Packages and Modules

A module is a single file containing Python code, while a package is a collection of related modules organized in a directory hierarchy. Packages provide a way to manage modules and sub-packages

Module:

Definition: A module is a single file containing Python code. It can define functions, classes, and variables.

Purpose: Modules provide a way to organize Python code logic into reusable files. Each module can be a library of functions and classes.

Usage: Modules are imported into other Python scripts using the import statement. For example, if you have a module named example_module.py, you can import it into another Python file like this: import example_module.

Example: example_module.py could contain functions like calculate_area() and calculate_perimeter(), which can be used in other scripts by importing the module.

Package:

Definition: A package is a way of organizing related modules into a single directory hierarchy. It contains multiple module files and a special file called __init__.py.

Purpose: Packages are used to organize modules into a directory hierarchy. They allow for better structuring of large Python applications by grouping related functionality together.

Usage: Packages are imported similar to modules but can contain sub-modules. For example, if you have a package named example_package with modules module1.py and module2.py, you can import them into another Python file like this: from example_package import module1, module2.

example_package/

├── __init__.py

├── module1.py

└── module2.py

Python comes with a wide range of built-in modules and packages that provide various functionalities.

Built-in Modules:

math: Mathematical functions and constants.

datetime: Date and time manipulation.

random: Generate pseudo-random numbers.

os: Operating system interfaces for file operations and directory manipulation.

sys: System-specific parameters and functions.

json: JSON encoding and decoding.

re: Regular expressions.

collections: Additional data structures like deque, OrderedDict, and defaultdict.

urllib: URL handling utilities (part of the standard library, split into submodules like urllib.request and urllib.parse).

csv: CSV file reading and writing.

sqlite3: SQLite database interface.

pickle: Object serialization and deserialization.

subprocess: Spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

argparse: Command-line argument parsing.

logging: Flexible event logging system for applications.

threading: Thread-based parallelism.

multiprocessing: Process-based parallelism.

socket: Low-level networking interface.

email: Package for managing email messages, including MIME and other RFC 2822-based message documents.

http: HTTP client and server classes (e.g., http.client, http.server).

Built-in Packages:

os.path: Submodule of os, provides methods for common pathname manipulations.

sys.path: A list of directories Python looks in for finding modules.

collections: Additional data structures like namedtuple and Counter.

queue: Implements multi-producer, multi-consumer queues.

multiprocessing: Process-based parallelism, including process pools and inter-process communication (IPC) tools.

http.server: Basic HTTP server classes for implementing web servers.

http.client: HTTP protocol client (low-level).

xml: Package for working with XML (e.g., xml.etree.ElementTree).

tkinter: Standard Python interface to the Tk GUI toolkit.

unittest: Unit testing framework.

doctest: Module for running docstring-based tests.

argparse: Command-line argument parsing (part of the standard library, often used as a standalone module).

38) Write a python program to illustrate the usage of built in package math.

```
import math
# Calculate square root
num = 16
print("Square root of", num, "is", math.sqrt(num))
# Calculate factorial
num = 5
print("Factorial of", num, "is", math.factorial(num))
# Calculate trigonometric functions
angle = math.radians(30)  # Convert degrees to radians
print("Sin(30 degrees):", math.sin(angle))
print("Cos(30 degrees):", math.cos(angle))
```

39) Write a python program to illustrate the usage of built in package datetime.

```
from datetime import datetime, timedelta

# Get current date and time
current_time = datetime.now()
print("Current Date and Time:", current_time)

# Calculate future date
days_to_add = 7
future_date = current_time + timedelta(days=days_to_add)
print("Date after", days_to_add, "days:", future_date)
```

40) Write a python program to illustrate the usage of built in package random.

```
import random
# Generate a random integer between 1 and 100
random_integer = random.randint(1, 100)
print("Random Integer:", random_integer)
```

```
# Generate a random floating-point number between 0 and 1
random_float = random.random()
print("Random Float between 0 and 1:", random_float)

# Shuffle a list
my_list = [1, 2, 3, 4, 5]
random.shuffle(my_list)
print("Shuffled List:", my_list)
```

41) Write a python program to illustrate the usage of built in package json.

