

## Une séance d'introduction à la loi binomiale

La séance décrite ci-après a été improvisée en terminale STI2D, en guise de révision du programme de 1ère sur la loi binomiale : Les élèves ayant unanimement dit qu'ils n'avaient pas vu la loi binomiale<sup>1</sup>, il a fallu au lieu de **révision**, faire rapidement une **introduction** à la loi binomiale... Celle-ci s'est concentrée sur la question suivante :

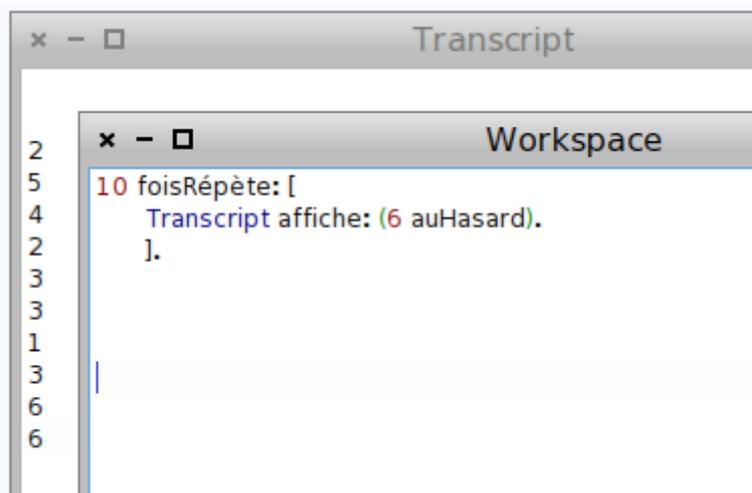
On lance 10 dés ; quelle est la probabilité qu'au moins deux d'entre eux donnent un « 6 » ?

Le but de l'exercice était, on s'en doute, de rappeler l'usage de la calculatrice pour calculer cette probabilité (voir la fin de cet article à ce sujet). Mais il a donc fallu (re)faire le cours de 1ère sur la question, en vitesse. MathsOntologie s'est montré utile

- pour présenter une variable binomiale sur un modèle d'urne,
- et pour programmer vite, grâce à la relative concision du langage ainsi que sa coloration syntaxique.

### I/ Avec MathsOntologie

Pendant le démarrage de l'ordinateur, a été effectuée une discussion sur la façon de simuler le lancer d'un dé avec une urne. Quelques élèves ont trouvé rapidement qu'il suffisait de numéroté 6 boules de 1 à 6 et les mettre dans une urne similaire à celle du Loto<sup>®</sup>. Cependant, comme ci-dessous on va détourner l'urne de sa fonction initiale, et pour accélérer l'activité, on a préféré choisir au hasard un entier inférieur ou égal à 6, avec `6 auHasard` :



```
Transcript
10 foisRépète: [
  Transcript affiche: (6 auHasard).
].

Workspace
1
2
3
3
1
3
6
6
```

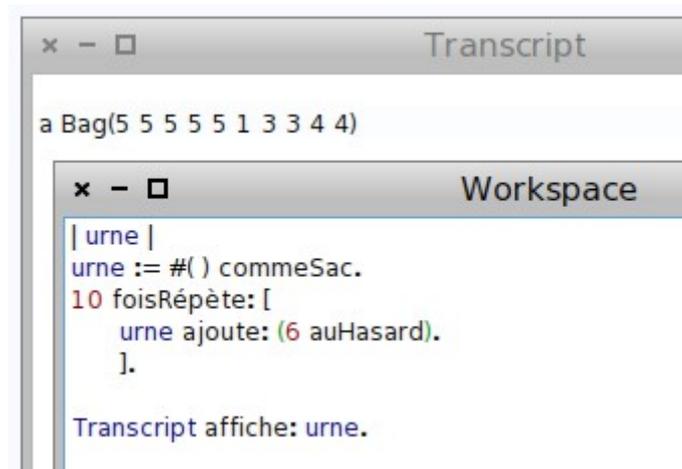
La boucle à 10 passages a pour but de tester l'algorithme, mais également de voir combien de fois le 6 peut sortir sur 10 lancers. Une comparaison entre les résultats obtenus par les élèves permet alors déjà de se faire une idée sur la rareté de l'évènement dont on veut la probabilité.

<sup>1</sup> Ce qui ne semble pas exact puisque quelques-uns d'entre eux se souvenaient qu'«il y a une histoire de boules dans une urne »...

