RESPUESTAS AL CUESTIONARIO CHECKPOINT 5

¿Qué es un condicional?

Descripción

Es una estructura de control que permite tomar decisiones en función de si una condición es verdadera o falsa. Es decir, te permite ejecutar un bloque de código si la condición resulta ser verdadera u otro bloque de código, si la condición es falsa.

Hay varios tipos de condicionales que te permiten controlar el flujo de ejecución del programa. Los principales son:

✓ **if:** Es la estructura condicional básica que permite ejecutar un bloque de código si una condición es verdadera.

✓ **if-else:** Permite ejecutar un bloque de código si la condición evaluada es verdadera, y otro bloque de código si la condición es falsa.

✓ **if-elif-else:** Cuando necesitas evaluar múltiples condiciones de manera secuencial. Se prueba cada condición en orden y se ejecuta el bloque de código correspondiente al primer resultado verdadero. Si ninguna condición es verdadera, se ejecuta el bloque de código dentro del "else".

✓ **Operador ternario (if-else en una línea):** Permite escribir condicionales simples en una sola línea de código, lo que resulta útil en situaciones donde la lógica es breve y clara.

✓ **Nested if (if anidados):** Consiste en colocar un condicional dentro de otro condicional. Esto se usa cuando necesitas realizar pruebas adicionales dentro de una rama condicional.

Ejemplo simple:

Intento de autenticación de un usuario.

El usuario introduce los datos que le pide la aplicación, por ejemplo, nombre de usuario y contraseña.

La aplicación realizará una consulta a la base de datos devolviendo True si esos datos son correctos, es decir, si los datos se encuentran en la base de datos y False si no los encuentró.

Si el resultado es verdadero, la aplicación le permitirá acceder a las partes de la aplicación que necesitan autenticarse. Si devuelve falso, le dará un mensaje de error con la posibilidad de que recupere la contraseña y/o nombre de usuario, si ese ha sido el motivo del error, además le propondrá que genere una cuenta de usuario, si es que aún no la tiene.

Ejemplo de código condicional

```
# Método para consultar los datos introducidos por el usuario
def autenticar(usuario, contraseña):
    # Se produce una consulta a la base de datos  donde nos devolverá True o False
      …….
    # Verificamos con un condicional el resultado de la autenticación
    if autenticado: # Si la autenticación recibida es True
          print("¡Bienvenido! Autenticación exitosa.")
   else: # Si el resultado ha sido False
          print("Lo siento, la autenticación falló. Verifique su nombre de usuario y
contraseña.")
```

Operador ternario

Existe la posibilidad de escribir un condicional en una sola línea mediante operadores ternarios.

La sintaxis de un operador ternario sería la siguiente:

resultado_si_verdadero if condición else resultado_si_falso

Ejemplo sencillo:

```
resultado = "par" if  X % 2 == 0 else "impar"
```

En este caso si el número X es par el resultado guardará el string “par”, sino guardará el string “impar”.

Documentación

Puedes consultar más información sobre los condicionales con ejemplos y diagramas de flujo. Estas serían algunas de las páginas:

https://www.mclibre.org/consultar/python/lecciones/python-if-else.html
https://ellibrodepython.com/if-python
https://docs.python.org/3/reference/compound_stmts.html#patterns

¿Cuáles son los diferentes tipos de bucles en python? ¿Por qué son útiles?

En Python hay principalmente dos tipos de bucles, “for” y “while”. Estos bucles te permiten repetir un bloque de código varias veces. Se diferencian principalmente, en la forma en que se controla dicha repetición.

Descripción simple del “for”

El bucle “for” se utiliza para iterar sobre una secuencia, como puede ser una lista, tupla, diccionario, cadena o un objeto iterable.

Sintaxis

La sintaxis básica de un bucle “for” es la siguiente: for elemento in secuencia:

Un ejemplo simple:

```
for num in range(1:11):
    print(num) # Imprimiría los números del 1 al 10.
```

La variable “num” podría llamarse de cualquier forma, aunque lo aconsejable es que sea en singular y que además el propio nombre aporte una descripción sobre lo que se

va a guardar en ella. Esta variable irá cambiando de valor, según va iterando sobre la secuencia de números del 1 al 10. Primer valor será 1 y lo imprimirá, segundo valor 2 y lo imprimirá y así hasta llegar a 10.

Descripción simple del “while”

El bucle “while” se utiliza para repetir un bloque de código, mientras la condición del while sea verdadera. No se conoce el número de veces que se va a iterar sobre el mismo.

