Chapter 1

Getting familiar with Python

Goal of the lecture. After this lecture, all students should:

- have Python3 and Jupyter notebook installed on their laptop;
- be familiar with using the python interpreter, using python scripts, and using Jupyter notebooks, and be able to write simple programs in each of these environments;
- know about variables in python and about simple data types, and how to use them appropriately to perform operations;
- be aware of the numerical limitations of number types (limits and precision);
- start checking error messages and using the `help` function, as well as the `tab` auto-completion tool in Jupyter notebook; start using relevant online documentation to answer their questions;
- know how install and import third-party python modules; have installed `numpy`; import `math` or `numpy` to use common mathematical functions;
- be familiar with good programming habits.

1.1 Introduction

1.1.1 Setting up Python

Python3. For this course, we will be working with python3, as the entire python community as now moved on to using it (the latest version is python3.9). On linux and macos, you can check your version of python by typing `python --version`, or `python3 --version` in your terminal (or in the Command Prompt application in Windows).

If you do not have Python3 already installed, please find here some online documentations on how to install it: for Linux, for Macos [1] (you will need to install Homebrew first), and for Windows. Please don't hesitate to let me know if you would like to suggest a more recent or better explained online reference for installing Python3.

Some of you may find it easier to install everything at the same time (including many useful packages and other resources) using anaconda (see below for more information about package managers). Please feel free to do so. Note that in this course we will not discuss anaconda.

Please let us know if you haven't managed to install Python3 on your laptop.

Pip. is python's default package manager. Pip installs packages, such as numpy, jupyter, pandas, or tensorflow, along with their dependencies. It usually comes built-in to Python, so if you have Python, you likely have pip already installed. Simple commands:

– To check which version of pip you have: `$ pip --version`

[1] For Linux and Mac users, I would advise against setting the PATH environment variable for python3 – we can discuss this in class.

– To upgrade pip to the latest version: `$ pip install --upgrade pip`

– To install a new package, for instance numpy: `$ pip install numpy`

If you have several version of python installed on your laptop, you have to be careful and check for which version of python is your `pip` command working (use `$ pip --version` and compare the result with `$ python --version`: do you get the same path to python?). To guarantee that you know for which version of python you are installing which packages, the proper way is to **replace `pip` by `python3 -m pip`** (i.e., the python command you are using followed by `-m pip`) at the beginning of all the commands above.

Package managers and Virtual environments. There exist other package managers that sit on top of pip and will allow you to create virtual environments to manage different versions of your packages depending on the project you are working on. A common package for data science work is Anaconda (its package manager is `conda`). Another interesting package manager is Pyenv. See here for a recent review of the most common package managers.

1.1.2 Working with python

Python Interpreter. The most basic way to execute Python code is line by line within the Python interpreter. The Python interpreter can be started by typing `python3` (or `python`) in your Terminal.

Start python3 in your terminal and execute small codes, such as basic calculations, assigning values to variables, or printing text or variables using the function `print()`. Press `Enter` at the end of each command to validate it. For example:

```
>>> 1 + 1
2
>>> x = 5
>>> x * 3
15
>>> a = 3 * x
>>> print(a)
15
```

Used this way, python can be handy when you need a quick access to a calculator. To quit the interpreter type `quit()`.

Python scripts. Running Python snippets line by line is useful in some cases, but for more complicated programs it is more convenient to save code in a file, and execute it all at once. By convention, Python scripts are saved in files with a `.py` extension. For example, create a file `test.py` with the following code:

```
# file:  test.py
print("Running test.py")
x = 5
print("Result is", 3 * x)
```

To run this file, first make sure it is located in the current directory, and then run the following command in your terminal: `$ python test.py`.

Jupyter Notebook. A useful hybrid of the interactive terminal and the self-contained script is the Jupyter notebook, a document format that allows executable code, formatted text, figures, and even interactive features to be combined into a single document. Though the notebook began as a Python-only format, it has since been made compatible with a large number of programming languages, and is now an essential part of the Jupyter Project.

