Chapitre 1

Nombres

I/ Vérité

1°) Affectation

Lorsqu'on souhaite que la variable x soit égale à 3, on place le nombre 3 dans x :

En *Python*, l'affectation se note par un signe "=".

```
x=3
print(x)
```

Des affectations peuvent se succéder :

```
x=3
x=8*7
print(x)
```

2°) Propositions

Une **proposition** est une affirmation qui est soit vraie, soit fausse. Par exemple :

1: « $2+2=4$ » est vraie ;

2: « $-8>5$ » est fausse ;

3: « $2x+1=3$ » n'est pas une proposition.

```
print(2+2==4)
print(-8>5)
print(2*x+1==3)
```

Pour la distinguer de l'affectation, l'égalité est notée par le signe "=" dédoublé ; de plus, « supérieur ou égal » se note >=.

3°) Négation

```
print(not True)
print(not False)
```

Exercice :

Donner les négations des propositions suivantes :

1: « Le triangle ABC est isocèle en A » :.....

2: « Les droites d_1 et d_2 sont parallèles » :.....

3: « L'entier naturel n est impair » :.....

4°) Opérations logiques

a) Conjonction

La conjonction de deux propositions n'est vraie que si chacune des deux propositions est vraie.

```
print(2+2==4 and 2>3)
```

Exercice :

Simplifier les énoncés des conjonctions suivantes :

1: « Le triangle ABC est isocèle et il a un angle de $60°$ » :

.....

2: « Les diagonales de $ABCD$ sont perpendiculaires et elles ont le même milieu » :.....

3: « Les droites d_1 et d_2 sont parallèles et elles passent par le point P » :

.....

b) Disjonction

Dès qu'une des propositions d'une disjonction est vraie, toute la disjonction est vraie :

```
print(2+2==4 or 2>3)
```

II/ Nombres entiers

1°) Opérations en Python

Les opérations sont notées respectivement +, -, * et /. L'élévation à une puissance se note avec l'astérisque dédoublée **. Par exemple pour afficher le cube de 5, on peut faire

```
print(5**3)
```

Exercice :

Que va afficher le programme *Python* ci-dessous ?

```
print(2+3*7)
print(-3**2)
print((-3)**2)
print(5+1/2+3)
print((5+1)/(2+3))
```

2°) Reconnaissance

Python reconnaît les nombres en affichant leur *type*. Ainsi, $\frac{3}{2}$ et « vrai » ne sont pas entiers, alors que 3 est entier :

```
print(type(3/2))
print(type(3))
print(type(True))
```

le type des propositions est une abréviation de *booléen*, d'après le nom de George Boole.

3°) Division euclidienne

a) Arrondi

L'arrondi entier par défaut d'un nombre positif se note *int* en *Python* :

```
print(int(3/2))
print(int(3))
```

b) Quotient

Pour obtenir le quotient euclidien de 34 par 13, il suffit de faire

```
print(34//13)
```

c) Reste

Pour obtenir le reste euclidien de 34 par 13, on utilise le symbole "pourcent" :

```
print(34%13)
```

III/ Nombres réels

1°) Écriture

Le carré de 2 millièmes est 4 millionnièmes :

```
print(0.002**2)
```

L'affichage $4e-6$ veut dire 4×10^{-6} : C'est l'**écriture scientifique** du réel. La présence de l'exposant évite de fixer la place de la virgule : en *Python*, les réels sont dits **flottants**.

Bien que $\frac{6}{2}$ soit entier, *Python* ne le sait pas :

```
print(type(6/2))
```

Exercice d'algorithmique :

Comment faire pour que *Python* reconnaisse $\frac{6}{2}$ comme un entier ?

2°) Programmation objet

a) Propriétés

Le réel π n'est pas connu par *Python*. Pour le charger, il faut le récupérer auprès de l'objet *math* dont il est une **propriété** :

```
from math import pi
print(pi)
```

b) Méthodes

L'expression "racine carrée" s'abrège *sqrt* (**sq**uare **r**oo**t**). Mais pour accéder à cette fonction en *Python*, on doit aussi l'importer depuis l'objet *math* dont elle est une **méthode** (ou algorithme) :

```
from math import sqrt
print(sqrt(2))
```

Le meilleur moyen pour faire des mathématiques avec *Python*, c'est d'importer toutes les propriétés et toutes les méthodes de l'objet *math* avec

```
from math import *
```

Dans ce cas, l'astérisque signifie « tout ».

Chapitre 2

Vocabulaire des évènements

I/ Évènements

1°) Description

a) Notation

Un évènement est décrit par la liste de ses issues possibles, notée entre accolades.

La même notation sera utilisée pour donner la liste des solutions d'une équation ou inéquation.

b) Exemples

1: On tire une carte d'un jeu de 32. L'évènement « la carte est une dame » se note $D = \{D\diamondsuit, D\heartsuit, D\spadesuit, D\clubsuit\}$.

2: Toujours avec un jeu de 32 cartes, l'évènement « la carte est un pique » se note $P = \{1\spadesuit, 7\spadesuit, 8\spadesuit, 9\spadesuit, 10\spadesuit, V\spadesuit, D\spadesuit, R\spadesuit\}$.

3: En lançant un dé, l'évènement « le résultat est pair » se note $A = \{2, 4, 6\}$, et l'évènement « le résultat est plus petit que 5 » se note $B = \{1, 2, 3, 4\}$.

2°) Cas particuliers

1: L'**évènement certain** ou **univers** contient toutes les éventualités. On le note Ω. Par exemple, en lançant un dé, $\Omega = \{1, 2, 3, 4, 5, 6\}$.

2: L'**évènement impossible** est au contraire celui qui ne contient aucune éventualité. Par exemple, en choisissant une carte au hasard dans un jeu de 32, l'évènement « la carte est un 37 de foie » est impossible.

