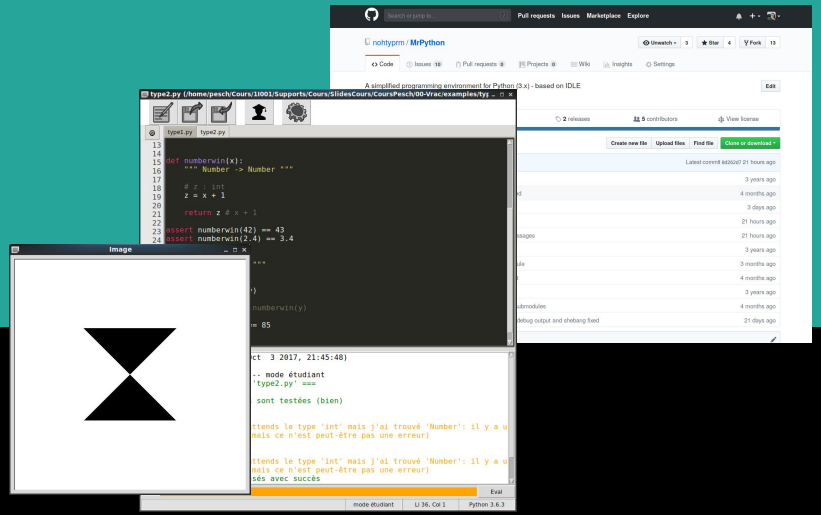


SORBONNE UNIVERSITÉ

PyParis

Python @ Sorbonne Université

Anatomie d'un cours d'introduction à l'Informatique

Présentation ...

Frédéric Peschanski

Frederic.Peschanski@lip6.fr

- **Maître de Conférences** en informatique à Sorbonne Université
  - enseignement de la programmation (Python en L1, Compilation en L3, Prog. fonctionnelle en M1) et “groupe de recherche” (en M2)

- **Chercheur** au LIP6
  - informatique “plutôt” théorique : automates, structures combinatoires, algorithmique (génération aléatoire), etc.

“Programming geek”

et adepte du logiciel libre...

- Depuis Linux 0.99pl12 (1994)
- Repo’s : https://github.com/fredokun
  - Langages favoris : Lisp (CL, Clojure), Haskell et Python bien sûr !

- Quelques projets **Python** :
  - MrPython : https://github.com/nohtyprm/MrPython
  - Tikz editor : https://github.com/fredokun/TikZ-Editor

Plan de la présentation

1. **Contexte** : cours d'introduction à l'université

2. **Survol du cours**
   "Eléments de Programmation 1"

3. **"Behind the scene"**
   Du cours magistral à l'examen final
   - Le point de vue étudiant
   - Le point de vue enseignant

1. Le Contexte

Cours d'introduction à l'informatique à l'université
(pour les étudiants en science et en ingénierie)

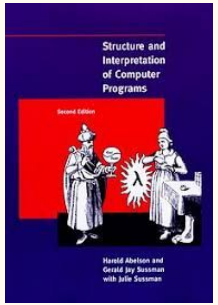
Approches variées, par exemple :

- **Approches "langage"** (le cas le plus fréquent)
  - Le cours est dirigé par la syntaxe du langage (variables, affectations, structures de contrôles, etc...)
  - Langages variés : Scheme, C, Java, Python, js, etc.
  - Exemple : Python @ Cornell CS1110

- **Approches "tour d'horizon"**
  - Introduction générale, la programmation n'étant qu'un aspect parmi d'autres (informatique & société, etc.)
  - Exemple : Scratch @ Harvard CS50

- **Approches "projet"**
  - Conduite d'un projet de bout-en-bout
  - Exemple : From Nand to Tetris @ Coursera

Hypothèse commune :
pas de prérequis informatique
(... un peu étonnant en 2018 ... mais les choses évoluent ...)



