Chapter 6: 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

🎬 **YouTube Playlists to Learn Python:**

- 🔗 Python Tutorials for Beginners
- 🔗 Python Code Challenges | Quiz
- 🔗 Python Exercises

Want to Learn Python, Join our WhatsApp Channel ✨:

<<<<<< HEAD

What is a Function?

=======

> "The only way to do great work is to love what you do."
> – Steve Jobs

6.1 What is a Function?

ef27128a4decf2953997546be3fe2a1a9231c939

A function is a block of reusable code that performs 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.

**Why Do We Use Functions? **

We use functions in Python for several reasons:

- **Code Reusability:** You can call a function multiple times instead of repeating code. This saves time and effort.
- **Modularity:** Breaking down a large program into smaller, manageable chunks (functions) makes it easier to understand and maintain.
- **Avoiding Repetition:** Functions prevent you from writing the same code over and over, reducing errors and improving efficiency."

6.2 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.

Syntax:

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

Example 6.1: Defining and Calling a Function

```
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.

Example 6.2: Function with a Return Value

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

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

6.3 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 6.3: Defining a Function with Parameters and Passing Arguments

```python
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.

Sure! Here's a simple task for beginners to practice writing functions in Python, along with input and output examples.

Task: Create a Function to Calculate the Area of a Rectangle

Function Requirements:

1. Define a function named `calculate_area` that takes two parameters: `length` and `width`.
2. The function should calculate the area of the rectangle (Area = Length × Width) and return the result.

Input:

- Length (a positive float or integer)
- Width (a positive float or integer)

Output:

- The area of the rectangle (a float)

Expected Output

```
The area of the rectangle with length 5 and width 3 is: 15
```

Additional Test Cases

1. **Input:** length = 7, width = 2

   - **Output:** The area of the rectangle with length 7 and width 2 is: 14

2. **Input:** length = 10.5, width = 4.2

   - **Output:** The area of the rectangle with length 10.5 and width 4.2 is: 44.1

Absolutely! Here's another beginner-friendly task.

Task: Create a Function to Check if a Number is Even or Odd

Function Requirements:

1. Define a function named is_even that takes one parameter: number.
2. The function should determine if the number is even or odd.
3. It should return the string "Even" if the number is even, and "Odd" if the number is odd.

Input:

- A single integer (positive or negative)

Output:

- A string: either "Even" or "Odd"

Example

```python
def is_even(number):
    # Check if the number is even or odd
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"

# Example usage:
num = 4
result = is_even(num)
print(f"The number {num} is: {result}")
```

Expected Output

```
The number 4 is: Even
```

Additional Test Cases

Encourage beginners to test the function with various numbers:

1. **Input:** num = 7

   ○ **Output:** The number 7 is: Odd

2. **Input:** num = -2

   ○ **Output:** The number -2 is: Even

3. **Input:** num = 0

   ○ **Output:** The number 0 is: Even

This task helps beginners understand conditional statements and how to return different outputs based on input in Python.

Sure! Here's a task that focuses on building logic through a simple number guessing game.

Task: Create a Number Guessing Game

Function Requirements:

1. Define a function named `guess_number` that takes no parameters.
2. The function should randomly select a number between 1 and 100.
3. Prompt the user to guess the number, providing feedback on whether their guess is too high, too low, or correct.
4. The game should continue until the user guesses the correct number.
5. Once the user guesses correctly, the function should print a congratulatory message and the number of attempts it took.

Input:

- User input (guesses) from the console

Output:

- Feedback on each guess and a congratulatory message upon a correct guess

Expected Output

When the user plays the game, the interaction might look like this:

```
Welcome to the Number Guessing Game!
Guess a number between 1 and 100.
Enter your guess: 50
Too low! Try again.
Enter your guess: 75
Too high! Try again.
Enter your guess: 60
Congratulations! You've guessed the number 60 in 3 attempts.
```

Notes for Beginners

1. **Random Number Generation:** You can use the `random` module to select a random number.
2. **Input Handling:** Use `input()` to get the user's guess and convert it to an integer.
3. **Loops and Conditionals:** This task will help practice loops for continuous guessing and conditionals for feedback.

