ASSIGNMENT - 24

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's allowed to have multiple import statements for the same module in Python. One goal might be to improve code readability by importing specific elements from the module in different parts of the code. It could be beneficial when organizing imports or when working in different sections of a large codebase.

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

Ans: A module in Python is a file containing Python definitions and statements. One characteristic is its ability to encapsulate reusable code, making it accessible through importing and helping in organizing code into logical units.

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: To avoid circular imports, refactor the code to eliminate mutual dependencies. You can often resolve circular imports by restructuring code into smaller, more modular components or by using techniques like importing modules within functions only when needed rather than at the top-level.

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

Ans: The \_ \_all\_ \_ attribute in Python specifies what symbols (functions, classes, variables) will be exported when using the from module import \* syntax. It acts as a list of public objects that the module intends to make available for import.

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

Ans: Checking if \_\_name\_\_ == '\_\_main\_\_': allows code in the script to be executed only when the script is run directly, not when it's imported as a module. This is helpful when creating modules that can be used both as standalone scripts and as imported modules.

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: Attaching a program counter to an RPN (Reverse Polish Notation) interpreter helps track the current execution position within the script. It enables sequential interpretation of the RPN script, allowing step-by-step execution or handling of control flow instructions.

Q7. What are the minimum expressions or statements (or both) that‘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: To render a basic but complete RPN language, you'd need at least:

* Stack data structure for operands.
* Operators/functions to perform arithmetic, logic, and control flow operations.
* Parser to read and interpret RPN expressions.
* Ability to handle variables, user-defined functions, and branching.