Introducción a los lenguajes de Programación - Python

Juan Antonio Cos García

Lenguaje de programación diseñado por el holandés **Guido van Rossum**.

La filosofía del lenguaje hace hincapié en mantener una sintaxis muy limpia que favorezca la legibilidad.

Administrado por la Python Software Foundation (PSF).

Python se distribuye con su propia licencia libre.

Python Software Foundation Licence (PSFL).

Compatible con la GPL.

Características:

- Lenguaje de programación multiparadigma:
  - Programación orientada a objetos.
  - Programación imperativa.
  - Programación funcional.
- Lenguaje interpretado.
- Tipado dinámicamente.
- Fuertemente tipado.
- Facilidad de extensión: es muy fácil escribir nuevos módulos en C/C++.

Características:

Python = lenguaje + biblioteca estándar + módulos

Lenguaje: sintaxis, semántica, palabras reservadas, funciones genéricas.

Biblioteca estándar: conjunto de módulos por defecto. Ejemplos: os, sys, re, pickle...

Módulos de extensión: librerías extras para interfaces gráficas, acceso a datos, persistencia, XML, desarrollo web, ...

Es multiplataforma.

Es Software Libre.

Lenguaje interpretado

Un lenguaje interpretado o de script es aquel que se ejecuta a través de un programa intermedio denominado **intérprete**.

...en vez de compilar el programa a código máquina y ejecutarlo directamente.

Los lenguajes compilados generan programas más rápidos.
Los lenguajes interpretados son más flexibles y más portables.

¡¡ Pero no olvidemos que Python también se compila a bytecodes !!

Tipado dinámico

En Python NO es necesario declarar el tipo de datos de una determinada variable.

El tipo se determina en tiempo de ejecución según el valor que se le asigne, además, este tipo puede variar durante la ejecución.

Fuertemente tipado

No se permite tratar a una variable como si fuera de un tipo distinto al que tiene.

Es necesario convertir de forma explícita dicha variable al nuevo tipo previamente.

Filosofía

Código "pythonico".

> Es el código que sigue los principios de legibilidad y transparencia.

Estos principios están descritos en "El Zen de Python" y fueron escritos por el desarrollador Tim Peters.

Los principios son…

(http://www.python.org/dev/peps/pep-0020/)

1.- Bello es mejor que feo.

2.- Explícito es mejor que implícito.

3.- Simple es mejor que complejo.

…….

¿Por qué Python?

- Fácil de aprender e implementar.
- Gran apoyo por parte de la Comunidad.

¡¡ Es software libre !!

- Gran apoyo por parte de la Industria. Empresas como:

  Google, YouTube, Eve online, Yahoo, Facebook, NASA, Pixar, Red Hat, Nokia, Intel, Cisco, HP, Netflix, etc.

- Multiplataforma (Linux, Windows, Mac OS, …).
- Multiparadigma (imperativo, funcional, OO).
- Facilita la depuración
- Muchísima documentación.
- Lenguaje muy potente.

  No muy adecuado para aplicaciones de bajo nivel o con requisitos de rendimiento críticos.

¿Dónde se utiliza Python?

Generalmente, Python es una excelente opción para:

- Desarrollo Web (por ejemplo, los frameworks Django y Pyramid, micro-frameworks Flask y Bottle)
- Computación científica y numérica (por ejemplo, SciPy, una colección de paquetes con fines matemáticos, científicos y de ingeniería; Ipython, un shell interactivo que permite la edición y grabación de sesiones de trabajo)
- Educación
- GUIs de Escritorio (por ejemplo, wxWidgets, Kivy, Qt)
- Desarrollo de software (control de compilación, gestión y pruebas: Scons, Buildbot, Apache Gump, Roundup, Trac)
- Aplicaciones empresariales (ERP y sistemas de comercio electrónico: Odoo, Tryton)
- .....etc

Herramientas básicas

Actualmente existen varias implementaciones de Python:

- **CPython** (o Python sin más)
  - Implementación original.
  - Al hablar de Python, se habla de ESTA implementación.
  - La más utilizada, la más madura y la más rápida.
  - Disponible para varias plataformas.
  - Escrita en C (tanto el intérprete como los módulos)
- **Jython**.
  - Implementación en Java de Python.
  - Desde Jython se pueden utilizar todas las librerías de Java.

