

# 1 Programme du premier semestre

Les séances de travaux pratiques du premier semestre poursuivent les objectifs suivants :

- consolider l'apprentissage de la programmation en langage Python qui a été entrepris dans les classes du lycée;
- mettre en place un environnement de travail;
- mettre en place une discipline de programmation : spécification précise des fonctions et programmes, annotations et commentaires, jeux de tests;
- introduire les premiers éléments de complexité des algorithmes : on ne présente que l'estimation asymptotique du coût dans le cas le pire;
- introduire des outils de validation : variants et invariants.

Le tableau ci-dessous présente les thèmes qui sont abordés lors de ces séances, et, en colonne de droite, une liste, sans aucun caractère impératif, d'exemples d'activités qui peuvent être proposées aux étudiants. L'ordre de ces thèmes n'est pas impératif.

Aucune connaissance relative aux modules éventuellement rencontrés lors de ces séances n'est exigible des étudiants.

Thèmes	Exemples d'activité, au choix du professeur et non exigibles des étudiants. Commentaires.
Recherche séquentielle dans un tableau unidimensionnel. Dictionnaire.	Recherche d'un élément. Recherche du maximum, du second maximum. Comptage des éléments d'un tableau à l'aide d'un dictionnaire. <i>Manipulations élémentaires d'un tableau unidimensionnel. Utilisation de dictionnaires en boîte noire. Notions de coût constant, de coût linéaire.</i>
Algorithmes opérant sur une structure séquentielle par boucles imbriquées.	Recherche d'un facteur dans un texte. Recherche des deux valeurs les plus proches dans un tableau. Tri à bulles. <i>Notion de complexité quadratique. On propose des outils pour valider la correction de l'algorithme.</i>
Utilisation de modules, de bibliothèques.	Lecture d'un fichier de données simples. Calculs statistiques sur ces données. Représentation graphique (histogrammes, etc.).
Algorithmes dichotomiques.	Recherche dichotomique dans un tableau trié. Exponentiation rapide. <i>On met en évidence une accélération entre complexité linéaire d'un algorithme naïf et complexité logarithmique d'un algorithme dichotomique. On met en œuvre des jeux de tests, des outils de validation.</i>
Fonctions récursives.	Version récursive d'algorithmes dichotomiques. Fonctions produisant à l'aide de <code>print</code> successifs des figures alphanumériques. Dessins de fractales. Énumération des sous-listes ou des permutations d'une liste. <i>On évite de se cantonner à des fonctions mathématiques (factorielle, suites récurrentes). On peut montrer le phénomène de dépassement de la taille de la pile.</i>
Algorithmes gloutons.	Rendu de monnaie. Allocation de salles pour des cours. Sélection d'activité. <i>On peut montrer par des exemples qu'un algorithme glouton ne fournit pas toujours une solution exacte ou optimale.</i>
Matrices de pixels et images.	Algorithmes de rotation, de réduction ou d'agrandissement. Modification d'une image par convolution : flou, détection de contour, etc. <i>Les images servent de support à la présentation de manipulations de tableaux à deux dimensions.</i>
Tris.	Algorithmes quadratiques : tri par insertion, par sélection. Tri par partition-fusion. Tri rapide. Tri par comptage. <i>On fait observer différentes caractéristiques (par exemple, stable ou non, en place ou non, comparatif ou non, etc).</i>

# A Langage Python

Cette annexe liste limitativement les éléments du langage Python (version 3 ou supérieure) dont la connaissance est exigible des étudiants. Aucun concept sous-jacent n'est exigible au titre de la présente annexe. Aucune connaissance sur un module particulier n'est exigible des étudiants. Toute utilisation d'autres éléments du langage que ceux que liste cette annexe, ou d'une fonction d'un module, doit obligatoirement être accompagnée de la documentation utile, sans que puisse être attendue une quelconque maîtrise par les étudiants de ces éléments.

## Traits généraux

- Typage dynamique : l'interpréteur détermine le type à la volée lors de l'exécution du code.
- Principe d'indentation.
- Portée lexicale : lorsqu'une expression fait référence à une variable à l'intérieur d'une fonction, Python cherche la valeur définie à l'intérieur de la fonction et à défaut la valeur dans l'espace global du module.
- Appel de fonction par valeur : l'exécution de  $f(x)$  évalue d'abord  $x$  puis exécute  $f$  avec la valeur calculée.

## Types de base

- Opérations sur les entiers (`int`) : `+`, `-`, `*`, `//`, `**`, `%` avec des opérandes positifs.
- Opérations sur les flottants (`float`) : `+`, `-`, `*`, `/`, `**`.
- Opérations sur les booléens (`bool`) : `not`, `or`, `and` (et leur caractère paresseux).
- Comparaisons `==`, `!=`, `<`, `>`, `<=`, `>=`.

## Types structurés

- Structures indicées immuables (chaînes, tuples) : `len`, accès par indice positif valide, concaténation `+`, répétition `*`, tranche.
- Listes : création par compréhension `[e for x in s]`, par `[e] * n`, par `append` successifs; `len`, accès par indice positif valide; concaténation `+`, extraction de tranche, copie (y compris son caractère superficiel); `pop` en dernière position.
- Dictionnaires : création `{c1 : v1, ..., cn : vn}`, accès, insertion, présence d'une clé `k in d`, `len`, `copy`.

## Structures de contrôle

- Instruction d'affectation avec `=`. Dépaquetage de tuples.
- Instruction conditionnelle : `if`, `elif`, `else`.
- Boucle `while` (sans `else`). `break`, `return` dans un corps de boucle.
- Boucle `for` (sans `else`) et itération sur `range(a, b)`, une chaîne, un tuple, une liste, un dictionnaire au travers des méthodes `keys` et `items`.
- Définition d'une fonction `def f(p1, ..., pn), return`.

## Divers

- Introduction d'un commentaire avec `#`.
- Utilisation simple de `print`, sans paramètre facultatif.
- Importation de modules avec `import module`, `import module as alias`, `from module import f, g, ...`
- Manipulation de fichiers texte (la documentation utile de ces fonctions doit être rappelée; tout problème relatif aux encodages est éludé) : `open`, `read`, `readline`, `readlines`, `split`, `write`, `close`.
- Assertion : `assert` (sans message d'erreur).