python function assignment 1

June 18, 2023

1. In Python, what is the difference between a built-in function and a user-defined function? Provide an example of each.

ANS: Functions

```
Built-in Functions
User Defined Functions
```

A built-in function is a function that is already defined in the Python interpreter and can be used directly. Examples of built-in functions include print( ), len( ), and input( ).

User Defined Functions is a block of reusable code that performs a specific task. We define a function using the def keyword, followed by the function name, parentheses (), and a colon :. Inside the parentheses, we can specify any parameters our function will take. The function's code block is indented under the function definition.Function can optionally return a value using return statementThe value that's returned can be used in other parts of your program. If no return statement is specified, the function will return None.

[5]:
```python
#Example of built-in function
# d, f and b are type

# integer
print(format(123, "d"))

# float arguments
print(format(123.4567898, "f"))

# binary format
print(format(12, "b"))
```

```
123
123.456790
1100
```

[4]:
```python
# Example of user defined function
def introduction(fname, lname):
  print("My first name is " + fname + " and my last name is " + lname)

introduction("John", "Alter")
```

```
My first name is John and my last name is Alter
```

2. How can you pass arguments to a function in Python? Explain the difference between positional arguments and keyword arguments.

ANS:A function can take multiple arguments, these arguments can be objects, variables(of same or different data types) and functions.Arguments are specified after the function name, inside the parentheses. We can add as many arguments as we want, just separate them with a comma.

```
[7]: def my_function(fname):
       print(fname + " id")

     my_function("Emil")
     my_function("Tobias")
     my_function("Linus")
```

```
Emil id
Tobias id
Linus id
```

Arguments are passed to functions as either positional arguments or keyword arguments. Positional arguments are passed in the order they are defined in the function signature. Keyword arguments are passed with a name-value pair and can be passed in any order. They are useful when we want to pass optional arguments to a function.

```
[2]: # Example of Positional Argument
     def my_function(a, b, c):
         print(a, b, c)    #a, b, and c are positional arguments. we must pass them␣
      ↪in the order they appear in the function definition.

     my_function(6,4,2)
```

```
6 4 2
```

```
[1]: #Example of Keyword Argument
     def my_function(a,b,c):
             print(a,b,c)

     my_function(c=3,b=2,a=1)
```

```
1 2 3
```

3. What is the purpose of the return statement in a function? Can a function have multiple return statements? Explain with an example.

ANS:The return statement is used to end the execution of a function and return the result (value of the expression following the return keyword) to the caller. When a return statement is used in a function body, the execution of the function is stopped. If specified, a given value is returned to the function caller. A function can have multiple return statements. When a return statement is executed, the function stops executing and returns the specified value

Overall, the return statement is crucial in user-defined functions as it enables the functions to produce outputs, pass data, allow result reusability, handle conditional returns, and control the flow of the function's execution.

[2]:
```
def sum(a,b):
    if type(a) != int or type(b) != int:
        return "Both arguments must be integers"
    return a+b

print(sum(2,3)) # Output: 5
print(sum("2",3)) # Output: Both arguments must be integers

#In this example, if both arguments are integers, then their sum is returned.
#If either argument is not an integer, then a string message is returned␣
 ↪instead.
```

```
5
Both arguments must be integers
```

4. What are lambda functions in Python? How are they different from regular functions? Provide an example where a lambda function can be useful.

ANS:Lambda functions are small, anonymous functions defined with the lambda keyword.The syntax of a lambda function is as follows: lambda parameters: expression.They can take any number of arguments but can only have one expression.Lambda functions do not require a return statement, the expression is implicitly returned.Lambda functions are useful for small tasks that are not reused throughout the code.

Lambda functions are different from regular functions in that they are anonymous and do not have a name. They are also not defined using the def keyword. Instead, they are defined using the lambda keyword. Lambda functions are often used when you need to pass a small function as an argument to another function.

[10]:
```
# A list of numbers
numbers = [1, 2, 3, 4, 5]

# Using filter() with lambda function to filter out even numbers
filtered_numbers = list(filter(lambda x:x % 2 == 0, numbers))

print(filtered_numbers) # Output: [2, 4]
```

```
[2, 4]
```

In the above example we have a list of numbers [from 1 to 5],we used a function filter to get an even number as a output.The lambda function takes one argument (x) and returns True if the number is even (x % 2 == 0). The filter() function then filters out all the elements from the list for which the lambda function returns False. Finally, we convert the filtered result into a list using the list() constructor.

5. How does the concept of "scope" apply to functions in Python? Explain the difference between local scope and global scope.

ANS:In Python, the concept of "scope" refers to the region of the program where a particular variable is accessible. The scope of a variable is determined by where it is defined in the code.

A variable defined inside a function has a local scope and can only be accessed within that function. On the other hand, a variable defined outside of any function has a global scope and can be accessed anywhere in the program

Local variables are not accessible outside the function. If you try to access them outside the function, you will get an error. global scope is created when a variable is defined outside of any function

[19]:
```
x = 10 # global scope

def my_function():
    y = 5 # local scope
    print(x) # prints 10

my_function()
print(y) # NameError: name 'y' is not defined
```

```
10
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [19], line 8
      5     print(x) # prints 10
      7 my_function()
----> 8 print(y)

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

In this example, x is defined in the global scope and can be accessed from inside the function my_function(). However, y is defined in the local scope of my_function() and cannot be accessed outside of it.

6. How can you use the "return" statement in a Python function to return multiple values?

ANS:We can return multiple values by bundling those values into a dictionary, tuple, or a list. These data types let us store multiple similar values. We can extract individual values from them in our main program. Or, we can pass multiple values and separate them with commas.

Python functions can return multiple values. To return multiple values, we can return either a dictionary, a Python tuple, or a list to our main program.

[15]:
```
def my_mini_calculator():
    num1 =int(input("Enter the first number: "))
    num2 =int(input("Enter the second number: "))

    #ADDITION
    add_result = num1 + num2
    #SUBTRACTION
    diff_result = num1 - num2

    #MULTIPLICATION
    mult_result = num1*num2

    #DIVISION
    div_result =num1 / num2

    return {                                    #passing multiple values
        "Addition": add_result,
        "Subtraction": diff_result,
        "Multiplication": mult_result,
        "Division": div_result,
    }

result= my_mini_calculator()

print("The sum is:",result["Addition"])
print("The diff is:",result["Subtraction"])
print("The product is:",result["Multiplication"])
print("The quotient is:",result["Division"])
```

```
Enter the first number:  10
Enter the second number:  9

The sum is: 19
The diff is: 1
The product is: 90
The quotient is: 1.1111111111111112
```

7. What is the difference between the "pass by value" and "pass by reference" concepts when it comes to function arguments in Python?

ANS:In "pass by value", the parameter value is copied to another variable. If you modify what you passed, it will not affect the original variable. In "pass by reference", the actual parameter is passed to the function. This means that what you passed is a reference to the original variable directly, and modifying it modifies the original variable as well.

[68]:
```
#Exaample of pass by value

def change(a):
    print("this is original a",id (a))
    a=a+10
    print("this is new a", id (a))
    print("inside fun a=",a)

a = 10
print("a before calling:",a)
print("this is main a",id (a))
change(a)
print("a after calling:",a)
```

```
a before calling: 10
this is main a 139651461251600
this is original a 139651461251600
this is new a 139651461251920
inside fun a= 20
a after calling: 10
```

[70]:
```
#Example of pass by refernce

def change (lst):
    lst[0]= lst[0]+10
    print("inside fun=",lst)

lst =[10]
print("before calling:",lst)
change(lst)
print("after calling:",lst)
```

```
before calling: [10]
inside fun= [20]
after calling: [20]
```

8. Create a function that can intake integer or decimal value and do following operations:

a. Logarithmic function (log x)
b. Exponential function (exp(x))
c. Power function with base 2 (2 x)
d. Square root

[41]:
```
import math
x = 12
def math_operations(x):
    print(f"Logarithmic function (log x): {math.log(x)}")
    print(f"Exponential function (exp(x)): {math.exp(x)}")
    print(f"Power function with base 2 (2^x): {math.pow(2,x)}")
    print(f"Square root: {math.sqrt(x)}")

math_operations(x)
```

```
Logarithmic function (log x): 2.4849066497880004
Exponential function (exp(x)): 162754.79141900392
Power function with base 2 (2^x): 4096.0
Square root: 3.4641016151377544
```

9. Create a function that takes a full name as an argument and returns first name and last name.

```
[62]: def get_first_last_name(full_name):
          first_name = full_name.split()[0]
          last_name = full_name.split()[-1]
          return first_name, last_name
```

```
[63]: # Example
      def get_first_last_name(full_name):
          full_name ="Pratiksha Prashant Mahadik"
          first_name = full_name.split()[0]
          last_name = full_name.split()[2]
          return (first_name, last_name)
      get_first_last_name('full_name')
```

```
[63]: ('Pratiksha', 'Mahadik')
```

```
[ ]:
```