SOME IMPORTANT MODULES IN PYTHON

sys :- Provides access to system specific parameters and functions. It's a built-in module, so you don't need to install anything extra to use it.

Some important Functions:-

1. **sys.argv :-** sys.argv is a list in the sys module that contains the command-line arguments passed to a Python script.
   - **List of Strings :-** sys.argv is a list where each element is a string. The strings represent command-line arguments provided when executing the script.
   - **Script Name :-** The first element (sys.argv[0]) is always the name of the script being executed. This is the file name of the script itself.
   - **Command-Line Arguments:-** Subsequent elements (sys.argv[1:]) are the arguments provided after the script name. These are the arguments you pass to the script when you run it.

Example:-

```
        import sys
        print("Arguments:", sys.argv)
```

Output:-

```
Arguments: ['/usr/local/lib/python3.10/dist-
packages/colab_kernel_launcher.py', '-f',
'/root/.local/share/jupyter/runtime/kernel-77b715d3-ab65-404d-a138-
b713261da856.json']
```

Note:- The output you're seeing indicates that the sys.argv list contains the command-line arguments passed to the script you're running. Specifically, it looks like you're running a script in a Jupyter environment or some environment that uses Colab or Jupyter kernels.

2. **sys.exit([arg])**:- Exits from Python. The optional argument can be an integer (which will be the exit status) or an object (which will be printed as an error message).
   - **Exit Status :-** If arg is an integer, it is used as the exit status code. By convention, an exit status of 0 indicates success, and any non-zero value indicates an error or abnormal termination**.** Exit status codes are typically used by operating systems and other programs to determine whether a script completed successfully or encountered an error.
   - **Error Message:-** If arg is a string (or any other non-integer type), it is treated as an error message. This message is printed to the standard error stream (stderr), and the exit status is set to 1, which generally indicates an error or failure.
   - **No Argument**:- If no argument is provided, sys.exit() behaves as if arg was 0, indicating a successful termination.

Example:-

```
#Exiting with Success Status (0)
        import sys
        try:
          sys.exit(0)
      except SystemExit as e:
          print("Caught SystemExit exception")

Output:- Caught SystemExit exception

# Exiting with an Error Status (non-zero integer):
      import sys
      try:
          sys.exit(1)  # Simulate an error condition
      except SystemExit as e:
          print("Exiting with code:", e.code)

Output:- Exiting with code: 1

# Exiting with an Error Message (string):
        import sys
        sys.exit("An error occurred!")

Output:- Will give an error
```

Note:- In Jupyter notebooks or other interactive environments (like Colab), sys.exit() might not work as expected because these environments handle kernel processes differently. They may prevent sys.exit() from terminating the kernel or might not support it as a standard exit mechanism.

3. **<u>sys.path</u>**<u>:-</u> A list of strings that specifies the search path for modules. It's initialized from the PYTHONPATH environment variable and can be modified at runtime

```
Example:-

# Checking Current Path
      import sys
      print(sys.path) '"This prints the list of directories Python is currently
searching for modules."'

# Adding Directories
      import sys
      sys.path.append('/path/to/your/module')

# Removing Directories
      import sys
      sys.path.remove('/path/to/remove')
```

4. **<u>sys.version</u>**<u>:-</u> A string that contains the version number of the Python interpreter.

```
Example:-
      import sys
      print(sys.version)
```

```
Output:- ‘”Depends upon the version you are using”’
```

5. **sys.modules**:- A dictionary that maps module names to modules that have already been loaded. Useful for inspecting or modifying module imports.

```
Example:-
        import sys
        print(sys.modules.keys())
```

os :- Provides a way of interacting with operating system. It offers functions to work with files and directories, environment variables, and other OS-specific functionality.

```
import os

# Current working directory
print("Current working directory:", os.getcwd())

# List files and directories
print("Files and directories:", os.listdir('.'))

# Create a new directory
os.mkdir('test_directory')

# Change to the new directory
os.chdir('test_directory')

# Create a new file
with open('example.txt', 'w') as f:
    f.write('Hello, world!')

# List files in the new directory
print("Files in test_directory:", os.listdir('.'))

# Get absolute path of the file
print("Absolute path of example.txt:", os.path.abspath('example.txt'))

#Rename the file or directory
os.rename('example.txt', 'new_example.txt')

# Remove the file and directory
os.remove('new_example.txt')
```

