
Unidad 0

Introducción a Python

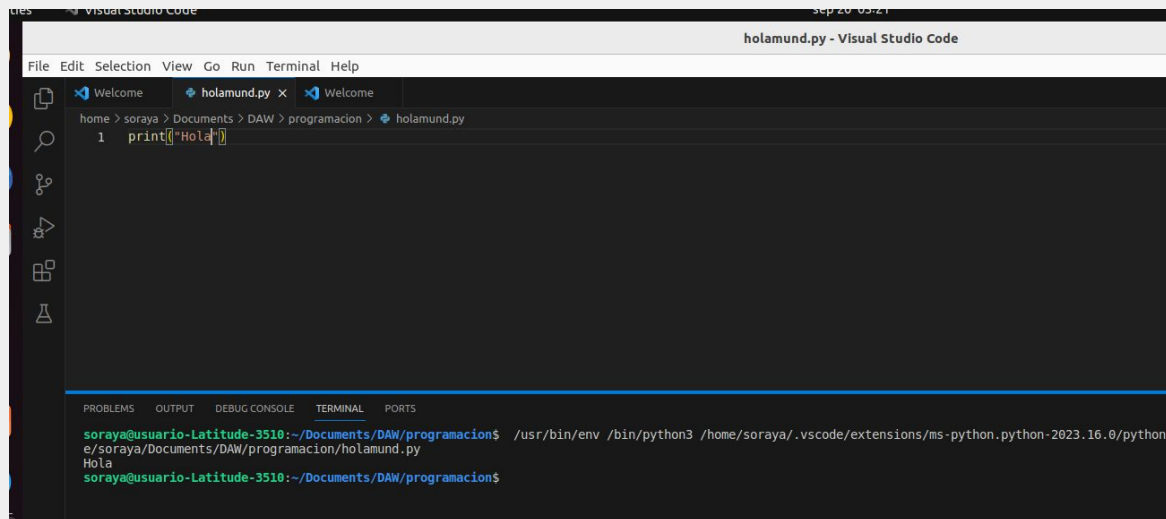
1º DAM

¿Cómo hacer programas en Python?

Software de desarrollo:

Existen diferentes entornos de desarrollo: Visual Studio Code, Sublime Text, Eclipse, NetBeans

En nuestro caso, usaremos Visual Studio Code



```
holamund.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Welcome holamund.py x Welcome
home > soraya > Documents > DAW > programacion > holamund.py
1 print('Hola')

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
soraya@usuario-Latitude-3510:~/Documents/DAW/programacion$ /usr/bin/env /bin/python3 /home/soraya/.vscode/extensions/ms-python.python-2023.16.0/pythonF
e/soraya/Documents/DAW/programacion/holamund.py
Hola
soraya@usuario-Latitude-3510:~/Documents/DAW/programacion$
```

¿Cómo hacer programas en Python?

Visual Studio Code. Instalación

Opción 1: Instalación desde Snap. Abre el Ubuntu software y busca Visual Studio Code

Opción 2:

1. Descarga el paquete con con extensión deb de [Visual Studio Code](#)
2. Abre una ventana de comando y escribe:

```
sudo dpkg -i nombrefichero.deb
```

1. Busca entre tus programas VS Code

Importante: Tras la instalación, deberás instalar en el VS Code una extensión para ejecutar Python. Para ello haz click en View/Extensions y busca Python

Conceptos básicos

Extensión

Un programa en lenguaje Python suele tener la extensión **py**.

Comentarios

Los comentarios nos sirven para aclarar la función de determinadas partes de nuestro código. Se pueden insertar de diversos modos y son obviados por el intérprete de python, es decir, ejecuta el código como si no existieran.

Un comentario de una línea comienza por el símbolo '#'

Se recomienda usar comentarios al principio para indicar qué hace el programa, autor/a, versión.

Conceptos básicos: Variables

Las variables son necesarias para almacenar información en un programa (p.ej.: la edad de una persona, la temperatura, el nombre, etc.).

Nombre de variables:

Las variables en Python solo incluyen letras (a-z, A-B), números y guión bajo (_), no pudiendo empezar por un número. Valores válidos son: usuario, usuario_1, usuario1, pero no 1usuario.

