1 Pyzo

Ouvrez **Pyzo**. S'il vous le demande cliquez sur "use this environement".
Pour la première utilisation de pyzo il faut faire quelques réglages. Pour installer ou mettre à jour des modules python on utilise `pip` ou `conda`.
Dans le shell python de **Pyzo** exécuter les séquences suivantes.

➔ Installer *jupyter*.

- `pip install --upgrade pip`
- `conda install jupyter` Cela prend du temps car python est mis à jour.
- `conda install anaconda-client`
- `conda install -c conda-forge jupyter_contrib_nbextensions`
- `notebook --no-browser --NotebookApp.token=''`

➔ Télécharger un de mes cours : Ils se trouvent à l'adresse https://github.com/debimax/cours-debimax.
Sinon utilisez un moteur de recherche avec les mots clefs : *git debimax*

Sur mon git, deux fichiers sont important pour vous aider en *ISN*.

- /documents/algorithmique-python3.pdf qui regroupe l'essentiel de mes cours
- /documents/memento_python.pdf qui est comme son nom l'indique, mon mémento pour python.

Téléchargez ces deux fichiers (Cliquez sur leur nom puis sur download).
Télécharger aussi mon 1° cours, ***bases.ipynb*** (cliquez sur ***base.ipynb*** puis sur ***Raw*** et enregistrer sous).

➔ Ouvrir un fichier ***.ipynb***

- Première méthode : Dans votre gestionnaire de fichier cliquez sur le fichier ***bases.ipynb*** ➔ *Sélectionner un programme dans la liste des programmes installés* ➔ Parcourir et indiquez alors le chemin `C://Programmes/Miniconda/Scripts/jupyter-notebook`
- Deuxième méthode : Ouvrez pyzo.
  Dans la console pyzo tapez *notebook* (votre navigateur s'ouvre) ➔ Upload ➔ Indiquez le fichier bases.ipynb

  Petit problème car il s'est chargé sous le nom ***bases.ipynb.txt*** et il faut le renommer. Cocher alors le fichier puis sur renommer. On doit bien avoir ***bases.ipynb***.
  Il ne reste plus qu'à double cliquer sur le fichier ***bases.ipynb***.

➔ Ouvrez un onglet à l'adresse localhost:8888/nbextensions
Cochez les extensions ***Exercise2*** , ***Tabe of content (2)*** , ***Chrome Clipboard*** , ***Codefolding in Editor***

2 Jupyter notebook

Mes cours sont au format **ipynb**. Ces fichiers sont lisibles avec le programme **jupyter-notebook** (son ancien nom était **ipython-notebook**).
L'intérêt de ce format est que le fichier se lit avec un navigateur (firefox ou chrome) et que l'on peut exécuter du code python en même temps en appuyant sur :

➔ **CTRL+Entrée** : pour exécuter le code et rester sur la même cellule.
➔ **Maj+Entrée** : pour exécuter le code et passer à la cellule suivante.
➔ Les flèches *UP* et *DOWN* pour naviguer dans la page sans exécuter les cellules.
➔ Entrée : pour éditer la cellule
➔ Esc : pour sortir du mode édition.

Il existe aussi une version de jupyter en ligne : https://try.jupyter.org/

3 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
```

4 Les principales méthodes

4.1 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
```

4.2 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
```

4.3 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]
```

5 Toutes les méthodes

5.1 Chaîne de caractères

- ***str.capitalize() :*** Retourne une copie de la chaîne avec son premier caractère en majuscule et le reste en minuscule.

```
In [1]: 'pablo neruda'.capitalize()
Out[1]: 'Pablo neruda'
```

- ***str.casefold() :*** *Casefolding* est similaire à *lowercasing* mais plus agressif. Par exemple, la lettre minuscule allemand "ß" est convertie en "ss"

```
In [2]: 'Pablo Neruda'.casefold()
Out[2]: 'pablo neruda'
```

- ***str.center(width[, fillchar]) :*** place *str* au centre d'une chaine de longueur *width* et complète avec le caractère *fillchar* (par un espace si on ne précise pas le caractère).

```
In [3](): 'Pablo Neruda'.center(20,'-')
Out[3](): '----Pablo Neruda----'
```

- ***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.encode(encoding="utf-8", errors="strict")*** Retourne une version codée de la chaîne comme un objet *bytes*. L'encodage par défaut est "utf-8" .

```
In [6]: 'Pablo Neruda'.encode('utf8')
Out[6]:  b'Pablo Neruda'
```

