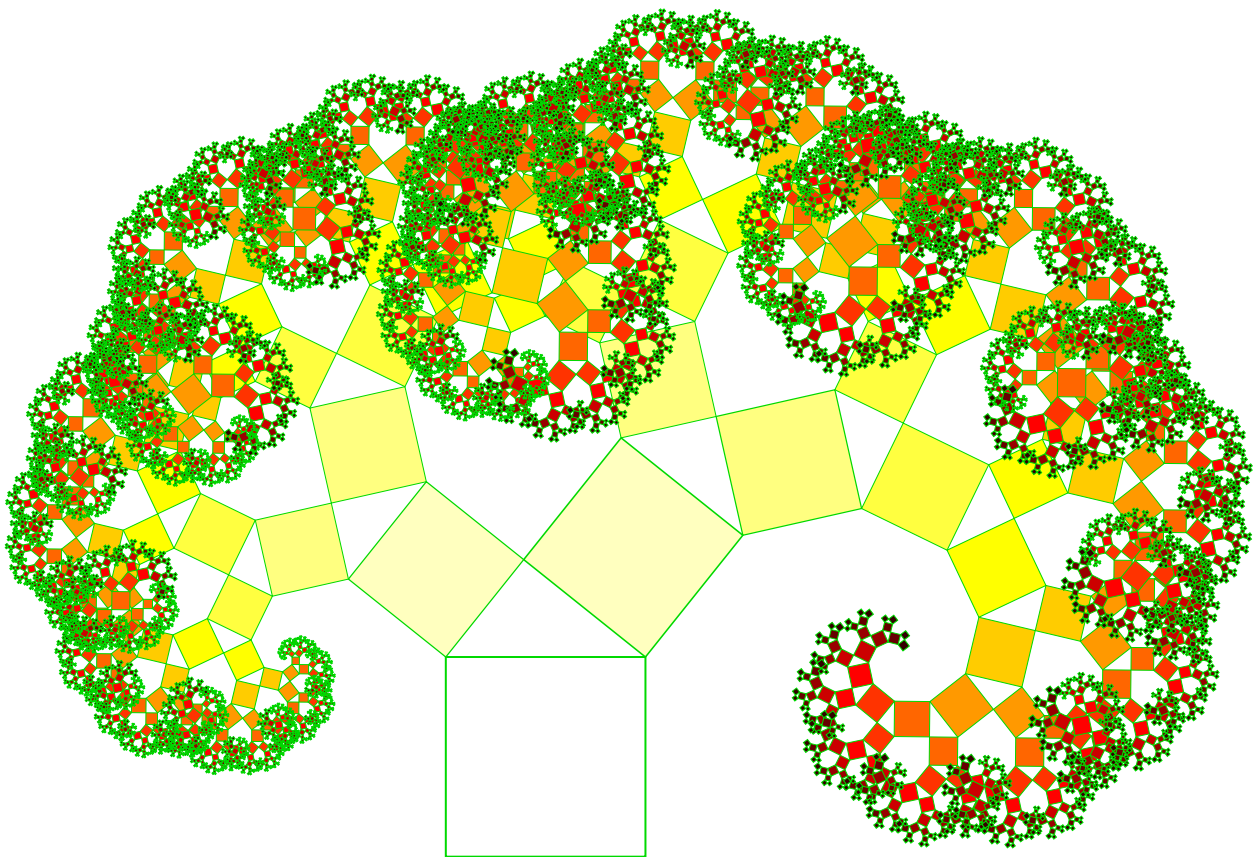


# Initiation à l'algorithmique avec Python 3

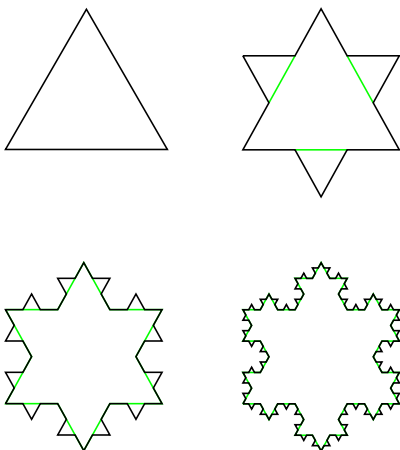
Franck CHAMBON

26 janvier 2011



Ci-dessus, l'arbre de PYTHAGORE,

ci dessous le flocon de VON KOCH



```
from turtle import *
T=[1,-2,1,0]
def koch(x,n):
    if n==0:
        forward(x)
    else:
        for i in T:
            koch(x/3, n-1)
            left(60*i)
for _ in range(3):
    koch(300, 5)
    right(120)
mainloop()
```


# Table des matières

<b>I]</b>	<b>Avant propos</b>	<b>3</b>
I-a)	Installation de Python . . . . .	3
I-b)	Amélioration de l'ergonomie . . . . .	3
<b>II]</b>	<b>Premières opérations</b>	<b>3</b>
II-a)	Les quatre opérations et les puissances . . . . .	3
II-b)	L'écriture scientifique . . . . .	4
<b>III]</b>	<b>Premières affectations</b>	<b>4</b>
III-a)	L'affectation . . . . .	4
III-b)	L'utilisation . . . . .	5
III-c)	Les noms de variables . . . . .	5
III-d)	Les <b>33</b> mots réservés . . . . .	5
III-e)	Python, un langage à typage dynamique . . . . .	5
III-f)	Affectation $\neq$ égalité . . . . .	6
III-g)	Structurer ses instructions . . . . .	6
III-h)	Utilisation en mathématiques au Collège . . . . .	6
<b>IV]</b>	<b>Les Fonctions</b>	<b>7</b>
IV-a)	Première fonction mathématique . . . . .	7
IV-b)	Premiers affichages . . . . .	7
IV-c)	Premières créations de procédures . . . . .	7
IV-d)	Première création de fonction . . . . .	8
<b>V]</b>	<b>L'environnement de travail</b>	<b>9</b>
V-a)	Premier programme : <i>Hello World</i> . . . . .	9
<b>VI]</b>	<b>Les tableaux</b>	<b>10</b>
VI-a)	Définition, première utilisation . . . . .	10
VI-b)	Tableaux automatiques : <b>range</b> . . . . .	10
<b>VII]</b>	<b>Les boucles</b>	<b>10</b>
VII-a)	Boucles <b>for</b> . . . . .	10
VII-b)	Boucle <b>while</b> . . . . .	11
<b>VIII]</b>	<b>Le module <b>turtle</b></b>	<b>11</b>
VIII-a)	Premières instructions . . . . .	11
VIII-b)	Autres instructions pour <b>turtle</b> . . . . .	12
<b>IX]</b>	<b>Approche de la récursivité</b>	<b>13</b>
<b>X]</b>	<b>Pour aller plus loin</b>	<b>14</b>
X-a)	Différences entre Python 2 et 3 . . . . .	14
X-b)	Des ouvrages de référence . . . . .	14
<b>A</b>	<b>Annexe : Feuilles d'exercices</b>	<b>14</b>
1	Pour commencer . . . . .	15
2	mordu ? . . . . .	16
3	recherche <b>while</b> désespérément . . . . .	17
4	Tableaux et boucle <b>for</b> . . . . .	18
5	Premières figures . . . . .	19
6	Tournicoti . . . . .	20
7	Tournicoton . . . . .	21
8	Arbre de Pythagore . . . . .	22
<b>B</b>	<b>Sur ce document</b>	<b>23</b>

# Initiation à l'algorithmique

## I] Avant propos

Python est un langage de programmation, très répandu et idéal pour débiter. Le langage est sous *licence libre* ; l'utilisation est et restera *gratuite*. Python est multiplateforme ainsi vous pourrez l'utiliser avec Windows, Mac/OS, ou GNU/Linux. Il existe deux branches actives : Python 2.x et Python 3.x.

Pour s'initier à l'algorithmique nous choisissons  Python 3

### I-a) Installation de Python



#### Installation sous GNU/Linux

Avec votre gestionnaire de paquet, installez le paquet **idle3**, c'est tout. *Félicitation.*



#### Installation sous Windows


Sur le site [www.python.org](http://www.python.org), vous trouverez le Windows Installer, fichier de 14Mo environ, c'est très peu. *Installation par défaut.*

Pour ouvrir Python, lancer le programme IDLE3. Vous devriez obtenir presque ceci :

```
Python Shell
File Edit Debug Options Windows Help
Python 3.1.2 (release31-maint, Sep 17 2010, 20:27:33)
[GCC 4.4.5] on linux2
Type "copyright", "credits" or "license()" for more information.
==== No Subprocess ====
>>> |
```

### I-b) Amélioration de l'ergonomie

Le choix par défaut de la fonte et de sa taille est mauvais, cliquer sur options, puis Configure IDLE... Choisir dans la liste des fontes : **DejaVu Sans Mono**, une belle fonte mécanique, **Mono** est important. Avec une taille de 16, pour commencer, vous pourrez réduire ensuite.

