Initiation au Python

A. Bazeniar

Enseignant chercheur
Centre universitaire Abdelhafid Boussouf
Mila, Algérie

Se reporter à des manuels de base et à certaines livres de spécialités

Janvier 2024

Chapitre 1

Introduction

Pourquoi choisir le langage Python ?

Il existe dans la littérature un grand nombre de langages de programmation. Python existe dans le monde de l'Open Source. Il figure parmi les interpréteurs et les compilateurs gratuits. Il est moderne, performants, portables et une documentation très riche.

Définition 1.0.1. *(Stéfane Fermigier)*
Python est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation. Python est développé depuis 1989 *par Guido van Rossum et de nombreux contributeurs bénévoles.*

Il se caractérise par :
— La syntaxe de Python est très simple.
— Python gère ses ressources (mémoire, descripteurs de fichiers...) sans intervention du programmeur.
— La bibliothèque standard de Python, et les paquetages contribués, donnent accès à une grande variété de services.
— Python est un langage qui continue à évoluer.
— Site web officiel pour téléchargement : http ://www.python.org
— Cours de Robert Cordeau : http ://www.afpy.org/Members/bcordeau/Python3v1-1.pdf/download
— L'environnement utilisé est *pycharm* via *https* : *//www.jetbrains.com/pycharm/* puis choisir *pycharm − community*.

1.1 Structure du code Python

1. **Code Python :** le code Python est un ensemble de package qui fournisse un programme final du problème implémenté en question (c'est un livre).
2. **Paquetage(Package) :** Un package est un moyen d'organiser des modules Python en une hiérarchie de répertoires (c'est des chapitres). Il contient un fichier spécial appelé *_ _init_ _.py* dans chaque répertoire pour indiquer que le répertoire doit être traité

comme un package.
Exemple :

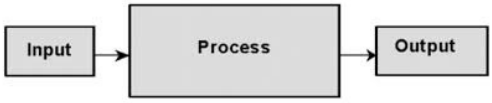
mon_package/
⊢ *__init__.py*
⊢ module1.py
⊢ module2.py
sous_package/
⊢ *__init__.py*
⊢ module3.py

3. **Module(Module) :** Un module est un fichier Python qui peut contenir des définitions de fonctions, de classes et de variables (c'est des paragraphes). On peut importer des modules dans d'autres scripts Python à l'aide du mot-clé *'import'*.
4. **Bloc(Block) :** Un bloc de code est souvent associé à des structures de contrôle telles que les boucles (for, while) ou les structures conditionnelles (if, else, elif (c'est des phrases).
5. **Instruction(Statement) :** est généralement une ligne de code qui accomplit une tâche particulière (c'est des mots clés).

1.2 Catégories d'instructions

Chaque instruction correspond à un mot-clé spécifique

1. Affectation : $=$ Opération Multiple $+=$ $-=$ $*=$ $/=$ Exemple : $z+=1$ Équivaut à $z = z + 1$.
2. Définition : [local]/nonlocal/global/def/class/with ... as/from ... import ... as
3. Répétition : while/for
4. Condition : if/elif/else/assert/try/except
5. Rupture : break/continue/return/yield/raise
6. Inaction : pass



La programmation d'un ordinateur consiste en effet à « expliquer » en détail à une machine ce qu'elle doit faire.

Remarque 1.2.1. *Les 33 « mots réservés » ci-dessous (ils sont utilisés par le langage lui-même) :*

and as assert break class continue def

del elif else except False finally for

from global if import in is lambda

None nonlocal not or pass raise return

True try while with yield

L'interpréteur fonctionne sur le modèle,

```
>>> instruction python
résultat
```

où le triple chevron correspond à l'entrée (input) que l'utilisateur tape au clavier, et l'absence de chevron en début de ligne correspond à la sortie (output) générée par Python.

Le \ (backslash) en Python indique que la ligne de code n'est pas finie. Les ... indique la suite du de la ligne précédente.

```
>>> Voici une ligne de code qui \
2 ... est vraiment très longue car \
3 ... elle est découpée sur trois lignes
4 résultat
```

1.2.1 Affectation

En Python comme dans de nombreux autres langages, l'opération d'affectation est représentée par le signe égale.

```
>>> n = 7    définir la variable n et lui donner la valeur 7
>>> msg = "Quoideneuf?"    affecter la valeur "Quoi de neuf ?" à msg
```

Deux noms de variables, à savoir n, msg. Deux séquences d'octets, où sont encodées le nombre entier 7, la chaîne de caractères Quoi de neuf ? **Affectations multiples**

```
>>> x = y = 7
>>> a, b = 4, 8.33
```

