Codificación y pensamiento computacional

La práctica de la codificación, en su forma tradicional, implica la manipulación de información simbólica y se reconoce generalmente que permite a los estudiantes crear y no sólo utilizar. En realidad, el discurso sobre las prácticas de codificación está incrustado en el más amplio sobre el pensamiento computacional. Recientemente, la CE ha dedicado un amplio estudio al estado del arte del pensamiento computacional (Bocconi, 2016). Lo que resulta es que si bien el sentido de urgencia para introducir el pensamiento computacional en la educación obligatoria es bastante fuerte, al mismo tiempo sigue existiendo una falta de consenso, incluso sobre la mera definición del concepto. Así, nos encontramos en la incómoda situación de actuar con urgencia pero sin tener la posibilidad de tener el control. Las cosas se mueven rápido, en realidad. Falta el tiempo para dar forma a las importantes lecciones aprendidas a través de las experiencias en curso y aprovecharlas plenamente. Esto es exactamente lo que está sucediendo con la evolución de las prácticas de codificación en el contexto educativo. Para decirlo en pocas palabras, parece que, frente a la explosión de lenguajes y dispositivos de cualquier tipo, las motivaciones pedagógicas y técnicas sólidas y profundas que inspiraron las anteriores experiencias de codificación educativa se han desvanecido en una nube centelleante de actividades de fantasía.

Definición del pensamiento computacional

Hay varias definiciones de "pensamiento computacional" por ahí.

El último dado en 2010 por Jeannette Wing, la informática que introdujo por primera vez en 2006 el término dice lo siguiente

El pensamiento computacional son los procesos de pensamiento que intervienen en la formulación de problemas y sus soluciones, de modo que las soluciones se representan de una forma que puede ser llevada a cabo eficazmente por un agente de procesamiento de información.

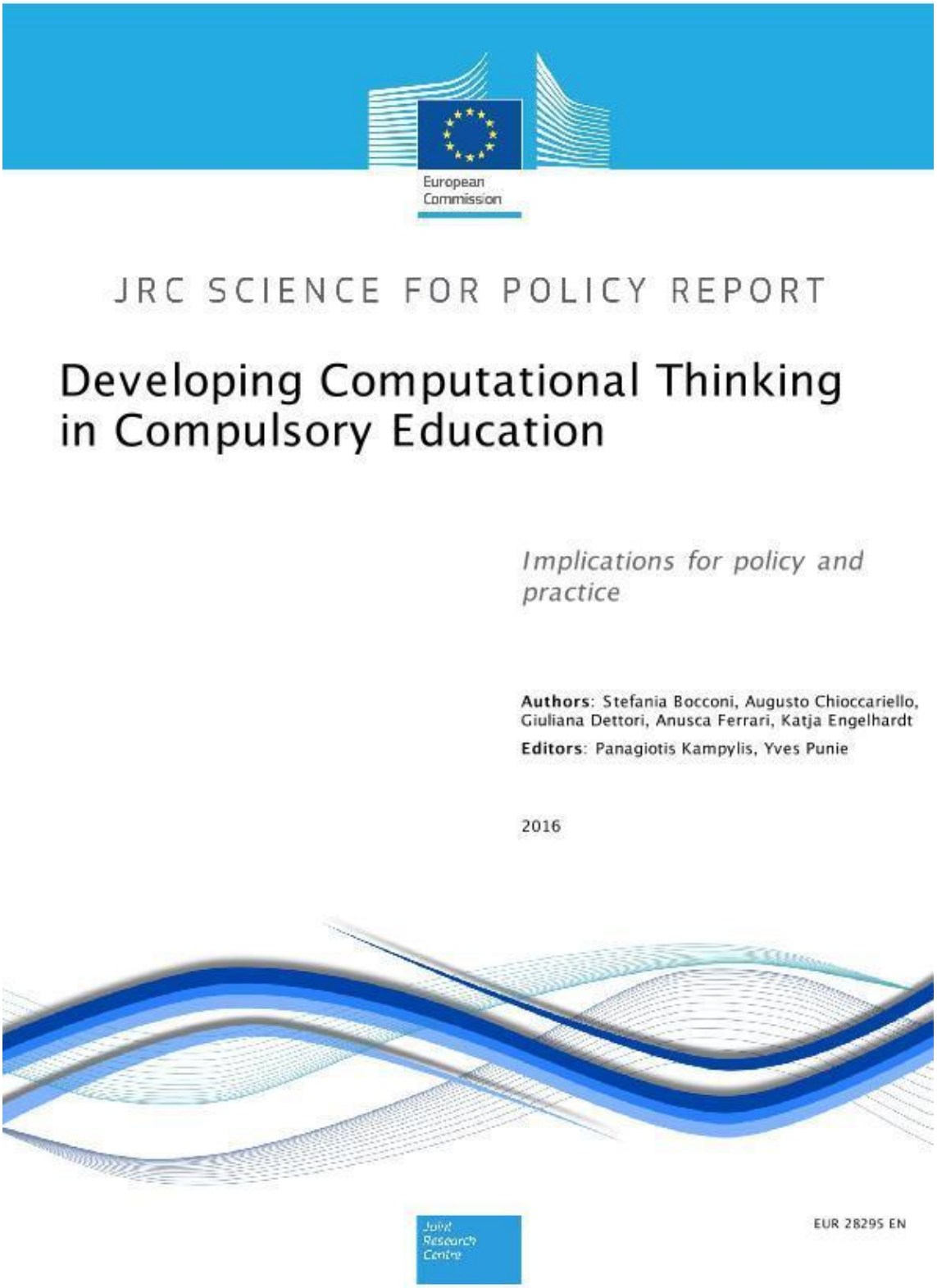
Más específicamente, con esta expresión se quiere decir un conjunto de habilidades:

- Usar abstracciones para representar el problema de maneras nuevas y diferentes
- Lógicamente organizando y analizando los datos
- *Divide et impera*
- Enfrentar problemas de uso de iteración, representación simbólica y operaciones lógicas.
- Reformular los problemas en pasos ordenados
- Aplicar estas habilidades a una amplia gama de problemas

Una buena revisión sobre el concepto de pensamiento computacional en la educación

Bocconi S., Chioccariello A., Dettori G., Ferrari A. y Engelhardt K. (2016). *Desarrollo del pensamiento computacional en la educación obligatoria.* ED. P. Kampylis e Y. Punie Science for Policy, informe del Centro Común de Investigación (CCI), el servicio de ciencia y conocimiento de la Comisión Europea.

Los autores muestran cuán amplio es el ámbito de las perspectivas propuestas por los expertos en la materia y cómo sigue siendo un concepto conmovedor.



European Commission

JRC SCIENCE FOR POLICY REPORT

Developing Computational Thinking in Compulsory Education

Implications for policy and practice

Authors: Stefania Bocconi, Augusto Chioccariello, Giuliana Dettori, Anusca Ferrari, Katja Engelhardt

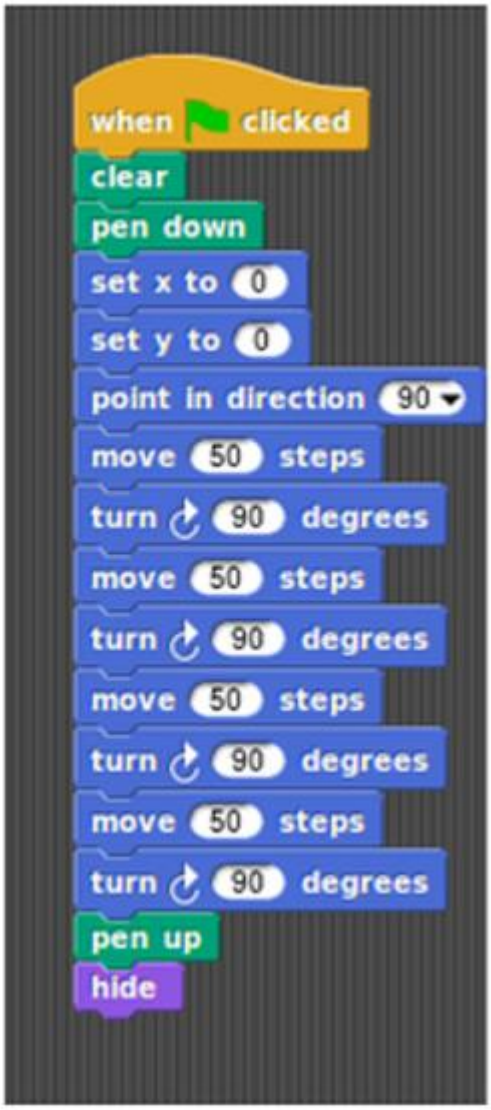
Editors: Panagiotis Kampylis, Yves Punie

2016

Joint Research Centre

EUR 28295 EN

Bloques vs. codificación textual




```
CLEARSCREEN
PENDOWN
HOME
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
PENUP
HIDETURTLE
```

Hoy en día los sistemas de programación visual son muy populares en los entornos escolares, en particular, los lenguajes basados en bloques en lugar de los lenguajes de texto convencionales. La figura muestra dos piezas de código, una en un lenguaje basado en bloques (Scratch) y la otra en un lenguaje textual (LibreLogo). En los sistemas basados en bloques, la instrucción individual se representa mediante bloques de color que se pueden arrastrar libremente dentro del área de trabajo. Los bloques se pueden combinar de forma similar a Lego para componer una secuencia de instrucciones, es decir, el programa. Por otro lado, en la programación textual las instrucciones deben escribirse en secuencia por medio de un editor de texto.

Ambas piezas de código producen el mismo resultado, ya que ambas dibujan un cuadrado aunque, a primera vista, no lo parezcan. Sin embargo, si compara cuidadosamente los dos objetos, verificará fácilmente la correspondencia entre las instrucciones.

Hoy en día, en la mayoría de los contextos escolares, la codificación es sinónimo de Scratch o, en todo caso, de lenguajes basados en bloques. Estos lenguajes son bastante inteligentes para proporcionar una primera experiencia de programación a los niños. Y son incluso poderosos, permitiendo una amplia gama de experiencias de codificación.

Además, Scratch fomentó mucho la difusión de la codificación, a través de su plataforma social. Sin embargo, dentro de todo el proceso, la idea general de la codificación se inclinó sobre la producción de animaciones, lo que podría estar bien, porque para realizarlas se requieren algunos métodos de programación bastante avanzados. ¿Qué nos estamos perdiendo entonces? Para entender esto tenemos que recordar la lección de Seymour Papert.

Entre los varios lenguajes basados en bloques, Scratch es, con mucho, el más popular. Puedes ir a explorarlo por un tiempo.

Todo comienza con Seymour Papert

La idea de incluir la programación informática entre las actividades educativas se debe a Seymour Papert. Papert, un matemático sudafricano, llegó a Estados Unidos a mediados de la década de 1960 después de haber trabajado con Jean Piaget durante cinco años. Publicó la primera versión de Logo en 1967, cuando trabajaba en el Laboratorio de Inteligencia Artificial del MIT. El logotipo era un lenguaje avanzado concebido en la intersección entre los campos de la inteligencia artificial y la psicología del desarrollo, como una herramienta para mejorar la forma en que los niños aprenden y resuelven problemas. Su idea clave, usando la famosa expresión de Papert, era permitir un piso bajo y un techo alto. Por esta razón, aunque aparentemente simple en los primeros pasos, su arquitectura interna permitía a los usuarios ampliar sus capacidades de una manera virtualmente ilimitada.

El papel clave de Seymour Papert es el de haber conectado los campos de la educación con los de la inteligencia artificial. Todo el bombo actual sobre las tecnologías educativas proviene de su trabajo fundamental y de su posición visionaria. Gracias a él hay un camino continuo desde el aprendizaje de teorías hasta el uso de la tecnología en las escuelas. Esto se olvida de alguna manera hoy en día y no debería.

Un gran número de idiomas educativos han sido derivados del Logo, entre los cuales Scratch, con mucho el más exitoso. Scratch es un pariente de Logo que está siendo desarrollado por Mitchel Resnick, un ex-estudiante y sucesivamente compañero de trabajo de Papert en el MIT. En realidad, las funcionalidades básicas de Logo también se pueden encontrar en Scratch que, sin embargo, tiene muchas más características, entre las que destacan la interfaz basada en bloques en lugar de texto y la posibilidad de construir animaciones o verdaderos videojuegos. Por otro lado, Logo se pensó como una forma de explorar conceptos matemáticos de forma físico-sintónica, otra expresión papeltiana que se refiere a la idea de construir una geometría -la Geometría de la Tortuga- en analogía con la geometría corporal que es bien conocida por los niños, antes de que se pongan en contacto con las matemáticas formales.

¿Y qué, si el Logo está incluido de alguna manera en Scratch? El hecho es que todas las consideraciones sobre lo que realmente sucede cuando se deja que los niños exploren con el Logo, toda la conciencia sobre la importancia del descubrimiento personal de los conceptos matemáticos, todo el fuerte énfasis en el enfoque creativo para el estudio de las ideas científicas han desaparecido casi por completo. No porque Scratch, u otros lenguajes similares, haga que tales perspectivas sean imposibles, sino porque toda la

interfaz está demasiado sesgada hacia el lado infantil. Esto no significa que no pueda hacer cosas muy complejas con Scratch, ni siquiera las más complejas. En cambio, se trata de que la mayoría de las actividades que se realizan en Scratch se centran en la producción de animaciones y juegos sencillos, y la mayoría de los proyectos son muy básicos y de corta duración, como han demostrado algunos estudios recientes basados en el raspado de la base de datos de Scratch (Aivaloglou & Hermans, 2016; Matias, Dasgupta & Hill, 2016; Scaffidi & Chambers, 2016). Podríamos decir que más no es necesariamente mejor....

Irónicamente, el sabor infantil, pensado para facilitar la introducción a la programación, resulta ser un factor limitante, básicamente porque los estudiantes anhelan mucho: si completas una tarea dura te has probado a ti mismo, si fallas... después de todo no fue tan fácil (Krouse, 2016A). Paradójicamente, Scratch puede ser frustrante porque todo parece tan fácil, pero pronto podría ser mucho más difícil. Porque codificar es difícil. Como las matemáticas. Hacer la vida mucho más fácil no siempre es una buena idea. Varios estudios revelaron que una introducción a la programación de Scratch no necesariamente facilita la transición a los lenguajes de codificación convencionales (Lewis, 2010; Lewis, Esper, Bhattacharyya, Fa-Kaji, Dominguez & Schlesinger 2014; Weintrop & Wilensky, 2015). Por eso están surgiendo nuevos enfoques para facilitar la transición a las "lenguas verdaderas" (Homer & Noble, 2014; Price & Barnes, 2015; Krouse, 2016B).

Biblio

- Krouse S. (2016A). *Scratch tiene un problema de marketing*. En Medium.
- Lewis C.M.(2010). *Cómo el entorno de programación determina la percepción, el aprendizaje y los objetivos: Logo vs. Rasguño*. Proceeding SIGCSE '10 Proceedings of the 41st ACM technical symposium on Computer science education, Pages 346-350.
- Lewis C., Esper S., Bhattacharyya V., Fa-Kaji N., Dominguez N. y Schlesinger A. (2014). *Percepciones de los niños sobre lo que se considera un lenguaje de programación*. J. Comput. Colección de Ciencias, 29(4): 123-133.
- Weintrop D., Wilensky U. (2015 A). *Bloquear o no bloquear, esa es la pregunta: las percepciones de los estudiantes sobre la programación basada en bloques*. Proceeding IDC '15 Proceedings of the 14th International Conference on Interaction Design and Children, 199-208.
- Homer M. & Noble J. (2014). *Combinando vistas en mosaico y texto del código*. Proceeding VISSOFT '14 Proceedings of the 2014 Second IEEE Working Conference on Software Visualization, 1-10.
- Price T.W., Barnes T. (2015). *Comparación de interfaces textuales y de bloques en un entorno de programación novato*. Proceeding ICER '15 Proceedings of the eleventh annual International Conference on International Computing Education Research , 91-99.
- Krouse S. (2016B). *WoofJS - haciendo que JavaScript sea aprendible*. En Medium.

Seymour Papert: sobre Logo, pensamiento y sentimientos

Bienvenido a las cintas de Logo. Estas cintas son sobre el logotipo, pero no son sólo sobre el logotipo, más allá del logotipo, son sobre el pensamiento. Se trata de cómo pensar en las computadoras y cómo usarlas para pensar en otras cosas. Se trata de cómo utilizar una experiencia de Logotipo, para desarrollar nuevas habilidades de pensamiento tanto para usted como para sus estudiantes. Pero incluso más allá del pensamiento, la cinta tiene mucho que decir sobre los sentimientos. Las personas, tanto adultos como niños, tienen fuertes sentimientos acerca de las computadoras, y su experiencia con las computadoras influye en la manera en que se sienten acerca de muchas otras cosas. Por ejemplo, sobre la escuela, sobre el aprendizaje y, lo que es más importante aquí, la experiencia de la gente con las computadoras a menudo influye en la forma en que piensan sobre sí mismos.

Papert S. (1986). *Seymour Papert en Logo*. En Logo Foundation.

La lección de Papert no es puramente técnica. Tampoco se limita a competencias específicas, al acceso a la información, al intercambio de información, etc. La idea de Papert de usar computadoras en la educación es holística. Logo fue concebido para explorar la geometría, las matemáticas o incluso la ciencia, mediante simulaciones inteligentes. Pero aún más que eso:

El propósito principal de Logo no es lo que ellos llaman "alfabetización informática" - por supuesto que sirve para eso, basado en cualquier otra cosa que se me ocurra, pero el propósito real no es tener una mejor comprensión de las computadoras, sino a través de las computadoras para tener una mejor comprensión de todo lo demás, incluyendo, me gustaría decir, a usted mismo. [...] No estoy tratando de darte una teoría de lo que hace que los niños estén tan involucrados y comprometidos con una computadora, estoy tratando de alentar una forma de pensar que mira más allá del papel de la computadora en la enseñanza de uno u otro rincón del plan de estudios y trata de ver las raíces emocionales de lo que está sucediendo.

El pensamiento de Papert enfatiza el placer y el beneficio de descubrir el aprendizaje, la apropiación, hacer suyo el conocimiento de una manera en la que uno se siente bien al respecto, buscando resonancia entre la experiencia de aprendizaje inmediata y la experiencia más amplia que constituye la vida del estudiante.

En esta cinta, traté de mostrar cómo un profesor puede usar el Logo para jugar el papel de pegamento intelectual, el papel que las matemáticas han hecho por mí. En otro extremo, por medio de algunas reflexiones, desempacando la intuición que cada maestro tiene, eso es bueno para hacer conexiones.... bueno, ¿por qué? Hay un lado cognitivo: las conexiones te ayudan a entender, entiendes lo nuevo refiriéndote a lo viejo, te ayudan a recordar. Pero hay un lado más profundo, uno que tiene que ver con cómo te sientes acerca del conocimiento y cómo te sientes acerca de ti mismo. Conectar el nuevo conocimiento con las cosas que sabes y amas y las cosas que puedes hacer te hace sentir bien al respecto, te hace tomarlo en una forma que es la tuya propia, pero al tomar el conocimiento en forma que te hace sentir como tú, también cambias tus sentimientos hacia ti. Ya no piensas en ti mismo como alguien que puede hacer matemáticas pero que no entiende realmente la poesía, o que puede dibujar pero que no tiene la cabeza para los números. En vez de eso, ustedes se apropian de todo el conocimiento en una forma que es la suya, que pueden hacer, que pueden amar. Y a través de amar lo que sabes que te amas más a ti mismo.

Los pasajes anteriores de Papert han sido extraídos de una serie de videos realizada en 1986. Sin embargo, aunque las tecnologías utilizadas por Papert en estos vídeos puedan parecer bastante obsoletas hoy en día, y aunque el software derivado de Logo en estos treinta años sea extremadamente valioso, creemos que la visión de Seymour Papert todavía pertenece al futuro y es algo por lo que todavía tenemos que luchar. Creemos que estas consideraciones dan el tono adecuado para llevar a las personas desde las escalas inferiores a las superiores del proceso de participación digital descrito en"Creciendo Ciudadanos Digitales": de mirar a compartir, a crear y, finalmente, a aprovechar el potencial de la tecnología para una sociedad mejor (Ferrari & Martens, 2016, p. 12).

Software Libre

Según la Fundación para el Software Libre:

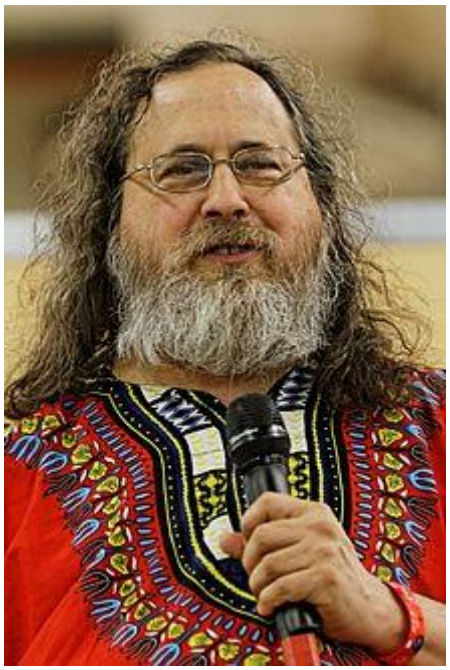
"Software libre" significa software que respeta la libertad de los usuarios y la comunidad. A grandes rasgos, significa que los usuarios tienen la libertad de ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software.

[...]

Un programa es software libre si los usuarios del programa tienen las cuatro libertades esenciales:

1.
    1. La libertad de ejecutar el programa como usted desee, para cualquier propósito (libertad 0).
    2. La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga su computación como usted desee (libertad 1). El acceso al código fuente es una condición previa para ello.
    3. La libertad de redistribuir copias para que pueda ayudar a su prójimo (libertad 2).
    4. La libertad de distribuir copias de sus versiones modificadas a otros (libertad 3). Al hacer esto, usted puede dar a toda la comunidad la oportunidad de beneficiarse de sus cambios. El acceso al código fuente es una condición previa para ello.

El concepto de Software Libre fue creado por Richard Matthew Stallman en 1983, un físico, que en ese momento trabajaba en el laboratorio de inteligencia artificial del MIT como programador. Renunció a su puesto de investigador cuando descubrió que el software se estaba convirtiendo en un componente estratégico de la propiedad intelectual de las empresas, en lugar de ser considerado como un conocimiento compuesto únicamente por ideas codificadas, que es un tipo particular de conocimiento y, como tal -así lo pensó Stallman-, debería ser libremente compartible entre todos los seres humanos.



Stallman creó

- el Proyecto GNU, un sistema operativo libre, sobre el cual se habría construido Linux, más adelante.
- la Free Software Foundation, dedicada a la promoción del Software Libre en todo el mundo
- la Licencia Pública General de GNU y el concepto de copyleft, que constituyen la base legal por medio de la cual los autores pueden ceder libremente una cierta cantidad de derechos de autor; un concepto que se generalizó sucesivamente con las licencias Creative Commons y otras, concebidas para hacer frente a las necesidades de los autores en el cambiante mundo de Internet.

Niveles de compromiso con el Software Libre

Las comunidades locales y las minorías pueden adaptar el software libre a sus necesidades e idiomas específicos, especialmente en el contexto multicultural y multilingüe de la Unión Europea. Es ético y útil utilizarlo y difundirlo libremente. Contrarresta la tendencia de romper el software privativo, lo que es contrario a la ley. Puede ser modificado y mejorado por cualquiera que sea capaz de hacerlo, y muchos jóvenes son perfectamente capaces de hacerlo. Fomenta la colaboración y la cooperación en proyectos complejos compartidos. El software libre es un poderoso incentivo para un enfoque creativo y ético en el uso de la tecnología. Por lo tanto, es un instrumento relevante de la democracia, especialmente en contextos educativos, en armonía con los valores de financiación de la Unión Europea.

Se puede usar software libre adoptando diferentes niveles de compromiso. La opción más radical es utilizar el sistema operativo Linux. Linux es un sistema operativo inteligente con varias ventajas para la mayoría de los usuarios y hoy en día se puede instalar con bastante facilidad. Sin embargo, incluso si prácticamente cualquier usuario pudiera permitirse la transición, en la práctica muchos usuarios podrían tener razones para mantener sus sistemas, sin importar si son Windows o Mac OS X. Sin embargo, el

software libre puede ser adoptado en el nivel mucho más fácil de las aplicaciones individuales. Hay muy buenas aplicaciones que se pueden instalar en todos los sistemas operativos, como la suite de oficina LibreOffice, el editor de imágenes Gimp y el editor de audio Audacity, por mencionar algunas de las más populares.

La idoneidad del modelo de software libre para contextos multiculturales destaca en el caso de LibreOffice: unas 50 comunidades de todo el mundo están activas para desarrollar y mantener sus respectivas versiones localizadas de LibreOffice, pero en este momento se admiten 178 idiomas, que pueden incluir hasta cierto punto una interfaz de usuario localizada, un sistema de ayuda localizado, listas de autotexto, listas de autocorrección, diccionarios de corrección ortográfica, patrones de separación de sílabas, gramática y tesauro (sinónimos). LibreOffice incluye todas las aplicaciones típicas de las suites de Office para escribir, presentar, organizar datos, dibujar, etc. Pero la razón por la que destacamos aquí el interés en LibreOffice es porque, entre las numerosas funcionalidades, se encuentra LibreLogo, una versión bastante completa del lenguaje de logotipos, disponible por defecto entre las herramientas estándar de LibreOffice desde la versión 4.2.3.3 (2014).

Recordando el principio de Seymour Papert de un piso bajo y un techo alto, LibreOffice es una implementación muy inteligente de Logo. El "piso" es extremadamente bajo ya que para empezar hay que introducir Writer (el procesador de textos estándar de LibreOffice), luego escribir algunas instrucciones del Logotipo y ejecutar el código con sólo pulsar un botón del menú. Si el código es correcto, se incrusta una imagen en el documento como gráfico vectorial estándar de LibreOffice. De esta manera es muy sencillo empezar a experimentar con la "geometría de la tortuga" de Papert. Como hemos dicho, en realidad Scratch se derivó de Logo pero, en lugar de ser codificado por medio de instrucciones de texto, utiliza bloques de colores que se pueden juntar de una manera similar a la de Lego para componer un programa. La ventaja de este sistema es que evita la posibilidad de errores ortográficos y sintácticos. Esto puede bajar la palabra al principio pero, sucesivamente, puede incluso dificultar la transición a las "verdaderas lenguas", como hemos señalado antes. En LibreLogo tienes que escribir instrucciones de texto, que al principio pueden ser más exigentes pero no más que escribir oraciones sencillas en inglés. De hecho, es bueno que el mismo tipo de habilidades pueda ser útil en diferentes áreas. LibreLogo se puede utilizar fuera de línea, sin tener que estar conectado a un servicio web, algo que puede causar algunos problemas de brecha digital - en muchas regiones esto sigue siendo un problema. Incluso la compartición de programas, para intercambiar problemas y soluciones, es extremadamente fácil, ya que simplemente requiere enviar textos cortos, por cualquier medio, de nuevo sin tener que depender de una plataforma en línea. Finalmente, con LibreLogo el énfasis se pone naturalmente en las matemáticas y las ciencias, lo que es bueno, ya que la difusión de una verdadera cultura científica sigue siendo un problema.

Dibujar con LibreLogo

¿La enseñanza de Logo a los alumnos de los profesores se parece a la enseñanza a los niños?

Esta es una introducción al Logotipo dirigida principalmente a los alumnos de los profesores, donde mostramos cómo iniciar a los niños en la práctica del Logotipo. Curiosamente, después de haber propuesto este asunto a varios cientos de estudiantes de magisterio, me di cuenta de que enseñar el Logo a los niños es muy similar a enseñar a los estudiantes adultos. Y, en promedio, durante el primer acercamiento los niños se desempeñan aún mejor! A menudo lo discutimos con aquellos estudiantes que afirman haber tenido grandes dificultades para hablar con la Tortuga. La mayoría de ellos experimentan una especie de "resurrección" después de una lucha inicial: "Al principio no entendía nada, pero luego... ¡qué maravilla!" Pero pocos nunca se liberan del mal humor, al menos en las dos semanas y media que dura el laboratorio. Hablar con estos estudiantes es siempre muy interesante porque se sorprenden cuando les digo que, la mayoría de las veces, los niños de 9 años se llevan muy bien con la Tortuga. Cuando se trata de ahondar en esta paradoja, parece que los niños captan con naturalidad el aspecto lúdico de la situación. Por otro lado, como hemos señalado en el capítulo anterior, los adultos se ven fácilmente atrapados en sus propios prejuicios: No tengo cabeza para los números, la computadora y yo estamos muy lejos, nunca estuve de acuerdo con las tecnologías. Especialmente la última es una declaración que escuché muchas veces, y, sorprendentemente, de verdaderos nativos digitales. Es decir, una persona que puede tener miles de contactos en Facebook, afirma con franqueza que no se lleva bien con las tecnologías.

Empecemos a escribir algo

Habiendo enfatizado que, por favor, enfrente a la Tortuga con el espíritu más juguetón posible, comenzando desde cero. Ya que supongo que usted sabe cómo los niños enfrentan un nuevo juego, trate de hacer lo mismo. Habiendo enfatizado que, por favor, enfrente a la Tortuga con el espíritu más juguetón posible, comenzando desde cero. Ya que supongo que usted sabe cómo los niños enfrentan un nuevo juego, trate de hacer lo mismo. Ahora, siempre que haya descargado LibreoOffice y activado la barra de herramientas Logo, como se explica en el primer capítulo, abra un nuevo documento de texto. Estás frente a una hoja en blanco: puedes escribir una carta, una lista de compras, un poema o lo que quieras. Trate de

escribir lo siguiente (usé caracteres en mayúsculas sólo para mayor claridad, LibreLogo es "insensible a mayúsculas y minúsculas"):

DELANTERO 100

¿Qué es lo que ves? Olvida tus "límites" y todo lo que piensas sobre tus relaciones con las matemáticas, la tecnología, etc. Mira el resultado que has obtenido y compáralo con lo que has escrito. Hágase preguntas. Experimenta con la Tortuga sobre las posibles respuestas que te vienen a la mente.

Antes de probar algo nuevo, escriba los siguientes comandos antes del anterior:

CLEARSCREENHOME

Con la primera se cancela el dibujo anterior, con la segunda se envía la Tortuga en su posición "home", es decir, en el centro de la hoja con la nariz apuntando hacia arriba.

CLEARSCREENHOMEFORWARD
100

A continuación, pulse de nuevo el botón "Run". ¿Qué ha cambiado?

Por favor, toca....

Ahora, con el fin de explorar mejor lo que la Tortuga puede hacer, les digo un par de otras órdenes:

DERECHA e IZQUIERDA.

No es difícil imaginar para qué son estos comandos, ¿verdad? Pero falta algo, para poder usarlo, ¿qué? Bueno, trata de adivinar y experimentar. Por favor, reflexiona sobre lo que realmente está pasando con la Tortuga, cuando apliques estos comandos. Luego, sólo tienes que jugar para ver qué puedes hacer. Recuerda, cuando eras un niño...

Guarde su trabajo de vez en cuando....

La razón por la que insisto tanto en lo que los niños pueden hacer es que tuve varias oportunidades de trabajar con ellos y con la Tortuga. Realmente algunos dicen que es demasiado difícil, la mayoría sólo juegan y se emocionan al dejar que el pequeño animal cree cosas divertidas. Una vez, tratando de dibujar una casa, una chica salió con una especie de castillo bizare. La elogié, comentando lo maravilloso que era este dibujo, estaba tan orgullosa. Entonces, cuando volví a ella, después de haber considerado las obras de otros niños, la encontré llorando, sollozando desesperadamente: había borrado accidentalmente los mandamientos de una manera que ya no he podido recuperar. Me sentí culpable por no haberle enseñado cómo salvar el trabajo de vez en cuando.

Por lo tanto, si empiezas a crear cosas más complejas que te gustan, **no olvides de guardar el documento de vez en cuando. Es tan fácil**.

En el pasado, esta introducción era mucho más larga. Siguiendo la experiencia que he acumulado durante los últimos años, tanto trabajando con los alumnos de los profesores como con los niños, prefiero parar aquí, invitándoles a explorar libremente las posibilidades. Desde aquí se puede ir en varias direcciones. Si desea indicaciones específicas y ejemplos para el uso de los comandos básicos, o si desea descubrir instrucciones más potentes, las encontrará en el siguiente capítulo y en los siguientes. Más tarde descubrirás las posibilidades de las exploraciones leyendo la historia de Marta, una ex-maestra estudiante mía. O, si quieres leer sobre una manera fascinante de introducir a los niños en el dibujo de un círculo, ve y lee la sección sobre el círculo, bueno, la primera parte de él, de lo contrario empezarás a volar....

Vamos a entrar en detalles

El programa permite crear gráficos a través de los movimientos de una "tortuga" que obedece órdenes específicas. Veamos un ejemplo.

Abra un nuevo documento de texto y escriba este comando:

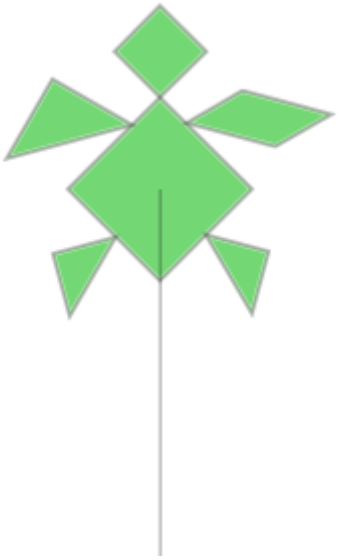
INICIO

Verás que la tortuga acaba de aparecer en el centro de la pantalla con la cabeza hacia arriba.

Por cierto, ten en cuenta que la Tortuga es en sí misma una gráfica de LibreLogo y es una composición de Tangram....

Ahora agregue un comando FORWARD:

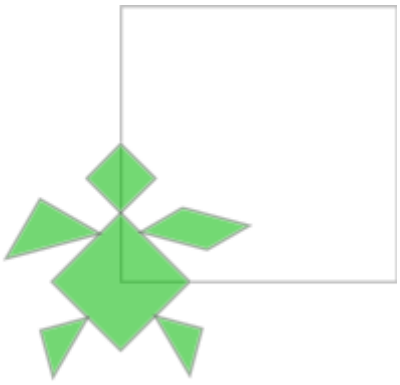
HOMEFORWARD
 100



La Tortuga viajó una distancia de 100 puntos de visualización dejando un rastro, por lo tanto drqwing un segmento de 100 puntos de largo. Al final, la Tortuga permanece donde está, al final del nuevo segmento, esperando otra orden.

Alternativamente, también puede especificar otras unidades espaciales, por ejemplo 10mm, 10cm, 10 in. Sólo inténtalo. Recuerde que 1 pt = 1/72 pulgadas, donde 1 pulgada = 25.4 mm. Por lo tanto, 1 pt = 2,54/72 = 0,3527 mm

Ahora vamos a intentar usar los comandos de giro que descubriste antes, por ejemplo RIGHT....



Hemos dibujado cuatro lados, y al final de cada lado giramos a la derecha 90°, obteniendo así un cuadrado.

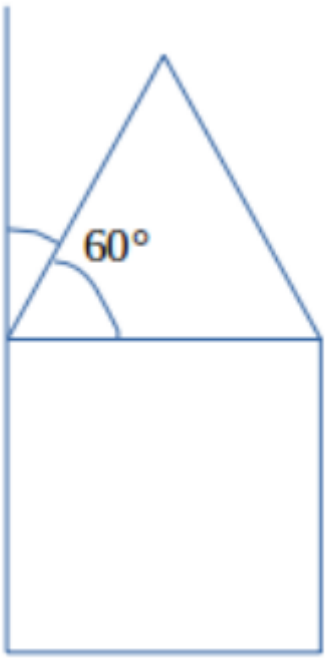
La peculiaridad de LibreLogo entre otros entornos de software

Sólo dos palabras sobre la peculiaridad de LibreLogo. La secuencia de instrucciones que escribimos es un poco de código, es un software. Lo escribimos en un tipo particular de lenguaje, que es el de Logo, y también es muy simple, pero es un software como cualquier otro. Por lo general, el software se escribe en documentos especiales usando edición de texto y edición de contexto simple y se guarda de esa manera. Luego, la computadora los ejecuta. Las formas en que se ejecutan estas operaciones varían mucho, dependiendo del tipo de lenguaje y del contexto. Hoy en día hay cientos de idiomas diferentes que se utilizan para los objetivos más diferentes.

