FSAB 1401 - Informatique Q1

Aurélie Maillard

Année académique 2018-2019

L'entièreté de ces notes est basée sur le cours d'informatique 1 de la première année de bachelier en science de l'ingénieur orientation ingénieur civil.
Les sources sont :

- le syllabus interactif du cours ;
- les remarques faites par les tuteurs lors des APE.

Table des matières

1 Introduction

1.1 The way of the program

1.1.1 Vocabulaire relatif au chapitre

— Un algorithme : Une suite d'étapes pour résoudre un type de problème

— Un bug : Une erreur dans le programme

— Un commentaire : Une information destinée à celui qui lit le code et qui n'a pas d'effet sur l'exécution du programme (signalé par le symbole #)

— *Debugging* : Trouver et retirer les erreurs d'un programme

— Le langage formel : Tous les langages créés pour un usage spécifique, dont le langage de programmation

— Un programme : Une séquence d'instructions qui spécifie à un ordinateur des actions et des calculs à exécuter

— Une erreur d'exécution (*runtime error*) : Une erreur qui n'apparaît que lorsqu'un programme est lancé mais qui l'empêche de s'exécuter

— Une erreur sémantique (*semantic error* : Une erreur qui altère l'exécution d'un programme, qui a un effet différent de celui attendu

— Une erreur de syntaxe (*syntax error* : Une erreur dans un programme qui le rend impossible à analyser et donc l'empêche de s'exécuter

— La syntaxe : La structure d'un programme et ses règles

1.1.2 Fonctions introduites dans le chapitre

print( )
✍ Afficher la valeur entre ( )

input( )
✍ Obtenir des données du clavier

1.2 Variables, expressions and statements

1.2.1 Vocabulaire relatif au chapitre

— Concaténer : Joindre deux strings bout-à-bout

— *Data type* : Ensemble de valeurs

— Une expression : Une combinaison de variables, d'opérations et de valeurs qui retourne une valeur finale

— *Float* : Type de données qui comprend tous les nombres à virgule

— *Int* : Type de données qui comprend tous les nombres entiers

— *Str* : Type de données qui comprend toutes les chaines de caractères

— *Bool* : Type de données qui comprend les valeurs *True* et *False*

— *Statement* : Une instruction que l'interpréteur Python peut exécuter

— Une valeur : Un nombre, une chaîne de caractère, une liste, etc. qui peut être stockée dans une variable ou calculée dans une expression

— Une variable : Un nom (doit commencer par une lettre ou _) qui se réfère à une valeur

1.2.2 Fonctions introduites dans le chapitre

type( )
✍ Indiquer le type de données de l'argument entre ( )

int( )
✍ Convertir l'argument entre ( ) en int

float( )
✍ Convertir l'argument entre ( ) en float

str( )
✍ Convertir l'argument entre ( ) en str

1.2.3 Opérateurs introduits dans le chapitre

\+ : addition

– : soustraction

* : multiplication

** : exposant

/ : division

// : division entière

% : reste de la division

== : égal

1.2.4 *Statements* introduits dans le chapitre

affectation
`variable = valeur`
✍ Affecter une valeur à une variable

1.3 Hello, little turtles !

1.3.1 Vocabulaire relatif au chapitre

— Une condition de fin : Une condition qui fait cesser la boucle de répéter son corps

1.3.2 Fonctions introduites dans le chapitre

range( )
✍ Générer une séquence d'*integers*
`ex : range (a,b,c)`
`a = valeur minimum`
`b = valeur maximum + 1`
`c = pas`

1.3.3 *Statements* introduits dans le chapitre

boucle for
```
for variable in list/range( ):
    instructions
```
✍ Répéter un certain nombre de fois les instructions situées dans le corps de la boucle for

1.4 Functions

1.4.1 Vocabulaire relatif au chapitre

— Une fonction : Une suite nommée d'instructions qui exécutes des opérations utiles, prenant (ou pas) en compte un (des) paramètre(s) et produisant (ou pas) un résultat

— L'argument d'une fonction : Une valeur fournie à une fonction lorsqu'elle est appelée