Principale source d'inspiration : SICP

- Cours d'introduction à l'informatique pour ingénieurs au MIT
  - de 1984 à 2005
- Approche "langage" mais structurée autour des concepts plutôt que des constructions syntaxiques
  - Abstractions (structures / contrôle)
  - Récursivité
  - Modularité
  - Etc.
- Basé sur le langage Scheme (dialecte de Lisp)

cf. également : How To Design Programs

```
(define (fact n)
  (if (zero? n)
      1
      (* n (fact (- n 1)))))

(fact 3)
→ (* 3 (fact 2))
→ (* 3 (* 2 (fact 1)))
→ (* 3 (* 2 (* 1 (fact 0))))
→ (* 3 (* 2 (* 1 1)))
→ (* 3 (* 2 1))
→ (* 3 2)
→ 6
```

2. Le Cours

«Eléments de Programmation 1»

Historique

Depuis 1999 le cours d'introduction à l'informatique à l'UPMC Campus Jussieu désormais **Sorbonne Université Campus Science**, suit les mêmes lignes directrices :

- Le thème principal est : **La résolution de problèmes par des principes d'algorithmique et de programmation**
- Le cours est basé sur **un langage de programmation généraliste**
  1999 - 2014 : un sous-ensemble de Scheme : Scheme101
  2015 - ... : un sous-ensemble de **Python 3** : Python101
- ... et supporté par un **environnement de développement pédagogique**
  1999 - 2010 : DrScheme (maintenant DrRacket)
  2011 - 2014 : MrScheme (dans le navigateur)
  2015 - ... : MrPython (un *fork* "amical" de IDLE3)

Quelques Statistiques

Un cours “offline” (présentiel) de taille importante

- ≃900 étudiants chaque année (bientôt $\geq$2k ?)
- ≃6 sessions en parallèle - **Cours magistral**
  - par un chargé de cours en amphi
- ≃40 groupes
  - **Travaux Dirigés** (TD) : chargé de TD
  - **Travaux sur Machine Encadrés** (TME) : chargé de TD + adjoint de TME

⇒ ce n’est pas un MOOC (... mais cours également utilisé au CNED)

L’équipe du cours

Romain Demangeon est responsable administratif du cours

Je m’occupe avec Romain de la ligne pédagogique et des supports, en plus du développement de MrPython

Autres contributeurs :

- Maryse Pelletier, Karine Heydemann, Fabien Tarissan, Clémence Magnien, Christophe Marsala, Antoine Genitrini, Choun Tong Lieu, Pascal Manoury, Mathilde Carpentier, Isabelle Mounier, Marie-Jeanne Lesot.
- Projets étudiants sur MrPython (en M1)
- ... et bien d’autres, dont nos étudiants qui expérimentent le cours chaque année !

Quelques grandes lignes directrices du cours

- La pensée algorithmique est au coeur du cours, la programmation permet de l'expérimenter en pratique, de la concrétiser
- Le langage de support est un "vrai" langage mais :
  - on se limite à un sous-ensemble du langage spécialement conçu pour le cours
    ⇒ idée du "pseudo-code exécutable" ⇒ **Python101**
  - Apprendre (en partie) le langage est plus un "effet de bord" qu'un objectif premier
    ⇒ cependant les programmes écrits sont de "vrais" programmes Python, bien écrits, bien commentés, bien testés, etc.
  - **Important** : nous ne cachons pas la différence entre Python101 (mode étudiant) et le "vrai" Python (mode expert) ... et nous expliquons/justifions nos choix de contraintes
- Emphase sur des activités "périphériques"
  - Test unitaire systématisé
  - Langage semi-formel de spécification/documentation des fonctions
  - Annotations de type et autres conventions de codage

À propos de Python101

Notre *leitmotiv* :

1 problème = 1 fonction (Python) pour le résoudre

- Les petits problèmes sont résolus par des fonctions Python simples / courtes
- Les problèmes plus complexes sont **décomposés** en problèmes plus simples ...
  (... jusqu'à l'obtention de sous-problèmes "suffisamment simples")
  ... et résolus par **composition fonctionnelle**

(remarque : dans la plupart des cas les étudiants ne décomposent pas eux-mêmes les problèmes, car c'est une tâche difficile, mais la méthode est expliquée et illustrée)

- La documentation *docstring* d'une fonction explique le problème qu'elle résout de façon "normalisée" ⇒ **spécification** (semi-formelle) **de fonction**