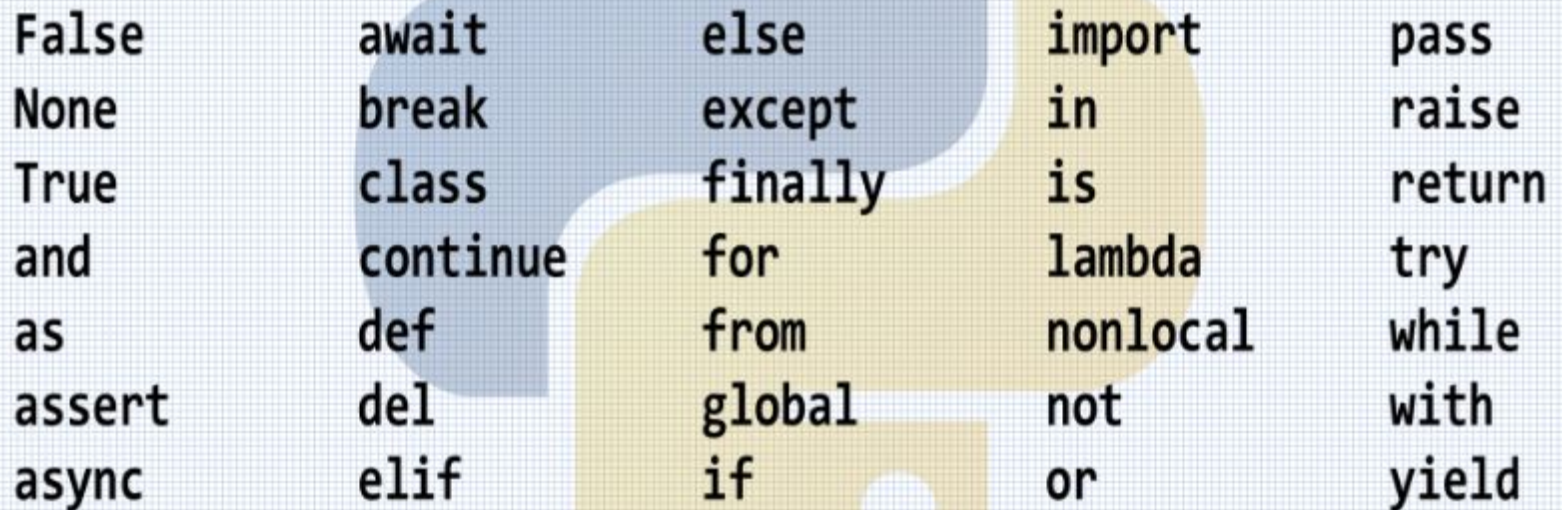
El nombre de las variables es **sensible a mayúsculas**, es decir, edad y Edad son dos variables diferentes.

Tampoco podemos utilizar **palabras reservadas** del lenguaje

Conceptos básicos: Variables

Palabras reservadas:

Las palabras reservadas son palabras que usa el lenguaje para poder funcionar. Es decir, son palabras que no vamos a usar: ni como variables ni como constantes y tampoco vamos a utilizarlas como nombres de funciones.



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

Conceptos básicos: Variables

Para almacenar información en un programa (p.ej.: la edad de una persona, la temperatura, el nombre, etc.) utilizamos las variables. Existen diferentes **tipos básicos**, entre ellos:

- *int*: para valores numéricos enteros
- *string*: para cadenas de texto
- *float*: para valores numéricos con decimales
- *bool*: para variables que almacenan información del tipo Verdadero/Falso

```
edad = 15
nombre = "Antonio"
peso = 63.5
mayorEdad = False
sinInformacion = None;

print(edad, type(edad))
print(nombre, type(nombre))
print(peso, type(peso))
print(mayorEdad, type(mayorEdad))
print(sinInformacion, type(sinInformacion))
```

Conceptos básicos: Variables

Conversión o casting entre tipos:

Cuando estamos trabajando con variables de diferentes tipos (string, int, float) podemos realizar conversiones de un tipo a otro.

Por ejemplo, si tenemos un valor numérico (33) lo podemos convertir a cadena y a la inversa, pero una cadena de texto ('hola') no podremos convertirla en valor numérico y el intérprete nos lanzará un error.

En programación, esta operación de conversión de tipos se denomina casting y es habitual en la mayoría de los lenguajes de programación. Cuando los **tipos son afines** (números, la conversión en ocasiones de entero a decimal) es **automática**, no requiere nuestra intervención, pero cuando es de **otros tipos (string a int)**, necesita una llamada **explícita**.

```
anyo_nacimiento = int(input('Introduce tu año de nacimiento: '))
print('Tu edad en el año 2050 será de %s años'%(2050-anyo_nacimiento))

# Si en lugar de int(input()) utilizáramos solo input(), la operación de resta
# nos daría error porque estamos trabajando con números y cadenas

anyo_nacimiento = input('Introduce tu año de nacimiento: ')
print('Tu edad en el año 2050 será de %s años'%(2050-anyo_nacimiento))
```


Conceptos básicos: Operaciones de E/S

Lectura por teclado

La función **input()** detiene la ejecución del programa y espera a que el usuario escriba un texto y pulse la tecla de retorno de carro; en ese momento prosigue la ejecución y la función devuelve una cadena con el texto que tecleó el usuario.

