Loops

Loops are used to repeat a block of code multiple times.

1. Purpose and Working of Loops

- **Purpose:** *To automate repetitive tasks and reduce code redundancy.*
- **Working:** A loop executes a block of code repeatedly until a specified condition is met.

2. While Loop

Executes a block of code as long as the condition is true.

```
while condition:
    # Code to execute
Example:
i = 1
while i <= 5:
    print(i)
    i += 1
```

3. For Loop

Iterates over a sequence (e.g., list, string, range) and executes a block of code for each item.

```
for item in sequence:
    # Code to execute
Example:
for i in range(1, 6):
    print(i)
```

4. Nested Loops

A loop inside another loop.

```
for i in range(1, 4):
    for j in range(1, 4):
        print(i, j)
```

5. Break and Continue

Break*: Exits the loop immediately.*

```
for i in range(1, 6):
    if i == 3:
        break
    print(i)
```

Continue*: Skips the current iteration and moves to the next.*

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

Functions

Functions are reusable blocks of code that perform a specific task.

Parts of a Function

- ***Function Definition****: Defines the function using the def keyword.*
- ***Function Call****: Executes the function.*
- ***Parameters****: Variables passed to the function.*
- ***Return Value****: Value returned by the function.*

Execution of a Function

```
def greet(name):
    return f"Hello, {name}!"


print(greet("Alice"))
```

Keyword and Default Arguments

Keyword Arguments*: Pass arguments by parameter name.*

```
def greet(name, message):
    return f"{message}, {name}!"


print(greet(message="Hi", name="Bob"))
```

Default Arguments*: Provide default values for parameters.*

```python
def greet(name, message="Hello"):
    return f"{message}, {name}!"


print(greet("Alice"))
```

Scope Rules

Local Scope*: Variables defined inside a function.*

Global Scope*: Variables defined outside a function.*

```python
x = 10  # Global variable
def func():
    y = 5  # Local variable
    print(x + y)
func()
```

Strings

Strings are sequences of characters.

Length of a String

Use the len() function.

```
s = "Hello"
print(len(s))  # Output: 5
```

Concatenation and Repeat Operations

***Concatenation**: Combine strings using +.*

```
s1 = "Hello"
s2 = "World"
print(s1 + " " + s2)
# Output: Hello World
```

***Repeat**: Repeat a string using *.*

```
s = "Hi"
print(s * 3)  # Output: HiHiHi
```

Indexing and Slicing

***Indexing**: Access individual characters using indices.*

```
s = "Python"
print(s[0])  # Output: P
```

Slicing: *Extract a substring using slicing.*

```
s = "Python"

print(s[1:4])

# Output: yth
```

Python Data Structures

Tuples

Immutable sequences of elements.

```
t = (1, 2, 3)
print(t[0])  # Output: 1
```

Unpacking Sequences

Assign elements of a sequence to variables.

```
t = (1, 2, 3)
a, b, c = t
print(a, b, c)  # Output: 1 2 3
```

Lists

Mutable sequences of elements.

```
l = [1, 2, 3]
l.append(4)
print(l)  # Output: [1, 2, 3, 4]
```

Mutable Sequences

Lists can be modified after creation.

```
l = [1, 2, 3]
l[0] = 10
print(l)  # Output: [10, 2, 3]
```

List Comprehension

Concise way to create lists.

```
squares = [x**2 for x in range(1, 6)]

print(squares)  # Output: [1, 4, 9, 16, 25]
```

Sets

Unordered collections of unique elements.

```
s = {1, 2, 2, 3}

print(s)  # Output: {1, 2, 3}
```

Dictionaries

Collections of key-value pairs.

```
d = {"name": "Alice", "age": 25}

print(d["name"])  # Output: Alice
```