L'évènement impossible est noté $\{\ \}$ ou $\emptyset$.

3: Lorsqu'un évènement ne contient qu'une éventualité, on dit qu'il est **élémentaire**. Par exemple :

(a) Avec un jeu de cartes, l'évènement « la carte est l'as de pique » ou $\{1\spadesuit\}$ est élémentaire.

(b) Avec un dé, l'évènement « le dé est tombé sur 6 » est élémentaire aussi : C'est $\{6\}$.

3°) Lancer de dé en Python

a) Simulation

Pour lancer un dé, on peut faire ceci :

```
from random import *
print(randint(1,6))
```

b) Univers

Pour faciliter la suite, on va stocker l'univers dans une variable appelée *omega*.

```
omega=set(range(1,7))
print(omega)
```

c) Autres évènements

```
pair={2,4,6}
petit={1,2,3,4}
print(pair)
print(petit)
```

II/ Opérations

1°) Contraire

Le contraire d'un évènement A, noté $\overline{A}$, est formé des éventualités qui ne sont pas dans A. On l'obtient en enlevant A à Ω :

```
omega=set(range(1,7))
pair={2,4,6}
impair=omega-pair
print(impair)
```

Exercice :
Donner le contraire des évènements suivants :

1: L'univers :.....

2: L'évènement impossible :.....

3: L'évènement « la carte est un pique » :.....

4: L'évènement « la carte est rouge » :.....

5: Quel est le contraire du contraire de A ?.....

2°) Conjonction

L'évènement « A et B » est formé de toutes les éventualités communes à A et B. On le note $A \cap B$.

Le symbole "$\cap$" ressemble à un "n", lettre centrale du mot anglais *and* qui s'abrège souvent par une esperluette &, elle-même obtenue par une déformation calligraphique de "Et".

```
omega=set(range(1,7))
pair={2,4,6}
petit={1,2,3,4}
print(pair&petit)
```

Lorsque $A \cap B = \emptyset$, on dit que A et B sont **incompatibles**. C'est le cas en particulier de A et $\overline{A}$: On ne peut pas avoir à la fois un évènement et son contraire.

Exercice : La réciproque est-elle vraie ?

3°) Disjonction

L'évènement « A ou B » est formé de toutes les éventualités qui sont soit dans A, soit dans B. On le note $A \cup B$.

Le symbole "$\cup$" ressemble à un "u", qui se prononce souvent "ou" dans plusieurs langues.

```
omega=set(range(1,7))
pair={2,4,6}
petit={1,2,3,4}
print(pair|petit)
```

Chapitre 3

Intervalles

I/ Ensembles de nombres

1: L'ensemble des entiers naturels est noté $\mathbb{N}$.

2: L'ensemble des entiers (relatifs) est noté $\mathbb{Z}$.

3: L'ensemble des fractions est noté $\mathbb{Q}$.

4: L'ensemble des réels est noté $\mathbb{R}$.

L'initiale de $\mathfrak{Zahl}$, « nombre » en Allemand.

Scoop : $\pi \notin \mathbb{Q}$

Pour noter qu'un nombre x est entier, on écrit $x \in \mathbb{Z}$. Pour noter qu'un nombre x n'est pas entier, on écrit $x \notin \mathbb{Z}$.

II/ Inclusion

Pour dire que tous les entiers sont des fractions, on note $\mathbb{Z} \subset \mathbb{Q}$. On dit que $\mathbb{Z}$ est **inclus** dans $\mathbb{Q}$.

III/ Intervalles de réels

1°) Segments

L'ensemble de tous les nombres compris entre a et b est noté $[a;b]$. Plus précisément

1: $x \in [a;b]$ veut dire $a \leqslant x \leqslant b$;

2: $x \in [a;b[$ veut dire $a \leqslant x < b$;

3: $x \in]a;b]$ veut dire $a < x \leqslant b$;

4: $x \in]a;b[$ veut dire $a < x < b$;

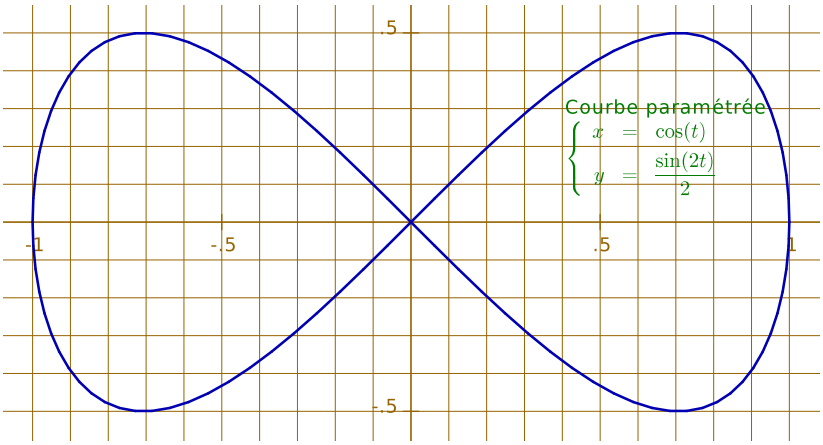
2°) Intervalles d'entiers

En *Python*, les intervalles d'entiers peuvent être définis par *range*, suivi par les deux bornes de l'intervalle. L'appartenance s'écrit *in*. Par exemple, les entiers entre 13 et 21 constituent l'« intervalle »

```
intervalle=range(13,21)
print(0 in intervalle)
print(13 in intervalle)
print(13.2 in intervalle)
print(20 in intervalle)
print(23 in intervalle)
print(21 in intervalle)
```

En bref, le "range" en question est mathématiquement décrit par $\mathbb{N} \cap [13; 21[$.

