MA4: Multiprocessing and integration with C++

This assignment primarily concerns parallel programming, higher order functions, and basic usage of `git`, Linux, and C++. The assignment consist of two major parts:

1. Parallel programming in Python
   - Use Monte Carlo to approximate π and the volume of a d-dimensional hypersphere
   - Comprehension, Lambda functions, `map`, `reduce`, `filter`, and `zip`
   - Plotting with `matplotlib`
   - Parallel programming in Python
2. C++ integration in Python
   - `git`
   - Linux
   - Write simple C++ code
   - Build a Python module written in C++
   - Compare execution time for Python vs C++vs Numba

Presentation MA4:
1. All parts (1.1, 1.2, 1.3, and 2.2) of the assignment orally, at one lesson, over Zoom or physically in a computer lab.
2. When the oral presentation has been approved, the files should be uploaded (use clear file names and include in the beginning of the files who approved the oral presentation at what date. Do not use .zip, .pdf, Jupyter notebooks, wordfiles or anything else but .cpp, .py, ...).

Important: You may collaborate with other students, but you must write and be able to explain your own code. You may not copy code neither from other students nor from the Internet except from the places explicitly pointed out in this lesson. Changing variable names and similar modifications does not count as writing your own code. Also, aiding somebopdy to cheat (for example, if somebody hands in your code as their own) is not permitted and will be reported.
Since the lesson tasks (MA's) are included as part of the examination, we are obliged to report failures to follow these rules.

1 Parallel programming in Python

In this part of the assignment you will first write a program that uses Monte Carlo to compute an approximation of π. Then you will modify this program so that the computation is done in parallel. Then, you will again modify the code so that you approximate the volume of a d-dimensional hypersphere.

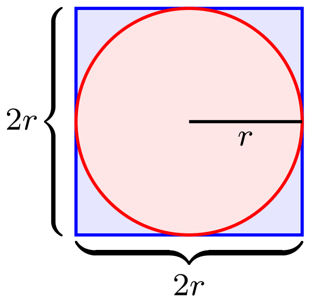
1.1 Monte Carlo approximation of π

Monte Carlo methods (https://en.wikipedia.org/wiki/Monte_Carlo_method) belong to a large and important class of methods and are used by many different algorithms to solve problems numerically; most importantly in optimization, integration, and probability theory.

One utilizes random tests to approximate the true underlying property. This is typically very useful and important for multidimensional problems where, for example, integration in all dimensions is impossible to do exactly with other conventional methods.

In this part of the assignment you will write a program that uses Monte Carlo to estimate the constant $\pi = 3.14159\ldots$

In Figure 1 is shown, in red, a circle with radius r and area $A_c = \pi r^2$. It is placed in a blue square, with sides $2r$, that has area $A_s = (2r)^2 = 4r^2$.



Figur 1: Circle with radius r inscribes in a square with sides $2r$.

If you divide the area A_c with A_s, you have,

$$\frac{A_c}{A_s} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4},$$

or

$$\pi = 4\frac{A_c}{A_s}.$$

Now assume that $r = 1$ and that the center of the circle is at the origin, (0,0). Now create n uniformly distributed random coordinates $(x, y) \in [-1, 1] \times [-1, 1]$ in the square, and sort these n_c points that lie inside the circle and n_s that lie outside the circle ($n = n_c + n_s$). One can then approximate π with

