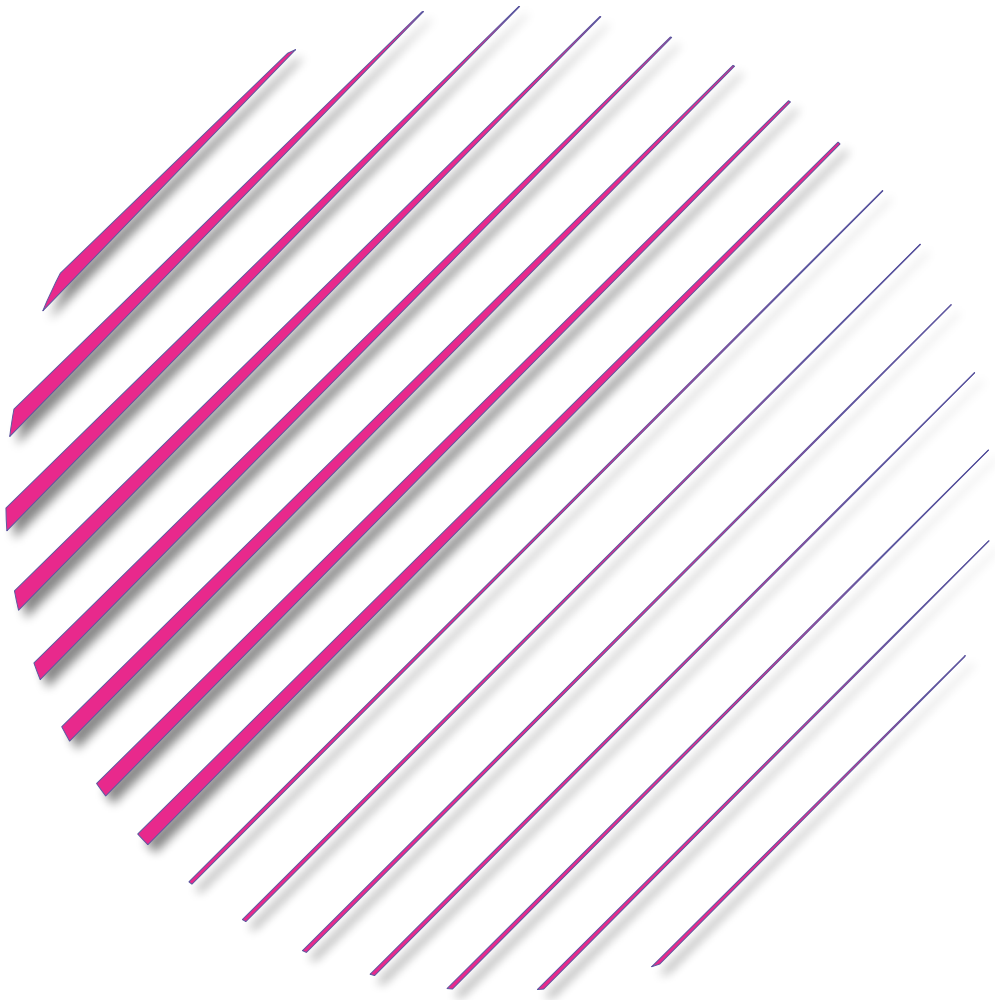
REPÈRES

Mathématiques-Physique-Chimie

Algorithmique et programmation avec le langage Python en voie professionnelle



http://pedagogie.ac-lille.fr/maths-physique-chimie

@ac-lille.fr

"Les tentatives de création de machines pensantes nous seront d'une grande aide pour découvrir comment nous pensons nous-mêmes."

Alan Turing

Dépôt légal : février 2021
Imprimé en France

Conception graphique et mise en pages : Ludovic Diana & Jérôme Lenoir

Cet ouvrage est imprimé sur du papier offset sans bois 110 g, papier issu de forêts gérées durablement

Sommaire

Sommaire

Préface

À l'instar du précédent opus, ce livret a pour vocation d'accompagner les enseignants de mathématiques et de physique-chimie dans la mise en oeuvre de l'algorithmique et de la programmation (A&P) dans leurs séquences d'enseignement. S'il ne s'agit pas ici d'atteindre un niveau élevé de maîtrise du langage de programmation en tant que tel, il convient néanmoins d'en faire un outil de développement de la pensée algorithmique. En effet, Python est un langage interprété ce qui signifie que les lignes du code sont exécutées successivement sans tenir compte, à priori, des suivantes. Le rapport à l'erreur de l'apprenant s'en trouve facilité, lui permettant ainsi une expérimentation mathématique bien plus aisée.

Dans une volonté d'inspirer et non de prescrire, ce document permet à chacun d'appréhender au mieux les notions développées dans les programmes de MPC de la voie professionnelle, mais également de :

- présenter quelques éléments de conceptualisation sur la notion d'algorithmique afin d'harmoniser les modes de représentation ;

- proposer quelques pistes d'exploitation des capacités et connaissances déclinées dans le module « algorithmique et programmation », à travers des situations pédagogiques illustrant les différents thèmes des programmes actuels ;

- tirer profit des nombreuses possibilités qu'offre le langage de programmation Python en appréhendant certaines bases et autres spécificités de cet outil.

Cette nouvelle édition a été écrite non seulement en tenant compte des nombreux retours du terrain, mais également dans le souci de mettre en exergue l'intérêt pédagogique à former les élèves à l'algorithmique et à la programmation aussi bien dans une démarche de résolution de problème en mathématiques, qu'au service des capacités numériques sollicitées à de nombreuses reprises dans les programmes de la voie professionnelle en physique-chimie.

Les différentes situations proposées dans ce livret ont uniquement un but formatif en évoquant des articulations possibles autour du questionnement propre à la programmation, et ainsi nourrir la réflexion des enseignants. Dans cette optique de former essentiellement à l'utilisation de l'A&P, les différents scénarios pédagogiques qui y sont développés ne présentent volontairement aucun observable ni critère d'évaluation, les compétences mobilisées ne tenant compte d'aucune progression annuelle.

> *Les situations et les pistes d'exploitation pédagogiques déclinées dans cette nouvelle édition constituent une base de réflexion en vue de la construction de séquences adaptées et réfléchies dans l'esprit des textes actuels et doivent être abordées de manière transversale.*

Le groupe A&P de l'académie de Lille

> Un algorithme est une liste finie de processus élémentaires, appelés instructions élémentaires, amenant à la résolution d'une problématique.
>
> Il ne doit contenir que des instructions compréhensibles par celui qui doit les exécuter.

Bon nombre d'exemples issus de la vie courante font appel à l'algorithmique : l'utilisation d'un smartphone, la gestion d'un feu tricolore mais également une recette de cuisine ou encore une notice de montage d'un meuble. Il existe des méthodes pour représenter un algorithme comme le langage naturel, le pseudo code, les algorigrammes... S'il n'est pas nécessaire de systématiser leur utilisation, celle-ci peut aider à comprendre plus facilement certains algorithmes simples.

Afin d'illustrer les éléments précédemment évoqués, nous allons présenter l'algorithme d'Euclide qui correspond à une suite finie de divisions euclidiennes aboutissant au calcul du PGCD de deux nombres entiers non nuls.

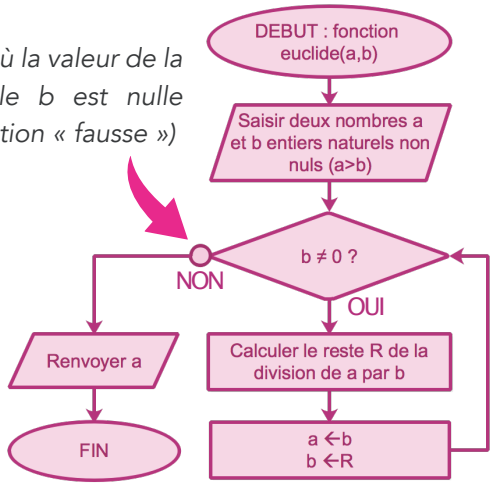
Pseudo code

```
Fonction euclide( a,b )
   Tant que b ≠ 0
      R ← a % b
      a ← b
      b ← R
   Fin Tant que
   Renvoyer a
Fin
```

: *a % b donne le reste de la division euclidienne de a par b*

Algorigramme

Cas où la valeur de la variable b est nulle (condition « fausse »)



Exemple de normalisation de la norme ISO 5807 :

Annoncer le début, une interruption temporaire ou la fin d'une procédure.

Utiliser des blocs d'instruction conditionnelle et des boucles.

Indiquer une phase de traitement.

Traiter les instructions de type entrée / sortie

Environnement

Il existe de nombreuses applications qui permettent à la fois d'écrire le code du programme (que l'on appelle communément un script en langage Python), de l'interpréter et de voir le résultat. Ces applications appelées environnements ou IDLE sont disponibles sous différentes plateformes comme les ordinateurs, calculatrices, tablettes ou encore les smartphones. De par sa compatibilité avec divers systèmes d'exploitation (Windows©, Linux©, Mac OSX©) nous utiliserons la suite Anaconda qui possède différentes applications dont notamment Spyder qui sera majoritairement utilisé dans ce document.

À noter que les différentes manipulations présentées dans ce document sont transposables sur d'autres environnements utilisant le langage Python.

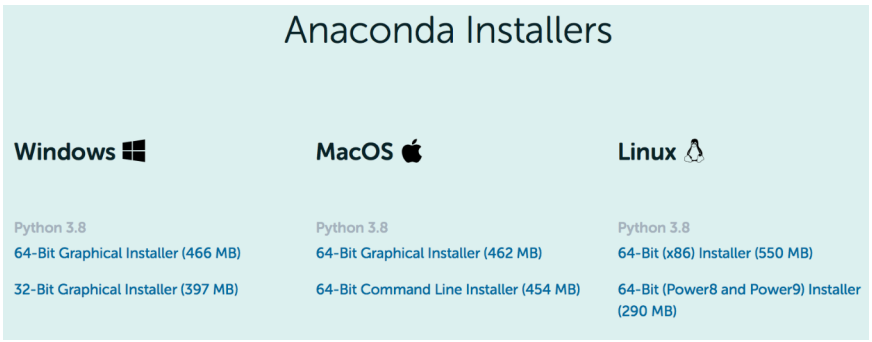
Suite Anaconda

Lien de téléchargement : https://www.anaconda.com/products/individual

Choisir le système d'exploitation.

Pour Windows , attention à bien choisir la version 64-Bit ou 32-Bit, elle se réfère à la version de votre système.

Une fois le fichier téléchargé, installer la suite Anaconda.

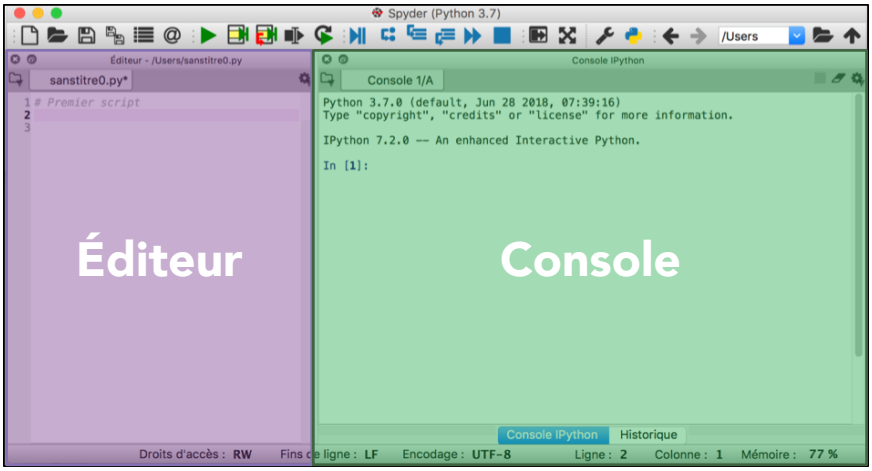


Pour les manipulations qui vont suivre, nous utiliserons l'application **Spyder** (Scientific **P**ython **D**eveloppement **E**nvi**R**onment), qui est un environnement de développement, disponible lorsqu'on lance la suite Anaconda.

À l'ouverture de l'application, deux sections sont visibles à l'écran : l'éditeur et la console.

Il est possible de configurer l'interface en français via les préférences du logiciel.

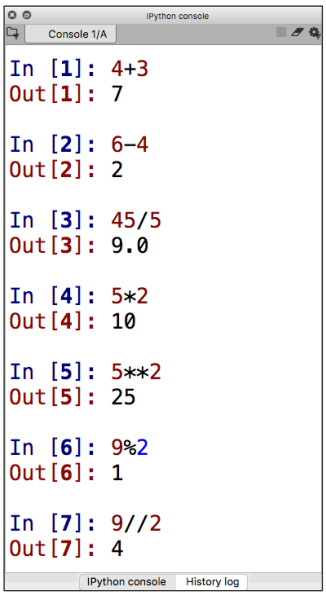
Nous allons exposer de manière succincte le rôle de chacune d'entre elles ci-après.



La console (ou Shell)

La console est une interface entre l'utilisateur et l'interpréteur Python. Son fonctionnement est comparable à celui d'une calculatrice : les instructions sont écrites directement dans la console puis exécutées par l'interpréteur. Le résultat éventuel est alors affiché à la ligne suivante. Ainsi, la console peut être utilisée pour réaliser divers calculs à l'aide d'une syntaxe spécifique tout en tenant compte des priorités.

Bien qu'elle reste limitée, la console est facile d'utilisation et permet d'effectuer des tests rapides, de vérifier une syntaxe et d'autres points que nous évoquerons ultérieurement.



L'Éditeur

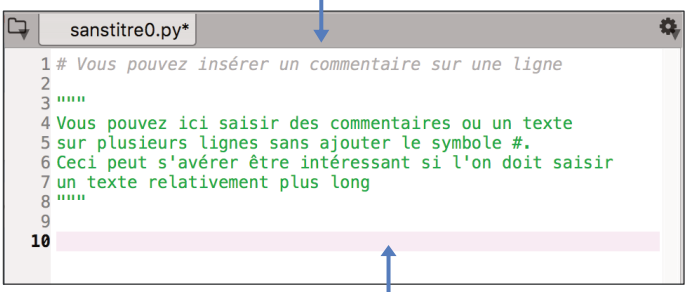
Lorsque le nombre d'instructions à traiter est trop conséquent, l'utilisation de la console a ses limites. Nous utiliserons dans ce cas l'éditeur (partie gauche) qui présente un certain nombre d'avantages par rapport à la console :

- conserver les instructions (et ainsi pouvoir les exécuter à plusieurs reprises) ;

- exécuter une séquence d'instructions relativement longue ;

- enregistrer un certain nombre d'instructions dans un fichier, appelé script, dont l'extension est **« *.py* »** que l'on pourra réutiliser à tout moment.

Pour saisir un commentaire sur une ligne, ou en regard d'une instruction, saisir un # juste avant le début du texte.



Lorsque la taille de ce commentaire est plus importante, une autre variante consiste à utiliser les triples guillemets.

Ainsi, tout le texte qui sera saisi entre ces symboles sera considéré comme un commentaire.

Le code est à saisir directement dans l'éditeur et non plus dans la console.

Les bibliothèques

Le langage Python permet l'utilisation de nombreuses instructions qui ne sont pas toutes disponibles par défaut dans l'éditeur. C'est pourquoi il est nécessaire, en fonction des besoins du script à saisir, d'importer soit une instruction unique, soit un module (ou bibliothèque) regroupant plusieurs instructions. Voici quelques-unes des bibliothèques les plus couramment utilisées :

- `math` permet d'avoir accès à certains nombres (« pi », « e », etc.), mais également à des fonctions mathématiques telles que `cos`, `sin`, `sqrt` (la racine carrée), `log` pour le logarithme népérien, `log10` pour le logarithme décimal, etc. ;

- `random` regroupe des instructions de gestion des nombres aléatoires ;

- `scipy` contient des fonctionnalités relatives à l'algèbre linéaire et aux statistiques ;

- `matplotlib` propose de nombreux types de représentations graphiques ;

- `turtle` permet la création de figures et de dessins ;

- `numpy` permet de manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux.

Les instructions suivantes permettent d'importer tout ou partie d'une bibliothèque :

```
import math                         # import de la bibliothèque math
from math import sqrt, cos          # import des instructions racine carrée et cosinus contenues dans la bibliothèque math

import matplotlib.pyplot as plt     # import de la bibliothèque matplot que l'on définit comme un objet plt

import numpy as np                  # import de la bibliothèque numpy que l'on définit comme un objet np
```

Il est possible d'obtenir de l'aide sur une instruction ou une bibliothèque en saisissant `help()` dans la console puis d'indiquer l'instruction à développer. Dans l'exemple suivant, on souhaite obtenir de l'aide sur l'instruction `abs`. Pour quitter la rubrique d'aide, il suffit de saisir `quit` dans la console.

```
In [1]: help()
```

```
help>
```

```
help> abs
Help on built-in function abs in module builtins:

abs(x, /)
    Return the absolute value of the argument.
```

Programmation fonctionnelle

La programmation fonctionnelle est clairement préconisée dans les programmes de mathématiques de la voie générale et technologique. C'est pourquoi, afin d'œuvrer pour la sécurisation du parcours de l'élève, l'ensemble des situations développées dans ce document se basera sur ce type de programmation, aussi bien en mathématiques qu'en physique-chimie.

Contrairement à la programmation de type « entrée(s) / sortie(s) » faisant appel aux instructions `input` ou `print`, la programmation fonctionnelle est de type déclaratif considérant le calcul en tant qu'évaluation de fonctions mathématiques ce qui peut favoriser la décomposition d'une tâche complexe en plusieurs tâches simples.

Ce paradigme permet de proposer des codes précis et concis en étant facilement testables et vérifiables. En ne gérant pas les objets, et donc les états, on limite de ce fait les effets de bord. D'autre part, le fait de diviser un programme en plusieurs fonctions permet de faciliter son débogage, c'est-à-dire le processus de diagnostic pas à pas qui permet de localiser les erreurs.

Les fonctions

Les fonctions Python ne sont pas exclusivement des fonctions qui dépendent de variables numériques. En programmation, une fonction permet de traiter des paramètres de type numérique (entier ou flottant), mais également des chaînes de caractères (string), des listes, et même une autre fonction.

De la même manière, lorsqu'une fonction renvoie un résultat, celui-ci pourra être de différents types (numérique, liste, fonction, tuple, dictionnaire, booléen, valeur aléatoire, etc.), et même des types différents de ceux utilisés en arguments.

Pour définir une fonction en langage Python, on utilise la commande `def` en prenant soin de respecter sa syntaxe (nom, parenthèses, « : » et indentation à la ligne suivante) comme le présentent les exemples ci-après.

Exemples en mathématiques

```
def PVTC(PVHT,taux):
    return PVHT*(1+taux/100)

from random import randint
def quatre():
    return randint(1,6)==4

def freq_4(n):
    c = 0
    for i in range(n):
        if quatre():
            c += 1
    return c/n
```

Exemples en physique-chimie

```
def poids(m):
    return m*9,81

def pression(F,S):
    return F/S

Vx = [6.6,6.6,6.6,6.6]
Vy = [8.45,6.45,4.50,2.55]
def Ec(m):
    Ec=[]
    for i in range(4):
        Ec.append(0.5*m*(Vx[i]**2+Vy[i]**2))
    return Ec
```

La commande `return` permet d'indiquer le résultat que doit renvoyer la fonction. Elle interrompt l'exécution de la fonction et renvoie un résultat. Il faut noter qu'on dit bien renvoyer et non retourner un résultat.

Une fois la fonction définie dans le script, elle peut être appelée dans l'éditeur par une autre fonction, mais elle peut également l'être dans la console.

Gérer les fonctions

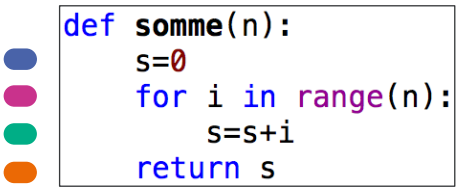
L'indentation

En plus de faciliter la lecture du code, l'indentation est une tabulation qui permet de hiérarchiser sa structure et plus particulièrement les différentes instructions ou blocs d'instructions qui la constitue. Avec le langage Python, l'indentation est aussi importante que le code. Une mauvaise indentation génère des erreurs de syntaxe, mais parfois également des erreurs invisibles lors de l'exécution du script, qui renvoie alors un résultat incohérent.

Afin de comprendre le principe d'indentation, étudions un exemple de fonction appelée `somme` qui renvoie la somme des nombres entiers successifs strictement inférieurs à un nombre `n` donné. Le nombre `n`, qui se trouve entre parenthèses est alors appelé argument de la fonction `somme`.

```
Fonction somme(n)
    s ←0
    Pour i allant de 0 à n-1
        s ← s +1
    Fin Pour
    Renvoyer s
Fin
```

Examinons en détail les étapes de la rédaction de ce premier script.



```
def somme(n):
    s=0
    for i in range(n):
        s=s+i
    return s
```

La variable `s` permet de stocker en mémoire les valeurs successives de la somme des nombres entiers. On initialise sa valeur à 0, c'est-à-dire que l'on affecte la valeur `0` à la variable `s`.

Nous allons maintenant utiliser une boucle dite bornée qui permet de répéter un certain nombre d'instructions : pour chaque nombre entier allant de `0` à `n-1`, on met à jour la valeur de la variable `s`.

Un nouveau niveau d'indentation est alors créé spécifiquement pour la boucle `for`. Cela signifie que l'ensemble des instructions saisies à ce nouveau niveau d'indentation seront répétées n fois.

Pour chaque valeur `i` comprise entre `0` et `(n-1)`, on ajoute cette valeur `i` à la valeur précédente de la variable `s`. Cette incrémentation est réalisée à l'aide d'une des deux commandes :

```
s+=i      s=s+i
```

Aucune instruction spécifique n'est à saisir pour sortir de la boucle `for` : il suffit de revenir à un niveau d'indentation inférieur.

La dernière ligne renvoie la valeur de la variable `s` qui correspond à la somme recherchée. Pour cela, on utilise l'instruction `return`.

Remarque importante

L'instruction `return` est utilisée pour renvoyer un résultat suite à l'appel à une fonction donnée (ici la fonction `somme`). Elle permet en outre de stopper tout processus en cours.

Dans l'exemple ci-contre, l'instruction `return s` est positionnée à un niveau d'indentation supérieur. En l'état, cette instruction mettrait donc fin à l'appel `somme(n)` immédiatement après la première itération de la boucle `for`, c'est à dire lorsque la variable `i` a pour valeur `0`.

```
def somme(n):
    s=0
    for i in range(n):
        s=s+i
        return s
```

Pour exécuter ce script, il faut cliquer sur l'icône ▶ disponible dans la barre d'outil. Il peut être nécessaire d'enregistrer un fichier d'extension .py si cela n'a pas encore été fait.

On appelle alors la fonction `somme` pour différentes valeurs de `n` dans la console d'exécution.

```
In [1]: runfile('/Users/[illegible].py',
wdir='/Users/[illegible]/Documents')

In [2]: somme(4)
Out[2]: 6

In [3]: somme(7)
Out[3]: 21
```

À noter que les touches directionnelles du clavier permettent de naviguer dans l'historique des opérations réalisées dans la console.

