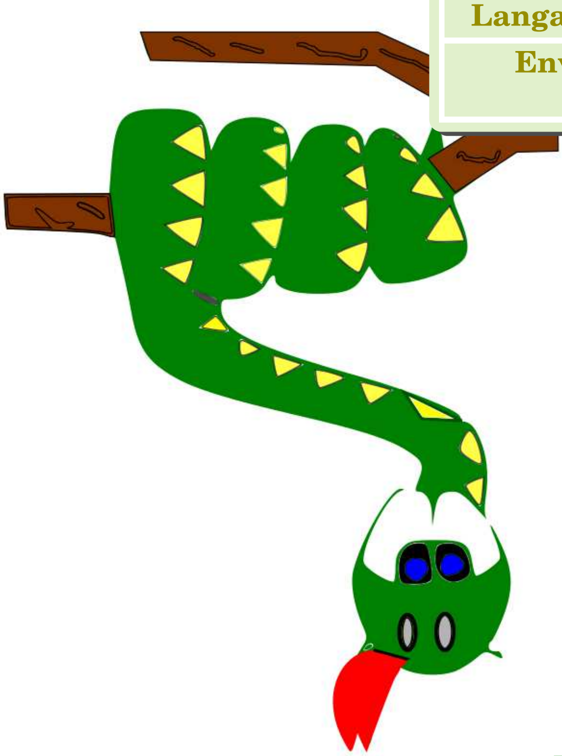
Livret Python

Classe de Seconde

Objectif	Programmer les bases pour les maths
Langage de programmation	Python
Environnement utilisé (EDI)	Jupyter Notebook



Frédéric Bro

2020 - 2021

Sommaire

Utilisation de Jupyter

A Python et Jupyter c'est quoi ?

- **Python** est le langage de programmation pour coder des algorithmes
- **Jupyter hub**, est un **environnement de calcul interactif** (EDI) qui permet de :
  - ▷ écrire du code écrit en langage **Python**
  - ▷ du texte enrichi
  - ▷ insérer des graphiques
- Un document réalisé avec **Jupyter** est appelé un **notebook** (callepin en français).

  Un notebook est un fichier de la forme : `nom_du_fichier.ipynb`.

B Peut-on utiliser un autre environnement Jupyter en cas de panne ?

- Utiliser **Try Jupyter with Python** depuis la page web :

  http://jupyter.org/try.

  Avantage : permet d'exécuter les notebooks pour dépanner.
  Inconvénient : l'environnement ne peut pas être utilisé indéfiniment dans le temps.

- Télécharger **anaconda** (une distribution **python** pour nous scientifiques) puis l'installer via l'adresse :

  https://www.anaconda.com/download/

  Avantage : on est indépendant des aléas d'internet.
  Inconvénient : tout se fait sur le même ordinateur.

C Manipuler les notebooks sur l'ENT

1 Créer un nouveau notebook

1. Ouvrir l'application **capytale** via l'ENT :

En sélectionnant **Jupyter**, on obtient le **Home** de **Jupyter** :

2. Sélectionner dans l'ordre

- **Nouveau**

- **Python 3** (l'interpréteur du code)