Ensuite, dans l'onglet General, section Autosave Preferences choisir **No Prompt**, pour At Start of Run (F5) Cela vous permettra de sauvegarder/compiler/lancer vos programmes en une seule touche : .

## II] Premières opérations

### II-a) Les quatre opérations et les puissances

```
>>> 5+3      # Un commentaire, il débute avec sharp # (AltGr+3) au clavier
8

>>> 35 - 2*10 # Espaces optionnelles. Priorités respectées
15

>>> (35-2)*10
370

>>> 23/4      # !!! attention avec Python2, le résultat était tronqué à 5 !!!
5.75

>>> 23:4      # Erreur de syntaxe, ":" n'est pas l'opérateur de la division !!!
SyntaxError: invalid syntax

>>> 23//4     # Si vous voulez le résultat entier tronqué
5
```

```
>>> 5**2      # 5 au carré, attention dans d'autres langages, on note 5^2
25

>>> 2*5**3    # Les puissances sont prioritaires, on obtient alors 2*(5**3)=2*125=250
250           #      ( et non 10**3=1000  )

>>> 55.6 + 21.13 # Dans le monde anglo-saxon, le séparateur décimal est le point .
76.73         # On ne doit pas utiliser la virgule pour un nombre décimal

>>> 55,6 + 21,13 # Une erreur sémantique, (vous donnez un autre sens)
(55 , 27 , 13) # sauf si vous désirez la liste (55 , 6+21 , 13)
```



### Résumé des opérations de bases

Avec Python 3 :

- + addition
- soustraction
- \* multiplication
- / division
- \*\* puissance
- // division entière
- % reste de cette division



### Particularités

- Les priorités sont respectées
- Les espaces autour des opérations : une option
- Le séparateur décimal est le point



### Divisions entières

```
>>> 147//9    # réponse : le quotient
16
>>> 147%9     # réponse : le reste
3
```

Ainsi : 147 divisé par 9 donne 16, et il reste 3.

## II-b) L'écriture scientifique

Avec Python, on peut entrer des nombres en écriture scientifique, suivant le modèle :

Pour  $1,3 \times 10^5$ , on entre :

```
>>> 1.3e5     # réponse :
130000.0
```

D'accord, mais pourquoi .0 ?

C'est que Python connaît différents **types** de nombres, il y a les nombres entiers (*integers*), et les nombres réels à virgule flottante (*floating point numbers*).

Une écriture scientifique sera toujours de **type float**, donc Python l'indique avec « .0 »

On reviendra sur la notion de **type**.



### Mini-problème

La masse d'un atome de carbone est environ  $2 \times 10^{-26}$  kg, celle de l'oxygène est environ  $2,7 \times 10^{-26}$  kg.

Quelle est la masse du dioxyde de carbone ( $\text{CO}_2$ ) ?

```
>>> 2e-26 + 2*2.7e-26 #Réponse :
7.4e-26
```

La molécule de  $\text{CO}_2$  a une masse de  $7,4 \times 10^{-26}$  kg.

Un réservoir de voiture contient 37 kg de carburant, et la combustion génère environ  $1,6 \times 10^{27}$  molécules de  $\text{CO}_2$ . À chaque plein de carburant, quelle masse de  $\text{CO}_2$  a été envoyée dans l'atmosphère ?

```
>>> 1.6e27*7.4e-26
118.4 # kg à chaque plein !!!
```

## III] Premières affectations

Python possède des « mémoires », que l'on appelle des *variables*.

### III-a) L'affectation

```
>>> a=12      # Ceci est une affectation, on "met" 12 dans 'a'.    a ← 12

>>> b=5+a     # Ceci est un calcul (5+12 qui donne 17), suivi de l'affectation b ← 17

>>> a=9       # Ceci est une réaffectation, a ← 9, maintenant a ≠ 12
```

### Conseils

1. Bien que l'affectation multiple soit possible, on déconseille cela aux débutants  
`>>> a = b = 7           # qui donne a ← 7 et b ← 7`
2. Bien que l'affectation parallèle soit possible, on déconseille cela aux débutants  
`>>> c, d = 8, 9       # qui donne c ← 8 et d ← 9`

Mais bientôt, on vous conseillera de l'utiliser!!!

### III-b) L'utilisation

Une fois qu'une valeur est « stockée » dans une variable, on peut l'utiliser.

```
>>> a=174.26       # Une affectation simple
>>> a               # Un appel
174.26       ← la réponse, quelle surprise
>>> a**2 - 3*a    # 'a' au carré, moins le triple de 'a'
29843.7676       ← le calcul de  $a^2 - 3a$ 
```

### III-c) Les noms de variables

On ne peut pas choisir n'importe quel nom pour une variable. De manière synthétique, on a :


**Obligatoire :** Débuter par une lettre et n'utiliser que les caractères :  $a \rightarrow z$  ,  $A \rightarrow Z$  ,  $0 \rightarrow 9$  et  $_$

**Conseil :** Débuter par une minuscule.

**Exemples :** `nombre_d0r` , `a15` sont des noms de variable valides.

**Remarques :** `niter`, `nIter`, `NITER` sont trois variables différentes.

On dit que Python est sensible à la casse ; les majuscules comptent.

Avec  Python 3, les lettres accentuées sont autorisées pour les noms de variables.

### III-d) Les 33 mots réservés

Interdiction de les utiliser en nom de variable

<b>and</b>	<b>as</b>	<b>assert</b>	<b>break</b>	<b>class</b>	<b>continue</b>	<b>def</b>
<b>del</b>	<b>elif</b>	<b>else</b>	<b>except</b>	<b>False</b>	<b>finally</b>	<b>for</b>
<b>from</b>	<b>global</b>	<b>if</b>	<b>import</b>	<b>in</b>	<b>is</b>	<b>lambda</b>
<b>None</b>	<b>nonlocal</b>	<b>not</b>	<b>or</b>	<b>pass</b>	<b>raise</b>	<b>return</b>
<b>True</b>	<b>try</b>	<b>while</b>	<b>with</b>	<b>yield</b>		

Chacun de ces mots a une utilisation que nous découvrirons, il ne faut pas les mélanger avec les variables.

**Exemple :** détruire (*delete*) une variable

```
>>> del nombre_d0r   # Détruit l'instance nombre_d0r
```

**Remarque :** **True** et **False** (vrai et faux) sont les seuls mots réservés commençant par une majuscule.

### III-e) Python, un langage à typage dynamique

On a déjà vu des types de variables : `int` pour entier (*integer*) et `float` pour nombre à virgule flottante (*floating point number*). Il existe d'autres types, comme `string` pour chaîne de caractères.

On peut changer le type d'une variable en la réaffectant, on dit que Python est un langage à typage dynamique.

```
>>> a=7       # Une affectation qui donne à 'a' le type 'int' pour integer (entier)
>>> type(a)   # Pour connaître le type d'une variable
<class 'int'>   # Réponse : 'int'
>>> a="bonjour"   # Une réaffectation qui modifie le type de 'a' en 'str'
          # 'str' pour string (chaîne de caractères)
          # C'est interdit avec d'autres langages (ex. C++) dont le typage est statique
```

On en profite pour signaler qu'une méthode pratique pour entrer une chaîne de caractères est de l'insérer entre une paire de " (guillemet droit double).

```
>>> phrase1 = "Exemple de chaîne de caractères"   # C'est la méthode que vous devez retenir.
>>> phrase2 = 'Un autre exemple'               # Certes le guillemet droit simple est accepté, mais
```

```

FAUX: >>> phrase3 = 'L'erreur'          # c'est le même caractère que l'apostrophe dactylographique.
''          # Une chaîne vide,
""          # une autre chaîne vide.
""""Du "texte" incluant des " et ', étonnant?"""" # Situation extrême : paire de triple double

```

### III-f) Affectation $\neq$ égalité

En pseudo-langage algorithmique, l'affectation se note  $\leftarrow$ , comme  $a \leftarrow 7$ .

En informatique, l'affectation se note souvent  $=$ , ainsi

```

>>> a=7 # Affectation classique,  a ← 7
>>> a=a+8 #Affectation légale,  a ← a + 8 = 7 + 8 = 15

```

En mathématiques,  $a = a + 8$  est une équation qui n'a pas de solution réelle, elle signifierait  $0 = 8$ . Ainsi, on veillera à ne pas confondre le  $=$  d'affectation en informatique, et le  $=$  en mathématiques.

Il faut toutefois disposer d'un symbole pour vérifier une égalité.

En informatique le « test d'égalité » se note souvent  $==$

```

>>> 10==8+2 # Un test d'égalité, qui répond True (vrai)
True
>>> 11==8+2 # Réponse : False (faux)
False

```

### III-g) Structurer ses instructions

L'ordre dans lequel on effectue les affectations est important.

```

1 >>> a=7  # a ← 7, 'a' est affecté à 7
2 >>> b=8  # b ← 8, 'b' est affecté à 8
3 >>> a=b  # a ← b, 'a' est affecté à 'b' qui est égal à 8, donc a = 8
4 >>> b=a  # b ← a, 'b' est affecté à 'a' qui est désormais égal à 8, donc b = 8
5 >>> a, b # Pour afficher les valeurs de 'a' et 'b'
6 (8,8)   # ← La réponse. Réessayer en intervertissant les lignes 3 et 4 !!!

```

### III-h) Utilisation en mathématiques au Collège

Python est capable de répondre à de nombreux problèmes numériques au Collège.

**Exemple 1** Calculer  $A = 3x(5x - 2) + 6x(5 + 3x)$ , avec  $x = 2$ ,  $x = 12$  et  $x = 103$ .

**Solution** avec Python :

```

>>> x=2          # Pour le premier calcul
>>> 3*x*(5*x-2)+6*x*(5+3*x) # Les multiplications doivent être explicites
180              # La réponse pour x=2

```

Pour les autres calculs, on modifie  $x$ , puis on refait le calcul. Il est possible de faire un copier-coller.

**Exemple 2** Tester l'égalité  $3x + 7 = 12x - 9$  pour  $x = 5$ ,  $x = 3$  et  $x = 3,5$

```

>>> x=5          # x ← 5, pour le premier test avec x=5
>>> 3*x+7==12*x-9 # == pour le test d'égalité
False            # La réponse est 'faux'

```

Pour les autres tests, on modifie  $x$ , on peut copier-coller le test.

C'est juste et rapide, mais il n'y a pas de justifications...

Pour une liste de valeurs plus longue, nous ferons un programme...

**Contre-exemple :** Pour travailler avec les fractions, on ne peut pas avoir directement les résultats exacts, il faut charger un *module* fractions. Utilisation de **import**. De manière générale, pour travailler avec des valeurs exactes, on utilisera un CAS (*Computer Algebra System*), comme SAGE ou XCas. Utilisation à partir du lycée.

## IV] Les Fonctions

### IV-a) Première fonction mathématique

Avant de faire de la programmation, montrons comment ajouter des fonctionnalités à Python.

Python n'est qu'un noyau, auquel on peut ajouter des programmes qui ont été écrits par des tiers.

Il y a, par exemple, un module **math** qui a été écrit pour apporter de nombreuses fonctions mathématiques usuelles, comme la racine carrée.

En informatique, la racine carrée (*square root*) est souvent notée **sqrt**. Pour demander à Python de charger (**import**) la fonction **sqrt** à partir (**from**) du module **math**, on entre :

```
>>> from math import sqrt
```

À partir de maintenant, on peut faire ces calculs :

```
>>> sqrt(169)
13
>>> sqrt(9876543210)
99380.79900061178
```


Si votre calculatrice bloque avec des nombres trop grands, pensez à Python, et vous aurez peut-être la réponse.

### IV-b) Premiers affichages

Pour afficher (**print**) un résultat, en mélangeant texte et nombre, suivre ce modèle pour commencer :

```
>>> x = 12 # Affectation    x ← 12
>>> calculA = 3*x*(5*x-2)+6*x*(5+3*x) # Calcul de A, et affectation dans 'calculA'
>>> print("Pour x =", x, ", on a : A =", calculA)
Pour x = 12 , on a : A = 5040
```

**print** est une fonction qui peut prendre plusieurs arguments, séparés par des virgules ; on peut faire se succéder des chaînes de caractères et des nombres. Pendant la saisie, une aide contextuelle apparaît.

Avec  Python 3, **print** est une fonction comme les autres, avec la version 2, elle avait un rôle spécial. Nous verrons les possibilités avancées de **print** plus tard.

IDLE3 possède une coloration syntaxique qui vous permet de bien distinguer les chaînes de caractères du reste, ainsi que les fonctions usuelles et les mots réservés.

### IV-c) Premières créations de procédures

Nous reprenons les exemples de III-h) pour automatiser des instructions.

**Exemple 1** Calculer  $A = 3x(5x-2) + 6x(5+3x)$ , avec  $x = 2$ ,  $x = 12$  et  $x = 103$ .

```
1 >>> def afficheA(x):
2 ...     print("L'expression A = 3x(5x-2) + 6x(5+3x) dépend de x.")
3 ...     print("Pour x =", x, ", A = ", 3*x*(5*x-2)+6*x*(5+3*x))
4 ...                                     # /\ ligne vide
5 >>> afficheA(2)
6 L'expression A = 3x(5x-2) + 6x(5+3x) dépend de x.
7 Pour x = 2 , A = 180
8 >>> afficheA(12)
9 L'expression A = 3x(5x-2) + 6x(5+3x) dépend de x.
10 Pour x = 12 , A = 5040
```

Explications ligne par ligne :

1. On définit (**def**), une procédure **afficheA**, qui prendra un argument **x**, on finit par **:** (mais que fait **afficheA**?)
2. Le curseur a été **indenté** (*décalé*) automatiquement.  
**afficheA** commence par écrire une simple chaîne de caractères, sans aucun calcul.
3. Ensuite, avec la même indentation, **afficheA** affiche des bouts de phrases avec le résultat du calcul demandé.
4. Une ligne vide implique que la définition de **afficheA** est finie.
5. On utilise **afficheA** avec un argument, on peut la réutiliser autant de fois qu'on veut.



**Exemple 2** Tester l'égalité  $3x + 7 = 12x - 9$  pour  $x = 5$ ,  $x = 3$  et  $x = 3,5$

```
1 >>> def exemple2(x):
2 ...     if 3*x+7==12*x-9 :
3 ...         print("L'égalité 3x+7=12x-9 est vérifiée pour x = ", x)
4 ...     else
5 ...         print("L'égalité 3x+7=12x-9 n'est pas vérifiée pour x = ", x)
6 ...
7 >>> exemple2(5)      # Puis 3, puis 3.5
8 L'égalité 3x+7=12x-9 n'est pas vérifiée pour x = 5 # La réponse
```

Explications ligne par ligne :

1. On définit **exemple2** (il faut choisir des noms de circonstance), qui prend un argument **x**. Ne pas oublier :
2. On débute le test SI (*if*). **if** <le test> : Ne pas oublier de finir par :
3. Ici, si <le test> est vrai (*True*), alors on affiche que l'égalité est vérifiée, avec la valeur de **x**.
4. Sinon (*else*)
5. Cette ligne de code n'est exécutée que si <le test> a échoué. On affiche que l'égalité n'est pas vérifiée.
6. Ligne vide pour finir la définition de **exemple2**.
7. Utilisation de **exemple2**, avec l'argument 5
8. La réaction de la procédure **exemple2**.

Le décalage dans les structures **def**, **if** et **else** est obligatoire. On l'appelle l'indentation. Par tradition, on utilise 4 espaces par niveau. Python interprète alors correctement leur début et fin.

D'autres langages utilisent des balises comme **begin** et **end**, ou { et } ce qui rend le code moins lisible et plus long. L'indentation est une bonne pratique, en Python elle est obligatoire.

#### IV-d) Première création de fonction

Nous sommes heureux d'avoir pu automatiser les exemples 1 et 2, nous avons eu un affichage correct, mais nous ne pouvons pas réutiliser automatiquement les résultats obtenus ; ils n'ont pas été stockés.

En fabriquant une fonction (*une procédure qui renvoie une valeur*), on peut réutiliser le résultat.

**Exemple 1** : définition de la fonction calculA :  $x \mapsto 3x(5x - 2) + 6x(5 + 3x)$

```
1 >>> def calculA(x):
2 ...     return 3*x*(5*x-2)+6*x*(5+3*x)
3
4 >>> calculA(103)%10 # le reste de la division par 10, donne le chiffre des unités.
5 9
```

Explications ligne par ligne :

1. On définit (**def**) la fonction **calculA** par :
2. Indentation. On renvoie (*return*) le résultat du calcul
3. Ligne vide ; fin de la définition de **calculA**
4. Utilisation : calcul pour  $x = 103$ , on demande le reste de la division par 10
5. Le résultat est affiché, c'est le dernier chiffre de **calculA(103)**.



#### Premiers exercices

Avant d'aller plus loin, il faut s'exercer à manipuler les mémoires. Affectation, réaffectation, fonction sont au programme.

