MultiThreading in Python?

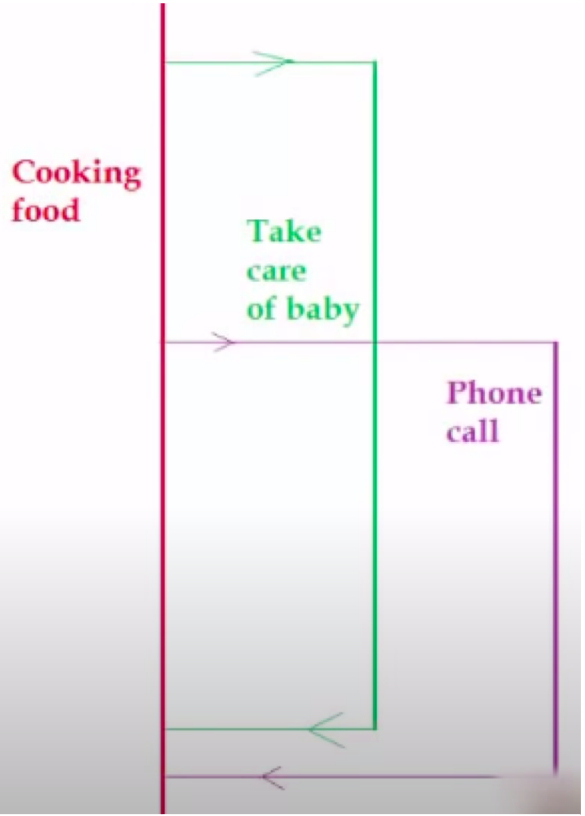
Ashish KC
Akshay Mawle

What is Thread?

- In Computer Science, threads are defined as the smallest unit of work which is scheduled to be done by an Operating System.
- Some points to consider about Threads are:
  1. Threads exists inside a process.
  2. Multiple threads can exist in a single process.
  3. Threads in same process share the state and memory of the parent process.

What is Multithreading?

- Multithreading is the ability of a program or an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer.
- Each user request for a program or system service (and here a user can also be another program) is kept track of as a thread with a separate identity. As programs work on behalf of the initial request for that thread and are interrupted by other requests, the status of work on behalf of that thread is kept track of until the work is completed.


Cooking
food
Take
care
of baby
Phone
call

What is Multithreading in Python?

- Python does not have multithreading because of GIL(Global Interpreter Lock)
- However, in Python, threading can be used to run multiple threads (tasks, function calls) at the same time.
- Note that this does not mean that they are executed on different CPUs.
- Threading allows python to execute other code while waiting; this is easily simulated with the sleep function.

Python Multithreading Modules

Python offers two modules to implement threads in programs.

- <*thread*> module and
- <*threading*> module.

Note:Python 2.x used to have the <*thread*> module. But it got deprecated in Python 3.x and renamed to <_*thread*> module for backward compatibility.

The principal difference between the two modules is that the module <_*thread*> implements a thread as a function. On the other hand, the module <*threading*> offers an object-oriented approach to enable thread creation.

For a given list of numbers print square and cube of every numbers

For example:

Input: [2,3,8,9]

Output: square list - [4, 9, 64, 81]
cube list - [8, 27, 512, 729]

```python
import time
def cal_square(numbers):
    print("calculate square numbers")
    for n in numbers:
        time.sleep(0.2)
        print("square: ",n*n)

def cal_cube(numbers):
    print("calculate cube numbers")
    for n in numbers:
        time.sleep(0.2)
        print("cube: ",n*n*n)

arr=[2,3,8,9]
t=time.time()
cal_square(arr)
cal_cube(arr)
print("Done in: ", time.time()-t)
print("I am done.")
```

