## **What is a Process in Python?**

In computing, a [**process**](https://www.geeksforgeeks.org/introduction-of-process-management/) is an instance of a computer program that is being executed. Any process has 3 basic components:

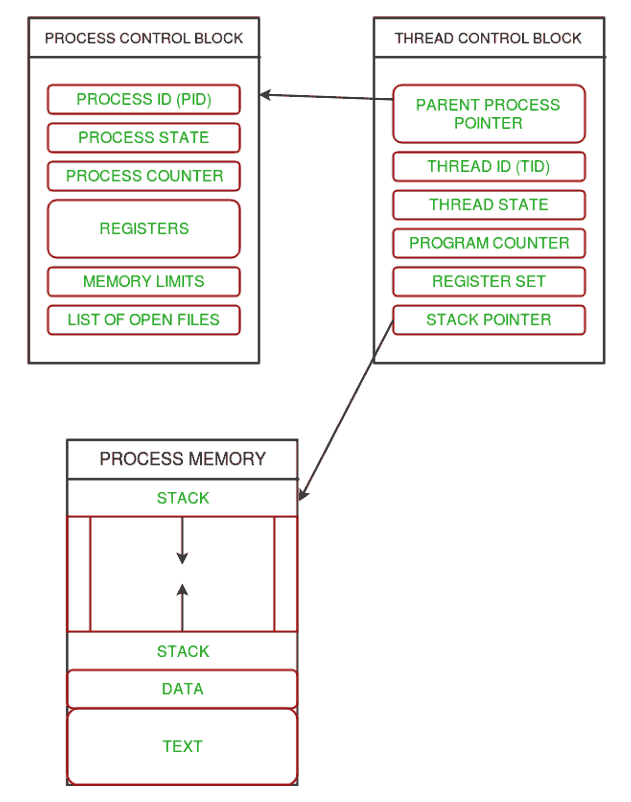
* An executable program.
* The associated data needed by the program (variables, workspace, buffers, etc.)
* The execution context of the program (State of the process)

## **An Intro to Threading in Python**

A **thread** is an entity within a process that can be scheduled for execution. Also, it is the smallest unit of processing that can be performed in an OS (Operating System). In simple words, a thread is a sequence of such instructions within a program that can be executed independently of other code. For simplicity, you can assume that a thread is simply a subset of a process! A thread contains all this information in a [**Thread Control Block**](https://www.geeksforgeeks.org/thread-control-block-in-operating-system/) **(TCB)**:

* **Thread Identifier:** Unique id (TID) is assigned to every new thread
* **Stack pointer:** Points to the thread’s stack in the process. The stack contains the local variables under the thread’s scope.
* **Program counter:** a register that stores the address of the instruction currently being executed by a thread.
* **Thread state:** can be running, ready, waiting, starting, or done.
* **Thread’s register set:** registers assigned to thread for computations.
* **Parent process Pointer:** A pointer to the Process control block (PCB) of the process that the thread lives on.

Consider the diagram below to understand the relationship between the process and its thread:

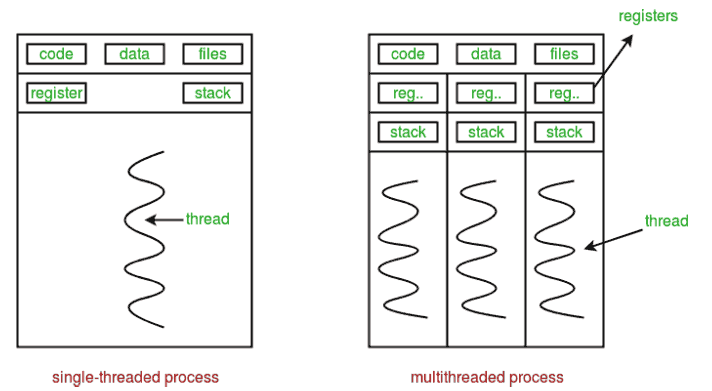


*Relationship between a Process and its Thread*

Multiple threads can exist within one process where:

* Each thread contains its own **register set** and **local variables (stored in the stack)**.
* All threads of a process share **global variables (stored in heap)** and the **program code**.

Consider the diagram below to understand how multiple threads exist in memory:

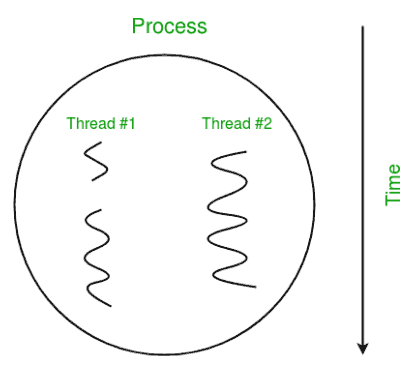


*Existence of multiple threads in memory*

## **An Intro to Threading in Python**

**Multithreading** is defined as the ability of a processor to execute multiple threads concurrently. In a simple, single-core CPU, it is achieved using frequent switching between threads. This is termed **context switching**. In context switching, the state of a thread is saved and the state of another thread is loaded whenever any interrupt (due to I/O or manually set) takes place. Context switching takes place so frequently that all the threads appear to be running parallely (this is termed **multitasking**).

Consider the diagram below in which a process contains two active threads:



*Multithreading*

### **Multithreading in Python**

In [Python](https://www.geeksforgeeks.org/python-programming-language/), the **threading** module provides a very simple and intuitive API for spawning multiple threads in a program. Let us try to understand multithreading code step-by-step.

**Step 1:** Import Module

First, import the threading module.

import threading

**Step 2:** Create a Thread

To create a new thread, we create an object of the **Thread** class. It takes the ‘target’ and ‘args’ as the parameters. The **target** is the function to be executed by the thread whereas the **args is** the arguments to be passed to the target function.

0 seconds of 15 secondsVolume 0%

This ad will end in 3

t1 = threading.Thread(target, args)

t2 = threading.Thread(target, args)

**Step 3:** Start a Thread

To start a thread, we use the **start()** method of the Thread class.

t1.start()

t2.start()

**Step 4:** End the thread Execution

Once the threads start, the current program (you can think of it like a main thread) also keeps on executing. In order to stop the execution of the current program until a thread is complete, we use the **join()** method.

t1.join()

t2.join()

As a result, the current program will first wait for the completion of **t1** and then **t2**. Once, they are finished, the remaining statements of the current program are executed.

**Example:**

Let us consider a simple example using a threading module.

**import** threading

**def** print\_cube(num):

# function to print cube of given num

print("Cube: {}" .format(num **\*** num **\*** num))

**def** print\_square(num):

# function to print square of given num

**print**("Square: {}" .format(num **\*** num))

**if** \_\_name\_\_ **==**"\_\_main\_\_":

# creating thread

t1 **=** threading.Thread(target**=**print\_square, args**=**(10,))

t2 **=** threading.Thread(target**=**print\_cube, args**=**(10,))

# starting thread 1

t1.start()

# starting thread 2

t2.start()

# wait until thread 1 is completely executed

t1.join()

# wait until thread 2 is completely executed

t2.join()

# both threads completely executed

print("Done!")

**Output:**

Square: 100

Cube: 1000