$$\pi \approx 4\frac{n_c}{n}.$$

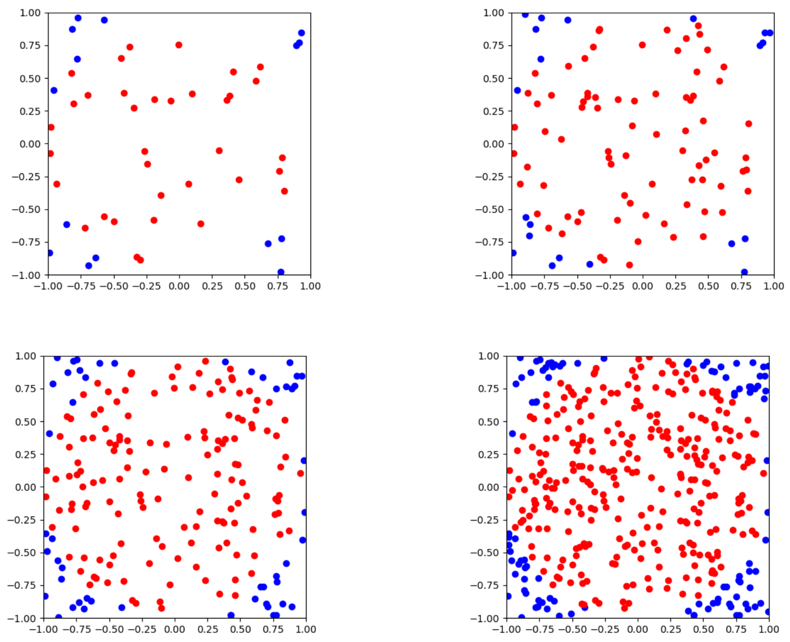
In Figure 2 is shown four examples, for $n = \{50, 100, 200, 400\}$. The approximations of π are then $\pi \approx \{2.8, 3.16, 2.96, 2.97\}$. Note that it can happen that one gets a better approximation of π for an n smaller than another, but as $n \to \infty$ the approximation will be arbitrarily close to π.

How do you decide if a point is inside the circle? In this simple case, when $r = 1$ and the center is in the origin, one can simply test if $x^2 + y^2 \leq 1$ (equivalent to $\sqrt{x^2 + y^2} \leq 1$). If one had a more complicated object, one can divide the domain into a mesh (for example a square grid or a triangulation) where each cell is either inside or outside the object.

Now write a program that has n, the number of random points that should be generated, as an argument and produces the following output:

1. Print the number of points n_c that are inside the circle.
2. Print the approximation of $\pi \approx 4n_c/n$.
3. Print the builtin constant π (`math.pi`) of Python.
4. Produce a `png` file that shows all points inside the circle as red dots and points outside the circle as blue dots (like Figure 2).

Tip: Use `random.uniform()` to create the n random points. Also, use `matplotlib.pyplot` to create the figures (Note, do not take screenshots write the figure to disk, you will need to know how to do this in part 2.2).


Figur 2: Approximation of π using Monte Carlo. Top: $n = 50, \pi \approx 2.8$ and $n = 100, \pi \approx 3.16$. Bottom: $n = 200, \pi \approx 2.96$ and $n = 400, \pi \approx 2.97$.

Oral presentation MA4 1.1:

1. Present and explain the code.
2. Show the figures for $n = \{1000, 10000, 100000\}$
3. Present the approximations of π for $n = \{1000, 10000, 100000\}$

1.2 Approximate the volume of a d-dimensional hypersphere

- In this part you will use higher order functions. See separate pdf in Studium (Module M4) for an introduction. You must use at least three of the concepts
  - Comprehension
  - Lambda function
  - `map()`
  - `functools.reduce()`
  - `filter()`
  - `zip()`

in this part of the assignment.

In this part of MA4 you should write a program that uses a Monte Carlo approximation of the volume $V_d(r)$ for a d-dimensional hypersphere radius r. A hypersphere is a generalization of the circle and sphere to higher dimensions than two and three. The program should have two arguments, n which is the number of random points to generate, and d which is the number of dimensions. One can assume $r = 1$ and the center of the hypersphere is the origin.

The test to see if a point is inside the hypersphere is in this case,

$$x_1^2 + x_2^2 + \ldots + x_d^2 \leq 1.$$

The furmula for the volume $V_d(r)$ (https://en.wikipedia.org/wiki/Volume_of_an_n-ball), to verify your code, is

$$V_d(r) = \frac{\pi^{d/2}}{\Gamma(d/2+1)} r^d, \qquad V_d(1) = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$$

where Γ is the Gamma function (https://en.wikipedia.org/wiki/Gamma_function). The Gamma function already exists in Python (in the module `math`) and you do not have to implement it yourselves. To solve this assignment you may use `numpy` if you want.

Oral presentation MA4 1.2:

1. Present and explain the code.
2. Show that you have used at least three higher order functions.
3. Present the approximation, and the exact value for $V_d(1)$, with $(n, d) = (100000, 2)$, $(n, d) = (100000, 11)$.

1.3 Parallel programming in Python

Read the separate pdf in Studium (Module M4) for an introduction to parallel programming in Python.

Modify your code from 1.2 so that it can run in parallel, by using `futures.ProcessPoolExecutor`. You can time the execution time with and without parallelization using `time.perf_counter`.

Oral presentation MA4 1.3:

1. Time the code from 1.2 with $(n, d) = (10000000, 11)$, and the parallel version of 1.3 with $(n, d) = (1000000, 11)$ and 10 processes (such that the total number of samplings n is the same for the two experiments)
2. Present the results. Which one was the fastest? How much faster and why?

2 Accelerating execution: C++ integration in Python and Numba

Pure Python code is often too slow to be practically usable. Then, one can write parts of the codes in other more efficient languages, e.g., C++, C, or Fortran. Many modules in Python are actually written in other languages due to this fact. We will now explore one (of many) ways to call C++ code, build a Python module and explore the performance difference compared to pure Python code. We will also briefly explore the *Numba* library. This is a way to speed up Python execution time to perform similarly to e.g. C++ without using another language.

As mentioned before Python is an interpreted language, where the code is interpreted and executed line by line. The language C++ on the other hand is a compiled language. That means that every time you modify your code it has to be compiled, that is translated to machine code, before the program can be run.

2.1 Preparation

This part of the assignment should be done on the Linux machines of the IT department. You could also do all the parts locally on your own machine, but in the oral presentation you have to show that you can do it on the Linux machines.

Linux is a free UNIX-like operating system (`https://en.wikipedia.org/wiki/Linux`) that is extensively used and you will learn how to use some basic features. This is useful for example if you ever need to setup servers (databases, web servers, etc.) or run programs in the cloud (with virtual machines), which is important in data engineering and data science.

2.1.1 IT department Linux machines

Choose one of the following Linux machines:

- arrhenius.it.uu.se
- atterbom.it.uu.se
- cronstedt.it.uu.se
- enequist.it.uu.se

Note that you can freely switch between the machines between different logins, all files are stored on a central filesystem and is available on all machines. You can also login multiple sessions at the same time.

The machines run the Ubuntu distribution of Linux and you do not have administrator (root) access, so you can't (and don't need to) use package manager to install any programs on the system.

To access a Linux machine you need to use a terminal program on your own machine (commands are run by writing them). Theoretically, you can run graphical programs also, but is out of the scope for this assignment (`https://en.wikipedia.org/wiki/X_Window_System`). You will use SSH (Secure Shell) to login to the Linux machines.

Some terminal programs (the recommended ones are highlighted in red), and you need to use one of them on your local computer.

Your OS	
Windows	If Windows 10 or later, the builtin Powershell
	WSL2 (search Microsoft Store)
	PuTTY (`https://www.putty.org/`)
	bitwise (`https://www.bitvise.com/`)
macOS	Terminal (preinstalled in /Applications/Utilities)
	iTerm2 (`https://iterm2.com/`)
Linux	xterm (preinstalled)
	(`https://www.tecmint.com/linux-terminal-emulators/`)

To login write (do not write the `$`, when it is in the beginning of a line it only indicates that you have a terminal started) in the terminal:

```
$ ssh abcde123@arrhenius.it.uu.se
```

Here `abcde123` is you UU account (same as you use to login to Studium) and you should use you password A (the same password you use to login to STUDIUM, not 'A') when asked for it (once you press return/enter). `arrhenius.it.uu.se` is the hostname you chose in the list of linux machines. If you use PuTTY or bitwise you will have to enter this information in a GUI (username: `abcde123` and hostname: `arrhenius.it.uu.se`).

If there are any problems, contact sven-erik.ekstrom@it.uu.se.

Here is a collection of some useful linux commands:

`ls`	List files in current directory (`ls -la` to show more information)
`pwd`	Show where you are (print working directory)
`cd abc`	Go into a directory named `abc`
`cd ..`	Go up one step in the file system (e.g.,`/` is the root directory, `/home` contains home directories and `/home/abcde123` is the home directory of user `abcde123`)
`cd`	Go to your home directory (where your personal files are, and this is where you are when you first log in)
`mkdir abc`	Create a directory called `abc`
`rm -fr abc`	Remove a file or directory called `abc`
`nano hej.txt`	Edit a file `hej.txt` with the editor `nano` (there are many different editors). Leave nano with `ctrl-x`
`python3 test.py`	Run the Python code in the file `test.py` with Python 3.x. Changing to `python test.py` would invoke Python 2.x on these linux machines (configurable so might be different elsewhere).

A tutorial with an introduction to linux (for further information ask teachers)

https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners

Python is installed on the Linux machines, however `matplotlib` is not installed. You need to install `matplotlib` once, and it is easiest to do with:

```
$ pip3.9 install matplotlib
```

We use `pip3.9` since `pip` is configured to handle Python 2.x packages on the Linux machines at the department (and `pip3.9` is for Python 3.9 there). Note: you only do this once on the Linux machine.

To exectute a Python file `hello.py`, you run it as

```
$ python3.9 hello.py
```

since you want to specify what version of Python to use (best use 3.9 and not an older version).

If you want to try this assignment locally on your home computer you will need a C++ compiler (for example, on the Linux machines you will use `g++` which is part of `gcc`). On Windows, if you installed WSL2 `gcc` is installed by default (you could also use https://visualstudio.microsoft.com/). On macOS you install XCode through the App Store. Again, note that for the oral presentation you need to have done the assignment on the Linux machines.

2.1.2 git

Get an account at some git provider, for example www.github.com. See separate instruction in Studium (Module M4).

2.1.3 C++

In this assignment you will use ctypes to communicate with C++ code from Python. This is a built-in module in Python that is written to communicate with code written in the language c, but with a simple modification it also works for C++. Note that there are a lot of ways to do this in Python, some other notable modules are Cython and SWIG (https://realpython.com/python-bindings-overview/).

Files of the assignment

Assume that you have created a git repository at your git provider and it is called prog 2. Now clone it to your local computer.

Download MA4_files.zip from Studium (in assignment MA4, where you downloaded this pdf). Unpack the zip in your local repository (we here assume it is prog2/MA4/MA4_files after you have unpacked the zip). Now add, commit, and push the files to the repository of your provider. Either do this with some specific git client, VSCode, or in the terminal,

```
$ cd prog2/MA4
$ git add MA4_files
$ git commit -m "added files for MA4"
$ git push
```

The files in MA4_files.zip are:

- person.cpp C++ code for the module person with a class Person.
- Makefile The file that gives instructions to the make command how the C++ code in person.cpp should be compiled.
- person.py Python code for the module person.
- MA4_2.py A test Python program that uses the module person.

We now briefly describe the contents of these files, and their function in the test program MA4_2.py.

person.cpp

```cpp
#include <cstdlib>
// Person class

class Person{
	public:
		Person(int);
		int get();
		void set(int);
	private:
		int age;
	};

Person::Person(int n){
	age = n;
	}

```

```cpp
int Person::get(){
        return age;
        }

void Person::set(int n){
        age = n;
        }
```

- `#include <cstdlib>` on line 1 means that definitions in `cstdlib` are included.
- Comments in C++ are made with `//` (the rest of the line is a comment) like on line 2 or with `/* comment */` which can span over several lines.
- `Person` (line 4–11) is a class with three public methods (constructor, `get`, and `set` on lines 6–8) and a private value (the persons `age` on line 10).
- Data types must be declared in C++ which is not necessary in Python. In this code we only have `int`, which stands for integer. Other examples are `double` (floating point number) and `char` (character).
  - `Person(int);` means that the constructor takes a variable of type `int` as an argument.
  - `int get();` means that the method `get` does not have any argument and returns an `int`
  - `void set(int);` means that the method `set` requires an argument of type `int` and returns nothing, `void`.
- Note that expressions are finalized with a `;`. C++ does not take into account white space/indentation like Python, and instead uses `{}` to organize code blocks.

