

SINTAXIS PYTHON



Variables

Todos los programas trabajan con información. Para poder gestionar la información se utilizan variables, que son contenedores de información de diferentes tipos de datos.

Para crear una variable simplemente le ponemos un identificador y con el **operador =** le asignamos un valor. Después, utilizaremos el identificador para referirnos a ese valor.

```
1
2 # Las variables se pueden declarar de forma independiente:
3 variable1 = 3
4 variable2 = 'Hola'
5 variable3 = 'Mundo'
6 variable4 = 5.7
7 variable5 = 5 + 6j
8 variable6 = True
9
10 print(variable1)
11 print(variable2)
12 print(variable3)
13 print(variable4)
14 print(variable5)
15 print(variable6)
16
17 # También se pueden declarar en una misma línea:
18 var1, var2, var3 = 'Hola', 'Mundo', 'Qué tal'
19 print(var1)
20 print(var2)
21 print(var3)
22
23 # Podemos imprimirlos también en una misma línea separados de espacio
24 print(var1 + " " + var2 + " " + var3)
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[Running] python -u "/home/alan/developments/python/M1/variables.py"
3
Hola
Mundo
5.7
(5+6j)
True
Hola
Mundo
Qué tal
Hola Mundo Qué tal
```

Variables: identificadores

Un identificador es el nombre que le damos a una variable para poder utilizarla en nuestro código. Se deben respetar unas convenciones a la hora de crear nuevos identificadores:

- Deben comenzar con una letra minúscula o una barra baja _
- No pueden empezar con un número
- El nombre debe estar compuesto por caracteres alfanuméricos o barra baja: A-z, 0-9, _
- Los nombres son case-sensitive, lo que quiere decir que hay distinción entre mayúsculas y minúsculas: Variable1 es distinto de VaRiable1
- Preferiblemente utilizar nombres en inglés

```
1
2 # Nombres correctos
3 ejemploVariable = 3
4 ejemplo_variable = 4
5 _ejemploVariable = 5
6 EJEMPLO_VARIABLE = 6
7 ejemplovariable = 7
8
9 #Nombres incorrectos
10 2ejemploVariable = 3
11 ejemplo-variable = 4
12 ejemplo variable = 5
```

Keywords

Python utiliza una serie de **keywords** o **palabras clave** predefinidas como parte de su sintaxis, se las conoce como **palabras reservadas** y sirven para hacer referencia a alguna funcionalidad de Python, permitiéndonos construir sentencias.

Es por esto que no podemos utilizar las keywords como identificadores para nuestras variables.

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

Comentarios

Para crear código fácilmente mantenible en el futuro, se emplea el uso de comentarios. Los comentarios simplemente son anotaciones sobre el código que sirven de ayuda para los desarrolladores pero que no tienen ningún efecto en la ejecución del código.

Simplemente sirven para aclarar cómo funciona un código y cómo utilizarlo de forma que cualquier persona que lo lea pueda modificarlo o seguir desarrollando a partir del mismo sin dificultad.

```
1
2 # Esto es un comentario de una línea
3
4 # Se pueden poner tantas
5 # líneas como queramos
6
7 """
8 Este es un comentario multilinea
9 el cual podemos abrir con triple comilla simple o doble,
10 tal y como hacemos en la creación de strings
11 """
12
13 print ("Comentarios") # También se puede comentar aquí
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
→ python /usr/bin/python3 /home/alan/developments/python/comments.py
Comentarios
→ python
```

Tipos de datos

Las variables pueden almacenar diferentes tipos de datos, que pueden hacer cosas diferentes: texto, numérico, booleano, binario, etc:

- Tipo texto: **str**
- Tipo numérico: **int**, **float**, **complex**
- Tipo booleano: **bool**
- Tipo secuencia: **list**, **tuple**, **range**
- Tipo mapa: **dict**
- Tipo conjunto: **set**, **frozenset**
- Tipo binario: **bytes**, **bytearray**, **memoryview**

No es necesario especificar manualmente el tipo de cada variable como ocurre en otros lenguajes como Java, Python lo infiere automáticamente.

```
2
3 # Cadena de texto
4 variable = str('Hola mundo')
5 # Numero entero
6 variable = int(20)
7 # Numero decimal
8 variable = float(1.5)
9 # Numero complejo
10 variable = complex(5j)
11 # Booleano, puede ser True o False
12 variable = bool(True)
13
14 ejemploFloat = 5.4
15 # Con la funcion type(x) podemos saber el tipo de datos de la variable x
16 print([type(ejemploFloat)])
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[Running] python -u "/home/alan/developments/python/python1.py"
<class 'float'>

[Done] exited with code=0 in 0.019 seconds
```

En este primer módulo nos centraremos en los tipos de datos básicos: texto, numéricos, booleanos.

