Rappels de première année

ψ

stephane@gonnord.org - http://blog.psi945.fr

Lundi 5 et 12 Septembre 2016

Plan

1. Coder
   - Logique globale
   - Éléments de programmation
   - Appels de fonction
2. Tester, évaluer
   - Faire des tests
   - Complexité théorique
   - Complexité empirique
3. Et aussi
   - Dans la machine
   - Entrées, sorties
   - S'entraîner régulièrement
   - Les bibliothèques

Avant toutes choses

- Des dossiers et sous-dossiers :
  - bien organisés ;
  - correctement nommés !

Avant toutes choses

- Des dossiers et sous-dossiers :
  - bien organisés ;
  - correctement nommés !
- Idle, Spyder, Pyzo : whatever works !
  - éditeur ;
  - interpréteur ;
  - savoir utiliser l'un et/avec l'autre ;
  - tester, rapatrier les résultats dans le .py, commenter ;
  - accès à l'aide.

Avant toutes choses

- Des dossiers et sous-dossiers :
  - bien organisés ;
  - correctement nommés !
- Idle, Spyder, Pyzo : whatever works !
  - éditeur ;
  - interpréteur ;
  - savoir utiliser l'un et/avec l'autre ;
  - tester, rapatrier les résultats dans le .py, commenter ;
  - accès à l'aide.
- Syntaxe claire, propre, lisible (PEPS...)

Sommaire

Vue d'ensemble

- Comprendre :
  - la démarche globale (épreuve de concours...) ;
  - le fonctionnement d'un algorithme ;
  - un programme donné ;
  - le rôle des différentes variables.

Vue d'ensemble

- Comprendre :
  - la démarche globale (épreuve de concours...) ;
  - le fonctionnement d'un algorithme ;
  - un programme donné ;
  - le rôle des différentes variables.
- Paramètres et résultat d'un programme :

```
def foo( ... ):
    ...
    return ...
```

Vue d'ensemble

- Comprendre :
  - la démarche globale (épreuve de concours...) ;
  - le fonctionnement d'un algorithme ;
  - un programme donné ;
  - le rôle des différentes variables.
- Paramètres et résultat d'un programme :

```
def foo( ... ):
    ...
    return ...
```

- Exemples : fonction qui détermine le maximum d'une liste ; qui en échange deux éléments ; qui la trie...

Sommaire

Dans tous les langages...

(ou presque) on trouvera :

- Des variables :
  - rôle ;
  - évolution en cours d'exécution.

Dans tous les langages...

(ou presque) on trouvera :

- Des variables :
  - rôle ;
  - évolution en cours d'exécution.
- Des boucles :
  - tant de fois ;
  - pour *i* allant de *telle valeur* à *telle valeur* ;
  - tant que *telle condition* est remplie.

Dans tous les langages...

(ou presque) on trouvera :

- Des variables :
  - rôle ;
  - évolution en cours d'exécution.
- Des boucles :
  - tant de fois ;
  - pour *i* allant de *telle valeur* à *telle valeur* ;
  - tant que *telle condition* est remplie.
- Des tests :
  - bien comprendre la logique booléenne ;
  - distinguer == et =
  - `else`... ou pas !

Dans tous les langages...

(ou presque) on trouvera :

- Des variables :
  - rôle ;
  - évolution en cours d'exécution.
- Des boucles :
  - tant de fois ;
  - pour *i* allant de *telle valeur* à *telle valeur* ;
  - tant que *telle condition* est remplie.
- Des tests :
  - bien comprendre la logique booléenne ;
  - distinguer == et =
  - `else`... ou pas !
- Des appels de fonction.

Types de données

- Scalaires :
  - booléens (`True, False`) ;
  - entiers ;
  - flottants (IEEE 754).

