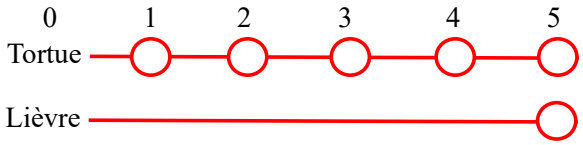
# Le lièvre et la tortue

**Objectif**

Règle du jeu :
à chaque tour, on lance un dé. Si le 6 sort, alors le lièvre gagne la partie, sinon la tortue avance d'une case.
La tortue gagne quand elle arrive à la position 5 (après avoir avancé de 5 cases).

L'objectif est de déterminer si le jeu est à l'avantage du lièvre ou de la tortue.

## Partie A quelques parties




Jouez quelques parties. A votre avis, le jeu est-il à l'avantage du lièvre ou de la tortue ?

## Partie B programme en langage Python

```
from random import randint

def partie():     # la fonction renvoie True si le lièvre gagne la partie

    PositionTortue = 0
    PositionLievre = 0

    while PositionTortue != 5 and PositionLievre != 5 :

        d = ..............     # on lance le dé

        if d==6:
            PositionLievre = ...
        else:
            PositionTortue = ........................

    if PositionLievre == 5 :
        return True
    else:
        return False
```

1. Compléter et saisir le programme ci-dessus.

2. Modifier le programme pour simuler 1000 parties.

3. Modifier le programme pour compter le nombre de parties gagnées par le lièvre et afficher la fréquence associée.

## Partie C calcul des probabilités

Déterminer la probabilité que la tortue gagne et la probabilité que le lièvre gagne.

Nom : Devoir maison 1

# Conjecture de Syracuse

**Introduction**

On considère l'algorithme ci-dessous où $n$ est un entier naturel avec $n > 1$.

```
1  n entier > 1
2
3  Tant que n > 1
4      Si n pair n ← n/2
5      Sinon n ← 3 × n + 1
6      Afficher n
```

**Exemple :**

si la valeur de départ de $n$ est 10, on obtient

l'affichage de la séquence de nombres : 5 ; 16 ; 8 ; 4 ; 2 ; 1.

## PARTIE A étude préliminaire

1. Inscrire dans le tableau les séquences affichées pour chaque valeur de départ.

| Départ | Séquence des valeurs affichées en sortie | | | | | | | | | | | | | | $T_{vol}$ | $A_{max}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 34 | 17 | 52 | 26 | 13 | 40 | 20 | 10 | 5 | 16 | 8 | 4 | 2 | 1 | | |
| 3 | | | | | | | | | | | | | | | | |
| 12 | | | | | | | | | | | | | | | | |
| 32 | | | | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | | | | |

2. Que remarque-t-on pour chaque séquence ? .....

3. Pour certaines valeurs de départ, les séquences seront uniquement constituées de nombres pairs avant d'atteindre 1. Sous quelle forme peut-on écrire les nombres de départ associés à de telles séquences ?

.....

4. Quelles sont les séquences uniquement constituées de nombres impairs ?

.....

**Définiton**

⋆ On appelle **temps de vol** de la séquence et on note $T_{vol}$, le nombre de valeurs affichées en sortie.
Pour notre exemple, si $n = 10$, $T_{vol} = 6$ car 6 valeurs sont affichées.

⋆ On appelle **altitude maximale** et on note $A_{max}$ la plus grande valeur de la séquence.
Pour $n = 10$, $A_{max} = 16$.

5. Compléter les deux dernières colonnes du tableau $T_{vol}$ et $A_{max}$.

## PARTIE B programmation Python

1. On a saisi sur colaboratory une fonction `syracuse` permettant d'obtenir la séquence associée au choix d'un entier de départ $n$. Ouvrir le fichier intitulé **syracuse** partagé et vérifier les séquences affichées en appelant `syracuse(11)` et `syracuse(32)`.

