Python for Beginners: Functions in Python

Connect with me: Youtube | LinkedIn | WhatsApp Channel | Web | Facebook | Twitter

- Download PDF
- To access the updated lecture notes, please click on the following link: https://yasirbhutta.github.io/python/docs/functions.html

Want to Learn Python, Join our WhatsApp Channel ✨:

What is a Function?

In Python, a function is a block of code designed to perform a specific task. It's reusable, which means you can call it multiple times in your program. This helps to organize your code, make it more readable, and avoid repetition.

How to Write a Function

To define a function, you use the `def` keyword followed by the function name, parentheses for parameters, and a colon. The code block that defines the function is indented.

```
def function_name(parameters):
  # Function body
  # Code to be executed
```

Example:

```
def greet(name):
  print("Hello,", name + "!")

# Calling the function
greet("Ahmad")  # Output: Hello, Ahmad!
```

Explanation:

- `def greet(name):` defines a function named `greet` that takes one parameter, `name`.
- `print("Hello,", name + "!")` is the function body, which prints a greeting message using the provided name.
- `greet("Ahmad")` calls the function with the argument "Ahmad".

Key Points:

- **Parameters:** These are variables passed to the function when it's called.
- **Return Value:** A function can optionally return a value using the `return` statement.

- **Docstrings:** It's good practice to include a docstring (a string that explains the function's purpose) after the function definition.

Function with a Return Value:

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

result = add(3, 5)
print(result)  # Output: 8
```

Parameters and Arguments

Parameters are defined by the names that appear in a function definition, whereas arguments are the values actually passed to a function when calling it. Parameters define what kind of arguments a function can accept.

Parameters

- **Definition:** Variables declared in a function's definition.
- **Purpose:** Act as placeholders for values that will be passed to the function when it's called.
- **Location:** Inside the function's parentheses.

Arguments

- **Definition:** Actual values passed to a function when it's called.
- **Purpose:** Provide data for the function to work with.
- **Location:** Inside the function call parentheses.

See also the `FAQ` question of `Python Documentation` on the difference between arguments and parameters.

Example:

```
def greet(name):  # 'name' is a parameter
    print("Hello,", name + "!")

greet("Alice")  # "Alice" is an argument
```

In this example:

- `name` is a parameter in the function `greet`.
- `"Alice"` is an argument passed to the function when it's called.

To summarize:

- Parameters are defined *before* the function is called.
- Arguments are provided *when* the function is called.

Think of it like this:

- A parameter is like an empty box that expects a value.
- An argument is the value you put into the box.

More on Defining Functions

Default Argument Values

- Video: Learn How to Use Default Parameters in Function Definition

Example: Function with Default Parameters:**

```
def greet(name="World"):
  print("Hello,", name + "!")

greet()  # Output: Hello, World!
greet("Alice")  # Output: Hello, Alice!
```

Keyword Arguments

Special parameters

Positional-or-Keyword Arguments

Positional-Only Parameters

Keyword-Only Arguments

Arbitrary Argument Lists

In Python, Arbitrary Argument Lists allow a function to accept a varying number of arguments. This is useful when you don't know beforehand how many arguments might be passed to the function. There are two types of arbitrary arguments:

1. **Arbitrary Positional Arguments (`*args`)**
2. **Arbitrary Keyword Arguments (`**kwargs`)**
3. Arbitrary Positional Arguments (`*args`)

These allow a function to take any number of positional arguments. Inside the function, `*args` collects all the positional arguments as a tuple.

- Video: How to Use *args in Python Functions
- Video: Understanding *args in Functions - How to Add Any Number of Arguments with *args

Example:

```
def greet(*names):
    for name in names:
```

```
        print(f"Hello, {name}!")

greet("Ali", "Hamza", "Ahmad")
```

Output:

```
Hello, Ali!
Hello, Hamza!
Hello, Ahmad!
```

In this example, the `greet` function can take any number of names. The `*names` collects them into a tuple (`names`), which can be iterated over.

1. Arbitrary Keyword Arguments (`**kwargs`)

These allow a function to accept any number of keyword arguments (arguments passed as key-value pairs). Inside the function, `**kwargs` collects these as a dictionary.

- Video: How to use **kwargs in Python

Example:

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

print_info(name="Ali", age=25, city="Multan")
```

Output:

```
name: Ali
age: 25
city: Multan
```

In this case, the function accepts any number of keyword arguments and collects them into a dictionary (`info`), which you can then work with inside the function.