2 Importer un nouveau notebook

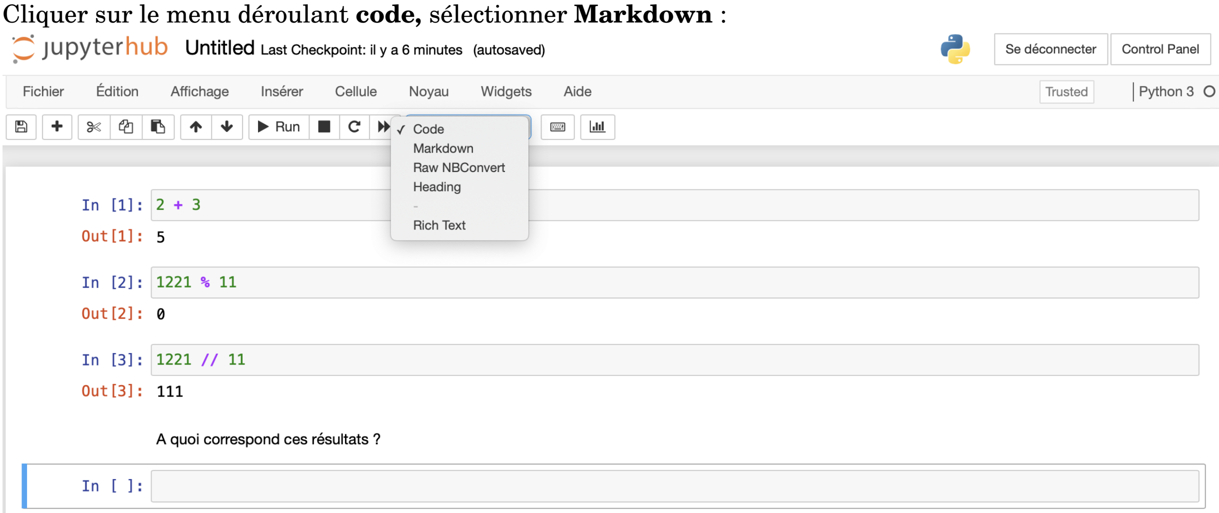
Depuis le **Home de Jupyter** :

- cliquer sur **Téléverser**
- puis enuite sléectionner un notebook du type `blabla.ipynb`

D Actions sur un notebook

1 Ajouter du texte

Cliquer sur le menu déroulant **code,** sélectionner **Markdown** :



2 Autres actions un notebook

Renommer un notebook	`Fichier ▷ Renommer`
Enregistrer un notebook	`File ▷ Download as ▷ Notebook (.ipynb)`
Exécuter une ligne	SHIFT + ENTRÉE | CTRL+ENTRÉE si on veut rester sur la cellule

Les calculs sont effectués dans un noyau connecté à python 3.
Toute trace de calculs est conservée en mémoire dans ce noyau. On peut :

Stopper une exécution	`Noyau ▷ Interrompre` *ou symbole carré noir*
Redémarrer le noyau	`Noyau ▷ Redémarrer`

II Premières instructions Python

A Conserver en mémoire

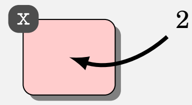
Pour conserver en mémoire un élément *(par exemple un nombre, un mot, etc)*, on utilise une **variable**.

Instruction d'affectation :

```
In[1] : x = 2
```

est l'instruction qui signifie que x **prend la valeur** 2

La variable est comme une boîte vide. Elle porte un nom, par exemple x et on lui place ou stocke le nombre 2



En langage naturel, on note $\boxed{x \leftarrow 2}$

Plusieurs affectations sur une même ligne :

```
In [1] : x,y,z = 2,3,4
```

signifie $x \leftarrow 2$, $y \leftarrow 3$ et $z \leftarrow 4$

B Type d'une variable

Définitions : on affecte à une variable x un élément.
Le type de la variable x dépend de l'élément. Voyons des exemples :

En langage naturel	En PYTHON	Type de x	Commentaires
x ← baba7:)	`x = 'baba7:)'`	**string**	x contient une chaine de caractères *C'est une succession de caractères mis entre des guillemets* ' ' *ou* " ".
x ← 64	`x = 64`	**int**	x contient l'entier 64
x ← 3,5	`x = 3.5`	**float**	x contient le nombre réel 3,5 *On dit que :* `3.5` *est **la représentation flottante** du réel* 3,5.

Problèmes de représentation :

- Pour écrire en machine le nombre réel 0,1 on saisit `0.1`

```
In [1]:  3*0.1

Out[1]:  0.30000000000000004
```

Ainsi :

$$\underbrace{\texttt{0.1}}_{\text{écriture flottante}} \neq \underbrace{0{,}1}_{\texttt{écriture décimale}}$$

  Un nombre réel x **est représenté en machine** par **son flottant** $fl(x)$ et <u>*il arrive*</u> que $x \neq fl(x)$.