— Un appel de fonction : Une instruction qui exécute une fonction

— Une définition de fonction : Une instruction qui crée une nouvelle fonction et qui spécifie son nom, ses paramètres et les instructions qu'il exécute

— Une composition de fonction : Un appel de fonction dans une autre fonction

— Le flux d'exécution : L'ordre dans lequel les instructions sont exécutées lors de l'exécution d'un programme

— Une fonction *fruitful* : Une fonction qui retourne une valeur lorsqu'elle est appelée

— Une fonction *void* : Une fonction qui ne retourne pas de valeur lorsqu'elle est appelée (>< *fruitful*)

— Une variable locale : Une variable définie dans une fonction qui ne peut être utilisée qu'à l'intérieur de celle-ci

— Un paramètre : Un nom utilisé dans une fonction qui se réfère à la valeur utilisée comme argument

— L'en-tête d'instruction : Première partie d'une instruction, qui commence avec un mot-clé et termine avec :

— Le corps d'une instruction : Suite d'instructions Python identée sur la même ligne

1.4.2 *Statements* introduits dans le chapitre

définition d'une fonction

```
def nom(paramètres):
""" documentation """
    instructions
```

✍ Créer une fonction et spécifier son nom, ses paramètres et les instructions qu'elle exécute

1.5 Conditionnals

1.5.1 Vocabulaire relatif au chapitre

— Une expression booléenne : Une expression soit *True* soit *False*

— Une condition : L'expression booléenne dans un *statement* conditionnel qui détermine quelle branche est exécutée

— Une condition enchaînée : Une branche conditionnelle qui a plus que deux flux d'exécution possibles

— Un opérateur de comparaison : Un opérateur qui produit comme résultat un bool

— Un opérateur logique : Un opérateur qui combine deux expressions booléennes

1.5.2 Fonctions introduites dans le chapitre

abs( )

✍ Donner la valeur absolue de la valeur entre ( )

min(, ,)

✍ Donner la valeur minimale parmi les valeurs entre ( )

max(, ,)
✍ Donner la valeur maximale parmi les valeurs entre ( )

pow(,)
✍ Donner la puissance n^{eme} d'une base

round( )
✍ Arrondir la valeur entre ( ) à l'unité la plus proche

1.5.3 Opérateurs introduits dans le chapitre

Opérateurs logiques

and : et

or : ou

not : pas

Opérateurs de comparaison

== : égal

!= : différent de

< : plus petit

> : plus grand

<= : plus petit ou égal

>= : plus grand ou égal

1.5.4 *Statements* introduits dans le chapitre

condition

```
if expression booléenne 1 :
    instructions   # exécutées si la condition 1 est True
elif expression booléenne 2 :
    instructions   # exécutées si la condition 1 est False et la condition 2 est True
else :
    instructions   # exécutées si la condition 1 et la condition 2 sont False
```

✍ Contrôler le flux d'exécution en fonctions de certaines conditions

return

```
return
```

✍ Permettre de terminer l'exécution d'une fonction avant (ou quand) on arrive à la fin

1.6 Fruitful functions

1.6.1 Vocabulaire relatif au chapitre

— Une fonction booléenne : Une fonction qui retourne comme résultat une valeur booléenne

— *None* : Une valeur spéciale de Python, souvent utilisée lorsqu'une fonction n'atteint pas de *return statement*

— Une suite de tests : Une collection de tests pour un code qu'on écrit

— *Unit testing* : Une procédure automatique pour s'assurer que des parties individuelles d'un code fonctionnent correctement

1.7 Iterations

1.7.1 Vocabulaire relatif au chapitre

— Une boucle infinie : Une boucle qui n'atteint jamais sa condition de fin

— Une itération : Une exécution répétée d'instructions

— Une itération définie : Une boucle où le nombre de répétition a une certaine limite (boucle for)

— Une itération indéfinie : Une boucle qui continue jusqu'à ce qu'une condition soit satisfaite (boucle while)

— Une boucle : Une structure qui permet de répéter une série d'instructions jusqu'à ce qu'une condition de fin soit satisfaite

— Une variable de boucle : Une variable qui intervient dans la condition de fin d'une boucle

