

Los ordenadores son dispositivos complejos pero están diseñados para hacer una cosa bien: ejecutar aquello que se les indica. La cuestión es cómo indicar a un ordenador lo que queremos que ejecute. Esas indicaciones se llaman técnicamente instrucciones y se expresan en un lenguaje. Podríamos decir que programar consiste en escribir instrucciones para que sean ejecutadas por un ordenador. El lenguaje que utilizamos para ello se denomina lenguaje de programación.

Pero aún seguimos con el problema de cómo hacer que un ordenador (o máquina) entienda el lenguaje de programación. A priori podríamos decir que un ordenador sólo entiende un lenguaje muy «simple» denominado código máquina. En este lenguaje se utilizan únicamente los símbolos 0 y 1 en representación de los niveles de tensión alto y bajo, que, al fin y al cabo, son los estados que puede manejar un circuito digital. Hablamos de sistema binario. Si tuviéramos que escribir programas de ordenador en este formato sería una tarea ardua, pero afortunadamente se han ido creando con el tiempo lenguajes de programación intermedios que, posteriormente, son convertidos a código máquina.

Si intentamos visualizar un programa en código máquina, únicamente obtendríamos una secuencia de ceros y unos:

```
00001000 00000010 01111011 10101100 10010111 11011001 01000000 01100010
00110100 00010111 01101111 10111001 01010110 00110001 00101010 00011111
10000011 11001101 11110101 01001110 01010010 10100001 01101010 00001111
11101010 00100111 11000100 01110101 11011011 00010110 10011111 01010110
```

Programación

Programar es traducir ideas y acciones a instrucciones que el computador pueda ejecutar, estas instrucciones deben hacerse de forma secuencial y proveer todos los pasos para obtener el resultado final.



El primer lenguaje de programación que encontramos en esta «escalada» es **ensamblador**. Veamos un ejemplo de código en ensamblador del típico programa que se escribe por primera vez, el «Hello, World»:

```
SYS_SALIDA equ 1
section .data
msg db "Hola, Mundo",0x0a
len equ $ - msg ;longitud de msg
section .text
global _start ;para el linker
_start: ;marca la entrada
mov eax, 4 ;llamada al sistema (sys_write)
mov ebx, 1 ;descripción de archivo (stdout)
mov ecx, msg ;msg a escribir
mov edx, len ;longitud del mensaje
int 0x80 ;llama al sistema de interrupciones
```

```
fin: mov eax, SYS_SALIDA ;llamada al sistema (sys_exit)
int 0x80
```

Aunque resulte difícil de creer, lo «único» que hace este programa es mostrar en la pantalla de nuestro ordenador la frase «**Hola, Mundo**», pero además teniendo en cuenta que sólo funcionará para una arquitectura x86.

Python es un lenguaje de programación potente y fácil de aprender. Tiene estructuras de datos de alto nivel eficientes y un simple pero efectivo sistema de programación orientado a objetos. La elegante sintaxis de Python y su tipado dinámico, junto a su naturaleza interpretada lo convierten en un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en muchas áreas, para la mayoría de plataformas.

Python te permite dividir tu programa en módulos que pueden reutilizarse en otros programas de Python. Tiene una gran colección de módulos estándar que puedes utilizar como la base de tus programas o como ejemplos para empezar a aprender Python

Python es un lenguaje interpretado, lo cual puede ahorrarte mucho tiempo durante el desarrollo ya que no es necesario compilar ni enlazar. El intérprete puede usarse interactivamente, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones cuando se hace desarrollo de programas de abajo hacia arriba.

Python permite escribir programas compactos y legibles. Los programas en Python son típicamente más cortos que sus programas equivalentes en C, C++ o Java por varios motivos:

- Los tipos de datos de alto nivel permiten expresar operaciones complejas en una sola instrucción;
- La agrupación de instrucciones se hace mediante indentación en vez de llaves de apertura y cierre;
- No es necesario declarar variables ni argumentos.

