Python Functions

1. Python - Functions

A function is a reusable block of code that performs a specific task. Functions help in modularizing the code and avoiding repetition.

Syntax:

```
def function_name(parameters):
    """docstring (optional)"""
    # body of function
    return value (optional)
```

Example:

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

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

Notes:

- Functions can return multiple values as tuples.
- Use `pass` when defining an empty function.

2. Python - Default Arguments

You can provide default values for parameters. If the argument is not provided, the default is used.

Example:

```
def greet(name="Guest"):
    print(f"Hello, {name}")

greet()           # Output: Hello, Guest
greet("Sohan")    # Output: Hello, Sohan
```

Notes:

- Default arguments must follow non-default ones.

3. Python - Keyword Arguments

Allows function calls using parameter names, enhancing readability and flexibility.

Example:

```
def student(name, age):
    print(f"Name: {name}, Age: {age}")

student(age=20, name="Amit")
```

Notes:

- Useful for functions with many optional parameters.

4. Python - Keyword-Only Arguments

Parameters after a `*` must be passed using keyword arguments.

Example:

```
def student(name, *, age):
    print(f"Name: {name}, Age: {age}")

student("Rahul", age=21)
```

Notes:

- Improves code readability and reduces ambiguity.

5. Python - Positional Arguments

These must be passed in the correct positional order.

Example:

```
def location(city, country):
    print(f"{city} is in {country}")

location("Paris", "France")
```

Notes:

- Common in simple, small functions.

6. Python - Positional-Only Arguments

Parameters before `/` can only be passed positionally.

Example:

```python
def add(x, y, /):
    return x + y

print(add(3, 4))
```

Notes:

- Introduced in Python 3.8
- Helps avoid unwanted keyword usage.

7. Python - Arbitrary Arguments

Used when you do not know the number of arguments beforehand.

*args (Non-keyword arguments):

```python
def total(*numbers):
    return sum(numbers)

print(total(1, 2, 3))
```

**kwargs (Keyword arguments):

```python
def show_details(**info):
    for key, value in info.items():
        print(f"{key}: {value}")

show_details(name="Amit", age=25)
```

Notes:

- `args` is a tuple, `kwargs` is a dictionary.
- Can be combined: `def func(a, *args, **kwargs):`

8. Python - Variable Scope

Scope defines the visibility and lifetime of a variable.

- **Local Scope**: Variables declared inside a function.
- **Global Scope**: Declared outside any function.
- **Nonlocal Scope**: Declared in an enclosing function.

Example:

```python
x = 10  # global

def func():
    x = 20  # local
    print(x)

func()
print(x)
```

```python
# nonlocal example

def outer():
    x = "outer"
    def inner():
        nonlocal x
        x = "inner"
    inner()
    print(x)

outer()
```

Notes:

- Use `global` keyword to modify global variables inside a function.
- Use `nonlocal` to modify variables from the outer (enclosing) function scope.

9. Python - Function Annotations

Used to attach metadata to function parameters and return types.

Example:

```python
def greet(name: str) -> str:
    return "Hello " + name

print(greet("Alice"))
print(greet.__annotations__)
```

Notes:

- Annotations are not enforced by Python.
- Useful for static type checkers, IDEs, documentation.

10. Python - Modules

Modules are Python files containing functions, classes, or variables. You can import and reuse them.

Example:

math_utils.py

```python
def add(a, b):
    return a + b
```

main.py

```python
import math_utils
print(math_utils.add(3, 4))
```

Creating and Using Modules:

```
# my_module.py
PI = 3.14

def area_circle(radius):
    return PI * radius * radius
```

```
# use_module.py
import my_module
print(my_module.area_circle(5))
```

Notes:

- Use `from module import function` for direct usage.
- Use `__name__ == "__main__"` to prevent code from running on import.

11. Python - Built-in Functions

Python provides many built-in functions for common tasks.

Examples:

```
print(len("hello"))       # Output: 5
print(type(123))          # Output: <class 'int'>
print(sum([1, 2, 3]))      # Output: 6
print(sorted([3, 1, 2]))   # Output: [1, 2, 3]
```

Common Built-in Functions:

- `print()` : Outputs to the screen.
- `len()` : Returns length.
- `type()` : Returns type of object.
- `input()` : Accepts user input.
- `sum()` : Sums items.
- `sorted()` : Returns sorted list.
- `abs()` : Absolute value.
- `max()` , `min()`
- `range()` : Generates a sequence of numbers.
- `enumerate()` : Adds index to iterable.
- `zip()` : Aggregates elements from multiple iterables.
- `map()` : Applies function to all items.
- `filter()` : Filters items using a function.

Example of map, filter, zip:

```
nums = [1, 2, 3, 4]
print(list(map(lambda x: x*x, nums)))  # [1, 4, 9, 16]
print(list(filter(lambda x: x%2 == 0, nums)))  # [2, 4]

names = ["Alice", "Bob"]
```

```
scores = [85, 90]
print(list(zip(names, scores)))  # [('Alice', 85), ('Bob', 90)]
```