

Introducción a Python, Pilas y Videojuegos (I)

¿Qué es un lenguaje de Programación?

Las tareas que realizas con los ordenadores, ya sea escribir, diseñar, navegar por Internet, comunicarte a través de las redes sociales, escuchar música o jugar (por poner algunos ejemplos) requieren del uso de **aplicaciones** o **programas**. De hecho, vivimos rodeados de dispositivos electrónicos que funcionan ejecutando diferentes programas que marcan su funcionalidad; desde tareas simples como la limpieza de ropa en una lavadora hasta otras mucho más complicadas como la gestión del tráfico de ferrocarriles de una gran ciudad.

En un mundo como éste, la pregunta de rigor es ¿por qué limitarnos a las tareas que otros han pensado? Somos consumidores, ¿por qué no, también, **creadores**?



Pero, ¿qué necesitamos para conseguirlo? Hacer programas de ordenador tiene muchos puntos de conexión con la buena cocina; escribir un libro de recetas requiere de mimo, ideas claras e instrucciones paso a paso que permitan a los lectores cocinar tus platos. De la misma manera, programar un videojuego (o cualquier otro tipo de aplicación) requiere de **cariño**, una **planificación** adecuada y la escritura de las **instrucciones paso a paso** que el ordenador del jugador realizará cuando éste arranque el juego.



No, sería estupendo, pero no. Para comunicarnos con los ordenadores y darles instrucciones detalladas y muy especializadas (piensa que en un juego vas a querer indicarle algo del tipo “cuando el jugador haga click en esa chimenea haz que salga la bruja desde dentro y en dos décimas de segundo haz que aparezca una escoba, la bruja la coja y salga volando hacia arriba a la izquierda emitiendo una carcajada terrorífica”) no basta con señalar y hacer click con el ratón, o seleccionar de un menú las diferentes opciones disponibles.