2. Finir de commenter le programme après les signes `#` :

```
1  def syracuse(n):
2    while n>1:
3      if n%2==0:       #
4        n = n//2       # division entière par 2
5      else:            #
6        n = 3*n+1      #
7      print(n)
```

3. On souhaite utiliser une fonction `tempsvol` qui prend en paramètre un entier `n` et qui renvoie le temps de vol de la séquence de premier terme `n`.

```
1  def tempsvol(n):
2    T=0                # On initialise le temps de vol à 0
3    while n>1:
4      T= ......       # à chaque passage dans la boucle, on incrémente T
5      if n%2==0:
6        n = n//2
7      else:
8        n = 3*n+1
9    return(T)          # la fonction renvoie la valeur finale de T
```

Compléter la ligne 4 de la fonction `tempsvol` sur le fichier colaboratory et exécuter la cellule.

4. Vérifier que le temps de vol si $n = 27$ vaut 111. (Appeler `tempsvol(27)` et exécuter)

5. On considère le programme ci-dessous :

```
1  for i in range(2,50):          # Boucle pour i allant de 2 à 49
2    print("Pour le nombre de départ ",i," le temps de vol est: ",tempsvol(i))
```

Exécuter ce programme et observer les valeurs obtenues, que peut-on dire du temps de vol associé à 27 ?

.....

6. Modifier le dernier programme pour qu'il affiche les temps de vol pour $n$ allant de 2 à 100 inclus. A-t-on constaté un nouveau record de temps de vol ? .....

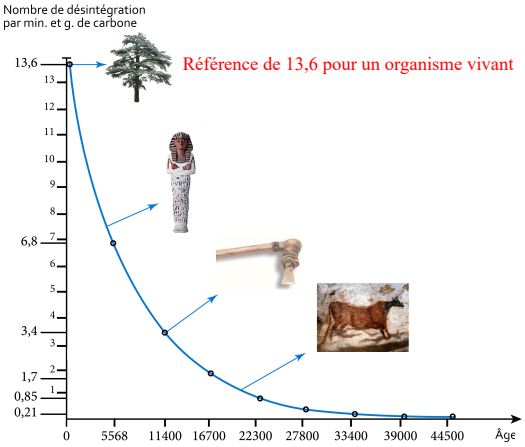
7. Si le nombre de départ $n$ s'écrit sous la forme $2^k$ où $k \in \mathbb{N}$ avec $k > 1$, quel sera le temps de vol ?

.....

# Datation au carbone 14

**Introduction**

La méthode de datation au carbone 14 est utilisée par les archéologues pour estimer l'âge d'objets constitués de matière organique. On sait qu'un gramme de carbone pur extrait d'un organisme vivant présente une activité due au carbone 14 de 13,6 désintégrations par minute. A la mort de l'organisme, il n'y a plus d'absorption de carbone, par contre le carbone 14 absorbé durant la vie étant radioactif se désintègre progressivement, son activité décroit de $1,2$ % par siècle. Pour dater un objet, on mesure l'activité du carbone 14 qu'il contient encore. La mesure de cette activité résiduelle permet de calculer l'âge de l'échantillon.




## Partie A Modélisation

On note $A_n$ l'activité du carbone 14 par minute et par gramme $n$ siècles après la mort de l'organisme étudié.
Ainsi, d'après l'introduction $A_0 = 13,6$ désintégrations par gramme et par minutes et $(A_n)$ décroit de $1,2$ % par siècle.

1. A quel coefficient multiplicateur correspond une diminution de $1,2$ % ?

   .....

2. Calculer $A_1$ et $A_2$, activités du carbone 14 un siècle et deux siècles après la mort d'un organisme :

   .....

3. Quelle est la nature de la suite $(A_n)_{n\geqslant 0}$ ?

   .....

4. Exprimer $A_n$ en fonction de $n$ :

   .....

5. Calculer l'activité du carbone 14 d'un organisme mort il y a 800 ans :

   .....

## Partie B Tableur

1. Reproduire judicieusement la feuille de calcul ci-dessous :

| | **A** | **B** |
|---|---|---|
| **1** | n (siècles) | An |
| **2** | 0 | 13,6 |
| **3** | 1 | |
| **4** | 2 | |
| **5** | 3 | |
| ⋮ | ⋮ | ⋮ |
| **502** | 500 | |

2. Quelle formule faut-il saisir dans la cellule **B3** et étendre vers le bas pour obtenir l'activité du carbone 14 pour chaque siècle écoulé depuis la mort d'un organisme ?

