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, it is permissible to use several import statements to import the same module. It is used in case when we have to import multiple functions from same module.

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

Ans. The following are some of a module's characteristics:

\_\_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 thingWe 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 provides list of all modules present in a library.

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. Advantage of reverse Polish notation is that it removes the need for parentheses that are required by infix notation, since the stack holds all arguments in a last-in, first out progression. For example, to compute the expression (3 × 4) + (5 × 6), one would type 3, press Enter ↑, and type 4. Upon pressing × (multiply), the intermediate product 12 appears visually at the bottom of the stack. Then one types 5, Enter ↑, and 6. The intermediate result 12 has been promoted to level three, with the 5 at level two and the 6 visible at level one. It is only required to press × and then + in succession. The intermediate product, 30, appears first in level one, and the final result, 42 appears at level one since the 12 at level two has now been added.

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. Reverse Polish notation was proposed by Burks, Warren and Wright in 1954 and so named because it was simply the reverse of Polish notation (prefix notation), invented by the Polish logician Jan Lukasiewicz, which puts the operator before the operands. In the 1960s, it was then independently reinvented by E.W. Dijkstra and F.L. Bauer for reducing the number of times computer memory is accessed and increasing performance. It made use of the computer’s stack to store its operands before executing the operator.

RPN leads to faster calculations for a couple of reasons. One is that there is less information to store. Therefore, instead of needing to store nine characters for the expression ((5 – 3) \* 2), computers using RPN only need to store five characters with the expression 5 3 – 2 \*. And because there are fewer characters to process, execution becomes faster.

So in a computer using RPN, the evaluation of the expression 5 1 – 3 \* is as follows:

1. Push 5 into the stack. This is the first value.
2. Push 1 into the stack. This is the second value and is on the position above the 5.
3. Apply the subtraction operation by taking two operands from the stack (1 and 5). The top value (1) is subtracted from the value below it (5), and the result (4) is stored back to the stack. 4 is now the only value in the stack and is in the bottom.
4. Push 3 into the stack. This value is in the position above 4 in the stack.
5. Apply the multiplication operation by taking the last two numbers off the stack and multiplying them. The result is then placed back into the stack. After this operation, the stack now only contains the number 12.