 **Syntaxes élémentaires en langage Python**

**Les principaux types de variables :**

**Description**

Nombres entiers

**int**

Nombres flottants (représentation des nombres décimaux)

**float**

Chaînes de caractères (string)

**str**

Booléens (valeur True(=1) ou False(=0))

**bool**

Liste d’éléments. NB : Les éléments de la liste peuvent être de tous types.

**list**

**Syntaxe Python**

a=-3 b=7

a=2/7 b=7.0 c=3.3

text=’’ Hello prof ’’

2+3==5 2.7>=5.5

L1=[2,5,7] L2=[‘’salut’’,3.8,8]

 **Typage :** Python est un langage typé, mais l’utilisateur n’a pas à déclarer le type de l’élément qu’il définit et utilise, l’interpréteur Python attribue lui-même le type adéquat. On peut vérifier le type d’un élément avec l’instruction type( ).
**Conversion de type :** On peut effectuer une conversion de type avec les instructions int( ) ; float( ) ; str( ) ; …
A savoir : Une saisie du type « a=3 » génère par défaut un int, mais une saisie « a=3.0 » génère par défaut un float.
L’instruction input() génère automatiquement un élément de type str et il peut donc être nécessaire de convertir le type. En général, on n’utilisera pas l’instruction input() (ni print() ), préférant la définition de fonctions, éventuellement appelées en console.

**Les instructions élémentaires :**

**Syntaxe Python**

**Description**

a**=**7

Affectation de la valeur 7 à la variable a

**Affectation**

a**+**b a**-**b a**\***b a**\*\***b
a**/**b a**//**b a**%**b

Somme, différence, produit, puissance,
quotient décimal, quotient entier, reste de la division euclidienne.

**Opérations élémentaires**

a**==**b a**!=**b
a**<**b a**<=**b a**>**b a**>=**b

Test d’égalité, de non-égalité, de comparaisons. Renvoie un booléen (True ou False). NB : On peut combiner avec **or** ; **and** ; **not**.

**Tests**

**Instruction conditionnelle**

**if** condition **:**
 | *instructions\_V*
**else :**
 | *instructions\_F*

Réalisation des instructions\_V si la condition est True, et réalisation des instructions\_F si la condition est False.
NB1 : else est facultatif
NB2 : dans une série d’instructions conditionnelles imbriquées, « else if » peut se condenser en « elif ».

**Tests sur flottants :** Le codage des nombres décimaux à l’aide du type float génère des approximations et il faut donc éviter d’effectuer des tests d’égalité sur des éléments de type float. Par exemple, 0.1+0.1+0.1==0.3 renvoie False.
**Opérations sur chaînes de caractères :** L’opération ‘sa’+’lut’ effectue la concaténation : ‘salut’. L’opération ‘miam’\*3 effectue une répétition : ‘miammiammiam’.
**Bloc conditionnel :** Une syntaxe experte permet de créer un bloc prenant une valeur en fonction de la réalisation ou non d’une condition :
« valeur\_V if Test else valeur\_F » vaut valeur\_V si le Test est True et vaut valeur\_F sinon.

**La structure fonctionnelle :**

**Syntaxe Python**

**Description**

**def** f(x)**:**
 |*instructions à réaliser*
 **return** resultat

Définition d’une fonction recevant un/des argument(s) x, … et renvoyant resultat. (arguments et résultat de tous types possibles)

**Création d’une fonction**

f(x)

Renvoie la valeur de l’image de x par la fonction f.

**Appel à une fonction**

 **Arguments d’une fonction :** Une fonction peut ne pas avoir d’argument, et on écrit dans ce cas « def f() : » ou prendre plusieurs valeurs en arguments, par exemple « def milieu(xA,yA,xB,yB) : ».On peut fixer une valeur par défaut à un argument : « def f(x=0) : ». **Instruction return :** Une fonctionpeut ne pas renvoyer de résultat (pas de return) et on parle alors plutôt de processus. L’exécution d’un return interrompt l’exécution de la fonction. Ainsi, pour renvoyer plusieurs informations, il faut les associer (par exemple en les séparant par des virgules).
**Variables locales et globales :** En général, par défaut et en général, une fonction n’altère pas les valeurs des variables globales du programme et agit sur des copies des variables (à l’exception de certaines opérations, notamment sur les listes).

