External & Built-in Modules in Python

Python provides a rich ecosystem of both built-in modules and external libraries (also known as packages) that you can use to enhance your programs. Here's a brief overview of both:

Built-in Modules:

Python comes with a set of modules that are available out of the box without the need for installing anything extra. These include modules for common tasks like working with files, handling dates and times, performing mathematical operations, and more. Some examples of built-in modules are:

1. `os`: Provides a way to interact with the operating system, including file operations, directory operations, etc.
2. `datetime`: Allows manipulation of dates and times.
3. `math`: Provides mathematical functions.
4. `random`: Enables generation of random numbers.
5. `json`: Allows encoding and decoding JSON data.

External Modules:

In addition to the built-in modules, Python has a vast ecosystem of external libraries that you can install using tools like pip (Python's package manager). These libraries cover a wide range of functionalities and can be easily integrated into your projects. Some popular external libraries include:

1. `numpy`: A powerful library for numerical computing, especially for working with arrays and matrices.
2. `pandas`: A data manipulation and analysis library, useful for working with structured data.
3. `matplotlib` and `seaborn`: Libraries for creating data visualizations and plots.
4. `requests`: A library for making HTTP requests, useful for interacting with web APIs.
5. `tensorflow` and `pytorch`: Libraries for machine learning and deep learning.

To use external modules, you typically need to install them first using pip. For example, to install the `requests` module, you would run:

```bash
pip install requests
```

Once installed, you can import and use these modules in your Python scripts just like you would with built-in modules:

```python
import requests

response = requests.get('https://api.example.com/data')
print(response.json())
```

By leveraging both built-in and external modules, you can extend Python's capabilities to suit your specific needs and build powerful applications.

CODE:

```python
import random
random_number = random.randint(a: 0, b: 1)
#print(random_number)
rand = random.random()*100
#print(rand)
First = ["Star Plus","DD1", "Aaj Tak", " pentest diaries"]
choice = random.choice(First)
print(choice)
```

Output:

```
/root/PycharmProjects/pythonProject/.venv/bin/python /root/PycharmProjects/pythonProject/22.py
Aaj Tak

Process finished with exit code 0
```

F-Strings & string format in Python

F-strings and string formatting are powerful tools in Python for creating formatted strings. Here's an overview of both:

F-Strings (Formatted String Literals):

Introduced in Python 3.6, f-strings provide a concise and readable way to embed expressions inside string literals. You can use f-strings by prefixing a string literal with `f` or `F` and then including expressions within curly braces `{}`. These expressions are evaluated at runtime and their values are inserted into the string.

Example:

```python
name = "Alice"
age = 30
greeting = f"Hello, my name is {name} and I am {age} years old."
print(greeting)
```

Output:

```
Hello, my name is Alice and I am 30 years old.
```

F-strings support various formatting options such as specifying precision for floating-point numbers, padding strings, and more.

String Formatting with `str.format()`:

Before the introduction of f-strings, `str.format()` was the preferred method for string formatting in Python. With `str.format()`, you create a string template with placeholder `{}` for the values you want to insert. You then call the `format()` method on the string and pass the values to be inserted as arguments.

Example:

```python
name = "Bob"
age = 25
greeting = "Hello, my name is {} and I am {} years old.".format(name, age)
print(greeting)
```

Output:

```
Hello, my name is Bob and I am 25 years old.
```

You can also specify the position of the arguments or use keyword arguments for more clarity and flexibility.

Example:

```
name = "Charlie"
age = 35
greeting = "Hello, my name is {name} and I am {age} years old.".format(name=name, age=age)
print(greeting)
```

Output:

```
Hello, my name is Charlie and I am 35 years old.
```

Both f-strings and `str.format()` offer similar functionality, but f-strings are often preferred for their readability and simplicity, especially when working with Python 3.6 and later versions. However, `str.format()` remains useful in cases where you need more control over formatting or when working with older versions of Python.

CODE:

```python
import math
me = "pentest"
a1 = 10

test = f"this is {me} {a1} {math.cos(100)}"
print(test)
```

Output:

```
/root/PycharmProjects/pythonProject/.venv/bin/python /root/PycharmProjects/pythonProject/23.py
this is pentest 10 0.8623188722876839

Process finished with exit code 0
```