Los programas están formados por código y datos. Pero a nivel interno de la memoria del ordenador no son más que una secuencia de bits. La interpretación de estos bits depende del lenguaje de programación, que almacena en la memoria no sólo el puro dato sino distintos metadatos.

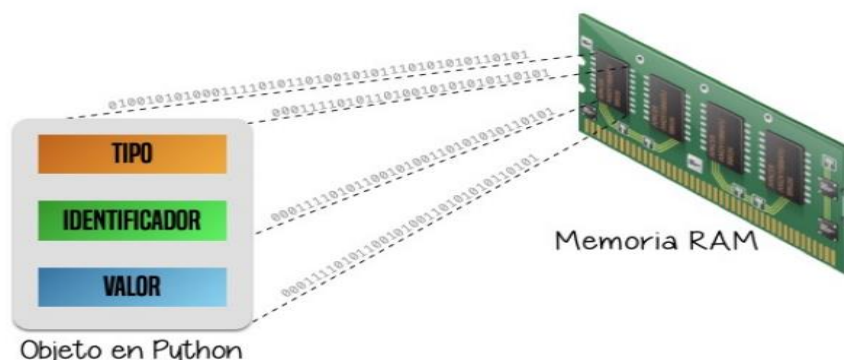


Figura 1: Esquema (*metadatos*) de un objeto en Python

Cada «trozo» de memoria contiene realmente un objeto, de ahí que se diga que en Python todo son objetos. Y cada objeto tiene, al menos, los siguientes campos:

- Un tipo del dato almacenado.
- Un identificador único para distinguirlo de otros objetos.
- Un valor consistente con su tipo.

Tipos de datos

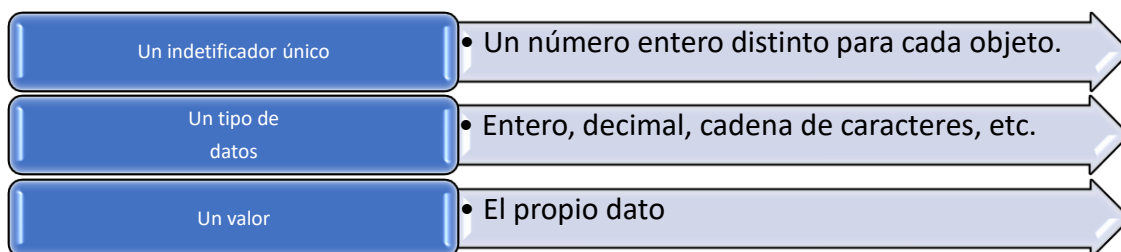
A continuación se muestran los distintos tipos de datos que podemos encontrar en Python, sin incluir aquellos que proveen paquetes externos:

Tabla 1: Tipos de datos en Python

Nombre	Tipo	Ejemplos
Booleano	bool	True, False
Entero	int	21, 34500, 34_500
Flotante	float	3.14, 1.5e3
Complejo	complex	2j, 3 + 5j
Cadena	str	'tfn', '''tenerife - islas canarias'''
Tupla	tuple	(1, 3, 5)
Lista	list	['Chrome', 'Firefox']
Conjunto	set	set([2, 4, 6])
Diccionario	dict	{'Chrome': 'v79', 'Firefox': 'v71'}

Variables

Python es un lenguaje de programación orientado a objetos y su modelo de datos también está basado en objetos. Para cada dato que aparece en un programa, Python crea un objeto que lo contiene. Cada objeto contiene:



Las variables son fundamentales ya que permiten definir nombres para los valores que tenemos en memoria y que vamos a usar en nuestro programa.

Reglas para nombrar variables

En Python existen una serie de reglas para los nombres de variables:

1. Sólo pueden contener los siguientes caracteres
 - Letras minúsculas.
 - Letras mayúsculas.

Dígitos.

Guiones bajos (_).

2. Deben empezar con una letra o un guion bajo, nunca con un dígito.
3. No pueden ser una palabra reservada del lenguaje («keywords»).

