# Threading

# Learning Objectives

By the end of this lesson, you will be able to:
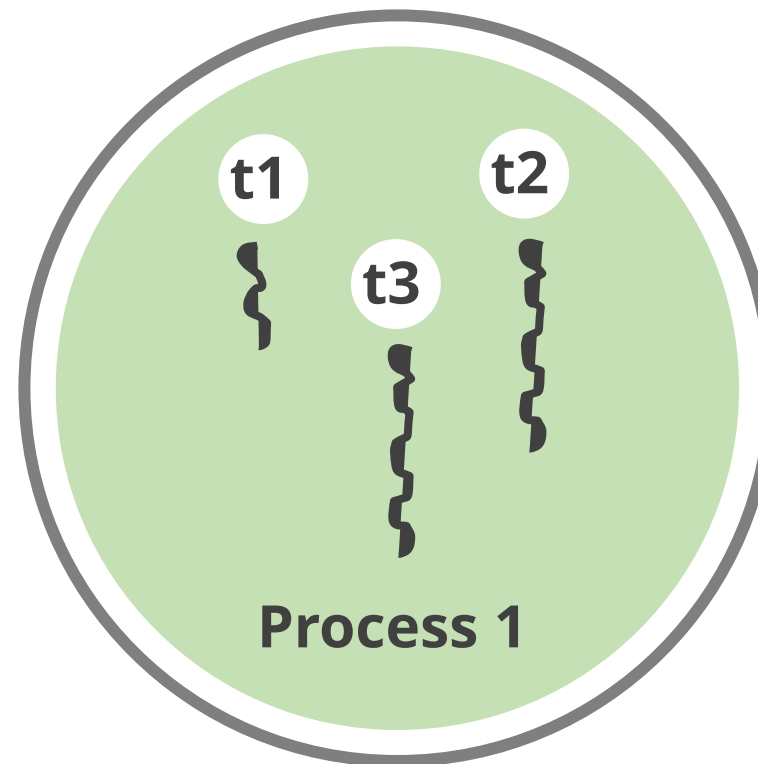
- Interpret threading

- Explain multi-threading

- Illustrate the steps to create a new thread

- Summarize the synchronization of threads

Introduction to Threading

# Threading

A thread is the smallest unit of execution with an independent set of instructions.



A process can have multiple threads in it as shown in the image.

# Threading

Threads are referred to as lightweight processes which have the below-mentioned features:

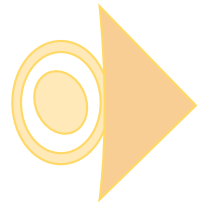They do not require as much memory overhead as processes do.

A thread runs within the same process and shares runtime resources like memory with the program.

A thread has a beginning, an execution path, and a result.

# Threading

Threads ensure an independent execution flow.

The thread's current state is stored in an instruction pointer, which also determines what will be executed next and in what sequence.
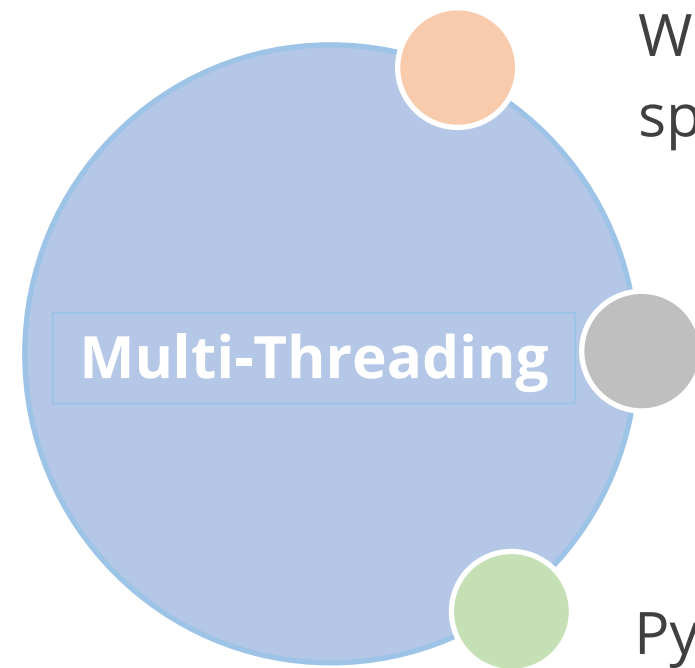
To provide the appearance of parallel programming, threading makes use of idle processes.

Introduction to Multi-Threading

# Multi-Threading

Multithreading is the ability of a process to run many threads concurrently.
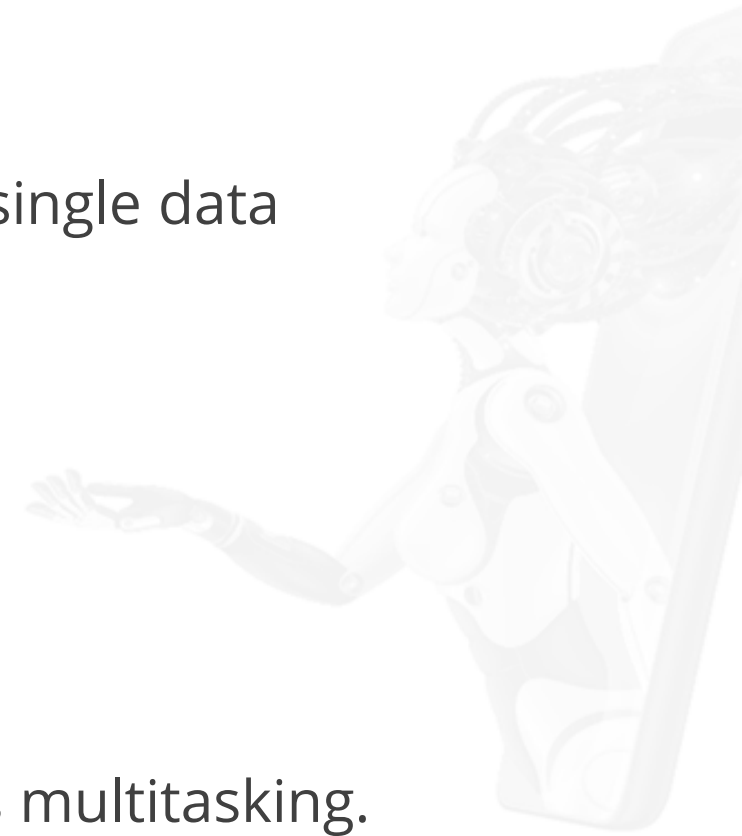
**Multi-Threading**

When multiple threads are operating concurrently, they share a single data space with the main thread.

This makes it easier for information to be shared.

Python programming makes use of multiple threads and enables multitasking.

# Multi-Threading: Advantages

The advantages of multi-threading are as follows:

Improves responsiveness significantly.

Allows a task to be broken into many subtasks and executed concurrently.

Improves application performance and rendering.

# Multi-Threading: Disadvantages

The disadvantages of multi-threading are as follows:

There would be no impact on the computation speed of a processor. It could reduce performance and result in overhead.

It is important to synchronize carefully to prevent mutual exclusion.

This raises the possibility of a deadlock.

# Threading Module

The *threading* module includes a class *Thread* which is used to implement threads.

| Class method | Method description |
| --- | --- |
| run() | Entry point function of a thread |
| start() | Triggers a thread when run method is called |
| join([time]) | Enables a program to wait for threads to terminate |
| isAlive() | Determines if a thread is active |
| getName() | Returns the name of a thread |
| setName() | Updates the name of a thread |

# Creating a New Thread

# Creating a New Thread

The steps to create a new thread are as follows.

| Step 1 | Step 2 | Step 3 |
|---|---|---|
| Define a Thread subclass. | Override the __init__ method to add additional arguments | Implement the required action by overriding the run function |

# Assisted Practice: Creating a new thread

**Objective:** In this demonstration, we will learn how to create a new thread.

**Tasks to perform:**

1. Define a thread subclass.

2. Override the __init__ method and add additional arguments.

3. Override the run() method.

# Synchronizing Threads

# Synchronizing Threads

Synchronization is done to make sure that only one thread at a time is using a resource when multiple threads are trying to access and modify a shared resource.

Here, two threads are trying to access the shared resources:



Thread &harr; Shared resources &harr; Thread

Improper synchronization might lead to a faulty program.

# Synchronizing Threads: Example

Consider two different threads T1 and T2 and *temporary.txt* is a shared resource

**Example**

- When T1 begins its execution, it saves certain values in a file called *temporary.txt* that will be used to compute a result when T1 has finished.

- Before T1 returns, T2 begins, and T2 modifies the values that T1 had stored in the file *temporary.txt*.

- T1 will hence produce an incorrect result.

