

#

DESARROLLO WEB FULLSTACK

con Python y JavaScript

polotic
misiones



SECUENCIAS DE DATOS



Una de las partes más poderosas del lenguaje Python es su capacidad para trabajar con secuencias de datos además de variables individuales.

list – Lista, o secuencia de valores cambiantes

tuple – Tupla, o secuencia de valores imutable

set – Colección de valores unicos

dict – Diccionario, o colección de pares de clave-valor

CADENAS

- Ordenable: Sí
- Mutable: No

Ya hemos analizado un poco los strings, pero en lugar de solo variables, podemos pensar en un string como una secuencia de caracteres. ¡Esto significa que podemos acceder a elementos individuales dentro de la cadena! Por ejemplo:

```
nombre = "Harry"  
print(nombre[0])  
print(nombre[1])
```

imprime el primer carácter (o índice-0) de la cadena, que en este caso resulta ser **H**, y luego imprime el segundo carácter (o índice-1), que es una **a**.



LISTAS


- Ordenable: Sí
- Mutable: Si

Una lista de Python le permite almacenar cualquier tipo de variable. Creamos una lista usando corchetes y comas, como se muestra a continuación.

De manera similar a los strings, podemos imprimir una lista completa o algunos elementos individuales. También podemos agregar elementos a una lista usando **append** y ordenar una lista usando **sort**

LISTAS

```
# Este es un comentario
nombres = ["Harry", "Ron", "Hermione"]
# Imprimir toda la lista:
print(nombres)
# Imprimir el segundo elemento de la lista:
print(nombres[1])
# Agregar un nombre a la lista:
nombres.append("Draco")
# Ordenar la lista:
nombres.sort()
# Imprimir la nueva lista:
print(nombres)
```

 Administrador: Windows PowerShell

```
PS P:\Cursos Polo\Python con Javascript\Codigo Fuente> py .\miLista.py
['Harry', 'Ron', 'Hermione']
Ron
['Draco', 'Harry', 'Hermione', 'Ron']
```

TUPLAS

- Ordenable: Sí
- Mutable: Si

Las tuplas se utilizan generalmente cuando necesita almacenar solo dos o tres valores juntos, como los valores x e y para un punto en un plano.

En el código Python, usamos paréntesis:

```
punto = (12.5, 10.6)
```

SETS

- Ordenable: No
 - Mutable: No Aplica
-
- Los conjuntos o sets se diferencian de las listas y tuplas en que no están ordenados.
 - También son diferentes porque, si bien puedes tener dos o más elementos iguales dentro de una lista / tupla, un set solo almacenará cada valor una vez.
 - Podemos definir un set vacío usando la función **set**. Luego podemos usar agregar y quitar para agregar y quitar elementos de ese conjunto, y la función **len** para encontrar el tamaño del conjunto.
 - Ten en cuenta que la función **len** funciona en todas las secuencias en Python. También ten en cuenta que a pesar de agregar 4 y 3 al conjunto dos veces, cada elemento solo puede aparecer una vez en un conjunto.

SETS



```
# Crear un conjunto (set) vacío:  
conjunto = set()
```

```
# Agregar elementos:  
conjunto.add(1)  
conjunto.add(2)  
conjunto.add(3)  
conjunto.add(4)  
conjunto.add(3)  
conjunto.add(1)
```

```
# Remover 2 elementos del set  
conjunto.remove(2)
```

```
# Imprimir el set s:  
print(conjunto)
```

```
# Encontrar el tamaño del set  
print(f"El set tiene {len(s)} elementos.")
```

```
""" Este es un comentario multilínea de Python:  
La salida será:  
{1, 3, 4}  
Este set tiene 3 elementos.  
"""
```

```
Administrador: Windows PowerShell  
PS P:\Cursos Polo\Python con Javascript\Codigo Fuente> py .\miSet.py  
{1, 3, 4}  
El set tiene 3 elementos.
```


DICCIONARIOS

- Ordenable: No
- Mutable: Si
- Los diccionarios o `dict` de Python, serán especialmente útiles en este curso.
- Un diccionario es un conjunto de pares clave-valor (key-value), donde cada clave tiene un valor correspondiente, al igual que en un diccionario, cada palabra (la clave) tiene una definición correspondiente (el valor).
- En Python, usamos llaves para contener un diccionario y dos puntos para indicar claves y valores. Por ejemplo:

```
# Definir un diccionario
casas = {"Harry": "Gryffindor", "Draco": "Slytherin"}
# Imprimir la casa de Harry
print(casas["Harry"])
# Agregar valores a un diccionario:
casas["Hermione"] = "Gryffindor"
# Imprimir la casa de Hermione:
print(casas["Hermione"])

""" Salida esperada:
Gryffindor
Gryffindor
"""
```



ESTRUCTURAS DE CONTROL Y TOMA DE DECISIONES



HACIÉNDONOS ENTENDER



El método principal de Python para hacer comparaciones es mediante el uso de **operadores**. De hecho, los operadores también juegan un papel importante en la manipulación de datos.

Utilizas operadores para definir *cómo* se compara un dato con otro y para *modificar* la información dentro de una sola variable.

Cuando utilizas un operador, debes proporcionar una **variable** o una **expresión**. Una expresión es una ecuación o fórmula que proporciona una descripción de un concepto matemático. En la mayoría de los casos, el resultado de evaluar una expresión es un valor Boolean (True o False).



TIPOS DE OPERADORES



Un *operador* acepta uno o mas **inputs** en forma de variables o expresiones, realiza una tarea (como una comparación o suma) y luego proporciona un **output** consistente con esa tarea.

Los operadores se clasifican parcialmente por su efecto y parcialmente por el número de elementos que requieren.

Por ejemplo, un operador **unario** trabaja con una sola variable o expresión; un operador **binario** requiere dos.

Los elementos proporcionados como entrada a un operador se denominan **operandos**. El operando en el lado izquierdo del operador se llama operando izquierdo, mientras que el operando en el lado derecho del operador se llama operando derecho.

TIPOS DE OPERADORES

La siguiente lista muestra las categorías de operadores que usa en Python:

- ✓ Unarios
- ✓ Aritmeticos
- ✓ Relacionales
- ✓ Lógicos
- ✓ Bitwise
- ✓ Asignación
- ✓ Membresía
- ✓ Identidad

Cada una de estas categorías realiza una tarea específica. Por ejemplo, los operadores aritméticos realizan tareas basadas en matemáticas, mientras que los operadores relacionales realizan comparaciones. Las siguientes secciones describen los operadores según la categoría en la que aparecen.



OPERADOR UNARIO



Los operadores unarios requieren una sola variable o expresión como entrada. A menudo, utilizarás estos operadores como parte de un proceso de toma de decisiones. Por ejemplo, es posible que desees buscar algo que no se parezca a otra cosa.

OPERADOR	DESCRIPCION	EJEMPLO
~	Invierte los bits en un número para que todos los 0 bits se conviertan en 1 bits y viceversa.	~ 4 resulta en un valor de -5
-	Niega el valor original para que lo positivo se vuelva negativo y viceversa.	- (- 4) resulta en 4 y -4 resulta en -4
+	Se proporciona únicamente para que esté completo. Este operador regresael mismo valor que proporciona como entrada.	+4 da como resultado un valor de 4

OPERADOR ARITMÉTICO



Las computadoras son conocidas por su capacidad para realizar operaciones matemáticas complejas. Sin embargo, las tareas complejas que realizan las computadoras a menudo se basan en tareas matemáticas mucho más simples, como la suma.

Operador	Descripcion	Ejemplo
+	Suma dos valores juntos	$5 + 2 = 7$
-	Resta el operando derecho del operando izquierdo	$5 - 2 = 3$
*	Multiplica el operando derecho por el operando izquierdo	$5 * 2 = 10$
/	Divide el operando izquierdo por el operando derecho	$5/2 = 2,5$
%	Divide el operando izquierdo por el operando derecho y devuelve el resto	$5\% 2 = 1$
**	Calcula el valor exponencial del operando derecho por el operando izquierdo	$5 ** 2 = 25$
//	Realiza la división de enteros, en la que el operando de la izquierda se divide por el operando de la derecha y solo se devuelve el número entero (también llamado división de piso)	$5 // 2 = 2$

OPERADOR RELACIONAL



Los operadores relacionales comparan un valor con otro y le dicen cuándo la relación que ha proporcionado es verdadera. Por ejemplo, 1 es menor que 2, pero 1 nunca es mayor que 2. El valor de verdad de las relaciones se usa a menudo para tomar decisiones en sus aplicaciones para garantizar que se cumpla la condición para realizar una tarea específica.