3°) Demi-droites



On prononce « plus l'infini ».

L'ensemble des tous les réels supérieurs à a est noté $]a; +\infty[$; l'ensemble de tous les réels supérieurs ou égaux à a est $[a; +\infty[$. Ce sont aussi des intervalles.

L'ensemble des nombres inférieurs à b est noté $]-\infty; b[$.

L'ensemble des réels est lui-même un intervalle : $\mathbb{R} =]-\infty; +\infty[$.

L'ensemble vide est aussi un intervalle : $]4; 4[= \emptyset$.

L'intersection de deux intervalles est toujours un intervalle. La réunion de deux intervalles n'est pas forcément un intervalle.

IV/ Boucles en Python

Les *range* de *Python* servent avant tout à faire quelque chose de répétitif, par exemple pour écrire beaucoup de texte :

```
for n in range(5):
        print('ligne␣numéro␣'+str(n))
```

Chapitre 4

Généralités sur les fonctions

I/ Exemple

Si on modifie le rayon x d'une boule, on modifie également son volume : On dit que le volume V est *fonction* du rayon x.

En Python, on peut écrire

```
from math import *
def V(x):
    return 4/3*pi*x**3
```

On rappelle que $V(x) = \frac{4}{3}\pi x^3$.

II/ Définitions

1°) Image

On dit que $V(x)$ est l'*image* de x.

Par exemple, l'image de 10 est $V(10) = \frac{4}{3}\pi \times 10^3 \simeq 4\,189$ cm^3 soit environ 4,2 litres.

2°) Antécédent

On dit que x est l'antécédent de $V(x)$ par V.

Pour trouver l'antécédent de 1000 par V, on doit résoudre l'équation $\frac{4}{3}\pi x^3 = 1000$.

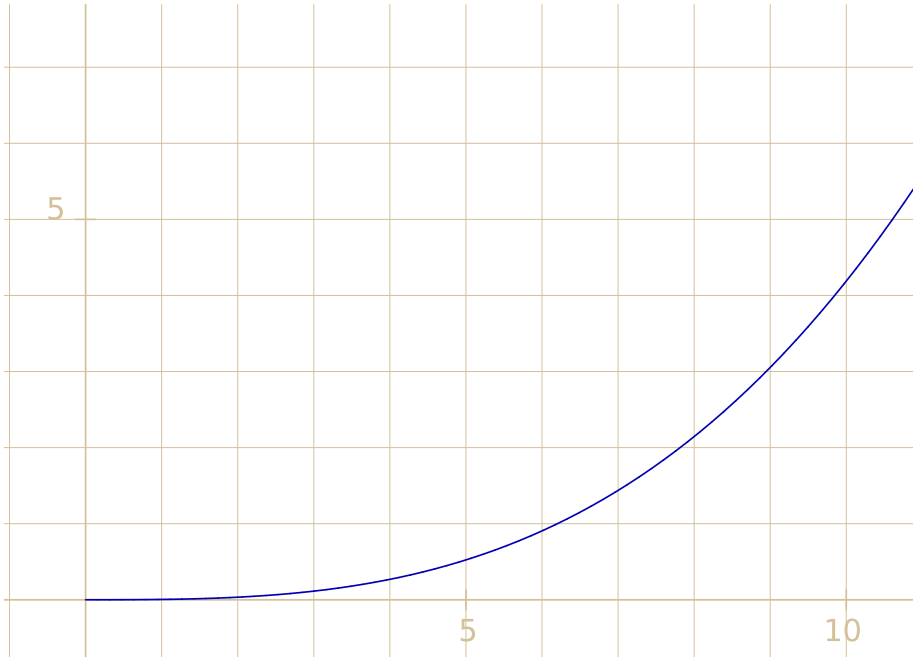
3°) Domaine

L'ensemble des réels qui ont une image par V s'appelle le *domaine de définition* de V. Ici c'est $[0; +\infty[$ parce qu'un rayon est toujours positif.

III/ Représentation graphique

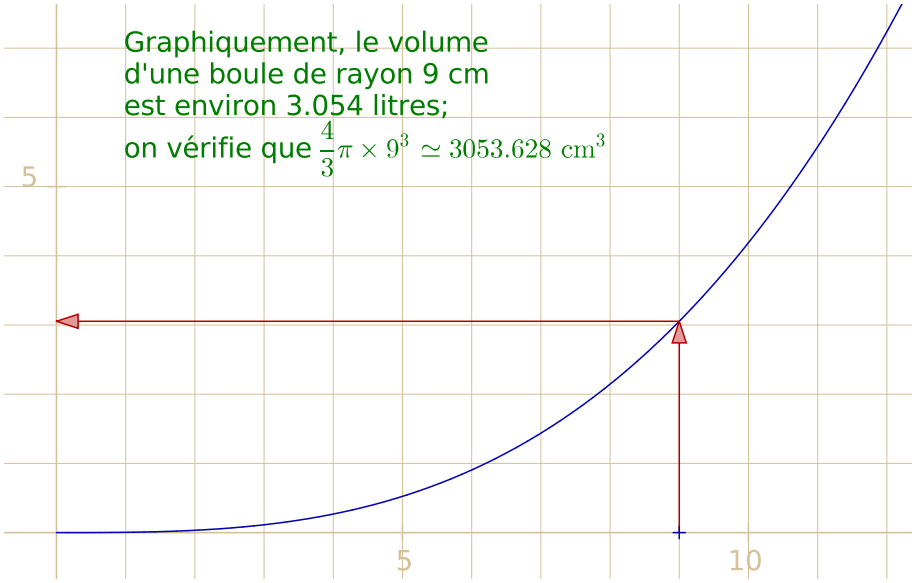
1°) Définition

Si, dans un repère, on place tous les points dont l'ordonnée y est l'image de l'abscisse x, on obtient une courbe appelée *représentation graphique* de la fonction dans le repère.

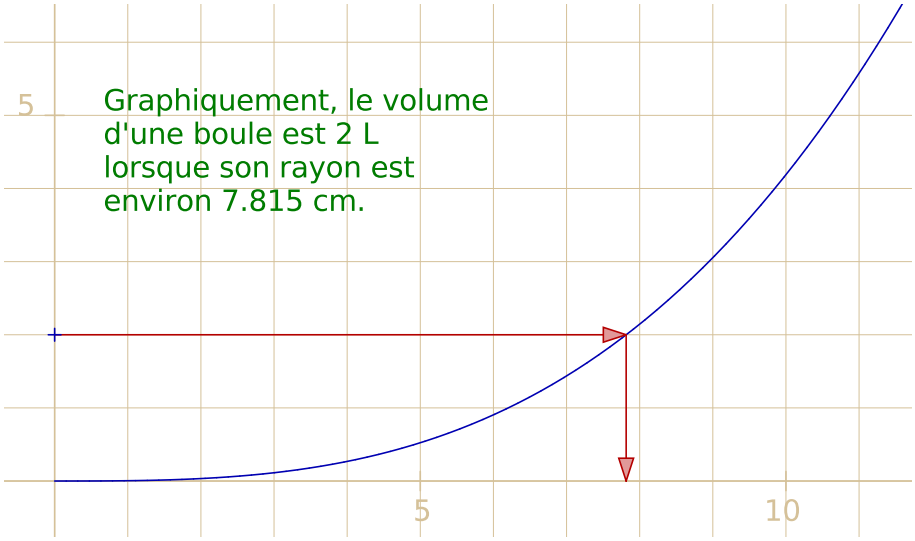


2˚) Utilisation

a) Recherche d'images

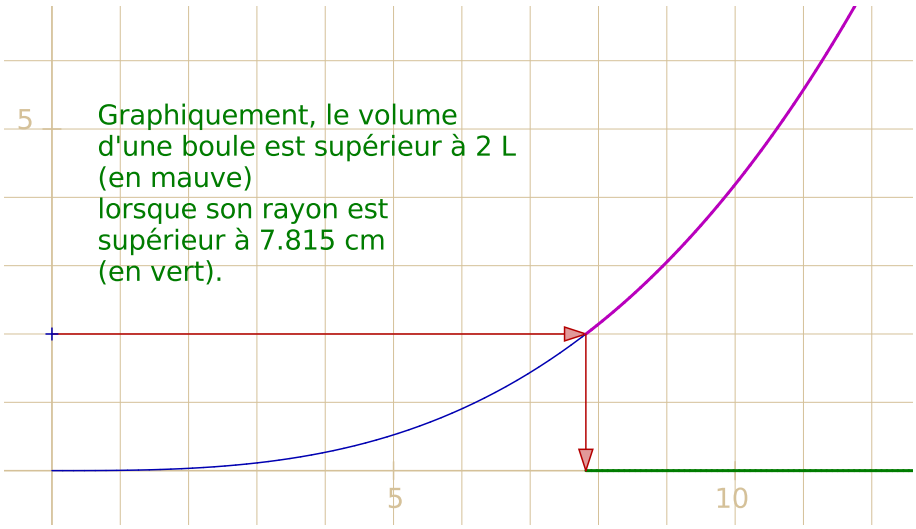


b) Recherche d'antécédents



Cette méthode permet de résoudre des équations graphiquement.

c) Résolution d'inéquations



On trouve graphiquement que l'ensemble des solutions de $V(x) \geqslant 2$ est $[7,8; +\infty[$.

IV/ Variations

1°) fonction croissante

Plus le rayon d'une boule est grand, plus son volume est important : On dit que la fonction V est *croissante* sur $[0; +\infty[$.

Une fonction est croissante sur un intervalle I si, chaque fois que $a \leqslant b$ dans I, $f(a) \leqslant f(b)$.

2°) fonction décroissante

Une fonction est décroissante sur un intervalle I si, chaque fois que $a \leqslant b$ dans I, $f(a) \geqslant f(b)$.

3°) extrema

a) Maximum

On dit que M est le *maximum* de f sur $[a; b]$ si pour tout x de $[a; b]$ on a $f(x) \leqslant M$.

Pour trouver le maximum de la fonction $f(x) = x - \dfrac{x^3}{25}$ sur l'intervalle $[0;5]$ on peut, avec une assez bonne approximation, chercher la plus grande valeur d'un tableau de valeurs de f :

```
def f(x):
    return x-x**3/25

M=max[f(x/1000) for x in range(5000)]
```

b) Minimum

On dit que m est le *minimum* de f sur $[a;b]$ si pour tout x de $[a;b]$ on a $f(x) \leqslant m$.

Chapitre 5

Translations

I/ Implication

1°) Exemple

L'énoncé « Si ABC est rectangle en A alors $AB^2 + AC^2 = BC^2$ » peut aussi s'écrire

1: ABC est rectangle en A seulement si $AB^2 + AC^2 = BC^2$;

2: Pour que ABC soit rectangle en A, il est nécessaire que $AB^2 + AC^2 = BC^2$;

3: Pour que $AB^2 + AC^2 = BC^2$, il suffit que ABC soit rectangle en A ;

4: Le fait que ABC est rectangle en A implique que $AB^2 + AC^2 = BC^2$;

5: Si $AB^2 + AC^2 \neq BC^2$, alors ABC ne peut pas être rectangle en A ;

6: Ou bien ABC n'est pas rectangle en A, ou alors $AB^2 + AC^2 = BC^2$.

