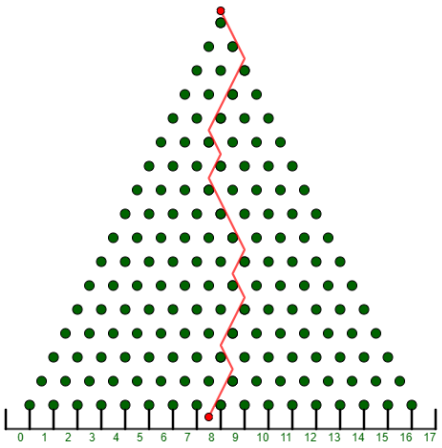
La planche de Galton

Présentation

Dans le programme (spécialité Terminale)

Contenus	**Capacités attendues**
Schéma de Bernoulli : répétition de n épreuves de Bernoulli indépendantes. Loi binomiale B(n, p) : loi du nombre de succès. Expression à l'aide des coefficients binomiaux.	Modéliser une situation par une succession d'épreuves indépendantes, ou une succession de deux ou trois épreuves quelconques. Représenter la situation par un arbre. Calculer une probabilité en utilisant l'indépendance, des probabilités conditionnelles, la formule des probabilités totales. Modéliser une situation par un schéma de Bernoulli, par une loi binomiale. Utilisation de boucle non bornée, utilisation des listes.

Situation déclenchante



Une bille roule à la surface d'une planche inclinée sur laquelle sont disposés des clous en quinconce. La bille passe aléatoirement d'un côté ou de l'autre des clous, et on note à l'arrivée la position de la bille à la sortie de la planche. Cette planche a été inventée par Sir Francis Galton.

Buts à atteindre

1. Écrire une fonction Python permettant de simuler la chute de plusieurs billes. Cette fonction prendra en paramètres le nombre de lignes de clous de la planche, le nombre de billes et donnera en sortie la liste des effectifs des billes associés aux différents numéros de sortie.
2. Représenter graphiquement les résultats obtenus par la simulation précédente.

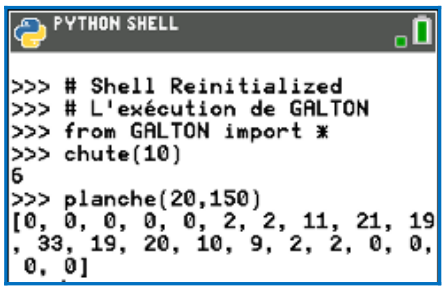
L. DIDIER & R. CABANE

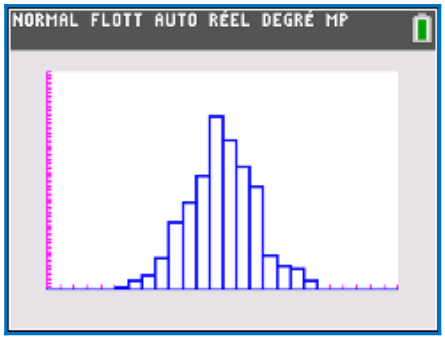
Fiche méthode

Proposition de résolution

On **crée trois fonctions dans ce script** :

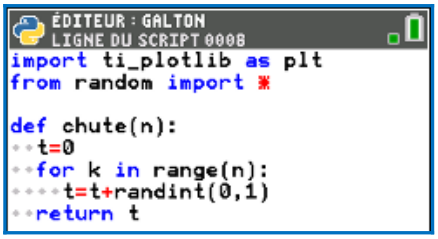
1. Une fonction `chute` qui prend comme argument un entier naturel (qui représente le nombre de lignes de clous) et qui renvoie la position finale d'une bille après sa chute.
2. Une fonction `planche` qui prend comme argument deux entiers naturels (qui représentent respectivement le nombre de lignes de clous de la planche ainsi que le nombre de billes lâchées) et qui renvoie la liste des effectifs associés à chaque numéro de sortie.
3. **TI-83** Une fonction `aff3` qui prend comme argument une liste et qui affiche le diagramme en bâtons associé à cette liste.