Combined Use

You can also use both `*args` and `**kwargs` in the same function to handle a combination of positional and keyword arguments.

Example:

```
def display_data(*args, **kwargs):
    print("Positional arguments:", args)
    print("Keyword arguments:", kwargs)

display_data(1, 2, 3, name="Ali", age=25)
```

Output:

```
Positional arguments: (1, 2, 3)
Keyword arguments: {'name': 'Ali', 'age': 25}
```

Key Points:

- `*args` collects all positional arguments into a tuple.
- `**kwargs` collects all keyword arguments into a dictionary.
- You can use both `*args` and `**kwargs` together to handle any type of arguments passed to a function.
-

Unpacking Argument Lists

Lambda Expressions

Documentation Strings

Function Annotations

See also:

- Python Quiz - Functions

True/False (Mark T for True and F for False)

Multiple Choice (Select the best answer)

1. What is the output of the following code?

```
def myfunction(val):
    return val % 4 == 0
print(myfunction (13) or myfunction (8))
```

- A) 0
- B) 13
- C) False
- D) True

- E) 3.5
- **Watch the Video Tutorial for the Answer:** https://youtu.be/laKpsLlq60I

2. **What is the output of the following code?** [Python Quiz #2]

```
def foo(x):
  if x == 1:
    return 1
  else:
    return x * foo(x - 1)

print(foo(5))
```

- A) 5
- B) 15
- C) 120
- D) None

Watch this video for answer: https://www.youtube.com/shorts/k50czTu7vao

For more details, see Appendix A

3. **What is the output of the following code?** [Python Quiz #30]

```
def calculate_sum(n):
    if n == 0:
        return 0
    else:
        return n + calculate_sum(n-1)

print(calculate_sum(4))
```

- A) 4
- B) 6
- C) 10
- D) 15

Watch the video for the answer: https://youtube.com/shorts/LQEfGgJYlT4?si=MDvSvVHiBc6hCJ0W

4. **What is the output of the following expression?** [Python Quiz #13]

```
def add(a,b,*parm):
    total = 0
    print(a+b)
    for n in parm:
        total += n
    return total
```

```
print(add(1, 2))
```

- A) 3 0
- B) 3
- C) 0
- D) Error

Watch this video for answer: https://youtube.com/shorts/k4KVCxU5oMg

5. **What is the output of the following code?** [Python Quiz #14]

```
def add(*args):
     print(type(args))

add(1, 2,8,9)
```

- A) set
- B) tuple
- C) list
- D) None

Watch this video for answer: https://youtube.com/shorts/VQT4CIlpf9M

6. **What is the output of the following code?** [#41 Python Quiz]

```
def display_data(**kwargs):
    print(type(kwargs))

display_data(name="Ali", age=25)
```

- A) <class 'set'>
- B) <class 'tuple'>
- C) <class 'list'>
- D) <class 'dict'>

Watch this video for answer: https://youtu.be/5IWmz7iWqUE?si=Wx0OeTwME3XEiL-h

7. **What will be the output of this code?**

```
def func(x, y=2):
    return x * y
print(func(3))
```

   - A) 2

- B) 6
- C) 3
- D) Error

Answer: B

8. **What is a function in Python?** [#42 Python Quiz]

   - A) A built-in tool that performs a specific operation.
   - B) A block of code that only executes when it is called.
   - C) A variable used to store data.
   - D) A loop structure for repetitive tasks.

9. **What is the main purpose of a function in Python?**

- A) To group a set of related code into a single unit
- B) To create a new type of data
- C) To write a program in a single line
- D) To change the value of global variables

8. **What is the purpose of the return statement in a function in Python?**

- A) To print the output of the function
- B) To exit the function and return a value
- C) To execute the function without returning anything
- D) To stop the function and start a new one

9. **What is the correct way to define a function in Python?**

- A) function my_function():
- B) def my_function():
- C) define my_function():
- D) my_function() {

11. **Which of the following is true about Python functions?**

- A) Functions are mandatory in Python programs.
- B) Functions can only return one value.
- C) Functions can return multiple values.
- D) A function must always take arguments.

Answer: C

12. **What happens if you don't include a return statement in a function?**

- A) The function will return None.
- B) The function will cause an error.
- C) The function will return 0.
- D) The function will return the last variable used.

Answer: A

Python Code Challenges

1. Write a Python program that takes two numbers as input and prints their sum.

- **Watch the Solution Now ✨**

2. **Exercise: Find the Maximum Value**

Task: Write a Python program that finds and prints the maximum value from a given list of numbers.