Types de données

- Scalaires :
  - booléens (`True, False`) ;
  - entiers ;
  - flottants (IEEE 754).
- Chaînes de caractères (`string`) : `foo = "blabla\n"`

Types de données

- Scalaires :
  - booléens (`True, False`) ;
  - entiers ;
  - flottants (IEEE 754).
- Chaînes de caractères (`string`) : `foo = "blabla\n"`
- (tuples : `(a, b) = (5, 17)`)

Types de données

- Scalaires :
  - booléens (`True, False`) ;
  - entiers ;
  - flottants (IEEE 754).
- Chaînes de caractères (`string`) : `foo = "blabla\n"`
- (tuples : `(a, b) = (5, 17)`)
- Tableaux :
  - « listes » `t = [10, 12, 42]` ...
  - vs « vecteurs » `t = numpy.array( [10, 12, 42] )`

  Création, accès, slicing...

Sommaire

Appels de fonction

- `<troll>` Un programme ne doit jamais dépasser 5 lignes ! `</>`

Appels de fonction

- `<troll>` Un programme ne doit jamais dépasser 5 lignes ! `</>`
- Importance du découpage (exemple du tri).
- Cahier des charges.

Appels de fonction

- `<troll>` Un programme ne doit jamais dépasser 5 lignes ! `</>`
- Importance du découpage (exemple du tri).
- Cahier des charges.
- Visibilité des variables ; paramètres ; mutables ou non.

Appels de fonction

- `<troll>` Un programme ne doit jamais dépasser 5 lignes ! `</>`
- Importance du découpage (exemple du tri).
- Cahier des charges.
- Visibilité des variables ; paramètres ; mutables ou non.
- Évaluation (un peu) simplifiée de la complexité.
- Récursif vs. non récursif.

Spécifier

- Quoi ?
  - Les entrées, les sorties.
  - Les effets de bord autorisés, demandés, attendus.
  - Les cas à la frontière du cahier des charges.

Spécifier

- Quoi ?
  - Les entrées, les sorties.
  - Les effets de bord autorisés, demandés, attendus.
  - Les cas à la frontière du cahier des charges.
- Qui ?

Spécifier

- Quoi ?
  - Les entrées, les sorties.
  - Les effets de bord autorisés, demandés, attendus.
  - Les cas à la frontière du cahier des charges.
- Qui ? Un peu le donneur d'ordre... et celui qui programme !

Spécifier

- Quoi ?
  - Les entrées, les sorties.
  - Les effets de bord autorisés, demandés, attendus.
  - Les cas à la frontière du cahier des charges.
- Qui ? Un peu le donneur d'ordre... et celui qui programme !
- Exemple : écrire un programme déterminant le deuxième plus gros élément d'une liste.

Sommaire

Faire des tests, débuguer

- « La définition de la fonction est acceptée/compile » : insuffisant !

Faire des tests, débuguer

- « La définition de la fonction est acceptée/compile » : insuffisant !
- Tester sur des cas favorables... et d'autres défavorables... et d'autres incorrects !

Faire des tests, débuguer

- « La définition de la fonction est acceptée/compile » : insuffisant !
- Tester sur des cas favorables... et d'autres défavorables... et d'autres incorrects !
- « Tracer » avant et/ou après l'exécution.
- Débug *old school* : `print( maximum_courant )` !!!
- Exemples de tests sur...
  - `maximum_liste`
  - `indice_maximum_liste`
  - `trier_liste`

Sommaire

Complexité théorique

- Qu'est-ce qu'un algorithme rapide ? lent ?

$$\ln N \ll \sqrt{N} \ll N \ll N^2 \ll 2^N$$

Complexité théorique

- Qu'est-ce qu'un algorithme rapide ? lent ?

$$\ln N \ll \sqrt{N} \ll N \ll N^2 \ll 2^N$$

- Règle du milliard.

Complexité théorique

- Qu'est-ce qu'un algorithme rapide ? lent ?

$$\ln N \ll \sqrt{N} \ll N \ll N^2 \ll 2^N$$

- Règle du milliard.

N	10^2	10^4	10^6	10^8
$\sqrt{N}$	10	10^2	10^3	10^4
$N\sqrt{N}$	10^3	10^6	10^9	10^{12}
$N\ln(N) \simeq$	$4,6 \times 10^2$	$9,2 \times 10^4$	$1,4 \times 10^7$	$1,8 \times 10^9$
N^2	10^4	10^8	10^{12}	10^{16}
N^3	10^6	10^{12}	10^{18}	10^{24}
2^N	$\simeq 10^{30}$			

Complexité théorique

- Qu'est-ce qu'un algorithme rapide ? lent ?

$$\ln N \ll \sqrt{N} \ll N \ll N^2 \ll 2^N$$

- Règle du milliard.