Variables informatiques

Une variable peut être imaginée comme une étiquette fichée sur une boîte qui peut contenir tous types de valeurs. Le contenu de chaque boîte varie au cours de l'exécution d'un programme, ce qui n'est pas le cas d'une variable mathématique.

Pour affecter une valeur à une variable, c'est-à-dire l'initialiser ou modifier sa valeur, on utilise l'opérateur d'affectation « = » à distinguer du signe égal employé en mathématiques pour énoncer une égalité. Quand on écrit « `a = 5` » en Python, on range 5 dans une boîte qui porte l'étiquette a et qu'on appelle, par simplification, a. On ne lira pas « a est égal à 5 » mais « a reçoit 5 » (en pseudo-code, on écrit : « `a ← 5` »).

À gauche de l'opérateur on retrouve le nom de la variable et à droite la valeur qu'on souhaite lui affecter.

Les noms de variable, toujours à gauche du "=", peuvent comporter plusieurs lettres simples, ainsi que des chiffres (sauf au début), mais pas de caractères spéciaux (sauf _). Il est recommandé de choisir des noms de variables évocateurs, pour faciliter la lecture des programmes.

Pour accéder au contenu d'une variable, il suffit d'utiliser son nom et l'on peut alors effectuer des opérations avec celle-ci. Par exemple, on change l'affectation de `a` en écrivant « `a = 3` » ou même « `a = a + 1` », ce qui augmente le contenu de la mémoire `a` de 1. Dans ce type d'écriture, l'ordinateur réalise d'abord l'opération qui est à droite du signe égal, puis affecte le résultat à la variable dont le nom est à gauche.

Affectations multiples : si l'on écrit "`a,b = 5,2`", on affecte 5 à `a` et 2 à `b`, et si l'on écrit "`a,b = b,a`" cela échange les contenus `a` et `b`.

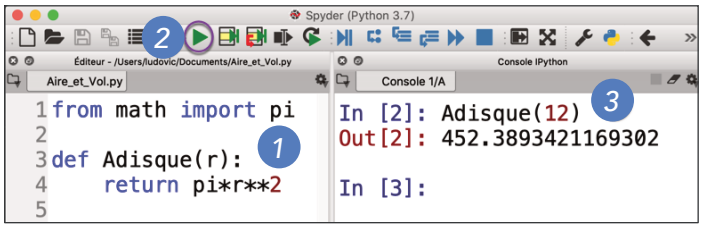
Premières applications

Les premières applications que nous proposons ont pour objectif de vous familiariser avec la programmation fonctionnelle à l'aide de plusieurs fonctions qui illustrent, entre autres, les exemples d'algorithmes proposés dans le module de géométrie spécifié dans le BO spécial N°5 du 11 Avril 2019. Ces fonctions seront regroupées dans un unique script que l'on nommera `Aire_et_Vol.py`

Surface d'un disque

Trouver l'aire d'un disque de rayon 12 centimètres.

Nous créons dans un premier temps un script nommé `Aire_et_Vol.py` dans lequel est saisie la fonction `Adisque`.



Voici les étapes à suivre :

1. Écrire le script dans l'éditeur et l'enregistrer ;
2. Exécuter ce script ▶. NB : chaque action sur cette flèche provoque l'enregistrement du script ;
3. Appeler `Adisque(12)` dans la console d'exécution afin de répondre à la consigne.

Remarque : Le nombre π fait partie de la bibliothèque « `math` », il est donc nécessaire de l'importer en début de script avant de définir la fonction `Adisque`.

```
from math import pi

def Adisque(r):
    return pi*r**2
```

À noter qu'une fois la fonction créée, elle peut être utilisée pour calculer l'aire de disques de rayons différents.

Volume d'un cylindre

Calculer le volume d'un cylindre de révolution de hauteur 15 cm et de rayon de base 4 cm.

Dans l'éditeur nous pouvons créer, après la fonction `Adisque`, la fonction `Vcyl` ci-contre.

```
def Vcyl(r,h):
    return pi*r**2*h
```

Le nombre π ayant été importé en début de script, il n'est pas nécessaire de le faire à nouveau. Il est maintenant disponible à tout moment. Ceci est valable pour toute instruction ou tout nombre importé d'une bibliothèque.

On exécute une fois de plus le programme dans l'éditeur ▶ puis on saisit `Vcyl(4,15)` dans la console.

```
In [3]: Vcyl(4,15)
Out[3]: 753.9822368615503
```

Remarque : Il aurait été possible d'exploiter la fonction `Adisque`.

```
def Vcyl2(r,h):
    return Adisque(r)*h
```

Diamètre d'un cylindre

Calculer le diamètre d'un cylindre de révolution de hauteur 15 cm et de volume 754 cm³.

La formule du diamètre D est donnée par la relation : $2 \times \sqrt{\frac{V_{cyl}}{\pi \times h}}$

où h représente la hauteur du cylindre et V_{cyl} son volume. Pour cette fonction, il convient d'importer la fonction racine carrée (`sqrt`) du module `math`. Deux syntaxes sont envisageables :

```
from math import pi
from math import sqrt
```

⇄

```
from math import pi,sqrt
```

Dans l'éditeur, saisir une nouvelle fonction nommée `diam`.

```
def diam(h,V):
    return 2*sqrt(V/(pi*h))
```

Exécuter le script ▶ puis saisir `diam(15,754)` dans la console.

```
In [4]: diam(15,754)
Out[4]: 8.00009423582128
```

Prolongement possible

Rechercher le rayon d'un cylindre de hauteur 15 cm et de volume 754 cm³ par balayage.

La fonction `Vcyl` étudiée précédemment est une fonction continue, strictement croissante sur $\mathbb{R}^+$ et donc plus particulièrement sur l'intervalle [0 ; 10].

```
In [2]: Vcyl(0,15)
Out[2]: 0.0

In [3]: Vcyl(10,15)
Out[3]: 4712.38898038469
```

Le théorème de la bijection nous permet alors d'affirmer qu'il existe une valeur unique r appartenant à l'intervalle [0 ; 10] telle que Vcyl(r,15) =754.

L'algorithme et la fonction détaillés ci-dessous permettent de trouver un encadrement du rayon de ce cylindre (la variable « `pas` » mise en argument permet de définir la précision de l'encadrement).

```
Fonction balayage ( pas )
    r ← 0
    Tant que Vcyl(r,15) ⩽ 754
        r ← r + pas
    Fin Tant que
    Renvoyer r - pas, r
Fin
```

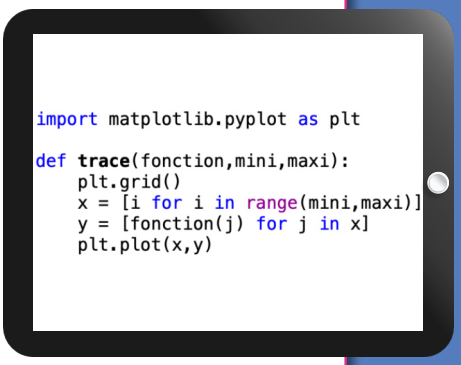
⇄

```
def balayage(pas):
    r = 0
    while Vcyl(r,15) <= 754:
        r = r + pas
    return r - pas, r
```

Exécuter le script ▶ puis appeler la fonction `balayage` en choisissant `pas = 0.1` :

```
In [5]: balayage(0.1)
Out[5]: (4.000000000000002, 4.100000000000001)
```

Dans cette partie, les activités proposées illustrent l'utilisation de l'algorithmique et de la programmation en Python dans les différents programmes de mathématiques & physique-chimie. L'accent est donc mis autour de ces notions mais il faut rappeler qu'elles doivent être abordées de manière transversale en mathématiques et constituer un appui au service des thématiques en physique-chimie.



```
import matplotlib.pyplot as plt

def trace(fonction,mini,maxi):
    plt.grid()
    x = [i for i in range(mini,maxi)]
    y = [fonction(j) for j in x]
    plt.plot(x,y)
```

Les scénarios pédagogiques ne sont que des propositions. Ils devront être adaptés en fonction de la progression retenue par l'enseignant. C'est pourquoi aucun critère d'évaluation n'y est annexé, les compétences mobilisées ne tenant compte d'aucune progression.

Activités

L'ensemble des scripts utilisés dans les activités est disponible à partir du lien suivant :
http://maths-physique-chimie-medias.discipline.ac-lille.fr/scripts.zip

Distance d'arrêt
Thème : Fonctions
Niveau : Seconde voie professionnelle

Date limite de consommation
Thème : Calcul commerciaux et financiers
Niveau : Seconde voie professionnelle

Perpendicularité de deux murs
Thème : Géométrie
Niveau : Seconde voie professionnelle

Part de marché
Thème : Statistiques à deux variables quantitatives
Niveau : Première voie professionnelle

Le jeu du craps
Thème : Fluctuations d'une fréquence selon les échantillons, probabilités
Niveau : Seconde voie professionnelle

Lois de Snell-Descartes

Thème : Optique
Niveau : Seconde voie professionnelle

Masques de protection

Thème : Fonctions
Niveau : Seconde voie professionnelle

Château de cartes

Thème : Suites numériques
Niveau : Première voie professionnelle

Émission de SO_2

Thème : Suites numériques
Niveau : Terminale voie professionnelle

Algorithme glouton

Thème : Accompagnement renforcé
Niveau : Première voie professionnelle

Analyse fréquentielle

Thème : Statistique à une variable
Niveau : Seconde voie professionnelle

Jeu du franc carreau

Thème : Fluctuations d'une fréquence selon les échantillons, probabilités
Niveau : Seconde voie professionnelle

Le défi du spaghetti

Thème : Fluctuations d'une fréquence selon les échantillons, probabilités
Niveau : Seconde voie professionnelle

Filtrage d'une tension

Thème : Obtenir un courant continu à partir d'un courant alternatif et inversement
Niveau : Terminale voie professionnelle

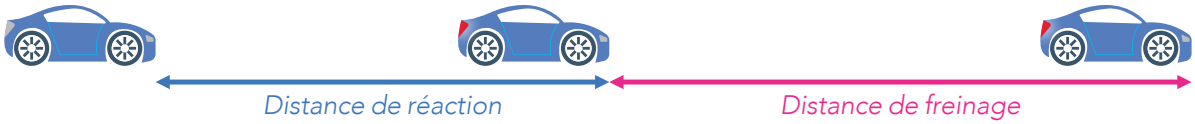
Distance d'arrêt

Situation

La distance d'arrêt correspond à la distance totale parcourue par un véhicule entre le moment où le conducteur perçoit un danger et le moment où le véhicule est totalement immobile. Celle-ci est calculée à partir de la somme de deux distances :

– La distance de réaction qui correspond à la distance parcourue le temps que le conducteur réagisse. Celle-ci est modélisée par la fonction d_R définie sur un intervalle $[10 ; 130]$ par $d_R = 0{,}28\ v - 0{,}2$ où v représente la vitesse du véhicule en km/h.

– La distance de freinage qui correspond à la distance parcourue pendant que le conducteur appuie sur la pédale de frein. Cette distance est modélisée par la fonction d_f définie sur l'intervalle $[10 ; 130]$ par $d_f = 0{,}0067\ v^2$ où v représente la vitesse du véhicule en km/h.



Sur une route de campagne où la vitesse est limitée à 80 km/h, les traces de freinage, liées à un accident, indiquent une distance d'arrêt de 82 m.

Le véhicule était-il en infraction lors de l'accident ?

Déroulement

L'ensemble des fonctions Python est à saisir dans un même script nommé DA.

1. Donner l'expression de la fonction d qui à tout nombre v de l'intervalle $[10 ; 130]$ fait correspondre la distance d'arrêt.

2. Écrire en Python la fonction `distance_arret` avec `v` comme argument qui traduit la fonction d précédente .

3. Effectuer les appels suivants : `distance_arret(10)`, `distance_arret(50)`, `distance_arret(80)`, `distance_arret(90)` et `distance_arret(130)` dans la console Python et conjecturer le sens de variation de la fonction d sur l'intervalle $[10 ; 130]$.

4. On cherche à déterminer la vitesse v_i du véhicule au moment où le conducteur a appuyé sur la pédale de frein. À l'aide des données de l'énoncé, reformuler mathématiquement la consigne.

5. Indiquer le rôle de la fonction `BDA` ci-contre.

6. Modifier la fonction `BDA` afin qu'elle permette d'étudier n'importe quelle distance d'arrêt que l'on appellera `Dlim`.

7. Répondre à la problématique en utilisant de manière appropriée la fonction `BDA`.

```
def BDA(pas=1):
    vi = 10
    while distance_arret(vi) <= 149:
        vi += pas
    return vi - pas,vi
```

- *Identifier la variable d'une fonction.*
- *Déterminer l'image ou des antécédents éventuels d'un nombre par une fonction définie sur un ensemble donné.*
- *Dans le cadre de problèmes modélisés par des fonctions, résoudre par une méthode algébrique une équation du type* $f(x) = c$.
- *Repérer les enchaînements logiques et les traduire en boucles.*
- *Réaliser un calcul à l'aide d'une ou plusieurs variables.*
- *Modifier un algorithme ou un programme.*

1 La distance d'arrêt correspond à la somme des distances de réaction et de freinage en fonction de la vitesse v exprimée en km/h. La fonction d est donc définie sur l'intervalle $[10 \ ; \ 130]$ par :

$$d(v) = 0{,}0067 \ v^2 + 0{,}28 \ v - 0{,}2$$

2 L'expression trouvée au point précédent permet d'écrire relativement facilement la fonction Python demandée :

```
def distance_arret(v):
    assert 10 <= v <= 130
    return 0.0067*v**2 + 0.28*v - 0.2
```

L'instruction `assert` permet de restreindre les valeurs de v au domaine de définition. On pourra, dans un premier temps, accepter de rendre cette étape facultative.

• savoir écrire une fonction Python simple

3

```
In [2]: distance_arret(10)
Out[2]: 3.27

In [3]: distance_arret(50)
Out[3]: 30.55

In [4]: distance_arret(80)
Out[4]: 65.08

In [5]: distance_arret(90)
Out[5]: 79.27

In [6]: distance_arret(130)
Out[6]: 149.43
```

Les différents appels successifs demandés sont réalisés dans la console d'exécution.

Au regard des valeurs obtenues, on peut conjecturer que la fonction d est strictement croissante sur l'intervalle $[10 \ ; \ 130]$.

• appeler une fonction Python dans la console d'exécution

4 L'objectif est ici de déterminer la valeur de la vitesse v_i correspondant à une distance d'arrêt de 82 m.

Ceci revient à résoudre sur l'intervalle $[10 \ ; \ 130]$ l'équation d'inconnue v_i :

$$d(v_i) = 82.$$

5 La fonction `BDA` permet de renvoyer un encadrement de la vitesse correspondant à une distance d'arrêt de 149 m.

On réalise un balayage sur le domaine de définition de la fonction d afin de déterminer la plus grande valeur v_i telle que $d(v_i) \leq 149$.

• comprendre et utiliser des fonctions

L'argument `pas` *correspond à l'amplitude d'encadrement que l'on souhaite faire de la vitesse. On définit ici la valeur 1 par défaut pour la variable* `pas` *ce qui la rendra facultative lors d'un appel ultérieur à la fonction* `BDA`*.*

```
In [2]: BDA()
Out[2]: (129, 130)

In [3]: BDA(0.1)
Out[3]: (129.69999999999692, 129.7999999999969)
```

```
def BDA(pas=1):
    vi = 10
    while distance_arret(vi) <= 149:
        vi += pas
    return vi - pas,vi
```

Tant que la distance d'arrêt est inférieure ou égale à 149, on incrémente la vitesse `vi` *d'une valeur correspondant au* `pas` *(qui ici vaut 1 par défaut).*

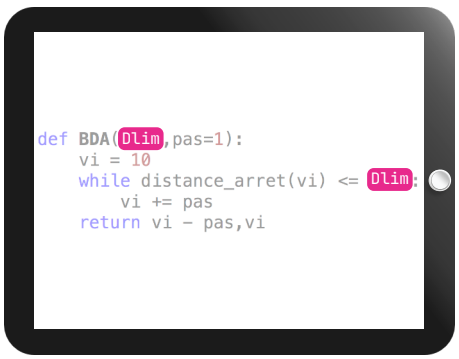
`vi` *correspond à la première valeur de la vitesse relative à une distance d'arrêt strictement supérieure à 149.*

`vi-pas` *correspond donc à la dernière valeur de la vitesse relative à une distance d'arrêt inférieure ou égale à 149.*

6 Le point précédent nous permet de constater que si un véhicule roule à 130 km/h, il aura besoin d'environ 149 m pour être totalement immobilisé.

Pour traiter une distance d'arrêt donnée, il convient donc de la passer en argument (`Dlim`) dans la fonction `BDA`.

• modifier une fonction Python




```
In [3]: BDA(82)
Out[3]: (91, 92)

In [4]: BDA(82,0.1)
Out[4]: (91.79999999999907, 91.89999999999907)
```

7 Les modifications précédentes ayant été apportées à la fonction `BDA`, l'appel `BDA(82)` permet de déterminer un encadrement à l'unité de la vitesse correspondant à une distance d'arrêt de 82 m. Le véhicule roulait donc à une vitesse comprise entre 91 et 92 km/h.

Le conducteur était en infraction, la vitesse sur route de campagne étant limitée à 80 km/h.

Date limite de consommation

Situation

Afin de lutter efficacement contre le gaspillage alimentaire, une chaîne de supermarchés souhaite créer une application permettant aux clients de bénéficier d'une remise exceptionnelle s'ils trouvent et achètent un produit non signalé en magasin et dont la date limite de consommation (DLC) est inférieure ou égale à un jour. Cette remise s'élève à :

- 40 % du prix affiché pour une DLC le lendemain ;
- 60 % du prix affiché pour une DLC le jour-même.

Comment mettre en place ce type d'application ?

Déroulement

L'ensemble des fonctions Python est à saisir dans un même script nommé `DLC`.

La fonction `DJ` ci-contre prend comme arguments deux chaînes de caractères correspondant à deux dates saisies au format jour/mois/année.

```
from datetime import datetime

def DJ(d1,d2):
    d1 = datetime.strptime(d1,'%d/%m/%Y')
    d2 = datetime.strptime(d2,'%d/%m/%Y')
    return (d2 - d1).days
```

1. Effectuer, dans la console Python, les appels suivants :
   - `DJ('10/10/1973','14/10/1973')`
   - `DJ('31/12/2021','01/01/2022')`

2. En déduire le rôle de la fonction `DJ`.

La chaîne de supermarché souhaite utiliser l'algorithme suivant afin de déterminer la remise à appliquer sur le prix d'un produit en fonction de sa DLC.

3. Indiquer les éléments manquants A et B de l'algorithme ci-contre en vous appuyant sur la situation.

4. Préciser la signification du résultat renvoyé par la fonction `prix_remise`.

5. Écrire en Python la fonction `prix_remise` correspondant à l'algorithme ci-contre.

```
Fonction prix_remise (P_init, date_jour,DLC):
    nb_j ← Résultat de l'appel DJ(date_jour, DLC)
    Si nb_j = 1 alors
        result = P_init x A
    Sinon si nb_j = 0 alors
        result = P_init x B
    Sinon
        result = False
    Fin Si
    Renvoyer nb_j,result
Fin
```

6. En utilisant la fonction `prix_remise`, calculer le montant à payer après remise pour un produit affiché à 7,84 € lors d'un achat le 05 Novembre 2021, et dont la DLC est :
   - le 06 Novembre 2021 ;
   - le 07/11/21 ;
   - le 04 Novembre 2021.

7. Analyser les résultats obtenus et préciser s'ils sont cohérents au regard de la situation.

8. En utilisant l'instruction `round`, modifier la fonction `prix_remise` afin qu'elle renvoie un prix remisé au centième.

- *Repérer des enchaînements logiques et les traduire en instructions conditionnelles.*
- *Réaliser un calcul à l'aide d'une variable.*
- *Modifier ou compléter un algorithme ou un programme.*
- *Lire et comprendre des fonctions Python simples.*

1 On réalise les deux appels dans la console d'exécution.

```
In [2]: DJ('10/10/1973','14/10/1973')
Out[2]: 4

In [3]: DJ('31/12/2021','01/01/2022')
Out[3]: 1
```

2 Le résultat renvoyé par la fonction `DJ` est un nombre entier dont la valeur correspond au nombre de jours écoulés entre les dates `d1` et `d2` mises en arguments.

3 L'algorithme présenté illustre la démarche de réduction évoquée dans la situation, c'est-à-dire la calcul de la valeur de la remise à appliquer au produit. Les éléments A et B correspondent aux coefficients multiplicateurs à appliquer au prix initial du produit en fonction des deux cas :

```
Si nb_j = 1 alors
      result = P_init x 0.6
```

Si la DLC est le lendemain de l'achat, on applique une remise de 40 % soit un coefficient multiplicateur de 0,6.

```
Sinon si nb_j = 0 alors
      result = P_init x 0.4
```

Si la DLC est le jour de l'achat, on applique une remise de 60 % soit un coefficient multiplicateur de 0,4.

- comprendre les différentes étapes d'un algorithme
- compléter un algorithme

4 La fonction `prix_remise` renvoie un tuple dont les valeurs correspondent :

- au nombre de jours séparant la date courante et la DLC du produit pour la variable `nb_j` ;
- au montant du produit après remise pour la variable `result`.

- comprendre une fonction Python
- connaître les instructions conditionnelles

5

```
def prix_remise (P_init,date_jour,DLC) :
    nb_j = DJ(date_jour,DLC)
    if nb_j == 1 :
        result = P_init*0.6
    elif nb_j == 0 :
        result = P_init*0.4
    else :
        result = False
    return nb_j,result
```

En tenant compte des éléments A et B précisés au point 4, la fonction Python `prix_remise` détaillée ci-contre traduit l'algorithme proposé.

• traduire un algorithme en une fonction Python

6

La fonction `prix_remise` fait appel à la fonction `DJ` étudiée au point 1. Cette dernière prend comme arguments deux dates qu'il convient de saisir sous la forme d'une chaîne de caractères.

Une fois la fonction `prix_remise` saisie, on exécute le script à l'aide du bouton ▶, puis on réalise les trois appels suivants dans la console d'exécution :

```
In [4]:
prix_remise(7.84,'05/11/2021','06/11/2021')
Out[4]: (1, 4.704)

In [5]:
prix_remise(7.84,'05/11/2021','07/11/2021')
Out[5]: (2, False)

In [6]:
prix_remise(7.84,'05/11/2021','04/11/2021')
Out[6]: (-1, False)
```

On peut donc conclure que pour une DLC affichée au :

- 06 Novembre 2021 : le montant à payer s'élève à 4,704 € ;
- 07/11/21 : aucune remise n'est accordée ;
- 04 Novembre 2021 : aucune remise n'est accordée, la date du jour étant supérieure à la DLC.

Remarque : chaînes de caractères

Les chaînes de caractères sont soumises à une syntaxe particulière, comme on peut le voir dans l'exemple de la fonction `DJ`.

Le langage Python permet de délimiter une chaîne de caractères soit avec des apostrophes, soit avec des guillemets, ceci dans le but de pouvoir intégrer ces deux symboles dans la chaîne de caractères étudiée.