Respect des conventions Python101

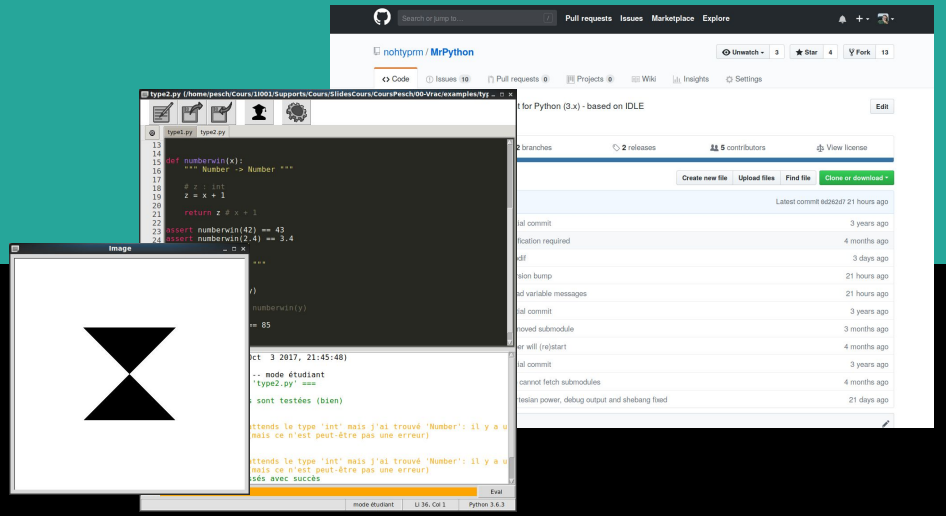
Question : comment s'assurer que les conventions (assez strictes) de codage sont respectées ?

Réponses :

1. encadrement par les enseignants

2. MrPython !

(message à caractère informatif)



MrPython

Un environnement de développement pédagogique pour Python

Mon MrPython en 180 secondes …

Cahier des charges :

Fonctionnalités minimales d'un éditeur Python :

- Indentation assistée
- Coloration syntaxique

Support spécifique pour Python 101 :

- Vérifications (si possibles statiques) des conventions Python101
- Messages d'erreurs contextualisés et précis
  - Erreurs syntaxiques
  - Erreurs de convention
  - Erreurs de l'interprète Python (on ne cache rien)

Solution : MrPython

- Un environnement *"distraction-free"*
- Un fork "amical" de IDLE3
  - Python Open Source License
- **Analyses statiques dédiées …**
- Développé (en partie) par des étudiants / pour des étudiants

⇒ DEMO !

Exemples sur les listes (cours 6 à 8)

- Du plus simple ...
  - Exercice 6.2 : Maximum d'une liste
- ... au plus complexe
  - Exercice 7.5 : Base de données des étudiants
  - Exercice 8.6 : Base de données compréhensive (variante du 7.5)

Python101 pour “language lawyers”

Un sous-ensemble stricte de Python 3.x

- Fonctionnel : les fonctions Python 101 sont “pures”
  - Notion de “pureté observationnelle”
- Portée lexicale stricte
- Typage fort et (quasiment) strict
- Tests unitaires obligatoires
- Pas de classes (pas d’OO)
- Bibliothèque standard réduite : quelques *builtins*, math, random et une bibliothèque “maison” pour les figures géométriques
- Quelques constructions avancées : compréhensions, with, etc.

⇒ Python101 n’est pas un langage complet
.... Mais tout programme Python101 est un programme Python 3 (bien typé, “joli”)

“Pureté Observationnelle” … Qu’est-ce ?

Les fonctions Python101 doivent être **composables** :

- Pour l’appelant : une fonction doit se comporter comme une “fonction mathématique”
  - Pour les mêmes arguments en entrée, retourne la même valeur en sortie, sans effet mémoire.
- Transparence référentielle :

  $\forall$ `EXPR,  EXPR[f(v1,v2,...,vn)]  = EXPR[u]  en supposant f(v1,v2,...,vn) = u`
- Effets de bord contrôlés :
  - On peut effectuer des effets de bord (écritures mémoire) dans le corps d’une fonction
    … mais il ne doivent pas “s’échapper”

Python101 : typage

- **Types simples :** `NoneType`, `bool`, `int`, `float`, `Number` (`int` *ou* `float`)
- **Types iterables**
  - **Sequences :** `range`, `str`, `list[α]`
  - **Tuples :** `tup[α₁,α₂,...,αₙ]` (non-sequentiels !)
  - **Ensembles :** `Set[α]` avec `α` *hashable*
  - **Dictionaires :** `Dict[α:β]` avec `α` *hashable*
- Définitions de types :

```
# type MyType ::= <type-expr>
```

⚠ Pas de classes !

Annotations :

- **Signatures de fonction :** `α₁ * α₂ *... * αₙ → β`
  - Fonctions partielles : `α₁ * α₂ *... * αₙ → β + NoneType`
- **Déclarations de Variable** (1ère affectation = initialisation)

```
# x : <type>
x = <init-val>
```

Python 101 : ce qu'il manque (plein de choses ...)