Sintaxis

La sintaxis básica de un bucle “while” es la siguiente:

```
while condición:
    # Cuerpo del bucle
    # Se ejecuta mientras la condición sea verdadera
```

Un ejemplo simple:

```
import random

# Generar un número aleatorio entre 1 y 100
numero_secreto = random.randint(1, 100)

# Bucle para que el usuario adivine el número
while True:
# Pedir al usuario que ingrese un número
intento = int(input("Intenta adivinar el número secreto: "))

# Comprobar si el número coincide con el número secreto
if intento == numero_secreto:
        print("¡Felicidades! Has adivinado el número secreto.")
        break  # Salir del bucle si el número es correcto
else:
        print("Número incorrecto. ¡Inténtalo de nuevo!")
```

Se trataría de un juego para adivinar el número secreto. Si no lo aciertas sigues teniendo la oportunidad de volver a introducir otro número y si lo aciertas te felicita y sales del bucle “while”.

Imprime el valor de “i” y le suma 1 a su valor. Ahora el valor de “i” pasa a ser 1 y sigue por tanto cumpliendo la condición y vuelve a repetir el código que hay dentro del while…. Así hasta que “i” pasa a valer 5. En ese momento ya no cumple con la condición y por tanto sale del bucle.

Los bucles son útiles porque permiten ejecutar un bloque de código varias veces de manera eficiente y controlada, con las siguientes diferencias:

Bucle “for”

- ✓ Se utiliza para iterar sobre secuencias. Perfecto cuando sabes la cantidad exacta de elementos a recorrer o deseas iterar sobre una secuencia de datos, como son listas, tuplas, diccionarios…
- ✓ Tiene una sintaxis simple y fácil de entender, repitiendo operaciones en cada elemento a recorrer.
- ✓ Es menos propenso a errores, ya que al iterar sobre una secuencia definida, es más difícil que se produzcan errores relacionados con la condición de salida del bucle.

Bucle “while”

- ✓ Es un bucle más flexible que el bucle “for”. Te permite ejecutar un bloque de código mientras la condición sea verdadera. Útil para cuando no sabes exactamente cuantas iteraciones necesitas para que se cumpla la condición de salida.
- ✓ Mayor control sobre la iteración, pudiendo poner una variedad de condiciones como condición de salida.
- ✓ Es muy útil para realizar tareas repetitivas.

Documentación

Puedes consultar más información sobre los bucles con ejemplos, diagramas de flujo y diferencias entre for y while, en las páginas siguientes:

https://ellibrodepython.com/estructuras-control-python
https://medium.com/@diego.coder/ciclos-en-python-for-y-while-20cbe73f7193
https://www.geeksforgeeks.org/difference-between-for-loop-and-while-loop-in-python/

¿Qué es una comprensión de listas en python?

Descripción simple

Es una expresión concisa para crear listas a partir de cualquier tipo de secuencia o iterable, aplicando un expresión a cada elemento de la secuencia.

Sintaxis

La sintaxis básica de una comprensión de listas es la siguiente:

```
nueva_lista = [expresión for elemento in iterable if condicion]
```

1. "expresión" es la operación, elemento… que se te ocurra, que se aplica a cada elemento del iterable, creando una nueva lista.
2. "elemento" es la variable que representa cada elemento del iterable.
3. "iterable" es la secuencia, lista, rango u otro tipo de iterable sobre el cual iterar.
4. "condición" es una expresión opcional que filtra los elementos del iterable (puede ser omitida).

Un ejemplo simple:

```
cuadrados = [x ** 2 for x in range(1, 6)]
print(cuadrados)  # Salida: [1, 4, 9, 16, 25]
```

Para cada valor “x” en el rango del 1 al 5, elevamos “x” al cuadrado y agregamos el resultado a la lista cuadrados.

Documentación

Puedes consultar más información sobre las listas de comprensión en las páginas siguientes:

https://ellibrodepython.com/list-comprehension-python
https://docs.python.org/es/dev/tutorial/datastructures.html
https://codigospython.com/listas-por-comprension-en-python-explicacion-y-ejemplos/

¿Qué es un argumento en Python?

Un argumento es un valor que se le pasa a una función o método cuando se le llama. Se trata de información que necesita para que el método o función sea ejecutado correctamente.

Hay varios tipos de argumentos:

✓ Argumentos posicionales: Son los valores que se pasan a una función en el orden en el que aparecen en la definición de la misma. También deben cumplir con el mismo número de parámetros que están definidos.

Ejemplo:

```
def saludar(nombre, saludo):
    print(f"{saludo}, {nombre}!")

saludar("Juan", "Hola") # Al llamar a la función se le pasan los
                        parámetros definidos en la función saludar.
```

En este ejemplo, "Juan" es el primer argumento (que corresponde al primer parámetro nombre), y "Hola" es el segundo argumento (que corresponde al segundo parámetro saludo).

✓ Argumentos de palabra clave (keyword arguments): Son argumentos que se pasan a una función con el nombre del parámetro al que se van a asignar. Por esta razón no importa el orden de los parámetros.

