**FSDS MAY BATCH 2022(Python Assignment -24)**

**Submitted by: Shubham Tiwari**

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: Import in python is similar to #include header\_file in C/C++. Python modules can get access to code from another module by importing the file/function using import. The import statement is the most common way of invoking the import machinery, but it is not the only way.

**import module\_name**

When the import is used, it searches for the module initially in the local scope by calling \_\_import\_\_() function. The value returned by the function is then reflected in the output of the initial code.

**import** math

pie **=** math.pi

print("The value of pi is : ",pie)

**Output:**

**The value of pi is : 3.141592653589793**

**import module\_name.member\_name**

In the above code module, math is imported, and its variables can be accessed by considering it to be a class and pi as its object.

The value of pi is returned by \_\_import\_\_(). pi as a whole can be imported into our initial code, rather than importing the whole module.

The value of pi is returned by \_\_import\_\_(). pi as a whole can be imported into our initial code, rather than importing the whole module.

When the import is used, it searches for the module initially in the local scope by calling \_\_import\_\_() function. The value returned by the function is then reflected in the output of the initial code.

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

Ans: The following are the desirable characteristics of a module.

1. Modules contain instructions, processing logic, and data.

2. Modules can be separately compiled and stored in a library.

3. Modules can be included in a program.

4. Module segments can be used by invoking a name and some parameters.

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 general, circular imports are the result of bad designs. A deeper analysis of the program could have concluded that the dependency isn't actually required, or that the depended functionality can be moved to different modules that wouldn't contain the circular reference.A simple solution is that sometimes both modules can just be merged into a single, larger module. The resulting code from our example above would look something like this:

**def function1():**

**function2()**

**def function2():**

**print('Hello, World!')**

**function3()**

**def function3():**

**print('Goodbye, World!')**

**function1()**

However, the merged module may have some unrelated functions (tight coupling) and could become very large if the two modules already have a lot code in them.

So if that doesn't work, another solution could have been to defer the import of module2 to import it only when it is needed. This can be done by placing the import of module2 within the definition of function1().In this case, Python will be able to load all functions in module1 and then load module2 only when needed.

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

Ans: A list of strings that define what variables have to be imported to another file is known as \_\_all\_\_ in Python. The variables which are declared in that list can only be used in another file after importing this file, the rest variables if called will throw an error.

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

Ans:

* Every Python module has a specified **\_\_name\_\_** and if this is '**\_\_main\_\_'**, it indicates that the module is being **executed independently by the user,** allowing us to take the necessary action.
* The **\_\_name\_\_ is assigned to the name of the script/module** if we acquire this script as a module in another script.
* Python files can function as **independent applications** or as **reusable modules**.
* If the file was executed directly, without being imported, then the statement **"if \_\_name\_\_ == "main"** is used to run some code.

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: Reverse Polish notation (RPN) is a method for conveying mathematical expressions without the use of separators such as brackets and parentheses. In this notation, the operators follow their operands, hence removing the need for brackets to define evaluation priority. The operation is read from left to right but execution is done every time an operator is reached, and always using the last two numbers as the operands. This notation is suited for computers and calculators since there are fewer characters to track and fewer operations to execute.Reverse Polish notation is also known as postfix notation. **The 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.

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:xxxxxxxxxxxxxxx