Multi-threading and Multi-processing in Python

This report covers the concepts of multi-threading and multi-processing in Python, including code examples and explanations.

1. Multi-threading

```
import threading
import time

def print_numbers():
    for i in range(5):
        time.sleep(2)
        print(f"Number:{i}")

def print_letter():
    for letter in "abcde":
        time.sleep(2)
        print(f"Letter:{letter}")

# create 2 threads
t1=threading.Thread(target=print_numbers)
t2=threading.Thread(target=print_letter)

t=time.time()

# start the thread
t1.start()
t2.start()

# wait for the threads to complete
t1.join()
t2.join()

finished_time = time.time() - t
print(finished_time)
```

In this code, we create two threads: one for printing numbers and another for printing letters. Each thread sleeps for a specified time to simulate I/O-bound tasks. The use of threads allows the program to perform these tasks concurrently, improving throughput.

Built-in Function Definitions:

1. start(): This method is called on a thread object to start the thread's activity. It invokes the run() method in a separate thread of control.

2. join(): This method is called on a thread object to block the calling thread until the thread whose join() method is called is terminated.

3. sleep(seconds): This function from the time module suspends execution for the given number of seconds. It is used to simulate a delay in the execution of the program.

4. print(): This built-in function outputs the specified message to the console or other standard output device.

2. Multi-processing

```
import multiprocessing
import time

def square_numbers():
    for i in range(5):
        time.sleep(1)
        print(f"Square: {i*i}")

def cube_numbers():
    for i in range(5):
        time.sleep(1.5)
        print(f"Cube: {i*i*i}")

if __name__ == '__main__':
    # create 2 processes
    p1=multiprocessing.Process(target=square_numbers)
    p2=multiprocessing.Process(target=cube_numbers)
    t=time.time()

    # start the process
    p 1.start()
    p2.start()

    # Wait for the process to complete
    p1.join()
    p2.join()

    finished_time = time.time() - t
    print(finished_time)
```

In this code, we create two processes: one for calculating squares and another for cubes. Each process runs independently, allowing for parallel execution on multiple CPU cores. This is particularly useful for CPU-bound tasks.

Built-in Function Definitions:

1. Process(): This class from the multiprocessing module is used to create a new process. It takes a target function as an argument to specify what the process should execute.

2. start(): Similar to the threading module, this method starts the process's activity, invoking the target function in a separate process.

3. join(): This method blocks the calling thread until the process whose join() method is called is terminated.

4. sleep(seconds): This function from the time module suspends execution for the given number of seconds, used to simulate delays.

5. print(): This built-in function outputs the specified message to the console or other standard output device.

In conclusion, multi-threading is suitable for I/O-bound tasks, while multi-processing is ideal for CPU-bound tasks. Understanding these concepts can help optimize performance in Python applications.