Opérateurs de comparaison :

$x == y$ x est égal à y
$x! = y$ x est différent de y
$x > y$ x est plus grand que y
$x < y$ x est plus petit que y
$x >= y$ x est plus grand que, ou égal à y
$x <= y$ x est plus petit que, ou égal à y

1.2.2 Variables

Sous Python, les noms de variables doivent en outre obéir à quelques règles simples :

- Python est un langage au typage dynamique :Python devine si la variable est un entier ou un caractère...
- Un nom de variable est une séquence de lettres $(a \to z, A \to Z)$ et de chiffres $(0 \to 9)$, qui doit toujours commencer par une lettre (Par convention : les lettres minuscules pour les variables et les majuscules pour les constantes).
- Seules les lettres ordinaires sont autorisées.

```
>>> x = 2
>>> x
2
```

Les trois principaux types dont nous aurons besoin dans un premier temps sont les entiers (integer ou *int*), les nombres décimaux que nous appellerons *floats* et les chaînes de caractères (string ou *str*). Pour une chaîne de caractères, il faut l'entourer de guillemets

```
>>> y = 3.14
>>> y
3.14
>>> a = " bonjour "
>>> a
'bonjour '
```

On peut écrire des nombres très grands ou très petits avec des puissances de 10 en utilisant le symbole e :

```
>>> 1e6
1000000.0
>>> 3.12e - 3
0.00312
```

1.2.2.1 Opérations sur les types numériques

L'opérateur / effectue une division. + la somme, − la soustraction, ∗ la multiplication, ∗∗ la puissance, // le quotient d'une division et mod le modulo.

```
>>> 2 * *3
8
>>> 5//4
1
>>> 8 mod 4
0 >>> i = 0
>>> i+ = 1
>>> i
1
```

Python propose les fonctions min() et max() qui renvoient respectivement le minimum et le maximum de plusieurs entiers et / ou floats :

```
>>> min(1, −2, 4)
−2
```

Exemple 1.2.1. — *(1+2)**3*
— *"Da" * 4*
— *"Da" + 3*
— *str(4) * int("3")*
— *int("3") + float("3.2")*

La fonction print() : Par la douche *entrer*. Ou par la fonction *print()* dans un programme.

```
>>> print(n)
7
>>> print(msg)
"Quoi de neuf ?"
```

le caractère";" sert à séparer plusieurs instructions Python sur une même ligne :

```
>>> print (" Hello "); print (" Ommar ")
Hello
Ommar
>>> print (" Hello ", end =""); print (" Ommar ")
HelloOmmar
>>> print (" Hello ", end =" "); print (" Ommar ")
Hello Ommar
```

La fonction $print()$ peut également afficher le contenu d'une variable quel que soit son type. Par exemple, pour un entier,

```
>>> var = 3
>>> print(var)
3
```

Tapez le script,

```
>>> x = 32 >>> print("Ommara", x, "ans")
ommar a 32 ans
```

La fonction input() : Elle est utiliser pour interagir avec l'utilisateur, soit pour obtenir des données, soit pour obtenir un résultat quelconque.

```
>>> age = input("Entrez votre age : ") print(age)
```

Exemple 1.2.2. *Écrire un programme qui calcule la circonférence et la surface d'un cercle.*

1.2.2.2 Liste

Une liste est une structure de données qui contient une série de valeurs. Python autorise la construction de liste contenant des valeurs de types différents (par exemple entier et chaîne de caractères), ce qui leur confère une grande flexibilité. Une liste est déclarée par une série de valeurs (n'oubliez pas les guillemets, simples ou doubles, s'il s'agit de chaînes de caractères) séparées par des virgules, et le tout encadré par des crochets. En voici quelques exemples,

```
1 >>> animaux = ["girafe","tigre","singe","souris"]
2 >>> tailles = [5,2.5,1.75,0.15]
3 >>> mixte = ["girafe",5,"souris",0.15]
4 >>> animaux
5['girafe',' tigre',' singe',' souris']
6 >>> tailles
7[5,2.5,1.75,0.15]
8 >>> mixte
9['girafe',5,' souris',0.15]
10 >>> animaux[0]
11'girafe'
```

La ligne 10 pour appeler l'élément de la liste. On peut ajouter dees éléments

```
>>> a = a + [15]
>>> a
[15]
>>> a.append(13)
>>> a
[15,13]
```

Exemple 1.2.3. *Générer une liste de* 20 *réels suivant un loi uniforme dans* $[1,3]$. *(Ind. importer le module random).*

```
>> animaux[1 : −1]
['tigre',' singe']
```