Herramientas básicas

IronPython.

Implementación en .NET de Python.

Desde IronPython se pueden utilizar todas las librerías de .NET.

Stackless Python. Variante de Python centrada en la programación basada en hilos.

Pippy. Implementación realizada para Palm.

PyPy. Implementación de Python hecha en Python.

ActivePython es una implementación privativa de Python con extensiones, para servidores en producción y aplicaciones de misión crítica desarrollado por ActiveState Software.

Herramientas básicas

CPython está instalado por defecto en casi todas las distribuciones de GNU/Linux.

Podemos comprobar si está instalado ejecutando desde un terminal con los comandos 'python' y 'python3':

```
alumno@ingeniero:~$ python3
Python 3.8.10 (default, May 3 2021, 11:48:03)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> credits
    Thanks to CWI, CNRI, BeOpen.com, Zope Corporation and a cast of thousands
    for supporting Python development.  See www.python.org for more information.
>>> exit()
alumno@ingeniero:~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
alumno@ingeniero:~$
```

Versiones actuales de Python
(http://www.python.org/getit/)

Dos versiones incompatibles:

- **Python 2.7.9**

- **Python 3.10**

Modos de ejecución de un programa en Python

Existen dos formas de ejecutar código Python:

1.- Escribimos las líneas directamente en un intérprete y obtenemos la respuesta línea a línea; ó

2.- Escribimos todo el código del programa en un archivo de texto y lo ejecutamos.

Modos de ejecución de un programa en Python

Directamente en el intérprete:

```
alumno@ingeniero:~$ python3
Python 3.10.0 (default, Apr 11 2022, 13:05:11)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> saludo = "A programar se aprende programando"
>>> print(saludo)
A programar se aprende programando
>>> exit()
alumno@ingeniero :~$
```

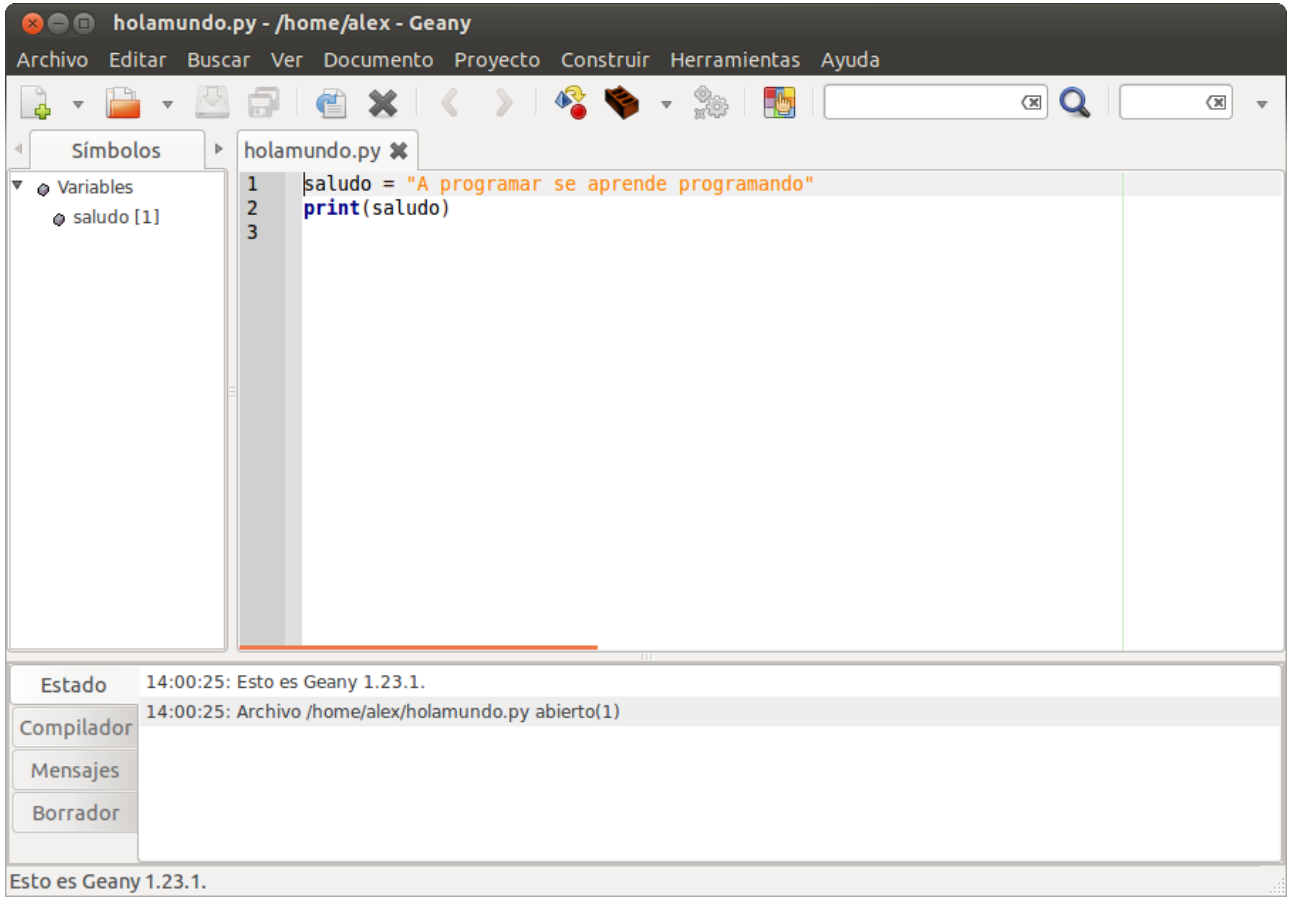
Modos de ejecución de un programa en Python

