

# DESCUBRIENDO EL PODER DE LA PROGRAMACION

CURSO INICIAL DE PYTHON

Jimmy Chung  
Alexander Solis



# CONTENIDO DEL CURSO

1. **Introducción, Instalación y Conceptos básicos**
2. **[Variables, Expresiones, Funciones y Operadores](#)**
3. **Condicionales y Ciclos – Patrones de sintaxis válidos**
4. **Listas, tuplas y diccionarios**
5. **Errores y Excepciones**
6. **Clases y funciones en Python**



# **Tema 3: Variables, Expresiones, Funciones y Operadores**



# Funciones «built-in»

Funciones «built-in»: Incorporadas por defecto en el propio lenguaje

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

Hemos utilizado `print()`, `type()`, `len()`.  
Pero existen muchas mas.

<https://docs.python.org/es/3/library/functions.html?highlight=built>



# FUNCIONES

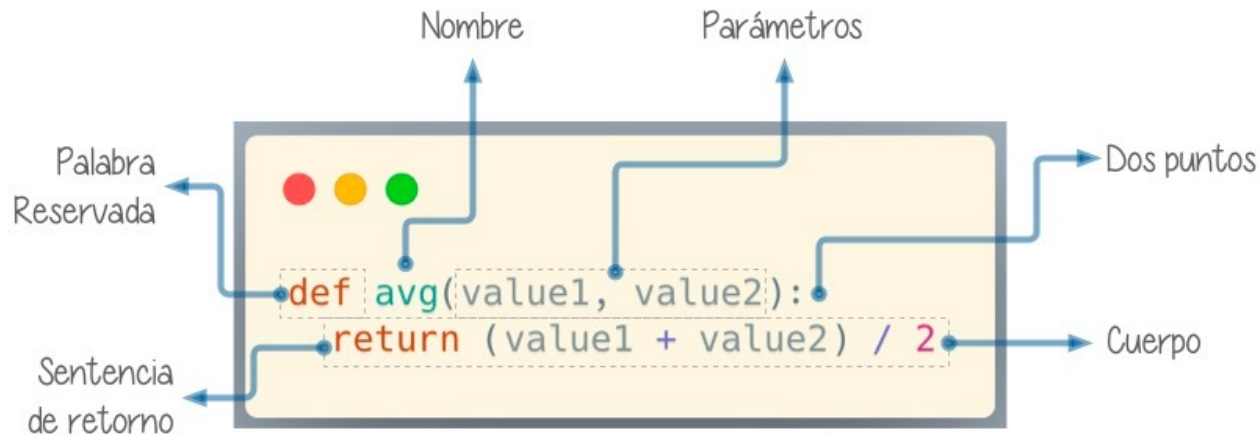
Una función es un bloque de código que se define con un nombre y que agrupa un conjunto de instrucciones relacionadas. Pueden recibir argumentos como entrada, realizar un procesamiento o cálculo, y devolver un resultado opcional. Las funciones nos permiten dividir nuestro programa en partes más pequeñas, mejorando la legibilidad, la **reutilización** del código y la **modularidad**.

```
def saludar():  
    print("¡Hola! ¡Bienvenido al curso de Python!")  
  
saludar() # Llamada a la función
```



# Definir una función

Para definir una función utilizamos la palabra reservada **def** seguida del nombre de la función. A continuación aparecerán **0 o más parámetros** separados por comas (entre paréntesis), finalizando la línea con **dos puntos**. En la siguiente línea empezaría el cuerpo de la función que puede contener 1 o más sentencias, incluyendo (o no) una sentencia de retorno con el resultado mediante **return**.



# Funciones: return y pass

**pass:** Permite «no hacer nada». Es una especie de «placeholder». Útil en programación Orientada a Objetos.

**return:** Permite:

- Salir de una función y transferir la ejecución a la parte del código donde se realizó la llamada.
- Devolver uno o varios parámetros, como resultado de la ejecución de la función.



