TP – CHAPITRE 2 - Récursivité

Sommaire

TP1 - factorielle

Définition : Pour tout entier naturel n la factorielle de n est définie par :

$$n\,! = 1 \times 2 \times 3 \times ... \times n \ .$$

- Calculer ci-dessous, les premières valeurs de la factorielle :

  $0\,! =$; $1\,! =$; $2\,! =$; $3\,! =$; $4\,! =$; $5\,! =$;

- Compléter ci-dessous la formule de récurrence de la factorielle, c'est à dire l'égalité donnant $n\,!$ en fonction de $(n-1)\,!$:

  $$n\,! =$$

- Sur votre compte, dans le dossier **NSI** , créer un sous-dossier **TP-Chapitre2** dans lequel on rangera les TP de ce chapitre.
- Récupérer le fichier **TP1 .py** donné en ressource et l'enregistrer dans le dossier **TP-Chapitre2** sur votre compte.
- Compléter le **pass** c'est à dire le code de la fonction récursive **factorielle(n)** .
- L'affichage attendu dans la console est suivant.

```
Factorielle(0) = 1
Factorielle(1) = 1
...
Factorielle(6) = 720
```

Lever la main pour valider ce TP.

TP2 - Somme

Définition : Pour tout entier naturel n on considère la somme S_n définie par :

$$S_n = 1 + 2 + 3 + ... + n \ .$$

- Calculer ci-dessous, les premières valeurs de cette somme :

$S_0 =$; $S_1 =$; $S_2 =$; $S_3 =$; $S_4 =$; $S_5 =$;

- Compléter ci-dessous la formule de récurrence de cette somme, c'est à dire l'égalité donnant S_n en fonction de S_{n-1} :

$$S_n =$$

- Récupérer le fichier **TP2 .py** donné en ressource et l'enregistrer dans le dossier **TP-Chapitre2** sur votre compte.
- Compléter le **pass** c'est à dire le code de la fonction récursive **somme(n)** .
- L'affichage attendu dans la console est suivant.

```
Somme(0) = 0
Somme(1) = 1
...
Somme(10) = 55
```

Lever la main pour valider ce TP.

TP3 - PGCD

Propriété : Pour tout entier naturel a et b on a :

$$\text{pgcd}(a, b) = \text{pgcd}(b, r)$$

avec $r = a \% b$: le reste de la division euclidienne de a par b.

- Déterminer ci-dessous, certains pgcd :

  pgcd (15, 6) = ; pgcd (84, 36) = ; pgcd (30, 165) = ;

- Récupérer le fichier **TP3 .py** donné en ressource et l'enregistrer dans le dossier **TP-Chapitre2** sur votre compte.
- Compléter le **pass** c'est à dire le code de la fonction récursive **pgcd(a, b)** .
- L'affichage attendu dans la console est suivant.

```
pgcd(18, 63) = 9
pgcd(143, 221) = 13
pgcd(31, 59) = 1
pgcd(561, 1071) = 51
```

Lever la main pour valider ce TP.

TP4 - Suite de Fibonacci

- La suite de Fibonacci (f_n) est définie par : $f_0 = 0$; $f_1 = 1$ et $f_n = f_{n-1} + f_{n-2}$ pour tout entier $n \geqslant 2$.
- Calculer les premiers termes de la suite de Fibonacci :

  $f_2 =$; $f_3 =$; $f_4 =$; $f_5 =$; $f_6 =$; $f_7 =$;

  $f_8 =$; $f_9 =$; $f_{10} =$.

- Dans Thonny créer un nouveau fichier nommé **TP4 .py** et l'enregistrer dans le dossier **TP-Chapitre2** sur votre compte.
- Ecrire une fonction récursive **f(n)** qui calcule et affiche le terme de rang n de la suite de Fibonacci.
- Rédiger ensuite une boucle **for** qui appelle la fonction **f(n)** pour n allant de 0 à 20 compris.
- L'affichage attendu dans la console est suivant.

```
f(0) = 0
f(1) = 1
...
f(20) = 6765
```

Lever la main pour valider ce TP.

TP5 - Palindrome

- Récupérer le fichier **TP5 .py** donné en ressource et l'enregistrer dans le dossier **TP-Chapitre2** sur votre compte.
- Compléter les **pass** pour que la fonction **palindrome(chaine)** renvoie **True** si la chaine est un palindrome et **False** si ce n'est pas le cas.
- L'affichage attendu dans la console est suivant.

```
False
True
True
False
```

Lever la main pour valider ce TP.

License CC BY-SA 4.0

TP6 - Suite de Syracuse

- La suite de Syracuse (s_n) est définie par : $s_0 = 10$ et $s_{n+1} = \dfrac{s_n}{2}$ si s_n est pair et $s_{n+1} = 3s_n + 1$ si s_n est impair pour tout entier $n \geqslant 0$.
- Calculer les premiers termes de la suite de Syracuse :

  $s_1 =$; $s_2 =$; $s_3 =$; $s_4 =$; $s_5 =$; $s_6 =$;

  $s_7 =$; $s_8 =$; $s_9 =$; $s_{10} =$.

- Dans Thonny créer un nouveau fichier nommé **TP6 .py** et l'enregistrer dans le dossier **TP-Chapitre2** sur votre compte.
- Ecrire une fonction récursive **s(n)** qui calcule et affiche le terme de rang n de la suite de Syracuse.
- Rédiger ensuite une boucle **for** qui appelle la fonction **s(n)** pour n allant de 0 à 15 compris .
- Le retour attendu dans la console est :

```
s(0) = 10
s(1) = 5
...
s(15) = 1
```

Lever la main pour valider ce TP.

 License CC BY-SA 4.0

TP7 - Nombre de chiffres d'un entier

- Créer un nouveau fichier **TP7 .py** dans le dossier **TP-Chapitre2** sur votre compte.
- Construire une fonction récursive nommée **nb_chiffres( )** qui prend en paramètre un entier naturel **n** et retourne le nombre de chiffres de cet entier.
- Sachant que la paramètre **n** doit être de type **int** et doit être positif, rajouter les préconditions sur le paramètre **n** , puis tester ces préconditions avec l'instruction **assert** . Pour les rappels concernant les préconditions, vous pouvez relire le cours vu en première sur les spécifications : **cours_specification.pdf** donné en ressource.
- Exemple de retours attendus dans la console :

```
>>> nb_chiffres(123456)
6
>>> nb_chiffres(-123456)
 Traceback (most recent call last):
   File "<pyshell>", line 1, in <module>
   File "/media/attar/TOSHIBA/Info/NSI/Terminale/Chapitre2_Recursiv
 ite/TP2NSI/Corrections/TP7-nb_chiffres.py", line 9, in nb_chiffres
     assert n >= 0, "Le paramètre doit être positif."
 AssertionError: Le paramètre doit être positif.
>>> nb_chiffres(-0.5)
 Traceback (most recent call last):
   File "<pyshell>", line 1, in <module>
   File "/media/attar/TOSHIBA/Info/NSI/Terminale/Chapitre2_Recursiv
 ite/TP2NSI/Corrections/TP7-nb_chiffres.py", line 8, in nb_chiffres
     assert type(n) == int, "Le paramètre doit être de type int."
 AssertionError: Le paramètre doit être de type int.
```

Lever la main pour valider ce TP.

 License CC BY-SA 4.0

TP8 - Nombre de 1 dans l'écriture binaire d'un entier

- Créer un nouveau fichier **TP8 .py** dans le dossier **TP-Chapitre2** sur votre compte.
- Construire une fonction récursive nommée **nb_1( )** qui prend en paramètre un entier naturel **n** et retourne le nombre de 1 dans l'écriture binaire de cet entier.
- Sachant que la paramètre **n** doit être de type **int** et doit être positif, rajouter les préconditions sur le paramètre **n** , puis tester ces préconditions avec l'instruction **assert** . Pour les rappels concernant les préconditions, vous pouvez relire le cours vu en première sur les spécifications : **cours_specification.pdf** donné en ressource.
- Exemple de retours attendus dans la console :

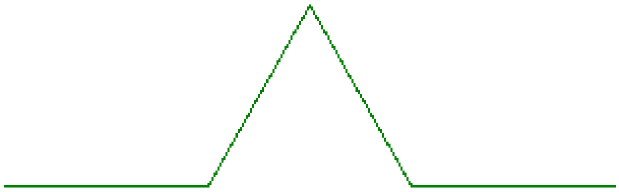
```
>>> nb_1(2)
1
>>> nb_1(15)
4
>>> nb_1(65)
2
>>> nb_1(255)
8
>>> nb_1(-5)
 Traceback (most recent call last):
   File "<pyshell>", line 1, in <module>
   File "/media/attar/TOSHIBA/Info/NSI/Terminale/Chapitre2_Recursiv
 ite/TP2NSI/Corrections/TP8-nb_1.py", line 10, in nb_1
     assert n >= 0, "Le paramètre doit être positif."
 AssertionError: Le paramètre doit être positif.
>>> nb_1(0.5)
 Traceback (most recent call last):
   File "<pyshell>", line 1, in <module>
   File "/media/attar/TOSHIBA/Info/NSI/Terminale/Chapitre2_Recursiv
 ite/TP2NSI/Corrections/TP8-nb_1.py", line 9, in nb_1
     assert type(n) == int, "Le paramètre doit être de type int."
 AssertionError: Le paramètre doit être de type int.
```

Lever la main pour valider ce TP.

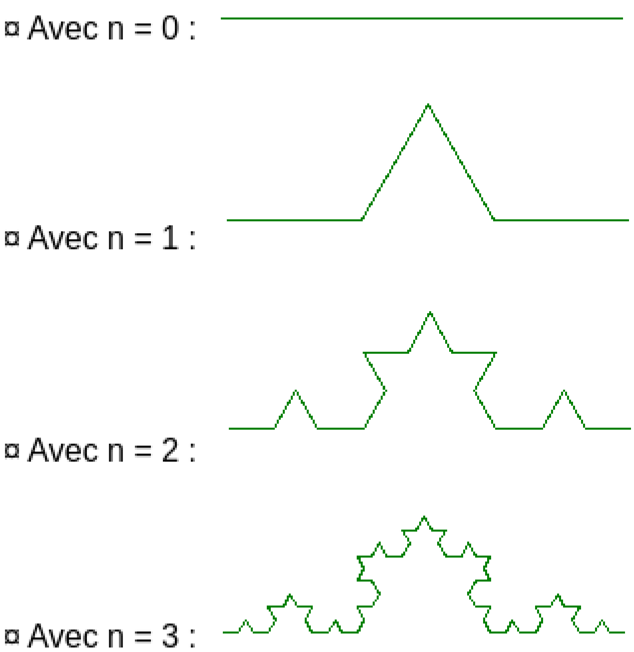
License CC BY-SA 4.0

TP9 - Courbe fractale de Koch

- Récupérer le fichier **TP9 .py** donné en ressource et l'enregistrer dans le dossier **TP-Chapitre2** sur votre compte.
- Compléter le **pass** dans le **else** de la fonction récursive **courbe_koch()** afin d'obtenir la courbe de Koch suivante au rang 1 :



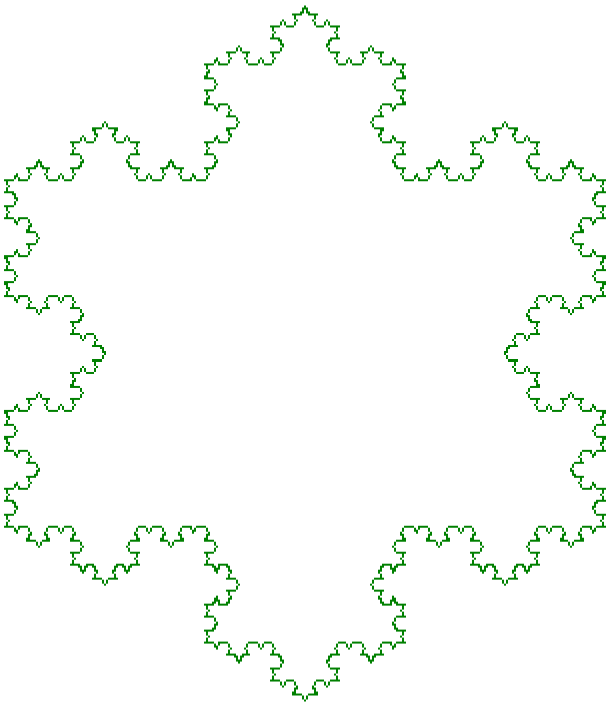
- Modifier la valeur de n1 qui correspond au rang de la courbe de Koch et vérifier qu'on obtient bien une fractale, comme ci-dessous :



Lever la main pour valider ce TP.

TP10 - Flocon de Koch

- Dupliquer le fichier **TP9 .py** en **TP10 .py** .
- Rédiger une fonction **flocon(n, longueur)** qui appelle trois fois la fonction récursive courbe_koch, afin d'obtenir le flocon de Koch.
- Modifier l'appelle de fonction à la fin du code, en appelant la fonction **flocon()** à la place de la fonction **courbe_koch()**.
- Le flocon de Koch attendu au rang n1 = 4 est suivant :

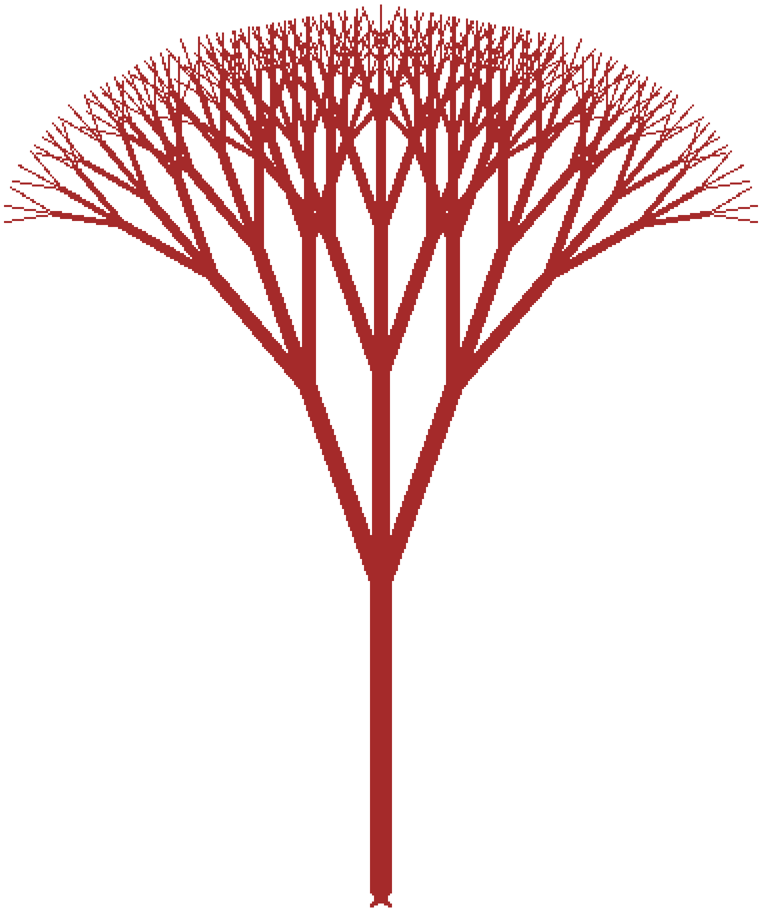


Lever la main pour valider ce TP.

License CC BY-SA 4.0

TP11 - Arbre fractale

- Récupérer le fichier **TP11 .py** donné en ressource et l'enregistrer dans le dossier **TP-Chapitre2** sur votre compte.
- Rajouter du code dans la fonction **branche(n, longueur)** afin d'obtenir un arbre fractale avec trois branches à chaque noeuds. Attention de bien gérer les rotations avec la fonction **left()**.
- L'arbre fractale attendu au rang n = 6 est suivant :

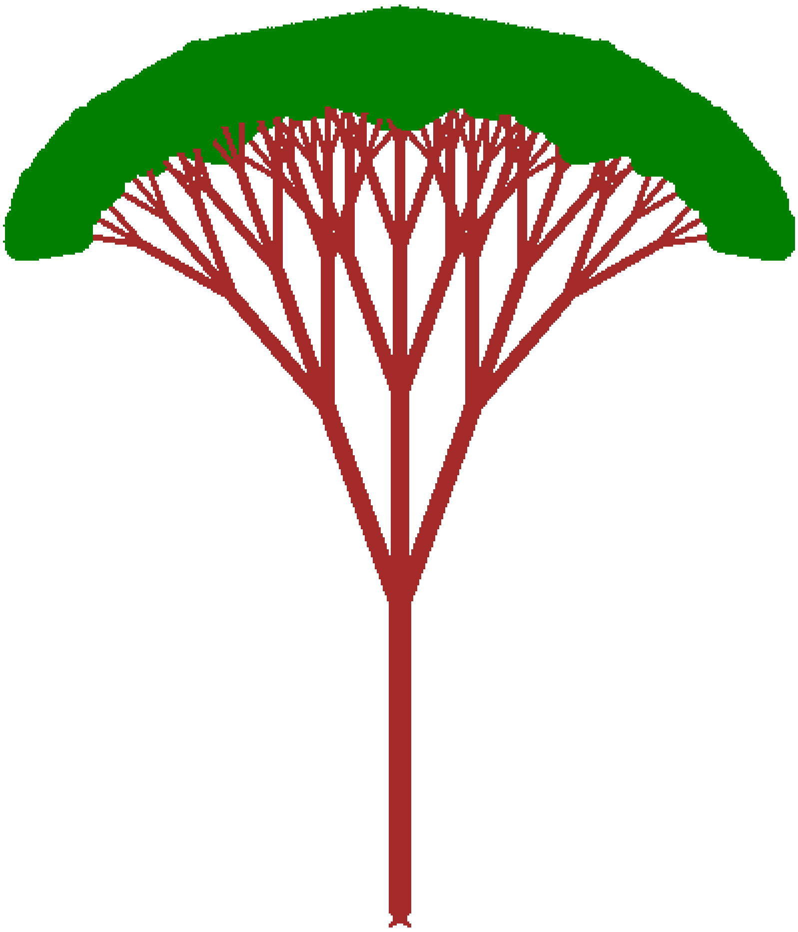


Lever la main pour valider ce TP.

Nicolas Blanc, Michel Courtieu, Lycée Anna de Noailles (Evian) License CC BY-SA 4.0

TP12 - Arbre fractale avec feuilles vertes

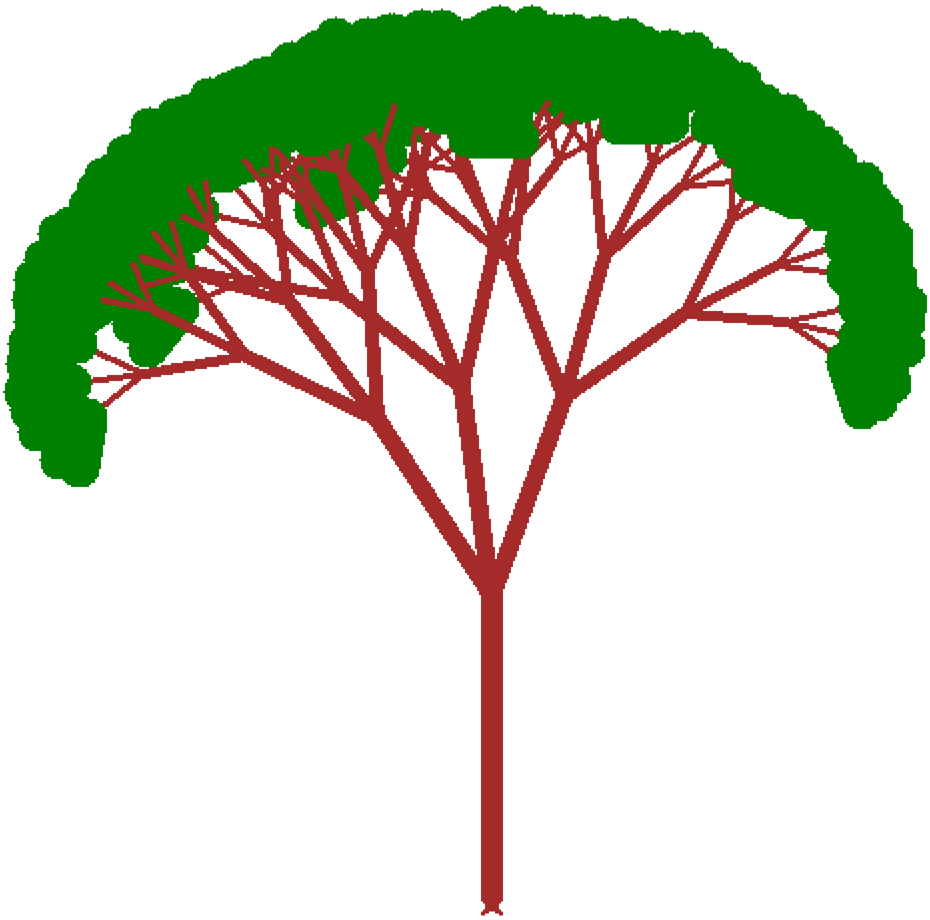
- Dupliquer le fichier **TP11 .py** en **TP12 .py** .
- Rajouter du code dans la fonction **branche(n, longueur)** afin d'obtenir un arbre fractale avec un feuillage vert au bout de branches.
- L'arbre fractale attendu au rang n = 6 est suivant :



Lever la main pour valider ce TP.

TP13 - Arbre fractale non équilibré

- Dupliquer le fichier **TP12 .py** en **TP13 .py** .
- A l'aide du module **random** et de la fonction **randint()** , modifier le code pour obtenir des angles aléatoires entre 20 ° et 40 ° entre deux branches d'un même noeuds.
  <u>Conseils :</u> Utiliser trois variables locales **angle1**, **angle2**, **angle3**, pour bien gérer les angles et bien revenir à la position initiale à la fin de la fonction récursive **branche()**.
- Un exemple d'arbre fractale attendu au rang n = 6 est suivant :



Lever la main pour valider ce TP.

 License CC BY-SA 4.0

TP14 - Mini-projet 2 : tapis de Sierpinsky

Voir le cahier des charges dans le fichier : mini_projet_2.pdf .

Lever la main pour valider ce TP.

Nicolas Blanc, Michel Courtieu, Lycée Anna de Noailles (Evian)
License CC BY-SA 4.0