Note: The json package in Python is used to work with JSON (JavaScript Object Notation) data. JSON is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. Python's json package provides methods to encode Python objects into JSON format and decode JSON data back into Python objects

```
import json

# Dictionary to JSON
data = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

json_data = json.dumps(data, indent=4)  # Convert dictionary to JSON string with indentation
print("JSON Data:")
print(json_data)

# JSON to Dictionary
json_string = '{"name": "Alice", "age": 25, "city": "London"}'
parsed_data = json.loads(json_string)
print("Parsed Data:")
print(parsed_data)
```

```
# JSON string
json_string = '{"name": "Alice", "age": 25, "city": "London"}'

# Convert JSON string to Python dictionary
parsed_data = json.loads(json_string)
print("Parsed Data:", parsed_data)
```

42) Write a python program to Write to JSON file:

Q)Create a package graphics with modules rectangle, circle and sub-package 3D-graphics with modules cuboid and sphere. Include methods to find area and perimeter of respective figures in each module. Write programs that finds area and perimeter of figures by different importing statements. (Include selective import of modules and import * statements)

To create the specified package structure and modules for the graphics package and the 3D-graphics sub-package, you can follow these steps:

Create a directory structure:

```
graphics_package/
├── graphics/
│   ├── __init__.py
│   ├── rectangle.py
│   └── circle.py
├── 3D_graphics/
│   ├── __init__.py
│   ├── cuboid.py
│   └── sphere.py
└── main.py
```

Implement the modules:

rectangle.py:

```
def area(length, width):
    return length * width

def perimeter(length, width):
    return 2 * (length + width)
```

circle.py:

```
import math

def area(radius):
    return math.pi * radius ** 2

def perimeter(radius):
    return 2 * math.pi * radius
```

cuboid.py:

```
def area(length, width, height):
    return 2 * (length * width + width * height + height * length)

def volume(length, width, height):
    return length * width * height
```

sphere.py:

```
import math

def area(radius):
    return 4 * math.pi * radius ** 2

def volume(radius):
    return (4 / 3) * math.pi * radius ** 3
```

Importing statements in main.py:

Selective import:

```
from graphics.rectangle import area as rect_area, perimeter as
rect_perimeter

from graphics.circle import area as circle_area, perimeter as
circle_perimeter

from graphics.3D_graphics.cuboid import area as cuboid_area, volume
as cuboid_volume

from graphics.3D_graphics.sphere import area as sphere_area, volume
as sphere_volume


rect_length, rect_width = 5, 10

circle_radius = 7

cuboid_length, cuboid_width, cuboid_height = 3, 4, 5

sphere_radius = 6


print("Rectangle Area:", rect_area(rect_length, rect_width))

print("Rectangle Perimeter:", rect_perimeter(rect_length,
rect_width))

print("Circle Area:", circle_area(circle_radius))

print("Circle Perimeter:", circle_perimeter(circle_radius))

print("Cuboid Area:", cuboid_area(cuboid_length, cuboid_width,
cuboid_height))

print("Cuboid Volume:", cuboid_volume(cuboid_length, cuboid_width,
cuboid_height))

print("Sphere Area:", sphere_area(sphere_radius))

print("Sphere Volume:", sphere_volume(sphere_radius))
```

Import everything (*):

```
from graphics.rectangle import *

from graphics.circle import *

from graphics.3D_graphics.cuboid import *

from graphics.3D_graphics.sphere import *


rect_length, rect_width = 5, 10
```

```
circle_radius = 7

cuboid_length, cuboid_width, cuboid_height = 3, 4, 5

sphere_radius = 6


print("Rectangle Area:", area(rect_length, rect_width))

print("Rectangle Perimeter:", perimeter(rect_length, rect_width))

print("Circle Area:", area(circle_radius))

print("Circle Perimeter:", perimeter(circle_radius))

print("Cuboid Area:", area(cuboid_length, cuboid_width,
cuboid_height))

print("Cuboid Volume:", volume(cuboid_length, cuboid_width,
cuboid_height))

print("Sphere Area:", area(sphere_radius))

print("Sphere Volume:", volume(sphere_radius))
```

In the first approach, specific functions are imported using the from module import function syntax. In the second approach, all functions from the modules are imported using the from module import * syntax. Choose the appropriate import method based on your needs and programming best practices.