**EXPERIMENT 5**

**AIM:** Python program to explore different types of Modules

**THEORY:**

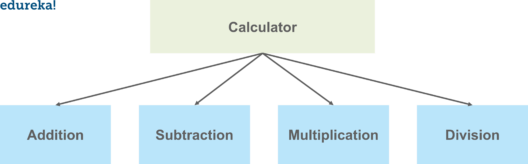
In Python, **Modules** are simply files with the “.py” extension containing Python code that can be imported inside another Python Program. A module can be considered as a code library or a file that contains a set of functions that you want to include in your application. With the help of modules, we can organize related functions, classes, or any code block in the same file.

Modular programming refers to the process of breaking a large, unwieldy programming task into separate, smaller, more manageable subtasks or modules.

The module contains the following components:

* Definitions and implementation of classes,
* Variables, and
* Functions that can be used inside another program.

Suppose we want to make an application for a calculator. We want to include few operations in our application such as addition, subtraction, multiplication, division, etc. Now, we will break the code into separate parts and simply create one module for all these operations or separate modules for each of the operations. And then we can call these modules in our main program logic.



**Types of Models:**

1. A module can be written in Python itself.
2. A module can be written in C and loaded dynamically at run-time, like the re (regular expression) module.
3. A built-in module is intrinsically contained in the interpreter, like the itertools module**.**

**Create and Access Python Modules**

To create a module, we have to save the code in a file with the file extension **“.py”**. Then, the name of the Python file becomes the name of the module. The module’s contents are accessed with the import statement.

**Advantages of Modules**

* Reusability: Working with modules makes the code reusable.
* Simplicity: The module focuses on a small proportion of the problem, rather than focusing on the entire problem.
* Scoping: A separate namespace is defined by a module that helps to avoid collisions between identifiers.

**CODE:**

**welcome.py - (MODULE)**

def welcome(name):

print("Hello, " + name +" how are you??")

**module.py - (MAIN FILE)**

from welcome import welcome

welcome("Meith Navlakha")

**OUTPUT:**

Text

Description automatically generated

**CODE:**

**person.py (MODULE)**

person1 = {

"name": "Meith Navlakha",

"age": 21,

"city": "India",

"college": "D. J. Sanghvi"

}

**Module.py (MAIN FILE):**

import person

a = person.person1["age"]

b = person.person1["name"]

c = person.person1["college"]

print("\nNAME : ", b)

print("AGE : ",a)

print("COLLEGE : ",c)

**OUTPUT:**

Text

Description automatically generated

**Rename a Python Module:**

We can name the file of the module whatever you like, but we have to note that it must have the file extension **“.py”**. To rename the module name, we can create an alias when you import a module, with the help of the **as keyword**

**CODE:**

import person as mod

a = mod.person1["age"]

b = mod.person1["name"]

c = mod.person1["college"]

print("\nNAME : ", b)

print("AGE : ",a)

print("COLLEGE : ",c)

**OUTPUT:**

Graphical user interface, text

Description automatically generated with medium confidence

**Python Built-in Modules**

Python has a number of built-in functions. The methods are loaded automatically and are always available, such as,

* print() and input() for I/O,
* Number conversion functions such as int(), float(), complex(),
* Data type conversions such as list(), tuple(), set(), etc.

In addition to these many built-in functions, there are also a large number of pre-defined functions available as a part of libraries bundled with Python distributions. These functions are defined in modules which are known as **built-in modules**.

These built-in modules are written in C language and integrated with the Python shell. Most useful and frequently used built-in modules of Python.

* Math Module
* Statistics Module

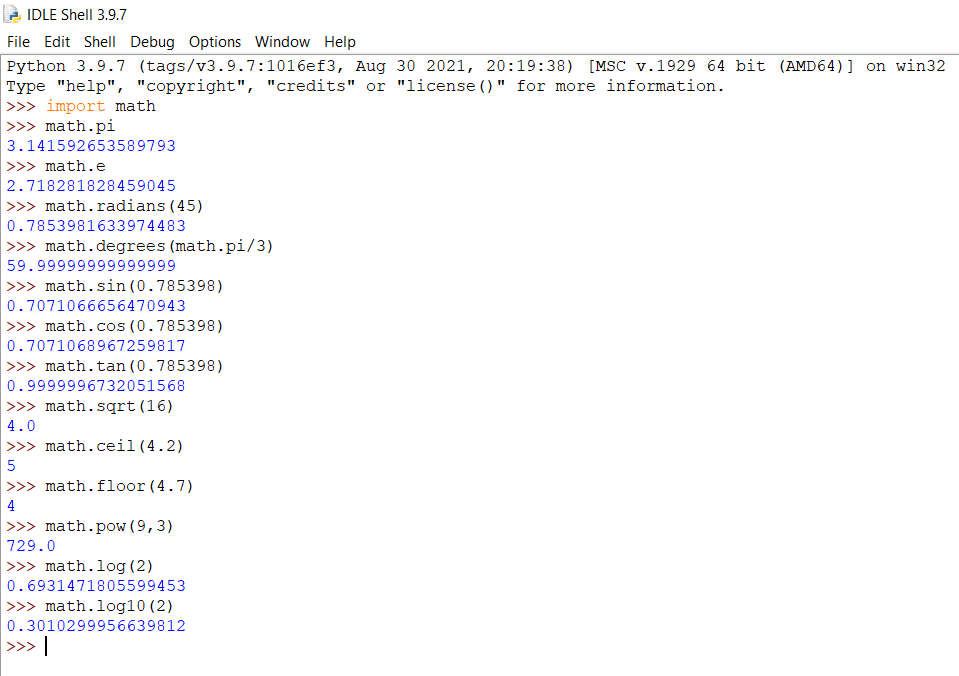
**Math Module of Python**

Some of the most popular mathematical functions that are defined in the math module include,

* Trigonometric functions,
* Representation functions,
* Logarithmic functions,
* Angle conversion functions, etc.

In addition, two mathematical constants- **pi** and **e** are also defined in this module.

**CODE:**

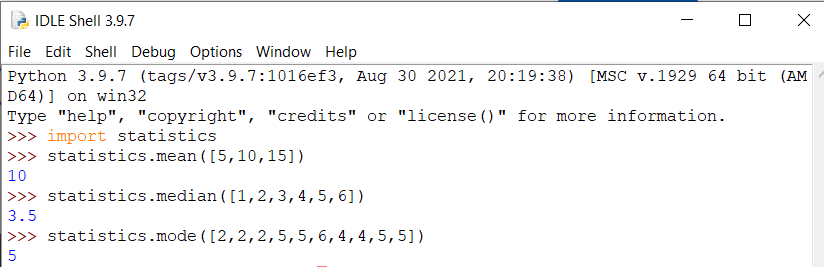


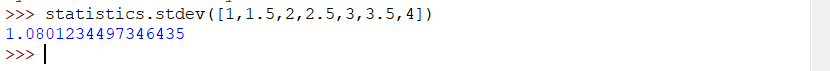
**Statistics Module of Python**

The statistics module provides functions to mathematical statistics of numeric data. Some of the popular statistical functions are defined in this module are as follows:

* Mean
* Median
* Mode
* Standard Deviation

**CODE**





**Datetime Module of Python**

In Python, date and time are not a data type of their own, but a module named **datetime** can be imported to work with the date as well as time. **Python Datetime module** comes built into Python, so there is no need to install it externally.

Python Datetime module supplies classes to work with date and time. These classes provide a number of functions to deal with dates, times and time intervals. Date and datetime are an object in Python, so when you manipulate them, you are actually manipulating objects and not string or timestamps.