. . . . . .

3. En 1991, deux randonneurs ont découvert dans les Alpes un corps momifié dans la glace. Sur cette momie baptisée Otzi, on a mesuré une activité du carbone 14 de 7,16 désintégrations par minute et par gramme.
A l'aide des colonnes **A** et **B**, donner un encadrement de l'âge de la momie d'une amplitude d'un siècle :

. . . . . < âge de la momie < . . . . . .

4. On a trouvé un morceau de bois de renne dans la grotte de Lascaux. On a mesuré une activité du carbone 14 de 1,44 désintégrations par minute et par gramme sur ce fragment. Donner un encadrement de l'âge du fragment :

. . . . . < âge du fragment < . . . . . .

## Partie C Algorithme et programme Python

On considère l'algorithme suivant :

```
1  fonction dater(mesure) :
2      n ← 0
3      A ← 13,6
4      tant que A > mesure :
5          n ← n + 1
6          A ← 0,988 × A
7      renvoyer n
```

1. À quoi correspond la valeur $n$ renvoyée en sortie de cet algorithme ?

. . . . . .

. . . . . .

2. Expliquer la ligne 6 de l'algorithme :

. . . . . .

. . . . . .

3. Traduire cet algorithme en un programme en langage Python, quel résultat renvoie l'appel *dater*(1.44) ?

. . . . . .

4. Modifier le programme pour obtenir l'âge d'Otzi : . . . . . .

# Primus et Doblus

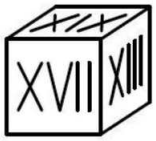
**Introduction**

Primus et Doblus sont deux citoyens romains qui passent le plus clair de leur temps à jouer.

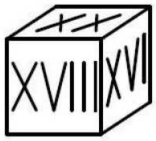
Chacun d'eux a inventé un dé cubique équilibré numéroté de façon inhabituelle :

⋆ le dé de Primus a pour numéros les nombres premiers consécutifs à partir de 13 ;

⋆ celui de Doblus a pour numéros les nombres pairs à partir de 16.



Dé de Primus



Dé de Doblus

Une partie acharnée s'engage, à chaque lancer, celui qui obtient le plus petit nombre doit donner un sesterce à son adversaire. Chacun d'eux est persuadé que son dé est plus performant que celui de l'autre.

On souhaite déterminer si l'un des joueurs a plus de chance de s'enrichir.

## Partie A Simulation à l'aide d'un programme Python

1. On a réalisé le programme Python ci-dessous :

```
1  from random import randint
2
3  primus=[13,17,19,23,29,31]
4
5  doblus=[16,18,20,22,24,26]
6
7  def partie():
8      d1=randint(0,5)       # on choisit un rang au hasard entre 0 et 5
9      d2=randint(0,5)
10
11     if primus[d1]>doblus[d2]:
12         return True            # True si Primus gagne
13     else:
14         return False
15
16
17 def pleindeparties():
18     compteur .......              # Nombre de victoires de Primus
19     for i in range(.........):
20         if partie():
21             ..................    # On incrémente le nombre de victoires de Primus
22     return compteur/100000
```

Expliquer les lignes 11 à 14 du programme.

. . . . . .

. . . . . .

. . . . . .

2. Ouvrir le fichier fourni et compléter la fonction *pleindeparties*.

3. Exécuter l'ensemble du programme et appeler plusieurs fois la fonction *pleindeparties*.

4. Quel joueur semble favorisé par ce jeu ? .....

## Partie B Calcul de probabilités

D'après la loi des grands nombres, si on répète une expérience aléatoire un grand nombre de fois, la fréquence observée d'un événement est proche de la probabilité de cet événement.

La probabilité que primus remporte la partie est donc proche de la fréquence observée à l'aide du programme Python. Dans cette partie, nous allons calculer la valeur exacte de cette probabilité.

1. Compléter le tableau ci-dessous :

| | 13 | 17 | 19 | 23 | 29 | 31 |
|---|---|---|---|---|---|---|
| 16 | | | | | | |
| 18 | | | | | | |
| 20 | | | | | | |
| 22 | | | | | | |
| 24 | | | | | | |
| 26 | | | | | | |

2. En déduire qui de Primus ou Doblus a la plus grande probabilité de s'enrichir.

.....

