GNU/Linux

Découverte du langage Python
Scripts d'administration en Python

Nicolas Delanoue

Université d'Angers - Polytech Angers

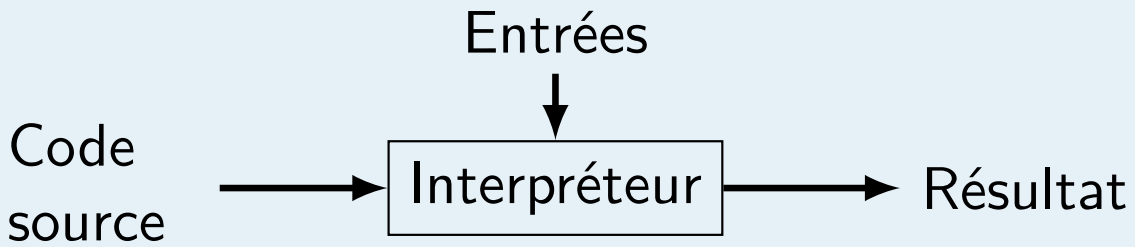
Quelques remarques sur Python

- Interprété,
- Multiplateforme,
- Multiparadigme (impératif, objet, fonctionnel, ...)
- Enorme communauté : plus de 50 000 librairies,
- Langage à usage général pour diverses applications :
  - développement web,
  - accès aux bases de données,
  - administration système,
  - GUI,
  - calculs scientifiques,
  - systèmes embarquées,...

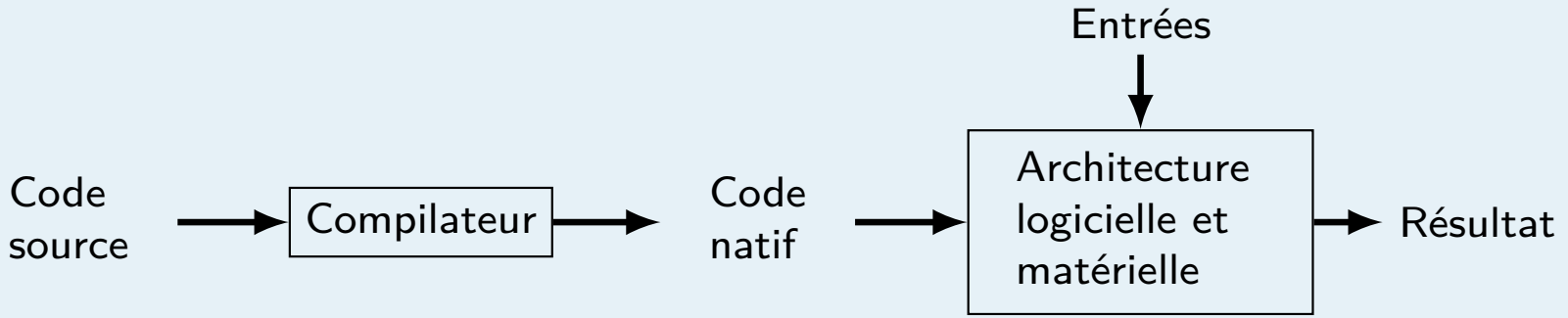
Définition

Un *interpréteur*, est un outil ayant pour tâche d'analyser, de traduire et d'exécuter les programmes écrits dans un langage informatique.

Un programme python est interprété :



Un programme c compilé avant d'être exécuté :



Premières différences

- Dans un langage interprété :
  - le même code source pourra marcher directement sur tout ordinateur, on parle de multiplateforme.
  - les erreurs de développement sont découvertes à l'exécution,
- Dans un langage compilé :
  - il faudra compiler le code source pour chaque architecture,
  - le programme est directement exécuté sur l'ordinateur, donc plus rapide.

Comment Python peut-il être multiplateforme ?

Réponse :

Simplement par qu'il existe un interpréteur pour chaque architecture, i.e un pour Windows, un pour Linux et un pour MacOS.

Pire

Il existe plusieurs interpréteurs pour chaque architecture : CPython, PyPy, IronPython et Jython.

En pratique, quand on utilisera la commande `python` sous GNU/Linux, on exécutera en fait CPython (qui est l'implémentation de référence : `https://www.python.org/`).

Python est multiparadigme

Définition

Un paradigme de programmation fournit (et détermine) la vue qu'a le développeur de l'exécution de son programme.

Exemples de paradigme

Programmation *impérative* , paradigme originel et très courant,

Programmation *orientée objet* définition et assemblage de briques logicielles appelées objets,

Programmation *déclarative* déclaration de données du problème, puis on demande au programme de le résoudre.

- *descriptive* : expressivité réduite, description de structures de données (e.g. html ou LaTeX)
- *fonctionnelle* : un programme est une fonction au sens mathématique (e.g. haskell)
- *logique*, déclaration des problèmes et des algorithmes sous forme de prédicats (e.g. Prolog)

Comparaison

- Python est lisible,
- Bash sera plus proche du système GNU/Linux :
  - Commandes spécifiques,
  - Mécanismes de redirection.

Exemple de code en Python

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

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

Algorithme d'inférence de types

On ne déclare pas le type des variables

```
1  # Déclaration d'un int
2  a=10
3  print(type(a))
4
5  # Déclaration d'un string
6  b = "python est trop cool"
7  print(b)
8
9  print(type(300))
```

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

L'indentation fait partie du langage Python

```
1 s = "Un petit texte"
2 cpt = 0
3 for c in s:
4   if c == "e":
5     cpt = cpt + 1
6 print("J'ai trouvé ",cpt,"fois le caractère 'e'.")
```

```
nico@pc:~/$ python3 example2.py
J'ai trouvé 3 fois le caractère 'e'.
```

L'affectation crée des références et non des 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']
```

L'affectation crée des références

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

Etat de la mémoire

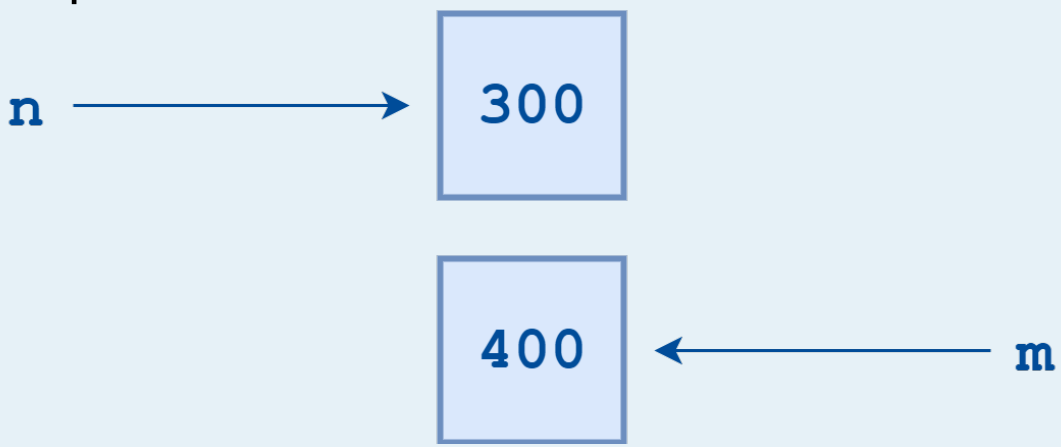
Etape 2

L'affectation crée des références

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

Etat de la mémoire

Etape 3

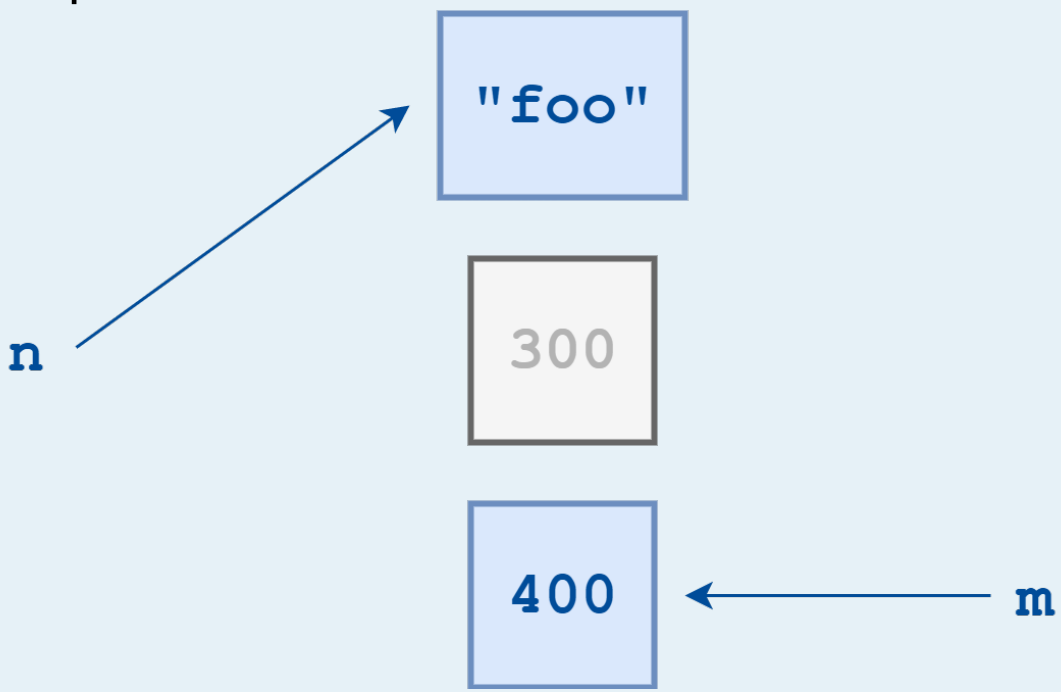


L'affectation crée des références

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

Etat de la mémoire

Etape 4



Supposons que l'on cherche à calculer $x = \cos(\pi)$.
Il existe (au moins) trois façons :

- Appel aux fonctions d'une via le mot clé import :

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

- Import de l'essentiel (plus lisible) :

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

- Import de tous :

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

Création de fonctions

```
# Création d'une fonction :
def compteur(stop):
    i = 0
    while i < stop:
        print(i)
        i = i + 1
    print("fin")

# Appel de la fonction :
compteur(3)
```

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

Interaction avec l'utilisateur

```
n=input("entrez un nombre\n")
print(type(n))
```

```
nico@pc:~/$ python3 example8.py
entrez un nombre
10
<class 'str'>
```

La reference card de Laurent Pointal

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

Installation de packages Python

Il existe différents outils de rajouter des packages Python : pip, conda, apt.

pip (Pip Installs Packages)

pip est un gestionnaire de paquets utilisé pour installer et gérer des paquets écrits en Python.

```
nico@pc:~/$ pip install nom-du-paquet
nico@pc:~/$ pip uninstall nom-du-paquet
```

Combinaison de Bash et Python

- Des scripts Bash courts pour une sous-partie spécifique (proche des commandes de bas niveau)
- Des scripts Python :
  - invocation de scripts Bash,
  - couplage avec des fonctionnalités de haut niveau : base de données, gui, internet, ...