- ***str.endswith(suffix[, start[, end]]) :*** Retour Vrai si la chaîne se termine avec le suffixe spécifié, sinon retourner faux. suffixe peut aussi être un tuple de suffixes à rechercher.

```
In [7]: 'Pablo Neruda'.endswith('lo')
Out[7]: False
In [8]:  'Pablo Neruda'.endswith('da')
Out[8]: True
```

- ***str.expandtabs(tabsize=8) :*** Retourne une copie de la chaîne où tous les caractères de tabulation sont remplacés par un ou plusieurs espaces, en fonction de la colonne courante et la taille de l'onglet donné. positions Tab se produisent tous les TabSize caractères.

```
In [9]: 'Pablo\tNeruda'.expandtabs()
Out[9]: 'Pablo       Neruda'
```

- ***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
```

Remarque : La méthode *find()* doit être utilisée que si vous avez besoin de connaître la position de *sub* . Pour vérifier si *sub* est une sous-chaîne ou pas, utilisez l'opérateur ***in*** :

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

- ***str.format() :*** Effectuer une opération de formatage de chaîne. La chaîne sur laquelle cette méthode est appelée peut contenir du texte littéral ou champs de remplacement délimités par des accolades {}.

```
In [13]: "1+2={0}".format(1+2)
Out[13]: '1+2=3'
```

- ***str.index(sub[, start[, end]]) :*** Comme find() , mais retourne une *ValueError* quand la sous-chaîne n'est pas trouvée.

```
In [10]: In [14]: 'Pablo Neruda'.index('a')
Out[10]: 1
In [11]: In [14]: 'Pablo Neruda'.index('y')
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-0a9111e44db1> in <module>()
----> 1 'Pablo Neruda'.index('y')

ValueError: substring not found
```

- ***str.isalnum() :*** Retourne vrai si tous les caractères de la chaîne sont alphanumériques (et s'il y a au moins un caractère), sinon false.
- ***str.isalpha() :*** Retourne vrai si tous les caractères de la chaîne sont alphabétiques (et s'il y a au moins un caractère), sinon false.
- ***str.isdecimal() :*** Retourne vrai si tous les caractères de la chaîne sont des caractères décimaux (et s'il y a au moins un caractère), sinon false
- ***str.isdigit() :*** Retourne vrai si tous les caractères de la chaîne sont des chiffres (et s'il y a au moins un caractère), sinon false.
- ***str.islower() :*** Retourne vrai si tous les caractères de la chaîne sont minuscules (et s'il y a au moins un caractère), sinon false.
- ***str.isnumeric() :*** Retourne vrai si tous les caractères de la chaîne sont des caractères numériques (et s'il y a au moins un caractère), sinon false.
- ***str.isprintable() :*** Retourne vrai si tous les caractères de la chaîne sont imprimables ou la chaîne est vide sinon, false.
- ***str.isspace() :*** Retourne vrai si il ya seulement des caractères blancs dans la chaîne (et s'il y a au moins un caractère), sinon false.
- ***str.istitle() :*** Retourne vrai si la chaîne est une chaîne titlecased (1^{er} lettre en majuscule) (et s'il y a au moins un caractère), sinon false.
- ***str.isupper() :*** Retourne vrai si tous les caractères de la chaîne sont en majuscules (et s'il y a au moins un caractère), sinon false.
- ***str.join(iterable) :*** Retourne une chaîne qui est la concaténation de str dans iterable

```
In [16]: '-'.join('abcd')
Out[16]: 'a-b-c-d'
```

- ***str*ljust()width[, fillchar])¶ :*** Retourne une chaîne justifiée à gauche dans une chaîne de longueur *largeur*. Le remplissage est fait en utilisant *FillChar* (par défaut un espace).

```
In [17]: 'pablo'.ljust(8,'*')
Out[17]: 'pablo***'
```

- ***str.lower() :*** Retourne une copie de la chaîne avec tous les caractères convertie en minuscules.

```
In [18]: 'Pablo Neruda'.lower()
Out[18]: 'pablo neruda'
```

- ***str.lstrip([chars]) :*** Retourne une copie de la chaîne avec les caractères de début supprimés. L'argument de caractères est une chaîne spécifiant le jeu de caractères à supprimer (par défaut espace, tabulation, newline etc...).

```
In [19]: '  \t \n  Pablo Neruda    '.lstrip()
Out[19]: 'Pablo Neruda    '
```

- ***str.maketrans(x[, y[, z]])¶ :*** Cette méthode statique retourne une table de traduction utilisable pour *str.translate()*.

