LE LANGAGE PYTHON

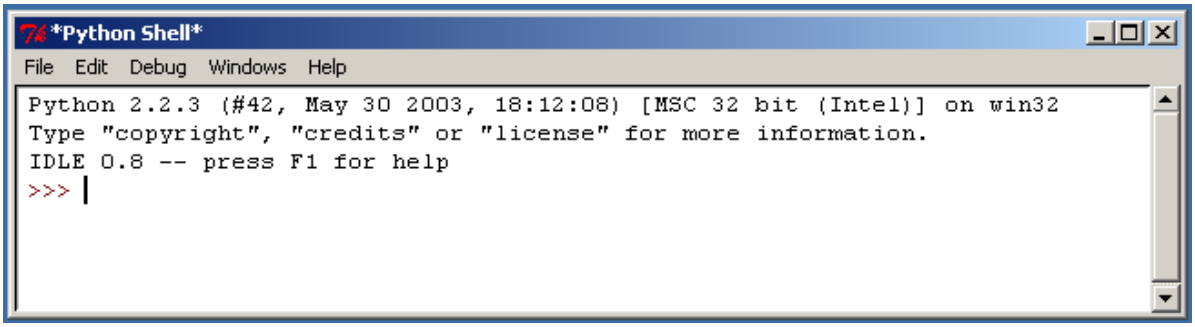
(version 3.1)

INITIATION AU LANGAGE PYTHON
séance n°1

L'environnement de travail

Python présente la particularité de pouvoir être utilisé de plusieurs manières différentes. Vous allez d'abord l'utiliser ***en mode interactif***, c'est-à-dire d'une manière telle que vous pourrez dialoguer avec lui directement depuis le clavier. Cela vous permettra de découvrir très vite un grand nombre de fonctionnalités du langage.

Dans un second temps, vous apprendrez comment créer vos premiers programmes (scripts) et les sauvegarder sur disque dur ou sur clef USB. Si vous utilisez une interface graphique telle que *Windows*, *Gnome, WindowMaker* ou *KDE*, vous préférerez vraisemblablement travailler dans une « fenêtre de terminal », ou encore dans un environnement de travail spécialisé tel que ***IDLE***. Voici par exemple ce qui apparaît dans une fenêtre de Windows :



Les trois caractères « supérieur à » constituent le signal d'invite, ou *prompt principal*, lequel vous indique que Python est prêt à exécuter une commande.

Calculer avec Python

Vous pouvez utiliser l'interpréteur comme une simple calculatrice de bureau. Testez par exemple les commandes suivantes dans l'environnement ***IDLE*** :

```
>>> 15+3

>>> 2 - 17                    # les espaces sont facultatifs

>>> 8 + 3 * 4                 # la hiérarchie des opérations mathématiques
                              # est-elle respectée ?
>>> (8+3)*4

>>> 20 / 3
```

Comme vous pouvez le constater, les opérateurs arithmétiques pour l'addition, la soustraction, la multiplication et la division sont respectivement +, -, * et /. Les parenthèses sont fonctionnelles.

Affectation-réaffectation

Nous avons vu en cours l'opération **d'affectation**, qui consiste à donner à une *variable* une valeur (nombre entier, réel, chaîne de caractères, etc...).

En pseudo-langage, l'opération d'affectation a été symbolisée par le sigle "←"
Exemple : a←5 # J'affecte à la variable a la valeur 5

En langage Python l'opération d'affectation s'écrit « = ».

Testez par exemple les commandes suivantes :

```
>>> a=3
>>>a

>>>a+10

>>>a/2
```

Ne croyez pas que cette affectation soit définitive. On peut **réaffecter** à la variable « a » une nouvelle valeur en réutilisant le symbole « = ».

```
>>>a                #Quelle est la valeur actuelle de a ?

>>>a=a+5            #On réaffecte à a sa valeur précédente augmentée de 5
>>>a

                    #Que vaut a maintenant ?
```

Remarque : le signe « = » utilisé ici n'a absolument aucun rapport avec l'égalité au sens mathématique du terme.

Python a une convention :si ce que l'on écrit est vrai, il renverra **true** en réponse ; si c'est faux, il renverra **false**.

Par exemple, testez les commandes suivantes (en vous souvenant bien de la dernière valeur affectée à la variable a).

```
>>> a<1

>>>a>2

>>>a==8              #le sigle « == » est l'égalité classique, au sens mathématique usuel.

>>>a!=4              #le sigle « != » signifie « différent ».
```

Que va renvoyer l'instruction a>=4 ?

Synthèse partielle : Quelle différence fondamentale faîtes-vous entre l'écriture a=10 et a==10 ?

Typage des variables

La variable a que nous avons rencontrée jusqu'ici appartient à une seule catégorie : celle des entiers. En anglais, cela se dit « Integer ».

Testez les commandes suivantes :

```
>>>a

>>>type(a)          #Quel est le résultat affiché ? Signification ?

>>>b=2.3            #Le séparateur décimal est le point, pas la virgule.

>>>type(b)          #Quel est le résultat affiché ? Signification ?

>>>c="Salut"

>>>type(c)

>>>d=[20,35,87,12]

>>>type(d)

>>>e=[5, "coucou",32.5,"Hugh"]          #Mélange de types dans un autre ?

>>>type(e)
```

Ces variables appartiennent à des **types** très différents. Nous verrons dans quelle mesure les utiliser séparément, puis ensemble. Rappeler ces types :

Opérations sur les variables

Commençons par réaffecter à chacune des valeurs a,b c et d d'autres valeurs.
Testez les commandes suivantes:

```
>>>a=2
>>>b=5.5
>>>c="Hello "          #avec l'espace entre Hello et le guillemet de fin.
>>>d="vous !"

>>>a+b

>>>c+d

>>>print(c+d)

#Quelle différence remarquez-vous entre les deux commandes précédentes ?
>>>a+c                #On ne mélange pas les genres!

>>> a*2

>>>a**2
```

```
>>>a**3

>>>a**4

>>>a**5
```

Que signifie l'opération « ** » entre un nombre et un entier ?

```
>>>c*2

>>>c*3
```

Même question entre une chaîne de caractères et un entier non nul pour « * ».

<u>Exercices</u> :

1. Signalons au passage la disponibilité de l'opérateur ***modulo***, représenté par le symbole **%**. Cet opérateur fournit ***le reste de la <u>division entière</u>*** d'un nombre par un autre. Essayez par exemple :
   >>> 10 % 3 (et notez ce qui se passe !)
   >>> 10 % 5

   Cet opérateur vous sera très utile plus loin, notamment pour tester si un nombre **a** est divisible par un nombre **b**. Il suffira en effet de vérifier que **a % b** donne un résultat égal à zéro.