# Funciones: Parámetros y Argumentos

**Parámetros:** Son variables que se definen en la declaración de una función.

**Argumentos:** Son los valores reales que se pasan a una función cuando se la llamamos.

The diagram illustrates the relationship between function parameters and arguments. It shows a function definition and a function call with corresponding labels and arrows.

```
def my_pretty_function(value_a, value_b, value_c):  
    pass
```

Parámetros

Argumentos

```
>>> my_pretty_function('Hola', 3.14, {1, 2, 3})
```

Three blue arrows point from the arguments in the function call to the parameters in the function definition: from 'Hola' to value\_a, from 3.14 to value\_b, and from {1, 2, 3} to value\_c.





# Funciones: Parámetros y Argumentos

- **Argumentos por posición**: Es la forma más básica e intuitiva de pasar parámetros (en el mismo orden de su definición).
- **Argumentos por nombre**: Llamar a una función, es usando el nombre del argumento con = y su valor.
- **Valores por defecto**: Parámetros opcionales que toman el valor por defecto.

```
def resta(a, b):  
    return a-b  
resta(5, 3) # 2
```

```
resta(a=3, b=5) # -2
```

```
def suma(a, b, c=0):  
    return a+b+c  
suma(5,5,3) # 13
```

```
suma(4,3) # 7
```



# Funciones: Documentación

Para documentar una función debemos usar la triple comilla `"""` al principio de la función. Es un comentario para indicar como debe ser usada la función, esto se conoce como **docstring**.

```
def mi_funcion_suma(a, b):  
    """  
    Descripción de la función. Como debe ser usada,  
    que parámetros acepta y que devuelve  
    """  
    return a+b
```

Para consultar la ayuda de la función (`?`, `help()`, `__doc__`).

```
help(mi_funcion_suma)
```

```
print(mi_funcion_suma.__doc__)
```



# Funciones: anotaciones de tipo

Las anotaciones de tipos o type-hints se introdujeron en [Python 3.5](#) y permiten indicar tipos para los parámetros de una función y/o para su valor de retorno

```
def multiplica_por_3(numero: int) -> int:  
    return numero*3
```

```
multiplica_por_3(6) # 18
```

Las anotaciones son útiles solo para documentar el código.

```
multiplica_por_3("Cadena")  
# 'CadenaCadenaCadena'
```



# CADENAS O STRING

Los Strings son una secuencia de caracteres como letras, números y símbolos, y deben escribirse entre comillas dobles ("texto") o sencillas ('texto').

```
s = "Esto es una cadena"  
print(s)          #Esto es una cadena  
print(type(s))    #<class 'str'>
```

```
s = 'Esto es otra cadena'  
print(s)          #Esto es otra cadena  
print(type(s))    #<class 'str'>
```



# STRINGS: Dar formato

El caracter '%', que identifica el inicio del marcador.

```
x = 5
s = "El número es: %d" % x
print(s) #El número es: 5
```

**Nota:** Las operaciones de formateo explicadas aquí tienen una serie de peculiaridades que conducen a ciertos errores comunes (como fallar al representar tuplas y diccionarios correctamente). Se pueden evitar estos errores usando las nuevas [cadenas de caracteres con formato](#), el método `str.format()`, o [plantillas de cadenas de caracteres](#). Cada una de estas alternativas proporcionan sus propios compromisos entre facilidad de uso, flexibilidad y capacidad de extensión.

<https://docs.python.org/es/3/library/stdtypes.html#printf-style-string-formatting>



# STRINGS: Dar formato

## Literales de cadena formateados.

Es una cadena que se prefija con 'f' o 'F'

```
a = 5; b = 10
s = f"Los números son {a} y {b}"
print(s) #Los números son 5 y 10
```

Se puede hacer un calculo o llamar una funcion al dar formato

```
a = 5; b = 10
s = f"a + b = {a+b}"
print(s) #a + b = 15
```

```
def funcion():
    return 20
s = f"El resultado de la función es {funcion()}"
print(s) #El resultado de la funcion es 20
```

[https://docs.python.org/es/3/reference/lexical\\_analysis.html#f-strings](https://docs.python.org/es/3/reference/lexical_analysis.html#f-strings)



# STRINGS: Dar formato

## Funcion format()

Realiza una operación de formateo, las marcas de reemplazo de texto están definidas por llaves {}.

```
s = "Los números son {} y {}".format(5, 10)
print(s) #Los números son {} y {}".format(5, 10)
```