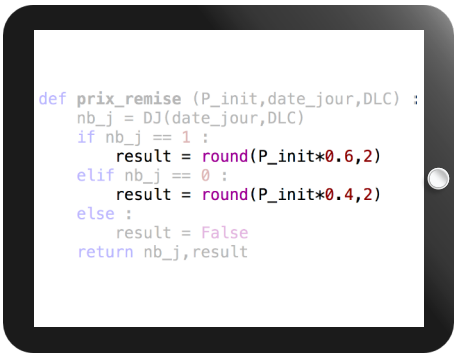
```
In [9]: ch1 = 'Utilisez des "guillemets" svp.'

In [10]: ch2 = "utilisez des 'apostrophes' svp."

In [11]: ch1,ch2
Out[11]: ('Utilisez des "guillemets" svp.', "utilisez des
'apostrophes' svp.")
```

7

Au regard des résultats précédents, la valeur de la variable `result` est un nombre à virgule flottante correspondant au produit d'un coefficient multiplicateur par le prix initial du produit à acheter.
Dans le premier cas de figure (DLC au 06 Nombre 2021), le prix après remise s'élève à 4,704 € ce qui n'est pas une valeur adéquate pour représenter un montant exprimé en euro.

8 On souhaite maintenant que la fonction `prix_remise` renvoie une valeur arrondie au centième du montant après remise.

Cette valeur correspond à la variable `result` dont il faut modifier l'affectation en utilisant l'instruction `round` :



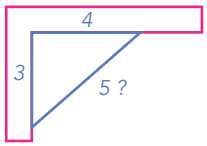
```
def prix_remise (P_init,date_jour,DLC) :
    nb_j = DJ(date_jour,DLC)
    if nb_j == 1 :
        result = round(P_init*0.6,2)
    elif nb_j == 0 :
        result = round(P_init*0.4,2)
    else :
        result = False
    return nb_j,result
```

- modifier une fonction Python
- connaître l'instruction round

Perpendicularité de deux murs

Situation

Pour vérifier si deux murs sont perpendiculaires, les maçons utilisent la règle du « 3 / 4 / 5 ». Le principe de cette règle consiste à repérer au crayon gris, depuis l'intersection des deux murs, une marque se situant à 3 m pour le premier et 4 m pour le second, puis de vérifier que les deux marques sont distantes de 5 m.



Comment vérifier simplement si deux murs sont perpendiculaires en utilisant la mesure directe des murs ?

Déroulement

L'ensemble des fonctions Python est à saisir dans un même script nommé MURS.
En accord avec l'appareil de mesure utilisé (le décamètre), les valeurs exprimées seront données au centimètre près.

1 Pour différentes valeurs des nombres `x`, `y` et `z`, réaliser différents tests de la fonction Python `T` ci-contre dans la console d'exécution.

```
def T(x,y,z):
    a = min( x, y, z)
    c = max( x, y, z)
    b = (x + y + z) - a - c
    return a,b,c
```

2 En déduire le rôle de la fonction `T`.

On considère maintenant un triangle où les nombres a, b et c représentent les longueurs de ces trois côtés avec c la longueur du côté le plus long.

3 En utilisant le théorème de Pythagore dans le triangle décrit ci-dessus, préciser parmi les trois fonctions Python suivantes celle qui permet de vérifier que ce triangle est rectangle. Justifier la réponse.

```
def triplet(a,b,c):
    a,b,c = T(a,b,c)
    return a**2 == c**2 + b**2
```

```
def triplet(a,b,c):
    a,b,c = T(a,b,c)
    return b**2 == a**2 + c**2
```

```
def triplet(a,b,c):
    a,b,c = T(a,b,c)
    return c**2 == a**2 + b**2
```

4 En utilisant la fonction Python `triplet`, préciser si les triangles suivants sont rectangles.
- ABC tel que AB = 15 cm, BC = 25 cm et AC = 20 cm ;
- PQR tel que PQ = 28 cm, QR = 21 cm et PR = 34 cm.

En pratique, les maçons considèrent qu'un écart de 5 % sur les mesures est acceptable. Cette liberté est liée à l'emploi du décamètre dont la précision est de cet ordre. Ainsi, on supposera de manière arbitraire la condition suivante : si le rapport de $(a^2 + b^2)$ par c^2 est compris entre 95 et 105 % alors les murs sont considérés comme perpendiculaires.

5 En utilisant un vocabulaire mathématique adapté, écrire la condition décrite ci-dessus.

6 Écrire en Python une fonction `triplet_app` prenant comme arguments les nombres `a`, `b` et `c` qui indique si le triangle est rectangle ou pas en tenant compte de la condition ci-dessus.

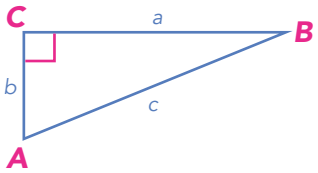
- *Reconnaître le type d'une variable.*
- *Modifier ou compléter un algorithme ou un programme.*
- *Comprendre et utiliser des fonctions Python.*
- *Calculer des longueurs.*
- *Utiliser le théorème de Pythagore.*

1 On réalise plusieurs appels dans la console d'exécution en prenant aléatoirement des nombres entiers pour définir les arguments x, y et z.

```
In [2]: T(5,3,9)
Out[2]: (3, 5, 9)

In [3]: T(1,7,45)
Out[3]: (1, 7, 45)

In [4]: T(23,19,5)
Out[4]: (5, 19, 23)

In [5]: T(12,45,6)
Out[5]: (6, 12, 45)
```

2 Les résultats obtenus suite à ces différents appels montrent que la fonction T a pour objectif de trier par ordre croissant trois valeurs mises en arguments, et de les renvoyer sous forme d'un tuple.

3 En considérant un triangle ABC donné, la réciproque du théorème de Pythagore permet de conclure que si $c^2=a^2+b^2$, alors le triangle ABC est rectangle en C.




```
def triplet(a,b,c):
    a,b,c = T(a,b,c)
    return c**2 == a**2 + b**2
```

Après avoir trié par ordre croissant les longueurs a, b et c des trois côtés du triangle, la fonction `triplet` ci-contre permet de comparer les valeurs de c^2 et de la somme $a^2 + b^2$, puis de renvoyer `True` si ces valeurs sont égales (et `False` sinon). Cela permet de vérifier si le triangle étudié est rectangle ou pas.

- comprendre les différentes étapes d'un algorithme
- compléter un algorithme

4 Au regard des éléments évoqués dans les points précédents, on effectue les deux appels ci-contre dans la console d'exécution qui permettent de montrer que le triangle ABC est rectangle en A alors que le triangle PQR ne l'est pas.

```
In [2]: triplet(15,25,20)
Out[2]: True

In [3]: triplet(28,21,34)
Out[3]: False
```

5 Ce point permet de mettre en œuvre différentes notations déclinées dans le thème « Vocabulaire ensembliste et logique ». Dans le cadre de cette étude, on considère que les murs sont perpendiculaires si et seulement si :

$$\frac{a^2 + b^2}{c^2} \in [\,0{,}95\,;\,1{,}05\,]$$

6 Tout comme la fonction `triplet`, la fonction `triplet_app` a pour objectif de vérifier que deux murs sont perpendiculaires à la différence près que cette dernière doit prendre en compte la condition précisée au point 5, d'où la modification à apporter sur le résultat du test renvoyé par la fonction `triplet_app` :

```
def triplet_app(a,b,c):
    a,b,c = T(a,b,c)
    return 0.95 <= (a**2 + b**2)/c**2 <= 1.05
```

• concevoir un programme simple pour résoudre un problème

Éléments de différenciation pédagogique

Un triplet (a, b, c) de nombres entiers naturels non nuls est appelé triplet pythagoricien lorsque les valeurs a, b et c vérifient la relation $a^2 + b^2 = c^2$.

En prolongement de l'activité, il est envisageable d'étudier comment déterminer l'ensemble des triplets pythagoriciens de 0 à n.

L'algorithme ci-contre consite à vérifier, pour tous nombres entiers naturels a et b compris entre 0 et n-1, si le nombre $\sqrt{a^2 + b^2}$ l'est également.

```
Fonction triplets (n):
    Pour a allant de 1 à n-1
        Pour b allant de a à n-1
            c ← √(a²+b²)
            Si c ∈ ℕ et c ≤ n
                Afficher a,b,c
            Fin Si
        Fin Pour
    Fin Pour
Fin
```

Pour cela, il est possible d'utiliser la méthode `is_integer()` qui renvoie `True` si un nombre est un entier naturel, et qui renvoie `False` dans le cas contraire.

```
def triplets(n):
  for a in range(1,n):
      for b in range(a,n):
          c = sqrt(a**2 + b**2)
          if c.is_integer() and c <= n:
              print(a,b,int(c))
```

Part de marché

Situation

Une grande marque de smartphone souhaite analyser ses ventes afin d'établir un plan prévisionnel d'investissement. Le tableau suivant recense les parts de marché mondiales de son modèle phare au cours des sept dernières années :

Rang de l'année (x_i)	1	2	3	4	5	6	7
Pourcentage de part de marché (y_i)	4,4	6,1	8,3	10	10,6	13,9	16,1

Source : IDC

En considérant que les ventes suivent la même évolution, quelle part de marché l'entreprise peut-elle espérer dans 10 ans ?

Déroulement

L'ensemble des fonctions Python est à saisir dans un même script nommé `SMARTPHONE`.

La fonction `ajustement` ci-dessous permet de réaliser l'ajustement affine d'un nuage de points de coordonnées (x ; y), toutes deux mises en arguments.

```
import numpy as np
import matplotlib.pyplot as plt

A = []
P = []

def ajustement(x,y):
    a,b, = np.polyfit(x,y,1)
    f = np.poly1d((a,b))
    plt.plot(x,f(x),"b--")
    plt.plot(x,y,'ro')
    return round(a,2), round(b,2)
```

1. Préciser l'utilité des deux premières lignes du script.

2. Au regard de la situation, indiquer à quoi correspondent les variables A et P, puis préciser leur type.

3. À partir du mémento mis à disposition à la fin du livret, expliquer l'utilisation de :
   a. l'instruction `a,b = np.polyfit(x,y,1)`.
   b. l'instruction `f = np.poly1d((a,b))`.
   c. L'instruction `round(a,2)`.

4. En utilisant la fonction `ajustement`, réaliser un ajustement affine du nuage de points relatif à la situation, et en déduire l'équation de la droite d'ajustement sous la forme $y=ax+b$.

La fonction R2 ci-contre permet de calculer le coefficient de détermination d'une série statistique à deux variables quantitatives x et y mises en arguments.

```
def R2(x,y):
    return np.corrcoef(x,y)[0,1]**2
```

5. En utilisant la fonction R2, déterminer le coefficient de détermination R^2 de la série statistique formée par le rang des années et les parts de marché correspondantes.

6. Valider la pertinence de l'ajustement affine réalisé pour cette série statistique.

7. Écrire en Python une fonction `part` qui prend comme argument le rang `x` de l'année et qui renvoie une estimation de la part de marché en pourcentage pour cette année.

8. En utilisant la fonction `part`, répondre à la problématique.

- *Représenter graphiquement à l'aide d'outils numériques un nuage de points associé à une série statistique à deux variables quantitatives.*
- *Réaliser un ajustement affine.*
- *Déterminer l'équation réduite d'une droite d'ajustement.*
- *Interpoler ou extrapoler des valeurs inconnues.*
- *Déterminer le coefficient de détermination d'une série statistique à deux variables quantitatives et évaluer la pertinence d'un ajustement affine.*
- *Concevoir un programme simple pour résoudre un problème.*
- *Comprendre et utiliser des fonctions Python.*
- *Reconnaître le type d'une variable.*

1

Les deux premières lignes du script ont pour objectif d'importer deux bibliothèques :

- `import numpy as np` : on crée un alias appelé `np` correspondant à la bibliothèque `numpy`. Cette bibliothèque permet de manipuler des tableaux multidimensionnels à l'aide de nombreuses fonctions notamment mathématiques ;
- `import matplotlib.pyplot as plt` : on crée un alias appelé `plt` correspondant à la bibliothèque `matplotlib`. Cette bibliothèque recense un certain nombre de fonctions permettant de représenter graphiquement des données.

2

Les variables `A` et `P` sont deux listes regroupant les valeurs de deux variables quantitatives, en l'occurrence ici le rang de l'année pour la variable `A` et le pourcentage de part de marché mondiale d'un modèle de smartphone pour la variable `P`. Au regard de la situation, on peut renseigner ces deux listes :

- connaître le type d'une variable

```
A = [i for i in range(1,8)]
P = [4.4,6.1,8.3,10,10.6,13.9,16.1]
```

3

Les indications données dans le mémento permettent de déterminer l'utilité des éléments suivants :

- L'instruction `np.polyfit(x,y,1)` vise à réaliser un ajustement polynomial de degré 1 (c'est-à-dire un ajustement affine) d'un nuage de points dont les coordonnées sont contenues dans deux listes notées `x` et `y`. L'instruction `a,b = np.polyfit(x,y,1)` affecte donc aux variables `a` et `b` les coefficients de ce polynôme (c'est-à-dire le coefficient directeur et l'ordonnée à l'origine de la droite d'ajustement du nuage de points étudié).
- L'instruction `f = np.poly1d((a,b))` permet de définir une fonction `f` définie sur $\mathbb{R}$ par $f(x) = ax + b$.
- L'instruction `round(a,2)` a pour objectif de renvoyer une valeur arrondie au centième de la variable `a`.

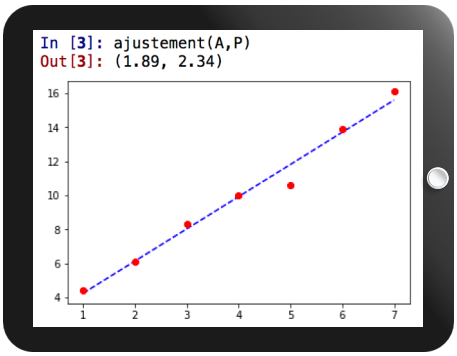
- connaître les fonctions de la bibliothèque numpy
- connaître l'instruction round

4 On réalise l'appel `ajustement(A,P)` afin de visualiser le nuage de points dont les coordonnées sont :

- Le rang de l'année en abscisse ;
- Le pourcentage de part de marché mondiale en ordonnée.

L'équation de la droite d'ajustement du nuage de points précédent est donc ici :

$$f(x) = 1{,}89x + 2{,}34$$



5 On réalise l'appel `R2(A,P)` dans la console d'exécution afin de connaître le coefficient de détermination de la série statistique à deux variables `A` et `P` étudiée dans cette situation :

```
In [4]: R2(A,P)
Out[4]: 0.9817284571939838
```

6 Le coefficient de détermination étant relativement proche de 1, on peut considérer que l'ajustement affine est pertinent dans cette situation.

7

```
def part(x):
    a,b = ajustement(A,P)
    return a*x+b
```

Au regard des différents éléments étudiés dans les points précédents, l'estimation de la part de marché pour une année de rang `x` est donnée par l'image de ce rang `x` par la fonction `f`, définie au point 3.

Les données proposées dans la situation nous indiquent les différents pourcentages de part de marché au cours des 7 dernières années, c'est-à-dire pour les années de rangs 1 à 7. Estimer le pourcentage de part de marché dans 10 ans revient donc à en déterminer la valeur pour l'année de rang 17. Cette valeur est déterminée dans la console d'exécution à l'aide de l'appel :

```
In [5]: part(17)
Out[5]: 34.47
```

En considérant la même évolution pour les dix prochaines années, on peut donc estimer à 34,5 % la part de marché mondial pour ce modèle de smartphone dans 10 ans.

- concevoir un programme simple pour résoudre un problème
- comprendre et utiliser des fonctions Python.

Le jeu du Craps

Situation

Le craps est un jeu de dés qui se pratique généralement dans les casinos et durant lequel on doit miser sur la chance qu'un joueur a de gagner une partie.

Comment estimer la chance de gagner au craps ?

Déroulement

L'ensemble des fonctions Python est à saisir dans un même script nommé DES.

1. Indiquer parmi les trois fonctions suivantes, celle qui permet de simuler la somme de deux dés. Justifier la réponse.

```
from random import randint

def d():
    return randint(1,6)+randint(1,6)
```

```
from random import randint

def d():
    return 2*randint(1,6)
```

```
from random import uniform

def d():
    return uniform(1,6)+uniform(1,6)
```

L'algorithme ci-contre permet de simuler une partie de craps avec des règles simplifiées et de vérifier si elle est gagnée ou pas.

```
Fonction craps (    ) :
    jeu ← Résultat de l'appel d()
    Si jeu = 7 ou jeu = 11
        Renvoyer "gagné"
    Sinon
        Renvoyer "perdu"
    Fin Si
Fin
```

2. Préciser, à l'aide de cet algorithme, les règles simplifiées du craps.

3. Écrire en Python la fonction `craps` correspondant à l'algorithme précédent.

4. Réaliser 10 essais avec la fonction `craps` et calculer la fréquence correspondante.

5. Indiquer si les 10 essais réalisés au point 4 sont suffisants pour répondre à la problématique. Justifier la réponse.

6. Compléter la fonction `freq` pour qu'elle renvoie la fréquence de parties de craps gagnées pour `n` parties jouées.

```
def freq(n):
    c=0
    for i in range(1,n+1):
        if craps()==  ?  :
            c+=1
    return  ?
```

7. Exécuter la fonction `freq` pour 10, 100, 1000 puis 10000 parties.

8. Répondre à la problématique « Comment estimer la chance de gagner au craps ? ». Justifier la réponse.

Pistes d'exploitation

- *Construire un modèle à partir des fréquences observées, en distinguant nettement modèle et réalité.*
- *Lire et comprendre une fonction Python renvoyant le nombre ou la fréquence de succès dans un échantillon de taille n pour une expérience aléatoire à deux issues.*
- *Observer la loi des grands nombres à l'aide d'une simulation sur Python ou tableur.*
- *Lire et comprendre des fonctions Python simples.*
- *Écrire des fonctions renvoyant le résultat numérique d'une expérience aléatoire.*

1

Les trois fonctions proposées utilisent deux types d'instructions permettant de générer un nombre aléatoire :

- `randint(1,6)` renvoie un nombre aléatoire entier compris entre 1 et 6 ;
- `uniform(1,6)` renvoie un nombre aléatoire suivant la loi uniforme compris entre 1 et 6.

Dans le contexte étudié, c'est donc la première fonction qui est à choisir, la seconde simulant la somme d'un double.

- connaître les instructions randint et uniform

2

La variable `jeu` correspond au résultat de l'appel `d()`, c'est à dire un nombre correspondant à la somme de deux dés après un lancer.

L'instruction conditionnelle permet alors de tester si cette somme a pour valeur 7 ou 11. Si c'est le cas, alors la fonction `craps` renvoie la chaîne de caractères `"gagné"` et, dans le cas contraire, elle renvoie la chaîne de caractères `"perdu"`.

On peut donc considérer qu'une partie de craps est gagnée lorsque la somme de deux dés vaut 7 ou 11.

- connaître les opérateurs logiques

3

```
from random import randint

def d():
    return randint(1,6)+randint(1,6)

def craps():
    jeu = d()
    if jeu == 7 or jeu == 11:
        return "gagné"
    else:
        return "perdu"
```

La fonction `craps` ci-contre correspond à l'algorithme proposé.

Cette fonction faisant appel à la fonction `d`, il convient de saisir celle-ci en amont de la fonction `craps`.

- savoir retranscrire un algorithme écrit en langage naturel en une fonction Python
- écrire une instruction conditionnelle

Éléments de différenciation pédagogique

Les fonctions `d` et `craps` permettent de travailler en accord avec une pensée algorithmique construite autour d'un champ lexical lié à la situation.

Il serait intéressant de mener une réflexion sur d'autres pistes de construction.

Par exemple, en travaillant sur les opérateurs de comparaison, les deux précédentes fonctions (`d` et `craps`) auraient pu être remplacées par la fonction `craps` ci-contre :

```
from random import randint

def craps():
    jeu = randint(1,6)+randint(1,6)
    return jeu == 7 or jeu == 11
```

Cette alternative renvoie un booléen : `True` si la partie est gagnée et `False` sinon.

4 Une fois les deux fonctions saisies, on exécute le script à l'aide de la touche ▶

Les fonctions peuvent alors être appelées dans la console.

La fréquence est à calculer ensuite, une fois les 10 appels à la fonction `craps` réalisés. Une mise en commun de l'ensemble des résultats obtenus sera utile pour répondre à la consigne suivante.

- savoir calculer une fréquence

5 L'hétérogénéité des résultats montre que le nombre d'essais réalisés est insuffisant pour répondre à la problématique, la taille de l'échantillon n'étant pas assez importante.

6 Les deux parties manquantes sont indiquées sur la capture d'écran ci-dessous. On pourra détailler les étapes de la fonction `freq`.

```
def freq(n):
    c=0
    for i in range(1,n+1):
        if craps()=="gagné":
            c+=1
    return c/n
```

La variable `c` stocke le nombre de parties gagnées tout au long des `n` lancers. Pour chaque valeur `i` variant de 1 à `n`, (c'est à dire pour tout nombre entier naturel `i` tel que $1 \leq i < n+1$), on teste si le résultat de l'appel `craps()` a pour valeur `"gagné"`, auquel cas on incrémente la valeur de la variable `c`.

Une fois les `n` lancers simulés, la fonction renvoie la fréquence `c/n` de parties gagnées correspondantes.

- compléter une fonction Python
- utiliser une boucle bornée
- connaître les différents types de variables

7 Les résultats obtenus suite aux différents appels précédents permettent d'émettre une conjecture quant à l'évolution des fréquences de parties gagnées pour différentes valeurs de `n`, et ainsi estimer la chance de gagner au craps, en l'occurrence ici environ 0,22.

- écrire une instruction conditionnelle

8

```
In [2]: freq(10)
Out[2]: 0.1

In [3]: freq(100)
Out[3]: 0.21

In [4]: freq(1000)
Out[4]: 0.228

In [5]: freq(10000)
Out[5]: 0.2278

In [6]: freq(100000)
Out[6]: 0.22185
```

Ainsi, on pourra arriver à la conclusion que la fréquence de parties gagnées tend vers la probabilité de gagner au craps lorsque la taille de l'échantillon `n` augmente.

Éléments de différenciation pédagogique

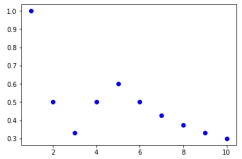
On propose la fonction `graph` du visuel ci-dessous qui est une variante de la fonction `freq` proposée au point 6. Celle-ci permet une visualisation plus concrète du phénomène présenté dans cette situation.

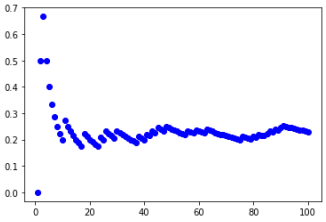
Les indications proposées dans le mémento en fin d'ouvrage permettent d'optimiser les paramètres graphiques et d'agrémenter le graphique renvoyé en fonction de la situation de départ.

