

{desafío}
latam_

Ciclos y métodos _

Sesión Presencial 2



Itinerario

Activación de
conceptos

Desarrollo Desafío

Panel de discusión

Activación de conceptos

¿Cuál de las siguientes afirmaciones es verdadera?

1. En el ciclo while el iterador aumenta su valor automáticamente
2. En el ciclo while la variable que evalúa la condición se define junto con el ciclo
3. En el ciclo while siempre se sabrá cuántas iteraciones hará el ciclo
4. En el ciclo while se debe manejar la condición de salida

¿Cuál es la ventaja del ciclo for?

1. Se debe manejar obligatoriamente la condición de salida
2. Se debe incrementar el valor del iterador
3. Se debe definir el iterador antes de declarar el ciclo
4. No se puede modificar el iterador

¿Cuál de las siguientes situaciones son posibles de solucionar con ciclos?

1. Realizar una sumatoria de números
2. Dibujar una lista anidada de HTML
3. Dibujar patrones
4. Dibujar y calcular las tablas de multiplicar
5. Todas son correctas

¿Por qué no es bueno anidar ciclos excesivamente?

1. Porque aumenta la complejidad del programa
2. Porque se generan errores de sintaxis
3. Porque después de más de 3 ciclos estos son omitidos
4. Porque no se puede incluir bloques if en ellos

Llamar una función

- Opción 1: **función()** → **print()**
- Opción 2: **objeto.función()** → **"gato".upper()**
- Opción 3: **Clase.función()** → **LinearRegression.fit()**
- Opción 4: **módulo.función()** → **math.sqrt()**

Definiendo una función

```
# Definimos con def y un nombre.  
def nombre_del_metodo(): # Se agrega () al final del nombre, y luego :  
    # Serie de instrucciones que ejecutará el método.  
    # (indentado por 4 espacios)  
    # ...  
    # ...
```

Definir y Llamar una función

```
1 def imprimir_menu():
2     print('Menú: Escoja una acción')
3     print('-' * 20)
4     print('1) Acción 1')
5     print('2) Acción 2')
6     print('Escribe "Salir" para terminar el programa')
7     print()
8
9 opt_menu = 'cualquier valor'
10
11 while opt_menu != 'salir' and opt_menu != 'Salir':
12     imprimir_menu() # Se llama a la función
13
14     opt_menu = input()
15
16     if opt_menu == '1':
17         print('Realizando acción 1')
18     elif opt_menu == '2':
19         print('Realizando acción 2')
20     elif opt_menu == 'salir' or opt_menu == 'Salir':
21         print('Saliendo')
22     else:
23         print('Opción inválida')
24     print()
```


Parametrizando funciones

```
1 def incrementar_en_uno(numero):  
2     total = numero + 1  
3     print("El resultado es: ", total)
```

: 1 incrementar_en_uno(10)

El resultado es: 11

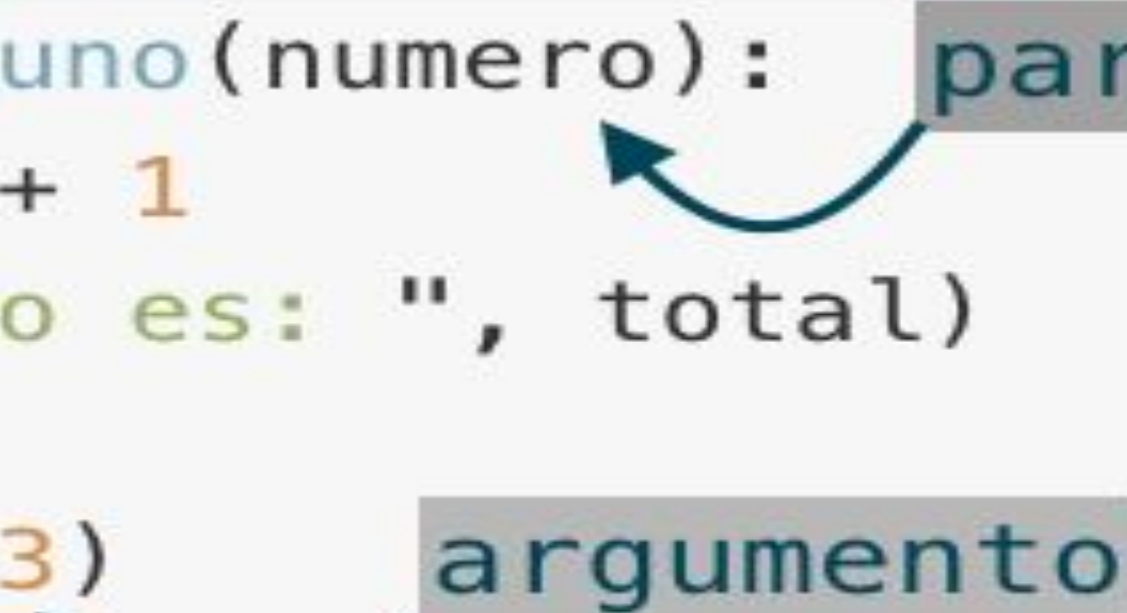
```
1 incrementar_en_uno()
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-12-ea1a5f1be123> in <module>()  
----> 1 incrementar_en_uno()
```

```
TypeError: incrementar_en_uno() missing 1 required positional argument: 'numero'
```