Creamos un fichero de texto con el contenido y lo ejecutamos:

```
jacos@ingeniero:~$ cat holamundo.py
saludo = "A programar se aprende programando"
print(saludo)
jacos@ingeniero:~$ python3 holamundo.py
A programar se aprende programando
jacos@ingeniero:~$
```

Modos de ejecución de un programa en Python

Como la anterior, pero con un IDE.



Elementos principales de un Programa

Los elementos principales de un programa informático clásico se dividen en varios grupos dependiendo del paradigma, del lenguaje y de la finalidad. A pesar de la variedad de elementos de un programa, prácticamente todos poseen los siguientes bloques:

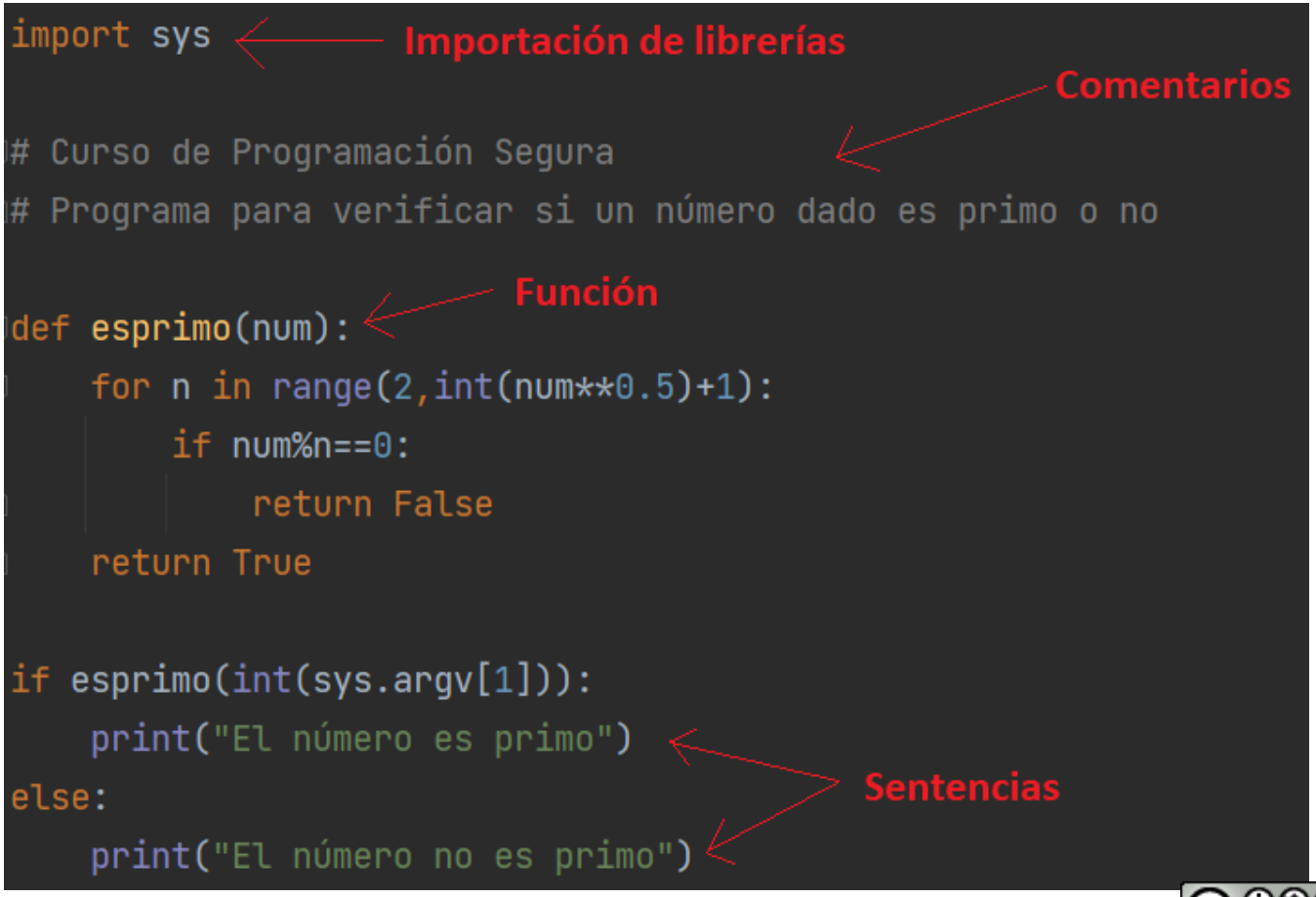
- **Bloque de declaraciones.** Incluye la declaración y normalmente la instanciación de todos los objetos y elementos a procesar como constantes o variables.

