



FULL STACK DEVELOPMENT

Checkpoint_5

2024



AUTOR: MOHAMED ALWASSY
ALWASSY

Preguntas del checkpoint 5:

1. ¿Qué es un condicional?
2. ¿Cuáles son los diferentes tipos de bucles en Python? ¿Por qué son útiles?
3. ¿Qué es una lista por comprensión en Python?
4. ¿Qué es un argumento en Python?
5. ¿Qué es una función Lambda en Python?
6. ¿Qué es un paquete pip?

Respuestas:

1. Un condicional en programación, como su propio nombre indica, es o son requisito o requisitos que se deben cumplir, para ejecutar una acción o otra dependiendo de los datos recopilados por el programa, y en Python para llevar esta acción a cabo se utiliza el método `if`, `elif` y `else`, que explicaremos a continuación con un ejemplo muy básico.

Ejemplo:

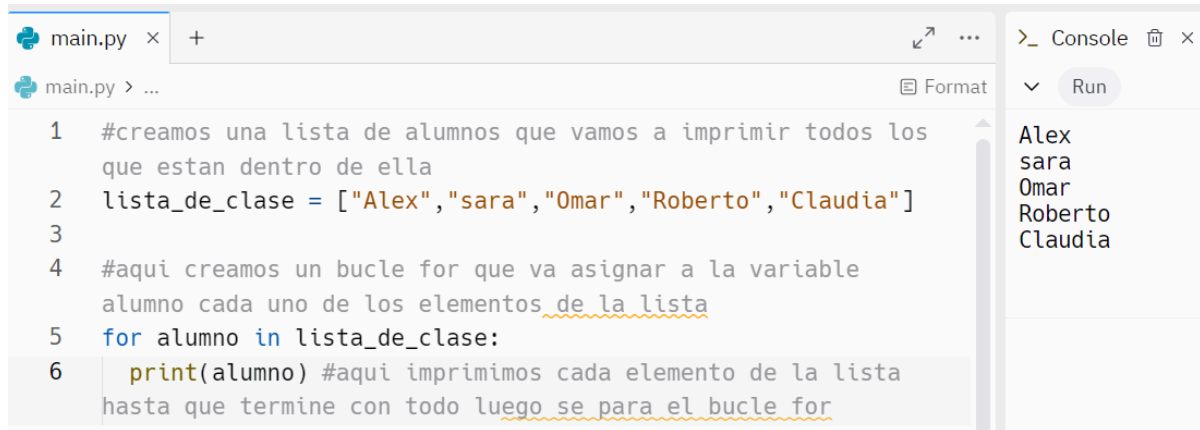
Vamos a crear un programa básico en el cual vamos a pedir al usuario que escriba su edad, y en base a eso vamos a imprimir "Eres un Niño" si su edad está entre 1 año y 10 años, "Eres un Adolescente" si su edad está entre 10 años y menor de 18, y "Eres un Adulto" si su edad es igual o superior a 18.

```
main.py > ...  
1  #aquí creamos una variable que va guardar la edad del usuario mediante input()  
2  su_edad = input("Escriba su edad aquí: ")  
3  
4  #Usamos try para asegurar que la edad es un número entero  
5  try:  
6      edad = int(su_edad)  
7  
8      # aquí está la primera condición que compara la edad si es de 1 a 10 años  
9      if(edad == 1 or edad <= 10):  
10         print("Eres es un niño") #imprime este mensaje  
11  
12         # aquí está la segunda condición que compara la edad si es de 11 a 17 años  
13         elif(edad == 11 or edad < 18):  
14             print("Eres un adolescente") #imprime este mensaje  
15  
16         #aquí la tercera condición si la edad introducida es igual o mayor de 18 años  
17         elif(edad >= 18):  
18             print("Eres un Adulto") #imprime este mensaje  
19  
20         #aquí else imprime este mensaje si el valor introducido no es un número  
21         else:  
22             print("Porfavor introduzca un número válido!")  
23  
24 #aquí except imprime este mensaje si el valor introducido no es un número  
25 except ValueError:  
26     print("Porfavor escriba un número entero!")
```

2 Los diferentes bucles en python se usan para ejecutar un bloque de código repetitivamente, o un número concreto de veces, verificar si se cumple una condición y en base a eso hacer una acción, vamos a ver con un ejemplo básico algunas de ellas:

- **For loop:** se usa para iterar sobre una secuencia como (lista, tupla, diccionario), o cualquier objeto iterable.

Ejemplo:



The screenshot shows a Python IDE with a file named 'main.py'. The code defines a list of names and uses a for loop to print each name. The console on the right shows the output of the loop.

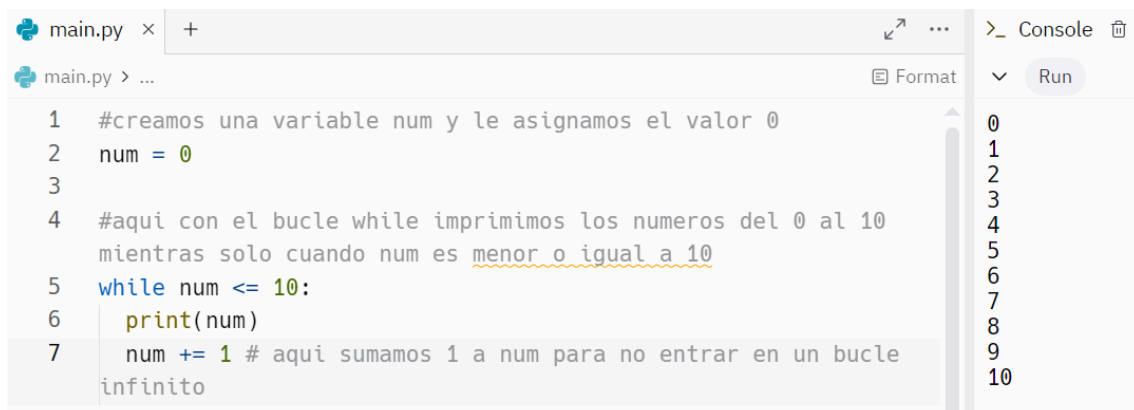
```
1 #creamos una lista de alumnos que vamos a imprimir todos los  
  que estan dentro de ella  
2 lista_de_clase = ["Alex", "sara", "Omar", "Roberto", "Claudia"]  
3  
4 #aqui creamos un bucle for que va asignar a la variable  
  alumno cada uno de los elementos de la lista  
5 for alumno in lista_de_clase:  
6     print(alumno) #aqui imprimimos cada elemento de la lista  
  hasta que termine con todo luego se para el bucle for
```

Console output:

```
Alex  
sara  
Omar  
Roberto  
Claudia
```

- **While loop:** es otro bucle que se usa en python, pero la diferencia que hay entre for loop y while loop es que while loop el bloque de código que está ejecutando se ejecuta al menos una vez antes de verificar la condición expuesta por el programador, mientras que for loop mira primero si se cumple la condición y luego ejecuta la acción por primera vez, también hay que tener mucho cuidado con no entrar en una interminable loop o infinite loop usando el while loop, vamos a poner un ejemplo donde imprimiremos los números del 0 al 10 usando el bucle while loop:

Ejemplo:



The screenshot shows a Python IDE with a file named 'main.py'. The code initializes a variable 'num' to 0 and uses a while loop to print numbers from 0 to 10. The console on the right shows the output of the loop.

```
1 #creamos una variable num y le asignamos el valor 0  
2 num = 0  
3  
4 #aqui con el bucle while imprimimos los numeros del 0 al 10  
  mientras solo cuando num es menor o igual a 10  
5 while num <= 10:  
6     print(num)  
7     num += 1 # aqui sumamos 1 a num para no entrar en un bucle  
  infinito
```

Console output:

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

- **For** algo **in range**(inicio, fin, paso): esta es otra bucle que tambien se usa en python para ejecutar algo en un rango determinado por el usuario. Vamos a poner un ejemplo donde imprimiremos una secuencia de numeros pares del 1 al 20 para ver como funciona este bucle:

Ejemplo:



```
main.py × +
main.py > ...
1 #aqui hemos creado una variable num que nos va a imprimir los
  numeros pares en rango del 0 a 20
2 for num in range(0,20,2):
3     print(num)
```

Console

```
0
2
4
6
8
10
12
14
16
18
```

- **For** con **Enumerate**: este es otro bucle que se usa en python para iterar sobre una secuencia o sacar el indice de cada elemento en una lista o tupla...etc. En este ejemplo a continuacion vamos a mostrar el indice que tiene cada uno de los elementos que componen una lista:

Ejemplo



```
main.py > ...
1 #hemos creado aqui una lista de alumnos
2 lista_alumnos = ["Adrian", "Amaia", "Lucas", "Joker", "Mario"]
3
4 #aqui hemos usado dos variables index y alumno y mediante la
  funcion enumerate sacamos el indice de cada alumno
5 for index , alumno in enumerate(lista_alumnos):
6     print(f'El nombre en el indice {index} es: {alumno}')
7
```

Console

```
El nombre en el indice 0 es: Adrian
El nombre en el indice 1 es: Amaia
El nombre en el indice 2 es: Lucas
El nombre en el indice 3 es: Joker
El nombre en el indice 4 es: Mario
```

- **For** con **Zip**: lo que hace este bucle es que itera sobre varias secuencias a la vez, y en este ejemplo vamos a utilizar el bucle for con zip para imprimir el nombre de cada alumno en una lista con su correspondiente edad:

Ejemplo:

```
main.py > ... Format Run Ask AI 69ms on 19:01
1 #aquí hemos creado dos listas de alumnos con sus
  correspondientes edades
2 alumnos= ["Ali","Carlos","Wasil","Karim"]
3 edades= [30,25,34,18]
4
5 #aquí hemos usado dos variables que con la función zip() el
  bucle itera sobre las dos listas asignando el valor de la
  primera lista a la variable alumno y el valor de la segunda
  edades a la variable edad
6 for edad , alumno in zip(edades,alumnos):
7     print(f'La edad de {alumno} es: {edad} años')
8
9
```

La edad de Ali es: 30 años
La edad de Carlos es: 25 años
La edad de Wasil es: 34 años
La edad de Karim es: 18 años

- **For** con **iter** y **next**: con este bucle se itera manualmente sobre una secuencia, he aquí un ejemplo:

```
main.py x + Console x
main.py > ... Format Run
1
2 numeros = [1,2,3,4,5,6,7,8,9,10]
3 iterator = iter(numeros)
4
5 while True:
6     try:
7         # Obtener el siguiente elemento del iterador
8         num = next(iterator)
9         print(num)
10    except StopIteration:
11        # Si se alcanza el final de la lista, salir del bucle
12        break
13
```

1
2
3
4
5
6
7
8
9
10

Como podemos ver en python hay diferentes tipos de bucles, y varia su función depende de lo que se desea conseguir implementar con cada bucle si es repetir de forma automática un bloque de código o ejecutar una acción una vez o cuando se cumplen algunas de las condiciones propuestas por el programador o mostrar un resultado específico.....y mucho más.

3 Es una manera eficiente de crear una lista en python con elementos como for in loop y condicionales, consiguiendo así simplificar varias líneas de código tradicionales en una línea de código, y para entenderlo mejor vamos a poner ejemplo.

En este ejemplo hemos creado un programa que multiplica al cubo los números de la lista, como se puede observar para llevar a cabo esto hay varias líneas de código, pero en el ejemplo de abajo con una línea de código se consigue la misma función.

```
main.py > ...  
1 num_list = range(1,11)  
2 numero_al_cubo = []  
3  
4 for num in num_list:  
5     numero_al_cubo.append(num ** 3)  
6  
7 print(numero_al_cubo)  
8
```

[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]

```
main.py > ...  
1 num_list = range(1,11)  
2 numero_al_cubo = [ num ** 3 for num in num_list]  
3 print(numero_al_cubo)  
4
```

[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]

He aqui otro ejemplo con un for in loop y condicional if:

```
main.py > ...  
1 num_list = range(1,11)  
2 num_par = []  
3 for num in num_list:  
4     if num % 2 == 0:  
5         num_par.append(num)  
6  
7 print(num_par)  
8  
9 #-----  
10 num_par = [num for num in num_list if num % 2 == 0]  
11 print(num_par)  
12
```

[2, 4, 6, 8, 10]
[2, 4, 6, 8, 10]

Como puedes ver con los dos ejemplos mostrados la list comprehension es una manera eficiente y bonita simplificar la cantidad de codigo tradicional para crear este tipo de soluciones.

4 Un argumento en python es un o varios parametros, que se le puede asignar a una funcion para devolver un operacion o resultado concreto. ejemplo:

```
def suma( numero_1, numero_2):  
  
    return numero_1 + numero_2  
  
resultado = suma(10, 14)  
  
print(resultado)
```

Como puedes ver en este ejemplo hemos creado una funcion que tiene dos parametros **numero_1** y **numero_2**, y que va a devolver la suma de estos dos numeros con los valores que

le hemos asignado en la **línea 3** del código guardados en la variable **resultado**.

Hay diferentes tipos de argumentos en python:

- **Argumentos posicionales**: básicamente el valor que tendrán sigue el orden en el que han sido creados en python por ejemplo:

```
def suma(a, b):
```

```
    return a + b
```

```
resultado = suma(3, 5) >> aquí el número 3 va a estar asignado a el argumento a y el número 5 al b  
en orden
```

```
print(resultado)
```

- **Argumentos de palabra clave (keyword arguments)**: no como el ejemplo de arriba con este tipo de argumentos nosotros asignamos los valores a los argumentos independientemente del orden de su creación. Ejemplo:

```
def resta(a, b):
```

```
    return a - b
```

```
resultado = resta(b=5, a=3) >> aquí nosotros asignamos los valores a los argumentos sin importar  
el orden en el que han sido creados arriba
```

- **Argumentos por defecto**: es el valor que se les asigna a los argumentos en el momento de creación de estos. Ejemplo:

```
def saludar(nombre="usuario"): >> argumento lleva el valor "usuario" por defecto
```

```
    return "Hola, " + nombre
```

```
saludo = saludar()
```

```
print(saludo) # Salida: "Hola, usuario"
```

5 **LAMBDA** es un tipo de función anónima en python que se usa cuando se quiere hacer una operación rápidamente sin la necesidad de declarar funciones formalmente, contiene una sola expresión y puede llevar varios argumentos.

Ejemplo:

```
# Función Lambda para calcular el cuadrado de un número
```

```
square = lambda x: x ** 2
```

```
print(square(3)) # Resultado: 9
```

```
# Funcion tradicional para calcular el cuadrado de un numero
```

```
def square1(num):
```

```
    return num ** 2
```

```
print(square(5)) # Resultado: 25
```

En el ejemplo de lambda anterior, `lambda x: x ** 2` produce un objeto de función anónimo que se puede asociar con cualquier nombre. Entonces, asociamos el objeto de función con `square`. De ahora en adelante, podemos llamar al objeto `square` como cualquier función tradicional, por ejemplo, `square(10)`

6 PIP es un sistema de gestion de paquetes utilizado para instalar y administrar paquetes de software escritos en python, muchos de ellos se pueden encontrar en bibliotecas de python packages PYPI y PIP3.

- PIP usa lineas de comandos para instalar un paquete especifico, y se hace asi:

```
pip install nombre-paquete
```

y para desinstalar un paquete se hace asi:

```
pip uninstall nombre-paquete
```

Los paquetes de pip pueden contener bibliotecas útiles, frameworks, herramientas y otros recursos que los desarrolladores pueden utilizar para construir aplicaciones y proyectos en Python de manera más eficiente.

Un ejemplo: **pip install requests** >> *Este comando descargará e instalará el paquete requests y todas sus dependencias*

import requests >> *importamos requests en nuestro programa para usar sus funciones*

```
# Ejemplo de uso de requests
```

```
response = requests.get('https://api.github.com') >> usamos la funcion requests.get para obtener una respuesta de la api de github
```



```
print(response.status_code) # Imprime el código de estado de la respuesta HTTP
```

devCamp

by BOTTEGA