2°) Définition

Étant données deux propositions a et b, la proposition « a est faux ou b est vrai » s'appelle **implication** de a vers b et se note $a \Rightarrow b$.

Par exemple, le théorème ci-dessus s'abrège en

$$\widehat{BAC} = 90° \Rightarrow AB^2 + AC^2 = BC^2$$

3°) Réciproque

a) Définition

En inversant le sens de la flèche dans une implication, on obtient la **réciproque** de celle-ci. Par exemple, la réciproque du théorème précédent peut s'écrire

$$\widehat{BAC} = 90° \Leftarrow AB^2 + AC^2 = BC^2$$

ou alors

$$AB^2 + AC^2 = BC^2 \Rightarrow \widehat{BAC} = 90°$$

Exercice :
Donner un exemple de théorème vrai, dont la réciproque est fausse.

b) Équivalence logique

La proposition « $a \Rightarrow b$ et $b \Rightarrow a$ » se dit « a est équivalent à b » et se note $a \Leftrightarrow b$. On dit souvent « a est vrai si et seulement si b est vrai ».

4°) Applications

a) Démonstration

Lorsque a et $a \Rightarrow b$ sont tous les deux vrais, alors b est vrai aussi : On dit qu'on a démontré b à partir de a.

b) Contraposée

La **contraposée** de « a implique b » est « le contraire de b implique le contraire de a ».

Toute implication est équivalente à sa contraposée.

c) Algorithmes et tests

En *Python*, il y a quelque chose qui ressemble à « si a alors b », mais alors que a est toujours une proposition, b est une instruction (ou suite d'instructions) en *Python*, donc un algorithme à effectuer, et pas une proposition : Au lieu de « si a est vrai alors b est vrai aussi » on dit plutôt « si a est vrai alors il faut faire b ». Par exemple, pour jouer à un jeu de dé :

```
from random import *
lancer=randint(1,6)
if lancer==6:
        print('gagné')
```

(on ne gagne que si le dé est tombé sur 6). Peut-être devrait-on aussi voir ce qu'il convient de faire dans les autres cas, pour cela on écrit

```
from random import *
lancer=randint(1,6)
if lancer==6:
        print('gagné')
else:
        print('perdu')
```

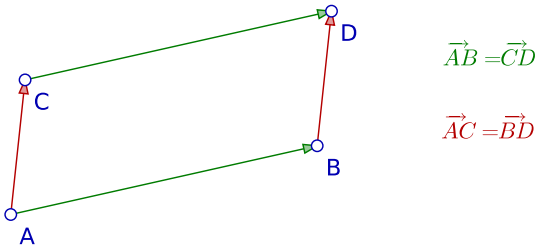
II/ Vecteurs

1°) Définitions

a) Notation

Le vecteur allant de A vers B est noté $\overrightarrow{AB}$ et représenté par un segment fléché allant de A vers B.

b) Égalité



On dit que $\overrightarrow{AB} = \overrightarrow{CD}$ lorsque $ABDC$ est un parallélogramme.

c) Translation

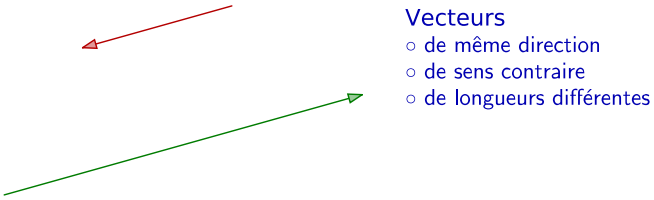
Dans le même cas, on dit que la même translation amène A en B et C en D. On dit que c'est la translation de vecteur $\overrightarrow{AB}$.

2°) Propriétés

1: Lorsque $(AB) \parallel (CD)$, on dit que les vecteurs $\overrightarrow{AB}$ et $\overrightarrow{CD}$ ont la **même direction**.

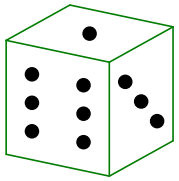
2: Si, en plus, ils ont la flèche du même côté, on dit qu'ils ont le même sens. Par exemple, $\overrightarrow{AB}$ et $\overrightarrow{BA}$ sont de sens contraires (mais ils ont la même direction).

3: La **longueur** du vecteur $\overrightarrow{AB}$ est la distance AB.



Chapitre 6

Probabilités



I/ Définitions

1°) Probabilité

La **probabilité** d'un évènement A, notée $P(A)$, est un nombre compris entre 0 et 1, qui mesure les chances de réalisation de A. Ce nombre ne présente d'intérêt que lorsque A ne s'est pas encore réalisé.

Il s'agit d'une fonction, mais elle n'est pas numérique puisque l'antécédent est un évènement, pas un nombre.

2°) Équiprobabilité

On dit qu'il y a **équiprobabilité** lorsque tous les évènements élémentaires ont la même probabilité.

Dans ce cas, la probabilité d'un évènement est le quotient de sa taille (le nombre d'éventualités qu'il contient), par la taille de l'univers :

len est le début du mot *length* qui veut dire "longueur" en Anglais.

```
omega=set(range(1,7))

def proba(evenement):
        return len(evenement)/len(omega)

pair={2,4,6}
petit={1,2,3,4}

print(proba(pair))
print(proba(petit))
```

Le second exemple n'est pas décimal, donc on lui préfère l'écriture en fraction. Avec *Python* on peut adapter l'exemple précédent en

```
from fractions import *
```

```
omega=set(range(1,7))

def proba(evenement):
        return Fraction(len(evenement),len(omega))

petit={1,2,3,4}
print(proba(petit))
```

L'affichage précédent était une valeur approchée de $\frac{4}{6} = \frac{2}{3}$, ce qu'on exprime parfois par « il y a deux chances sur trois d'avoir un nombre inférieur à 5 en lançant un dé équilibré ».