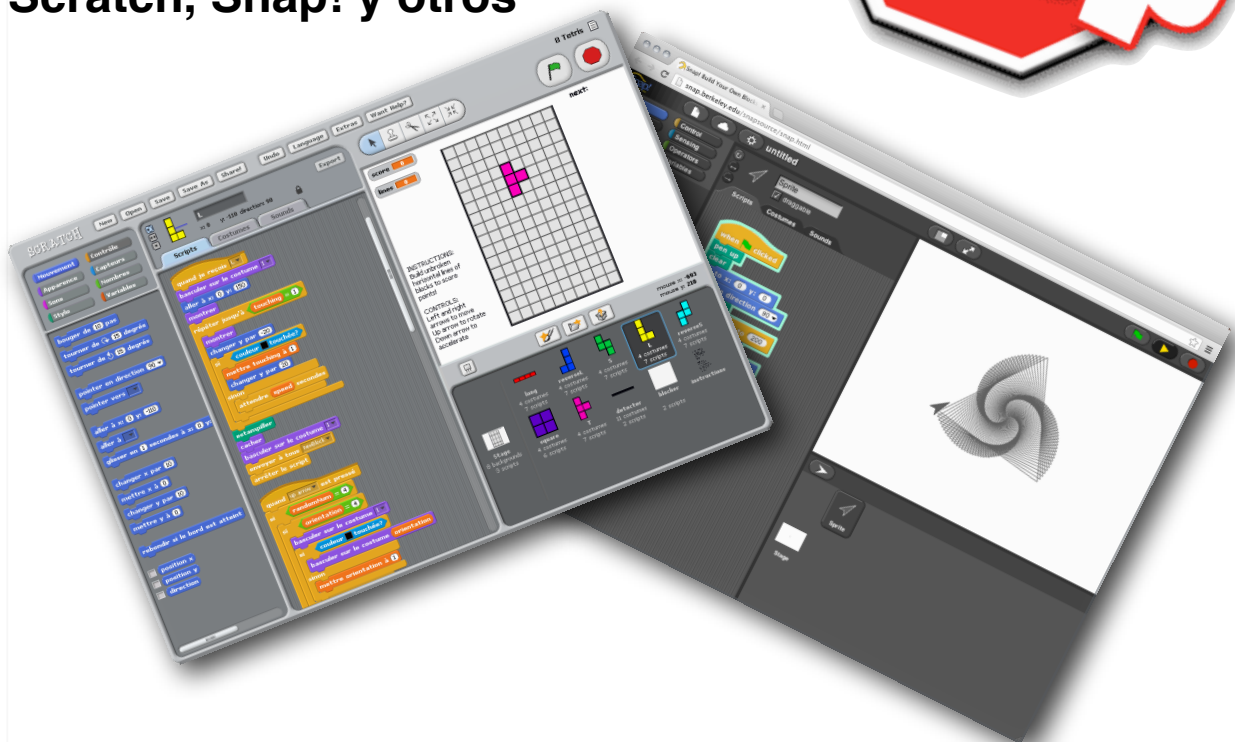


... Sniff... A mí también me entienden...

¡Un momento!

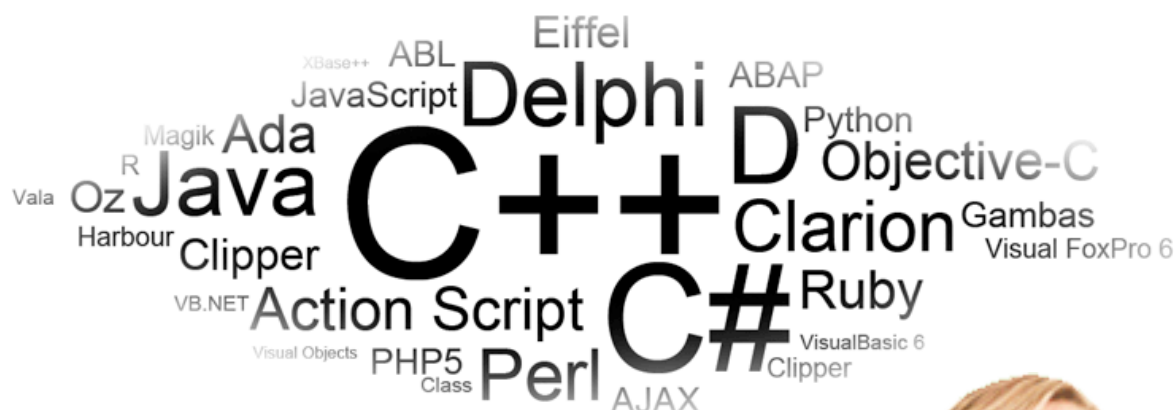


Scratch, Snap! y otros



En realidad sí que hay sistemas para programar tareas más específicas usando botones y arrastrando elementos sobre una pantalla indicando de esta manera qué queremos que aparezca y qué comportamiento queremos que tenga, ¡y son fantásticos! ¡Y podemos hacer videjuegos con ellos de manera extremadamente sencilla! Puede que conozcas alguno de ellos: **Scratch** es el ejemplo más famoso... Pero también es cierto que poseen ciertas limitaciones y son algo menos flexibles.

Como decimos, para comunicarnos con un ordenador necesitamos de un lenguaje común, igual que los humanos nos comunicamos en castellano, inglés o japonés. Y lenguajes de programación hay muchos y muy diferentes entre sí, aunque la funcionalidad final sea muy parecida.



Y entonces, ¿en qué se diferencian?

En la facilidad de uso, en la potencia, en la flexibilidad... y en los gustos personales, ¡claro!



Los lenguajes más amigables, intentan parecerse, en la medida de lo posible, al lenguaje humano. De esta manera, es más fácil aprenderlo y usarlo para que el ordenador haga las tareas que tú quieras.

En inglés, eso sí

¡Lo sabía!



Nuestra elección ha sido el lenguaje **Python**...



No, tú no



Tú tampoco



... nombre inglés de un tipo de serpiente y, al mismo tiempo, un guiño al famoso grupo humorístico británico, Monty Python, por parte de su creador, el holandés **Guido van Rossum**.



