Other languages and C++
Writing scripts

Florido Paganelli
Lund University
florido.paganelli@hep.lu.se

Outline

- Presentation of the languages we will use
- Practical use of interpreted languages to ease a programmer's life
  - Writing Bash script
  - Writing a Python script

Languages we will use

- Bash, seen in tutorial
- C++, seen in previous lecture
- Python: an interpreted language

Goals and non-goals of this tutorial

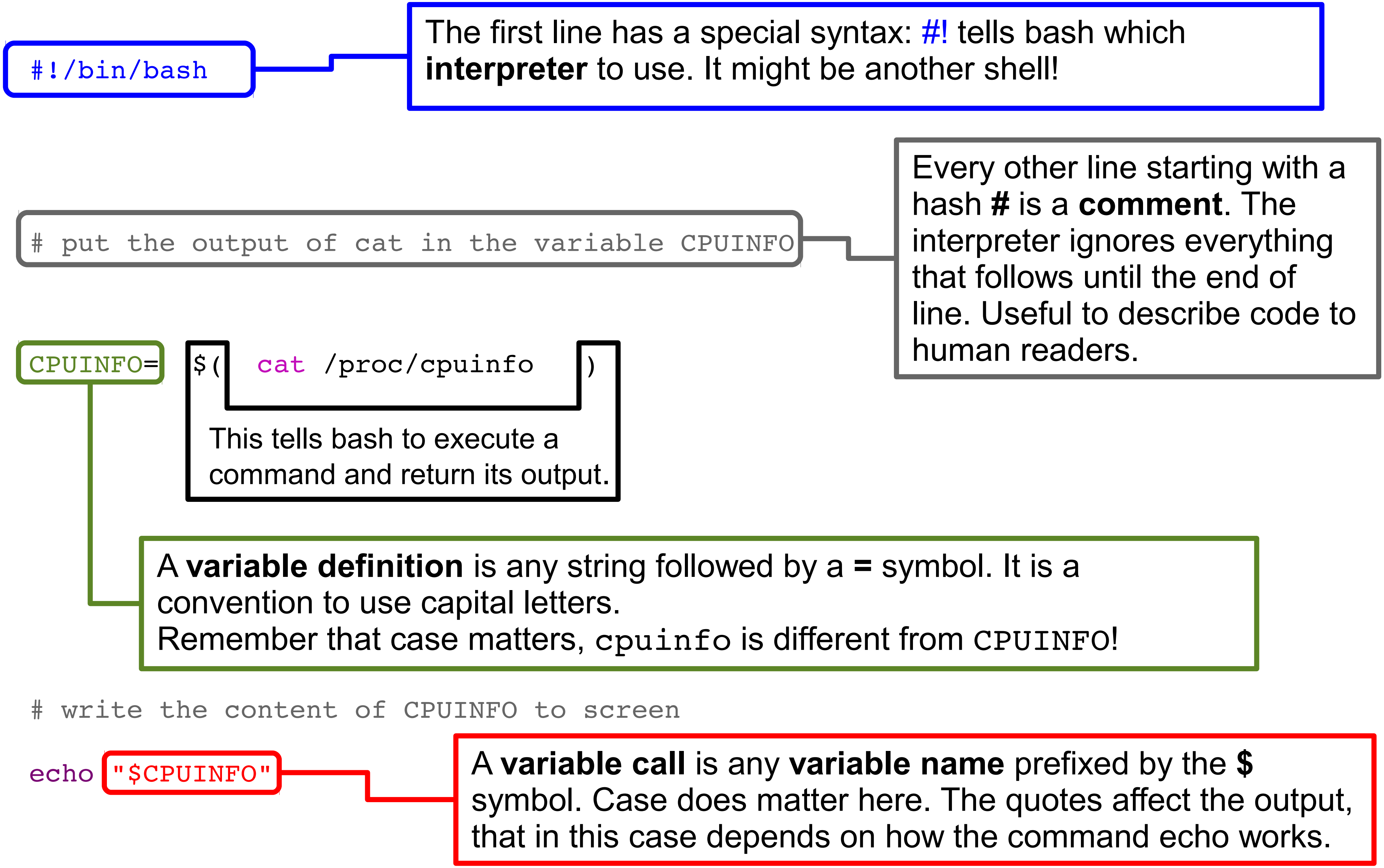
- Goals:
  - Being able NOT TO PANIC when somebody gives you something you've never seen before (will happen in your entire career)
  - Being able to write a bash script.
  - Being able to understand basic python syntax.
  - Being able to search for information depending on a task one wants to achieve.
- Non-goal:
  - Become a script-fu master. It takes long time for the black belt :)
  - Become a python coder. We cannot do this in a lecture, there's full courses out there

A bash script and its components

- A **bash script** is nothing more that a sequence of commands written in a file.
- The bash interpreter will process those in sequence, from the top line to the bottom
- Like C++, is possible to define **variables** and **control structures** in the scripting language.
- However, the bash script language has little to share with the complexity of C++. All that it can do is to execute commands and store things in variables.
- **Exercise:** Open geany, write and save the following code as `getcpuinfo.sh`

```
#!/bin/bash

# put the output of cat in the variable CPUINFO
CPUINFO=$(cat /proc/cpuinfo)

# write the content of CPUINFO to screen
echo "$CPUINFO"
```

Anatomy of a bash script



```
#!/bin/bash

# put the output of cat in the variable CPUINFO

CPUINFO=$( cat /proc/cpuinfo )

# write the content of CPUINFO to screen

echo "$CPUINFO"
```

The first line has a special syntax: #! tells bash which **interpreter** to use. It might be another shell!

Every other line starting with a hash **#** is a **comment**. The interpreter ignores everything that follows until the end of line. Useful to describe code to human readers.

This tells bash to execute a command and return its output.

A **variable definition** is any string followed by a **=** symbol. It is a convention to use capital letters.
Remember that case matters, `cpuinfo` is different from `CPUINFO`!

A **variable call** is any **variable name** prefixed by the **$** symbol. Case does matter here. The quotes affect the output, that in this case depends on how the command echo works.

Features of bash scripting

- The script can be **made executable** as if it was a command. Commands not in the PATH must have a directory path identified. To run those in the current directory, prefix them with **./**

```
pflorido@tjatte:~> chmod +x getcpuinfo.sh
pflorido@tjatte:~> ./getcpuinfo.sh
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model          : 15
model name : Intel(R) Core(TM)2 CPU      6400 @ 2.13GHz
stepping  : 6
cpu MHz          : 2127.650
```

- The **environment** of a bash script does NOT affect the environment where the script is called. If one wants to affect the external environment, the script must be "*sourced*", that is, imported in the existing environment.

```
pflorido@tjatte:~> ./getcpuinfo.sh
Processor : 0 ...
pflorido@tjatte:~> echo "$CPUINFO"

pflorido@tjatte:~> source ./getcpuinfo.sh
pflorido@tjatte:~> echo "$CPUINFO"
processor : 0
vendor_id : GenuineIntel
cpu family     : 6
model          : 15
model name     : Intel(R) Core(TM)2 CPU          6400  @ 2.13GHz
stepping  : 6
cpu MHz        : 2127.650
```

Features of bash scripting

- One can define functions to reduce complexity and increase readability

```
#!/bin/bash

# a function that gets meminfo
getmeminfo(){
MEMINFO=$(cat /proc/meminfo)
}

# execute the function, it will change the environment
getmeninfo

# write the content of MEMINFO to screen
echo "$MEMINFO"
```

- The example above also shows that the variables are **always global** (any part of the program can access them). There is a way of scoping them, but since is not widely used, we will not cover it.
  Bash variables have **no type**, but most of the time is just **strings**.

Exercise D6.1: Add to the getprocinfo.sh script a line that outputs information about the number of cores per processor. Use the pipe | with `echo`, `grep` and `wc` to count the number of cores per processor.
Hint: the line starting with '`processor`' in `/proc/cpuinfo` is repeated as many times as the number of cores.

Exercise D6.2: Debugging to debug your script, that is, see what is doing while running, modify the first line this way:

```
#!/bin/bash -x
```

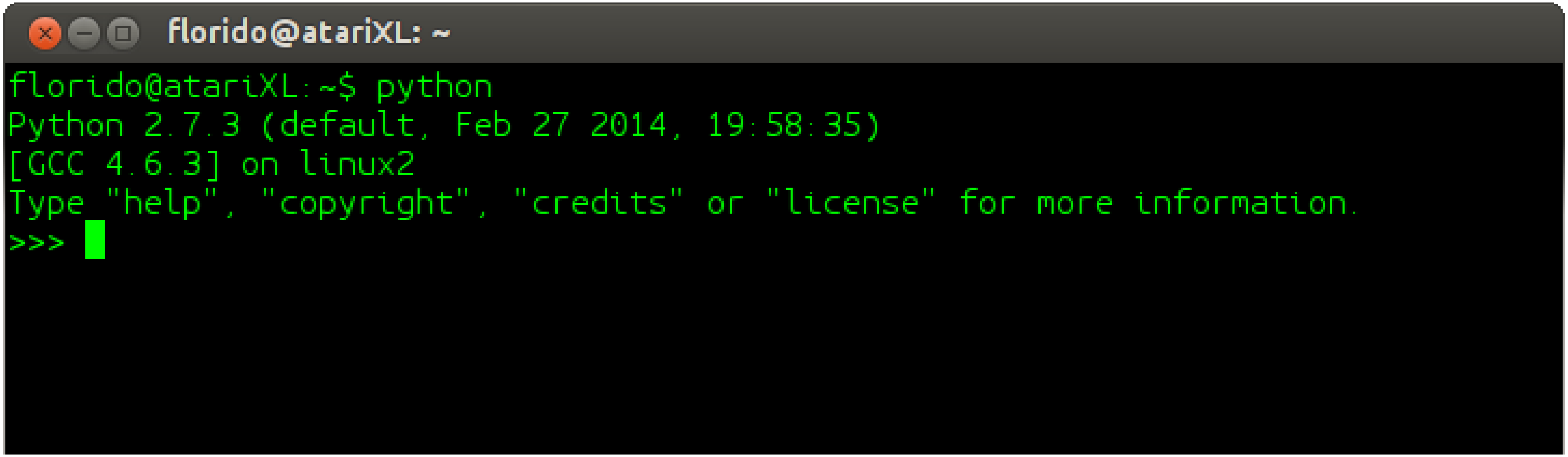
Python

- Interpreted, code is compiled on the fly
- Widely used in the scientific community
- Easy to learn
- Good for quick proof-of-concepts, even involving complex calculations (there are a lot of nice libraries out there)

The Python interpreter

1) Open the terminal

2)Run the python interpreter:



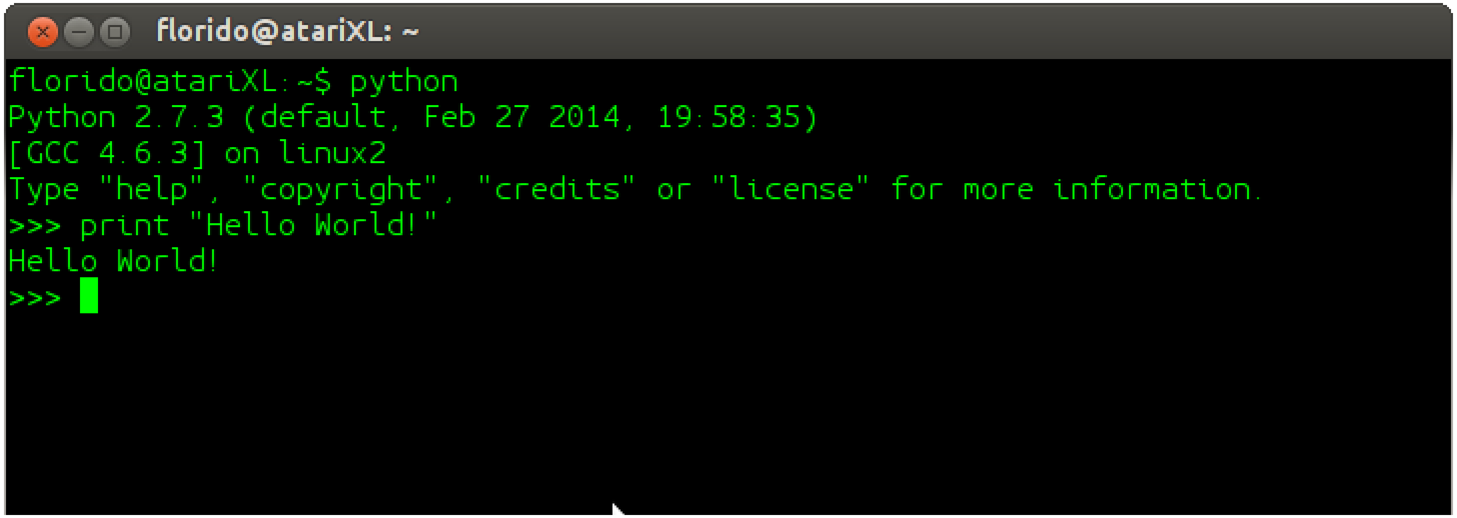
```
florido@atariXL:~$ python
Python 2.7.3 (default, Feb 27 2014, 19:58:35)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Your first Python program

- As python is interpreted, you can directly write programs in the interpreter console.
- Try to write:

```
print “Hello World!”
```

and press enter.



```
florido@atariXL:~$ python
Python 2.7.3 (default, Feb 27 2014, 19:58:35)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello World!"
Hello World!
>>> 
```

Your first python program cont.

- It is however very unpractical to write a program on the fly. It's better to save it to a file as seen for C++.
- Python code is conventionally added in a file with extension `.py`. This is not very important for the code to work, but on some systems like windows the extension matters.

Your first python program cont.

- Let's create a python *script* that prints “Hello Word”.

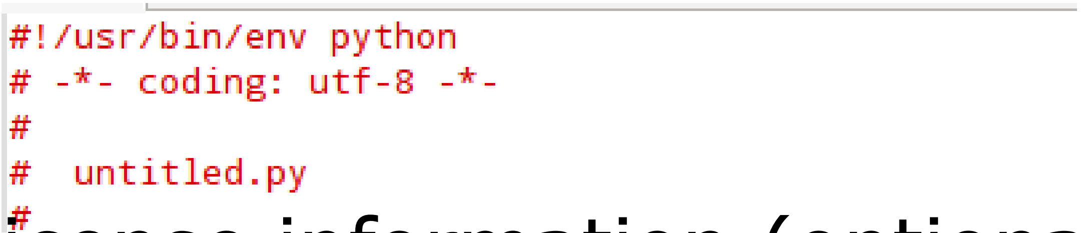
1) Open you favorite editor. In this tutorial we will use *Geany*.

2) Click on the File menu → New (with template) → main.py

3) Let's analyze the structure of the shown python file. Any analogy with C++?

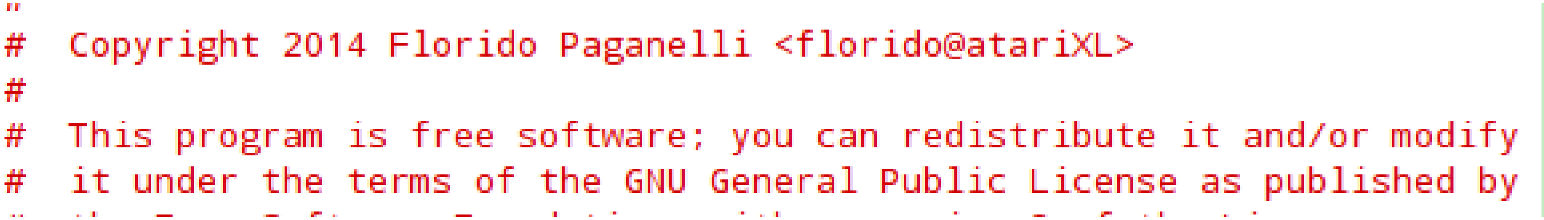
Python program structure

1) The header



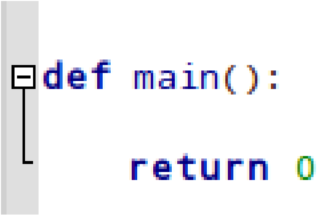
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#  untitled.py
#
```

2) License information (optional)



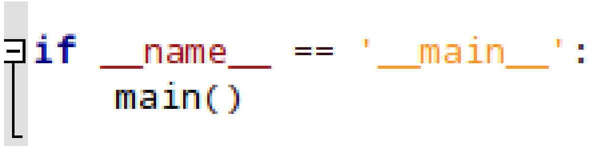
```
#  Copyright 2014 Florido Paganelli <florido@atariXL>
#
#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
```

3) The main function



```
def main():

    return 0
```

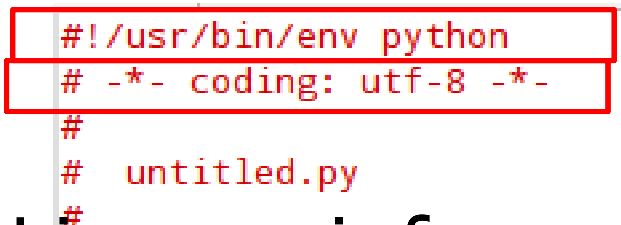
4) The main function callback



```
if __name__ == '__main__':
    main()
```

Python program structure

1) The header

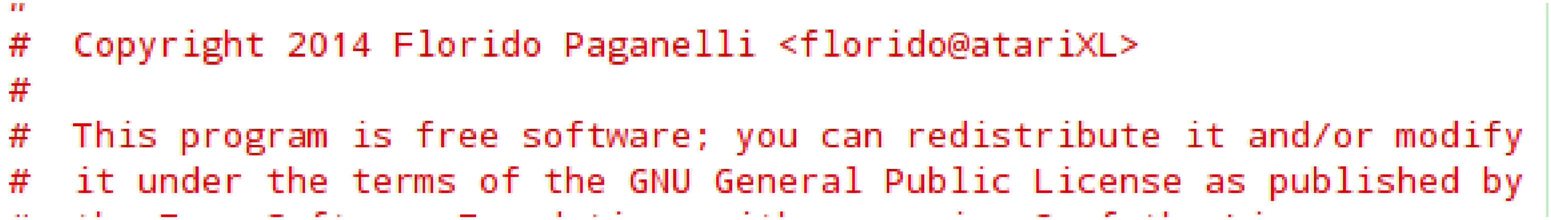


```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#  untitled.py
#
```

Required: Tells command line to use Python interpreter

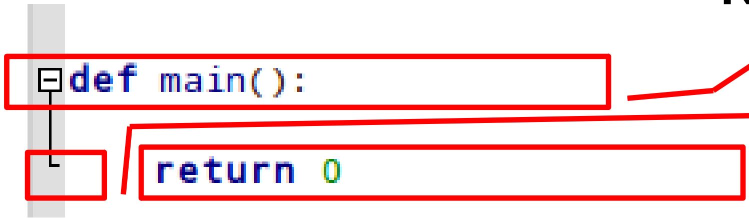
Optional but recommended: Info about encoding

2) License information (optional)



```
#  Copyright 2014 Florido Paganelli <florido@atariXL>
#
#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
```

3) The main function



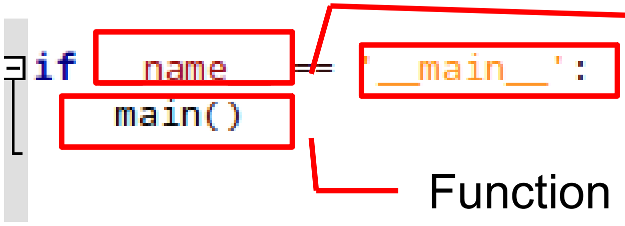
```
def main():

    return 0
```

Required: definition of the main function

Required: function body indentation. All you code goes before `return 0`

Recommended: function return value

4) The main function callback



```
if __name__ == '__main__':
    main()
```

Special variable that asks the interpreter the predefined variable __name__ that tells the name of the default function

Special name of a function

Function call

Python syntax and execution

- Syntax features:
  - **Indentation** (tabs and spaces) is one of the ways to identify a *block of code* in Python. It is fundamental: the author enforced it for readability of code. Python will fail to compile and write out an error if indentation is bad.
  - **;** is the instruction **separator**, is not as important in Python as in C; it can be omitted if indentation is well done.
- Runtime features:
  - `main` will be executed as the first function by the python interpreter.
  - Therefore our print "Hello World" command goes right before the return statement, indented as the return statement, followed by a ;
  - See `helloworld.py`
- Question: why is the `if` executed?

helloworld.py

```
def main():
TAB print "Hello World!";
TAB return 0

if __name__ == '__main__':
main()
```

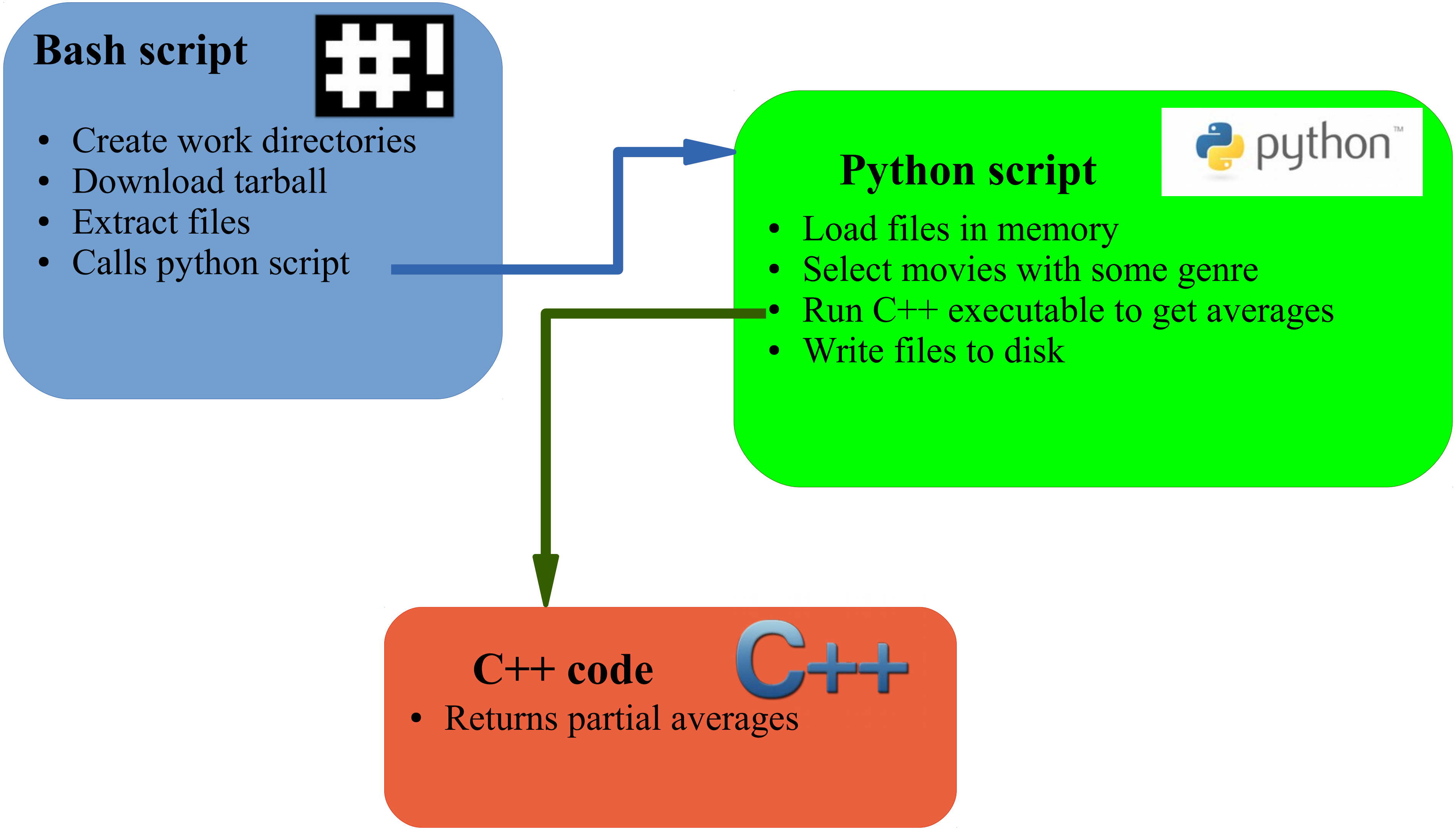
Composition of langues

- Cornerstone of open source programming: if something exist that does a task, and it does it good, use it and do not rewrite code
- Automation of repetitive tasks and interoperability within languages
- Technique: identify subproblems and separate tasks, increasing debuggability

Toy problem we will solve

- Fetch a document (or a collection of documents) from the internet (bash)
- Modify their contents adjusting it to our needs (python)
- Write the result on disk according to some strategy that involves creation of folders and structured filenames (python)
- Integrate a C++ program that calculates the average
- Download the source code from http://www.hep.lu.se/staff/paganelli/doku.php/it_services:other

Toy problem overview



Bash script

- Create work directories
- Download tarball
- Extract files
- Calls python script

Python script

- Load files in memory
- Select movies with some genre
- Run C++ executable to get averages
- Write files to disk

C++ code

- Returns partial averages

Task D6.3: using bash

1. Create a folder called `wip` (use `mkdir`) where we will do other tasks, and `cd` into it.
2. Download the file located at:
   http://www.hep.lu.se/staff/paganelli/lib/exe/fetch.php/it_services:movies.tar.gz
   And give it the filename: `tarball.tar.gz`
   (Hint: see `man wget`)
3. Extract the file with `tar`
   (Hint: see `man tar`)

Tasks using python

- Inspect the files we just downloaded with `geany` or `cat`. How do they look like? This is a know format called CSV (Comma Separated Values).
  - Discuss with the teacher about notable info on the format.

Sample data file

```
"imdbID","Title","Genre","Director","Country","imdbRating","imdbVotes"
"tt0090084","Storm","Action, Comedy","David Winning","Canada","5.2","53"
"tt0090086","Strannaya istoriya doktora Dzhekila i mistera Khayda","Mystery, Sci-
Fi","Aleksandr Orlov","N/A","6.2","21"
"tt0091002","Eleven Days, Eleven Nights","Drama, Romance","Joe
D'Amato","Italy","3.3","370"
"tt0091012","Equalizer 2000","Action, Adventure, Sci-Fi","Cirio H. Santiago","USA,
Philippines","3.9","180"
"tt0091017","L'escot","N/A","Antoni Verdaguer","Spain","4.8","8"
"tt0091026","Eye of the Eagle","Action, Adventure, War","Cirio H. Santiago","USA,
Philippines","4.5","72"
"tt0091062","Florida Straits","Action, Adventure, Romance","Mike
Hodges","USA","5.5","160"
"tt0091073","Francesca","Comedy, Drama","Vérénice Rudolph","West Germany","N/A","N/A"
"tt0091090","Fu gui bi ren","Comedy, Family, Fantasy","Clifton Ko","Hong
Kong","6.6","97"
"tt0091092","Fuegos","N/A","Alfredo Arias","France","4.0","8"
"tt0091094","Funland","Comedy","Michael A. Simpson","USA","4.4","227"
```

Python variables

Start the Python interpreter (command: python) and try the following:

```
Python 2.6.6 (r266:84292, Aug 12 2014, 07:57:07)
[GCC 4.4.5] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> a = 3
>>> b = 'hello!'
>>> print a,b
3 hello!
```

- The Python interpreter allows you to see the content of every variable by writing its name. Try writing a and b and then press enter!
- Use the builtin `len(variable_name_here)` function to see how “big” is a variable. What happens?
- More about builtin functions: https://docs.python.org/2/library/functions.html

Python dict

- Start the Python interpreter (command: python) and try the following:

```
>>> dict = { 'name': 'florido', 'surname': 'paganelli' }
>>> print dict
{'surname': 'paganelli', 'name': 'florido'}
>>> print dict['name']
florido
>>> dict['name']='Rudolph'
>>> print dict['name']
Rudolph
>>> dict['Address']='unknown'
>>> print dict
{'surname': 'paganelli', 'name': 'Rudolph', 'Address': 'unknown'}
```

See:
https://docs.python.org/2/library/stdtypes.html#mapping-types-dict

Python list

- Start the Python interpreter (command: python) and try the following:

```
>>> list = [ 'apple', 'pear', 'banana' ]
>>> print list[1]
pear
>>> list[3]='orange'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
>>> list[2]='orange'
>>> print list
['apple', 'pear', 'orange']
>>> list.append('peach')
>>> print list
['apple', 'pear', 'orange', 'peach']
```

See:

https://docs.python.org/2/library/stdtypes.html#sequence-types-str-unicode-list-tuple-bytearray-buffer-xrange

Task D6.4 – load data in memory

- Load the data in the files we just downloaded into a variable
  - Learn how to open a file in python
  - Learn how to use the csv library https://docs.python.org/2/library/csv.html
  - Organize the movie records in a python dictionary **dict**
  - Add each record in a python **list**
  - Print the list (and learn how to PrettyPrint)
- Let's look at the code!

exerciseD6.4.py

- Go in the wip folder you created:
  ```
  cd wip
  ```
- Copy exerciseD6.4.py into the folder with:
  ```
  cp ../python/exerciseD6.4.py .
  ```
- Let's discuss about it and then run it!

Python function

- Declaration:

```
>>> def myfunction(adictionary):
...  TAB return adictionary.keys()
...
>>> print myfunction
<function myfunction at 0x7ffe99b35230>
```

Remember tabs!

- Function Call:

```
>>> myfunction(dict)
['surname', 'name', 'Address']
>>>
```

Task D6.5 - exercise
Refactor code into functions

- Identify chunks of code that can be moved inside functions
- Replace blocks of code with function calls
- Try to refactor the code that opens a file and creates the db into a new function called `createdb(dirpath)`
- `dirpath` is the input argument of the function; the function should be called with a string that is the directory where the movies folder is located.

ExerciseD6.5.x.py

- exerciseD6.5.first.py shows a solution for the previous exercise
- exerciseD6.5.better.py shows a better refactoring. Let's have a look at it.

Task D6.6
Select subset of the dataset

- Select only movies that belong to a **genre** and write the selection to a file. We will use **Comedy**

Task D6.7
Integration with external programs

- You are given a C++ program that does partial averages. Call the code to calculate the average vote of a Genre.
- Let's have a look at the C++ code quickly, and then we'll see how to integrate an external program into python.

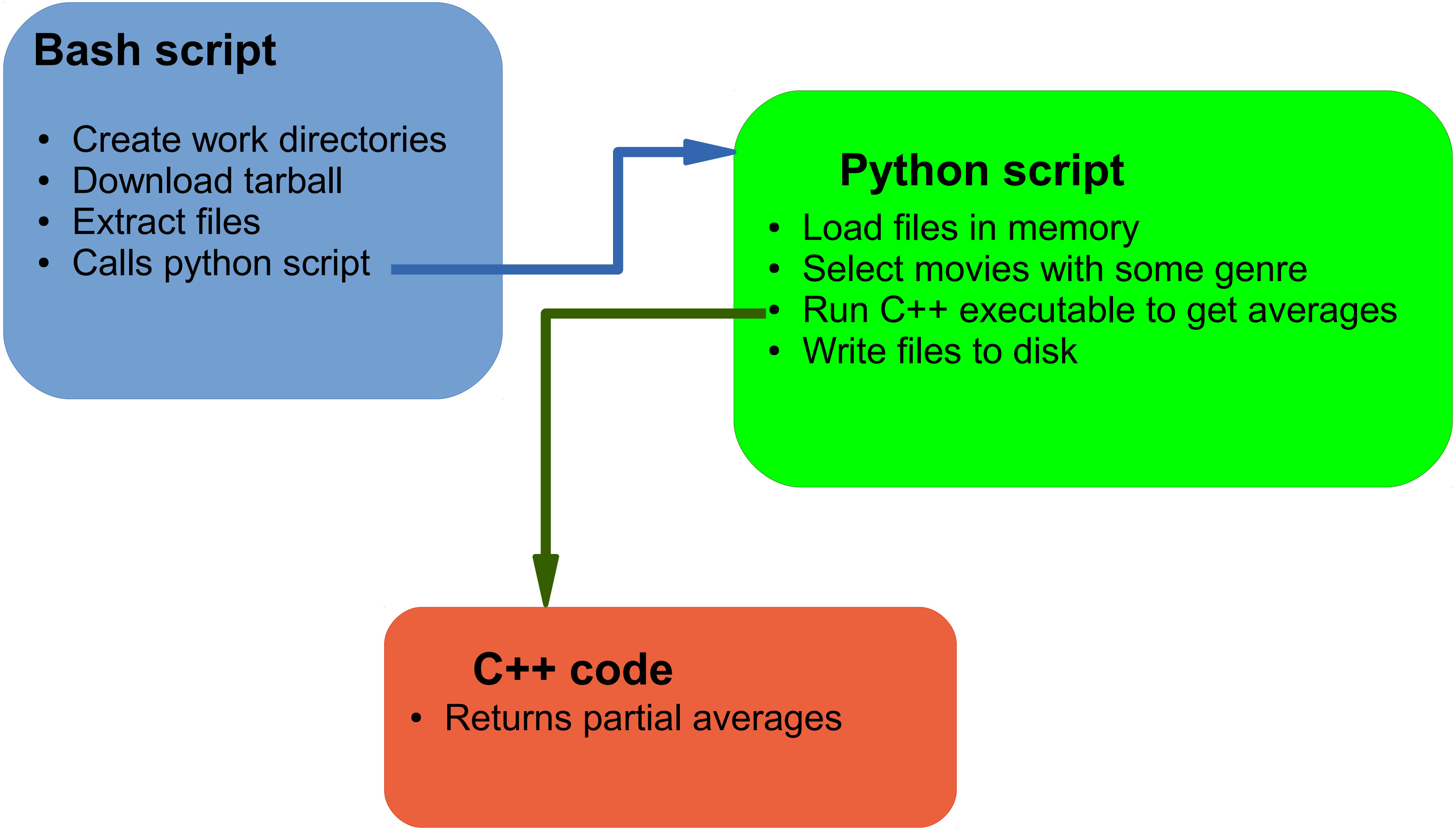
Task D6.8: integration with bash

- Modify the Python script to accept Genre as a parameter
- Modify the bash script to call the Python script on some genres (guided, we will use Bash control structures)

Task D6.9: exercise
create a fully automated script

- Enhance the python script with the ability to write out selected data, that includes:
  - Title
  - imdbRating
  - imdbVotes
- Extend the bash script to download the input data and pass the proper options to python to calculate the average and create the files.

Task D6.9: create a fully automated script



Bash script

- Create work directories
- Download tarball
- Extract files
- Calls python script

Python script

- Load files in memory
- Select movies with some genre
- Run C++ executable to get averages
- Write files to disk

C++ code

- Returns partial averages

Notable Python libraries and IDEs

- Libraries:
  - **Scipy**, for scientific computing
  - **Matplotlib**, to draw plots from scientific data
  - **Ipython**, an interactive environment like mathematica or matlab
- IDEs:
  - **Eclipse**, written in java
  - **Spyder**, specific for scientific programming
  - **Eric**

Missing but worth a look

- Regular expressions and string operations: Python is very good at it https://docs.python.org/2/library/re.html
- C++ libraries compatibility https://docs.python.org/2/extending/extending.html
- Python objects: https://docs.python.org/2/tutorial/classes.html

References

- Bash scripting: http://tldp.org/LDP/abs/html/
- Python documentation: https://docs.python.org/

Hacker's wisdom fun

```
#!/bin/ssh
#The Unix Guru's View of Sex
unzip ; strip ; touch ; grep ; finger ; mount ; fsck ;
more ; yes ; umount ; sleep
```

http://www.ee.ryerson.ca/~elf/hack/ugvs.html