```
In [18]: str='Pablo Neruda'
In [19]: intab = "aeiou"
In [H]: outtab = "12345"
In [20]: str.maketrans(intab, outtab)
Out[20]: {97: 49, 111: 52, 117: 53, 101: 50, 105: 51}
In [21]: print("97 est le code acsci de 'a'")
97 est le code acsci de 'a'
In [22]: str.translate(str.maketrans(intab, outtab))
Out[22]: 'P1bl4 N2r5d1'
```

- ***str.partition(sep) :*** Divise la chaîne à la première occurrence de *sep* , et retourner un 3-uplet contenant la partie avant le séparateur, le séparateur lui-même, et la partie après le séparateur.

```
In [23]: 'Pablo Neruda'.partition(' ')
Out[23]: ('Pablo', ' ', 'Neruda')
```

- ***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
```

- ***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
In [27]: 'Pablo Neruda'.rfind('y')
Out[27]: -1
```

- ***str.rindex() :*** Comme *rfind()* , mais elle retourne une *ValueError* quand la *sub* n'est pas trouvé.

```
```

- ***str.rjust(width[, fillchar]) :*** Retourne une chaîne justifié à droite dans une chaîne de longueur *width*. Le remplissage est fait en utilisant le spécifié *FillChar* (valeur par défaut est un espace).

```
In [24]: 'Neruda'.rjust(10,'*')
Out[24]: '****Neruda'
```

- ***str.rpartition(sep) :*** Divise la chaîne lors de la dernière apparition de *sep*, et retourne un 3-uplet contenant la partie avant le séparateur, le séparateur lui-même, et la partie après le séparateur.

```
n [25]: 'Neruda'.rpartition(' ')
Out[25]: ('', '', 'Neruda')
```

- ***str.rsplit(sep=None, maxsplit=-1) :*** Retourne une liste des mots de la chaîne, en utilisant *sep* pour séparateur. Utiliser *split()* de préfrence

```
In [26]: 'Pablo Neruda'.split(' ')
Out[26]: ['Pablo', 'Neruda']
```

- ***str.rstrip([chars]) :*** Retourne une copie de la chaîne avec les caractères de fin supprimés. L'argument de caractères est une chaîne spécifiant le jeu de caractères à supprimer.

```
In [27]: 'Pablo Neruda'.rstrip('da')
Out[27]: 'Pablo Neru'
```

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

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

La seule différence avec *rsplit()* 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.splitlines([keepends]) :*** Retourne une liste de lignes dans la chaîne, la rupture au niveau des limites de ligne.

```
In [31]: 'Pablo\nNeruda'.splitlines()
Out[31]: ['Pablo', 'Neruda']
```

- ***str.startswith(prefixe[, start[, end]]) :*** Retourne **True** si la chaîne commence par le *préfixe*, **False** sinon. préfixe peut aussi être un tuple de préfixes à rechercher

```
In [32]: 'Pablo Neruda'.startswith('Pa')
Out[32]: True
```

- ***str.strip([chars]) :*** Retourne une copie de la chaîne avec les avant et arrière caractères supprimés. L'argument de chars est une chaîne spécifiant le jeu de caractères à supprimer (espace par defaut).

```
In [33]: '   Pablo Neruda   '.strip()
Out[33]: 'Pablo Neruda'
```

- ***str.swapcase() :*** Retourne une copie de la chaîne avec des caractères majuscules converties en minuscules et vice versa.

```
In [34]: 'Pablo Neruda'.swapcase()
Out[34]: 'pABLO nERUDA'
```

- ***str.title() :*** Retourner une chaîne où les mots commencent par une majuscule et les autres caractères sont minuscules.

```
In [35]: 'pablo neruda'.title()
Out[35]: 'Pablo Neruda'
```

- ***str.translate(map) :*** Renvoyer une copie où tous les caractères ont été remplacés par le dictionaire map obtenu avec *maketrans*.

```
In [36]: str='Pablo Neruda'
In [37]: intab = "aeiou"
In [38]: outtab = "12345"
In [39]: str.translate(str.maketrans(intab, outtab))
Out[39]: 'P1bl4 N2r5d1'
```

- ***str.upper() :*** Retourne une copie de la chaîne avec tous les caractères convertie en majuscules.

```
In [40]: 'pablo neruda'.upper()
Out[40]: 'PABLO NERUDA
```

- ***str.zfill(width) :*** Retourne à gauche de chaîne numérique rempli de zéros dans une chaîne de longueur *width*. La chaîne d'origine est renvoyé si la largeur est inférieure ou égale à len(s).

```
In [41]: '125'.zfill(5)
Out[41]: '00125'
```