BOTTEGA. FULL STACK DEVELOPMENT. CHECKPOINT 5

NOMBRE: Juan Camero Muñoz

PREGUNTAS:

1. **¿Qué es un condicional?**

Un condicional en Python es un comando que va a direccionar el flujo de ejecución de un programa. Ello quiere decir, que cuando se esté ejecutando un programa, la dirección del mismo es vertical y hacia abajo, sin embargo, al encontrarse un condicional, van a existir como mínimo dos posibilidades o más para continuar el programa.

Quedan bien representados con las palabras que se usan para determinar la condicionalidad de la ejecución del programa. Por ejemplo, con el uso de la palabra "IF": "Si eres mayor de 18 años podrás conducir un coche. Si no es así, no podrás". Expresa que si cumples la condición de tener 18 años, podrás conducir sin problemas, y en caso contrario, no será posible. De ahí, que el programa en ese punto, podrá tomar dos rutas alternativas. O bien ser apto para conducir, o no serlo.

En Python ello se escribe de la siguiente manera:

```
If edad >= 18:                          # Aquí se muestra la condición a cumplir: edad >= 18
        print("Apto para conducir")     # Si edad es mayor o igual de 18,será apto (1er camino)
else:                                   # Se usa para aquellos que no cumplen la condición
        print("No eres apto")           # No son aptos
```

El comando "if" también admite otras posibilidades, es decir, se pueden plantear no dos posibilidades de flujo, sino más. Para ello se necesitan más condiciones para reconocer qué flujo ha de tomar el programa al llegar a este tipo de condicionales. Ello se consigue con el comando "elif".

Por ejemplo:

```
if mapa == 'azul':                      # Si el color del mapa es azul (1ra condición)
   print("Temperatura baja")            # Si se confirma azul, advertir temperaturas bajas
elif mapa == 'verde':                   # Si es verde (2da condición)
   print("media")                       # Si se confirma verde, advertir temperaturas medias
elif mapa == 'amarillo':                # Si es amarillo (3ra condición)
   print("media-alta")                  # Si se confirma, advertir temperaturas medias-altas
elif mapa == 'naranja':                 # Si es naranja (4ta condición)
   print("alta")                        # Si se confirma, advertir temperaturas altas
else:                                   # Si tampoco es naranja, y es otra cualquiera
   print("muy alta")                    # Advertir temperaturas muy altas.
```

También es posible hacer más complejas las condiciones añadiendo operadores lógicos como:

Con1 and con2 -> deben cumplirse dos condiciones simultáneas.

Ej: if edad >= 18 and vision = 'normal':

Con1 or con2 -> al cumplirse una de ellas ya es determinante.

Ej: if mapa = 'naranja' or mapa = 'rojo':

Not -> Se usa para indicar que debe cumplir la condición contraria.

Ej:

pagado = True

cancelado = True

if pagado and not cancelado:

Por último, también vamos a mencionar la posibilidad de utilizar una tercera variable u operador ternario con la que aumentar la complejidad de la condición.

Ej:

pagado = True

inscripción = 'validado' if pagado else 'no válido'

2. ¿Cuáles son los diferentes tipos de bucles en Python? ¿Por qué son útiles?

En Python hay dos tipos de bucles más importantes:

- El bucle For:

  Se construye con la sintaxis básica siguiente:

  for "variable" in "iterable":

  operaciones del bucle.

  Donde "variable" es como vamos a llamar a cada elemento iterado. E "iterable" es el elemento que puede iterarse, ya sea una lista, una tupla, un diccionario, o una cadena string.

  Ejemplo:

  for n in range(1,10):

  print(n) -> Imprimirá un número del 1 al 9 en cada vuelta del bucle.

- El bucle While:

  El bucle while en Python se utiliza para ejecutar un bloque de código de forma repetitiva mientras se cumpla una condición específica. La estructura básica del bucle while es la siguiente:

  ```
  while “condición”:
      operaciones del bucle.
  ```

  Mientras la condición siga resultando “True” a lo largo del bucle, se seguirán repitiendo las operaciones internas. En el momento que resulte “False”, el bucle habrá terminado y no se reproducen las operaciones internas.

  Ejemplo:

  ```
  i = 0

  while i < 5:

      print(i)

      i += 1    -> En sus iteraciones, i irá sumando hasta llegar a 5, momento en que i<5 será “False” y ya no se imprimirá el número.
  ```

3. ¿Qué es una lista por comprensión en Python?

Una lista por comprensión en Python es una manera de crear listas más concreta y rápida que la manera tradicional. La sintaxis básica de una lista por comprensión es la siguiente:

[“Expresión que usa la variable” for “Variable” in “Iterable”]

1. La “Expresión que usa la variable”, es la operación que va a utilizar cada elemento del iterable.

2. El bucle for, va tomando cada valor del iterable, actualizando la “Expresión que usa la varable” en cada bucle.

Ejemplo:

```
cuadrados = [n * n for n in range(1, 11)]

print(cuadrados);        -> Imprimirá los cuadrados de los números del 1 al 10.
```

4. ¿Qué es un argumento en Python?

Un argumento es una parte de la definición de una función, en la cual podemos introducir información necesaria para la culminación de la función que queremos utilizar.

Es posible utilizar un número, un string, una lista, una tupla, un objeto, etc como argumento, o incluso varios de ellos, o a veces las funciones o presentan argumentos porque no necesitan información añadida para su culminación.

Veamos algunos ejemplos:

```
def buenos_dias()
        return print("Buenos días")       -> Imprime: Buenos días (sin argumentos)

nombre = 'Juan'
def saludos(nombre):
        return print(f'Saludos, {nombre}')         -> Imprime: Saludos, Juan (1 argumento String)

horas = 7
minutos = 30
def la_hora(horas, minutos):
        return print(f'Son las {horas} horas y {minutos} minutos')
                        -> Imprime: Son las 7 horas y 30 minutos (2 argumentos numéricos)
```

5. **¿Qué es una función Lambda en Python?**

Una función lambda, es otra forma de declarar funciones de una forma más compacta y corta, sin necesidad de utilizar la palabra clave def. La sintaxis básica sería:

Las funciones lambda se definen utilizando la siguiente sintaxis:

"Nombre de la función" = lambda "argumento/s": "expresión de retorno"

Donde:

- "Nombre de la función": el propio nombre de la función lambda.
- "Argumento/s": donde se introducen el/los argumento/s que se necesitan para la ejecución de la función.
- "Expresión de retorno": Operaciones que se realizan y resultado que devuelve la función.

Ejemplo:

```
multiplicar = lambda x, y: x * y

resultado = multiplicar(5, 3)

print(resultado)  # Salida: 15
```

Las funciones lambda en Python son herramientas útiles para definir funciones pequeñas de forma rápida y sencilla. Son especialmente útiles para tareas simples de procesamiento de datos o como argumentos en funciones de orden superior.

6. ¿Qué es un paquete pip?

Un paquete pip es un módulo, biblioteca o herramienta que proviene de la comunidad de desarrolladores del lenguaje y para la comunidad de desarrolladores. Existen distintos repositorios donde la comunidad de desarrolladores deja estos paquetes útiles para otros desarrolladores. Podemos hablar por tanto de determinadas fuentes externas de herramientas Python que podemos utilizar para desarrollar nuestras aplicaciones, y en la cual también podemos participar, ofreciendo elementos útiles para la comunidad.

El más reconocido es PyPI (Python Package Index), repositorio central de paquetes para Python. PyPI es una herramienta esencial para el ecosistema Python, ya que facilita la distribución y el uso de software de terceros.

Estos paquetes se instalan y administran utilizando la herramienta pip, que es el sistema de gestión de paquetes estándar para Python. Y una vez instalados, pueden ayudar a la programación del código de manera más rápida o sencilla. Sin ellos, también se puede programar una aplicación, pero con ellos, el desarrollo culminará antes e incluso de forma más eficiente, gracias al sometimiento continuo del trabajo de la comunidad de forma diaria, es decir, en base a su experiencia y su empeño de optimizar los resultados.

Si hablamos de modularidad, viene determinado por la forma en que se almacenan estos bloques de código. Ya que permiten dividir el código en unidades reutilizables y organizadas, para utilizarlos como módulos independientes. De manera que dos aplicaciones con un mismo objetivo, puede presentar distintos módulos, y con ellos, ser igual de eficientes, o no.

Por tanto, facilitan la reutilización de código desarrollado por otros programadores, ahorrando tiempo y esfuerzo en el desarrollo de aplicaciones. Y con pip, resolvemos las dependencias entre paquetes, asegurando que se instalen las versiones correctas de las bibliotecas necesarias para que una aplicación funcione de forma correcta y eficiente.

Un ejemplo, en lugar de calcular una media aritmética codificando línea por línea, usar una función predefinida como np.mean() del módulo numpy:

- Vamos a crear una función para calcular la media aritmética sin importaciones:
  Utilizaremos una lista de números con la variable “datos”.

```
datos = [10, 15, 20, 25, 30]
def calcular_media(datos):
        if not datos:
                return None
        suma = 0
        for dato in datos:
                suma += dato
                media = suma / len(datos)
                return print("La media aritmética es: ", media)
```

- Ahora usando el paquete de importación:

```
import numpy as np
datos = [10, 15, 20, 25, 30]
array_numpy = np.array(datos)
media = np.mean(array_numpy)
print("La media aritmética es: ", media)
```