   Vérifier à l'aide de la commande modulo si le nombre 33294 est divisible par 3 puis par 17.

2. Écrire un programme qui permet d'échanger la valeur de 2 nombres a et b indépendamment des valeurs affectées à a et à b au départ.

Récapitulatif

A faire sur feuille, sans ordinateur

I) Affectation

Définition : La notion **d'affectation** (ou d'assignation) désigne l'opération par laquelle on établit un lien entre le nom de la variable et sa valeur (son contenu). On l'a noté en langage usuel avec le symbole ←

Question 1 : Comment se note-t-elle en Python ?

Question 2 : Comment fait-on pour affecter à une variable a la valeur 3 ?

Question 3 : Cette affectation est-elle définitive ? Comment fait-on pour lui ré-affecter une autre valeur ?

Question 4 : Quelle est la signification du symbole « == » en Python ?

Question 5 : On écrit les lignes de commande suivantes. Quel résultat va renvoyer l'ordinateur ?

```
>>>a=5

>>>a

>>>a==3

>>>a==5

>>>a=18

>>>a<20

>>>a>10

>>>a=a+6

>>>a

>>>b=30

>>>b>a

>>>a+b
```

Question 6 : On écrit les lignes de commande suivantes. Quel résultat va renvoyer l'ordinateur ?

```
>>>a=10
>>>b="salut"
>>>a+b
```

Même question avec :

```
>>>a=10
>>>b= 5.2
>>>a+b
```

II) Typage des variables

Question 7 : Rappeler les différents types de variables rencontrés.

Question 8 :
a) Comment assigne-t-on à une variable a le type string ? (chaîne de caractères)

b) Que signifie l'opération **concaténer** deux chaînes de caractères ?

c) Que va renvoyer les commandes suivantes ?
```
>>>a="Bonjour "
>>>b="vous !"
>>>a+b

>>>print(a+b)
```

Question 9 : Que va renvoyer la commande suivante ?
```
>>> a="Oui "
>>>a*3
```

Question 10 : On admet que le premier élément d'une liste LIST se note LIST[0], le second LIST[1], etc...
On se donne une liste par la commande suivante :
```
>>>LIST=[5,48,-57,"ABC",451.9,"soleil",-789]
```
Que va renvoyer les commandes suivantes ?
```
>>>LIST[0]

>>>LIST[2]

>>>LIST[5]
```

Pour un complément sur la notion d'affectation avec Python, se référer à la fiche : **Plus sur l'affectation**.

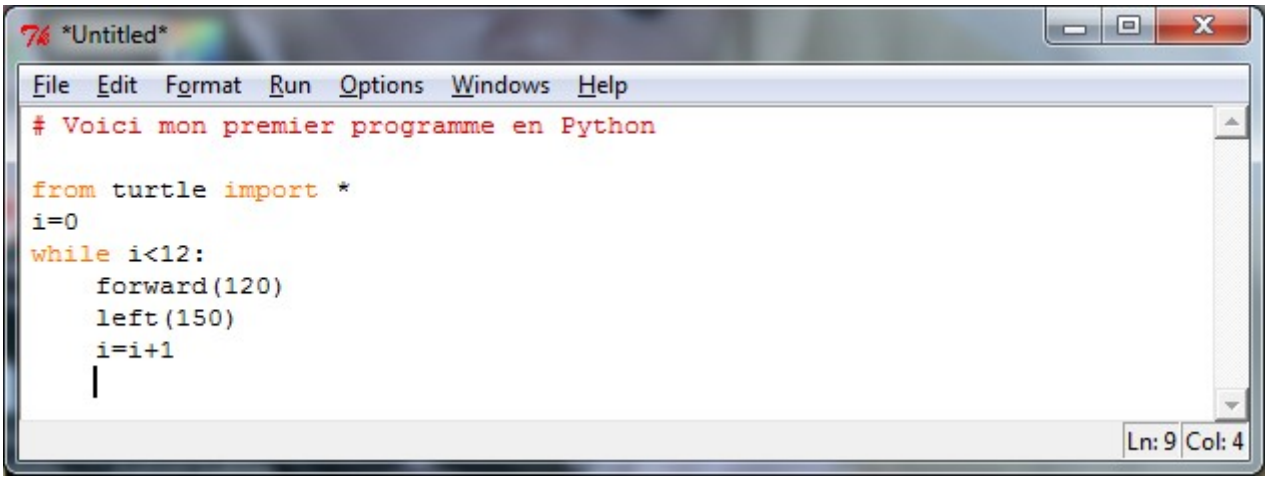
INITIATION AU LANGAGE PYTHON
séance n°2

Objectifs : savoir écrire un script en Python et l'enregistrer pour une utilisation ultérieure. Travailler la notion de boucle.

Consignes : La dernière fois, nous avons utilisé Python en mode interactif grâce à l'interface IDLE (Python Gui). De cette manière, nous avons été en mesure d'apprécier « en direct » quelques outils néanmoins fondamentaux : ***affectation*** d'une valeur à une ***variable***, les différents ***types*** de variables : int, float, string, list... Mais dès lors que nous fermons la fenêtre dans laquelle nous avons travaillé, toutes les données sont irrémédiablement perdues.
Au cours de cette séance, vous allez apprendre comment écrire un script et l'enregistrer pour une éventuelle ré-utilisation dans le futur.

Exercice 1 : votre premier programme
Remarque : Tout ce qui suit le signe dièse : # est considéré par Python comme un commentaire. Cela n'apparaîtra pas dans le script final et n'est destiné qu'à votre intention ou à celle des lecteurs du programme. Il sera ESSENTIEL que vous commentiez vos programmes. D'une cela facilite leur compréhension, et de deux cela vous aidera à les reprendre plus tard si vous ne les avez pas terminé.

1. Rechercher le logiciel Python dans menu démarrer>programmes>Python 3.1>IDLE(Python GUI)
2. Ouvrir une nouvelle fenêtre à l'aide de l'onglet **file** puis **New window**. Il apparaît une fenêtre vierge attendant vos instructions.
3. Saisir alors **exactement** le texte ci-dessous. La mise en forme se fait automatiquement. En particulier **l'indentation** du bloc d'instructions suivant while est ESSENTIELLE. Elle se fera automatiquement sous IDLE.



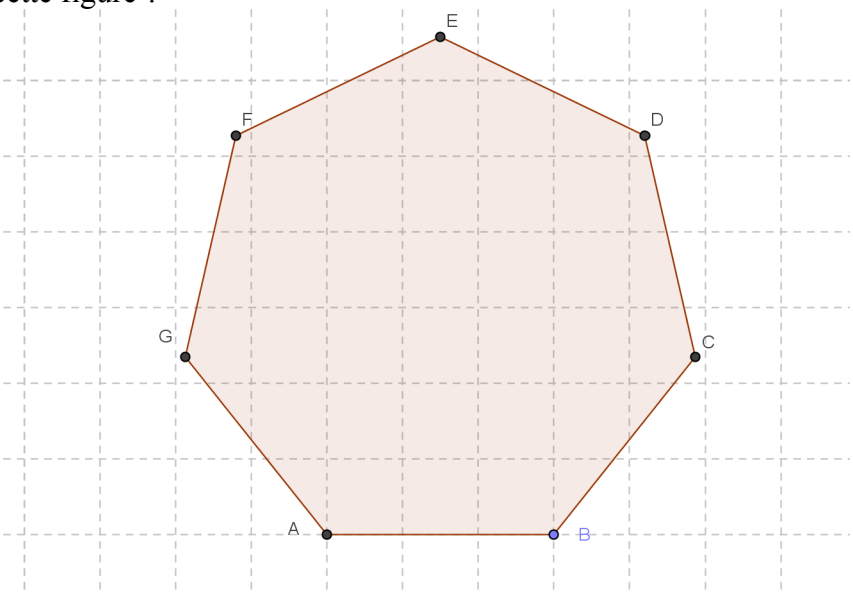
4. TRES IMPORTANT : enregistrez ce fichier sur votre clef USB dans le dossier TP Python en lui donnant le nom suivant : TP2_exercice1.py (l'extension .py est capitale)
5. Vous allez **compiler** le programme en appuyant simultanément sur les touches Ctrl et F5.
6. Alors, que se passe-t-il ?

Exercice 2

En créant une nouvelle feuille vierge grâce à l'instruction **file New Window**, répondez au problème suivant. Vous l'enregistrerez sous le nom : TP2_exercice2.py

Construire la figure suivante en utilisant l'instruction **while** (longueur d'un côté du quadrillage 50).

Quel nom porte cette figure ?



Quelques instructions pour obtenir les figures demandées.

from turtle import *	Permet d'importer les commandes pour les tracés.
reset()	On efface tout et on recommence
goto(x,y)	Aller à l'endroit(x,y)
forward(x)	Avancer de la distance x
backward(x)	Reculer de la distance x
up()	Relever le crayon (pour pouvoir avancer sans dessiner).
down()	Abaisser le crayon pour recommencer à dessiner.
left(x)	Tourner à gauche d'un angle en degrés égal à x.
right(x)	Tourner à droite d'un angle en degrés égal à x.
sqrt(x)	racine carré de x ($\sqrt{x}$)

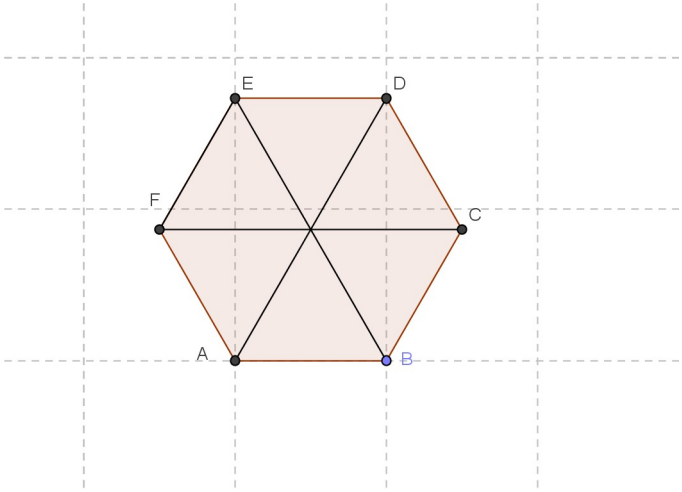
Instructions en Python	Commentaires
while condition**:** Bloc d'instructions	**while** signifie « **tant que** » Les « **:** » sont indispensables pour dire « faire » L'indentation après while est obligatoire.

Exercice 3

De même, après avoir écrit votre programme dans une nouvelle feuille vierge, enregistrez-le sur votre clef USB sous le nom : TP2_exercice3.py.

En utilisant toujours l'instruction while, soit une fois soit deux fois, obtenir la figure suivante (longueur d'un côté d'un petit carré : 50.

Cette figure **devra être construite en une seule fois**. On construira d'abord les côtés de l'hexagone puis les trois diagonales.



Exercice 4

Écrire un programme dans une nouvelle feuille vierge calculant et affichant le résultat de $S = 5+9+13+17+...+101+105$.

Vous l'enregistrerez sur votre clef USB sous le nom : TP2_exercice4.py

La commande pour afficher une variable S est : **print(S)**

Récapitulatif

Écrire une boucle « Tant que » avec Python

Syntaxe en langage usuel	Syntaxe en Python
Tant que condition vérifiée faire **bloc d'instructions**	**while condition vérifiée :** **bloc d'instructions**

Remarques :

- **Les : sont obligatoires après la condition associée à while.**
- **l'indentation du bloc d'instructions associé à while est obligatoire.**

Utilisation du module turtle

Toute utilisation des outils graphique du module turtle nécessite qu'on l'ait « appelé » par la commande : **from turtle import *** avant toute chose. On redonne les commandes usuelles pour tracer des figures :

reset()	On efface tout et on recommence
goto(x,y)	Aller à l'endroit(x,y)
forward(x)	Avancer de la distance x
backward(x)	Reculer de la distance x
up()	Relever le crayon (pour pouvoir avancer sans dessiner).
down()	Abaisser le crayon pour recommencer à dessiner.
left(x)	Tourner à gauche d'un angle en degrés égal à x.
right(x)	Tourner à droite d'un angle en degrés égal à x.

INITIATION AU LANGAGE PYTHON
séance n°3

Objectifs : Travailler la notion de boucle sur des exemples plus poussés.

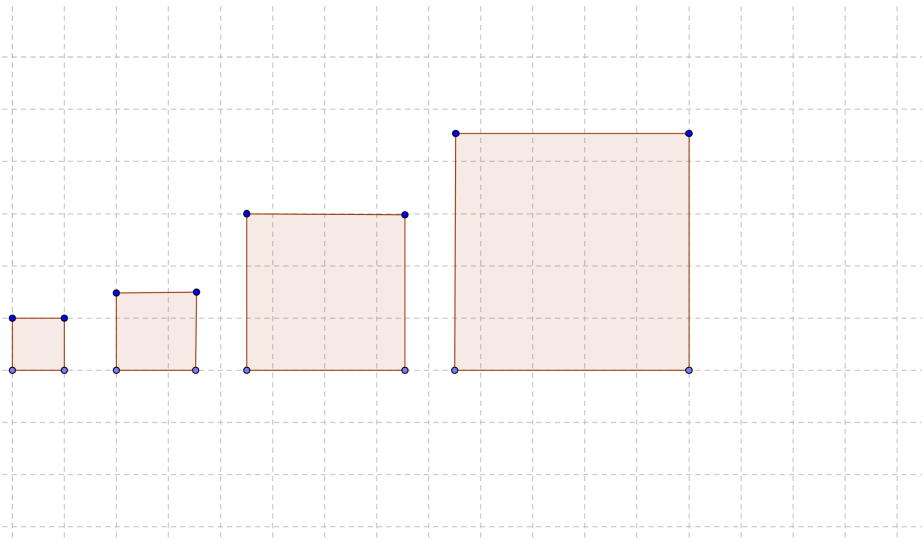
Consignes : on veillera dans cette séance à enregistrer soigneusement ses programmes pour une utilisation ultérieure dans certains scripts.

Exercice 1

1. Rappeler dans l'encart de texte ci-dessous un script écrit en Python qui permet de dessiner un carré de côté 10. Vous utiliserez une boucle **while**.

2. En modifiant ce script, en écrire un autre qui trace une série de dix carrés de côté 10. L'espace entre les carrés est de 10. Vous enregistrerez votre programme sous le nom : TP3_exercice1.py

3. En modifiant le script précédent, en écrire un autre qui trace une série de quatre carrés dont les côtés augmentent de 15 en 15. Le premier carré a un côté de 10 et l'espace entre chaque carré est de 10. Vous l'enregistrerez sous le nom : TP3_exercice1bis.py



Exercice 2

Écrire un script en Python qui trace une série alternée de carrés et d'hexagones espacés de 30. Les carrés comme les hexagones ont leurs côtés égaux à 20. Vous l'enregistrerez sous le nom TP3_exercice2.py.

Exercice 3

Considérons une liste de nombres entiers ou flottants :
a=[20,3.5,44, 37, 151, 28,15.5]

Le premier élément de la liste se note a[0], le second a[1],etc...
Il existe une fonction prédéfinie en Python qui renvoie le nombre d'éléments d'une liste : c'est la fonction **len**. Dans le cas présent, si l'on tape len(a), l'ordinateur affichera 7.

Écrire un script qui :

- part d'une liste de nombres entiers ou flottants, par exemple on prendra la liste précédente.
- En parcourant la liste, renvoie au final la somme des éléments qui la constitue. Vous utiliserez une boucle **while** et la fonction **len**.

Vous enregistrerez votre programme sous le nom TP3_exercice3.py

Pour un complément sur la notion de saisie de données avec Python, se référer à la fiche : **Plus sur la saisie de données**.

INITIATION AU LANGAGE PYTHON
séance n°4

Objectifs : Travailler la notion d'instruction conditionnelle.

Consignes : on enregistrera également tous les programmes réalisés au cours de la séance.

Écrire une instruction conditionnelle « Si...alors » avec Python

Syntaxe en langage usuel	Syntaxe en Python
Si condition vérifiée faire **bloc d'instructions**	**if condition vérifiée:** **bloc d'instructions**

Remarques :
- **Les : sont obligatoires après la condition associée à if.**
- **l'indentation du bloc d'instructions associé à if est obligatoire.**

Exercice 1
Écrivez un programme qui affiche les 20 premiers termes de la table de multiplication par 7, en signalant au passage (à l'aide d'une astérisque) ceux qui sont des multiples de 3.
Exemple : 0 7 14 21* 28 35 42* 49 (penser à utiliser la commande modulo %)
Enregistrez votre programme sous le nom TP4_exercice1.py

Exercice 2
Écrivez un programme qui :
- demande à l'utilisateur de saisir une chaîne de caractères
- compte le nombre d'occurrences du caractère « g » dans cette chaîne.

Enregistrez votre programme sous le nom TP4_exercice2.py

Écrire une instruction conditionnelle « Si...alors...sinon » avec Python

Syntaxe en langage usuel	Syntaxe en Python
Si condition vérifiée faire **bloc d'instructions 1** **Sinon** faire **bloc d'instructions 2**	**if condition vérifiée:** **bloc d'instructions 1** **else:** **bloc d'instructions 2**

Remarques :
- **Les : sont obligatoires après la condition associée à if ainsi qu'après else.**
- **l'indentation du bloc d'instructions associé à if est obligatoire ; même chose pour celui associé à else.**
- **if et else sont indentés identiquement**

Exercice 3
Écrire un programme qui demande à l'utilisateur de saisir un entier naturel N.
Si N est un multiple de 5, il sera affiché « multiple de 5 », sinon ce sera « pas multiple de 5 »
Enregistrez votre programme sous le nom TP4_exercice3.py

Exercice 4
Écrire un programme qui demande à l'utilisateur de saisir un entier naturel N.
Si N est pair, le programme dessinera un carré de côté 50, sinon il dessinera un triangle équilatéral de côté 70.
Enregistrez votre programme sous le nom TP4_exercice4.py

Écrire une instruction conditionnelle « Si...Sinon si...sinon » avec Python

Syntaxe en langage usuel	Syntaxe en Python
Si (condition 1 vérifiée) faire bloc d'instructions n°1 **Sinon si (condition 2 vérifiée)** faire bloc d'instructions n°2 …etc... **Sinon si (condition N-1 vérifiée)** faire bloc d'instructions n° N-1 **Sinon** faire bloc d'instruction n° N	**if (condition 1 vérifiée):** bloc d'instructions n°1 **elif (condition 2 vérifiée):** bloc d'instructions n°2 ...etc... **elif (condition N-1 vérifiée):** bloc d'instructions n° N-1 **else:** bloc d'instruction n°N

Remarques :

- **elif signifie sinon si**
- **Les : sont obligatoires après la condition associée à if ainsi qu'après celles associées aux elif et à else.**
- **l'indentation du bloc d'instructions associé à if est obligatoire ; même chose pour celui associés aux elif et à else.**
- **if, elif et else sont indentés identiquement**
- **Toutes les conditions énoncées sont par construction incompatibles deux à deux.**

Exercice 5
Lors d'un sondage, une personne interrogée a le choix entre trois réponses :

- satisfait
- moyennement satisfait
- pas satisfait

Écrire un programme qui demande à l'utilisateur son avis parmi ces trois propositions dans l'ordre précité. Si la première réponse est donnée, il affichera "Merci ! Very Good" ; si c'est la seconde il affichera "Ok ok" sinon il affichera "Snif ! "

INITIATION AU LANGAGE PYTHON
séance n°5

Objectifs : Travailler la notion de chaîne de caractères et de liste.

