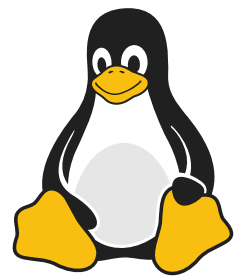
GNU/Linux

A short introduction to Python
Administration script with Python

Nicolas Delanoue

Université d'Angers - Polytech Angers

POLYTECH°
ANGERS



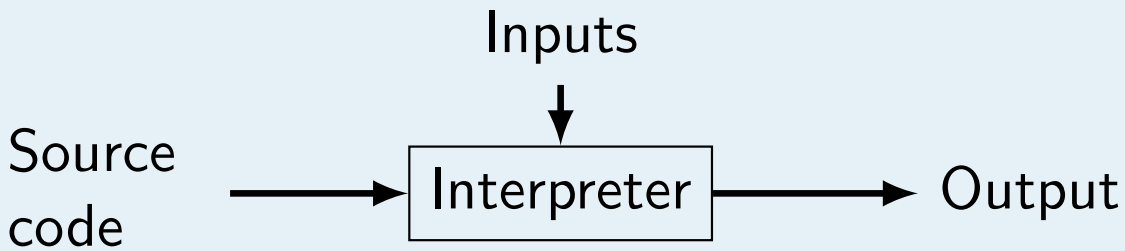
Some remarks about Python

- Interpreted,
- Multiplateform,
- Multiparadigm (imperative, object, fonctional, ...)
- Huge community : more than 50 000 libraries,
- Langage with a lot of applications :
  - web development,
  - datebase access,
  - system administration,
  - GUI,
  - scientific computations,
  - embedded systems,...

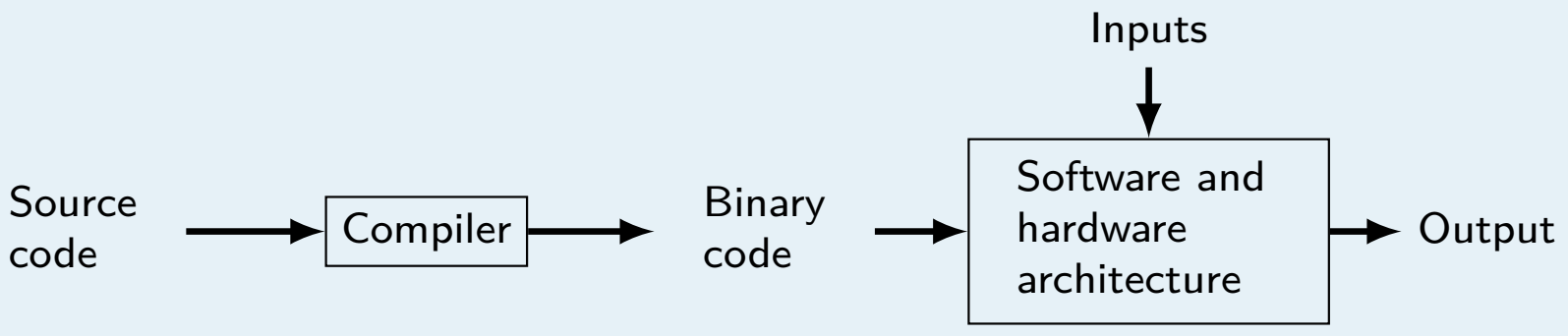
Definition

An *interpreter* is a software whose task is to analyze, translate and execute programs written in a computer language.

A python program is interpreted :



A c program is compiled before been executed :



First differences

- With an interpreted language :
  - the same source code will be able to work directly on any computer computer, we talk about cross-platform..
  - development errors are discovered at runtime execution,
- With a compiled language :
  - you will have to compile the source code for each architecture„
  - the program is directly executed on the computer, therefore faster.

How can Python be cross-platform ?

Answer :

Simply because there is an interpreter for each architecture architecture, i.e. one for Windows, one for Linux and one for MacOS.

Remark

There are several interpreters for each architecture : CPython, PyPy, IronPython and Jython.

In practice, when we use the command `python` under GNU/Linux, we will in fact execute CPython (which is the reference implementation : `https://www.python.org/`).

Python is multiparadigm

Definition

Programming *paradigms* are a way to classify programming languages based on their features.
Languages can be classified into multiple paradigms.

Exemples de paradigme

Imperative or procedural , in which the programmer instructs the machine how to change its state, (e.g. c)

Object-oriented programming which groups instructions with the part of the state they operate on, (e.g. c++)

declarative programming in which the programmer merely declares properties of the desired result, but not how to compute it

- *descriptive* : reduced expressiveness, description of data structures (e.g. html or LaTeX)
- *fonctional* : in which the desired result is declared as the value of a series of function applications (e.g. haskell)
- *logic*, in which the desired result is declared as the answer to a question about a system of facts and rules, (e.g. Prolog)

Comparison : bash vs Python

- Python is readable,
- Bash is closer to the GNU/Linux OS :
  - Specific commands,
  - Redirection mechanisms.

Example of Python code

```
#Addition
a=10
b=20
c=a+b

#Condition
if c == 30 :
    print("equal")
```

Type inference refers to the automatic detection of the type of an expression in a formal language.

We do not declare the type of the variables

```
1  # Int declaration
2  a=10
3  print(type(a))
4
5  # String declaration
6  b = "python is so cool"
7  print(b)
8
9  print(type(300))
```

```
nico@pc:~/$ python3 test.py
<class 'int'>
python is so cool
<class 'int'>
```

Indentation is part of the Python language

```
1 s = "Un petit texte"
2 cpt = 0
3 for c in s:
4   if c == "e":
5     cpt = cpt + 1
6 print("I find the caracter 'e' ",cpt," times.")
```

```
nico@pc:~/$ python3 example2.py
I find the caracter 'e' 3 times.
```

Assignment creates references, not copies

```
1 L = [1,2,3]
2 M = ['X',L,'Y']
3 print(M)
4 L[1] = 0
5 print(M)
```

```
nico@pc:~/$ python3 example4.py
['X', [1, 2, 3], 'Y']
['X', [1, 0, 3], 'Y']
```

Assignment creates references

```
1 n = 300
2 m = n
3 m = 400
4 n = "foo"
```

Memory state

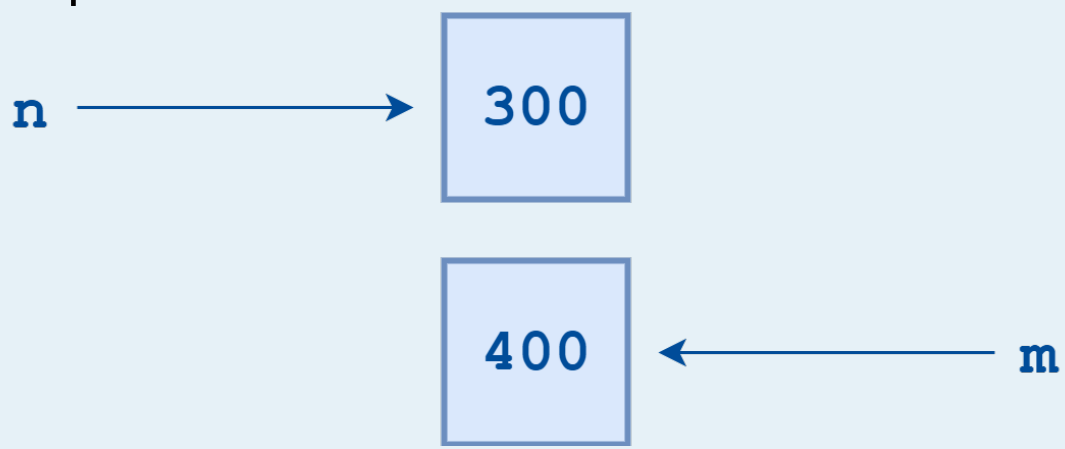
Step 2

Assignment creates references

```
1 n = 300
2 m = n
3 m = 400
4 n = "foo"
```

Memory state

Step 3

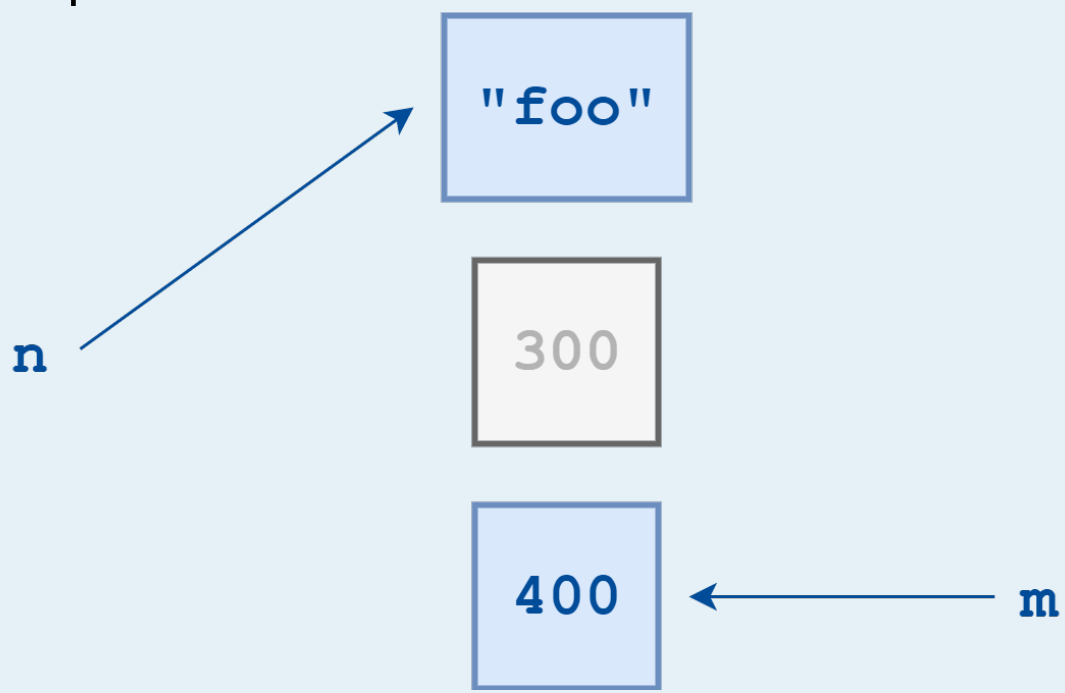


Assignment creates references

```
1 n = 300
2 m = n
3 m = 400
4 n = "foo"
```

Memory state

Step 4



Let us suppose that we want to compute $x = \cos(\pi)$.
There are (at least) three ways : :

- Call to the functions of a via the import keyword :

```
import math
x = math.cos(math.pi)
```

- Import of the essential (more readable) :

```
from math import cos,pi
x = cos(pi)
```

- Import all the library :

```
from math import *
x = cos(pi)
```

Creation of functions

```
# Creation of a function :
def compteur(stop):
    i = 0
    while i < stop:
        print(i)
        i = i + 1
    print("fin")

# Call of a function :
compteur(3)
```

```
nico@pc:~/$ python3 example6.py
0
1
2
fin
```

User interaction

```
n=input("Enter a number\n")
print(type(n))
```

```
nico@pc:~/$ python3 example8.py
Enter a number
10
<class 'str'>
```

Reference card of Laurent Pointal

http://perso-laris.univ-angers.fr/~delanoue/polytech/gnu_linux_python/ref_card_python.pdf

Installation of Python packages

There are different tools to add Python packages : pip, conda, apt.

pip (Pip Installs Packages)

pip is a package manager used to install and manage packages written in Python.

```
nico@pc:~/$ pip install package-name
nico@pc:~/$ pip uninstall package-name
```

Combining Bash and Python

- Short Bash scripts for a specific subpart (close to low-level commands)
- Python scripts :
  - invocation of Bash scripts,
  - coupling with high level functionalities : database, gui, internet, ...