Listado de las palabras reservadas del lenguaje

Here is a list of the Python keywords. Enter any keyword to get more help.			
False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

False – Valor booleano, resultado de operaciones de comparación u operaciones lógicas en Python

Una buena práctica, es escribir el nombre de las variables siempre en minúsculas, de ese modo es más difícil equivocarse y no tienes que recordar si escribiste alguna letra de la variable en Mayúscula.

Está claro que si quieres llamarlas así, puedes hacerlo, pero si en el futuro quieres retocar tu programa o lo tienen que hacer otros, la lectura y entendimiento va a ser más pesada, cuando vayas a escribir un nombre de variable formado por varias palabras, nunca dejes espacios tal y como te he dicho. Debes escribirlo con barra baja o con mayúsculas, como te vaya mejor.

Ejemplo: `nombre_empleado` o `nombreEmpleado`

Ejemplos de nombres de variables

Veamos a continuación una tabla con nombres de variables:

Válido	Inválido	Explicación
a	3	Empieza por un dígito
a3	3a	Empieza por un dígito
a_b_c__95	another-name	Contiene un caracter no permitido
_abc	with	Es una palabra reservada del lenguaje
_3a	3_a	Empieza por un dígito

Importante: Los nombres de variables son «case-sensitive»³, Por ejemplo, `stuff` y `Stuff` son nombres diferentes.

Asignación. se usa el símbolo = para asignar un valor a una variable:



Nota: Hay que diferenciar la asignación en Python con la comparación, el símbolo = lo hemos aprendido desde siempre como una equivalencia entre dos expresiones algebraicas, sin embargo en Python nos indica una sentencia de asignación, del valor (en la derecha) al nombre (en la izquierda).

Algunos ejemplos de asignaciones a variables:

```
total= 157503
temperatura = 16.8
city_name = 'San Cristóbal de La Laguna'
```

Python nos ofrece la posibilidad de hacer una asignación múltiple de la siguiente manera:

```
tres = three = drei = 3 #en este caso las tres variables utilizadas
en el «lado izquierdo» tomarán el valor 3.
```

```
a=10 #Asignación tradicional
print(a)
a+=1 #al valor de la vble a, se aumenta en 1, equivale a = a + 1
print(a)
```

La asignación se puede clasificar de la siguiente forma:

- Simples: Almacena un valor constante a una variable. Ejemplo: $a \leftarrow 5$
- Contador: Para llevar un conteo del número de veces que se realiza un proceso. Ejemplo: $a \leftarrow a + 1$
- Acumulador: Es un sumador en un proceso. Ejemplo: $a \leftarrow a + b$
- De trabajo: Guarda el resultado de una operación matemática o expresión. Ejemplo: $a \leftarrow c + b * 2 / 4$

Tipos de datos en Python



A continuación lo que realiza este programa es mostrar en la pantalla de nuestro ordenador la frase «Hola, Mundo»,

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

La función `print()` en los programas, para que python nos muestre texto o variables hay que utilizar la función `print()`, Las cadenas se pueden delimitar tanto por comillas dobles

(") como por comillas simples ('), La función `print()` permite mostrar texto en pantalla. El texto a mostrar se escribe como argumento de la función:

```
print("Hola") #print('Hola')
```

La función `print()` permite incluir variables o expresiones como argumento, lo que nos permite combinar texto y variables:

```
nombre = "Pepe"
edad = 25
print("Me llamo", nombre, "y tengo", edad, "años.")
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\Johan\Documents\CEL\Python> & "C:/Program Files (x86)/Microsoft Visual Studio
/Shared/Python37_86/python.exe" c:/Users/Johan/Documents/CEL/Python/marzo18.py
Me llamo Pepe y tengo 25 años.
PS C:\Users\Johan\Documents\CEL\Python>
```

```
semanas = 4
print("En", semanas, "semanas hay", 7 * semanas, "días.")
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prue... multiplataforma https://aka.ms/pscore6
Enfocar carpeta en el explorador (ctrl + clic)
PS C:\Users\Johan\Documents\CEL\Python> & "C:/Program Files (x86)/Microsoft Visual Studio
/Shared/Python37_86/python.exe" c:/Users/Johan/Documents/CEL/Python/marzo18.py
En 4 semanas hay 28 días.
PS C:\Users\Johan\Documents\CEL\Python>
```

OPERADORES ARITMÉTICOS: Un operador aritmético toma dos operandos como entrada, realiza un cálculo y devuelve el resultado.