- Arrondir un nombre :

```
In [2]:  7/3

Out[2]:  2.3333333333333335
```

$\frac{7}{3}$ est **représentée** par défaut avec 18 chiffres.

```
In [3]:  round(1/3,6)

Out[3]:  2.333333
```

$\frac{7}{3}$ est arrondie par défaut avec 6 décimales.

```
In [4]:  round(7/3)

Out[4]:  2
```

- $\frac{7}{3}$ est arrondie par défaut à l'unité.
- Si $x > 0$, alors **`round`**(x) renvoie la partie entière son écriture décimale.

Remarque : il existe bien d'autres types que l'on rencontrera plus tard, par exemple :

- **set** (ensemble)
- **list** (une liste)
- **bool** (le booléen **True** « vrai » ou **False** « faux »)

C Afficher le contenu d'une variable

Instruction print :

Objectif	Langage naturel	Python
Affichage d'une valeur numérique x :	Afficher « x »	`print(x)`
Affichage d'un texte :	Afficher « les 3 petits cochons »	`print('les 3 petits cochons')`
Affichage d'un texte avec une apostrophe :	Afficher « l'entier »	`print("l'entier")`
Affichage d'un texte et d'une variable numérique x :	Afficher « la valeur de x est :x »	`print('la valeur de x est :',x)`

D Opérations de base

1 Opérateurs

Opérations	$+$	$-$	$\times$	$/$	$=$	$\neq$	$>$	$<$	$\leqslant$	$\geqslant$
Avec Python	+	-	*	/	==	!=	>	<	<=	>=

Puissance d'un nombre a	a^n
Avec Python	a**n

2 Quotient et reste de la division euclidienne

Soient deux entiers naturels a avec b.

Posons la division euclidienne : $\begin{array}{c|c} a & b \\ \hline r & q \end{array}$

Alors il existe deux uniques entiers q et r vérifiant :

$$a = bq + r \text{ et } 0 \leqslant r < b.$$

Obtenir	Avec Python
Quotient q	a//b
Reste r	a%b

3 Pour faire d'autres calculs

Instructions contenues dans le module math :

```
In [1] : from math import *
```

est l'instruction qui permet de charger toutes les fonctions contenues dans le module nommé **math**

Par exemple, on peut faire :

Calcul	Avec Python
$\sqrt{3}$	sqrt(3)
π	pi
partie entière de 2,7	floor(2.7)

☐ **Exercice 1 :** *Division dans tous ses états*

1. Pour chaque opération, recopier exactement le résultat obtenu après exécution :

```
In [1]:  4/2

Out[1]:

In [2]:  4//2

Out[2]:
```

2. Laquelle des deux divisions donne un résultat entier (de type **int**) ?

☐ **Exercice 2 :** *Minis calculs*

1. **a.** Calculer avec Python $3\times\left(\frac{1}{3}+1\right)$.

   b. Retrouver ce résultat à la main.

2. Calculer 2^{10}.

3. **a.** Que teste l'instruction suivante ?

```
In [3]:  18**2+80**2 == 82**2

Out[3]:  True
```

   b. Que peut-on en déduire pour le triangle de mesures 18cm, 80cm et 82cm ?

   c. Quelle est la nature d'un triangle de mesures $0,18$cm, $0,80$cm et $0,82$cm ?

☐ **Exercice 3 :** *Diviseurs*

1. Dans un nouveau notebook, on saisit l'instruction suivante :

```
In [1]:  51 % 3

Out[1]:  0
```

Compléter « 51 est .... par .... »

2. Que donne l'instruction suivante ?

```
In [2]:  51 // 3

Out[2]:
```

Intérepréter le résultat obtenu.

III Les fonctions

A Somme d'entiers consécutifs

1. On donne le programme de calcul suivant :

> S prend la valeur 0
> k prend la valeur 1
> Répéter 5 fois de suite les instructions suivantes :
> - S prend la valeur S + k
> - k prend la valeur k + 1

A l'issue de ce programme que valent S et k ?
Point avec le professeur : écrivons et exécutons ce programme en langage Python.

2. Pour répéter l'instruction S prend la valeur S + k avec k allant de 1 à 5, on peut écrire dans JUPYTER NOTEBOOK :

```
In [7]: s = 0
        for k in range(1,6):
            s = s+k
            print(s,k)
```

a. En exécutant cette ligne, indiquer les affichages obtenus.

b. Citer toutes les variables qui interviennent dans cet algorithme ?

c. Compléter les pointillés S = ... + ... + ... + ... + ... + ...

