W03 LEC

Defining a Function

Introduction to User-Defined Functions in Python

- Welcome to the lecture on user-defined functions in Python.
- In this lecture, we will learn about the concept of functions and how to create our own functions in Python.
- Functions play a crucial role in structuring our code, promoting reusability, and making our programs more modular.

What are Functions?

- Functions are a way to group code into reusable blocks that perform specific tasks.
- They allow us to break down complex programs into smaller, more manageable parts.
- Functions can take input values called arguments or parameters, perform operations, and return output values.
- Functions are a key way to define interfaces so programmers can share their code.

Why Use Functions?

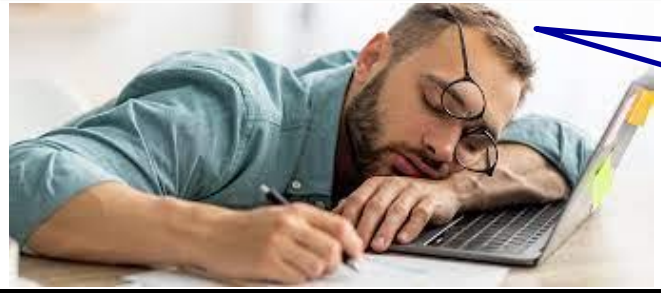
- Functions provide several benefits:
  - Code Reusability: Once a function is defined, it can be used multiple times throughout the program, eliminating code duplication.
  - Modularity: Functions help organize code into logical blocks, making it easier to understand and maintain.
  - Abstraction: Functions allow us to encapsulate complex operations behind a simple interface, making the code more readable.

Why Use Functions?

- Greeting

```
print("Hello, John")
print("Welcome to our 204101 class")
print("Hope you have fun with Python")

print("Hello, Ann")
print("Welcome to our 204101 class")
print("Hope you have fun with Python")

print("Hello, Dan")
print("Welcome to our 204101 class")
print("Hope you have fun with Python")
```

```
Hello, John
Welcome to our 204101 class
Hope you have fun with Python

Hello, Ann
Welcome to our 204101 class
Hope you have fun with Python

Hello, Dan
Welcome to our 204101 class
Hope you have fun with Python
```



Me, after greeting all 150 students in 204101 class

How do we write functions in Python?

Function Syntax

```
def function_name(param1, param2, …):
    line of code 1
    line of code 2
    …
    line of code N
    return result #Optional
```

4 spaces
(or a tab)

- **def**: Keyword used to define a function.
- **function_name**: Name of the function, following Python naming conventions.
- **param1, param2,** ...: Input parameters that the function expects. A function can have zero or many parameters (optional)
- **return**: Keyword used to specify the value the function should return (optional).

Function Example

function name

parameter

```
def greet(name):
    # Function to greet the user
    print("Hello,", name)

# Function call
greet("Alice")
```

function body

function call

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

```
def greet( "Alice"  ):
    # Function to greet the user
    print("Hello,", name     )
```

Output:

Hello, Alice

Why Use Functions?

- Greeting

```
def greet(name):
    print("Hello,", name)
    print("Welcome to our 204101 class")
    print("Hope you have fun with Python")

greet("John")
greet("Ann")
greet("Dan")
```

```
Hello, John
Welcome to our 204101 class
Hope you have fun with Python

Hello, Ann
Welcome to our 204101 class
Hope you have fun with Python

Hello, Dan
Welcome to our 204101 class
Hope you have fun with Python
```

Me, after greeting all 150 students in 204101 class using function

- An example of a function with no parameter (and no return statement):

No parameter is specified inside the parentheses. That's okay!

```
def my_function():
        print("Welcome to our 204101 class")
```

No return statement in the function body. That's okay too!

Return Statement

- Functions can return values using the `return` statement.
- The `return` statement specifies the value or expression that the function should return.
- Once a `return` statement is encountered, the function execution terminates and the result is returned to the caller.