Task: Create a Function to Find the Maximum Number in a List

Function Requirements:

1. Define a function named `find_max` that takes one parameter: `numbers`, which is a list of integers.
2. The function should return the maximum number in the list.
3. If the list is empty, the function should return `None`.

Input:

- A list of integers

Output:

- The maximum integer in the list or `None` if the list is empty

Expected Output

```
The maximum number in the list is: 9
```

Additional Test Cases

1. **Input:** `numbers = [10, 20, 30, 5]`

   - **Output:** `The maximum number in the list is: 30`

2. **Input:** `numbers = [-5, -1, -10]`

   - **Output:** `The maximum number in the list is: -1`

3. **Input:** `numbers = []`

   - **Output:** `The maximum number in the list is: None`

6.4 More on Defining Functions

6.4.1 Default Argument Values

- [Video: Learn How to Use Default Parameters in Function Definition](https://yasirbhutta.github.io/)

Example 6.4: Function with Default Parameters

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

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

6.4.2 Keyword Arguments

6.4.3 Special parameters

6.4.3.1 Positional-or-Keyword Arguments

6.4.3.2 Positional-Only Parameters

6.4.3.3 Keyword-Only Arguments

6.4.3.4 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:

```python
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:

```python
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:

```python
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.
-

video: Guard Statements in Python: Explained Simply

Unpacking Argument Lists

Lambda Expressions

Documentation Strings

Function Annotations

'nonlocal' keyword

In Python, the `nonlocal` keyword is used to declare that a variable inside a nested function refers to a variable in the nearest enclosing scope that is not global. This allows you to modify a variable from an outer (but not global) scope within a nested function.

Here's an example to illustrate how `nonlocal` works:

```python
def outer_function():
    x = 10  # This is the enclosing variable

    def inner_function():
        nonlocal x  # Declare that we want to use the outer variable x
        x += 5      # Modify the outer variable
        print("Inner x:", x)

    inner_function()
    print("Outer x:", x)

outer_function()
```

Output:

```
Inner x: 15
Outer x: 15
```

Explanation:

1. `outer_function` defines a variable `x`.
2. `inner_function` modifies `x` using the `nonlocal` keyword.

3. When `inner_function` is called, it updates `x`, and both the inner and outer prints show the updated value.

When to Use `nonlocal`:

- When you have nested functions and you want to modify a variable from the outer function.
- When you need to avoid using global variables and want to keep your code cleaner and more modular.

If you don't use `nonlocal`, Python will treat the variable as a new local variable in the inner function, which can lead to unexpected behavior or errors.

See also:

- [Python Quiz - Functions](#)

Fix the Errors

Assigning a value to a function (functions can't be assigned to variables)

```python
def greet():
    print("Hello World!")

greeting = greet
```

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?

```python
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

7. **What is the output of the following code?** Python Quiz #88

```python
def greet(name="User"):
    return "Hello, " + name
print(greet("Ahmad"))
```

A) `Hello, User`
B) `Hello, Ahmad`
C) `Hello`
D) `Error`

17. **What is the output of the following code?** Python Quiz #89

```python
def my_function():
    pass
print(my_function())
```

- A) `None`
- B) `0`
- C) `True`
- D) `Error`

20. **What is the output of the following code?** Python Quiz #90

```python
def my_func(a, b=2, c=3):
    return a + b + c
print(my_func(5, c=4))
```

- A) `11`
- B) `12`
- C) `10`
- D) `Error`

23. **Which of the following function calls is invalid for this function definition?** [Python Quiz #93]

```python
def my_func(a, b, c=3):
    return a + b + c
```

- A) `my_func(1, 2)`
- B) `my_func(1, 2, 4)`
- C) `my_func(a=1, b=2, c=5)`
- D) `my_func(1, c=4, b=2, 5)`

`<<<<<< HEAD`

25. **What is the output of the following code?** [Python Quiz #91]

```python
def change_value(x):
    x = 10
num = 5
change_value(num)
print(num)
```

