Assignment-3

1. **Why are functions advantageous to have in your programs?**
   Ans.
   - Using functions we can avoid rewriting the same logic/code again and again in a program.
   - Python functions can be called multiple times in the program and anywhere in the program.
   - It becomes easy to track the umpteen lines of code when the code is divided into multiple functions.
   - Reusability is the main achievement of Python functions.
   - However, function calling is always overhead in a Python program.

2. **When does the code in a function run: when it's specified or when it's called?**
   Ans.

```
In [2]: def evenOdd(n):
            if n%2 == 0:
                print("The number is even.")
            else:
                print("The number is odd.")

        evenOdd(88)

        The number is even.

In [ ]:
```

   So, as per the above program for checking if an integer passed is an even number or odd number. evenOdd() method is specified but it only gets executed after calling it.
   Parameter should be passed when calling a method or a function.

3. **What statement creates a function?**
   Ans.
   def keyword is used to create a method.

   Sytax:

   def function_name(parameters):
   “ “ “docstring” ” ”
   statements(s)

return expression

Ex.

```
In [4]: def square(n):
            """This method returns the
            square value of entered number"""
            return n**2

        print(square(2))
        print(square(-14))
```

```
4
196
```

4. **What is the difference between a function and a function call?**
   Ans.
   Ex.

```
def func():
        print("Pratik Kodilkar")
```

- In the following example, func() method has been created which has as print statement which is an expression which is going to be executed.
- Creating a function is all about naming or creating it by using def keyword and after that operation which have to be carried out can be defined in it.

```
def func():
        print("Pratik Kodilkar")

func()
```

- Now, by taking the above example we are using func(), we're calling it, so that the statements defined it in can be executed.

5. How many global scopes are there in a Python program? How many local scopes?
Ans.

In [26]:
```
x = 'global x'

def test():
    y = 'local y'
    print(y)

test()
```

```
local y
```

Here, local variable 'y' gets printed because its in the local scope in the test() method. Local variable is not accessible outside the test() method.

In [27]:
```
x = 'global x'

def test():
    y = 'local y'
#     print(y)
    print(x)

test()
```

```
global x
```

x gets printed as it is in the global scope. The compiler checks if the x is in the local scope or not. If x isn't in the test() method which means its not in the local scope, then compiler checks x in Enclosing scope, here as well compiler fails it to find out. Finally, x get shows up into the global scope, and x gets printed. Global variable are accessible inside the method as as well outside the method.

In [28]:
```
## Global keyword

# x = 'global x'

def test():
    global x
    x = 'global x-2'
#     print(y)
    print(x)

test()
print(x)
```

```
global x-2
global x-2
```

x is redefined with a new value as well as it is converted to global variable. x variable has been overwritten. It is now accessible inside the local and global scope. x = 'global x' which has been defined earlier is no longer necessary.

In [29]:
```
## Nonlocal keyword

def outer():
    x = 'outer x'

    def inner():
        nonlocal x
        x = 'inner x'
        print(x)

    inner()
    print(x)

outer()
```

```
inner x
inner x
```

In Python, nonlocal keyword is used in the case of nested functions. This keyword works similar to the global, but rather than global , this keyword declares a variable to point to the variable of outside enclosing function, in case of nested functions. Here, x variable inside the inner() method has been executed. The nonlocal variable x is accessible inside the inner() method as well as outside the inner method. x variable outside the inner() method is global to it. Now, by using nonlocal keyword, x inside inner() method has overwritten the value of x in the outer() method. And nonlocal variable x is executed inside the inner() method as well as outside of it.

6. **What happens to variables in a local scope when the function call returns?**
   Ans.

```
In [31]: num1 = 0
         num2 = 0

         def calc():
             num1 = 0
             num2 = 0
             num1 += 1
             num2 += 1
             return num1, num2

         calc()
         print(num1)
         print(num2)
```

```
0
0
```

- In this program, global variable num1 = 0 and num2 = 0, the variables num1 and num2 in the local scope has '0' as value respectively.
- But, after addition operation on num1 as well as num2 should have to be 1 as result for num1 and num2.
- Local variables are not accessible outside the method. So, the global variables num1=0 and num2 =0 have been printed.

- The num1 and num2 variables should be passed as arguments within the calc() method. Then it overwrites the value of variables after addition and gives result as '1'. This is the solution to get a value true value after addition operation from the local scope.

```
In [38]: num1 = 0
         num2 = 0

         def calc(num1, num2):
             num1 = 0
             num2 = 0
             num1 += 1
             num2 += 1
             return(num1, num2)

         num1, num2 = calc(num1, num2)

         print(num1)
         print(num2)
```

```
1
1
```

7. **What is the concept of a return value? Is it possible to have a return value in an expression?**
   Ans.
   - A return statement is used to end the execution of the function call and "returns" the result to the caller. The statements after the return statements are not executed.
   - If the return statement is without any expression, then the special value **None** is returned.
   - A **return statement** is overall used to invoke a function so that the passed statements can be executed.
   - return statement cannot be used outside the function.
   - It is possible to return a value in an expression.

     Ex.

```
In [40]: def cube(n):
             return n**3

         cube(5)

Out[40]: 125
```

8. **If a function does not have a return statement, what is the return value of a call to that function?**
   Ans.
   If a function does have an explicit return statement it returns **None**.

9. **How do you make a function variable refer to the global variable?**
   Ans.
   global keyword:

```
In [42]: ## Global keyword

         x = 'global x'

         def test():
             global x
             x = 'global x-2'
         #     print(y)
             print(x)

         test()
         print(x)

global x-2
global x-2
```

- In this Python program, local variable x is refered as global variable by using global keyword. The variable x inside the local scope has converted into global variable, which is now accessible inside the local as well as global scope.
- x variable inside the test() method has overwritten the value of global variable x.

nonlocal keyword:

```
In [47]: print("Value of a using nonlocal is ", end="")
def outer():
    a = 5
    def inner():
        nonlocal a
        a = 10
    inner()
    print (a)

outer()
```

```
Value of a using nonlocal is 10
```

- In Python, nonlocal keyword is used in the case of nested functions.
- This keyword works similar to the global, but rather than global, this keyword declares a variable to point to the variable of outside enclosing function, in case of nested functions.
- Now, a in inner() method has overwritten the value by referring the a variable in the outer() method.

10. What is the data type of None?

Ans.

- In Python, None keyword is an object, and it is a data type of the class NoneType .
- We can assign None to any variable, but we can not create other NoneType objects.

11. What does the sentence import areallyourpetsnamederic do?

Ans.

It gives an error because, this module ‘areallyourpetsnamederic’ does not consist in Python Library.

12. **If you had a bacon() feature in a spam module, what would you call it after importing spam?**

    Ans.

    This function can be called with spam.bacon().

13. **What can you do to save a program from crashing if it encounters an error?**

    Ans.

    - The code shall be processed inside the **try** and **except** statement.
    - When it encounters an error, the control is passed to the **exept** block, skipping the code in between.
    - After execution of program, it will throw an error instead of crashing.

14. **What is the purpose of the try clause? What is the purpose of the except clause?**

    Ans.

    - The code that could potentially cause an error goes in the try clause.
    - The code that executes if an error happens goes in the except clause.