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:

Any duplicate import statement for the same module in the same program is ignored by Python.

What happens if a module is imported twice? The module is only loaded the first time the import statement is executed and there is no performance loss by importing it again.

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

ANS:

Characteristics of Modules

* Modules contain instructions, processing logic, and data.
* Modules can be separately compiled and stored in a library.
* Modules can be included in a program.
* Module segments can be used by invoking a name and some parameters.
* Module segments can be used by other modules.

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:

In python, a module can be made by importing other modules. In some cases, a Circular dependency is created. Circular dependency is the case when some modules depend on each other. It can create problems in the script such as tight coupling, and potential failure.

To run the script without any error the Circular dependency has to be removed. Circular dependency can be created in many ways. With a well-designed program, these cases can be avoided.

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

ANS:

\_\_all\_\_ is used to document the public API of a Python module. Although it is optional, \_\_all\_\_ should be used.

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

ANS:

You use this idiom when you want to create an additional entry point for your script, so that your file is accessible as a stand-alone script as well as an importable module. You might want that when your script needs to collect user input.

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:

The advantage of reverse Polish notation is that it removes the need for order of operations and parentheses that are required by infix notation and can be evaluated linearly, left-to-right. For example, the infix expression (3 × 4) + (5 × 6) becomes 3 4 × 5 6 × + in reverse Polish notation.

Stacks can be used to evaluate postfix notation equations (also known as Reverse Polish notation ). So the algorithm moves along the expression, pushing each operand on the stack while operators cause two items to be popped off the stack , evaluated and the result pushed back on the stacks

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:

An assignment statement sets the current value of a variable, field, parameter, or element. The statement consists of an assignment target followed by the assignment operator and an expression. When the statement is executed, the expression is evaluated and the resulting value is stored in the target.

For example: a = 6; c = a + b; ++j; The first two expression statements both cause the value of the expression on the right of the equal sign to be assigned to the variable on the left. The third expression statement causes the value of j to be incremented by 1.

An Assignment statement is a statement that is used to set a value to the variable name in a program.

**Assignment statements** initialize or change the value stored in a variable using the assignment operator =. An assignment statement always has a single variable on the left hand side. The value of the **expression** (which can contain math operators and other variables) on the right of the = sign is stored in the variable on the left.