math :- The math module in Python provides mathematical functions and constants that are not available in the basic set of Python operators. It is part of Python's standard library and is commonly used for performing mathematical calculations and operations.

```
import math

# Constants
```

```
print("Pi:", math.pi)   #Pi: 3.141592653589793
print("Euler's number:", math.e)  #Euler's number: 2.718281828459045

# Square root
print("Square root of 16:", math.sqrt(16))  #Square root of 16: 4.0

# Trigonometric functions
print("Sine of pi/2:", math.sin(math.pi / 2))  #Sine of pi/2: 1.0
print("Cosine of 0:", math.cos(0))  #Cosine of 0: 1.0

# Logarithms
print("Log base 10 of 100:", math.log10(100))  #Log base 10 of 100: 2.0
print("Natural log of e:", math.log(math.e))  #Natural log of e: 1.0

# Factorial
print("Factorial of 5:", math.factorial(5))   #Factorial of 5: 120

# Angle conversions
print("180 degrees in radians:", math.radians(180))   #180 degrees in radians: 3.141592653589793
print("Pi radians in degrees:", math.degrees(math.pi))  #Pi radians in degrees: 180.0
```

<u>DateTime :-</u> Provides classes for manipulating dates and times in both simple and complex way.

```
from datetime import datetime, date, time, timedelta

# Date example
d = date(2024, 8, 9)   #Handles date-only values.
print("Date:", d)    #Date: 2024-08-09

# Time example
t = time(14, 30)   #Handles time-only values
print("Time:", t)   #Time: 14:30:00

# Datetime example
dt = datetime(2024, 8, 9, 14, 30)  #Handles date and time together.
print("Datetime:", dt)  #Datetime: 2024-08-09 14:30:00

# Current date and time
now = datetime.now()
print("Current datetime:", now)

# timedelta
delta = timedelta(days=5, hours=3)  # Represents a duration or difference between two dates/times.
print("Timedelta:", delta)  #Timedelta: 5 days, 3:00:00

# Adding a timedelta to a datetime
delta = timedelta(days=10, hours=5)
future_dt = now + delta
```

```
print("Future datetime:", future_dt) #Future datetime: 2024-08-19 10:45:08.455878

# Combining date and time
combined = datetime.combine(d, t)
print("Combined datetime:", combined)  #Combined datetime: 2024-08-09 14:30:00

# Time difference
time_diff = future_dt - now
print("Time difference:", time_diff)  #Time difference: 10 days, 5:00:00

# Parse a datetime from an ISO format string
# ISO format: 'YYYY-MM-DDTHH:MM:SS'
parsed_dt = datetime.fromisoformat('2024-08-09T14:30:00')
print("Parsed datetime:", parsed_dt)  #Parsed datetime: 2024-08-09 14:30:00
```

Random :- For generating random numbers and performing random selections. It is part of Python's standard library and is commonly used for tasks involving randomness, such as simulations, games, and testing.

```
import random

# Random float between 0.0 and 1.0
# Generates a random float number where 0.0 <= number < 1.0
print("Random float between 0 and 1:", random.random())
#output: Random float between 0 and 1: 0.3744487175646646

# Random float between 1 and 10
# Generates a random float number where 1 <= number <= 10
print("Random float between 1 and 10:", random.uniform(1, 10))
#output: Random float between 1 and 10: 7.853589681161471

# Random integer between 1 and 10
# Generates a random integer number where 1 <= number <= 10
print("Random integer between 1 and 10:", random.randint(1, 10))
#output: Random integer between 1 and 10: 8

# Random choice from a list
# Chooses a random element from the provided list
choices = ['apple', 'banana', 'cherry']
print("Random choice from list:", random.choice(choices))
#output: Random choice from list: banana

# Random choices with replacement
# Chooses k elements from the list with replacement (elements may be repeated)
print("Random choices with replacement:", random.choices(choices, k=2))
#output: Random choices with replacement: ['cherry', 'apple']

# Random sample without replacement
# Chooses k unique elements from the list without replacement (no repeated elements)
print("Random sample without replacement:", random.sample(choices, 2))
```

```
#output: Random sample without replacement: ['banana', 'apple']

# Shuffle a list
# Shuffles the list in place, changing the order of its elements randomly
items = [1, 2, 3, 4, 5]
random.shuffle(items)
print("Shuffled list:", items)
#output: Shuffled list: [3, 1, 4, 5, 2]

# Set the seed for reproducibility
# Initializes the random number generator with a specific seed value
random.seed(42)
print("Random number with seed 42:", random.random())
#output: Random number with seed 42: 0.6394267984578837
```