- **Bloque de instrucciones**
  - **Entrada:** Su función es obtener la información aportada desde el exterior necesaria para realizar el procesamiento.
  - **Proceso:** Instrucciones cuya finalidad es alcanzar el objetivo del programa a partir de la información proporcionada en la entrada.
  - **Salida:** Una vez realizado los cálculos este bloque se encarga de almacenar, por ejemplo en un fichero o en una base de datos o mostrarlo en algún dispositivo de salida como un monitor o impresora.

Elementos de un Programa

Dentro de nuestro programa o código fuente podemos distinguir varios componentes:

- Comandos de pre-procesamiento: Importación de módulos/librerías.
- Funciones.
- Declaración de variables.
- Sentencias & Expresiones
- Comentarios



Introducción al lenguaje de programación Python

Sentencias.
Mostrando cadenas de texto.
Comentarios.
Variables.
Entrada de datos.
Identificadores.
Palabras reservadas.
Literales: números y cadenas.
Una cuestión de estilo.
    Identación.

Juan Antonio Cos García

Nuestro primer programa en Python

En Linux se puede conseguir que el sistema operativo abra el fichero con el intérprete adecuado añadiendo una nueva línea:

A esta línea se la conoce como shebang, hashbang ó sharpbang:

```
#!/usr/bin/python3

print("A programar se aprende programando.")
```

Para ejecutarlo ahora basta con:

Añadir el permiso de ejecución.

Ejecutarlo con ./holamundo.py

Nuestro “Hola Mundo” completo

```
#!/usr/bin/env python3

def main():
	print("A programar se aprende programando.")
	return 0

if __name__ == '__main__':
	main()
```

Sentencias

Una sentencia es un comando a ser ejecutado cuando lanzamos nuestro programa.

En Python, las sentencias NO TERMINAN con el carácter **;**

En nuestro ejemplo básico solo había una sentencia:

```
print("A programar se aprende programando.")
```

La ejecución de sentencias se realiza de forma secuencial.

Juan Antonio Cos García

Mostrando cadenas de texto

print es una función utilizada para mostrar cadenas de caracteres.

Una cadena de caracteres (string) es un serie de caracteres cerrados entre comillas dobles:

"Esto es un ejemplo de cadena."

"" (cadena vacía)

"Otro ejemplo de cadena.\n"

print imprime retornos de carro a no ser que se lo indiquemos explícitamente:

```
print("Ser o no ser: esa es la cuestión.")
print("Ser o no ser: ", end="")
print("esa es la cuestión.")
```

Comentarios

Hasta ahora nuestro programa carece de algo importante: documentación.