Operador	Descripcion	Ejemplo
==	Determina si dos valores son iguales. Observa que el operador relacional usa dos signos iguales. Un error que cometen muchos programadores es usar solo un signo igual, lo que da como resultado que un valor se asigne a otro.	1==2 es False
!=	Determina si dos valores no son iguales. Algunas versiones anteriores de Python le permitían usar el operador <> en lugar del operador !=. El uso del operador <> da como resultado un error en las versiones actuales de Python.	1!=2 es True
>	Verifica que el valor del operando izquierdo sea mayor que el valor del operando derecho.	1>2 es False
<	Verifica que el valor del operando izquierdo sea menor que el valor del operando derecho.	1<2 es True
>=	Verifica que el valor del operando izquierdo es mayor que o	1>=2 es False
<=	Verifica que el valor del operando izquierdo sea menor o igual que el valor del operando derecho.	1<=2 es True



OPERADOR LÓGICO



Los operadores lógicos combinan el valor verdadero o falso de variables o expresiones para que pueda determinar su valor de verdad resultante. Utiliza los operadores lógicos para crear expresiones booleanas que ayudan a determinar si se deben realizar las tareas.

Operador	Descripcion	Ejemplo
and	Determina si ambos operandos son verdaderos.	True and True es True True and False es False False and True es False False and False es False
or	Determina cuando uno de los dos operandos es verdadero.	True or True es True True or False es True False or True es True False or False es False
not	Niega el valor de verdad de un solo operando. Un valor verdadero se vuelve falso y un valor falso se vuelve verdadero.	not True es False not False es True



OPERADOR BITWISE

Los operadores bitwise (o bit a bit) interactúan con los bits individuales en un número. Por ejemplo, el número 6 es en realidad 0b0110 en binario. Un operador bit a bit interactuaría con cada bit dentro del número de una manera específica. Cuando se trabaja con un operador lógico bit a bit, un valor de 0 cuenta como False y un valor de 1 cuenta como True

Operador	Descripcion	Ejemplo
& (And)	Determina si ambos bits individuales dentro de dos operadores son verdaderos y establece el bit resultante en verdadero cuando lo son.	0b1100 & 0b0110= 0b0100
(Or)	Determina si alguno de los bits individuales dentro de dos operadores es verdadero y establece el bit resultante en verdadero cuando uno de ellos lo es.	0b1100 0b0110 =0b1110
^ (Or excluyente)	Determina si solo uno de los bits individuales dentro de dos operadores es verdadero y establece el bit resultante en verdadero cuando uno lo es. Cuando ambos bits son verdaderos o ambos bits son falsos, el resultado es falso.	0b1100 ^ 0b0110 = 0b1010
~ (Complemento de uno)	Calcula el valor de complemento a uno de un número.	~0b1100 = ~0b1101 ~0b0110 = ~0b0111
<< (shift a la izquierda)	Desplaza los bits del operando izquierdo a la izquierda por el valor del operando derecho. Todos los bits nuevos se establecen en 0 y todos los bits que fluyen al final se pierden.	0b00110011 << 2 =0b11001100
>> (Shift a la derecha)	Desplaza los bits del operando izquierdo a la derecha por el valor del operando derecho. Todos los bits nuevos se establecen en 0 y todos los bits que fluyen al final se pierden.	0b00110011 >> 2 =0b00001100



OPERADOR DE ASIGNACIÓN



Los operadores de asignación colocan datos dentro de una variable. El operador de asignación simple aparece en capítulos anteriores del libro, pero Python ofrece una serie de operadores de asignación interesantes que puede utilizar. Estos otros operadores de asignación pueden realizar tareas matemáticas durante el proceso de asignación, lo que hace posible combinar la asignación con una operación matemática.

OPERADOR DE ASIGNACIÓN

Operador	Descripcion	Ejemplo
=	Asigna el valor encontrado en la operación de la derecha ya la izquierda operando.	MiVar = 2 resulta en MiVar conteniendo 2
+=	Agrega el valor encontrado en el operando derecho al valor encontrado en el operando izquierdo y coloca el resultado en el operando izquierdo.	MiVar += 2 resulta en MiVar conteniendo 7
-=	Resta el valor encontrado en el operando derecho del valor encontrado en el operando izquierdo y coloca el resultado en el operando izquierdo.	MiVar = 2 resulta en MiVar conteniendo 3
*=	Multiplica el valor encontrado en el operando derecho por el valor encontrado en el operando izquierdo y coloca el resultado en el operando izquierdo.	MiVar *= 2 resulta en MiVar conteniendo 10
/=	Divide el valor encontrado en el operando izquierdo por el valor encontrado en el operando derecho y coloca el resultado en el operando izquierdo.	MiVar /= 2 resulta en MiVar conteniendo 2.5
%=	Divide el valor encontrado en el operando izquierdo por el valor encontrado en el operando derecho y coloca el resto en el operando izquierdo.	MiVar %= 2 resulta en MiVar conteniendo 1
**=	Determina el valor exponencial encontrado en el operando izquierdo cuando se eleva a la potencia del valor encontrado en el operando derecho y coloca el resultado en el operando izquierdo.	MiVar **= 2 resulta en MiVar conteniendo 25
//=	Divide el valor encontrado en el operando izquierdo por el valor encontrado en el operando derecho y coloca el resultado entero (número entero) en el operando izquierdo.	MiVar //= 2 resulta en MiVar conteniendo 2