Consignes : on enregistrera également tous les programmes réalisés au cours de la séance.

Les chaînes de caractères (type string)

Sous Python, une donnée de type ***string*** est une suite quelconque de caractères (texte ou même nombres) délimitée soit par des apostrophes (*simple quotes*), soit par des guillemets (*double quotes*).

Exemples :

```
>>> phrase1='Voilà un écrit '
>>> phrase2='bien intéressant'
>>> print(phrase1,phrase2)
Voilà un écrit  bien intéressant
```

A l'intérieur d'une chaîne de caractères, l'*antislash* « \ » permet d'insérer un certain nombre de codes spéciaux (sauts à la ligne, apostrophes, guillemets, etc.). On ne donne que l'exemple des apostrophes.

```
>>> phrase3='Comment s\'appelle-t-il ?'          # L'antislash est placé avant l'apostrophe
>>> print(phrase3)
Comment s'appelle-t-il ?
```

Python considère les chaînes de caractères comme une **collection ordonnée d'éléments**. Pour accéder à un élément de la chaîne, on saisit le nom de la variable et on lui accole entre crochets le numéro de l'élément considéré. Attention, la numérotation commence à zéro !

Exemples :

```
>>> mot='erg4ju6op'
>>> print(mot[0])
e
>>> print(mot[3])
4
```

Opérations sur les chaînes de caractères

1. La concaténation

Cette opération consiste à mettre bout à bout les éléments constitutifs de plusieurs chaînes de caractères. Elle se symbolise à l'aide de l'opération +

Exemple :

```
>>> mot1='Mais c\'est '
>>> mot2='merveilleux !'
>>> print(mot1+mot2)          # on concatène mot1 et mot2
Mais c'est merveilleux !
>>> print(mot2+mot1)
merveilleux !Mais c'est       # L'ordre est important.
```

2. Déterminer la longueur d'une chaîne

Cette opération s'effectue grâce à la fonction **len()**.
Exemple :

```
>>> mot="adseghbm55f"
>>> len(mot)
11
```

3. Convertir une chaîne de caractères en nombre

Exemple :

```
>>> nb1='54'
>>> nb1+5
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    nb+5
TypeError: Can't convert 'int' object to str implicitly
>>> int(nb1)+5                # on convertit la chaîne nb1 en entier grâce à la fonction int()
59
>>> nb2='20.36'
>>> float(nb2)+4              # on convertit la chaîne nb2 en flottant grâce à la fonction float()
24.36
```

Remarque : La chaîne qui ne contient aucun élément se note "" (rien entre les guillemets)

Exercice 1
Écrire un programme qui :

- demande à l'utilisateur de saisir une chaîne de caractères
- compte le nombre d'occurrences du caractère « a » dans cette chaîne.
- Si « a » n'apparaît pas, le programme le rajoute à la fin de la chaîne.

Enregistrez votre programme sous le nom TP5_exercice1.py

Exercice 2
Écrire un programme qui recopie une chaîne (dans une nouvelle variable) en l'inversant. Ainsi par exemple, « `bouchon` » deviendra « **`nohcuob`** ».
Enregistrez votre programme sous le nom TP5_exercice2.py

Les listes d'éléments (type list)

Sous Python, on peut définir une liste comme **une collection d'éléments (éventuellement de types divers) séparés par des virgules, l'ensemble étant encadré par des crochets.**

Exemple :

```
>>> maliste=['abc','hello',20,3.14,'coucou']
>>> print(maliste)
['abc', 'hello', 20, 3.14, 'coucou']
```

Remarque : Comme les chaînes de caractères, les éléments d'une liste sont numérotés en commençant à zéro.

Exemple :
```
>>> maliste=['abc','hello',20,3.14,'coucou']
>>> maliste[0]
'abc'
>>> maliste[2]
20
```

A retenir : l'indice d'une liste de *n* éléments commence à 0 et se termine à *n*-1

Opérations sur les listes

1. Concaténation et duplication

Comme pour les chaînes de caractères, l'opérateur + sert pour concaténer des listes et * pour les dupliquer.
Exemple :
```
>>> liste1=[325,'bonjour',65,69]
>>> liste2=['ha','gros',564]
>>> liste1+liste2
[325, 'bonjour', 65, 69, 'ha', 'gros', 564]
>>> liste2*3
['ha', 'gros', 564, 'ha', 'gros', 564, 'ha', 'gros', 564]
```

2. Ajouter ou supprimer un élément dans une liste

Python est un langage « orienté objet ». On dispose de ***méthodes*** spécifiques permettant d'effectuer certaines actions sur des objets. Nous considérerons ici qu'une liste est un objet.

- Ajouter un élément **à la fin** d'une liste

Exemple :
```
>>> maliste=[201,45.8,12]
>>> maliste.append(35)
>>> maliste
[201, 45.8, 12, 35]
```

On a appliqué la méthode **append()** à l'objet maliste avec l'argument 35. La syntaxe est : liste.append(objet)

- Ajouter un élément **à l'intérieur** d'une liste

Exemple :
```
>>> maliste=[201,45.8,12,35,89]
>>> maliste.insert(3,100)
>>> maliste
[201, 45.8, 12, 100, 35, 89]
```

La commande liste.insert(indice,objet) insère l'objet dans la liste avant l'indice précisé.

- Supprimer un élément dans une liste

Cette opération s'effectue grâce à la commande del liste[indice]

Exemple :

```
>>> maliste=[55,201,89,33]
>>> del maliste[1]
>>> maliste
[55, 89, 33]
```

Remarque : la liste vide se note []. La méthode append() est particulièrement adaptée pour construire une liste à partie d'une boucle.

La fonction **len()** renvoie également le nombre d'éléments d'une liste.

Exemple :

```
>>> maliste=[55,120,40,-8]
>>> len(maliste)
4
```

Exercice 3

Écrire un programme qui :

- demande à l'utilisateur de saisir une liste de nombres, par exemple [20,35,11,56,41,10].
- détermine le plus grand élément de cette liste,
- affiche cet élément.

Par exemple si l'utilisateur saisit [20,35,11,56,41,10] il s'affichera :
le plus grand élément de cette liste est 56
Enregistrez votre programme sous le nom TP5_exercice3.py

Exercice 4

Écrire un programme qui :

- demande à l'utilisateur de saisir deux listes,
- alterne les éléments de la première liste avec ceux de la seconde.