Admite como parámetro una cadena de caracteres que se mostrará antes del cursor.

```
>>> s = input('--> ')\n--> Monty Python's Flying Circus\n>>> s\n'Monty Python's Flying Circus'
```

Con la instrucción `s = input('-->')`, estamos:

1. Imprimiendo '-->' por pantalla
2. Leyendo un valor
3. Guardando ese valor en una variable llamada s

Conceptos básicos: Operaciones de E/S

Salida

La función **print** nos permite mostrar en la salida estándar (habitualmente la consola) el mensaje que le pasemos como

```
print("Hola mundo")
```

Conceptos básicos: Operaciones

Operadores aritméticos

Operador	Descripción	Ejemplo
+	Suma	<pre>>>> 3 + 2 5</pre>
-	Resta	<pre>>>> 4 - 7 -3</pre>
-	Negación	<pre>>>> -7 -7</pre>
*	Multipliación	<pre>>>> 2 * 6 12</pre>
**	Exponente	<pre>>>> 2 ** 6 64</pre>
/	División	<pre>>>> 3.5 / 2 1.75</pre>
//	División entera	<pre>>>> 3.5 // 2 1.0</pre>
%	Módulo	<pre>>>> 7 % 2 1</pre>

```
# multiplicación de números enteros
print(num_1 * num_2)
```

```
# multiplicación de números decimales
print(num_dec_1 * num_dec_2)
```

```
# multiplicación de entero y decimal
print(num_1 * num_dec_1)
```

Conceptos básicos: Operaciones

Operadores lógicos

Operator	Meaning	Example	Result
and	Logical and	$(5 < 2)$ and $(5 > 3)$	False
or	Logical or	$(5 < 2)$ or $(5 > 3)$	True
not	Logical not	not $(5 < 2)$	True

Habitualmente se utilizan tres tipos:

- **and**: devuelve cierto si las dos expresiones que une son ciertas; falso en caso contrario. Tiene prioridad sobre or.
- **or**: devuelve cierto si una cualquiera de las expresiones que une es cierta; si ninguna lo es, devuelve falso.
- **not**: niega o devuelve lo contrario de lo que recibe. Tiene prioridad sobre or.

Conceptos básicos: Operaciones

Operadores lógicos

Los operadores AND Y OR reciben dos expresiones lógicas. El operador NOT recibe una expresión lógica.

En las siguientes tablas puedes consultar el comportamiento de los operadores según los valores que reciba. Considera que:

- El valor 0 en Python False
- El valor 1 en Python True

INPUT OUTPUT	
A	NOT A
0	1
1	0

INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Conceptos básicos: Operaciones

Operadores lógicos

```
llueve = True  
hace_frio = True  
es_invierno = llueve and hace_frio  
es_invierno
```

True

```
guardar_paraguas = not(llueve)  
guardar_paraguas
```

False

```
coger_chaqueta = llueve or hace_frio  
coger_chaqueta
```

True

```
es_primavera = llueve and not(hace_frio)  
es_primavera
```

False

Conceptos básicos: Operaciones

Operadores Comparación. Devuelven un valor lógico que será True o False si se cumple o no la condición.

operador	comparación
==	es igual que
!=	es distinto de
<	es menor que
<=	es menor o igual que
>	es mayor que
>=	es mayor o igual que

Conceptos básicos: Operaciones

Operadores Comparación. Devuelven un valor lógico que será True o False si se cumple o no la condición.

```
print('3 < 5:', 3 < 5, '\t 3 < 2:', 3 < 2, '\t 3 < 3:', 3 < 3)
print('3 <= 5:', 3 <= 5, '\t 3 <= 2:', 3 <= 2, '\t 3 <= 3:', 3 <= 3)
print('4 > 5:', 4 > 5, '\t 4 > 2:', 4 > 2, '\t 4 > 4:', 4 > 4)
print('4 >= 5:', 4 >= 5, '\t 4 >= 2:', 4 >= 2, '\t 4 >= 4:', 4 >= 4)

print('3==3: ', 3==3, '\t 3!=3: ', 3!=3)
print('1==3: ', 1==3, '\t 1!=3: ', 1!=3)
```