OPERADOR DE MEMBRESÍA



Los operadores de membresía detectan la aparición de un valor dentro de una lista o secuencia y luego generan el valor de verdad de esa apariencia. Piense en los operadores de membresía como si fuera una rutina de búsqueda de una base de datos. Ingresa un valor que cree que debería aparecer en la base de datos y la rutina de búsqueda lo encuentra o informa que el valor no existe en la base de datos.

Operador	Descripcion	Ejemplo
In	Determina si el valor del operando izquierdo aparece en la secuencia que se encuentra en el operando derecho.	“Hola” in “Hola Adios” es True
not in	Determina si el valor del operando izquierdo falta en la secuencia encontrada en el operando derecho.	“Hola” not in “Hola Adios” es False



OPERADOR DE IDENTIDAD



Los operadores de identidad determinan si un valor o expresión es de cierta clase o tipo. Utiliza operadores de identidad para asegurarse de que realmente está trabajando con el tipo de información que cree que es. El uso de operadores de identidad puede ayudarlo a evitar errores en su aplicación o determinar el tipo de procesamiento que requiere un valor.

Operador	Descripcion	Ejemplo
Is	Se evalúa como verdadero cuando el tipo de valor o expresión en el operando derecho apunta al mismo tipo en el operando izquierdo.	<code>type(2) is int</code> es <code>True</code>
Is not	Se evalúa como verdadero cuando el tipo de valor o expresión en el operando derecho apunta a un tipo diferente al valor o expresión en el extremo izquierdo.	<code>type(2) is not int</code> es <code>False</code>



Orden de precedencia de operadores



Operador	Descripcion
()	Utiliza paréntesis para agrupar expresiones y anular la precedencia predeterminada para poder forzar una operación de menor precedencia (como la suma) para que tenga prioridad sobre una operación de mayor precedencia (como la multiplicación).
**	La exponenciación eleva el valor del operando izquierdo a la potencia del operando derecho.
~ + -	Los operadores unarios interactúan con una sola variable o expresión.
* / % //	Multiplica, divide, módulo y división de piso.
+ -	Adición y sustracción.
>> <<	Desplazamiento bit a bit a derecha e izquierda.
&	AND bit a bit.
^	OR exclusivo bit a bit y OR estándar.

Orden de precedencia de operadores

Operador	Descripcion
<= < > >=	Operadores de comparacion
== !=	Operadores de igualdad
= %= /= //=- += *= **=	Operadores de asignación
Is is not	Operadores de identidad
In not in	Operadores de membresía
not or and	Operadores Lógicos

Control de Flujo: IF

```
x = int(input("Por favor ingrese un numero#:"))
```

```
if x < 0:
```

```
    x = 0
```

```
    print 'Negativo cambiado a cero'
```

```
elif x == 0:
```

```
    print 'Cero'
```

```
elif x == 1:
```

```
    print 'El numero es 1'
```

```
else:
```

```
    print 'Mas de uno'
```

Condicional

Defino otra condición si no se cumple la primera

Todo aquello que se cumpla por defecto si no se cumple lo anterior

Entendiendo el único operador ternario de Python



Un operador ternario requiere tres elementos.

Python admite solo uno de esos operadores y lo usa para determinar el valor de verdad de una expresión.

Este operador toma la siguiente forma:

```
ValorCuandoTrue if Expresion else  
ValorCuandoFalse
```

Cuando la expresión es **verdadera**, el operador genera ValorCuandoTrue. Cuando la expresión es **falsa**, genera ValorCuandoFalse. Un ejemplo sería:

```
“Hola” if True else  
“Adios”
```



Otra manera de hacerlo...



