Q1. Is it permissible to use several import statements to import the same module? What would the goal be? Can you think of a situation where it would be beneficial?

Ans:- *Yes we can use more than one import statement to import the same module more than once.*

*# This is com einto action when support we want to import some other function or class from the base*

*# class of any module for example.*

*# from sklearn.preprocessing import LabelEncoder*

*# from sklearn.linear\_model import LinearRegression*

Q2. What are some of a module's characteristics? (Name at least one.)

Ans:- *name : It returns the name of the module. doc : It denotes the documentation string line written in a module*

*# code. file : It holds the name and path of the module file from which it is loaded . dict : It return a*

*# dictionary object of module attributes, functions and other definitions and their respective values.*

Q3. Circular importing, such as when two modules import each other, can lead to dependencies and bugs that aren't visible. How can you go about creating a program that avoids mutual importing?

Ans:- *Circular importing means importing the two modules in each other. If suppose we are wokring in MOD1.py file*

*# and it is importing some function say F2() from some other module say MOD2.PY file or we can do vice-versa.*

*# What will happen is: This will give an import error. This is because when we import F2() function from module*

*# MOD2.py, then this will execute MOD2.py file. And in MOD2.py file there is an another statement of importing*

*# MOD1.py module. This will result in endless loop. To avoid this error just do one thing- We can use if*

*# name == 'main'.In the function, you can't directly refer to the function in the program. The addition of*

*# this sentence avoids the endless loop of the program*

Q4. Why is \_ \_all\_ \_ in Python?

Ans:- *It specifies all the modules present in the particular library and those can be called when we use import \**

Q5. In what situation is it useful to refer to the \_ \_name\_ \_ attribute or the string '\_ \_main\_ \_'?

Ans:- *During the time of execution of the code if we want to refer the module in which we are working on then we*

*# uses name attribute. In that case it will return the module in which we are working on. Suppose if that moudle*

*# is being imported from some other module then name will have the name of that moudle from where the current*

*# module has been imported. The current module in which we are working is refer to the string '\_ main \_'.*

Q6. What are some of the benefits of attaching a program counter to the RPN interpreter application, which interprets an RPN script line by line?

Ans:- *RPN saves time and keystrokes. You avoid using and keeping track of parentheses while doing calculations.*

*# The process is similar to the way you learned math on paper. You can see the intermediary results as you*

*# perform your computations rather than just the answer at the end.*

Q7. What are the minimum expressions or statements (or both) that you'd need to render a basic programming language like RPN primitive but complete— that is, capable of carrying out any computerised task theoretically possible?

Ans:- *Notations : +-/\**

*# These are the basic notations we require to carry out a computerised task , like RPN Primitive.*

*# We also need a particular data structure for storing elements from a statements except operators*