Ejemplo: saludar(saludo="Hola", nombre="Juan")

✓ Argumentos predeterminados (default arguments): Son argumentos que tienen un valor predeterminado en la definición de la función y por tanto pueden ser omitidos al llamar a dicha función.

Ejemplo:

```
def saludar(nombre, saludo="Hola"):
    print(f"{saludo}, {nombre}!")

saludar("Juan")
```

✓ Argumentos indefinidos: Cuando lo que queremos es pasar una cantidad indefinida de argumentos, se utiliza la expresión *args, como nombre en la definición del método o función. Si además lo que queremos es que sean argumentos clave= valor lo que se pase se usará **kwargs

Ejemplo para 3 tipos de variables:

```
def greeting(time_of_day, *args, **kwargs):
    print(f"Hi {' '.join(args)}, I hope that you're having a good {time_of_day}.")

    if kwargs:
        print('Your tasks for the day are:')
        for key, val in kwargs.items():
            print(f'{key} -> {val}')
greeting('Morning',
        'Kristine', 'Hudgens',
        first = 'Empty dishwasher',
        second = 'Take pupper out',
        third = 'math homework')
```

En este ejemplo 'Morning' será un argumento posicional, tiene que ir el primero porque así está definido. Seguido van dos argumentos, 'Kristine', 'Hudgens', que se corresponden con argumentos indefinidos, que se definien en la función como *arg. Y por último en tercer lugar van los argumentos con clave, first = 'Empty dishwasher', second = 'Take pupper out', third = 'math homework', que se corresponden con los argumentos indefinidos definidos en la función como **kwargs.

Documentación

Puedes consultar más información sobre las argumentos en las páginas siguientes:

https://recursospython.com/guias-y-manuales/argumentos-args-kwargs/
https://www.learnpython.org/es/Multiple%20Function%20Arguments

¿Qué es una función de Python Lambda?

Descripción simple

Se trata de una función anónima, esto significa que no tiene nombre. Esta función se crea utilizando la palabra clave lambda, seguida de una lista de parámetros y una expresión que define lo que la función debe devolver.

Es muy útil para pasar una función, como argumento a otra función, como son las funciones de orden superior map(), filter() y reduce().

Sintaxis

La sintaxis general de una función lambda es la siguiente:

lambda parametros: expresion

1. "parametros" representa los parámetros que la función lambda puede tomar.
2. "expresion" es el valor que la función devuelve basado en esos parámetros.

Un ejemplo simple:

```
suma = lambda x, y: x + y
print(suma(3, 5))  # Salida: 8
```

En este caso, la función recibe dos parámetros “x” e “y” en este caso 3 y 5 respectivamente y devuelve la suma de ambos parámetros.

Ejemplo pasando como argumento a una función:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # Output: [2, 4, 6, 8, 10]
```

Se imprime una lista con los números que cumplen la condición, es decir los números pares.

Documentación

https://www.freecodecamp.org/espanol/news/expresiones-lambda-en-python/
https://ellibrodepython.com/lambda-python
https://atareao.es/pyldora/las-maravillas-de-las-funciones-lambda-en-python/

¿Qué es un paquete pip?

Un paquete pip es un conjunto de archivos que contienen código Python, que permite que sea fácilmente distribuido e instalado usando la herramienta pip.

Pip es un sistema de gestión de paquetes estándar para Python que facilita la instalación y gestión de los paquetes y dependencias de Python.

Los paquetes pip son distribuidos a través del Python Package Index (PyPI), que es un repositorio centralizado de software de Python de código abierto. PyPI contiene miles de paquetes que cubren una amplia gama de funcionalidades, desde bibliotecas para ciencia de datos y desarrollo web hasta herramientas de automatización etc...

Algunos ejemplos de paquetes pip populares incluyen:

1. NumPy y pandas para computación científica y análisis de datos.
2. Django y Flask para desarrollo web.
3. TensorFlow y PyTorch para aprendizaje profundo y machine learning.
4. Matplotlib y Seaborn para visualización de datos.

Para instalar un paquete pip, generalmente se utiliza el comando **pip install seguido del nombre del paquete** que se desea instalar. Por ejemplo, para instalar el paquete requests, que es comúnmente utilizado para realizar solicitudes HTTP en Python, se ejecutaría el siguiente comando en la línea de comandos:

pip install requests

Una vez instalado, el paquete estará disponible para ser importado y utilizado en el programa Python que estás desarrollando.

Los paquetes pip facilitan mucho la gestión de dependencias y la distribución de software, permitiendo a los desarrolladores utilizar y compartir código de manera eficiente y efectiva.

Documentación

Para más información sobre el paquete pip, su instalación, documentación, código…:

https://pypi.org/project/pip/