Parámetro v/s Argumento

```
def incrementar_en_uno(numero):  
    total = numero + 1  
    print("El número es: ", total)  
  
incrementar_en_uno(3)
```



The diagram illustrates the relationship between a function parameter and its argument. A curved arrow points from the word 'parámetro' (parameter) to the variable 'numero' in the function definition. Another curved arrow points from the word 'argumento' (argument) to the value '3' in the function call.

Parámetros opcionales

```
def incrementar_por_cantidad(numero, cantidad = 2):  
    total = numero + cantidad  
    print("El número es: ", total)
```

```
incrementar_por_cantidad(5)      # 7  
incrementar_por_cantidad(5, 1)  # 6
```



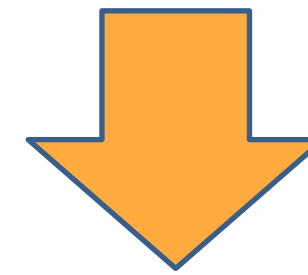
Un valor opcional
tiene un valor asignado
por defecto

Retorno

```
def transformar_a_fahrenheit(f):  
    celsius = (f + 40) / 1.8 - 40  
    return celsius  
  
print(transformar_a_fahrenheit(110))
```

1) 110

2) 43.3



```
1 grados_celsius = transformar_de_fahrenheit(110)  
2  
3 print("La variable grados_celsius es del tipo: ", type(grados_celsius))  
4 print(grados_celsius)
```

La variable grados_celsius es del tipo: <class 'float'>
43.33333333333333

Retorno

```
1 def prueba_return():  
2     a = "Esta línea se va a imprimir"  
3     b = "Esta línea no se va a imprimir"  
4     return a # Punto de salida  
5     print(b)  
6  
7 prueba_return()
```

'Esta línea se va a imprimir'

Tipos de variable

- Globales
- Locales
- De instancia
- De clase

Alcance de una variable local

```
def aprobado(promedio, nota_aprobacion = 4):  
    if promedio >= nota_aprobacion:  
        status = True  
    else:  
        status = False  
    return status  
  
print(status)
```

Scope de la función aprobado
Aquí existe status

Espacio principal __main__
Aquí NO existe status



```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-19-48c6094df5ce> in <module>()  
      6     return status  
      7  
----> 8 print(status)  
  
NameError: name 'status' is not defined
```

Alcance de una variable global

```
1 continent = 'South America'
2
3 def print_continent():
4     print(continent)
5
6 print_continent()
```

South America

Uso conjunto de variables locales y globales

```
1 # estas variables se definen dentro del ambiente __main__
2 name = 'Alan Turing'
3 age = 41
4
5 def any_function():
6     # Esta variable está siendo definida en un ambiente nuevo: el de any_function
7     birthplace = 'Londres'
8
9     # Como age es una variable global, se puede acceder desde este scope
10    # Y también se puede acceder a birthplace, porque aunque sea local, se está en su mismo scope
11    print("Edad de", age, "años. Residencia: ", birthplace)
12
13    # Acá se vuelve al __main__ (se elimina la indentación de 4 espacios)
14    # La variable birthplace no existe en este espacio.
15
16    print("Nombre: ", name)
17    any_function()
18
```

Nombre: Alan Turing
Edad de 41 años. Residencia: Londres

/* Desafío */

Panel de discusión

{desafío}
latam_

*Academia de
talentos digitales*

www.desafiolatam.com