- A) 5
- B) 10
- C) Error
- D) None

Answer: a) 5

Scope and Variables

41. **What is the output of the following code?** [Python Quiz #92]

```python
x = 5
def my_func():
    global x
    x = 10
my_func()
print(x)
```

a) 5

b) 10

c) None

d) Error

26. **What is the output of the following code?** [Python Quiz #93] =======

25. **What is the output of the following code?** [Python Quiz #91]

```python
def change_value(x):
    x = 10
```

```
num = 5
change_value(num)
print(num)
```

- A) `5`
- B) `10`
- C) `Error`
- D) `None`

41. What is the output of the following code? [Python Code #92]

```python
def my_func():
    global x
    x = 10

x = 5
my_func()
print(x)
```

- A) `5`
- B) `10`
- C) `None`
- D) `Error`

ef27128a4decf2953997546be3fe2a1a9231c939

```python
def outer():
    x = 1
    def inner():
        print(x)
    return inner

func = outer()
func()
```

- A) None
- B) Error
- C) 1
- D) Function object

27. What will be the output of the following code? [Python Quiz #95]

```python
def outer():
    x = 5
    def inner():
        nonlocal x
        x = 10
    inner()
    return x
print(outer())
```

- A) 5
- B) 10
- C) None
- D) Error

What is the output of the following code? [Python Quiz #96]

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

result = greet(5)
print(result)
```

```
 - A) Hello, 5!
 - B) TypeError
 - C) None
 - D) Hello, !
```

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

```python
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

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

```python
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]

```python
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]

```python
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

30. **What is the output of the following code?** [Python Quiz #97]

```python
def f(a, b, *args):
    return len(args)
print(f(1, 2, 3, 4, 5))
```

- A) 2
- B) 3
- C) 5
- D) None

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

```python
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?** [Python Quiz #87]

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

- A) 2
- B) 6
- C) 3
- D) Error

What is the output of the following code? [Python Quiz #98]

```python
def outer_function(message):
    def inner_function():
```

```
        print(message)
    return inner_function

my_function = outer_function("Hello, world!")
my_function()
```

- A) Hello, world!
- B) Error
- C) None
- D) outer_function

1. **What is the output of the following code?** [Python Quiz #99]

```
def apply_function(func, x):
    return func(x)

def square(x):
    return x * x

result = apply_function(square, 5)
print(result)
```

- A) 25
- B) 5
- C) 10
- D) Error

Function Composition

21. **What is function composition in Python?**

- A. Combining multiple functions into a single function
- B. Applying a function multiple times
- C. Creating a new function from existing functions
- D. All of the above

22. **What is the output of the following code?**

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

def add_one(x):
    return x + 1

def compose(f, g):
    def composed_function(x):
        return f(g(x))
    return composed_function
```

```
result = compose(add_one, square)(5)
print(result)
```

- A. 26
- B. 36
- C. 25
- D. 11

Partial Application

23. **What is partial application in Python?**

- A. Applying a function to some of its arguments
- B. Creating a new function with fewer arguments
- C. Applying a function multiple times
- D. All of the above

24. **What is the output of the following code?**

```python
from functools import partial

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

add_5 = partial(add, 5)
result = add_5(3)
print(result)
```

- A. 8
- B. 5
- C. 3
- D. Error
-

21. **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.

22. **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:

```python
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:**

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

Output:

```
15
```

- **Watch the Solution Now** ✨

4. **Problem Statement:**

Write a function `add(*args)` that takes a variable number of arguments and returns the sum of all the arguments. The function should handle any number of arguments, including zero arguments. If no arguments are passed, the function should return `0`.

Function Signature:

```python
def add(*args):
```

Input:

- The function accepts a variable number of integer arguments. These integers can be positive, negative, or zero. The number of arguments can range from 0 to any positive integer.

Output:

- The function returns an integer, which is the sum of all the arguments passed to it. If no arguments are passed, the function should return `0`.

Sample Input:

```
add(1, 2, 3)
```

Sample Output:

```
6
```

- **Watch the Solution Now ✨**

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:

```python
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.