The DateTime module is categorized into 6 main classes –

* [**date**](https://www.geeksforgeeks.org/python-datetime-date-class/) : An idealized naive date, assuming the current Gregorian calendar always was, and always will be, in effect. Its attributes are year, month and day.
* [**time**](https://www.geeksforgeeks.org/python-datetime-time-class/) : An idealized time, independent of any particular day, assuming that every day has exactly 24\*60\*60 seconds. Its attributes are hour, minute, second, microsecond, and tzinfo.
* [**datetime**](https://www.geeksforgeeks.org/python-datetime-datetime-class/) : Its a combination of date and time along with the attributes year, month, day, hour, minute, second, microsecond, and tzinfo.
* [**timedelta**](https://www.geeksforgeeks.org/python-datetime-timedelta-class/) : A duration expressing the difference between two date, time, or datetime instances to microsecond resolution.
* **tzinfo** : It provides time zone information objects.
* **timezone** : A class that implements the tzinfo abstract base class as a fixed offset from the UTC

**CODE:**

**1) Basic**

from datetime import date

Date = date(2021,1,12)

print("Date: ", Date)

**OUTPUT:**

Text

Description automatically generated

**2) today()**

from datetime import date

today = date.today()

print("Today's date is", today)

**OUTPUT:**

Text

Description automatically generated

**3) Month, Year, Day**

from datetime import date

today = date.today()

print("Current year:", today.year)

print("Current month:", today.month)

print("Current day:", today.day)

**OUTPUT:**

Text

Description automatically generated

**4) Get date from Timestamp**

from datetime import datetime

date\_time = datetime.fromtimestamp(1887639468)

print("Datetime from timestamp:", date\_time)

**OUTPUT:**

A screenshot of a computer

Description automatically generated with medium confidence

**5)Convert Date to String :** Can convert date object to a string representation using two functions isoformat() and strftime().

from datetime import date

today = date.today()

Str = date.isoformat(today)

print("String Representation", Str)

print(type(Str))

**OUTPUT:**

Text

Description automatically generated

**6) Time object representing time in Python**

from datetime import time

my\_time = time(13, 24, 56)

print("Entered time", my\_time)

my\_time = time(minute=12)

print("\nTime with one argument", my\_time)

my\_time = time()

print("\nTime without argument", my\_time)

**OUTPUT:**

A screenshot of a computer

Description automatically generated with medium confidence

**6) Get hours, minutes, seconds, and microseconds**

from datetime import time

Time = time(12, 15, 47)

print("hour =", Time.hour)

print("minute =", Time.minute)

print("second =", Time.second)

print("microsecond =", Time.microsecond)

**OUTPUT:**

Graphical user interface, text

Description automatically generated

**7)Add days to datetime object**

from datetime import datetime, timedelta

ini\_time\_for\_now = datetime.now()

print("initial\_date", str(ini\_time\_for\_now))

future\_date\_after\_2yrs = ini\_time\_for\_now + timedelta(days=730)

future\_date\_after\_2days = ini\_time\_for\_now + timedelta(days=2)

print('future\_date\_after\_2yrs:', str(future\_date\_after\_2yrs))

print('future\_date\_after\_2days:', str(future\_date\_after\_2days))

**OUTPUT:**

Text

Description automatically generated

**8) Difference between two date and times**

from datetime import datetime, timedelta

ini\_time\_for\_now = datetime.now()

print("initial\_date", str(ini\_time\_for\_now))

new\_final\_time = ini\_time\_for\_now + timedelta(days=5)

print("new\_final\_time", str(new\_final\_time))

print('Time difference:', str(new\_final\_time -ini\_time\_for\_now))

**OUTPUT:**

Text

Description automatically generated

**CONCLUSION:** In this experiment I have successfully studied and implemented different types of modules in python. Python has 3 times of modules: module in Python itself, module can be written in C and loaded dynamically and built-in module. I have created my module welcome.py and imported it in the module.py file. Also, in this experiment I have implemented some built-in modules like statistics, time and math.