ISARA-Lyon F. BILLY

Programmer en Python

L'essentiel pour débuter

Ce document est inspiré de la documentation officielle du langage Python :
`https://docs.python.org/fr/3.7/`
Il n'a pas la prétention de présenter Python de manière exhaustive, mais plutôt de servir de manuel de référence pour les exercices et travaux qui vous seront proposés.
Bonne lecture !

Table des matières

1 La console

La console est l'environnement interactif le plus simple pour coder en Python.

En début de ligne figure l'invite de commande `>>>` (on l'appelle aussi le « prompt »).
Comme son nom l'indique, c'est un symbole à droite duquel on peut taper une commande en Python.
Le retour (la réponse) de l'interpréteur Python se fait sur la ligne suivante.

La console peut permettre d'effectuer des calculs tout simples :

```
>>> 2 + 5
7
>>> 3 * 8
24
>>> (2 + 3) * 5
25
```

Dans la suite de ce document, vous reconnaîtrez facilement les exemples impliquant l'utilisation de la console grâce à la présence du prompt `>>>`.

2 Les variables

Contrairement à d'autres langages de programmation, Python n'a pas de commande pour déclarer une variable.
Une variable est donc créée au moment où on lui affecte une valeur pour la première fois !

2.1 Types de variables

Nous utiliserons essentiellement les types de variables suivants :

- **int** (pour un nombre entier)
- **float** (pour un nombre décimal, dit "à virgule flottante", d'où le nom du type)
- **bool** (pour une variable booléenne, limitée à 2 valeurs (*True* ou *False*))
- **str** (pour une chaîne de caractères, donc un texte)
- **list** (pour une liste de variables, assimilable à un tableau)

2.2 Affectation : l'opérateur « = » et le typage dynamique

Pour affecter une valeur à une variable, on utilise l'opérateur « = ».
Par exemple :

```
a = 5
b = 7.2
prenom = "Lucas"
```

Python est un langage **à typage dynamique**, ce qui signifie (entre autres) que le type est défini automatiquement à l'affectation. Par conséquent, dans l'exemple précédent :
a est de type *int*, b est de type *float* et prenom est de type *str*.

2.3 Détermination du type d'une variable

On utilise l'instruction **type(*variable*)** :

```
>>> a = 5
>>> type(a)
<class 'int'>
>>> prenom = "Lucas"
>>> type(prenom)
<class 'str'>
```

Notons que si on déclare un nombre avec une partie décimale nulle, il est considéré comme un décimal :

```
>>> a = 5.0
>>> type(a)
<class 'float'>
```

2.4 Transtypage

Le typage dynamique de Python permet la conversion du type d'une variable en cours de programme : c'est ce qu'on appelle le **transtypage**.
Ainsi, on peut forcer le type d'une variable avec les fonctions suivantes :

- **int(*variable*)** : permet la conversion en entier
- **float(*variable*)** : permet la conversion en décimal
- **str(*variable*)** : permet la conversion en chaîne de caractères

```
>>> str(8.3)
'8.3'
>>> a = 5
>>> type(a)
<class 'int'>
>>> a = str(a)
>>> type(a)
<class 'str'>
```

3 Les opérateurs

Pour manipuler les variables, on a besoin d'opérateurs.
La très grande majorité d'entre eux concerne les variables numériques (*int* ou *float*).

3.1 Opérateurs de calcul

Opération	Résultat
`x + y`	somme de *x* et *y*
`x - y`	différence de *x* et *y*
`x * y`	produit de *x* et *y*
`x / y`	quotient de *x* et *y*
`x // y`	quotient entier de *x* et *y*
`x % y`	reste de `x / y`
`-x`	négatif de *x*
`abs(x)`	valeur absolue de *x*
`x ** y`	*x* à la puissance *y*

3.2 Opérateur de concaténation

Pour concaténer (c'est-à-dire mettre bout à bout) deux chaînes de caractères, Python utilise l'opérateur « + ».

```
>>> "Hello" + " " + "world"
'Hello world'
```

ATTENTION :
Si, dans une concaténation, on veut ajouter une variable numérique, il faut que celle-ci soit transtypée en chaîne de caractères, sinon une erreur est signalée.

```
>>> a = 60
>>> "Une heure comporte " + a + " minutes"
 Traceback (most recent call last):
   File "<pyshell>", line 1, in <module>
 TypeError: can only concatenate str (not "int") to str
>>> "Une heure comporte " + str(a) + " minutes"
'Une heure comporte 60 minutes'
```

3.3 Opérateurs de comparaison

Ces opérateurs renvoient un booléen : True ou False.