Operador	Descripción	Ejemplo
+	Realiza Adición entre los operandos	6 + 3 = 9
-	Realiza Sustracción entre los operandos	6 - 3 = 3
*	Realiza Multiplicación entre los operandos	6 * 3 = 18
/	Realiza División entre los operandos	6 / 3 = 2
%	Realiza un módulo entre los operandos	6 % 3 = 2
**	Realiza la potencia de los operandos	6 ** 3 = 216
//	Realiza la división con resultado de número entero	6 // 3 = 2

OPERADORES RELACIONALES: Los valores booleanos son además el resultado de expresiones que utilizan operadores relacionales (comparaciones entre valores)

Operador	Descripción	Ejemplo
>	Devuelve True si el operador de la izquierda es mayor que el operador de la derecha	12 > 3 devuelve True
<	Devuelve True si el operador de la derecha es mayor que el operador de la izquierda	12 < 3 devuelve False
==	Devuelve True si ambos operandos son iguales	12 == 3 devuelve False
>=	Devuelve True si el operador de la izquierda es mayor o igual que el operador de la derecha	12 >= 3 devuelve True
<=	Devuelve True si el operador de la derecha es mayor o igual que el operador de la izquierda	12 <= 3 devuelve False
!=	Devuelve True si ambos operandos no son iguales	12 != 3 devuelve True

OPERADORES LÓGICOS

Se utiliza un operador lógico para tomar una decisión basada en múltiples condiciones. Los operadores lógicos utilizados en Python son and, or y not)

Operador	Descripción	Ejemplo
and	Devuelve True si ambos operandos son True	a and b
or	Devuelve True si alguno de los operandos es True	a or b
not	Devuelve True si alguno de los operandos False	not a

Condicionales

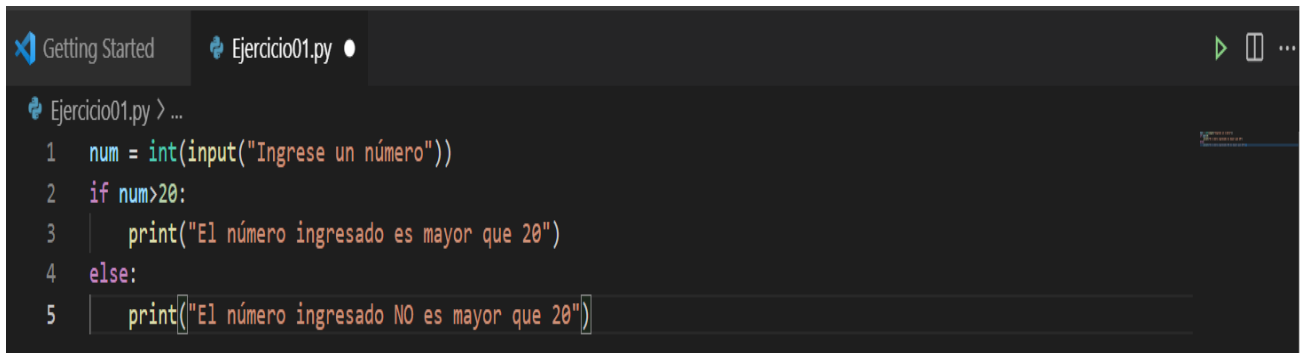
Estas construcciones permiten condicionar la ejecución de uno o varios bloques de sentencias al cumplimiento de una o varias condiciones. Sentencias condicionales: if ...

La estructura de control if ... permite que un programa ejecute unas instrucciones cuando se cumplan una condición.