T1 → Balance = 200

T2 → Balance = 100

T1 ← Balance = 100
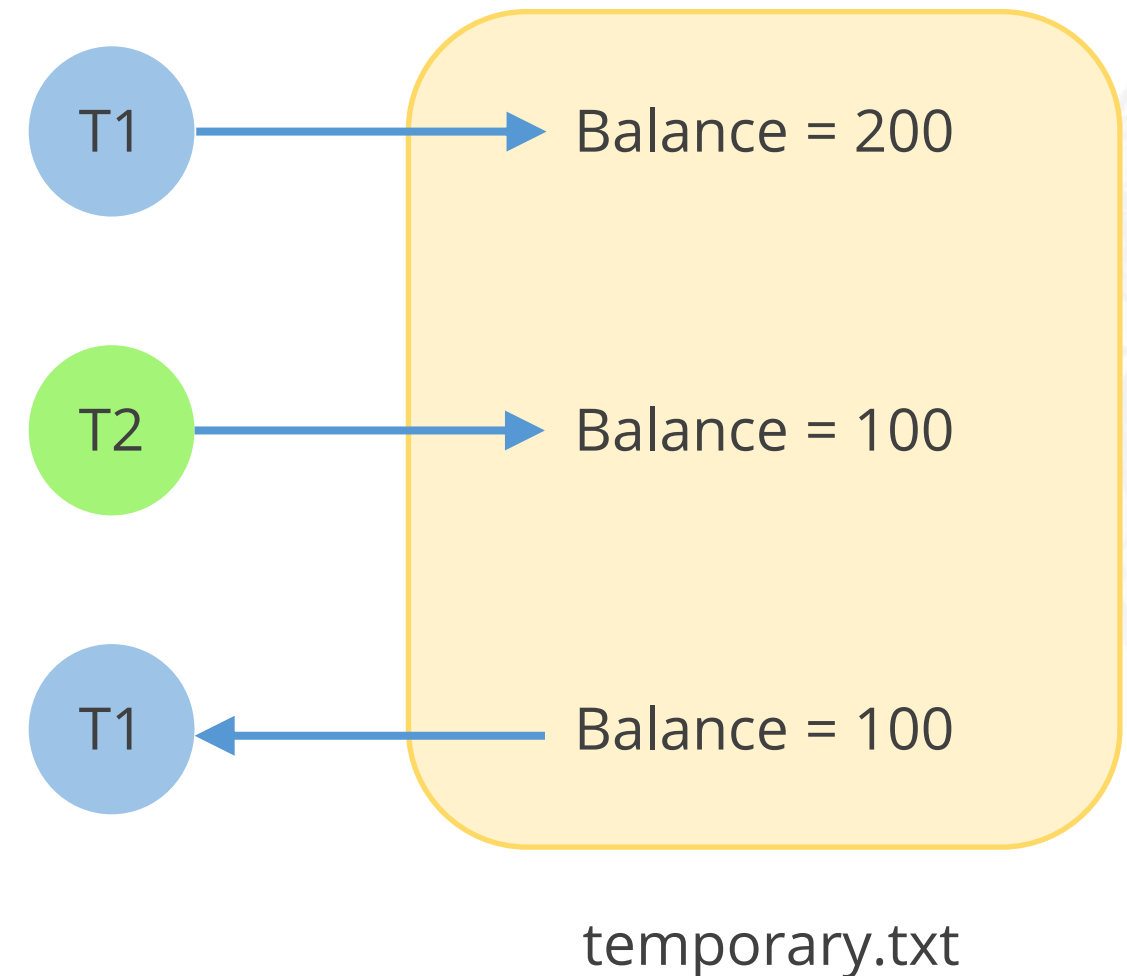
temporary.txt

# Synchronizing Threads: Lock

If synchronization is performed in the example, once thread T1 begins using the file *temporary.txt,* it will be locked and no other thread will be able to access or modify it until thread T1 returns.