3. La commande **range** :

a. En exécutant le programme suivant, donner les valeurs de la variable k qui sont affichées.

```
In [8]: for k in range(1,10):
            print(k)
```

b. Compléter le tableau ci-dessous :

	Ensemble des entiers k allant		
	de	jusque	nombre d'entiers
range(1,8)			
range(8)			
range(2,101)			
range(-4,5)			

4. Modifier le programme pour calculer $1+2+\cdots+10$.

5. Pour calculer la somme $1+2+\cdots+n$, pour n'importe quelle valeur de n, on peut définir une **fonction**.

a. Recopier puis exécuter l'instruction suivante :

```
In [9]: def somme(n):
            s = 0
            for k in range(1,n+1):
                s = s+k
            return s
```

On vient de définir la fonction nommée **somme** de paramètre n *(un entier supérieur ou égal à* 1*)*.
Elle renvoie la valeur de $1+2+\cdots+n$.

b. Exécuter l'instruction et compléter le résultat obtenu :

```
In [10]: somme(10)

Out[10]:
```

c. Calculer $1+2+\cdots+100$.

B Synthèse

Boucle for :

Code Python :

```
for k in range(3):
    Instruction 1
    Instruction 2
```

Signifie en langage naturel que

Pour chaque **entier** k allant de 0 jusque 3 **exclu Faire**
    Instruction 1
    Instruction 2
FinPour

Commande **range** :

Code Python	Ensemble des entiers associés	Comptage du nombre d'entiers consécutifs :
`range(10)`	**0**, 1, 2, 3, 4, 5, 6, 7, 8, **9**	$9-0+1=10$
`range(1,11)`	**1**, 2, 3, 4, 5, 6, 7, 8, 9, **10**	$10-1+1=10$
`range(10,111)`	**10**, 11, 12, 13, ... , 109, **110**	$110-1+1=110$
`range(n,m+1)`	n, $n+1$, $n+2$, ... , $m-1$, m	$m-n+1$
`range(1,10,2)`	1, 3, 5, 7, 9	
`range(2,15,2)`	2, 4, 6, 8, 10, 12, 14	

- A chaque fois que `k` prend sa valeur dans l'ensemble des entiers **`range(...,...)`**, le bloc **d'insturctions** est **répété**.
  On dit que c'est une **boucle for** ou boucle « **Pour** » en français.
- Une boucle « **Pour** » **permet de répéter** un nombre fini de fois un bloc d'instructions.
- `k` est une **variable**. Son nom aurait pu aussi bien s'appeler `i`, `j`, `voiture`, etc.

Généralité sur les fonction :

Une fonction est un programme qui a un **nom** et éventuellement des **paramètres** *(variables qui sont utilisées à l'intérieur de ce programme)* :

Nom de la fonction *(au choix)*

Paramètre(s) : *(ils sont séparés par des virgules)*

```
In [1] : def nom_fonction(....,....) :
             Bloc d'instructions
             return ........
```

Variable(s) dont on souhaite **renvoyer** la (ou les) valeur(s)

Exemple : la surface latérale d'une boîte *(parallélépipède rectangle)* de dimensions x, y et z est $\boxed{2xy+2xz+2yz}$.
Écrivons la fonction nommée **surface** de paramètres x *(la largeur)*, y *(la longueur)* et z *(la hauteur de la boîte)* et qui renvoie sa surface latérale :

```
In [1] : def surface(x,y,z) :
             return 2*x*y + 2*x*z + 2*y*z
```

Pour calculer la surface latérale de la boîte lorsque $x=3$, $y=4$ et $z=5$, on écrit :

```
In [2] : surface(3, 4, 5)
Out [2] : 94
```

☐ **Exercice 4 :** *(Un amusement)*

1. Écrire un algorithme qui affiche : 10 fois de suite le mot « banane ».
2. Écrire un algorithme qui affiche successivement les entiers 1, 2, 3 jusque 50.