En el módulo 2 trabajaremos con los tipos orientados a estructuras de datos: listas, tuplas, diccionarios... etc

Tipos de datos: Strings o cadenas de texto

Las cadenas de texto pueden ser desde letras, hasta palabras o incluso párrafos. Hay diferentes formas de declarar las cadenas de texto:

- Comillas simples 'hola'
- Comillas dobles "hola"
- Triple comilla para poder crear cadenas de texto multilínea

```
python1.py  strings.py X
strings.py > ...
1  variable1 = "Hola"
2  variable2 = 'Hola'
3
4  variable3 = """Lorem ipsum dolor sit amet, consectetur adipiscing elit,
5  sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
6  Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat."""
7
8  variable4 = '''Lorem ipsum dolor sit amet, consectetur adipiscing elit,
9  sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
10 Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.'''
11
12 print(variable1)
13 print(variable2)
14 print(variable3)
15 print(variable4)

OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS
[Running] python -u "/home/alan/developments/python/strings.py"
Hola
Hola
Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

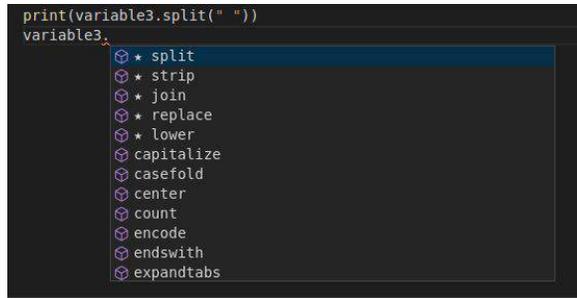
[Done] exited with code=0 in 0.03 seconds
```

Tipos de datos: operaciones con Strings

Python incluye una serie de métodos que podemos utilizar sobre variables de tipo String para realizar todo tipo de operaciones:

- strip()
- lower()
- upper()
- replace()
- split()
- format()

```
print(variable3.split(" "))
variable3.
```



```
1 variable1 = "Hola mundo"
2 # La funcion len(x) nos permite conocer la longitud de una cadena de texto
3 variable2 = len(variable1)
4 print(variable2)
5
6 # El método strip() nos permite eliminar espacios al principio y final
7 variable3 = " Texto con espacios al principio y final "
8 print(variable3)
9 print(variable3.strip())
10
11 # El método lower() nos permite pasar un texto a minúsculas
12 variable3 = "TEXTO EN MAYÚSCULAS"
13 print(variable3.lower())
14
15 # El método upper() nos permite pasar un texto a mayúsculas
16 variable3 = "texto en minúsculas"
17 print(variable3.upper())
18
19 # El método replace() nos permite reemplazar caracteres en un texto
20 variable3 = "Hola Mundo"
21 print(variable3.replace("u","a"))
22 print(variable3)
23 print(variable3.replace("Hola","Adiós"))
24
25 # El método split(x) nos permite reemplazar caracteres en un texto
26 variable3 = "Hola, mundo, qué tal"
27 print(variable3.split(","))
28 variable3 = "Hola mundo"
29 print(variable3.split(" "))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
+ python /usr/bin/python3 /home/alan/developments/python/strings-operations.py
10
Texto con espacios al principio y final
Texto con espacios al principio y final
texto en mayúsculas
TEXTO EN MINUSCULAS
Hola Mundo
Hola Mundo
Adiós Mundo
['Hola', ' mundo', ' qué tal']
['Hola', 'mundo']
```

Tipos de datos: formateo de Strings

El método `format` nos permite formatear partes de una cadena de texto.