Es posible también darle nombre a cada elemento

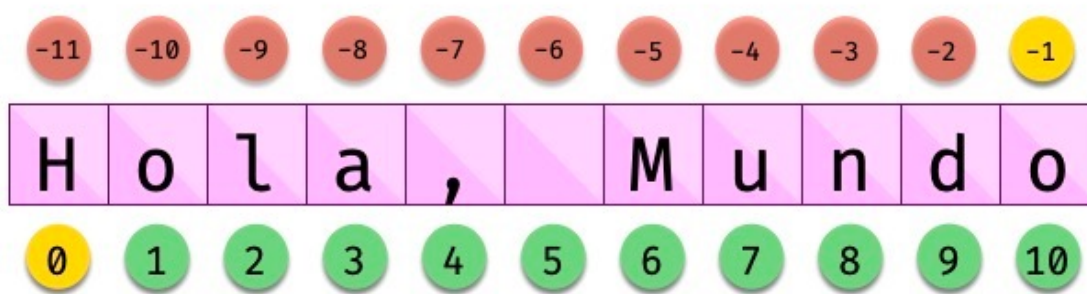
```
s = "Los números son {a} y {b}".format(a=5, b=10)
print(s) #Los números son 5 y 10
```

<https://docs.python.org/es/3/library/stdtypes.html#str.format>



# Operaciones con Strings

Obtener un caracter: Cada caracter tiene su posición (Índice)



```
>>> sentence = 'Hola, Mundo'
>>> sentence[0]
'H'
>>> sentence[-1]
'o'
>>> sentence[4]
','
>>> sentence[-5]
'M'
```

Si intentamos acceder a un índice que no existe, obtendremos un error "fuera de rango"

```
>>> sentence[50]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```





# Extraer trozos (partes) de un String

**[:]** Extrae toda la cadena (una copia)

**[start:]** Extrae desde start hasta el final de la cadena.

**[:end]** Extrae desde el inicio de la cadena hasta end menos 1.

**[start:end]** Extrae desde start hasta end menos 1.

**[start:end:step]** Extrae desde start hasta end menos 1, haciendo saltos de tamaño step.

```
>>> proverb = 'Agua pasada no mueve molino'

>>> proverb[:]
'Agua pasada no mueve molino'

>>> proverb[12:]
'no mueve molino'

>>> proverb[:11]
'Agua pasada'

>>> proverb[5:11]
'pasada'

>>> proverb[5:11:2]
'psd'
```



# String: Principales funciones

Pertenencia de un elemento. `'lago' in 'murcielago': True`

`len()`: Longitud de una cadena. `len('casa'): 4`

`split()`: Dividir una cadena. `len(str, set=' ')`

`startswith(str)`: Indica si una cadena inicia con str

`endswith(str)`: Indica si una cadena finaliza con str

`find(str)`: Encontrar la primera ocurrencia de str. (-1 si no existe)

`index(str)`: Encontrar la primera ocurrencia de str. (Error si no existe)

`count()`: contar el número de veces que aparece str.



# String: Principales funciones

repalce(): Reemplazar elementos.

```
>>> proverb = 'Quien mal anda mal acaba'

>>> proverb.replace('mal', 'bien')
'Quien bien anda bien acaba'

>>> proverb.replace('mal', 'bien', 1)  # sólo 1 reemplazo
'Quien bien anda mal acaba'
```



# String: Principales funciones

## Conversión de mayúsculas y minúsculas

`capitalize()`: Primera letra en mayúscula

`title()`: En mayúscula la primera letra de cada palabra.

`upper()`: En mayúscula todas las letras.

`lower()`: En minúscula todas las letras.

`swapcase()`: Cambia entre minúsculas y mayúsc.