In the end of `person.cpp` there is also,

person.cpp (cont.)

```cpp
extern "C"{
        Person* Person_new(int n) {return new Person(n);}
        int Person_get(Person* person) {return person->get();}
        void Person_set(Person* person, int n) {person->set(n);}
        void Person_delete(Person* person){
                if (person){
                        delete person;
                        person = nullptr;
                        }
                }
        }
```

This code, which is in `c`, is a bridge code so that the Python module `ctypes` can communicate with the C++ code. `Person_delete` is a destructor, which removes an object.

As mentioned before, C++ code has to be compiled every time it has been modified (otherwise the compiled binary will be of an older version of the code).

To compile the code in `person.cpp` (to be done remotely on the linux machine when you have uploaded the files there, which is described later in this pdf), the following two commands are to be executed:

```
$ g++ -std=c++11 -c -fPIC person.cpp -o person.o
$ g++ -std=c++11 -shared -o libperson.so  person.o
```

These command create a so-called “shared object” `libperson.so` which Python can import. The compiler on the Linux machines is `gcc`, and `g++` the C++ compiler.

Since every time you have edited the file `person.cpp`, you have to run the two commands above you can instead use the command `make` which gets its commands from a file called `Makefile`.

Makefile

```
# Makefile for MA4
all:
	g++ -std=c++11 -c -fPIC person.cpp -o person.o
	g++ -std=c++11 -shared -o libperson.so  person.o
clean:
	rm -fr *.o *.so __pycache__
```

Comments in makefiles start with a #-sign, and then the rest of the line is a comment, like on line 1. It is important to have a <tab> on the rows that start with `g++` and `rm`. To run the two `g++` commands, all one have to do in the terminal is to type `make` (or `make all`). This is because `all:` is the first block of commands. The second block of commands is named `clean:` which runs the command on the last line (`rm -fr *.o *.so __pycache__`, which deleted files `*.o`, `*.so` and the directory `__pycache__`); it is called by running `make clean` in the terminal.

You can add other blocks of consecutive commands in the make file, just have a unique identifier like `all:` and `clean:` above. A more advanced build system of the same type is `CMake` that in a similar way can compile on all major platforms (like Windows, macOS, Linux, ...)

To read more about make files you can for example read `https://opensource.com/article/18/8/what-how-makefile` and `https://www.gnu.org/software/make/manual/make.html`, and about `Cmake` on `https://cmake.org/`.

The actual Python module (that you will import with `import person` in other Python programs) is

person.py

```python

""" Python interface to the C++ Person class """
import ctypes
lib = ctypes.cdll.LoadLibrary('./libperson.so')

class Person(object):
	def __init__(self, age):
		lib.Person_new.argtypes = [ctypes.c_int]
		lib.Person_new.restype = ctypes.c_void_p
		lib.Person_get.argtypes = [ctypes.c_void_p]
		lib.Person_get.restype = ctypes.c_int
		lib.Person_set.argtypes = [ctypes.c_void_p,ctypes.c_int]
		lib.Person_delete.argtypes = [ctypes.c_void_p]
		self.obj = lib.Person_new(age)

	def get(self):
		return lib.Person_get(self.obj)

	def set(self, age):
```

```
20                  lib.Person_set(self.obj, age)
21
22          def __del__(self):
23                  return lib.Person_delete(self.obj)
```

- On line 2 the module `ctypes` is imported, which handles the bridging to the `C` and C++ code.
- On line 3 the shared object `libperson.so`, from the compiled C++ code, is imported.
- From line 5 we have the class defintion of an `Person` in Python. In the constructor argument types (`argypes`) and return types (`restype`) are defined for the different functions in the `extern "C"` part of `person.cpp`. The types are here `ctypes.c_int` (`int`) and `ctypes.c_void_p` (`void`).
- On lines 15 and 18 the Python methods for `get()` and `set(val)` are defined.

The last file in `MA4_files.zip` is a test code in Python that uses the module `person`.

MA4_2.py

```
1   #!/usr/bin/env python3.9
2
3   from person import Person
4
5   def main():
6           f = Person(5)
7           print(f.get())
8           f.set(7)
9           print(f.get())
10
11  if __name__ == '__main__':
12          main()
```

- The first line is a so-called *shebang* (https://en.wikipedia.org/wiki/Shebang_(Unix)) that defines what program should be run if we execute the script directly by `$ ./MA4_2.py` (may be necessary to make the script executable by `$ chmod 755 MA4_2.py` first). One can also execute the script with `$ python3.9 MA4_2.py`.
- On line 3 `Person` is imported from the module `person`.
- On lines 6–7 an object `f` of type `Person` is created with a value of 5, which is printed.
- On lines 8–9 the value of `f` is changed to 7, which is printed.

Todo on the Linux machines

Once you are logged into a Linux machine write

```
git clone https://github.com/your_username/prog2
```

where the url is modified to fit your `git` server provider, username, and repository name.

Note: If you use Github to handle your repository, it is required to use so-called *Personal access token* to login when you want to clone the repository (it wil not work with the password you use to login to www.github.com). Here is a description how to create an access token, and in point 8 you just check "repo".
Then you use the created access token instead of a password when git asks for it. To not be required to use it everytime you access the repository, yoiu can store it on the Linux machine, this can be achieved by doing the following once you have cloned the repository:

```
sveek137@atterbom:~/prog2$ git config credential.helper store
sveek137@atterbom:~/prog2$ git pull
Username for 'https://github.com': sverek
Password for 'https://sverek@github.com':
Already up to date.
```

Now write (moduify paths if they are different for you)

```
$ cd prog2/MA4/MA4_files
$ ls
```

and you should now see the files described in the previous section.

To test your code run:

```
$ make
$ python3 MA4_2.py
5
7
```

You can also test the following

```
$ ./MA4_2.py
$ ls -la
$ chmod 755 MA4_2.py
$ ls -la
$ ./MA4_2.py
```

The first time you run `$ ./MA4_2.py` you get an error message. Note the difference between the first and second time you execute `$ ls -la` for the file `MA4_2.py`. The command `chmod` changes the rights of the file, and 755 makes the file executable. When you have done this once you can execute the code by just typing `$ ./MA4_2.py` (until you change the rights of the file to something else).

2.2 Numba

While Python in general is slow, it is also easy to read. If a piece of code cannot be understood, it cannot be maintained easily and will quickly become obsolete. Especially for scientific uses, being able to peer-review code and easily try different things is essential, meaning readability is of particular value. To get the best of both worlds and try to speed up Python without needing to write C++-code (sometimes called "the two language problem"), special libraries exist. One such library is *Numba*, which you will use to speed up your Python code and compare it to native Python and C++.

Numba is a so-called “Just-In-Time” (JIT) compiler. Essentially, it compiles a piece of code at run time (as opposed to C++, which is compiled before execution), meaning the first time code is run, it is converted to fast machine code. This is particularly useful on loops, where many executions of the same piece of code are conducted. The one-time compilation will greatly accelerate performance of the following executions.

In order to be able to accelerate code beyond Python-speeds, Numba needs to make a few assumptions about the code, e.g. regarding data types (like in C++, as opposed to generic Python that uses duck typing). We consider an example:

```
 1  # Example of a dynamically written function
 2  def maxOfList():
 3      result = 'Nothing yet'
 4      lst = [1, 8, 3, 9, 14]
 5      for x in lst:
 6          if result=='Nothing yet':
 7              result = x
 8          elif x > result:
 9              result = x
10      return result
```

This code would run perfectly fine in Python, but cannot run with Numba. Can you guess why? It has to do with the fact that the variable `result` is of different types in different parts of th code; first a string (`str`) on line 3, then an integer (`int`) on lines 7 and 9. Hence, the comparison on line 8, `>`, is not well defined (might be comparing an integer to a string). Numba analyses the whole function when it is compiling it to make it faster, and we have here an ambiguity that is not acceptable for Numba. Numba is not as strict as C++, which does not handle any conversions at all, but much more strict than standard Python. For the interested, you have here some examples of cases where you need to put extra care for Numba to work and give faster code https://numba.pydata.org/numba-doc/latest/user/troubleshoot.html. However, your task using Numba is so simple you will not have to worry about this.