Una de las ventajas de usar Python es que normalmente tiene más de una forma de hacer las cosas. Python tiene una forma alternativa de este operador ternario: un atajo aún más corto. Toma la siguiente forma:

```
(ValorCuandoFalse, ValorCuandoTrue)  
[Expresion]
```

Como vimos antes, cuando `Expresion` es verdadera, el operador genera `ValorCuandoTrue`; de lo contrario, genera `ValorCuandoFalse`. Observe que los elementos `ValorCuandoTrue` y `ValorCuandoFalse` se invierten en este caso. Un ejemplo de esta versión es:

```
("Hello", "Goodbye")[True]
```

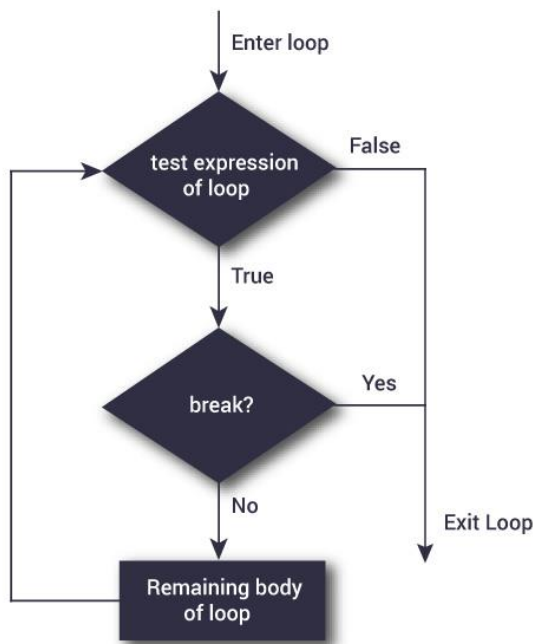
Control de Flujo: FOR

```
hogwarts = ['Harry', 'Ron', 'Hermione']  
  
for estudiante in hogwarts:  
    print estudiante, len(estudiante)
```

Accediendo mediante indices:

```
for i in range(len(hogwarts)):  
    print i, hogwarts[i]
```

Bucles: break, continue, else



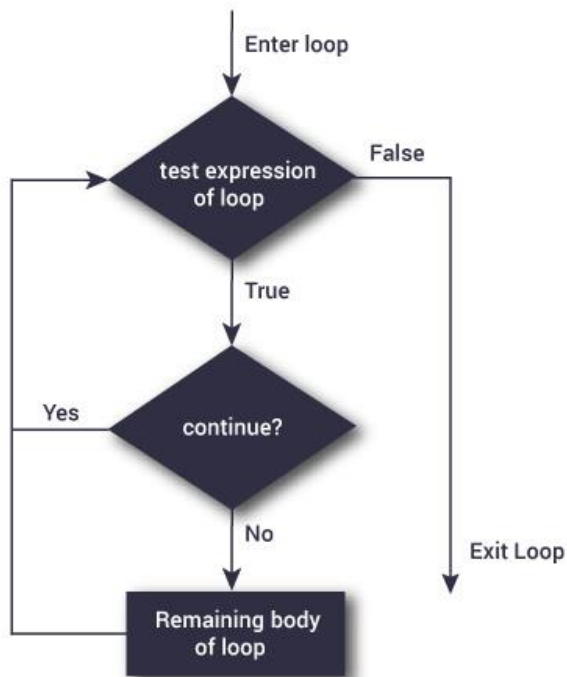
break

La sentencia break termina el bucle que la contiene. El control del programa fluye a la declaración inmediatamente después del cuerpo del bucle. Si la instrucción break está dentro de un bucle anidado (bucle dentro de otro bucle), la declaración break terminará el bucle más interno.

```
for val in "Harry":  
    if val == "a":  
        break  
    print(val)
```

```
print("Fin")
```

Bucles: break, continue, else



continue

La instrucción continue se usa para omitir el resto del código dentro de un bucle solo para la iteración actual. El bucle no termina sino que continúa con la siguiente iteración.

```
for val in "Harry":  
    if val == "a":  
        continue  
    print(val)  
  
print("Fin")
```



Bucles: break, continue, else

else

Luego del agotamiento del bucle puedo ejecutar código adicional:

```
for n in range(2,10):  
    for x in range(2,n):  
        if n % x == 0:  
            print n, ' igual a ', x, '*', n/x  
            break  
    else:  
        # el loop finalizó sin encontrar un factor  
        print n, ' es primo'
```



¿Cómo hacer nada?



pass

Pass es un relleno sintáctico que simplemente sirve para determinar que algo “no hace nada”.

```
while 1:  
    pass
```


Ingreso de Datos

```
nombre = input("Nombre: ")  
print("Hola, " + nombre)
```



Formateo de la salida de datos

```
print(f"Hola, {input(`Nombre: ``)}")
```

Errores en Python



Una **excepción** es lo que sucede cuando ocurre un **error** mientras estamos ejecutando nuestro código Python y, con el tiempo, mejorarás cada vez más en la interpretación de estos errores, lo cual es una habilidad muy valiosa.

