Débuter la programmation en Python (Numworks)

Ressources pour la programmation en Python Numworks :

- https://www.numworks.com/fr/ressources/manuel/python/
- https://www.numworks.com/fr/ressources/python/premiers-pas/
- https://www.numworks.com/fr/ressources/python/activites/

Ce qui suit est la copie d'écran à peine transformée des six premières activités (consulter les corrections sur le site)

1. Opérateurs et notion de fonction

1) Découverte de quelques opérateurs

(a) Dans la console d'exécution, saisir `5**2` (au clavier (5) (x) (x) (2)) puis `2**3`. A quoi correspond l'opérateur `**`?

(b) Dans la console d'exécution, saisir `4//2` (au clavier (4) (/) (/) (2)) puis `9//3` et enfin `5//2`. A quoi correspond l'opérateur `//` ?

Aide : Essayer d'autre valeurs

(c) Dans la console d'exécution, saisir `4%2` (au clavier (Toolbox) puis **Catalogue**) puis `9%3` et enfin `5%2`. A quoi correspond l'opérateur `%`

Aide : Essayer d'autres valeurs

2) Découverte de la notion de fonction en programmation

(a) Nous allons commencer par créer notre premier script nommé `activite1.py`. Voici une fonction nommée `double` qui prend en entrée un nombre et qui renvoie son double.

La saisir dans le script `activite1.py` puis faire **Exécuter le script** dans le menu à droite du titre du script. La fonction ainsi créée est disponible dans le menu de la touche (var). Appeler cette fonction avec différentes valeurs, par exemple `double(2)`, `double(5)`, ...

```
deg                PYTHON
from math import *
def double(a):
  d=a*2
  return d
```

(b) Ecrire dans le même script une fonction que vous nommerez `carre` qui prend en entrée un nombre et qui renvoie sa valeur au carré.

(c) Ecrire dans le même script une fonction que vous nommerez `cube` qui prend en entrée un nombre et renvoie sa valeur au cube.

2. Fonctions imbriquées et à plusieurs arguments

1) Découverte d'une fonction à deux arguments

(a) Nous allons commencer par créer un nouveau script nommé `activite2.py`. Voici une fonction nommée `maximum` qui prend en entrée deux nombres et qui renvoie le maximum des deux valeurs.

La saisir dans le script `activite2.py` puis faire **Exécuter le script** dans le menu à droite du titre du script. La fonction ainsi créée est disponible dans le menu de la touche (var). Appeler cette fonction avec différentes valeurs, par exemple `maximum(12,15)`, `maximum(6,-8)`, ...

```
deg                PYTHON
from math import *

def maximum(a, b):
  if a > b:
    return a
  else:
    return b
```

(b) Ecrire dans le même script une fonction que vous nommerez `minimum` qui prend en entrée deux nombres et qui renvoie le minimum des deux valeurs.

2) Découverte d'une fonction à plus que deux arguments

(a) Ecrire dans le même script une fonction que vous nommerez `maximum3` qui prend en entrée trois nombres et qui renvoie le maximum des trois valeurs.

(b) Ecrire dans le même script une fonction que vous nommerez `maximum4` qui prend en entrée quatre nombres et qui renvoie le maximum des quatre valeurs.

Aide : Vous pouvez utiliser votre fonction `maximum` et vous appuyer sur un schéma

(c) Ecrire dans le même script une fonction que vous nommerez `maximum8` qui prend en entrée huit nombres et qui renvoie le maximum des huit valeurs en utilisant la fonction maximum4.

Aide : faire un schéma avec le résultat de chaque appel aux fonctions

3. Instruction conditionnelle

Au sein d'une fonction, il peut être intéressant d'exécuter des instructions sous certaines conditions.

1) If ... elif ... else

Par exemple, si vous souhaitez écrire une fonction `majorite(age)` qui renvoie `"Majeur"`/`"Mineur"` selon l'âge renseigné, il est nécessaire de séparer les cas où l'âge est supérieur à 18 et celui où il est inférieur.