```
>>> proverb = 'quien a buen árbol se arrima Buena Sombra le cobija'

>>> proverb
'quien a buen árbol se arrima Buena Sombra le cobija'

>>> proverb.capitalize()
'Quien a buen árbol se arrima buena sombra le cobija'

>>> proverb.title()
'Quien A Buen Árbol Se Arrima Buena Sombra Le Cobija'

>>> proverb.upper()
'QUIEN A BUEN ÁRBOL SE ARRIMA BUENA SOMBRA LE COBIJA'

>>> proverb.lower()
'quien a buen árbol se arrima buena sombra le cobija'

>>> proverb.swapcase()
'QUIEN A BUEN ÁRBOL SE ARRIMA buena sombra LE COBIJA'
```



# String: Principales funciones

isalnum(): Detectar si todos los caracteres son letras o números.  
isnumeric(): Detectar si todos los caracteres son números.  
isalpha(): Detectar si todos los caracteres son letras.  
isupper(): Detectar si todos los caracteres son mayúsculas.  
islower(): si todos los caracteres son minúsculas.  
istitle(): Detectar si la primera letra de cada palabra es mayúscula.

La clase **str** tiene muchas funciones adicionales, que podemos describir y con la función built-in **dir()**,

```
>>> texto = 'Hola!'
>>> dir(text)
```



# OPERADORES

Los operadores son símbolos reservados por el propio lenguaje que se utilizan para llevar a cabo operaciones sobre uno, dos o más elementos llamados operandos. Los operandos pueden ser variables, literales, el valor devuelto por una expresión o el valor devuelto por una función.

```
>>> 7 + 3 #7 y 3 son los operandos
```

```
>>> 10 #10 es el resultado
```



# Operador de concatenación de cadenas

Combinar cadenas: Signo u operador +

```
>>> proverb1 = 'Cuando el río suena'  
>>> proverb2 = 'agua lleva'  
  
>>> proverb1 + proverb2  
'Cuando el río suenaagua lleva'  
  
>>> proverb1 + ', ' + proverb2  # incluimos una coma  
'Cuando el río suena, agua lleva'
```

Repetir cadenas: Signo u operador \*

```
>>> reaction = 'Wow'  
  