Par exemple si l'utilisateur saisit ['coucou','bonjour','ça va ?'] et ['Charles','Edouard','Au poil !'], il s'affichera : ['coucou','Charles','bonjour','Edouard','ça va ?','Au poil !']
Enregistrez votre programme sous le nom TP5_exercice4.py

INITIATION AU LANGAGE PYTHON
séance n°6

Objectifs : Initiation à la notion de fonction.

Motivation : De même qu'un organisme vivant (donc une structure biologiquement complexe) est constitué d'un assemblage de nombreuses cellules, un programme « compliqué » se structure en un assemblage de plus petits sous-programmes, chacun ayant un rôle bien défini. Nous allons découvrir au cours de cette séance, comment créer ces petits sous-programmes et les utiliser pour en construire un autre plus compliqué.
L'utilité de ces sous-programmes, que l'on appellera ***fonctions*** est primordiale lorsque l'on veut réaliser plusieurs fois la même opération au sein d'un même programme.

La syntaxe Python pour définir une fonction est la suivante :

```
def   nom_de_la_fonction(liste de paramètres):

        Blocs d'instructions
```

- Vous pouvez choisir n'importe quel nom pour la fonction que vous créez, à l'exception des mots réservés du langage, et à la condition de n'utiliser aucun caractère spécial ou accentué (le caractère souligné « _ » est permis).
- Comme les instructions **if** et **while** que vous connaissez déjà, l'instruction **def** est une *instruction composée*. La ligne contenant cette instruction se termine obligatoirement par un double point, lequel introduit un bloc d'instructions que vous ne devez pas oublier d'indenter.
- La ***liste de paramètres*** spécifie quelles informations il faudra fournir en guise d'arguments lorsque l'on voudra utiliser cette fonction (Les parenthèses peuvent parfaitement rester vides si la fonction ne nécessite pas d'arguments). (*Gérard Swinnen. Programmer avec Python*)

Tout en suivant les instructions, vous répondrez sur cette feuille aux questions posées au cours des exercices.

Exercice 1
Pour le moment nous allons utiliser Python en mode interactif. Ouvrez un fichier vierge via : **menu démarrer>tous les programmes>Python 3.1>IDLE (Python GUI).** Recopiez ensuite le script ci-dessous et tapez deux fois entrée. Le prompt de commande >>> apparaît. Écrire alors table5()

```
>>> def table5():                          # Pas de paramètres en argument ici
        i=0                                # Remarquez l'indentation du bloc d'instructions de def
        while i<11:
                print("5 *",i," = ",5*i)   # Remarquez l'indentation du bloc d'instructions de while
                i=i+1


>>> table5()
                        # Que s'affiche-t-il à l'écran ?
```

Analyse :

- L'intérieur des parenthèses est vide. Dans cette fonction, nous n'avons utilisé aucun paramètre, c'est-à-dire que l'utilisateur ne se donne pas le choix de faire varier par exemple le choix de la table : 5 est fixé.
- Bien remarquer l'indentation après l'instruction **def** suivie de deux points. Elle est obligatoire.

Exercice 2

On aimerait pouvoir écrire n'importe quelle table d'entiers. Encore faut-il le choisir ! Pour cela, on mettra celui-ci en paramètre. A la suite de ce que vous avez saisi précédemment, écrire le script ci-dessous :

```
>>> def table_d_entiers(n):
        i=0
        while i<11:
            print(n," * ",i," = ",n*i)
            i=i+1
```

Appuyez deux fois sur la touche Entrée. Le prompt de commande >>> réapparait.

Saisir :

```
>>>table_d_entiers(4)
                    # Que s'affiche-t-il à l'écran ?
```

Saisir :

```
>>>table_d_entiers(8)
                    # Que s'affiche-t-il à l'écran ?
```

Analyse :

- Le fait d'avoir mis un entier n en paramètre (à l'intérieur des parenthèses) permet d'afficher la table de multiplication de l'entier n. Il suffit de remplacer n par la valeur choisie en appelant la fonction.
- On aimerait néanmoins demander à l'utilisateur de saisir n à l'écran pour qu'ensuite la fonction soit appelée et affiche la table de multiplication de n. On séparera donc en deux la construction de la fonction et le programme principal.

Exercice 3

A partir de la fenêtre présentement ouverte, ouvrir une fenêtre vierge via **file>New Window**.

Saisir ensuite le script qui suit :

```
def table_d_entiers(n):                    # Construction de la fonction
    i=0
    while i<11:
        print(n," * ",i," = ",n*i)
        i=i+1

# programme principal

n=int(input("Saisissez un entier "))
table_d_entiers(n)
```

1. Enregistrez ce programme sous le nom : TP6_exercice3.py
2. Appuyez sur Ctrl + F5 simultanément pour le faire fonctionner. Vous choisirez n'importe quelle valeur de n pour le tester.

Remarque : Tout programme en Python un peu complexe prendra nécessairement la forme suivante :

Liste de fonctions
il peut n'y en avoir qu'une
Programme principal
on utilise les fonctions précédentes

Exercice 4

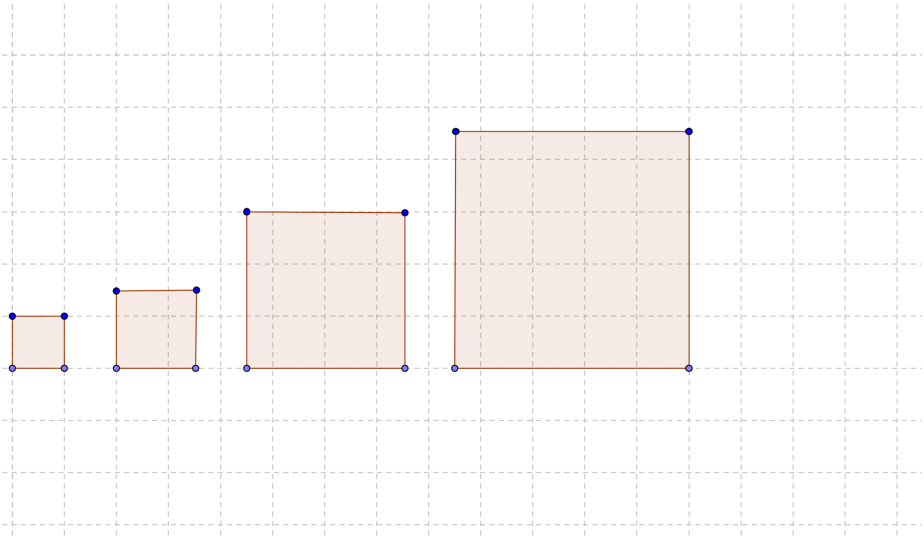
Ouvrir une autre fenêtre vierge via **file>New Window**

1. Écrire une fonction **carre(taille)** permettant de dessiner un carré de côté taille.
2. Écrire le programme principal à la suite. Celui-ci :
   a) demandera à l'utilisateur de saisir la taille du côté d'un carré
   b) dessinera 5 carrés de la taille choisie, l'espace entre les carrés étant de 10.
3. Enregistrez le programme sous le nom TP6_exercice4.py
4. Le tester en appuyant sur Ctrl+F5 simultanément.

