Introducción a la Programación con PYTHON

José Rodriguez

Unidad 1 - Introducción

UNIDAD 1 – INTRODUCCIÓN A PYTHON

INDEX

1.- Lenguajes De Script

Los lenguajes de script son lenguajes de programación diseñados principalmente para automatizar tareas, interpretar instrucciones y controlar otros programas o sistemas, sin necesidad de un proceso de compilación previo.

Un script es un archivo de texto que contiene una serie de instrucciones que se ejecutan de forma secuencial mediante un intérprete, el cual lee y ejecuta el código línea a línea.

Las características más habituales de los lenguajes de script son:

* Lenguajes interpretados
* Sintaxis sencilla y legible
* Desarrollo rápido de aplicaciones
* Alta portabilidad
* Tipado dinámico en muchos casos
* Uso frecuente en automatización

Ejemplos de lenguajes de script:

* Python
* JavaScript
* PHP
* Bash
* PowerShell
* Perl

Las diferencias entre lenguajes compilados e interpretados se pueden resumir en la siguiente tabla

Lenguajes compilados	Lenguajes de script
Requieren compilación	Se ejecutan directamente
Código más complejo	Código más legible
Mayor rendimiento	Menor rendimiento
Control bajo nivel	Automatización y rapidez

Ambos tipos se utilizan según las necesidades del proyecto.

Los lenguajes de script se utilizan principalmente en:

* Automatización de tareas repetitivas
* Administración de sistemas
* Desarrollo web
* Procesamiento de ficheros
* Ciencia de datos
* DevOps

2.- Características de Python

Python es un lenguaje de programación interpretado, de alto nivel y multipropósito. Fue creado por Guido van Rossum y destaca especialmente como lenguaje de script por su sencillez y legibilidad.

Python permite crear scripts para automatizar tareas, gestionar sistemas, trabajar con datos y desarrollar aplicaciones completas.

Las principales características de Python son:

* Lenguaje interpretado
* Sintaxis clara y legible
* Uso de indentación obligatoria
* Tipado dinámico
* Gran biblioteca estándar
* Multiplataforma

Ejemplo de script en Python:

```
nombre = input("Introduce tu nombre: ")
print("Hola", nombre)
```

Python se utiliza ampliamente para automatizar tareas como:

* Gestión de archivos y directorios
* Ejecución de comandos del sistema
* Copias de seguridad
* Administración de servidores

Ejemplo:

```
import os
os.system("ls")
```

Python se diferencia de otros lenguajes de script por:

Una comunidad muy activa
Gran cantidad de módulos externos
Uso tanto en scripting como en aplicaciones grandes
Curva de aprendizaje suave

Ventajas

Código fácil de leer y mantener
Ideal para principiantes
Muy potente para tareas complejas
Amplio soporte y documentación

Desventajas

Menor rendimiento que lenguajes compilados
Gestión manual de entornos y dependencias
No es el mejor para tiempo real estricto

3.- Ejecutando scripts

Existen dos formas de ejecutar código Python, bien en una sesión interactiva (línea a línea) con el intérprete, o bien de la forma habitual, escribiendo el código en un archivo de código fuente y ejecutándolo.

El primer programa que vamos a escribir en Python es el clásico Hola Mundo, y en este lenguaje es tan simple como:

```
print ("Hola Mundo")
```

Vamos a probarlo primero en el intérprete. Ejecuta Python, escribe la línea anterior y pulsa Enter. El intérprete responderá mostrando en la consola el texto Hola Mundo.

Vamos ahora a crear un archivo de texto con el código anterior, de forma que pudiéramos distribuir el programa. Abre tu editor de texto preferido o bien el

IDE que hayas elegido y copia la línea anterior. Guárdalo como hola.py, por ejemplo.

Ejecutar este programa es tan sencillo como indicarle el nombre del archivo a ejecutar al intérprete de Python

python hola.py

Pero vamos a ver cómo simplificarlo aún más. Si utilizas Windows los archivos .py ya estarán asociados al intérprete de Python, por lo que basta hacer doble clic sobre el archivo para ejecutar el programa. Sin embargo como este programa no hace más que imprimir un texto en la consola, la ejecución es demasiado rápida para poder verlo si quiera. Para remediarlo, vamos a añadir una nueva línea que espere la entrada de datos por parte del usuario.

```
print (“Hola Mundo”)
input()
```

De esta forma se mostrará una consola con el texto Hola Mundo hasta que pulsemos Enter.

A veces resulta aconsejable comentar algunas líneas de los script. Para ello utilizamos el carácter # indicando de esta forma que lo que viene a continuación es un comentario.

```
>>> print "Hola Mundo" # Esto es un comentario
Hola Mundo
```

4.- Variables.

En Python encontramos los siguientes tipos de datos simples:

- **Numéricos**
- **Cadenas** de Texto
- Valores **booleanos**:

Para crear una variable de tipo numérico simplemente tenemos que asignarle un valor numérico. Existen tres tipos de variables numéricas que se corresponden con los tipos de números: enteros (int), reales (float) y complejos (complex).

```
>>> a=6
>>> b=12.7
>>> c=5+6j
>>> print a,b,c
6 12.7 (5+6j)
```

Para obtener información sobre el tipo de una variable utilizamos la función type(nombre_variable)

```
>>> type(a)
<type 'int'>
>>> type(b)
<type 'float'>
>>> type(c)
<type 'complex'>
```

4-1 Variables numéricas.

