**PW SKILLS**

**MULTITHREADING**

1. \*\*Multithreading in Python\*\*:

- Multithreading in Python refers to the ability of a program to execute multiple threads concurrently.

- It is used to perform multiple tasks simultaneously and make efficient use of CPU resources, especially in I/O-bound applications.

- The module used to handle threads in Python is the `threading` module.

- The `threading` module provides a higher-level interface for working with threads compared to the lower-level `thread` module.

2. \*\*Functions in the threading module\*\*:

- `activeCount()`: Returns the number of Thread objects currently alive.

- `currentThread()`: Returns the current Thread object.

- `enumerate()`: Returns a list of all Thread objects currently alive.

3. \*\*Explanation of Thread-related functions\*\*:

- `run()`: This method represents the code to be executed by the thread when it is started. It should be overridden in subclasses to implement the desired behavior.

- `start()`: This method starts the execution of the thread's `run()` method.

- `join()`: This method blocks the calling thread until the thread whose `join()` method is called terminates.

- `isAlive()`: This method returns `True` if the thread is alive and `False` otherwise.

4. \*\*Python program to create two threads\*\*:

```python

import threading

def print\_squares():

squares = [i \*\* 2 for i in range(1, 6)]

print("Squares:", squares)

def print\_cubes():

cubes = [i \*\* 3 for i in range(1, 6)]

print("Cubes:", cubes)

thread1 = threading.Thread(target=print\_squares)

thread2 = threading.Thread(target=print\_cubes)

thread1.start()

thread2.start()

thread1.join()

thread2.join()

print("Main thread exiting")

```

5. \*\*Advantages and disadvantages of multithreading\*\*:

- \*\*Advantages\*\*:

- Increased responsiveness: Multithreading allows a program to remain responsive to user input even when performing intensive tasks.

- Improved performance: Multithreading can improve performance by utilizing multiple CPU cores simultaneously.

- Simplified coding: Multithreading can simplify coding for complex applications by breaking them down into smaller, manageable threads.

- \*\*Disadvantages\*\*:

- Complexity: Multithreading introduces complexity due to issues such as race conditions and deadlocks, making debugging and maintenance more challenging.

- Resource overhead: Each thread consumes system resources such as memory and CPU time, leading to potential resource contention and overhead.

- Synchronization overhead: Synchronizing access to shared resources among threads can introduce overhead and reduce performance.

6. \*\*Deadlocks and race conditions\*\*:

- \*\*Deadlocks\*\*: Deadlocks occur when two or more threads are blocked indefinitely, waiting for each other to release resources that they need. This typically happens when multiple threads acquire locks in different orders.

- \*\*Race conditions\*\*: Race conditions occur when the behavior of a program depends on the timing or interleaving of multiple threads, and the outcome is unpredictable. Race conditions can lead to incorrect results or unexpected behavior.