2.2.1 Function decorators and setup

To use Numba, you first need to install the Numba-library. This is easily done using `pip3.9` on the Linux machines, similarly to installing `matplotlib` or similar libraries. To install, simply open a command window and type:

```
$ pip3.9 install numba
```

To apply Numba to a Python-function, you will use a *function decorator*. You can read more about these online, but essentially these are functions that modify another function in some way (in our case: speed it up through compilation).

The syntax is adding `@decoratorName` above a function definition. So to JIT-compile a function using Numba, you simply add

```
@njit
```

above your function definition. Note that you must have imported Numba to your program:

```
from numba import njit
```

Example:

```python
@njit
def someFunction(x):
    ...
```

Numba will then JIT-compile the decorated function the first time it is called.

2.3 Compute Fibonacci numbers in Python, with Numba, and in C++

To see how compiled languages (such as C++) can be more efficient than interpreted ones (such as Python), you will now calculate Fibonacci numbers using both. You will also use Numba to speed up your Python code and see how it compares.

You will now modify the code you have on the linux machine, so that you can compute Fibonacci numbers for some given n. This should be done in pure Python, Python with Numba, and in C++ (by modifying the `person` module and computing the Fibonacci number of the persons `age`). Use a recursive version for computing the Fibonacci number,

```python
def fib_py(n):
        if n <= 1:
                return n
        else:
                return(fib_py(n-1) + fib_py(n-2))
```

In the assignment you should measure how long time the three functions take to complete, for some different Fibonacci numbers. You will plot these timings in a figure, that you should save to disk. Note that you can not look at the images you create on the Linux machine, and you have to save to disk.

1. You can either use the existing files and add functionality, or copy the files to some other place (by using for example the commands `cp`, `mkdir`, `mv`). If you have new names for files, then remember to modify accordingly in the `Makefile`.

2. Use either some editor in the terminal, e.g., `nano` (recommended), `vim`, or `emacs`. One can also use VSCode to edit files remotely (`https://code.visualstudio.com/docs/remote/ssh`).

3. Create an "equivalent" C++ method as the Python method `fib_py` above, so that the following works in a Python script

   ```
   f=Person(n)
   f.fib()
   ```

   Hint: One way to implement this in C++ is to create one public and one private method in the `Person` class. Do not forget to modify the `C` bridging code, and also the Python module file `person.py`.

4. Make timings for `fib_py(n)`, `fib_numba(n)` and `f=Person(n);f.fib()` (the C++ code) for $n = 30, \ldots, 45$. Use for example `time.perf_counter`. Save the results to a figure (.png) (x-axis is n and y-axis is seconds). Use, for example, `matplotlib.pyplot`. Remember that you can not show the figure that you create on the Linux machine, but you have to save the figure to disk.

5. Also time `fib_py(n)` and `fib_numba(n)` for $n = 20, ..., 30$ and save the results as a figure.
6. Compute the Fibonacci number for $n = 47$ using the C++-code and Numba (too slow with pure Python).
7. Remember to use `git` as a backup of your code, and send the finished code to you main `git` repository

```
$ git add newfile.txt  # add a file to \code{git}'s indexing
$ git commit -m "added newfile.txt" # commit changes with a message
$ git push # send commited changes to the main repository
$ git pull # get changes from the repository
$ git stash # save what you have done and return to the lastet unaged statestate
```

Oral presentation MA4 2.2:

1. Present and explain your code.
2. Login to a linux machine and show that you have used `git`. (e.g., `ls -la` in the directory with assignment files).
3. Show the pictures with your timings. Comment on the results.
4. What is the result for Fibonacci with $n = 47$? Why?