II/ Propriétés

1°) Intervalle

Une probabilité est toujours comprise entre 0 et 1.

On évitera l'usage des pourcentages pour exprimer une probabilité ; cette notation est réservée aux statistiques.

2°) Cas particuliers

1: $P(\emptyset) = 0$

2: $P(\Omega) = 1$

3°) Disjonction

```
omega=set(range(1,7))

def proba(evenement):
        return len(evenement)/len(omega)

pair={2,4,6}
petit={1,2,3,4}

print(proba(pair|petit)+proba(pair&petit))
print(proba(pair)+proba(petit))
```

On admettra que dans le cas général,

$$P(A \cup B) + P(A \cap B) = P(A) + P(B)$$

4°) Théorème

$$P(\overline{A}) = 1 - P(A)$$

III/ Cas non équiprobable

1°) Exemple

Si le dé est truqué, le 6 sort plus souvent que les autres résultats, par exemple, en donnant les probabilités de chaque chiffre dans un tableau :

La donnée de ces probabilités s'appelle la loi de probabilité de l'univers.

Évènements	1	2	3	4	5	6
Probabilités	0,15	0,15	0,15	0,15	0,2	0,2

En *Python*, un tableau s'écrit entre crochets, mais ses éléments sont comptés à partir de 0, pas de 1. On va donc ajouter un 0 au début du tableau *loi* (la probabilité que le dé donne 0 est nulle) :

```
omega=set(range(1,7))
loi=[0]+[0.15]*4+[0.2]*2

def proba(evt):
        return sum(loi[x] for x in evt)

petit=set(range(1,5))
pair=set(range(2,7,2))

print(proba(petit))
print(proba(pair))
```

2°) Cas général

La probabilité d'un évènement est la somme des probabilités des évènements élémentaires qui le constituent (ses issues).

Chapitre 7

Fonctions affines

I/ Définitions

1°) Fonction affine

Une fonction f est dite **affine** s'il existe deux réels a et b tels que $f(x) = ax + b$.

Exemple : Un transporteur demande 1,5 € par kilomètre parcouru, auxquels il ajoute 20 € pour l'assurance. Alors le prix est fonction affine de la distance.

```
def prix(distance):
        return 1.5*distance+20
```

2°) Coefficient directeur

Le nombre a s'appelle le coefficient directeur de la fonction.
Exemple : Les 1,5 € par kilomètre.

3°) Ordonnée à l'origine

Le nombre b s'appelle l'ordonnée à l'origine de la fonction.
Exemple : Les 20 €.

4°) Cas particuliers

a) Fonctions linéaires

Si $b = 0$, $f(x) = ax$ est **linéaire**.

b) Fonctions constantes

Si $a = 0$, $f(x) = b$ est **constante**.

II/ Représentation graphique

1°) Allure

La représentation graphique d'une fonction affine est une droite.

2°) Réciproque

Réciproquement, si une fonction est représentée par une droite, elle est affine, et le nombre a s'appelle coefficient directeur de la droite, et b s'appelle l'ordonnée à l'origine de la droite.

3°) Détermination

a) Ordonnée à l'origine

Comme l'image de 0 est b, la droite coupe l'axe des ordonnées au point de coordonnées $(0; b)$.

b) Coefficient directeur

Si x_1 et x_2 sont deux nombres,

$$a = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

III/ Variations

1°) Cas où $a > 0$

Si $a > 0$, la fonction $f(x) = ax + b$ est croissante sur $\mathbb{R}$. Son tableau de variation ressemble à

x	$-\infty$	$+\infty$
$ax + b$	↗	

2°) Cas où $a < 0$

Si $a < 0$, la fonction $f(x) = ax + b$ est décroissante sur $\mathbb{R}$. Son tableau de variation ressemble à

x	$-\infty$		$+\infty$
$ax + b$		$\searrow$	

IV/ Signe

1°) Antécédent de 0

La fonction $ax + b$ s'annule lorsque $x = -\dfrac{b}{a}$.

2°) Tableau de signe

a) Cas où $a > 0$

x	$-\infty$		$-\frac{b}{a}$		$+\infty$
$ax + b$		$-$	0	$+$	

b) Cas où $a < 0$

x	$-\infty$		$-\frac{b}{a}$		$+\infty$
$ax + b$		$+$	0	$-$	

Chapitre 8

Coordonnées

I/ Point

1°) Repère

Un repère du plan est donné par trois points O, I et J non alignés. La droite (OI) s'appelle l'**axe des abscisses** et elle est généralement horizontale, dirigée vers la droite. La droite (OJ) s'appelle l'**axe des ordonnées** et elle est en général dirigée vers le haut. Le point O est l'**origine** du repère. En posant $\vec{\imath} = \overrightarrow{OI}$ et $\vec{\jmath} = \overrightarrow{OJ}$, on note aussi $(O, \vec{\imath}, \vec{\jmath})$ le repère.

Si $(OI) \perp (OJ)$, le repère est dit **orthogonal**. Si, de plus, $OI = OJ$, le repère est **orthonormé**.

2°) Coordonnées

a) Définition

Un point est déterminé par ses deux coordonnées, l'abscisse et l'ordonnée, notées respectivement x et y. Pour noter que le point M a pour coordonnées x et y, on écrit $M(x; y)$. Par exemple, $O(0; 0)$, $I(1; 0)$ et $J(0; 1)$.

b) Programmation objet

Pour créer un objet *point* en *Python*, on va l'appeler *classe*, et dans la classe, on va placer des *méthodes* de l'objet. La première est celle qui s'exécute chaque fois qu'on crée un point, elle s'appelle toujours __init__ :

Le caractère "_" s'obtient en appuyant sur la touche 8 du haut du clavier.

```
class Point:
        def __init__(self,x,y):
                self.x=x
```

```
            self.y=y
```

Pour accéder à l'abscisse d'un objet *obj*, on écrit *obj.x* ; ici l'objet est le point lui-même, d'où le *self* (*soi* en Anglais).