You can install Jupiter notebook from your terminal using `pip` (or `pip3` – or see here, if you use `conda` as a package manager):

```
$ pip install notebook
```

To open a notebook, run the following command in your terminal:

```
$ jupyter notebook
```

1.2 Python variables and objects

1.2.1 Variables in python.

In python, a variable is a name given to a memory location, where will be stored a value. A variable is created the moment we assign a first value to it (we don't need to declare a variable before using it). The value of a variable can change during program execution. Calling an undefined variable returns a `NameError` message. For instance:

```
>>> print(a)    # Return NameError message, as ‘a‘ is not declared yet
NameError: name 'a' is not defined
>>> a=1     # Declare a variable 'a'
>>> print(a) # Print 'a'
1
>>> a=20        # Redeclare 'a' (i.e., change the value of 'a')
>>> print(a)    # Print 'a'
20
>>> del(a)      # Clear the variable 'a'
>>> print(a)    # Return NameError message, as 'a' is not defined
NameError: name 'a' is not defined
```

Rules for creating variables. In python, variable names:
- can only contain letters, numbers, or underscores;
- cannot start with a number;
- are case sensitive;
- keywords can't be used to for naming variables.

Random tips: ◦ Assigning a single value to multiple variables:

```
>>> a = b = c = 10
>>> print(a, b, c)
10 10 10
```

◦ Assigning different variables in a single line with the operator ",":

```
>>> a, b, c = 10, "test", 200.3
>>> print(a, b, c)
10 test 200.3
```

1.2.2 Objects

Objects. Python is an object-oriented programming language, and in Python everything is an object. Each object has a type, and can have attributes and methods. Attributes contain data. Methods are functions that can act on the attributes of the object.

Dot syntax. Once an object is given an identity (variable name), its attributes and methods can be accessed via the dot syntax.

For instance, in Python even simple types have attached attributes and methods. Numerical types have a `real` and `imag` attribute that returns the real and imaginary part of the value, if viewed as a complex number:

```
>>> x = 4.5
>>> print(x.real, "+", x.imag, 'i')
4.5 + 0.0 i
```

Methods are like attributes, except they are functions that you can call using opening and closing parentheses. For example, floating point numbers have a method called `is_integer` that checks whether the value is an integer:

```
>>> x = 4.5
>>> x.is_integer()
```

```
False
>>> x = 4.0
>>> x.is_integer()
True
```

1.2.3 Types.

Every value in python has a type. There are several simple build-in types, such as the number types (integer, float and complex), Boolean, and string. Python also has more complex types that are built-in data structure, and which we will discuss in the next lecture. The following table summarises python's simple types, and is followed by a few comments on integer and float. You can find more information on simple types here.

Type	Example	Description
int	`x = 1`	Integers (i.e., whole numbers)
float	`x = 1.0`	Floating-point numbers (i.e., real numbers)
complex	`x = 1 + 2j`	Complex numbers (i.e., numbers with real and imaginary part)
bool	`x = True`	Boolean: True/False values
str	`x = 'abc'`	String: characters or text
NoneType	`x = None`	Special object indicating nulls

Integer. Operations on integers are exact. In python2 integers are coded on 32 or 64 bits, but in python3 integer actually have no limits! See here for more information how python3 does it. This however comes at the cost of computational efficiency for integer larger than 64-bits (try `print(sys.maxsize)`). You can convert a string or a float to an integer using `int()`. For a float, the function will just take whatever value that appears before the coma. To properly round a float (up or down) to an integer, use the function `round()`.

Float. Real values in python are of the type "float" (or floating point numbers). Most of the time float in python are coded on 64 bits. The larger absolute value that can be coded with the maximum precision is then $max = 1.7976931348623157e+308$ and the smallest is $min = 2.2250738585072014e-308$. If you want to know more about how float are encoded, check here. To convert a string or an integer into a float, use the function `float()`.

Variable types. In python, a variable automatically takes the type of the value used to define the variable. Besides, variables do not have a fixed type. The type of a variable can change during program execution (i.e. you can use the same name for objects of different types). This happens as variables are in fact pointers to objects, and that the type is a property of an object. While objects have a specific type, variables can point to objects of any type. For example, below, we create a variable `a` and assign it (i.e., make it point) to an integer object with value 1. After that, we re-assign variable `a` so that it points to a string object.

Finally, you can check the current type of a variable using the function `type()`.

```
>>> a=1 # Declare a variable 'a'
>>> print("a =", a, ": the variable 'a'  is an integer.")
>>> type(a)
>>> a="test" # re-declare the variable 'a'
>>> print("a =", a, ": the variable 'a'  is a string.")
>>> type(a)
```

1.3 Operations

1.3.1 Operations.

Operators are used to perform operations, compare objects, or check membership. Below is a list of Python's arithmetic operators with their descriptions. These operators can be used and combined in intuitive ways, using

standard parentheses to group operations. In additions to numerical operations, are also available: Bitwise operations, assignment operations, comparison operations, Boolean operations, and Identity and membership operations. See here for an overview of all the basic operations in python. We will test most of them in the tutorials.

Operator	Name	Description
`a + b`	Addition	Sum of `a` and `b`
`a - b`	Subtraction	Difference of `a` and `b`
`a * b`	Multiplication	Product of `a` and `b`
`a / b`	True division	Quotient of `a` and `b`
`a // b`	Floor division	Integer remainder after division of `a` by `b`
`a % b`	Modulus	Integer remainder after division of `a` by `b`
`a ** b`	Exponentiation	`a` raised to the power of `b`
`-a`	Negation	The negative of `a`
`+a`	Unary plus	`a` unchanged (rarely used)

Example: Using the operator "+" with variables of the same type allows to sum numbers, or concatenate strings for instance:

```
>>> a=10; b=20
>>> print(a+b)
30
>>> a="Hello "; b="world!"
>>> print(a+b)
Hello world!
```

This does not work with variables of different types. The call of the operator "+" with different types will return a `TypeError` message.

```
>>> a=10          # a is an integer
>>> b="Test"      # b is a string
>>> print(a+b)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

1.3.2 Illegal operations.

An "illegal operation" is an operation that is not authorized or not understood by the python interpreter. The operation is terminated and the interpreter returns an error message. See for instance the following error messages:

– `NameError`, which is returned when you call a variable/function that hasn't been defined;

– `TypeError`, which is returned when an operation can't be performed because the provided values are not of the expected types.

Be careful: This is a common error in python, as variable types are not fixed. It is important to properly track the types of your variables while coding. Illegal type operations will raise an exception from your python interpreter. However, because variable types are not fixed, one can end up performing operations that that would differ from the desired operation, but that would still be understood and run by the interpreter.

– `ZeroDivisionError,` returned when there is a division by 0 in the code.

To code more efficiently, learn to recognise error messages.

1.4 Python Language Syntax

- comments are marked with an `#`
- the end of a line terminates a statement; if you would like a statement to continue to the next line, it is possible to indicate it with a backslash "\";
- You can terminate a statement with a semicolon;

- indentation at the beginning of a line matters (they would corresponds to curly braces in C for instance);
- white space within a line doesn't matter;
- parentheses are used to indicate priority of operations, but also for calling a function.

For instance, can you comment on the different syntax points that you see in the following code:

```
# define two variables a and b
a=10;  b=20

# define c and x
c = 3*(b-a)
d = 1 + 2 + 3 + 4 +\
    5 + 6 + 7 + 8

# silly test:
if c < 0:
    print(b)
elif a < 0 and b < 0:
    print(c)
else:
    print(d)
```

For more detailed explanations on the python syntax, check this page.
Add syntax of conditions here

1.5 Python modules

Help command. Within the python interpreter you can use the `help()` command to get information on modules, class, functions, etc. that are available in your current python session. For instance, try `help(int)` to get information about the class int. Try `help(sqrt)` to see what happens when Python does not recognize the name you entered. To quit the help, press the letter `q`.

Within Jupyter notebook, you can also use the command "`?`" to get help. For instance, try `int?` to get information about the integer class. Similar, you can only get information about objects that are available in your current session (for instance, try `sqrt?` to see what happens).

Available objects. To find out which objects are currently available to you, you can use Python's `dir()` command. This is short for "directory," and it returns a list of all the modules, functions, and variable names that have been created or imported during the current session. Try to enter `dir()` in your python interpreter. There is an item called `__builtin__`, which is the collection of all the functions and other objects that Python recognizes when it first starts up. It is Python's "last resort" when hunting for a function or variable. To see the list of built-in functions, type `dir(__builtin__)`. Note that there is no `sqrt` function, or any other common mathematical functions, such as sine, cosine, or exponential.

Modules. The sqrt function we seek belongs to a library. Python also has a large community of developers who have created entire libraries of useful functions. To gain access to these, however, you need to import them into your working environment. Use the import command to access functions that do not come standard with Python.

Useful built-in modules: Python comes with a `math` module that contains most basic mathematical functions, as well as a `cmath` modules for mathematical functions defined for complex numbers. To import the module, run `import math`, or `import cmath`.

1.6 Good programming habits

- **Give meaningful names** to your variables and functions.
- **Add comments** across your code. To comment a part of a line, precede it with the character `#`. For example: `>>> a=1 # a is an integer`
- Decompose your programs in small functions, rather than working out a large program; i.e., defined intermediate functions, as well as variables.
- **Clarity:** Clean up useless variables and operations. In some situations, you may wish to use a very long command that doesn't fit on one line. For such cases, you can end a line with a backslash. Python will then continue reading the next line as part of the same command. Try this:
- **Test!** Don't wait for having written the whole program to start testing; test each function as soon as they are written, and test sub-part of a function if the function is long;
- Errors: read and try to understand the error messages; google error messages that you don't understand;
- Bugs: comment parts of the program and run test on smaller pieces of the program to figure out where a bug or an error comes from. Use debugging function of Jupyter notebook.
- **look for information/help in the documentation and online.** A python documentation is available online at https://www.python.org/doc/. You will also often find answers to your questions on the web or by visiting stackoverflow.com. You can also get help directly from Python for build-in functions by using the `help()` command. For instance to ask information about the function `round`, type `help(round)` at the command prompt. To quite the "help", press "q";
- More advanced: Remember to free out memory.

The Zen of Python, by Tim Peters. Type `import this` in your python interpreter.

1.7 Comment on using python virtual environments

Why using virtual environments? It is common to have program that only work with specific version of python and libraries. In that sense you may need to have several versions of python installed on your laptop, as well as for each version different set of libraries. Having virtual environments allow you to manage all of these versions, and always run your programs with the correct version of python.

How to create your own virtual environment? You can set as many virtual environment on your laptop as you need. The best is to install each of them in a different folder, at the root of your user profile.

How to create to start your virtual environment? To start a virtual environment, use the command `source` in your terminal in the following way:

```
source ./.venv/bin/activate
```

in which you can replace `.venv` by the address of the folder in which is installed the virtual environment that you want to activate (i.e. the one you want to use). Once you are in the virtual environment, you can see its name appearing between parenthesis at the beginning of each prompt line:

```
(.venv39) username@laptop ~ %
```

Finally, to quit the virtual environment, just type `deactivate` and press enter.

Pyenv. The advantage of pyenv is that it comes with many virtual environment already created, such as anaconda, or mini-conda, and it is then easy to download and install them. To see the list of virtual environment you can install with pyenv, type

```
pyenv install --list
```