En Python, un bloc d'instruction conditionnelle s'écrit en indiquant : `python if condition: instruction`

L'instruction n'est exécutée que si la condition suivant `if` est vérifiée. Les deux points servent à indiquer à Python que vous commencez un bloc d'instructions.
Pour notre exemple, on peut écrire :

```
def majorite(age):
  if age >= 18:
    return "Majeur"
```

La fonction ci-dessus répond donc le texte "Majeur" si l'âge indiqué entre parenthèses est supérieur ou égal à 18. Voir l'application ci-contre.

```
>>> from exemple import *
>>> majorite(25)
'Majeur'
```

Si l'on souhaite ajouter la possibilité de répondre `"Mineur"`, il faut ajouter un cas. On souhaite que la fonction réponde `"Majeur"` si l'âge est supérieur à 18 et `"Mineur"` sinon. Pour signifier ce SINON à Python, on utilise `else` suivi de deux points avec la même indentation que le premier `if`.

```
def majorite(age):
  if age >= 18:
    return "Majeur"
  else:
    return "Mineur"
```

En résumé :

- S'il n'y a qu'un seul cas à distinguer, on utilisera :
  `python if condition: instruction`
- S'il n'y a que deux cas à distinguer, on utilisera :
  `python if condition: instruction_1 else: instruction_2`
- S'il y a plus de deux cas, on utilisera `elif` pour ajouter des conditions :
  `python if condition_1: instruction_1 elif condition_2: instruction_2 elif condition_3: instruction_3 ...`

2) Les conditions

Voici la syntaxe des conditions dans Python :

Condition	Syntaxe Python
Si x est égal à y	`if x == y:`
Si x est différent de y	`if x != y:`
Si x est strictement supérieur à y	`if x > y:`
Si x est strictement inférieur à y	`if x < y:`
Si x est supérieur ou égal à y	`if x >= y:`
Si x est inférieur ou égal à y	`if x <= y:`

Si l'on souhaite vérifier deux conditions on utilise `and` entre les deux conditions.
Par exemple : `python if x == 1 and y > 0:`
Si l'on souhaite vérifier l'une ou l'autre des deux conditions, on utilise `or` entre les deux conditions. Par exemple :

`python if x >= 1 or x == 0:`

Appuyez sur la touche Toolbox de votre calculatrice pour faire apparaître un menu de raccourcis. Dans **Boucles et tests** vous trouverez des blocs d'instructions pré-remplis pour vous éviter d'écrire lettre par lettre au clavier.

3) Exercice

Écrire une fonction `vabsolue(x)` qui prend un réel en argument et renvoie sa valeur absolue.

4) Autre exercice

Écrire une fonction `mediane` qui prend une série de nombres de taille quelconque en argument et qui renvoie la médiane de la série.

4. Boucle bornée

La boucle `for` est communément utilisée lorsque l'on souhaite répéter plusieurs fois une même instruction.
Il faut connaitre le nombre de fois que vous souhaitez répéter l'instruction.
Vous aurez besoin d'une variable permettant de compter le nombre de fois que vous répétez l'instruction. En Python, on indique les valeurs que doit prendre cette variable "compteur" dans une liste.

```
for i in [1,2,3,4,5]:
  print("Hello!")
```

Dans cet exemple, la variable qui permet de compter est `i`. À la première iteration, `i` prend la valeur du premier élément de la liste, c'est-à-dire 1, et l'instruction est exécutée : on affiche `"Hello!"`. Une fois l'instruction exécutée, `i` prend la valeur du deuxième élément de la liste et l'instruction est à nouveau exécutée : on affiche `"Hello!"` une deuxième fois. Une fois la boucle terminée, on aura donc affiché cinq fois `"Hello!"`.

1) for i in range(n)

La plupart du temps, on n'écrira pas la liste des valeurs que prend `i`. On préfèrera utiliser `range(n)` qui génère la liste des n premiers entiers. Et pour afficher cinq fois `"Hello!"`, on écrira :

```
for i in range(5):
  print("Hello!")
```

ATTENTION `range(n)` génère la liste des n premiers entiers en commençant par 0 (et en finissant donc en n-1). Ainsi :

```
>>> range(5)
[0,1,2,3,4]
```

Vous pouvez aussi utiliser `range(n,m)` qui liste les entiers entre n et m-1 :

```
>>> range(1,5)
[1,2,3,4]
```

Enfin, la commande `range(n,m,p)` liste les entiers de n à m-1 par pas de p :

```
>>> range(1,10,2)
[1,3,5,7,9]
```

2) Exercice

Écrire une fonction `somme(n)` qui prend un nombre entier en argument et renvoie la somme des n premiers entiers $(1+2+3+\ldots+n)$.

3) Autre exercice

Écrire une fonction `puissance(x,n)` qui prend un réel x et un entier naturel n en argument et renvoie la puissance n-ième de x, x^a.

5. Boucle non bornée

On utilise généralement la boucle `while` lorsqu'on souhaite répéter un nombre de fois une même instruction et qu'on ne sait pas combien de fois cette instruction va être répétée.