*Benevolente
Dictador
Vitalicio*





Una de las virtudes de **Python** es que viene preinstalado en dos de los sistemas operativos más importantes, **OS X** y **GNU/Linux** y que es muy fácil de instalar en el caso de que no lo tengas o quieras actualizarlo a una versión posterior: Sólo tienes que dirigirte a

www.python.org

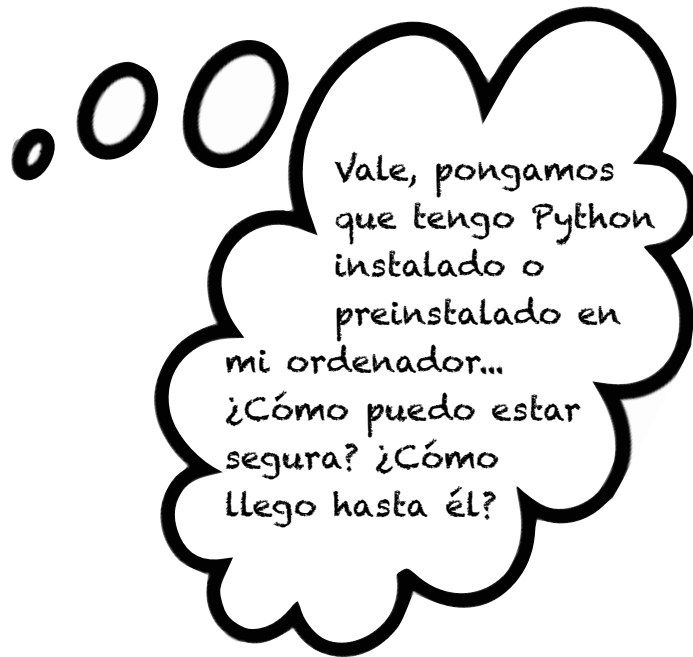
The screenshot shows the Python.org website's 'Download Python' page. The page is titled 'Download Python' and lists the current production versions as Python 2.7.5 and Python 3.3.2. It provides instructions on how to download Python for different operating systems (Windows, Mac OS X, Linux, Unix) and offers links to download Python 2.7.x or Python 3.3.x. The page also includes a section for 'Alternative Implementations' and a 'Release Schedule' section.

Si tienes Windows, por ejemplo, aquí está el instalador

Es mejor que instales, para nuestros propósitos, la versión 2.7

También tienes métodos para usar **Python** directamente en dispositivos móviles, pero vamos a centrarnos, de momento, en el desarrollo de videojuegos desde un ordenador clásico.





Según cuál sea tu sistema operativo, podrás acceder a él desde una aplicación, el menú de inicio o bien tendrás que abrir una ventana de terminal y escribir el comando **python**. En cualquier caso, deberías llegar a algo parecido a esto:

```
jdestudios@ubuntu-jdestudios: ~  
jdestudios@ubuntu-jdestudios:~$ python  
Python 2.7.4 (default, Apr 19 2013, 18:32:33)  
[GCC 4.7.3] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>>
```

El comando que has escrito

Tu versión de Python

Aquí puedes empezar a escribir en tu nuevo lenguaje :-)

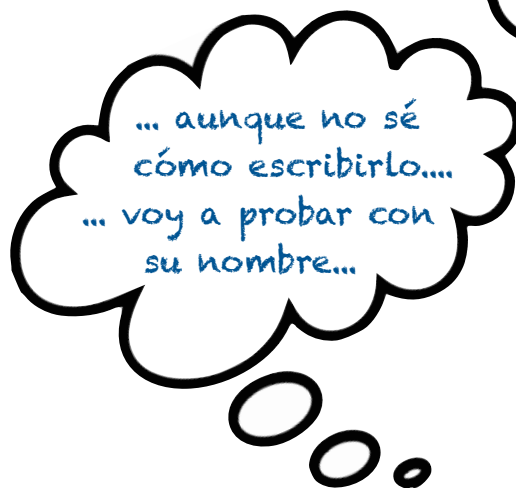
Esos tres símbolos “*mayor que*” consecutivos que puedes apreciar en la ventana, **>>>**, son lo que se conoce como el **prompt** de Python; nuestro nuevo amigo está esperando a que escribas algo allí y pulses la tecla *enter*, para empezar a procesarlo...


```
Type "help", "copyright"
>>> 
```