- Es importante indicar en todos nuestros programas, al menos:
  - Nombre del programa.
  - Fecha en la que fue escrito.
  - Autor.
  - Qué hace el programa.
- Podemos usar los comentarios para incluir cualquier información adicional.
- Los comentarios son ignorados por el intérprete.

Comentarios

Los comentarios empiezan con el carácter #

Ejemplo:

```
#!/usr/bin/env python3
#
#       holamundo.py
#
#       Copyright 2022 IES Ingeniero Cierva <ingeniero@midominio.local>
#
#       Programa que muestra un saludo en pantalla
#       Octubre 2022

def main():
    print("A programar se aprende programando.")
    return 0

if __name__ == '__main__':
    main()
```

Comentarios

También podemos insertar comentarios en nuestro código:

```
print("Hola")          # Mostramos un saludo
```

Los comentarios ayudan a mantener el código.

- Debe ser una ayuda para todo aquel que vaya a leer nuestro código, incluido nosotros mismos.
- Solo se ponen comentarios cuando sea necesario.

El carácter # hace que el intérprete ignore todo el resto de la línea.

Nuestro primer programa

```
#!/usr/bin/env python3
#
#       holamundo.py
#
#       Copyright 2022 IES Ingeniero <ies@midominio.local>
#
#       Programa que muestra un saludo en pantalla
#       Enero 2022

def main():
    print("A programar se aprende programando.")
    return 0

if __name__ == '__main__':
    main()
```

Interactuando con el intérprete

Las sentencias anteriores también las podemos utilizar directamente con el intérprete:

```
alumno@ingenier:~$ python3
Python 3.10.0 (default, Apr 11 2022, 13:05:11)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("A programar se aprende programando")
A programar se aprende programando
>>>
```

Juan Antonio Cos García

Interactuando con el intérprete

También podemos hacer operaciones aritméticas:

+ - * / % **

```
>>> print(5 + 7)
12
>>> print(2 * 3)
6
>>> print(20 / 2)
10.0
>>> print("Hola " * 10)
Hola Hola Hola Hola Hola Hola Hola Hola Hola Hola
>>> print("Superman " + "Batman")
Superman Batman
>>>
```

Variables y asignación

Hasta ahora nuestros programas han sido muy simples...
... pero muchos programas necesitan hacer cálculos antes de mostrar información por pantalla,

.. por lo que se necesita un lugar temporal donde almacenar dichos resultados intermedios.

En Python, y en otros muchos lenguajes, esos lugares de almacenamiento temporal se denominan **variables**.

Variables y asignación: tipos

Cada variable debe tener un **tipo** (type).

- El tipo de una variable determina qué clase de datos puede almacenar.

La elección del tipo es crítica ya que determina:

- Cómo se almacena la variable.
- Qué tipo de operaciones podemos realizar.

El tipo de una variable numérica determina además:

- El número más grande y el número más pequeño que podemos almacenar.
- Si podemos almacenar números decimales (reales).

Variables y asignación: tipos

Por ahora vamos a limitar el número de tipos que vamos a manejar a:

Números enteros (**int**, integer, entero).

Puede almacenar números enteros, tales como 0, 2345 ó -2.

El rango de posibles valores está limitado. En arquitecturas i386 de 32 bits, los límites son:

Desde el número -2.147.483.648 hasta el número +2.147.483.647.

Números reales (**float**, floating point, coma flotante).

Puede almacenar números decimales (reales) como el -234'8287, así como números enteros (como el 1.0). También es capaz de almacenar números mucho más grandes que el tipo int (como el 2).

Sin embargo la aritmética en coma flotante es mucho más lenta que la aritmética entera. Además se pueden producir errores de redondeo.

Variables y asignación: declaraciones

Python es un lenguaje tipado dinámicamente, las variables NO SE DECLARAN antes de poder ser utilizadas.
Basta con utilizar la variable directamente.
Podemos consultar el valor de una variable a través de la función *type*.

```
>>> a = 62
>>> type(a)
<class 'int'>
>>> f = 18.3
>>> type(f)
<class 'float'>
>>>
```

Variables y asignación: asignación

Una variable puede tomar un valor a través de una asignación.