1. Lire des instructions et comprendre le mécanisme.
2. Savoir appliquer manuellement des instructions.
3. Créer ses propres instructions.



Faire les exercices Python 3 sur la **feuille n° 1** et la **feuille n° 2**.



## V] L'environnement de travail

Pour un exercice rapide comme précédemment, on utilise une console (*shell*) Python.  
Notre travail ne sera pas enregistré sur l'ordinateur.

Pour un projet plus long, que l'on souhaite retravailler, on utilise un IDE (*Integrated Development Environment*), pour débiter avec Python, l'IDE d'origine est parfait. On peut éditer (écrire/modifier) un programme dans un fichier et l'exécuter (le lancer — *run*).



### Les premières étapes à effectuer :

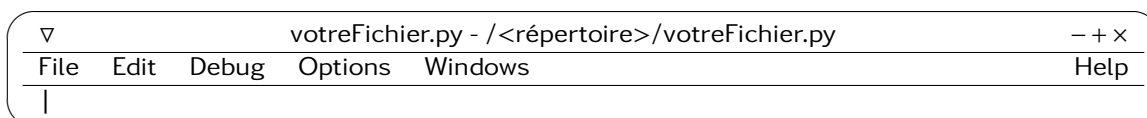
1. Ouvrir une nouvelle feuille avec l'éditeur, avec **Ctrl** + **N**.
2. Écrire un programme en python. (à suivre)
3. Sauvegarder (*Save*) le code, avec **Ctrl** + **S**.  
Choisir l'extension **.py**. Est-elle automatique ?
4. Vérifier la syntaxe, avec **Alt** + **X**.
5. Lancer (*Run*) votre programme avec **F5**

**Remarque :** Pour sauvegarder votre fichier, choisir un répertoire dans lequel vous avez des droits d'écriture. Avec une installation en réseau, ce n'est pas le cas par défaut.



### pendant la construction

- Alt** + **X** pour vérifier le code
- F5** pour sauver et exécuter le programme.
- Alt** + **↔** pour alterner entre vos fenêtres.



Exemple minimaliste de IDLE.

La décoration variera en fonction de votre gestionnaire de fenêtre.  
Il est conseillé de laisser l'extension par défaut **.py** à vos fichiers.  
Ils pourront ensuite être exécutés facilement dans un terminal.

### V-a) Premier programme : *Hello World*

Tradition oblige, notre premier programme saluera le monde.

Ouvrir un nouvelle fenêtre IDLE — **Ctrl** + **N**  
Sauvegarder sous le nom **HelloWorld(.py)** — **Ctrl** + **S**  
Taper le code suivant :

```
1 print("Hello World!")
2 nom=input("Comment t'appelles-tu, toi ? >")
3 print("Bien le bonjour",nom)
```

Résultat dans le shell :

```
Hello World!
Comment t'appelles-tu, toi ? >Cémoi
Bien le bonjour Cémoi
```

Explications :

1. On affiche une première ligne, facile !
2. On demande une entrée (*input*), on affecte le résultat dans **nom**.
3. On affiche les deux chaînes de caractères à la suite.



### Utilisation d'input

**input** est une fonction qui renvoie toujours une chaîne de caractères. Ainsi :

```
1 nombre=input("Donne un nombre >")
2 print("Son carré est :",nombre*nombre)
```

provoquera une erreur ; on ne multiplie pas les chaînes de caractères entre elles.



### Changer le type d'une variable

Les types déjà rencontrés :  
'str' pour les chaînes de caractères  
'int' pour les entiers  
'float' pour les nombres à virgule flottante.

```
1 réponse=input("Donne un nombre >")
2 nombre=float(réponse)
3 print("Son carré est :",nombre*nombre)
```

fonctionnera correctement.

**float()**, **int()**, **str()** transforment dans la mesure du possible leur argument dans le type choisi.

## VI] Les tableaux

### VI-a) Définition, première utilisation


```
1 >>> tablo = [100, 101, 102, 103]
2 >>> len(tablo)
3 4      # La réponse
4 >>> tablo[0]
5 100    # La réponse
6 >>> tablo[len(tablo)-1]
7 103    # La réponse
8 >>> tablo[len(tablo)]
9 IndexError: list index out of range
```

Les indices vont de 0 à `len(.) - 1`

Explication ligne par ligne

1. Un tableau : valeurs entre crochets, séparées par des virgules.
2. `len(.)` renvoie la longueur (*length*) du tableau.
3. Ici 4 valeurs pour `tablo`.
4. On demande la valeur d'indice 0 du tableau.
5. 100 est la première (indice 0) valeur du tableau.
6. On demande la valeur d'indice  $4 - 1 = 3$  du tableau.
7. 103 est la dernière (indice 3) des 4 valeurs du tableau.
8. On demande la valeur d'indice 4 du tableau.
9. Une erreur d'indice est annoncée, `tablo[4]` n'existe pas.

### VI-b) Tableaux automatiques : `range`

 Python 3 dispose de tableaux pré-remplis virtuellement, ils ne prennent presque aucune place en mémoire.

`range(100)` est un tableau virtuel à 100 éléments, de 0 à  $99 < 100$

`range(100)`  $\leftrightarrow$  `[0, 1, 2, ... 99]`.

`range(5, 100)` est un tableau virtuel à  $100 - 5 = 95$  éléments, de 5 à  $99 < 100$

`range(5, 100)`  $\leftrightarrow$  `[5, 6, 7, ... 99]`.

`range(0, 100, 7)` est un tableau virtuel à  $14$  éléments, de 0 à  $14 \times 7 = 98 < 100$

`range(0, 100, 7)`  $\leftrightarrow$  `[0, 7, 17, ... 98]`.

La syntaxe générale est `range([start,] stop[, step])`

Il y a deux arguments optionnels, on ne peut pas choisir le deuxième sans le premier.

On retrouve très souvent `range(.)` dans les structures en boucle `for`.

## VII] Les boucles

L'outil de base en programmation, pour répéter, est la boucle. Il y a les boucles décrivant un tableau : boucle `for`. Il y a les boucles fonctionnant suivant une condition : boucle `while`.

### VII-a) Boucles `for`

Pour  $x \in [0; 3]$  calculer et afficher  $5x^2 + 3x - 1$  se traduit par `for x in [0, 1, 2, 3]:` calculer...

Comme dans l'exemple réel :

```
>>> for x in [0, 1, 2, 3]:
      print(5*x**2 + 3*x - 1)

-1
7
25
53
```

Ou avec `range(.)`

```
>>> for x in range(4): # de 0 à 3<4
      print(5*x**2 + 3*x - 1)

-1
7
25
53
```

Exemple : Tester ceci dans un fichier.

```
1 réponse = input("Donne un entier positif : ")
2 monEntier = int(réponse) # Pour obtenir un entier
3 for x in range(monEntier+1): # x, z, c ou truc ...
4     print("Si x =", x, "alors -x =", -x)
5     print("Si x =", x, "alors (-x)^2 =", (-x)**2)
6     print("") # Une ligne vide sera affichée
7 print("Nous avons fait", monEntier+1, "boucles,")
8 print("nous le savions dès la ligne 2,")
9 print("ceci ne sera affiché qu'une fois")
```

Dans un programme, les boucles sont délimitées par des parties indentées.

La boucle `for` commence en ligne 3, et se termine en ligne 6.

L'indentation est une spécificité de Python, cela offre un code clair et court.

## VII-b) Boucle while

Une boucle qui tourne **tant que** (*while*) une condition est remplie se construit suivant le modèle :

```
>>> while condition :  
        instruction 1  
        instruction 2...
```

```
1 ça_repart = True # ça_repart est un booléen, ici vrai  
2 while ça_repart: # pas de cédille avec python2 !!!  
3     réponse = input("Donne un nombre entre 10 et 20 :")  
4     nombre = float(réponse)  
5     ça_repart = (nombre<10 or nombre>20)  
6 print("Il manque",20-nombre,"pour arriver à 20")
```

Explications ligne par ligne :

1. On crée une variable booléenne (vrai ou faux — **True or False**), ici nommée `ça_repart` affectée à **True**, ainsi la boucle pourra se faire au moins une fois.
2. Tant que (**while**) `ça_repart` est vrai (**True**), alors :
3. On demande un nombre, mais on obtient une chaîne de caractères.
4. On transforme cette chaîne en nombre à virgule (**float**).
5. On teste si le nombre est trop petit ou (**or**) trop grand, (on voulait un nombre entre 10 et 20) si oui alors `ça_repart` ← **True**, sinon `ça_repart` ← **False** (alors le nombre est bon).

La boucle a fait un tour, si `ça_repart` est vrai, on recommence la boucle.


Si `ça_repart` est faux, on passe à la suite du programme.

6. Fin de l'indentation, la boucle est finie, on écrit un petit message avec un calcul.

Avec tout ça on peut faire les exercices sur la **feuille n° 3** et la **feuille n° 4**.


## VIII] Le module turtle

Afin de travailler sur des algorithmes avec des boucles, un moyen intéressant est de construire des figures ; c'est plus visuel que des suites de nombres. On peut obtenir des figures simples mais aussi des fractales.

 Python 3 possède un module de dessin adapté à l'apprentissage de l'algorithmique : la tortue (*turtle*).

On lui donne des instructions graphiques : avance ou recule, tourne à gauche ou à droite... et laisse une marque lors de son passage.

### VIII-a) Premières instructions



#### Découverte du module turtle

```
up() # relève le crayon  
forward(50) # avance de 50 pixels  
left(60) # tourne à gauche de 60°  
down() # repose le crayon  
circle(+100, 37) # trace un arc de cercle  
# vers la gauche (+), de rayon 100 pixels,  
# et d'angle 37°.  
circle(-50, 180) # arc à droite (-),  
# rayon 50, demi-tour (180°)  
pensize(5) # change l'épaisseur du  
# crayon, ici 5 pixels  
right(25) # tourne à droite de 25°  
backward(30) # recule de 30 pixels  
towards(0,0) # tourne en direction de 0.  
  
mainloop() # pour finir avec la tortue
```

Rangés par catégorie :

- `left(x)` tourne à gauche de  $x$  degrés.
- `right(x)` tourne à droite de  $x$  degrés.
- `up()` `down()` lève ou pose le crayon.
- `forward(x)` avance de  $x$  pixels.
- `backward(x)` recule de  $x$  pixels.
- `circle(±R, x)!` trace un arc de  $x$  degrés à gauche (+) ou à droite (-) de rayon  $R$ .
- `circle(±R)` pour le cercle complet.
- `pensize(x)`  $\mapsto$  l'épaisseur du trait à  $x$  pixels.
- `speed(x)` change la vitesse de la tortue,  $x$  de 1 (lent) à 10 (rapide).  
Ou `speed(0)` ultra rapide, pas d'animation.

Avec ces instructions, on peut découvrir visuellement le fonctionnement des boucles.

Vous pouvez faire les exercices de la **feuille n° 5**.

## VIII-b) Autres instructions pour turtle

Créons une fonction `segment(...)` qui enregistre les paramètres de la tortue, trace ailleurs un segment, et rétablit les paramètres.

```
def segment(x1, y1, x2, y2, ma_couleur):
    départ = pos() # enregistre la position de départ, pour la rétablir en suite
    orientation = heading() # enregistre l'orientation de la tortue
    posé = isdown() # dit si le crayon était posé ou non
    ancienne_couleur = pencolor() # renvoie la couleur du crayon
    # c'est parti
    up() # on relève
    setpos(x1, y1) # on va en (x1, y1)
    pencolor(ma_couleur) # couleur demandée
    down() # on pose le crayon
    dot() # marque un petit disque
    setpos(x2, y2) # on va en (x2, y2)
    dot() # un autre
    # le segment est tracé !
    up() # on relève
    setpos(départ) # on retourne au point de départ
    setheading(orientation) # on oriente la tortue comme avant
    pencolor(ancienne_couleur)
    if posé: # s'il était posé,
        down() # on repose le crayon

# Utilisation
circle(20, 127) # Un arc de cercle débuté
segment(44, 55, 144, 155, "red") # Un segment rouge est tracé
circle(20, 53) # l'arc de cercle se poursuit normalement.

mainloop()
```

À retenir : `pos` : position, `heading` : orientation, `set` : placer, `pen` : crayon, `color` : couleur.

`towards(M)` renvoie la direction du point *M*, ainsi `setheading(towards(M))` tourne dans la direction de *M*.

```
def carré_Plein(a, ma_couleur):
    fillcolor(ma_couleur)
    begin_fill()
    for _ in range(4):
        forward(a)
        left(90)
    end_fill()
    left(45)
    forward(1.4142*a)
    write("Carré")

# Utilisation
if not(isvisible()):
    showturtle()
carré_Plein(50, "blue")
hideturtle()

mainloop()
```

Les instructions placées entre `begin_fill()` et `end_fill()` définissent une région qui sera remplie avec la couleur choisie.

`fillcolor("blue")` prépare le remplissage en bleu.

Pour choisir une couleur on peut utiliser le système (r,g,b) : (red, green, blue).

`ma_couleur = (r, g, b)` où *r*, *g*, *b* sont des nombres entre 0 et 1.

`ma_couleur = (0, 0, 0)` rien, c'est noir.

`ma_couleur = (1, 0, 0)` que du rouge.

`ma_couleur = (1, 1, 0)` rouge et vert : jaune

`ma_couleur = (1, 1, 1)` rouge, vert et bleu : blanc

C'est une méthode additive des couleurs, pas comme avec des tubes de peintures où les couleurs se soustraient.

`isvisible()` renvoie **True** ou **False**.

`show-` (montre) et `hide-` (cache) la -turtle (tortue).

Pour plus de fonctions et de précisions, voir la [documentation officielle](#) du module `turtle`.

Vous pouvez faire les exercices de la [feuille n° 6](#). (et la [feuille n° 7](#) – difficile)

## IX] Approche de la récursivité



### Définition

La récurrence et la récursivité sont des notions proches, l'apprentissage de l'une aide à comprendre l'autre. Voici une suite définie par récurrence :  $u_0 = 7$  et pour  $n \in \mathbb{N}^*$ , on a :  $u_n = 3 \times u_{n-1} + 1$ .

Si on demande de calculer  $u_5$ , on peut avoir deux approches :

1.  $u_0 = 7$ , donc  $u_1 = 3 \times 7 + 1 = 22$ , donc  $u_2 = 3 \times 22 + 1 = 67$ , donc  $u_3 = 3 \times 67 + 1 = 202$ , donc  $u_4 = 3 \times 202 + 1 = 607$ , donc  $u_5 = 3 \times 607 + 1 = 1822$ .
2.  $u_5 = 3 \times u_4 + 1$ , donc je demande la valeur de  $u_4$ . Or  $u_4 = 3 \times u_3 + 1$ , donc je demande la valeur de  $u_3$ , etc. Lorsque  $u_0$  est demandé, on répond 7, donc on peut, en remontant, calculer  $u_5$ .

La récursivité fonctionne suivant la deuxième approche, c'est un outil très puissant, parfois indispensable et qui souvent simplifie un programme. En Python, on peut programmer cette suite ainsi :

```
def suiteU(n):  
    if n==0:  
        return 7  
    else:  
        return 3*suiteU(n-1)+1
```

```
# suite: utilisation  
nombre = suiteU(5)  
print("u(5) est égal à", nombre)
```



### Exemples

Un bon exemple :

```
def facto(n):  
    if n==0:  
        return 1  
    else:  
        return n*facto(n-1)
```

Calculer à la main `facto(5)`.

N'oubliez pas que l'ordinateur ne conserve pas en mémoire ce qui n'a pas été demandé ; ainsi `fibonacci(3)` sera appelé deux fois, une première fois pour `fibonacci(5)`, et encore une fois pour `fibonacci(4)`, en recommençant tous les calculs!!! Exercice : reprogrammer `fibonacci(n)` de manière plus efficace.

Un mauvais exemple :

```
def fibo(n):  
    if n<2:  
        return n  
    else:  
        return fibo(n-1)+fibo(n-2)
```

Calculer à la main `fibonacci(5)`.

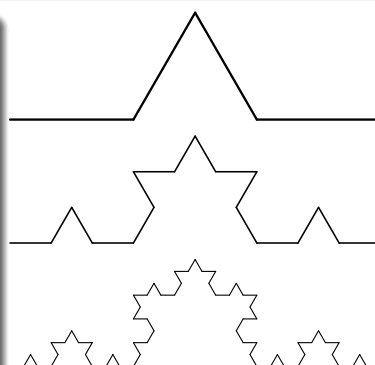


### Des fractales

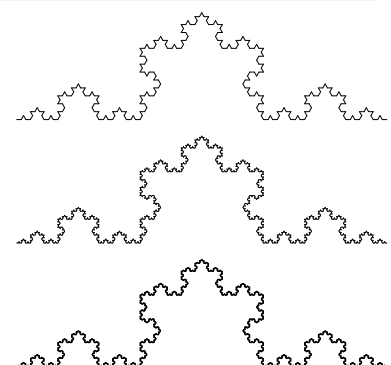
La courbe de VON KOCH

```
# avec la tortue  
def koch(x, n):  
    if n==0:  
        forward(x)  
    else:  
        koch(x/3, n-1)  
        left(60)  
        koch(x/3, n-1)  
        right(120)  
        koch(x/3, n-1)  
        left(60)  
        koch(x/3, n-1)  
koch(270,6) # Appel n°1
```

Tracer à la main `koch(9, 0)`, puis `koch(9, 1)`, puis `koch(9, 2)` et `koch(9, 3)`.



Étapes 1 à 6 de la courbe de VON KOCH.





À l'étape  $n$ , chaque segment de l'étape 1 est remplacé par une réduction au tiers de l'étape  $n-1$ . C'est un appel récursif. Lorsqu'on appelle `koch(270,6)`, on souhaite une courbe qui a une base de 270 pixels de large, et une **profondeur de récursion** de 6 ; il y a 6 étapes.

Mais on peut aussi considérer que chaque segment de l'étape  $n$  se transforme en un chemin de l'étape 1 ; un triangle équilatéral est placé au tiers du segment.


Vous pouvez faire la construction de l'arbre de PYTHAGORE avec l'aide de la **feuille n°8**.

## X] Pour aller plus loin

### X-a) Différences entre Python 2 et 3

Nous avons choisi  Python 3 parce qu'il est très répandu et idéal pour débiter. Mais c'est la version 2 qui est aujourd'hui la plus répandue, et de loin. Pourquoi débiter avec  Python 3 ?


1. Guido VAN ROSSUM, le créateur de Python, le conseille.
2. L'encodage des fichiers est naturellement en UTF-8, il n'y a donc plus aucun problème d'accent.
3. `print( . )` est une fonction comme les autres, son fonctionnement est simplifié et amélioré.
4. La division est redéfinie, elle n'est plus source d'erreurs : `13/3 == 13.0/3` mais faux avec la version 2 !
5. Il n'y a pas de limite aux nombres entiers, seuls le temps de calcul et la capacité d'affichage sont un frein.
6. De manière générale le code a été épuré. . .


Ainsi la version d'avenir, pour un débutant est  Python 3.

Les détracteurs répondront que toutes les bibliothèques de la version 2 n'ont pas encore été portées à la version 3 ; les débutants s'en moquent, cela ne les concernerait que dans plusieurs années, et alors tout aura été porté.

### X-b) Des ouvrages de référence

En français :


Un cours complet facile sur  Python 3 : faire une recherche Google avec [apprendre python 3](#).


Et télécharger gratuitement le livre libre *Apprendre à programmer avec  Python 3*, de Gérard SWINNEN.

Pour les étudiants, une approche plus rapide, proposée par [Bob CORDEAU](#) sur l'afpy.

En anglais :


Un autre cours, très détaillé et facile : chercher [non-programmer tutorial python 3](#).

La [documentation officielle](#) de  Python 3 propose un tutoriel et toutes les références possibles.

Enfin, pour ceux qui connaissent déjà l'algorithmique et souhaite apprendre  Python 3 plus vite : [Plongez au cœur de Python 3](#).

## A Annexe : Feuilles d'exercices

Les pages suivantes sont des feuilles d'exercices de difficulté variée.

Pour comparer  Python 3 à une calculatrice : [feuille n° 1](#)

Pour faire son premier programme : [feuille n° 2](#)

Pour travailler sur la boucle `while` : [feuille n° 3](#)

Pour travailler sur la boucle `for` et les tableaux : [feuille n° 4](#)

Pour faire vos premiers dessins avec la tortue : [feuille n° 5](#), [feuille n° 6](#) et [feuille n° 7](#)

Pour construire un arbre de PYTHAGORE : [feuille n° 8](#)

3. Pour plusieurs valeurs de  $x$ , calculer **progA**( $x$ ) et **progB**( $x$ ). Quelle conjecture faire?
4. Démontrer cette conjecture. (Niveau 4<sup>e</sup>-3<sup>e</sup>)
5. Écrire **progC**( $x$ ) en Python une fonction d'un argument  $x$ , qui calcule  $C = x(3x + 8) - 15$ .  
*Attention, ici on ne veut qu'une seule opération mathématique par instruction. C'est la règle du jeu, certes stupide !*
6. Même chose avec  $D = (5x - 2)(2x + 8)$



**A2 : mordu ?****Exercice n° 1 : Tout ça pour ça ?**

On suppose qu'au départ  $A = 1$ ,  $B = 1$ ,  $C = 1$ .

On lance l'algorithme suivant :

N°	Instructions	A	B	C
1	$A \leftarrow 2$	2	1	1
2	$B \leftarrow 10$			
3	$C \leftarrow A + B$			
4	$C \leftarrow B - C$			

1. Compléter le tableau.
2. Refaire l'exercice en remplaçant l'instruction 1 :  
 $A \leftarrow 2$  par  $A \leftarrow 5$ , puis par  $A \leftarrow (-3)$ .
3. Quelle conjecture faites-vous ?  
Démontrez-la avec  $A \leftarrow x$  à l'instruction 1.

**Exercice n° 2 : Compléter le tableau**

	X
Instructions	0
$A \leftarrow 1$	
$B \leftarrow 9$	
$C \leftarrow 4$	
$D \leftarrow 8$	
$X \leftarrow A$	
$X \leftarrow X * 10 + B$	
$X \leftarrow X * 10 + C$	
$X \leftarrow X * 10 + D$	

Conseil : écrire à droite, ligne par ligne, les valeurs de A, B, C et D.

**Exercice n° 3 : Échange de variables ?**

	A	B
Instructions	a	b
$A \leftarrow B$		
$B \leftarrow A$		

1. Que fait cet algorithme ?
2. En utilisant une troisième variable C, écrire un algorithme qui permute les valeurs des variables A et B.
3. [Bien plus dur] Et sans utiliser d'autre variable ?

**Exercice n° 4 : B = A ?**

	A	B
Instructions	a	b
$A \leftarrow A + B$		
$B \leftarrow A - B$		
$A \leftarrow A - B$		

Que fait cet algorithme ?

**Exercice n° 5 : Permutation circulaire**

Donner un algorithme qui réalise ceci :

Situation initiale :

A	B	C
a	b	c

Situation finale :

A	B	C
c	a	b

On utilisera une autre variable D.

**Exercice n° 6 : La fonction carré**

Avec Python (*shell*), définir la fonction carré.  
(/!\ Avec Python 2, accents interdits !)

```
>>> def carré(x):
    return x*x

>>> carré(11) # le carré du nombre 11
121
```

Calculer les carrés des entiers jusqu'à 13.  
Il est conseillé de connaître ces nombres.

**Exercice n° 7 : Premier programme**

Voici un programme de calcul :

- ↪ Choisir un nombre entier N
- ↪ Lui ajouter 4
- ↪ Multiplier la somme obtenue par N
- ↪ Ajouter 4 à ce produit.

1. Faire fonctionner « à la main » ce programme de calcul pour les entiers N compris entre 0 et 6.
2. Faire une conjecture.
3. Traduire ceci en un algorithme, qui contiendra exactement trois instructions d'affectation.
4. Programmer ceci en Python, vérifier pour  $N \in \llbracket 0; 6 \rrbracket$ .  
Tester d'autres valeurs pour conforter votre conjecture.
5. Prouver la conjecture. (Ça, c'est des maths)

### A3 : recherche **while** désespérément

#### Exercice n° 1 : S'arrêtera ?

```
N = 153
i = 121
while (N>i):
    N=2*N-5
    i=2*N+7
    print(N, i)
```

Ce programme fait

- $N \leftarrow 2 \times 153 - 5 = 301$
- $i \leftarrow 2 \times 121 + 7 = 249$
- Affichage de : 301 249
- $301 > 249$  donc on boucle à nouveau
- $N \leftarrow 2 \times 301 - 5 = 597 \dots$

Écrire la suite.



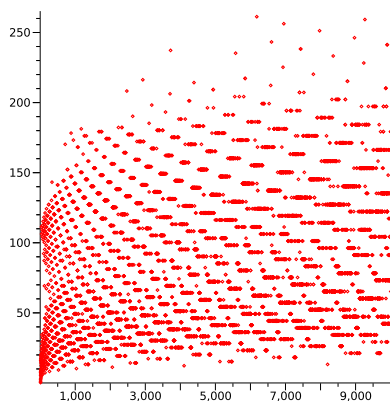
#### Exercice n° 2 : Liste des diviseurs

```
N = 154
i = 1
while (i**2<N):
    if (N%i==0):
        print(i, "*", N/i, "=", N)
    i=i+1
print(N, "possède ? diviseurs.")
```

1. Que fait ce programme ? Écrire toutes les étapes.  
Aide : % donne le reste d'une division euclidienne.
2. Modifier le pour qu'il affiche un nombre correct au lieu de « ? ».
3. Vérifier avec  $N = 7$ ,  $N = 8$  et  $N = 9$  après avoir fait le calcul à la main.

#### Exercice n° 3 : La suite de Syracuse

1. Comment en Python peut-on savoir si un entier est pair ou impair ?  
(Penser à % : reste de division euclidienne)
2. Voici une suite : 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.  
Règle : on part de  $N = 14$ , qui est pair, donc on le divise par 2.  
On obtient 7 qui est impair, dans ce cas on calcule  $3 \times 7 + 1 = 22$ .  
Et on recommence : pair ou impair, tant que le résultat n'est pas égal à 1.  
Personne ne sait si cette suite s'arrête pour tout entier  $N$  au départ, on conjecture que oui.  
Écrire en Python cette boucle avec  $N = 14$  au départ.
3. Essayer avec d'autres valeurs de départ, comme  $N = 27$  par exemple.
4. Modifier votre programme pour afficher le nombre de boucles (qu'on appelle *durée de vol*), et la valeur maximale atteinte (qu'on appelle *altitude maximale*).  
Vérifier que pour  $N = 14$ , la durée de vol est 17, et l'altitude maximale est 52.
5. Quel est le plus petit entier  $N$  qui donne une durée de vol supérieure ou égale à 100 ?
6. Quel est le plus petit entier  $N$  qui donne une altitude maximale supérieure ou égale à 9232 ?



7. Ce graphique est en lien avec l'exercice.  
Trouver lui une légende et un titre.

## **A4 : Tableaux et boucle `for`**

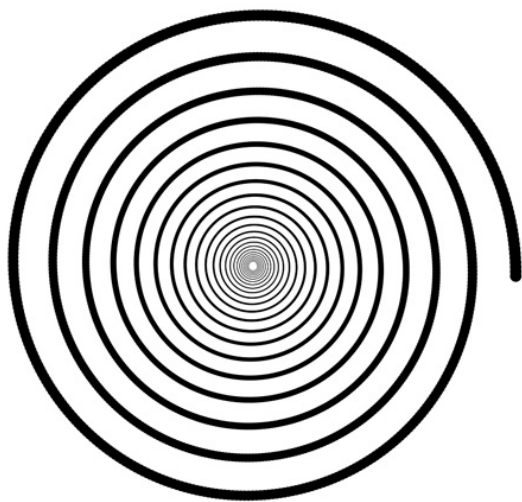
à suivre

## **A5 : Premières figures**

à suivre

**A6 : Tournicoti**

Objectif : construire une spirale  
avec le module `turtle`

**Exercice n° 1 : Première tentative**

1. Faire le programme suivant :
  - Choisir un rayon  $R$  de 200 pixels
  - faire un quart de cercle avec ce rayon
  - diminuer le rayon de 3 pixels
  - recommencer en boucle 50 fois.
2. Que se passe-t-il si on fait 100 boucles ?
3. Au lieu de faire une réduction de 3 pixels par quart de cercle, faire une réduction de 2 % par quart de cercle.
4. Modifier le programme pour avoir un trait d'épaisseur 5 pixels au départ, et 1 pixel à la fin de la boucle.

*Aide : vous aurez surement besoin d'arrondir un nombre.*

`from math import *` # Ça sert toujours  
`ceil(2.568)` renvoie 3, l'entier arrondi par excès.

Cette spirale n'était qu'une approximation d'une spirale d'ARCHIMÈDE.  
Essayons maintenant l'approximation d'une spirale de FIBONACCI.

**Exercice n° 2 : La suite de FIBONACCI**

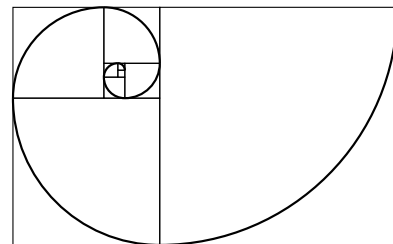
Voici la suite de FIBONACCI : 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

1. Vérifier que chaque terme est la somme des deux précédents (s'ils existent).
2. Calculer de tête les huit termes suivants (et les écrire).
3. Faire un programme Python qui stocke dans un tableau **N** termes de la suite à partir de [1, 2, ...]. (*0 n'est pas un bon candidat pour un côté.*)

*Aide : Créer un tableau `Tablo=[1, 2]` avec les deux premières valeurs.*

`Tablo.append(3)` ajoute la valeur 3 à la suite de votre tableau.

Spirale d'Or



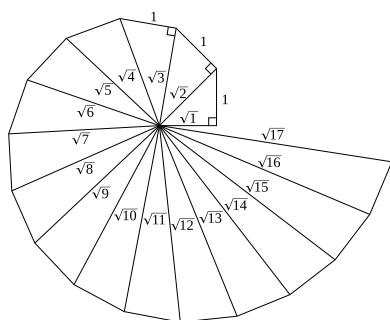
Les côtés des carrés sont des termes de la suite de FIBONACCI.

Pour faire une approximation de la spirale d'Or, on peut faire des quarts de cercle dans chaque carré.

**Exercice n° 3 : Votre spirale d'Or**

On choisit **N=15**.

1. Avec l'exercice précédent, créer un tableau de **N** valeurs successives de la suite de FIBONACCI, en commençant par [1, 2, ...].
2. En appliquant un coefficient de proportionnalité, construire un tableau dont le plus grand élément est égal à la moitié de la hauteur de votre écran en pixel.
3. Tracer des quarts de cercle de rayons décroissants issus de votre tableau.

**Exercice n° 4 : Et THÉODORE DE CYRÈNE alors ?**

On choisit **a = 100** pixels comme côté commun et **N = 17** triangles.

1. Construire un triangle rectangle isocèle de petit côté  $a$  pixels, les coordonnées de ses sommets seront  $(0;0)$ ,  $(a;0)$ ,  $(a;a)$ .  
*Aide : (Aller-retour) ici = `pos()` et `setpos(ici)` sont vos amis.*
2. Placée en  $(a,a)$ , l'instruction `setheading(towards(0,0))` dirige la tortue vers l'origine. La tourner de  $90^\circ$  et construire le triangle suivant.
3. Construire cette spirale (dite de THÉODORE DE CYRÈNE).
4. Prendre **a = 10** et **N = 500** et ne plus tracer l'hypoténuse.

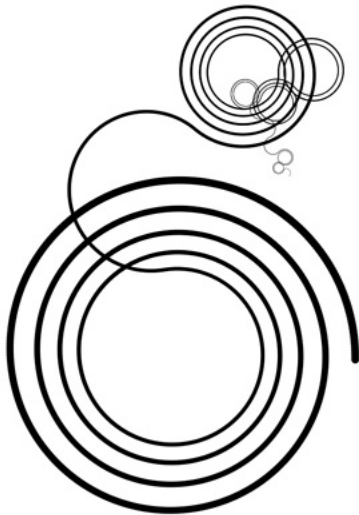


Coupe d'un nautilus.

## A7 : Tournicoton

## Feuille très difficile

Objectif : modifier une spirale construite précédemment.



## Exercice n° 1 : Première bascule de spirale

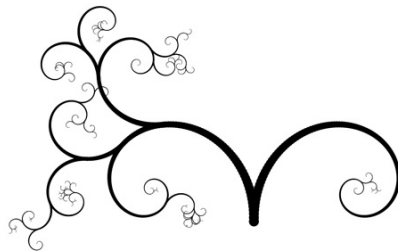
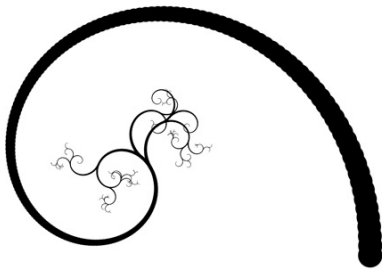
```
from random import *
# idem : turtle, math
R=800/4 # écran : 800 pixels de haut ?
e=3 # pourcentage de réduction de R : 3%
N=100 # nombre de quarts de tour
for i in range(1,N):
    pensize(ceil(4*(1-i/N)))
    circle(R, 90)
    R=R*(100-e)/100

mainloop()
```

Ce programme est une solution à l'exercice 1 de la feuille précédente. Modifier le pour introduire une bascule aléatoire de côté pour la courbure.  $R = -R$

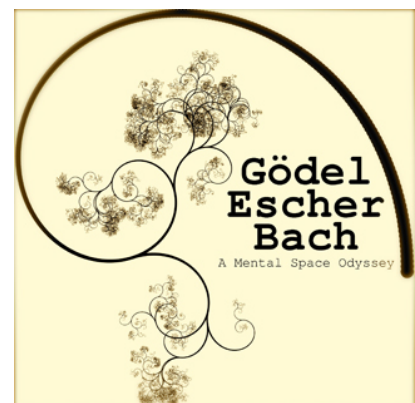
Pour forcer un caractère aléatoire visuellement agréable, le test `random() < (0.125*i/N)` est intéressant ; sa probabilité de succès est nulle à la première boucle, et de  $\frac{1}{4}$  à la dernière.

## Exercice n° 2 : Une branche récurrente



Reprendre l'exercice avec la spirale de FIBONACCI, et la modifier afin d'obtenir, de manière aléatoire, des branches du même genre qu'au dessus.

Inspiration :



## A8 : Arbre de Pythagore

On construit des rectangles de même format  $k = \frac{AB}{AC}$ , autour d'un triangle rectangle.

1. Démontrer que  $\mathcal{A}_{\text{Rectangle 1}} = \mathcal{A}_{\text{Rectangle 2}} + \mathcal{A}_{\text{Rectangle 3}}$
2. Construire cette figure, avec les contraintes suivantes :
  - On ne trace pas deux fois le même segment,
  - on ne relève pas le crayon.

On choisira  $k = 0,9$  et  $\widehat{CDE} = 30^\circ$ .

Aide : `from math import *` #est toujours votre ami  
`cos(radians(30))` renvoie le cosinus de 30 degrés.

Essayer `cos(30)` est une mauvaise idée.

Par défaut `cos()`, `sin()`, `tan()` attendent des radians.

3. Modifier votre programme afin que les tracés de  $[CF]$ ,  $[FG]$ ,  $[GE]$  et de  $[EH]$ ,  $[HI]$ ,  $[ID]$  se fassent par deux appels à une même procédure.
4. Modifier votre programme afin d'obtenir  $N = 9$  en « profondeur » de branches dans votre arbre de PYTHAGORE.
5. Tester d'autres valeurs pour  $N$  et  $k$ .
6. Mettre de la couleur dans votre arbre.

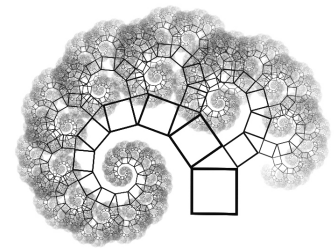
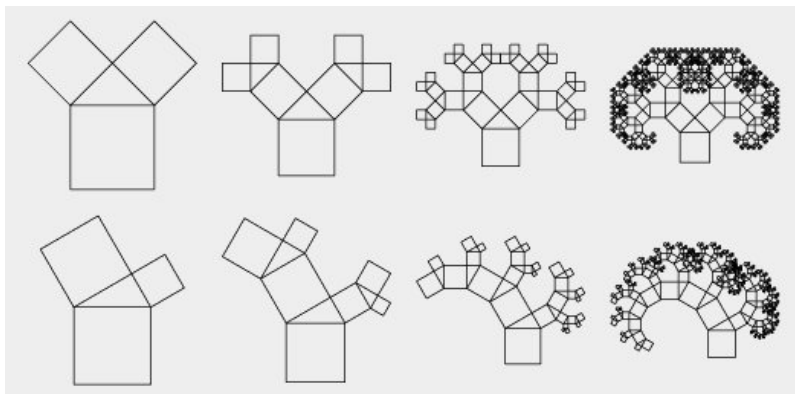
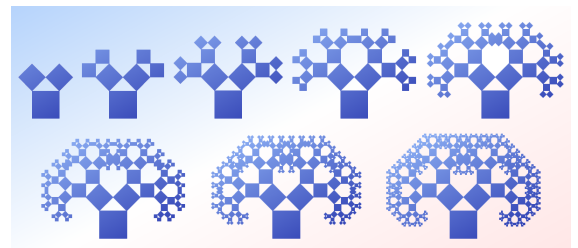
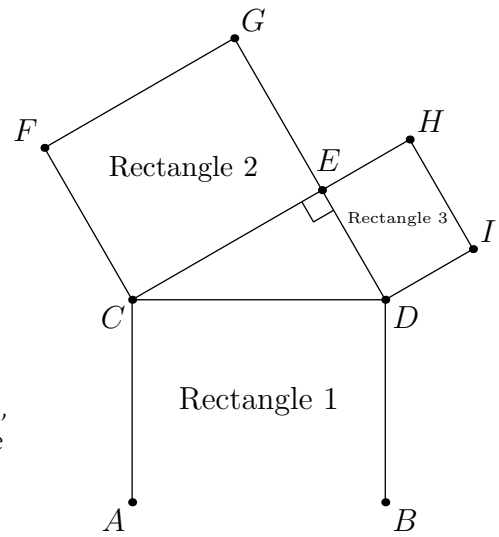
Exemple : Si vous utilisez :

```
def maCouleur(n):
    pencolor(0, n/N, 0) # (r g b)
```

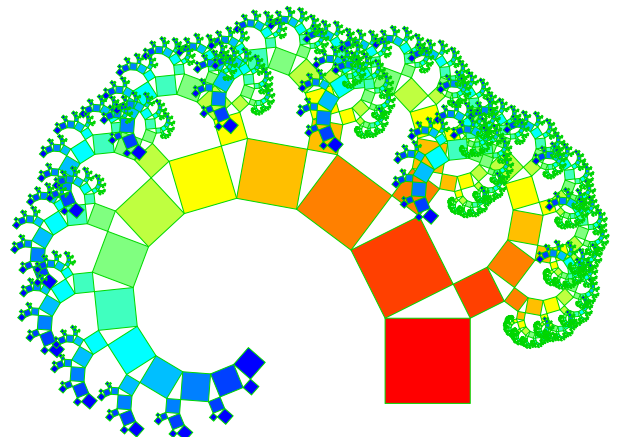
aux étapes  $n$ , vous aurez un arbre dont le tronc est noir  $(0,0,0)$ , et des feuilles vertes  $(0, 1, 0)$ .

Vous pouvez mettre toute formule à votre goût.

`pencolor` attend trois nombres entre 0 et 1 pour le rouge (*red*), le vert (*green*) et le bleu (*blue*).



7. Pour remplir les rectangles de couleur, vous devrez réécrire une partie du cœur, vous pourrez tracer des segments deux fois.





## B Sur ce document

### B1 : Licence option 1

DO WHAT THE FUCK YOU WANT TO PUBLIC LICENSE  
Version 2, December 2004

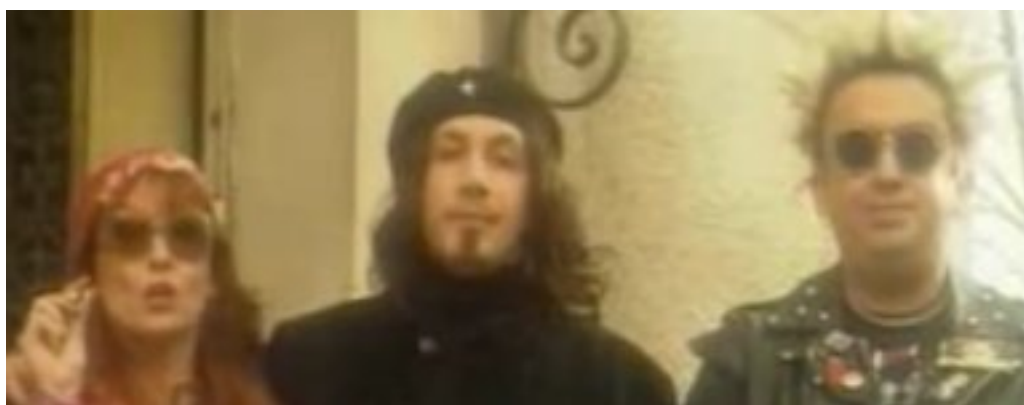
Copyright © 2004 Sam Hocevar

14 rue de Plaisance, 75014 Paris, France

*Everyone is permitted to copy and distribute verbatim or modified copies of this license document, and changing it is allowed as long as the name is changed.*

DO WHAT THE FUCK YOU WANT TO PUBLIC LICENSE  
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

o. You just DO WHAT THE FUCK YOU WANT TO.



### B2 : Licence option 2



Cet article est publié sous la licence Creative Commons-BY-NC-SA :

BY [Paternité] Vous devez citer le nom de l'auteur original

NC [Pas d'Utilisation Commerciale] Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.

SA [Partage des Conditions Initiales à l'Identique] Si vous modifiez, transformez ou adaptez cette création, vous n'avez le droit de distribuer la création qui en résulte que sous un contrat identique à celui-ci.

### B3 : Licence option 3

Copyright © 2011 Franck CHAMBON

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Lesser General Public License (see the file LICENSE in the top-level source directory).

### B4 : Auteur

Franck CHAMBON : <[Franck.Chambon@ac-grenoble.fr](mailto:Franck.Chambon@ac-grenoble.fr)>

Remerciements à IsaT, fidèle lectrice.