Muchas veces un string se compone de datos dinámicos, que no conocemos y que son cambiantes, como aquellos datos que proceden de una base de datos o de la entrada del usuario.

```
1 cost = 49
2 description = "El precio del producto es {} euros"
3 print(description.format(cost))
4
5 quantity = 5
6 itemID = 567
7 price = 49
8 orderDescription = "Compra de {} piezas del producto con id {} con precio {:.2f} euros."
9 print(orderDescription.format(quantity, itemID, price))
10
11 orderDescription = "Compra de {0} piezas del producto con id {1} con precio {2:.2f} euros."
12 print(orderDescription.format(quantity, itemID, price))
13
14 orderDescription = "Compra de {0} piezas del producto con id {1} con precio {2:.2f} euros. Cantidad total = {0}"
15 print(orderDescription.format(quantity, itemID, price))
16
17 orderDescription = "Compra de {quantity} piezas del producto con id {itemID} con precio {price:.2f} euros."
18 print(orderDescription.format(quantity = quantity, itemID = itemID, price = 59.9))
19 |
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
+ python /usr/bin/python3 /home/alan/developments/python/strings-format.py
El precio del producto es 49 euros
Compra de 5 piezas del producto con id 567 con precio 49.00 euros.
Compra de 5 piezas del producto con id 567 con precio 49.00 euros.
Compra de 5 piezas del producto con id 567 con precio 49.00 euros. Cantidad total = 5
Compra de 5 piezas del producto con id 567 con precio 59.98 euros.
+ python
```

Tipos de datos: formateo de Strings

El método `format` nos permite formatear partes de una cadena de texto.

Muchas veces un string se compone de datos dinámicos, que no conocemos y que son cambiantes, como aquellos datos que proceden de una base de datos o de la entrada del usuario.

```
7 price = 49
8 # Al utilizar llaves le indicamos que las rellene con las variables que pasamos en format
9 orderDescription = "Compra de {} piezas del producto con id {} con precio {:.2f} euros."
10 print(orderDescription.format(quantity, itemID, price))
11
12 # Al especificar una posición dentro de las llaves le estamos dando un orden
13 orderDescription = "Compra de {0} piezas del producto con id {1} con precio {:.2f} euros."
14 print(orderDescription.format(quantity, itemID, price))
15
16 # Podemos repetir variables o cambiarlas de orden
17 orderDescription = "Compra de {0} piezas del producto con id {1} con precio {:.2f} euros. Cantidad total = {0}"
18 print(orderDescription.format(quantity, itemID, price))
19
20 # En vez de usar números podemos usar nombres
21 orderDescription = "Compra de {quantity} piezas del producto con id {itemID} con precio {price:.2f} euros."
22 print(orderDescription.format(quantity = quantity, itemID = itemID, price = 59.9))
23
24 # Para formatear podemos usar :< dentro de las llaves para indicar que queremos
25 # forzar a alinear a la izquierda un texto con los espacios indicados
26 head = "Producto\tCantidad"
27 description = "{0:<15}\t{1}"
28 print(head)
29 print(description.format("Coche", 2))
30 description = "{0}\t{1}"
31 # Observar la diferencia sin el formateo de espacios
32 print(description.format("Coche", 2))
33
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
+ python /usr/bin/python3 /home/alan/developments/python/strings-format.py
El precio del producto es 49 euros
Compra de 5 piezas del producto con id 567 con precio 49.00 euros.
Compra de 5 piezas del producto con id 567 con precio 49.00 euros.
Compra de 5 piezas del producto con id 567 con precio 49.00 euros. Cantidad total = 5
Compra de 5 piezas del producto con id 567 con precio 59.90 euros.
Producto      Cantidad
Coche         2
Coche         2
```

Tipos de datos: Booleanos

Utilizamos este tipo de datos cuando queremos expresar un valor verdadero o falso. Por tanto una variable de tipo booleano tendrá uno de dos posibles valores:

- True
- False

Cuando se utilizan operadores de comparación se obtiene como resultado un booleano.

```
1
2 print (5 < 7)
3 booleano1 = 10 > 20
4 print(booleano1)
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
→ python /usr/bin/python3 /home/alan/developments/python/boolean.py
True
False
→ python
```

Tipos de datos: Booleanos

Utilizamos este tipo de datos cuando queremos expresar un valor verdadero o falso. Por tanto una variable de tipo booleano tendrá uno de dos posibles valores:

- True
- False

Cuando se utilizan operadores de comparación se obtiene como resultado un booleano.

```
7 # podemos evaluar cualquier valor o variable
8 print(bool("Hello"))
9 print(bool(15))
10 x = "Hello"
11 y = 15
12 print(bool(x))
13 print(bool(y))
14
15 #Cualquier string es True a menos que esté vacío
16 print(bool(""))
17 # Cualquier número es True a menos que sea 0
18 print(bool(0))
19 print(bool(1))
20
21 # La función predefinida isinstance devuelve boolean
22 x = 200
23 print(isinstance(x, int))
24 print(isinstance(x, float))
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
→ python /usr/bin/python3 /home/alan/developments/python/boolean.py
True
False
True
True
True
True
True
False
False
True
True
False
```