La particularidad de LibreLogo es que el software está escrito en un documento y la tortuga "trabaja" en el mismo documento, dejando su marca de forma gráfica. Así, se obtiene tanto el código como su resultado gráfico en un solo documento. Los gráficos se pueden seleccionar con el ratón y, eventualmente, se pueden transferir a varios contextos. Por ejemplo, creé las fotos de arriba jugando con la tortuga en otro documento, luego seleccioné los gráficos y los traje aquí. Otra notación: un grupo de instrucciones hechas para operar en secuencia se llama script, expresión que usaremos profusamente.

Un clásico: construyamos una casa

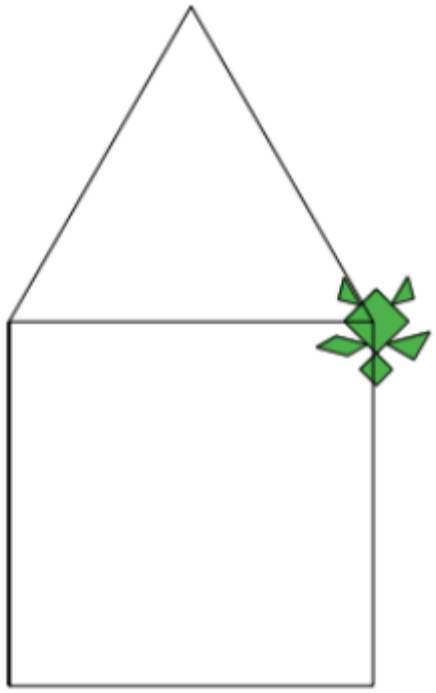
Ahora queremos construir una casa, por lo tanto necesitamos un techo. Podemos hacerlo dibujando un triángulo equilátero sobre el cuadrado, cuya base coincide con la parte superior del cuadrado. Suponiendo que construimos un cuadrado de 50 mm de lado, siendo el techo equilátero, los otros lados del triángulo tendrán que ser de 50 mm de largo también. Por lo tanto, para hacer el lado izquierdo del techo, la tortuga tendrá que moverse 50 mm, pero antes de eso, tiene que cambiar de dirección. ¿Cuánto? Dado que los ángulos internos de un triángulo equilátero son de 60° cada uno, la tortuga necesita desviarse 90° - 60° = 30° a la derecha. Muy a menudo, los principiantes, e incluso los adultos, dejan que la Tortuga gire 60°, enfocándose en el valor del ángulo interno del triángulo, en lugar de en la desviación de la dirección actual de la Tortuga. Una vez que haya llegado a la parte superior, dibujando el lado izquierdo del techo, tendrá que girar 120° a la derecha para dibujar el lado derecho. Finalmente, girará 30° a la derecha para alinearse con la pared de la casa.



Aquí estamos....

PANTALLA PANTALLA CLARA HACIA ARRIBA
50mm DERECHA 90FORA
50mm DERECHA 90
Adelante 50mm DERECHA 90Adelante
50mm DERECHA 90

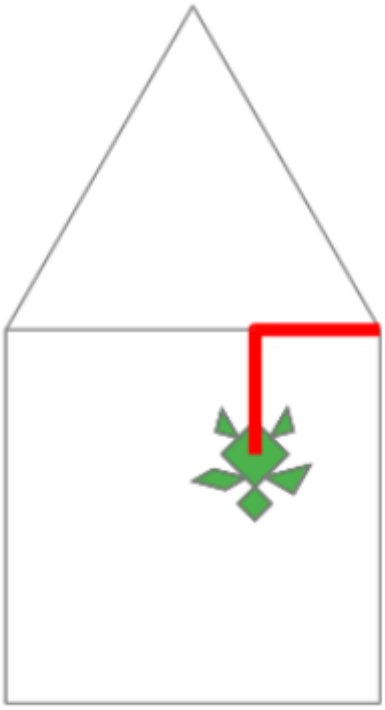
Pusimos algo más en este código. En primer lugar hemos añadido, en la parte superior, algunas instrucciones de limpieza: limpiar la pantalla de cualquier otro tipo de gráficos eventualmente colgando alrededor de casa y - es conveniente hacer esto antes de cualquier nuevo dibujo. Segundo, hemos escrito un par de instrucciones seguidas: Adelante 50mm a la derecha 90. Con LibreLogo usted puede hacer esto, el comportamiento es el mismo. Es sólo una cuestión de legibilidad del código. Es una buena idea hacer que el código sea lo más legible posible, especialmente cuando se vuelve más complejo.



Levantar el lápiz

Ahora, supongamos que queremos dibujar una ventana en medio de la pared. Para ello es necesario introducir dos nuevos comandos - PENUP y PENDOWN - que permiten mover la tortuga sin dibujar.

PANTALLA PANTALLA DELANTERA
50mm DERECHA 90PORTAL
50mm DERECHA 90PORTAL 50mm DERECHA 90PORTAL
50mm DERECHA 90PORTAL 50mm DERECHA 30PORTAL
50mm DERECHA 120PORTAL
50mm DERECHA 120
PENUPFORWARD
50mm/3 DERECHA 90FORWARD
50mm/3 DERECHA 90



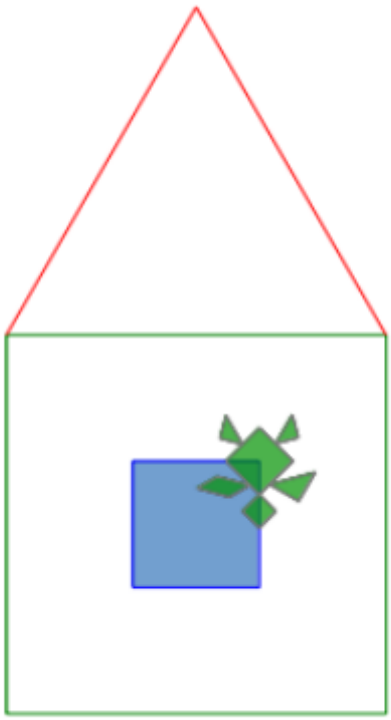
En la figura el camino con el lápiz hacia arriba se hizo rojo.

Finalmente:

PANTALLA PANTALLA DELANTERA
50mm DERECHA 90PORTAL
50mm DERECHA 90PORTAL 50mm DERECHA 90PORTAL
50mm DERECHA 90PORTAL 50mm DERECHA 30PORTAL
50mm DERECHA 120PORTAL
50mm DERECHA 120
PENUPFORWARD
50mm/3 DERECHA 90FORWARD
50mm/3 DERECHA 90
PENDOWFORWARD
50mm/3 DERECHA 90FORWARD
50mm/3 DERECHA 90FORWARD
50mm/3 DERECHA 90FORWARD
50mm/3 DERECHA 90

Dibujo con lápices de color

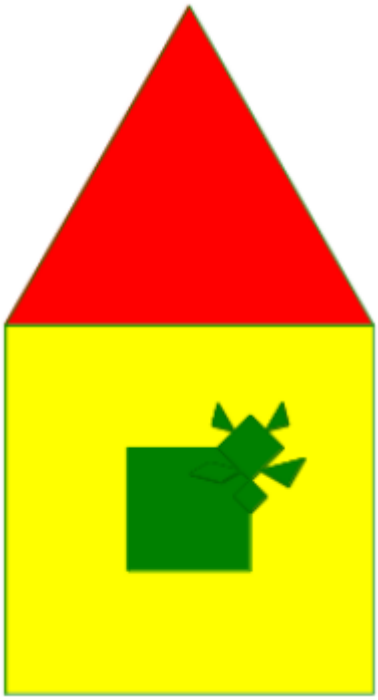
CLEARSCREENHOME
PENCOLOR "verde "
FORWARD 50mm RIGHT 90FORWARD
50mm RIGHT 90FORWARD
50mm RIGHT 90FORWARD 50mm RIGHT 90FORWARD
90FORWARD
50mm RIGHT 30PENCOLOR
"rojo "
FORWARD 50mm RIGHT 120FORWARD 50mm RIGHT
120PENUPFORWARD
50mm/3 LEFT 90FOR
50mm/3PENDOWN
PENCOLOR "azul "
FORWARD 50mm/3 RIGHT 90FORWARD
50mm/3 RIGHT 90FORWARD
50mm/3 RIGHT 90FORWARD 50mm/3
RIGHT 90
PENCOLOR "gris



¿Y por qué no colorear el interior también?

CLEARSCREEN HOMEFORWARD
50mm RIGHT 90FORWARD
50mm RIGHT 90FORWARD
50mm RIGHT 90FORWARD 50mm RIGHT 90FORWARD
50mm RIGHT

30
FILLCOLOR "amarillo " FILLFORWARD
50mm DERECHO 120FORWARD
50mm DERECHO 120PENSIONES
HACIA ADELANTE 50mm/3 IZQUIERDA
90 HACIA ADELANTE
50mm/3 PENDIENTE
FILLCOLOR "rojo" FILLFORWARD
50mm/3 RIGHT 90FORWARD
50mm/3 RIGHT 90FORWARD
50mm/3 RIGHT 90FORWARD 50mm/3
RIGHT 90
FILLCOLOR "verde " FILL



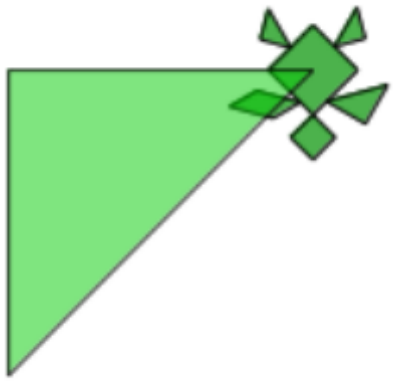
Cómo funciona el coloreado

En este código hemos agrupado dos instrucciones. Con el primero establecimos qué color usar: FILLCOLOR "verde". Con la segunda -FILL- le pedimos a la Tortuga que coloree la figura. La instrucción FILL hace dos cosas, en realidad: cierra la figura y la rellena con un color. Para entender lo que significa el cierre, intente ejecutar

ADELANTE 100 DERECHA 90ADELANTE
100 DERECHA 90

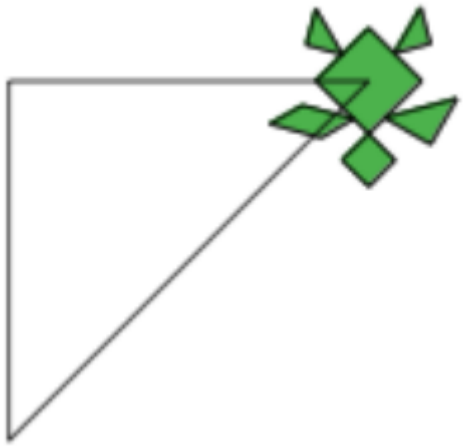
Obtendrá un triángulo rectángulo abierto, ya que falta el tercer lado, la hipotenusa. Puede cerrar y rellenar la figura con

ADELANTE 100 DERECHA 90ADELANTE 100 DERECHA 90RELLENAR



O, si lo que quieres es simplemente cerrarlo, sin colorear

ADELANTE 100 DERECHA 90ADELANTE 100 DERECHA 90CIERRE



Cierre a la manera estándar de la tortuga

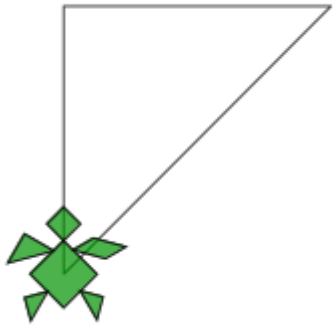
De lo contrario, debe calcular el ángulo de giro y la longitud correctos de la hipotenusa y, a continuación, dar las órdenes de movimiento adecuadas.

ADELANTE 100 DERECHA 90ADELANTE 100 DERECHA 135ADELANTE SQRT(10000+10000) DERECHA 135

Bueno, hay algo nuevo aquí, trate de descubrirlo pensando en el teorema de Pitágoras....

En cualquier caso, esto es lo que obtenemos.

Note que aquí la Tortuga regresó en su **estado** inicial: la misma **posición** inicial y el mismo **rumbo** inicial. Recuerde este concepto: **estado**.



Manos a la obra

Nacimiento del logotipo (como robot)

Logo fue creado por Seymour Papert en los años setenta para mejorar el aprendizaje de las matemáticas. Seymour Papert, nacido en Sudáfrica en 1928, estudió matemáticas por primera vez en Johannesburgo y sucesivamente en Cambridge. Entre 1958 y 1964 se doctoró en la Universidad de Ginebra con Jean Piaget: interesante colaboración entre un matemático y un pedagogo. Desde 1964 fue investigador del laboratorio de Inteligencia Artificial del MIT donde, en 1967, obtuvo el cargo de codirector con Marvin Minsky, un científico relevante en el campo de la inteligencia artificial. El laboratorio es el mismo donde Richard Stallman, padre del software libre, habría trabajado unos años más tarde. Seymour Papert es famoso por haber inventado el Logo, un lenguaje de programación para dibujar dando órdenes a una "Tortuga". Sin embargo, en la primera versión, concebida en los años setenta, la Tortuga era un robot que sabía dibujar con lápiz. En esos días se necesitaba un laboratorio avanzado para construir un dispositivo como ese. Hoy en día se pueden encontrar robots educativos que son muy similares al primer robot Papert's Turtle a precios asequibles.

Entonces Logo se convirtió en un lenguaje de software

Cuando las computadoras llegaron a los hogares en los años 80, el Logo se convirtió en un software y como tal fue descrito por Seymour Papert en Mindstorms. Sin embargo, es importante señalar la perspectiva pedagógica del pensamiento de Papert. Leamos este pasaje, tomado de Mindstorms (pp. 7-8)[1]:

Tomo de Jean Piaget un modelo de niños como constructores de sus propias estructuras intelectuales. Los niños parecen ser alumnos innatamente dotados, adquiriendo mucho antes de ir a la escuela una gran cantidad de conocimientos mediante un proceso que yo llamo "aprendizajeiagetiano", o "aprendizaje sin ser enseñado". Por ejemplo, los niños aprenden a hablar, aprenden la geometría intuitiva necesaria para moverse en el espacio, y aprenden lo suficiente de lógica y retórica como para moverse alrededor de los padres - todo esto sin ser enseñado. Debemos preguntarnos por qué algunos aprendizajes tienen lugar tan temprano y espontáneamente, mientras que otros se retrasan muchos años o no se producen en absoluto sin una instrucción formal deliberada.
Si realmente miramos al "niño como constructor" estamos en camino a una respuesta. Todos los constructores necesitan materiales para construir. Lo que me diferencia de Piaget es el papel que atribuyo a las culturas circundantes como fuente de estos

materiales. En algunos casos, la cultura les proporciona abundancia, facilitando así el aprendizaje constructivo de losiagetianos. Por ejemplo, el hecho de que tantas cosas importantes (cuchillos y tenedores, madres y padres, zapatos y calcetines) vengan en pares es un "material" para la construcción de un sentido intuitivo del número. Pero en muchos casos en los que Piaget explicaría el desarrollo más lento de un concepto en particular por su mayor complejidad o formalidad, veo el factor crítico como la relativa pobreza de la cultura en aquellos materiales que harían que el concepto fuera simple y concreto.

1 Seymour Papert (1993) Mindstorms, Children, Computers, and Powerful Ideas. Basic books, Nueva York, 2 edición.

Versiones de logotipos

En los 90's el Logo circuló como un programa instalable desde un disquete. Una vez lanzado, produjo una pantalla negra en la que se podían escribir instrucciones en secuencia, una tras otra. Las instrucciones representaban los movimientos dados a la tortuga en la pantalla. Luego, con un comando especial, podrías "ejecutar" la secuencia de comandos, y así la tortuga se movería dejando una marca en la pantalla. Logo tuvo una gran resonancia como método experimental para la enseñanza de las matemáticas y se han derivado una gran variedad de versiones, llegando a generalizaciones como la actual Scratch. Sin embargo, no se ha distribuido ampliamente en las escuelas y tal vez ha tenido más éxito entre los niños que entre los maestros. Probablemente era demasiado pronto. Usar el Logo significa escribir código, una actividad que no es parte de la preparación de la mayoría de los maestros, incluyendo aquellos que enseñan ciencias. Hoy es quizás diferente, hablamos mucho de texto codificado, aunque quizás no siempre con pleno conocimiento de los hechos. La situación ha evolucionado tanto que la codificación puede significar muchas cosas diferentes. Además, desde los años 80 hasta la actualidad, la variedad de lenguajes de programación ha crecido enormemente. Hay muchas versiones de Logotipos disponibles. Aquí enumeramos sólo aquellos con los que hemos tenido la oportunidad de jugar:

- LibreLogo
- Kojo
- Academia de Tortugas
- Xlogo desarrollado en ETH Zürich

A continuación se ofrece una breve descripción de estos sistemas.

Librelogo

LibreLogo es la implementación del famoso lenguaje de logotipos dentro del procesador de texto Writer. Writer es la aplicación de procesamiento de texto de LibreOffice, análogamente a word, que forma parte de la suite MS Office. Se puede descargar desde http://libreoffice.org para cualquier tipo de plataforma. Pero, ¿qué significa usar Logo dentro de un procesador de textos como Writer, considerando que se trata de un procesador de textos normal, mientras que Logo es una especie de lenguaje de dibujo? Simple: Con la barra de herramientas de LibreLogo puede producir imágenes que se integran en el documento como si fueran importadas. Es una idea brillante, gracias a Németh László, que reprodujo las características del Logo en LibreOffice. En realidad, los mejoró aún más, aprovechando el lenguaje Python, con el que escribió el plugin. El uso de LibreLogo es muy sencillo: se abre un documento en Writer, se escribe un código en el lenguaje del logotipo, como si se tratara de cualquier otro texto, y luego se ejecuta pulsando el botón correspondiente en la barra de herramientas de LibreLogo; si el código es correcto, el sistema ejecuta el código dibujando una figura en el texto, justo en el centro de la página. El dibujo se incluirá como gráfico estándar de LibreOffice y puede ser manipulado en consecuencia. Más adelante veremos cómo configurar LibreLogo en LibreOffice.

La implementación de LibreLogo de Logo encaja muy bien con la famosa idea de Papert: un lenguaje de software educativo debe tener un piso bajo y un techo alto. Es decir, debería ser muy fácil de empezar y, al mismo tiempo, debería permitir el crecimiento de la complejidad. LibreLogo es probablemente la implementación más fácil para los principiantes: basta con abrir un documento de texto, escribir algunas instrucciones y pulsar un botón para ver el dibujo resultante.

Kojo

Kojo es un entorno de codificación muy potente pensado para enseñar matemáticas y otros temas relacionados con las ciencias. Es un programa Java que se puede descargar y ejecutar en cualquier sistema operativo. Puede descargarse de http://www.kogics.net. Las capacidades de Kojo van mucho más allá de las de un entorno típico de Logotipo. Actualmente es una implementación de Scala, un lenguaje de programación muy actualizado y potente. Entre otras muchas características, permite dibujar muchos cuadros simultáneamente, abriendo las puertas, por ejemplo, al mundo de las simulaciones de sistemas biológicos. Algo que no es posible hacer con la versión actual de LibreLogo. El inconveniente es que la sintaxis w es algo más compleja. Por ejemplo, mientras que para dibujar un segmento de longitud 100 con LibreLogo tiene que escribir lo siguiente

LibreLogo	**Kojo**
REPITE 4[ADELANTE 100 DERECHA 90]	repetir(4) {adelante (100) derecha(90)}

Incluso en este ejemplo trivial tienes que molestarte con más niveles de brackets.

Academia de Tortugas

Turtle Academy es un sitio Web para el aprendizaje del idiomaLogo. Los comandos son bastante similares a los de LibreLogo. Una diferencia es que los nombres de las variables tienen que ser declarados preponiendo dos puntos al nombre. Por ejemplo :a o :foo en lugar de a o foo. Hay lecciones gratuitas, y muchos ejemplos para explorar, un patio de recreo gratuito para experimentar. Después de haberte registrado (gratis) puedes guardar tus propios programas y compartirlos con otros. El patio de juegos es muy cómodo, basta con teclear comandos y con el retorno se ejecutan inmediatamente. A la derecha el manual está siempre disponible.

Simplemente copie la siguiente cadena....

repetir 4 [adelante 100 derecha 90]

y pégalo en la caja de comandos del patio de recreo, el tipo return....

Xlogo

Xlogo es un sitio Web para aprender Logo también. Es el sistema más simple disponible, por lo que sabemos, adecuado para niños de k1-5 años. Como con la Academia de las Tortugas.

repetir 4 [adelante 100 derecha 90]

y péguelo en la caja de comandos de la derecha. Escribiendo"enter" usted obtiene el resultado.

Xlogo tiene un sistema muy directo de procedimientos de construcción, es decir, nuevos comandos. Anticipémonos a este concepto aprovechando este sistema tan sencillo. Utilice el cuadro de edición en el lado izquierdo de la pantalla de Xlogo para escribir lo siguiente:

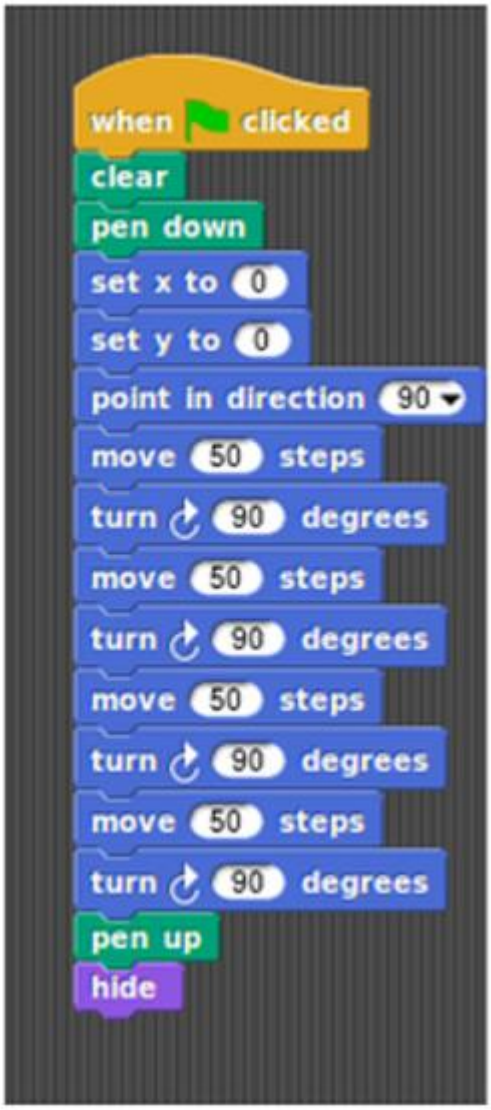
hasta la repetición cuadrada
4[
adelante 100 derecha
90
] (
sqrepeat
4)

cuadrado

Rasguño

Desde el surgimiento del Logo, en los años 80, la variedad de lenguajes de programación ha crecido enormemente. Hoy en día, entre las derivaciones de logotipos, Scratch es el lenguaje de programación educativo más reconocido. Mitchel Resnick, un ex-estudiante de Papert, es el líder del proyecto Scratch y ahora todavía lo sigue en el Laboratorio de Medios del MIT. Scratch va mucho más allá de la producción de gráficos y permite crear animaciones y videojuegos, lo que permite experimentar con técnicas de programación bastante sofisticadas. Otro aspecto innovador es que está estructurado como un servicio web y esto ha permitido la creación de una gran comunidad de difusión e intercambio de programas vivos. Desde el punto de vista operativo, Scratch se diferencia del Logo en que es un lenguaje visual. De hecho, los comandos están formados por bloques de colores que pueden ser entrelazados. El programa sale de la ejecución de estas secuencias de comandos conectados entre sí, como en un puzzle. Es un sistema atractivo que se parece un poco a Lego, donde las instrucciones que das están pegadas como ladrillos. Las articulaciones garantizan que las instrucciones se combinen sólo de forma legítima, protegiendo contra los errores ortográficos y sintácticos típicos y frecuentes que cualquier persona que escriba software en el modo de texto convencional encontraría. Muchos de estos idiomas han surgido, además de Scratch. Los más famosos son Snap! Alice, Blockly, Android App Inventor, entre otros.

Aquí mostramos una comparación entre un lenguaje basado en bloques y otro basado en texto. A la izquierda tenemos un código de Snap! y a la derecha uno de LibreLogo. Ambos producen el mismo resultado, es decir, el dibujo de un cuadrado de 100 puntos laterales de la pantalla. Es fácil encontrar las correspondencias, por ejemplo, el bloque "move 50 steps" corresponde a "FORWARD 50" y así sucesivamente. Hay algunas pequeñas diferencias, como por ejemplo para llegar al centro de la página antes de empezar el dibujo: se utilizan los bloques "set x a 0" y "set y a 0" en un lado y "HOME" en el otro.




```
CLEARSCREEN
PENDOWN
HOME
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
PENUP
HIDETURTLE
```

Aquí usamos Snap! pero el mismo código de Scratch se ve exactamente igual. Utilicé Snap! ya que tengo cierta preferencia por este idioma. Snap! representa una mejora de Scratch, que lo hace más similar a un lenguaje genérico, manteniendo al mismo tiempo la forma visual. Entre estas características se encuentra la posibilidad de guardar el código en un formato estándar (XML) que puede ser leído y editado por cualquier editor de texto. Para aquellos que están acostumbrados a trabajar con software, este es un elemento muy importante.

Una característica especial de Scratch es que ha creado una gran comunidad de intercambio de software. Esto sucedió gracias a que fue concebido como un servicio web, que permite la escritura de programas y la posibilidad de ejecutarlos, pero también la realización de un entorno social para compartir y remezclar los programas. Los lenguajes visuales también tienen inconvenientes. Son (aparentemente) fáciles, divertidos y coloridos, su efectividad parece garantizada pero la evidencia científica no es tan clara. De hecho, hay varios estudios que demuestran que los lenguajes visuales no facilitan tanto el aprendizaje de las lenguas "reales" (Weintrop, 2015a). Parece que son ventajosos para entender las construcciones más simples de la

programación, pero se estudia donde la comprensión profunda de lo que un algoritmo dado no muestra diferencias sustanciales entre los lenguajes visual y textual (Weintrop, 2015b). La investigación de Colleen Lewis es de particular interés. Comparó los resultados obtenidos con Logo y Scratch en una clase de niños de entre 10 y 12 años (Lewis, 2010). Los resultados mostraron que el aprendizaje de algunas construcciones de codificación específicas fue facilitado por Scratch pero, por otro lado, los niños mostraron un mayor nivel de autoestima cuando se les introdujo en la programación con Logo.

E incluso si en las fases iniciales los niños prefieren herramientas visuales, más tarde, una vez que prueban la programación textual convencional, pueden percibir los límites de la codificación visual:

- como limitando su creatividad porque menos poderosos
- porque sienten que el código visual es menos real: "si tienes que hacer algo real, nadie te pedirá nunca que lo codifiques con software educativo visual" (Weintrop, 2015c).

Biblio

- Weintrop D. (2015a) *Comparing block-based, text-based, and hybrid blocks/-text programming environments in introductory computer science classes*, ICER '15 Proceedings of the eleventh annual International Conference on International Computing Education Research, 1-4.
- Weintrop D., y Wilensky U. (2015b) *Bloquear o no bloquear, esa es la pregunta: las percepciones de los estudiantes sobre la programación basada en bloques*. Proceeding IDC '15 Proceedings of the 14th International Conference on Interaction Design and Children, 199-208.
- Weintrop David y Wilensky U. (2015c) Using commutative assessments to compare conceptual understanding in blocks based and text based programs, ICER '15 Proceedings of the eleventh annual International Conference on International Computing Education Research, 101-110.
- Lewis C.M.(2010). *Cómo el entorno de programación determina la percepción, el aprendizaje y los objetivos: Logo vs. Rasguño*. Proceeding SIGCSE '10 Proceedings of the 41st ACM technical symposium on Computer science education, 346-350.

Sobre la base de estas consideraciones decidimos centrarnos en el lenguaje de logotipos, como herramienta de introducción a la programación. Un buen número de versiones de Logotipos están disponibles hoy en día. Aquí nos centramos en una versión que está disponible por defecto en Writer, la aplicación de procesamiento de texto incluida en la suite LibreOffice office, análoga a la extensa suite MS Office. Existe otro proyecto similar llamado OpenOffice. Muchos se preguntan cuáles son las diferencias con LibreOffice. En la actualidad, LibreOffice es conveniente porque incorpora más funciones y se actualiza con mayor frecuencia. Los productos de

software como MS Office son "propietarios", es decir, la empresa los vende pero sin distribuir el código fuente de forma clara, según el modelo industrial convencional, con el que la propiedad intelectual se mantiene celosamente en secreto. LibreOffice es un software libre, y como tal es ideal para su uso en cualquier contexto educativo. En primer lugar, porque lleva un mensaje ético - recordemos aquí la definición de las cuatro libertades del Software Libre (lección 1, diapositiva 19):

- La libertad de ejecutar el programa como usted desee, para cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga su computación como usted desee (libertad 1). El acceso al código fuente es una condición previa para ello.
- La libertad de redistribuir copias para que pueda ayudar a su prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a otros (libertad 3). Al hacer esto, usted puede dar a toda la comunidad la oportunidad de beneficiarse de sus cambios. El acceso al código fuente es una condición previa para ello.

Donde enfatizamos las frases de orientación ética.

Cabe señalar -en este punto muchos están confundidos- que el software de *código abierto* es diferente del *software libre*, ya que falta el aspecto ético: El software de código abierto asume que el código fuente está disponible de forma clara, pero no menciona las cuatro libertades mencionadas anteriormente y, en particular, las dos especificaciones que caracterizan el valor ético del Software Libre: "para ayudar a otros" en la tercera libertad y "para que toda la comunidad se beneficie" en la cuarta libertad. El Software Libre es desarrollado por comunidades que pueden unirse en sociedades sin fines de lucro. El Código Abierto es desarrollado por actores económicos privados que se adhieren al paradigma de desarrollo compartido porque encaja bien en sus estrategias de marketing: hay empresas que desarrollan proyectos de Código Abierto junto a productos propietarios tradicionales porque lo encuentran conveniente para sus estrategias de marketing, lo cual es perfectamente correcto, pero es diferente, y bastante relevante en contextos educativos, por una serie de buenas razones. Proponiendo el uso de Software Libre en la escuela

- enseñas que hay gente que hace cosas porque creen que es bueno, y no necesariamente por dinero.
- enseñas un modelo de trabajo fuertemente colaborativo

- usted potencialmente empodera a los estudiantes porque algunos de ellos pueden descubrir que pueden contribuir a algún proyecto de software.
- deja claro el punto crucial de que descifrar la propiedad es malo porque es contra la ley, mientras que hay alternativas de software libre de alta calidad

LibreOffice y LibreLogo son software libre. LibreLogo es un plugin de LibreOffice y está integrado en la versión descargable por defecto. Sin embargo, cuando ejecute Libreoffice por primera vez no verá la barra de herramientas de LibreLogo. Para activarlo tienes que comprobar la opción Ver->Barras de herramientas->Logo en el menú, luego cerrar y relanzar el programa. Pero, ¿qué significa usar Logo en un programa de escritura como Writer, considerando que se trata de un procesador de textos normal, mientras que Logo es una especie de lenguaje de dibujo? Simple: Con la barra de herramientas de LibreLogo puede producir imágenes que se integran en el documento como si fueran importadas. Es una idea brillante, gracias a Németh László, que reprodujo las características del Logo en LibreOffice. En realidad, los mejoró aún más, aprovechando el lenguaje Python, con el que escribió el plugin. Usar LibreLogo es muy sencillo: se abre un documento en Writer, se escribe un código en el lenguaje del logotipo, como si se tratara de cualquier otro texto, y luego se ejecuta pulsando el botón correspondiente en la barra de herramientas de LibreLogo; si el código es correcto, la tortuga ejecuta el código dibujando una figura en el texto, justo en el centro de la página.

File Edit **View** Insert ...

Print Layout
Web Layout
Toolbars ->
Status Bar
...

3D-Settings
Align Objects
Bullets and numbering
Changes
Drawing
Drawing Objects Properties
Find
Fontwork
Form Controls
Form Design
Form navigation
Formatting
Frame
Logo
Media Playback
...

Una vez hecho esto, debe cerrar el programa y relanzarlo para ver, entre las otras barras de herramientas, también la de LibreLogo:

donde los iconos tienen los siguientes significados:

	FORWARD 10	Forward by 10 points (we will see the meaning of point successively)
	BACK 10	Back by 10 points
	LEFT 15	Left by -15 degree
	RIGHT 15	Right by 15 degree
		Executes the program. Starting with version 4.3, in a newly opened document it executes a standard sampled program.
		Stops the running program
	HOME	Brings the Turtle in its initial condition: at the center with nose up.
	CLEARSCREEN	Cancels all the graphics - leaving the text.
		Allows to write a command and run it at once.
		Adjust the whole text making it uppercase. At the same time, it translates the commands in the language of the document. Currently, the dictionary file for the Italian language is set up but the commands have to be written in English. I will fix this.

Al ejecutar un programa en LibreLogo, incrusta un objeto gráfico en su documento de LibreOffice. Los objetos gráficos producidos por LibreLogo son completamente similares a los producidos con las herramientas de escritura a mano disponibles en Writer, accesibles a través de la barra de herramientas especial, bajo el punto de menú Ver}. ->Barras de herramientas->Dibujo:

Como tal, los dibujos realizados con LibreLogo pueden ser movidos, copiados o guardados como cualquier otro objeto gráfico. Una cosa útil a entender es que tales objetos son a menudo en realidad una composición de objetos distintos. Haremos muchas de ellas en este manual. Para utilizarlos como un solo objeto, utilice la función de agrupación de la siguiente manera: primero, delimite la región que incluye los objetos a agrupar, seleccionando el puntero. en la barra de dibujo y luego delineando el rectángulo deseado con el ratón y manteniendo pulsado el botón izquierdo. Tenga en cuenta que el cursor del ratón debe tener la forma de una flecha

y no la típica que tiene al insertar texto, en forma de I mayúscula, porque aquí es donde se inserta el texto y no los gráficos. El hecho de que el cursor gráfico (y no textual) esté activo también se entiende por el hecho de que, al mismo tiempo, se activa otra barra de herramientas para controlar los gráficos:

Al seleccionar la región que contiene los objetos gráficos, se activan iconos en esta barra, incluido el icono de la función de agrupación, que está representada por un grupo de tres rectángulos. Al presionar esta tecla se agruparán todos los objetos gráficos de la región seleccionada en un único objeto gráfico que puede copiarse o guardarse en otro lugar.

Otro truco útil es"anclar" correctamente los gráficos al documento, donde tenemos que usarlos. La clave para determinar el anclaje en la barra gráfica habitual tiene la forma de un ancla. Haciendo clic en la flecha a la derecha del ancla, puede seleccionar cuatro tipos de ancla: 1) "en la página", 2) "en el párrafo", 3) "en el carácter" y 4) "como carácter". En el primer caso los gráficos están asociados a la página y no se mueven de ella, en el segundo a un párrafo, en el tercero a un carácter y en el cuarto caso se comporta como si fuera un carácter. Lo que es el anclaje más apropiado es algo que se aprende de la experiencia. La mayoría de los gráficos de este manual han sido anclados "al párrafo", excepto las pequeñas imágenes que están en línea con el texto, que, como en el anterior, están ancladas "como un carácter".

Estos se refieren a la gestión de los gráficos en Writer en general. Usando LibreLogo, la única diferencia es que los gráficos se producen a través de las instrucciones que ponemos en el código. LibreLogo coloca los gráficos en el centro de la primera página del documento, incluso si el texto del código se extiende en las páginas siguientes. Puede ocurrir que los gráficos se superpongan al texto del código mismo. A primera vista el resultado puede ser confuso y uno puede creer que algo malo está pasando. Nada de esto. Los gráficos se producen para ser utilizados en otro lugar. Se trata simplemente de seleccionarla, como acabamos de describir, y llevarla a otra parte, en una página limpia, simplemente para verla con claridad, o en algún otro documento en el que deba integrarse.

La historia de Marta - El teorema de “Turtle Total Trip”

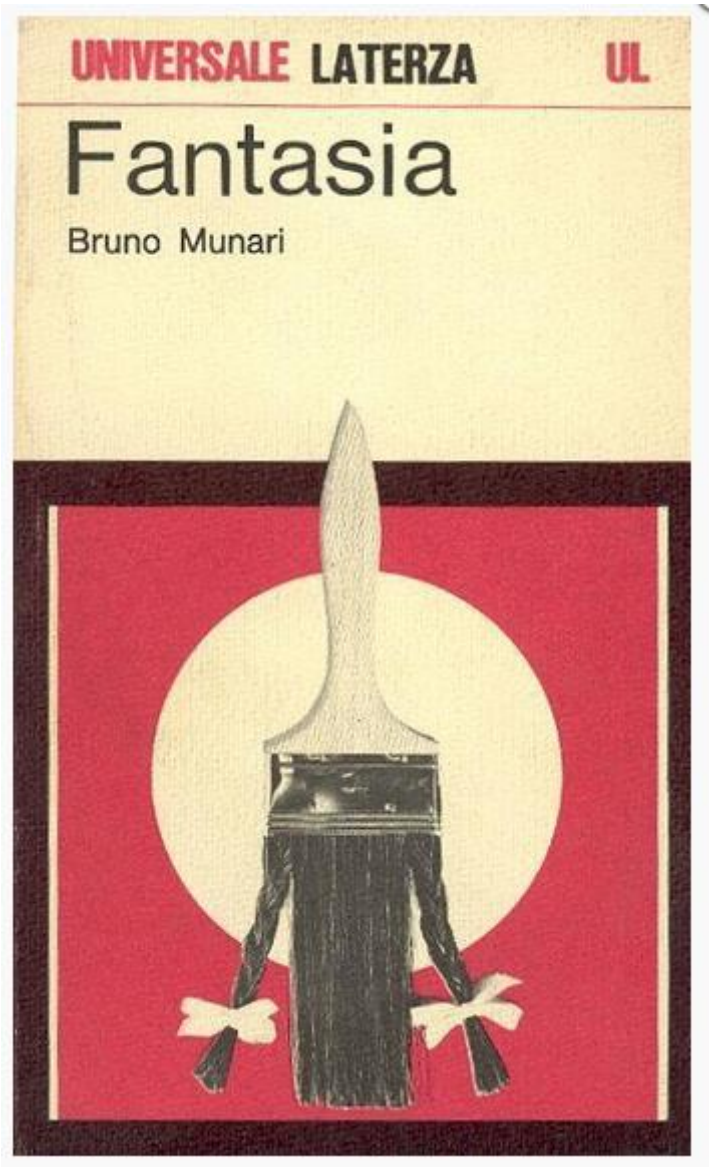
El mensaje de Marta

Hace un par de años, recibí el siguiente mensaje de Marta, una alumna de mi escuela primaria.

Les escribo porque he tratado de reflexionar sobre algunos temas, utilizando el Logo, y he desarrollado un breve recorrido con algunos "ejercicios creativos", como los llamé, posiblemente para ser realizados con niños, basados en la operación de repetición de elementos para formar nuevas imágenes. Me inspiré en el libro de Munari1, "Fantasía"; ¡qué bonito! A partir de ahí, desarrollé algunas reflexiones, incluso de carácter geométrico, que, debo ser sincero, me volvieron un poco "loco"; recogí algunas hipótesis y datos y, aunque no llegué a ninguna conclusión definitiva, encontré sin embargo algunos aspectos interesantes sobre los que quizás podría reflexionar.

1] Bruno Munari, *Fantasía*, Laterza, Bari, 1977.

Bruno Munari fue un autor famoso en el siglo XX en Italia en el diseño gráfico. Aportó contribuciones destacadas a las artes visuales, el diseño industrial, la escultura, la pintura, la literatura, el cine y el arte concreto.



Prólogo

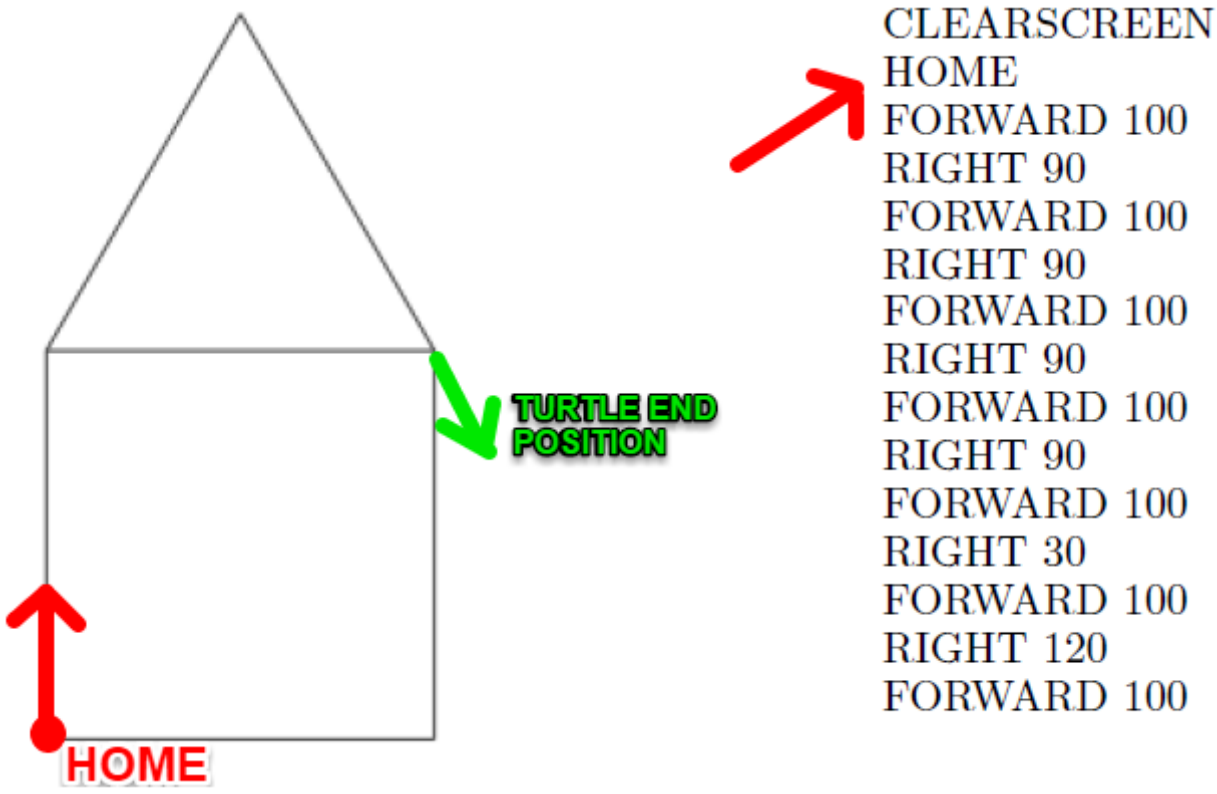
Vale la pena empezar con el prólogo original de Marta:

Nuestro viaje parte de la lectura de un hermoso texto de Bruno Munari, Fantasía. El texto es un elogio a la fantasía. Según Munari, una persona culta sin imaginación es como un diccionario, lleno de palabras pero sin poesía. Uno de los dispositivos mentales implicados en la fantasía, consiste en la repetición de unidades, sin variaciones. ¿Qué pasa si repetimos algunos objetos una y otra vez? Intentamos dar una respuesta, utilizando el Logo y explotando su gran potencial gráfico: "A Logo la precisión técnica, a nosotros la experimentación, la reflexión.... y la diversión! El viaje no se limita a dar libre fantasía, sino que también da lugar a algunas reflexiones geométricas y aritméticas, a través del método inductivo y según un enfoque científico (observación-hipótesis-verificación). Se supone que trabaja con una clase de niños de primaria.

Primer paso: la casa

El trabajo comienza con uno de los ejercicios básicos que habitualmente proponemos en el laboratorio: el dibujo de una casa compuesta por un rectángulo y un triángulo.

CLEARSCREENHOMEFORWARD
100 RIGHT 90FORWARD
100 RIGHT 90FORWARD
100 RIGHT 90FORWARD 100 RIGHT 90FORWARD 100 RIGHT 90FORWARD
100 RIGHT 30FORWARD
100 RIGHT 120FORWARD
100



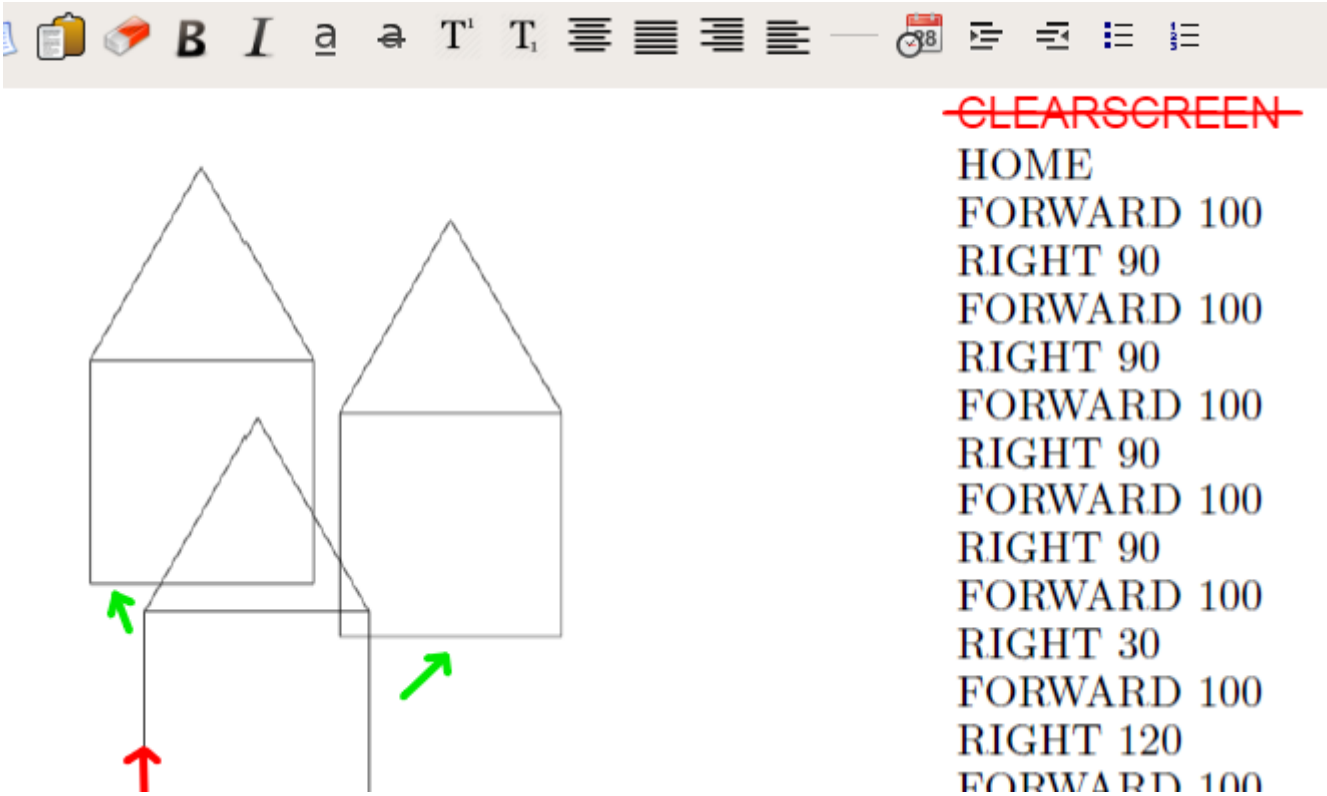
Conseguir un pueblo

Intente borrar el primer comando CLEARSCREEN, luego inicie el programa de nuevo y agin. Luego, si voy y arrastro el dibujo, descubro que las casas se superponen, arrastrándolas de aquí para allá, ¡tenemos una aldea!

HOMEFORWARD

100 DERECHA 90
ADELANTE 100 A LA DERECHA 90ADELANTE
100 A LA DERECHA 90ADELANTE 100 A LA DERECHA
90ADELANTE 100 A LA DERECHA
30ADELANTE
100 A LA DERECHA 120ADELANTE
100

Ejecutar repetidamente y arrastrar los resultados....

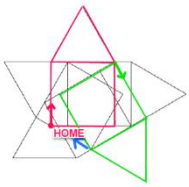


¿Una estrella? ¿O una flor?

Ahora vamos a deshacernos del comando HOME, también, y ejecutar el código repetidamente....

ADELANTE 100 A LA DERECHA 90ADELANTE
100 A LA DERECHA 90ADELANTE
100 A LA DERECHA 90DELANTE 100 A LA DERECHA
90DELANTE 100 A LA
DERECHA 30DELANTE
100 A LA DERECHA 120DELANTE
100

¿Qué es esto? ¿Qué es esto? Alguien verá una estrella aquí, alguien más una flor....



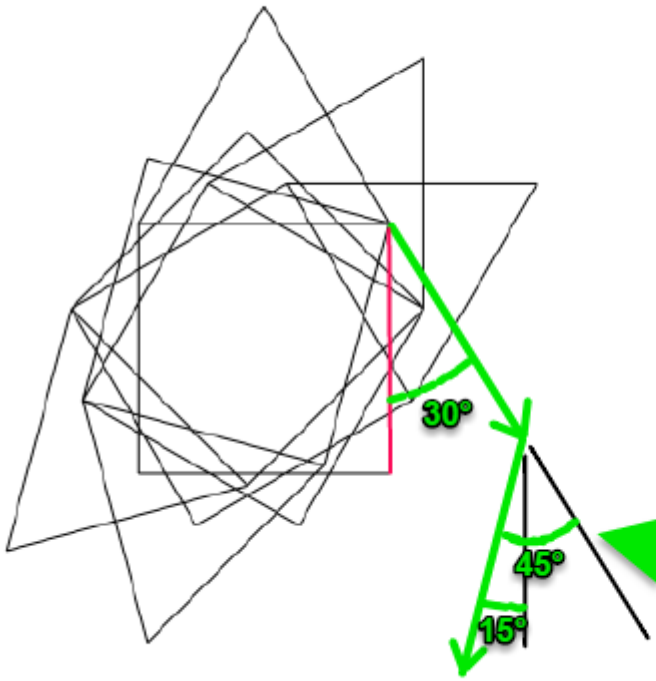
~~CLEARSCREEN~~
~~HOME~~
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100

Una curiosidad desbordante... ¿un cohete?

Aquí la curiosidad de Marta es desbordante: ¿y si añadimos un giro arbitrario final, después de cada dibujo de casa?

REPETIR 5[
ADELANTE 100 DERECHA 90ADELANTE
 100 DERECHA 90ADELANTE
 100 DERECHA 90ADELANTE 100 DERECHA 90DELANTE
 100 DERECHA 90DELANTE
 100 DERECHA 30DELANTE
 100 DERECHA 120DELANTE
 100DELANTE
 45
]

Aquí Marta decidió elegir como parámetro principal el ángulo de giro final expresado con respecto a la vertical.



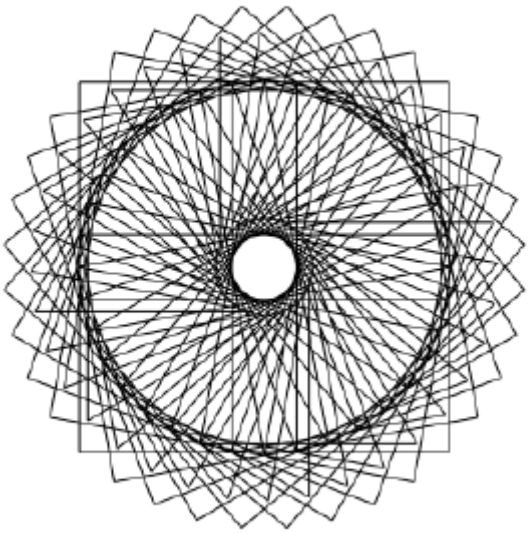

```
REPEAT 5 [
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT 45
]
```

Exagerando... el sol....

Luego, entre otras variaciones, trató de exagerar, porque a los niños les gusta hacerlo....

REPETIR 100[
ADELANTE 100 DERECHA 90ADELANTE
 100 DERECHA 90ADELANTE
 100 DERECHA 90ADELANTE 100 DERECHA 90ADELANTE
 100 DERECHA 90DELANTE
 100 DERECHA 30DELANTE
 100 DERECHA 120DELANTE
 100DELANTE
 10
]

¡Un sol!



```
REPEAT 100 [
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT 100
]
```

La conjetura de Marta

Al jugar con estos códigos, tarde o temprano, descubres que después de un cierto número de repeticiones, la figura está terminada y, si sigues aumentando las iteraciones, la tortuga simplemente repetirá el mismo dibujo una y otra vez. Marta observó que el número de iteraciones requeridas para completar la figura dependía del turno final añadido al final de cada dibujo de la casa, y construyó la siguiente tabla:

Giro final (expresado con respecto a la dirección vertical)	Número de ieraciones necesarias para completar la figura
15	24
30	12
60	3
70	36
100	9
120	6
240	6

Y ella concluyó:

"Desafortunadamente, no parece que surja ninguna relación. . . . Ahora detente y reflexiona un momento... ¿Podríamos hacer otras suposiciones?"

Una primera solución eurística

La primera respuesta que le dimos a Marta fue eurística:

donde N es el número requerido de iteraciones, θ es el ángulo de giro final con respecto a la vertical y "mod" es la operación *módulo*, que da el resto de la división entre los dos operandos, θ un 60 aquí. Eurístico significa un enfoque de la resolución de problemas que emplea un método práctico, no garantizado como óptimo, perfecto, lógico o racional, sino suficiente para alcanzar un objetivo inmediato. Sólo funciona para los casos que tuvo que resolver, pero sin el sustento de una teoría general subyacente, es decir, sin un verdadero conocimiento del fenómeno.

Más tarde, descubrimos otros casos que requerían cambiar esta fórmula, para incluirlos en la solución. La expresión se hizo más general, pero no se lograron avances significativos en la comprensión.

Una solución numérica

En un curso sucesivo, otro estudiante propuso aplicar una solución numérica, es decir:

¿por qué no dejar que la tortuga reconozca su estado inicial por sí misma?

La idea era correcta, desde un punto de vista lógico porque, si después de cierto número de iteraciones la tortuga asume el mismo estado que tenía al principio, a partir de ese momento no puede hacer otra cosa que repetir el mismo patrón. Aplicamos este enfoque probando insturcciones como la siguiente:

SI POS = P_START Y DIR = D_START [STOP]

Desde el punto de vista práctico, el enfoque no fue impecable, porque al comparar los números digitales producidos en cálculos largos puede haber errores de redondeo que requieren la aplicación de algunos trucos. Y, en todo caso, no ganamos nada con el conocimiento del fenómeno.

¿La solución matemática?

La respuesta a la conjetura de Marta se encuentra en el primer capítulo de Turtle Geometry (p. 4-54). La solución matemática a los problemas está hecha de teoremas y los teoremas tienen que ser probados. Esto no es fácil, es el trabajo del matemático. Pero no es necesario que sigamos todos los pasajes porque los conceptos pueden ser entendidos reuniendo las declaraciones de los teoremas relevantes. Si algunos lectores quieren profundizar en este tema, pueden consultar el capítulo anterior. Estaré encantado de compartir con ellos las partes relevantes de esta publicación.

En primer lugar, no debemos tomar uno de los objetos matemáticos básicos de la geometría de las tortugas, el procedimiento **POLY**:

```
TO POLY SIDE ANGLEREPEAT
[
FORWARD SIDERIGHT
 ANGLE
]
```

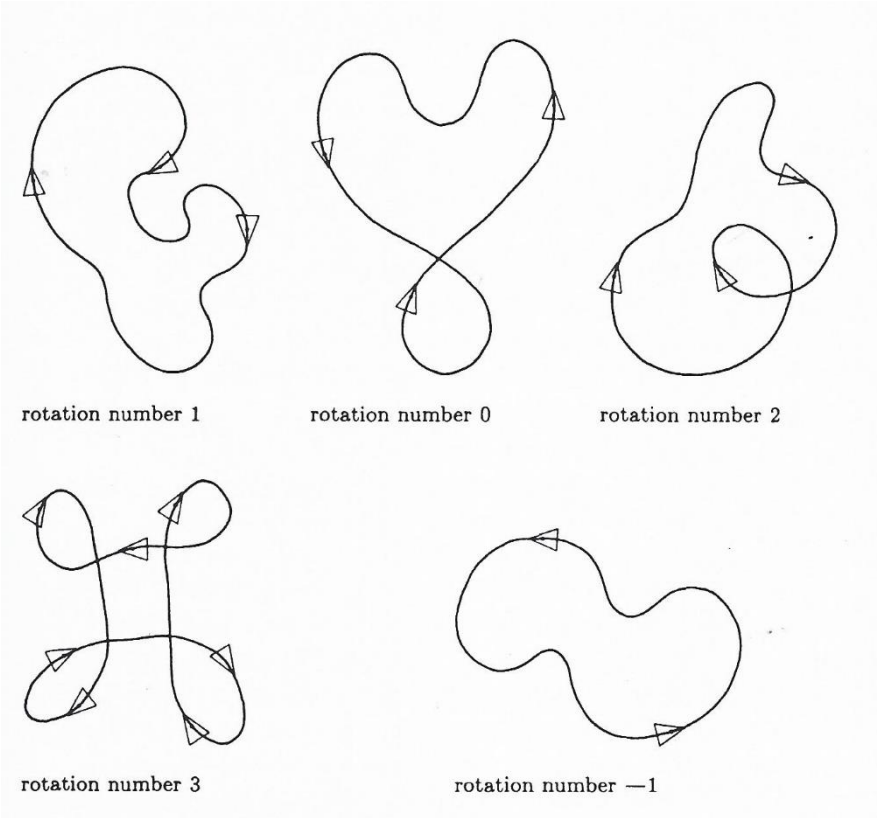
Ya hemos visto esta construcción varias veces. Ahora nos vamos a dar cuenta de que es realmente un objeto fundamental. Se llama **POLY**, porque es la construcción básica para dibujar polígonos regulares. Pero, tan pronto como empezamos a jugar libremente con los parámetros **SIDE** y **ANGLE**, descubrimos formas asombrosas.

El teorema del camino cerrado

El total de vueltas a lo largo de cualquier trayectoria cerrada es un múltiplo entero de 360°.

donde el múltiplo entero se denomina "número de rotación" del trayecto cerrado

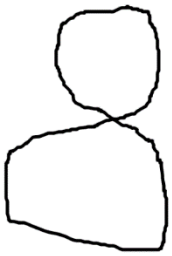
Conviene precisar que, aquí y en lo sucesivo, por *cierre* no se entiende sólo llegar al punto de partida, sino también llegar allí con la misma dirección inicial. Es decir, nos referimos a *alcanzar el mismo estado*.



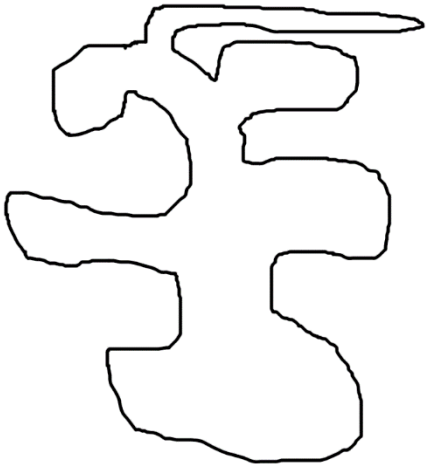
El teorema de la vía simple cerrada

El giro total en una trayectoria cerrada simple es de 360° (a la derecha o a la izquierda). Es decir, el número de rotación de cualquier trayectoria cerrada simple es ± 1

Por simple cerrado entendemos un camino que nunca se cruza, aunque sea muy complejo. Esta es la formulación que usualmente se cita como teorema del Viaje Total de la Tortuga. Es útil tener en cuenta cuando se dibujan polígonos regulares, por ejemplo.



This is not a simple closed path



This is a simple closed path

Teorema de cierre POLY

Una trayectoria trazada por el procedimiento POLY se cerrará precisamente cuando el giro total alcance un múltiplo de 360°.

Basta con ejecutar el código **POLY**, cambiando los parámetros **SIDE** y **ANGLE**. Intente también fijar pequeños valores de repetición en la sentencia REPEAT y auméntelos progresivamente, observe lo que sucede y reflexione sobre ello. Reportamos aquí el código POLY por conveniencia.

```
TO POLY SIDE ANGLEREPEAT
[
FORWARD SIDERIGHT
 ANGLE
]
```

El lema en bucle

Cualquier programa que sea sólo una repetición de un bucle básico de instrucciones de tortuga tiene precisamente la estructura de POLY con una entrada de ángulo igual a T, el giro total en el bucle.

Este es un resultado muy importante porque permite vincular los teoremas anteriores a la conjetura de Marta, abriendo así el camino a su solución. El dibujo de la casa más la rotación final utilizada por Marta en sus exploraciones constituye un "bucle básico de instrucciones de la tortuga" y, como tal, tendrá un valor T de giro total. Este lema establece que esto será equivalente a un programa POLY donde ANGLE=T, es decir

```
AL
 TREPEAT DEL LADO DEL POLÍGRAFO
[
FORWARD SIDERIGHT
 T
]
```

Entonces enfoquémonos en POLY

Si ***n*** es el número de iteraciones en **POLY** y ***A*** es el **ángulo de** entrada, el teorema de cierre de **POLY** dice que POLY se hace cuando la tortuga ha girado un múltiplo de 360°, es decir, cuando *n* vueltas de *A* cada una es un múltiplo de 360°:

$$nA = 360R$$

donde ***R*** es el número de rotación que hemos definido en la diapositiva 13. Reescribimos aquí el código de POLY con A para ANGULO y nombrando N el número de repeticiones.

```
TO POLY SIDE
 AREPEAT N[
FORWARD SIDERIGHT
 A
]
```

El lector notará que no se ha dicho nada sobre **SIDE**. En realidad, **SIDE** no juega ningún papel significativo en la forma de la figura. Sólo determina su escala. Pruebe....

Hacia la respuesta a la conjetura de Marta

Con la ecuación$nA = 360R$, que define un múltiplo común de ***A*** y 360, no cuenta toda la historia. ***R*** y ***n*** no son un número entero cualquiera que satisface la ecuación; son los más pequeños (positivos) que lo hacen, correspondiendo a la primera vez que el cambio de rumbo alcanza un múltiplo de 360. Es por eso que el número $nA = 360R$ se llama el múltiplo menos común de A y 360, denotado $\mathbf{LCM}(A, 360)$.

Por lo tanto, el número de iteraciones necesarias para cerrar la cifra viene dado por

$$n = \frac{\mathbf{LCM}(A, 360)}{A}$$

y el número de rotación viene dado por

$$R = \frac{\mathbf{LCM}(A, 360)}{360}$$

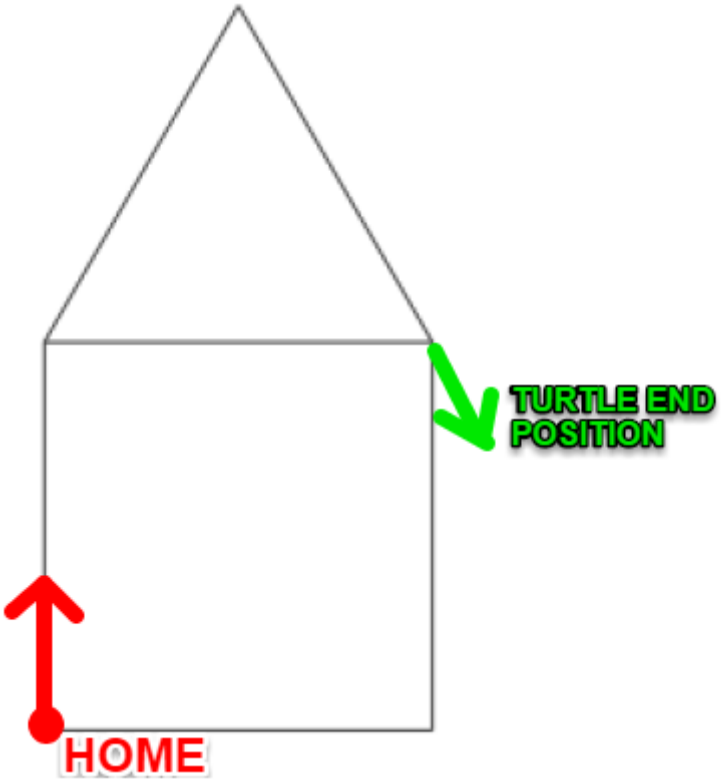
La respuesta a la conjetura de Marta

En este punto, al juntar este último resultado que es válido para los programas de POLY y el lema de bucle (diapositiva 16) que asocia bucles genéricos de comandos de tortuga, como el de las exploraciones de Marta, y los procedimientos de POLY, tenemos la respuesta a nuestra pregunta, dado que **T** es el giro total a lo largo del camino:

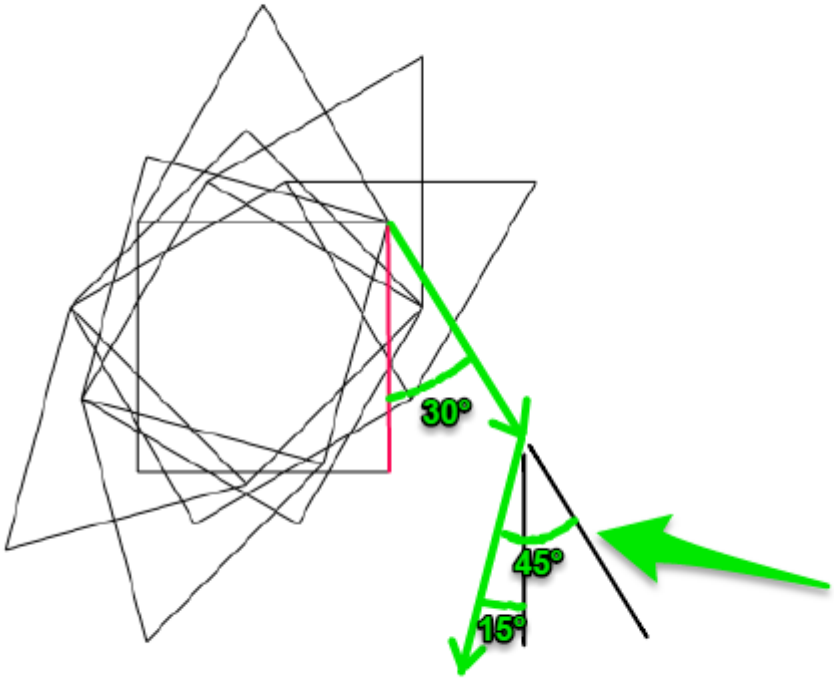
$$n = \frac{\mathbf{LCM}(T, 360)}{T}$$

La respuesta que hemos de dar a Marta es justamente eso. Y, lo que es importante, esta respuesta es válida para cualquier combinación de los parámetros. Es la respuesta matemática.

Para aplicar esta fórmula tenemos que calcular el giro total ***T***. El giro total para dibujar una casa simple, como en la diapositiva 4 (volvemos a poner aquí la cifra), es igual a 510°. Entonces, cuando tengas que añadir el último turno a este número. Por ejemplo, para el caso de la diapositiva 7 tenemos 510 + 45 = 555°.




```
CLEARSCREEN
HOME
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 30
FORWARD 100
RIGHT 120
FORWARD 100
```




```
REPEAT 5 [
  FORWARD 100
  RIGHT 90
  FORWARD 100
  RIGHT 90
  FORWARD 100
  RIGHT 90
  FORWARD 100
  RIGHT 90
  FORWARD 100
  RIGHT 30
  FORWARD 100
  RIGHT 120
  FORWARD 100
  RIGHT 45
]
```

Escribir un código para calcular el número de iteraciones

El Múltiple Mínimo Común (MCM) y el Divisor Más Grande Común (DPC) pueden calcularse de muchas maneras. Acabamos de elegir el más práctico en esta circunstancia. Los detalles sobre estos algoritmos están fuera de lugar aquí. Es suficiente saber que el MCV puede calcularse directamente, mientras que el DPC requiere un método de aproximación sucesiva. En realidad, esta complejidad se encuentra en el hecho de que el DPC que vamos a utilizar es un procedimiento recursivo. Por otro lado, el MSC es directo pero requiere el conocimiento del DPC. Así, en el código encontrará primero GCD, luego LCM y finalmente el procedimiento principal. Este puede ser llamado de la siguiente manera:

PRINT NITER FINAL_TURN

o

N = NITER FINAL_TURN

donde **FINAL_TURN** es el último turno de Marta, después de haber dibujado una sola casa.

Hay algo nuevo aquí, en lo que se refiere al uso de los comandos de LibreLogo. Centrémonos en la siguiente construcción: **N = NITER FINAL_TURN**. **NITER** es el nombre del procedimiento, **FINAL_TURN** es el argumento que necesita el procedimiento. Pero, ¿qué significa escribir **N = NITER**...? Aquí estamos usando otras posibles características de los procedimientos, que es devolver algún resultado numérico, en este caso el número de iteraciones. Este efecto se puede conseguir, insertando al final del procedimiento algo así como **OUTPUT N**, donde **N** se calculó de alguna manera en las instrucciones anteriores de **NITER**. Veamos el código en la siguiente diapositiva.

Un código LibreLogo para obtener la solución

```
A GCD A B

GLOBAL R

IF B = 0 [ R=A STOP ][ C=A%B GCD B C ]

FIN



A LCM A B

GLOBAL R, L
```

```
GCD A B

L = A*B/R

FIN


TO NITER FINAL_TURN

T = 90+90+90+90+90+30+120+30+FINAL_TURN

LCM 360 T

N = L/T

SALIDA N

FIN


IMPRIMIR NÚMERO 30
```

Cálculo de la rotación total en el código anterior

En el procedimiento **NITER** calculamos el total del giro **T de la** siguiente manera:

T = 90+90+90+90+90+30+120+30+FINAL_TURN

Los plazos de la suma **90+90+90+90+90+30+120** son las sucesivas desviaciones necesarias para construir la casa (ver figura). El último término es porque usamos la convención de usar la desviación final espresse con respecto a la vertical, para que podamos calcular directamente los valores de la tabla de Marta (diapositiva 9). Si prefiere utilizar el valor de la desviación final, este es simplemente dado por **FINAL_TURN + 30** - usted puede cambiar fácilmente el código.

Recapitulación

Aquellos que se sienten incómodos con las matemáticas pueden encontrar la segunda parte de esta lección un poco rara. Tratemos de retomar los puntos básicos.

1. Primero, informamos sobre la exploración de Marta, lo que la llevó a formular una conjetura significativa.
2. Esbozamos una solución heurística.
3. Luego una numérica.
4. Definimos lo que es un objeto POLY, es decir, un tipo específico de procedimiento.
5. El teorema del camino cerrado nos dio el marco general dentro del cual podíamos buscar la respuesta a la conjetura de Marta.
6. El teorema del camino cerrado simple fue un caso particular útil para los caminos que no se cruzan.
7. El teorema de cierre de POLY nos permitió aplicar el teorema del camino cerrado a las cifras generadas por POLY.
8. El lema en bucle nos permitió reconocer la relación entre las figuras de Marta y los objetos POLY, lo que nos llevó a la fórmula de solución.

Comandos más elegantes

Colores predefinidos

En LibreLogo hay 24 colores predefinidos:

Utilizando colores predefinidos

Si quieres dibujar con un lápiz de color puedes usar uno de los nombres anteriores así:

PENCOLOR "ROJO"
FORWARD 100

O si quieres rellenar una figura con un cierto color:

FILLCOLOR "GREEN"
REPEAT 3 [FORWARD 100 RIGHT 120]
FILL

Más colores

Actualmente los colores disponibles son muchos más. Pueden expresarse a través del llamado código RGB, que significa Rojo, Verde, Azul. En los gráficos por ordenador, los colores se pueden expresar como una combinación de tres colores principales, que son precisamente el rojo, el verde y el azul. Todos los demás pueden expresarse mezclando y dosificando adecuadamente estos tres colores básicos. Para ello, la intensidad de cada color se expresa con un número que va de 0 a 255 y se mezclan escribiendo por ejemplo[255,0,0] para el rojo,[0,255,0] para el verde o[0,0,255] para el azul. Todos los demás colores se obtienen variando los parámetros dentro de los corchetes. Con[0, 0, 0] se obtiene el negro y con[255, 255, 255, 255] el blanco. Este código se utiliza en cualquiera de los comandos que permiten especificar el color. Por ejemplo

PENCOLOR[45, 88, 200]

FILLCOLOR[255, 200, 100]

Juega un rato con estos números y verás qué colores salen.

Color transparente

Volvamos a los colores predefinidos por un momento, hay una rareza: hay dos elementos con un nombre diferente pero el color parece el mismo. ¿Los encontraste? En realidad difieren, pero debido a un cuarto atributo, que es la transparencia. Veamos mientras seguimos el juego. Por el bien de esta explicación, anticiparemos una nueva orden. En los ejemplos anteriores ya hemos dibujado un cuadrado, usando las instrucciones Adelante y Derecha, convenientemente combinadas. En realidad, para dibujar las principales formas geométricas, en Logo también hay instrucciones preestablecidas, que pueden ayudarnos a escribir el código sintéticamente. La plaza es una de ellas. Hagamos un cuadrado lateral de 50 mm:

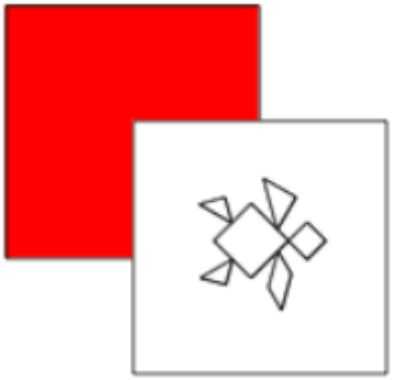
CUADRADO(50mm)

De esta manera, la tortuga dibuja un cuadrado y luego se coloca en el centro con la cabeza hacia arriba. Por cierto, no elegimos el color, pero salió "automáticamente", por defecto.

Colores opacos

Usemos esta instrucción, primero para dibujar una casilla que colorearemos de rojo, luego nos moveremos un poco hacia abajo, para dibujar una segunda casilla, haciendo que se superponga un poco a la anterior, luego la colorearemos de blanco.

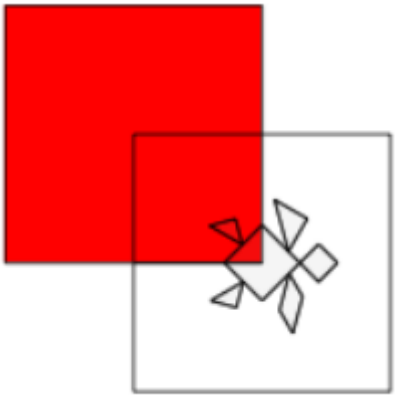
CLEARSCREENHOMEFILLCOLOR
[255, 0, 0] ; esto es rojo, equivalente a **FILLCOLOR["rojo"].**
CUADRADO(50mm)
DELANTERO -25mmDERECHO
90FORO
25mmFILLCOLOR
[255, 255, 255, 255] ; este es blanco (opaco), equivalente a **FILLCOLOR["blanco"].**
CUADRADO(50mm)



Colores transparentes

Bien, ahora el cuadrado blanco se superpone al rojo. Un resultado previsible: los colores se superponen porque son opacos. Un pintor diría que "cubren". En realidad, con los códigos RGB en Logo se puede asignar un cuarto parámetro: la "transparencia", que se puede dar a un determinado color. Con un valor 0 se obtiene una opacidad total, mientras que con el valor 255 se obtiene una transparencia total. Con valores entre 0 y 255 se pueden obtener diferentes niveles de transparencia. Por lo tanto, intentemos que el cuadrado blanco sea transparente, añadiendo el cuarto parámetro igual a 255:
FILLCOLOR[255, 0, 0] ; esto es rojo, equivalente a **FILLCOLOR["rojo"].**
SQUARE(50mm)
FORWARD -25mmRIGHT
90FORWARD
25mmFILLCOLOR
[255, 255, 255, 255, 255] ; esto es blanco (transparente), equivalente a **FILLCOLOR["invisible"].**
CUADRADO(50mm)

Pruebe con **FILLCOLOR[255, 255, 255, 255, 125]** para ver qué pasa....



Color invisible

El cuadrado blanco se ha vuelto transparente. Si en lugar de **FILLCOLOR[255, 255, 255, 255, 255] hubiéramos** utilizado **FILLCOLOR "invisible"**, habríamos obtenido el mismo resultado. Esa es la diferencia entre "blanco" e "invisible", en la tabla de colores que vimos al principio.

Una última observación. Mirando de cerca, se puede ver que en estas fotos nuestra tortuga tiene diferentes colores. De hecho, la tortuga está coloreada con el mismo color que seleccionamos para llenar la última foto, quizás con una transparencia diferente. Después de dibujar el cuadrado verde claro, la tortuga salió un poco más oscura. Después del cuadrado blanco, salió gris claro y después del color invisible salió gris oscuro. Este comportamiento no afecta a los resultados que queremos obtener porque, si queremos utilizar los gráficos que estamos produciendo de alguna manera, al final sólo tendremos que añadir la instrucción **HIDETURTLE**.

Hay muchos más efectos que se pueden aplicar a los colores, como los diferentes tipos de degradados. Por ejemplo, eche un vistazo a las opciones de los comandos **FILLCOLORAND FILLTRANSPARENCY** en el manual de comandos de LibreLogo.

Cuadrado y círculo (el camino disponible)

Ahora vamos a ver más comandos disponibles para hacer gráficos de logotipos.

Por favor, recuerde: pueden ser convenientes si está usando LibreLogo como una herramienta de gráficos, lo cual es perfectamente correcto. Sin embargo, si está pensando en utilizarlo en su aula k1-5 (e incluso más tarde) concéntrese en lo que mostramos en la lección 5.

Dicho esto, los comandos para dibujar cuadrados y círculos son así de simples:

CUADRADO 100CIRCULO
100

Por cierto, en el estado de ánimo de "construir tus propios bloques" podrías construir tus propios comandos SQUARE y CIRCLE (todavía necesitamos el concepto de "variable" pero vamos a descubrirlo pronto) pero tienes que evitar estos nombres predefinidos. Por lo tanto, en general, compruebe siempre en el

manual la existencia de un comando estándar con un nombre, antes de utilizarlo para sus procedimientos.

Prueba algo por ti mismo

En la diapositiva anterior hemos visto cómo dibujar dos imágenes en secuencia tiene el efecto de superponerlas y hacer coincidir el centro de simetría, y cómo la tortuga siempre se reposiciona en ese centro. También es evidente cómo el argumento (por "argumento" se entiende el valor que un comando requiere para ser operado. Puede haber más de un argumento - veremos algunos ejemplos) del comando **CUADRATURA** representa el lado del cuadrado y el argumento del comando **CÍRCULO** representa el diámetro del círculo que queremos construir.

Intenta practicar con los comandos **CUADRADO** y **CÍRCULO**, por ejemplo, construyendo esta locomotora. Puedes practicar a respetar las proporciones dadas, como en el ejemplo, donde el lado del cuadrado más grande es dos o tres veces más grande que los cuadrados más pequeños y el diámetro de los círculos es la misma longitud del lado del cuadrado más pequeño. Encontrará una solución ("a", no "la"...!) en la siguiente diapositiva, pero primero trate de resolverlo por su cuenta.

Una solución para la locomotora

Una cosa importante a entender con el código es que la misma meta puede ser alcanzada de muchas maneras diferentes. No existe un criterio absoluto para establecer el mejor procedimiento. Por lo tanto, no existe una "respuesta correcta". Un procedimiento puede ser mejor que otro desde un cierto punto de vista: claridad del código escrito, rapidez de ejecución, memoria total utilizada desde el código, gestión de posibles recursos, etc. Puede suceder que un código muy bueno bajo uno de estos términos pueda resultar muy malo para otro.

```
FILLCOLOR "rojo"
 CUADRADO 60PENÚMETRO
 15 DERECHO 90 HACIA ADELANTE
 45 IZQUIERDA
 90
 PENÚMETRO 30PENÚMETRO
 25 P
ENÚMETRO
 "NEGRO"
 CUADRADO 20PENÚMETRO
 40 PENÚMETRO 40 PENÚMETRO
 20PENÚMETRO
 IZQUIERDO 90 HACIA ADELANTE 60
 PENÚMETRO 20HIDETURTLE
```

El rectángulo

El cuadrado y el círculo sólo necesitan un parámetro para ser ejecutados. En cambio, el rectángulo y la elipse necesitan dos parámetros. Para el rectángulo, los dos parámetros son la longitud del lado largo y el lado corto. La instrucción para construir el rectángulo es la siguiente:

RECTÁNGULO[60,40]

Este comando produce un rectángulo de 60 puntos y otro de 40. En comparación con los casos del cuadrado y del círculo, este comando tiene otra particularidad: en este caso, para dar los dos parámetros necesarios, recurrimos a la escritura**[60,40]**. Se trata de una "lista" de valores. Es una forma de considerar un montón de valores como un todo, una lista de hecho, y la forma de obtener este resultado es listar los valores, dividirlos con comas y cerrarlos entre corchetes. Las listas pueden ser útiles en varias circunstancias, no sólo en este caso, pero veremos cómo más adelante. Intenta hacer una figura como ésta.

Una solución para la serie de rectángulos

RECTANGULO[40mm, 20mm]
PENDIENTE HACIA LA IZQUIERDA 2,5mm HACIA LA IZQUIERDA 90 HACIA LA DERECHA 90 HACIA LA PENDIENTE
2,5mm HACIA LA IZQUIERDA 90 HACIA LA IZQUIERDA
2,
5mm HACIA LA
IZQUIERDA 90 HACIA LA DERECHA 90 HACIA LA IZQUIERDA 90 HACIA LA IZQUIERDA 90 HACIA LA IZQUIERDA 90 HACIA LA IZQUIERDA
25mm HACIA LA
DERECHA 90 HACIA LA DERECHA 90 HACIA LA DERECHA 90 HACIA LA PENDA 90 HACIA LA DERECHA
25mm HACIA LA RIELA

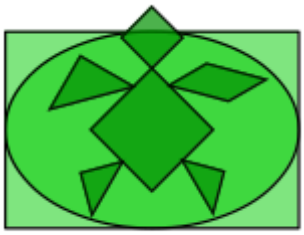
La elipse

Otra imagen que puede ser útil es la elipse. En LibreLogo la elipse se dibuja con el comando

ELIPSE[60, 40]

Obviamente la elipse no tiene un diámetro fijo, como el círculo. Por este motivo, para definirlo se necesitan dos parámetros: los dos ejes, mayor y menor. En el ejemplo la elipse tiene el eje mayor igual a 60 puntos y el menor igual a 40 puntos. Es posible combinar varias formas y ajustar sus parámetros para obtener una variedad de efectos. Por ejemplo, un círculo inscrito en un cuadrado puede obtenerse también partiendo de las instrucciones para dibujar rectángulos y elipses:

RECTÁNGULO[60, 40]
ELIPSE[60, 40]



Trata de dibujar una figura como ésta.

Una solución para las elipses

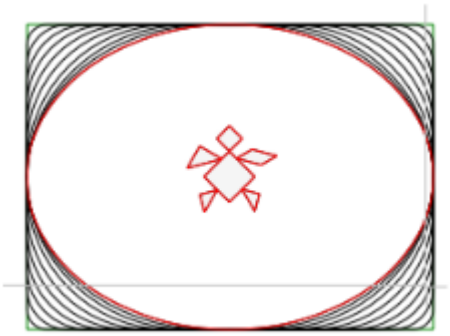
ELIPSE[120, 80]
PENUP ADELANTE 10PENDOWNELLIPSE

[90, 60]
PENUP ADELANTE 10PENDOWNELLIPSE
[60, 40]
PENUP ADELANTE 10PENDOWNELLIPSE
[30, 20]
PENUP ADELANTE 10PENDOWN

Parámetros para rectángulos

Los comandos RECTANGULO y ELLIPSE aceptan algunos parámetros adicionales. Por ejemplo, se pueden producir rectángulos con vértices redondeados, mediante un tercer parámetro, más allá de las medidas de los dos lados. Por ejemplo....

FILLCOLOR "invisible"
PENCOLOR "verde"
RECTANGULO[200, 150, 0]
PENCOLOR "negro"
RECTANGULO[200, 150, 10]
RECTANGULO[200, 150, 20]
RECTANGULO[200,
150
, 30]RECTANGULO[200, 150, 150
, 40]
RECTÁNGULO[200, 150, 50] RECTÁNGULO[200,
150, 60] RECTÁNGULO[200,
150, 150, 70] RECTÁNGULO[200
, 150, 80] RECTÁNGULO[200,
150, 90]
RECTÁNGULO[200, 150, 100] RECTÁNGULO[200, 150, 100]
PENCOLOR "rojo"
ELLIPSE[200, 150].



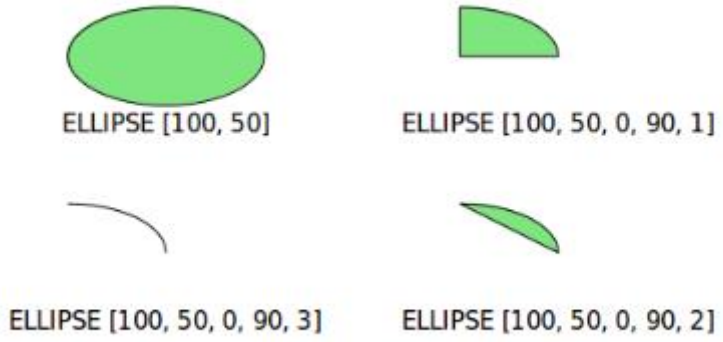
De esta manera hemos aprendido a controlar los rectángulos redondeados. También nos dimos cuenta de que, jugando con el tercer

parámetro, podemos pasar del caso extremo de un rectángulo normal a la elipse real.

Parámetros para elipses

Veamos ahora los parámetros del comando ELLIPSE.

Los parámetros adicionales son tres y todos sirven para dibujar sólo una parte de la elipse. Los dos primeros representan el ángulo inicial y final que limita un sector, expresado en grados. En el ejemplo, habiendo escogido 0 y 90 establecimos para dibujar el primer cuadrante de la elipse. El cuarto parámetro determina si se debe dibujar un sector de la elipse (1), un segmento de la elipse (2) o sólo un arco (3).



El espacio de la página

Para entender cómo está organizado el espacio de la página en LibreLogo, es conveniente pensar en términos de geometría cartesiana. Pero en cualquier caso, como se trata de una tortuga, necesitamos un par de coordenadas para la posición y un ángulo para la dirección. Para entender cómo funcionan estas referencias, vamos a presentar y utilizar dos nuevas instrucciones a la vez: **POSICIÓN** y **RÚBRICA**. Además, la instrucción **PRINT** es útil para conocer el valor actual de la posición y de la dirección. Estos dos comandos, **POSITION** y **HEADING**, pueden utilizarse con y sin parámetros. Cuando se utilizan sin parámetros, proporcionan los valores actuales. De hecho, si abro un nuevo documento en Writer y ejecuto el comando **PRINT POSITION**, obtendrás lo siguiente en un mensaje: **297.4393700787401, 420.71811023622047]** o, utilizando una cantidad razonable de dígitos,**[297.4, 420.7]**. Estas son las coordenadas de la tortuga cuando se coloca en el centro de la página siguiendo un comando **HOME**.

Para conocer las dimensiones del espacio de la página se utiliza el comando **PAGESIZE**, con **PRINT PAGESIZE** obtenemos**[595.303937007874, 841.8897637795275]**, es decir**[595.3, 841.8]**. Estos números representan las

dimensiones de una hoja A4 expresadas en "puntos" a la densidad de 72 DPI: **595,3 / 72 x 25,4 = 210 mm** y **841,8 / 72 x 25,4 = 297 mm**, ya que **1 pulgada = 25,4 mm**.

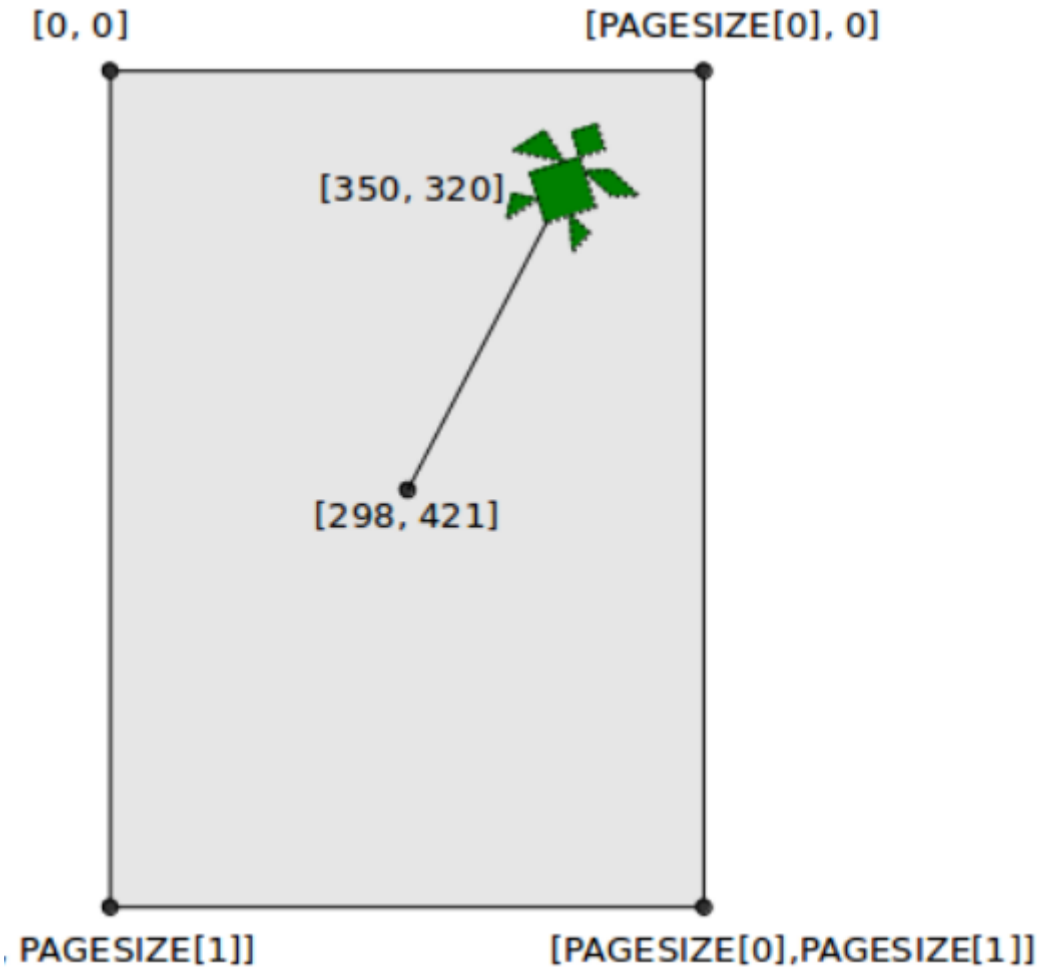
Tabla de espacio de página

Cada vez que queremos utilizar posiciones absolutas podemos referirnos a este esquema que nos da la orientación del sistema de referencia en la hoja y sus dimensiones. En la siguiente tabla podemos ver los valores numéricos equivalentes:

[0, 0]	**[0, 0]**
TAMAÑO DE PÁGINA[0], 0]	**[596, 0]**
0, TAMAÑO DE PÁGINA[1]]	**[0, 842]**
[TAMAÑO DE PÁGINA[0], TAMAÑO DE PÁGINA[1]]	**[596, 842]**

PAGESIZE es una variable tratada como una constante dentro de LibreLogo porque contiene las dimensiones de la página. Además es ese tipo particular de variable que se llama vector, porque está compuesta por más elementos, precisamente dos, las dos dimensiones de la página: **PAGESIZE[0]** es el ancho y **PAGESIZE[1]** el largo.

Ahora, ¿cómo podemos enviar la tortuga a una posición determinada? Supongamos que queremos enviar la tortuga desde su posición de origen a las coordenadas**[350,320]**. El comando es **POSICIÓN[350,320]**.... (la escala del tamaño de la página no se respeta en la figura)

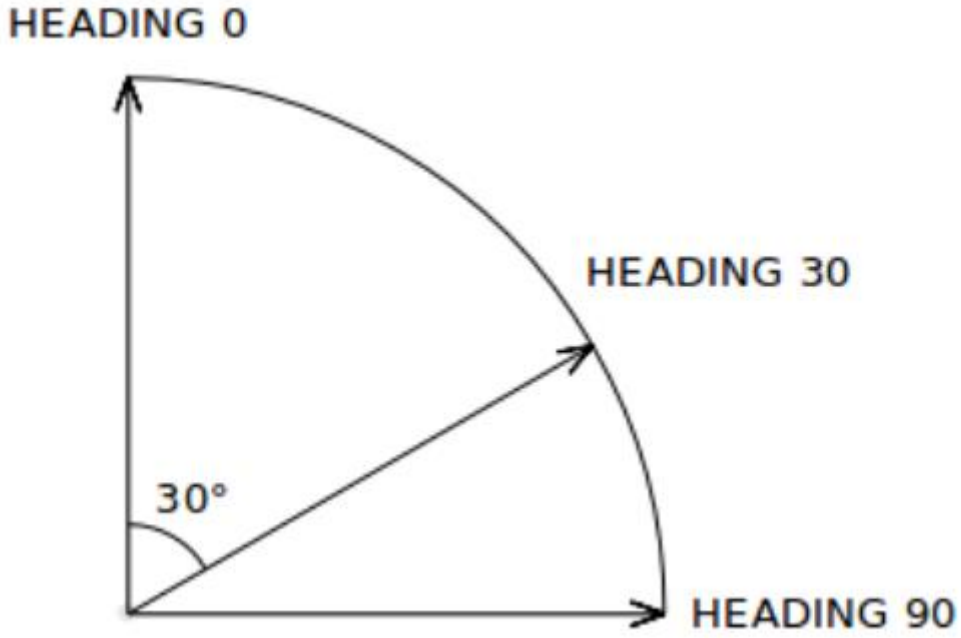


RÚBRICA

Así como podemos usar el comando **POSITION** para controlar la posición, también podemos usar el comando **HEADING** para controlar la dirección hacia donde se dirige la tortuga. Con **HEADING**, sin ningún parámetro, obtenemos la posición actual.

Usando otro parámetro, por ejemplo el **HEADING[30]**, hacemos que la tortuga gire 30° .

Cuando abrimos un nuevo documento, o después de la instrucción **HOME**, la tortuga apunta siempre hacia la parte superior de la hoja, y esta dirección corresponde a 0°.



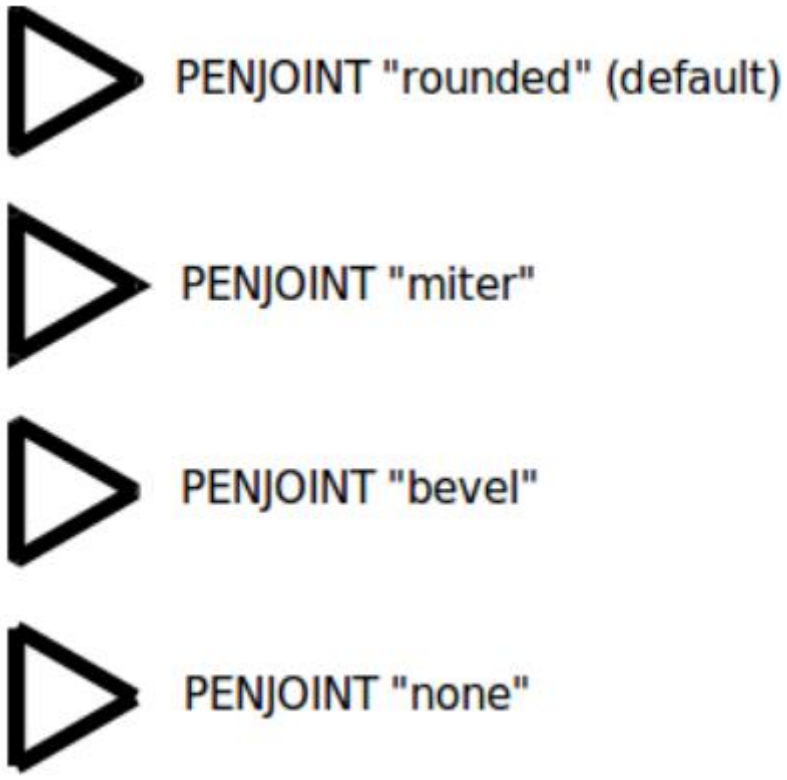
ANCHO DE PENSIÓN

PENJOINT (unión de segmentos)

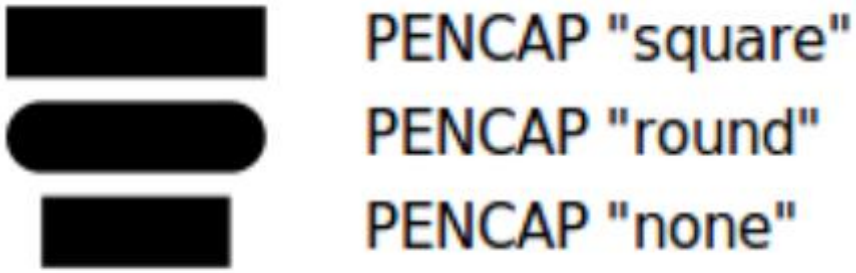
Dibujemos un triángulo, por ejemplo, de la siguiente manera:

PENWIDTH 5DELANTERA
40 DERECHA 120DELANTERA
40 DERECHA 120DELANTERA
40 DERECHA 120HIDETURTLE

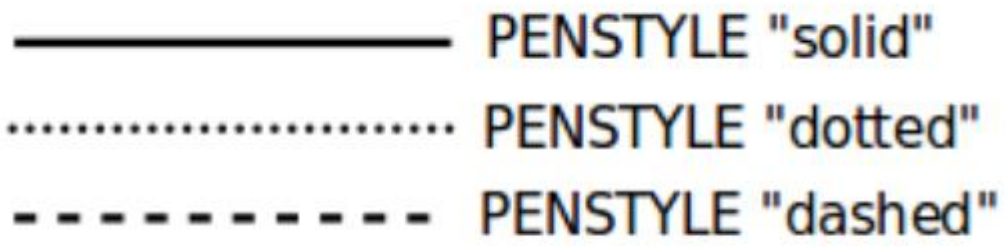
Podemos alterar la superposición de los vértices dando la instrucción PENJOINT y un argumento apropiado.



PENCAP (fin de segmentos)



PENSTYLE (punteado de segmentos)



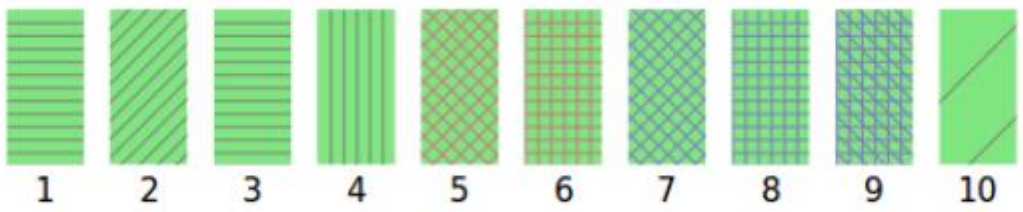
Dashing de estilo pluma

También es posible determinar la elegancia.

Estas son las reglas
:- Parámetro 1: número de puntos-
Parámetro 2: longitud de los puntos-
Parámetro 3: número de caracteres- Parámetro
4: longitud de los caracteres- Parámetro
5: longitud de los espacios-
Parámetro 6: opcional, si vale 2 entonces los rectángulos son forzados a los cuadrados

El ejemplo de la figura se obtiene con **PENSTYLE[3, 1mm, 2, 4mm, 1mm].**

FILLSTYLE (sombreado cruzado)



Parámetros FILLSTYLE

FILLSTYLE 0 ; relleno sin trampillas (por defecto)
FILLSTYLE 1 ; trampillas simples negras (horizontal)

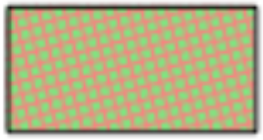
FILLSTYLE 2; trampillas simples negras (45 grados)
FILLSTYLE 3 ; ventanillas simples negras (-45 grados)
FILLSTYLE 4 ; ventanillas simples negras (verticales)
FILLSTYLE 5 ; ventanillas cruzadas rojas (45 grados)
FILLSTYLE 6 ; ventanillas cruzadas rojas (0 grados)FILLSTYLE
7 ; ventanillas cruzadas azules (45 grados)FILLSTYLE
8 ; ventanillas cruzadas azules (0 grados)
FILLSTYLE 9 ; ventanillas
simples azules (
0 grados)
FILLSTYLE 9
; ventanillas simples
cruzadas tridimensionales triples azules (
45 grados)

Los sombreados personalizados se pueden especificar mediante una lista con los siguientes argumentos:

estilo (1 = simple, 2 = doble, 3 = triple eclosión)
grado de coloración

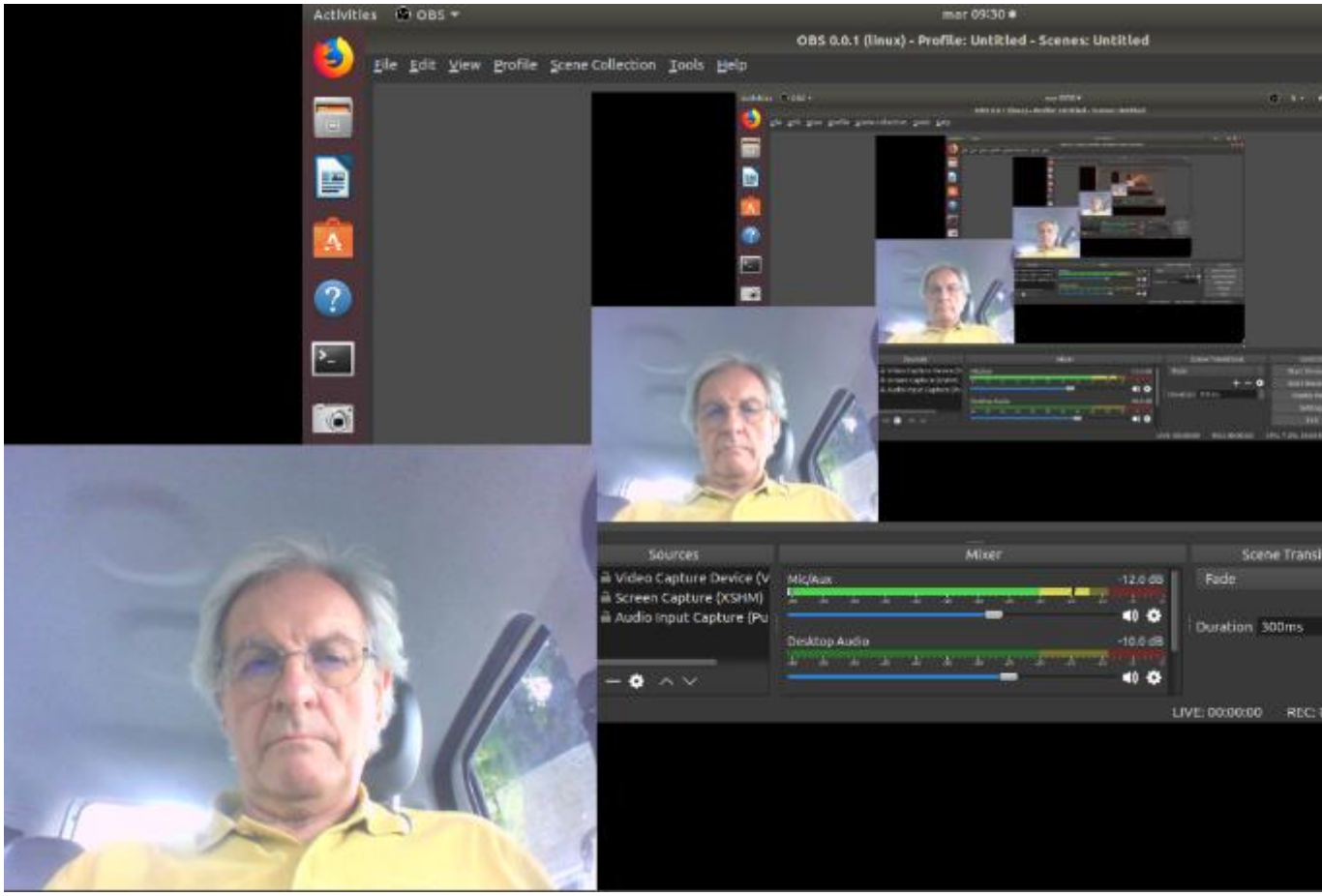
FILLSTYLE[2, "verde", 3pt, 15°]; escotillas cruzadas verdes (15 grados)

FILLSTYLE [2, "red ", 3pt, 15°]



Recursividad, crecimiento y fractales

La magia de dos espejos.... o de un software de screencasting



Desde espejos y screencasting hasta software

Todo el mundo sabe cómo dos espejos opuestos generan una sorprendente fuga de imágenes... El espejo número 1 sólo puede hacer una cosa: reproducir la escena delante de él. Incluso el espejo número 2 puede hacer sólo lo mismo, pero al hacerlo también reproduce el espejo número 1, incluyendo la escena que contiene, que a su vez reproduce la escena en el espejo número 2 y así sucesivamente, ad libitum. El mismo efecto se puede ver en un software de screencasting que reproduce la pantalla en su ventana, pero aquí se ve otra versión más pequeña de la pantalla con la ventana dentro, y así sucesivamente. Es un fenómeno que golpea

porque nos permite asomarnos al infinito, normalmente inaccesible a la experiencia humana. Esa es la recursión.

Es muy fácil crear un esquema recursivo a través del software:

1 A RECURSIÓN2
RECURSIÓN3
FIN

4 RECURSIÓN

¿Qué está pasando aquí? Recuerde: con la construcción TO...END usted define el procedimiento e (instrucciones 1-3) y luego lo utiliza (4).

Trate de averiguar...

Manos a la obra

En este fragmento de código la Tortuga está ejecutando sólo un comando: RECURSIÓN. Como no existe tal comando en el Logo, ha sido necesario definirlo con un procedimiento, pero aquí no ocurre nada, excepto llamar al comando RECURSIÓN. Sucede como en los espejos: cuando se ejecuta el comando RECURSION, lo que sucede es que RECURSION se llama de nuevo y así sucesivamente. En realidad, si intentas escribir este código en LibreOffice y lo ejecutas no pasará nada excepto que, al cabo de un tiempo, aparecerá un cuadro de mensaje diciéndote "Program terminated: maximum recursion depth (1000) exceeded". Esta es una especie de medida de seguridad, ya que los programas recursivos pueden atascar el ordenador si no se proporciona un criterio de parada. Cada vez que se llama a un procedimiento, el sistema asigna la memoria necesaria para sus actividades. Si deja que la máquina continúe con este proceso, se reclamará una cantidad infinita de memoria, lo que no es el caso de su ordenador, por supuesto.

Hagamos que este tonto programa sea un poco más interesante.

AL CÍRCULO DE RECURSIÓN
DRECURSIÓN
D+1 FIN

RECURSIÓN 1

Esto es más divertido, pruébalo: deberías tener un globo inflable.

Tratar de seguir el proceso de recursividad

En esta versión hemos proporcionado al procedimiento RECURSION un argumento, D, luego hemos añadido sólo una instrucción más, CIRCULO D, y hemos cambiado la llamada recursiva como sigue: RECURSIÓN D+1. Luego se le pide a la tortuga que comience con la RECURSIÓN. Así, la primera vez que se llama RECURSIÓN, se llevan a cabo las siguientes acciones....

Dentro de la RECURSIÓN:

- el valor 1 se da a la variable D
- se dibuja un círculo de diámetro 1
- La RECURSIÓN se llama de nuevo pero con el argumento D+1, es decir, 2, causando, a su vez, lo siguiente.....

Dentro de la RECURSIÓN dentro de la RECURSIÓN:

- el valor 2 se da a la variable D
- se dibuja un círculo de diámetro 2
- La RECURSIÓN se llama de nuevo pero con el argumento D+1, es decir, 3, causando, a su vez, lo siguiente.....

Dentro de la RECURSIÓN dentro de la RECURSIÓN dentro de la RECURSIÓN:

- el valor 3 se da a la variable
- se dibuja un círculo de diámetro 3
- La RECURSIÓN se llama de nuevo pero con el argumento D+1, es decir, 4, causando, a su vez, lo siguiente.....

Y así sucesivamente....

Otro ejemplo

Intenta adivinar lo que está haciendo este programa, luego intenta ejecutarlo

```
A LA RECURSIÓN D
 CIF D + C > 70 [ C = -1 ]IF
 D + C < 50 [ C = 1 ]CÍRCULO
 D + CRECURSIÓN
 D+C CEND

RECURSIÓN 50 1
```

Y otro más

Y ahora intenta averiguar qué está pasando aquí, antes de intentar ejecutarlo....

```
TO RECURSION NLABEL
 NPENUP
 FORWARD 10 PENDOWNIF
 N > 20 [ STOP ]
RECURSIÓN N+1 FIN

RECURSIÓN 1
```

Recursividad y repetición

Algunos lectores se preguntarán cuál es la diferencia entre recursión y repetición. De hecho, los ejemplos que hemos visto también se pueden hacer con un REPEAT - el lector puede hacer esto como un ejercicio.

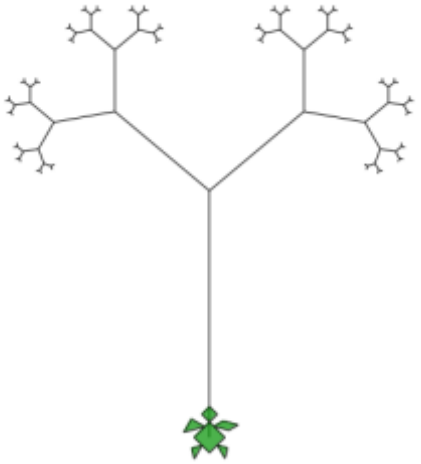
A su vez, el efecto de un ciclo de repetición también puede lograrse mediante la recursión, colocando las operaciones a realizar en un procedimiento e insertando la llamada recursiva como última instrucción.

Pero, ¿quién nos impide hacer la llamada recursiva en otras partes del procedimiento, o incluso hacer más de una? Ahí es donde las cosas se ponen interesantes....

```
AL ÁRBOL LLIF
 LL < 2 [ STOP ]
ADELANTE LL IZQUIERDA 50
ÁRBOL LL/2RIGHT
 100
ÁRBOL LL/2LEFT
 50 ATRÁS LLEND

ÁRBOL 200
```

Estamos entrando en el increíble mundo de los fractales....

El código anterior produce un árbol fractal (tree-simple.odt)



Profundización en el procedimiento TREE

1 AL **ÁRBOL** LL2
SI LL < 2 [PARADA]3
ADELANTE LL IZQUIERDO 504 **ÁRBOL LL/25**
DERECHO 1006
ÁRBOL **LL/27**
IZQUIERDO 50 ATRÁS LL8
FIN

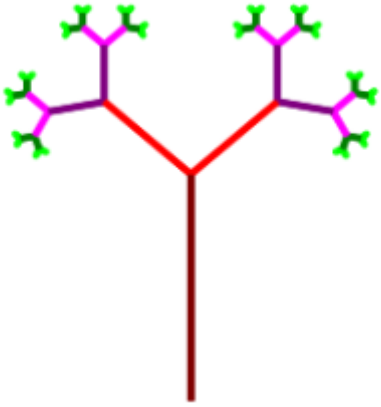
10 **ÁRBOL** 200

El procedimiento TREE es recursivo porque se llama a sí mismo, dos veces. El hecho de que las llamadas sean dos está relacionado con las recurrentes estructuras bifurcadas que representan la "idea básica" de este árbol. A decir verdad, no hay nada más que bifurcaciones en este árbol, que simplemente está hecho de bifurcaciones, siendo la única diferenciación la escala espacial. Vamos a entrar en detalles. El procedimiento se llama primero en la instrucción #10: ÁRBOL 200. Por lo tanto, cuando se introduce TREE, se verifica el valor de LL si es menor de 2 (Instr. #2). Como este tiempo su valor es 200, la ejecución sigue adelante. Con la instrucción #3 la tortuga avanza por LL=200 puntos y gira a la izquierda por 50°, dibujando así el tronco y girándose a la izquierda. Luego, en lugar de seguir dibujando algo que llama ÁRBOL (#4), pero pasándole un valor de LL/2. Abstengámonos de zambullirnos en este llamado al ÁRBOL y asumamos que lo hemos hecho. En el #5 la tortuga gira a la derecha por 100°, en el #6 llama de nuevo al ÁRBOL LL/2, luego en el #7 gira a la izquierda por 50° y regresa por la última rama. Veamos una animación en la siguiente diapositiva.

Demostración del crecimiento del árbol fractal

Un equipo de tortugas para explicar el proceso de recursión

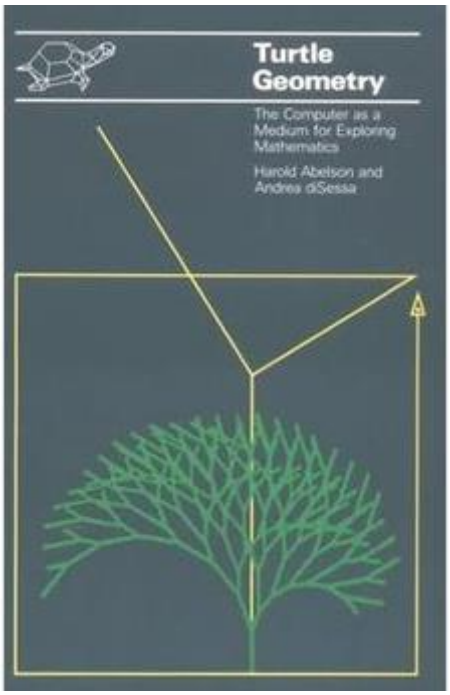
Pero, ¿qué está pasando en las sucesivas llamadas "internas" del ÁRBOL? Para captar el proceso puede ser conveniente pensar no en una sola tortuga sino en todo un equipo de tortugas, cada una cuidando su nivel de recursión específico. Estamos distinguiendo a las tortugas que trabajan en diferentes niveles de recursividad por color. En el primer nivel tenemos a la tortuga marrón, que está dibujando el tronco. En la segunda tenemos la roja que es llamada dos veces, para las dos primeras ramas sucesivas, más grandes. Luego en la tercera tenemos a la tortuga violeta que es llamada cuatro veces para el siguiente grupo de ramas más pequeñas, y así sucesivamente....



Cada vez más natural....

Hay que reconocer que tenemos una especie de árbol pegajoso y un poco demasiado simétrico. ¿Cómo conseguir una versión más natural? Hay muchas posibilidades. En primer lugar, podemos intentar diferenciar entre las ramas izquierda y derecha, para hacer las cosas más asimétricas. Una manera limpia de implementar esta idea es tener procedimientos separados a la izquierda y a la derecha que produzcan tallos de diferentes longitudes.

El siguiente ejemplo es tomado de Turtle Geometry, por Harold Abelson y Andrea diSessa (MIT 1986, p. 84). El ejemplo reproduce exactamente el dibujo que aparece en la portada de su libro.



Más árboles naturales

AL NIVEL DE ÁNGULO DE LA LONGITUD DE LA **RAMA**

LONGITUD DELANTERA*2

LONGITUD DEL NODO NIVEL DE ÁNGULO

LONGITUD DE ESPALDA*2

FIN

AL NIVEL DE ÁNGULO DE LA LONGITUD DE LA **RAMIFICACIÓN**

LONGITUD ANTERIOR

LONGITUD DEL NODO NIVEL DE ÁNGULO

LONGITUD DE ESPALDA

FIN

AL NIVEL DE ÁNGULO DE LONGITUD DEL **NODO**

SI LEVEL=0 [STOP]

ÁNGULO IZQUIERDO

LONGITUD DE LA RAMA ÁNGULO NIVEL 1

ÁNGULO DERECHO*2

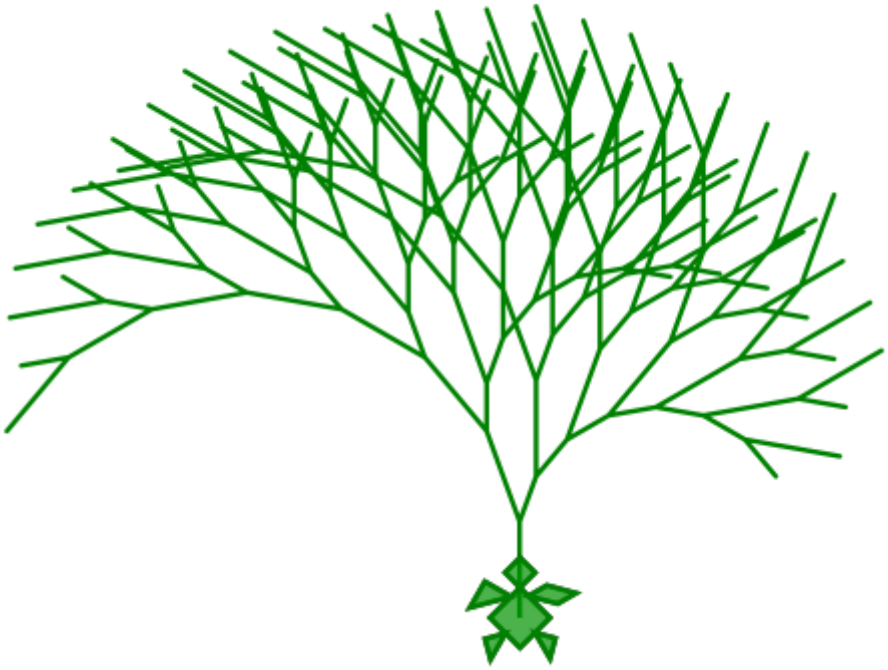
LONGITUD DE LA RAMA ÁNGULO NIVEL-1

ÁNGULO IZQUIERDO

FIN

RAMO 20 20 20 7

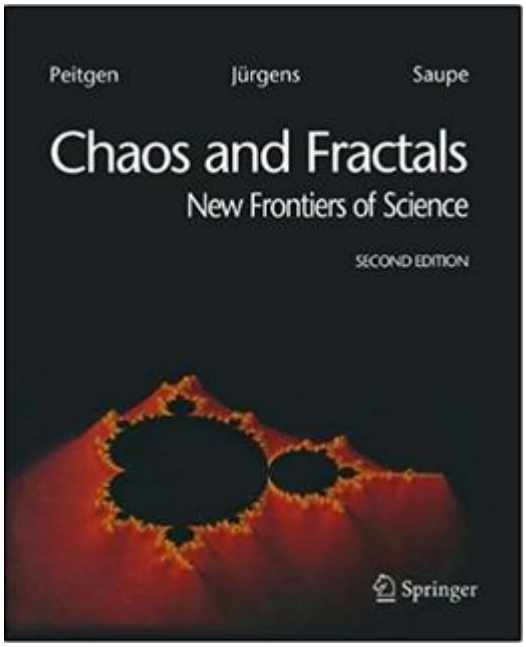
Un árbol más natural



Descripción de los sistemas en L para los procesos de crecimiento

Como dijimos sobre el primer ejemplo simple, en la diapositiva 10, no había nada más que bifurcaciones en ese árbol. Existe una forma más formal de describir la idea básica que subraya estos **procesos de crecimiento**. El formalismo es el de los sistemas L, que es un lenguaje para modelar el crecimiento desarrollado por Aristid LindenMayer en 1968 para describir el desarrollo de las plantas a nivel celular. Posteriormente amplió la interpretación de los sistemas L para representar partes enteras de plantas (Development models of herbaceous plants for computer imagery purposes Prusinkiewicz, P.; Lindenmayer, A. ; Hanan, J. Computer Graphics, 1988, Vol.22 (4), p.141-291.). Los sistemas L pueden ser interpretados en términos de gráficos de tortugas por su representación gráfica (Heinz-Otto Peitgen, Hartmut

Jürgens, Dietmar Saupe, Chaos and Fractals - New Frontiers of Science, Springer Verlag, Nueva York, 1992, p. 376).



Ejemplos de código - Archivos ODT para ser ejecutados con LibreLogo en LibreOffice

Aquí puedes descargar los códigos completos de todos los ejemplos que vamos a discutir. Son archivos ODT, puedes abrirlos en LibreOffice y ejecutarlos con LibreLogo inmediatamente. Es una buena idea hacerlo mientras lee las siguientes explicaciones.

- tree-simple.odt - La versión más simple posible de árbol pegajoso, similar a la de "Turtle Geometry" Abelson & diSessa, p.83. Este código es apropiado para bucear primero en un proceso recursivo.
- tree-simple-L-system.odt - Árbol pegajoso más sencillo - "Turtle Geometry", Abelson & diSessa, p.83Mi variante en el lenguaje del sistema L - Axioma: B - Reglas de producción: **F** → **FF**, **B** → **F[-B]+B**

- árbol-abelson-diSessa.odt - Árbol de ramas izquierda y derecha diferenciadas, de "Turtle Geometry", H. Abelson y A. diSessa, MIT press, 1986, p.84, 85
- árbol-abelson-diSessa-L-system.odt - Árbol de ramas izquierda y derecha diferenciadas, de "Turtle Geometry", H. Abelson y A. diSessa, MIT press, 1986, p.84, 85 Mi versión en el lenguaje del sistema L - Axioma: **B**, Reglas de producción: **B → F[+FB]-B**
- weed.odt - Fractal: weed, de "Chaos and Fractals", Peitgen et al, Springer Verlag, 1992, p. 398 - Axiom: **F**, Regla de producción: **F → F[+F]F[-F]F**
- weed-2.odt - Fractal: another weed, de "Chaos and Fractals", Peitgen et al, Springer Verlag, 1992, p. 398 - Axiom: **B**, **F → F**, **FB → F[+B]F[-B]+B**
- bush.odt - Fractal: bush, de "Chaos and Fractals", Peitgen et al, Springer Verlag, 1992, p. 398 - Axiom: **F**, Regla de producción: **F → FF+[+F-F-F-F]-[-F+F+F]**

Símbolos del sistema L

Aquí tienes una tabla de algunos símbolos del sistema L....

Símbolo del sistema L	Instrucciones para la tortuga
F	desplazarse hacia delante con una determinada longitud de paso fija l y trazar una línea desde la posición anterior a la posición nueva
f	Avanzar como en el caso de F, pero sin trazar la línea.
+	girar a la izquierda por un ángulo fijo
-	girar a la derecha por el mismo ángulo
[	punto de ramificación: comienza la ramificación....
]	... cierre de sucursal
B	no hacer nada rama

Veamos un ejemplo sencillo....

Crecen los fractales clásicos como sistemas en L: curva de Koch

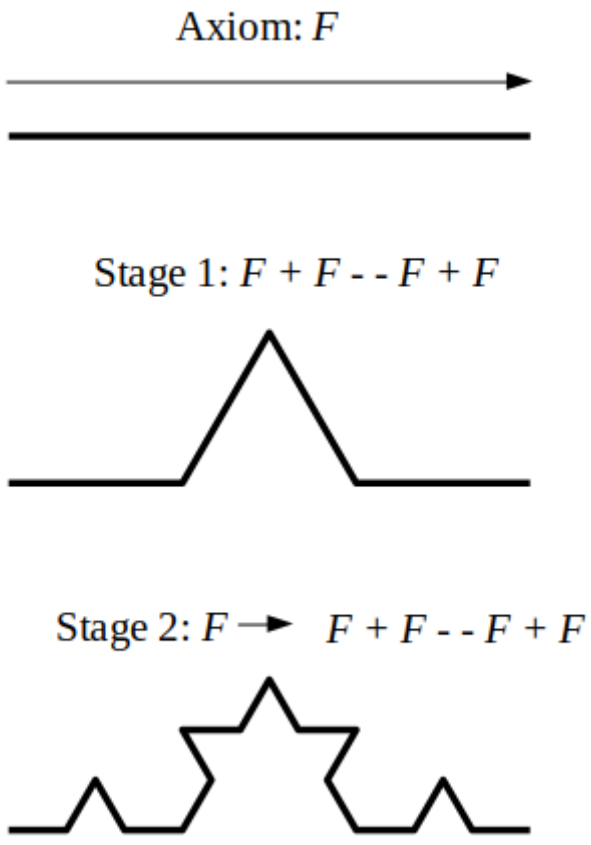
La figura muestra las tres primeras etapas del crecimiento del fractal de Koch. La regla es muy simple: comenzamos con un segmento horizontal de longitud F. Luego se sustituye por el patrón que se muestra en la figura de la etapa 1. En la etapa 2, cada uno de los segmentos más pequeños se sustituye por el mismo patrón, pero escalado por un factor de 1/3. Este ejemplo explica muy claramente la naturaleza de la auto similitud de un fractal, basada en la repetición de una forma geométrica básica reducida por un factor de escala en cualquier iteración. Los sistemas L describen este proceso como un crecimiento, por medio de un axioma y un conjunto de reglas de producción para pasar de una etapa a otra. En el caso de la curva de Koch tenemos:

Axioma: ***F***

Normas de producción:

- ***F -> F + F - - - F + F (1*** paso F, gire a la izquierda, 1 paso F, gire a la derecha 2 veces, 1 paso F, gire a la izquierda, 1 paso F)
- **+ -> + *(los*** giros a la izquierda permanecen donde están)
- **- *(los*** giros a la derecha permanecen donde están)

A + o - son giros a la izquierda o a la derecha con un ángulo de 60°.



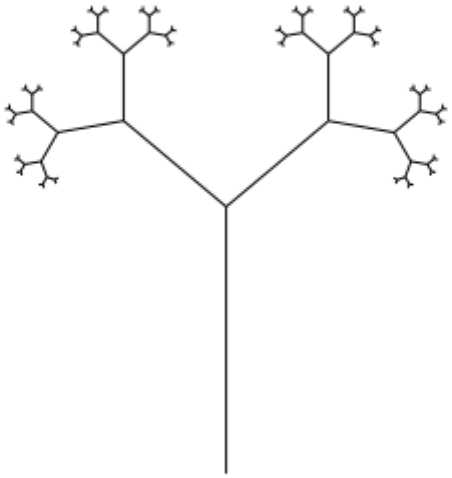
Recodificación del primer árbol simple en la perspectiva del lenguaje del sistema L (tree-simple-L-system.odt)

Axiom: B

Production rules: F → FF

B → F[-B]+B

```
TO WEED LL DELTA ITER
  IF ITER = 1 [
  ]
  [
    FORWARD F
    SALVOPOSDIR ITER
    RIGHT DELTA
    WEED F/2 DELTA ITER-1
    RECUPEROPOSDIR ITER
    LEFT DELTA
    WEED F/2 DELTA ITER-1
  ]
END

LL = 100 DELTA = 50.0 ITER = 8
WEED LL DELTA ITER
```



Comentarios sobre el código anterior

Las diapositivas anteriores muestran lo que se necesita estrictamente para entender la implementación del sistema L. Los procedimientos SETGL para establecer las variables de estado, SAVESTATE para guardar el estado de la tortuga y RESTORESTATE para restaurarla se pueden leer en los códigos que puede descargar de una de las siguientes diapositivas y ejecutarlas en su computadora.

En la parte superior se encuentran el axioma y las reglas de producción. Axioma es B, que es una ramificación de "no hacer nada". Puedes verificar esto si intentas ejecutar el código con iteración ITER=1: no pasará nada visible. Tener a B como axioma significa que empezamos con nada. Hay dos reglas de producción. La primera, F -> FF, significa que el espacio cubierto por un paso F, en la siguiente etapa será cubierto por dos pasos F, por lo tanto la longitud del paso F se reduce a la mitad en cada nueva etapa. La segunda, B -> F[-B]+B, se reconoce dentro del código de procedimiento WEED, si asociamos las llamadas WEED F/2 DELTA ITER a B y FORWARD F a F. Además, la DELTA DERECHA se representa con el símbolo "-" y la DELTA IZQUIERDA con el símbolo "+". El corchete izquierdo corresponde al comando de guardar el estado actual, antes de la ramificación, y el corchete derecho al restablecimiento del estado al final de la ramificación.

Es necesario reflexionar un poco para enfocar este código, pero entonces, el lenguaje del sistema L puede ser usado para generar una amplia gama de posibles simulaciones de crecimiento.

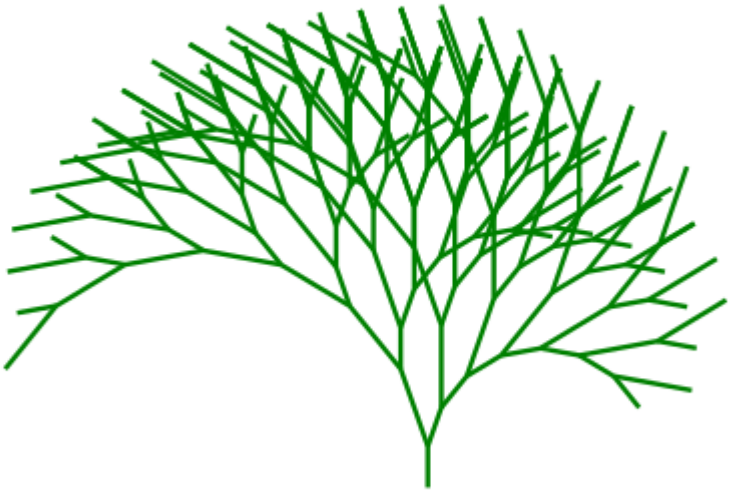
A continuación vamos a ver la versión del sistema L del "árbol más natural" que hemos visto en las diapositivas 14 y 15, luego dos ejemplos de plantas de maleza y un arbusto.

Versión del sistema L del árbol más natural (tree-abelson-diSessa-L-system.odt)

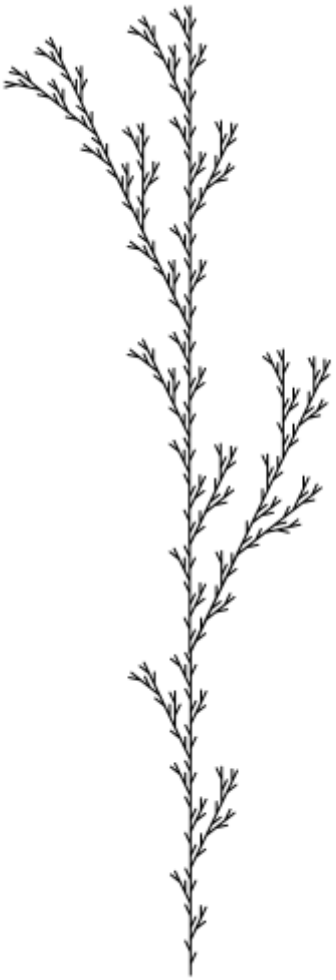
Axiom: **B**

Production rule: **B → F[+FB]-B**

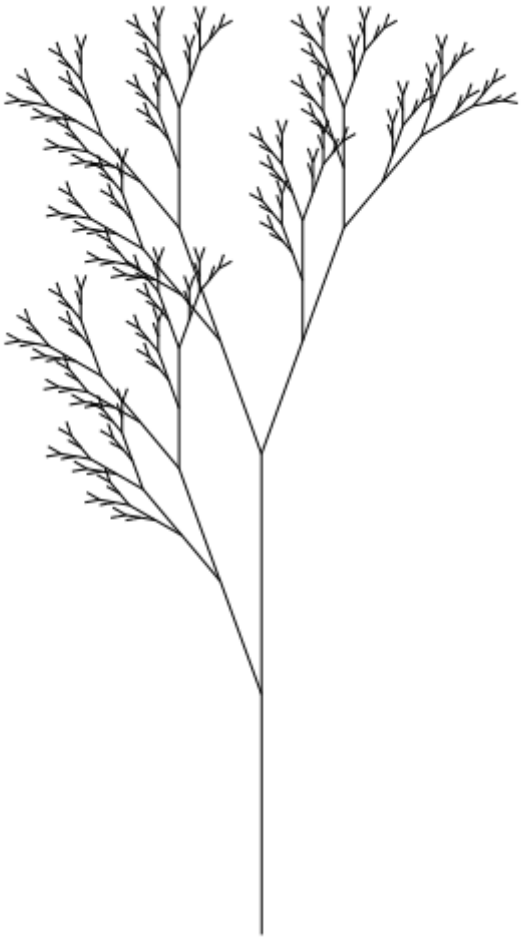
```
TO WEED LL DELTA ITER
  IF ITER = 1 [
    FORWARD F
  ]
  [
    FORWARD F
    SAVESTATE ITER
    LEFT DELTA
    FORWARD F
    WEED F DELTA ITER-1
    RESTORESTATE ITER
    RIGHT DELTA
    WEED F DELTA ITER-1
  ]
END

LL = 20 DELTA = 20.0 ITER = 8
WEED F DELTA ITER
```



Crecimiento fractal de una planta de maleza



Crecimiento fractal de otra planta de maleza



Crecimiento fractal de un arbusto

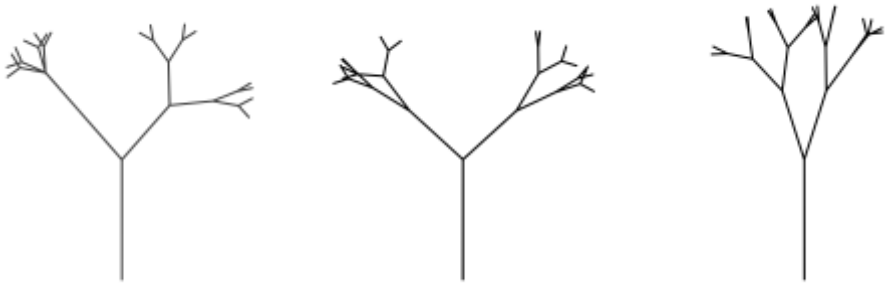
Otra manera de ser natural

Hasta ahora hemos buscado crear formas de aspecto natural trabajando en patrones de crecimiento fractal. Tenemos otra posibilidad, sin embargo, que es el uso de generadores de números aleatorios, como lo hicimos dejando que la tortuga juegue con la tortuga. Consigamos nuestro primer árbol pegajoso más simple:

```
A ÁRBOL LL
A = ALEATORIO 50IF
 LL < 5 [ STOP ]
FORWARD LL LEFT
 ATREE
 LL*3/5RIGHT
 A*2TREE
 LL*3/5LEFT
 ABACK
 LLEND

ÁRBOL 50
```

La diferencia única, con respecto al original, está en la elección del ángulo de giro A, que aquí es un número aleatorio entre 0° y 49°.

Árboles con ángulos aleatorios

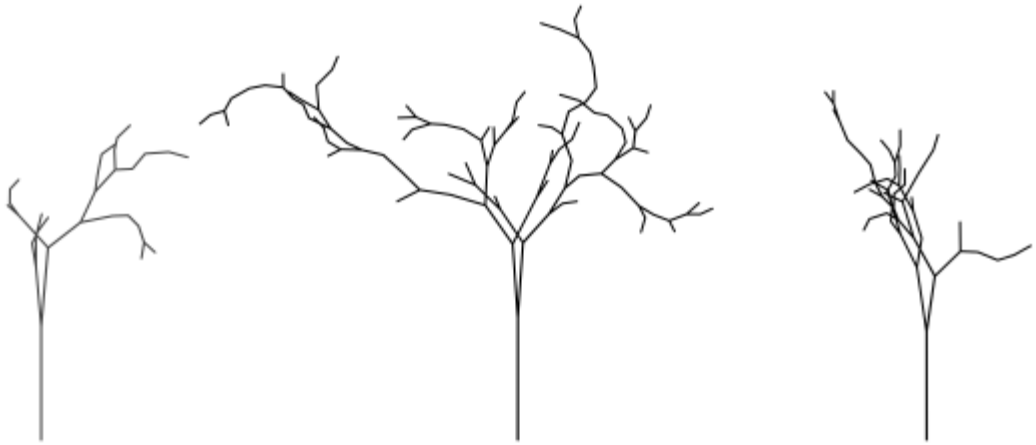


Más aleatoriedad

Aquí añadimos la elección aleatoria del paso siguiente, añadiendo un número aleatorio comprendido entre -5 y 5 a la longitud del paso regular - en este ejemplo 50 en la primera iteración.

```
A ÁRBOL LL
A = RANDOM 50
L = LL - 5 + 10IF
 L < 5 [ STOP
FORWARD LLEFT
 ATREE
 L*3/5RIGHT
 A*2TREE
 L*3/5LEFT
 ABACK
 L*3/5LEFT
 ABACK
 LEND

ÁRBOL 50
```

Árboles con ángulos y ramas de bifurcación aleatoria



Recapitulación

La lección anterior trataba sobre la simulación del comportamiento animal. Esta última lección trataba sobre la simulación de las formas naturales de las plantas. Sin embargo, acabamos de arañar la superficie del mundo de la simulación. Nuestro objetivo era mostrar lo fácil que es construir experimentos de simulación simples pero significativos con una herramienta gratuita como LibreLogo, y lo instructivo que puede ser intentar modelar fenómenos naturales. Posiblemente, otras ideas son incontables. La siguiente, a partir de las investigaciones propuestas en este trabajo, podría consistir en mezclar el modelado del sistema L y la aleatoriedad para producir formas aún más realistas. En una lección sucesiva volveremos sobre la simulación de fenómenos naturales al hablar de algunos problemas mecánicos físicos.

Empezando a dar vueltas y vueltas... al cometa Halley.

La dimensión vertical del Logotipo

En esta lección vamos a conectar un piso bastante bajo con un techo bastante alto. Empezaremos con actividades para niños de 7-8 años, terminando con la integración de las ecuaciones diferenciales de Newton y el cálculo de las órbitas de los cuerpos celestes. En contextos aparentemente muy diferentes, trataremos de mostrar cómo comparten la misma perspectiva matemática básica.

Seymour Papert explicó eficazmente cómo es posible que los niños que trabajan con el Logo estén expuestos a poderosas ideas matemáticas. Invitamos al lector a leer cuidadosamente las siguientes tres diapositivas.

Jugar a la tortuga

La meta de las primeras experiencias de los niños en el ambiente de aprendizaje de las Tortugas no es aprender reglas formales sino desarrollar una comprensión de la manera en que se mueven en el espacio. Estos conocimientos se describen en TURTLE TALK y por lo tanto se convierten en "programas" o "procedimientos" o "ecuaciones diferenciales" para la Tortuga.

Observemos de cerca cómo un niño, que ya ha aprendido a mover la Tortuga en líneas rectas para dibujar cuadrados, triángulos y rectángulos, puede aprender a programarla para dibujar un círculo.

Imaginemos, pues, como he visto cientos de veces, a un niño que exige: ¿Cómo puedo hacer que la Tortuga dibuje un círculo? El instructor en un entorno LOGO no proporciona respuestas a tales preguntas, sino que introduce al niño en un método para resolver no sólo este problema, sino también una gran clase de otros.

Este método se resume en la frase "jugar a la tortuga". Se anima al niño a que mueva su cuerpo ya que la Tortuga en la pantalla debe moverse para hacer el patrón deseado.

A MI CÍRCULO REPITA [ADELANTE 1 DERECHA 1] FIN

Para el niño que quería hacer un círculo, moverse en un círculo puede llevar a una descripción como: "Cuando caminas en círculo das un pequeño paso adelante y giras un poco. Y sigues haciéndolo". De esta descripción es sólo un pequeño paso hacia un programa formal de Tortugas.

A MI CÍRCULO REPITA [ADELANTE 1 DERECHA 1] FIN

Otro niño, quizás menos experimentado en programación simple y en la heurística de "jugar a las tortugas", podría necesitar ayuda. Pero la ayuda no consistiría principalmente en enseñar al niño cómo programar el círculo de la Tortuga, sino en enseñarle un método, un procedimiento heurístico. Este método (que incluye el consejo resumido como "play Turtle") intenta establecer una conexión firme entre la actividad personal y la creación de conocimiento formal.

Aprendizaje sintónico

El incidente del círculo de tortugas ilustra el aprendizaje sintónico. Este término se toma prestado de la psicología clínica y puede ser contrastado con el aprendizaje disociado ya discutido. Algunas veces el término se usa con calificativos que se refieren a tipos de sintonía.

Por ejemplo, el Círculo de la Tortuga es sintónico en el sentido de que el círculo está firmemente relacionado con el sentido y conocimiento de los niños sobre sus propios cuerpos. O es sintónico del ego en el sentido de que es coherente con el sentido que los niños tienen de sí mismos como personas con intenciones, metas, deseos, gustos y disgustos.

Un niño que dibuja un círculo de Tortuga quiere dibujarlo; hacerlo produce orgullo y emoción. La geometría de la tortuga se aprende porque es sintónica. Y es una ayuda para aprender otras cosas porque fomenta el uso consciente y deliberado de estrategias de resolución de problemas y matemáticas.

Ahora vamos a la historia de un profesor

Ahora, vamos a bosquejar la historia contada por Antonella, una experimentada maestra de primaria que quería explotar el concepto de aprendizaje sintónico de Papert con sus hijos.

Antes de ir a la computadora para dibujar un cuadrado, yo"usaba" su cuerpo: Les pedí que caminaran y describieran un cuadrado, tratando de imaginar instrucciones precisas: paso y vuelta, paso y vuelta... al punto de partida.

Aún no somos precisos, volvamos a intentarlo.

Le di a cada niño una muñeca pequeña y le pedí que caminara sobre el borde de un azulejo, al mismo tiempo que le presenté los nombres de los comandos, Adelante un paso y así sucesivamente.

Buscando algo redondo en el suelo

Intenta dibujar un círculo....

Reflejando....

¿Qué tortuga manda...?

Discutiendo en la computadora:

...

"Necesitas cambiar los números"

"Tenemos que ir en la curva"

"Debemos continuar para hacer lo correcto"

"Si siempre vamos bien, tenemos un pequeño punto"

...

Luego la perspicacia:

"¡Debemos avanzar 1 a la derecha 1!

"Intentémoslo", te sugiero.

"Tiene que repetirse muchas veces...."

Del aprendizaje corporal al conocimiento formal

Es fascinante cómo estas primeras experiencias pueden llevarnos a reflexionar sobre el concepto de círculo sin tener todavía un conocimiento formal del mismo.

Sin embargo, al hacerlo, estamos trabajando sobre un concepto mucho más avanzado que todo lo que los niños verán antes de la madurez, a saber, el concepto de cálculo diferencial.

Luego, tan pronto como tomen posesión de la noción formal de circunferencia, será posible volver al Logo y reanudar la discusión a través de la forma sintética de producir un círculo, por medio de la instrucción CIRCULO D, donde D representa el diámetro del círculo, reflexionando sobre la definición como lugar de puntos equidistantes de un punto dado.

Desde el suelo bajo hasta el techo alto

Vamos a pedirle ayuda a Papert, otra vez....

El niño en la experiencia del círculo de tortugas no estaba aprendiendo sobre el formalismo del cálculo, por ejemplo que el derivado de x^n es nx^{n-1}, sino sobre su uso y su significado. De hecho, el programa Círculo de Tortugas conduce a un formalismo alternativo para lo que tradicionalmente se llama una "ecuación diferencial" y es un poderoso portador de las ideas que hay detrás del diferencial. Por eso es posible entender tantos temas a través de la Tortuga; el programa Tortuga es un análogo intuitivo de la ecuación diferencial, un concepto que se encuentra en casi todos los ejemplos de matemáticas aplicadas tradicionalmente.

El cálculo diferencial deriva gran parte de su poder de la capacidad de describir el crecimiento por lo que está sucediendo en la punta de crecimiento. Esto es lo que lo convirtió en un buen instrumento para los intentos de Newton de entender el

movimiento de los planetas. A medida que se traza la órbita, son las condiciones locales en el lugar donde el planeta se encuentra ahora las que determinan a dónde irá después. En nuestras instrucciones a la Tortuga, Adelante 1, Derecha 1, nos referimos solamente a la diferencia entre dónde está la Tortuga ahora y dónde estará momentáneamente. Esto es lo que hace que las instrucciones sean diferentes.

Comparación con las geometrías de Euclides y Descarte

No hay referencia en ellos a ninguna parte distante del espacio fuera del camino mismo. La Tortuga ve el círculo a medida que avanza, desde dentro, por así decirlo, y está ciega ante cualquier cosa que esté lejos de él. Esta propiedad es tan importante que los matemáticos tienen un nombre especial para ella: La geometría de la tortuga es "intrínseca". El espíritu de la geometría diferencial intrínseca se observa observando varias maneras de pensar en la curva, por ejemplo, el círculo.

Para Euclides, la característica definitoria de un círculo es la distancia constante entre los puntos del círculo y el punto, el centro, que no forma parte del círculo. En la geometría de Descarte, más probable que la de Euclides que la de la Tortuga, los puntos se sitúan a la distancia de algo fuera de ellos, es decir, de los ejes de coordenadas perpendiculares. Las líneas y curvas se definen mediante ecuaciones que conectan estas coordenadas. Así, por ejemplo, un círculo se describe como

$$(x-a)^2+(y-b)^2=R^2$$

Una geometría diferencial

En la geometría de la Tortuga un círculo se define por el hecho de que la Tortuga sigue repitiendo el acto: Adelante un poco, gire a la DERECHA un poco. Esta repetición significa que la curva que dibuja tendrá una curvatura constante, donde la curvatura significa cuánto gira para un movimiento de avance dado.

La geometría de la tortuga pertenece a una familia de geometrías con propiedades que no se encuentran en el sistema euclidiano o cartesiano. Estas son las geometrías diferenciales que se han desarrollado desde Newton y que han hecho posible gran parte de la física moderna. Hemos notado que la ecuación diferencial es el formalismo a través del cual la física ha sido capaz de describir el movimiento de una partícula o de un planeta.

[...]

La geometría de la tortuga tiene vínculos tanto con la experiencia de un niño como con los logros más poderosos de la física. Porque las leyes del movimiento del niño, aunque de forma menos precisa, comparten la estructura matemática de la ecuación diferencial con las leyes del movimiento de los planetas que giran alrededor del sol y con las de las polillas que giran alrededor de una llama de vela. Y la Tortuga es nada más y nada menos que una reconstrucción en forma computacional intuitiva del núcleo cualitativo de esta estructura matemática.

Lenguaje de programación versus notación algebraica para la física

Las siguientes consideraciones han sido inspiradas por la lectura del artículo en profundidad de Bruce Sherin: A comparison of programming languages and algebraic notation as expressive languages, International Journal of Computers for Mathematical Learning 6: 1-61, 2001. De lo abstracto:

El propósito del presente trabajo es considerar algunas de las implicaciones de reemplazar, a los efectos de la instrucción en física, la notación algebraica por un lenguaje de programación. Lo novedoso es que, más que en trabajos anteriores, me tomo en serio la posibilidad de que un lenguaje de programación pueda funcionar como el principal sistema de representación para la enseñanza de la física. Esto significa tratar la programación como si tuviera potencialmente un estado similar y realizar una función similar a la notación algebraica en el aprendizaje de la física. [...]

Una conclusión de este trabajo es que la física algebraica puede ser caracterizada como ***una física de equilibrio y equilibrio****, y la física de programación como* ***una física de procesos y causalidad****. De manera más general, este trabajo proporciona una base teórica y empírica para comprender cómo el uso de determinados sistemas de símbolos afecta la conceptualización de los estudiantes.*

Cuerpo en caída libre - notación algebraica

Comparemos el concepto de Bruce de una física de equilibrio y equilibrio (notación algebraica) versus una física de procesos y causalidad (enfoque de programación). Con la notación algebraica clásica, la descripción de la física básica comienza con el segundo principio de dinámica, $\mathbf{F} = m\mathbf{a}$, que para un cuerpo en caída libre junto a la superficie de la tierra es $\mathbf{F} = m\mathbf{g}$, donde $\mathbf{g}$ es la constante de aceleración de la gravedad dirigida hacia el suelo. El módulo $\mathbf{g}$ es de 9,8 m/s2. Esta es una ecuación diferencial que dice que, en cada punto de la trayectoria de caída, la aceleración es constante y es igual a g. Integrándola dos veces con respecto al tiempo, obtenemos:

$$y = y_0 + v_0 t + \frac{1}{2} g t^2$$

donde y es la posición a lo largo de la vertical, y_0 es la posición inicial del cuerpo, v_0 es su velocidad inicial y t es el tiempo. Esta es la solución del problema, ya que esta ecuación te dice dónde estará el cuerpo en cualquier momento. El punto desarrollado por Sherin es que los estudiantes tienen dificultades para captar la dinamicidad del fenómeno leyendo estas ecuaciones, tales como $\mathbf{F} = m\mathbf{a}$ o $y = y_0 + v_0 t + \frac{1}{2} g t^2$ La información está ahí, por supuesto, pero la capacidad de extraer toda la información que contienen no es trivial y requiere tiempo.

Cuerpo en caída libre - lenguaje de programación

El del lenguaje de programación es básicamente un enfoque de simulación: tienes que averiguar qué está pasando con cada variable en cualquier momento. Primero hay que saber dónde se encuentra el cuerpo al principio del proceso y cuál es su velocidad en ese punto, o si está quieto. Entonces, la pregunta es: dado el valor de aceleración, ¿cuál será el valor de la velocidad después de un *pequeño* intervalo de tiempo? ¿Y cuál es su nueva posición, después del mismo intervalo de tiempo? Como señaló Papert, aquí estamos en el centro del cálculo diferencial, ya que nos centramos en los *pequeños* cambios en los parámetros del sistema (posición y velocidad) después de un *pequeño* intervalo de tiempo, dadas las fuerzas (gravedad) que actúan sobre el cuerpo.

```
YPOS = 0.0 ; posición inicialVEL = 0.0 ; velocidad inicialACC = 9.8 ; aceleración constanteDVEL = ACC ; incremento de la velocidad de ov en un intervalo de tiempo

REPEAT 10 [ ; bucle en intervalos de tiempo sucesivosVEL = VEL + DVEL ; velocidad de actualización para el siguiente pasoYPOS = YPOS + VEL; ; otras instrucciones: mostrar posición y así sucesivamente.....
]
```

Estas son sólo las instrucciones principales, ver el código completo y el resultado en la siguiente diapositiva. Otras variaciones en diapositivas sucesivas. Los códigos para ser ejecutados en un archivo ODT pueden ser descargados aquí.

Caída libre del cuerpo - lenguaje de programación - resultado

```
; Free-falling body with constant acceleration of gravity

CLEARSCREEN
HOME
SHOWTURTLE

PENUP
FORWARD 350
RIGHT 180
PENDOWN

YPOS = 0.0 ; initial position
VEL = 0.0 ; intitial velocity
ACC = 9.8 ; constant acceleration
DVEL = ACC ; increment ov velocity in one time interval

REPEAT 10 [
  VEL = VEL + DVEL ; update velocity for next step
  YPOS = YPOS + VEL ; update position for next step
  PENDOWN
  CIRCLE 5
  PENUP
  FORWARD VEL
  ]

HIDETURTLE
```

Cuerpo en caída libre con aceleración constante y componente de velocidad horizontal

```
; Free-falling body with constant acceleration of gravity and
; constant horizontal componente of velocity

CLEARSCREEN

TO PARAB ACC
HOME
PENUP
FORWARD 350
RIGHT 180
PENDOWN

YPOS = 0.0
VEL = 0.0

REPEAT 10 [
  VEL = VEL + ACC
  YPOS = YPOS + VEL
  PENDOWN
  CIRCLE 5
  PENUP
  FORWARD (VEL/2-1)
  LEFT 90 FORWARD 10 RIGHT 90
  ]

END

PARAB 5
PARAB 9.8
PARAB 30

HIDETURTLE
```

Cuerpo en caída libre con aceleración constante y resistencia al aire

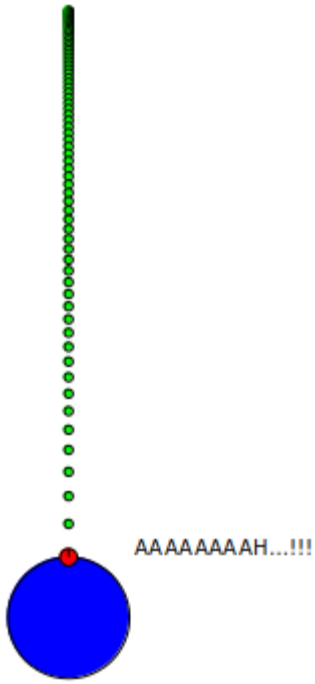
```
; Free-fall of body with constant acceleration of gravity and
; air resistance

CLEARSCREEN
HOME

YPOS = 0.0
VEL = 0.0
ACC = 9.8
RES = 0.0

G = 9.8
M = 10.0
K = 2.0

PENUP
FORWARD 350
RIGHT 180
PENDOWN

REPEAT 20 [
  RES = VEL * K
  ACC = G - RES / M
  VEL = VEL + ACC
  YPOS = YPOS + VEL
  PENDOWN
  CIRCLE 5
  PENUP
  FORWARD VEL
  ]

HIDETURTLE
```

Cuerpo colgando de una cuerda - oscilaciones con el tiempo

```
; Body hanging from a spring - oscillations with time
; (y axis: x-time)

CLEARSCREEN
HOME

YPOS = 0.0
VEL = 0.0
ACC = 9.8
RES = 0.0

G = 9.8
M = 10.0
KA = 0.5
K = 1.0

PENUP
FORWARD 250
RIGHT 180
PENDOWN

REPEAT [
 RES = KA * VEL
 SPR = K * YPOS
 ACC = G - RES / M – SPR / M
 VEL = VEL + ACC
 YPOS = YPOS + VEL
 PENDOWN
 CIRCLE 5
 PENUP
 FORWARD VEL
 LEFT 90 FORWARD 3 RIGHT 90 ; comment this if want to see oscillating along y-axis
 ]
```

Un cuerpo celeste que cae sobre la tierra desde el espacio

```
; Free-fall of body from space

CLEARSCREEN
HOME

FILLCOLOR "BLUE"
CIRCLE 70
FILLCOLOR "LIME"
XPOS0 = POSITION[0]
YPOS0 = POSITION[1]

PENUP
FORWARD 350
RIGHT 180
PENDOWN
HIDETURTLE

XPOS = POSITION[0]
YPOS = POSITION[1]
VEL = 0.0
KG = 9800

REPEAT [
 DY = YPOS-YPOS0
 VEL = VEL + KG / DY**2
 YPOS = YPOS + VEL
 PENDOWN
 CIRCLE 5
 IF YPOS0-YPOS < 10 [ BREAK ]
 PENUP
 POSITION [XPOS,YPOS]
;FORWARD (VEL)
 ]
FILLCOLOR "RED"
CIRCLE 10
HEADING 0
FORWARD 5
LABEL "                    AAAAAAAAH...!!!"
```



Recapitulando por un tiempo y preparándose para ir al espacio....

Mostramos una progresión en la que los ejercicios han sido progresivamente complicados alrededor del cuerpo en caída libre, incluyendo diferentes efectos secundarios o un contexto cambiante. Podrían utilizarse en una actividad de la escuela secundaria, permitiendo que los estudiantes desarrollen la programación de la manera más autónoma posible. Podrían trabajar en grupos pequeños, por ejemplo, enfrentándose por sí mismos a problemas como: ¿debemos aumentar primero la posición o la velocidad? De esta manera, los estudiantes pueden

1. se centran en el papel crucial del concepto de condiciones iniciales.
2. tener una percepción más clara de la naturaleza dinámica del fenómeno
3. ser inducidos a centrarse en los valores numéricos efectivos y sus respectivas unidades de medida de los parámetros pertinentes

Invitamos al lector interesado a descargar los ejemplos prácticos, intentando experimentar y desarrollarlos. Pueden descargarse como un único archivo sip: logo-odt-files.zip

en el último ejemplo dejamos caer la suposición de la aceleración constante que pone el cuerpo lejos en el espacio. Ahora vamos a intentar introducir una componente de velocidad horizontal inicial....

Después de todo, los planetas giran...

La adición de una componente inicial de velocidad horizontal en el último ejercicio nos permite la simulación de órbitas de cuerpos. Por lo tanto, esbocemos el problema en el contexto de la gravedad de Newton, primero en notación algebraica y, por supuesto, limitándonos al problema bidimensional.

En lo que respecta al enfoque algebraico, aquí sólo planteamos el problema. La solución analítica de las ecuaciones diferenciales es un problema de nivel de educación superior. Sin embargo, el enfoque algebraico sirve como punto de partida para la solución numérica programada.

Para resolver el problema tenemos que partir del segundo principio de la dinámica $\mathbf{F} = m\mathbf{a}$, donde la fuerza es dada por la ley de Newton $\mathbf{F} = -\frac{GMm}{r^2}\frac{\mathbf{r}}{r}$, Ges la constante gravitacional, Mes la masa del más grande de los dos cuerpos, $m$el más pequeño, $r$es la distancia entre las dos masas, entre los baricentros de los dos cuerpos para ser precisos. Los símbolos en negrita son vectores, los otros son escalares.

Primero tenemos que derivar la aceleración $\mathbf{a} = -\frac{GM}{r^2}\frac{\mathbf{r}}{r}$, y luego hay que resolver la ecuación diferencial $\frac{d}{dt}\frac{d\mathbf{r}}{dt} = -\frac{GM}{r^2}\frac{\mathbf{r}}{r}$. Por supuesto, esto va más allá de nuestros objetivos. Sin embargo, vamos a utilizar esta relación como base para el enfoque numérico.

Preparación de la solución numérica

Como hemos visto en los ejercicios anteriores, lo que siempre necesitamos es el valor de la aceleración en cada punto del camino. Las pocas líneas de código a las que nos vamos a referir están extraídas del archivo Halley-RK-4-AU-90-en.odt que se encuentra en la colección descargable del archivo logo-odt-files.zip. También encontrará allí una explicación de las variables. Las expresiones de aceleración anteriores se traducen en código de la siguiente manera

XACC = GG/R2 ∗ DX/RYACC = GG/R2 ∗ DY/R

Estos valores de aceleración se utilizan para obtener incrementos de velocidad

XVEL = XVEL + XACC ∗ DtYVEL = YVEL + YACC ∗ Dt

y a partir de éstos los respectivos incrementos de posición

XPOS = XPOS + XVEL ∗ DtYPOS = YPOS + YVEL ∗ Dt

El mismo contexto de cálculo diferencial del círculo de los niños anteriores

La esencia del código reportada en Halley-RK-4-AU-90-es.odt es simple:

- empezar a formar un punto dado con una velocidad dada, luego
- repita hasta que la órbita se cierre:
    - evaluar la aceleración en el punto actual
    - a partir de esto, evalúe los siguientes incrementos de velocidad y posición
    - pasar a la siguiente posición

La descripción respectiva del círculo de niños, REPEAT 360 [adelante 1 a la derecha 1], es la siguiente

- partir de un punto dado
- Repita hasta que la órbita se cierre.
    - desplazarse al siguiente punto con la tecla ARRIBA 1 DERECHA 1

Incluso si la evaluación de la siguiente posición es algo compleja en el contexto de la gravedad de Newton, mientras que es trivial en el caso del círculo de los niños, el punto importante que comparten es que en **ambos casos el proceso es local**. Esa es la característica básica del cálculo diferencial. Y ese es un ejemplo de una poderosa idea matemática incrustada en las primeras actividades del Logo.

Condiciones iniciales

La última diapositiva fue la más importante de esta sección, sin embargo necesitamos especificar otro elemento importante del código de órbita de Halley, el de las condiciones iniciales.

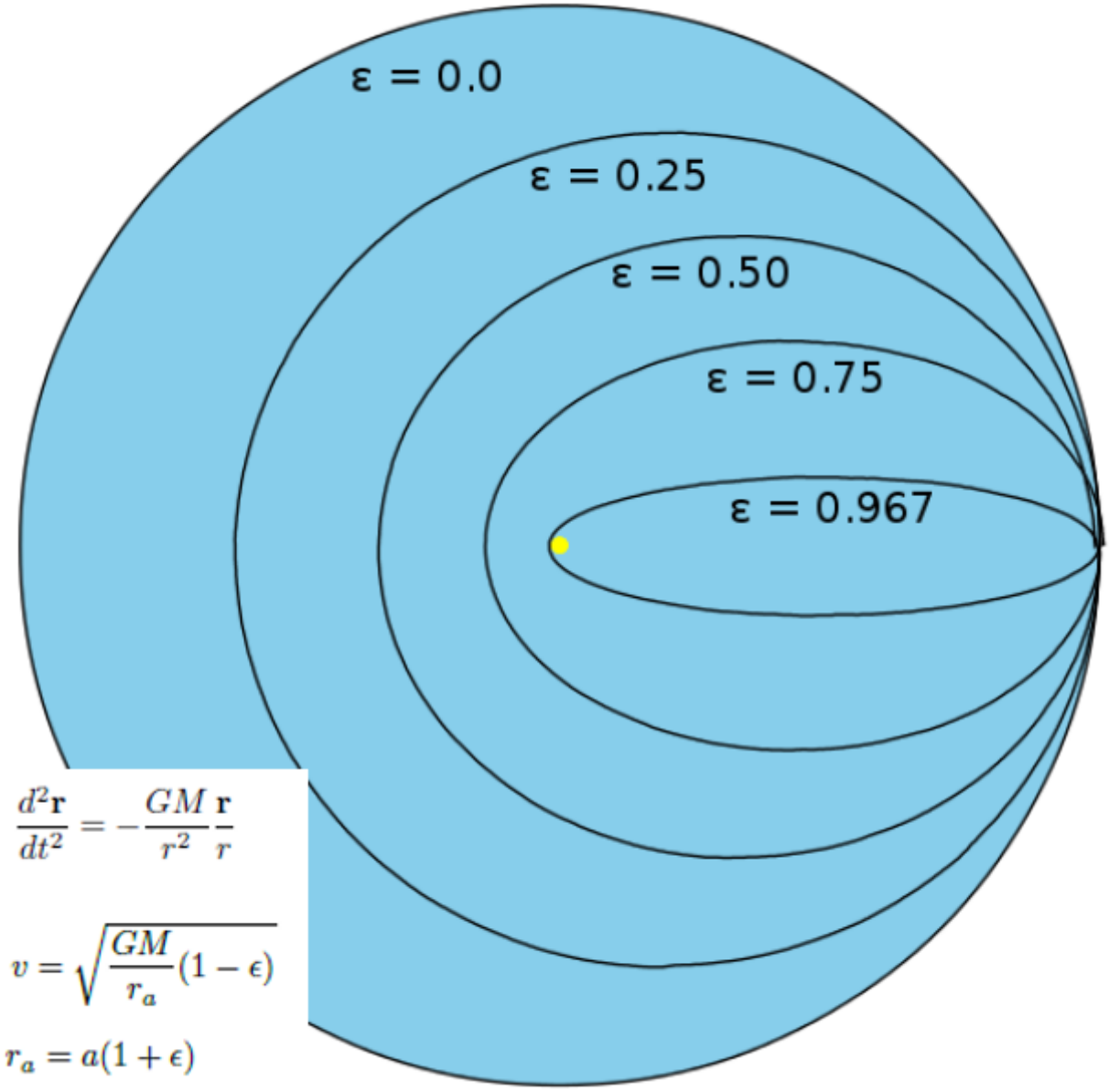
Determinar las condiciones iniciales es un aspecto crucial de cualquier problema físico. El marco matemático típico de los fenómenos físicos es el de las ecuaciones diferenciales y éstas, para ser resueltas, necesitan establecer cuáles son las condiciones iniciales.

En nuestro caso, las condiciones iniciales se pueden establecer utilizando la ley Kepler II y las propiedades de elipsis. La ecuación relevante aquí es

$$v = \sqrt{\frac{GM}{r_a}(1-\epsilon)}$$

donde v es la velocidad inicial, G es la constante de gravitación, M la masa solar, r_a es el aphelion y ϵ la excentricidad de la órbita. para obtener la órbita de Halley se han utilizado los r_a y ϵ valores encontrados en los datos astronómicos.

Orbita para varios valores de excentricidad - 0.967 es el valor del cometa Halley



Recapitulación

La duración de esta lección fue realmente desafiante. Empezamos con un juego de niños y terminamos con un ejercicio probablemente adecuado para un curso de física en la universidad.

Los objetivos eran mostrar el potencial de un (aparentemente) simple software libre para la programación educativa y, al mismo tiempo, dar un ejemplo real y fuerte de cómo las ideas matemáticas fundamentales pueden ser ofrecidas muy pronto por medio de un uso inteligente del lenguaje de programación Logo.

El miedo a las matemáticas

La motivación fundamental para la génesis del Logo reside en la cuestión aún no resuelta de la enseñanza de las matemáticas. El lenguaje del Logo fue concebido por Papert precisamente para tratar de resolver este viejo problema por el que también había acuñado un nombre preciso, *Mathofobia*, para describir la antipatía generalizada hacia este tema. El interés de Papert por este tema ha acompañado toda su vida laboral, que se extendió a lo largo de la segunda mitad del siglo XX. Su contribución fue excepcional, tanto en términos de elaboración teórica desde el punto de vista pedagógico, como de la creatividad que lo llevó a idear un lenguaje específico para acercar a los niños a las matemáticas. Su profunda competencia tanto en la matemático-informática como en la pedagogía hace que su trabajo sea único y explica su rara capacidad para proponer soluciones concretas.

La sensación es que no ha cambiado mucho, al menos en promedio desde los años ochenta; que la motivación inicial, basada en una revisión seria y difícil de la forma en que se introduce a los jóvenes en las matemáticas, ha terminado diluyéndose en el pote de la "codificación", una especie de Disneylandia, superficialmente excitante para algunos, objeto de burla para otros; que el mensaje de Papert, en cierto modo extremo y provocador, que sin duda ha de ser descifrado con respecto a un contexto diferente, es en gran medida erróneo; que todo es derrotado por el fracaso del Logo, restringido a una minoría de círculos experimentales, sin haber revolucionado nada, en contraste con las legítimas expectativas de Papert; que invocar la magia de las matemáticas para introducir a los jóvenes en un dominio comúnmente considerado "frío", es un sueño concebible sólo por un matemático, algo "idealista". En otras palabras, una utopía.

El logo falló, dijimos. Fracasó en las intenciones iniciales de Papert. No está generalizado en las escuelas y no se adopta como una forma estándar de apoyar el aprendizaje de matemáticas y ciencias. Pero no falló en el sentido de no haber dejado rastro, sino todo lo contrario. Existen muchas versiones del Logo en todo el mundo, algunas de las cuales se han convertido en herramientas importantes para la investigación educativa, por ejemplo en el campo de la simulación de sistemas biológicos complejos. Y es siempre a partir de Logo que el extenso mundo de los lenguajes de los bloques visuales tomó su ejemplo, ante todo, Scratch. Logo y Scratch no se oponen. En cierto modo, Scratch deriva del Logo y "contiene" muchas de sus

funcionalidades. Muchas de las cosas que puede hacer en Logo también se pueden hacer en Scratch. Pero Scratch está mucho más orientado a la construcción de "tu propio videojuego" o a la narración de historias. El problema, sin embargo, es que este abanico más amplio de posibilidades, desplegado en un contexto de profesores con escasa formación tecnológica, ha acabado dispersando las intenciones educativas originales del Logo. Una de las intenciones de este manual es recuperar el "sabor matemático" original de la práctica de codificación en la escuela. En la siguiente sección vamos a comentar lo que Papert llamó "Mathophobia", un tipo de enfermedad contra la que el Logo estaba destinado a luchar.

Matrofobia

Seymour Papert eligió el título del segundo capítulo de Mindstorms "Mathophobia: The Fear for Learning" . Su punto de partida es la escisión esquizofrénica entre las humanidades y la ciencia, una división profundamente construida en el lenguaje, la cosmovisión, la organización social y el sistema educativo. Una división que en los últimos 30-40 años de deriva neoliberal, se ha ampliado aún más. Por ejemplo, en las universidades, desde los años ochenta, los académicos luchan en la competencia por conseguir que sus investigaciones sean financiadas. Desafortunadamente, la otra cara de la moneda es que a casi nadie le importa enseñar, o muy poco. La carrera de un profesor no depende de la calidad de su enseñanza, sino casi sólo de la cantidad de su producción científica. Las consecuencias son muy malas. Los académicos tienden a transformarse en gestores cuando investigan y en funcionarios públicos cuando enseñan: empresarios dinámicos por un lado y conservadores (fuertes) por el otro. De esta manera, incluso en las humanidades vemos una deriva técnica del papel académico. La misión de enseñar, que en cierto modo es el "lado humanista" del trabajo, se reduce a una especie de Cenicienta. Así, la dicotomía entre lo científico y lo humanístico es aún más fuerte y desequilibrada.

Romper la línea entre ciencia y humanidades

No se trata de un equilibrio adecuado, sino de romper la línea entre las dos culturas. Papert veía la computadora como una fuerza para atenuar la distinción. La práctica de la codificación se pensó como una forma de introducir una matemática más humanista y de explotar algunos razonamientos científicos en las humanidades. Es en este contexto que Papert habló de un Mathland, donde las matemáticas se convertirían en un vocabulario natural, con la idea de que podríamos cambiar no sólo la forma en que enseñamos matemáticas sino también la forma en que nuestra cultura piensa sobre el conocimiento y el aprendizaje.

Papert afirma que sus argumentos no se limitan al aprendizaje de las matemáticas, sino que se refieren a la actitud hacia el aprendizaje en general. La palabra \textit{matofobia} sugiere dos asociaciones. Una es la aversión generalizada a la matemática. La otra deriva de la raíz "matemáticas", que en griego antiguo significa aprendizaje en sentido general. Así, si los niños empiezan por ser aprendices espontáneos hábiles, más tarde "aprenden" el miedo al aprendizaje y no sólo a las matemáticas. Irónicamente, parece que cuanto más te instruyen, más temes aprender.

El aprendizajeiagetiano

Los niños aprenden miles de palabras antes de entrar al primer grado. Menos obvio para muchas personas es el hecho de que los niños también aprenden una gran cantidad de matemáticas. Entre estos conocimientos preescolares existen, por ejemplo, nociones como la conservación del volumen de líquidos en recipientes de forma diferente, o la independencia del número total de objetos del orden en que han sido contados. Papert llamó a esto

El aprendizajeiagetiano, un proceso de aprendizaje que tiene muchas características que las escuelas deben envidiar: es efectivo (todos los niños llegan allí), es barato (parece no requerir ni maestros ni desarrollo curricular), y es humano (los niños parecen hacerlo con un espíritu despreocupado sin recompensas y castigos externos explícitos).

De hecho, las prácticas de la enseñanza de las matemáticas subestiman en gran medida el aprendizajeiagetiano, imponiendo conocimientos formales que resultan estar mayormente disociados de las antiguas nociones espontáneas. Las consecuencias para los futuros adultos son graves. La pérdida de la actitud positiva del niño hacia el aprendizaje es un fenómeno muy común en la edad adulta y no se refiere sólo a las matemáticas.

La deficiencia se convierte en identidad: "No puedo aprender francés, no tengo oído para los idiomas.""Nunca podría ser un hombre de negocios, no tengo cabeza para las cifras."

El argumento de Papert sobre las actitudes y los prejuicios

Alrededor del 80% de mis alumnos de Educación Primaria se declaran "distantes de las matemáticas". Muchos afirman que también tienen dificultades con las tecnologías, a pesar de que se supone que son "nativos digitales". La noción de que hay gente inteligente y gente tonta para una actividad dada es generalizada. Es extremadamente difícil erradicar los prejuicios sobre las propias actitudes. El punto es que las creencias aceptadas sobre la aptitud matemática no se derivan de la evidencia disponible. Con el fin de reforzar el concepto, Papert refundió el argumento de la siguiente manera (Papert, 1993, p.43):

Imaginemos que los niños se ven obligados a pasar una hora al día dibujando pasos de baile en el papel cuadriculado y tienen que pasar pruebas en estos "datos de baile" antes de que se les permita bailar físicamente. ¿No esperaríamos que el mundo se llenara de "bailarines"? ¿Diríamos que los que llegaron a la pista de baile y a la música tenían la mayor "aptitud para la danza"? En mi opinión, no es más apropiado sacar conclusiones sobre la aptitud matemática de la falta de voluntad de los niños para dedicar muchos cientos de horas a hacer sumas.

La escuela construye aptitudes

Considere el caso de un niño que observé durante sus 8 y 9 años. Jim era un niño muy verbal y matofóbico de una familia profesional. Su amor por las palabras y por hablar se manifestó muy pronto, mucho antes de ir a la escuela. La matofobia se desarrolló en la escuela. Mi teoría es que vino como resultado directo de su precocidad verbal. Aprendí de sus padres que Jim había desarrollado un hábito temprano de describir con palabras, a menudo en voz alta, lo que sea que estuviera haciendo mientras lo hacía. Este hábito le causó dificultades menores con sus padres y maestros de preescolar. El verdadero problema vino cuando llegó a la clase de aritmética. Para entonces ya había aprendido a mantener bajo control el "hablar en voz alta", pero creo que seguía manteniendo su comentario de funcionamiento interno sobre sus actividades. En su clase de matemáticas estaba bloqueado: simplemente no sabía cómo hablar de hacer sumas. Carecía de un vocabulario (como la mayoría de nosotros) y de un sentido de propósito. De esta frustración de sus hábitos verbales creció el odio hacia las matemáticas, y del odio creció lo que las pruebas confirmaron más tarde como una mala actitud.

Para mí la historia es conmovedora. Estoy convencido de que lo que aparece como debilidad intelectual crece muy a menudo, como lo hizo Jim, a partir de las fortalezas intelectuales. Y no son sólo las fuerzas verbales las que socavan a los demás. Cada observador cuidadoso de los niños debe haber visto procesos similares trabajando en diferentes direcciones: por ejemplo, un niño que se ha enamorado del orden lógico está

preparado para ser apagado por la ortografía inglesa y para continuar desde allí y desarrollar una aversión global por la escritura.

El modelo de aprendizaje "disociado

La idea de Papert era que podíamos usar las computadoras como vehículos para escapar de la situación de Jim o de la lógica amorosa de los niños pero con problemas disléxicos. En ambos casos son víctimas de la aguda separación de nuestra cultura entre lo verbal y lo matemático. Se imagina un *Mathland* donde el amor y la habilidad de Jim para el lenguaje podrían ser movilizados para servir a su aprendizaje matemático formal en lugar de oponerse a él, mientras que para el otro tipo de niños, el amor por la lógica podría nutrir el interés en la lingüística. Los métodos de enseñanza prevalecientes dan a los estudiantes de matemáticas posibilidades limitadas para dar sentido a lo que están aprendiendo. En consecuencia, los niños se ven obligados a seguir el peor modelo de aprendizaje de las matemáticas, que es el aprendizaje memorístico, en el que el material carece de sentido. Es lo que Papert llamó un modelo *disociado*.

No sólo matemáticas....

Ya en un estudio de un año de duración que puso computadoras poderosas en las aulas de un grupo de alumnos de 7º grado "promedio", los estudiantes estaban trabajando en lo que ellos llamaban "poesía informática". Usaban programas de computadora para generar frases. Le dieron a la computadora una estructura sintáctica dentro de la cual hacer elecciones al azar a partir de listas de palabras dadas. El resultado es el tipo de poesía concreta que vemos en la siguiente ilustración. Una de las estudiantes, Jenny, de 13 años de edad, había conmovido profundamente al personal del proyecto al preguntar el primer día de su trabajo con la computadora: "¿Por qué elegimos un lugar para esto? Nosotros no somos el cerebro". El estudio había elegido deliberadamente a niños con un rendimiento escolar "medio". Un día Jenny llegó muy emocionada. Ella había hecho un Descubrimiento. "Ahora sé por qué tenemos sustantivos y verbos", dijo. Durante muchos años en la escuela Jenny había sido entrenada en categorías gramaticales. Nunca había entendido las diferencias entre sustantivos y verbos y adverbios. Pero ahora era evidente que la dificultad con la gramática no se debía a la incapacidad de trabajar con categorías lógicas. Era otra cosa. Simplemente no había visto ningún propósito en la empresa. No había sido capaz de entender de qué se trataba la gramática en el sentido de para qué podía ser. Y cuando le preguntó para qué era, las explicaciones que le dieron sus maestros le parecieron manifiestamente deshonestas. Dijo que le habían dicho que "la gramática te ayuda a hablar mejor".

(Papert, 1993, p.48)

Nota: Reporté este ejemplo porque lo encontraremos de nuevo, en forma diferente, entre las aplicaciones más avanzadas de Logo. Curiosamente, en los años 70 se necesitaba un ordenador potente y un equipo de investigación avanzado, hoy en día se puede hacer lo mismo con una simple implementación de Logo en un PC.

Ejemplo de poesía informática

Reporté este ejemplo porque lo encontraremos de nuevo, en forma diferente, entre las aplicaciones más avanzadas de Logo. Curiosamente, en los años 70 se necesitaba un ordenador potente y un equipo de investigación avanzado, hoy en día se puede hacer lo mismo con una simple implementación de Logo en un PC, a continuación se muestra un ejemplo realizado con LibreLogo.

Aquí tenéis un ejemplo citado por Papert (1993, p.49)

DEMENTE RETARDADO HACE PORQUE DULCE SNOOPY SCREAMSSEXY CHICA AMA ES POR ESO QUE LA DAMA SEXY ODIA ENORMEMENTE AL HOMBRE PORQUE EL PERRO FEO ODIA AL LOBO LOCO ODIA PORQUE EL LOBO DEMENTE SKIPSSEXY RETARDADO GRITA ES POR ESO QUE EL SEXY RETARDADO SNOOPY CORRE PORQUE EL LOBO GORDO LÚPULOWEET FOGINY SALTA A UNA DAMA GORDA CORRE

Doble discurso

Papert puso su argumento de una manera muy fuerte: a menudo los niños no pueden entender para qué son las matemáticas y la gramática porque perciben las explicaciones de los adultos como una doble charla.

Es fácil entender por qué las matemáticas y la gramática parecen tener sentido para los niños cuando no lo tienen para todos los que los rodean y por qué ayudar a los niños a darles sentido requiere más que un maestro que haga el discurso correcto o ponga el diagrama correcto en la pizarra. He preguntado a muchos profesores y padres qué pensaban que eran las matemáticas y por qué era importante aprenderlas. Pocos tenían una visión de las matemáticas que fuera lo suficientemente coherente como para justificar la dedicación de varios miles de horas de la vida de un niño a su aprendizaje, y los niños lo perciben. Cuando un profesor le dice a un estudiante que la razón de esas muchas horas de aritmética es poder comprobar el cambio en el supermercado, simplemente no se cree al profesor. Los niños ven estas razones como un ejemplo más de doble discurso adulto. El mismo efecto se produce cuando a los niños se les dice que las matemáticas escolares son "divertidas" cuando están bastante seguros de que los profesores que lo dicen pasan sus horas de ocio en cualquier cosa

excepto en esta supuesta actividad llena de diversión. Tampoco ayuda el decirles que necesitan las matemáticas para convertirse en científicos - la mayoría de los niños no tienen un plan así. Los niños pueden ver perfectamente bien que a la maestra no le gustan las matemáticas más de lo que a ellos les gustan y que la razón para hacerlo es simplemente que ha sido inscrito en el plan de estudios. Todo ello erosiona la confianza de los niños en el mundo de los adultos y en el proceso de educación. Y creo que introduce un profundo elemento de deshonestidad en la relación educativa.

El legado de la enseñanza de matemáticas como el problema QWERTY

Es importante tener en cuenta la diferencia entre las matemáticas -un vasto dominio de indagación cuya belleza rara vez es sospechada por la mayoría de los legos- y las matemáticas escolares. Esta última es una especie de construcción social, es decir, un conjunto de temas matemáticos determinados por una sucesión de circunstancias específicas. Un proceso que no garantiza, *per se*, el logro de un resultado óptimo. Recuerda la historia de la disposición del teclado QWERTY. QWERTY representa las primeras cinco teclas en las filas superiores. Este arreglo no tiene una explicación racional, sino sólo una explicación histórica. Se introdujo porque las llaves de las primeras máquinas de escribir tendían a atascarse. Por lo tanto, se dispusieron para reducir las colisiones separando las teclas que se sucedían con más frecuencia. La tecnología de las máquinas de escribir mejoró rápidamente y en pocos años el problema de las interferencias ya no era un problema, pero el sistema QWERTY se mantuvo. En este punto, demasiada gente hablaba con fluidez el diseño QWERTY y la producción de máquinas de escribir estaba demasiado lejos para dar un paso atrás en el rediseño de un diseño más racional, por ejemplo, agrupando las teclas más utilizadas.

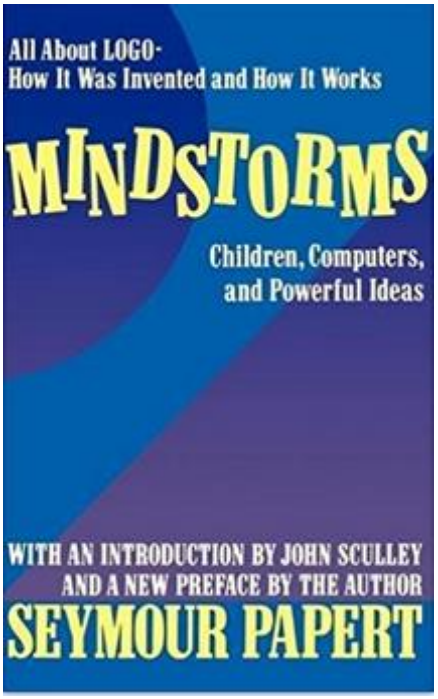
El problema del QWERTY es un buen ejemplo de cómo los hábitos consolidados pueden no ser necesariamente la mejor opción. Al igual que el diseño del QWERTY, las matemáticas escolares se moldearon en un contexto histórico diferente. De la misma manera, esta idea de las matemáticas se ha profundizado y, aún hoy en día, para la mayoría de la gente es inconcebible que las matemáticas puedan ser también otra cosa. Recuerdo a un conocido profesor de cálculo que, al comienzo del primer año del plan de estudios de matemáticas, exhortaba a sus alumnos a olvidar lo que habían aprendido en la escuela secundaria, ya que las matemáticas eran algo diferente.

Los principios de apropiabilidad de Papert

La geometría de la tortuga fue concebida para adaptarse a los niños y, en primer lugar, para ser *apropiada*. Podríamos describir este concepto por medio de algunos principios.

1. El *principio de continuidad*: el nuevo conocimiento matemático debe ser continuo con el existente, el que los niños tienen antes de ir a la escuela.
2. El *principio de poder}*: el nuevo conocimiento debe empoderar a los estudiantes para que realicen proyectos personalmente significativos, que no podrían realizarse sin él.
3. El *principio de la resonancia cultural*: los nuevos conceptos deben tener sentido para los niños en su contexto social. Irónicamente, incluso en el contexto social de los adultos: no debemos infligir a los niños algo que no hayamos comprendido a fondo y, desafortunadamente, con las "matemáticas básicas clásicas" este es el caso.

Seymour Papert. Tormentas mentales, niños, computadoras e ideas poderosas. Basic books, Nueva York, 2ª edición, 1993.



Un ejemplo de poesía informática con LibreLogo

Recordando el ejemplo de la poesía informática mostramos aquí algo similar al ejemplo reportado por Papert codificado en LibreLogo. Es una anticipación e incluso divergente. Primero no dijimos casi nada acerca de la codificación hasta ahora, segundo parece un poco raro usar un software orientado a gráficos, como LibreLogo, para construir un ejercicio de gramática. No te preocupes, por ahora sólo vuela sobre el ejemplo. Lo que estamos haciendo es mostrar cómo, hoy en día, con un simple sistema de software libre instalado en un pequeño ordenador es posible realizar algo que hace treinta años requería un potente ordenador dirigido por un equipo avanzado de científicos. Esto significa que las herramientas, incluso las bastante complejas, para introducir aplicaciones avanzadas de la tecnología en las escuelas están disponibles gratuitamente: herramientas gratuitas para una escuela verdaderamente democrática.

Un poco de limerick

Nuestro ejemplo es algo diferente ya que está concebido para producir una especie de limericks, una forma de verso de cinco versos, a menudo humorístico o sin sentido. El código está protegido por Stefano Penge 2017, publicado bajo Licencia Pública General v.3.0; ligeramente modificado por Andreas Formiconi (2018): Stefano Penge, *Lingua, codificación y creatividad. Fare coding con le materie umanistiche*, Anicia, Roma, 2018.

En este punto, querido lector, puede pasar directamente a la siguiente lección. Por si acaso tienes mucha curiosidad, vuela sobre el ejemplo tratando de comprender el panorama general. Volverás aquí más tarde, una vez que estés familiarizado con tales construcciones, si te gusta jugar con este tipo de código. El código completo de limerick se puede descargar desde este enlace. Aquí sólo comentaremos las piezas básicas.

En primer lugar, un ejemplo de salida:

Un elefante muy eficiente del
día de Algheroun
jugó tressette bajo un peroma
más allá de tres orerimases
en el fondoEse
soñoliento cirujano de Como

Que en inglés podría sonar como:

Un elefante muy eficiente de Algheroone un día jugó a la tressette bajo un peralpero pasó tres horas de pie a un ladoEse soñoliento cirujano de Como

El comando LIMERICK

El resultado se obtiene definiendo un solo comando nuevo - LIMERICK - mediante una *subrutina, de la* siguiente manera:

```
TO LIMERICKLABEL
 VERSO_1BACK
 12LABEL
 VERSO_2BACK
 12LABEL
 VERSO_3BACK
 12LABEL
 VERSO_4BACK
 12
ETIQUETA VERSO_5END
```

Aquí hay cinco comandos, cada uno para producir un verso, VERSE-1, VERSE-2 y así sucesivamente.....

El comando VERSE

Cada comando de versículo describe la estructura de la oración, por ejemplo para VERSE_1 que tenemos:

```
TO VERSE_1OUTPUT
 'Un' + _CHARACTER_ + _ADJECTIVE_ + 'di' + _PROVENANCE_END
```

La frase del primer verso está compuesta por el artículo (italiano)"Un", luego un carácter, un adjetivo, la preposición"di" y una procedencia.

El comando CHARACTER

Para elegir el carácter se define una subrutina CHARACTER adecuada

```
PARA SALIR DEL
 RANDOM [' signore ',' elefante ',' chirurgo ']
END
```

que elige entre los personajes"signore" (caballero),"elefante" (elefante) y"chirurgo" (cirujano). Del mismo modo, se podría aumentar el número de caracteres posibles. Del mismo modo para los otros tipos de palabras clave: adjetivos, porveniencias, etc.

Un buen ejercicio sería cambiar todo este código para producir oraciones en inglés, u otras variantes.

Sólo tome esto como un ejemplo de lo fácil que es navegar en una pieza de código de software libre, código LibreLogo aquí, y, posiblemente, para cambiarlo de acuerdo a sus deseos. Por supuesto que tienes que estudiar y entender algunos elementos básicos de la codificación, pero estamos aquí sólo para esto....

La Tortuga hace la tortuga

¿Hay lugar para el azar en las computadoras?

Ahora queremos dar a la Tortuga la capacidad de tomar decisiones, pero primero hagamos que el escenario sea más interesante.

Hasta ahora hemos representado una especie de visión determinista de la programación informática. En nuestro contexto esto significa dar órdenes claras a la Tortuga - hacer esto, hacer aquello. Una vez que el programa es escrito, el juego termina. No importa la complejidad, el dibujo está congelado en el código. Por lo tanto, ¿no hay lugar para el azar en la computadora?

Sí y no. Una explicación técnica para esta respuesta sería demasiado compleja en este caso. En una primera aproximación podemos decir que, no, una computadora no puede producir aleatoriedad verdadera sino una especie de pseudo-aleatoriedad, gracias a trucos matemáticos apropiados.

Básicamente, para producir aleatoriedad hay que ser capaz de generar números aleatorios, por medio de los llamados generadores de números aleatorios. En realidad, generan secuencias periódicas, en el sentido de que después de un cierto número de extracciones se vuelve a iniciar la misma secuencia inicial. El truco consiste en utilizar algoritmos que producen períodos extremadamente largos para que nunca se llegue al final de la secuencia por medio de extracciones sucesivas de números.

Por lo tanto, aparecen como números aleatorios verdaderos.

RANDOM

La Tortuga entiende el comando RANDOM:

X = RANDOM 100 ; devuelve un número de flotador aleatorio[*Nota*] ($0 \leq X < 100$), que es igual o mayor que 0 y menor que 100.

X = RANDOM "abcde" ; devuelve una letra aleatoria entre a, b, c, d, e

X = RANDOM[1, 2, 3] ; devuelve un elemento aleatorio entre 1, 2 y 3

También puede mezclar diferentes posiciones, por ejemplo

X = RANDOM[1, "pippo", 3.14]

Pero la ranodmnidad también está incrustada en la variable por defecto especial **ANY**, por ejemplo con

PENCOLOR CUALQUIERA

vas a usar un color de pluma al azar.

Nota: En informática, un número flotante es un número con dígitos decimales. Por ejemplo, 3.14 es un número flotante, 18 es un número entero.

Experimentar con RANDOM

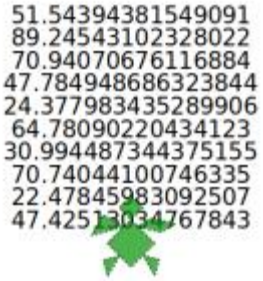
En los ejemplos anteriores, las opciones aleatorias se memorizan en la variable llamada X. Por supuesto, puede elegir el nombre que prefiera, o incluso utilizar las instrucciones de alguna manera diferente. Por ejemplo, intente jugar con el uso diferente del comando RANDOM por medio del siguiente código:

```
HOMECLEARSCREENPENUPFORWARD
 300REPEAT
 10[
ETIQUETA RANDOM 100BACK
 12
]
```

Con este código la Tortuga va primero al lado de la parte superior de la página, luego escribe una columna de diez números aleatorios sucesivos entre 0 y 100, imprimiéndolos mediante la instrucción LABEL.

En la siguiente diapositiva verás el resultado que tengo ahora mismo. Tenga en cuenta que no obtendrá la misma secuencia de números y eso es correcto! Después de todo se supone que son números aleatorios.... Y, por supuesto, puedes cambiar lo que quieras en este pedazo de código - siempre es bueno experimentar....

Una muestra aleatoria



A algo REPETIR nnn [Adelante xxx A LA DERECHA yyy] FIN

Recordemos algunos códigos

TR REPEAT 3 [FORWARD 100 RIGHT 120] END ; con esto se obtiene un triángulo llamando a **TR**

TO SQ REPEAT 4 [FORWARD 100 RIGHT 90] END; con esto se obtiene un cuadrado llamando a **SQ**

TO CR REPEAT 360 [FORWARD 1 RIGHT 1] END; with this you obtain a circle by calling **CR**

Estos son sólo ejemplos de un patrón de código muy recurrente:

A *algo* **REPETIR** *nnn* **[Adelante** *xxx A* **LA DERECHA** *yyy* **] FIN**

Con esto se obtiene "lo que sea" llamando a *algo*....

Comenzando con este patrón Abelson y diSessa elaboraron los teoremas básicos de la geometría de las tortugas, por ejemplo.

Desplazamiento por un importe indeterminado

Y por medio del mismo patrón vamos a darle vida a nuestra Tortuga. Dentro del ciclo de repetición, primero vamos a decirle a la Tortuga que se mueva hacia adelante por una cantidad aleatoria entre **1** y **2**. Para conseguir este efecto escribimos la instrucción

HACIA ADELANTE ALEATORIO(1) + 1

Aquí le decimos a la Tortuga que avance en una cantidad igual a **RANDOM(1) +1**. En la diapositiva anterior se mostró una sintaxis ligeramente diferente: **RANDOM 1** en lugar de **RANDOM(1)**. Sin embargo, ambos pueden utilizarse. La primera es la versión estándar de LibreLogo reportada en el manual oficial de la Barra de Herramientas de LibreLogo. La segunda es una versión alternativa que se basa en el

código Python subyacente mediante el cual se ha escrito LibreLogo. En el ejemplo que estamos discutiendo esta versión es preferible porque es más clara. En realidad, usando **RANDOM 1 + 1** no está claro si estamos dispuestos a sumar **1** al resultado de **RANDOM 1** o si queremos llamar **RANDOM 2**. Usando **RANDOM(1) + 1** no hay tal ambigüedad. Emitiendo el comando **RANDOM(1) + 1** obtenemos un número aleatorio entre **0** y **1**, por lo tanto, si sumamos **1** a esto, obtenemos un número aleatorio que está entre **1** y **2**.

En este caso, por ejemplo, los resultados de cinco ejecuciones sucesivas de **RANDOM(1) + 1**

1.71, 1.93, 1.73, 1.43, 1.99, 1.07, 1.26, 1.47, 1.27, 1.38

Volteo por una cantidad no determinada completamente

Luego giramos la Tortuga por una cantidad aleatoria, digamos entre -45 y 45 grados. Podemos lograr este resultado de manera similar, por medio de la instrucción

IZQUIERDO ALEATORIO(90) - 45

Con **RANDOM(90) - 45** obtenemos un número aleatorio entre **-45** y **45** grados.

Aquí los resultados de cinco corridas sucesivas de **RANDOM(90) - 45**:

37.8, 22.5, -12.1, -0.78, 2.3, -27.3, 34.0, 15.9, -26.7, 26.6

Inyectando aleatoriedad en algo REPEAT 360 [Adelante xxx A LA DERECHA yyy] FIN

Ahora ponemos los dos códigos "generadores aleatorios" en nuestro patrón básico "TO something REPEAT 360 [FORWARD xxx RIGHT yyyy] END"

TO **RANDOM_MOVE** REPEAT 100 [FORWARD **RANDOM(1) + 1** RIGHT **RANDOM(90) - 45**] END

donde se enfatizan las piezas de código inyectado. Además, escribimos aquí 100 repeticiones, pero esto se puede cambiar libremente, por supuesto. Si primero escribes este código en un documento de Libreoffice, entonces después de esto, abajo, añades el comando **RANDOM_MOVE** y dejas que la tortuga ejecute el código, obtienes un resultado similar a esta figura. Sólo intenta.... ¿estás tomando la misma foto? Quiero decir exactamente lo mismo.

El fascinante mundo de la simulación científica

Bueno, podría apostar que tu tortuga no viajó exactamente por el mismo patrón que la mía. A decir verdad, la probabilidad de obtener la misma imagen no es exactamente cero, pero es extremadamente baja: estoy bastante seguro....

¿Y qué? ¿Cómo podemos estar contentos de producir resultados impredecibles con una máquina concebida para realizar cálculos complejos y precisos? Bueno, en realidad, estamos muy contentos porque con este sencillo código abrimos una ventana al enorme y crucial mundo de la simulación científica. Hoy en día, en todos los campos científicos, la simulación es una herramienta de investigación esencial, la única posible para hacer frente a la abrumadora complejidad de los fenómenos naturales.

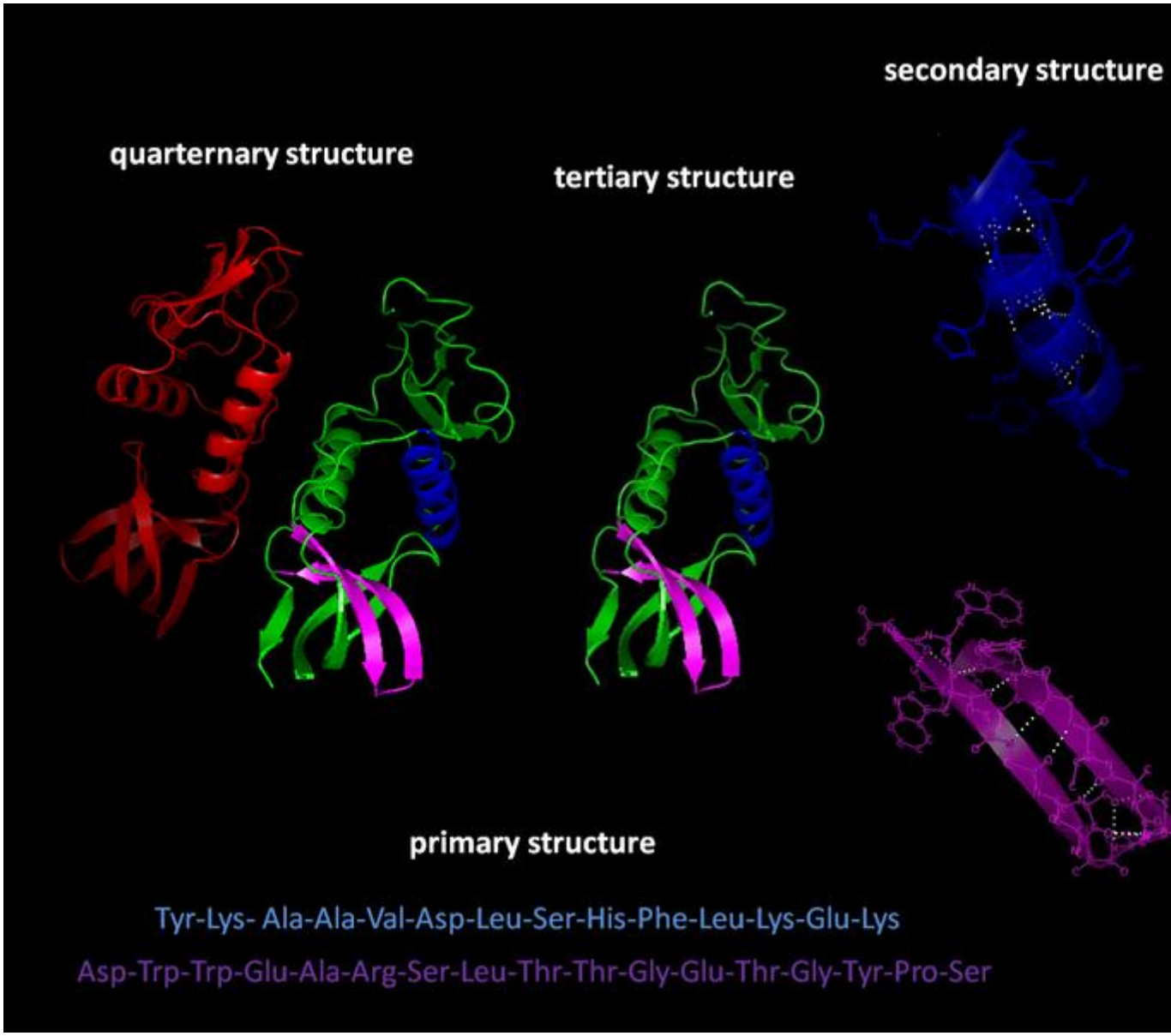
Sin embargo, "hoy en día" no significa exactamente hoy. Mi tesis de graduación en física, en los años setenta, se basó en gran medida en el uso de la simulación por ordenador de la radiación dispersa en el cuerpo humano. Un contexto demasiado complejo para enfrentarse únicamente con las herramientas del análisis matemático. En física, las simulaciones por ordenador utilizan los llamados métodos de *Monte Carlo*, una amplia clase de algoritmos computacionales que se basan en muestreos aleatorios repetidos para obtener resultados numéricos. Por ejemplo, en mi trabajo de tesis, simulé una fuente de radioactividad dentro del cuerpo, reproduciendo la emisión de fotones, invocando así un generador de números aleatorios en cualquier paso aleatorio, como la dirección de la emisión, la interacción con la materia, la dirección de la desviación, etc.

Así, "hoy" puede significar también hace cuarenta años....

Simulación y complejidad

Pero las simulaciones no se limitan al ámbito de la física, al contrario, cuanto más complejo es el fenómeno, más simulaciones pueden resultar ser la única herramienta de investigación posible. En el contexto de las disciplinas biológicas hablamos de experimentos *in silico*, refiriéndonos al hecho de que se realizan a

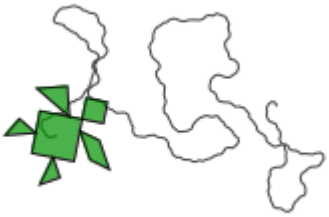
través de cálculos realizados con ordenadores digitales que funcionan con chips de silicio - la expresión es una alusión a las frases latinas *in vivo*, *in vitro* e *in situ*, comúnmente utilizadas en biología. Hoy en día, los experimentos *in silico* incluyen biología molecular, ensayos genéticos, crecimiento tumoral, dermatología, remodelación ósea, insuficiencia orgánica, ensayos clínicos, entre otros. En medicina, por ejemplo, los estudios *in silico* se utilizan para descubrir nuevos medicamentos porque es más rápido y cuesta mucho menos. En biología se utilizan para estudiar la formulación de modelos de comportamiento de las células y en genética para analizar la expresión génica (en biología molecular, "expresión génica" significa la forma en que un conjunto de genes determina el funcionamiento de la célula a nivel macromolecular).



¿Qué estamos simulando aquí?

¿Pero qué estamos simulando con nuestro trozo de código? La respuesta obvia es una tortuga vagando al azar. Por supuesto, si imaginamos observar a un animal desde la altura en su entorno, la impresión será la de un movimiento aleatorio. En este sentido, nuestro código podría considerarse una simulación del comportamiento del animal, aunque sea muy pobre, ya que nuestro supuesto de aleatoriedad se basa en el hecho de que ignoramos lo que el animal está haciendo en realidad.

Los animales se mueven porque buscan comida o pareja, por ejemplo. O, quizás, la simulación podría ser más apropiada para una persona borracha que anda por ahí en la noche, buscando una felicidad inexistente. La distribución de alimentos para una especie dada de animales en su ambiente natural podría ser modelada, hasta cierto punto, pero la lucha interna de un hombre es mucho más difícil de traducir en términos matemáticos.



¿Qué queremos decir con"modelar algo"?

Pero ¿qué significa exactamente "modelar algo", como la distribución de alimentos, en el contexto de nuestro ejemplo? Sí, aparentemente, inyectamos una gota de vida en nuestro código, reproduciendo una aleatoriedad que todos experimentamos en nuestra vida, en forma de buena o mala suerte. Sin embargo, a pesar de esta fuerte percepción de incertidumbre, todos sabemos que nuestra capacidad de observar el mundo exterior y de tomar decisiones es también crucial.

En esencia, nuestra tortuga es totalmente pasiva! Sólo tiene algo de energía para proceder y girar: sin olor, sin vista, sin decisiones. Este es el punto: ¡necesitamos una forma de dejar que la Tortuga tenga la capacidad de tomar decisiones!

Condición IF [bloque verdadero] [bloque falso]

En ejemplos anteriores hemos visto códigos que se ejecutan secuencialmente. Sí, acabamos de introducir algún comportamiento aleatorio en la vida de la tortuga, pero la secuencia de instrucciones siempre ha sido rígidamente predeterminada, hasta ahora. Ahora vamos a mostrar la forma de cambiar la secuencia de instrucciones dependiendo del estado del proceso. Hay un comando específico en LibreLogo para hacer esto:

Condición **IF [** *bloque verdadero*] **[** *bloque falso*]

lo que significa: si se verifica la *condición* dada, ejecutar el primer bloque de instrucciones (*bloque verdadero*), en caso contrario, ejecutar el segundo (*bloque falso*). Veamos un par de ejemplos a la vez:

SI a < 10 [IMPRIMIR "Pequeño"]SI
a < 10 [IMPRIMIR "Pequeño"] [IMPRIMIR "Grande"]

En la primera línea, si la variable **a** contiene un número menor que 10, entonces la tortuga imprimirá la palabra "Small".

En la segunda línea, si la variable **a** contiene un número menor que 10, entonces la tortuga imprimirá la palabra "Pequeño", de lo contrario (una mayor o igual a 10) imprimirá la palabra "Grande".

Ve e inténtalo, y recuerda: ¡la mejor manera de aprender a codificar es siempre explorando por ti mismo!

Más sobre las condiciones

Las sentencias condicionales utilizan operadores relacionales:

- **==** igual
- < menor que
- <= inferior o igual a
- > mayor que
- >= mayor que o igual a
- **!=** no es igual a

Las condiciones pueden ser construidas de maneras bastante complejas usando operadores lógicos: **Y**, **O**, **NO**. Por ejemplo:

SI a < 10 Y NO **a** = 5 [IMPRIMIR "0, 1, 2, 3, 4, 6, 7, 8 o 9"]SI
a < 10 Y a **!=** 5 [IMPRIMIR "0, 1, 2, 3, 4, 6, 7, 8 o 9"]].

Reflexiona, intenta jugar...

Preparación para la toma de decisiones....

Dicho esto, ampliemos la línea de código que leímos antes:

TO **RANDOM_MOVE** REPEAT 100 [FORWARD **RANDOM(1) + 1** RIGHT **RANDOM(90) - 45**] END

A **RANDOM_MOVEREPEAT**
100[
FORWARD **RANDOM(1) + 1RIGHT RANDOM(90) - 45**

END (FIN)

Para ejecutar este código tienes que escribir debajo de **RANDOM_MOVE**

Bien, lo que vamos a hacer ahora es dejar que la tortuga se pregunte sobre su estado en relación a un estímulo positivo, el cual asumimos que es causado por la presencia de alimento sme food en alguna parte. En la siguiente diapositiva vamos a ver el código completo, luego lo discutiremos pieza por pieza.

La tortuga está oliendo comida en alguna parte.

1. **A OLER**
2. **PF =[350,350]** ; **PF** es el punto donde hemos puesto la comida, digamos que alguna ensalada
3. **PENUP POSICIÓN PF RÚBRICA PENDIDA 0 ETIQUETA "ALIMENTOS"** ; escriba "ALIMENTOS" en[350,350]
4. **PENUP HOME PENDOWN** ; volver a casa, en el centro de la página
5. **P1 = POSICIÓN** ; colocar la posición actual en la variable **P1**
6. **FORWARD 1** ; un pequeño paso adelante, sólo para inicializar variables
7. **P2 = POSICIÓN** ; colocar la posición actual en la variable **P2**
8.
9. **REPEAT 100 [** ; comenzar iteraciones....
10. **RANDOM(1) + 1** ; un paso adelante, entre **1** y **2**
11. **D1 = SQRT((P1[0]-PF[0])**2 + (P1[1]-PF[1])**2)**; distancia entre **P1** y **PF**
12. **D2 = SQRT((P2[0]-PF[0])**2 + (P2[1]-PF[1])**2)**; distancia entre **P2** y **PF**
13. **IF D1 < D2 [A LA DERECHA 20] ;** si la distancia del alimento desde el punto viejo es menor que desde el nuevo punto, gire 20°.
14. **P1 = P2** ; el nuevo punto se vuelve viejo
15. **P2 = POSICIÓN** ; fijar nuevo punto
16. **]**
17. **FIN**

Descripción del procedimiento de Olfateo

Los números a la izquierda indican los números de las instrucciones en la lista de diapositivas anteriores.

2-7: inicialización.

2-4: La posición del alimento **PF** está ajustada.

5-7: Se ajustan los puntos **P1** y **P2**, que tienen algún valor significativo al entrar en el ciclo de repetición (9-16). En cada iteración, **P1** es la ubicación anterior de la tortuga, **P2** es la nueva.

9: Comienza el ciclo de repetición. El número de iteraciones es irrelevante. Siempre se puede detener el programa con el botón rojo de la barra de herramientas de LibreLogo.

10: **RANDOM(1) + 1** ; un paso adelante, longitud aleatoria entre **1** y **2**

11: **D1 = SQRT((P1[0]-PF[0])**2 + (P1[1]-PF[1])**2)** ; cálculo de la distancia entre la posición previa de la tortuga **P1** y la posición de alimento **PF** con el teorema de Pitágoras

12: **D2 = SQRT((P2[0]-PF[0])**2 + (P2[1]-PF[1])**2)** ; cálculo de la distancia entre la nueva posición de tortuga **P2** y la posición de alimento **PF** con el teorema de Pitágoras

13: **SI D1 < D2 [A LA DERECHA 20] ;** si la distancia del alimento D1 desde el punto viejo es menor que la distancia D2 desde el punto nuevo, entonces gire en 20, de lo contrario no haga nada.

14: **P1 = P2** ; por último, preparándose para la siguiente iteración, el nuevo punto se convierte en el anterior

15: **P2 = POSICIÓN** ; el nuevo punto se ajusta a la posición actual a la que llega la tortuga

16: Ciclo de repetición de fin de partida

Ahora puedes invocar a SMELLING cuando quieras.

Mejorar el modelado

Sin duda, hicimos a la tortuga un poco consciente del entorno pero su comportamiento parece un poco estereotipado, siendo capaz de reaccionar con sólo girar a la derecha. Esto podría considerarse una forma elegante de dibujar una espiral (hay muchas otras), pero queríamos explorar el modelado aquí.

Por lo tanto, en la siguiente diapositiva vamos a ver un ejemplo más sofisticado, aumentando la cantidad de aleatoriedad en el comportamiento de la tortuga.

Interesante que para aproximar mejor la vida tengamos que aumentar los ingredientes caóticos, ¿no es así?

La tortuga sigue oliendo la comida pero se pone más animada

1. A OLER
2. PF =[350,350] ; PF es el punto donde hemos puesto la comida, digamos que alguna ensalada
3. PENUP POSICIÓN PF RÚBRICA PENDIDA 0 ETIQUETA "ALIMENTOS" ; escriba "ALIMENTOS" en[350,350]
4. PENUP HOME PENDOWN ; volver a casa, en el centro de la página
5. P1 = POSICIÓN ; colocar la posición actual en la variable P1
6. FORWARD 1 ; un pequeño paso adelante, sólo para inicializar variables
7. P2 = POSICIÓN ; colocar la posición actual en la variable P2
8.
9. REPEAT 100 [; comenzar iteraciones....
10. RANDOM(1) + 1 ; un paso adelante, entre 1 y 2
11. D1 = SQRT((P1[0]-PF[0])**2 + (P1[1]-PF[1])**2); distancia entre P1 y PF
12. D2 = SQRT((P2[0]-PF[0])**2 + (P2[1]-PF[1])**2); distancia entre P2 y PF
13. **RANDOM DERECHO(40) - 20 ;** un nuevo componente de giro aleatorio, entre -20° y 20°.
14. IF D1 < D2 [A LA DERECHA 20] ; si la distancia del alimento desde el punto viejo es menor que desde el nuevo punto, gire 20°.
15. P1 = P2 ; el nuevo punto se vuelve viejo
16. P2 = POSICIÓN ; fijar nuevo punto
17.]
18. FIN

Lo que ha cambiado con respecto a lo que olía anteriormente....

Sólo se ha añadido una instrucción aquí:

13:DERECHA ALEATORIO(40) - 20

Este es un nuevo componente de giro aleatorio, comprendido entre -20° y 20°, que añadimos antes del giro a la **derecha 20** aplicado condicionalmente sólo si la tortuga se estaba acercando a la localización de alimento:

14: SI D1 < D2 [DERECHA 20]

Debe quedar claro, a partir de estos ingenuos experimentos, que existen innumerables posibilidades para explorar el modelado, comenzando por cambiar el equilibrio entre el giro fijo y el aleatorio en este código.

Exploraciones de modelado adicionales

Lo que hemos modelado antes es el sentido del olfato, que es algo que sólo depende de la distancia. Podríamos intentar modelar la vista, que es diferente del olfato porque es direccional: sólo se ve si se mira en una dirección determinada, mientras que con el olfato se percibe independientemente de la dirección. Podríamos intentar configurar un modelo de dos ojos, o incluso tener en cuenta la intensidad del estímulo. Podemos poner obstáculos que hay que sortear para llegar a los objetivos.

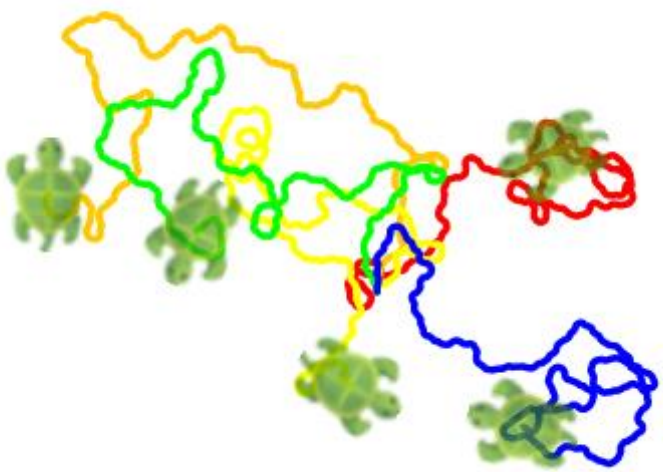
Otro campo de exploración es el de las poblaciones, para el estudio de la presa depredadora u otras dinámicas. Sin embargo, en este caso necesitamos la habilidad de correr múltiples tortugas simultáneamente y con la versión actual de LibreLogo esto no es posible. Hay muchas versiones de Logotipos, y algunas de ellas permiten correr más de una tortuga a la vez. Como ya hemos dicho, la razón para usar LibreLogo como la primera, aunque no trivial, introducción al Logo se basa en su piso extremadamente bajo, lo cual es útil en la escuela primaria.

Como ejemplo, extendamos el primer ejemplo mostrado en esta lección (diapositiva 10) a múltiples tortugas. Vamos a usar otro entorno de Logotipo que es Kojo, primero vamos a ver el código, que realiza exactamente el mismo comportamiento que se muestra en la diapositiva 10 pero para cinco tortugas totalmente independientes. Comentaremos el código más adelante.