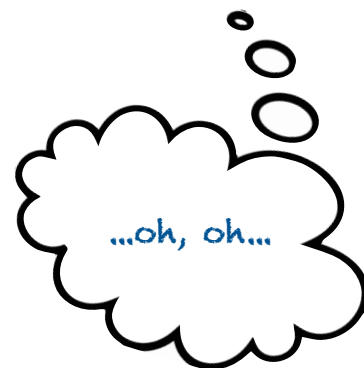
```
Type "help", "copyright"
>>> 3 + 5
```

```
Type "help", "copyright"
>>> 3 + 5
8
>>>
```

Genial, suma estupendo...



```
>>> 3 + 5
8
>>> 2 x pi
File "<stdin>", line 1
  2 x pi
    ^
SyntaxError: invalid syntax
>>> 
```



Errores en Python

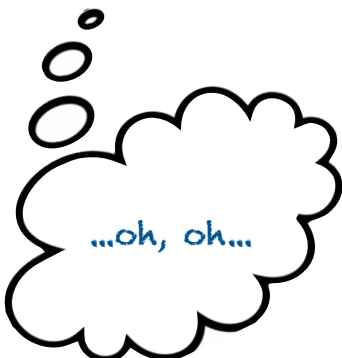


Cuando **Python** no entiende lo que le dices, devuelve, naturalmente un error. Y lo hace a su modo, tratando de explicarte qué tipo de error es y dónde se ha producido (aunque a veces no lo acierte del todo; no le critiques, a fin de cuentas estamos hablando de errores). En el ejemplo anterior observarás que, entre otras cosillas, **Python** se queja diciéndote

SyntaxError: invalid syntax

es decir, que ha encontrado un error de sintaxis (lo que quiere decir que lo que has escrito no está bien dicho) y por eso no te entiende. Aún hay más; intenta indicarte dónde está el error, con una especie de flecha: ¿ves que te señala el signo de multiplicación?

Un secreto: En **Python**, la multiplicación se escribe con el símbolo del asterisco *****, no con el signo del aspa o la x.



```
>>> 2 x pi
      File "<stdin>", line 1
        2 x pi
          ^
SyntaxError: invalid syntax
>>> 2 * pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pi' is not defined
>>>
```

¡Un nuevo error! ¡Tranquilo! Parece que **Python** te está diciendo que ese nombre que has escrito, **pi**, no lo entiende (él te dice, literalmente, '*no está definido*'). Claro, si **Python** pudiera entender cualquier cosa que le pongas, podrías pedirle algo así como *¿me haces la cena de esta noche?* y a ver cómo se come eso...

Todo idioma, todo lenguaje, tiene sus palabras y para comunicarte con él tienes que usarlas. **Python** no es distinto es esto. Poco a poco las irás aprendiendo, con el uso; de momento, acabas de ver que, para **Python**, **pi** no es tu π ni vale 3.14... Hay maneras de decírselo, por supuesto, pero hay un viejo dicho en este mundillo y es que **Python viene con baterías incluidas**, y esa es una ventaja muy interesante de este lenguaje y que le da una gran flexibilidad. ¿Qué queremos decir con esto? En el próximo cuadro tienes la respuesta...

Módulos (o librerías) de Python



Con lo que nos referimos con **baterías incluidas**, es que Python tiene una serie de extras que podemos usar cuando nos interese. Estos extras vienen en forma de **módulos** (también llamados **librerías**), cada uno con su propio nombre. Para cargarlos, hemos de escribir

```
>>> import nombre_del_módulo
```

y, a partir de entonces, para usar cualquier cosa que haya en él, hemos de utilizar el nombre del módulo, un punto y el nombre de lo que queremos.

