Syntaxes élémentaires en langage Python

Les principaux types de variables :

	Syntaxe Python	Description
int	a=-3 b=7	Nombres entiers
float	a=2/7 b=7.0 c=3.3	Nombres flottants (représentation des nombres décimaux)
str	text='' Hello prof ''	Chaînes de caractères (string)
bool	2+3==5 2.7>=5.5	Booléens (valeur True(=1) ou False(=0))
list	L1=[2,5,7] L2=[''salut'',3.8,8]	Liste d'éléments. NB : Les éléments de la liste peuvent être de tous types.

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
Affectation	a=7	Affectation de la valeur 7 à la variable a
Opérations élémentaires	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.
Tests	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**.

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

Création d'une fonction	**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)

Appel à une fonction	f(x)	Renvoie la valeur de l'image de x par la fonction f.

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 fonction peut 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 :

	Syntaxe Python	Description
Boucle bornée « Pour »	**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 non bornée « Tant que »	**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)

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
Création	L=[ ] L=[2 , -7 , 5 , 6.3]	Création d'une liste vide, création d'une liste contenant 4 valeurs
Longueur	**len**(L)	Fonction renvoyant le nombre d'éléments de la liste L (len=length)
Accès à une valeur	L[k] L[k]=v Attention : Les rangs sont indexés à partir de 0 donc la valeur de rang k est la (k+1)ème valeur de la liste.	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.
Test	v **in** L	Renvoie un booléen indiquant si la valeur v est dans la liste L.
Ajout et suppression d'éléments	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.
Création d'une liste en compréhension	L=[ ] **for** k **in range**(11): L**.append**(k**2) *est equivalent à* L=**[** k**2 **for** k **in range**(11) **]**	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.

	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 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 :
		$\{\ k^2\ \ / \ k\ entier\ ;\ 0 \leq k < 11\ ;\ k\ impair\ \}$

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 :

	Syntaxe Python	Description
Création	t = (2,-3,1) *ou* t = 2,-3,1	Création d'un 3-uplet contenant les trois valeurs 2, -3 et 1
Accès à une valeur	t[k]	Donne accès à la valeur de rang k du tuple t.
	a,b,c = t	Stocke dans a,b,c les valeurs du tuple t.

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 ».

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, …

```
from math import*
```

```
>>> log(2)
0.6931471805599453
>>> e
2.718281828459045
>>> pi
3.141592653589793
>>> cos(pi/6)
0.8660254037844387
>>> sqrt(2)
1.4142135623730951
>>> exp(3)
20.085536923187668
```

```
from random import*
```

```
>>> random()
0.05515514516853437
>>> randint(1,6)
5
```