Logiciels Scientifiques

Licence Mathématiques
Licence MIASHS
Mineur Mathématiques

Cours 00

Andrzej Stos

Validation

- Épreuve CC 1h30 : TP noté (en binôme), 30%
- Devoir maison : 3-4 exercices individuels 20%
- Épreuve CC 1h30 : épreuve individuelle type TP 50%

Python

Quelques mot à propos du langage Python :

- un langage de programmation *généraliste* : une énorme collection de bibliothèques (= *modules* ou *packages*)
- un langage *interprété*
- syntaxe simple et lisible
- recommandé pour l'apprentissage de l'algorithmique au lycée
- bien adapté au domaine d'analyse de données et de visualisation (bibliothèques !)
- parmi les plus populaires langages de programmation, Python est le "tissu" de plusieurs projets, logiciels et infrastructures informatiques...
- "A quoi ça sert, les langages de programmation ?"
  "A dire à l'ordinateur ce qu'il doit faire..." (réponse courante)
  "A décrire et communiquer ses idées – et les destinataires, ce sont surtout des humains, et non des ordinateurs"
  (Guido van Rossum, créateur de Python)

Méthode de travail

- Dans cette UE (basée sur les TP), il s'agit de *votre* travail. Vraiment. J'insiste. Explications.
- Vous êtes évalués sur ce que *vous produisez et comprenez* durant le semestre
  (et non pas sur ce quelqu'un présente au tableau).
- En particulier, on ne prévoit pas de corrigés dans le cours ou sur l'ENT.
- Méthode de travail : faire tous les exercices en TP
  mais pas qu'en TP si besoin
  - en équipe (binôme),
  - en échangeant avec vos camarades (pas forcément qu'en TP !)
  - surtout avec de l'aide du professeur si besoin !
  - *Vérifiez* votre solution avec votre professeur ! (pas de corrigés sur l'ENT)
- En revanche, le contrôle se fera à livre ouvert.
  Seront autorisés : vos notes manuscrites, vos programmes de TP, vos feuilles de TP, l'aide mémoire.
  Seront interdits : ENT, internet, portables, calculettes.

Méthode de travail

- Planning : 8 feuilles de TP pour 16 séances (24h)
  - TP 1 : 1 séance
  - TP 2, 3, ..., 7 : 2 séances chacune
  - TP 8 : 3 séances
- Sur l'ENT / emploi du temps, on peut déterminer l'avancement actuel (numéro de la séance pour son groupe de TP).
- Il est important de travailler régulièrement et respecter ce planning.

Installation

- Python est un logiciel libre
- On utilisera la version 3 du langage

  $\hookrightarrow$ plus de support pour la version 2 après 1er janvier 2020
- On utilise l'éditeur de texte (ang. *IDE*) **Pyzo** :
  CAPES, Agrégation
- Pour installer logiciel**s** sur sa machine : suivre les instructions sur la page web de Pyzo `www.pyzo.org` ("Quickstart")
  - Étape 1 : **installer l'éditeur Pyzo IDE**
  - Étape 2 : **installer Python Anaconda** (ou Miniconda ou "regular Python")
  - Si vous optez Pour Anaconda, vous avez tout ce qu'il faut (Python et les modules, utiles et non utiles)
  - Si vous optez pour Miniconda ou "regular Python", **n'oubliez pas** d'installer les modules : `numpy`, `matplotlib` et `sympy` à partir de console Pyzo (voir la page web "Step 4")
  - Au premier lancement, Pyzo détecte Anaconda et demande de l'utiliser (acceptez)

Remarques pratiques

- Sauvegardez vos programmes sur le disque `M` (pas d'autre !), dans le dossier `LogSci`.
- Sauvegarder votre travail **souvent** : `CTRL-S` est votre ami !
- Pensez à créer un script par exercice, par exemple : `tp2ex4.py`
- Calculettes sont interdites en TP (et aux contrôles) !