En inglés "if" significa "si" (condición).

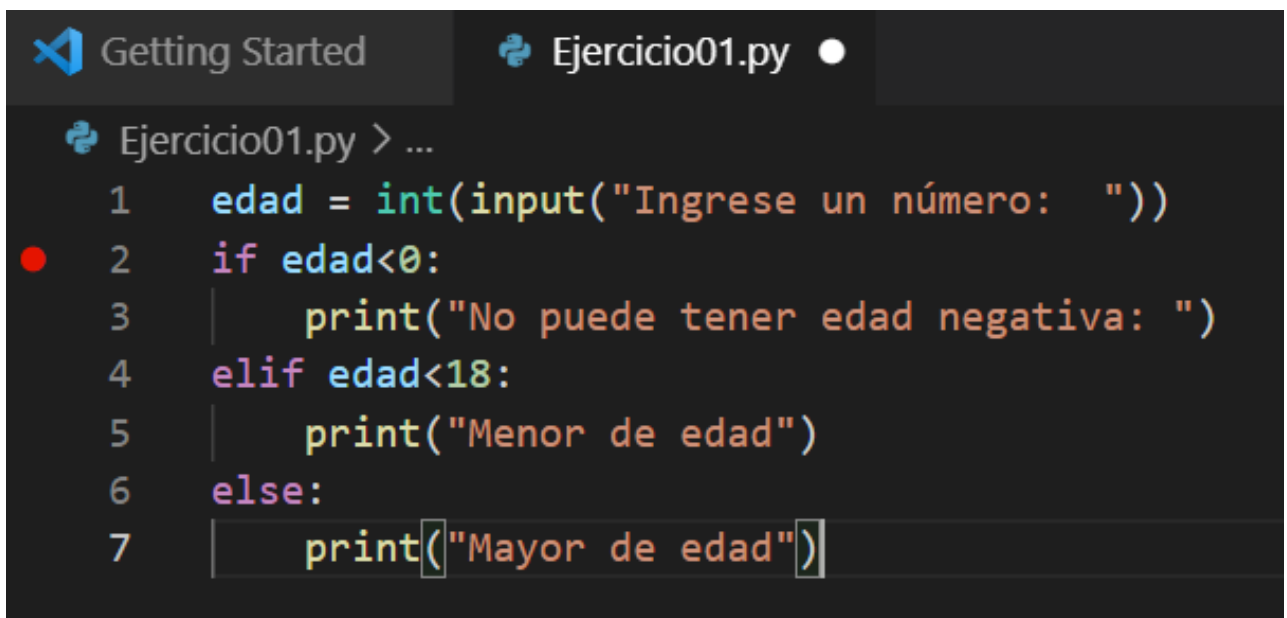
if condición:
 aquí van las órdenes que se ejecutan si la condición es cierta
 y que pueden ocupar varias líneas

else:
 y aquí van las órdenes que se ejecutan si la condición es
 falsa y que también pueden ocupar varias líneas



```
Getting Started  Ejercicio01.py ●
Ejercicio01.py > ...
1 num = int(input("Ingrese un número"))
2 if num>20:
3     print("El número ingresado es mayor que 20")
4 else:
5     print("El número ingresado NO es mayor que 20")
```

Las estructuras condicionales anidadas/múltiples permiten elegir entre varias opciones o alternativas posibles de forma encadenada en función del cumplimiento o no de una determinada condición. Se representa de la siguiente forma:



```
Getting Started  Ejercicio01.py ●
Ejercicio01.py > ...
1 edad = int(input("Ingrese un número: "))
2 if edad<0:
3     print("No puede tener edad negativa: ")
4 elif edad<18:
5     print("Menor de edad")
6 else:
7     print("Mayor de edad")
```

"elif" sirve para enlazar varios "else if", sin tener que aumentar las tabulaciones en cada nueva comparación.

En Python no existe una orden "switch" o "case", sino que se deben realizar enlazando varios casos con "elif".