Veamos un ejemplo. En la imagen se ve que ocurrió un **TypeError**, lo que generalmente significa que Python *esperaba que una determinada variable fuera de un tipo*, pero descubrió que era de otro. tipo. En este caso, la excepción nos dice que *no podemos usar el símbolo > para comparar str e int*, y luego arriba podemos ver que esta comparación ocurre en la línea 2.

En este caso, es obvio que -2 es un número entero negativo, por lo que debe ser el caso de que nuestra variable **num** sea un **string**. Esto sucede porque resulta que la función **input** siempre devuelve un **string**, y tenemos que especificar que se debe convertir (o transformar) en un número entero usando la función **int**.

```
Administrador: Windows PowerShell
PS P:\Cursos Polo\Python con Javascript\Codigo Fuente> py .\miAplicacion.py
Numero: -2
Traceback (most recent call last):
  File ".\miAplicacion.py", line 2, in <module>
    if num > 0:
TypeError: '>' not supported between instances of 'str' and 'int'
PS P:\Cursos Polo\Python con Javascript\Codigo Fuente> █
```

Funciones en Python



- Ya hemos visto algunas funciones de Python como `print` e `input`, pero ahora vamos a sumergirnos en escribir nuestras propias funciones.
- Para comenzar, escribiremos una función que toma un número y lo eleva al cuadrado:

```
def cuadrado(numero):  
    return numero * numero
```

- Observe cómo usamos la palabra clave `def` para indicar que estamos definiendo una función, que estamos tomando una única entrada llamada `numero` y que usamos la palabra clave `return` para indicar cuál debería ser la salida de la función.
- Entonces podemos "llamar" a esta función tal como hemos llamado a otras: usando paréntesis:

```
for i in range(10):  
    print(f"El cuadrado de {i} es {cuadrado(i)}")
```

Modulos en Python



Si bien ya tengo definido mi archivo `misFunciones.py`, voy a poner parte del código en otro archivo llamado `miScript.py` de manera que ambos queden así:

```
def cuadrado(numero):  
    return numero * numero
```

misFunciones.py

```
for i in range(10):  
    print(f"El cuadrado de {i} es {cuadrado(i)}")
```

miScript.py

Si intentamos ejecutar `miScript.py` nos saldrá el siguiente mensaje:

```
Traceback (most recent call last):  
  File ".\miScript.py", line 2, in <module>  
    print(f"El cuadrado de {i} es {cuadrado(i)}")  
NameError: name 'cuadrado' is not defined
```

Modulos en Python

Nos encontramos con este problema porque, de forma predeterminada, los archivos de Python no se conocen entre sí, por lo que tenemos que importar explícitamente la función cuadrado del módulo de funciones que acabamos de escribir en `miScript.py`

```
def cuadrado(numero):  
    return numero * numero
```

`misFunciones.py`

```
from misFunciones import cuadrado  
  
for i in range(10):  
    print(f"El cuadrado de {i} es {cuadrado(i)}")
```

`miScript.py`

Alternativamente, podemos optar por importar todo el módulo de funciones y luego usar la notación de puntos para acceder a la función cuadrada:

```
import misFunciones  
for i in range(10):  
    print(f"El cuadrado de {i} es {misFunciones.cuadrado(i)}")
```



TRABAJEMOS JUNTOS



1. Escribe un programa Python que imprima “Hola Mundo”, si **a** es mayor que **b**.
2. Escribe un programa Python que acepte 5 números decimales del usuario.
3. Escribe un programa en Python que reciba 5 números enteros del usuario y los guarde en una lista.
Luego recorrer la lista y mostrar los números por pantalla.
4. Dada una lista (`lista1 = [12, 15, 32, 42, 55, 75, 122, 132, 150, 180, 200]`), iterarla y mostrar números divisibles por cinco, y si encuentra un número mayor que 150, detenga la iteración del bucle.



#

¡HASTA LA
próxima!

www.polotic.misiones.gob.ar

[f](#) [@](#) [v](#) [d](#) /poloticmisiones

polotic
misiones



Gobierno
de Misiones