>>> reaction * 4  
'WowWowWowWow'
```



# Operadores de comparación

Nombre	Operador	Ejemplo	Resultado
# Igual que	<code>==</code>	<code>1 == 1</code>	True
# Diferente a	<code>!=</code>	<code>"a" != "a"</code>	False
# Mayor que	<code>&gt;</code>	<code>10 &gt; 5</code>	True
# Menor que	<code>&lt;</code>	<code>5 &lt; 1</code>	False
# Mayor o igual que	<code>&gt;=</code>	<code>30 &gt;= 30</code>	True
# Menos o igual que	<code>&lt;=</code>	<code>20 &lt;= 10</code>	False

## Notas:

1. El resultado siempre es un Boolean (True/False).
2. Los objetos a comparar deben ser compatibles entre sí.





# Operadores lógicos o booleanos

## OR, AND, NOT

Operación	Resultado	Descripción
<code>a or b</code>	Si <code>a</code> se evalúa a falso, entonces devuelve <code>b</code> , si no devuelve <code>a</code>	Solo se evalúa el segundo operando si el primero es falso
<code>a and b</code>	Si <code>a</code> se evalúa a falso, entonces devuelve <code>a</code> , si no devuelve <code>b</code>	Solo se evalúa el segundo operando si el primero es verdadero
<code>not a</code>	Si <code>a</code> se evalúa a falso, entonces devuelve <code>True</code> , si no devuelve <code>False</code>	Tiene menos prioridad que otros operadores no booleanos

Los operadores `and`, `or` y `not` realmente no devuelven `True` o `False`, devuelven uno de los operandos.



# Operadores aritméticos

Operador	Descripción
+	Suma dos operandos.
-	Resta al operando de la izquierda el valor del operando de la derecha. Utilizado sobre un único operando, le cambia el signo.
*	Producto/Multiplicación de dos operandos.
/	Divide el operando de la izquierda por el de la derecha (el resultado siempre es un <code>float</code> ).
%	Operador módulo. Obtiene el resto de dividir el operando de la izquierda por el de la derecha.
//	Obtiene el cociente entero de dividir el operando de la izquierda por el de la derecha.
**	Potencia. El resultado es el operando de la izquierda elevado a la potencia del operando de la derecha.



# Operadores aritméticos

Expresiones numéricas: Orden de evaluación.

- Cuando introducimos una cadena de operadores, Python debe saber cuál tiene que hacer primero
- Esto recibe el nombre de “precedencia del operador”
- Ahora, ¿qué operador “tiene precedencia” sobre los otros?

`x = 1 + 2 * 3 - 4 / 5 ** 6`



# Operadores aritméticos

## Reglas de Precedencia del

### Operador:

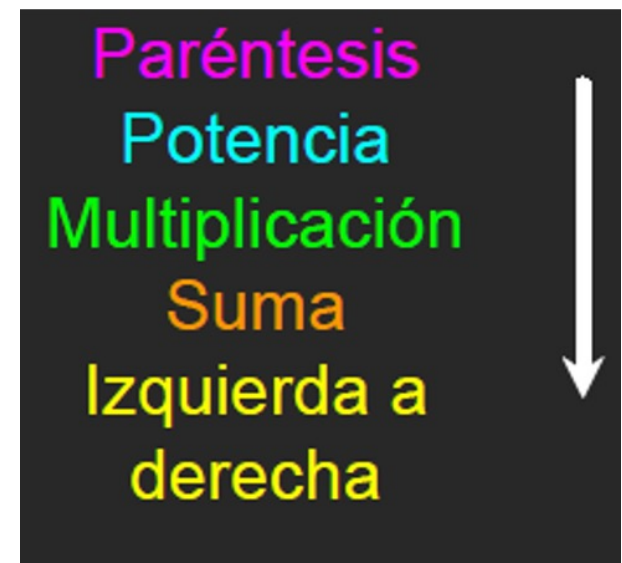
- De la regla de precedencia más alta a la regla de precedencia más baja:

- + Siempre se respetan los paréntesis

- + Potenciación (elevar a la potencia)

- + Multiplicación, división, resto


- + Suma y resta



# Operadores aritméticos

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```

Paréntesis  
Potencia  
Multiplicación  
Suma  
Izquierda a  
derecha



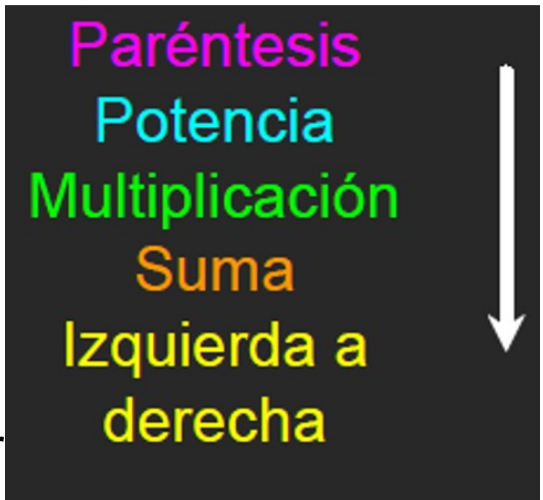
1 + 2 \*\* 3 / 4 \* 5  
↓  
1 + 8 / 4 \* 5  
↓  
1 + 2 \* 5  
↓  
1 + 10  
↓  
11



# Operadores aritméticos

## Reglas de Precedencia del Operador:

- Recuerde las reglas de arriba hacia abajo
- Cuando escribe un código, utilice paréntesis
- Cuando escribe un código, use las expresiones matemáticas más simples que le sea posible para que sean fáciles de entender
- Divida las series de operaciones matemáticas largas para que sean más claras



# Operadores aritméticos

## División de Números enteros:

- La división de números enteros arroja un resultado con punto flotante.

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(10.0 / 2.0)
5.0
>>> print(99.0 / 100.0)
0.99
```



# Operadores de asignación

Operador	Ejemplo	Equivalencia
<code>+=</code>	<code>x += 2</code>	<code>x = x + 2</code>
<code>-=</code>	<code>x -= 2</code>	<code>x = x - 2</code>
<code>*=</code>	<code>x *= 2</code>	<code>x = x * 2</code>
<code>/=</code>	<code>x /= 2</code>	<code>x = x / 2</code>
<code>%=</code>	<code>x %= 2</code>	<code>x = x % 2</code>
<code>//=</code>	<code>x //= 2</code>	<code>x = x // 2</code>