**Les boucles :**

**for** k **in range**(n) :
 |*instructions à réaliser*

Boucle effectuée pour k prenant les valeurs entières de 0 inclus à n exclu (0≤k<n donc n valeurs générées)

**Boucle bornée « Pour »**

**while** *condition* :
 |*instructions à réaliser*

Boucle effectuée tant que la condition (de type bool) est vérifiée.
NB : Pour créer une condition, voir Tests (Rappel : Il faut éviter les tests d’égalité sur les float)

**Syntaxe Python**

**Description**

**Boucle
non bornée « Tant que »**

Formation Python –2019 – Franck CHEVRIER

**Options du range :** range(n1,n2) permet de générer les entiers k tels que n1≤k<n2 ;
 range(n1,n2,p) permet de générer les entiers k tels que n1≤k<n2 avec un pas p entre chaque valeur.
**Compléments sur for :** range est une instruction permettant de générer des int, mais on peut aussi faire itérer la boucle for sur une liste prédéfinie avec d’autres typages : « for mot in [‘rouge’,’vert’,’bleu’] » ou « for lettre in ‘python’ ».

**Gestion des listes :**

**Syntaxe Python**

**Description**

L=[ ] L=[ 2 , -7 , 5 , 6.3 ]

Création d’une liste vide, création d’une liste contenant 4 valeurs

**Création**

L[k]
L[k]=v

Donne accès à la valeur de rang k de la liste L.

Affecte la valeur v au terme de rang k de la liste L.

**Accès à une valeur**

L**.append**(v)
L**.remove**(v)
L**.pop**(k)

 **append** : Place la valeur v à la fin de la liste L (ajout d’un élément).

 **remove** : Supprime la première apparition de la valeur v dans la liste L.

 **pop** : Supprime (et renvoie) la valeur de rang k de la liste L.

**len**(L)

Fonction renvoyant le nombre d’éléments de la liste L (len=length)

**Longueur**

**Ajout et suppression d’éléments**

v **in** L

Renvoie un booléen indiquant si la valeur v est dans la liste L.
Attention : Les rangs sont indexés à partir de 0 donc la valeur de rang k est la (k+1)ème valeur de la liste.

**Test**

L=[ ]
**for** k **in range**(11):
 L**.append**(k\*\*2)

 *est equivalent à*

L=**[** k\*\*2 **for** k **in range**(11) **]**
L=[ ]
**for** k **in range**(11):
  **if** k%2:
 L**.append**(k\*\*2)

 *est equivalent à*

L=**[** k\*\*2 **for** k **in range**(11) **if** k%2 **]**

**Création d’une liste en compréhension**

Création d’une liste à partir d’une liste vide, par extensions successives (ici, création de la liste des carrés des entiers de 0 à 10)
et son équivalent en syntaxe rapide.

Création d’une liste à partir d’une liste vide, par extensions successives sous condition
(ici, création de la liste des carrés des entiers k de 0 à 10 pour k impair)
et son équivalent en écriture de liste en compréhension.

Cette écriture est à rapprocher de l’écriture mathématique :

Attention : Les rangs sont indexés à partir de 0 donc la valeur de rang k est la (k+1)ème valeur de la liste.

 **Listes et tuples :**
La liste est un type d’objet de taille dynamique, contrairement à un élément de type tuple (équivalent d’un tableau), dont la taille est fixée. Ce type tuple peut être utile notamment pour stocker des coordonnées en dimension fixée (2 ou 3 par exemple), mais la valeur est ensuite non modifiable.
**Gestion des tuples :**

**Autres types de collections :**Il existe en Python d’autres types de collections que les listes, par exemple les dictionnaires (dict).
Mais dans les programmes officiels du lycée, il est indiqué : « Afin d’éviter les confusions, on se limite aux listes dans présenter d’autres types de collections ».

**Syntaxe Python**

**Description**

t = (2,-3,1) *ou* t = 2,-3,1

Création d’un 3-uplet contenant les trois valeurs 2, -3 et 1

**Création**

**Accès à une valeur**

t[k]
a,b,c = t

Donne accès à la valeur de rang k du tuple t.
Stocke dans a,b,c les valeurs du tuple t.

**Complément : Utilisation de modules / bibliothèques de fonctions**

**Installation d’une bibliothèque :**Il existe des bibliothèques de fonctions utiles, que l’on peut importer. Pour ajouter une bibliothèque à l’environnement Python, il faut exécuter une seule fois en console l’instruction « pip install *nom\_module* ».

**Utilisation d’une bibliothèque :**
Il convient d’importer en début de programme les fonctions du module à l’aide de l’instruction « from nom\_module import\* ». (\* signifie toutes les fonctions)
NB : On peut aussi importer les fonctions du module à l’aide de l’instruction « import nom\_module » mais les fonctions devront ensuite être appelées avec la syntaxe « nom\_module.nom\_fonction() », moins pratique à l’usage.
NB : Import sélectif : Si on ne souhaite récupérer que certaines fonctions ou variables d’un module, on peut également utiliser une syntaxe du type « from math import sqrt,pi » en listant les éléments souhaités.

 **Bibliothèques courantes :** math, random, matplot, scipy, …

