

# Variables, Operadores, Expresiones y Sentencias



Universidad Católica  
**San Pablo**

Departamento de Ciencia  
de la Computación

# Contenido

## ❖ Variables y Expresiones

- Valores y variables
- Sentencias
- Expresiones
- Operadores

# Variables



# Valores y tipos

- Un valor es una de las cosas fundamentales como una letra o un número que un programa manipula.
- Los valores que hemos visto hasta ahora son 2 (el resultado cuando añadimos 1 + 1), y "Hola todo el Mundo!".

- $1+1=2$



ENTERO.

- "Hola todo el mundo"



CADENA

# Valores y tipos

- Para revisar que tipo tiene un valor se usa la función **type( )**.



```
1 print(type("Hola, Mundo!"))
2 print(type(17))
3 print(type(3.2))
4
```




```
<class 'str'>
<class 'int'>
<class 'float'>
```

# Valores y tipos - Observaciones

- Se está usando la función **print( )** para mostrar información por pantalla.
  - `print(variable)`
  - `print(valor)`
- Los tipos de **datos básicos** de Python son los booleanos, los numéricos (enteros, punto flotante y complejos) y las cadenas de caracteres.

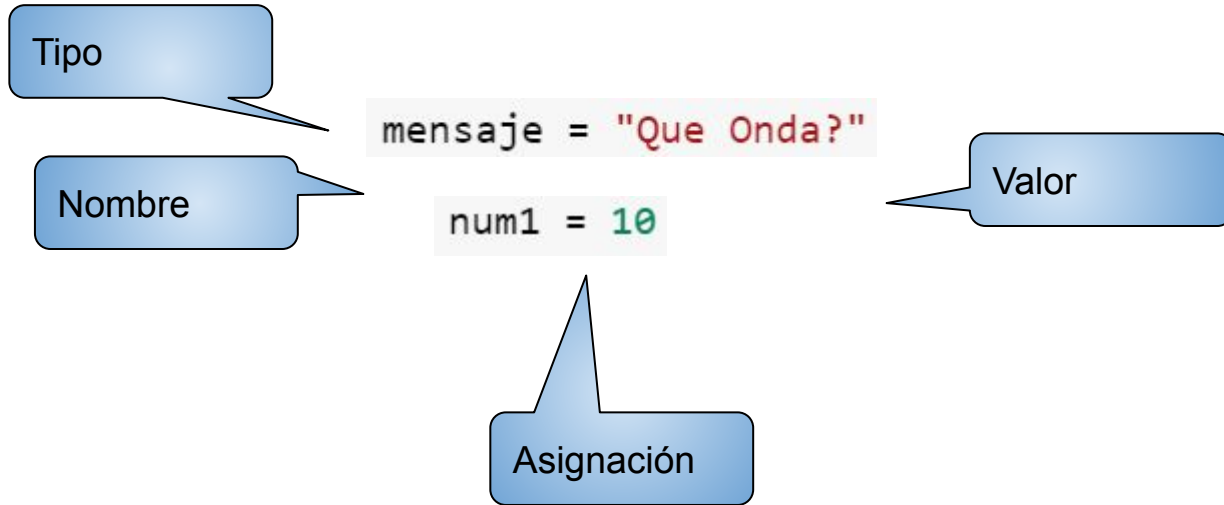
# Variables

- Una de las características más poderosas en un lenguaje de programación es la capacidad de **manipular variables**.
- Las variables permiten **asignar nombres** coherentes a información para ser reutilizada con facilidad.
- Una variable es un **nombre** que se refiere a un valor.
- La **sentencia de asignación** crea nuevas variables y les da valores:



```
1 mensaje = "Que Onda?"  
2 n = 17  
3 pi = 3.14159
```

# Variables



The diagram shows a memory layout with four rows. Each row has a memory address on the left, a value in a colored box in the middle, and a variable name on the right. A callout labeled **Dirección memoria** (Memory address) points to the address column.

	Memoria	
100	10	num1
104	20	num2
108	100	ptr1
112	104	ptr2



# Variables

- La función **print( )** también funciona con variables.
- En cada caso el resultado es el valor de la variable.



```
1 mensaje = "Que Onda?"  
2 num1 = 10  
3 pi = 3.14159  
4 print(mensaje)  
5 print(num1)  
6 print(pi)
```



```
Que Onda?  
10  
3.14159
```

# Variables

- Las variables tienen tipos.
- El **tipo de una variable** es el **tipo del valor** al que se refiere.



```
1 mensaje = "Que Onda?"
2 num1 = 10
3 pi = 3.14159
4 print(type(mensaje))
5 print(type(num1))
6 print(type(pi))
7
```

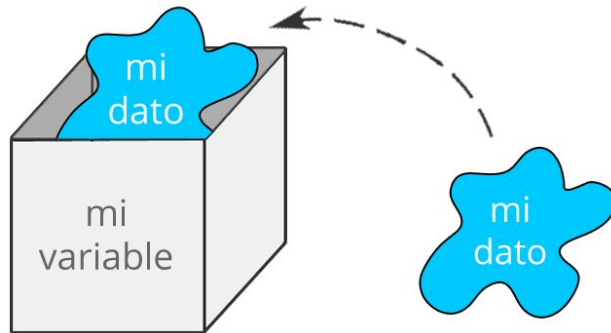


```
<class 'str'>
<class 'int'>
<class 'float'>
```

# Variables

- **Nombres de las variables**

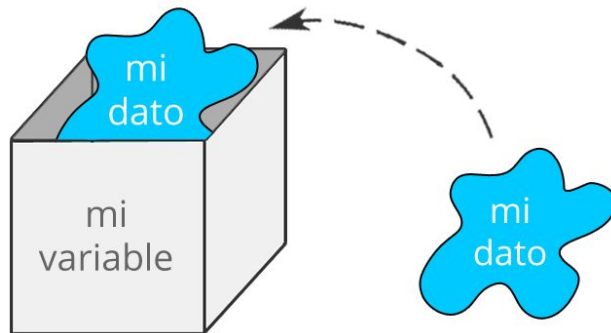
- Escoger nombres significativos para sus variables , que se usa la variable.
- Los nombres pueden ser arbitrariamente largos, teniendo en consideración:
  - Tienen que empezar por una letra o guión bajo.
  - Pueden ser compuestos, separado por guión bajo.
  - Pueden estar en minúsculas o mayúscula (CASE SENSITIVE)
  - El uso de **keywords (palabras reservadas)** como nombres está prohibido.



# Variables

- **Nombres de las variables**

- Tienen que empezar por una letra o guión bajo.
- Pueden ser compuestos, separado por guión bajo.



```
1 cat_color = 'Brown'  
2 cat_Color = 'White'  
3 _threads = 8  
4 variable = 10  
5 2phone_number = 78469334212  
6
```

File "<ipython-input-12-b4b0e9d3fdd3>", line 5

```
2phone_number = 78469334212
```

^

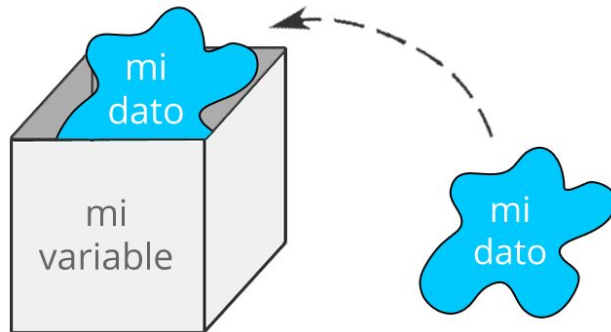
SyntaxError: invalid decimal literal

# Variables

- **Nombres de las variables**

- Pueden estar en minúsculas o mayúscula

**Recuerde:** la capitalización importa, por lo cual, **Pedro** y **pedro** son variables diferentes.



```
1 cat_color = 'Brown'
2 cat_Color = 'White'
3 print(cat_Color)
4 print(cat_color)
```

```
White
Brown
```

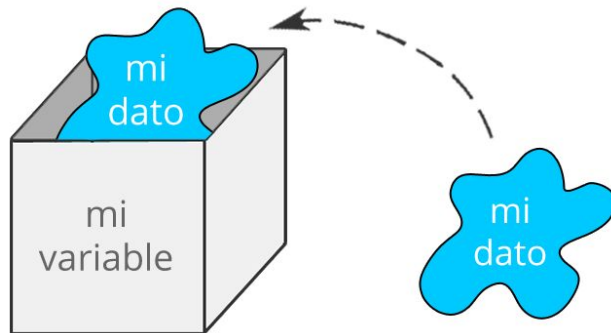
# Variables

- **Nombres de las variables**

- El uso de **keywords (palabras reservadas)** como nombres está prohibido.

**Palabras reservadas:** definen las reglas del lenguaje y su estructura, y no pueden ser usadas como nombres de variables. Python tiene un poco más de treinta palabras reservadas:

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



# Operadores

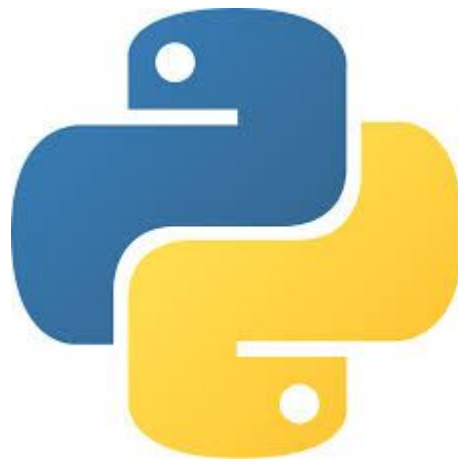


# Operadores

Los operadores son símbolos que le indican al intérprete que realice una operación específica, como aritmética, comparación, lógica, etc.

Estos son los diferentes tipos de operadores en Python:

- Operadores aritméticos
- Operadores relacionales
- Operadores de asignación
- Operadores lógicos





# Operadores y operandos

- Los Operadores son símbolos especiales que representan cálculos como la suma y la multiplicación.
- Los valores que el operador usa se denominan **operandos**.
- Las siguientes son expresiones válidas en Python:

`20+32`

`hora-1`

`hora*60+minuto`

`minuto/60`

`5**2`

`(5+9) * (15-7)`

# Operadores aritméticos

OPERADOR	DESCRIPCIÓN	USO
+	Realiza Adición entre los operandos	$12 + 3 = 15$
-	Realiza Sustracción entre los operandos	$12 - 3 = 9$
*	Realiza Multiplicación entre los operandos	$12 * 3 = 36$
/	Realiza División entre los operandos	$12 / 3 = 4$
%	Realiza un módulo entre los operandos	$16 \% 3 = 1$
**	Realiza la potencia de los operandos	$12 ** 3 = 1728$
//	Realiza la división con resultado de número entero	$18 // 5 = 3$



```
1 print(12 + 3)
2 print(12 - 3)
3 print(12 * 3)
4 print(12 / 3)
5 print(16 % 3)
6 print(12 ** 3)
7 print(18 // 5)
8 print(18 / 5)
```

```
15
9
36
4.0
1
1728
3
3.6
```

# Operadores aritméticos

- Cuando hay más de un operador en una expresión, el orden de evaluación depende de las reglas de precedencia.
  - Python sigue las mismas reglas de precedencia a las que estamos acostumbrados para sus operadores matemáticos.
  - El acrónimo **PEMDAS** es útil para recordar el orden de las operaciones:
    - Paréntesis
    - Exponenciación
    - Multiplicación / División
    - Adición / Sustracción

# Operadores relacionales

OPERADOR	DESCRIPCIÓN	USO
>	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



```
1 print(12 > 3)
2 print(12 < 3)
3 print(12 == 3)
4 print(12 >= 3)
5 print(12 <= 3)
6 print(12 != 3)
7
```

True  
False  
False  
True  
False  
True

# Operadores de asignación

OPERADOR	DESCRIPCIÓN	USO
=	a = 5. El valor 5 es asignado a la variable a	=
+=	a += 5 es equivalente a a = a + 5	+=
-=	a -= 5 es equivalente a a = a - 5	-=
*=	a *= 3 es equivalente a a = a * 3	*=
/=	a /= 3 es equivalente a a = a / 3	/=



```
1 a = 5
2 print(a)
3 a += 5
4 print(a)
5 a -= 5
6 print(a)
7 a *= 3
8 print(a)
9 a /= 3
10 print(a)
11
```



```
5
10
5
15
5.0
```

# Operadores lógicos

OPERADOR	DESCRIPCIÓN	USO
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

# Operaciones sobre cadenas

En general, no se puede calcular operaciones matemáticas sobre cadenas, incluso si las cadenas lucen como números.