Exercice 5

Ouvrir une autre fenêtre vierge via **file>New Window**

1. Copier le script de la fonction carré précédente.
2. Écrire le programme principal à la suite. Celui-ci :
   a) demandera à l'utilisateur de saisir la taille du côté du carré initial.
   b) construira une suite de 4 carrés dont les côtés s'accroissent de 15 à chaque fois.
   c) sera tel que l'espace entre les carrés est de 10.
3. Vous enregistrez ce programme sous le nom TP6_exercice5.py et le testerez.



Exercice 6

Écrire un programme utilisant deux fonctions carre(taille) et hexagone(taille) permettant de dessiner le motif suivant. Les carrés comme les hexagones ont leur côtés égaux à 20 et l'espace entre chaque figure est de 30. Vous l'enregistrerez sous le nom TP6_exercice6.py et le testerez.

Exercice 7

En créant deux fonctions pentagone et triangle, écrire un programme qui dessine une alternance pentagone (de côté 15, de couleur bleue) et triangle équilatéral (sommet vers le bas, de côté 25, de couleur rouge) 5 fois de suite. L'espacement entre la base d'un pentagone et d'un triangle est de 20. On partira du point de coordonnées (-100,0). Vous l'enregistrerez sous le nom TP6_exercice7.py et le testerez.

Liste de couleurs : 'blue','red','green','yellow','pink',etc...
Pour affecter une couleur à une ligne la commande est : color(couleur)

Remarque : Les exercices 4, 5 et 6 ont déjà été traités avant. Quelle différence notez-vous ?

Pour un complément sur la notion de fonction avec Python, se référer à la fiche : **Plus sur la notion de fonction**

Fiches de complément Python

PLUS SUR LA NOTION D'AFFECTATION

L'affectation

Dans le langage de programmation Python, on dispose de l'opération d'affectation symbolisée par le signe « égale » : =
Par exemple, analysez les séquences d'instructions suivantes :

1. L'affectation simple

```
>>> a = 3                    # J'affecte à la variable a la valeur 3
>>> b = 4                    # J'affecte à la variable b la valeur 4
```

2. L'affectation parallèle

```
>>> a,b = 3,4                # J'affecte simultanément à a la valeur 3 et à b la valeur 4
```

Ici, la ligne de code ci-dessus parvient exactement au même résultat que les 2 lignes de codes précédentes.

♦♦♦ ATTENTION ! Il convient cependant de bien comprendre comment fonctionne l'affectation parallèle par rapport à l'affectation simple.

Exercice résolu : on souhaite échanger les valeurs de 2 variables a et b.

- Résolution avec l'affectation simple

```
>>> a = 3
>>> b = 4
                             # Fausse bonne idée : échanger directement les valeurs de a et b
>>> a = b                    # a prend la valeur 4
>>> b = a                    # La valeur 4 étant à présent affectée à la variable a, b la prend aussi !
                             # Au final, a et b valent 4
                             # On va avoir besoin d'une variable supplémentaire pour stocker
                               la valeur initiale de a. Corrigeons-donc notre script...
>>> a = 3
>>> b = 4
>>> c= a                     # J'affecte à c la valeur initiale de a, ç'est-à-dire 3
>>> a = b                    # a prend la valeur de b, ç'est-à-dire 4
>>> b = c                    # b prend la valeur de c, ç'est-à-dire la valeur initiale de a : 3
```

- Résolution avec l'affectation parallèle

```
>>> a,b= 3,4                 # J'affecte simultanément à a la valeur 3 et à b la valeur 4
>>> a,b = b,a                #ET LA, GROSSE DIFFERENCE : les expressions de la partie
                               droite sont d'abord toutes évaluées avant qu'aucune affectation
                               ne se fasse. Comme b vaut 3 et a vaut 4, le tour est joué !
```

Exercice
Quelles valeurs vont être affectées à a et à b au final après les lignes de codes suivantes ?

```
>>> a,b = 6,10                          >>> a,b= 6,10
>>> a = b                   OU          >>> a,b = b, a - b
>>> b = a – b
```

On a aussi la notion d'*affectation multiple*. Exemple a=b=5 On affecte à a et b la valeur 5

PLUS SUR LA SAISIE DE DONNEES

La commande input()

Dans le langage de programmation Python, on dispose de la fonction prédéfinie input() qui permet à l'utilisateur de saisir des données à l'écran.

Par exemple testez :

```
>>> a=input()                    # l'ordinateur attend la saisie de l'utilisateur.
3
>>> a
'3'                              # Attention même si c'est un entier qui est saisi, a est du type « string »
>>> type(a)
<class 'str'>                    # Comment le convertir en entier ?
>>> a=int(input())
3
>>> a
3
>>> type(a)
<class 'int'>
```

La commande input() permet aussi d'afficher du texte. Il suffit d'écrire entre guillemets ou simple quotes le message à l'intérieur des parenthèses.

Exemple :

```
>>> a=input("Saisissez un nombre décimal ")
Saisissez un nombre décimal          # l'ordinateur attend la saisie de l'utilisateur

# Si l'on veut que a soit du type float, on doit composer deux instructions.
# On modifie la ligne précédente par :
>>> a=float(input("Saisissez un nombre décimal "))
Saisissez un nombre décimal 20.6          # je saisis 20.6 par exemple
>>> type(a)
<class 'float'>                           # La variable a a été convertie en flottant
```

On retiendra : la commande a=input() permet d'affecter à la variable a ce qui est saisi par l'utilisateur à l'écran. **ATTENTION !** On gardera à l'esprit que a est du type string. Si l'on souhaite travailler avec des nombres ou des listes, il faudra convertir a en composant une instruction adéquate avec la commande input().

PLUS SUR LA NOTION DE FONCTION

Comment retourner une valeur ?

Lorsqu'une fonction est destinée à renvoyer une valeur, on utilise l'instruction **return**(argument)

Exercice 1

Dans une fenêtre vierge, recopier le script suivant :

```
def puissance3(x)
        return(x**3)
```

Puis testez ce programme en l'enregistrant sous le nom : TP_fonctions_calcul1.py . Appuyez ensuite sur Ctrl+F5. Testez:

1. puissance3(2)
2. puissance3(10)

Exercice 2

On souhaite dans cet exercice calculer le volume d'un parallélépipède rectangle. Écrire une fonction volume dépendant de trois paramètres qui renvoie le volume d'un tel parallélépipède. Testez-la ensuite sur quelques exemples. Vous enregistrerez ce programme sous le nom : TP_fonctions_calcul2.py

Variables locales et globales

Lorsque l'on définit une fonction, il est nécessaire de connaître la portée des variables.

