PARALLEL PROGRAMMING

Assoc. Prof. Dr. Bora Canbula

Manisa Celal Bayar University
Department of Computer Engineering

https://github.com/canbula/ParallelProgramming/ 302kib9

Parallel Programming

Instructor

Assoc. Prof. Dr.
Bora CANBULA

Phone

0 (236) 201 21 08

Email

bora.canbula@cbu.edu.tr

Office Location

Dept. of CENG

Office C233

Office Hours

4 pm – 5 pm, Mondays

Course Overview

Parallel Programming (Teams Code: 302kib9)

We are going to learn the basics of asynchronous programming, creating multiple threads and processes in this course. Python is preferred as the programming language for the applications of this course.

Required Text

Python Concurrency with asyncio, Manning, *Matthew Fowler*

A Practical Approach to High-Performance Computing, Springer, *Sergei Kurgalin – Sergei Borzunov*

Python Parallel Programming Cookbook, Packt, *Giancarlo Zaccone*

Course Materials

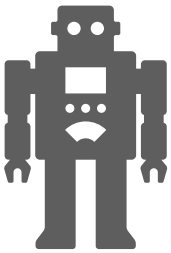
- Python 3.x (Anaconda is preferred)
- Jupyter Notebook from Anaconda
- Pycharm from JetBrains / Microsoft Visual Studio Code
- PC with a Linux distro or a Linux terminal in Windows 10/11.

Course Schedule

Week	Subject	Week	Subject
01	Data Structures in Python	08	Deadlock and Semaphore
02	Functions and Decorators in Python	09	Barriers and Conditions
03	Coroutines and Concurrency with asyncio	10	Creating Processes with multiprocessing
04	IO-bound Problems and Concurrency	11	Pipes and Queues
05	Creating Threads in Python with threading	12	CPU-bound Problems and Parallelism
06	Global Interpreter Lock and JIT Compiler	13	Creating Clusters
07	Protecting Resources with Lock	14	Load Balancing with Containers

Natural Languages vs. Programming Languages

A language is a tool for expressing and recording thoughts.



Computers have their own language called **machine** language. Machine languages are created by humans, no computer is currently capable of creating a new language. A complete set of known commands is called an instruction list (IL).

The difference is that human languages developed naturally. They are still evolving, new words are created every day as old words disappear. These languages are called **natural** languages.

Elements of a Language

- **Alphabet** is a set of symbols to build words of a certain language.
- **Lexis** is a set of words the language offers its users.
- **Syntax** is a set of rules used to determine if a certain string of words forms a valid sentence.
- **Semantics** is a set of rules determining if a certain phrase makes sense.

Machine Language vs. High-Level Language

The IL is the alphabet of a machine language. It's the computer's mother tongue.

High-level programming language enables humans to write their programs and computers to execute the programs. It is much more complex than those offered by ILs.

A program written in a high-level programming language is called a **source code**. Similarly, the file containing the source code is called the **source file**.

Compilation vs. Interpretation

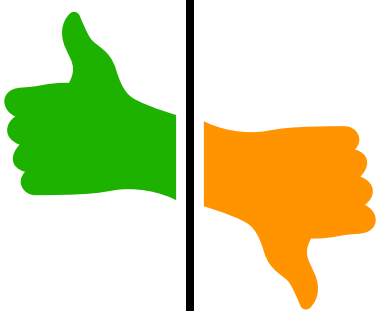
There are two different ways of transforming a program from a high-level programming language into machine language:

Compilation: The source code is translated once by getting a file containing the machine code.

Interpretation: The source code is interpreted every time it is intended to be executed.

Compilation

- The execution of the translated code is usually faster.
- Only the user has to have the compiler. The end user may use the code without it.
- The translated code is stored using machine language. Your code are likely to remain your secret.



- The compilation itself may be a very time-consuming process
- You have to have as many compilers as hardware platforms you want your code to be run on.

Interpretation

- You can run the code as soon as you complete it, there are no additional phases of translation.
- The code is stored using programming language, not machine language. You don't compile your code for each different architecture.

- Don't expect interpretation to ramp up your code to high speed
- Both you and the end user have the interpreter to run your code.

What is Python?

Python is a widely-used, interpreted, object-oriented, and high-level programming language with dynamic semantics, used for general-purpose programming.

Python was created by Guido van Rossum. The name of the Python programming language comes from an old BBC television comedy sketch series called Monty Python's Flying Circus.

Python Goals

- an **easy and intuitive** language just as powerful as those of the major competitors
- **open source**, so anyone can contribute to its development
- code that is as **understandable** as plain English
- **suitable for everyday tasks**, allowing for short development times

Why Python?

- easy to learn
- easy to teach
- easy to use
- easy to understand
- easy to obtain, install and deploy

Why not Python?

- low-level programming
- applications for mobile devices

Python Implementations

An implementation refers to a program or environment, which provides support for the execution of programs written in the Python language.

- **CPython** is the traditional implementation of Python and it's most often called just "Python".
- **Cython** is a solution which translate Python code into "C" to make it run much faster than pure Python.
- **Jython** is an implementation follows only Python 2, not Python 3, written in Java.
- **PyPy** represents a Python environment written in Python-like language named RPython (Restricted Python), which is actually a subset of Python.
- **MicroPython** is an implementation of Python 3 that is optimized to run on microcontrollers.

Start Coding with Python

- **Editor** will support you in writing the code. The Python 3 standard installation contains a very simple application named IDLE (Integrated Development and Learning Environment).
- **Console** is a terminal in which you can launch your code.
- **Debugger** is a tool, which launches your code step-by-step to allow you to inspect it.

first.py

```
print("Python is the best!")
```

https://form[illegible].com/r/3qrM5Gj66X

in-class quizzes

h[illegible]forms.office.com/r/GNNHg5B4c7

Function Name
A function can cause some effect
or evaluate a value, or both.

Where do functions come from?
- From Python itself
- From modules
- From your code

first.py

```
print("Python is the best!")
```

Argument
- Positional arguments
- Keyword arguments

`print``(*objects, sep=' ', end='\n', file=None, flush=False)`

Print *objects* to the text stream *file*, separated by *sep* and followed by *end*. *sep*, *end*, *file*, and *flush*, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like `str()` does and written to the stream, separated by *sep* and followed by *end*. Both *sep* and *end* must be strings; they can also be `None`, which means to use the default values. If no *objects* are given, `print()` will just write *end*.

The *file* argument must be an object with a `write(string)` method; if it is not present or `None`, `sys.stdout` will be used. Since printed arguments are converted to text strings, `print()` cannot be used with binary mode file objects. For these, use `file.write(...)` instead.

Output buffering is usually determined by *file*. However, if *flush* is true, the stream is forcibly flushed.

Literals

A literal is data whose values are determined by the literal itself. Literals are used to encode data and put them into code.

literals.py

```
print("7")
print(7)
print(7.0)
print(7j)
print(True)
print(0b10)
print(0o10)
print(0x10)
print(7.4e3)
```

- String
- Integer
- Float
- Complex
- Boolean
- Binary
- Octal
- Hexadecimal
- Scientific Notation

Basic Operators

An operator is a symbol of the programming language, which is able to operate on the values.

Multiplication

```
print(2 * 3)      Integer
print(2 * 3.0)    Float
print(2.0 * 3)    Float
print(2.0 * 3.0)  Float
```

Division

```
print(6 / 3)      Float
print(6 / 3.0)    Float
print(6.0 / 3)    Float
print(6.0 / 3.0)  Float
```

Exponentiation

```
print(2**3)       Integer
print(2**3.0)     Float
print(2.0**3)     Float
print(2.0**3.0)   Float
```

Floor Division

```
print(6 // 3)     Integer
print(6 // 3.0)   Float
print(6.0 // 3)   Float
print(6.0 // 3.0) Float
```

Modulo

```
print(6 % 3)      Integer
print(6 % 3.0)    Float
print(6.0 % 3)    Float
print(6.0 % 3.0)  Float
```

Addition

```
print(-8 + 4)     Integer
print(-4.0 + 8)   Float
```

Operator Priorities

An operator is a symbol of the programming language, which is able to operate on the values.

priorities.py

```
print(9 % 6 % 2)
print(2**2**3)
print(2 * 3 % 5)
print(-3 * 2)
print(-2 * 3)
print(-(2 * 3))
```

- + (unary)
- - (unary)
- ** (right-sided binding)
- *
- /
- //
- % (left-sided binding)
- + (binary)
- - (binary)

Variables

Variables are symbols for memory addresses.

Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

Built-in Functions

A	E	L	R
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
anext()			round()
any()	F	M	
ascii()	filter()	map()	S

hex(*x*)

Convert an integer number to a lowercase hexadecimal string prefixed with "0x". If *x* is not a Python `int` object, it has to define an `__index__()` method that returns an integer. Some examples:

```
>>> hex(255)
'0xff'
>>> hex(-42)
'-0x2a'
```

classmethod()	help()	ord()	type()
compile()	hex()		
complex()		P	V
	I	pow()	vars()
D	id()	print()	

id(*object*)

Return the "identity" of an object. This is an integer which is guaranteed to be unique and constant for this object during its lifetime. Two objects with non-overlapping lifetimes may have the same `id()` value.

https://docs.python.org/3/library/functions.html

Identifier Names

For variables, functions, classes etc. we use identifier names. We must obey some rules and we should follow some naming conventions.

- Names are case sensitive.
- Names can be a combination of letters, digits, and underscore.
- Names can only start with a letter or underscore, can not start with a digit.
- Keywords can not be used as a name.

keyword — Testing for Python keywords

Source code: Lib/keyword.py

This module allows a Python program to determine if a string is a keyword or soft keyword.

`keyword.`**`iskeyword`**`(s)`
Return `True` if *s* is a Python keyword.

`keyword.`**`kwlist`**
Sequence containing all the keywords defined for the interpreter. If any keywords are defined to only be active when particular `__future__` statements are in effect, these will be included as well.

`keyword.`**`issoftkeyword`**`(s)`
Return `True` if *s* is a Python soft keyword.

New in version 3.9.

`keyword.`**`softkwlist`**
Sequence containing all the soft keywords defined for the interpreter. If any soft keywords are defined to only be active when particular `__future__` statements are in effect, these will be included as well.

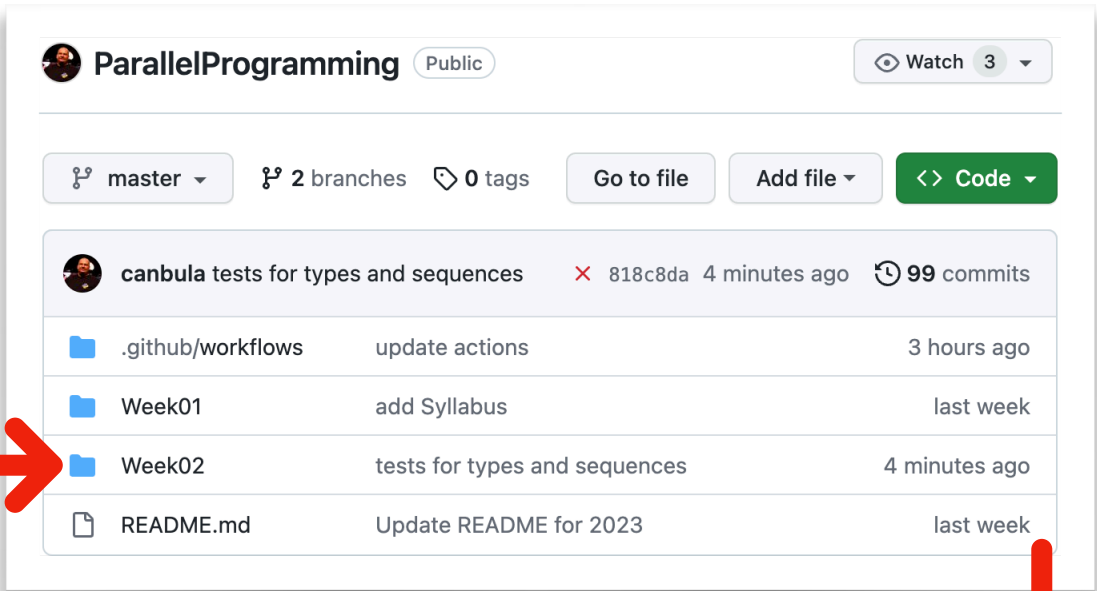
New in version 3.9.

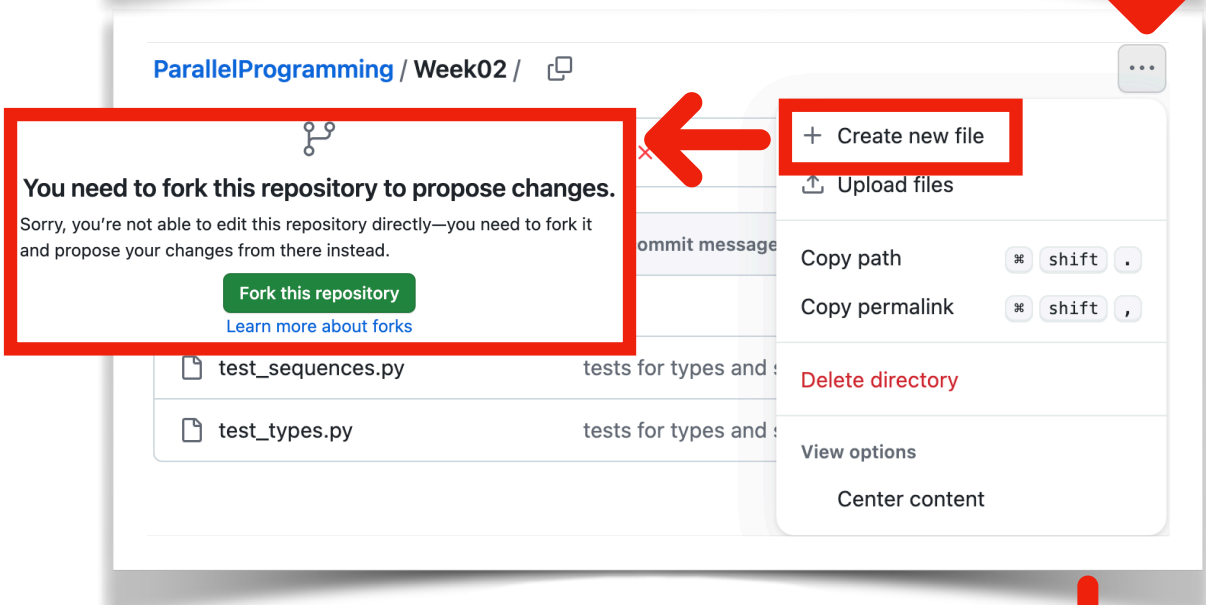
https://form[illegible].com/r/VBi2yJZiX5

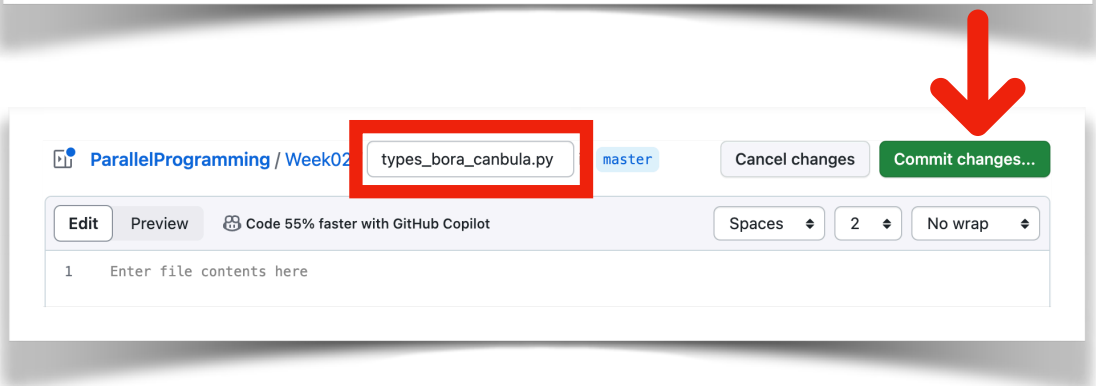
in-class quizzes

h[illegible]forms.office.com/r/RRCyEr8RE8

Your First Homework







- An integer with the name: my_int
- A float with the name: my_float
- A boolean with the name: my_bool
- A complex with the name: my_complex