1 Les bases de python3

Les types

Les différents types entiers, flottants, complexes, chaînes de caractères, boléens, listes, tuples et dictionnaires.

- ***entier*** : 3
- ***flotant*** : 2.3
- ***complexe*** : 1+2j
- ***chaînes de caractères (string)*** : 'ISN'
- ***Boléen*** : True
- ***Listes*** : [0,1,2,3]
- ***Tuples*** : (0,1,2,3)
- ***Dictionnaires*** : 'zero' :0, 'un' :1, 'deux' :2,'trois' :3
- ***Byte*** : b'toto'
- On aura toujours à l'esprit que $0.2 + 0.7 = 0.8999999999999999 \neq 0.9$
- ***int()*** permet de convertir, un nombre ou une chaîne de caractère en un entier.
- ***float()*** permet de convertir, un nombre ou une chaîne de caractère en un flottant.
- On utilise le caractère # pour écrire un commentaire

Les opérateurs

Opération	algorithme	python
Addition	2+3	2+3
Soustraction	12-5	12-5
Multiplication	3*6	3*6
Division	7/2	7/2
Quotient de la division euclidienne	7 div 2 ou div(7,2)	7//2
Reste de la division euclidienne	7 mod 2 ou mod(7,2)	7%2
puissance	7^2	7**2
racine carrée	$\sqrt{2}$ ou sqrt(2)	sqrt(2)

Variables

- Python utilise le symbole = pour affecter une valeur à une variable.
- Attention à ne pas confondre $a = 12$ et $a == 12$.

'Python3'

```
A=3; B=5
C='toto'
```

Entrées

- ***input("texte")*** permet de saisir du texte pour un programme.
- Il faudra éventuellement convertir ce texte dans le type voulu avec ***int()*** ou ***float()***

```
a=input("Entrer un texte a:  ")
b=int(input("Entrer un entier b:    "))
c=float(input("Entrer un réel c:    "))
```

Affichage

- On utilise de préférence la méthode ***format()*** pour afficher du texte et des variables.

```
print("La valeur de la variable a est",str(a),".") # si a est un nombre
print("La valeur de la variable a est " + str(a) + ".") # autre forme
print("La valeur de la variable a est {}.".format(a)) # je préfére
print("Le produit de {} par {} est {}".format(a,b,a*b))
print("{0}*{1}={2} et {0}/{1}={3}".format(a,b,a*b,a/b))
```

Connecteurs logiques

algorithmique	python
a = b	a==b
a≠b	a !=b
A et B	A and B A & B

algorithmique	python
A ou B	A or B A | B
A xor B	A ^ B
non A	not(A)

Condition SI

```
if x<=0:
    print('Pas de racines')
```

```
if x<=0:
    print('Pas de racines')
else:
    print('1 ou  2 racines')
```

```
if x<=0:
    print('Pas de racines')
elif x==0:
    print('1 racines double')
else:
    print('2 racines distinctes')
```

```
print("Saisissez deux valeurs numériques")
a=float(input("Saisir a: "))
b=float(input("Saisir b: "))
if a==b :
    print("Vous avez saisi deux fois la même valeur, à savoir {}.".format(a))
else :
    print("Vous avez saisi deux valeurs différentes {} et {}.".format(a,b))
```

Boucle pour

```
for i in range(4,21,2): # 4 6 ... 20
    print(i)
for i in range(6):      # de 0 à 5 sinon utiliser range(1,7) pour 1 2 ...6
    print(i)
```

La syntaxe générale est ***for i in range(m,n,p)*** :
i prend alors toutes les valeurs de ***m à n-1 par pas de p***

Tant que

```
i=1
while i<=5:
    print(i)
    i=i+1 #où en plus concis i+=1
#À la sortie de la boucle i=6
```

2 Les principales méthodes

2.1 Chaîne de caractères

- ***str.split() :*** Retourne une liste des mots de la chaîne, en utilisant *sep* pour séparateur.

```
In [28]: 'Pablo Neruda'.split(' ')
Out[28]: ['Pablo', 'Neruda']
```

Il existe aussi une variante avec *rsplit()*, (r pour right) la différence se produit lorsque le deuxième argument est spécifié.

```
In [29]: 'Pablo Neruda Meilland jean claude'.rsplit(' ',2)
Out[29]: ['Pablo Neruda Meilland', 'jean', 'claude']
In [30]: 'Pablo Neruda Meilland jean claude'.split(' ',2)
Out[30]: ['Pablo', 'Neruda', 'Meilland jean claude']
```

- ***str.join(iterable) :*** Retourne une chaîne qui est la concaténation de str dans iterable

```
In [16]: '-'.join('abcd')
Out[16]: 'a-b-c-d'
In [16]: ''.join(['a','e','i','o','u','y'])
Out[16]: 'aeiouy'
```

- ***len()*** compte le nombre de caractère de la chaîne.

```
In [20]: len('Pablo Neruda')
Out[20]: 12
```

- ***str.count(sub[, start[, end]]) :*** Retourne le nombre d'occurrences du caractère *sub* dans l'intervalle [*start* ;*end*].

```
In [4](): 'Pablo Neruda'.count('a')
Out[4](): 2
```

- ***str.find(sub[, start[, end]]) :*** Retourne l'indice le plus faible dans la chaîne où sub se trouve. Retourne -1 si sub n'est pas trouvé.

```
In [10]: 'Pablo Neruda'.find('a')
Out[10]: 1
In [11]: 'Pablo Neruda'.find('ll')
Out[11]: -1
```

- ***str.index(sub[, start[, end]]) :*** Comme find() , mais retourne une *ValueError* quand la sous-chaîne n'est pas trouvée.
- ***str.rfind(sub[, start[, end]]) :*** Retourne l'indice le plus élevé dans la chaîne où *sub* sous se trouve. Retourne -1 en cas d'échec.

```
In [26]: 'Pablo Neruda'.rfind('a')
Out[26]: 11
```

- ***str.replace(old, new[, count]) :*** Retourne une copie de la chaîne de toutes les occurrences de la chaîne *old* remplacées par de *new*

```
In [24]: 'Pablo Neruda'.replace('a','*')
Out[24]: 'P*blo Nerud*'
In [25]: 'Pablo Neruda'.replace('a','*',1)
Out[25]: 'P*blo Neruda
```

- ***sub in str :*** Retourne *True* si *sub* est une sous-chaîne de *str* False sinon.

```
In [12]: 'a' in  'Pablo Neruda'
Out[12]: True
```

2.2 Les tuples

- ***t.count(x) :*** Retourne le nombre d'occurrences de x dans le tuple t.

```
In [1]: t=(2,3,2,4,2)
In [2]: t.count(2)
Out[2]: 3
```

- ***t.index(x) :*** Retourne l'indice le plus faible dans le tuple t où x se trouve. retourne une *ValueError* quand x n'est pas trouvée.

```
In [3]: t=(2,3,4,2
In [4]: t.index(2)
Out[4]: 0
```

2.3 Les listes

- ***l.append(x) :*** Ajoute x à la fin de la liste.

```
In [1]: l=[1,2,3,4,5]
In [2]: l.append(6)
In [3]: l
Out[3]: [1, 2, 3, 4, 5, 6]
```

- ***l.clear()*** Efface la liste l *(depuis python3.3)*.

```
In [4]: l.clear()
In [5]: l
Out[5]: []
```

- ***l.extend (t) :*** concatène la liste l avec le contenu de la liste t sans réaffectation.

```
In [6]: t=(2,3,4,2)
In [7]: l1=[1,2,3]
In [8]: l2=[4,5,6]
In [9]: l1.extend(l2)
```

- ***l.insert(n,x) :*** insert x à la liste l à l'indice n.

```
In [10]: l=[1,2,3,4]
In [11]: l.insert(1,8)
In [12]: l
Out[13]: [1, 8, 2, 3, 4]
```

- ***l.pop ([i]) :*** affiche l'élément d'indice i et aussi supprime cet élément de l.

```
In [14]: l=[1,2,3,1]
In [15]: a=l.pop(2)
In [16]: a
Out[16]: 3
In [17]: l
Out[17]: [1, 2, 1]
```

- ***l.remove (x) :*** supprime le premier élément x de *s* et affiche une *ValueError* s'il ne le trouve pas.

```
In [9]: l=[1,2,3,1]
In [10]: l.remove(1)
In [11]: l
Out[11]: [2, 3, 1]
In [12]: l.remove(1)
In [13]: l
Out[13]: [2, 3]
In [14]: l.remove(1)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-14-ce3fdfde5d67> in <module>()
----> 1 l.remove(1)
ValueError: list.remove(x): x not in list
```

- ***l.reverse()*** inverse les éléments de l.

```
In [16]: l=[1,2,3]
In [17]: l.reverse()
In [18]: l
Out[18]: [3, 2, 1]
```

- ***l.sort()*** Trie les éléments de l.

```
In [19]: l=[4,6,2,1,0,5,3]
In [20]: l.sort()
In [21]: l
Out[21]: [0, 1, 2, 3, 4, 5, 6]
```