2025C1-G01-FUNDAMENTOS-DE-PYTHON / Clases / Clase 06

/ NOTEBOOK 03 - PARTE I Ejecución Condicional, Bucles, Listas y su Procesamiento.ipynb

Jia231 Update NOTEBOOK 03 - PARTE I Ejecución Condicional, Bucles, Listas y ... c07995b · now History

Preview Code Blame 905 lines (905 loc) · 36.6 KB Raw

NOTEBOOK 03 - PARTE I Ejecución Condicional, Bucles, Listas y su Procesamiento

Fundamentos de Python | UMCA

Profesor: Ing. Andrés Mena Abarca

Nombre del estudiante: Jia Ming Liou

---

1. Valores Booleanos

1.1. **Teoría**

En Python, los valores booleanos son un tipo de dato que solo pueden tener dos valores: `True` (verdadero) y `False` (falso). Estos valores son fundamentales para el control de flujo en los programas, permitiendo que se tomen decisiones basadas en condiciones.

- `True` y `False` son palabras clave en Python.
- Se usan principalmente en expresiones condicionales y bucles.

Operadores de comparación: Estos operadores devuelven un valor booleano como resultado de comparar dos valores.

- `==` (igual a)
- `!=` (diferente de)
- `<` , `>` , `<=` , `>=` (menor, mayor, menor o igual, mayor o igual)

1.2. **Ejemplo de Código**

In [5]:
```
# Ejemplo de valores booleanos
x = 10
y = 20

# Operación de comparación
es_mayor = x > y  # False porque 10 no es mayor que 20
es_igual = x == y  # False porque 10 no es igual a 20

print(f"¿Es {x} mayor que {y}? {es_mayor}")
print(f"¿Es {x} igual a {y}? {es_igual}")
```

```
¿Es 10 mayor que 20? False
¿Es 10 igual a 20? False
```

1.3. **Ejercicio Práctico**

Caso de Aplicación: Evaluación de Edades para una Competencia Deportiva

Imagina que estás organizando una competencia deportiva y necesitas verificar si un participante es elegible para competir en una categoría específica de edad. En esta categoría, los participantes deben tener entre **18 y 30 años** (inclusive). También se deben evaluar dos edades de los participantes, para ver quién es mayor o si tienen la misma edad.

El ejercicio requerirá:

- Evaluar si un participante es elegible para competir según su edad.
- Comparar las edades de dos participantes para determinar quién es mayor.

Proceso

1. **Solicita** al usuario que ingrese la edad de dos participantes (edad1 y edad2).
2. **Verifica** si cada uno de ellos es elegible para la competencia (la edad debe estar entre 18 y 30 años inclusive).
3. **Compara** las edades de los dos participantes: - ¿Quién es mayor? - ¿Son de la misma edad?
4. **Imprime** resultados detallados de las evaluaciones.

In [6]:
```
#Respuesta

edad1 = int(input("Digite la edad jugador 1"))
edad2 = int(input("Digite la edad jugador 2"))

jugador1_elegible = (edad1 >= 18 and edad1 <= 30)
jugador2_elegible = (edad2 >= 18 and edad2 <= 30)
if(jugador1_elegible):
```

```
    print("Usuario 1 es elegible en la competencia")
else:
    print("Usuario 1 NO es elegible en la competencia")


if(jugador2_elegible):
    print("Usuario 2 es elegible en la competencia")
else:
    print("Usuario 2 NO es elegible en la competencia")

if(edad1 == edad2):
    print("Los 2 participantes son de la misma edad")
elif(edad1 > edad2):
    print("Usuario 1 es mayor a Usuario 2")
else:
    print("Usuario 2 es mayor a Usuario 1")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[6], line 3
      1 #Respuesta
----> 3 edad1 = int(input("Digite la edad jugador 1"))
      4 edad2 = int(input("Digite la edad jugador 2"))
      6 jugador1_elegible = (edad1 >= 18 and edad1 <= 30)

ValueError: invalid literal for int() with base 10: ''
```

2. Ejecución Condicional

2.1. Teoría

La ejecución condicional permite que el programa tome decisiones y ejecute diferentes bloques de código basados en ciertas condiciones. En Python, las estructuras condicionales más comunes son `if`, `elif` (else if) y `else`.

Estructura básica:

```
if condición:
    # Código si la condición es True
elif otra_condición:
    # Código si otra_condición es True
else:
    # Código si ninguna condición es True
```

2.2. Ejemplo de Código: Clasificación de Calificaciones

Contexto:

Imagina que estás desarrollando un sistema para clasificar las calificaciones de un grupo de estudiantes. El sistema debe recibir una calificación y asignar una letra basada en la siguiente escala:

- Si la calificación es mayor o igual a 90, el estudiante obtiene una **A**.
- Si la calificación está entre 80 y 89, el estudiante obtiene una **B**.
- Si la calificación está entre 70 y 79, el estudiante obtiene una **C**.
- Si la calificación está entre 60 y 69, el estudiante obtiene una **D**.
- Si la calificación es menor que 60, el estudiante obtiene una **F**.

