Functions in Python

A **function** is a block of reusable code that performs a specific task. Functions make the code **organized, readable, and reusable**.

1. Types of Functions

There are **two main types** of functions in Python:

1. **Built-in Functions** – Predefined in Python (e.g., print(), len(), type()).
2. **User-defined Functions** – Created by the user to perform specific tasks.

2. Defining a Function

A function is defined using the def keyword.

Syntax:

```
def function_name(parameters):
    # Function body
    return value  # (optional)
```

Example:

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

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

Note:

- The return statement is **optional**.
- Functions **execute only when called**.

3. Function Parameters & Arguments

- **Parameters**: Variables listed inside the function definition.
- **Arguments**: Values passed to the function when calling it.

Types of Arguments:

1. **Positional Arguments** – Passed in order.
2. **Keyword Arguments** – Passed with parameter names.
3. **Default Arguments** – Default value is used if no argument is given.
4. **Variable-length Arguments** – *args (multiple positional arguments), **kwargs (multiple keyword arguments).

Example:

```
def describe_pet(animal="dog", name="Buddy"):
    print(f"I have a {animal} named {name}.")


describe_pet("cat", "Whiskers")
describe_pet(name="Max", animal="rabbit")
describe_pet()
```

Note:

- describe_pet() uses the default values (dog and Buddy).
- Order matters for **positional arguments**, but not for **keyword arguments**.

4. Return Statement

A function can return a value using return.

Example:

```
def square(num):
    return num * num

print(square(5))  # Output: 25
```

Note:

- The return statement **stops** function execution and sends back a result.

5. Lambda (Anonymous) Functions

A **lambda function** is a small function with **no name**.

Syntax:
lambda arguments: expression

Example:

```
square = lambda x: x * x
print(square(4))

add = lambda a, b: a + b
print(add(3, 5))
```

Note:

- Used for **short, simple** functions.
- Cannot have multiple statements.

6. *args and **kwargs

- *args allows passing multiple **positional** arguments.
- **kwargs allows passing multiple **keyword** arguments.

Example:

```
def add_numbers(*args):
    return sum(args)

print(add_numbers(1, 2, 3, 4))
```

```
def print_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_info(name="Alice", age=25, city="NY")
```

Note:

- *args collects values as a tuple ((1, 2, 3, 4)).
- **kwargs collects key-value pairs as a dictionary ({"name": "Alice", "age": 25, "city": "NY"}).

7. Scope & Lifetime of Variables

1. **Local Scope** – Variable inside a function (exists only inside).
2. **Global Scope** – Variable outside a function (exists throughout).
3. **Nonlocal** – Used in nested functions to modify outer function variables.

Example:

```
x = 10  # Global variable

def my_func():
    global x
    x = x + 5  # Modifies the global variable
    print(x)

my_func()  # Output: 15
```

Note:

- global allows modifying a global variable inside a function.
- Without global, x inside my_func() would be **local**.