Operador	Ejemplo	Equivalencia
<code>**=</code>	<code>x **= 2</code>	<code>x = x ** 2</code>
<code>&amp;=</code>	<code>x &amp;= 2</code>	<code>x = x &amp; 2</code>
<code> =</code>	<code>x  = 2</code>	<code>x = x   2</code>
<code>^=</code>	<code>x ^= 2</code>	<code>x = x ^ 2</code>
<code>&gt;&gt;=</code>	<code>x &gt;&gt;= 2</code>	<code>x = x &gt;&gt; 2</code>
<code>&lt;&lt;=</code>	<code>x &lt;&lt;= 2</code>	<code>x = x &lt;&lt; 2</code>





# Operadores de asignación

```
a=7; b=2
print("Operadores de asignación")
x=a; x+=b; print("x+=", x) # 9
x=a; x-=b; print("x-=", x) # 5
x=a; x*=b; print("x*=", x) # 14
x=a; x/=b; print("x/=", x) # 3.5
x=a; x%=b; print("x%=", x) # 1
x=a; x//=b; print("x//=", x) # 3
x=a; x**=b; print("x**=", x) # 49
x=a; x&=b; print("x&=", x) # 2
x=a; x|=b; print("x|=", x) # 7
x=a; x^=b; print("x^=", x) # 5
x=a; x>>=b; print("x>>=", x) # 1
x=a; x<<=b; print("x<<=", x) # 28
```



# Operadores de identidad

Los operadores de identidad se utilizan para comprobar si dos variables son, o no, el mismo objeto

Operador	Descripción
is	Devuelve True si ambos operandos hacen referencia al mismo objeto; False en caso contrario.
is not	Devuelve True si ambos operandos no hacen referencia al mismo objeto; False en caso contrario.

**Recuerda:** Para conocer la identidad de un objeto se usa la función `id()`.



# Operadores de pertenencia

Los operadores de pertenencia se utilizan para comprobar si un valor o variable se encuentran en una secuencia (list, tuple, dict, set o str).

Operador	Descripción
in	Devuelve True si el valor se encuentra en una secuencia; False en caso contrario.
not in	Devuelve True si el valor no se encuentra en una secuencia; False en caso contrario.



# Ejercicios

- Crea dos variables nombre y apellido y asígnales tus datos, concatena las dos variables, e imprime en pantalla el resultado.
- Crea una variable, asígnale un texto, e imprime en pantalla la longitud del texto.
- Crea una variable, asígnale un texto, e imprime en pantalla: El primer caracter, los tres primeros, y los dos últimos.
- Crea una variable, asígnale el texto "Bienvenidos al curso de Python", busca el texto "curso". Imprime el resultado.
- Crea una variable, asígnale el texto "Hoy es lunes", luego reemplaza "lunes" por "jueves". Imprime el resultado.



# Ejercicios

- Calcule la raíz cuadrada de 82.
- Escriba una función para las operaciones, sumar, restar, multiplicar, dividir. Escriba e imprima el llamado a cada una de estas funciones.
- Calcule e imprima las siguientes potencias:  $2^5$ ,  $10^3$ ,  $14^2$ .
- Pida un número por pantalla e imprima su raíz cuadrada.
- Ejecute:  $(1 == 1)$  and  $(2 > 1)$ , qué resultado obtiene?
- Ejecute:  $(0 != 0)$  or  $(10 > 20)$ , qué resultado obtiene?



# Ejercicios de funciones

- Escribe una función llamada `es_par()` que tome un número como argumento y devuelva `True` si es par y `False` si es impar.
- Escribe una función llamada `convertir_texto(cadena, a_mayusculas)` que devuelva una cadena convertida a letras mayúsculas o minúsculas dependiendo del valor del parámetro `a_mayusculas`.
- Escribe una función llamada `invertir_cadena()` que tome una cadena como parámetro y devuelva la cadena invertida.
- Escribe una función llamada `calcular_area_rectangulo()` que tome la longitud y el ancho de un rectángulo como argumentos y devuelva el área del rectángulo.
- Escribe una función llamada `imprimir_pares()` que tome un número entero como argumento y imprima todos los números pares desde 1 hasta ese número.
- Escribe una función llamada `convertir_fahrenheit_a_celsius()` que tome una temperatura en grados Fahrenheit como argumento y devuelva la temperatura equivalente en grados Celsius.
- Escriba una función que reciba un carácter (`c`) y una longitud (`n`), y devuelva una cadena de texto de largo (`n`).



# Ayuda de ejercicios funciones

- `es_par`: Un numero es par si al dividirlo por dos, la división es exacta..
- `area_rectangulo()`: Multiplicamos el largo por el ancho
- `fahrenheit_a_celsius()` :  $\text{celsius} = (\text{Fahrenheit} - 32) \times 5/9$ , o también,  $\text{celsius} = (\text{Fahrenheit} - 32) / 1.8$



# Ejercicios con Strings

Dado el siguiente texto: "Un programa de computador siempre hará lo que le ordenes que haga, no lo que quieres que haga". Realice los siguientes operaciones:

- Imprime los primeros tres caracteres.
- Imprime los últimos cinco caracteres.
- Imprime los caracteres en las posiciones pares.
- Imprime los caracteres en las posiciones impares.
- Imprime los caracteres desde la tercera posición hasta la sexta posición.
- Imprime los últimos cuatro caracteres utilizando un índice negativo.
- Imprime los caracteres en las posiciones múltiplos de 3.
- Imprime los caracteres en reversa.
- Imprime los caracteres en las posiciones impares en reversa.
- Imprime una subcadena formada por los caracteres desde la segunda posición hasta la penúltima posición.





# Ejercicios con operadores matematicos

- Escriba una función que devuelva la suma de tres números.
- Escriba una función que devuelva resta de cuatro números, dejando valor por defecto de 2 y 0 para el tercer y cuarto numero respectivamente.
- Escriba una función que devuelva la multiplicación de dos números.
- Escriba una función que devuelva la división de dos números, y diga si es exacta.
- Escriba una función que devuelva el módulo o resto de una división.
- Escriba una función que devuelva el resultado de elevar un número a una potencia .
- Escriba una función que devuelva solo la parte entera (con tipo de dato int), de la división de dos números.



# Nuestro primer programa

Programa de cálculo de descuento en una tienda.

- Capturamos el precio del producto
- Capturamos el descuento.
- Calculamos el precio final, a partir de los datos ingresados.



# Nuestro primer programa

```
# Programa de cálculo de descuento en una tienda.
# Capturamos el precio del producto y su descuento.
# Calculamos el precio final a partir de los datos ingresados.

# Función para calcular el descuento
def calcular_descuento(precio, porcentaje_descuento):
    descuento = precio * (porcentaje_descuento / 100)
    precio_final = precio - descuento
    return precio_final

# Función para mostrar el resultado
def mostrar_resultado(precio_inicial, porcentaje_descuento, precio_final):
    print("Precio inicial: $", precio_inicial)
    print("Porcentaje de descuento: ", porcentaje_descuento, "%")
    print("Precio final: $", precio_final)

# Solicitar el precio del producto al usuario
precio_producto = float(input("Ingrese el precio del producto: "))

# Solicitar el porcentaje de descuento al usuario
porcentaje_descuento = int(input("Ingrese el porcentaje de descuento: "))

# Calcular el precio final llamando a la función calcular_descuento
precio_final = calcular_descuento(precio_producto, porcentaje_descuento)

# Mostrar el resultado llamando a la función mostrar_resultado
mostrar_resultado(precio_producto, porcentaje_descuento, precio_final)
```



# Referencias bibliográficas

Documentación oficial Python.

<https://docs.python.org/es/3/tutorial/index.html>

<https://docs.python.org/es/3/library/stdtypes.html#printf-style-string-formatting>

[https://docs.python.org/es/3/reference/lexical\\_analysis.html#f-strings](https://docs.python.org/es/3/reference/lexical_analysis.html#f-strings)

<https://docs.python.org/es/3/library/stdtypes.html#str.format>