Las sentencias...

```
base = 18
altura = 12
```

... asignan valores a las variables base y altura. Los números 8 y 12 se denominan constantes.

En Python, al contrario que en otros lenguajes, el siguiente código sería legal:

```
a = 23          # Ahora a es de tipo <int>.
a = 8.52        # Ahora a es de tipo <float>.
```

Variables y asignación: asignación

Una vez que hemos asignado valores a las variables, podemos utilizarlas para calcular otros valores:

```
largo = 34
ancho = 23
alto = 9
volumen = largo * ancho * alto    # Volumen vale ahora 7038
```

En Python, '*' representa la multiplicación.

La sentencia multiplica las variables *largo*, *ancho* y *alto*, almacenando el resultado en *volumen*.

La parte derecha de una asignación puede ser cualquier expresión que contenga constantes, variables y operadores.

Variables y asignación: mostrando valores

Para mostrar el valor de una variable se utiliza también la función **print**.

Para escribir el mensaje:

Volumen: *v*

... utilizaríamos la siguiente llamada a print:

print("Volumen:", volumen)

print muestra un retorno de carro al terminar.

Variables y asignación: mostrando valores

... pero print también es capaz de mostrar el resultado de cualquier expresión numérica:

Para escribir el mensaje:

Volumen: *v*

... podemos utilizar también la llamada:

print("Volumen:", largo * ancho * alto)

Ejemplo completo: volumen de un cubo

```
#!/usr/bin/env python3

largo = 34
ancho = 23
alto = 9
volumen = largo * ancho * alto

print("Dimensiones", largo, "x", ancho, "x", alto)
print("El volumen del cubo es:", volumen)
```

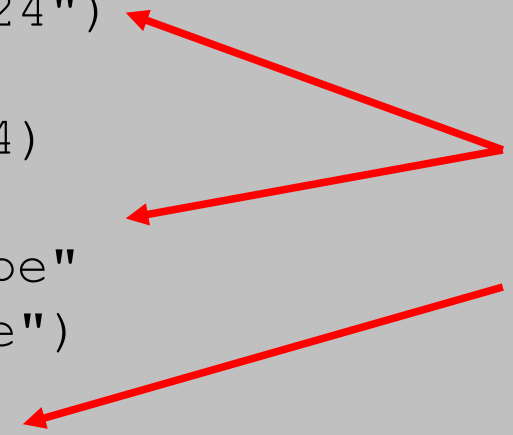
Juan Antonio Cos García

Ejemplo: longitud circunferencia

```
#!/usr/bin/env python3

import math

radio = 18
longitud = 2 * math.pi * radio
superficie = math.pi * radio * radio

print("Radio:     ", radio)
print("Longitud:  ", longitud)
print("Superficie:", superficie)
```

No nos confundamos

```
>>> print("23 + 24")
23 + 24
>>> print(23 + 24)
47
>>> nombre = "Pepe"
>>> print("nombre")
nombre
>>> print(nombre)
Pepe
>>>
```

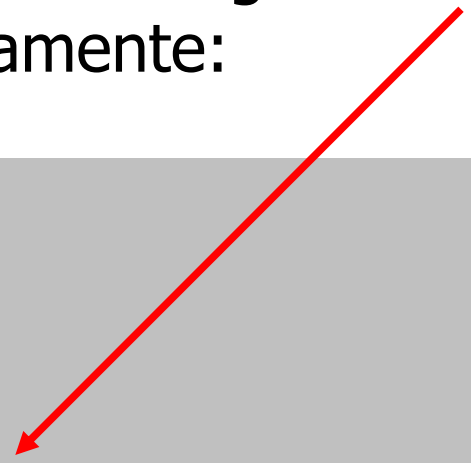


¡¡¡ Hay que pensar como programadores !!!

Variables y asignación: inicialización

En Python no podemos utilizar ninguna variable que no hayamos inicializado previamente:

```
alumno@ingeniero:~$ python3
>>> a = 128
>>> print(a)
128
>>> print(b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
>>>
```



Entrada de datos

El programa que calculaba el volumen del cubo no era especialmente útil, ya que siempre calculaba el volumen del mismo cubo.

Para mejorar nuestro programa, necesitamos que el usuario sea capaz de introducir las dimensiones.

De eso se encarga la función **input**.