Cette fonction ne fait que créer les coordonnées du point et les stocker dans les variables x et y. Pour créer le point $M(3;2)$ on fera donc

```
M=Point(3,2)
```

Une méthode d'affichage possible est celle-ci :

```
    def __str__(self):
        return '('+str(self.x)+';'+str(self.y)+')'
```

(le décalage vers la droite (« indentation ») signifie que la définition de la méthode est à l'intérieur de la « classe »)

II/ Milieu

1°) Coordonnées

On considère deux points $A\,(x_A;y_A)$, $B\,(x_B;y_B)$ et leur milieu $M\,(x_M;y_M)$

a) Abscisse

L'abscisse du milieu est la moyenne des abscisses

$$x_M = \frac{x_A + x_B}{2}$$

b) Ordonnée

L'ordonnée du milieu est la moyenne des ordonnées

$$y_M = \frac{y_A + y_B}{2}$$

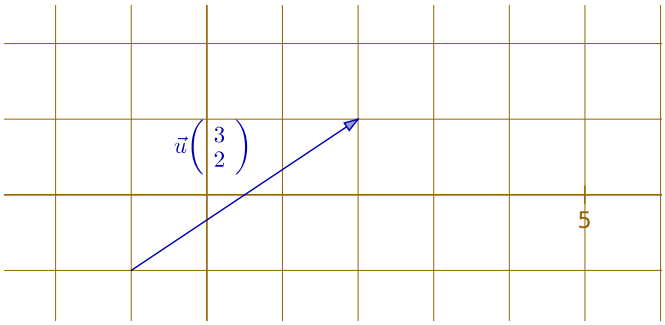
c) En Python

```
    def milieu(self,p):
        xM=(self.x+p.x)/2
        yM=(self.y+p.y)/2
        return Point(xM,yM)
```

III/ Vecteur

1°) Coordonnées

Un vecteur aussi a des coordonnées dans un repère. Pour écrire que le vecteur $\vec{u}$ a pour coordonnées 3 et 2, on écrit $\vec{u}(3;2)$ ou $\vec{u}\begin{pmatrix}3\\2\end{pmatrix}$.



a) Abscisse

L'abscisse du vecteur $\overrightarrow{AB}$ est la différence entre les abscisses de A et B.

$$x_{\overrightarrow{AB}} = x_B - x_A$$

b) Ordonnée

L'ordonnée de $\overrightarrow{AB}$ est la différence entre les ordonnées.

$$y_{\overrightarrow{AB}} = y_B - y_A$$

2°) En Python

a) Vecteur

Comme pour le point, on peut donc créer un objet *Vecteur* avec les mêmes propriétés x et y et la même méthode d'affichage :

```
class Vecteur:
        def __init__(self,x,y):
                self.x=x
                self.y=y
        def __str__(self):
                return '('+str(self.x)+';'+str(self.y)+')'
```

b) Points

On peut aussi définir un vecteur à partir de deux points, mais ici ce sera une méthode de l'objet *Point* :

```
        def vecteur(self,p):
                return Vecteur(p.x-self.x,p.y-self.y)
```

Exemple : Dans un repère, on donne $A(-3;5)$ et $B(3;-2)$. On demande les coordonnées, dans le même repère, du milieu M de $[AB]$, ainsi que du vecteur $\overrightarrow{AB}$. On suppose que les classes précédentes ont été enregistrées dans des fichiers appelés *points.py* et *vecteurs.py*.

```
from points import *
from vecteurs import *
A=Point(-3,5)
B=Point(3,-2)
print(A.milieu(B))
print(A.vecteur(B))
```

Chapitre 9

Espace

Chapitre 10

Addition des vecteurs

I/ Somme de vecteurs

1°) Coordonnées

Étant donnés deux vecteurs $\vec{u}$ et $\vec{v}$, on définit leur somme comme le vecteur $\vec{u}+\vec{v}$ dont

1: l'abscisse est la somme des abscisses de $\vec{u}$ et $\vec{v}$

$$x_{\vec{u}+\vec{v}} = x_{\vec{u}} + x_{\vec{v}}$$

2: l'ordonnée est la somme des ordonnées

$$y_{\vec{u}+\vec{v}} = y_{\vec{u}} + y_{\vec{v}}$$

2°) Python

La méthode *__add__* correspond au symbole $\boxed{+}$, c'est donc celle-là qu'on va définir pour l'objet *Vecteur* (associé à un autre vecteur) :

```
class Vecteur:
        def __init__(self,x,y):
                self.x=x
                self.y=y
        def __str__(self):
                return '('+str(self.x)+';'+str(self.y)+')'
        def __add__(self,v):
                return Vecteur(self.x+v.x,self.y+v.y)
```

II/ Addition des vecteurs

1°) Translations successives

Si on va de A vers B, puis de B vers C, on accomplit une translation de A vers C : $\overrightarrow{AB} + \overrightarrow{BC} = \overrightarrow{AC}$.

Cette relation porte le nom de Michel CHASLES, mathématicien français du 19e siècle.