Opérateur	Description	S'applique à des...
x == y	égal à	nombres ET chaînes
x != y	différent de	nombres ET chaînes
x > y x >= y x < y x <= y	supérieur à supérieur ou égal à inférieur à inférieur ou égal à	nombres

3.4 Opérateurs logiques

Ces opérateurs renvoient eux aussi un booléen : True ou False.
Ils servent essentiellement à combiner plusieurs expressions lors d'un test.

Opérateur	Description	**Valeur renvoyée**
(expression 1) and *(expression 2)*	ET logique	True si les deux expressions sont vraies, False dans le cas contraire
(expression 1) or *(expression 2)*	OU logique	True si l'une des expressions est vraie, False dans le cas contraire
not*(expression)*	négation logique	True si l'expression est fausse, False si l'expression est vraie

4 L'éditeur de code

Utiliser la console c'est bien, mais elle ne permet que d'enchaîner les instructions en validant une à une chacune d'entre elles.
Pour écrire un code Python plus élaboré, on utilise donc un éditeur de code.
Une fois écrit, le code peut être enregistré dans un fichier **.py** (extension pour un code Python).

4.1 Inclusion de commentaires

Un bon code inclut nécessairement des commentaires.
Ce sont des lignes **non interprétées par le langage, donc ignorées au moment de l'exécution**, qui permettent une meilleure compréhension du code.
En Python, une ligne de commentaire débute avec le caractère « # »(dièse) :

```
1 t = "une phrase"
2 # on récupère la taille de la chaîne
3 taille_t = len(t)
```

4.2 Utilisation de la console pour les entrées/sorties

4.2.1 Ecrire dans la console avec print()

Au cours de l'exécution du programme, on peut donner des informations à l'utilisateur par l'intermédiaire de la console.
On écrit dans la console avec ***print()*** :

```
mess = "Python, c'est cool !"
# on récupère la taille de la chaîne
taille = len(mess)
print("La chaîne est de taille " + str(taille))
```

Dans la console, on aura l'affichage suivant (le programme lancé ici s'appelle "test") :

```
>>> %Run test.py
  La chaîne est de taille 20
```

4.2.2 Lire dans la console une saisie de l'utilisateur avec input()

Au cours de l'exécution du programme, on peut aussi avoir besoin de récupérer des informations fournies par l'utilisateur. Comme pour l'écriture (avec *print()*), une lecture peut aussi se faire par l'intermédiaire de la console.
On utilise la fonction ***input()*** :

```
1 print("Quel est ton prénom ?")
2 prenom = input()
3 print("Bonjour " + prenom + " !")
```

Dans la console, on aura l'affichage suivant :

```
>>> %Run test3.py
  Quel est ton prénom ?
  Paul
  Bonjour Paul !
```

Notons que la variable renvoyée par la fonction ***input*** (ici *prenom*) est de type "chaîne de caractères". On peut donc l'utiliser directement dans une concaténation, comme sur la ligne 3 du code ci-dessus.

5 Les séquences

En Python, une séquence est une **suite ordonnée de données**.
Dans la suite, on va détailler trois types de séquences très utilisées :

- les **listes** (type ***list***)
- les **chaînes de caractères** (type ***string***)
- les **intervalles** ou **plage** (type ***range***)

5.1 Les listes

La **liste** est utilisée en Python pour regrouper plusieurs valeurs, à la manière d'un tableau.
Elle s'écrit comme unesuite, placée entre crochets, de valeurs (éléments) séparées par des virgules.
Contrairement au type "tableau" dans d'autres langages, les éléments d'une liste ne sont pas obligatoirement tous du même type (entier, chaîne,...), bien qu'à l'usage ce soit souvent le cas. Néanmoins, les listes se manipulant comme des tableaux, nous les assimilerons à ces derniers.