- **input** lee una **cadena** desde la entrada estándar.
- Elimina el retorno de carro que el usuario teclea.
  - En Linux se utiliza internamente la librería readline, por lo que podemos editar la línea normalmente.
- Se puede proporcionar un prompt como parámetro.

Entrada de datos

Para leer un número entero y almacenarlo en la variable i, se utiliza la función **input** de la siguiente manera:

i = int(input())

... ya que **input** lee una cadena de texto.

La función **int** convierte la cadena en un número entero.

i es la variable en la que se va a almacenar el valor leído.

De forma similar, se puede leer un número real con:

f = float(input())

Ejemplo: volumen de un cubo v2

```
#!/usr/bin/env python3

largo = int(input("Introduzca la longitud: "))
ancho = int(input("Introduzca la anchura:  "))
alto = int(input("Introduzca la altura:   "))

volumen = largo * ancho * alto

print("Dimesiones", largo, "x", ancho, "x", alto)
print("El volumen del cubo es:", volumen)
```

Juan Antonio Cos García

Ejecución del programa

```
alumno@ingeniero:~$ python3 cubo.py
Introduzca la longitud: 12
Introduzca la anchura:  23
Introduzca la altura:   34
Dimesiones 12 x 23 x 34
El volumen del cubo es: 9384
alumno@ingeniero:~$
```

... pero el programa tiene un fallo:

¿Qué pasa si la entrada que se teclea no son números?

De momento no nos preocupamos...

... lo resolveremos más adelante

Juan Antonio Cos García

Funciones de conversión

Python proporciona las siguientes funciones de conversión, entre otras:

str([object])

Devuelve una representación como cadena del objeto.

Ejemplos: str(8) -> '8' str(8.32) -> '8.32'

float([x]):

Convierte el parámetro en un número en coma flotante.

Ejemplo: float('3') -> 3.0 float(3) -> 3.0

int([x[, base]])

Convierte una cadena o un número a un entero.

Se puede especificar una base.

Ejemplos: int('8') -> 8 int(2.6) -> 2 int('101', 2) -> 5

Identificadores

Al escribir programas es imprescindible seleccionar nombres que representen funciones, variables, objetos, clases o cualquier otra entidad.

A estos nombres utilizados para *identificar* algo se les denomina **identificadores**.

En Python, un identificador es una secuencia de letras, números y guiones bajos (_).

Palabras reservadas

Las palabras reservadas del lenguaje tienen un significado especial para el compilador y NO PUEDEN ser utilizadas como identificadores.
Las palabras reservadas en Python3 son:

and	finally	or
as	for	pass
assert	from	raise
break	global	return
class	if	try
continue	import	True
def	in	while
del	is	with
elif	lambda	yield
else	None	
except	nonlocal	
False	not	

Literales (constantes)

Un literal es un valor constante que aparece en el programa.

Los literales pueden ser:

- Literales numéricos (numeric literal).
- Cadenas (string literal).

Constantes numéricas

Cuatro tipos en Python:

- Booleanos: **True** y **False**.
  - En Python las constantes True y False se interpretan con el valor entero 1 (verdadero) y 0 (falso) respectivamente.
- Enteros.
  - Números como 1234 se interpretan como números enteros.
  - Añadiendo los prefijos 0, 0x y 0b se pueden expresar las constantes en octal, hexadecimal y binario respectivamente.
    - El número 255 se puede expresar como: 0377, 0xFF, 0b11111111
  - En Python los enteros pueden tener un tamaño arbitrario, basta escribirlos:
  - a = 12345678901234567890123456789012345678901234567890

Constantes numéricas

Coma flotante.

Números como 123.34 y 8.323e3 se interpretan como números en coma flotante

Números complejos.

Números como 1.2 + 12.34j

Cadenas

Una constante cadena se utiliza para especificar una secuencia de caracteres.

Se definen encerrando el texto entre:

- Comillas simples: 'Esto es una cadena'
- Comillas dobles: “Esto es una cadena”
- Tripes comillas (con ''' ó """): '''Esto es una cadena'''

Una cadena es una cadena al margen de las comillas que se usen.

- Hay que usar las mismas comillas para iniciar y finalizar la cadena.
- Hay ciertas diferencias.

Cadenas

Con comillas simples/dobles:

- Se definen en una sola línea.
- Permiten mostrar comillas dobles en el caso de las simples y viceversa.
- Permiten el uso de caracteres especiales como retornos de carro, tabuladores, etc.