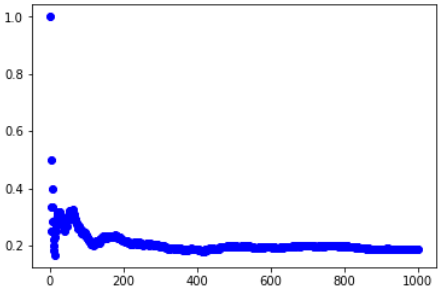
Remarque : le script n'utilise pas les listes ce qui rend son exécution plus longue.

```
import matplotlib.pyplot as plt

def graph(n):
    c=0
    for i in range(1,n+1):
        if craps()=="gagné":
            c+=1
        plt.plot(i,c/i,"bo")
```

Les captures d'écran ci-dessous sont obtenues respectivement suite aux appels `graph(10)`, `graph(100)` et `graph(1000)`.





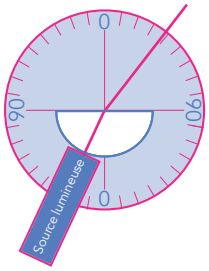


Il est alors possible de valider la conjecture émise aux points 7 et 8 en observant la stabilisation relative des fréquences vers une valeur proche de la probabilité étudiée lorsque la taille de l'échantillon `n` augmente. Lors d'une phase de mutualisation, un temps peut être accordé à la mise en commun des résultats et ainsi procéder à la rédaction de la trace écrite, notamment en ce qui concerne la vulgarisation de la loi des grands nombres.

Lois de Snell-Descartes

Situation

Durant une séance de travaux pratiques, on réalise le montage expérimental ci-contre, une cuve semi-circulaire contenant par exemple un liquide à étudier. On souhaite réaliser en Python un script permettant de simuler les lois de Snell-Descartes mises en œuvre dans le protocole précédent.



Comment réaliser ce script ?

Déroulement

L'ensemble des fonctions Python seront à saisir dans un script nommé SNELL.

La fonction Python `angle_i2` ci-contre renvoie la valeur de l'angle de réfraction dans le cadre de l'expérimentation citée dans la situation.

```
def angle_i2(i1,n1,n2):
    return degrees(asin(n1*sin(radians(i1))/n2))
```

1. Préciser la signification des arguments i_1, n_1 et n_2 utilisés dans la fonction `angle_i2`.

2. Après avoir saisi la fonction `angle_i2` dans l'éditeur, réaliser l'appel `angle_i2(60,1,1.5)` dans la console d'exécution. Interpréter le résultat obtenu.

3. Modifier la fonction `angle_i2` afin qu'elle renvoie une valeur arrondie à l'unité de l'angle de réfraction i_2.

On souhaite maintenant représenter graphiquement le trajet d'un rayon lumineux suivant les lois de la réflexion et de la réfraction de Snell-Descartes. On utilise pour cela la fonction `graphe` ci-contre :

```
import matplotlib.pyplot as plt
from math import pi

def graphe(i1,n1,n2):
    fig = plt.figure()
    ax = fig.add_axes([0,0,1,1],polar=True)
    ax.set_rticks([])
    ax.set_theta_zero_location("N")
    ax.margins(0)
    i1r = radians(i1)
    i2r = A

    ax.arrow(0,0,i1r,1,color='b',label='i1',
             head_width=0.0)
    B et C
    plt.legend()
```

4. Compléter l'élément A correspondant à la valeur en radian de l'angle de réfraction i_2.

5. Réaliser l'appel `graphe(60,1,1.5)` dans la console d'exécution et préciser la nature des éléments obtenus sur le graphique au regard de l'expérimentation.

Dans la fonction `graphe` ci-dessus, les instructions B et C manquantes visent à représenter graphiquement le rayon réfracté en rouge et le rayon réfléchi en vert.

6. En vous appuyant sur l'instruction `ax.arrow(0,0,i1r,1,color='b',label='i1')`, compléter les éléments B et C.

7. Valider vos résultats en réalisant quelques tests de la fonction `graphe` pour différents cas de figure.

- *Lire et comprendre des fonctions Python simples.*
- *Réaliser un calcul à l'aide d'une ou plusieurs variables.*
- *Modifier ou compléter un programme simple pour résoudre un problème.*
- *Comprendre et utiliser des fonctions.*
- *Structurer un programme en ayant recours à des fonctions pour résoudre un problème donné.*

***Remarque** : le script proposé est optimisé pour des versions de Python supérieures à 3.8. Pour des versions plus anciennes, il sera nécessaires de faire quelques ajustements. (Par exemple, l'instruction `ax.margins(0)` est à enlever).*

1 Les trois arguments de la fonction `angle_i2` correspondent à :

- la valeur en degré de l'angle d'incidence pour `i1` ;
- la valeur de l'indice de réfraction du premier milieu pour `n1` ;
- la valeur de l'indice de réfraction du second milieu pour `n2`.

2 L'appel `angle_i2(60,1,1.5)` dans la console d'exécution renvoie une erreur.

```
In [3]: angle_i2(60,1,1.5)
Traceback (most recent call last):

  File "<ipython-input-3-4f8ebffa0ffb>", line 1, in <module>
    angle_i2(60,1,1.5)

  File "/[illegible]/snell_LD4.py",
line 4, in angle_i2
    return degrees(asin(n1*sin(radians(i1))/n2))

NameError: name 'degrees' is not defined
```

Tout comme les instructions `asin`, `sin` et `radians` utilisées dans cette fonction, l'instruction `degrees` fait partie de la bibliothèque `math`. L'erreur renvoyée indique qu'il convient d'importer ces instructions (ou à défaut la bibliothèque `math` dans son intégralité) en amont de la fonction.

- comprendre une fonction Python
- savoir importer une bibliothèque

3 L'instruction `round` permet de renvoyer la valeur arrondie d'un nombre donné à une précision indiquée en argument. Au regard du point précédent, on modifie donc la fonction `angle_i2` :

```
from math import asin,sin,degrees,radians

def angle_i2(i1,n1,n2):
    return round(degrees(asin(n1*sin(radians(i1))/n2)))
```

- connaître l'instruction round

4 La valeur en degrés de l'angle de réfraction `i2` est obtenue suite à l'appel `angle_i2(i1,n1,n2)`. La variable `i2r` stocke la valeur courante de l'angle de réfraction `i2` exprimée en radians à l'aide de l'instruction :

```
i2r = radians(angle_i2(i1,n1,n2))
```

• connaître l'instruction radians

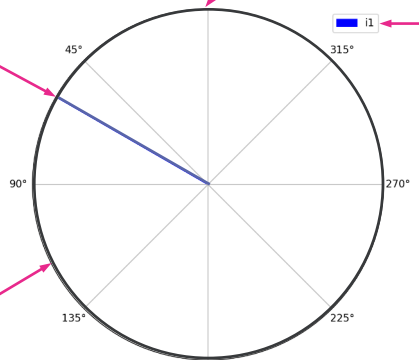
5 L'appel `graphe(i1,n1,n2)` permet de représenter graphiquement le trajet d'un rayon lumineux :

Définition de l'origine angulaire au Nord

```
ax.set_theta_zero_location("N")
```

Affichage du rayon incident

```
ax.arrow(0,0,i1r,1,color='b',label='i1')
```




```
plt.legend()
```

Affichage de la légende

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1],polar=True)
ax.set_rticks([])
```

Définition d'un repère en coordonnées polaires

6 L'instruction `ax.arrow(0,0,i1r,1,color='b',label='i1')` permet de tracer un segment [OA] bleu où O est l'origine du repère, A le point de coordonnées polaires (`i1r` ; 1) dans ce repère. Une étiquette notée « `i1` » est enregistrée.

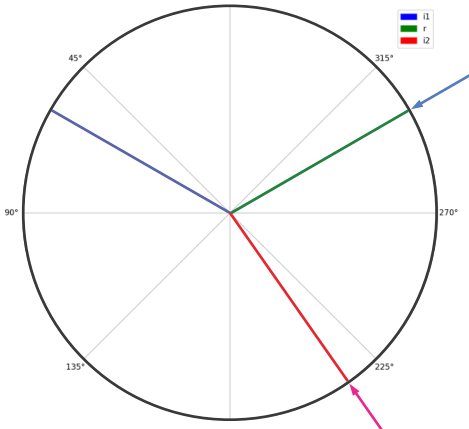
En suivant cette syntaxe, on déduit les instructions B et C qui représentent respectivement le rayon réfracté et le rayon réfléchi :

```
ax.arrow(0,0,-i1r,1,color='g',label='r')
ax.arrow(0,0,pi+i2r,1,color='r',label='i2')
```

• comprendre une fonction Python

7 Plusieurs appels sont réalisés dans la console d'exécution pour différents cas de figure (étude de plusieurs milieux de propagation, modification de l'angle d'incidence) Le schéma ci-dessous représente le cas étudié tout au long de l'activité à savoir un angle d'incidence de 60° traversant un dioptre Air/Verre.

B `ax.arrow(0,0,-i1r,1,color='g',label='r')`




```
graphe(60,1,1.5)
```

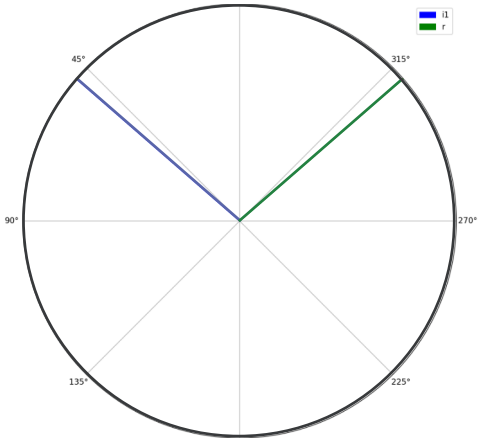
`ax.arrow(0,0,pi+i2r,1,color='r',label='i2')` C

Éléments de différenciation pédagogique

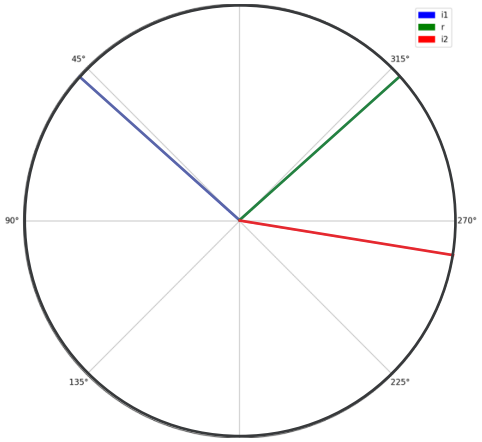
En prolongement de cette activité, il serait intéressant d'illustrer le phénomène de réflexion totale en ajoutant à la fonction **graphe** une instruction conditionnelle afin de contrôler l'affichage du rayon réfracté. En application de la loi de Snell-Descartes, ce dernier n'existe que dans le cas où le rapport entre n1*sin(i1) et n2 est inférieur ou égale à 1, d'où la condition utilisée ci-contre.

```
def graphe(i1,n1,n2):
    fig = plt.figure()
    ax = fig.add_axes([0,0,1,1],polar=True)
    ax.set_rticks([])
    ax.set_theta_zero_location("N")
    ax.margins(0)
    i1r = radians(i1)

    ax.arrow(0,0,i1r,1,color='b',label='i1',
             head_width=0.0)
    ax.arrow(0,0,-i1r,1,color='g',label='r')

    if n1*sin(radians(i1)) < = n2:
        i2r = radians(angle_i2(i1,n1,n2))
        ax.arrow(0,0,pi+i2r,1,color='r',
                 label='i2')
    plt.legend()
```

À titre d'exemple on peut étudier la situation d'un dioptre Eau/Air avec deux valeurs différentes pour l'angle d'incidence :




```
graphe(49,1.33,1)
```




```
graphe(48,1.33,1)
```

Masques de protection

Situation

Un analyste doit conseiller un autoentrepreneur qui souhaite produire et commercialiser des masques en tissu répondant à la norme AFNOR, en coton biologique. Le bénéfice de l'entreprise dépend de différents facteurs liés à la quantité de masques produits et vendus, auxquels doivent se rajouter des frais fixes (publicités, tissus, élastiques, …).
Il définit ainsi le bénéfice mensuel, noté B, exprimé en euros en fonction du nombre de masques x (fabriqués et vendus) modélisé par la fonction suivante :

$B(x) = -0{,}01x^2+5x-100$, définie pour tout nombre x appartenant à l'intervalle $[0\ ;\ 500]$

Que doit conseiller l'analyste à l'autoentrepreneur ?

Déroulement

L'ensemble des fonctions Python est à saisir dans un même script nommé `MASQUES`.

1. Reformuler la problématique en questionnement mathématique.

2. Ecrire en Python une fonction `benefice` prenant comme argument le nombre x et qui renvoie l'image du nombre x par la fonction B donnée dans l'énoncé.

On donne la fonction simplifiée `trace` ci-contre permettant de tracer des points de la représentation graphique d'une fonction sur un intervalle [a ; b].

```
import matplotlib.pyplot as plt

def trace(fonction,mini,maxi):
    plt.grid()
    x = [i for i in range(mini,maxi)]
    y = [fonction(j) for j in x]
    plt.plot(x,y)
```

3. Recopier ce script puis effectuer un appel dans la console Python afin de visualiser la représentation graphique de la fonction B sur [0 ; 500].

4. À l'aide de la représentation graphique obtenue, proposer un encadrement possible du nombre de masques à produire pour obtenir un bénéfice maximal.

On ajoute à la fonction `trace` précédente l'instruction suivante :

```
return (y.index(max(y)))
```

5. En utilisant le mémento disponible en fin d'ouvrage, expliquer ce qu'effectue l'instruction précédente.

6. Ajouter cette instruction à la fonction `trace` puis effectuer l'appel réalisé au point 3 afin de noter la valeur renvoyée.

7. Rédiger une note que l'analyste pourrait proposer à l'autoentrepreneur pour le conseiller.

- *Etude d'une fonction polynôme de degré 2*
- *Représentation graphique d'une fonction sur un intervalle donné*
- *Ajouter une instruction dans un script écrit en langage Python*
- *Détermination du maximum d'une fonction*

1 L'analyste devra prodiguer à l'autoentrepreneur des conseils sur la quantité de masques à produire afin de pouvoir dégager un bénéfice maximal. Mathématiquement, cela revient à déterminer le nombre de masques à produire pour engendrer un bénéfice maximal.

• traduire une problématique en questionnement mathématique

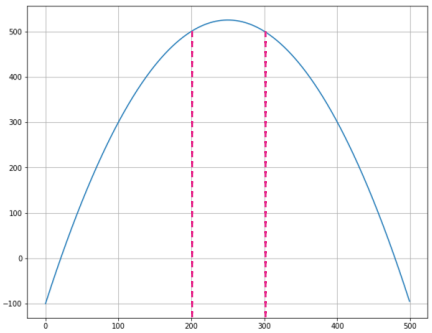
2 La fonction ci-contre permet de répondre à la question :

```
def benefice(x):
    assert 0 <= x <= 500
    return -0.01*x**2 + 5*x - 100
```

L'instruction `assert` permert de restreindre les valeurs de x au domaine de définition en renvoyant une erreur si celui-ci n'est pas respecté.

3 On effectue l'appel suivant `trace(benefice,0,500)` qui permet de visualiser la représentation graphique de la fonction `benefice` sur l'intervalle `[0 ; 500]`.

4 D'après le graphique obtenu, on peut proposer un encadrement approximatif compris entre 200 et 300 masques pour pouvoir espérer obtenir un bénéfice maximal.



• écrire une fonction Python

• connaître le vocabulaire lié aux fonctions

• utilisation d'un script écrit en langage Python

• interpréter une représentation graphique

5 L'instruction `return(y.index(max(y)))` permet de déterminer le rang de la valeur maximale dans la liste `y`, soit le maximum de la fonction `benefice` sur l'intervalle d'étude.

6 On obtient la fonction `trace` modifiée ci-contre.

Dans la console Python, l'appel `trace(benefice,0,500)`, renvoie la valeur 250.

```
def trace(fonction,mini,maxi):
    plt.figure(figsize=(10,8))
    plt.grid()
    x = [i for i in range(mini,maxi)]
    y = [fonction(j) for j in x]
    plt.plot(x,y)
    return (y.index(max(y)))
```

• connaître les méthodes liées aux listes

7 On peut proposer la note suivante :

« Pour réaliser un bénéfice maximal, votre entreprise de masques doit fabriquer et vendre 250 masques par mois, le bénéfice mensuel maximal ainsi réalisé sera de 525 €. Votre entreprise réalisera 2,10 € de bénéfice par masque vendus».

• interpréter un résultat afin de répondre à une problématique

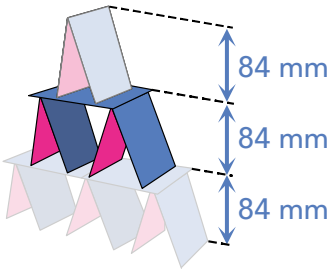
Remarque

Il est possible de faire un prolongement autour de la valeur maximale trouvée qui pourra être confrontée à la formule du sommet, élément caractéristique d'une fonction polynôme de degré 2.

Château de cartes

Situation

Lors d'un défi, un adolescent mesurant 1,85 m souhaite contruire un château de cartes dont la hauteur se rapproche le plus possible de sa taille. Il utilise le modèle ci-contre où le second niveau du château est mis en surbrillance. On considère que le premier niveau est celui qui se trouve en haut du château.



Combien de cartes devra-t-il poser à chaque niveau de son château ?

Combien lui faudra-t-il de jeux de 32 cartes pour cette réalisation ?

Déroulement

L'ensemble des fonctions Python est à saisir dans un même script nommé `CARTES`*.*
Tout au long de la situation, on appellera (U_n) *la suite numérique formée par les nombres successifs de cartes nécessaires à chaque niveau du château.*

1. Calculer le nombre maximum de niveaux constituant le château.

2. Conjecturer la nature de la suite (U_n).

3. Préciser l'objectif mathématique de l'algorithme ci-contre et en donner une interprétation relative à la situation.

```
Fonction sa(U1,r,n) :
    Un ← U1
    Pour i allant de 1 à n-1
        Un ← Un + r
    Fin Pour
    Renvoyer Un
Fin
```

4. Écrire en Python la fonction `sa` traduisant l'algorithme ci-dessus.

5. En vous aidant de la fonction `sa` et des points précédents, déterminer le nombre de cartes constituant le niveau de base du château.

On souhaite créer une liste des nombres de cartes nécessaires à chaque niveau de ce château.

6. Écrire en Python une fonction `liste_cartes` utilisant la fonction `sa` précédente et qui renvoie une liste des nombres de cartes constituant les niveaux du château.

7. Préciser l'appel à réaliser dans la console d'exécution afin de répondre à la première problématique.

8. Expliquer mathématiquement le contenu de la liste renvoyée par cet appel.

9. Écrire en Python une fonction `nb_jeu`, faisant appel à la fonction `liste_cartes`, permettant de répondre à la seconde problématique.

- *Générer à l'aide d'un outil numérique les termes d'une suite.*
- *Calculer un terme de rang donné d'une suite arithmétique définie par son premier terme et par relation de récurrence.*
- *Concevoir un programme simple pour résoudre un problème.*
- *Comprendre et utiliser des fonctions Python.*
- *Comprendre et utiliser des méthodes sur les listes en langage Python.*

1 On souhaite ici réaliser un château dont la hauteur maximale est de 1,85 m soit 1 850 mm. Le schéma fourni dans l'énoncé indique qu'au regard de la taille des cartes utilisées, la hauteur d'un niveau est de 84 mm. Le nombre de niveaux correspond donc au quotient de la division euclidienne de ces deux nombres, c'est-à-dire 22 :

```
In [1]: 1850//84
Out[1]: 22
```

2 Le schéma fourni indique que le premier niveau est constitué de 2 cartes. Les second et troisième niveaux sont quant à eux constitués respectivement de 5 et 8 cartes. On peut donc conjecturer que (U_n) est une suite arithmétique de premier terme $U_1 = 2$ et de raison r = 3.

3 L'algorithme de la fonction `sa` a pour objectif de calculer le terme de rang n d'une suite arithmétique de premier terme `U1` et de raison `r` par une relation de récurrence :

```
Un ← U1
```

La terme de rang n (variable `Un`) est d'abord initialisé à la valeur du premier terme `U1`.

```
Pour i allant de 1 à n-1
    Un ← Un + r
Fin Pour
```

On ajoute (n-1) fois la valeur de la raison r à la valeur de la variable `Un`.

```
Renvoyer Un
```

On sort de la boucle `for`, puis on renvoie la valeur du terme de rang n.

Au regard de la situation, la fonction `sa` permet de calculer le nombre de cartes constituant le n-ième niveau du château.

• comprendre un algorithme

4

```
def sa(U1,r,n):
    Un = U1
    for _ in range(1,n):
        Un += r
    return Un
```

La fonction `sa` suivante est une traduction de l'algorihtme proposé au point précédent.

- traduire un algorithme à l'aide d'une fonction en Python.

La syntaxe de la boucle `for` nécessite d'utiliser un itérable (on utilise généralement `i`). Dans l'exemple de la fonction `sa`, la valeur de l'itérable n'est pas utilisée dans la boucle, c'est pourquoi il est possible d'utiliser «`_`» plutôt que «`i`» pour l'itérable.

5

Le point 1 précise que le château doit compter 22 niveaux. L'appel `sa(2,3,22)` permet donc de conclure que le niveau de base du château compte 65 cartes.

```
In [2]: sa(2,3,22)
Out[2]: 65
```

6

L'objectif de la fonction `liste_cartes` est de renvoyer une liste contenant le nombre de cartes nécessaires de chacun des niveaux du château. D'après le point 3, la fonction `sa` renvoie le nombre de cartes du n-ième niveau d'où la fonction suivante :

```
def liste_cartes(U1,r,n):
    L = [U1]
    for i in range(2,n+1):
        L.append(sa(U1,r,i))
    return L
```

On crée une liste `L` dans laquelle on insère le nombre `U1` de cartes du premier niveau.

Pour `i` allant de 2 à n ...

On ajoute à la liste `L` la valeur du nombre de cartes du niveau `i`.

La fonction renvoie alors la liste `L` contenant les nombres de cartes constituant les niveaux du château.

