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, accès, insertion, `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` $f(p_1, \ldots, p_n)$, `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` $f, g, \ldots$
- 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).

A Langage OCaml

La présente annexe liste limitativement les éléments du langage OCaml (version 4 ou supérieure) dont la connaissance, selon les modalités de chaque sous-section, est exigible des étudiants. Aucun concept sous-jacent n'est exigible au titre de la présente annexe.

A.1 Traits et éléments techniques à connaitre

Les éléments et notations suivants du langage OCaml doivent pouvoir être compris et utilisés par les étudiants dès la fin de la première année sans faire l'objet d'un rappel, y compris lorsqu'ils n'ont pas accès à un ordinateur.

Traits généraux

- Typage statique, inférence des types par le compilateur. Idée naïve du polymorphisme.
- Passage par valeur.
- Portée lexicale : lorsqu'une définition utilise une variable globale, c'est la valeur de cette variable au moment de la définition qui est prise en compte.
- Curryfication des fonctions. Fonction d'ordre supérieur.
- Gestion automatique de la mémoire.
- Les retours à la ligne et l'indentation ne sont pas signifiants mais sont nécessaires pour la lisibilité du code.

Définitions et types de base

- `let`, `let rec` (pour des fonctions), `let rec ... and ....fun` x y `->` e.
- `let` v `=` e `in` e', `let rec` f x `=` e `in` e'.
- Expression conditionnelle `if` e `then` e_V `else` e_F.
- Types de base : `int` et les opérateurs +, -, *, /, l'opérateur `mod` quand toutes les grandeurs sont positives ; `float` et les opérateurs +., -., *., /. ; `bool`, les constantes `true` et `false` et les opérateurs `not`, &&, || (y compris évaluation paresseuse). Entiers et flottants sont sujets aux dépassements de capacité.
- Comparaisons sur les types de base : =, <>, <, >, <=, >=.

Types structurés

- n-uplets ; non-nécessité d'un `match` pour récupérer les valeurs d'un n-uplet.
- Listes : type `'a list`, constructeurs `[]` et `::`, notation `[`x`;` y`;` z`]` ; opérateur `@` (y compris sa complexité) ; `List.length`. Motifs de filtrage associés.
- Type `'a option`.
- Déclaration de type, y compris polymorphe.
- Types énumérés (ou sommes, ou unions), récursifs ou non ; les constructeurs commencent par une majuscule, contrairement aux identifiants. Motifs de filtrage associés.
- Filtrage : `match` e `with` p_0 `->` v_0 `|` p_1 `->` v_1 ... ; les motifs sont exhaustifs, ils ne doivent pas comporter de variable utilisée antérieurement ni deux fois la même variable ; motifs plus ou moins généraux, notation _, importance de l'ordre des motifs quand ils ont des instances communes.

Programmation impérative

- Absence d'instruction ; la programmation impérative est mise en œuvre par des expressions impures ; `unit`, `()`.
- Références : type `'a ref`, notations `ref`, `!`, `:=`. Les références doivent être utilisées à bon escient.
- Séquence ;. La séquence intervient entre deux expressions.
- Boucle `while` c `do` b `done` ; boucle `for` v `=` d `to` f `do` b `done` (on rappelle, quand cela est utile au problème étudié, que les deux bornes sont atteintes).

Divers

- Usage de `begin ... end`.
- Exceptions : `failwith`.

— Utilisation d'un module : notation $M.f$. Les noms des modules commencent par une majuscule.
— Syntaxe des commentaires, à l'exclusion de la nécessité d'équilibrer les délimiteurs dans un commentaire.

A.2 Éléments techniques devant être reconnus et utilisables après rappel

Les éléments suivants du langage OCaml doivent pouvoir être utilisés par les étudiants pour écrire des programmes dès lors qu'ils ont fait l'objet d'un rappel et que la documentation correspondante est fournie.

Définition et types de base

— Types de base : opérateur `mod` avec opérandes de signes quelconques, opérateur `**`.
— Types `char` et `string` ; `'`x`'` quand x est un caractère imprimable, `"`x`"` quand x est constituée de caractères imprimables, `String.length`, $s.[i]$, opérateur `^`. Existence d'une relation d'ordre total sur `char`. Immuabilité des chaines.
— Fonctions de conversion entre types de base.
— `print_int`, `print_string`, `print_float`.

Types structurés et structures de données

— Listes : les fonctions `mem`, `exists`, `for_all`, `filter`, `map` et `iter` du module `List`.
— Tableaux : type `'a array`, notations $[\vert \ldots \vert]$, $t.(i)$, $t.(i)$ `<-` v ; les fonctions suivantes du module `Array` : `length`, `make`, `make_matrix`, `init`, `copy`, `mem`, `exists`, `for_all`, `map` et `iter`.
— Types enregistrements immuables et mutables, notations associées.
— Types mutuellement récursifs.
— Piles et files mutables : fonctions `create`, `is_empty`, `push` et `pop` des modules `Queue` et `Stack`.
— Dictionnaires mutables réalisés par tables de hachage sans liaison multiple ni randomisation par le module `Hashtbl` : fonctions `create` (on élude toute considération sur la taille initiale), `add`, `remove`, `mem`, `find`, `find_opt` et `iter`.