☐ **Exercice 5 :**

On propose la fonction suivante écrit en langage **PYTHON** :

```
In [1]: def factorielle(n):
            p = 1
            for k in range(1, n+1):
                p = p*k
            return p
```

1. Donner le nom de cette fonction et le nom de son paramètre.
2. **a.** Écrire l'instruction permettant d'exécuter cette fonction pour n = 5.
   b. Donner le résultat obtenu en sortie.
3. Que renvoie `factorielle(6)` ?

☐ **Exercice 6 :**

1. On propose l'algorithme ci-dessous :

```
S ← 0
Pour k allant de 1 à 5 Faire
    S ← S + 2
FinPour
```

	Au départ					
k		1	2	3	4	5
S	0					

Tout en exécutant l'algorithme ci-dessus, remplir les cases vides de ce tableau.

2. Écrire cet algorithme avec PYTHON.

IV Instructions conditionnelles : les boucles « Si » et « Si - Sinon »

Boucle if :

Langage naturel :

```
Si condition Alors
    Bloc 1 d'instructions
Sinon
    Bloc 2 d'instructions
FinSi
```

Correspond à

Code Python :

```
In [1] : if condition:
             Bloc 1 d'instructions
         else:
             Bloc 2 d'instructions
```

Le bloc **else** est **optionnel**

On teste si condition est vraie. Si c'est le cas, alors on exécute :

`Bloc 1 d'instructions`

sinon on exécute :

`Bloc 2 d'instructions`

— L'instruction **if** ou « **Si** » est aussi appelée **boucle « Si »**.
— C'est une **instruction conditionnelle**

☐ **Exercice 7 :** *Minimum*

On considère la fonction nommée **distance** de paramètres a et b *(deux nombres réels)* et qui renvoie la distance entre a et b. Compléter alors les instructions en pointillés.

```
In [1]: def distance(a,b):
            if a .... b:
                return b−a
            else:
                return ....
```

```
In [2]: distance(2.5,4.3)
```

```
Out[2]:
```

☐ **Exercice 8 :** *Parité*

1. Que signifie le test ci-dessous ?

```
In [1]: 1236 % 2 == 0
```

2. Écrire une fonction nommée **parité** de paramètre n *(un entier naturel)* et qui affiche « pair » si n est pair sinon affiche « impair ».

☐ **Exercice 9 :** *Nombre de diviseurs*

On considère la fonction nommée **nb_div** qui a pour paramètre n *(un entier naturel non nul)* et qui renvoie le nombre de diviseurs de n.

Principe :

- La variable c désigne le nombre de diviseurs de n.
- Au départ c = 0.
- Pour chaque entier k allant de 1 jusque n, on teste si k divise n.
  Si cela est vrai, alors c est augmenté de 1.

```
In [1]: def nb_div(.....):
            c = ....
            for k in range(...., ....):
                ..................
                    ..............
            return c
```

Compléter les instructions en pointillé de cette fonction.

*Remarque : on dit que c est un **compteur**.*

☐ **Exercice 10 :** *Liste de Diviseurs*

On souhaite collecter dans une **liste** tous les diviseurs d'un nombre `n` *(un entier naturel supérieur ou égal à 2)*.

1. On considère la fonction nommée **diviseurs2** de paramètre `n` *(un entier naturel supérieur ou égal à 2)*.
Elle **renvoie** la liste de tous les diviseurs de `n`.
Procédure :

- Au départ, on crée la **liste**, notéeD, contenant le diviseur 1 grâce à l'instruction `D = [1]`.
- Au fur et à mesure, on teste pour chaque entier nommé `div` (compris entre 1 et `n`), s'il est oui ou non un diviseur de n.
  Si c'est le cas, alors on l'ajoute à la liste `D`, grâce à la commande append (voir encadré PYTHON).

Avec python

`D.append(3)` permet d'ajouter 3 à `D`.

Compléter alors les instructions en pointillé :

```
In [1]: def diviseurs2(n):
            D = [1]
            for div in range(2, .....):
                if ..............:
                    D.append(......)
            return D
```

2. Dresser la liste `D1` de tous les diviseurs de 75, puis `D2` la liste de tous les diviseurs de 105.

3. Exécuter l'instruction suivante :

```
In [4]: D3 = [d1 for d1 in D1 if d1 in D2]
        D3

Out[4]: ..............................................
```

Compléter : « `D3` est l'ensemble des .... de 75 et 105 »

4. Comment obtenir le plus grand diviseur commun de 75 et 105 ?

Avec python

- `max(D)` renvoie le maximum des éléments de `D`.
- `min(D)` renvoie le minimum des éléments de `D`.

5. **a.** Écrire une fonction nommée **pgcd** de paramètres `a` et `b` *(2 entiers non nuls)* qui utilise la fonction diviseurs2 et qui renvoie le plus grand diviseur commun de `a` et de `b`.

b. Déterminer le plus grand diviseur commun de 20 400 avec 265 625.

☐ **Exercice 11 :** *PGCD*

Le PGCD de `a` et de `b` est égal à `a` si $\mathtt{b} = 0$ sinon il est égal au PGCD de `b` avec le reste de la division euclidienne de `a` par `b` :

$$\mathtt{PGCD(a,b)} = \begin{cases} \mathtt{a} & \text{si} \quad \mathtt{b} = 0 \\ \mathtt{PGCD(b,r)} & \text{sinon} \end{cases} .$$

1. Écrire une fonction nommée **pgcd** qui a pour paramètre `a` et `b` (entiers naturels avec $\mathtt{a} \geqslant \mathtt{b}$) et qui renvoie le PGCD de `a` avec `b`.

2. Calculer le PGCD de 212 et 86.

☐ **Exercice 12 :**

Une année bissextile est une année multiple de 4, non multiple de 100 à moins qu'elle ne soit multiple de 400.

Écrire une fonction nommée **bissextile** qui prend pour paramètre `n` *(une année)* et qui renvoie **`True`** si l'année `n` est bissextile et **`False`** sinon.

V Instructions répétitives avec test d'arrêt : la boucle « tant que »

Boucle while :

Langage naturel :

TantQue condition est vraie **Faire**
Bloc d'instructions
FinTantQue

Correspond à

Code Python :

```
In [1]: while condition:
            Bloc d'instructions
```

A chaque fois on teste si `condition` est vraie. Si c'est le cas, on exécute alors
Bloc d'instructions

- L'instruction **`while`** ou « **TantQue** » est aussi appelée **boucle « Tant que »**.
- Le bloc est répété tant que `condition` est vraie.

☐ **Exercice 13 :** *Un peu de logique*
Compléter le tableau ci-dessous :

Condition	Condition contraire
$x > 2$	
$x \geqslant 3$	
$x < 4$	
$U \geqslant 10^3$	
$U < 10^{-4}$	
$U \geqslant 0$ et $U < 1$	

☐ **Exercice 14 :**
Exécuter chaque algorithme et indiquer le résultat obtenu :

	Résultats ?		Résultats ?

In [1]: `n = 0`
`while n != 10:`
`    n = n+1`
`print(n)`
		In [2]: `x=0`
		`while x < 1:`
		`    x = x-1`
		`    print(x)`	

☐ **Exercice 15 :**

```
In [1]: n = 0
        while n**2 < 5004:
            n = n+1
        n

Out[1]: 71
```

1. Compléter avec l'un des symboles >, <, ⩾ ou ⩽ :
   « La valeur de `n` en mémoire a son carré .... 5 004 ».
2. Quel est le plus petit entier ayant son carré supérieur ou égal à 5 004 ?
3. Quel est le plus grand entier qui a son carré strictement inférieur à 5 004 ?

☐ **Exercice 16 :**

1. Déterminer le plus grand entier n tel que $n^3 \leqslant 5\,004$.
2. Déterminer le plus petit entier n tel que $\left(\frac{2}{3}\right)^n \leqslant 0,01$.

☐ **Exercice 17 :** *Chiffres dans l'écriture d'un entier*

Avec python

Soit la division euclidienne :
`a=b×q+r`
- `a//b` renvoie `q`
- `a%b` renvoie `r`.

