

# Introduction to Programming Lecture 7

- Instructor : Andrew Yarmola [andrew.yarmola@uni.lu](mailto:andrew.yarmola@uni.lu)
- Course Schedule : Wednesday 14h00 - 15h30 Campus Kirchberg B21
- Course Website : [sites.google.com/site/andrewyarmola/itp-uni-lux](https://sites.google.com/site/andrewyarmola/itp-uni-lux)
- Office Hours : Thursday 16h00 - 17h00 Campus Kirchberg G103 and by appointment.

## Remarks on homework and questions

### Writing files

Writing files is a slightly more dangerous operation as you can accidentally write over a file or completely erase it. To open a file for writing, we use the `open(path_to_file, mode)` command again, but here the `mode` parameter will be different.

Here are all the mode parameter options.

- 'r' open for reading (default)
- 'w' open for writing, **ERASING** the file first
- 'x' open for exclusive creation, failing if the file already exists
- 'a' open for writing, appending to the end of the file if it exists
- 'b' binary mode
- 't' text mode (default)
- '+' open a disk file for updating (reading and writing)

If you are creating a **new** file that shouldn't already exist, use the 'x' option to write it.

In [2]:

```
import random

random_ints = random.sample(range(1000), 5)
# if I change the 'x' to a 'w' this will overwrite the
# file everytime I run this code
file_handle = open('random_numbers.txt', 'x')
for i in random_ints :
    file_handle.write(str(i)+'\n')
file_handle.close()

f = open('random_numbers.txt', 'r')
print(f.read())
f.close()
```

117  
489  
361  
312  
154

In [3]:

```
random_ints = random.sample(range(1000), 5)
file_handle = open('random_numbers.txt', 'a')
for i in random_ints :
    file_handle.write(str(i)+'\n')
file_handle.close()

f = open('random_numbers.txt', 'r')
print(f.read())
f.close()
```

117  
489  
361  
312  
154  
466  
559  
762  
520  
723

The final three modes 'b', 't', '+' are modifier modes. For example, to read a binary file, you would use the mode 'rb', to write a 'wb', etc.

The '+' mode modifies allows for simultaneous reading and writing with, for example, the 'r+' mode. However, I do **not recommend** using this mode unless you really really have to. It can also behave differently on windows and unix systems.

In [5]:

```
# open the file as binary
file_handle = open('../pdf/Lecture 1.pdf', 'rb')
# read the first 20 bytes
print(file_handle.read(20))
file_handle.close()
```

```
b'%PDF-1.3\n%\xc4\xe5\xf2\xe5\xeb\xa7\xf3\xa0\xd0\xc4'
```

## Saving python data

It may happen that you want to save some python list or dictionary for later use. This can be accomplished by using the `repr` and `eval` commands in conjunction.

- `repr(object)` (tries to) returns a string representation of an object
- `eval(string)` (tries to) evaluate string as a string representation of an object and returns the object

In [6]:

```
a = {float(2**65) : 'some bad example here'.split()}
x = {1 : a, 2 : 4.4 }
print(type(repr(x)))
print(repr(x))
```

```
<class 'str'>
{1: {3.6893488147419103e+19: ['some', 'bad', 'example', 'here']},
2: 4.4}
```

In [7]:

```
z = {1: {3.6893488147419103e+19: ['some', 'bad', 'example', 'here']}, 2: 4.4}
y = eval(repr(x))
print(y == x)
print(z == x)
```

```
True
```

```
True
```

You can now save python `repr` data to a file and load it later.

In [8]:

```
fp = open('test_file.txt', 'w')
fp.write(repr(x))
fp.close()
```

## Writing and reading json data

A better and more portable way to store list, dictionary, or other text data is to use a file format called **json**. It is used widely in web development, server configurations, and data messaging. This file format is very similar to a python dictionary.

In [9]:

```
import json

# show the json representaion of an object
print(json.dumps(x))
print(json.dumps(a))

{"1": {"3.6893488147419103e+19": ["some", "bad", "example", "here"]}, "2": 4.4}
{"3.6893488147419103e+19": ["some", "bad", "example", "here"]}
```

Notice the quotation marks around the data. We can also write straight to a file

In [10]:

```
file_handle = open('json_data.json', 'x')
# writes the json repr of x into the file_handle
json.dump(x, file_handle)
file_handle.close()
```

In [11]:

```
cat json_data.json
```

```
{"1": {"3.6893488147419103e+19": ["some", "bad", "example", "here"]}, "2": 4.4}
```

We can then `json.load(file_handle)` to get our data back!

In [12]:

```
fp = open('json_data.json', 'r')
data = json.load(fp)
fp.close()

print(data)

{'1': {'3.6893488147419103e+19': ['some', 'bad', 'example', 'here']}, '2': 4.4}
```

**Remark :** json always loads all your **keys** as strings!

Note : we will talk about how to check if a file exists, move, rename, and delete files and directories a bit later.

## with statement of opening files

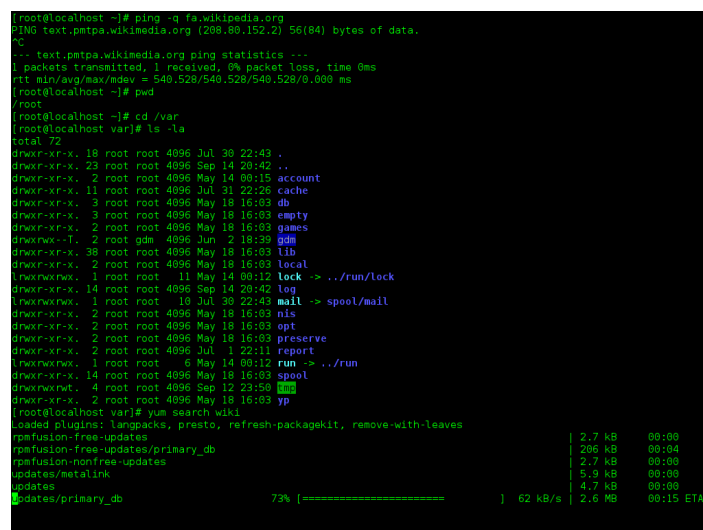
Python has a nice advanced statement called the `with` statement. I won't go into the details, but on a basic level, the `with` statement allows you to define a startup and cleanup action for a block of code. For opening files this is very useful as you will always want to make sure the file is closed after you have opened it. For example, we can rewrite the above code

In [13]:

```
with open('json_data.json', 'w') as file_handle :
    json.dump(x, file_handle)
# the with statement will automatically close the file!
```

## Command line

Having some familiarity with the command line is a very useful (and perhaps essential tool) for programmers. The command line is a text based prompt where you can execute various commands. Different operating systems tend to have a different list of commands available. You can actually run different versions of the command line, called **shells**.



```
[root@localhost ~]# ping -q fa.wikipedia.org
PING text.patpa.wikimedia.org (208.60.152.2) 56(84) bytes of data.
^C
--- text.patpa.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 548.528/548.528/548.528/0.000 ms
[root@localhost ~]# pwd
/root
[root@localhost ~]# cd /var
[root@localhost var]# ls -la
total 72
drwxr-xr-x. 18 root root 4096 Jul 30 22:43 .
drwxr-xr-x. 23 root root 4096 Sep 14 20:42 ..
drwxr-xr-x. 2 root root 4096 May 14 08:15 account
drwxr-xr-x. 11 root root 4096 Jul 31 22:26 cache
drwxr-xr-x. 3 root root 4096 May 18 16:03 db
drwxr-xr-x. 3 root root 4096 May 18 16:03 empty
drwxr-xr-x. 2 root root 4096 May 18 16:03 games
drwxr-xr-x. 2 root gdm 4096 Jun 2 18:39 gdm
drwxr-xr-x. 38 root root 4096 May 18 16:03 lib
drwxr-xr-x. 2 root root 4096 May 18 16:03 local
lrwxrwxrwx. 1 root root 11 May 14 08:12 lock -> ../run/lock
drwxr-xr-x. 14 root root 4096 Sep 14 20:42 log
lrwxrwxrwx. 1 root root 10 Jul 30 22:43 mail -> spool/mail
drwxr-xr-x. 2 root root 4096 May 18 16:03 nis
drwxr-xr-x. 2 root root 4096 May 18 16:03 opt
drwxr-xr-x. 2 root root 4096 May 18 16:03 preserve
drwxr-xr-x. 2 root root 4096 Jul 1 22:11 report
lrwxrwxrwx. 1 root root 6 May 14 08:12 run -> ../run
drwxr-xr-x. 14 root root 4096 May 18 16:03 spool
drwxrwxrwt. 4 root root 4096 Sep 12 23:58 tmp
drwxr-xr-x. 2 root root 4096 May 18 16:03 yp
[root@localhost var]# yum search wiki
Loaded plugins: langpacks, presto, refresh-packagekit, remove-with-leaves
rpmfusion-free-updates                                | 2.7 kB    00:00
rpmfusion-free-updates/primary_db                     | 286 kB    00:04
rpmfusion-nonfree-updates                             | 2.7 kB    00:00
updates/metalink                                      | 5.9 kB    00:00
updates                                                | 4.7 kB    00:00
updates/primary_db                                    73% [=====] | 62 kB/s | 2.6 MB    00:15 ETA
```

Here is the basic idea :

In [14]:

```
# we can see our present working directory
%pwd
```

Out[14]:

```
'/Users/yarmola/Teaching/python-course/lectures/week-7'
```

In [15]:

```
# we can list files in our current directory
%ls
```

```
Lecture 7.ipynb      json_data.json      random_numbers.txt  test_f
ile.txt
```

In [16]:

```
# we can remove files
%rm random_numbers.txt
```

In [17]:

```
# check that we did really remove it
%ls
```

```
Lecture 7.ipynb  json_data.json  test_file.txt
```

In [18]:

```
# we can dump the contents of files to the terminal as text
%cat json_data.json
```

```
{"1": {"3.6893488147419103e+19": ["some", "bad", "example", "here"]}, "2": 4.4}
```

In [19]:

```
# change directory (the .. means go up one level)
%cd ..
```

```
/Users/yarmola/Teaching/python-course/lectures
```

In [20]:

```
# list a directory with more information
%ls -l
```

```
total 0
drwxr-xr-x  3 yarmola  staff   96 May  4 20:09 pdf/
drwxr-xr-x  6 yarmola  staff  192 May  4 19:48 week-1/
drwxr-xr-x  4 yarmola  staff  128 May  4 20:10 week-10/
drwxr-xr-x  4 yarmola  staff  128 May  4 20:09 week-11/
drwxr-xr-x  4 yarmola  staff  128 May  4 20:09 week-12/
drwxr-xr-x  4 yarmola  staff  128 May  4 20:09 week-13/
drwxr-xr-x  6 yarmola  staff  192 May  4 19:54 week-2/
drwxr-xr-x  6 yarmola  staff  192 May  4 19:57 week-3/
drwxr-xr-x  4 yarmola  staff  128 May  4 20:01 week-4/
drwxr-xr-x  4 yarmola  staff  128 May  4 20:02 week-5/
drwxr-xr-x 11 yarmola  staff  352 May  4 20:07 week-6/
drwxr-xr-x  7 yarmola  staff  224 May  4 20:10 week-7/
drwxr-xr-x  4 yarmola  staff  128 May  4 20:09 week-8/
drwxr-xr-x  4 yarmola  staff  128 May  4 20:09 week-9/
```

In [22]:

```
# chande directory relative to our current place
%cd pdf
```

```
/Users/yarmola/Teaching/python-course/lectures/pdf
```

**Note** : The paths pdf/ and .. are **relative paths to my present working directory** while "/Users/yarmola/Teaching/python-course/homework" is a **global path**.

Above, I was using the % character to tell the ipython interpreter to call command line operations (and not python functions). This only works for a select set of "magic" commands allowed by ipython. When working with the command line, you will work in a **separate** program called a **terminal emulator**.

On macOS you can run the terminal emulator /Application/Utilities/Terminal.app and on Windows, things are a little more complicated. There is the built-in cmd prompt, which has a very different and much more limited set of commands. On Windows 10, you can enable the bash shell (see <https://msdn.microsoft.com/en-us/commandline/wsl/about?f=255&MSPPError=-2147217396>). On earlier versions of windows, you should install Babun (<https://babun.github.io>).

You can now try to run some of the above commands (without the % sign in front). As you can see, the general structure of a command is

- command -<option1> -<option2> ... <arg1> <arg2> ...

Here, the < > means that you replace what is between **and including** them with your desired argument (or option). Here is a basic list of commands.

- ls <path\_to\_directory\_or\_omit>
  - list the contents of a directory (or present working directory)
- cd <directory\_path>
  - change the present working directory
- pwd
  - display the path of the current directory
- mkdir <new\_directory\_name\_or\_path>
  - make a new directory at the path specified (gives error if there is already a file or directory at that path)
- rmdir <new\_directory\_name\_or\_path>
  - remove the directory at the path specified
- cp -i <file\_1> <new\_path\_2>
  - copy a file, without the -i this would **overwrite new\_path\_2 such a file already exists**.
- mv -i <file\_or\_dir\_1> <new\_path\_2>
  - copy a file or directory, without the -i this would **overwrite new\_path\_2 if such a file already exists**.
- rm <path\_to\_file>
  - remove (or delete) a file.
- man <command\_name>
  - show the manual for the command (to quit out of a man page, press q)
- cat <path\_to\_file>

- output the contents of a file
- `diff <file_1> <file_2>`
  - print the difference between two text files

**Remark :** there are a few keyword paths to be aware of :

- `~`
  - shorthand for your home directory, for example `cd ~/Teaching` is the same on my machine as `cd /Users/yarmola/Teaching` because my home directory is `/Users/yarmola`
- `..`
  - shorthand for the directory up one level up
- `.`
  - short hand for the current directory

**Note :** you can use TAB completion to help you along!

**Warning :** The command line can be rather dangerous if you are not careful!

## Git and GitHub

One of the goals for this course is to introduce you to the most essential tools of software development. Perhaps the most important such tool is **version control software**. As you work on projects and files, you will often make changes and additions. It is a good idea to not just always have a backup of you work, but to also see what changes others have made. Additionally, if both of you work on at the same time, you might want an automated method for combining your work at the end. For this purpose we will start learning how to use **git**.

Git is a simple, fast, and very very popular version control software. I will mostly spend time working with git **from the command line**, however, you can try learning to use the GitHub Desktop software if you want.

**Note :** Git is the version control software that runs on your computer. GitHub is the cloud service that stores a copy of you code on their servers.

Git manages different versions of your project. It is based around **commits** and **branches**. **Commits** are a frozen version of your code (like a save, but for a whole directory). Git takes over the directory where you are writing your code and manages it in such a way that at any given time the files in the directory represent a **commit** (or an edit to a commit). Commits are organized into **branches**, which are feature paths through your code. Branches usually correspond to feature additions that you work on separately from the stable version of your project. The whole system is called a **repository**.

**GitHub** is an online (i.e. **remote**) repository storage solution for **git**. In particular, you will have a **local** and a **remote** copy of your repository that you will have to keep in sync.

If you are running macOS or Linux, you most likely have git already installed and you can just open a command line and try using it. For Windows users, please download git from <https://git-scm.com/downloads>.



The key words for git are :

- `git clone <repo>` : you can make a copy someones existing repository to use, modify, or submit suggested changes.
- `git add <file>` : you can add/mark a file as updated so that it will be added next commit
- `git status` : ask git to tell you which files have been modified and which have been staged for the next commit.
- `git diff <file or omit>` : show the changes since the last commit.
- `git commit -m "Description"` : commit (i.e. save) all files for which you have called add.
- `git branch --all` : show all existing branches
- `git checkout <branch_name>` : switch the contents of your directory to the latest commit in branch <branch\_name>.
- `git checkout -b <new_branch_name>` : make a new branch copying the current.
- `git merge <branch_name_or_commit_id>` : you can merge branches together using git's automated tools. Most of the time it will work out for you all the differences and how to combine your code with someone else's code.
- `git fetch` : download all new data from GitHub
- `git pull` : download **and** merge all new data form GitHub into your local repository
- `git push origin <branch_name>` : push your chnages to branch <branch\_name> to GitHub.

Let's do a little demo and take a look at the GitHub website.

GitHub learning videos :

- <https://www.youtube.com/playlist?list=PLg7s6cbtAD165JTRsXh8ofwRw0PqUnkVH>
- <https://www.youtube.com/watch?v=0fKg7e37bQE>

A very barebones guide : <http://rogerdudler.github.io/git-guide/>

# Project 1

Over the next three weeks. Project 1 will be due Wednesday Nov 23. You will work in teams of 2 on one of the following projects. You can also suggest your own projects pending my approval. Each team will have to do a different project. The goal of each project is for you to write a piece of software that solves a specific programming challenge. Here are the general topics for each project.

- Topic 1 : (Homology) Implement the notions of a simplex, simplicial complex, and write a method to compute homology with  $\mathbb{Z}/p\mathbb{Z}$  coefficients
- Topic 2 : (Chebyshev Polynomials) Implement the notions of a fraction (i.e. rational numbers), polynomials (with rational coefficients), Chebyshev polynomials, the Chebyshev inner product for polynomials, and a method for decomposing a polynomials in the Chebyshev basis.
- Topic 3 : (Finite Fields) Implement the notions of  $\mathbb{Z}/p\mathbb{Z}$  for  $p$  prime, polynomials over  $\mathbb{Z}/p\mathbb{Z}$ , irreducibility over  $\mathbb{Z}/p\mathbb{Z}$ , and field extension over  $\mathbb{Z}/p\mathbb{Z}$ .
- Topic 4 : (Elliptic Curve Cryptography) Implement the notions of an Elliptic curve over  $\mathbb{Z}/p\mathbb{Z}$  of  $p$  prime, points and arithmetic on an Elliptic curve, the Diffie-Hellman key exchange, a simple encryption and decryption algorithm.
- Topic 5 : (Graphs) Implement the notion of an undirected graph. Create methods to check graph connectedness, find a minimal spanning tree, find a shortest path between two vertices, compute the closure of a graph, and find the graphs strongly connected components.
- Topic 6 : (Itinerary) Given an itinerary file of flights find the number of days spent in each country. You will have to look up countries from airport codes. You will generate a CSV spread sheet of the resulting data.
- Topic 7 : (Markov Chain Monte Carlo) Given a probability distribution, implement the Metropolis-Hastings random walk on a square lattice and apply it to computing expectations of a given function. This is called Markov Chain Monte Carlo.

Please let me know your team and choice of project by Wednesday Nov 9. You will submit your work on GitHub, so please make a GitHub team for this project. Additionally, **each team with have to write a series of tests to check that their code does what it should.**