N	10^2	10^4	10^6	10^8
$\sqrt{N}$	10	10^2	10^3	10^4
$N\sqrt{N}$	10^3	10^6	10^9	10^{12}
$N\ln(N) \simeq$	$4,6 \times 10^2$	$9,2 \times 10^4$	$1,4 \times 10^7$	$1,8 \times 10^9$
N^2	10^4	10^8	10^{12}	10^{16}
N^3	10^6	10^{12}	10^{18}	10^{24}
2^N	$\simeq 10^{30}$			

- Complexités dans le cas de boucles simples, imbriquées ou successives.

Complexité théorique

- Qu'est-ce qu'un algorithme rapide ? lent ?

$$\ln N \ll \sqrt{N} \ll N \ll N^2 \ll 2^N$$

- Règle du milliard.

N	10^2	10^4	10^6	10^8
$\sqrt{N}$	10	10^2	10^3	10^4
$N\sqrt{N}$	10^3	10^6	10^9	10^{12}
$N\ln(N) \simeq$	$4,6 \times 10^2$	$9,2 \times 10^4$	$1,4 \times 10^7$	$1,8 \times 10^9$
N^2	10^4	10^8	10^{12}	10^{16}
N^3	10^6	10^{12}	10^{18}	10^{24}
2^N	$\simeq 10^{30}$			

- Complexités dans le cas de boucles simples, imbriquées ou successives.
- Complexité spatiale vs. temporelle.

Complexité théorique

- Qu'est-ce qu'un algorithme rapide ? lent ?

$$\ln N \ll \sqrt{N} \ll N \ll N^2 \ll 2^N$$

- Règle du milliard.

N	10^2	10^4	10^6	10^8
$\sqrt{N}$	10	10^2	10^3	10^4
$N\sqrt{N}$	10^3	10^6	10^9	10^{12}
$N\ln(N) \simeq$	$4,6 \times 10^2$	$9,2 \times 10^4$	$1,4 \times 10^7$	$1,8 \times 10^9$
N^2	10^4	10^8	10^{12}	10^{16}
N^3	10^6	10^{12}	10^{18}	10^{24}
2^N	$\simeq 10^{30}$			

- Complexités dans le cas de boucles simples, imbriquées ou successives.
- Complexité spatiale vs. temporelle.
- IRL : cache, transferts de données.

Sommaire

Complexité empirique

- Complexités typiques (exemples ?) :

$$C(n) = Kn^{\alpha}$$

Complexité empirique

- Complexités typiques (exemples ?) :

$$C(n) = Kn^{\alpha}$$

- Évaluations successives :

$$C(10^2), C(10^3), C(10^4)...$$

Complexité empirique

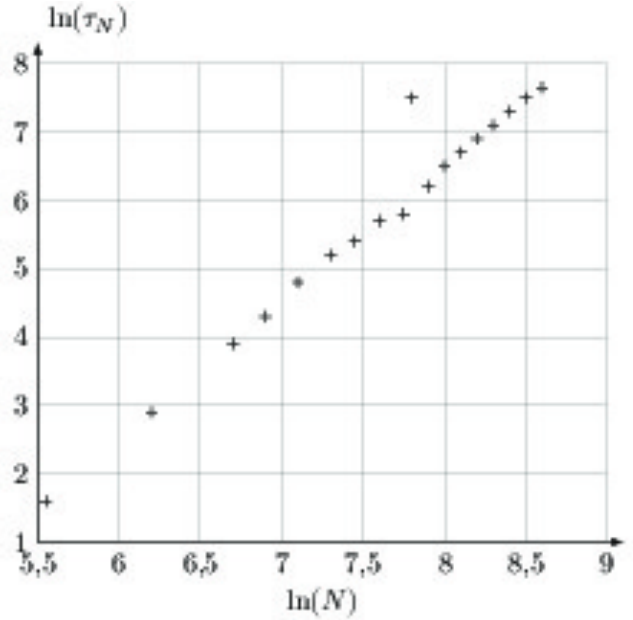
- Complexités typiques (exemples ?) :

$$C(n) = Kn^{\alpha}$$

- Évaluations successives :

$$C(10^2), C(10^3), C(10^4)...$$

- Représenter $\ln(C(n))$ en fonction de $\ln(n)$.