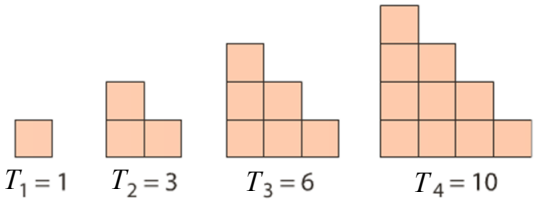
.....

.....

# Nombres triangulaires

**Introduction**

On s'intéresse aux nombres triangulaires. On a schématisé les quatre premiers nombres triangulaires ci-dessous :




## Partie A Modélisation

1. Réaliser un schéma pour déterminer le cinquième nombre triangulaire $T_5$ obtenu à la cinquième étape.

   $T_5 = ...$

2. Expliquer comment on peut obtenir $T_2$ à partir de $T_1$, $T_3$ à partir de $T_2$, $T_4$ à partir de $T_3$ :

   .....

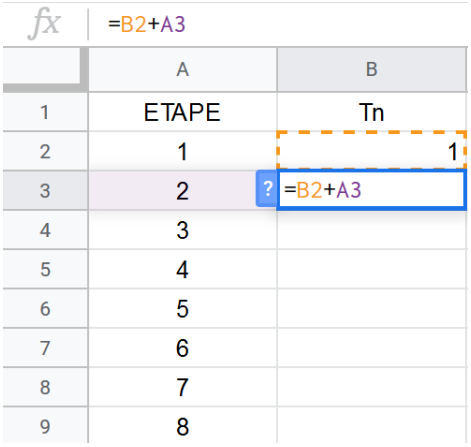
   .....

   .....

## Partie B Tableur

1. La colonne A de la feuille de calcul ci-dessous contient les numéros d'étapes de 1 à 200. Expliquer la formule saisie dans la cellule B3.

   .....




2. Reproduire judicieusement cette feuille de calcul et étirer la formule saisie en B3 jusqu'à la ligne 201.

3. Lire la valeur de $T_{200}$ : .....

## Partie C Algorithmique et programmation

On considère l'algorithme ci-dessous :

```
1  fonction f()
2      n ← 1
3      T_1 ← 1
4      Tant que T_n < 1000000
5          n ← n + 1
6          T_n ← T_n + n
7      renvoyer n
```

1. A quoi correspond la valeur renvoyée par l'algorithme ci-dessus ?

   .....

2. Saisir puis exécuter le programme Python ci-dessous :

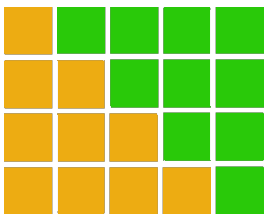
```
1  def f():
2      n=1
3      T=1
4      while T<1000000:
5        n=n+1
6        T=T+n
7      return n
```

3. A partir de quelle valeur de $n$ a-t-on $T_n \geqslant 1\ 000\ 000$ ?

   .....

## Partie D une formule

En considérant le schéma ci-dessous, Kazuki affirme que $T_4 = \dfrac{4 \times 5}{2}$ et qu'il peut calculer la valeur de n'importe quel nombre triangulaire sans outils informatiques.



1. Comment calculera-t-il $T_5$ et $T_{99}$ ?

   .....

   .....

   .....

2. Donner une formule permettant de calculer $T_n$ pour tout entier $n \geqslant 1$ :

   .....

   .....

# Paradoxe du Duc de Toscane

**Introduction**

A la cour de Florence, un jeux de société faisait intervenir la somme des numéros sortis lors du lancer de trois dés. Le Duc de Toscane avait remarqué que la somme 10 était obtenue légèrement plus souvent que la somme 9. Le Duc est intrigué par cette observation car il constate qu'il y a autant de façons d'écrire 10 que 9 comme sommes de trois entiers compris entre 1 et 6 :

$10 = 6 + 3 + 1$ ; $10 = 6 + 2 + 2$ ; ... (6 décompositions)

$9 = 6 + 2 + 1$ ; $9 = 5 + 3 + 1$ ; ... (6 décompositions)

Le Duc de Toscane demande à Galilée d'élucider ce paradoxe. Galilée rédigea vers 1620 un mémoire sur les jeux de dés pour répondre à la demande du Duc de Toscane.