Rappels et compléments de TP 1 : variables et types

Qu'est-ce que c'est qu'une variable ?
Exemples :

- `a = 10`
  Ceci n'est pas une égalité au sens mathématique ni une comparaison !
  C'est une *affectation* d'une valeur à une variable !
  On lit : "a reçoit 10".
  ↪ a est de type entier (`int`)
- `b = 2.0`
  ↪ nombre à virgule flottant (`float`)
- `s = "Hello"`
  `t = 'world'`
  ↪ `s` et `t` sont des chaînes de caractères (`string`)
  "L'apostrophe est permise entre guillemets"

Rappels et compléments : variables et types

En Python, un bloc (un groupement de commandes) se fait à l'aide d'indentation (décalage)

- ```
  for n in range(2,5):
      print(n, n**2, "carre dans la boucle")
  print("ceci n'est pas dans la boucle")
  ```
- `range(DE, A, PAS)` : la seconde borne est exclue !
  `range(3, 14, 5)` $\hookrightarrow$ `3, 8, 13`
  `range(3, 13, 5)` $\hookrightarrow$ `3, 8`
  `range(4)` est un raccourci pour `range(0, 4)`
- La boucle `for` accepte :

une liste arbitraire	une chaîne de caractères
`for i in [1,3,"mardi"]:` `  print(i)`	`for i in "octobre":` `  print(i)`
  ```

Rappels et compléments : Tests

- tests `if` :
  `a == b` test d'égalité
  `a != b` test de non égalité
  `and`, `or`, `not` : opérations logiques

  Structure générale :

```
if a >= b:
    print("a est plus grand ou égal")
else:
    print("b est strictement plus grand")
```

  Remarque : `else` n'est pas obligatoire.

Rappels et compléments : Fonctions

```
def affine(x):
    y = 2*x + 5
    return y
```

Attention : `return` c'est different de `print` (!)

```
def muffin(x):
    y = 2*x + 5
    print(y)
```

```
print(affine(2) + affine(3))
```
↪ 20

```
print(muffin(2) + muffin(3))
```
↪ erreur

Règle générale : aucun `print` dans les fonctions ! !

(sauf pour débogguer dans la phase de mise au point)

Style de programmation (!)

- Espaces autour des opérateurs (+, =, ==)
- Un espace après la virgule
- *Pas* d'espace entre une fonction et son argument
- Une ligne vide après une procédure

Oui	Non
`x = 10`	`x=10`
`y = x + 1`	`y=x+1`
`y = 2*x + 1`	`y=2*x+1`
`y = 2 * x + 1`	`y=2*x+1`
`x += 1`	`x+=1`
`if n < 10:`	`if n<10 :`
`f(x)`	`f (x)`

The Zen of Python : `import this`

Beautiful is better than ugly.

Simple is better than complex.

Readability counts.

concrètement -1p au contrôle

Listes

Liste est une suite de termes entre crochets :

`L = [2, 5.0, "mardi", "jeudi"]`

`L[0]` vaut 2 (l'indice 0 désigne le premier element)

`L[1:3]` vaut `[5.0, "mardi"]`

On utilisera des listes pour stocker des résultats de calculs en série.

Exemple :

```
L = [ ]
for i in range(10):
    L.append(i**2)
```

Interdit :

```
L = [ ]
for i in range(10):
    L[i] = i**2
```

Attention ! Dupliquer une liste :

```
L = [1,5,10]
K = L
K[0] = 23
print(K)
  ↪ [23, 5, 10]
print(L)
  ↪ [23, 5, 10]
```