Código Kojo para cinco tortugas vagabundas

```
clear()

def path(t: Turtle, delay: Int) = runInBackground {
    t.setAnimationDelay(delay)
    repeat(10000) {
        t.forward(random(1) + 1)
        t.right(random(90) - 45)
    }
}

val t1 = newTurtle(0, 0)
t1.setPenColor(red)
val t2 = newTurtle(0, 0)
t2.setPenColor(orange)
val t3 = newTurtle(0, 0)
t3.setPenColor(yellow)
val t4 = newTurtle(0, 0)
t4.setPenColor(green)
val t5 = newTurtle(0, 0)
t5.setPenColor(blue)

path(t1, 1000)
path(t2, 1000)
path(t3, 1000)
path(t4, 1000)
path(t5, 1000)
```

Cinco tortugas errantes



Diccionario LibreLogo-Kojo mínimo

Action	LibreLogo	Kojo
Move forward by 100	FORWARD 100	forward(100)
Move back by 100	BACK 100	back(100)
Turn left by 90°	LEFT 90	left(90)
Turn right by 90°	RIGHT 100	right(90)
Raise the pen	PENUP	penUp()
Lowers the pen	PENDOWN 100	penDown()
Moves to new position of coordinates (100,200) drawing a line	POSITION [100,200]	changePosition(100, 200)
Clears the screen and goes home	CLEARSCREEN HOME	clear()
Hides the Turtle	HIDETURTLE	invisible()
Shows the TURTLE	SHOWTURTLE	visible()

Repetition	REPEAT 4 [
	FORWARD 100
	RIGHT 90
	]
		repeat(4) {
		forward(100)
		right(90)
		}

Definition of procedures ("functions" in Kojo) example shows a procedure for drawing a square of side 50	TO MYSQ SIDE
	REPEAT [
	FORWARD SIDE
	RIGHT 90
	]
	END
	
	MYSQ 50
		def mysq(side: Int) {
		repeat {
		forward(side)
		right(90)
		}
		}
		
		mysq(50)

Objetos de tortuga en Kojo

El logotipo no distingue entre mayúsculas y minúsculas, puede escribir comandos en mayúsculas o minúsculas, indistintamente. Suelo mostrar ejemplos escritos en mayúsculas porque parece ser más fácil para los principiantes. En cambio, Kojo distingue entre mayúsculas y minúsculas y los comandos están escritos en estilo "camel-case", lo que significa que las mayúsculas y minúsculas pueden mezclarse en la misma palabra y que ese mayúscula es relevante: por ejemplo, **penUp()** es correcto y **penup()** no funciona. Los paréntesis son algo diferentes, como se puede ver en la diapositiva anterior, pero la correspondencia entre los comandos de las tortugas es bastante fácil de reconocer. Sin embargo, el código Kojo parece algo complejo. En realidad, se basa en el lenguaje Scala, que es un lenguaje moderno muy poderoso. Esto tiene un cierto costo en complejidad.

Por ejemplo, nuestro diccionario LibreLogo-Kojo mínimo carece de una construcción específica:

val t1 = nuevoTurtle(0, 0)

que ha utilizado cinco veces, para **t1**, **t2**, **t3**, **t4** y **t5**. Bueno, estas son las cinco tortugas. Este tipo de instrucción es peculiar de los llamados *lenguajes orientados a objetos* y representa a un *constructor*. La expresión **val** significa que un nuevo objeto de tipo tortuga será creado y asignado a la variable **t1**. De manera similar, para **t2**.... **t5**. ser un objeto significa tener un número de características: constantes numéricas o literales, variables, funciones específicas, etc.

Kojo funciona en todo tipo de sistemas operativos, estando escrito en Java. Se puede descargar desde el [sitio de la fundación kogics](#).

Descripción del código Kojo de cinco tortugas

1: **clear()** ; clear screen

3-9: **ruta def(t: Tortuga, retraso: Int) = ruta de** función (procedimiento en LibreLogo). Para iniciar una tortuga tienes que invocar **el sendero**, ver la invocación de los cinco **senderos** en las instrucciones 22-26. La definición de la función de **trayectoria** toma argumentos: un objeto **t** del tipo **Tortuga** y un **retardo** numérico del tipo **Int**, es decir, un número entero. El número de **retardo** es para controlar la velocidad de la Tortuga: cuanto mayor sea el número, más lenta será la velocidad. El argumento del tupé de **tortuga** sirve para iniciar diferentes tortugas, como en las instrucciones 22-26.

5-8: mismo ciclo de repetición de la versión Logotipo.

11-20: construyendo cinco objetos diferentes de **Tortuga** y colocándolos en cinco colores diferentes.

22-26: inicio de las cinco tortugas invocando la función de **ruta** en las diferentes cinco tortugas, todas con retraso 1000. El retraso es la cantidad de tiempo (en milisegundos) que tarda la tortuga en moverse a través de una distancia de cien pasos.

Recapitulación

Recapitulando, puede ser que algunos lectores encontraran extraño el entorno del Kojo con respecto al del Logo. Hemos puesto esta parte para que conozca todo el potencial del lenguaje del Logotipo, presentando también una implementación más sofisticada. La discusión sobre el modelado del comportamiento animal fue una buena oportunidad para captar algunas de las características más avanzadas. Sin embargo, a continuación seguiremos usando LibreLogo porque, para los niños más

pequeños, el aspecto de "piso bajo" de un lenguaje de software educativo es importante.

Geometría de la tortuga

Todos conocemos la existencia de la geometría euclidiana, y también de la geometría cartesiana. ¿Son diferentes? Por supuesto, son muy diferentes. ¿Cuál es la verdadera geometría? Esta pregunta es absurda. La geometría euclidiana y la geometría cartesiana son perfectamente ciertas. Son diferentes perspectivas de los mismos objetos geométricos. Cuál es la que usamos es una cuestión de conveniencia en la lucha contra el problema en cuestión. Existen otras geometrías (no euclídeas, diferenciales...) pero basta con centrarse en éstas, por el bien de nuestra discusión sobre la geometría de las tortugas. Sí, porque lo que queremos mostrar es que esta ene es una geometría como las otras y que es la más adecuada para introducir y dejar experimentar algunos conceptos geométricos a los niños.

Veamos cómo el objeto más fundamental de la geometría está representado en estas tres geometrías.

El punto

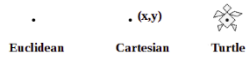
En la geometría euclídea el punto es la única entidad sin dimensión, o el "objeto sin partes".

En la geometría cartesiana un punto en un plano es dado por un par ordenado de números (x, y)... en el espacio tridimensional es dado por tres números (x, y, z)... y así sucesivamente, en los espacios n-dimensionales por $(x_1, x_2, \ldots, x_n)$ números. Centrándonos en el plano, donde seguiremos trabajando dentro de este curso (la tortuga también puede trabajar en espacios más complejos), un punto **es** un par de números, es decir, para actuar sobre un punto, hay que trabajar sobre los dos números que lo representan.

En la geometría de la tortuga el punto es... ¡la tortuga!

No es una broma: la tortuga se describe completamente una vez que conocemos su posición y su rumbo. La expresión "completamente descrita" nos permite pensar en la noción de **estado**, una noción básica de la física. En realidad la tortuga tiene un estado, hecho de posición y rumbo. El hecho de que el objeto fundamental se caracterice por tal estado es lo que hace que la geometría de la tortuga sea peculiar con respecto a las demás. Además, es lo que hace que sea apropiado ser utilizado para enseñar geometría a los niños, porque trabajar con la tortuga nos permite

pensar en términos de movimiento y, en el caso de los niños, identificarnos con el animal.

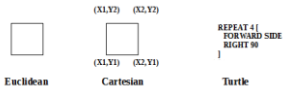


La plaza

En la geometría euclídea el cuadrado es una figura plana con cuatro lados iguales y cuatro ángulos iguales.

En la geometría cartesiana, un cuadrado se define por cuatro pares de números, tales como $[(x_1, y_1), (x_2, y_1), (x_2, y_2), (x_1, y_2)]$. A decir verdad, se trata de un cuadrado con lados paralelos al eje del sistema de referencia, para un cuadrado orientado en cualquier dirección, la respuesta es mucho más compleja y nos sacaría de contexto.

¿Y con la tortuga? Qué extraña es la expresión en la figura! Bueno, es simplemente un trozo de código, el código del Logotipo que tienes que escribir en LibreLogo para obtener un cuadrado. ¿Pero cómo podemos comparar una expresión como $[(x_1, y_1), (x_2, y_1), (x_2, y_2), (x_1, y_2)]$ésta con este código? Esa es la belleza de las matemáticas: la capacidad de trabajar con los objetos más extraños, una vez que se ha establecido un conjunto perfecto de reglas con las que trabajar. No te preocupes por entender ese código ahora mismo. Sólo enfócate en el hecho de que, en el reino de la geometría de la tortuga, ese trozo de código representa un cuadrado, e incluso de una manera más general con respecto a la expresión $[(x_1, y_1), (x_2, y_1), (x_2, y_2), (x_1, y_2)]$, porque es perfectamente adecuado para representar un cuadrado en cualquier orientación posible. Volveremos sobre este tema más adelante.

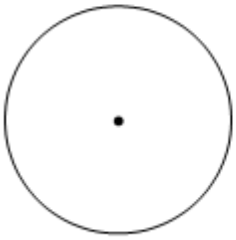


El círculo

En la geometría euclídea el cuadrado es una figura plana con cuatro lados iguales y cuatro ángulos iguales.

En la geometría cartesiana, un cuadrado se define por cuatro pares de números, tales como $[(x_1, y_1), (x_2, y_1), (x_2, y_2), (x_1, y_2)]$. A decir verdad, se trata de un cuadrado con lados paralelos al eje del sistema de referencia, para un cuadrado orientado en cualquier dirección, la respuesta es mucho más compleja y nos sacaría de contexto.

¿Y con la tortuga? Qué extraña es la expresión en la figura! Bueno, es simplemente un trozo de código, el código del Logotipo que tienes que escribir en LibreLogo para obtener un cuadrado. ¿Pero cómo podemos comparar una expresión como $[(x_1, y_1), (x_2, y_1), (x_2, y_2), (x_1, y_2)]$ ésta con este código? Esa es la belleza de las matemáticas: la capacidad de trabajar con los objetos más extraños, una vez que se ha establecido un conjunto perfecto de reglas con las que trabajar. No te preocupes por entender ese código ahora mismo. Sólo enfócate en el hecho de que, en el reino de la geometría de la tortuga, ese trozo de código representa un cuadrado, e incluso de una manera más general con respecto a la expresión $[(x_1, y_1), (x_2, y_1), (x_2, y_2), (x_1, y_2)]$, porque es perfectamente adecuado para representar un cuadrado en cualquier orientación posible. Volveremos sobre este tema más adelante.

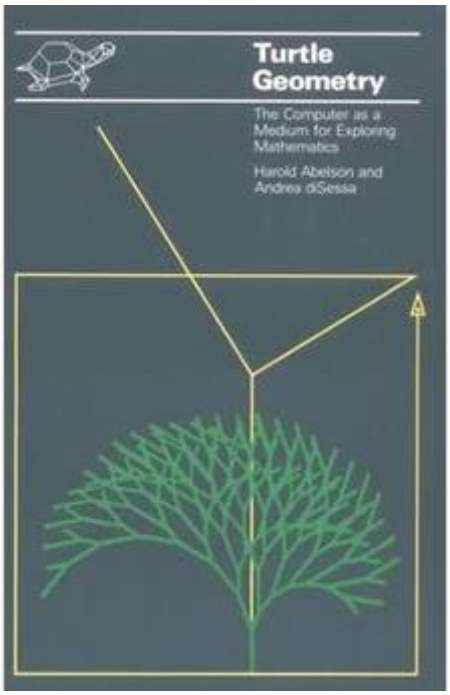


$$x^2 + y^2 = r^2$$

```
REPEAT 360 [
  FORWARD 1
  RIGHT 1
]
```

Geometría de la tortuga - H. Abelson y A. diSessa

La geometría de las tortugas fue descrita a fondo por Harold Abelson y Andrea diSessa en Turtle Geometry - The Computer as a Medium for Exploring Mathemathics, MIT press, Cambridge, 1986. Este es un libro de matemáticas a nivel universitario, lleno de teoremas y ejercicios. Lo que es asombroso es que fue escrito cuando las computadoras apenas entraban en cada casa, en los años ochenta. Tal vez, algunos de ustedes, leyendo los primeros comandos de Logo, pensaron que este es un programa muy básico para niños. Por supuesto que lo es, pero está respaldado por una base teórica muy sólida. Con las palabras de Papert: tiene *un piso bajo y un techo alto*.



Geometría de la tortuga - contenido

Es interesante explorar el índice de este libro de 477 páginas:

- introducción a la geometría de la tortuga
    - Gráficos de tortugas
    - POLYs y otros caminos cerrados
    - Programas de bucle
    - Simetría de los programas en bucle
- Retroalimentación, crecimiento y forma
    - La tortuga como animal
    - Tortugas interactuando
    - Crecimiento
- Métodos vectoriales en la geometría de las tortugas
    - Análisis vectorial de programas vectoriales
    - Coordenadas para vectores
    - Implementación de gráficos vectoriales de tortugas en una computadora
    - Maniobrando una Tortuga Tridimensional
    - Mostrando una Tortuga Tridimensional
- Topología de los senderos de las tortugas
    - Deformación del trayecto cerrado
    - Deformación de curvas y planos
    - Corrección del Algoritmo de la Promesa
- La tortuga escapa del avión
    - Geometría de la tortuga en una esfera
    - Curvatura
    - Curvatura y topología total

- Explorando el Cubo
    - Un cubo de ordenador
    - Observaciones y preguntas sobre los cubos
    - Resultados
- Una segunda mirada a la esfera
    - Una simulación por computadora
    - Explorando
    - Resultados
- Superficies Planas Piecewise
    - Un Programa para Superficies Planas a Pedazos
    - Orientaciones
    - Curvatura y características de Euler
- Espacio curvo y relatividad general
    - Representaciones de cuña
    - Fenómenos de espacio y tiempo curvos
    - La Teoría General de la Relatividad
    - Un simulador de la relatividad general

Increíble, ¿verdad? A continuación nos referiremos sólo a algunos de estos contenidos porque algunas partes son bastante complejas. Sin embargo, lo que queremos enfatizar aquí es el potencial vertical extendido de Logo:f usado inteligentemente, puede jugar un papel significativo en el apoyo al aprendizaje de las matemáticas desde la edad de los niños hasta el nivel universitario.

Los comandos básicos de la Tortuga

Incluso en una simple implementación como el LibreLogo hay muchas instrucciones disponibles, mucho más allá de las que necesita Turtle Graphics. Por ejemplo, no se han usado más de una docena de instrucciones en todo el libro de Geometría de Tortugas. Con una perspectiva minimalista, se pueden realizar proyectos bastante complejos mediante comandos, FORWARD y RIGHT, suponiendo que se utilicen números negativos, por ejemplo, distancias negativas para retroceder y ángulos negativos para girar a la izquierda. Como estamos interesados en trabajar con niños, es apropiado introducir un comando de giro a la izquierda, que es IZQUIERDA. Si desea también un comando BACK, incluso si esto es menos importante.

Pero para explotar todo el potencial de los gráficos de Turtle, vamos a anticiparnos brevemente a las construcciones básicas del software, una que nos permite repetir cualquier secuencia de instrucciones cuantas veces queramos y la otra que nos da

la habilidad de incapsular secuencias de instrucciones en nuevos comandos. Volveremos sobre estas construcciones más adelante.

Repetir secuencias de instrucciones: REPETIR

Ya hemos visto que se puede obtener un cuadrado con el siguiente código:

ADELANTE 100 A LA DERECHA 90ADELANTE
100 A LA DERECHA 90ADELANTE 100 A LA DERECHA
90ADELANTE
100 A LA DERECHA 90

Bajar cuatro veces la misma instrucción no es tan malo pero imagina tener una secuencia mucho más larga, pueden ser cientos o miles de repeticiones. Incluso mediante un uso intensivo de cortar y pegar puede resultar en una pesadilla. La instrucción REPEAT está hecha para resolver estas situaciones:

REPITE 4[
ADELANTE 100 DERECHA 90
]

O, si lo prefiere, puede escribirlo en una sola línea:

REPETIR 4 [ADELANTE 100 DERECHA 90]

Sólo inténtalo.

Sintaxis REPEAT

Así que la sintaxis es: primero escribe REPEAT, luego el número de repeticiones - si no lo especificas, repetirá el código *ad libitum*. Finalmente, incrusta el código que tiene que ser repetido dentro de un par de corchetes.

El hecho de que utilicemos aquí corchetes es específico de la implementación de LibreLogo. Por ejemplo, en la versión Kojo de Logo, se usarán barcas rizadas {}.

en la versión Logo utilizada en el libro Turtle Geometry los bloques de código a repetir no están entre corchetes sino que están sangrados. Una sintaxis utilizada también en el lenguaje Python más reciente. Así que, verá, la sintaxis puede ser algo diferente entre las lenguas, pero la racionalidad es la misma.

LibreLogo

Kojo

Lenguaje del logotipo del Libro de Geometría de Tortugas

Procedimientos: A ..... FIN

Esta es una de las construcciones más poderosas en todos los lenguajes de programación. En esencia, ponga una envoltura alrededor de un montón de instrucciones y déle un nombre, luego use ese nombre como una orden. En LibreLogo estas piezas de código encapsuladas se llaman procedimientos. Se puede poner cualquier cantidad de código dentro, llamadas a otros procedimientos incluidos. Cuando se trabaja con niños, se les puede decir que, cada vez que la Tortuga encuentra una instrucción TO, se sienta, sigue escuchando hasta la siguiente instrucción END, memorizando toda la secuencia y su nombre. Luego, cuando más adelante en el siguiente paso encuentre un comando con ese nombre, ejecutará la secuencia memorizada. Los niños aceptan muy bien esta explicación y de esta manera se exponen al paradigma de la encapsulación, fundamental en la programación pero también en las matemáticas, en la figura se han enfatizado las dos ocurrencias del nombre **SQ**, la primera vez cuando se define en la frase TO y la segunda vez cuando se utiliza. Intente hacer variaciones en este ejemplo.

A LA
DERECHA 90END
.

SQ

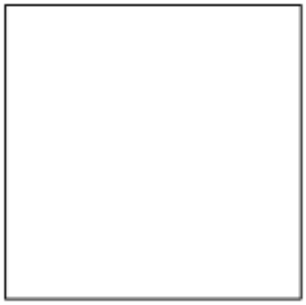
Variables

Hasta ahora hemos visto algunos comandos que requieren *argumentos*. El argumento es el valor que un comando puede requerir para ser ejecutado. Por

ejemplo, FORWARD no tendría ningún sentido sin un argumento que represente la distancia que la tortuga necesita cubrir. La expresión FORWARD 50 significa que la tortuga tiene que avanzar 50 puntos; 50 es el valor del argumento. También encontraremos comandos que requieren más de un argumento. Básicamente, en todos los ejemplos anteriores usamos argumentos numéricos. En realidad, cada idioma permite comprender "variables". Son nombres simbólicos a los que se les pueden asignar valores numéricos convenientes. Intentemos ejecutar el siguiente código:

LATO = 100ANGOLO
= 90

FORWARD LATO LEFT ANGOLOFORWARD
LATO LEFT ANGOLO
FORWARD LATO LEFT ANGOLOFORWARD
LATO LEFT ANGOLO



Variables, como en álgebra

Hemos dibujado un cuadrado, pero en lugar de usar directamente el valor 100 como argumento de las instrucciones FORWARD, primero asignamos el valor 100 a la variable llamada "SIDE" y luego lo usamos como argumento de todas las instrucciones FORWARD. La utilidad de este método es evidente: supongamos que no estoy satisfecho con las dimensiones de este cuadrado y que quiero probar otros valores del lado. Bueno, sólo necesito cambiar el valor 100 por la instrucción SIDE=100, cambiándolo por ejemplo con SIDE=150. Siéntete libre de experimentar! Incluso puede que quieras cambiar el valor de un ANGLE, probando diferentes valores....

Para aquellos que estudiaron los fundamentos del álgebra, ciertamente habrán reconocido el concepto de variable, que en esa disciplina se usa ampliamente para hacer cálculos simbólicos simples. También recordarán que el concepto de variable se utiliza de varias maneras, para expresar cantidades que se consideran variables - por ejemplo una variable dependiente en función de otras variables independientes - luego cantidades que asumimos constantes, finalmente cantidades que asumen el

significado de parámetros, que son como constantes que nos puede interesar cambiar de vez en cuando.

En cualquier caso, todas estas cantidades están representadas de forma simbólica. En realidad, aquellos que tuvieron tiempo para profundizar en los estudios de álgebra, sabrán que el concepto de variable es pasable de una serie de generalizaciones. No se preocupe, esta no es una clase de matemáticas disfrazada, o tal vez lo es, en cierto modo: después de todo, Logo representa el objetivo de Seymour Papert de hacer que las matemáticas sean más accesibles.

Variables para describir posiciones en el plano

Pero lo que estamos haciendo aquí no requiere actitudes o habilidades especiales. Sólo estamos introduciendo una de las posibles generalizaciones que vamos a necesitar pronto. La generalización que proponemos se refiere al concepto de la posición de la tortuga en la pantalla.

La posición a lo largo de la línea se determina por un simple número - por ejemplo, la posición a lo largo de una carretera: "Trabajos en carretera en el km...". Un caso diferente es el de la posición en una superficie. En una navegación por satélite, un ejemplo conocido, se puede dar la posición en términos geográficos, pero ésta debe ser dada a través de dos valores: latitud y longitud, que denotan el paralelo y el meridiano. Para hundir un barco en un juego de tablero de batalla naval, debes dar dos coordenadas, por ejemplo b7, donde "b representa la columna y 7 la línea". Del mismo modo, para identificar celdas en una hoja de cálculo, etc. Nuestra tortuga también necesita dos valores para identificar un lugar específico en la hoja, que podemos imaginar como la x y la y de la tortuga en el espacio de la página. Por lo tanto, la manera de expresar este concepto en el mundo de las tortugas (¡y no sólo eso!) es la siguiente:

P=[200, 300]
IMPRIMIR P

LibreLogo imprime el "valor"**[200, 300]**. Obviamente elegimos estos números al azar, sólo para dar un ejemplo. El objetivo es mostrar cómo representar de forma simbólica una posición, que en realidad se expresa mediante dos números. En álgebra este tipo de variable se llama "vector".

Vectores

Existe también otra forma de "aislar" los elementos individuales dentro de un vector. Veamos el siguiente ejemplo:

P=[200, 300]
IMPRIMIR PPRINT
P[0]
IMPRIMIR P[1]

Si se ejecuta este fragmento de código, primero se imprime el "valor"**[200, 300]**, luego el valor **200**, luego **300**. Por lo tanto, está claro que con **P[0]** se obtiene el primer elemento del vector de posición, que contiene el número **200**, y con **P[1]** el segundo elemento, que contiene el número **300**. Por lo tanto, esto es lo que se necesita para ir y ver cómo se puede controlar aún más fácilmente la posición de la tortuga en la pantalla.

¿Por qué deberíamos usar sólo estos comandos.... Aprendizaje sintónico

Ahora, realmente, con los comandos que hemos visto hasta ahora, podemos reproducir todos los ejemplos reportados en el libro de Geometría de Tortugas, pero, lo que es aún más importante, son los más adecuados para que los niños trabajen con el Logo. Hagamos un ejemplo simple. Como veremos más adelante, en LibreLogo, como en todos los demás entornos de Logo, hay órdenes primitivas que facilitan la construcción de figuras geométricas. Por ejemplo, con SQUARE 100 se obtiene un cuadrado de 100 puntos laterales. ¿Y qué? ¿Por qué molestarse en"viajar con la Tortuga"?

Bueno, la respuesta es exactamente ésta: viajar con la Tortuga. Sí, porque para los niños la experiencia física es un primer canal de aprendizaje fundamental. Papert usó el término aprendizaje sintónico corporal. Primero tratas de "dibujar" una figura en el suelo moviéndote con tu cuerpo, luego externalizas el proceso proyectándolo a otra entidad, que podría ser un robot aceptando órdenes, o nuestra Tortuga. Si usted "enseña" a los niños que para obtener un cuadrado sólo tienen que escribir CUADRATURA, memorizan algo - obtienen una "competencia"? - pero son una experiencia de aprendizaje bastante pobre. Por otro lado, si los acompañas en un recorrido exploratorio, al principio sólo moviendo la Tortuga, luego descubriendo desviaciones, luego tratando de elegir, o mucho mejor, dejándolos elegir metas, pueden llegar a la plaza pero ésta será "su plaza". La plaza se asociará a la mayor experiencia que han tenido sobre sus componentes y propiedades.

El papel del error

Por supuesto, cuando exploren, cometerán errores. Esto es crucial para poner los errores en la perspectiva correcta. El entorno escolar convencional induce a los niños a tener miedo de cometer errores. Los padres tienen una responsabilidad relevante sobre este problema, especialmente hoy en día, ya que todos vivimos una fuerte presión hacia el mero desempeño. Tomemos nuestra última definición de Tortuga de un cuadrado:

```
REPITE 4[
ADELANTE 100 DERECHA
 90
]
```

y pídele a tu discípulo que transforme este código para obtener un triángulo equilátero. Muy a menudo obtendrá la siguiente respuesta:

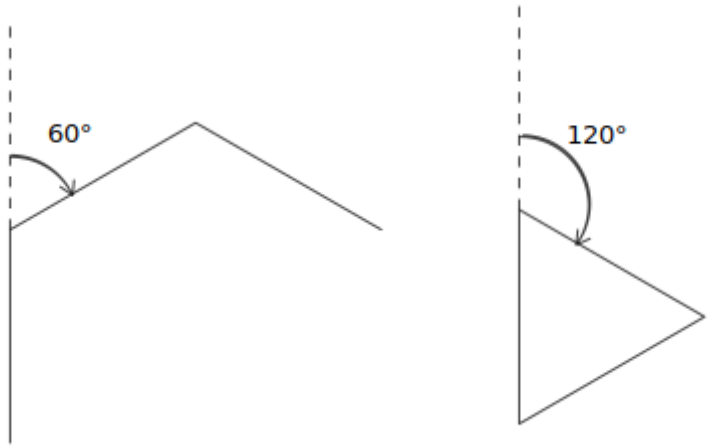
```
REPITE 3[
ADELANTE 100 DERECHA
 60
]
```

Incluso los adultos suelen responder así. La razón es que se centran en el hecho de que los ángulos internos de un triángulo equilátero miden 60° grados, pero el resultado es erróneo.... Este error es tan común que se cita en "Turtle Geometry", Fig. 1.2, p.7.

Solucionar el error

Al estudiar los resultados, los alumnos pronto se dan cuenta de que tienen que centrarse en la desviación en lugar de en el ángulo interno y que los dos ángulos son adyacentes. Por lo tanto, la Tortuga debe girar 180-60 = 120°.

```
REPETIR 3[
ADELANTE 100 DERECHA
 120
]
```



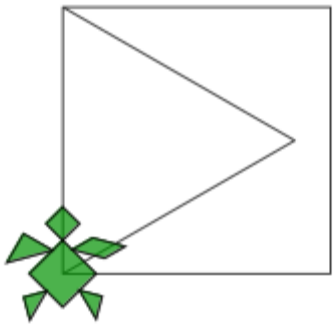
Intentando construir algo: un error de nuevo....

Ahora que tenemos un triángulo y un cuadrado, podemos usarlos como bloques de construcción en dibujos más complejos - una casa, por ejemplo. Exploremos esto....

A TRREPEAT
 3[
ADELANTE 100 DERECHA
 120
]
 FIN

HASTA LA REPETICIÓN CUADRADA
 4[
ADELANTE 100 DERECHA
 90
] (
SQREPEAT
 4)

SQTR



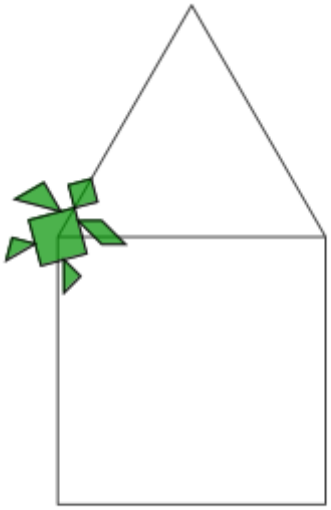
Solucionando el error....

Mirando más cuidadosamente nos damos cuenta de que, una vez que el cuadrado es dibujado con SQ, la Tortuga tiene que moverse a lo largo del lado vertical hacia arriba, desde su posición final y luego, debe girar 30° para poder dibujar el techo en la posición correcta.

A TRREPEAT
3[
ADELANTE 100 DERECHA
120
]
FIN

HASTA LA REPETICIÓN CUADRADA
4[
ADELANTE 100 DERECHA
90
] (
SQREPEAT
4)

HACIA ADELANTE
100 HACIA LA
DERECHA
30TR

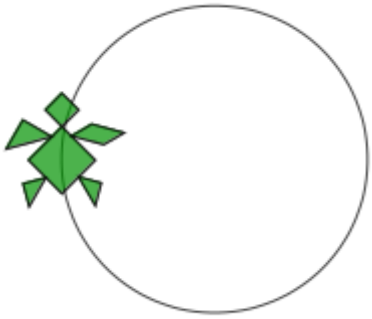


círculo de Papert

Comparando el código del triángulo y el cuadrado, es natural intentar cambiar los números para obtener otros polígonos. En otros lugares desarrollamos esta idea en profundidad, tanto desde el punto de vista matemático como en el contexto de la exploración infantil. Cuando juegan con polígonos a menudo algunos niños pueden hacer un descubrimiento: ¡pero parece un círculo! Aquí está el código:

REPETIR 360 [ADELANTE 1 DERECHA 1]

Esto parece el código de un triángulo, de un cuadrado o de otros polígonos regulares. En realidad es el código del polígono de 360 lados, pero parece un círculo porque los lados son muy pequeños.



Cuerpo construcción sintónica del círculo

Geometría de tortuga versus geometría de coordenadas

Al principio de esta lección comparamos la descripción de las figuras de la tortuga con la de la geometría euclídea y cartesiana. Con este último se describen las formas geométricas especificando las coordenadas de sus puntos. Es idea cartesiana relacionar los números con la geometría.

Pero esta no es la única manera, por ejemplo, los comandos FORWARD y RIGHT de la tortuga dan una forma alternativa de medir las cifras en el avión.

La geometría de las coordenadas se llama geometría de coordenadas; aquí nos referimos a la geometría de Adelante y Derecha como geometría de tortuga. Mientras que el estudio de la geometría de coordenadas conduce a gráficos y ecuaciones algebraicas, la geometría de tortuga introduce a otras ideas matemáticas, y algunas de ellas de una manera muy apropiada para introducir a los niños en las matemáticas.

Intrínseca versus Extrínseca

Una gran diferencia entre la geometría de las tortugas y la geometría de las coordenadas reside en la noción de las propiedades *intrínsecas* de las figuras geométricas. Una propiedad intrínseca es aquella que depende únicamente de la figura en cuestión, no de su relación con un marco de referencia. El hecho de que un rectángulo tenga cuatro ángulos iguales es intrínseco al rectángulo. Pero el hecho de que un rectángulo particular tenga dos lados verticales es *extrínseco*, ya que un marco de referencia externo es necesario para determinar qué dirección es "vertical". Supongamos que tenemos comandos que pueden dejar que la tortuga dibuje enviándola a posiciones específicas, dadas sus coordenadas. En realidad, tal programa existe y se llama POSICIÓN. El código para dibujar un rectángulo sería algo de este tipo (volveremos más tarde sobre algunos comandos nuevos, de todos modos su significado es fácil de adivinar):

POSICIÓN DE PENUP[200.200] POSICIÓN DE PENDOWN [400.200] POSICIÓN DE[400.300] POSICIÓN DE[200.300] POSICIÓN DE[200.200] POSICIÓN DE[200.200].

Así obtenemos un rectángulo con el lado horizontal de 200 y el vertical de 100. El código "natural" en la geometría de las tortugas sería:

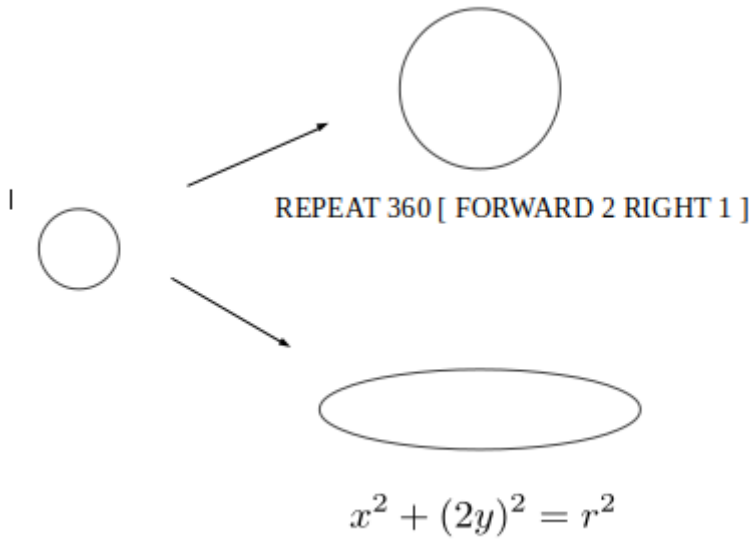
REPETIR 2 [ADELANTE 100 DERECHA 90 ADELANTE 200 DERECHA 90]

Estos códigos dibujan el mismo rectángulo, pero ¿qué pasa si el rectángulo está orientado a lo largo de una dirección oblicua? En el caso del código cartesiano hay que calcular las nuevas coordenadas de los puntos, dado el ángulo de la dirección, mientras que con el código de la tortuga basta con girar la tortuga en la dirección correcta antes de empezar a dibujar el rectángulo.

Intrínseco versus Extrínseco - otro ejemplo

Otra propiedad intrínseca es ilustrada por el programa de tortugas para dibujar un círculo: avanzar un poco hacia adelante, girar un poco a la DERECHA y repetirlo una y otra vez. Contrasta esto con la representación en coordenadas cartesianas de un círculo, $x^2 + y^2 = r^2$. La representación de la tortuga, **REPEAT 360 [FORWARD 1 RIGHT 1]**, hace evidente que la curva es en todas partes la misma, ya que el proceso que la dibuja hace lo mismo una y otra vez.

Esta propiedad del círculo, sin embargo, no es evidente en absoluto a partir de la representación cartesiana. Intente modificar de la misma manera el comportamiento de los dos objetos a lo largo de la dirección y: **REPEAT 360 [FORWARD 2 RIGHT 1]** y$x^2 + (2y)^2 = r^2$... En la versión tortuga se obtiene un círculo de nuevo, pero con diámetro doble, en la versión cartesiana se obtiene una elipse. Así, la representación de la tortuga de un círculo es más intrínseca con respecto al concepto de círculo que la cartesiana.



Local versus global

La representación de la tortuga de un círculo no sólo es más intrínseca que la descripción de las coordenadas cartesianas, sino que también es más *local*, es decir, trata de la geometría de una pequeña pieza a la vez. La tortuga puede olvidarse del resto del plano cuando dibuja un círculo y sólo se ocupa de la pequeña parte del plano que rodea su posición actual.

Por el contrario, $x^2 + y^2 = r^2$ se basa en un sistema de coordenadas *global a* gran escala para definir sus propiedades. Una definición de un círculo como el conjunto de puntos equidistantes de un punto fijo es tan global como la de usar $x^2 + y^2 = r^2$... La representación de la tortuga no necesita hacer referencia a ese punto especial "lejano", el centro. Esta propiedad de las representaciones de tortugas es fundamental para las ideas básicas de cálculo diferencial que se enseñan a nivel universitario.

Procedimientos versus ecuaciones

Una diferencia final importante entre la geometría de la tortuga y la geometría de las coordenadas es que la geometría de la tortuga describe característicamente los objetos geométricos en términos de procedimientos en lugar de ecuaciones. En la formulación de descripciones geométricas de tortugas tenemos acceso a toda una gama de mecanismos de procedimiento (como las repeticiones) que son difíciles de captar en el formalismo algebraico tradicional. Además, es muy fácil modificar una descripción de procedimiento en términos de tortugas. Esto hace que la geometría de las tortugas sea un campo fructífero para la exploración matemática. Esta es una característica extremadamente atractiva para un sistema que se va a utilizar en un contexto educativo.