Finalement, on se rend compte que l'accès au contenu d'une liste fonctionne sur le modèle liste[début :fin :pas].

```
>>> x = [0,1,2,3,4,5,6,7,8,9]
>>> x
[0,1,2,3,4,5,6,7,8,9]
>>> x[:: 2]
[0,2,4,6,8]
>>> x[1 : 6 : 3]
[1,4]
```

L'instruction *len()* vous permet de connaître la longueur d'une liste, c'est-à-dire le nombre d'éléments que contient la liste. L'instruction *range()* est une fonction spéciale en Python qui

génère des nombres entiers compris dans un intervalle. Lorsqu'elle est utilisée en combinaison avec la fonction *list()*, on obtient une liste d'entiers.

```
>>> len([1,2,3,4,5,6,7,8])
8
>>> list(range(10))
[0,1,2,3,4,5,6,7,8,9]
>>> list(range(15,20))
[15,16,17,18,19]
>>> sum(list)
45
>>> min(liste)
0
>>> max(liste)
9
```

L'instruction range() fonctionne sur le modèle range(début, fin, pas).

```
>>> list(range(10,0,-1))
[10,9,8,7,6,5,4,3,2,1]
```

Exemple 1.2.4. *Constituez une liste semaine contenant les 7 jours de la semaine.*

1. À partir de cette liste, comment récupérez-vous seulement les 5 premiers jours de la semaine d'une part, et ceux du week-end d'autre part ? Utilisez pour cela l'indiçage.

2. Cherchez un autre moyen pour arriver au même résultat (en utilisant un autre indiçage).

3. Accéder au dernier jour de la semaine. *4. Inversez les jours de la semaine en une commande.*

1.2.3 Instructions conditionnelle

Une instruction composée se compose :

- D'une ligne d'en-tête terminée par deux-points.
- D'un bloc d'instructions indenté par rapport à la ligne d'en-tête.

```
>>> a = 150
>>> if (a > 100) :
...         print("a dépasse la centaine")
...
```

L'indentation est cruciale en Python, car elle est utilisée pour délimiter les blocs de code. Toutes les instructions au même niveau d'indentation appartiennent au même bloc

```
if (condition) :
    <instruction>
```

```
if (condition) :
    <instruction>
else :
    <instruction>
```

```
if (condition) :
    <instruction>
elif(condition) :
    <instruction>
else :
    <instruction>
```

Instructions composées-blocs d'instructions

```
if embranchement == "vertébrés" :
  if classe == "mammifères" :
    if ordre == "carnivores" :
      if famille == "félins" :
        print("c'est peut-être un chat")
      print("c'est en tous cas un mammifère")
    elif classe == "oiseaux" :
  print("c'est peut-être un canari")
print("la classification des animaux est complexe")
```

1.2.4 Instructions de répétition

Une itération permet de d'exécuter un bloc pour chaque donnée présente dans une séquence.

Instructions for

```
>>> for < var > in < seq >:
...         <block>
... [else :
        < block >]
```

Exemple

```
>>> b = 0
>>> for a in(1,2,3,4) :
...         b+ = a
...else :
        print('a=',a,'b=',b)
```

```
>>> animaux = ["girafe","tigre","singe","souris"]
>>> for i in range ( len ( animaux )) :
...         print (f"L' animal i est un(e)  animaux [i ]")
...
L' animal 0 est un(e) girafe
L' animal 1 est un(e) tigre
L' animal 2 est un(e) singe
L' animal 3 est un(e) souris
```

Instructions break et continue Ces deux instructions permettent de modifier le comportement d'une boucle (for ou while) avec un test. L'instruction break stoppe la boucle.

```
>>> for i in range(5) :
...if i > 2 :
... break
... print (i)
...
0
1
2
```

L'instruction continue saute à l'itération suivante, sans exécuter la suite du bloc d'instructions de la boucle.

```
>>> for i in range(5) :
... if i == 2 :
... continue
... print (i)
...
0
1
3
4
```

Instructions while

```
>>> while < test >:
...          < block >
...[else :
          < block >]
```

Exemple

```
>>> a, b = 1, 0
>>> while b <= 5 :
...          b+ = a
...else :
          print('a =', a,' b =', b)
```

Exemple 1.2.5. *Soit la liste ["vache", "souris", "levure", "bacterie"]. Affichez l'ensemble des éléments de cette liste (un élément par ligne) de trois façons différentes (deux méthodes avec for et une avec while).*