— Une boucle imbriquée : Une boucle dans le corps d'une boucle

1.7.2 *Statements* introduits dans le chapitre

boucle while

```
while condition :
    instructions
```

✍ Exécuter une série d'instructions tant que certaines conditions sont satisfaites

break

```
break
```

✍ Quitter immédiatement le corps d'une boucle

continue

```
continue
```

✍ Quitter immédiatement le corps d'une boucle seulement pour l'itération en cours

incrémentation

```
variable += valeur_à_ajouter
```

✍ Assigner à la variable son "ancienne" valeur incrémentée d'une autre valeur (valable pour tous les autres opérateurs de base)

1.8 Modules

1.8.1 Vocabulaire relatif au chapitre

— Un module : Un fichier contenant des définitions et *statements* Python prêts à l'emploi pour d'autres programmes, rendu disponible grâce à l'instruction import

— Un objet : Un nom auquel une méthode peut se référer

— Une propriété (*attribute*) : Une variable définie dans un module, obtenu avec le *dot operator* (.)

— Une méthode : Une fonction rattachée à un objet

— Un espace-nom : Une collection d'indentifiants qui appartiennent à un module ou à une fonction

```
ex : turtle
import turtle #module
alex = turtle.Turtle #objet = module.méthode
alex.color(red) #object.attribute(argument)
alex.forward(30) #objet.méthode(argument)
```

1.8.2 *Statements* introduits dans le chapitre

import

```
import module                # importe le nom du module dans l'espace-nom
from math import méthodes  # importe le nom des fonctions dans l'espace-nom
from math import *           # importe tous les identifiants dans l'espace-nom
```

✍ Permettre d'utiliser des variables et fonctions définies dans un autre module Python

1.8.3 Modules introduits dans le chapitre

- turtle
- random
- time
- string

Les méthodes et *attributes* des différents modules sont disponibles dans la documentation de Python

2 Data Structures

2.1 Strings

2.1.1 Vocabulaire relatif au chapitre

— Une valeur par défault : Une valeur donnée à un paramètre si aucun argument n'est foruni lors de l'appel d'une fonction

— *Immutable* : Qui ne peut pas être modifié (strings et tuples)

— *Mutable* : Qui peut être modifié (listes)

— Un index : Une variable ou une valeur utilisée pour sélectionner un élément d'une collection ordonnée

— *Slice* : Une partie d'un string

2.1.2 Fonctions introduites dans le chapitre

len( )

✍ Retourner le nombre de caractères de l'argument entre ( )

list( )

✍ Créer une liste à partir du string entre ( )

find( )

✍ Déterminer l'index d'un caractère/string

2.1.3 Opérateurs introduits dans le chapitre

Opérateur d'indexation

[ ] : sélectionne un caractère

```
ex : CI = "bière"
     t = CI[3]     # index commence à 0
     print (t)
     >>> r
```

Opérateur *slide*

[:] : sélectionne une chaîne de caractère

```
ex : CI = "bière"
     t = CI[2:4]     # sélectionne le 2 et 3^eme caractère
     print(t)
     >>> èr
```

Opérateurs d'appartenance

in : apparaît dans

not in : n'apparaît pas dans

Opérateurs de comparaison

== : même position dans l'alphabet

!= : différente position dans l'alphabet

< : avant dans l'alphabet

> : après dans l'alphabet

<= : même position ou avant dans l'alphabet

>= : même position ou après dans l'alphabet

Les majuscules précèdent toujours les minuscules.

2.2 Lists

2.2.1 Vocabulaire relatif au chapitre

— Une liste : Une collection ordonnée de valeurs

— Une liste imbriquée : Une liste dans une liste

— Cloner : Créer un objet qui a la même valeur qu'un objet déjà existant

— *A modifier* : Une fonction qui change ses arguments dans le corps de celle-ci

— Un fonction pure : Une fonction qui n'a pas d'effet secondaire

2.2.2 Opérateurs introduits dans le chapitre

+ : concatène deux listes

***** : répète une liste un certain nombre de fois

is : réfère au même objet

2.2.3 *Statements* introduits dans le chapitre

deletion

```
del liste[index]
```

✍ Supprimer de la liste l'élément correspondant à l'index

clonage

```
liste[:]
```

✍ Créer une copie exacte d'une liste