ESTRUCTURAS REPETITIVAS

Permiten ejecutar un conjunto de instrucciones varias veces, de acuerdo con el valor que genere la expresión relacional y/o lógica, esto significa que una instrucción repetitiva permite saltar a una instrucción anterior para volver a ejecutarla.

Las estructuras repetitivas se utilizan cuando se quiere que un conjunto de instrucciones se ejecuten un cierto número de veces

Ejemplo, escribir algo en pantalla cierta cantidad de veces, mover un objeto de un punto a otro cierta cantidad de pasos, o hacer una operación matemática cierta cantidad de veces

Ciclo While

En Python tiene una palabra reservada llamada `while` que nos permite ejecutar ciclos, o bien secuencias periódicas que nos permiten ejecutar código múltiples veces.

El ciclo `while` nos permite realizar múltiples iteraciones basándonos en el resultado de una expresión lógica que puede tener como resultado un valor `True` o `False`.

Python evalúa la condición: si el resultado es `True` se ejecuta el cuerpo del bucle. Una vez ejecutado el cuerpo del bucle, se repite el proceso (se evalúa de nuevo la condición y, si es cierta, se ejecuta de nuevo el cuerpo del bucle) una y otra vez mientras la condición sea cierta, si el resultado es `False`, el cuerpo del bucle no se ejecuta y continúa la ejecución del resto del programa.

La variable o las variables que aparezcan en la condición se suelen llamar variables de control.

Nota: Las variables de control deben definirse antes del bucle `while` y modificarse en el bucle `while`.

Sintaxis

```
while condición
    cuerpo del bucle
```

```
i=1
while i<=10:
    print(i)
    i +=1
```

Otra ventaja del bucle `while` es que el número de iteraciones no está definida antes de empezar el bucle, por ejemplo porque los datos los proporciona el usuario.

Por ejemplo, el siguiente ejemplo pide un número positivo al usuario una y otra vez hasta que el usuario lo haga correctamente:

```

numero = int(input("Escriba un número positivo: "))
while numero < 0:
    print("¡Ha escrito un número negativo! Inténtelo de nuevo")
    numero = int(input("Escriba un número positivo: "))
print("Gracias por su colaboración")

```

Ejercicios para practicar:

1. Se dispone de un termómetro para medir con exactitud la temperatura en un determinado lugar. Sin embargo, les basta con saber de manera aproximada si la temperatura se ajusta a los rangos siguientes

Temperatura	Sensación térmica
[-10,10)	Mucho frío
[10,15)	Poco frío
[15,25)	Temperatura normal
[25, 30)	Poco calor
[30,45)	Mucho calor

2. Escriba un programa que pida tres números y que escriba si son iguales, hay dos iguales o son distintos.
3. Escriba un programa que pida un año y que escriba si es bisiesto o no. Se recuerda que los años bisiestos son múltiplos de 4, pero los múltiplos de 100 no lo son, aunque los múltiplos de 400 sí.
4. Calcular la suma de los primeros cinco números naturales (1,2,3,4,5)
5. Diseñar un algoritmo que calcule el promedio de notas del primer parcial de un curso de N estudiantes.
6. Realiza las siguientes expresiones en Python para obtener el valor con el que quedará la variable x.

$$x = 7 + 3 * 6 / 2 - 1;$$

$$x = (3 * 9 * (3 + (9 * 3 / (3))));$$

$$x = 3 + 4 * (8 * (4 - (9 + 3) / 6));$$

$$x = a + b * (c * (d - (e + f) / g));$$
7. Dados los valores iniciales de $a = 15$, $b = 3$, cuál será el valor final de las variables después de ejecutar las siguientes expresiones (individualmente).

$$a = a * 6 / a ++;$$

$$b = --b * a++ / b;$$
8. Escriba un programa que pida dos números enteros y que calcule su división, escribiendo si la división es exacta o no.
9. Escriba un programa que pida el año actual y un año cualquiera y que escriba cuántos años han pasado desde ese año o cuántos años faltan para llegar a ese año.