Las siguientes operaciones son inválidas (asumiendo que mensaje tiene el tipo cadena):

- mensaje = "Soy de Peru"
- mensaje-1
- "Hola"/123
- mensaje\*"Hola"
- "15"+2

```
1 mensaje = "Soy de Peru"
2 mensaje-1
3 "Hola"/123
4 mensaje*"Hola"
5 "15"+2
6
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-20-1466dca96904> in <cell line: 2>()
      1 mensaje = "Soy de Peru"
----> 2 mensaje-1
      3 "Hola"/123
      4 mensaje*"Hola"
      5 "15"+2
```

TypeError: unsupported operand type(s) for -: 'str' and 'int'

# Operaciones sobre cadenas

- El **operador +** funciona con cadenas, aunque no calcula lo que esperamos. Representa la concatenación, que significa unir los dos operandos enlazándolos en el orden en que aparecen.
- El **operador \*** funciona con cadenas, concatenando varias veces una de ellas.



```
1 msj1 = "Hola"  
2 msj2 = "Amigo"  
3 print(msj1+msj2)
```

HolaAmigo



```
1 msj1 = "Hola"  
2 print(msj1*5)
```

HolaHolaHolaHolaHola



# Operaciones sobre cadenas

Las letras mayúsculas vienen antes que las minúsculas es decir son menores que las minúsculas.

Para ordenarlas, se usa el orden ortográfico.



```
1 print("hola" <= "ola")
2 print("hola" < "ola")
3 print("hola" >= "ola")
4 print("hola" > "ola")
5 print("hola" == "ola")
6 print("hola" == "HOLA")
7 print("hola" > "HOLA")
8
```



```
True
True
False
False
False
False
True
```

# Expresiones




# Expresiones

- Una **expresión** es una combinación de valores, variables y operadores.
  - $1 + 3$
  - $\text{valor1} + 5$
  - $(\text{valor1} * 5) + \text{radio}$
  - $1\ 3\ * \ -$  (no es aceptado porque no sigue las reglas de sintaxis)
- Las expresiones son **evaluadas** por el intérprete
  - Si usted digita una expresión en la línea de comandos, el intérprete la evalúa y despliega su resultado:

```
1 valor1 = 7
2 nro = 9
3 total = (valor1 * 5) + nro
4 print(total)
```

# Expresiones

- Un valor, por sí mismo, se considera como una expresión, lo mismo ocurre para las variables.
- Aunque es un poco confuso, evaluar una expresión no es lo mismo que imprimir o desplegar un valor. En el primer caso (líneas 1 y 2) la expresión se evalúa, es decir, se va a calcular la operación, sin embargo este resultado no se va a imprimir. En el segundo caso (línea 4), además de evaluarse la expresión está también se imprime.



```
1 17
2 x = 2
3
4 print(4+5)
5
6
```

9

# Comentarios

- A medida que los programas se hacen más grandes y complejos, se hacen más difíciles de leer. Los lenguajes formales son densos; y a menudo, es difícil mirar una sección de código y saber que hace, o por qué lo hace.
- Por esta razón, es una muy buena idea añadir notas a sus programas para explicar en lenguaje natural lo que el programa hace. Estas notas se denominan **comentarios** y se marcan con el símbolo # para cada línea, o entre 3 apóstrofes (') para varias líneas.



```
1 # calcula el porcentaje de la hora que ha pasado
2 minuto = 60
3 porcentaje = (minuto * 100) / 60
4 ''' calcula el porcentaje
5 de la hora
6 que ha pasado '''
7 porcentaje = (minuto * 100) / 60
8
```



*Gracias!*