Tipos de datos: tipos numéricos

Existen diferentes formas de representar datos numéricos en Python:

- Números enteros: **int** (longitud de bits infinita, es dinámica, no tiene una longitud máxima como ocurre en otros lenguajes como Java)
- Números coma flotante (decimales): **float**
- Números complejos: **complex**

Nos centraremos en los 2 primeros.

```
1 # números enteros
2 entero1 = 1
3 entero2 = -1
4
5 # números decimales
6 decimal1 = 25.5
7 decimal2 = -25.5
8
9 # números complejos
10 complex1 = 3 + 6j
11 complex2 = -3 - 9j
12
13 print(type(entero1))
14 print(type(entero2))
15 print(type(decimal1))
16 print(type(decimal2))
17 print(type(complex1))
18 print(type(complex2))
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
+ python /usr/bin/python3 /home/alan/developments/python/numbers.py
<class 'int'>
<class 'int'>
<class 'float'>
<class 'float'>
<class 'complex'>
<class 'complex'>
+ python
```

Tipos de datos: conversiones de tipos o casting

Existen determinadas situaciones en las que podemos necesitar especificar un tipo de dato o convertir un dato de un tipo a otro.

Para ello utilizaremos el constructor del tipo de datos que necesitemos, es decir, una función que permite crear una instancia de ese determinado tipo.

```
6
7 print(x)
8 print(y)
9 print(z)
10
11 # Conversiones a float
12 x = float(1)
13 y = int(2.5)
14 z = int("34")
15 w = int(34.4)
16
17 print([x])
18 print(y)
19 print(z)
20 print(w)
21
22 # Conversiones a string
23 x = str(1)
24 y = str(2.5)
25 z = str("Hola")
26
27 print(x)
28 print(y)
29 print(z)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
+ python /usr/bin/python3 /home/alan/developments/python/casting.py
2
2
34
1.0
2
34
34
1
2.5
Hola
```

Tipos de datos: None

En otros lenguajes de programación cuando se quiere crear una variable y se necesita instanciar pero no queremos asignarle un valor, esa variable se inicializa como None.

En Python utilizamos la palabra reservada **None**

```
1
2 variable1 = None
3 print(variable1)
4 print(type(variable1))
5
6 variable1 = 4
7 print(variable1)
8 print(type(variable1))
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[Running] python -u "/home/alan/developments/python/M1/none.py"
None
<class 'NoneType'>
4
<class 'int'>
```

Operadores

Los operadores nos permiten realizar operaciones entre variables y valores. Se agrupan por categorías:

- **Aritméticos**
- Asignación
- Comparación
- Lógicos
- Identidad
- Membresía
- Bit a bit

```
1 |
2 | x = 5
3 | y = 7
4 |
5 | # Operadores aritmeticos
6 |
7 | # suma
8 | x + y
9 | # resta
10 | x - y
11 | # multiplicacion
12 | x * y
13 | # division
14 | x / y
15 | # modulo (resto de la division)
16 | x % y
17 | # exponente
18 | x ** y
19 | # division floor o division suelo
20 | x // y
21 |
```

Operadores

Los operadores nos permiten realizar operaciones entre variables y valores. Se agrupan por categorías:

- Aritméticos
- **Asignación**
- Comparación
- Lógicos
- Identidad
- Membresía
- Bit a bit

```
3 # Operadores de asignacion
4
5 # usamos = para asignar un valor a una variable
6 x = 5
7
8 # Combinar operadores ariméticos, de comparacion y bit a bit
9 # con el operador de asignación para abreviar
10
11 # Equivale a x = x + 5
12 x += 5
13 # Equivale a x = x - 5
14 x -= 5
15 # Equivale a x = x * 5
16 x *= 5
17 # Equivale a x = x / 5
18 x /= 5
19 # Equivale a x = x % 5
20 x %= 5
21 # Equivale a x = x // 5
22 x //= 5
23 # Equivale a x = x ** 5
24 x **= 5
25 # Equivale a x = x & 5
26 x &= 5
27 # Equivale a x = x | 5
28 x |= 5
29 # Equivale a x = x ^ 5
30 x ^= 5
31 # Equivale a x = x >> 5
32 x >>= 5
33 # Equivale a x = x << 5
34 x <<= 5
```