Un ejemplo; el valor de π está definido dentro del módulo **math**, con el nombre (lo adivinaste) **pi**. Así que, tras importar el módulo **math**, podremos escribir **math.pi**

```
>>> 2 * pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pi' is not defined
>>> import math
>>> 2 * math.pi
6.283185307179586
>>> █
```

¡Bravo!



¡Otro módulo al rescate!



pilas engine



Pilas es un módulo de **Python** que implementa un motor para hacer videojuegos. ¡Es fantástico! En castellano y con mucha documentación, puedes obtenerlo de su página web:

pilas-engine.com.ar

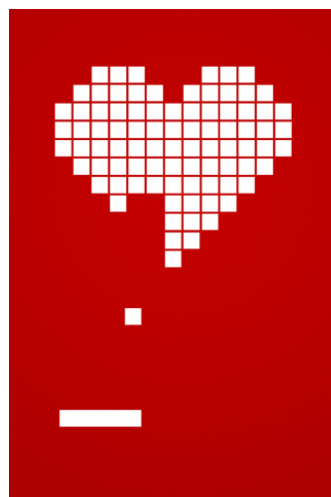
Como no viene incluido de serie en **Python**, necesitarás instalarlo en tu ordenador. Busca el archivo adecuado para tu sistema operativo en el apartado de descargas de la web.