1. En utilisant ce tableau, expliquer comment obtenir tous les chiffres de 1789 :

n	n%10	n//10
1789		

2. Écrire une fonction nommée **chiffres** qui prend pour paramètre `n` *(un entier naturel)* et qui affiche tous les chiffres dans l'écriture de `n`.

☐ **Exercice 18 :** *Décomposition en facteurs premiers*

Méthode :

On cherche le plus petit nombre premier qui divise le nombre n, on fait la division de n par ce nombre premier et si le quotient obtenu est différent de 1, on recommence ... jusqu'à obtenir pour quotient 1.

De coutume, on peut mener les calculs dans un tableau :

Diviseurs	$n = 240$
2	120
2	60
2	30
2	15
3	5
5	1

1. A partir du tableau compléter :

$$240 = 1 \times \ldots\ldots\ldots \times \ldots\ldots\ldots \times \ldots\ldots\ldots$$

2. Compléter la fonction nommée **décomposition** de paramètre n *(un entier supérieur à 2)* qui affiche le produit des facteurs premiers de n.

```
In [1]: def decomposition(n):
            print(n,' = 1', end = '')
            div = 2
            while n > ....:
                while n%div .........:
                    print(" x ", div, end = '')
                    n = .........
                div = div + 1
```

3. Tester ce programme pour un grand nombre de votre choix.

☐ **Exercice 19 :**

En 2018, on place 1 000 € sur un compte à la banque.
Chaque année, ce capital augmente de 4%.

A partir de quelle année le capital aura -t-il doublé ?

☐ **Exercice 20 :**

2012, augmenté de la somme de ses chiffres donne 2017.
Trouver tous les nombres entiers qui, augmentés de la somme de leurs chiffres, donnent 2017.

☐ **Exercice 21 :**

Une rue contient 1 000 immeubles. On doit les numéroter de 1 à 1 000.
Combien de fois va-t-on peindre le chiffre 9 ?

☐ **Exercice 22 :** *PGCD - (Méthode d'Euclide)*

Rappel :

- Si un nombre est un diviseur de 2 nombres a et b, alors il est aussi un diviseur de b et du reste r de la division euclidienne de a par b.
- Plus précisément :

$$\mathtt{PGCD}(a\,;b) = \mathtt{PGCD}(b\,;r)$$

Soient deux entiers naturels a et b avec $a \geqslant b > 0$.

L'algorithme d'EUCLIDE consiste à répéter le processus suivant :

① Calculer q et r de la division euclidienne : $a = b \times q + r$.

② Remplacer a par b

③ Remplacer b par r

jusqu'à ce que b soit nul.
Dans ce cas $\mathtt{PGCD}(a\,,b)$ est la dernière valeur de b contenue en mémoire.

Calculons $\mathtt{PGCD}(26\,,7)$:

$$\begin{aligned} 26 &= 7 \times 3 + 5 \\ 7 &= 5 \times 1 + 2 \\ 5 &= 2 \times 2 + \mathbf{1} \\ 2 &= 2 \times 1 + \boxed{0} \end{aligned}$$

1 est le dernier reste non nul, donc $\mathtt{PGCD}(26\,,7) = \mathbf{1}$.

1. On considère la fonction ci-dessous, nommée **PGCD**. Elle a pour paramètres les nombres a et b *(deux entiers naturels non nuls)*. Elle calcule et renvoie $\mathtt{PGCD}(a\,,b)$.

```
In [1]: def PGCD(a,b):
            while ............:
                r = ......
                a = ......
                b = ......
            return .....
```

2. Calculer $\mathtt{PGCD}\left(2^{13}+1\,,2^{12}+1\right)$.

☐ **Exercice 23 :**

Un distributeur peut rendre des billets de 20 euros, des pièces de 5 euros et des pièces de 1 euros.
Il est programmé pour commencer à rendre le plus de billets de 20, puis le plus de pièces de 5 possible et enfin le reste en pièces de 1.
Écrire une fonction nommée **distributeur** qui a pour paramètre n *(le montant à donner)* et qui renvoie les nombres de billets de 20, de pièces de 5 puis de pièces de 1 rendus.