# Python Threading Tutorial

Contents

[Python Threading Tutorial 1](#_Toc188626735)

[Video 1](#_Toc188626736)

[Threading 1](#_Toc188626737)

[Process 2](#_Toc188626738)

[Time Slicing 2](#_Toc188626739)

[Tutorial 1: Without Threading 3](#_Toc188626740)

[Tutorial 2: Threading 4](#_Toc188626741)

[Tutorial 3: Threading with Join 5](#_Toc188626742)

[Tutorial 4: Daemon Threads and Tkinter 7](#_Toc188626743)

[Assignment Submission 10](#_Toc188626744)

Time required: 60 minutes

# Video

This is a 3 hour class that goes into this subject very deeply. Watch this if you want to take a deep dive into threading.

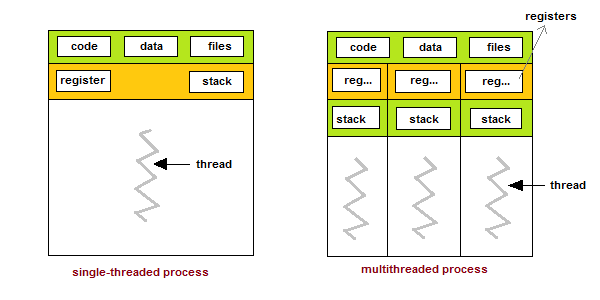
[Asynchronous Python Mastery: Threads, processes, and Async Development](https://youtu.be/fgitcCAjqQs?si=VUMtQwJEE1ixICrY)

# Threading

Threading is built into Python. Threading in Python is used to run multiple threads (tasks, function calls) at the same time. Python threads are used in cases where the execution of a task involves some waiting. One example would be interaction with a service hosted on another computer, such as a webserver. Threading allows python to execute other code while waiting.

# Process

A process has at least one thread and some resources set aside by the operating system. This could be a hard drive, processor cores, network, etc. The resources change as the process’s needs change.



# Time Slicing

Your computer can have hundreds of processes running at the same time. Time slicing or scheduling allows these threads to share process time. Time slicing is not free. The OS has to save the current status of the thread so that when it comes back to the core, it doesn’t start from the beginning. There is some overhead in switching from one thread to another.



# Tutorial 1: Without Threading

All program runs on at least one single thread. All code executes nose to tail, one after the other. Functions must run one after the other. These are called blocking calls. Nothing else can happen while the function is running.

* function1()
* function2()
* function3()

This program shows an example of functions running in a single thread.

Graphical user interface, text, application, email

Description automatically generated

Example run:



# Tutorial 2: Threading

Threading allows us to speed up programs by executing multiple tasks at the SAME time.

* Each task will run on its own thread
* Each thread can run simultaneously and share data with each other
* Every thread you start must do SOMETHING
* Threads will finish at different times. The OS task scheduler runs them when it has time.

We will define 3 different functions, one for each thread. Our threads will then target these functions. When we start the threads, the target functions will be run.

Modify the existing code as shown.

A picture containing text

Description automatically generated

Example run (Each run may be different):

A picture containing text

Description automatically generated

# Tutorial 3: Threading with Join

Modify the existing code to add a print statement. Notice that it executes right away on the program thread.

Text

Description automatically generated

Example run (Each run may be different):

Text

Description automatically generated

Uncomment the **.join()** methods. The .join() method releases the thread and allows the program to “join” the main thread.

This is an example run with join. All three threads must complete before the next command is run.

Example run (Each run may be different):

A picture containing application

Description automatically generated

# Tutorial 4: Daemon Threads and Tkinter

A daemon is a background service. If you are running a function or method in a separate thread that you want to keep going until the program stops, setting daemon to True will stop the thread when the program ends.

In a Tkinter application, the main event loop (mainloop()) is responsible for handling all GUI updates and user interactions. If a long-running task is executed within this main thread, it will block the event loop, causing the GUI to become unresponsive until the task completes.

Threading helps by allowing long-running tasks to run in separate threads, freeing up the main thread to continue handling GUI updates and user interactions. This ensures that the GUI remains responsive even while background tasks are being executed.

In the provided code, the background\_task method runs in a separate thread created by the start\_thread method. This allows the label to be updated with the current timestamp every second without blocking the main event loop, keeping the GUI responsive.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Example run:

A screenshot of a computer error

Description automatically generated

## Assignment Submission

1. Attach all program files.
2. Attach a screenshot of each successful program run.
3. Submit the assignment in Blackboard.