Pour aller plus loin (faire des statistiques sur les lancers), on va ajouter une variable `urne` qui contiendra les résultats des 10 lancers, ajoutés au fur et à mesure desdits lancers. En effet, il revient au même de lancer 10 dés ou de lancer 10 fois un même dé. La variable `urne` est un « sac », et on la crée par l'algorithme suivant :

- créer un tableau vide avec `# ( )` ;
- le transformer en un sac avec `commeSac`.

L'affichage en ligne et trié, permet de mieux voir combien de fois le 6 est sorti parmi les 10 lancers :

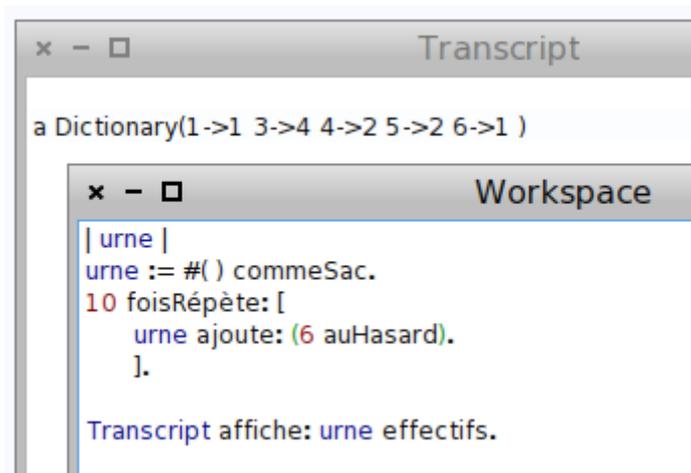


```
Transcript
a Bag(5 5 5 5 1 3 3 4 4)

Workspace
| urne |
urne := #( ) commeSac.
10 foisRépète: [
  urne ajoute: (6 auHasard).
].

Transcript affiche: urne.
```

Pour répéter l'expérience un grand nombre de fois, il faut faire compter les 6 automatiquement par MathsOntologie. Ce qui peut se faire par le tableau d'effectifs de l'urne :



```
Transcript
a Dictionary(1->1 3->4 4->2 5->2 6->1)

Workspace
| urne |
urne := #( ) commeSac.
10 foisRépète: [
  urne ajoute: (6 auHasard).
].

Transcript affiche: urne effectifs.
```

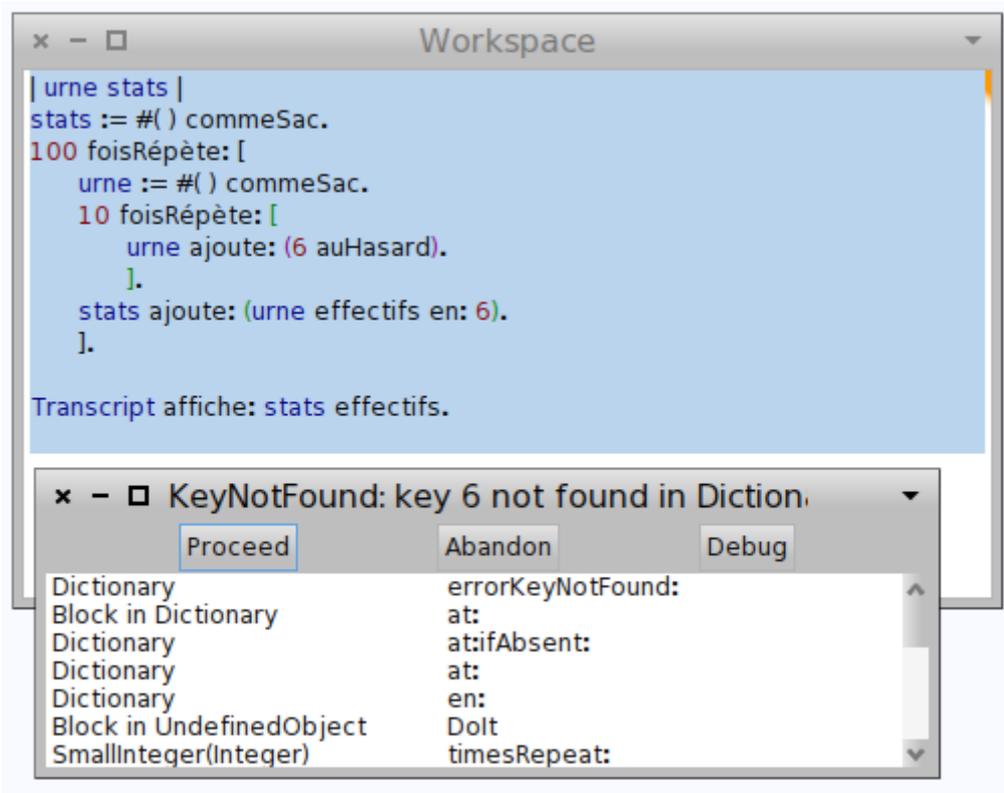
On apprend que ce tableau d'effectif est un « dictionnaire », qui à chaque résultat qui est sorti, associe le nombre de fois qu'il est sorti. Ce genre d'objet peut donc aider à aborder la notion de fonction...