Con comillas triples:

- Se puede definir una cadena a través de varias líneas.
- Permiten el uso de caracteres especiales como retornos de carro, tabuladores, etc.

Cadenas

```
#!/usr/bin/env python3

cadena1 = "Esto es una cadena."
cadena2 = 'Y esto también.\n'
cadena3 = "Ser o no ser,\nesa es la cuestión.\n"
cadena4 = "Mostramos una 'comilla simple'..."
cadena5 = '... o una "comilla doble".\n'
cadena6 = """Este texto podría ser la ayuda de un comando
o algo similar, ya que respeta los retornos de carro que
vayamos tecleando."""

print(cadena1)
print(cadena2)
print(cadena3)
print(cadena4)
print(cadena5)
print(cadena6)
```

Secuencias de escape

Una secuencia de escape permite que las cadenas de caracteres (strings) contengan caracteres que podrían causar problemas al intérprete, incluyendo:

- Caracteres no imprimibles (de control).
- Caracteres que tienen un significado especial para el Python (como " ó \).

Ejemplos:

- \n Nueva línea.
- \t Tabulador

Secuencias de escape

Las secuencias de escape representan acciones a realizar sobre la impresión.
El carácter de escape \ también se utiliza para poder mostrar caracteres como \ ó “

\n realiza un retorno de carro.

\\ muestra una contrabarra

Ejemplos:

```
print("Y él dijo: \"¡¡ Me voy !!\"\n")
print('Y él dijo: "¡¡ Me voy !!"\n')
```

Estilo de codificación

(Tomamos C como ejemplo)

Podemos pensar en un programa en C como un conjunto de tokens.

De eso se encargará el analizador léxico del compilador.

La sentencia:

```
printf("El volumen del cubo es: %d\n", volumen);
```

...consta de los tokens:

1.- printf

2.- (

3.- "El volumen del cubo es: %d\n"

4.- ,

5.- volumen

6.-)

7.- ;

Estilo de codificación

La cantidad de espacios (o tabuladores) entre los tokens no es importante.

Los siguientes programas SON equivalentes y generarán el mismo código máquina. Para el compilador son el mismo programa.

Estilo de codificación

```
#include <stdio.h>

int main(void)
{
    printf("A programar se aprende programando.\n");
    return(0);
}
```

```
#include <stdio.h>
int main(void)
{printf("A programar se aprende programando.\n");
return(0);}
```

```
#include <stdio.h>
int main(void) {printf("A programar se aprende  programando.\n");
return(0);}
```

¿Cuál es más legible?

Competición código ofuscado en C:
http://www.ioccc.org/main.html

Código ofuscado

C puede ser muy críptico, IOCCC es un concurso de creación de código C ofuscado.
Ejemplo:

```
char M[2],A,Z,E=40,J[40],T[40];main(C){for(*J=A=scanf("%d",&C);
--            E;            J[            E]            =T
[E   ]=  E)   printf("._");  for(;(A-=Z=!Z)  ||  (printf("\n|"
)    ,   A    =             39              ,C            --
)    ;   Z    ||    printf   (M   ))M[Z]=Z[A-(E   =A[J-Z])&&!C
&    A   ==            T[                                 A]
|6<<27<rand()||!C&!Z?J[T[E]=T[A]]=E,J[T[A]=A-Z]=A,"_.":" |"];}
```

Indentación en Python

En Python no es posible escribir el código anterior.
El intérprete **obliga** a indentar el código.

- Los espacios son importantes para Python.
- A los espacios situados al principio de cada línea se denomina indentación.
  - Se pueden utilizar espacios o tabuladores.
  - NO MEZCLARLOS NUNCA.
  - Se recomienda usar **cuatro espacios por nivel**.
    - Se puede configurar cualquier IDE para que utilice la tecla tabulador como sinónimo de 4 espacios.
- Otros lenguajes usan { } ó begin...end para marcar los bloques, **en Python se usa la indentación**.

Ejemplos:

No:

```
a=2
x=(2+4)*4
numero=int ('23')
a=int (raw_input ('Número: '))
```

Sí:

```
a = 2
x = (2 + 4) * 4
numero = int('23')
a = int(input('Número: '))
```

Bibliografía

- *Apuntes Curso de Python*. Alejandro Roca Alhama