L'accès à un élément donné se fait à l'aide d'une variable entière appelée **indice**. Dans l'exemple suivant, une liste *l1* regroupe des décimaux :

```
l1 = [1.2, 4.0, -5.1, 8.6, -3.7]
i = 2
# on récupère l'élément d'indice 2 dans une variable elt
elt = l1[i]
# on l'affiche dans la console
print(elt)
# on affiche l'élément d'indice 0 dans la console
print(l1[0])
```

La console affiche :

```
>>> %Run test.py
  -5.1
  1.2
```

Comme vous pouvez le remarquer, **l'indice du 1er élément est 0**. Par conséquent, 2 est l'indice du 3ème élément.
Pour toute liste de taille N (donc de N éléments), l'indice du dernier élément sera donc N-1.

5.2 Les chaînes de caractères

La **chaîne de caractères** est une suite ordonnées de caractères.
A ce titre, c'est donc une séquence sous Python.

L'accès à un caractère donné se fait, de la même manière que pour une liste, à l'aide d'un **indice** entre crochets.
Là encore, **l'indice du 1er élément est 0**. Et là aussi, comme pour les listes, pour une chaîne de taille N (donc de N caractères), l'indice du dernier caractère sera donc N-1.

Si on veut afficher la première lettre d'une chaîne, on écrit :

```
ch = "Une chaîne est une séquence"
premLettre = ch[0]
print("La première lettre est " + premLettre)
```

La console affiche :

```
>>> %Run test.py
  La première lettre est U
```

On peut également extraire tout ou partie d'une chaîne.
Par exemple :

```
mot = "informatique"
debutMot = mot[0:4]
```

On indique entre crochets l'indice du 1^{er} caractère à extraire, puis le séparateur **:**, et enfin l'indice du 1^{er} caractère **à ne pas extraire** (même logique que le $2^{ème}$ paramètre de *range*). Ainsi, on va stocker dans la variable *debutMot* les caractères d'indice 0 à 3... donc les 4 premières lettres.
Si on omet le 1^{er} indice (comme ci-dessous), Python considère qu'il faut démarrer au début de la chaîne :

```
debutMot = mot[:4]
```

On obtient le même résultat !

Pour obtenir une suite de caractères au milieu de la chaîne (par exemple les caractères d'indice 5 et 6), on écrira naturellement :

```
milieuMot = mot[5:7]
```

Enfin, pour obtenir la fin de chaîne, par exemple à partir du caractère d'indice 5, on codera :

```
finMot = mot[5:]
```

Si on affiche les variables *debutMot*, *milieuMot* et *finMot* dans la console...

```
print(debutMot)
print(milieuMot)
print(finMot)
```

... la console affiche :

```
info
ma
matique
```

5.3 Les intervalles (ou plages)

Un **intervalle** ou **plage** est une séquence d'**entiers consécutifs**, délimitée par deux bornes (une valeur minimale et une valeur maximale).
La création d'un intervalle sous Python se fait avec la fonction **range**. Comme pour toute fonction (nous y reviendrons !), **range** est suivi de parenthèses (qui attestent qu'il s'agit bien d'une fonction).

On indique dans ces parenthèses les **paramètres** suivants, séparés par des virgules, dans cet ordre :
- la valeur minimale. Ce paramètre est **optionnel** ; lorsqu'il est omis, cette valeur minimale vaut 0.
- la valeur maximale
- le pas ("distance" entre deux éléments consécutifs). Ce paramètre est **optionnel** ; lorsqu'il est omis, le pas prend sa valeur par défaut : 1

ATTENTION :
L'intervalle est **ouvert du côté de sa borne maximale**. Ainsi, **range(1, 5)** renvoie tous les entiers de 1 inclus à **5 exclu**.

Et, comme pour toute séquence :
- on accède à un entier de la plage en spécifiant son indice entre crochets
- l'indice du premier entier (valeur minimale de l'intervalle) vaut 0

5.4 Longueur d'une séquence : la fonction len

Cette fonction **len**(*variable*) ne s'applique qu'aux séquences et renvoie la longueur de celles-ci.