Operadores

Los operadores nos permiten realizar operaciones entre variables y valores. Se agrupan por categorías:

- Aritméticos
- Asignación
- **Comparación**
- Lógicos
- Identidad
- Membresía
- Bit a bit

```
1
2 x = 5
3 y = 7
4
5 # Operadores de comparacion. Los utilizamos para
6 # verificar si se cumplen condiciones entre las variables
7 # el resultado es un valor booleano True o False
8
9 # verifica si x es igual a y
10 x == y
11 # verifica si x es distinto a y
12 x != y
13 # verifica si x es mayor que y
14 x > y
15 # verifica si x es menor que y
16 x < y
17 # verifica si x es mayor o igual que y
18 x >= y
19 # verifica si x es menor o igual que y
20 x <= y
```

Operadores

Los operadores nos permiten realizar operaciones entre variables y valores. Se agrupan por categorías:

- Aritméticos
- Asignación
- Comparación
- **Lógicos**
- Identidad
- Membresía
- Bit a bit

```
x = 5
k = 7
z = 10
# Operadores logicos: Los utilizamos para
# verificar si se cumplen condiciones entre las variables
# el resultado es un valor booleano True o False

# verifica primero si x es menor que k y despues que k es menor que z
# devuelve True solo si se cumplen las dos condiciones, en caso contrario
# devuelve False
x < k and k < z

# verifica primero si x es menor que k y despues que k es menor que z
# devuelve True solo si se cumple una de lass dos condiciones
# si no se cumple ninguna entonces devuelve False
x < k or k < z

# negacion de un booleano, la comparacion x < k nos devuelve True
# al envolverlo en un not negamos ese resultado y se convierte en False
not(x < k)

# Podemos combinar los operadores logicos tantas veces
# como queramos en una misma sentencia
```

Operadores

Los operadores nos permiten realizar operaciones entre variables y valores. Se agrupan por categorías:

- Aritméticos
- Asignación
- Comparación
- Lógicos
- **Identidad**
- Membresía
- Bit a bit

```
1
2 x = 5
3 k = 7
4
5 # Operadores de identidad: comparan si dos objetos son el mismo
6 # no si son iguales, si son el mismo en cuanto a su alojamiento en memoria
7
8 # verifica si x es k y devuelve True o False
9 print(x is k)
10
11 # verifica si x no es k y devuelve True o False
12 print(x is not k)
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

[Running] python -u "/home/alan/developments/python/operators5.py"
False
True

Operadores

Los operadores nos permiten realizar operaciones entre variables y valores. Se agrupan por categorías:

- Aritméticos
- Asignación
- Comparación
- Lógicos
- Identidad
- **Membresía**
- Bit a bit

```
1
2 x = 'hola'
3 k = 'hola mundo'
4
5 # Operadores de membresía: verifican si una secuencia esta contenida
6 # en una variable
7
8 # verifica si el valor de x está contenido en k y devuelve True o False
9 print(x in k)
10
11 # verifica si el valor de x no está contenido en k y devuelve True o False
12 print(x not in k)
13
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

[Running] python -u "/home/alan/developments/python/operators5.py"

True
False

Operadores

Los operadores nos permiten realizar operaciones entre variables y valores.

Se agrupan por categorías:

- Aritméticos
- Asignación
- Comparación
- Lógicos
- Identidad
- Membresía
- **Bit a bit**

El sistema binario es un sistema de numeración con base 2, es decir, que utiliza dos números para representar cantidades, el 0 y el 1, lo que se conoce como un bit de información.

Este sistema representa aquellos números mayores que 1 como secuencias de bits.

Cadena de Bits	0	0	0	0	0
Posición de los Bits	4	3	2	1	0
Valor del bit en numeros decimales	16	8	4	2	1

Operadores

Los operadores nos permiten realizar operaciones entre variables y valores.