Étapes de résolution

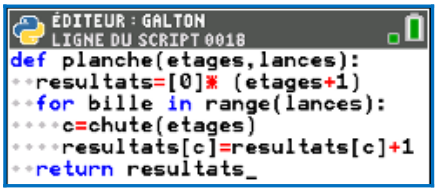
L'instruction `from random import *` permettra d'importer une bibliothèque pour utiliser la fonction `randint(0,1)` qui renvoie un nombre au hasard valant 0 ou 1 (de manière équiprobable)[1]. **La fonction `chute` simule en fait une loi binomiale de paramètres *n* et 0,5.**

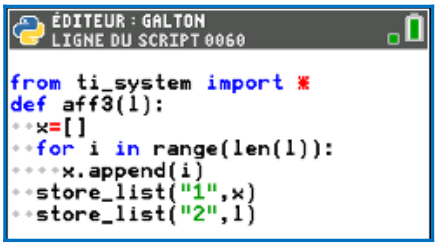


La boucle permet de compter le nombre total de choix à droite lors des *n* rencontres entre la bille et les clous.

Quelques remarques :

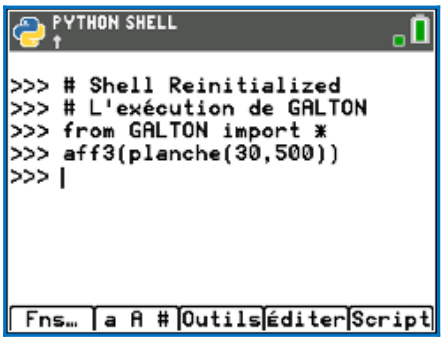
1. L'instruction `L=[0]*k` permet de créer une liste `L` de longueur `k` et d'initialiser chaque élément de la liste à 0.
2. La boucle permet de simuler le lâcher de plusieurs billes et de stocker les différents résultats de sortie dans la liste `resultats`.
3. La fonction `aff3` a pour objectif de préparer la représentation graphique de la liste `l`.
4. L'instruction `from ti_system import *` permettra d'utiliser les fonctions associées à cette bibliothèque, et notamment l'instruction `store_list("1",x)` qui permet d'exporter la liste `x` dans le menu [listes] de la calculatrice sous le nom `L1`.

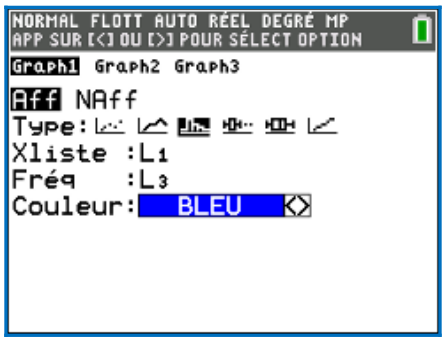


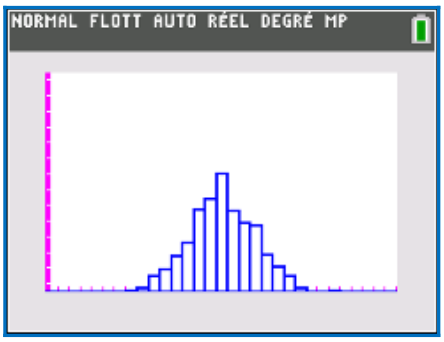


1 L'appel `randint(0,1)` revient à simuler un jeu de « pile ou face ».

On peut donc exécuter la fonction `aff3` avec `planche(30,500)` en paramètre. Une fois l'instruction tapée, il faut quitter l'application Python et aller dans le menu [graph stats] (touche [2nde] et [f(x)]), régler les paramètres comme ci-dessous puis enfin régler la fenêtre d'affichage (touche [fenêtre]).







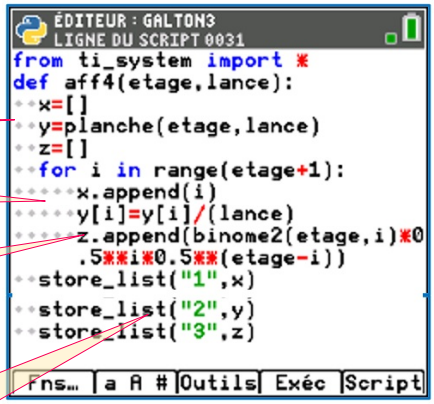
On pourrait aussi faire une comparaison de nos résultats avec le modèle théorique. Voir la fiche « triangle de Pascal »pour déterminer un coefficient binomial à l'aide de la fonction `binome2`.

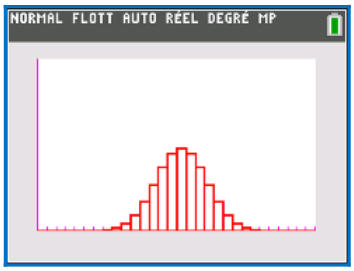
On lance une simulation.