On connait alors une condition d'arrêt, c'est-à-dire un test qui permet de savoir si l'instruction va être répétée ou non.
Par exemple, si on veut ajouter 1 à un nombre tant que ce nombre est inférieur à 50, on utilisera une boucle `while` :
"TANT QUE nombre < 50, ajouter 1 à nombre".

En Python, la structure de la boucle `while` est la suivante : `python while condition: instruction ]`

Dans notre exemple, pour un nombre initial égal à 8, on écrira donc :

```
deg PYTHON
nombre = 8
while nombre<50:
  nombre=nombre+1
```

A la fin de la boucle, on aura :

```
deg PYTHON
>>> from exemple import *
>>> nombre
50
```

En effet, lorsque nombre contient 49 la condition est toujours vérifiée et l'instruction s'exécute. `nombre` est donc incrémenté de 1 et vaut alors 50.
La condition n'est plus vérifiée et la boucle `while` se termine.

1) Exercice

Écrire une fonction `max_cube(n)` qui prend un entier naturel n en argument et qui renvoie le plus grand entier dont le cube est inférieur ou égal à n.

2) Autre exercice

Écrire une fonction `facteurs(n)` qui prend un nombre entier en argument et renvoie la liste des facteurs de sa decomposition en facteurs premiers.

6. Introduction aux listes en Python

Une liste en Python est une structure de données qui peut contenir n'importe quel type de données (nombre entier, nombre flottant, chaîne de caractères, liste). Les éléments d'une liste sont rangés et séparés par des virgules. Le premier élément de la liste L a pour rang 0 ; il est noté `L[0]`.

1) Déclaration et initialisation

Une liste L peut être simplement déclarée par l'instruction `L=list()` ou `L=[]`. Elle existe alors mais ne contient aucun élément. On peut aussi la déclarer et l'initialiser en écrivant explicitement ses éléments : `L=[2,3,5,7,11,13,17,19,23,29]`.

```
deg PYTHON
>>> L=7*[0]
>>> L
[0, 0, 0, 0, 0, 0, 0]
```

On peut encore déclarer et initialiser une liste avec N éléments égaux à 0 en écrivant `L=N*[0]`.

Il y a bien d'autres façons de déclarer et d'initialiser une liste, par exemple avec la fonction `range()` : l'instruction `L=list(range(1,10))` conduit à la liste `L= [1,2,3,4,5,6,7,8,9]`.

2) Accès et modification d'un élément d'une liste

Lorsqu'une liste L a été créée, on peut accéder à l'élément de rang R en écrivant `L[R]` et le modifier ; en commençant par la fin : `L[-1]` est le dernier élément, `L[-2]` l'avant-dernier.
On peut aussi accéder au rang d'un élément : `L.index(E)` renvoie le premier rang de l'élément E.

```
deg PYTHON
>>> L=[1,2,4,2,5]
>>> L.index(2)
1
>>> L.index(3)
  Last command
Error: object not in sequence
```

Attention : s'il n'y a pas l'élément ou le rang demandé, ces instructions provoquent une erreur !
On peut extraire une liste E d'une liste L à partir d'un rang d jusqu'à un rang f en écrivant `E=L[d:f]`. Si un des rangs est omis, les valeurs par défaut sont d=0 (le début) et f=len(t) (la fin).
Si par exemple, `L=[12,5,7]` alors `L[1:2]` renvoie `[5]`, `L[:2]` renvoie `[12,5]` et `L[1:]` renvoie `[5,7]`.

3) Ajout et suppression d'un élément dans une liste

On peut ajouter un élément E en dernière position dans une liste L en écrivant `L.append(E)`.
Après `L= []`, l'instruction `L.append(1.5)` conduit à avoir L= [1.5].
Si on écrit encore `L.append(2.1)`, on aura L=[1.5,2.1].
Pour supprimer le dernier élément d'une liste L, écrire `L.pop()`.
Cette instruction renvoie l'élément supprimé qui peut alors être utilisé.
`L.pop(R)` supprime et renvoie l'élément de rang R.
La méthode `insert(R,E)` insère l'élément E au rang R, en décalant tous les éléments suivants d'un rang. Si L=[1,2,3], l'instruction `L.insert(2,5)` insère l'élément 5 au rang 2 ce qui conduit à avoir L=[1,2,5,3].

```
deg PYTHON
>>> L=list()
>>> L.append(1.5)
>>> L
[1.5]
>>> L.append(2.1)
>>> L
[1.5, 2.1]
>>> L.pop()
2.1
>>> L
[1.5]
```

```
>>> L=[1,2,3,2,1,5]
>>> L.remove(2)
>>> L
[1, 3, 2, 1, 5]
>>> del L[1]
>>> L
[1, 2, 1, 5]
```

On peut aussi, avec `L.remove(E)`, enlever la 1ère occurrence de l'élément E de la liste L. L'instruction `del L[R]` supprime L[R], l'élément de rang R, de la liste L.

4) Longueur d'une liste

Très utile pour éviter les erreurs de rang, l'instruction `len(L)` donne la longueur de la liste L (son nombre d'éléments).

Avec la liste `J=["Lu","Ma","Me","Je","Ve","Sa","Di"]`, l'instruction `len(J)` renvoie 7. Si j'ai une liste de listes, par exemple `M=[[0,0],[0,1],[1,0]]`, l'instruction `len(M)` renvoie 3 (c'est la longueur de la liste M), par contre `len(M[0])` renvoie 2 (c'est la longueur de la liste M[0]).

5) Test de la présence d'un élément dans une liste

Le mot-clé `in` permet de tester la présence d'un élément dans une liste. `N in L` renvoie `True` si N est un élément de la liste L et `False` sinon. On peut tester si un élément N n'est pas dans la liste L en écrivant `N not in L`.

L'instruction `L.count(E)` donne le nombre d'occurrences d'un élément E dans une liste L.

```
>>> L=[2,3,5,11]
>>> 5 in L
True
>>> N=7
>>> if N not in L: L.append(N)
>>> L
[2, 3, 5, 11, 7]
```

6) Tri d'une liste

```
>>> L=[2, 3, 5, 11, 7]
>>> L.sort()
>>> L
[2, 3, 5, 7, 11]
```

Une liste L peut être triée dans l'ordre alphanumérique croissant avec l'instruction `L.sort()`. On obtiendra le tri dans l'ordre inverse en utilisant la méthode `reverse()`.

7) Parcours d'une liste

L'instruction `for x in L` : est une déclaration de boucle bornée qui donne successivement à x la valeur des éléments de la liste L. Si P=1, l'instruction `for x in [2,3,5,7] : P*=x` conduit à P=210 (la syntaxe `P*=x` condense `P=P*x`).

On peut parcourir une liste pour en créer une autre avec ce type de déclaration qui conduit à `L=[4,9,25,49]` :

`L=[n*n for n in [2,3,5,7]]`

```
>>> P=1
>>> for x in [2,3,5,7]:P*=x
>>> P
210
>>> L=[n**2 for n in [2,3,5,7]
>>> L
[4, 9, 25, 49]
```

On peut même effectuer un filtrage avec une condition `L=[n*n for n in [1,2,3,4,5] if n%2==1]` conduit à `L=[1,9,25]`.

8) Choix d'un élément aléatoire dans une liste

Cette possibilité est offerte par le module `random` qu'il faut donc importer au préalable par l'instruction `from random import *`.

Une liste L contenant au moins deux éléments, on en tire un N au hasard en écrivant `N=choice(L)`. Si `L=[1,10,100,1000]` alors `choice(L)` renvoie un élément au hasard de L : 1, 10, 100 ou bien 1000.

9) Exercice

Écrire une fonction `elimine(L)` qui supprime les doublons d'une liste : `elimine([1,2,2,1,2,1])` renvoie `[1,2]`.

10) Autre exercice

Écrire une fonction `cartes(N)` qui renvoie une main de N cartes prises au hasard dans un jeu de 32 cartes.
Par exemple, avec `cartes(2)`, on devrait pouvoir obtenir `[As de Trefle, Dix de Pique]`.

Compléments :
Pour obtenir un cours plus complet sur Python (pas forcément Numworks) :

- Mon cours de NSI sur mathadomicile : http:/ph.moutou.free.fr/1ereNSI/Chap2_Cours.pdf
- « mes notes sur Python » qui se trouvent sur la page « Python » du site mathadomicile : http:/ph.moutou.free.fr/Python/coursPython.pdf
  Sur cette page, il y a des exercices avec corrections, mais ils ne sont pas vraiment basiques...

- Le cours de Gérard Swinnen « Apprendre à programmer avec Python 3 » (467 pages) est téléchargeable à l'adresse « inforef.be/swi/download/apprendre_python3.pdf »

- L'ancien site du zéro qui s'appelle maintenant Openclassroom « Apprenez à programmer en Python » (leur cours imprimable est à l'adresse https://openclassrooms.com/fr/old-courses-pdf)