Le `6->1` veut dire que le 6 est sorti une fois, ou que `effectifs en: 6` vaut 1.

On peut donc faire une simulation sur 100 lancers des 10 dés, à l'aide d'une boucle imbriquée dans une autre boucle : On rajoute une autre urne appelée `stats`, et on l'initialise à une urne vide. Ensuite, pour y accumuler les nombres de « 6 » au cours des 100 lancers, on effectue une boucle à 100 passages, où

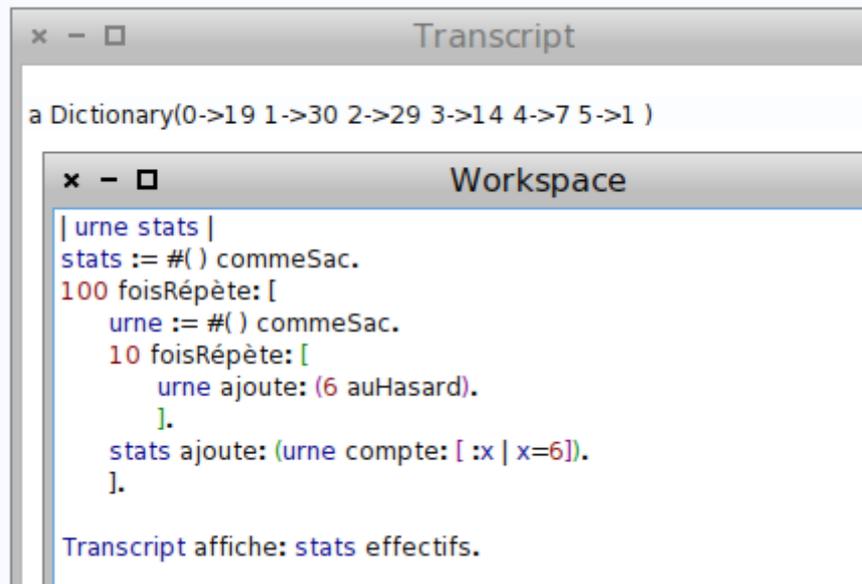
- on vide l'urne des 10 dés ;
- on effectue 10 fois le rajout d'un résultat de lancer de dé ;

Puis on ajoute au sac « stats » le nombre de fois que le 6 apparaît. Mais ceci mène souvent à une erreur, le nombre de fois que le 6 apparaît n'existant pas si le 6 n'est jamais sorti, ce qui, sur 100 répétitions, arrive quasiment à coup sûr. MathsOntologie ouvre le débogueur en signalant que le dictionnaire n'a pas trouvé la clé 6 :



Il faut donc s'y prendre autrement...

On va alors compter les x tels que x égale 6 :



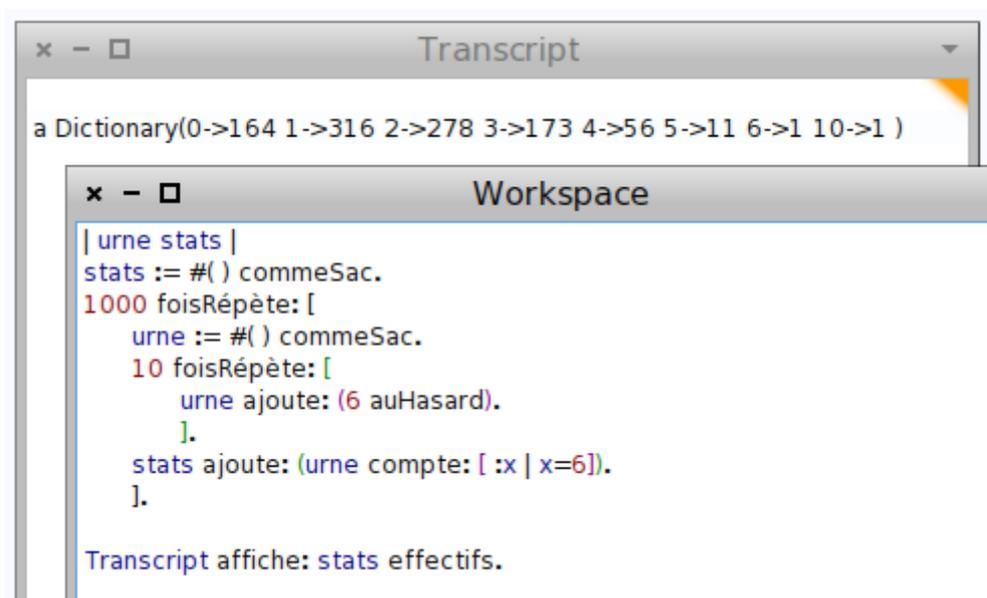
```
Transcript
a Dictionary(0->19 1->30 2->29 3->14 4->7 5->1 )

Workspace
| urne stats |
stats := #( ) commeSac.
100 foisRépète: [
  urne := #( ) commeSac.
  10 foisRépète: [
    urne ajoute: (6 auHasard).
  ].
  stats ajoute: (urne compte: [ :x | x=6]).
].

Transcript affiche: stats effectifs.
```

Parmi les 100 répétitions, il est arrivé une fois que le 6 sorte 5 fois, soit la moitié des dés ! Plusieurs élèves ont atteint ou dépassé les 6 résultats gagnants, ce qui a commencé à donner l'idée que la probabilité voulue n'est peut-être pas si faible que cela.

Il est facile de modifier le script pour faire 1000 expériences au lieu de 100 :



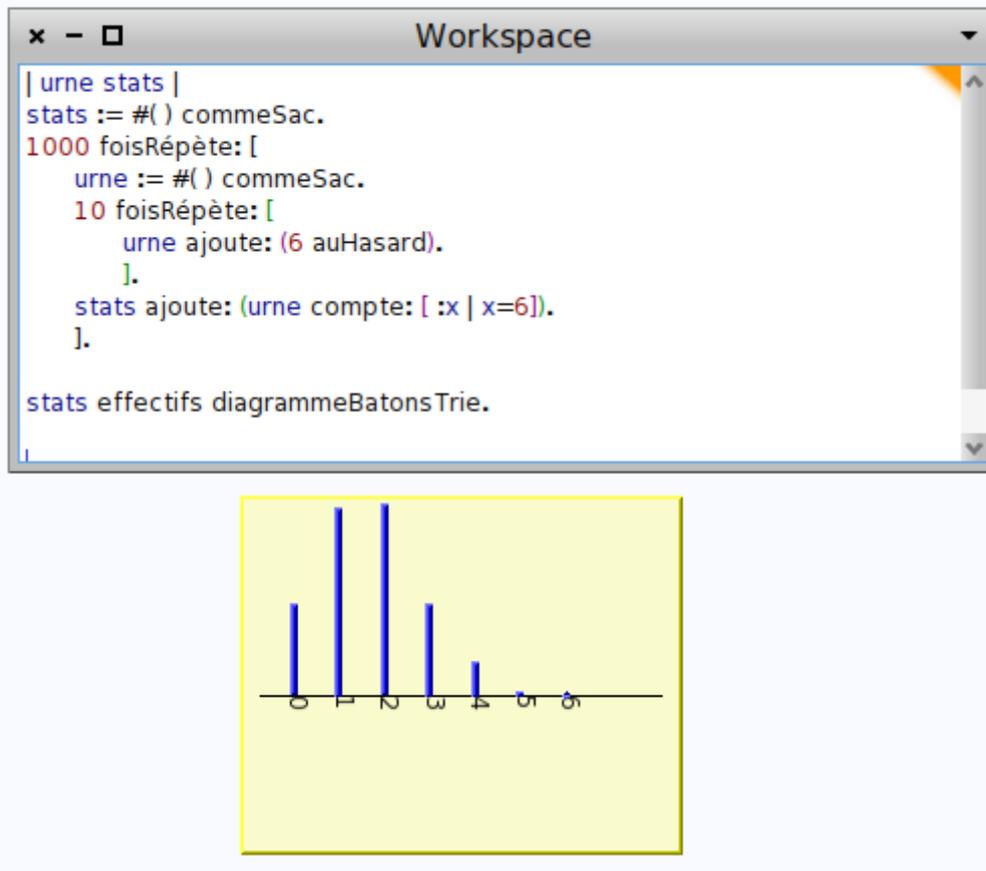
```
Transcript
a Dictionary(0->164 1->316 2->278 3->173 4->56 5->11 6->1 10->1 )

Workspace
| urne stats |
stats := #( ) commeSac.
1000 foisRépète: [
  urne := #( ) commeSac.
  10 foisRépète: [
    urne ajoute: (6 auHasard).
  ].
  stats ajoute: (urne compte: [ :x | x=6]).
].

Transcript affiche: stats effectifs.
```