On considère l'expérience aléatoire consistant à lancer trois dés discernables équilibrés et à noter la somme des faces obtenues.

## Partie A Simulation sur tableur

1. Reproduire la feuille de calcul ci-dessous :

| | **A** | **B** | **C** | **D** | **E** | **F** |
|---|---|---|---|---|---|---|
| **1** | dé 1 | dé 2 | dé 3 | somme | | |
| **2** | | | | | | |
| **3** | | | | | fréq 9 : | |
| **4** | | | | | fréq 10 : | |
| **5** | | | | | | |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| **1001** | | | | | | |

2. En utilisant judicieusement les formules $= ALEA.ENTRE.BORNES(1;6)$ et $= SOMME(A2 : C2)$ réaliser une simulation de 1000 expériences.

3. Dans les cellules $F3$ et $F4$, saisir une formule permettant de calculer la fréquence observée de la somme 9 et de la somme 10. Utiliser la formule $= NB.SI(D2 : D1001; 9)$

4. On a regroupé dans le tableau ci-dessous les observations de 10 élèves :

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Fréq 9** | 0,110 | 0,122 | 0,108 | 0,115 | 0,101 | 0,126 | 0,121 | 0,110 | 0,115 | 0,116 |
| **Fréq 10** | 0,129 | 0,118 | 0,134 | 0,127 | 0,140 | 0,121 | 0,123 | 0,118 | 0,134 | 0,114 |

Commenter ces observations :

.....

.....

5. Comment pourrait-on obtenir des observations plus décisives ?

.....

.....

## PARTIE B Simulation à l'aide d'un programme Python

Dans cette partie, nous allons réaliser un programme Python permettant de simuler l'expérience aléatoire 100 000 fois.

1. Compléter et saisir le programme ci-dessous :

```
from random import randint

def toscane(n):      # Le paramètre n correspond au nombre de lancers

    compteurneuf=0
    compteurdix=0

    for i in range(n) :
        d1= ............
        d2= ............
        d3= ............
        s= .........        # Calcul de la somme des 3 dés
        if s==9:
            ..........................
        if s==10:
            ..........................
    return "Fréquence du 9:",compteurneuf/n,"Fréquence du 10:",compteurdix/n
```

2. Exécuter 10 fois ce programme et compléter le tableau ci-dessous en arrondissant au millième :

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Fréq 9** | | | | | | | | | | |
| **Fréq 10** | | | | | | | | | | |

3. Conclure :. . . . . .

## PARIE C Calcul des probabilités

1. On considère l'expérience consistant à lancer trois dés discernables équilibrés et à noter la somme des faces obtenues. On associe à chaque issue de l'expérience un triplet ordonné $(a;b;c)$ où $a$, $b$ et $c$ sont des entiers de 1 à 6. Calculer le cardinal de l'univers des possibles de cette expérience aléatoire.

. . . . . .

2. On dénombre 6 issues équiprobables correspondant à la somme $1+3+6=10$. Compléter le tableau et calculer la probabilité d'obtenir une somme égale à 10.

. . . . . .

| 6 issues pour $1+3+6=10$ | | | 6 issues pour $1+4+5=10$ | | | 3 issues pour $2+2+6=10$ | | | ... issues pour $2+3+5=10$ | | | ... issues pour $2+4+4=10$ | | | ... issues pour $3+3+6=10$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| d1 | d2 | d3 | d1 | d2 | d3 | d1 | d2 | d3 | d1 | d2 | d3 | d1 | d2 | d3 | d1 | d2 | d3 |
| 1 | 3 | 6 | 1 | 4 | 5 | 2 | 2 | 6 | | | | | | | | | |
| 1 | 6 | 3 | 1 | 5 | 4 | 2 | 6 | 2 | | | | | | | | | |
| 3 | 1 | 6 | 4 | 1 | 5 | 6 | 2 | 2 | | | | | | | | | |
| 3 | 6 | 1 | 4 | 5 | 1 | | | | | | | | | | | | |
| 6 | 1 | 3 | 5 | 1 | 4 | | | | | | | | | | | | |
| 6 | 3 | 1 | 5 | 4 | 1 | | | | | | | | | | | | |