Return Function

```
def  square(number):
      # Function to calculate the square of a number
      result  =  number  **  2
      return  result

# Function call
outcome =  square(5)
print("Square of 5 is",  outcome)
```

- In this example, the **square** function takes one parameter, **number**.
- Inside the function body, we calculate the square of the number and assign it to the **result** variable.
- We then use the **return** statement to return the calculated result.
- The returned value is stored in the **outcome** variable outside the function and displayed on the screen.

Return Function

```
def  square(number):
      # Function to calculate the square of a number
      result  =  number  **  2
      return  result

# Function call
outcome =  square(5)
print("Square of 5 is",  outcome)
```

```
# Function call
outcome  =  square(5)
```

```
def  square(   5   ):
      25   =   5   **  2
    return  25
```

```
print("Square of 5 is",  outcome)
```

Check Your Understanding

What is the output of the following code?

```
1 def greet(name):
2     print("Hello, " + name)
3
4 greet("Alice")
5 greet()
```

Check Your Understanding

What is the output of the following code?

```
1  def numval(num):
2      num += 1
3      print("num is: ",num)
4
5  value = 10
6  numval(value)
```

Variable Scope

Variable Scope and Parameter Passing in Python

- In this section, we will discuss the concept of variable scope and how parameters are passed to functions in Python.
- Understanding variable scope and parameter passing is crucial for writing efficient and bug-free code.

Variable Scope

- Variable scope refers to the visibility or accessibility of a variable within a program.
- In Python, variables can have either global scope or local scope.
  - Global variables are accessible throughout the entire program,
  - Local variables are only accessible within a specific block of code, such as a function.

Global Variable

- Global variables are defined outside of any function and are accessible throughout the program.
- They can be accessed and modified by any part of the code.

Global variable

global keyword to indicate that we want to modify the global variable

```
# Global variable
global_var = "Hello, I'm a global variable"

def print_global_var():
    # Accessing the global variable inside a function
    print(global_var)

def modify_global_var():
    # Modifying the global variable inside a function
    global global_var
    global_var = "Modified global variable"

# Calling the functions
print_global_var()
modify_global_var()
print_global_var()
```

```
Hello, I'm a global variable
Modified global variable
```

Global variables should be used with caution to avoid naming conflicts.

Local Variable

- Local variables are defined inside a function and are accessible only within that function.
- They are created when the function is called and destroyed when the function finishes execution.
- Local variables have a limited lifespan and do not interfere with variables outside the function.

Local Variable

- `result` variable is a local variable because **it is defined within the function.**
- It is accessible only within the scope of the `calculate_sum` function.

```
def calculate_sum(a, b):
    # Local variable
    result = a + b
    print("The sum is:", result)
```

```
The sum is: 8
```

```
# Calling the function
answer = calculate_sum(5, 3)
print("The sum is :",result)
```

```
NameError: name 'result' is not defined
```

'result' is only visible within the scope of the calculate_sum function

Check Your Understanding

What is the output of the following code?

```
1 def test():
2     x = 10
3     print("x is:", x)
4
5 x = 20
6 test()
7 print("x is:", x)
```

Check Your Understanding

What is the output of the following code?

```
1 def foo():
2     y = "Good day!"
3
4 foo()
5 print(y)
```

Check Your Understanding

What is the output of the following code?

```
z = 10

def bar():
    global z
    z = z + 10

bar()
print(z)
```

Parameter Passing

Parameter Passing

- When we call a function, we can pass values to the function using parameters.
- **Parameters** are variables that act as placeholders for the values we want to pass to the function.
- The values passed to the function are called **arguments.**

parameter

```
def greet(name):
    # Function to greet the user
    print("Hello,", name)

# Calling the function with an argument
greet("Alice")
```

argument

Parameter Passing Modes

- In Python, there are two main parameter passing modes: pass by value and pass by reference.
- In **pass by value**,
  - a **copy of the variable's value** is passed to the function.
  - Any changes made to the parameter inside the function **do not affect** the original variable outside the function.

Pass by Reference

- Python uses a variation of pass by reference called "pass by assignment" or "pass by object reference"
- In pass by reference,
  - **the memory address of the variable** is passed to the function.
  - Any changes made to the parameter inside the function **affect the original variable** outside the function.

