Input

1. input() always returns a string (convert if needed).
2. Use .split() for multiple inputs.

In [27]:
```
name = input("Enter your name: ")
print("Hello, " + name + "!")
```

```
Hello, Taimur!
```

In [35]:
```
age = int(input("Enter your age: "))  # Converts input to an integer
print("You are", age, "years old.")
```

```
You are 12 years old.
```

In [29]:
```
height = float(input("Enter your height in meters: "))
print("Your height is:", height, "m")
```

```
Your height is: 5.6 m
```

In [30]:
```
# Multiple Inputs in One Line
a, b = input("Enter two numbers: ").split()  # Splits input by spaces
a, b = int(a), int(b)  # Convert to integers
print("Sum:", a + b)
```

```
Sum: 7
```

In [32]:
```
# List Input (Multiple Values)
numbers = list(map(int, input("Enter numbers: ").split()))
print("You entered:", numbers)
```

```
You entered: [1, 2, 3, 4, 5]
```

In [33]:
```
names = input("Enter names separated by commas: ").split(",")
print("Names:", names)
```

```
Names: ['Alice', 'Bob', 'Charlie']
```

Loop

Instead of writing the same code multiple times, loops allow us to execute a block of code multiple times with different values.

In [1]:
```
# Print Hello 6 times
print("Hello")
print("Hello")
print("Hello")
print("Hello")
print("Hello")
print("Hello")
```

```
Hello
Hello
Hello
Hello
Hello
Hello
```

In [3]:
```
for i in range(6):
    print("Hello")
```

```
Hello
Hello
Hello
Hello
Hello
Hello
```

For Loop

A for loop in Python is used to iterate over a sequence/Mapping (like a list, tuple, dictionary, string ...)

1. The loop picks each element from the sequence, one at a time.
2. The loop executes the block of code inside it.
3. It continues until all elements in the sequence are processed.

In [ ]:
```
# Basic Syntax
for variable in sequence:
    # Code to execute
```

Write for loop in C++ #include <iostream> using namespace std; int main() { for (int i = 0; i < 5; i++) { // Initialization, Condition, Increment cout << i << endl; } return 0; }

In [15]:
```
# Using range()
for i in range(5):  # range(5) generates numbers 0 to 4
    print(i)
```

```
0
1
2
3
4
```

1. range(5) generates numbers from 0 to 4 (excluding 5).
2. Each number is assigned to i in each iteration.

range(start, stop, step) handles loop control internally.

In [12]:
```
for i in range(2, 10, 2):  # Start=2, End=10 (exclusive), Step=2
    print(i, end=' ')
```

```
2 4 6 8
```

In [6]:
```
# Iterating Over a List
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
# The loop picks one item (fruit) from the list of fruits in each iteration.
```

```
apple
banana
cherry
```

In [14]:
```
# Looping Through a String
word = "Python"
for letter in word:
    print(letter)
```

```
P
y
t
h
o
n
```

In [1]:
```
my_tuple = ("apple", "banana", "cherry")

for item in my_tuple:
    print(item)
```

```
apple
banana
cherry
```

In [3]:
```
my_tuple = ("red", "green", "blue")

for i in range(len(my_tuple)):
    print("Index", i, "has value", my_tuple[i])
```

```
Index 0 has value red
Index 1 has value green
Index 2 has value blue
```

#Looping Over a Dictionary person = {"name": "Alice", "age": 25, "city": "New York"} for key in person.keys(): print(key) for value in person.values(): print(value) for key, value in person.items(): print(key, ":", value)

In [20]:
```
# Nested for Loop
for i in range(3):  # Outer Loop
    for j in range(2):  # Inner Loop
        print(f"i={i}, j={j}")
```

```
i=0, j=0
i=0, j=1
i=1, j=0
i=1, j=1
i=2, j=0
i=2, j=1
```

While Loop

A while loop is used when we want to repeat a block of code as long as a condition is True

1. The condition is checked before each iteration.
2. If the condition is True, the code inside the loop executes.
3. If the condition becomes False, the loop stops.

Basic Syntax while condition: # Code to execute

In [24]:
```
# Basic while Loop
x = 1
while x <= 5:  # Loop runs as long as x is 5 or less
    print(x)
    x += 1  # Increment x
```

```
1
2
3
4
5
```

In [25]:
```
x = 5
while x > 0:
    print("Counting down:", x)
    x -= 1
print("Finished!")
```

```
Counting down: 5
Counting down: 4
Counting down: 3
Counting down: 2
Counting down: 1
Finished!
```

1. x starts at 1.
2. The condition x <= 5 is checked.
3. The loop prints x and then increases x by 1.
4. When x becomes 6, the condition fails, and the loop stops.

Infinite Loop (Be Careful!) # If the condition never becomes False, the loop will run forever. x = 1 while x > 0: # Condition is always True print(x)

Break Statement

The break statement stops the loop immediately when executed.

In [36]:
```
for i in range(1, 6):
    if i == 4:
        break  # Stops the loop when i is 4
    print(i)
```

```
1
2
3
```

In [37]:
```
x = 1
while x <= 5:
    if x == 3:
        break  # Stops when x is 3
    print(x)
    x += 1
```

```
1
2
```

Continue Statement

The continue statement skips the current iteration and moves to the next one.

In [38]:
```
for i in range(1, 6):
    if i == 3:
        continue  # Skips when i is 3
    print(i)
```

```
1
2
4
5
```

In [39]:
```
for i in range(1, 6):
    if i % 2 == 0:
        continue
    print(i)
```

```
1
3
5
```

Arithmetic Operations

In [40]:
```
# Examples of Arithmetic Operations
a = 10
b = 3

print("Addition:", a + b)       # 13
print("Subtraction:", a - b)    # 7
print("Multiplication:", a * b) # 30
print("Division:", a / b)       # 3.3333
print("Floor Division:", a // b) # 3
print("Modulus:", a % b)        # 1
print("Exponentiation:", a ** b) # 1000
```

```
Addition: 13
Subtraction: 7
Multiplication: 30
Division: 3.3333333333333335
Floor Division: 3
Modulus: 1
Exponentiation: 1000
```

1. / always returns a float (3.3333).
2. // returns the integer part of division (3).
3. % gives the remainder of division (1).
4. ** raises a to the power of b (10^3 = 1000).

In [41]:
```
# Division by Zero
print(10 / 0)  # ZeroDivisionError
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[41], line 2
      1 # Division by Zero
----> 2 print(10 / 0)

ZeroDivisionError: division by zero
```

In [42]:
```
# Negative Exponent
print(2 ** -3)  # 0.125 (1/8)
```

```
0.125
```

Operator Precedence

PEMDAS Rule: P → Parentheses E → Exponents MD → Multiplication & Division (from left to right) AS → Addition & Subtraction (from left to right)

In [1]:
```
# Parentheses Have the Highest Precedence
print(10 + 5 * 2)    # 20 (Multiplication first, then addition)
print((10 + 5) * 2)  # 30 (Parentheses first)
```

```
20
30
```

In [2]:
```
# Multiplication and Division Before Addition and Subtraction
print(10 + 3 * 2)  # 16 (Multiplication first, then addition)
print(10 / 2 - 3)  # 2.0 (Division first, then subtraction)
```

```
16
2.0
```

In [4]:
```
# Logical Operator Precedence (not > and > or)
print(True or False and False)  # True (AND evaluated first: False and False = Fals
print(not True and False)       # False (NOT first, then AND)
# (not has higher precedence, so not True → False, then False and False → False)
```

```
True
False
```

In [5]:
```
print(3 + 5 * 2 ** 2)
```

```
23
```

In [6]:
```
result = (2 + 3) * 2 ** 2
print(result)
```

```
20
```

Explanation: Parentheses: 2 + 3 = 5 Exponent: 2 ** 2 = 4 Multiplication: 5 * 4 = 20 Output: 20

Math Function

In [10]:
```python
import math

print(math.sqrt(16))     # Square root → 4.0
print(math.pow(2, 3))    # Power → 8.0
print(math.floor(3.7))   # Floor → 3
print(math.ceil(3.1))    # Ceil → 4
print(math.pi)           # Pi → 3.141592...
```

```
4.0
8.0
3
4
3.141592653589793
```

In [ ]: