Huawei HCCDA-AI certification

Trainer: Fawad Bahadur Marwat

5

Introduction to Function in Python

Definition

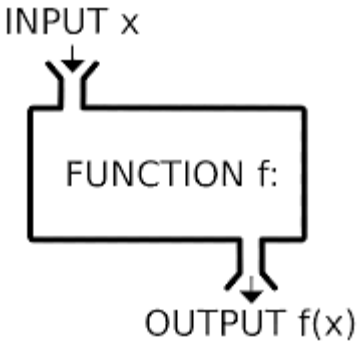
Named, reusable code blocks for specific tasks.

Purpose

- Organize code into manageable parts.
- Promote reusability, reduce redundancy.
- Enable multiple calls throughout a program.

Syntax

Use def keyword, function name, and parameters in ().



Example

```
def greet(name):
    print(f"Hello, {name}!")
greet("Alice")  # Output: Hello, Alice!
```

Explanation

greet function takes name parameter and prints a greeting.

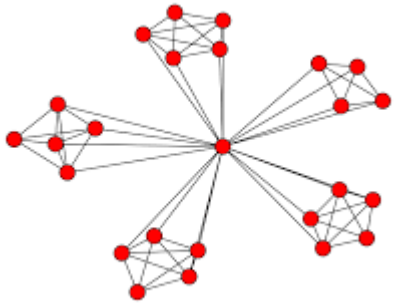
CORVIT
Reaching end to end

Trainer: Fawad Bahadur Marwat

Functions: Modularity & Reusability

Modularity

- Encapsulate functionality for organized, navigable code.
- Develop, test, and debug functions independently.
- Enhances software quality and maintainability.



Code Reusability

- Reuse functions across programs or sections.
- Eliminates redundant code.
- Keeps codebase clean and maintainable.

CORVIT
Reaching end to end

Trainer: Fawad Bahadur Marwat

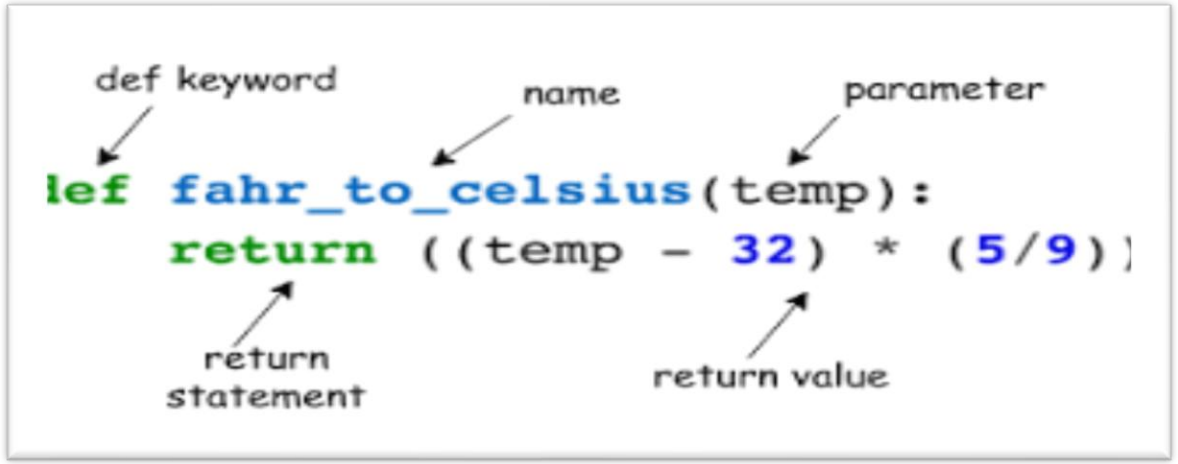
Function Definition in Python

Syntax

Use def keyword, function name, and () for parameters.
Parameters (optional) accept inputs for the function.

Purpose

- Define reusable code to perform specific tasks.
- Return results using return (optional).



Example

```
def add(a, b):
    return a + b
print(add(3, 4))  # Output: 7
```

Explanation

- add function takes parameters a and b.
- Returns their sum using return.

Trainer: Fawad Bahadur Marwat

Executing Functions in Python

Function Execution

Runs when called, executing the function's code body.

Calling Syntax

- Use function name followed by ().
- Include arguments in () if required.

Multiple Calls

Reuse functions by calling them multiple times.

Example


```
def greet(name):
    print(f"Hello, {name}!")
greet("Alice")    # Output: Hello, Alice!
greet("Bob")      # Output: Hello, Bob!
greet("Charlie")  # Output: Hello, Charlie!
```

Explanation

greet function is called multiple times with different arguments.

CORVIT
Reaching end to end

Trainer: Fawad Bahadur Marwat

Function Arguments in Python

Definition

Values (arguments) passed to a function for processing.
Used as inputs for calculations or manipulations.

Importance

Enable generic, flexible functions.
Avoid hardcoding, work with varied data.

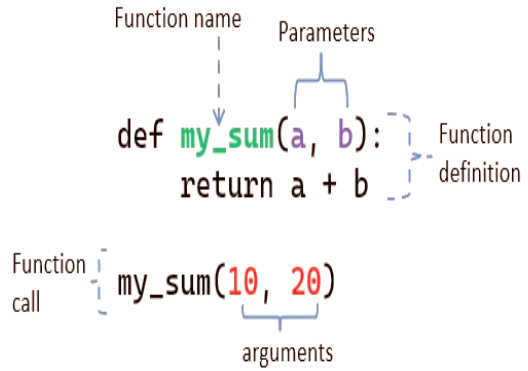
Benefits

Dynamic functionality with different inputs.
Improves code readability and maintainability.

Example

Functions

```
def multiply(a, b):
    return a * b
result = multiply(4, 5)
print(result)  # Output: 20
```



Trainer: Fawad Bahadur Marwat

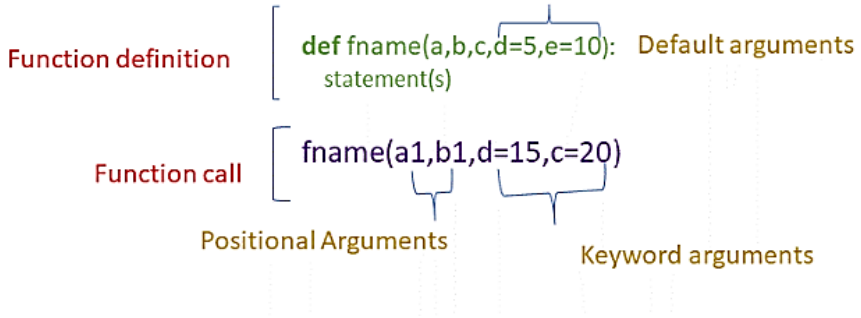
Positional Arguments in Python

Definition

Arguments passed in the order of function parameters.
First argument matches first parameter, and so on.

Key Point

Order matters; incorrect order causes logical errors.



Example

Functions

```
def divide(numerator, denominator):
    if denominator == 0:
        return "Error: Cannot divide by zero."
    return numerator / denominator
result1 = divide(10, 2)  # Output: 5.0
result2 = divide(2, 10)  # Output: 0.2
print(result1)
print(result2)
```

Explanation

Correct: divide(10, 2) → 10 / 2 = 5.0.
Incorrect order: divide(2, 10) → 2 / 10 = 0.2.

Trainer: Fawad Bahadur Marwat

Keyword Arguments in Python

Functions

Definition

Specify parameter names and values when calling a function.
Allows arguments to be passed in any order.

Example

```
def profile(name, age, city):
    return f"{name} is {age} years old and lives in {city}."
info = profile(age=30, name="Alice", city="New York")
print(info)  # Output: Alice is 30 years old and lives in New York.
```

Benefits

Clarity: Clearly indicates what each argument represents.
Flexibility: Order-independent, ideal for optional parameters.

```
def num(a,b):
    print(a,b)
num(b=3,a=5)
```

Keyword Arguments

Explanation

Arguments passed using parameter names, ignoring order.

Trainer: Fawad Bahadur Marwat

CORVIT
Reaching end to end

*args in Python

Definition

Allows a function to accept any number of positional arguments.
Arguments are collected as a tuple.

Benefits

Ideal when the number of arguments is unknown.

Example

Functions

```
def concatenate_strings(*args):
    return " ".join(args)
result = concatenate_strings("Hello", "world",
"from", "Python!")
print(result)  # Output: Hello world from Python!
```

Explanation

- *args gathers all arguments into a tuple.
- join() combines them into a single string.

```
def num(a,b):
    print(a,b)
num(b=3,a=5)
```

Keyword Arguments

CORVIT
Reaching end to end

Trainer: Fawad Bahadur Marwat

Global vs. Local Variables in Python

Global Variables

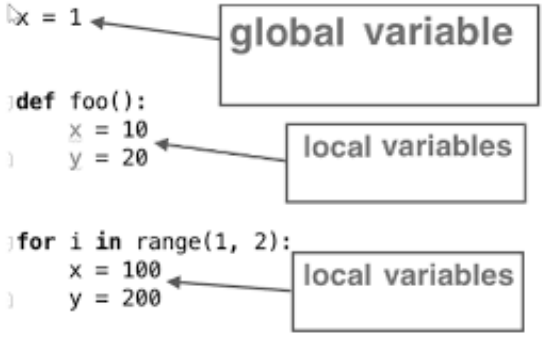
- Defined outside functions.
- Accessible anywhere in the code.

Local Variables

- Defined inside a function.
- Accessible only within that function's scope.

Example

```
global_var = 10  # Global variable
def function():
    local_var = 5  # Local variable
    print(global_var)  # Output: 10
function()
# print(local_var)  # Error: local_var is not accessible
```



Explanation

- global_var is accessible everywhere.
- local_var is limited to the function; accessing it outside raises an error.

CORVIT
Reaching end to end

Trainer: Fawad Bahadur Marwat