Immutable vs Mutable Objects

- Immutable objects, such as **numbers and strings**, are **passed by value**.

- Mutable objects, such as **lists, sets, and dictionaries**, are **passed by reference**.
  - *We'll discuss them later*

Check Your Understanding

What is the output of the following code?

```
def modify_value(x):
    x += 5

value = 10
modify_value(value)
print(value)
```

Check Your Understanding

What is the output of the following code?

```
1 def modify_value(value):
2     value += 5
3
4 value = 10
5 modify_value(value)
6 print(value)
```

Exercises

[Recap] Computational Thinking

1. Decomposition: Breaking down complex problems into smaller, more manageable parts.
2. Pattern Recognition: Identifying patterns, similarities, and relationships within a problem.
3. Abstraction: Focusing on the essential details while ignoring irrelevant information.
4. Algorithm Design: Developing step-by-step instructions to solve a problem.

Algorithm Breakdown

- Breaking down the algorithm helps in understanding the problem and designing a modular and reusable solution.
- Algorithm breakdown involves breaking down a complex problem into smaller, more manageable steps.
- Each step should be well-defined and focused on solving a specific part of the problem.

Steps for Algorithm Breakdown

- Understand the Problem: Clearly define the problem and its requirements.
- Identify Inputs and Outputs: Determine the data or information required as input and the expected results as output.
- Break Down the Problem: Divide the problem into smaller sub-problems or tasks.
- Define Steps: For each sub-problem or task, define the specific steps needed to solve it.
- Sequence the Steps: Arrange the steps in a logical sequence, considering dependencies and order of execution.
- Test and Refine: Test the algorithm with sample inputs and refine it if necessary.

Example

- Problem: Calculate the square of a number.
- Algorithm Breakdown:
  1. Get the input number. (with the help of which Python built-in function?)
  2. Calculate the square of the number.
  3. Display the result. (with the help of which Python built-in function?)

Function Definitions:

```
def get_input_number():
    # Function to get the input number from the user
    # Code to retrieve and return the number

def calculate_square(num):
    # Function to calculate the square of a number
    # Code to perform the calculation
    # Return the square value

def display_result(result):
    # Function to display the calculated result
    # Code to display the result

#-----------------------------------
#main -----------------------------------
number = get_input_number()
outcome = calculate_square(number)
display_result(outcome)
```

Scan for this coding exercise

https://cmu.to/awXPC

Example

Problem: Body Mass Index Calculator (Input in cm and kg)

Algorithm Breakdown:

1. Get the input weight in kg from the user.
2. Get the input height in cm from the user.
3. Convert the height from cm to meters by dividing it by 100.
4. Calculate the square of the height.
5. Divide the weight by the square of the height to calculate the BMI.
6. Display the calculated BMI value.

BODY MASS INDEX FORMULA (Metric)

$$BMI = \frac{\text{Weight (kgs)}}{[\text{Height (m)}]^2}$$

Function Definitions: BMI Calculator

```
def get_weight():
    # Function to get the input weight in kg from the user
    # Code to retrieve and return the weight

def get_height():
    # Function to get the input height in cm from the user
    # Code to retrieve and return the height

def convert_to_meters(height_cm):
    # Function to convert height from cm to meters
    # Code to divide height_cm by 100 and return the result
```

Function Definitions: BMI Calculator

```
#Program BMI calculation
def get_user_input():
    #your code here
    return (input_w, input_h)

def calculate_bmi(in_weight, in_height):
    #your code here
    return bmi

def display_result(bmi):
    #your code here
#--------------------------------
#--------------------------------
#main function
weight, height = get_user_input()
bmi = calculate_bmi(weight, height)
display_result(bmi)
```

Scan for this coding exercise

https://cmu.to/Mbm1K

Summary

- A function is a section of reusable code.
- To call/invoke a function, type its name followed by a pair of parentheses.
- You have the option to pass data, known as arguments, to the function, but this requires a matching set of parameters in the function's definition.
- A function can return data back to the location where it was called.

References

- Hands-on Python tutorial
  - anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/functions.html
- Think Python: How to Think Like a Computer Scientist