```
def liste_cartes(U1,r,n):
    L = [U1]
    for i in range(2,n+1):
        L.append(sa(U1,r,i))
    return L
```

- concevoir un programme simple pour résoudre un problème
- comprendre et utiliser des fonctions Python.
- générer et manipuler une liste

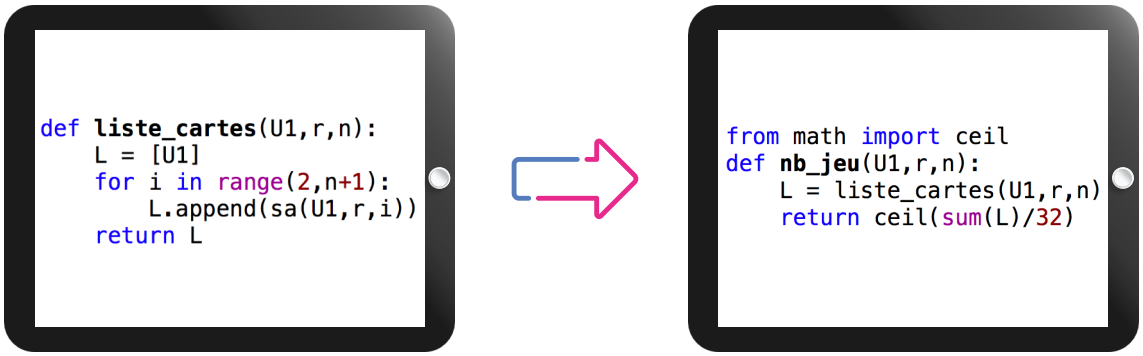
7 Le point 1 précise que le château de cartes compte 22 niveaux d'où l'appel suivant :

```
In [2]: liste_cartes(2,3,22)
Out[2]:
[2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32,
35, 38, 41, 44, 47, 50, 53, 56, 59, 62, 65]
```

8 La liste obtenue suite à l'appel `liste_cartes(2,3,22)` regroupe les 22 premiers termes d'une suite arithmétique de premier terme $U_1 = 2$ et de raison $r = 3$.

9 Pour répondre à la seconde problématique, la fonction `nb_jeu` doit pouvoir renvoyer le nombre de jeux de cartes nécessaires à la réalisation. Pour cela, on utilise deux instructions :

- l'instruction `sum(L)` qui permet de calculer la somme des termes exclusivement numériques constituant la liste `L`.
- l'instruction `ceil` qui a pour objectif de renvoyer la valeur plafond d'un nombre donné (c'est-à-dire le plus petit nombre entier supérieur ou égal de ce nombre). Cette instruction faisant partie de la bibliothèque `math`, il faut l'importer en amont de la fonction `nb_jeu` :



Au regard des éléments développés tout au long de l'activité, on réalise l'appel `nb_jeu(2,3,22)` dans la console d'exécution ce qui permet de conclure que 24 jeux de 32 cartes sont nécessaires à la construction du château.

```
In [14]: nb_jeu(2,3,22)
Out[14]: 24
```

- comprendre et utiliser une fonction Python
- connaître l'instruction ceil
- manipuler les listes

Émission de SO_2

Situation

Les entreprises dont l'activité est considérée comme polluante sont soumises à une taxe appelée TGAP (Taxe Générale sur les Activités Polluantes).
Une entreprise a pour objectif de réduire de moitié ses émissions de dioxyde de soufre (actuellement correspondant à une masse de 2 tonnes par an dans l'air). Pour cela, elle adopte une série de mesures visant à diminuer ses émissions de SO_2 de 12 % chaque année.

Combien d'années seront nécessaires à l'entreprise pour atteindre son objectif ?

Déroulement

L'ensemble des fonctions Python est à saisir dans un même script nommé ***S02****.*
Tout au long de la situation, on appellera (U_n) la suite numérique formée par les masses annuelles de SO_2 rejetées dans l'air par l'entreprise.

1. Reformuler la problématique afin de faire apparaître les attentes mathématiques.

2. Conjecturer la nature de cette suite puis donner son premier terme U_1 et sa raison q.

3. Écrire en Python une fonction `sg` prenant comme arguments le premier terme `U1`, la raison `q` et le nombre entier naturel `n` correspondant au nombre d'années écoulées. Cette fonction renvoie la valeur du terme de rang n de la suite *(U_n)* calculé à partir des termes précédents.

4. Réaliser différents appels dans la console Python afin de tester la fonction `sg`.

5. Expliquer le rôle de la fonction `GEN` ci-contre, où `N` correspond à un nombre d'années, en donnant les principales étapes mobilisées dans celle-ci.

```
import matplotlib.pyplot as plt

def GEN(U1,q,N):
    L = []
    for n in range(1,N+1):
        Y = sg(U1,q,n)
        L.append(Y)
        plt.plot(n,Y,'b+')
    plt.grid()
    return L
```

6. Préciser l'appel à réaliser dans la console d'exécution permettant de générer et représenter graphiquement les 10 premiers termes de la suite *(U_n)*.

7. Recopier la fonction `GEN` puis effectuer l'appel précisé au point précédent dans la console d'exécution.

8. À partir des points précédents, répondre à la problématique.

- *Générer à l'aide d'un outil numérique les termes d'une suite.*
- *Calculer un terme de rang donné d'une suite géométrique définie par son premier terme et par relation de récurrence.*
- *Réaliser et exploiter une représentation graphique du nuage de points (n ; U_n) dans le cas d'une suite géométrique.*
- *Concevoir un programme simple pour résoudre un problème.*
- *Comprendre et utiliser des fonctions Python.*
- *Comprendre et utiliser des méthodes sur les listes en langage Python.*

1 La problématique proposée vise à déterminer le nombre d'années nécessaires à l'entreprise afin de réduire de moitié ses émissions de SO_2 . On souhaite donc déterminer le plus petit nombre entier naturel *n* tel que $U_n \leq 1$.

2 La diminution annuelle envisagée par l'entreprise s'élève à 12 % des émissions de l'année précédente. On peut donc conjecturer que (U_n) est une suite géométrique de raison q = 0,88 et de premier terme $U_1 = 2$.

3 Pour traiter cette consigne, il est possible de créer une fonction Python renvoyant directement le terme de rang n à l'aide de la formule du terme général. La consigne nous oblige ici à ce que cette fonction calcule ce terme à l'aide d'une relation de récurrence :

```
def sg(U1,q,n):                  Fonction sg(U1,q,n):
    Un = U1                          Un ← U1
    for i in range(1,n):             Pour i allant de 1 à n-1
        Un *= q                          Un ← Un x q
                                     Fin Pour
    return Un                        Renvoyer Un
```

• structurer un programme en ayant recours à des fonctions pour résoudre un problème donné

4 Les trois appels suivants présentent différents exemples de calcul d'un terme de rang n pour plusieurs suites géométriques :

```
In [2]: sg(7,5,6)
Out[2]: 21875
```

Calcul du 6e terme d'une suite géométrique de premier terme $U_1 = 7$ et de raison $q = 5$.

```
In [3]: sg(2.5,3,4)
Out[3]: 67.5
```

Calcul du 4e terme d'une suite géométrique de premier terme $U_1 = 2,5$ et de raison $q = 3$.

```
In [4]: sg(4000,0.5,9)
Out[4]: 15.625
```

Calcul du 9e terme d'une suite géométrique de premier terme $U_1 = 4\,000$ et de raison $q = 0,5$.

5 La fonction `GEN` prend comme arguments le premier terme `U1` d'une suite géométrique, sa raison `q` et `N`, un nombre de termes à déterminer. Cette fonction renvoie une liste des `N` premiers termes de la suite géométrique étudiée et en affiche une représentation graphique sous la forme d'un nuage de points de coordonnées (n ; U_n).

• comprendre une fonction en Python.

Dans un premier temps, on importe la bibliothèque `matplotlib` à l'aide de l'instruction :

```
import matplotlib.pyplot as plt
```

```
def GEN(U1,q,N):
    L = []
```
On crée une liste vide que l'on nomme L.

```
    for n in range(1,N+1):
```
Pour n allant de 1 à N ...

```
        Y = sg(U1,q,n)
```
On affecte à la variable Y le terme de rang n de la suite géométrique étudiée ;

```
        L.append(Y)
```
On ajoute la valeur de la variable Y à la liste L ;

```
        plt.plot(n,Y,'b+')
```
On affiche dans un repère le point de coordonnées (n ; Y) à l'aide d'une croix bleue.

```
    plt.grid()
```
On affiche une grille dans le repère étudié.

```
    return L
```
La fonction renvoie alors la liste L correspondant à l'ensemble de N premiers termes de la suite (U_n).

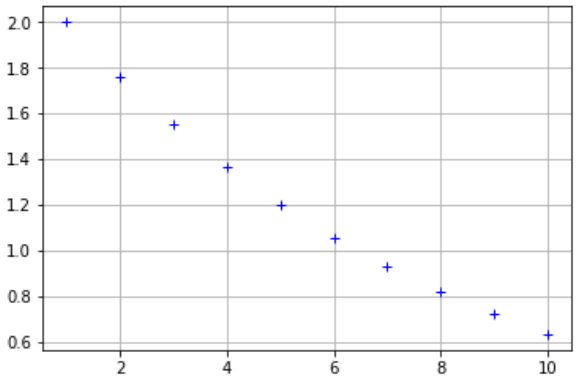
6 La suite (U_n) est une suite géométrique de premier terme $U_1 = 2$ et de raison $q = 0{,}88$. Il convient donc de réaliser l'appel suivant dans la console d'exécution :

```
GEN(2,0.88,10)
```

• comprendre une fonction en Python.

7 L'appel `GEN(2,0.88,10)` permet de renvoyer une liste contenant la valeur (en tonne) des 10 premiers termes de la suite (U_n), et d'en afficher la représentation graphique sous la forme d'un nuage de points (n ; U_n).

```
In [2]: GEN(2,0.88,10)
Out[2]:
[2,
 1.76,
 1.5488,
 1.362944,
 1.19939072,
 1.0554638336,
 0.9288081735679999,
 0.8173511927398399,
 0.7192690496110591,
 0.632956763657732]
```



8 Au regard de la liste et du nuage de points précédents, on remarque que le 7e terme de la suite (U_n) a une valeur arrondie au centième de 0,93 tonne.

Cette valeur étant la première valeur inférieure à la moitié de la masse initiale de dioxyde de soufre émise, soit une tonne, il faudra donc 6 années à l'entreprise pour atteindre son objectif.

Algorithme glouton

Situation

Il est possible d'acheter, avec de l'argent liquide, différents produits dans des distributeurs automatiques. Or, lorsque l'on donne une somme supérieure à la valeur du produit, le distributeur nous rend la monnaie avec une répartition des billets et des pièces bien définie.

Comment rendre la monnaie de manière optimale ?

Déroulement

L'ensemble des fonctions Python est à saisir dans un même script nommé `GLOUTON`.
On utilisera la liste : `monnaie=[20,10,5,2,1,0.50,0.20,0.10,0.05,0.02,0.01]`, représentant les montants d'un système monétaire de manière décroissante de 20 € à 1 centime.

1. Expliciter brièvement le terme « optimale » utilisé dans la problématique en s'appuyant sur un exemple de rendu d'un montant de 56 €.

2. Dans la fonction `rendre` présentée ci-contre, expliquer l'utilisation des commandes suivantes :
   - la méthode `.append()` ;
   - le rôle des opérateurs `//` et `%`
   - Le rôle de `int(compteur)` .

```
monnaie=[20,10,5,2,1,0.50,0.20,0.10,0.05,0.02,0.01]

def rendre(montant):
    repartition=[]
    for argent in monnaie:
        compteur = round(montant,2) // argent
        montant = montant % argent
        repartition.append((int(compteur),argent))
    return repartition
```

3. Effectuer les appels suivants : `rendre(56)` et `rendre(14.03)` dans la console Python puis commenter les résultats obtenus.

4. En utilisant les points précédents, décrire les différentes étapes de la fonction `rendre`. On pourra les décrire à l'aide d'un algorithme écrit en pseudo-code.

On souhaite simplifier l'affichage du résultat renvoyé par la fonction `rendre` selon l'exemple ci-contre.

```
In [4]: rendre(153)
Out[4]:
[(7, 20),
 (1, 10),
 (0, 5),
 (1, 2),
 (1, 1),
 (0, 0.5),
 (0, 0.2),
 (0, 0.1),
 (0, 0.05),
 (0, 0.02),
 (0, 0.01)]
```

5. Préciser en quoi l'affichage est simplifié.

6. Modifier la fonction `rendre` en ajoutant une instruction conditionnelle afin d'obtenir un affichage simplifié selon l'exemple ci-contre.

7. Valider la fonction modifiée en effectuant à nouveau les appels suivants : `rendre(56)` et `rendre(14.03)` dans la console Python.

```
In [6]: rendre(153)
Out[6]: [(7, 20), (1, 10), (1, 2), (1, 1)]
```

Cette activité s'inscrit dans le cadre d'une séance de remédiation en 1ère bac professionnelle et fait suite à l'introduction des listes effectuée en classe.

- *Rappel des opérateurs // et %*
- *Modifier ou compléter un algorithme ou un programme.*
- *Comprendre et utiliser des méthodes sur les listes en langage Python.*

1

Le terme « optimale » utilisé dans la problématique fait référence au rendu monnaie qui, pour être considéré ainsi, doit être composé d'un nombre minimal de pièces ou de billets.

Le rendu monnaie « optimal » pour un montant de 56 € est composé de :

2 billets de 20 €, 1 billet de 10 €, 1 billet de 5 € et 1 pièce de 1 €.

2

- La méthode `.append()` utilisée ici permet d'ajouter à la liste `repartition` le tuple `(int(compteur),argent)` correspondant au nombre de pièces (ou de billets) nécessaires pour le rendu de la monnaie.
- L'opérateur `//` permet d'obtenir le quotient de la division du montant par argent lors de chaque itération et l'opérateur `%` permet d'obtenir le reste de la division du montant par l'argent lors de chaque itération.
- Le rôle de `int(compteur)` permet d'obtenir un nombre entier pour le résultat donnant le nombre de billets ou de pièces nécessaires au rendu optimal de la monnaie.

- connaître les méthodes sur les listes
- connaître les opérateurs // et %
- connaître le typage en Python

3

Une fois la fonction `rendre` saisie, on exécute le script `GLOUTON` à l'aide de la touche ▶. Dans la console Python, on effectue les appels :

`rendre(56)`

```
In [2]: rendre(56)
Out[2]:
[(2, 20),
 (1, 10),
 (1, 5),
 (0, 2),
 (1, 1),
 (0, 0.5),
 (0, 0.2),
 (0, 0.1),
 (0, 0.05),
 (0, 0.02),
 (0, 0.01)]
```

`rendre(14.03)`

```
In [3]: rendre(14.03)
Out[3]:
[(0, 20),
 (1, 10),
 (0, 5),
 (2, 2),
 (0, 1),
 (0, 0.5),
 (0, 0.2),
 (0, 0.1),
 (0, 0.05),
 (1, 0.02),
 (1, 0.01)]
```

- interpréter un script Python

Par conséquent, le rendu de monnaie pour 56 € sera composé de 2 billets de 20 €, d'1 billet de 10 €, d'1 billet de 5 €, et d'1 pièce de 1 € ce qui est en accord avec le point 1.

Le rendu de monnaie pour 14,03 € sera composé d'1 billet de 10 €, de 2 pièces de 2 €, d'1 pièce de 2 centimes et d'une pièce de 1 centime.

4 L'algorithme écrit en pseudo-code suivant illustre la fonction `rendre` :

```
Fonction rendre( montant ) :
    repartition ← on crée une liste vide
    Pour argent dans monnaie
          Compteur ←  quotient de la division de
                      montant par argent
          Montant ← reste de la division de montant par argent
          repartition  ← on ajoute le couple (compte,argent)
                         à la liste répartition
    Renvoyer repartition
Fin
```

- comprendre une fonction en Python.
- traduire les étapes d'un script à l'aide d'un algorithme.

5 L'affichage est simplifié car il permet de faire apparaître uniquement la monnaie utilisée pour réaliser le rendu. Les billets ou pièces qui n'interviennent pas dans le rendu de la monnaie n'apparaissent plus dans le résultat.

6 On propose la modification suivante :

```
def rendre(montant):
    repartition=[]
    for argent in monnaie:
        compteur = round(montant,2) // argent
        if compteur != 0:
            montant = montant % argent
            repartition.append((int(compteur),argent))
    return repartition
```

- compléter un programme à l'aide d'une instruction conditionnelle.

7 Une fois la fonction `rendre` saisie, on exécute le script `GLOUTON` à l'aide de la touche ▶ . Dans la console Python, on effectue les appels :

`rendre(56)`

```
In [5]: rendre(56)
Out[5]: [(2, 20), (1, 10), (1, 5), (1, 1)]
```

`rendre(14.03)`

```
In [6]: rendre(14.03)
Out[6]: [(1, 10), (2, 2), (1, 0.02), (1, 0.01)]
```

- valider un script Python.

Remarque

Il est possible de poursuivre le questionnement autour de la réflexion sur le terme « optimal ». Par exemple, réfléchir sur le fait qu'il n'y ait pas de pièce de 7 € dans ce système.

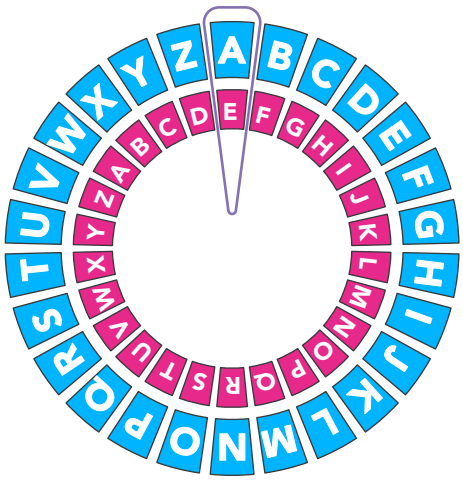
Évidemment, il ne s'agit pas de faire l'étude des algorithmes gloutons dont le principe consiste à choisir des solutions locales optimales d'un problème dans le but (ou l'espoir !) d'obtenir une solution optimale globale à celui-ci.

Analyse fréquentielle

Situation

En cryptographie, le code de César est une méthode de chiffrement très simple utilisée par Jules César lui même dans ses correspondances secrètes (ce qui explique son nom !).
Le texte chiffré s'obtient en remplaçant chaque lettre du texte clair original par une lettre à distance fixe, toujours du même côté, dans l'ordre de l'alphabet. Pour les dernières lettres (dans le cas d'un décalage à droite), on reprend au début. Ce décalage constitue la clef de chiffrement.

Comment décrypter un texte chiffré avec le code de César sans connaître la clef de chiffrement ?



Exemple avec un décalage de 4

Déroulement

L'ensemble des fonctions Python est à saisir dans un même script nommé `CESAR`.
Pour l'écriture de ce script, nous utiliserons une variable globale, notée `alpha`, déclarée à l'extérieur de toute fonction, pouvant être utilisée n'importe où dans ledit script.

```
alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

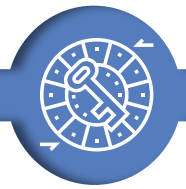
Étude préliminaire

1. À partir du mémento mis à disposition à la fin du livret, dans la fonction `code` ci-contre, expliquer l'utilisation de :
   a. la méthode `.upper()`
   b. la méthode `.find()`
   c. le calcul `(ordre+clef) % 26`
   c. l'instruction `alpha[indice]`

```
def code(texte,clef):
    res = ''
    for lettre in texte.upper():
        if lettre==' ':
            res += ' '
        else:
            ordre = alpha.find(lettre)
            indice = (ordre+clef) % 26
            res +=alpha[indice]
    return res
```

2. En utilisant la fonction `code` précédente, proposer une fonction `decode` ayant pour arguments `texte`, correspondant au message à déchiffrer, et `clef` pour la clé de déchiffrement.

3. Écrire les deux fonctions précédentes et réaliser les appels suivants dans la console Python :
   a. pour coder : `code('ALGORITHME',12)`
   b. pour décoder : `decode("JOPTPL",7)`



Analyse fréquentielle

L'analyse fréquentielle est basée sur le fait que, dans chaque langue, certaines lettres ou combinaisons de lettres apparaissent avec une certaine fréquence.
Par exemple, en français, on donne la répartition suivante :

Lettre	A	B	C	D	E	F	G	H	I	J	K	L
%	7.68	0.80	3.32	3.60	17.76	1.06	1.10	0.64	7.23	0.19	0.00	5.89

M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
2.72	7.61	5.34	3.24	1.34	6.81	8.23	7.30	6.05	1.27	0.00	0.54	0.21	0.07

On souhaite déchiffrer un texte qui a été crypté avec le code de César dont on ne connait pas la clef (disponible via le QR Code ci-contre).
À noter que le texte d'origine a été « nettoyé », c'est-à-dire qu'on a enlevé les majuscules, les symboles de ponctuation, les accents ou les caractères spéciaux.

1. Proposer un algorithme qui permet de calculer la fréquence d'une lettre choisie dans un texte.

2. Écrire en Python la fonction `Flettre` correspondant à l'algorithme présenté au point 1. Exécuter la commande `Flettre('A','TATA')` dans la console Python.

3. Donner le nom de la lettre la plus utilisée dans la langue française.

4. Expliquer le rôle de la fonction `AF` ci-contre en donnant les principales étapes mobilisées dans celle-ci.

```
def AF(crypte):
    f = []
    for lettre in alpha:
        f.append(Flettre(lettre,crypte))
        p = f.index(max(f))
    return decode(crypte,(p-4) % 26)
```

5. Après avoir téléchargé le fichier texte.txt, créer une variable `crypte` dont la valeur correspond au texte codé contenu dans ce fichier.

6. Exécuter le script `CESAR` et réaliser l'appel `AF(crypte)`. Commenter le résultat obtenu au regard du point 3.

- *Déterminer la fréquence d'apparition d'une lettre dans un texte.*
- *Décomposer un problème en sous-problèmes.*
- *Reconnaître le type d'une variable.*
- *Comprendre et utiliser des fonctions Python.*

Étude préliminaire

1 La fonction `code` permet de crypter un texte à l'aide d'une clef de cryptage, tous deux mis en arguments.

a. La méthode `.upper()`, utilisée ici dans l'instruction `texte.upper()` permet de passer le texte `texte` en majuscules.

b. La méthode `.find()`, utilisée ici dans l'instruction `ordre=alpha.find(lettre)` permet d'affecter à la variable `ordre` le rang de la lettre `lettre` dans la chaîne de caractères `alpha`. Autrement dit, la variable `alpha` prendra donc successivement comme valeur le rang alphabétique des différentes lettres constituant le texte à crypter.

c. Le calcul `(ordre+clef) % 26` renvoie le rang alphabétique de la lettre `lettre` une fois qu'elle aura été codée avec la clef `clef` mise en argument de la fonction `code`.

d. L'instruction `alpha[indice]` renvoie la lettre courante codée à l'aide de la clef de cryptage `clef` mise en argument.

- connaître les méthodes liées à l'utilisation des chaînes de caractères
- connaître les opérateurs arithmétiques

Éléments de différenciation pédagogique

La fonction `code` a pour objectif de crypter un texte à l'aide d'une clef dont le rang alphabétique est passé en argument.

L'instruction conditionnelle permet ici de détecter dans le texte à étudier les espaces et ainsi les inclure au texte crypté. Il est donc possible de simplifier la fonction en ne tenant pas compte des espaces dans l'étude, c'est-à-dire en supprimant l'instruction conditionnelle.

```
def code(texte,clef):
    res = ''
    for lettre in texte.upper():
        if lettre==' ':
            res += ' '
        else:
            ordre = alpha.find(lettre)
            indice = (ordre+clef) % 26
            res +=alpha[indice]
    return res
```

Message à coder	C	E	S	A	R
Variable `ordre`	2	4	18	0	17
Clef de cryptage	K				
Variable `clef`	10				
`ordre + clef`	12	14	28	10	27
`(ordre+clef)%26`	12	14	2	10	1
`alpha[indice]`	M	O	C	K	B

Exemple d'utilisation

```
In [1]:

In [2]: code('cesar',10)
Out[2]: 'MOCKB'

In [3]:
```

2 Au regard des différents éléments vus au point précédent, le principe de décodage est réciproque à celui de codage.

```
def decode(texte,clef):
    res = ''
    for lettre in texte.upper():
        if lettre==' ':
            res += ' '
        else:
            ordre = alpha.find(lettre)
            indice = (ordre-clef) % 26
            res +=alpha[indice]
    return res
```

Les fonctions `code` et `decode` sont donc similaires si ce n'est le sens de décalage des lettres qui se trouve inversé.