Se agrupan por categorías:

- Aritméticos
- Asignación
- Comparación
- Lógicos
- Identidad
- Membresía
- **Bit a bit**

```
# Decimal vs Binario
# 1 = 0000 0001
# 2 = 0000 0010
# 3 = 0000 0011
# 4 = 0000 0100
# 5 = 0000 0101
# 6 = 0000 0110
# 7 = 0000 0111
# 8 = 0000 1000
# 9 = 0000 1001
# 10 = 0000 1010
# 11 = 0000 1011
# 12 = 0000 1100
# 13 = 0000 1101
# 14 = 0000 1110
# 15 = 0000 1111
# 16 = 0001 0000
```

```
18 a = 16          # 16 = 0001 0000
19 b = 13         # 13 = 0000 1101
20 c = 0
21
22
23 # Binary AND. Resultado: 0 = 0000 0000
24 c = a & b
25 print ("Line 1 - Value of c is ", c)
26
27 # Binary OR. Resultado: 29 = 0001 1101
28 c = a | b
29 print ("Line 2 - Value of c is ", c)
30
31 # Binary XOR. Resultado: 29 = 0001 1101
32 c = a ^ b
33 print ("Line 3 - Value of c is ", c)
34
35 # Binary NOT. Equivale: -a-1 lo que es -17 = 1110 1111
36 # Invierte todos los bits de a, es decir, de 16 que es 0001 0000
37 c = ~a
38 print ("Line 4 - Value of c is ", c)
39 print ("-17 en binario ", bin((1<<8) - 17))
40 print (["16 en binario ", bin((1<<8) + 16)])
41
42 # Desplazamiento a la izquierda. 64 = 0100 0000
43 c = a << 2
44 print ("Line 5 - Value of c is ", c)
45
46 # Desplazamiento a la derecha. 4 = 0000 0100
47 c = a >> 2
48 print ("Line 6 - Value of c is ", c)
49
50
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[Running] python -u "/home/alan/developments/python/M1/bitwise.py"
Line 1 - Value of c is 0
Line 2 - Value of c is 29
Line 3 - Value of c is 29
Line 4 - Value of c is -17
-17 en binario 0b11101111
16 en binario 0b10001000
Line 5 - Value of c is 64
Line 6 - Value of c is 4
```

Indentación

El término inglés 'indentation' se traduce al castellano como sangrado o sangría. Hace referencia a nivel de jerarquía que tiene cada fragmento de código y que permite una mejor legibilidad del mismo.

En el caso de Python realizar una correcta indentación es imprescindible para que el programa pueda ejecutarse correctamente.

```
1
2 x = 5
3 y = 6
4
5 # Ejemplo de indentación
6 if x < 6:
7     print ('x es menor que y')
8
9 # Mismo ejemplo, sin indentar, da fallo
10 if x < 6:
11     print ('x es menor que y')
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 1

```
[Running] python -u "/home/alan/developments/python/functions.py"
File "/home/alan/developments/python/functions.py", line 11
    print ('x es menor que y')
    ^
IndentationError: expected an indented block
```

Funciones

Las funciones son bloques de código, secuencias de instrucciones que se agrupan bajo un identificador y que podemos reutilizar en diferentes partes del código simplemente con su identificador, sin necesidad de duplicar ese código.

Contribuyen a:

- Mejor legibilidad
- Evitar código duplicado
- Código más eficiente
- Sencillez

```
1
2 def imprimirHolaMundo():
3     print("Hola mundo")
4
5 def imprimirSaludo(name):
6     print("Hola " + name)
7
8 def imprimirAviso(avis, name):
9     print(avis + " " + name)
10
11 imprimirHolaMundo()
12 imprimirSaludo("Alan")
13 imprimirAviso(["Hola", "Alan"])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
+ python /usr/bin/python3 /home/alan/developments/python/functions.py
Hola mundo
Hola Alan
Hola Alan
```

Funciones

En la conceptualización de un programa, cuando se está definiendo su diseño es posible que se creen métodos vacíos de implementación, para desarrollar el código en el futuro.

Python dará error si un método aparece vacío, es por eso que en estos casos se utiliza la palabra reservada

pass

```
1
2 def myFunction():
3     pass # Si dejamos el método vacío sin poner pass dará error
4
5 myFunction()
```

Funciones

Built-in functions en Python:

<https://docs.python.org/3/library/functions.html>

Built-in Functions				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Resumen sintaxis básica

Hemos visto:

- Variables
- Palabras clave (keywords)
- Comentarios
- Tipos de datos
 - Cadenas de texto
 - Numéricos
 - Booleanos
 - Conversiones
- Indentación (sangrías)
- Operadores
 - Asignación
 - Comparación
 - Lógicos
 - Identidad
 - Membresía
 - Bit a bit
- Funciones