Estructura Condicional

Usaremos una serie de sentencias `if`, `elif` y `else` para implementar esta lógica.

In [ ]:

```
#Respuesta
# Ejemplo mejorado: Clasificación de calificaciones de un estudiante

calificacion = float(input("Digite la nota del estudiante"))
try:
    if(calificacion >= 90):
        print("A")
    elif(calificacion >=80):
        print("B")
    elif(calificacion >=70):
        print("C")
    elif(calificacion >=60):
        print("D")
    else:
        print("F")
except ValueError:
    print("error: ingrese un valor valido para la nota")
```

```
F
```

2.3. Ejercicio Práctico

Escribe un programa que pida al usuario ingresar su edad. Luego, el programa debe imprimir si es mayor o menor de edad (asumiendo 18 años como el límite).

- El programa pide al usuario ingresar su edad.
- Si el usuario ingresa un valor no válido (por ejemplo, texto o números negativos), le solicita nuevamente que ingrese un número válido.
- Dependiendo de la edad ingresada, el programa:
  - Imprime si es "menor de edad" (menor de 18).
  - Imprime si es "mayor de edad" (18 años o más).
  - Opcionalmente, imprime un mensaje especial para niños menores de 13 años o personas mayores de 65 años.
- Permite que el usuario repita el proceso tantas veces como desee. ***OPCIONAL

In [ ]:
```python
# Declaración de variables
# Ejercicio para verificar si una persona es mayor o menor de edad
numero_valido = False
quieres_seguir_participando = True

while(quieres_seguir_participando):
    while(numero_valido == False):
        edad = input("Digite su edad")
        if(edad.isdigit() and int(edad) > 0):
            numero_valido = True

    edad_num = int(edad)

    if(edad_num >= 18):
        if(edad_num > 65):
            print("Todavia estas joven")
        print("El usuario es mayor de edad")
    else:
        if(edad_num <= 13):
            print("Eres un nino")
        print("Eres un menor de edad")

    seguir_participando = input("Quieres seguir participando? (Si/No)")

    if(seguir_participando == "No"):
        quieres_seguir_participando = False
```

```
Eres un nino
Eres un menor de edad
```

3. Bucles

3.1. Teoría

Los bucles permiten repetir un bloque de código varias veces. Python tiene dos tipos principales de bucles: `for` y `while`.

- **Bucle `for`:** Se utiliza para iterar sobre una secuencia (listas, tuplas, cadenas, etc.).
- **Bucle `while`:** Repite un bloque de código mientras una condición sea `True`.

3.2. Ejemplo de Código

- **Bucle `for`:**

In [ ]:
```python
# Ejemplo de bucle for

for i in range(1,10):
    print(f"Iteracion {i}")

for i in range(10):
    print(f"Iteracion {i}")

for i in range(0,100,10):
    print(f"Iteracion {i}")
```

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

```
Iteracion 3
Iteracion 4
Iteracion 5
Iteracion 6
Iteracion 7
Iteracion 8
Iteracion 9
Iteracion 0
Iteracion 10
Iteracion 20
Iteracion 30
Iteracion 40
Iteracion 50
Iteracion 60
Iteracion 70
Iteracion 80
Iteracion 90
```

Caso: Suma de numeros del 1 a la cantidad que defina el usuario.

Vamos a suponer que queremos calcular la suma de los números del 1 a la catidad que deficna el suario. Este caso es útil para entender cómo funciona el bucle `for`, ya que iteramos sobre una secuencia de números y realizamos una operación en cada iteración.

In [ ]:
```python
#Respuesta

suma = 0

num = int(input("Por favor ingrese el numero a sumar en secuencia."))

for i in range(0,num + 1):
    suma = suma + i

print(f"La suma de los numeros del 1 al {num} es : {suma}")
```

```
La suma de los numeros del 1 al 9 es : 36
```

- **Bucle** `while` :

In [ ]:
```python
# Ejemplo de bucle while
contador = 1
suma = 0
while(contador <= 10):
    print(f"Contador: {contador}")
    suma = suma + contador

print(f"La suma del numero 1 al 10 es: {contador}")
```

Caso: Cálculo de Total de Compras

Imagina que estás desarrollando un pequeño programa para una tienda. Los clientes ingresan el precio de varios productos que están comprando, uno por uno. El programa debe seguir pidiendo el precio de cada producto hasta que el usuario indique que no va a ingresar más productos, y al final, debe calcular y mostrar el total a pagar.