Sample Input:

```
numbers = [3, 7, 1, 9, 5]
```

Sample Output:

```
9
```

Instruction: please don't use the `max()` function to find the maximum value in a list.

- **Watch the Solution Now ✨**

3. **Problem Statement:** Write a Python function `find_length` that takes a string input `word` and returns the length of the word by counting the number of characters in it. You are not allowed to use the built-in `len()` function.

Function Signature:

```
def find_length(word: str) -> int:
```

Input:

- A string `word` which can contain letters, spaces, or special characters. **Output:**
- The function returns an integer representing the total number of characters in the input string. **Sample Input and Output: Input:**

```
find_length("python language")
```

Output:

```
15
```

- **Watch the Solution Now ✨**

Youtube@yasirbhutta

1. Write a function `sum3(num1,num3,num3)` that takes three numbers as input and returns the sum.
2. Write a function `SumNum(num1)` that takes a number as input and returns the sum of numbers from 1 to that number (num1).
3. Write a function `sumSquares(x)` that takes a vector of numbers as input and returns the sum of their squares.
4. Write a function `isEven(x)` that takes a number as input and returns true if it is even, and false otherwise.
5. Write a program with three functions:
6. **`isEven(n)`:** This function takes an integer `n` as input and returns `True` if `n` is even and `False` otherwise. You can use the modulo operator (`%`) to check for evenness.
7. **`printTable(n)`:** This function takes an integer `n` as input and prints its multiplication table. The table should show the product of `n` with each number from 1 to 10, formatted like `n * i = n * i`, where `i` is the current number in the loop.
8. **`main`:** The main program should:
    - Prompt the user to enter an integer.
    - Use the `isEven(n)` function to check if the entered number is even.
    - If the number is even, call the `printTable(n)` function to print its multiplication table.
    - If the number is odd, print a message indicating the number is odd and not eligible for printing a table.

Example output:

```
Enter an integer: 4
4 is even! Here's its multiplication table:
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
...
4 * 10 = 40
```

9. Write a function `avgPositive(data)` that takes a list of numbers as input and returns the average of all positive numbers in the list.

Review Questions

References and Bibliography

Which of the following will cause a syntax error due to incorrect indentation in Python?

A)

```
print("Hello World!")
```

B)

```
def my_function():
print("Hello World!")
```

C)

```
if x == 10:
    print("x is 10")
```

D)

```
x = 10
```

Answer: B

Appendices

Appendix A: Recursive program

- A recursive program is one that calls itself in order to solve a problem. In Python, this usually happens within a function where the function continues to call itself with a modified argument until a base condition is met.

In the example, the function `foo(x)` is a recursive function that calculates the factorial of `x`.

The code:

```
def foo(x):
    if x == 1:
        return 1
    else:
        return x * foo(x - 1)

print(foo(5))
```

Step-by-Step Explanation:

1. **Base Case**:
    - The function has a base case `if x == 1: return 1`. This stops the recursion. Without this base case, the function would keep calling itself indefinitely, leading to a "stack overflow" or "maximum recursion depth exceeded" error.
2. **Recursive Case**:
    - If `x` is not equal to `1`, the function returns `x * foo(x - 1)`. This is the recursive step, which calls `foo` again with `x - 1`.
3. **Example with `foo(5)`**: Let's break down the flow when you call `foo(5)`:
    - `foo(5)` checks if `x == 1`. Since `x = 5`, the base case is not satisfied, so the function returns `5 * foo(4)`.
    - Now, the function evaluates `foo(4)`. Again, `x == 1` is false, so the function returns `4 * foo(3)`.
    - Next, `foo(3)` is evaluated. It returns `3 * foo(2)`.
    - Then, `foo(2)` returns `2 * foo(1)`.
    - Finally, `foo(1)` hits the base case and returns `1`.

    Now, the recursive calls start to resolve from the deepest level:

    - `foo(2)` returns `2 * 1 = 2`
    - `foo(3)` returns `3 * 2 = 6`
    - `foo(4)` returns `4 * 6 = 24`
    - `foo(5)` returns `5 * 24 = 120`
4. **Output**: The result of `foo(5)` is `120`, which is the factorial of 5. Hence, `print(foo(5))` will output `120`.

Conclusion:

This is a classic example of recursion being used to calculate the factorial of a number. The function continues to break down the problem (finding factorial of smaller numbers) until it hits the simplest case (`x == 1`), after which it multiplies the results together to get the final answer.