```
L = [1,5,10]
M = L[:]
M[0] = 23
print(M)
  ↪ [23, 5, 10]
print(L)
  ↪ [1, 5, 10]
```

Arithmétique des ordinateurs

Pour la machine $(\sqrt{10})^2 \neq 10$...

Développement décimal :
$\sqrt{10} \approx 3.16227766016837933199889354443...$

Arrondi machine : `sqrt(10)` $\hookrightarrow 3.162277660168379 \neq \sqrt{10}$

donc `3.162277660168379 ** 2` $\neq 10$

Un truc qui cloche...

3.16227766016837933199889354443...
3.162277660168379**5**

Explication ? ? ?

Arithmétique des ordinateurs

Pour la machine $2 * 0.25 + 0.5 = 1$ et $3 * 0.3 + 0.1 \neq 1$

Explication : ordinateur fait ses calculs en base 2.

Dans cette base, 0.25 possède un développement fini :

$\frac{1}{4} = 2^{-2} = 0.01_2$.

Or 0.1 et 0.3 possèdent un développement infini,

par exemple $0.1 = 0.0001100110011001 \ldots_2$

un autre exemple bien connu : $\frac{1}{3} = 0.333333333 \ldots_{10}$

Arithmétique des ordinateurs - les floats

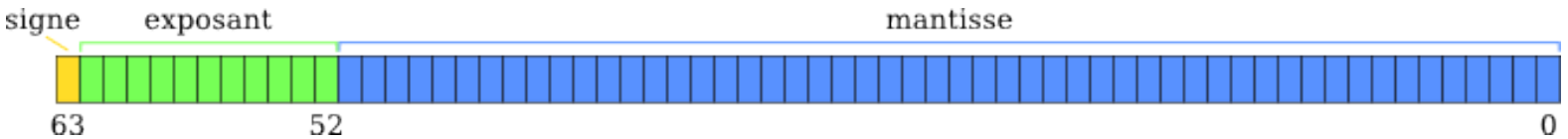
Sur la machine, les nombres `float` sont représentés selon la norme universelle IEEE-754.

Il s'agit d'une représentation exponentielle :

$$\pm \left(1 + \frac{M}{2^{52}}\right) 2^{E-1023}$$

où M est appelé *mantisse* et E l'*exposant*.

Les registres du processeur (récent) ont 64 bits :



- 1 bit pour le signe
- 11 bits pour l'exposant E
- 52 bits pour la mantisse M

Arithmétique des ordinateurs - les flottants

Conséquences intéressantes :

- On a environ 15 chiffres de précision ($\log_{10}(2^{52}) \approx 15.65$)
- Les flottants ne sont pas distribués "régulièrement" :
- Il y a 2^{52} flottants dans l'intervalle $[1, 2]$ et autant dans [2,4], [4,8], [8,16] etc. Par conséquent :
  `2.0**53` $\hookrightarrow$ `9007199254740992.0`
  `2.0**53 + 1` $\hookrightarrow$ `9007199254740992.0`
  Avec les entiers : `2**53 + 1` $\hookrightarrow$ `9007199254740993`
- L'approximation bizarre de $\sqrt{10}$ dans l'exercice 4.1 était en fait la *meilleure possible* selon la norme
- Le plus grand nombre positif représentable est $1.7976931348623157 \times 10^{308}$
- Pourquoi donc utiliser les flottants ?
  $\hookrightarrow$ un bon compromis entre la précision et l'étendu (l'intervalle couvert)

Écriture en base 2

- Conversion de la base 2 vers la base 10 (facile !).
  Supposons que une liste `L` contient les chiffres d'écriture binaire d'un entier dans l'ordre *inverse*, c.à.d. que `L[0]` correspond au chiffre des unités.
  Algorithme : (dans une boucle) pour la `i`-ième chiffre `c`, rajouter $c \times 2^i$ à la somme.
- Conversion de la base 10 vers la base 2
  Entrée : un entier `m` (positif) à convertir
  Initialiser une liste `L`
  tant que `m` non nul, faire :
  rajouter à `L` le reste de la division de `m` par 2
  diviser `m` par 2
  fin tantque

  Remarque : la liste obtenue est dans l'ordre *inverse*.
  Pour inverser une liste, on peut utiliser l'*idiome* suivant :
  `L = L[: :-1]`.

Chaînes de caractères (`string`)

```
c = "ceci est une chaine"
s = 'cela aussi'
```

"l'apostrophe est permis entre guillemets"

Remarque : on évitera d'utiliser les lettres avec des accents é à etc..

`s[0]` $\hookrightarrow$ "c"

`s[1:4]` $\hookrightarrow$ "ela"

s[0] = "a" $\hookrightarrow$ erreur

Remarque : On dit que les chaînes de caractères sont *immuables*

Concaténation de deux chaînes : `b = c + s`

Rajouter un caractère à une chaîne : `s += "x"`

Exemple fondamental - une boucle `for` sur une chaîne :

```
for lettre in "ma chaine":
    print(ord(lettre))
```

Chaînes de caractères (`string`)

Codage : code ASCII
`ord("a")` $\hookrightarrow$ 97
`ord("b")` $\hookrightarrow$ 98...
`chr(97)` $\hookrightarrow$ `"a"`

Appliquer une fonction `f` à tous les éléments d'une liste `L` :
`list(map(f, L))`
$\hookrightarrow$ utile pour le codage

`map` fonctionne aussi avec les chaînes :
`list(map(f, "ma chaine"))`

Remarque : `map` est un *générateur*
(donc `list` est souvent nécessaire)