- Artefacts, effets de seuils.

Sommaire

Dans la machine

- Entiers, flottants : des 0 et des 1... mais :
  - combien ?
  - comment ?

Dans la machine

- Entiers, flottants : des 0 et des 1... mais :
  - combien ?
  - comment ?
- Listes vs. `numpy.array`.

Dans la machine

- Entiers, flottants : des 0 et des 1... mais :
  - combien ?
  - comment ?
- Listes vs. `numpy.array`.
- Fichiers (textes, images) : bits, octets, etc. Différents ordres de grandeur ?

Sommaire

Entrées, sorties

D'où viennent les données traitées par nos programmes ?

- Des programmes eux-mêmes !
- De l'interpréteur.

Entrées, sorties

D'où viennent les données traitées par nos programmes ?

- Des programmes eux-mêmes !
- De l'interpréteur.
- Du clavier via `input` :
  - années 80 ;
  - autres langages que Python ;
  - Python par des gens qui programment comme dans d'autres langages ou comme il y a 30 ans ;
  - sujets de concours...

Entrées, sorties

D'où viennent les données traitées par nos programmes ?

- Des programmes eux-mêmes !
- De l'interpréteur.
- Du clavier via `input` :
  - années 80 ;
  - autres langages que Python ;
  - Python par des gens qui programment comme dans d'autres langages ou comme il y a 30 ans ;
  - sujets de concours...
- Fichiers texte (`.txt`, `.csv`).
- Bases de données.

Sommaire

S'entraîner régulièrement

- Pratiquer les `numpy.array` :
  - création, slicing,
  - utilisation dans `linalg, scipy`

S'entraîner régulièrement

- Pratiquer les `numpy.array` :
  - création, slicing,
  - utilisation dans `linalg, scipy`
- Mais aussi (un peu) `matplotlib`.

S'entraîner régulièrement

- Pratiquer les `numpy.array` :
  - création, slicing,
  - utilisation dans `linalg, scipy`
- Mais aussi (un peu) `matplotlib`.
- Savoir récupérer des données dans des fichiers... mais aussi au clavier (...) !

S'entraîner régulièrement

- Pratiquer les `numpy.array` :
  - création, slicing,
  - utilisation dans `linalg, scipy`
- Mais aussi (un peu) `matplotlib`.
- Savoir récupérer des données dans des fichiers... mais aussi au clavier (...) !
- Penser « bases de données ». Imaginer le type de bases et de requêtes pour les questions suivantes :
  - Quelles sont les villes de plus de 100000 habitants ?

S'entraîner régulièrement

- Pratiquer les `numpy.array` :
  - création, slicing,
  - utilisation dans `linalg, scipy`
- Mais aussi (un peu) `matplotlib`.
- Savoir récupérer des données dans des fichiers... mais aussi au clavier (...) !
- Penser « bases de données ». Imaginer le type de bases et de requêtes pour les questions suivantes :
  - Quelles sont les villes de plus de 100000 habitants ?
  - Qui a gagné le plus de sets contre des Suisses entre 2000 et 2010 lors de tournois du grand chelem ?
  - Quels clients ont acheté en 2016 au moins un CD de Céline Dion et un CD de ZZ Top ?

Sommaire

Utiliser des bibliothèques...

- `from numpy import *`
- `from numpy import array, cos`

Utiliser des bibliothèques...

- `from numpy import *`
- `from numpy import array, cos`
- `import numpy` puis `numpy.array(...)`

Utiliser des bibliothèques...

- `from numpy import *`
- `from numpy import array, cos`
- `import numpy` puis `numpy.array(...)`
- `import numpy as np` puis `np.array(...)`

Utiliser des bibliothèques...

- `from numpy import *`
- `from numpy import array, cos`
- `import numpy` puis `numpy.array(...)`
- `import numpy as np` puis `np.array(...)`
- `import numpy.linalg as lin` puis `lin.det(...)`

C’est fini !

Merci de votre attention