- Programmation Orientée Objet
- Gestion des fichiers et autres E/S
  ⇒ à l'exception d'un exercice illustratif en TME
- La majeure partie de la bibliothèque standard
  (uniquement quelques builtins, math et random)
- De nombreuses constructions de Python : exceptions, générateurs, contextes, décorateurs, etc, etc.

⇒ A l'issu du cours, les étudiants sont prêts pour un "Cours Python" (*think like a computer scientist*, etc.) ... ainsi qu'un "Cours d'algorithmique"

A Sorbonne Université, ils vont faire du C au second semestre ... mais il y a aussi des projets en Python ...

3. Behind the Scene

du cours magistral ...

... aux examens

1) Le Cours Magistral

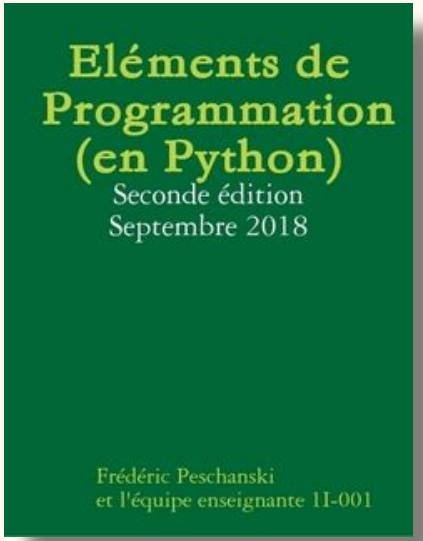
- **Amphi / vidéoprojecteur / chargé de cours / 200+ étudiants**
- **Objectifs :**
  - Présenter les concepts fondamentaux sans connaissance préliminaire (sauf le cours précédent)
- **Supports :**
  - Transparents du cours (pour ma part : jupyter + RISE)
  - Le livre du cours

Enseignement efficace ? ... on critique souvent et facilement les cours magistraux ... mais j'aurais tendance à penser qu'on peut le rendre "efficace" et difficilement "remplaçable"

... mais pour cela il faut atteindre un certain niveau d'interactivité ... ce qui demande quelques efforts (de l'enseignant en amont, et des étudiants sur place).

Le livre du cours

Un cours bien rédigé nous semble indispensable pour un cours introductif de cette ampleur et pour lequel des choix spécifiques (et assez idiosyncrasiques) ont été décidés...



- dans notre cas :
  Eléments de Programmation (en Python)
  - 230+ pages
  - Licence Creative Commons : CC-BY-SA 3.0
    - Ebook gratuit (PDF)
    - Livre imprimé (coût réduit) sur lulu.com
      (discussions en cours avec des éditeurs potentiels)

Plan du livre (et semainier du cours)

1. Premiers pas
2. Instructions, variables et alternatives
3. Répétitions et boucles
4. Plus sur les boucles
5. Séquences, intervalles et chaînes de caractères
6. Listes
7. N-uplets et décomposition de problèmes
8. Compréhensions de listes
9. Ensembles et dictionnaires
10. Compréhensions d'ensembles et de dictionnaires compréhensions
11. Ouverture sur la programmation orientée objet

Les concepts “*avant*” les constructions syntaxiques ⇒ une illustration avec les boucles

- Nous introduisons (bien sûr) les constructions de boucle ainsi que leur sémantique : while et for ... in ...
  (chapitre 3)
- Mais les concepts fondamentaux suivent rapidement (chapitre 4) :
  - Notion de **simulation de boucle** (où les boucles “sur papier”)
  - Notion de **correction** : invariants de boucles
    Important : les étudiants n’inventent pas les invariants en examen
    (ils les vérifient ou choisissent un “meilleur” invariant parmi une liste de possibilités)
    - Notion de **terminaison** : variants de boucles
    - Notion d’**efficacité** : plutôt par l’expérimentation, les notions de complexité sont juste abordées de façon succincte et en complément dans le livre

2) Les Travaux Dirigés

- **≈30 étudiants, 1 chargé de TD, une petite salle de TD et un tableau noir**
- **Objectifs :**
  - Illustrer les concepts vus en cours, résoudre les problèmes “ensemble”
  - Donner un cadre de réponse le plus précis possible
- **Supports :**
  - Recueil d’exercices (énoncés seuls pour les étudiants / consignes de corrections pour l’enseignant)

Efficace ? … exercices “papier & crayon”: est-ce utile ?