- comprendre une fonction Python
- modifier un programme

3 Une fois les deux fonctions `code` et `decode` saisies, on exécute le script `CESAR` à l'aide de la touche ▶ :

```
In [2]: code('ALGORITHME',12)
Out[2]: 'MXSADUFTYQ'
```

```
In [3]: decode("JOPTPL",7)
Out[3]: 'CHIMIE'
```

`MXSADUFTYQ` correspond au mot `ALGORITHME` codé à l'aide de la lettre `M`.

Le mot codé `JOPTPL` correspond au mot `CHIMIE` si on le décode à l'aide de la lettre `H`.

Analyse fréquentielle

1 Calculer la fréquence d'apparition d'une lettre `lettre` dans un texte `txt` revient à déterminer le quotient des valeurs des variables :

- `cpt` qui renvoie le nombre d'occurrences de la lettre `lettre` étudiée dans le texte nommé `txt` ;
- `long` qui renvoie le nombre de caractères contenus dans la variable `txt`.

- traduire le calcul d'une fréquence à l'aide d'une fonction Python

L'algorithme suivant illustre donc la fonction `Flettre` qui renvoie la fréquence d'apparition de la lettre `lettre` dans le texte `txt` :

```
Fonction Flettre( lettre, txt ) :
    long ← longueur du texte txt
    cpt ← nombre d'occurrences de
          lettre dans le texte txt
    Renvoyer cpt/long
```

• concevoir un algorithme simple pour résoudre un problème

2 La fonction `Flettre` suivante est une traduction de l'algorithme proposé au point précédent.

• traduire un algorithme à l'aide d'une fonction Python

```
def Flettre(lettre,txt):
    long = len(txt)
    cpt = txt.count(lettre)
    return cpt/long
```

```
In [2]: Flettre('A','TATA')
Out[2]: 0.5
```

3 Le tableau fourni dans le contexte nous indique que c'est la lettre E qui correspond à la fréquence d'apparition la plus élevée pour un texte écrit en Français.

4 La fonction AF prend comme argument la variable `crypte` correspondant à un texte crypté.

```
def AF(crypte):
    f = []
```
On créé une liste vide que l'on nomme `f`.

```
    for lettre in alpha:
```
Pour chaque `lettre` appartenant à la chaîne de caractères `alpha`...

```
        f.append(Flettre(lettre,crypte))
```
On ajoute à la liste `f` la fréquence d'apparition de la lettre étudiée dans le texte `crypte`.

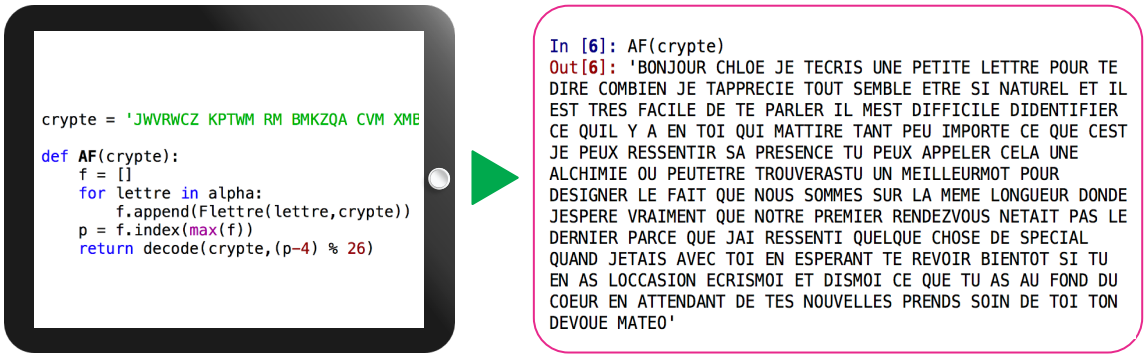
```
    p = f.index(max(f))
```
On affecte à la variable `p` le rang alphabétique de la lettre correspondant à la fréquence d'apparition la plus élevée.

```
    return decode(crypte,(p-4) % 26)
```
La fonction renvoie alors le texte décodé.

5 La variable `crypte` est une chaîne de caractères que l'on place dans le script `CESAR`.

```
crypte = 'JWVRWCZ KPTWM RM BMKZQA CVM XMBQBM TMBBZM XWCZ BM ...
```

6 L'appel `AF(crypte)` permet de décoder un texte crypté sans connaître la clef utilisée à l'aide d'une analyse fréquentielle.



```
crypte = 'JWVRWCZ KPTWM RM BMKZQA CVM XME

def AF(crypte):
    f = []
    for lettre in alpha:
        f.append(Flettre(lettre,crypte))
    p = f.index(max(f))
    return decode(crypte,(p-4) % 26)
```

```
In [6]: AF(crypte)
Out[6]: 'BONJOUR CHLOE JE TECRIS UNE PETITE LETTRE POUR TE
DIRE COMBIEN JE TAPPRECIE TOUT SEMBLE ETRE SI NATUREL ET IL
EST TRES FACILE DE TE PARLER IL MEST DIFFICILE DIDENTIFIER
CE QUIL Y A EN TOI QUI MATTIRE TANT PEU IMPORTE CE QUE CEST
JE PEUX RESSENTIR SA PRESENCE TU PEUX APPELER CELA UNE
ALCHIMIE OU PEUTETRE TROUVERASTU UN MEILLEURMOT POUR
DESIGNER LE FAIT QUE NOUS SOMMES SUR LA MEME LONGUEUR DONDE
JESPERE VRAIMENT QUE NOTRE PREMIER RENDEZVOUS NETAIT PAS LE
DERNIER PARCE QUE JAI RESSENTI QUELQUE CHOSE DE SPECIAL
QUAND JETAIS AVEC TOI EN ESPERANT TE REVOIR BIENTOT SI TU
EN AS LOCCASION ECRISMOI ET DISMOI CE QUE TU AS AU FOND DU
COEUR EN ATTENDANT DE TES NOUVELLES PRENDS SOIN DE TOI TON
DEVOUE MATEO'
```

La chaîne de caractères renvoyée par la fonction `AF` dans cet exemple montre que l'analyse fréquentielle proposée est efficace dès lors que le texte contient assez de caractères pour que la répartition des fréquences par lettre soit pertinente.

En considérant les éléments de réponse donnés au point 3 et ci-dessus, on peut émettre la conjecture que le rang `p` de la lettre dont la fréquence d'apparition est la plus élevée dans le texte crypté correspond à celui de la lettre E, c'est-à-dire 4. Ceci explique que l'on utilise l'instruction `(p-4) % 26` pour définir la clef utilisée ici dans le décodage du texte.

Remarque

Outre la nécessité d'avoir un échantillon suffisamment important de caractères, la fréquence d'apparition des lettres relève de données statistiques qui peuvent varier en fonction de nombreux paramètres (type du document, style, vocabulaire...). À noter également qu'il existe des cas particuliers. Par exemple, le roman « La disparition » de Georges Perec ne contient pas la lettre « e » malgré ses 300 pages !

Jeu du franc carreau

Situation

Lors d'une partie de franc carreau, un joueur lance une pièce sur un quadrillage. Si la pièce ne touche qu'un seul carreau, c'est-à dire sans toucher aucune ligne du quadrillage, la partie est gagnée. En revanche, si le centre de la pièce tombe à l'extérieur du quadrillage, le joueur la relance à nouveau.

Dans le cadre de cette activité, on se limitera à l'étude d'un carreau de 1 cm de côté et une pièce de 0,2 cm de rayon représentés sur le schéma ci-dessous.

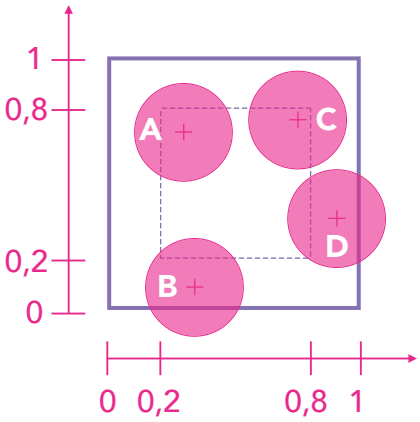
Quelles sont les chances de gagner au franc carreau ?

Déroulement

L'ensemble des fonctions Python est à saisir dans un même script nommé `FC`.

Partie N°1 : étude préliminaire

1 Le schéma ci-contre représente 4 lancers de pièces nommées A, B, C et D dans un carreau. Indiquer en justifiant la réponse si ces lancers représentent une partie gagnée.



On souhaite écrire en Python une fonction `valide` prenant comme arguments les coordonnées `x` et `y` du centre de la pièce et qui renvoie `True` si la partie est gagnée et `False` sinon.

2 Indiquer parmi les quatre fonctions suivantes, celle qui correspond au schéma ci-dessus. Justifier votre réponse.

```
def valide(x,y):
    return (0.2<x%1<0.8 and 0.2<y%1<0.8)
```

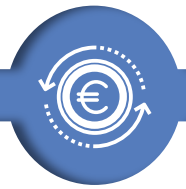
```
def valide(x,y):
    return (0<x%1<1 and 0<y%1<1)
```

```
def valide(x,y):
    return (0.8<x%1<1 and y>1)
```

```
def valide(x,y):
    return 0<x+y<1
```

3 Émettre une hypothèse quant aux résultats obtenus suite aux appels `valide(0.3,0.6)` et `valide(0.9,0.4)` au regard de la situation.

4 Réaliser les deux appels précédents dans la console d'exécution afin de vérifier votre hypothèse.



Partie N°2 : création de la grille de jeu

1. Saisir successivement les deux instructions ci-dessous dans la console d'exécution.
   - `import matplotlib.pyplot as plt` ;
   - `plt.plot([0,1],[2,3],'b')`.

2. En déduire l'objectif mathématique de l'instruction `plt.plot([xA,xB],[yA,yB],'b')`.

La fonction `Grille` ci-contre permet de créer un quadrillage de 10 carreaux sur 10.

```
import matplotlib.pyplot as plt

def Grille():
    plt.xticks([]),plt.yticks([])
    for i in range(  ?  ):
        plt.plot([0, 10],  ?  , 'b')
        plt.plot([i, i],  ?  , 'b')
    plt.axis('scaled')
```

3. Compléter les éléments manquants et tester la fonction `Grille` dans la console d'exécution.

Partie N°3 : simulation du jeu et étude fréquentiste

La fonction `fc` adossée à l'algorithme ci-dessous prend comme argument le nombre `n` de pièces lancées et simule le jeu du franc carreau pour ces `n` lancers.

```
Fonction fc(n):
    Afficher le quadrillage
    X ← [abscisses du centre des n pièces]
    Y ← [ordonnées du centre des n pièces]
    gagne ← [résultat True/False des n parties]
    Afficher les n pièces dans le quadrillage
Fin
```

```
def fc(n):
    Grille()
    X = [    I1    for _ in range(n)]
    Y = [    I1    for _ in range(n)]
    gagne = [    I2    for (a,b) in zip(X,Y)]
    plt.scatter(X,Y ,c=gagne,s=70)
```

1. Les coordonnées du centre de la pièce sur le quadrillage sont générées par deux nombres aléatoires compris entre 0 et 10. Préciser l'instruction notée **I1** qui permet de générer ces coordonnées.

2. En vous aidant des deux premières parties, compléter l'instruction notée **I2** dans la fonction `fc` ci-dessus.

3. Préciser le rôle de la dernière ligne de la fonction `fc`.

4. Modifier la fonction `fc` pour qu'elle renvoie la fréquence de parties gagnées sur `n` lancers de pièces.

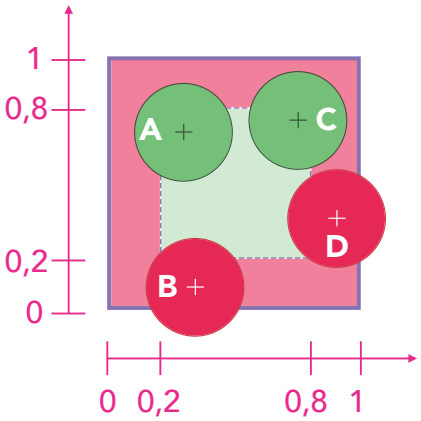
5. Réaliser plusieurs essais de la fonction `fc` dans la console d'exécution et répondre à la problématique.

- *Expérimenter pour observer la fluctuation des fréquences.*
- *Estimer la probabilité d'un événement à partir des fréquences.*
- *Faire preuve d'esprit critique face à une situation aléatoire simple.*
- *Modifier une situation donnée pour percevoir une version vulgarisée de la loi des grands nombres.*
- *Analyser un problème.*
- *Reconnaître le type d'une variable.*
- *Compléter un programme.*
- *Comprendre et utiliser des fonctions Python.*
- *Structurer un programme en ayant recours à des fonctions pour résoudre un problème donné.*

Partie N°1 : étude préliminaire

1 Une partie est considérée comme gagnée lorsque le jeton ne touche aucun des contours des cases constituant la grille. La partie du schéma ci-contre coloriée en rouge présente, pour un élément du quadrillage, l'ensemble des positions centrales d'une pièce correspondant à une partie perdue.

On en déduit assez aisément que seules les pièces A et C représentent une partie gagnée.



2 Au regard des éléments évoqués au point précédent, une partie est gagnée dès lors que le centre de la pièce se situe dans la zone verte du schéma ci-dessus. Les coordonnées du centre de la pièce doivent donc appartenir à l'intervalle] 0,2 ; 0,8 [. La fonction `valide` ci-dessous prend comme arguments les coordonnées `x` et `y` du centre de la pièce et renvoie :

- `True` dans le cas où elles appartiennent toutes les deux à l'intervalle] 0,2 ; 0,8 [;
- `False` dans le cas contraire.

```
def valide(x,y):
    return (0.2<x%1<0.8 and 0.2<y%1<0.8)
```

• connaître les opérateurs logiques et de comparaison

Remarque

La commande `x%1` permet de calculer le reste de la division euclidienne de l'abscisse du centre de la pièce par 1. Cette opération permet, dans le cadre de la situation, de limiter l'étude à un carreau et non plus sur l'ensemble du quadrillage.

3 Les appels `valide(0.3,0.6)` et `valide(0.9,0.4)` visent à vérifier si les parties sont gagnées ou non, dans le cas où le centre de la pièce a pour coordonnées respectives (0,3 ; 0,6) puis (0,9 ; 0,4).

4 Les résultats obtenus dans la console d'exécution suite à ces appels corroborent les éléments d'analyse évoqués au point 1, et valident l'hypothèse précédente.

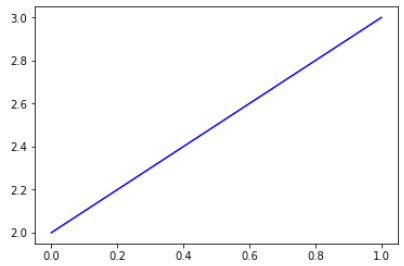
```
In [2]: valide(0.3,0.6)
Out[2]: True

In [3]: valide(0.9,0.4)
Out[3]: False
```

Partie N°2 : création de la grille de jeu

1 L'instruction `import matplotlib.pyplot as plt` crée un alias nommé `plt` permettant d'utiliser la bibliothèque graphique `matplotlib`.

La fenêtre graphique ci-dessous est alors obtenue suite à l'exécution de la seconde commande dans la console.

```
In [2]: plt.plot([0,1],[2,3],'b')
Out[2]: [<matplotlib.lines.Line2D at 0x115f66208>]
```



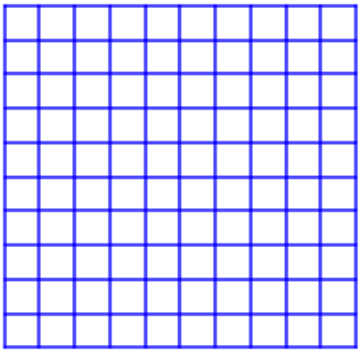
2 On peut déduire du point précédent que l'instruction `plt.plot([0,1],[2,3],'b')` permet de tracer en bleu le segment [AB] où A et B sont deux points de coordonnées respectives (x_A ; y_A) et (x_B ; y_B) dans un repère donné.

3 La grille à créer est un quadrillage composé de 10 lignes et de 10 colonnes. Le principe utilisé dans la fonction `Grille` est le suivant :

- pour `i` allant de 0 à 10 :
- On trace un segment horizontal reliant les points de coordonnées (0 ; `i`) et (10 ; `i`).
- On trace un segment vertical reliant les points de coordonnées (`i` ; 0) et (`i` ; 10).

```
for i in range(11):
    plt.plot([0, 10], [i, i], 'b')
    plt.plot([i, i], [0, 10], 'b')
```

```
def Grille():
    plt.xticks([]),plt.yticks([])
    for i in range(11):
        plt.plot([0, 10], [i, i], 'b')
        plt.plot([i, i], [0, 10], 'b')
    plt.axis('scaled')
```



Partie N°3 : simulation du jeu et étude fréquentiste

- connaître les instructions de la bibliothèque random
- connaître l'itérable ZIP et la compréhension d'une liste

1 L'instruction `random()` renvoie un nombre aléatoire compris entre 0 et 1 qui suit la loi uniforme. Les coordonnées X et Y du centre d'une pièce sont donc modélisées à partir de l'instruction **I1** suivante :

```
10*random()
```

2 La variable `gagne` est une liste contenant l'ensemble des résultats (sous forme d'un booléen) obtenus suite aux `n` simulations d'un lancer de pièce. Ces simulations sont réalisées à l'aide de la fonction `valide` détaillée dans la partie N°1 d'où l'instruction **I2** :

```
valide(a,b)
```

Remarque : utilisation de l'itérable ZIP

L'instruction `zip` renvoie une liste de n-uplets, chacun contenant un élément (lu de gauche à droite) des séquences mises en arguments.

```
In [2]: L1 = [1,2,3]
   ...: L2 = [4,5,6]
   ...: L3 = [a+b for (a,b) in zip(L1,L2)]
   ...: print(L3)
[5, 7, 9]
```

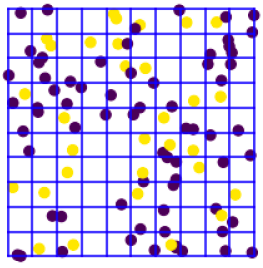
Dans l'exemple ci-contre, on affiche une liste L3 où sont enregistrées les sommes d'éléments de même rang contenus dans les listes L1 et L2.

3 L'instruction `plt.scatter(X,Y,c=gagne,s=70` place dans le repère obtenu suite à l'appel de la fonction `Grille()` les points de coordonnées (x ; y) appartenant respectivement aux listes X et Y. L'argument `c=color` permet de différencier la couleur de ces points en fonction des valeurs contenues dans la liste gagne (`True` ou `False`). Il est à noter que l'argument `s` fixe la superficie des points.

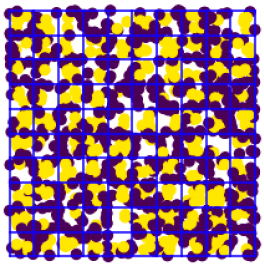
4 La liste `gagne` recense les résultats (`True` ou `False`) des `n` parties réalisées. La fréquence de parties gagnées correspond donc au rapport d'itérations de la valeur `True` contenues dans la liste `gagne` et du nombre `n` de parties. On obtient cette fréquence en ajoutant en dernière ligne l'instruction :

```
return sum(gagne)/n
```

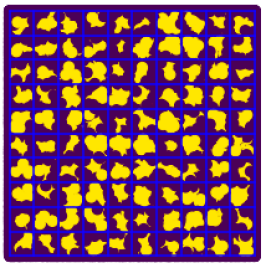
5 On réalise plusieurs appels de la fonction `fc` dans la console, pour différentes valeurs de `n` :



```
In [3]: fc(100)
Out[3]: 0.31
```



```
In [4]: fc(1000)
Out[4]: 0.37
```



```
In [5]: fc(1000000)
Out[5]: 0.360548
```

Au regard des résultats ci-dessus, on peut estimer les chances de gagner au franc carreau à environ 36 %.

Le défi du spaghetti

Situation

Qu'on les préfère à la carbonara ou à la bolognaise, les spaghettis sont un des aliments les plus consommés dans le monde. En moyenne, la longueur d'un spaghetti est d'environ 25 cm. En cassant un spaghetti au hasard en trois morceaux, il est parfois possible de construire un triangle.

Comment estimer la probabilité de pouvoir construire un triangle à partir des trois morceaux obtenus ?

Cette probabilité dépend-elle de la façon dont le spaghetti a été coupé ?

Déroulement

L'ensemble des fonctions Python est à saisir dans un même script nommé SPAG.

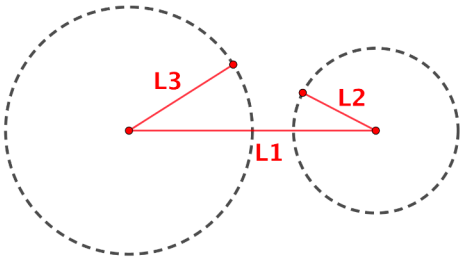
Étude préliminaire

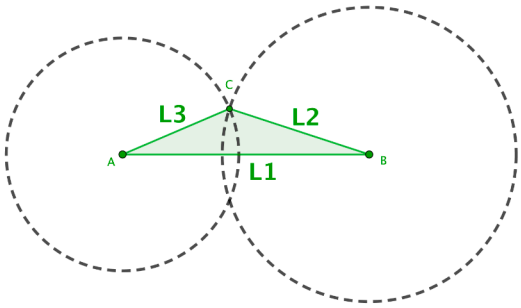
1 Reproduire et renseigner le tableau suivant après avoir coupé en trois parties au hasard les cinq spaghettis mis à votre disposition.

	Longueur L1 du 1er morceau	Longueur L2 du 2e morceau	Longueur L3 du 3e morceau	Triangle constructible?
Spaghetti N°...				

En géométrie on peut démontrer qu'un triangle est constructible en utilisant l'inégalité triangulaire :

« Dans un triangle, la longueur de chaque côté est inférieure ou égale à la somme des deux autres. »





2 Écrire en Python une fonction `constructible` prenant comme arguments les longueurs `AB`, `BC` et `AC` qui renvoie `True` si le triangle ABC est constructible et `False` sinon.

3 Valider les résultats obtenus au point 1 en utilisant la fonction `constructible` dans la console d'exécution.

4 Indiquer si les tests réalisés sur les cinq spaghettis précédents permettent de répondre à la première problématique. Justifier la réponse.

Le défi du spaghetti

Étude de deux cas de figure

On souhaite maintenant évaluer la probabilité de construire un triangle après avoir coupé un spaghetti en trois, selon deux méthodes.