On voit qu'ici il est arrivé que tous les dés donnent 6, ce qui est quand même rare...

On peut même afficher le diagramme en bâtons :



Il faut une vingtaine de minutes pour la partie « simulation » effectuée jusqu'ici. Le débogage est en effet assez rapide parce que les erreurs de syntaxe sont affichées en rouge. Il s'agit le plus souvent de l'oubli d'un point, de l'insertion d'une espace par exemple entre « fois » et « Répète » (alors que « foisRépète » doit être écrit d'un seul tenant), ou de fautes d'orthographe dans les noms de variables, ce qui fait que les noms lors de l'utilisation ne sont pas identiques aux noms lors de la déclaration (entre deux traits verticaux sur la première ligne).

## II/ Théorie

La loi binomiale est alors définie comme loi du nombre de succès dans la répétition indépendante d'événements identiques à deux issues. Ici, le succès est l'obtention d'un 6 et les paramètres de la loi binomiale sont

1. Le nombre total de répétitions, ici 10 ;
2. La probabilité de succès lors du lancer d'un seul dé, ici 1/6.

Il est alors rappelé qu'il existe un moyen de calculer les probabilités d'une telle loi binomiale, mais que ce moyen n'est pas au programme et que seule l'utilisation de la calculatrice est à connaître.

Il reste alors une vingtaine de minutes pour calculer la probabilité à l'aide de la calculatrice.

## III/ Avec la calculatrice

Pour calculer les probabilités avec la calculatrice, il faut utiliser une « distribution » de probabilité, qui se trouve à l'aide du bouton « distrib » ; la première impression est que des distributions, il y en a beaucoup sur une calculatrice qui s'appelle « stats » :

```
0:STAT DESSIN
1:normalFdp(
2:normalFRép(
3:FracNormale(
4:studentFdp(
5:studentFRép(
6:X²Fdp(
7↓X²FRép(
```

En descendant un peu, on trouve les variables binomiales :

```
8:STAT DESSIN
9↑FRép(
X:binomFdp(
H:binomFRép(
B:poissonFdp(
C:poissonFRép(
D:géomtFdp(
E:géomtFRép(
```

En choisissant la loi (« Fdp ») puis en entrant les paramètres, on obtient une liste de 11 probabilités élémentaires :

```
binomFdp(10,1/6)
(.1615055829 .3...
█
```

C'est cette liste qui s'appelle « loi » de la variable aléatoire, les nombres qui sont dedans étant les probabilités de n'avoir aucun 6, d'avoir un seul 6, d'avoir exactement 2 « 6 » etc.

Par exemple la probabilité d'avoir exactement deux dés qui affichent un 6, est environ 0,29 :

```
binomFdp(10,1/6)
(.1615055829 .3...
binomFdp(10,1/6,
2)
.2907100492
```

Ce qui n'est pas la réponse à la question initiale : Avoir au moins deux « 6 », c'est avoir ou bien 2 « 6 », ou bien 3 « 6 », ou bien 4 « 6 » etc (jusqu'à 10), On peut donc additionner tous les nombres de la liste sauf les deux premiers pour finir l'exercice, Mais il est plus simple de passer par la probabilité du contraire, en constatant déjà qu'il est possible de calculer directement la probabilité d'avoir au maximum deux dés qui affichent « 6 » avec la fonction de répartition « FRép » :

```
(.1615055829 .3...
binomFdp(10,1/6,
2)
.2907100492
binomFRép(10,1/6
,2)
.7752267978
█
```

Pour savoir quelle est la probabilité d'avoir au moins deux fois un « 6 » parmi les 10, il suffit donc de soustraire à 1 la fonction de répartition calculée en 1 :

```
1-binomFRép(10,1  
/6,1)  
■ .5154832525
```

On a donc un peu plus d'une chance sur deux que le nombre de dés affichant « 6 » soit au moins deux...