Hugo Ruscitti,
el creador de Pilas



¡ Gracias ,

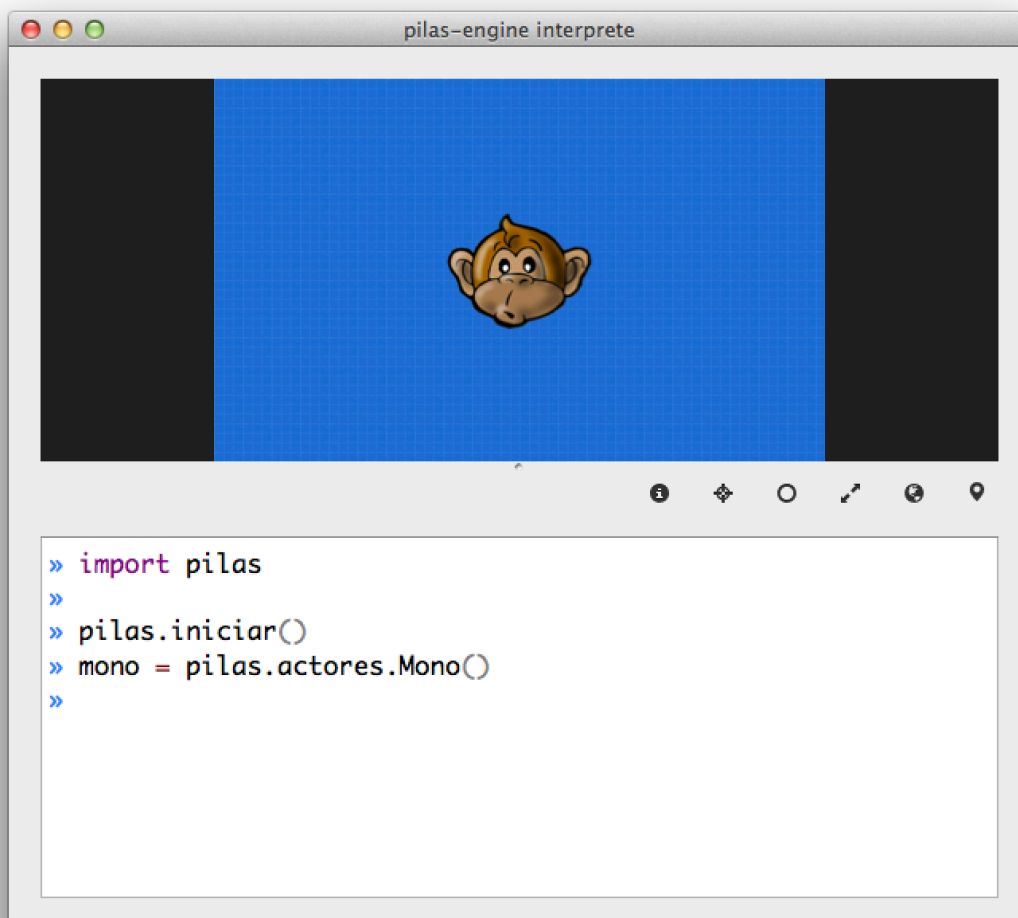


Hugo !

De la misma forma que con **Python**, tras instalar **Pilas**, podrás ejecutarlo desde una aplicación, el menú de inicio o desde una ventana de la terminal escribiendo el comando **pilas** (según sea tu sistema operativo). Al hacerlo, esta vez, se te abrirá una ventana (*la ventana de Pilas engine*) y verás algo similar a lo siguiente:



Y si pulsas el botón “*abrir intérprete*”, ¡voilà!





De paso, observarás que **Pilas** es muy amable y su *intérprete* (en lenguajes de programación se utiliza este término ya que el ordenador lo que hace es *interpretar* lo que le dices), ya ha añadido unas cuantas líneas de código por ti. Veámoslas, una a una:

```

» import pilas ← 1
» pilas.iniciar() ← 2
» mono = pilas.actores.Mono() ← 3

```

1 Recuerda, como hemos dicho, que **Pilas** es un módulo de **Python**. Por lo tanto, para poder usarlo, primero hay que importarlo. ¡Observa que se pone en minúsculas!

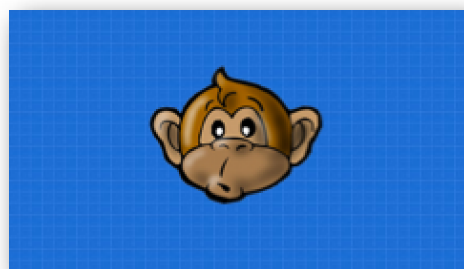
2 Pero **Pilas** es mucho más; nada más y nada menos que un *motor* (en inglés, *engine*) de juegos. Para ponerlo a punto necesitas inicializarlo y para hacer esto debes usar una **función** de **Pilas** que se llama **iniciar()**.

