Python 101
Introduction

PYTHON 101 TEAM

Instructors

Bruno Costa
Bruno Vieira @bmpvieira
Diogo Silva @odiogosilva
Francisco Pina-Martins github
Joana Fino
João Silva

What is python?

Python is an **interpreted**, **object-oriented**, **high-level** programming language, that emphasizes in code readability:

- **Interpreted**: Programs can be run as soon as they are written, no need to compile;
- **Object-oriented**: The programming paradigm, where "objects" can be somewhat separated from the rest of the program. An object is an entity that has a:
  - **Identity**: Unique identifier of the object
  - **Value**:
  - **Type**: whether it is a string, number, etc.
- **High-level**: Abstracts from the computer it is being run on;

Course overview

- **Day 1**: Installing and configuring Python; Introduction to Python programming
- **Day 2**: Data structures
  - Strings
  - Lists
  - Tuples
  - Dictionaries
  - Sets
- **Day 3**: Control flow

    - Conditional statements (if, elif)
    - Loops (while, for)
- **Day 4**: Functions and Input/Output
- **Day 5**: Modules (*sys*, *os*, *re*, *Bio*) and Do It Yourself day

Introducing Python programming

Data structures (Day 2)

- Basic operators ("+","-","/","*")
- Variable assignment
- Standard data structures (Strings, numbers, lists and dictionary)

Control flow (Day 3)

- Control flow is used in Python whenever you want to:
    - **Make choices**, using *conditional statements* when conditions are met
    - **Perform repeated actions**, using a set of statements executed *n* times in a *loop* until a condition is met

Introducing Python programming

Functions (Day 4)

- Blocks of code that perform specific tasks (procedures) repeatedly
- How to create new functions
- Using arguments to bring flexibility to those procedures

Input/Output (Day 4)

- The three input/output channels: standard input, standard output and standard error
- Handling input files (opening and parsing) and output files (writing)
- Interactive scripts that request user input
- Printing messages to the terminal

Introducing Python programming

Modules (Day 5)

- What are modules, how do they work and how to use them

- Overview of some of the most useful modules:
  - *re*, regular expressions
  - *sys*, system tools
  - *os*, operative system tools and commands
  - *biopython*, the bioinformatic's module of choice

Online resources

Some of the most usefull resources that will help you solving problems and getting new ideas during your coding sessions:

- Official Python tutorial: From the official Python documentation, this resource covers all basic and advance built-in features of Python.
- Python wiki: Includes several oficial Python resources, such as the beginners guide, beginners errors and Python books.
- Stack Overflow: An extremely usefull language-independent collaboratively edited question and answer site for programmers. Most of the questions and problems that you may face when coding have been probably already answered in this forum.

Google is your friend

- Giving the current flood of digital data, the importance of google's search engine to sort over all information cannot be overstated. The trick, however, is knowing how to put your problem in order to obtain the most usefull answers.

UNIX

The UNIX shell

- The UNIX shell acts as an interface between the **user** and the **kernel**, which is the hub of the operating system

```
diogo@diogo-RF510-RF410-RF710:~$ 
```

- Through the shell, the user has a wide range of programs at its disposal that allow file manipulations, navigation through the directory structure, etc. Here we will provide a short tour through the shell using some of the most useful programs:
  - **ls**

- **cd**
- **touch**
- **cp**
- **mv**
- **rm**
- **mkdir**

A tour through the shell

Asking for help

- Linux provides documentation and help for all of its programs:
  - using the **man** command before the program name

```
man ls # Manual for the ls command
```

- using the **whatis** command before the program name

```
whatis ls # Brief description of the ls command
```

- using the **--help**/**-h** option after the program name

```
ls --help # Provides information on the usage and options of the ls command
```

A tour through the shell

Navigating through the directory structure

- We will start in the home directory (~), which is familiar to most unix users. To view the contents of the home or any other directory, we can use the **ls** command

```
ls # Shows the contents inside the current dir
ls ~/ # Shows contents inside the home dir
ls path/to/dir # Shows contents inside the specified dir

```

- The **ls** command has some useful options:
  - **-a**: Shows hidden files
  - **-d**: Lists directories instead of contents
  - **-l**: Long listing format, with aditional details

A tour through the shell

Navigating through the directory structure

- To change our current directory from home, we can use the **cd** command

```
cd path/to/dir
cd # Returns you to the home directory
cd ~/ # Equivalent to "cd"
cd ./ # Stays on the current directory
cd ../ # Goes to the parent directory

```

- If you ever get lost and wish to know your location you can use the **pwd** command

```
pwd # Returns the current directory

```

A tour through the shell

Creating new directories

- We can create a new directory for the material of this course using the **mkdir** command

```
mkdir ~/Python101 # Creates the Python101 directory in the home dir
mkdir Python101 # Creates the Python101 directory in the current dir
```

Manipulating files

- To create new files in which python scripts can be written, use the **touch** command

```
touch my_script.py # Creates a new file name my_script.py in the current dir
```

- To copy or move/rename files or directories use the **cp** and **mv** commands, respectively

```
cp my_script.py /usr/local/bin # copies the my_script.py file in the current dir to the /usr/local/bin/ dir
cp -r my_folder/ ~/ # Copies my_folder/ dir recursively to the HOME dir
mv my_script.py ~/ # Moves the my_script.py file to the home dir
mv my_script.py my_program.py # Renames my_script.py to my_program.py
```

A tour through the shell

Manipulating files

- To remove files or directories, use the **rm** command

```
rm my_script.py # Removes my_script.py in the current dir
rm ~/my_script.py # Removes my_script.py from the HOME dir from any dir
rm -r my_folder/ # Removes my_folder/ in the current dir recursively
```

- To change the file owner and group, use the **chown** command

```
chown USER FILENAME # Changes the user owner of the FILENAME
chown USER:GROUP FILENAME # Changes the user and group ownership of the FILENAME
chown :GROUP FILENAME # Changes the group owner of the FILENAME
```

A tour through the shell

Manipulating files

- To change file access permissions, use the **chmod** command

```
chmod MODE FILENAME
 Symbolic mode
chmod -rwx FILENAME # Removes ("-" symbol) read ("r"), write ("w") and executable ("x") permissions of the FILENAME for the owner, group and others
chmod +rwx FILENAME # Gives ("+" symbol) read, write and executable permissions to FILENAME for the owner, group and others
 Numerical mode
chmod 755 FILENAME # Changes FILENAME permisions to read, write and executable for the owner ("7") and read and executable for the group and others
```

- **Numerical mode rational**: File permissions are represented by a three-digit octal number. These numbers are added accordingly, using these values:
    - 4 = read (r)
    - 2 = write (w)
    - 1 = execute (x)
    - 0 = no permissions (-)

A tour through the shell

Using the terminal screen

- To clear the terminal window from previous commands so that the output of subsequent commands can be more clearly visualized, use the **clear** command

```
clear
```

- To display the contents of a file on the screen, use the **cat** command

```
cat FILENAME
```

- To display the contents of a file on the screen one page at a time, use the **less** command

```
less FILENAME # Use the keywords "space" and "b" to move forward and backwards, respectively
```

A tour through the shell

Using the terminal screen

- The initial or final contents of a file can also be visualized on screen using the **head** and **tail** commands

```
head FILENAME # prints the first 10 lines of the FILENAME
head -n 50 # The -n option changes the lines read to 50
less FILENAME # prints the last 10 lines of the FILENAME
less -n 30 # Same as the -n options for the head command

```

Filtering the screen output

- The output that is printed on the terminal screen can be filtered/searched using the **grep** command

```
grep [options] PATERN FILENAME # grep searches the FILENAME for a given PATTERN and prints lines with that pattern on the screen

ls | grep PATERN # grep can be used with the pipe (|) to filter the output of other shell commands
```

A tour through the shell

Redirection and piping

- To redirect output the output of any shell command, use the or symbols

```
cat FILENAME > new_file.txt # The output of the cat command is redirected to a new file, instead of the terminal screen. The ">" symbol overwrites any contents in the original new_file.txt
grep PATERN FILENAME > new_file.txt # The output of the grep command is redirected to a new file. The ">" symbol appends the output to the end of the new_file.txt

ls | grep PATERN > new_file.txt
```

- Pipes ("|") allow separate processes to communicate implicitly, enabling narrow tools to be combined in complex ways

```
ls | grep PATERN # allows communication between "ls" and "grep". Searches for PATERN using grep, from the output of ls

cat FILENAME | wc # The FILENAME output is fed to the "wc" command, which counts the no. of words in a file
```

Installing python

Linux

- In order to install everything you will need for the course all you have to do is issue the following command in the terminal:

```
sudo apt-get install python2 ipython-notebook geany idle python-biopython
```

- Your package manager will handle everything for you.

Mac (Part I)

These commands will install Python and some Scientific modules.

- 1 Download and install Xcode from app store
- 2 Download and Install Command Line Tools from Xcode

```
xcode-select --install
```

- 3 Install Homebrew package manager

```
ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)"
```

- 4 Add Homebrew repositories for science and python

```
brew tap homebrew/science
brew tap homebrew/python
```

- 5 Update Homebrew packages

```
brew update
```

Mac (Part II)

- 6 Install Python

```
brew install python
```

- 7 Install SciPy

```
brew install gfortran
pip install scipy
```

- 8 Install matplotlib

```
pip install matplotlib
```

- 9 Install ipython

```
brew install pyqt
pip install pyzmq
pip install pygments
pip install jinja2
pip install tornado
pip install ipython
```

- 10 Install biopython

```
pip install biopython
```

Windows

Installing python

Python installation

- For this course, we will use the EPD (Enthought Python Distribution) package as it already contains a number of usefull modules and a text editor.

Biopython installation

- The Biopython module can be installed through an executable file here.

IPython installation

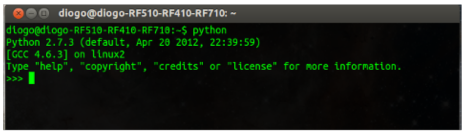
- The installation of IPython on Windows is greatly facilitated by EPD. Just open a terminal window and write:

```
enpkg ipython
```

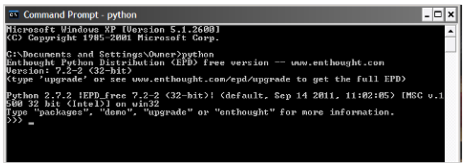
Two ways of running python code

The interactive mode

- Ideal for **training** and **testing blocks of code** on the fly. During an interactive session, instructions are directly executed. However, the code cannot be saved between different sessions. Some interactive interpreters include:
  - Python's built-in interpreter
  - Python IDLE
  - Dreampie



```
diogo@diogo-RF510-RF410-RF710:~$ python
Python 2.7.3 (default, Apr 20 2012, 22:39:59)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Owner>python
Enthought Python Distribution (EPD) free version -- www.enthought.com
Version: 7.2-2 (32-bit)
(type 'upgrade' or see www.enthought.com/epd/upgrade to get the full EPD)

Python 2.7.2 |EPD_free 7.2-2 (32-bit)| (default, Sep 14 2011, 11:02:05) [MSC v.1
500 32 bit (Intel)] on win32
Type "packages", "demo", "upgrade" or "enthought" for more information.
>>> _
```

- The symbols "> > >" are the prompt of the interactive mode. To execute an instruction, write on the prompt line and press enter.
- The symbols "**...**" are the secondary prompt. They represent continuation lines, i.e., when the instruction is not yet complete and ready to be executed.
- To exit the interactive mode, you can use the key combo "**Ctrl + D**" or type "**quit()**"

Script mode

Writing scripts using a text editor (geany, vim) will make the majority of your coding experience. A python script that contains a set of instructions can be executed in more than one way:

- **Shebang way** (Unix only):
  1. The first line of the script file must contain a "shebang" (#!) followed by the path of the Python executable (e.g. #!/usr/bin/python). This will tell the program loader which interperter should be used to execute the file.
  2. Make the script file executable (usually with the "chomd +x filename" command)
  3. Run the script by typing "./my_script.py" in the terminal prompt.
- **Script argument way**:
  - Run your python interpreter in the prompt with the script as an argument (e.g. python my_script.py). This mode is prefered for scripts that require arguments to be passed when executed
- **Double clicking**:
  - Run your script simply by double-cliking it. However, this mode will only be useful when the user input is collected using the prompt feature of the **input()** and **raw_input** functions, that will be covered in the 4th Day

Notes for windows users:

- Script files **must** end with a ".py" extension