In [12]:
```python
#Respuesta
listo_pagar = False
total = 0
while(listo_pagar == False):
    precio = float(input("Digite el precio del producto: "))
    total = total + precio
    es_final_compras = input("Desea finalizar la compra? (Si/No)")
    if(es_final_compras.upper() == "SI"):
        listo_pagar = True

print(f"Total a pagar: {total}")
```

```
Total a pagar: 5000.0
```

3.3. **Ejercicio Práctico** - Tabla de Multiplicar

Descripción: Crea un programa que pida al usuario un número entero `n` y muestre la tabla de multiplicar de ese número del 1 al 10. Utiliza un ciclo `while` para generar la tabla.

Instrucciones:

1. Solicita al usuario que ingrese un número entero.
2. Inicializa una variable `i` en 1.
3. Utiliza un ciclo `while` para que mientras `i` sea menor o igual a 10:
   - Multiplica `n` por `i` y muestra el resultado en el formato "n x i = resultado".
   - Incrementa `i` en 1.
4. Al final, muestra la tabla completa.

Ejemplo de salida:

```
Ingrese un número entero: 3
Tabla de multiplicar de 3:

3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
```

In [10]:

```python
#Respuesta

tabla = int(input("Digite un numero"))
contador = 1
while(contador <=10):
    print(f"{tabla} x {contador} = {tabla*contador}")
    contador = contador + 1
```

```
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10
```

3.3. Ejercicio Práctico

Escribe un programa que imprima los números del 1 al 10 usando un bucle for. Luego, realiza el mismo ejercicio con un bucle while.

In [11]:

```python
#Respuesta
for i in range (1, 10):
    print(i)

contador = 1
while(contador <= 10):
    print(contador)
    contador = contador + 1
```

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

Caso 2: Suma de Números

- **Presentación:** Supongamos que tienes una lista de números y quieres calcular la suma total. ¿Cómo podrías hacer esto en Python?
- **Salida Esperada:**
  - La suma total de los números en la lista.
- **Preguntas adicionales:**
  - ¿Cómo podrías calcular el promedio de los números?
  - ¿Y si quisieras encontrar el número más grande y el más pequeño?
  - ¿Puedes modificar el código para que solo sume los números pares?

4. Listas y su Procesamiento

4.1. Teoría

Una lista es una estructura de datos en Python que almacena múltiples elementos en un solo objeto. Las listas son mutables, lo que significa

que podemos modificar su contenido después de haberlas creado.

Operaciones comunes con listas:

- **Añadir elementos:** `append()`, `insert()`
- **Eliminar elementos:** `remove()`, `pop()`
- **Acceso a elementos:** Índices
- **Slicing:** Obtener sublistas

4.2. **Ejemplo de Código**

In [12]:
```
# Ejemplo de procesamiento de listas
```

4.3. **Ejercicio Práctico**

Imagina que quieres crear una lista de tus amigos. ¿Cómo podrías almacenar estos nombres en Python? Una vez que tengas la lista, ¿cómo podrías agregar un nuevo amigo? ¿Y cómo podrías saludar a cada uno de ellos?

```
amigos = ["Ana", "Juan", "María", "Pedro"\] \# ... (resto del código)
```

Salida Esperada:

```
- Una lista impresa con todos los nombres de los amigos.
- El nombre del tercer amigo.
- Un saludo personalizado para cada amigo.
```

Preguntas adicionales:

```
- ¿Cómo podrías eliminar a un amigo de la lista?
- ¿Qué pasa si quieres cambiar el nombre de un amigo?
- ¿Puedes ordenar la lista alfabéticamente?
```

In [13]:
```
#Respuesta
```

4. Listas y su Procesamiento

4.1. **Teoría**

Una lista es una estructura de datos en Python que almacena múltiples elementos en un solo objeto. Las listas son mutables, lo que significa que podemos modificar su contenido después de haberlas creado.

Operaciones comunes con listas:

- **Añadir elementos:** `append()`, `insert()`
- **Eliminar elementos:** `remove()`, `pop()`
- **Acceso a elementos:** Índices
- **Slicing:** Obtener sublistas

4.2. **Ejemplo de Código**

In [14]:
```
# Ejemplo de procesamiento de listas
#            0         1         2
```

append(elemento):

- Agrega un elemento al final de la lista.

In [ ]:
```

```

insert(índice, elemento):

- Inserta un elemento en una posición específica de la lista.
- El índice indica la posición antes de la cual se insertará el elemento.

In [ ]:
```

```

remove(elemento):

- Elimina la primera ocurrencia de un elemento específico de la lista.
- Si el elemento no existe, se produce un error.

In [ ]:

pop(índice):

- Elimina y devuelve el elemento en la posición indicada.
- Si no se especifica un índice, elimina y devuelve el último elemento.

In [ ]:

Caso Final

Un profesor quiere llevar un registro de los nombres de sus estudiantes. Ayúdale a crear un programa que le permita agregar nuevos estudiantes, ver la lista completa y acceder a la información de un estudiante en particular.

In [ ]: