

## Séance 2 : BOUCLES WHILE ET RÉCURSIONS

L1 – Université Côte d’Azur

### Création du répertoire de travail

**Avant de commencer ce TP** : sur votre ordinateur, si ce n’a pas été déjà fait au TP 1, créez un répertoire Python dans lequel vous rangerez tous vos fichiers de l’UE. Dans votre répertoire Python créez un répertoire TP2. Durant cette séance, tous vos fichiers devront être sauvegardés dans ce répertoire TP2.

Les prochaines semaines, vous devrez créer les répertoires TP3, puis TP4. Ainsi il sera facile de vous retrouver dans tous les fichiers.

#### Exercice 1 — Récursion versus boucle `while` (★)

On appelle factorielle de  $n$ , que l’on note  $n!$ , le produit  $n! = 1 \times \dots \times n$ .

1. Écrire *par récurrence* la fonction `fac_rec(n)` prenant un entier  $n \geq 0$  et retournant la factorielle de  $n$ . Testez sur `fac_rec(5)` qui vaut 120.
2. Écrire en utilisant une boucle `while`, sous la forme d’une fonction `fac(n)`. Testez sur `fac(5)`.
3. Python est-il capable de calculer  $3000!$  avec chacune de ces définitions? Pour bien comprendre ce qui pose problème, observez l’exécution de `fact_rec(5)` et `fact(5)` en mode débogage rapide (`⏏`+`F5`) puis en appuyant sur la touche `F7`.

#### Exercice 2 — Suite de Syracuse (★)

On définit la suite de Syracuse de la manière suivante. On part d’un entier  $u_0$  quelconque auquel on applique les règles de récurrence suivantes jusqu’à ce que l’on tombe sur 1.

$$u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair,} \\ 3u_n + 1 & \text{si } u_n \text{ est impair.} \end{cases}$$

Par exemple en partant de 12 on obtient la suite suivante :

$$12 \rightarrow 6 \rightarrow 3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

1. Écrire une fonction `suisvant(u)` qui **renvoie** l’élément suivant une valeur  $u$ . Par exemple `suisvant(12)` renvoie 6 et `suisvant(3)` renvoie 10. Écrire au moins quatre tests.
2. On souhaite écrire `syracuse_rec(u)` qui affiche les différentes valeurs de la suite en partant de  $u$ .
  - a) Quel est le cas d’arrêt?
  - b) Écrire la fonction `syracuse_rec(u)` qui affiche la valeur  $u$  puis fait l’appel récursif.
3. Écrire un programme `syracuse(u)` avec une boucle `while` qui affiche les différentes valeurs de la suite.
4. Modifier vos programmes précédents pour qu’ils calculent le nombre d’étapes nécessaires avant de terminer.
5. (*bonus : pour les plus rapides ou à faire chez soi*) Parmi tous les nombres inférieurs à 1000, lequel nécessite-t-il le plus d’étapes pour terminer?

**Exercice 3** – Dessiner un tapis (\*\*)

Ajoutez à votre programme les définitions de fonctions ci-dessous.

```

1 def étoile():
2     print('*', sep='', end='')
3
4 def dièse() :
5     print('#', sep='', end='')
6
7 def nouvelle_ligne() :
8     print()
    
```

Sans utiliser la fonction `print`, mais uniquement les fonctions `étoile`, `dièse`, et `nouvelle_ligne` ci-dessus, écrivez des fonctions `tapis_a(l,h)`, `tapis_b(l,h)`, `tapis_c(l,h)` et `tapis_d(l,h)` qui affichent des tapis de largeur `l` et de hauteur `h` avec les motifs ci-dessous.

a)	b)	c)	d)
*****	*#*#*#*#*#*	*#*#*#*#*#*	*#*#*#*#*#*
*****	*#*#*#*#*#*	#*#*#*#*#*#*	#*#*#*#*#*#*
*****	*#*#*#*#*#*	*#*#*#*#*#*	*****
*****	*#*#*#*#*#*	#*#*#*#*#*#*	*#*#*#*#*#*
*****	*#*#*#*#*#*	*#*#*#*#*#*	#*#*#*#*#*#*
*****	*#*#*#*#*#*	#*#*#*#*#*#*	*****
*****	*#*#*#*#*#*	*#*#*#*#*#*	*#*#*#*#*#*
*****	*#*#*#*#*#*	#*#*#*#*#*#*	#*#*#*#*#*#*
*****	*#*#*#*#*#*	*#*#*#*#*#*	*****

**Exercice 4** – Fermat, Grothendieck, et les nombres premiers (\*\*)

Le  $n$ -ième nombre de Fermat est  $2^{2^n} + 1$ . Fermat avait conjecturé que tous ces nombres étaient des nombres premiers. Votre mission : montrer à l’aide de Python que cette conjecture est fautive !

1. Écrivez une fonction `fermat(n)` qui renvoie le  $n$ -ième nombre de Fermat. Par exemple, `fermat(3)` renvoie 257.

```
1 def fermat(n):
2     return 2 ** (2 ** n) + 1
```

2. Écrivez une fonction `premier_facteur(n)` qui renvoie le plus petit nombre supérieur ou égal à 2 qui divise  $n$ . Par exemple, `premier_facteur(35)` renvoie 5 et `premier_facteur(31)` renvoie 31.

```
1 def premier_facteur(n):
2     d = 2
3     while n%d != 0 :
4         d = d+1
5     return d
```

3. Que pensez-vous de 57, appelé nombre premier de Grothendieck? Testez de même la primalité de `fermat(3)` et `fermat(4)` en utilisant la fonction `premier_facteur`.<sup>1</sup>

```
1 >>> print('Le premier facteur de',57,'est',premier_facteur(57))
2 Le premier facteur de 57 est 3
```

4. Écrivez un programme qui affiche le premier  $n$  tel que `fermat(n)` n’est pas premier. Vous afficherez aussi un diviseur premier de `fermat(n)`.

*Une première méthode naïve :*

```
1 def est_premier(n):
2     return n == premier_facteur(n)
3
4 def dernière_question() :
5     n = 0
6     # Heureusement que Fermat s'est trompé, sinon notre boucle ne terminerait jamais !
7     while est_premier(fermat(n)) :
8         n = n + 1
9     print('fermat(',n,')=',fermat(n),' n'est pas premier',sep='',end=' ')
10    print('car il a pour diviseur ', premier_facteur(fermat(n)),sep='')
```

*La solution ci-dessus n’est pas optimale, car on calcule 2 fois `fermat(n)` et on cherche 2 fois son diviseur pour le  $n$  qu’on affiche. On peut faire plus efficace en sauvant les calculs intermédiaires dans des variables comme ci-dessous*

```
1 def dernière_question_v2() :
2     n = 0
3     f = fermat(n)
4     p = premier_facteur(f)
5     # Pour que le test du while ait un sens la première fois qu'on
6     # rentre dans la boucle, il faut initialiser f et p avant.
7     while f == p :
8         n = n + 1
9         f = fermat(n)
10        p = premier_facteur(f)
11    print('fermat(',n,')=',f,' n'est pas premier',sep='',end=' ')
12    print('car il a pour diviseur ',p,sep='')
```

*On peut faire encore plus efficace! En effet, le calcul de `fermat(n)` peut se faire de manière incrémentale : on passe de `fermat(n)-1 = 2^{2^n}` à `fermat(n+1)-1 = 2^{2^{n+1}}` en élevant au carré. C’est la propriété qu’on utilise ci-dessous*

1. Mais alors pourquoi parler de premier de Grothendieck? Et qui était Grothendieck? Lire <http://images.math.cnrs.fr/Alexandre-Grothendieck.html> si cela vous intéresse.

```

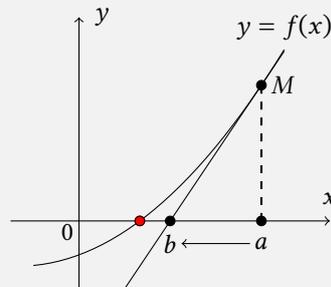
1 def dernière_question_v3() :
2     n = 0
3     f = fermat(0)
4     p = premier_facteur(f)
5     while f == p :
6         n = n + 1
7         f = (f-1)**2 + 1
8         p = premier_facteur(f)
9     print('fermat(',n,')=',f, ' n'est pas premier', sep='', end=' ')
10    print('car il a pour diviseur ',p, sep='')

```

**Exercice 5 – Méthode de Newton (\*\*)**

S’il existe des formules pour résoudre les équations du second degré, de nombreuses équations ne possèdent pas de telles formules. Il existe cependant des algorithmes pour trouver des valeurs approchées de ces solutions. Nous allons programmer nous même une méthode générale (la méthode de Newton) qui résout de manière approchée les équations de la forme  $f(x) = 0$  (en tous cas elle trouvera une solution si elle existe et si  $f$  vérifie certaines propriétés).

Nous allons supposer que la fonction  $f$  est une fonction dérivable et à dérivée non nulle presque partout (de sorte que la tangente à la courbe existe avec une probabilité quasi-nulle d’être horizontale) pour appliquer la méthode des tangentes de Newton mentionnée en cours.



On suppose que l’approximation courante est un nombre strictement positif  $a$ . L’équation de la droite tangente à la courbe de  $f$  au point  $M(a, f(a))$  s’écrit  $y - f(a) = f'(a)(x - a)$ . En  $y = 0$ , la droite coupe l’axe des abscisses en un point d’abscisse  $b$  donnée par la formule :

$$b = a - \frac{f(a)}{f'(a)}$$

Ainsi, pour  $f(x) = x^2 - r$ , on a  $f'(x) = 2x$  et  $b = a - \frac{a^2 - r}{2a} = \frac{1}{2} \left( a + \frac{r}{a} \right)$  comme vu en cours.

- Écrivez une fonction `dériver(f, a, h)` qui renvoie  $\frac{f(a+h) - f(a)}{h}$  : pour un  $h$  petit, c’est une bonne approximation de  $f'(a)$ .

```
1 def dériver(f, a, h) :
2     return (f(a+h) - f(a)) / h
```

- Écrivez une fonction `résoudre(f, a0, h)` qui renvoie un nombre  $a$  tel que  $|f(a)| < h$  en prenant pour approximation initiale  $a_0$ . *Indice* : on va calculer une suite d’approximation de  $a$  en utilisant la formule de récurrence :

$$a = a - \frac{f(a)}{f'(a)}$$

```
1 def résoudre(f, a, h) :
2     while abs(f(a)) >= h :
3         a = a - f(a) / dériver(f, a, h)
4     return a
```

- Testez votre fonction : calculez les cinq premières décimales de  $\sqrt{2}$ ; vous devez trouver 1,41421 ...

```
1 def f(x) :
2     return x*x - 2
3
4 print("racine(2) =", résoudre(f, 1, 0.000001))
```

**Exercice 6** – Dessiner des tapis, bonus (\*\*\*)

Même consigne que pour l’exercice 3. Vous utiliserez lorsque ce sera nécessaire les deux fonctions suivantes.

```
1 def barre_1():
2     print('/', end='')
3
4 def barre_2():
5     print('\', end='') # Il faut échapper le symbole de backslash
```

a)	b)	c)
*****	#####	###/\###
*****	#####*	##/##\###
*****	#####*##	##/####\##
*****	#####*###	/#####\##
*****	#####*####	/#####\
*****	#####*#####	\#####/
*****	#####*#####	#\#####/##
*****	#####*#####	##\####/##
*****	#####*#####	###\##/###
*****	#####*#####	####\/####

```
1 # Pour afficher n fois le même symbole
2 def ligne_simple(n,symbole):
3     i=0
4     while i<n:
5         symbole()
6         i=i+1
7
8 def tapis_A(largeur,hauteur):
9     ligne_simple(largeur,étoile) # première ligne
10    nouvelle_ligne()
11    i=0
12    while i<hauteur-2:
13        étoile()
14        ligne_simple(largeur-2,dièse)
15        étoile()
16        nouvelle_ligne()
17        i=i+1
18    ligne_simple(largeur,étoile) # dernière ligne
19    nouvelle_ligne()
20
21 # Pour afficher ##### où l'* est à la position p
22 def ligne_position(n,p,symbole1,symbole2):
23     i=0
24     while i<n:
25         if i==p:
26             symbole2()
27         else:
28             symbole1()
29         i=i+1
30
31
32 def tapis_B(largeur,hauteur):
33     i=0
34     while i<hauteur:
35         ligne_position(largeur,largeur-i-1,dièse,étoile)
36         nouvelle_ligne()
37         i=i+1
38
39 def barre_1():
40     print('/',end='')
41
42 def barre_2():
43     print('\',end='') # Il faut échapper le symbole de backslash
44
45 # Ne marche que si largeur est pair
46 def tapis_C(largeur,hauteur):
47     i=0
48     m=hauteur//2
49     n=largeur//2
50     while i<m:
51         ligne_position(n,n-i-1,dièse,barre_1)
52         ligne_position(n,i,dièse,barre_2)
53         nouvelle_ligne()
54         i=i+1
55     i=0
56     while i<m:
57         ligne_position(n,i,dièse,barre_2)
58         ligne_position(n,n-i-1,dièse,barre_1)
59         nouvelle_ligne()
60         i=i+1
```

