

# Cours python\_190

Types	Opérateurs de base
<p><b>int</b> #nombre entier  <b>float</b> #nombre décimal  <b>str</b> #chaîne de caractères  <b>bool</b> #booléen (vrai ou faux)</p>	<p>Les opérateurs arithmétiques sont : +, -, *, /            Puissance : **            Division entière : //            Reste de la division entière : %</p> <p>== #Pour tester une égalité il faut utiliser un double égal</p> <p>!= #Pour tester une différence</p> <p>&lt;, &gt;, &lt;=, &gt;=</p> <p>Les opérateurs logiques :</p> <p><b>and</b> &amp;  <b>or</b>    <b>not</b> ~  <b>xor</b> ^</p> <p><b>Opérateur sur les chaînes de caractères</b></p> <p>'bon'+jour' #vaut 'bonjour'            'Ha'*3 #vaut 'HaHaHa'            'bon'&lt;'jour' #vaut True            'na'=='Ha' #vaut False            'ou' in 'jour' #vaut True            'on' not in 'bon' #vaut False</p>
Variables, affectation et expression	Types structurés
<p>Les noms de <b>variables</b> ne peuvent contenir que des signes alphanumériques (a-z, A-Z, et 0-9) et des underscore (blancs soulignés). Le nom ne peut pas commencer par un chiffre.</p> <p><b>L'affectation</b> attribue une valeur à une variable à l'aide du symbole =            Syntaxe : nom_variable=expression</p>	<p>Une fonction qui renvoie plusieurs valeurs les renvoie dans un tuple.</p> <p>Les listes :</p> <p>Initialisation : l=[val1, val2,...]            Accès : l[rang]            Ajout : l.append(valeur)            Suppression : del l[rang]            Modification : l[rang]=valeur            Taille : len(l)</p> <p>Compréhension de liste :            Liste=[expression for i in conteneur]</p> <p>Les dictionnaires :</p> <p>Initialisation : d={clé1 :val1, clé2 :val2,...}            Accès : d[clé]            Ajout : d[cléN]=valN (ajoute à la fin)            Suppression : del d[clé]            Modification : d[clé]=val            Taille : len(d)</p> <p>Compréhension de dictionnaire :            dico={k :v for k,v in liste de tuple}</p>

