Assignment_24

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

   Keyword def that marks the start of the function header. A function name to uniquely identify the function.

   Function naming follows the same rules of writing identifiers in Python

   lambdas are one-line methods without a name.

   They work practically the same as any other method in Python

   Lambdas differ from normal Python methods because they can have only one expression,

   can't contain any statements and their return type is a function object

2. What is the benefit of lambda?

   Fewer Lines of Code

   Lambda functions are inline functions and thus execute comparatively faster

   Many times lambda functions make code much more readable by avoiding the logical jumps caused by function calls

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

   map applies as a transformation to an element.

   The map() function iterates through all items in the given iterable and executes the function we passed as an argument on each of them.

Syntax : map(function, iterable(s))

filter accumulates only elements matching a condition.

filter() forms a new list that contains only elements that satisfy a certain condition, i.e. the function we

passed returns True

Syntax : filter(function, iterable(s))

reduce accumulates all elements to a single value, by using immutable values

reduce() works by calling the function we passed for the first two items in the sequence. The result returned by

the function is used in another call to function alongside with the next (third in this case), element

Syntax : reduce(function, sequence[, initial])

In [ ]:

```
### Map functions
fruit = ["Apple", "Banana", "Pear", "Apricot", "Orange"]
map_object = map(lambda s: s[0] == "A", fruit)
```

```
#print(list(map_object))
for i in map_object:
   print(i)
```

In [12]:

```
### Filter functions
fruit = ["Apple", "Banana", "Pear", "Apricot", "Orange"]
filter_object = filter(lambda s: s[0] == "A", fruit)

for i in filter_object:
   print(i)
```

```
Apple
Apricot
```

In [14]:

```
### Reduce function
from functools import reduce

list = [2, 4, 7, 3]
print(reduce(lambda x, y: x + y, list))
print("With an initial value: " + str(reduce(lambda x, y: x + y, list, 10)))
```

```
16
With an initial value: 26
```

In [15]:

```
### Reduce function
from functools import reduce

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

list = [2, 4, 7, 3]
print(reduce(add, list))
```

```
16
```

4. What are function annotations, and how are they used?

Function annotation is the standard way to access the metadata with the arguments and the return value of the function.

These are nothing but some random and optional Python expressions that get allied to different parts of the function.

They get evaluated only during the compile-time and have no significance during the run-time of the code.

They do not have any significance or meaning associated with them until accessed by some third-party libraries.

They are used to type check the functions by declaring the type of the parameters and the return value for the functions.

The string-based annotations help us to improve the help messages.

Syntax :

```
def func(a: 'int') -> 'int':

    pass
```

Annotations for simple parameters:

```
def func(x: 'float'=10.8, y: 'argument2'):
```

In the above code the argument, ‘x’ of the function func,

has been annotated to float data type and the argument ‘y’

has a string-based annotation. The argument can also be

assigned to a default value using a ‘=’ symbol followed

by the default value. These default values are optional to the code.

Annotations for return values:

def func(a: expression) -> 'int':

The annotations for the return value is written after the '->' symbol.

In [2]:

```
def fib(n:'float', b:'int')-> 'result':
   pass
print(fib.__annotations__)
{'n': 'float', 'b': 'int', 'return': 'result'}
```

5. What are recursive functions, and how are they used?

A recursive function is a function that calls itself during its execution.

This means that the function will continue to call itself and repeat its behavior until some condition is met to return

a result

In [1]:

```
# example
def fact(x):
   if x == 1 :
      return 1
   else :
      return x * fact(x-1) # recurtion

fact(3)
```

Out[1]:

```
6
```

6. What are some general design guidelines for coding functions?

1. Use 4-space indentation and no tabs.

2. Use docstrings

3. Wrap linethat they don’t exceed 79 characters

4. Use of regular and updated comments are valuable to both the coders and users

5. Use of trailing commas : in case of tuple -> ('good',)

6. Use Python’s default UTF-8 or ASCII encodings and not any fancy encodings

7. Naming Conventions

8.Characters that should not be used for identifiers :

‘l’ (lowercase letter el),

‘O’ (uppercase letter oh),

‘I’ (uppercase letter eye) as single character variable names as these are similar to the numerals one and zero.

9. Don’t use non-ASCII characters in identifiers

10. Name your classes and functions consistently

11. While naming of function of methods always use self for the first argument

7. Name three or more ways that functions can communicate results to a caller.

1. Function can return single value

2. Can return multiple values, tuple

3. can return list,dictionary

4. can return function object

5. can return class object