Como hemos visto antes, Python dispone de los tipos necesarios para representar los diferentes formatos numéricos.

- Para representar números enteros se utiliza el tipo **int** y, para valores muy grandes de enteros, el tipo **long.** Obviamente, el tipo long ocupa mucha más memoria por lo que su uso debería estar limitado salvo que fuese necesario. El tipo long de Python permite almacenar números de cualquier precisión, estando limitados solo por la memoria disponible en la máquina.

Por defecto, el valor asignado a una variable entera es de tipo int.

Los valores de las variables del tipo int pueden expresarse en decimal, octal o bien hexadecimal.

```
#El valor viene expresado en hexadecimal
>>> a=0x24
>>> type(a)
<type 'int'>

#El valor viene expresado en Octal
>>> a=0x34
>>> type(a)
<type 'int'>
```

- Las variables que representan a los números reales se caracterizan por el punto decimal y el tipo asociado es float.

Python implementa su tipo **float** a bajo nivel mediante una variable de tipo double de C, es decir, utilizando 64 bits (doble precisión), y en concreto se sigue el estándar IEEE 754: 1 bit para el signo, 11 para el exponente, y 52 para la mantisa.

```
>>> a=8.3
>>> type(a)
<type 'float'>
```

Los datos asignados a una variable de este tipo se pueden escribir en formato normal (con el punto decimal) o bien en notación científica (mantisa+expontente)

```
>>> a=0.4e+2
>>> print a
40.0
```

Los números complejos se caracterizan por tener una parte imaginaria y una parte real. La parte imaginaria va acompañada por la letra j.

```
>>> a=4+5J
>>> print a
(4+5j)
```

4.2 Variables de cadena

Una variable de cadena es un texto encerrado entre comillas dobles o simples. La cadena puede ir precedida del carácter r para indicar el tipo raw (texto plano) o bien por el carácter u (codificación unicode). Una de las diferencias entre ambos formatos es que las cadenas unicode interpretan los caracteres seguidos del símbolo “\” tales como salto de línea, retorno de carro, etc.

Observa la diferencia entre ambas declaraciones

```
>>> a= r"Hola \n mundo"
>>> print a
Hola \n mundo

>>> a=u"Hola \n Mundo"
>>> print a
Hola
 Mundo
```

Otra particularidad de las cadenas en Python es la posibilidad de escribir texto formateado utilizando para ello las comillas triples

```
>>> a= '''Esto es una cadena muy extensa
 que hay que escribirla en varias líneas'''
>>> print a
 Esto es una cadena muy extensa
  que hay que escribirla en varias líneas
```

4.3 Variables Boolenas

Una variable de tipo booleano sólo puede tener dos valores: True (cierto) y False (falso). Estos valores son especialmente importantes para las expresiones condicionales y los bucles,

```
>>> a=True
>>> type(a)
<type 'bool'>
```

Observa que Python es sensible a Mayúsculas, es decir, no es lo mismo true que True. Si asignas a una variable el valor true dará un error.

5.- Operadores

Como todos los lenguajes de programación, Python dispone de una serie de operadores para indicar que debemos hacer con las variables. Estos operadores se pueden clasificar en

5.1 Operadores numéricos.

Indican operaciones aritméticas entre variables de tipo numérico.

Operador	Descripción	Ejemplo
-	Negación	`r = -7 # r es -7`
*	Multiplicación	`r = 2 * 6 # r es 12`
**	Exponente	`r = 2 ** 6 # r es 64`
/	División	`r = 3.5 / 2 # r es 1.75`
//	División entera	`r = 3.5 // 2 # r es 1.0`
%	Módulo	`r = 7 % 2 # r es 1`

5.2 Operadores a nivel de bit.

Expresan operaciones lógicas que operan sobre números bit a bit.

Operador	Descripción	Ejemplo
&	and	`r = 3 & 2 # r es 2`
|	or	`r = 3 | 2 # r es 3`
^	xor	`r = 3 ^ 2 # r es 1`
~	not	`r = ~3 # r es -4`
<<	Desplazamiento izq.	`r = 3 << 1 # r es 6`
>>	Desplazamiento der.	`r = 3 >> 1 # r es 1`

Analicemos algunos de los ejemplos puestos en la tabla.

r = 3 &2

Ambos números, expresados en binario, se escriben como

3= 111
2= 010

El operador & a nivel de bits devuelve 1 si ambos bits son 1 y cero en cualquier otro caso, por lo tanto, el resultado será 010 que resulta ser el número 2 expresado en binario.

r = 3 << 1.

El 3 se escribe en binario como 11, si desplazamos todos los bits a la derecha un lugar introduciendo un 0 por la izquierda, tendremos 110 que resulta ser el número 6 expresado en binario.

5.3 Operadores booleanos.

Estos operadores actúan sobre variables booleanas y podemos clasificarlos en dos grupos:

Operadores lógicos

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	`r = True and False # r es False`
or	¿se cumple a o b?	`r = True or False # r es True`
not	No a	`r = not True # r es False`

Operadores de comparación

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	`r = 5 == 3 # r es False`
!=	¿son distintos a y b?	`r = 5 != 3 # r es True`
<	¿es a menor que b?	`r = 5 < 3 # r es False`
>	¿es a mayor que b?	`r = 5 > 3 # r es True`