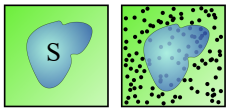
(On montre de la même façon que la probabilité d'obtenir une somme égale à 9 vaut : $\frac{25}{216}$)

3. Conclure : . . . . . .

# Méthode de Monte-Carlo

**Introduction**

La méthode de Monte-Carlo est une méthode d'approximation de l'aire d'une surface basée sur une approche probabiliste. On souhaite estimer l'aire de la surface $S$ ci-dessous incluse dans un rectangle d'aire connue.




On considère l'expérience aléatoire consistant à définir $N$ points aléatoires uniformément répartis sur la surface du rectangle. La probabilité qu'un point aléatoire soit inclus dans la surface $S$ est alors $p = \dfrac{Aire\ de\ S}{Aire\ du\ rectangle}$.

Lorsque $N$ prend de grandes valeurs, la fréquence observée des points inclus dans $S$ est proche de $p$ donc :

$$\text{le nombre } f_{\text{observée}} = \frac{Nombres\ de\ points\ inclus\ dans\ S}{Nombre\ total\ de\ points} \text{ est proche de } \frac{Aire\ de\ S}{Aire\ du\ rectangle}.$$

Une estimation de l'aire de $S$ est donc donnée par : $Aire\ du\ rectangle \times f_{\text{observée}}$.

## Partie A Testons la méthode de Monte-Carlo dans un cas particulier

Nous allons pour cela estimer l'aire d'une surface simple par la méthode de Monte-Carlo et confronter l'approximation obtenue à la valeur exacte de l'aire que nous savons calculer.

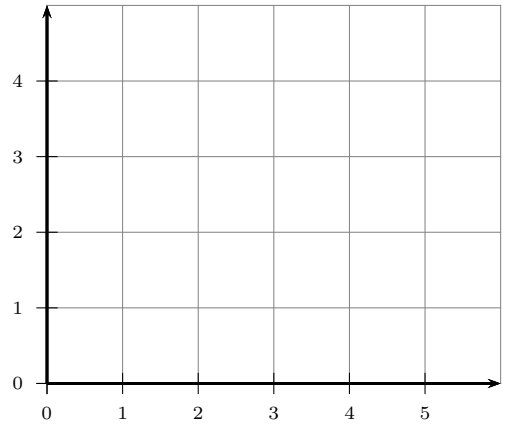
1. On considère la fonction $f$ définie sur l'intervalle $[0;6]$ par $f(x) = \dfrac{1}{2}x$. Représenter la courbe $C_f$ dans le repère orthonormé ci-contre.

2. Hachurer la surface $S_1$ délimitée par $C_f$, l'axe des abscisses et la droite d'équation $x = 6$.

3. Calculer l'aire de la surface hachurée.

.....

4. Expliquer ce que fait la fonction **montecarlo()** dans le programme ci-dessous.




.....

.....

```
from random import*

def f(x):
    return x/2

def montecarlo():
    x=uniform(0,6)      # abscisse aléatoire dans l'intervalle [0;6]
    y=uniform(0,5)      # ordonnée aléatoire dans l'intervalle [0;5]
    if y<f(x):
        return True
    else:
        return False
```

5. On souhaite appeler 1000 fois la fonction **montecarlo()** et compter le nombre de points inclus dans la surface étudiée. Compléter et saisir le programme ci-dessous :

```
from random import*

def f(x):
    return x/2

def montecarlo():
    x=uniform(0,6)      # abscisse aléatoire dans l'intervalle [0;6]
    y=uniform(0,5)      # ordonnée aléatoire dans l'intervalle [0;5]
    if y<f(x):
        return True
    else:
        return False

compteur=0

for i in range(1000):
    if montecarlo()== ..... :
        compteur= ..............
print(compteur/1000)
```

6. A quoi correspond l'affichage obtenu en sortie de programme ?

. . . . . .

7. D'après l'introduction, une estimation de l'aire de $S_1$ est donnée par : *Aire du rectangle* $\times f_{\text{observée}}$.
Modifier le programme pour qu'il affiche une estimation de l'aire de $S_1$.

8. Calculer |valeur affichée $- 9$| et interpréter le résultat.

. . . . . .

. . . . . .

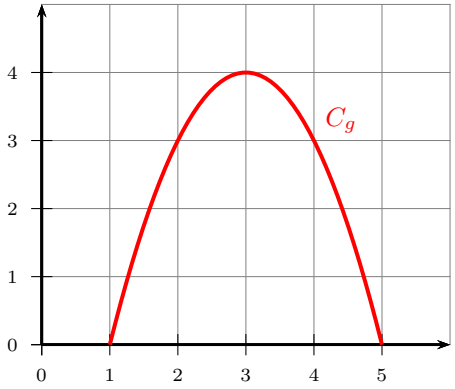
## Partie B Application de la méthode dans un cadre général

On souhaite à présent appliquer la méthode de Monte-Carlo pour estimer l'aire d'une surface que nous ne savons pas calculer. On considère la fonction $g$ définie sur $[1;5]$ par $g(x) = -x^2 + 6x - 5$.
On note $S_2$ la surface délimitée par $C_g$ et l'axe des abscisses.

1. Hachurer la surface $S_2$.

2. Modifier le programme pour obtenir une estimation de l'aire de $S_2$.

3. Le programme ci-dessous permet d'afficher un point rouge et un bleu dans un repère. En vous inspirant de cet exemple, compléter le programme de la question 2 pour obtenir un graphique où les points inclus dans la surface $S_2$ sont affichés en rouge et les autres en bleu.




```
from matplotlib.pyplot import*          # module graphique

plot(1,2,'ro')        # point de coordonnées (1;2) motif 'o' de couleur rouge
plot(2,3,'bo')        # point de coordonnées (2;3) motif 'o' de couleur bleue

savefig("figure")         # sauvegarde et affichage du graphique
```

# Suite de Fibonacci

**Introduction**

La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent. Elle doit son nom à Leonardo Fibonacci qui, dans l'ouvrage Liber abaci publié en 1202, décrit la croissance d'une population de lapins. Les nombres de Fibonacci apparaissent parfois dans la nature, par exemple, le nombre de pétales de la marguerite appartient souvent à la suite de Fibonacci.

$$\begin{cases} f_0 = 0 \\ f_1 = 1 \\ f_{n+2} = f_{n+1} + f_n \quad \textit{pour} \quad n \in \mathbb{N} \end{cases}$$

## Partie A questions préliminaires

1. On considère l'algorithme en langage naturel ci-dessous :

**Initialisation :** $A = 0$
$B = 1$
**Traitement :** Répéter 4 fois :
$C \leftarrow A$
$A \leftarrow B$
$B \leftarrow C + B$
**Sortie :** Afficher $B$

compléter le tableau précisant la valeur des variables à chaque itération.

| Variables | $A$ | $B$ | $C$ |
|---|---|---|---|
| état initial | 0 | 1 | |
| première itération | | | |
| deuxième itération | | | |
| troisième itération | | | |
| quatrième itération | | | |

2. Quelle est la valeur affichée en sortie d'algorithme, à quoi correspond-elle ?

. . . . . .

3. Écrire, en utilisant une boucle bornée, une fonction `fibonacci(k)` qui calcule $f_k$, c'est-à-dire qui prend en argument un entier $k \geqslant 0$ et qui renvoie le terme d'indice $k$ de la suite de Fibonacci.

4. Compléter le tableau ci-dessous :

| $f_0$ | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ | $f_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | | | | | | | |

5. Écrire un programme qui détermine le premier nombre de Fibonacci à 4 chiffres.

## Partie B Codage de Fibonacci

Théorème de Zeckendorf :

tout entier naturel $n > 1$ peut être décomposé, de manière unique, comme somme de nombres de Fibonacci distincts et non consécutifs.

Par exemple, $12 = 8 + 3 + 1$. La décomposition de Zeckendorf $12 = f_6 + f_3 + f_2$ est unique.

1. Écrire une fonction `pgf(n)` prenant en argument un entier $n > 1$ et renvoyant le plus grand terme $f_k$ de la suite de Fibonacci vérifiant $f_k \leqslant n$.

