Actividad-3-MAUV-AD

August 20, 2024

1 Analítica de Datos 2024-B MAUV

1.1 Actividad 3-A

1.2 1. Variables, Tipos de Variables y Operadores en Python

Dentro del presente cuaderno se explicarán las variables, los tipos de variables y los operadores. Las variables permiten almacenar y manipular datos, los tipos de variables definen la naturaleza de esos datos, y los operadores permiten realizar operaciones sobre ellos, facilitando la ejecución de cálculos y la toma de decisiones en el código.

1.3 Variables

Las variables se utilizan para referirse a valores almacenados en la memoria del ordenador. Una variable actúa como un contenedor que guarda datos que se pueden usar y modificar a lo largo del programa.

1.4 Ejemplo

```
edad = 25  # Variable de tipo entero
nombre = "Juan Pablo"  # Variable de tipo
```

Como podemos observar en el ejemplo anterior, edad es una variable que almacena un número entero, y nombre es una variable que almacena una cadena de texto.adena de texto

[12]:
```
# Ejemplo de variables

# Variable de tipo entero
edad = 30  # Un número entero sin decimales

# Variable de tipo flotante
altura = 1.75  # Un número decimal (flotante)

# Variable de tipo cadena de texto
nombre = "Ana"  # Una secuencia de caracteres entre comillas

# Variable de tipo booleano
es_estudiante = True  # Puede ser True (verdadero) o False (falso)

# Imprimir los valores para ver el resultado
```

```
print("Edad:", edad)
print("Altura:", altura)
print("Nombre:", nombre)
print("Es estudiante:", es_estudiante)
```

```
Edad: 30
Altura: 1.75
Nombre: Ana
Es estudiante: True
```

1.5 Tipos de Operadores en Python

Los operadores en Python son símbolos que permiten realizar operaciones sobre variables y valores. Existen varios tipos de operadores que cumplen diferentes funciones en el código.

1.6 Ejemplo de Operadores

```
# Operadores Aritméticos
suma = 10 + 5  # Suma
resta = 10 - 5  # Resta
producto = 10 * 5  # Multiplicación
cociente = 10 / 5  # División
modulo = 10 % 3  # Resto de la división
exponente = 2 ** 3  # Exponente

# Operadores de Comparación
igual = (10 == 5)  # Igual a
diferente = (10 != 5)  # Diferente de
mayor = (10 > 5)  # Mayor que
menor = (10 < 5)  # Menor que
mayor_igual = (10 >= 5)  # Mayor o igual que
menor_igual = (10 <= 5)  # Menor o igual que

# Operadores Lógicos
y_logico = (10 > 5) and (5 < 3)  # Y lógico
o_logico = (10 > 5) or (5 < 3)  # O lógico
no_logico = not (10 > 5)  # No lógico
```

1.6.1 1. Operadores Aritméticos:

Realizan operaciones matemáticas básicas como suma, resta, multiplicación y división. ### 2. Operadores de Comparación: Comparan dos valores y devuelven un valor booleano (True o False) dependiendo de si la comparación es verdadera o falsa. ### 3. Operadores Lógicos: Se utilizan para combinar condiciones booleanas y devuelven True o False según las condiciones combinadas.

[19]:
```
# Operadores Aritméticos
suma = 10 + 5  # Suma
resta = 10 - 5  # Resta
```

```
producto = 10 * 5  # Multiplicación
cociente = 10 / 5  # División
modulo = 10 % 3  # Resto de la división
exponente = 2 ** 3  # Exponente

# Operadores de Comparación
igual = (10 == 5)  # Igual a
diferente = (10 != 5)  # Diferente de
mayor = (10 > 5)  # Mayor que
menor = (10 < 5)  # Menor que
mayor_igual = (10 >= 5)  # Mayor o igual que
menor_igual = (10 <= 5)  # Menor o igual que

# Operadores Lógicos
y_logico = (10 > 5) and (5 < 3)  # Y lógico
o_logico = (10 > 5) or (5 < 3)  # O lógico
no_logico = not (10 > 5)  # No lógico

# Imprimir los valores para ver el resultado
print("Suma:", suma)
print("Resta:", resta)
print("Producto:", producto)
print("Cociente:", cociente)
print("Módulo:", modulo)
print("Exponente:", exponente)

print("Igual:", igual)
print("Diferente:", diferente)
print("Mayor:", mayor)
print("Menor:", menor)
print("Mayor o igual:", mayor_igual)
print("Menor o igual:", menor_igual)

print("Y lógico:", y_logico)
print("O lógico:", o_logico)
print("No lógico:", no_logico)
```

```
Suma: 15
Resta: 5
Producto: 50
Cociente: 2.0
Módulo: 1
Exponente: 8
Igual: False
Diferente: True
Mayor: True
Menor: False
```

```
Mayor o igual: True
Menor o igual: False
Y lógico: False
O lógico: True
No lógico: False
```

1.7 2. Funciones y Módulos en Python

En esta sección, exploraremos las funciones y módulos en Python. Las funciones son bloques de código reutilizables que realizan una tarea específica, mientras que los módulos permiten organizar y reutilizar código a lo largo de diferentes programas.

1.7.1 Funciones

Las funciones en Python son bloques de código definidos por el usuario que realizan tareas específicas. Se definen utilizando la palabra clave def, seguida del nombre de la función y los paréntesis que pueden contener parámetros.

1.7.2 Etapas para Crear una Función:

Definir la Función: Utilizamos la palabra clave def, seguida del nombre de la función y los paréntesis. Los parámetros, si los hay, se colocan dentro de los paréntesis. El bloque de código de la función se indenta.

Implementar el Cuerpo de la Función: Dentro del bloque indentado, se escribe el código que realiza la tarea deseada.

####Retornar un Valor (Opcional): Utilizamos la palabra clave return para devolver un valor desde la función. Si no se usa return, la función devolverá None por defecto.

####Llamar a la Función: Una vez definida, la función puede ser llamada en cualquier parte del programa utilizando su nombre y pasando los argumentos necesarios.

1.7.3 Ejemplo de Funciones

```python
# Definición de la función para calcular el área de un círculo
def calcular_area_circulo(radio):
    """
    Calcula el área de un círculo dado su radio.
    """
    import math  # Importar el módulo math para usar la constante pi
    area = math.pi * (radio ** 2)  # Fórmula para calcular el área
    return area  # Devolver el resultado

# Llamada a la función con un radio de 5
resultado = calcular_area_circulo(5)
print("Área del círculo con radio 5:", resultado)
```

La función calcular_area_circulo toma un parámetro (radio) y utiliza la fórmula del área del círculo (* radio^2) para calcular y devolver el área. Se utiliza el módulo math para acceder a la constante pi.

[28]:
```python
# Definición de la función para calcular el área de un círculo
def calcular_area_circulo(radio):
    """
    Calcula el área de un círculo dado su radio.
    """
    import math  # Importar el módulo math para usar la constante pi
    area = math.pi * (radio ** 2)  # Fórmula para calcular el área
    return area  # Devolver el resultado

# Llamada a la función con un radio de 5
resultado = calcular_area_circulo(5)
print("Área del círculo con radio 5:", resultado)
```

```
Área del círculo con radio 5: 78.53981633974483
```

1.8 Módulos

Los módulos en Python son archivos que contienen definiciones de funciones, clases y variables que pueden ser reutilizadas en diferentes programas. Un módulo se importa utilizando la palabra clave import.

1.8.1 Etapas para Usar Módulos:

1. Crear un Módulo: Define un archivo .py con funciones, clases o variables que quieras reutilizar.
2. Importar el Módulo: Utiliza la palabra clave import para incluir el módulo en tu script.
3. Usar el Contenido del Módulo: Llama a las funciones o usa las variables del módulo como si fueran parte de tu script actual.

Ejemplo de Módulo

```python
# matematica.py


# Módulo con funciones matemáticas básicas

def suma(a, b):
    """
    Retorna la suma de dos números.
    """
    return a + b

def resta(a, b):
    """
    Retorna la resta de dos números.
    """
    return a - b
    # Importar el módulo 'matematica'
```

```
# main.py
# import matematica
##Se comenta esta linea para la explicación, el archivo de modulo se encuentran en la misma ca

# Usar las funciones del módulo
resultado_suma = matematica.suma(10, 5)
resultado_resta = matematica.resta(10, 5)

print("Resultado de la suma:", resultado_suma)
print("Resultado de la resta:", resultado_resta)
```

1.8.2 Descripción del Código:

matematica.py define dos funciones (suma y resta) que realizan operaciones matemáticas básicas main.py importa el módulo matematica y utiliza sus funciones para realizar cálculos, demostrando cómo reutilizar código.

[47]:
```
# Importar el módulo 'matematica'
import matematica

# Usar las funciones del módulo
resultado_suma = matematica.suma(10, 5)
resultado_resta = matematica.resta(10, 5)

print("Resultado de la suma:", resultado_suma)
print("Resultado de la resta:", resultado_resta)
```

```
Resultado de la suma: 15
Resultado de la resta: 5
```

1.9 3. Estructuras de Control en Python

Las estructuras de control en Python permiten dirigir el flujo de ejecución del código, permitiendo que los programas tomen decisiones, repitan acciones y manejen diversas condiciones. En Python, las estructuras de control principales son las sentencias condicionales (if, elif, else) y los bucles (for, while).

1.9.1 Estructuras Condicionales

Las estructuras condicionales permiten ejecutar diferentes bloques de código según ciertas condiciones. La estructura básica en Python es la sentencia if, complementada por elif (else if) y else para manejar múltiples casos.

1.9.2 Etapas Propuestas para Usar Estructuras Condicionales:

Definir la Condición: Usar if para definir la primera condición. La condición debe ser una expresión que se evalúe a True o False.

Agregar Condiciones Alternativas (Opcional): Usar elif para agregar más condiciones que se evalúan si la condición del if es False.

Agregar una Condición por Defecto (Opcional): Usar else para manejar todos los casos que no cumplen ninguna de las condiciones anteriores.

1.9.3 Ejemplo de Estructuras Condicionales

```
# Definir una variable de edad
edad = 20

# Estructura condicional para determinar si la persona es mayor de edad
if edad >= 18:
    print("Eres mayor de edad.")
elif edad >= 13:
    print("Eres un adolescente.")
else:
    print("Eres un niño.")
```

1.9.4 Descripción del Código:

La sentencia if verifica si la edad es mayor o igual a 18. Si es True, se imprime "Eres mayor de edad." Si la primera condición no es True, elif verifica si la edad es mayor o igual a 13. Si es True, se imprime "Eres un adolescente." Si ninguna de las condiciones anteriores es True, else imprime "Eres un niño." Estructuras de Repetición (Bucles) Las estructuras de repetición permiten ejecutar un bloque de código varias veces. Los bucles más comunes en Python son for y while.

1.9.5 Etapas Propuestas para Usar Estructuras de Repetición:

Definir el Bucle: Usar for para iterar sobre una secuencia (como una lista o un rango) o while para repetir mientras una condición sea True.

Escribir el Cuerpo del Bucle: El bloque de código dentro del bucle se ejecutará en cada iteración.

Manejar la Terminación (si es necesario): En un bucle while, asegurarse de que la condición eventualmente se vuelva False para evitar un bucle infinito.

1.9.6 Ejemplo de Estructuras de Repetición

Bucle for

```
# Imprimir los números del 1 al 5 usando un bucle for
for i in range(1, 6):
    print(i)
```

1.9.7 Descripción del Código:

range(1, 6) genera una secuencia de números del 1 al 5. El bucle for itera sobre esta secuencia, imprimiendo cada número. ### Bucle while

```
# Imprimir los números del 1 al 5 usando un bucle while
contador = 1
while contador <= 5:
    print(contador)
    contador += 1  # Incrementar el contador para evitar un bucle infinito
### Descripción del Código:
```

El bucle **while** ejecuta el bloque de código mientras contador sea menor o igual a 5.
El valor de contador se incrementa en cada iteración para eventualmente terminar el bucle.

[60]:
```
# Definir una variable de edad
edad = 20

# Estructura condicional para determinar si la persona es mayor de edad
if edad >= 18:
    print("Eres mayor de edad.")
elif edad >= 13:
    print("Eres un adolescente.")
else:
    print("Eres un niño.")

print("--------")  #Salto de lunea
# Imprimir los números del 1 al 5 usando un bucle for
for i in range(1, 6):
    print(i)

print("--------")  #Salto de lunea
# Imprimir los números del 1 al 5 usando un bucle while
contador = 1
while contador <= 5:
    print(contador)
    contador += 1  # Incrementar el contador para evitar un bucle infinito
```

```
Eres mayor de edad.
--------
1
2
3
4
5
--------
1
2
3
4
5
```

[ ]: