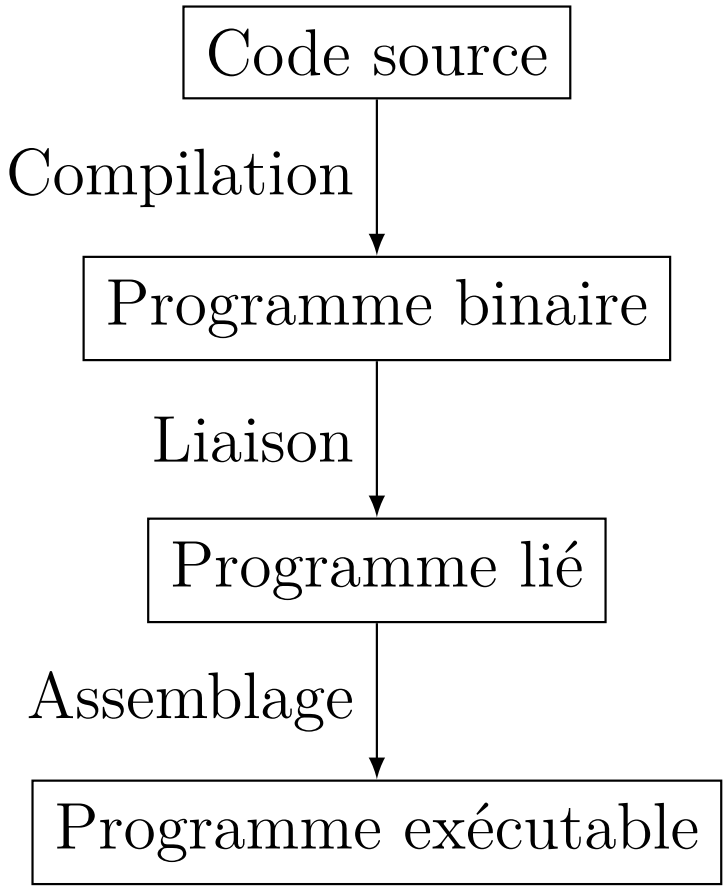
Programmer en python

Damien Nouvel

Langages informatiques

- Langages compilés et/ou **interprétés**
- Du **programme** à l'**exécution** :



⇒ La machine exécute le code **écrit par le développeur**

Ligne de commande

- Rappel de **commandes utiles** sous Unix :

```
[damien] mkdir dossier # Créer un dossier
[damien] cd dossier # Changer de dossier
[damien] ls # Lister les fichiers
[damien] touch fic.txt # Créer un fichier
[damien] less fic.txt # Voir le contenu d'un fichier
[damien] man less # Aide pour une commande (ici less)
[damien] cp fic.txt fic2.txt # Copie de fichier
[damien] mv fic.txt fic3.txt # Déplacer un fichier
[damien] rm fic2.txt # Effacer un fichier
[damien] chmod u+x prog # Rendre un fichier éxécutable
[damien] ./prog # Exécuter un programme
```

- **Exécution** d'un programme python :

```
[damien] python monfichier.py
[damien] ./monfichier.py # Si le fichier est exécutable
```

Méthode de développement

- Possibilité d'utiliser des **IDE** :
  - Surtout pour les entreprises
  - Très utile pour les gros projets avec de nombreux fichiers

  $\Rightarrow$ Pas nécessaire, mais pas interdit
- Méthode proposée :
  - Ouverture de fenêtres :
    - Editeur de textes léger (gedit, NotePad++, Sublime)
    - Ligne de commande (bash)

    $\Rightarrow$ Alt+Tab pour basculer de l'une à l'autre
- Développement du programme :
  - Ecriture de petites portions de code
  - Mise en commentaires avec le caractère dièse #
  - Limiter la sortie avec less : `python prog.py | less`

$\Rightarrow$ Ne pas perdre du temps !