T1 → Balance = 200

T2 ✕→ 🔒 Balance = 100

T1 ← Balance = 100

temporary.txt
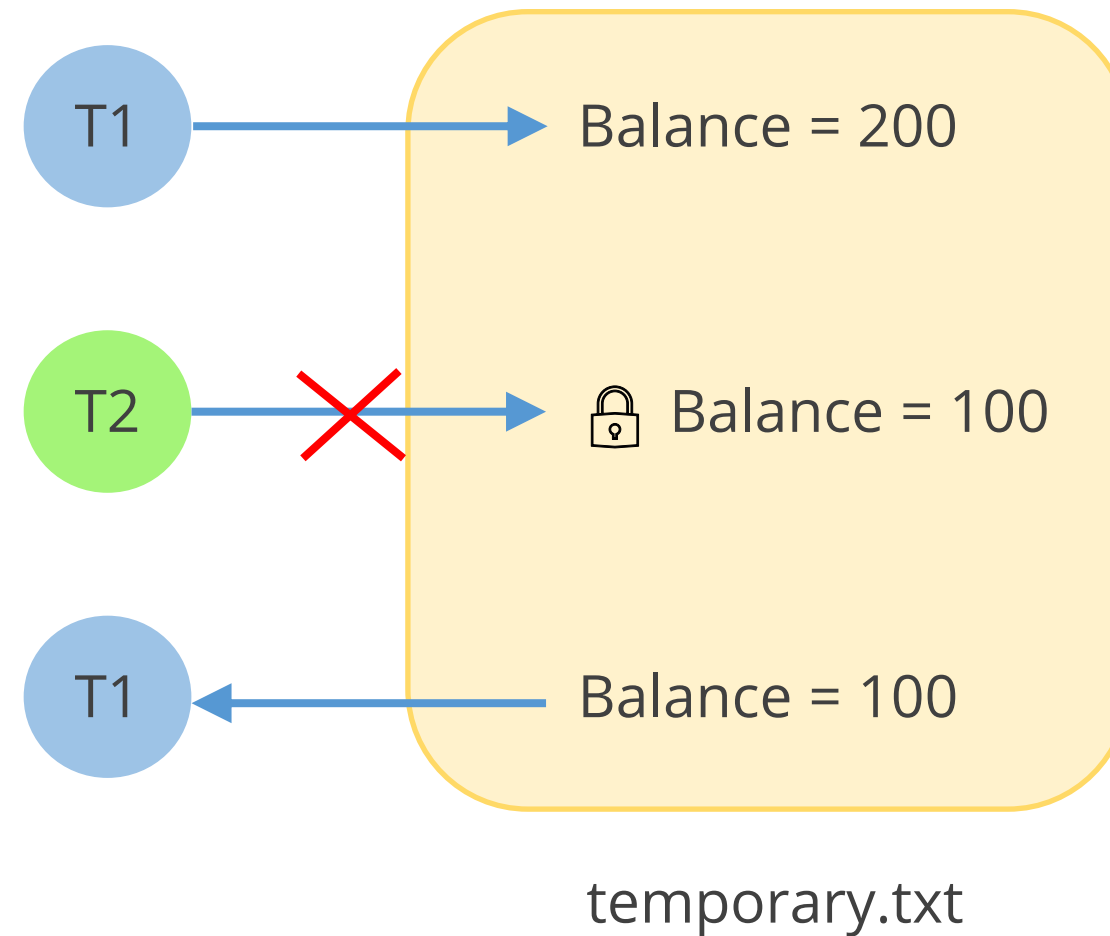
# Synchronization in Python

The *Threading* module has built-in locking functionality that allows you to synchronize threads.

Locking is essential to manage access to shared resources in order to prevent data corruption or loss.

The *lock* object can be used to generate a new lock.

The *lock* object *acquire(blocking)* method is used to enable threads to execute synchronously.

# Synchronization in Python: acquire()

The lock object has two methods: acquire() and release().

## acquire([blocking])

- The optional blocking parameter specifies whether the thread waits for the lock to be acquired.

- If blocking is set to 0, then a thread will return instantly with a value of 0 if the lock can't be obtained and a value of 1 if it can.

- If blocking is set to 1, the thread blocks and awaits the release of the lock.

# Synchronization in Python: release()

The lock object has two methods acquire() and release().

**release()**

- The release() function is used to release the lock when it is no longer necessary.

- When the lock is locked, unlock it and then return.

- Allow just one thread to proceed if multiple threads are waiting for the lock.

# Use of Synchronization in Python

To prevent concurrent access to an object, we must utilize a *lock* object.

Python's built-in data structures, such as lists and dictionaries, are thread-safe as a result of their use of atomic byte codes.

Other Python data structures or fundamental data types, such as integers and floats, do not have this security.

# Key Takeaways

- Threading is making use of idle processes to give the appearance of parallel programming.

- A thread is the smallest unit of execution with an independent set of instructions

- Python 3 provides a threading module to implement threads

- Synchronization is used to ensure the sharing of resources between multiple threads, which will not cause loss of data.

- Synchronization is achieved using Lock, acquire and release methods of the module.