Instructions	Fonctions
<p><b>input</b>('texte permettant de savoir ce qui doit être entré') <i>#le résultat est une chaîne de caractères #affecter le résultat à une variable après avoir éventuellement modifié le type.</i></p> <p><b>print</b>('texte',variable) <i>#Le texte est entre guillemets, si il y a plusieurs éléments à afficher, les séparer par une virgule.</i></p> <p>Un <b>conteneur</b> est une variable d'un type structuré ou un intervalle défini par la fonction <b>range</b>            Syntaxe :  <b>range</b>(4) <i># conteneur d'entiers de 0 à 3</i>  <b>range</b>(d, n, p) <i>#conteneur d'entiers de d inclus à n exclu par pas de p.</i>  <b>range</b>(0,4,1) est identique à <b>range</b>(4)  <b>range</b>(d,n) est identique à <b>range</b>(d,n,1)</p> <p>Une <b>boucle bornée (for)</b> exécute un bloc un nombre prédéfini de fois en changeant la valeur d'une variable.            Syntaxe :  <b>for var in conteneur :</b> <i>#attention aux :</i>                <b>Bloc</b> <i>#attention à l'indentation</i></p> <p><b>if condition :</b> <i>#attention aux :</i>                <b>bloc</b> <i>#attention à l'indentation</i>  <b>elif condition :</b> <i>#C'est le sinon quand il y</i>                <b>bloc</b> <i>en a plusieurs</i>            :            :  <b>else :</b> <i>#C'est le dernier sinon</i>                <b>bloc</b> <i>Le else n'est pas obligatoire</i></p> <p>Une <b>boucle non bornée (while)</b> exécute un bloc tant que la condition est vérifiée.            Syntaxe :  <b>while condition :</b> <i>#attention aux :</i>                <b>bloc</b> <i>#attention à l'indentation</i></p>	<p>Une <b>fonction</b> permet de donner un nom à un bloc de code.            Syntaxe :  <b>def nom_fonction(para1,para2,...):</b> <i>#att aux :</i>                <b>corps</b> <i>#att indentation</i></p> <p>Pour des raisons de lisibilité, l'on peut indiquer le type des paramètres.            Syntaxe :  <b>def nom_fonction(para1 :int, para2 :float,...) :</b></p> <p>L'appel d'une fonction à la forme suivante :            Syntaxe :  <b>nom_fonction(exp1,exp2,...)</b></p> <p>para1, ... sont les paramètres : des noms de variables qui peuvent être utiliser dans le bloc corps, et qui sont initialisés par les expressions exp1, ... passées en arguments lorsque la fonction est appelée.</p> <p>La fonction <b>return</b> permet à une fonction de renvoyer un ou plusieurs résultats.</p> <p>Lorsque la fonction return est utilisée, l'on peut préciser le type de retour.            Syntaxe exemple :  <b>def nom_fonction(para1 :int, ...)-&gt;bool :</b></p> <p>Lorsqu'avec return, l'on renvoie plusieurs valeurs :  <b>return val1,val2,val3</b>            en réalité, c'est un <b>tuple</b> qui est renvoyé.            On accède aux valeurs de la façon suivante :            v1,v2,v3=fonction</p> <p>L'instruction <b>assert</b> permet de tester une fonction.            Syntaxe :  <b>assert fonction()==valeur</b>  <b>assert not fonction()</b> <i>#Résultat attendu False</i>  <b>assert fonction()</b> <i>#Résultat attendu True</i></p>
<b>Les méthodes</b>	
<p style="text-align: center;">Les méthodes sont des instructions que l'on appelle de la façon suivante :  <b>var.méthode</b>            La méthode est appliquée sur la variable var.</p> <p><b>append()</b> : permet d'ajouter à une liste ce qui est défini dans les parenthèses.  <b>items()</b> : permet d'obtenir la liste des tuples (clé, valeur) d'un dictionnaire.  <b>keys()</b> : permet d'obtenir la liste des clés d'un dictionnaire (l'ordre peut être modifié)  <b>values()</b> : permet d'obtenir la liste des valeurs d'un dictionnaire (l'ordre est le même que pour keys())  <b>reverse()</b> : permet de renverser une liste</p>	
<b>Bibliothèques</b>	
<p>Une bibliothèque contient des fonctions qui ne sont pas directement accessibles. Il faut importer celle-ci en début de programme pour y avoir accès.            Syntaxe :  <b>from turtle import*</b> <i>#ici, nous importons toutes les fonctions contenues dans la bibliothèque turtle.</i></p>	

Pour avoir accès à la liste de ces fonctions, il suffit de taper dans le shell :

```
import turtle
help(turtle)
```

Pour des raisons de capacités mémoires, l'on peut souhaiter de n'importer qu'une fonction contenue dans une bibliothèque. Il suffit de la spécifier. L'on peut également la renommer pour plus de lisibilité.

Syntaxe :

```
from bibliothèque import fonction as nouveau_nom
```

Exemple :

```
from math import sqrt as racine #si l'on souhaite importer plusieurs fonction, les séparer par une virgule.
```

<b>math</b>	<b>random</b>
sqrt() #racine carrée round() #arrondi floor() #arrondi inférieur ceil() #arrondi supérieur pi cos(), sin(), tan() degrees(), radians()	randint(a,b) #entier au hasard entre a et b au sens large random() #un réel au hasard entre 0 et 1 non compris
<b>turtle</b>	<b>nsi_ui</b>
<b>forward()</b> ou <b>fd()</b> #avance de ... pixels <b>backward()</b> ou <b>back()</b> #recule de ... pixels <b>left()</b> #tourne à gauche de ... ° <b>right()</b> #tourne à droite de ... ° <b>clear()</b> #efface <b>penup()</b> #lève le crayon <b>pendown()</b> #descend le crayon <b>setx()</b> #déplace le crayon horizontalement de ... pixels <b>sety()</b> #déplace le crayon verticalement de ... pixels	
<b>matplotlib.pyplot</b>	<b>fractions</b>
	Fraction(entier_a,entier_b) #est égale à la fraction simplifiée de a/b méthodes : numerator denominator exemple : fr.numerator #prend le numérateur de la fraction fr