On calcule les pourcentages associés à cette simulation

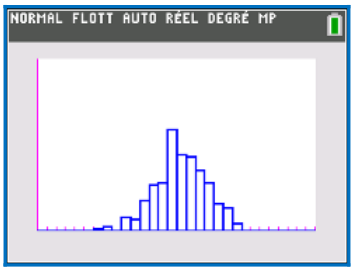
On détermine le modèle à l'aide de la loi binomiale.

On stocke les résultats pour les exporter dans menu graphstats et ainsi pouvoir les représenter.

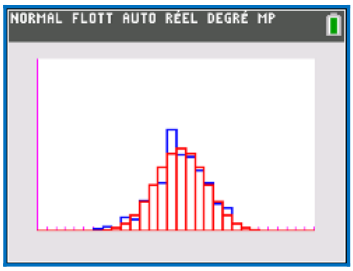




Modèle mathématique



Simulation



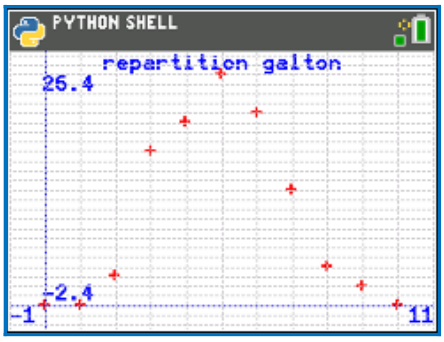
Comparaison

Pour aller plus loin

Approfondissement possible

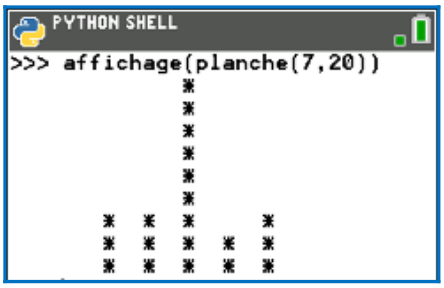
Une deuxième manière de représenter les résultats est d'utiliser la bibliothèque `plotlib`. On peut représenter graphiquement les résultats à l'aide d'un nuage de points. Pour cela, il faut créer une fonction `aff2` qui prend en paramètre une liste `l` et qui va régler les paramètres d'affichage.

```
def aff2(l):
    x=[]
    for i in range(len(l)):
        x.append(i)
    plt.cls()
    plt.auto_window(x,l)
    plt.color(255,0,0)
    plt.scatter(x,l,"+")
    plt.color(0,0,255)
    plt.axes("on")
    plt.color(0,0,255)
    plt.title("repartition galton"
        )
    plt.color(255,255,255)
    plt.grid(1,1,"dot")
    plt.show_plot()
```



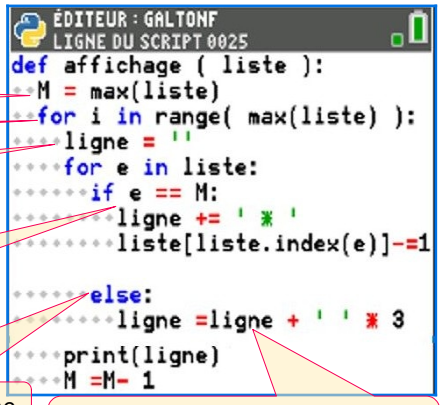
Prolongement possible

Imaginer une fonction prenant en entrée une liste et représentant les billes dans la répartition finale de la planche de Galton.



Codes Python proposés

L'affichage de la calculatrice étant réduit par rapport à celui d'un ordinateur, on exécutera ce code avec de petites valeurs pour que cela reste lisible. On pourra être plus ambitieux sur un écran plus grand.



On repère le maximum.

On va construire le graphique ligne par ligne.

À chaque passage de boucle on initialise la ligne avec l'ensemble vide.

En balayant la liste on va construire les colonnes sur chaque ligne, si on rencontre le maximum de la liste, on met une étoile sur la ligne puis on diminue de 1 ce maximum...

... sinon on met 3 espaces sur la ligne.

On affiche la nouvelle ligne construite à chaque passage de boucle.

Le principe est de repérer la colonne ayant le plus de billes puis de représenter la première ligne à l'aide d'espace et du symbole `*`. Ensuite on passe à la ligne suivante en diminuant de 1 le maximum de la liste.

Indication : étant donné une liste `L`, l'appel `L.index(x)` renvoie l'index du premier élément de la liste `L` ayant pour valeur `x`.