Syntaxe

- Principes généraux :
  - Commande `print` pour les **affichages**
  - Deux points après les **tests** / **boucles**
  - **Tabulations** pour les blocks de code
- Langage faiblement typé :
  - **Chaînes de caractères** (`"Bonjour"`)
  - **Entiers** (`5`)
  - **Nombre flottants** (`2.5`)

  $\Rightarrow$ Pour concaténer un nombre, utiliser `str()`

```
print "Bonjour, le monde !"
x = 5*2.5
if x > 10:
  print "x est plus grand que 10"
for i in range(3):
  print i
```

Listes et dictionnaires

- **Listes** d'éléments (ordonnés) :
  - Initialisation : `lst = [1, 2, 3]`
  - Accès aux éléments : `lst[2]`
  - Ajout d'éléments : `lst += [4, 5]`
- **Dictionnaires** (table de hachage, paires clés/valeurs) :
  - Initialisation : `dic = {'age': 25, 'taille':170}`
  - Accès aux éléments : `dic['age']`
  - Ajout d'éléments : `dic['nom'] = 'Damien'`

```
etudiants = [{'nom': 'Roger', 'note': 15}]
etudiants += [{'nom': 'Julie', 'note': 13}]
for e in etudiants:
  print "L'étudiant", e['nom'], "a eu", e['note']
moyenne = 0.0
for e in etudiants:
  moyenne += e['note']
moyenne /= len(etudiants)
```

Fonctions

- Rappels sur les fonctions :
  - **Isolées** du reste du code
  - **Paramètres** en entrée
  - Sortie **retournée** par le mot-clé `return`
- Définition et appel d'une fonction
  - Définition simple : `def fonc(param1, param2):`
  - Valeurs par défaut : `def fonc(param1 = 1, param2 = 3):`
  - Appel : `x = fonc(3, 2)`

```
def compterLettre(chaine = "", lettre =" "):
  compte = 0
  for c in chaine:
    if c == lettre:
      compte += 1
  return compte
print compterLettre("Botte de foin", "o")
```

Quelques fonctions utiles

- Tests et boucles avec `in`

⇒ Fonctionne avec les listes, chaînes, boucles

```
if 4 in var:
   ...
for x in var:
   ...
```

- Pour les chaînes de caractères :
  - Concaténation avec `+`
  - Séparation avec `split(chaine, car)`
  - Réunion avec `car.join(lst)`
  - Conversion en entier avec `int(chaine)`
  - Suppression du saut de ligne `strip(chaine)`

Lecture de fichier ligne par ligne

- Lecture de fichier :
  - Import du module codecs : `import codecs`
  - Chemin : `f = codecs.open("fic.txt", "r", "utf-8")`
  - Modes (r/w) :
    `f = codecs.open("fic.txt", "rw", "utf-8")`
  - Lecture ligne par ligne : `for ligne in f:`

```
import codecs
fichier = codecs.open("80jours.txt", "r", "utf-8")
caracteres = 0
nbLignes = 1
for ligne in fichier:
  caracteresLigne = len(ligne)
  print "La ligne", nbLignes, "a", caracteresLigne, "caractères"
  caracteres += caracteresLigne
  nbLignes += 1
print "Il y a en moyenne", caracteres/nbLignes, "caractères"
```

Exercice

- Implémenter un **segmenteur** simple pour le français :
  - Pour chaque phrase :
    - Lire un fichier ligne par ligne
    - Se servir d'une variable "mot" qui accumule les caractères
    - Dès qu'un caractère est un séparateur, enregistrer le mot

  $\Rightarrow$ Créer un dictionnaire du nombre de mots dans le texte

Exercice (solution)

```
caractere = ''
mot = ''
dictMots = {} # Dictionnaire du nombre d'occurrences par mot
for ligne in codecs.open("80jours.txt", "r", "utf-8"):
  ligne += ' '
  ligneMots = []
  for caractere in ligne:
    # Caractere non-separateur : ajout au mot courant
    if caractere not in [' ', ',', '.', '!', '?']:
      mot += caractere
    # Separateur : enregistrement et passage au suivant
    else:
      if len(mot):
        ligneMots += [mot]
        dictMots[mot] = dictMots.get(mot, 0) + 1
      mot = ''
  print ligneMots
```

Compléments pour python

- Entête pour faire travailler python en UTF8 :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import codecs, sys
sys.stdin = codecs.getreader('utf8')(sys.stdin)
sys.stdout = codecs.getwriter('utf8')(sys.stdout)
```

- Diverses choses utiles :
  - Utiliser `range()` pour les boucles

    ```
    for i in range(10):
    ```

  - Récupérer les arguments du programme avec la liste `sys.argv`
  - Lire l'entrée standard avec `sys.stdin`
  - Utiliser la fonction `sorted()` pour les tris :

    ```
    listeTriee = sorted(liste)
    listeTriee = sorted(liste, reverse = True)
    cleTriees = sorted(dict.items(), key=lambda x:x[1])
    ```