Type de séquence	Valeur renvoyée
liste (type *list*)	nombre d'éléments de la liste
chaîne de caractères (type *str*)	nombre de caractères de la chaîne
intervalle (type *range*)	nombre d'entiers dans la plage

Un exemple d'utilisation avec ces trois types :

```
interv = range(1, 5) # ATTENTION : interv représente la séquence 1,2,3,4 (le 5 est exclu !)
liste1 = ["titi", "tata", "toto"]
mot = "Python"
print(len(interv))
print(len(liste1))
print(len(mot))
```

La console affiche :

```
>>> %Run test.py
4
3
6
```

6 Les structures de contrôle de flux

Dans un algorithme (on l'a vu en modélisant sous forme d'organigramme), le flux logique des étapes comporte la plupart du temps des ruptures.

Elles sont de deux sortes :
- le **test (structure conditionnelle)**
- la **boucle (structure itérative)**

6.1 Structure conditionnelle : l'instruction « if »

Pour implémenter un test, on utilise l'instruction « if ».

La syntaxe est la suivante :

if (condition1) **:**
    (bloc d'instructions 1)
elif (condition2) **:**
    (bloc d'instructions 2)
else :
    (bloc d'instructions 3)

Le mot clé "elif" est un raccourci pour "else if" (sinon si) et permet d'effectuer un second test si (condition1) n'est pas vérifiée.
Il peut y avoir un nombre quelconque de blocs "elif".

TRES IMPORTANT :

- les blocs sont obligatoirement **indentés**, c'est-à-dire décalés par rapport aux mots-clés. C'est cette indentation qui relie le bloc à son mot clé.
- "elif" et "else" (et leurs blocs évidemment) sont optionnels : on peut très bien avoir un simple **if** (sans elif ni else) ou un **if...else** (sans elif).

6.2 Structures itératives

Une **itération** est la répétition d'une ou plusieurs instructions.
Itérer, c'est donc **répéter un bloc d'instructions**.

Il existe deux manières d'implémenter une structure itérative (donc **une boucle**), et il est impératif de bien les distinguer.

6.2.1 La boucle « for »

On utilise une boucle « for » **dans le cas où on connaît à l'avance le nombre d'itérations (tours de boucle)**.

On utilise un entier qui varie à chaque itération : c'est l'**indice** de la boucle.

Par exemple, si on veut afficher le double de chaque nombre entier de 1 à 4 :

```
# on indique l'intervalle de variation [1,5[ de l'indice
# (ATTENTION : la borne supérieure est exclue !)
for i in range(1, 5) :
    double = i * 2
    print("Le double de " + str(i) + " est " + str(double))
```

La console affiche :

```
>>> %Run test.py

  Le double de 1 est 2
  Le double de 2 est 4
  Le double de 3 est 6
  Le double de 4 est 8
```

On peut aussi parcourir une liste avec ce type de boucle.
Dans l'exemple suivant, j prendra successivement les valeurs de tous les éléments de la liste.

```
ch = ""
liste1 = ["Python,", "un", "langage", "sympa"]
for j in liste1 :
    # ch est un accumulateur qui concatène les éléments de la liste
    # (on l'a initialisé avec une chaîne vide)
    ch = ch + " " + j
print(ch)
```

La console affiche :

```
>>> %Run test.py

   Python, un langage sympa
```

6.2.2 La boucle « while »

La boucle « while » est utilisée quand **on ne connaît pas à l'avance le nombre d'itérations**. La sortie d'une boucle while se fait donc sur un test qui implique une ou plusieurs variables **qui évolue(nt)** lorsque la boucle est parcourue.
ATTENTION : si ce n'est pas le cas, on rentre dans une **boucle infinie** ! Ce risque n'existe pas avec une boucle de type « for ».

Dans l'exemple ci-dessous, on parcourt la boucle **tant que** la réponse de l'utilisateur est "O" :

```
rep = "O"
while (rep == "O") :
    nb = input("Donnez un nombre entier \n")
    double = int(nb) * 2
    print("Le double de " + str(nb) + " est " + str(double))
    rep = input("Voulez-vous continuer ? (O/N) \n")
```

La console affiche :

```
>>> %Run test.py

  Donnez un nombre entier
  5
  Le double de 5 est 10
  Voulez-vous continuer ? (O/N)
  O
  Donnez un nombre entier
  4
  Le double de 4 est 8
  Voulez-vous continuer ? (O/N)
  N
```