… Oui ! la réflexion est algorithmique et Python est utilisé comme du “pseudo-code”… Il faut aussi penser à des principes “papier & crayon” intéressants permettant de “réfléchir” aux algorithmes/programmes (ex.: simulations de boucles) …

Condition nécessaire : un contrat d’apprentissage équilibré

- les enseignants doivent un peu se préparer (lecture du livre et des consignes de correction)
- les étudiants doivent venir avec quelques connaissances sur le cours !

(une “lutte” de tous les instants ☻)

Recueil d’exercices

Eléments de Programmation (en Python)
Recueil d’exercices

- 210+ pages (340+ pages avec solutions) ... 70+ exercices

- Enoncés en Creative Commons : CC-BY-SA 3.0
  - Impression pour les étudiants
  - Ebook or impression “à la demande” pour les enseignants
    ⇒ *copyright restreint* sur les solutions
    (consignes plutôt que solutions) !

3) Travaux sur Machine Encadrés (TME)

- ≈15 binômes (pair programming since 1999 !), 1 enseignant, 1 adjoint
- **Objectifs :**
  - Mettre les concepts en pratique sur ordinateur
- **Material :**
  - Sélection dans le recueil d'exercices, après préparation en TD pour les exercices les plus difficiles
    ⇒ le chargé de TD prépare lui-même la sélection
  - Soumission des programmes via un serveur Web

Efficace ?... peu de gens se plaignent !
⇒ mais pour de nombreux exercices se placer derrière l'écran tout de suite n'est pas une bonne idée, ce n'est pas le bon cadre pour la préparation / réflexion préalable ...
... souvent les étudiants n'arrivent pas à réfléchir devant l'ordinateur (distractions etc.) et on constate aussi qu'une ébauche de solution au papier se fait beaucoup plus rapidement ... que la saisie complète du programme solution sur machine (surtout avec MrPython qui rouspète tout le temps)...

4) Evaluation

- Jargon interne : les “épreuves”
- **Sur machine :**
  - Evaluation en contrôle continu des solutions codées par les étudiants
  - TME “solo” : 45 minutes pour résoudre un problème seul face à la machine
- **Sur papier :**
  - Interrogations écrites en TD (à la discrétion des chargés de TD)
  - Devoir sur table : 1h30 d’exercices de type TD
  - Examen final : 2h00 idem

Efficace ? … beaucoup d’étudiants se plaignent (des notes finales) et ce n’est pas non plus une activité “agréable” pour de nombreux enseignants
⇒ il s’agit probablement d’une bonne estimation du niveau de compréhension des étudiants “typiques” (mais ce n’est pas une façon très “sympathique” d’obtenir cette information)
(personnellement j’aimerais qu’on revienne un peu sur ce principe d’évaluation et son usage immodéré dans notre système éducatif … mais le typeur de MrPython m’occupe déjà pas mal …)

(ceci dit, vu le système établi, on peut essayer de faire “au mieux”)

Illustration : l’examen final

- ≈900 étudiants assis et studieux ≈20 enseignants surveillants, du papier et des crayons …
- **Objectifs** :
  - Évaluer (en 2 heures) le niveau de compréhension des étudiants puisque c’est ce qui nous est ultimement demandé (vous avez compris … ce n’est pas l’aspect du métier qui me plait le plus…)
- **Support** :
  - Sujet d’examen avec cadres de réponses (et guides d’indentations 😊) !
  - Également toutes les annales corrigées avec les solutions pour les étudiants

Efficace ? … (subjectivement) Oui !
… Python101 en tant que “pseudo-code exécutable” se révèle plutôt efficace, je suis plutôt surpris de la qualité des réponse (des étudiants qui ont sérieusement suivis le cours… donc environ 50% des effectifs (sic !)).

Conclusion :

Pour nous, produire un “bon” cours nécessite une vision globale :

Livre du cours, recueil d’exos, solutions pour enseignant, environnement de développement dédié ....

Prochaines étapes :

- Plus d’analyses statiques et des messages d’erreurs (encore) plus clairs
  - En particulier : contrôle des effets, vérifications des *returns* ...
- Un peu plus de “marketing”
  - ... édition un peu plus “sérieuse” du livre du cours
  - Un site web pour rendre le cours visible “à l’extérieur”
  - MrPython “ailleurs” ?
- ... voir “plus grand” ?
  - Bientôt $>$ 2k étudiants
  - MOOC (?)