La fonction `methode1` ci-dessous traite une des façons de couper un spaghetti en trois et vérifie si les trois morceaux obtenus permettent de construire un triangle.

```
from random import random

def methode1():
    coupe1 = 25*random()
    coupe2 = 25*random()
    M1 = min(coupe1,coupe2)
    M2 = max(coupe1,coupe2)
    return constructible(M1,M2-M1,25-M2)
```

1. À partir du mémento mis à disposition à la fin du livret, expliquer, dans la fonction `methode1` l'utilisation de :
   a. l'instruction `25*random()`
   b. l'instruction `min(coupe1,coupe2)`
   c. l'instruction `max(coupe1,coupe2)`

2. Expliquer le principe de coupe du spaghetti utilisé dans la fonction `methode1`. Un schéma pourra éventuellement illustrer la démarche.

3. Réaliser quelques tests de la fonction `methode1` dans la console d'exécution.

4. Proposer une démarche permettant d'estimer la probabilité de construire un triangle après avoir coupé un spaghetti selon la méthode 1.

5. Écrire en Python une fonction `frequence` prenant comme argument le nombre `n` de spaghettis testés et qui illustre la démarche proposée au point précédent.

6. Réaliser plusieurs appels à la fonction `frequence` pour différents nombres de spaghettis et répondre à la première problématique.

On souhaite maintenant évaluer cette probabilité avec une seconde méthode de coupe :

- on coupe aléatoirement une première fois le spaghetti ;
- on repère le morceau le plus long ;
- on coupe aléatoirement celui-ci en deux morceaux.

7. Écrire en Python une fonction `methode2` traduisant la méthode précédente.

8. En utilisant dans la console d'exécution une version de la fonction `frequence` adaptée à la méthode 2 , répondre à la seconde problématique.

- *Expérimenter pour observer la fluctuation des fréquences.*
- *Estimer la probabilité d'un événement à partir des fréquences.*
- *Faire preuve d'esprit critique face à une situation aléatoire simple.*
- *Reconnaître le type d'une variable.*
- *Analyser un problème.*
- *Concevoir un algorithme ou un programme simple pour résoudre un problème.*
- *Comprendre et utiliser des fonctions Python.*

Étude préliminaire

1 L'expérimentation, qui est réalisée avec cinq spaghettis, peut entraîner instinctivement une découpe spécifique des spaghettis de manière à obtenir des morceaux sensiblement de la même longueur. Il est intéressant d'en faire l'analyse car cela reviendrait à conjecturer qu'un triangle est toujours constructible. La mise en œuvre d'un contre-exemple concret permet alors une réflexion plus approfondie.

2 L'utilisation de l'inégalité triangulaire et des schémas proposés permettent d'écrire la fonction `constructible` suivante :

```
def constructible(AB,BC,AC):
    return AB<=BC+AC and AC<=AB+BC and BC<=AB+AC
```

- connaître les opérateurs arithmétiques et les opérateurs logiques

3 Les différents appels réalisés ci-dessous présentent plusieurs cas de coupe de spaghettis réalisés au point 1 :

	L1 (en cm)	L2 en cm)	L3 (en cm)	Triangle constructible?
Spaghetti N°1	5	5	15	NON
Spaghetti N°2	8	7	10	OUI
Spaghetti N°3	8	4	13	NON
Spaghetti N°4	9	7	9	OUI
Spaghetti N°5	6,5	9,2	9,3	OUI

```
In [2]: constructible(5,5,15)
Out[2]: False

In [3]: constructible(8,7,10)
Out[3]: True

In [4]: constructible(8,4,13)
Out[4]: False

In [5]: constructible(9,7,9)
Out[5]: True

In [6]: constructible(6.5,9.2,9.3)
Out[6]: True
```

4 Au regard des points précédents, la taille de l'échantillon (n=5) ne permet pas d'estimer avec pertinence la probabilité de pouvoir construire un triangle à partir des trois morceaux obtenus dans chacun des cinq cas.

• faire preuve d'esprit critique vis à vis d'une situation aléatoire simple

Étude de deux cas de figure

1 La fonction `methode1` permet de simuler le découpage d'un spaghetti en trois parties, et de renvoyer si les morceaux obtenus permettent de construire un triangle.

a. l'instruction `25*random()` renvoie un nombre aléatoire compris entre 0 et 25 ;

b. l'instruction `min(coupe1,coupe2)` renvoie la plus petite valeur des variables `coupe1` et `coupe2` ;

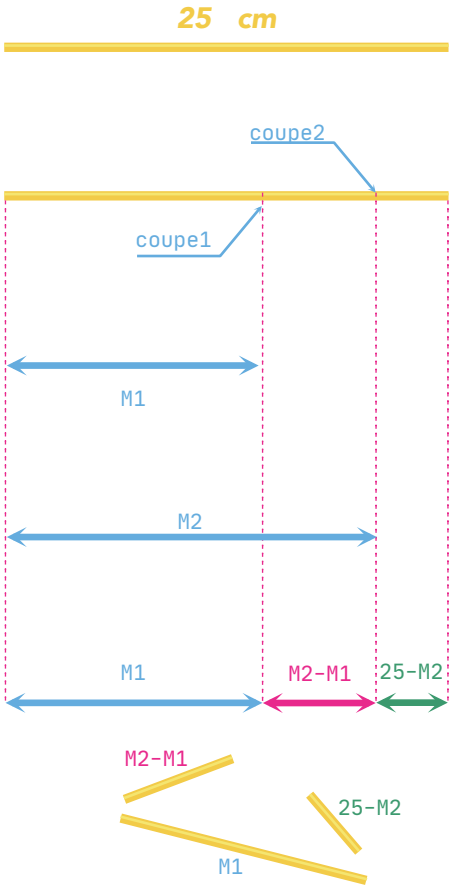
c. l'instruction `max(coupe1,coupe2)` renvoie la plus grande valeur des variables `coupe1` et `coupe2`.

• connaître les instructions de gestion des nombres aléatoires

• connaître les instructions min et max

2 Le principe de coupe d'un spaghetti utilisé dans la fonction `methode1` se déroule en plusieurs étapes :

• analyser et comprendre une fonction Python



- On utilise un spaghetti de 25 cm de long ;
- On choisit au hasard deux longueurs notées `coupe1` et `coupe2` entre 0 et 25 cm ;

```
coupe1 = 25*random()
coupe2 = 25*random()
```

- On appelle `M1` le morceau ayant la plus petite valeur des deux ;

```
M1 = min(coupe1,coupe2)
```

- On appelle `M2` le morceau ayant la plus grande valeur des deux ;

```
M2 = max(coupe1,coupe2)
```

- On vérifie si les trois morceaux de longueur `M1`, `M2-M1` et `25-M2` permettent de construire un triangle ou non.

```
return constructible(M1,M2-M1,25-M2)
```

3 On exécute la fonction `methode1` en appuyant sur le bouton ▶ .

```
In [2]: methode1()
Out[2]: False

In [3]: methode1()
Out[3]: False

In [4]: methode1()
Out[4]: True

In [5]: methode1()
Out[5]: False

In [6]: methode1()
Out[6]: False
```

Celle-ci renvoie un booléen correspondant au résultat renvoyé par la fonction `constructible` prenant comme arguments les longueurs correspondant aux trois morceaux de spaghetti:

- `False` dans le cas où le triangle formé par les trois morceaux de spaghetti obtenus n'est pas constructible ;
- `True` dans le cas où ce triangle est constructible.

4 Les différents essais réalisés pour l'ensemble de la classe permettent de conjecturer qu'une taille d'échantillon trop petite ne permet pas d'estimer avec pertinence la probabilité de pouvoir construire un triangle avec les trois morceaux de spaghetti. Pour pouvoir estimer cette probabilité, on pourra par exemple proposer d'écrire en Python une fonction `frequence` prenant comme argument le nombre `n` de spaghettis testés et qui renvoie la fréquence de cas où le triangle formé par les trois morceaux obtenus à l'aide de la méthode N°1 est constructible.

• analyser un problème et utiliser une fonction Python pour le résoudre.

5 La fonction `frequence` utilise un compteur (variable `s`) initialisé à 0. Pour chaque spaghetti testé, on vérifie si le triangle formé par les trois morceaux obtenus est constructible ou non. Si c'est le cas, on incrémente la valeur de la variable `s`. Une fois les `n` spaghettis testés, la fonction `frequence` renvoie le quotient de la valeur de la variable `s` par le nombre `n` de spaghettis testés, ce qui correspond à la fréquence recherchée.

• utiliser une boucle bornée et une instruction conditionnelle.

```
Fonction frequence(n):
    s ← 0
    Pour i allant de 0 à n-1
        Si le résultat de l'appel
        methode1( ) est VRAI
    alors
            s ← s + 1
        Fin Si
    Fin Pour
    Renvoyer s/n
Fin
```

▶

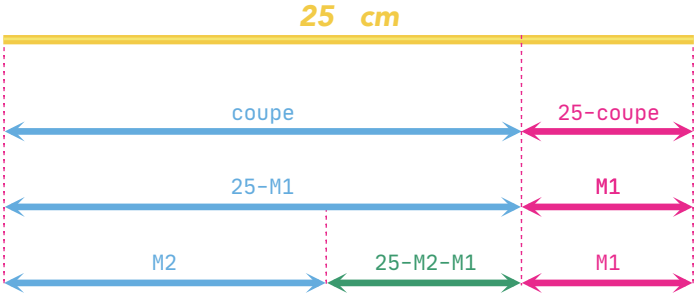
```
def frequence(n):
    s=0
    for i in range(n):
        if methode1():
            s+=1
    return s/n
```

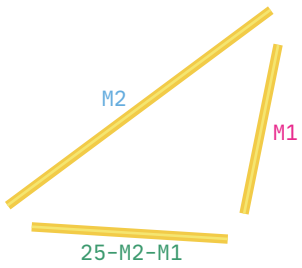
▼

6 Les différents appels ci-contre permettent d'exploiter une version vulgarisée de la loi des grands nombres et ainsi pouvoir estimer la probabilité recherchée à environ 0,25 soit 1 chance sur 4.

```
In [2]: frequence(10)
Out[2]: 0.2

In [3]: frequence(100)
Out[3]: 0.29

In [4]: frequence(10000)
Out[4]: 0.2461

In [5]: frequence(1000000)
Out[5]: 0.2500133
```

7 Le schéma ci-dessous présente les différentes étapes illustrant le principe de découpe du spaghetti selon la méthode N°2. Ce schéma pourra éventuellement faire l'objet d'une différenciation pédagogique pour traduire directement le principe de découpe explicité dans la méthode N°2 à l'aide d'une fonction Python.





Tout comme la fonction `methode1`, la fonction `methode2` prend comme argument la variable `n` correspondant au nombre de spaghettis testés.

```
def methode2():
    coupe = 25*random()
    M1 = min(coupe,25-coupe)
    M2 = (25-M1)*random()
    return constructible(M1,M2,25-M1-M2)
```

On affecte à la variable `coupe` un nombre aléatoire suivant la loi uniforme compris entre 0 et 25.

On affecte à la variable `M1` la longueur du morceau ayant la plus petite longueur.

On affecte à la variable `M2` un nombre aléatoire suivant la loi uniforme compris entre 0 et 25-`M1`.

La fonction renvoie un booléen indiquant si le triangle est constructible (`True`) ou non (`False`).

8 On modifie la fonction `frequence` développée au point 5 en remplaçant la méthode de découpe initiale par la méthode N°2. Les résultats obtenus suite aux différents appels ci-dessous permettent d'estimer une probabilité d'environ 0,39 pour la méthode N°2. Ces éléments amènent à la conclusion que la probabilité de pouvoir construire un triangle à partir de trois morceaux de spaghetti dépend donc directement de la méthode utilisée pour les couper.

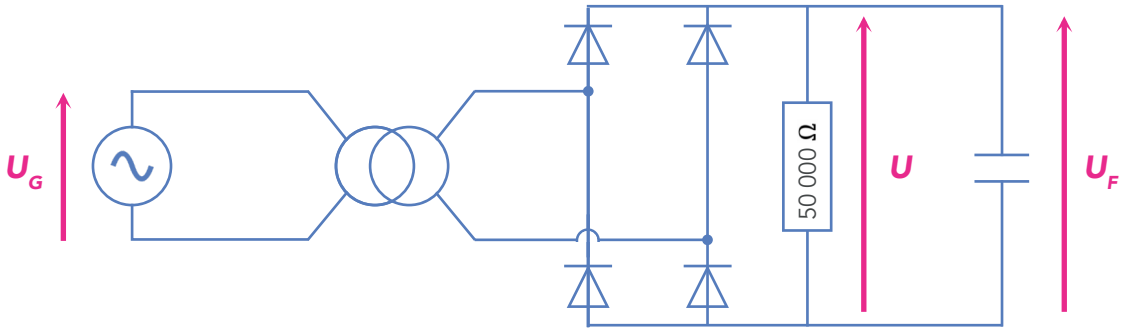
```
def frequence(n):
    s=0
    for i in range(n):
        if methode2():
            s+=1
    return s/n
```

```
In [2]: frequence(10)
Out[2]: 0.4

In [3]: frequence(10000)
Out[3]: 0.381

In [4]: frequence(1000000)
Out[4]: 0.386074

In [5]: frequence(100000000)
Out[5]: 0.38629843
```

Filtrage d'une tension

Situation

Les cours de physique-chimie ont montré que bon nombre de chargeurs électriques permettent de convertir la tension alternative fournie par le réseau pour l'adapter à la tension de l'appareil à charger. Le schéma ci-dessous illustre les premières étapes de cette conversion.



Comment modéliser la tension U en sortie d'un pont de diodes ?

Quel est le rôle du condensateur monté en dérivation sur le résistor ?

Déroulement

Dans cette étude, on définit la tension U_G fournie par le générateur par la fonction mathématique définie pour tout nombre t appartenant à l'intervalle $[0\ ;\ 4\pi]$ par :

$U_G(t) = U_{max}\ sin(t)$

L'ensemble des fonctions Python est à saisir dans un même script nommé TENSION.

Modélisation de la tension en sortie du pont de diodes.

```
import matplotlib.pyplot as plt
from math import pi , sin , exp

def dA(Um,R):
    t=[(4*pi)*i/1000 for i in range(1000)]
    y = [Um*sin(i) for i in t]
    U = [-j if j<0 else j for j in y]
    plt.style.use('bmh')
    plt.plot(t,U)
    return t,U
```

1. Indiquer à quoi correspond la variable `t` et préciser son type.
2. Expliquer le rôle de l'instruction `U = [-j if j<0 else j for j in y]`.
3. Justifier que la fonction `dA` permet de modéliser la tension `U` en sortie du pont de diodes.
4. Valider la réponse faite au point précédent en réalisant l'appel `dA(17,50000)` dans la console d'exécution et donner le nom du signal obtenu.

Filtrage d'une tension

Rôle du condensateur

Pour établir le rôle du condensateur présenté à la page précédente, on simule l'évolution de la tension `Uf` aux bornes du condensateur.

Pour cela on veut visualiser l'évolution de la tension redressée `U` et de la tension `Uf` aux bornes d'un condensateur de capacité C. Dans cette optique on utilise la fonction `filtrage` ci-dessous où certains éléments ont été cachés (dans l'algorithme ou dans la fonction) :

```
Fonction filtrage( Um,R,C ) :                    def filtrage(Um,R,C):
    t,U ← [résultats de l'appel dA(Um,R)]            ?
    Uf ← [ ]                                         Uf = []
    Pour chaque valeur i allant de 0  à 999          ?

        D ← Um × e^(−(t[i]−π/2)[π] / R×C)            D = Um*exp(-((t[i]-pi/2)%pi)/(R*C))

        Si ?                                         if U[i]< D and t[i] > pi/2 :
            Uf ← Uf + D                                  ?
        ?                                            else :
            Uf ← Uf + U[i]                               ?
        Fin Si
    Fin Pour
    ?                                                plt.plot(t,Uf)
Fin
```

1. Compléter les éléments manquants de l'algorithme et de la fonction `filtrage` ci-dessus.

2. Réaliser différents appels à la fonction `filtrage` dans la console d'exécution. On pourra par exemple tester les capacités suivantes : C_1 = 100 µF, C_2 = 470 µF, C_3 = 1000 µF.

3. Analyser les représentations graphiques obtenues afin d'expliquer l'influence de la valeur de la capacité du condensateur sur l'évolution de la tension redressée.

4. Répondre à la seconde problématique.

- *Concevoir un algorithme ou un programme simple pour résoudre un problème.*
- *Analyser un problème.*
- *Reconnaître le type d'une variable.*
- *Comprendre et utiliser des fonctions Python.*
- *Générer et parcourir une liste.*
- *Itérer une ou plusieurs instructions sur les éléments d'une liste.*

Modélisation de la tension de sortie du pont de diodes

1 La variable `t` est une liste générée par compréhension :

```
t = [(4*pi)*i/1000 for i in range(1000)]
```

Pour chaque valeur `i` allant de 0 à 999, on ajoute à la liste `t` la valeur $\frac{4\pi \times i}{1000}$. Cette liste nous sert de base temporelle pour les différents signaux à représenter dans les points suivants.

• choisir ou reconnaître le type d'une variable

2 La liste `y` permet d'illustrer la tension d'alimentation U_G pour tout nombre de la liste `t`.

L'instruction `U=[ -j if j<0 else j for j in y ]` crée une liste notée `U` par compréhension. Pour tout élément `j` de la liste `y`, on ajoute à la liste `U` :

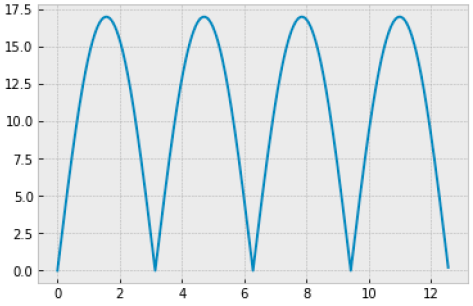
- la valeur opposée de `j` si elle est négative ;
- la valeur de `j` dans le cas contraire.

3 Au regard du schéma électrique proposé en énoncé et des capacités travaillées durant les séances de physique-chimie, la situation expérimentale traite d'un redressement double alternance. On déduit du point précédent que les éléments de la liste `U` permettent de modéliser la tension U en sortie du pont de diodes.

L'instruction `plt.plot(t,U)` donne la représentation graphique de la fonction U pour tout nombre compris entre 0 et $4\,\pi$, c'est-à-dire la tension en sortie du pont de diodes.

• comprendre et utiliser des fonctions

4 Le signal obtenu suite à l'appel `dA(17,50000)` permet de valider la conjecture émise au point précédent. On observe bien ici un redressement double alternance de la tension fournie par le générateur.



Rôle du condensateur

1 La fonction `filtrage` utilise la fonction `dA` étudiée en première partie afin d'exploiter la tension aux bornes du pont de diodes :

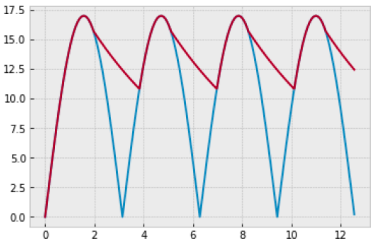
- la variable `t` correspond à la base temporelle de l'étude expérimentale sous forme d'une liste de 1 000 valeurs ;
- la variable `U` correspond à la modélisation de la tension en sortie du pont de diodes sous forme d'une liste de 1 000 valeurs ;
- la variable `Uf` correspond à la modélisation de la tension aux bornes du condensateur ;
- la variable `D` correspond à la valeur de la tension déchargée par le condensateur à l'instant `t` ;

• comprendre un algorithme sa traduction en langage Python

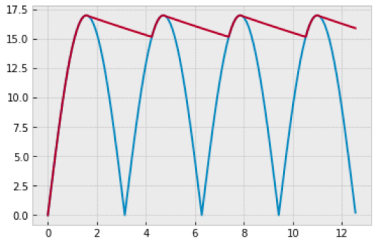
```
Fonction filtrage( Um,R,C ) :                    def filtrage(Um,R,C):
    t,U ← [résultats de l'appel dA(Um,R)]            t,U = dA(Um,R)
    Uf ← [ ]                                         Uf = []
    Pour chaque valeur i allant de 0  à 999          for i in range(1000) :

        D ← U_m × e^{-(t[i]-π/2)[π]/(R×C)}               D = Um*exp(-((t[i]-pi/2)%pi)/(R*C))

        Si U[i]<D et t[i]>π/2 alors                      if U[i]< D and t[i] > pi/2 :
            Uf ← Uf + D                                      Uf.append(D)
        Sinon                                            else :
            Uf ← Uf + U[i]                                   Uf.append(U[i])
        Fin Si
    Fin Pour
    Afficher le nuage de points (t,Uf)               plt.plot(t,Uf)
Fin
```

2 On réalise les appels suivants dans la console d'exécution :

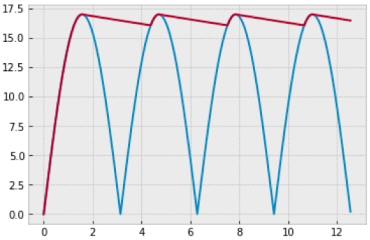
```
In [2]: filtrage(17,50000,100e-6)
```



```
In [3]: filtrage(17,50000,470e-6)
```



```
In [4]: filtrage(17,50000,1000e-6)
```



3 Au regard des signaux obtenus au point précédent, on remarque que plus la valeur de la capacité du condensateur placé en dérivation sur la charge résistive est importante et moins l'ondulation de la tension est grande.

4 Le rôle du condensateur monté en dérivation sur le résistor est de filtrer une tension redressée. Ce dispositif est fréquemment utilisé dans les différents types d'alimentation stabilisée permettant la charge d'appareils comme les smartphones, les ordinateurs ou tout autre appareil fonctionnant sur batterie. Ces alimentations convertissent une tension alternative sinusoïdale en une tension continue nécessaire à la recharge d'une batterie d'accumulateurs.

Éléments complémentaires de progression

La bivalence est un des points clés de la transformation de la voie professionnelle. Garantir la cohérence de la formation mathématique et scientifique des élèves fait partie intégrante des lignes directrices stipulées en préambule des programmes de la voie professionnelle.

En complément de l'étude de la fonction qui, à tout nombre réel t, associe le nombre $a \sin(\omega t + \varphi)$, cette situation traite une partie du programme d'électricité (obtenir un courant continu à partir d'un courant alternatif). De nombreuses capacités et connaissances déclinées dans le module Algorithmique & Programmation y sont également mises en exergue

En outre, l'utilisation de la fonction exponentielle de base e dans la seconde partie de cette activité fait de celle-ci une introduction possible au module sur les fonctions logarithme népérien et exponentielle dans le cadre des programmes complémentaires en vue de la préparation à une poursuite d'études.

