Functions Theory

Assignment Solution

PW SKILLS

1. **Define a function and explain its syntax.**

Answer:

In Python, a function is a block of code that performs a specific task and can be reused throughout a program. Functions are defined using the `def` keyword followed by the function name and a set of parentheses containing optional parameters. The syntax to define a function is as follows:

Syntax:

```
def function_name(parameters):
    """
    Docstring: Description of the function
    """
    # Function body: Code to be executed
    statement(s)
```

- `def`: Keyword used to define a function.
- `function_name`: Name of the function, which should follow the naming conventions for variables.
- `parameters`: Optional inputs to the function, separated by commas.
- `Docstring`: Optional documentation string that describes the purpose and usage of the function.
- `statement(s)`: Code block that specifies the tasks to be performed by the function.

2. **Differentiate between types of functions such as built-in functions and user-defined functions:**

Answer:

Built-in functions: These functions are pre-defined in Python and are readily available for use without needing to be explicitly defined. Examples include `print()`, `len()`, `max()`, `min()`, etc.

User-defined functions: These functions are defined by the programmer to perform specific tasks according to their requirements. They are created using the `def` keyword followed by the function name, and they can accept parameters and return values as needed.

3. **Explain the difference between parameters and arguments in the context of function definition and function call:**

Answer:

Parameters: Parameters are variables listed in the function definition. They are placeholders for the values that will be passed to the function when it is called.

- Arguments: Arguments are the actual values passed to a function when it is called. They can be constants, variables, expressions, etc. Arguments are supplied to the function during its invocation.

Example:

```
def my_function(fname):  #fname is parameter
  print(fname + " Skills")

my_function("PW") #”PW” is argument
```

4. Define and discuss global scope/variables in Python functions with examples:

Answer:

Global variables are defined outside any function and can be accessed from anywhere in the program. They have global scope, meaning they are visible throughout the entire program.

Example:

```
global_var = 10

def func():
    print(global_var)  # Accessing global variable inside function

func()  # Output: 10
```

5. Define and discuss local scope/variables in Python functions with examples:

Answer:

Local variables are defined inside a function and can only be accessed within that function. They have local scope, meaning they are accessible only within the block of code where they are defined.

Example:

```
def func():
    local_var = 20
    print(local_var)  # Accessing local variable inside function

func()  # Output: 20
```

6. Describe the types of arguments in Python functions, including default arguments, keyword arguments, and positional arguments with examples:

Answer:

Default arguments: Parameters with default values specified in the function definition.

Code:

```
def greet(name, message="Love you Dad,"):
    print("Message from Morgan Stark:")
    print(message, name)

greet("Morgan")
Output:
Message from Morgan Stark:
Love you Dad, Morgan
```

Keyword arguments: Arguments passed with their parameter names.

Example:

```
def greet(name, message):
    print(message, name)

greet(message="Hi", name="Tony")  # Output: Hi Tony
```

Positional arguments: Arguments passed based on their position in the function call.

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

print(add(2, 3))  # Output: 5
```

7. **Discuss the concept of keyword arguments with an example function, explaining how changing the order of arguments does not affect the function call:**

Answer:

Keyword arguments allow you to specify arguments using their parameter names, which makes the code more readable and flexible. Changing the order of keyword arguments does not affect the function call.

Example:

```
def display_info(name, age, city):
    print("Name:", name)
    print("Age:", age)
    print("City:", city)

display_info(age=20, city="New York", name="Tony")
```

Output:

```
Name: Tony
Age: 20
City: New York
```

8. **Define a function that accepts a variable number of arguments using *args and demonstrate its usage with an example:**

Answer:

The `*args` syntax allows a function to accept any number of positional arguments, which are then accessed as a tuple inside the function.

Example:

```
def sum_numbers(*args):
    total = 0
    for num in args:
        total += num
    return total

print(sum_numbers(1, 2, 3, 4, 5))  # Output: 15
```

9. **Explain keyword arguments in Python functions with an example:**

Answer:

Keyword arguments allow you to specify arguments using their parameter names, which makes the code more readable and flexible. In Python functions, you can use the parameter names to explicitly pass values to the corresponding parameters, irrespective of their order.

Example:

```
def display_info(name, dob, Place):
    print("Name:", name)
    print("DOB:", dob)
    print("Place:", Place)

display_info(dob='23rd Jan 1897', Place="Cuttack, Orissa IN", name="Netaji")
```

Output:

```
Name: Netaji
DOB: 23rd Jan 1897
Place: Cuttack, Orissa IN
```

10. **Describe *args and kwargs in Python functions. When and why is it used?**

Answer:

- `***args**` allows a function to accept any number of positional arguments, which are then accessed as a tuple inside the function.
- `**kwargs**` allows a function to accept any number of keyword arguments, which are then accessed as a dictionary inside the function.

These are used when the number of arguments to be passed to a function is not fixed or known in advance. They provide flexibility in defining functions that can handle varying numbers of arguments.

11. How does Python handle functions with *args when different numbers of arguments are passed?

Answer:

Python handles functions with `*args` by packing any extra positional arguments into a tuple. If the number of arguments passed is less than the number of parameters in the function definition, Python assigns `None` to the remaining parameters.

12. Define nested functions in Python. Give an example and explain its structure.

Answer:

Nested functions are functions defined within another function. They have access to variables in the outer (enclosing) function's scope.

```
def outer_func():
    print("Outer function")

    def inner_func():
        print("Inner function")

    inner_func()

outer_func()()
```

Output:
Outer function
Inner function

In this example, `inner_func()` is defined inside `outer_func()`, making it a nested function. It can access variables from `outer_func()` but not vice versa.

13. What is a lambda function in Python? How is it different from regular functions, explain with example:

Answer:

A lambda function is an anonymous function defined using the `lambda` keyword. It can take any number of arguments but consists of only a single expression.

Example:

```
square = lambda x: x **2
print(square(7))  # Output: 49
```

Unlike regular functions defined using `def`, lambda functions are single-line expressions and do not have a name. They are typically used when a small, anonymous function is needed for a short period of time.

14. What is call by value and call by reference in Python. Explain with an example:

Answer:

- Call by Value: In call by value, a copy of the variable's value is passed to the function. Any modifications made to the parameter inside the function do not affect the original variable.

Example:

```
def increment(num):
    num += 1
    print("In inside function value of x:", num)

x = 10
increment(x)
print("In outside function, value of x:", x)
```

Output:

```
In inside function value of x: 11
In outside function, value of x: 10
```

Call by Reference: In call by reference, a reference to the variable is passed to the function. Any modifications made to the parameter inside the function affect the original variable.

Example:

```
def modify_list(lst):
    lst.append(4)
    print("Inside function:", lst)

my_list = [1, 2, 3]
modify_list(my_list)
print("Outside function:", my_list)
```

Output:

```
Inside function: [1, 2, 3, 4]
Outside function: [1, 2, 3, 4]
```

Programming Assignments:

Note: for Programming Assignments Please refer to the colab Notebook.
Programming Assignments: **Programming-solution :Week-04**