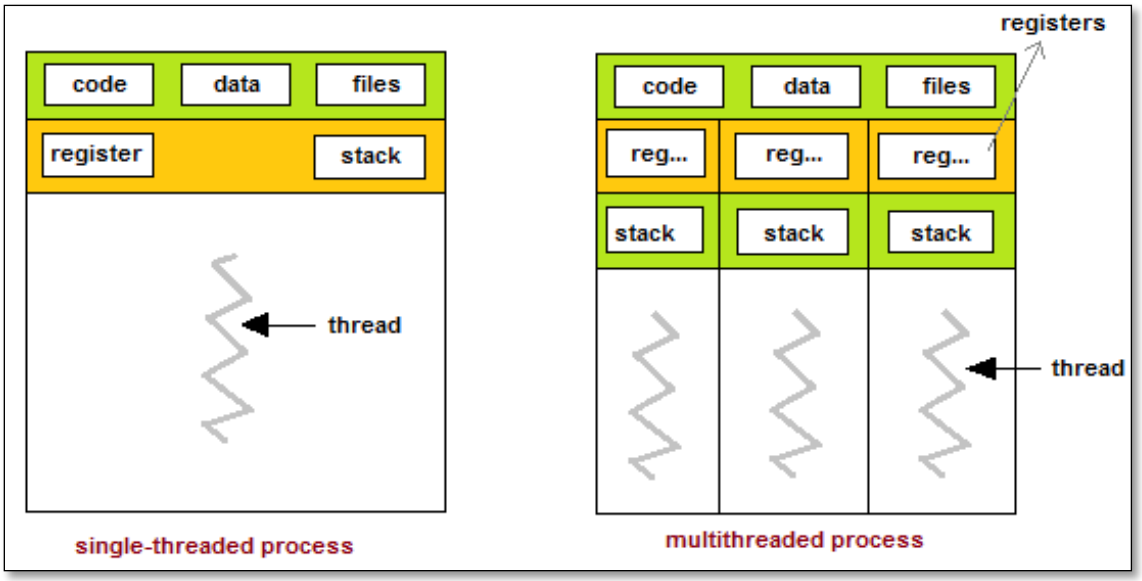
Python Threading Tutorial

Contents

Time required: 30 minutes

Threading

Multithreading 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.



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.

```python
"""
    Filename: threading_example.py
"""
import threading


# ---------------------- TEST FUNCTIONS -----------------------------------#
def function1():
    for i in range(5):
        print("ONE ")


def function2():
    for i in range(5):
        print("TWO ")


def function3():
    for i in range(5):
        print("THREE ")


def main():
    # ------------------- NORMAL LINEAR FUNCTIONS --------------------------#
    # If we call these functions, the first function call
    # MUST complete before the next, they are executed linearly
    function1()
    function2()
    function3()


# If a standalone program, call the main function
# Else, use as a module
if __name__ == "__main__":
    main()
```

Example run:

```
ONE
ONE
ONE
ONE
ONE
TWO
TWO
TWO
TWO
TWO
THREE
THREE
THREE
THREE
THREE
```

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.

```python
def main():
    # ------------------- NORMAL LINEAR FUNCTIONS -------------------------#
    # If we call these functions, the first function call
    # MUST complete before the next, they are executed linearly
    # function1()
    # function2()
    # function3()

    # ---------------------- THREADED FUNCTIONS ----------------------------#
    # We can execute these functions concurrently using threads!
    # We must have a target for a thread.
    t1 = threading.Thread(target=function1)
    t2 = threading.Thread(target=function2)
    t3 = threading.Thread(target=function3)

    t1.start()
    t2.start()
    t3.start()
```

Example run (Each run may be different):

```
ONE
ONE
THREE ONE TWO


THREE ONE TWO


THREE ONE TWO


THREE TWO

THREE TWO
```

Tutorial 3: Threading with Join

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

```python
def main():
    # ------------------- NORMAL LINEAR FUNCTIONS -------------------------#
    # If we call these functions, the first function call
    # MUST complete before the next, they are executed linearly
    # function1()
    # function2()
    # function3()

    # ---------------------- THREADED FUNCTIONS ----------------------------#
    # We can execute these functions concurrently using threads!
    # We must have a target for a thread.
    t1 = threading.Thread(target=function1)
    t2 = threading.Thread(target=function2)
    t3 = threading.Thread(target=function3)

    t1.start()
    t2.start()
    t3.start()

    # This pauses the main program until the thread is complete
    # t1.join()
    # t2.join()
    # t3.join()
    print("Threading rules!")
```

Example run (Each run may be different):

```
ONE TWO

ONE TWO

ONE TWO Threading rules!THREE



ONE TWO THREE


TWO ONE
THREE

THREE
THREE
```

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):

```
ONE
ONE
ONE THREE TWO


ONE THREE TWO


ONE THREE TWO


TWO THREE

TWO THREE

Threading rules!
```

Daemon Threads

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.

Let’s do a simple Tkinter program to illustrate. This program will update the label each second to the computer clock time.

```python
import tkinter as tk
# Import the Thread class from the threading module
# to handle concurrent operations
from threading import Thread
# Import the time module to use sleep() and time() functions
import time


class ThreadingApp:
    def __init__(self):
        # Create the main window of the application
        self.root = tk.Tk()
        # Set the title of the window
        self.root.title("Threading with Tkinter")
        # Call the method to set up the GUI elements
        self.setup_gui()

    def setup_gui(self):
        # Create label widget to display text
        self.lbl_display = tk.Label(self.root, text="Threading with Tkinter")
        # Place label in the window with 20 pixels padding on top and bottom
        self.lbl_display.pack(pady=20)

        # Create a button that will start the thread when clicked
        start_button = tk.Button(
            self.root,                    # Parent widget is the main window
            text="Start Thread",          # Text shown on the button
            command=self.start_thread     # Function to call when button is clicked
        )
        # Place button in the window with 20 pixels padding on top and bottom
        start_button.pack(pady=20)

    def background_task(self):
        """This method runs in a separate thread
           and performs a background task"""
        while True:  # Infinite loop
            # Update the label text with the current timestamp
            now = time.localtime()
            now = f"{now.tm_min:02d}:{now.tm_sec:02d}"
            self.lbl_display.config(
                text=f"Running background task {now}"
                )

            # Pause for 1 second before the next update
            time.sleep(1)
```

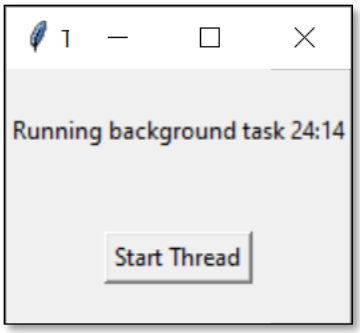
```python
    def start_thread(self):
        # Create a new thread object
        thread = Thread(
            target=self.background_task,  # Function to run in the thread
            daemon=True                   # Thread stops when program ends
        )
        # Start the thread's execution
        thread.start()

    def run(self):
        # Start the main event loop of the application
        self.root.mainloop()


def main():
    # Create an instance of our application
    app = ThreadingApp()
    # Start running the application
    app.run()


# Only run the app if this file is run directly (not imported)
if __name__ == "__main__":
    main()
```

Example run:



Assignment Submission

1. Attach all program files.
2. Attach a screenshot of each successful program run.

3. Submit the assignment in Blackboard.