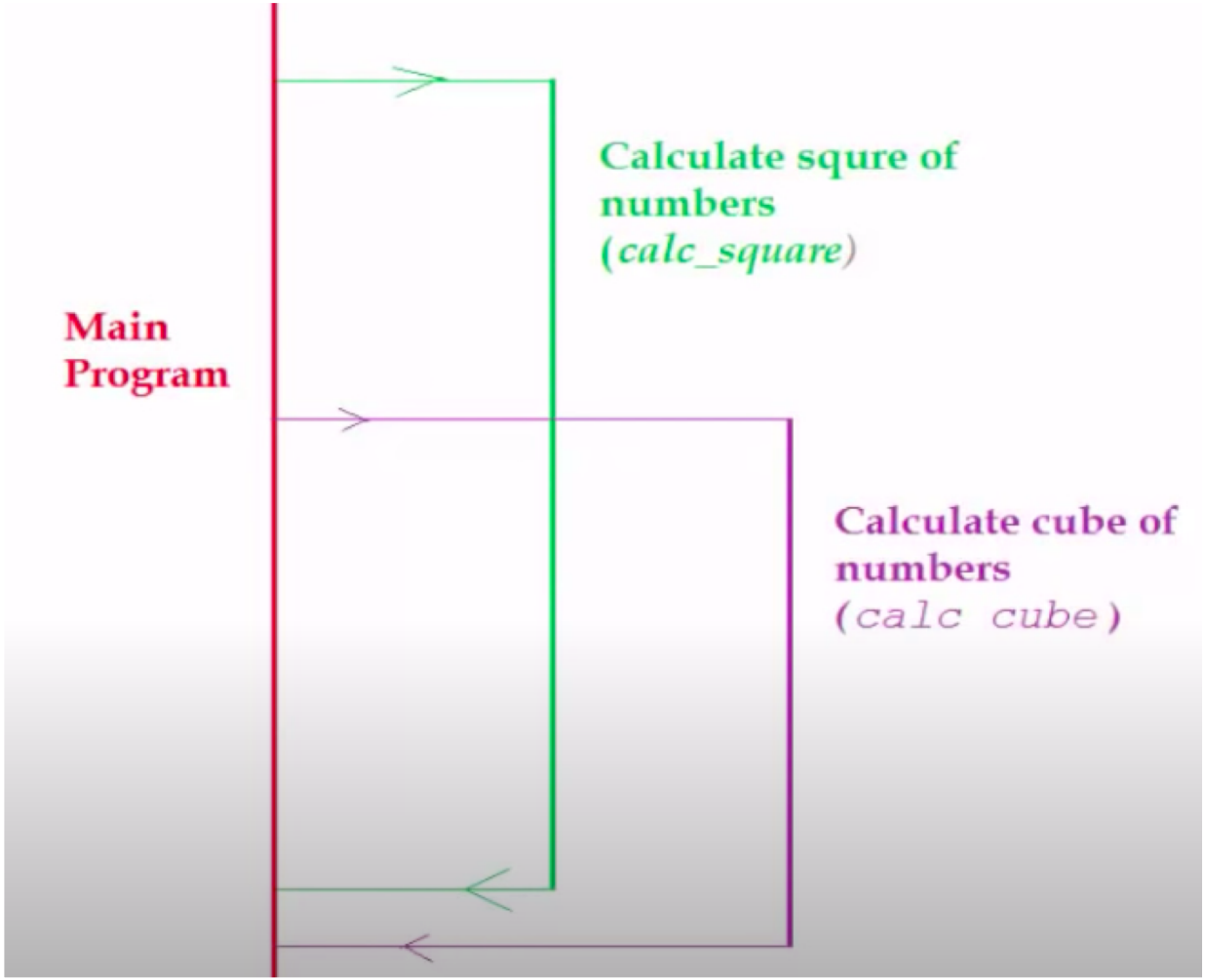
OUTPUT:

```
Ashishs-MacBook-Air:Assignments ashish$ python3 m.py
calculate square numbers
square:  4
square:  9
square:  64
square:  81
calculate cube numbers
cube:  8
cube:  27
cube:  512
cube:  729
Done in:  1.6261799335479736
I am done.
```

```python
import time
import threading
def cal_square(numbers):
    print("calculate square numbers")
    for n in numbers:
        time.sleep(0.2)
        print("square: ",n*n)

def cal_cube(numbers):
    print("calculate cube numbers")
    for n in numbers:
        time.sleep(0.2)
        print("cube: ",n*n*n)

arr=[2,3,8,9]
t=time.time()
t1=threading.Thread(target=cal_square, args=(arr,))
t2=threading.Thread(target=cal_cube, args=(arr,))

t1.start()
t2.start()

t1.join()
t2.join()
print("Done in: ", time.time()-t)
print("I am done.")
```

OUTPUT:

```
calculate square numbers
calculate cube numbers
square:  4
cube:  8
square:  9
cube:  27
square:  64
cube:  512
square:  81
cube:  729
Done in:  0.8174951076507568
I am done.
```


Calculate squre of
numbers
(calc_square)
Main
Program
Calculate cube of
numbers
(calc cube)

Advantages of Multithreading in Python

- Multithreading can significantly **improve the speed of computation** on multiprocessor or multi-core systems because each processor or core handles a separate thread concurrently.
- Multithreading allows a program to remain **responsive** while one thread waits for input, and another runs a GUI at the same time. This statement holds true for both multiprocessor or single processor systems.
- All the threads of a process have **access** to its global variables. If a global variable changes in one thread, it is visible to other threads as well. A thread can also have its own local variables.

Disadvantages of Multithreading in Python

- On a single processor system, multithreading won't hit the speed of computation. The **performance may downgrade** due to the overhead of managing threads.
- Synchronization is needed to prevent mutual exclusion while accessing shared resources. It directly leads to **more memory and CPU utilization.**
- Multithreading **increases the complexity** of the program, thus also making it **difficult to debug**.
- It raises the possibility of **potential deadlocks**.
- It may cause starvation when a thread doesn't get regular access to shared resources. The application would then fail to resume its work.

Thank you! Any Questions?