# ASSIGNMENT 24

**QUESTION 1. What is the relationship between def statements and lambda expressions ?**

ANSWER :

Using `def`: we need to define a function with a name and need to pass a value to it. After execution, we also need to return the result from where the function was called using the `return` keyword.

Using `Lambda`: Lambda definition does not include a "return" statement, it always contains an expression which is returned. We can also put a lambda definition anywhere a function is expected, and we don't have to assign it to a variable at all. This is the simplicity of lambda functions.

Lambda vs. def.

1. Def can hold multiple expressions while lambda is a uni-expression function.
2. Def generates a function and designates a name to call it later. Lambda forms a function object and returns it.
3. Def can have a return statement. Lambda can't have return statements.
4. Lambda can be used inside a list and dictionary.

**QUESTION 2. What is the benefit of lambda?**

ANSWER :

The most important benefit of lambda function is **IIFE** (**immediately invoked function execution**). It means that a lambda function is callable as soon as it is defined. This provides Lambda functions a way to perform operations using built-in methods, such as Map() and Filter() in Python. Also, using lambda functions, small and single-use functions can be created that can save time and space in the code.

**QUESTION 3. Compare and contrast map, filter, and reduce.**

ANSWER:

**`filter`** is a function which when given a list and a function applies the function to each item of the list returning a list of those items for which the function returned `True`.

**`map`** is a function which when given a list and a function, applies the function to each item of the list and returns the resultant list having the same number of elements as in the given list.

**`reduce`** is a function which when given a list, a function (operator) and initial value of the accumulator, it applies the function to the combination of each value and the accumulator and returns the final value of the accumulator. If the initial value is not given then the first value of the list is the initial value and the reduction is applied to the remaining elements.

**QUESTION 4. What are function annotations, and how are they used?**
ANSWER :

- Function annotations are nothing more than a way of associating arbitrary Python expressions with various parts of a function at compile-time and these expressions have no life in python's runtime environment.
- Python does not attach any meaning to these annotations. They take life when interpreted by third party libraries.
- For example, Annotations for excess parameters : Excess parameters for e.g. `*args` and `**kwargs`, allows arbitrary number of arguments to be passed in a function call.

**QUESTION 5. What are recursive functions, and how are they used?**
ANSWER:
Recursive function is a function that calls itself directly or indirectly. It is always made up of 2 portions, the base case and the recursive case.

- The recursive case is the part where the function calls on itself.
- Base Case: One critical requirement of recursive functions is the termination point or base case. Every recursive program must have a base case to make sure that the function will terminate. Missing base case may result in unexpected behavior.

```
def factorial(x):
    if x == 1: # This is the base case
        return 1

    else: # This is the recursive case
        return(x * factorial(x-1))

print(factorial(4))
```

**QUESTION 6. What are some general design guidelines for coding functions?**
ANSWER:

1. Use 4-space indentation and no tabs. Tabs should be used solely to remain consistent with code that is already indented with tabs. Python disallows mixing tabs and spaces for indentation.
2. Maximum Line Length : Limit all lines to a maximum of 79 characters. For flowing long blocks of text with fewer structural restrictions (docstrings or comments), the line length should be limited to 72 characters.
3. It is permissible to break before or after a binary operator, as long as the convention is consistent locally.
4. Imports are always put at the top of the file, just after any module comments and docstrings, and before module globals and constants.
    a. Imports should be grouped in the following order:
    b. Standard library imports.
    c. Related third party imports.

    d. Local application/library specific imports.
5. Wildcard imports (`from <module> import *`) should be avoided, as they make it unclear which names are present in the namespace, confusing both readers and many automated tools.
6. Class names should normally use the CapWords convention. The naming convention for functions may be used instead in cases where the interface is documented and used primarily as a callable.
7. Code should be written in a way that does not disadvantage other implementations of Python (PyPy, Jython, IronPython, Cython, Psyco, and such).
8. Comparisons to singletons like `None` should always be done with `is` or `is not`, never the equality operators.
9. When catching exceptions, mention specific exceptions whenever possible instead of using a bare `except:` clause. A bare `except:` clause will catch `SystemExit` and `KeyboardInterrupt` exceptions, making it harder to interrupt a program with Control-C, and can disguise other problems.

**QUESTION 7. Name three or more ways that functions can communicate results to a caller.**
ANSWER:
1. By using **`return`** statements in the function, we can collect and store result from the function in a variable and either use the result in some other function or print the value.
2. Using Object: We can create a class to hold multiple values and return an object of the class.
3. Using Tuple: A Tuple is a comma separated sequence of items. It is created with or without `()`. Tuples are immutable.

```
def fun():
   string = "hello"
   x = 20
  return (string, x)

string, x = fun() # Assign returned tuple
print(string)
print(x)
```

   NOTE: This can also be done using lists.
4. Python functions are first class objects, so we can return a function from another function.
5. Using **`lambda`** function: In the main body, especially while using `lists` with `filter()`, we can equate the lambda to a variable and get the returned value stored in the variable and use it further.
6. Using **`yield`** statement, used to return from a function without destroying the states of its local variable.