Exemple : Définissons une fonction en mode interactif sous IDLE.

```
>>> def fonction1():
        x=3
        print("Dans cette fonction x est égal à ",x)


>>> fonction1()
Dans cette fonction x est égal à  3
>>> x                                    # message d'erreur
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    x
NameError: name 'x' is not defined
```

Remarque : **la variable x n'existe pas en dehors de la fonction où elle a été définie.**
On dit que x est une une variable locale.

Par contre, une valeur déclarée à la racine du module principal est visible partout.

Exemple :

```
>>> def fonction2():
        print(x)


>>> x=4
```

```
>>> fonction2()
4
>>> x=5
>>> fonction2()
5
```

On dit que x est une variable globale. Cette variable est visible dans tout le module.

††† ATTENTION aux types modifiables comme les listes †††

```
>>> liste=[1,2,3]                    # liste est une variable globale
>>> def change_liste():
        liste[1]=-15


>>> change_liste()
>>> liste
[1, -15, 3]                          # liste a été modifiée par la fonction change_liste()
```

De la même manière, en passant une liste en arguments, elle est tout autant modifiable !

```
>>> liste=[1,2,3]
>>> def change_liste2(x):            # x est passé en argument
        x[1]=-20


>>> change_liste2(liste)
>>> liste
[1, -20, 3]                          # liste a été modifiée par la fonction change_liste2()
```

La règle LGI

Nous avons vu au paragraphe précédent les notions de **variables locales** ou **globales**. Il existe aussi un troisième type de variables : les **variables internes**. Exemple : la fonction len() qui renvoie la taille d'une liste ou d'une chaîne de caractères et qui existe dès qu'on lance Python.

Python traite les variables par ordre de priorité :

1. D'abord, il regarde si la variable considérée est une variable locale,
2. Ensuite, si elle n'existe pas localement, il regarde si c'est une variable globale,
3. Enfin, il regarde si c'est une variable interne.

Exemple :

```
>>> def fonction():
        x=10                         # x est une variable locale
        print('Dans la fonction x vaut ',x)


>>> x=20                             # x est une variable globale
>>> fonction()
Dans la fonction x vaut  10
>>>print('Dans le module principal x vaut ',x)
Dans le module principal x vaut  20
```

Remarque : Notez bien ce qui s'est passé. x a pris en priorité la valeur qui lui était définie localement par rapport à celle qui lui était définie globalement.

Il est possible de forcer une variable à prendre sa valeur globale dans une fonction avec le mot clé : **global**

Exemple :

```
>>> def double():
        global x
        x=x*2


>>> x=3
>>> double()
>>> x
6
```

Remarque : cette notion a son importance lorsque dans une fonction qui renvoie une ou plusieurs valeurs, on souhaite que ces dernières servent dans le programme principal. On doit donc les déclarer en tant que variables globales.

Exemple : On souhaite subdiviser un intervalle [a ; b] en n sous-intervalles égaux. On accepte le chevauchement aux points de subdivision.

```
def subdivision(a,b,n):
   global subdi                          # la variable subdi est déclarée comme globale
   subdi=[]                              # on initialise subdi par la liste vide
   i,c=0,0
   while i<=n:
      c=a+i*(b-a)/n                      # on construit les points de la subdivision
      subdi.append(c)
      i=i+1

   return subdi

# programme principal
a=float(input("Entrer la borne inférieure de l\'intervalle "))
b=float(input("Entrer la borne supérieure de l\'intervalle "))
n=int(input(" Entrer le nombre d\'intervalles n de la subdivision "))
subdivision(a,b,n)
print (subdi)
                                         # on compile le programme avec Ctrl+F5
Entrer la borne inférieure de l'intervalle 2
Entrer la borne supérieure de l'intervalle 10
Entrer le nombre d'intervalles n de la subdivision 20
[2.0, 2.4, 2.8, 3.2, 3.6, 4.0, 4.4, 4.8, 5.2, 5.6, 6.0, 6.4, 6.8, 7.2, 7.6, 8.0, 8.4, 8.8, 9.2, 9.6, 10.0]
```

LA RECURSIVITE

Définition : Un algorithme est dit **récursif** s'il s'appelle lui-même.

Reprenons l'exemple de la factorielle vu en classe. On rappelle que :

- n doit être un entier naturel.

1. Si n=0, on a n! = 1 ie 0! = 1
2. Si n est supérieur ou égal à 1, on a $n! = 1 \times 2 \times 3 \times ... \times n$.
   n! se lit « factorielle n »

Écrire une fonction factorielle(n) qui renvoie la factorielle d'un entier naturel choisi par l'utilisateur.

Quel lien existe-t-il entre $n!$ et $(n-1)!$?

Ce lien nous suggère l'algorithme récursif suivant, où la fonction factorielle va s'appeler elle-même.

```
def factorielle(n):
    if n = = 0:
        return 1
    else:
        return n * factorielle(n-1)        # la fonction factorielle est appelée à l'intérieur d'elle
                                             même.
```

Essayez de comprendre ce qui se passe. Schématisez-le pour n=2.

Exercice : on pose $u_0 = 2$ et pour tout entier naturel n non nul, $u_n = 3u_{n-1} + 8$. Créer une fonction suite(n) en utilisant un algorithme récursif capable de calculer n'importe quelle valeur u_n .

Test avec un tableur :

Calcul des 20 premiers termes

	A	B
1	n	u_n
2	0	2
3	1	14
4	2	50
5	3	158
6	4	482
7	5	1454
8	6	4370
9	7	13118
10	8	39362
11	9	118094
12	10	354290
13	11	1062878
14	12	3188642
15	13	9565934
16	14	28697810
17	15	86093438
18	16	258280322
19	17	774840974
20	18	2324522930
21	19	6973568798
22	20	20920706402

Formule dans B3 :

=3*B2+8

Utilisation du logiciel Python :

```
def suite(n):
    if n==0:
        return 2
    else:
        return 3*suite(n-1)+8


# programme principal

a=0
while a<=20:
    print suite(a)
    a=a+1
```

On compile ensuite ce code.

Résultat à l'écran

```
>>>
2
14
50
158
482
1454
4370
13118
39362
118094
354290
1062878
3188642
9565934
28697810
86093438
258280322
774840974
2324522930
6973568798
20920706402
>>>
```