Une variable est une étiquette liée à un objet en mémoire. Son nom ne peut pas être un mot clé du langage (`def`, `return`, `for`, `if`, etc.), ni même commencer par un chiffre. Une variable est sensible à la casse, c'est-à-dire que Python distingue les lettres capitales des lettres minuscules.

```
In [1]: nb=31
In [2]: a=b=c=2
In [3]: var1,var2 = 5,"txt"
In [4]: var1,var2 = var2,var1
In [5]: nb
Out[5]: 31
In [6]: a
Out[6]: 2
In [7]: var1
Out[7]: 'txt'
```

Affectation

- `nb = 31` : affectation de la valeur 31 à la variable `nb`.
- `a = b = c = 2` : affectation de la valeur 2 aux variables `a`, `b` et `c`.
- `var1,var2 = 5, "txt"` : affectations simultanées de la valeur 5 à la variable `var1` et la chaîne de caractères « txt » à la variable `var2`.
- `var1,var2 = var2,var1` : permutation des valeurs des variables `var1` et `var2`.

```
In [1]: type(3)
Out[1]: int

In [2]: type(4.93)
Out[2]: float

In [3]: type(2+3j)
Out[3]: complex

In [4]: type("coucou")
Out[4]: str

In [5]: round(1/3,2)
Out[5]: 0.33

In [6]: type([2,4,6,8])
Out[6]: list

In [7]: type((2,'mot'))
Out[7]: tuple
```

Types de base

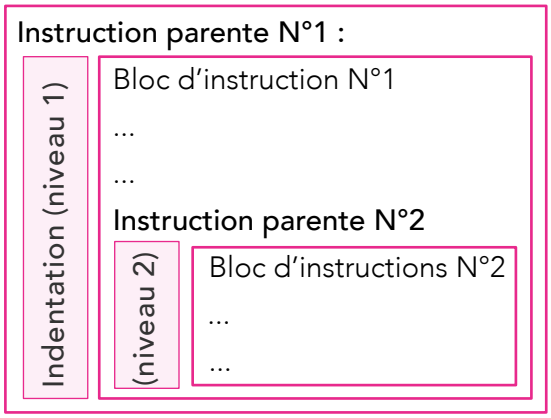
- `int` : nombres entiers.
- `float` : nombres à virgule flottante.
- `complex` : nombres complexes.
- `str` : chaînes de caractères.
- `round(a,2)` : valeur arrondie au centième de la variable `a`.
- `list` : liste de valeurs (numériques ou non).
- `type(test)` : renvoie le type de la variable `test`.
- `tuple` : n-uplet de valeurs non modifiables.

Il est possible de convertir une variable dans un autre type. Par exemple, l'instruction `str(a)` convertit la variable `a` en une chaîne de caractères.

```
In [1]: a,b = 20,2
In [2]: a+b
Out[2]: 22
In [3]: a-b
Out[3]: 18
In [4]: a/b
Out[4]: 10.0
In [5]: a*b
Out[5]: 40
In [6]: a//b
Out[6]: 10
In [7]: a%b
Out[7]: 0
In [8]: a**b
Out[8]: 400
```

Opérateurs arithmétiques

- `a+b` : somme des valeurs `a` et `b`.
- `a-b` : différence des valeurs `a` et `b`.
- `a/b` : rapport des valeurs `a` et `b`.
- `a*b` : produit des valeurs `a` et `b`.
- `a//b` : quotient de `a` par `b`.
- `a%b` : reste de la division euclidienne de `a` par `b`.
- `a**b` : `a` puissance `b`.



Les boucles et les instructions conditionnelles sont des blocs de code où chaque instruction est indentée (ajout d'une tabulation par rapport à la ligne précédente).

```
def testif(a):
    if a == 1:
        txt = "Gagné"
    elif a == 2:
        txt = "Perdu"
    else:
        txt = "Rejouer"
    return txt
```

Instruction conditionnelle

```
def testif(a):
    Si la variable a vaut 1:
        Alors txt ← "Gagné".
    Si la variable a vaut 2:
        Alors txt ← "Perdu".
    Sinon:
        txt ← "Rejouer".
    Renvoyer la valeur de txt
```

Les blocs d'instructions `elif` et `else` sont facultatifs. Deux commandes d'ajout ou de suppression d'un niveau d'indentation sont disponibles dans le menu Outils :

```
def twhile(a):
    while a >= 0:
        print(a)
        a -= 1
```

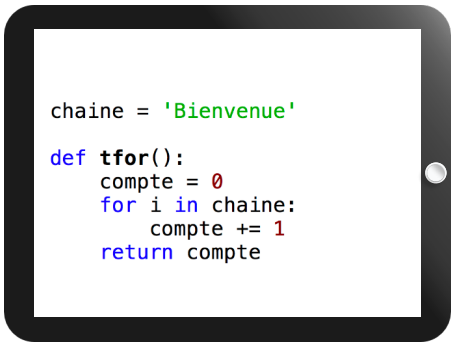
Boucle non bornée

```
def twhile(a):
    Tant que la variable a ≥ 0:
        On affiche la valeur de a.
        a ← a-1
```

Autres commandes optionnelles disponibles :

- `break` : sortir immédiatement de la boucle.
- `continue` : passer directement à l'itération suivante.
- `else` : exécuter un bloc d'instructions en sortie de boucle.

Bien que l'instruction `print` soit à éviter, son utilisation dans la fonction précédente n'a pour seul objectif que de proposer un exemple simple d'utilisation d'une boucle non bornée `while`.



```
chaine = 'Bienvenue'

def tfor():
    compte = 0
    for i in chaine:
        compte += 1
    return compte
```

Boucle bornée

```
def tfor():
    compte ← 0
    Pour chaque élément i de la variable chaine:
        compte ← compte + 1.
    Renvoyer la valeur de compte:
```

Autres commandes optionnelles disponibles :

- `break` : sortir immédiatement de la boucle.
- `continue` : passer directement à l'itération suivante.
- `else` : exécuter un bloc d'instructions en sortie de boucle.

De manière générale, une boucle bornée fait référence à un itérable qui peut être un nombre entier, une chaîne de caractères ou une liste en fonction de l'objectif visé.

La variable `i` prend successivement comme valeurs :

- `for i in range(5)` : tous les nombres entiers naturels compris entre 0 et (5−1).
- `for i in range(3,8)` : tous les nombres entiers naturels compris entre 3 et (8−1).
- `for i in range(1,9,2)` : tous les nombres entiers naturels compris entre 1 et (9−1) avec un pas de 2.
- `for i in 'chimie'` : tous les caractères de la chaîne `'chimie'`.
- `for i in [3,14,-4.3]` : toutes les valeurs contenues dans la liste `[3,14,-4.3]`.

Listes : définition, opérations & méthodes (partie 1)

Une liste est une séquence d'éléments modifiables. Elle est définie entre crochets et peut contenir différents types de variables : nombres entiers, nombres à virgule flottante, chaînes de caractères, booléens. Contrairement à une liste, un tuple est une séquence non modifiable qui est définie entre parenthèses.

Les différentes opérations et méthodes concernent donc uniquement les listes.
L'indexation commence à 0 pour les listes.

```
In [1]: L = list((1,'ok',True))
In [2]: len(L)
Out[2]: 3
In [3]: L.append(2.5)
In [4]: L
Out[4]: [1, 'ok', True, 2.5]
In [5]: L.remove('ok')
In [6]: L
Out[6]: [1, True, 2.5]
In [7]: L.insert(0,"a")
In [8]: L
Out[8]: ['a', 1, True, 2.5]
In [9]: L.count(2.5)
Out[9]: 1
```

Principales opérations et méthodes des listes

- `L=list((1, "ok",True))` : créer la liste `L=[1, "ok",True]`.
- `len(L)` : nombre de valeurs de la liste L.
- `L.append(2.5)` : ajouter la valeur `2.5` à la fin de la liste L.
- `L.remove("ok")` : supprimer la valeur `"ok"` de la liste L.
- `L.insert(0,"a")` : ajouter la valeur `"a"` au rang 0 de la liste L.
- `L.count(2.5)` : compter le nombre d'occurrences de la valeur `2.5` dans la liste L.

```
In [1]: L = [8,1,3,-5.4,1.5]

In [2]: sum(L)
Out[2]: 8.1

In [3]: sorted(L)
Out[3]: [-5.4, 1, 1.5, 3, 8]

In [4]: L
Out[4]: [8, 1, 3, -5.4, 1.5]

In [5]: L.sort()

In [6]: L
Out[6]: [-5.4, 1, 1.5, 3, 8]

In [7]: min(L)
Out[7]: -5.4

In [8]: max(L)
Out[8]: 8
```

Cas d'une liste exclusivement numérique

On considère pour l'exemple la liste `L=[8,1,3,-5.4,1.5]`.

- `sum(L)` : calculer la somme des valeurs contenues dans la liste L.
- `sorted(L)` : afficher les valeurs de la liste L par ordre croissant.
- `L.sort()` : trier par ordre croissant et mettre à jour les valeurs de la liste L.
- `min(L)` : afficher la plus petite valeur de la liste L.
- `max(L)` : afficher la plus grande valeur de la liste L.

```
In [1]: L1 = [3,False,'non']

In [2]: L2 = [2,9,2,4]

In [3]: L2[3]
Out[3]: 4

In [4]: L2[-1]
Out[4]: 4

In [5]: L1.index(False)
Out[5]: 1

In [6]: L = L1+L2

In [7]: L
Out[7]: [3, False, 'non', 2, 9, 2, 4]
```

Autres instructions sur les listes

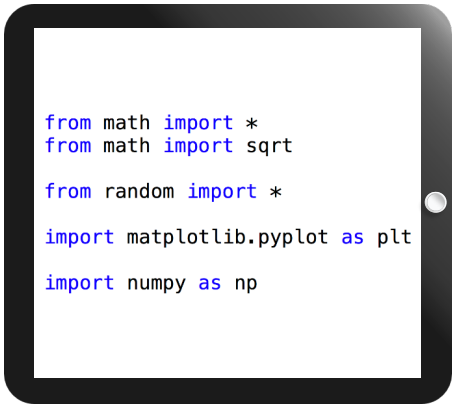
- `L2[3]` : valeur de rang 4 (d'indice 4) dans la liste L2.
- `L2[-1]` : dernière valeur de la liste L2.
- `L1.index[False]` : rang de la valeur `False` dans la liste L1.
- `L=L1+L2` : créer une liste L concaténant les listes L1 et L2.

```
In [8]: L2.extend([1,2,3])

In [9]: L2
Out[9]: [2, 9, 2, 4, 1, 2, 3]

In [10]: L1[1] = 15

In [11]: L1
         [3, 15, 'non']
In [12]:
Out[12]: del L1[1]

In [13]: L1
         [3, 'non']
In [14]:
Out[14]: L2.reverse()

In [15]: L2
         [3, 2, 1, 4, 2, 9, 2]
```

Autres instructions sur les listes (suite)

- `L2.extend([1,2,3])` : étendre la liste `L2` avec la liste `[1,2,3]`.
- `L1[1]=15` : remplacer la 2e valeur de la liste `L1` par `15`.
- `del L1[1]` : supprimer la 2e valeur de la liste `L1`.
- `L2.reverse()` : inverser l'ordre des valeurs de la liste `L2`.

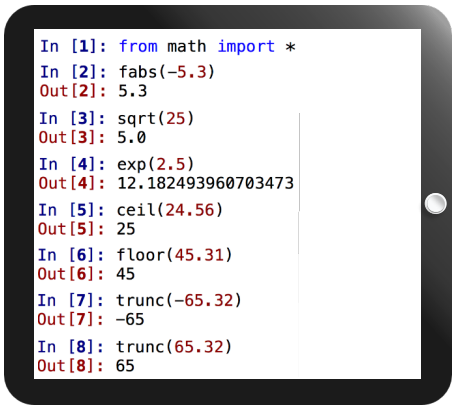
Il est possible de créer une liste par compréhension et/ou en tenant compte d'une instruction conditionnelle.



```
from math import *
from math import sqrt

from random import *

import matplotlib.pyplot as plt

import numpy as np
```

Import des modules

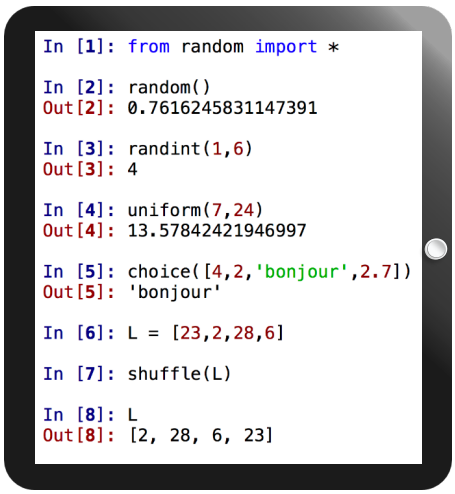
- `from math import *` : import de l'ensemble des instructions du module `math`.
- `from random import *` : import de l'ensemble des instructions du module `random`.
- `import matplotlib.pyplot as plt` : import du module `matplotlib` en tant qu'objet appelé `plt`.
- `import numpy as np` : import du module `numpy` en tant qu'objet appelé `np`.



```
In [1]: from math import *
In [2]: fabs(-5.3)
Out[2]: 5.3
In [3]: sqrt(25)
Out[3]: 5.0
In [4]: exp(2.5)
Out[4]: 12.182493960703473
In [5]: ceil(24.56)
Out[5]: 25
In [6]: floor(45.31)
Out[6]: 45
In [7]: trunc(-65.32)
Out[7]: -65
In [8]: trunc(65.32)
Out[8]: 65
```

Module math

- `fabs` : valeur absolue d'un nombre.
- `sqrt` : racine carrée d'un nombre.
- `exp` : image d'un nombre par la fonction exponentielle de base e.
- `ceil` : premier entier supérieur à un nombre.
- `floor` : premier entier inférieur à un nombre.
- `trunc` : valeur tronquée d'un nombre.



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

In [2]: random()
Out[2]: 0.7616245831147391

In [3]: randint(1,6)
Out[3]: 4

In [4]: uniform(7,24)
Out[4]: 13.57842421946997

In [5]: choice([4,2,'bonjour',2.7])
Out[5]: 'bonjour'

In [6]: L = [23,2,28,6]

In [7]: shuffle(L)

In [8]: L
Out[8]: [2, 28, 6, 23]
```

Module random

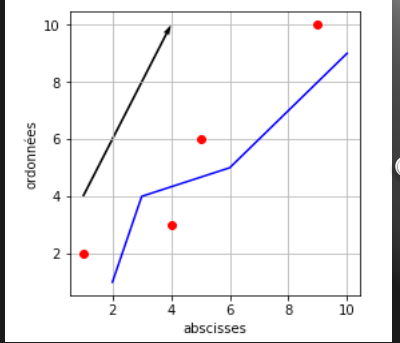
- `random` : nombre compris entre 0 et 1 qui suit la loi uniforme.
- `uniform(min,max)`: nombre compris entre `min` et `max` qui suit la loi uniforme.
- `randint(min,max)` : nombre aléatoire entier compris entre `min` et `max`.
- `choice` : choisir un élément au hasard dans une séquence.
- `randrange(debut,fin,pas)` : choisir au hasard un multiple de `pas` compris entre `debut` et `fin`.
- `shuffle(seq)` : mélange aléatoirement les éléments de la séquence `seq`.

```
In [1]: import numpy as np
In [2]: print(np.array([1,2,3,4]))
[1 2 3 4]
In [3]: np.array([[1,2],[3,4]])
Out[3]:
array([[1, 2],
       [3, 4]])
In [4]: print(np.linspace(0,5,5))
[0.   1.25 2.5  3.75 5.  ]
In [5]: print(np.arange(6))
[0 1 2 3 4 5]
In [6]: np.arange(9).reshape(3,3)
Out[6]:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
In [7]: X,Y = [1,2,3,4],[5,6,7,8]
In [8]: print(np.polyfit(X,Y,1))
[1. 4.]
In [9]: f = np.poly1d([5,2])
In [10]: print(f)
5 x + 2
```

Module numpy

- `np.array([a,b,c,d])` : créer une matrice 1x4.
- `np.array([[a,b],[c,d]])` : créer une matrice 2x2.
- `np.linspace(a,b,n)` : créer une matrice 1xn contenant n nombres répartis uniformément sur l'intervalle [a ; b [.
- `np.arange(n)` : créer une liste de nombres entiers de 0 à n-1.
- `np.arange(n).reshape(a,b)` : créer une matrice axb de nombres entiers de 0 à n-1.
- `np.poly1d([a,b,...,n])` : représenter le polynôme de degré n-1 de coefficients *a, b, ..., n*.
- `np.poly1d([a,b,...,n],True)` : représenter le polynôme de degré n *(x - a) (x - b) ... (x - n)* sous forme développée.
- `np.polyfit(X,Y,n)` : créer la liste des coefficients du polynôme de degré n modélisant le nuage de points de coordonnées (X;Y).

```
import matplotlib.pyplot as plt

X,Y = [2,3,6,10],[1,4,5,9]
plt.plot(X,Y,'b-')
plt.plot(Y,X,'ro')
plt.grid()
plt.xlabel('abscisses')
plt.ylabel('ordonnées')
plt.axis('scaled')
plt.quiver(1,4,3,6,scale_units
           ='xy',scale=1)
```



Module matplotlib

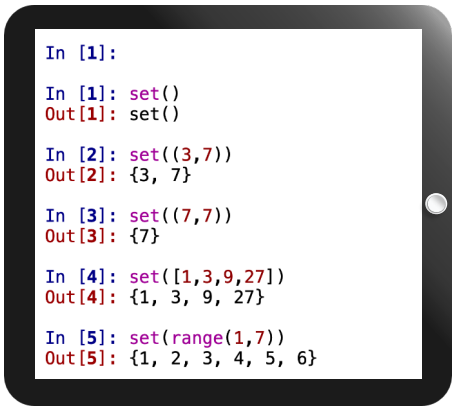
- `plt.plot(X,Y,'ro')` : afficher le nuage de points de coordonnées (X;Y) à l'aide de **ronds rouges**.
- `plt.title('titre')` : afficher un titre au graphique.
- `plt.xlabel('tx')`: afficher une légende sur l'axe des abscisses.
- `plt.ylabel('ty')`: afficher une légende sur l'axe des ordonnées.
- `plt.axis([Xmin,Xmax,Ymin,Ymax])` : définir la fenêtre graphique.
- `plt.grid()` : afficher une grille.
- `plt.show()` : afficher le graphique courant.
- `plt.quiver(X,Y,Vx,Vy,scale_units='xy',scale=1)` : Tracer un vecteur de composantes (Vx,Vy) au point d'application de coordonnées (X;Y) dans le repère étudié.
- `plt.bar(X,Y,width=1)` : afficher un diagramme en barre. (X est une liste contenant les abscisses des coins inférieurs gauches des barres et Y les hauteurs)

L'instruction `plt.plot(X,Y,'chaîne')` offrent plusieurs possibilités de représentation graphique en fonction de la nature de la chaîne de caractères mise en argument :

o	**rond**	**x**	**croix**	**c**	**cyan**
+	**+**	*****	**étoiles**	**m**	**magenta**
-	**ligne continue**	**b**	**bleu**	**y**	**jaune**
–	**tirets**	**r**	**rouge**	**k**	**noir**
:	**pointillés**	**g**	**vert**	**w**	**blanc**

Complément : utilisation de la fonction set

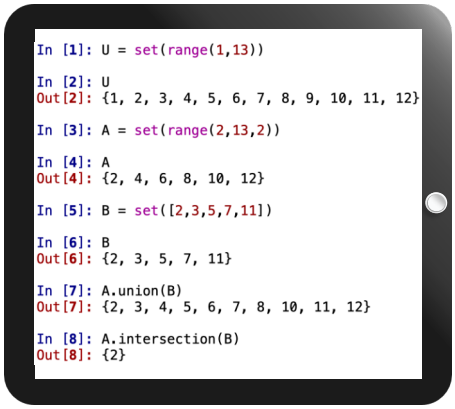
La fonction `set` en Python permet de créer des ensembles correspondant à une collection non ordonnée sans élément dupliqué qui peut être utilisée dans différents cas. On pourra faire une analogie avec cette structure de données équivalente à des ensembles en mathématiques dans le cadre du module transversal « Vocabulaire ensembliste et logique ».



```
In [1]:

In [1]: set()
Out[1]: set()

In [2]: set((3,7))
Out[2]: {3, 7}

In [3]: set((7,7))
Out[3]: {7}

In [4]: set([1,3,9,27])
Out[4]: {1, 3, 9, 27}

In [5]: set(range(1,7))
Out[5]: {1, 2, 3, 4, 5, 6}
```

Un exemple : mobilisation de la fonction `set`

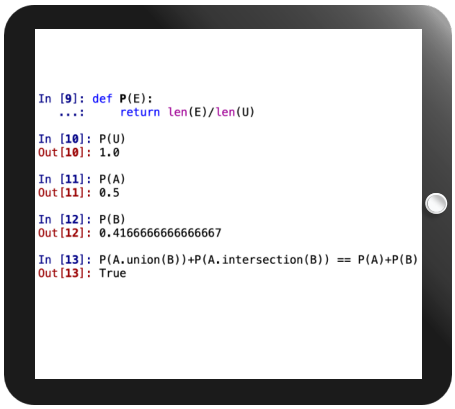
- l'ensemble vide se contruit en ne donnant aucun argument à la fonction : `set()`.
- `set` peut convertir un couple en paire ou en singleton.
- `set` peut construire un ensemble à partir d'une liste.
- `set` peut convertir un range en un ensemble.



```
In [1]: U = set(range(1,13))

In [2]: U
Out[2]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

In [3]: A = set(range(2,13,2))

In [4]: A
Out[4]: {2, 4, 6, 8, 10, 12}

In [5]: B = set([2,3,5,7,11])

In [6]: B
Out[6]: {2, 3, 5, 7, 11}

In [7]: A.union(B)
Out[7]: {2, 3, 4, 5, 6, 7, 8, 10, 11, 12}

In [8]: A.intersection(B)
Out[8]: {2}
```

Utilisation possible en Probabilités :

- Pour l'univers de probabilité obtenu en lançant un dé à 12 faces , on peut écrire `U = set(range(1,13))`
- On considère l'événement A : « le résultat est pair »
  `A = set(range(2,13,2))`
- On considère l'événement B : « le résultat est un nombre premier »
  `B = set(range(1,13,2))`
  - Pour calculer l'événement « le résultat est pair ou premier » :
    `A.union(B)`
- Pour calculer l'événement « le résultat est à la fois pair et premier »,
  `A.intersection(B)`



```
In [9]: def P(E):
   ...:     return len(E)/len(U)

In [10]: P(U)
Out[10]: 1.0

In [11]: P(A)
Out[11]: 0.5

In [12]: P(B)
Out[12]: 0.4166666666666667

In [13]: P(A.union(B))+P(A.intersection(B)) == P(A)+P(B)
Out[13]: True
```

Dans le cas équiprobable, on définit la probabilité d'un événement comme une fonction qui à chaque événement associe un nombre compris entre 0 et 1.

- On peut calculer les probabilités `P(U)`, `P(A)` et `P(B)`.
- On peut vérifier la formule :
  `P(A.union(B))+P(A.intersection(B)) == P(A)+P(B)`

Ont participé à la rédaction de ce document

Raphaël BOUCLY
Emelyne DE-JAEGHERE
Ludovic DIANA
Jérôme LENOIR
Véronique LEPAÎTRE
Paolo RIDOLFI