2. Montrer l'existence d'une décomposition de Zeckendorf pour tout entier $n > 1$. (Nous admettrons son unicité.)

3. Le codage de Fibonacci est une représentation des entiers naturels non nuls fondée sur la décomposition de Zeckendorf. Si $n = \sum_{i=2}^{k} a_i \times f_i$ est la décomposition de Zeckendorf de l'entier $n$ alors la chaîne de caractères $a_2a_3a_4...a_k$ est son code de Fibonacci.

   Par exemple, "00100101" est le code de Fibonacci de 50 car $50 = 0.f_2+0.f_3+1.f_4+0.f_5+0.f_6+1.f_7+0.f_8+1.f_9$.

   Écrire une fonction `decode` qui prend en paramètre un code de Fibonacci et qui renvoie l'entier $n$ représenté par ce code. Par exemple, `decode("00100101")` devra retourner 50.

4. Écrire une fonction `code` qui prend en paramètre un entier strictement positif et renvoie le code de Fibonacci de ce nombre. Par exemple, `code(50)` retournera la chaîne de caractères "00100101".

# Nombres premiers

**Définiton**

Un nombre premier est un entier naturel qui admet exactement deux diviseurs entiers naturels.

7 admet pour seuls diviseurs 1 et 7, c'est donc un nombre premier.
4 est divisible par 1, 2 et 4, il n'est donc pas premier (un nombre entier qui n'est pas premier est dit composé)

## Partie A test de primalité avec une boucle bornée

1. On considère l'algorithme ci-dessous, traduire cet algorithme en une fonction Python `estprema(n)` qui renvoie `True` si l'entier `n` est premier et `False` sinon.

Fonction estprema(n) :
  Si $n < 2$
    renvoyer False
  Si $n = 2$
    renvoyer True
  Pour $i$ allant de 2 à $n - 1$
    Si $i$ divise $n$
      Renvoyer False
  Renvoyer True

2. Modifier la fonction afin qu'elle renvoie également une décomposition de l'entier `n`, par exemple `estprema(10)` doit renvoyer `False,5,2`.

   Tester la fonction avec un nombre le nombre 50370952483.

## Partie B test de primalité avec une boucle non bornée

1. On considère la fonction Python ci-dessous :

```
1  def estpremb(n):
2    if n<2:
3      return False
4    if n==2:
5      return True
6    d=2
7    while d*d<=n:
8      if n%d==0:
9        return False
10     d=d+1
11   return True
```

   Expliquer et justifier le choix de la condition à la ligne 7 du programme :

   . . . . . .

   . . . . . .

   . . . . . .

2. Saisir le programme et vérifier si le nombre 567373 est premier : . . . . . .

## Partie C une fonction polynôme

1. En 1772, Léonard Euler étudie la fonction polynôme $P$ qui à un entier $n$ associe $P(n) = n^2 + n + 41$.
   Écrire une fonction Python $P(n)$ qui prend un entier $n$ en paramètre et renvoie l'image de $n$ par $P$.

2. En utilisant la fonction `estpremb(n)` de la partie B et une boucle, déterminer le premier entier $n$ dont l'image par $P$ n'est pas un nombre premier.

3. Modifier votre programme pour qu'il affiche les nombres premiers obtenus avant d'atteindre un nombre composé.

## Partie D une question de rang

On considère le programme Python ci-dessous :

```
1  def g():
2      compteur=0
3      n=1
4      while compteur<100:
5          n=n+1
6          if estpremb(n)==True:
7              compteur=compteur+1
8      return compteur
```

1. Expliquer à quoi correspond le nombre renvoyé par la fonction $g$.

   .....

   .....

   .....

   .....

2. En modifiant ce programme, déterminer le 1001 ème nombre premier.

   .....

3. Les nombres de Mersenne sont les entiers de la forme $M_n = 2^n - 1$ ou $n$ est un entier naturel non nul.
   Calculer les nombres de Mersenne $M_3$ et $M_4$.

   .....

4. La propriété « si $2^n - 1$ est premier alors $n$ est premier » a été démontrée.
   A l'aide d'un contre-exemple trouvé grâce à une modification de votre programme, montrer que la réciproque de cette propriété est fausse.

   .....

   .....

   .....