1.2.5 Instructions de définition

La définition de fonction permet d'associer un identificateur avec un bloc d'instructions. Une fonction peut utiliser des données en entrée (appelées arguments) et renvoyer des données en sortie (appelées retours). Après la définition d'une fonction, les instructions du bloc associé peuvent être exécutées par un appel de fonction (Ex. f(5)).

```
>>> def (nam) :
...          < block >
```

Exemple

```
>>> def f(x) :
...          print(x)
... def g(x) :
          returnx, x + 1, x + 2
```

1.2.6 Types de données standards

Les types prédéfinis de Python ne correspondent pas à des mots-clés du langage, mais à des classes, au sens de la programmation par objets

$$[i] = itrable,\ [o] = ordonn,\ [m] = mutable$$

— Vide : None : Ensemble ne contenant qu'une seule valeur (mot réservé) : None

```
>>> False, not
False
(False, True)
```

— Booléen : *bool* : Ensemble ne contenant que deux valeurs (motsréservés) : True,False

— Entier : *int* : Les valeurs numériques sont représentables en décimal (sans préfixe), binaire (préfixe 0B ou 0b)octal (préfixe 0O ou 0o), hexadécimal (0X ou 0x)

```
>>> bin(7), hex(273)
('0B111', '0X111')
```

— Séquence : tuple [i,o] Série ordonnée, itérable, non-mutable de valeurs arbitraires, séparées par des virgules ; Lorsque la série est un singleton, une virgule finale est obligatoire.

```
>>> a = (0,1,4,9,16,25)
>>> a[0], a[2], a[5]
(0, 4, 25)
>>> a[0] = 1
TypeError : tuple object does not support item assignment
```

— Ensemble : $set[i, m]$: Les valeurs d'un ensemble sont obligatoirement délimitées par des {} ; Quelques méthodes : add,update,pop,discaerd,clear,copy.

```
>>> a.clear()
>>> a.add(1)
1
>>> a.update([2,3,4])
1, 2, 3, 4
>>> a.pop()
1
```

— Dictionnaire : $dict[i, m]$: L'accès aux valeurs d'un dictionnaire (en lecture et en écriture) se fait à partir de la clé, en utilisant la notation par crochets ; Quelques méthodes :
len(dictionnaire) : retourne le nombre d'éléments dans le dictionnaire
in : vérifie si une clé est présente dans le dictionnaire
dict.get(cle, valeur_par_defaut) : retourne la valeur associée à la clé, ou la valeur par défaut si la clé n'existe pas
dict.keys() : retourne une liste des clés du dictionnaire
dict.values() : retourne une liste des valeurs du dictionnaire
dict.items() : retourne une liste de tuples (clé, valeur)

```
alpha_dictionnaire = dict()  #dictionnqire vide
alpha_dictionnaire["nom"]="Ahmed"
alpha_dictionnaire["age"] = 30
alpha_dictionnaire["ville"] = "Paris"
print(f"Adresse : alpha_dictionnaire")
```

1.3 fichiers

Lecture dans un fichier Une grande partie de l'information est stockée sous forme de texte dans des fichiers. Pour traiter cette information, vous devez le plus souvent lire ou écrire dans un ou plusieurs fichiers. Python possède pour cela de nombreux outils qui vous simplifient la vie.

```
f=open('n1.txt','w')
s='ecouter le message'
f.write(s)
f = open("n1.txt")
for ligne in f :
   print(ligne[ :-1])
f.close()
```

1.4 importation des fonctions

L'import du module math par exemple autorise toutes les opérations mathématiques usuelles :

```
import math as m                    from math import sqrt
import random                       from random import uniform

a=m.sqrt(2)                         a=sqrt(2)
b=random.uniform(1,2)               b=uniform(1,2)
```

1.5 Les Classes

La classe : Une classe en Python est un plan ou un modèle qui permet de créer des objets avec des caractéristiques et des comportements communs. Elle définit les attributs et les méthodes qui seront disponibles pour tous les objets créés à partir d'elle.

```
class Livre :
    def__init__(self, titre, auteur, annee_publication) :
      self.titre = titre
      self.auteur = auteur
      self.annee_publication = annee_publication

    def afficher_infos(self) :
      print(f"Titre : self.titre")
      print(f"Auteur : self.auteur")
      print(f"Année de publication : self.annee_publication")

    def age(self) :        return 2024 − self.annee_publication

####### creation d'objet
livre1 = Livre("important An Introduction to Optimization ", "Edwin P. Chong, Stanislaw H. Zak", 2013)
livre1.afficher_infos()
print()
print(f"L'âge de {livre1.titre} est de {livre1.age()} ans")
```