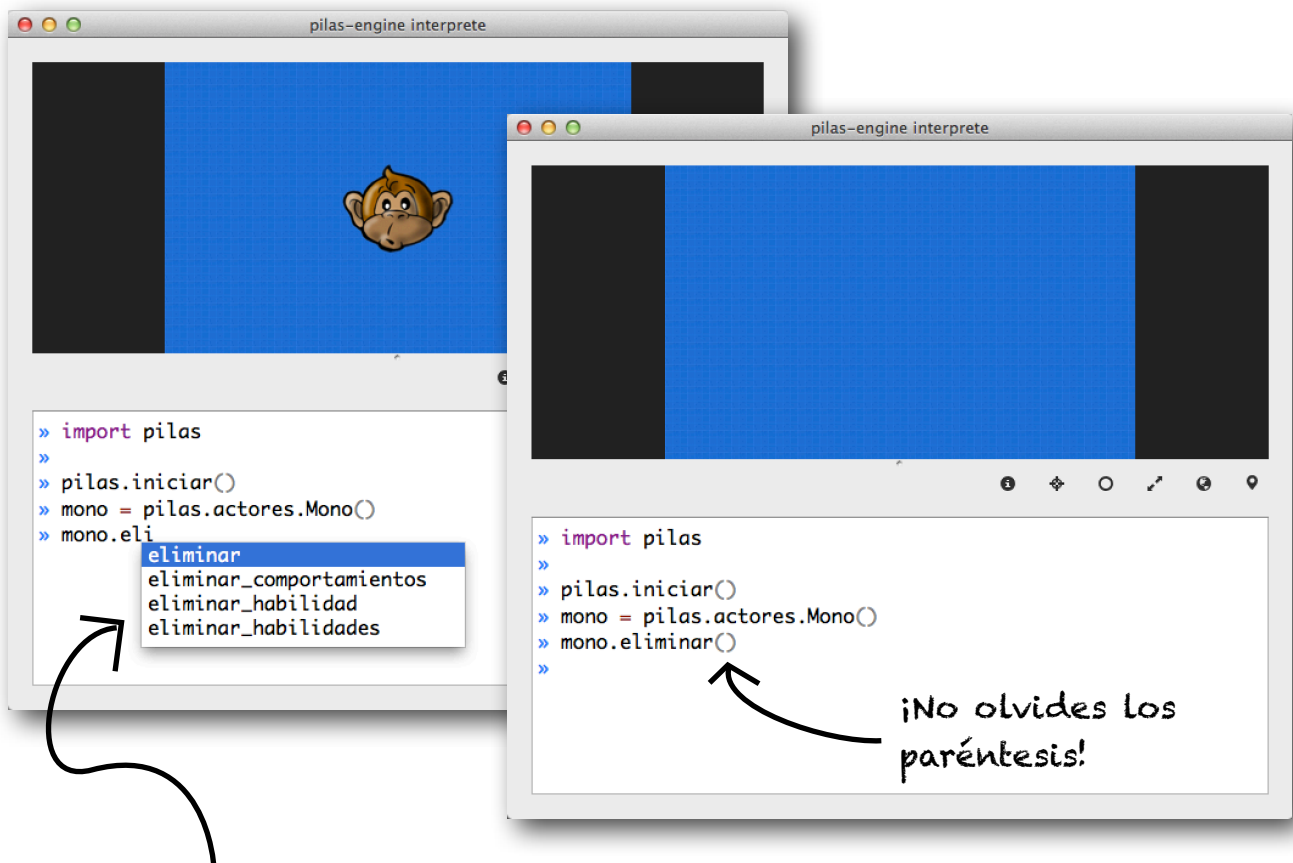
3 Y **Pilas** trae consigo una serie de **objetos** que puedes utilizar en tus programas, algunos de ellos son personajes o **actores** y un ejemplo es el **Mono**. Para crear uno puedes escribir **pilas.actores.Mono()** y ¡allí está, en medio de la ventana!

Fíjate que, en esta última línea, usando el signo = se le asigna un nombre al personaje (en este caso, **mono**). Esto permite que puedas referirte a él más adelante (es lo que se llama, en Python, una **variable**).

¿Quieres probarlo? Escribe el comando

mono.eliminar()





A medida que lo escribes, Pilas te ayuda intentado completar lo que vas poniendo. Si lo prefieres, selecciona la opción que deseas y pulsa "intro".

¿Has visto cómo ha desaparecido? ¡Estupendo! Y sí, lo has adivinado, la forma de decirle a un **actor** que haga algo, es igual que la que usamos con los módulos; su nombre, seguido de un punto y de lo que queremos hacer (es lo que se llama **notación dot**, de "dot", punto, en inglés). Igual que con los módulos podemos usar así lo que contienen, los actores (los **objetos** en general) podrán hacer ciertas cosas que tengan definidas...



Andar para después correr

Antes de seguir adelante, **necesitas con urgencia** familiarizarte más con el **lenguaje Python**. Vas a ver una serie de conceptos que quizá de primeras te resulten algo pesados pero que luego agradecerás para llevar a cabo la estupenda tarea de crear tu propio **videojuego**. ¿No te parece que ya han aparecido muchos términos nuevos? ¿Se te empieza a escapar algo de las manos?

¡No te preocupes, en la **segunda parte** de este tutorial lograrás centrarte un poco!

En él, aprovechando el **intérprete de Pilas**, aprenderás los diferentes tipos de elementos que puede manejar **Python** por defecto y cómo manipularlos, entenderás por qué todo en **Python** son objetos, qué son las funciones y muchas otras cosas...