2°) Exemple

On donne $A(-3;5)$, $B(3;-2)$ et $C(5;8)$ dans un repère. On peut vérifier que $\vec{w} = \overrightarrow{AC}$ est la somme de $\vec{u} = \overrightarrow{AB}$ et $\vec{v} = \overrightarrow{BC}$, tout simplement en comparant leurs coordonnées :

```
from points import *
from vecteurs import *
A=Point(-3,5)
B=Point(3,-2)
C=Point(5,8)
u=A.vecteur(B)
v=B.vecteur(C)
w=A.vecteur(C)
print(u+v)
print(w)
```

3°) parallélogramme

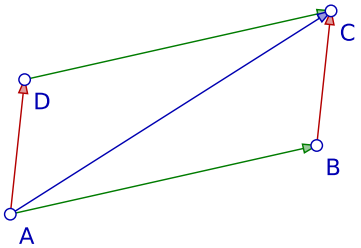
Soit $ABCD$ un parallélogramme. Alors les égalités suivantes sont vraies :

1: $\overrightarrow{AB} = \overrightarrow{DC}$

2: $\overrightarrow{AB} + \overrightarrow{BC} = \overrightarrow{AC}$

3: $\overrightarrow{AB} + \overrightarrow{AD} = \overrightarrow{AC}$

La règle du parallélogramme



Chapitre 11

Trinômes

I/ Fonction "carré"

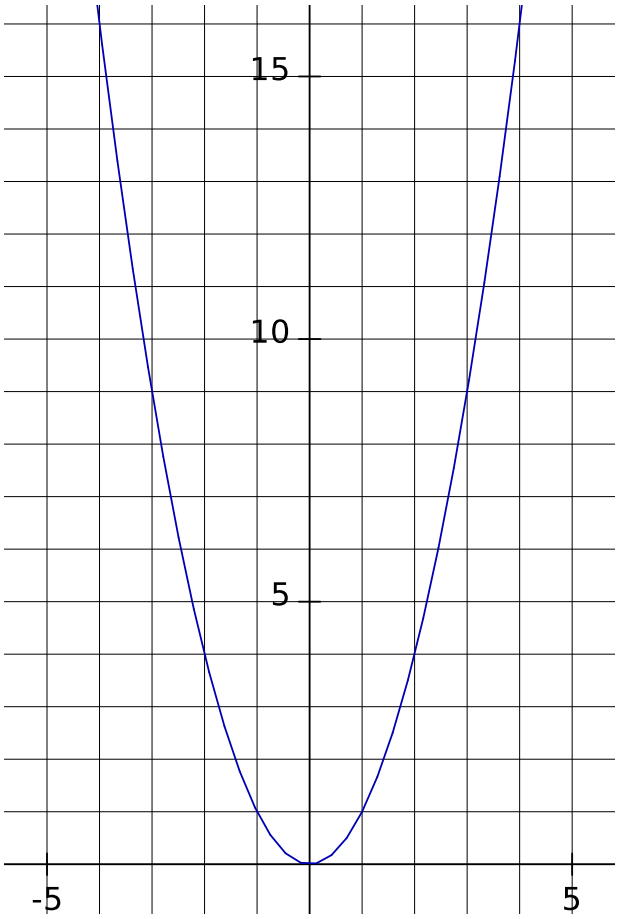
1°) Définition

Le carré d'un réel x est le produit de x par lui-même : $x^2 = x \times x$.

```
def carré(x):
        return x**2
```

La fonction "carré" est définie sur $\mathbb{R}$.

2°) Représentation graphique



3°) Variations

a) Tableau de variations

x	$-\infty$		0		$+\infty$
x^2		$\searrow$	0	$\nearrow$	

b) Minimum

La fonction "carré" admet un minimum en 0, et ce minimum est 0 : $x^2 \geqslant 0$.

Un carré est donc toujours positif.

II/ Trinômes

1°) Définition

On dit qu'une fonction f est du second degré (ou *trinôme*) s'il existe trois réels $a \neq 0$, b et c tels que $f(x) = ax^2 + bx + c$.

2°) Variations

a) Extremum

La fonction $f(x) = ax^2 + bx + c$ passe par un minimum ou un maximum en $x = -\dfrac{b}{2a}$. Sa représentation graphique est une **parabole** d'axe vertical (l'équation de l'axe est $x = -\dfrac{b}{2a}$)

b) Cas où $a > 0$

x	$-\infty$		$-\frac{b}{2a}$		$+\infty$
$f(x)$		$\searrow$	$f\left(-\frac{b}{2a}\right)$	$\nearrow$	

c) Cas où $a < 0$

x	$-\infty$		$-\frac{b}{2a}$		$+\infty$
$f(x)$		$\nearrow$	$f\left(-\frac{b}{2a}\right)$	$\searrow$	

Chapitre 12

Statistique

I/ Tableaux

Un tableau T est une liste de données, la première se notant T[0], la suivante T[1], etc. Pour trier le tableau T, on écrit T.sort(). Dans ce chapitre, on va faire des statistiques sur les 1000 premières décimales de $\sqrt{2}$ (pour des indices allant de 0 à 999 dans le tableau).

En anglais, trier se dit *to sort*...

```
from decimal import *
getcontext().prec=1001
rac2=str(Decimal(2).sqrt())
rac2=rac2[2:]
chiffres=[int(c) for c in rac2]
```

II/ Moyenne

```
def moyenne(tableau):
        return sum(tableau)/len(tableau)

print(moyenne(chiffres))
```

En choisissant des chiffres complètement au hasard, leur moyenne est 4,5 et ici, on a $4,482 \simeq 4,5$.

L'erreur d'approximation est $\frac{4,5-4,482}{4,5} \simeq 0,004$ soit 0,4 %.

III/ Quantiles

1°) Médiane

```
def mediane(tableau):
        trie=sorted(tableau)
        n=len(trie)
        if n%2==1:
                return trie[n//2]
        else:
                return (trie[n//2-1]+trie[n//2])/2

print(mediane(chiffres))
```

Parmi les 1000 premières décimales de $\sqrt{2}$, le *chiffre médian* est donc 5.

2°) Quartiles

a) Premier quartile

```
def Q1(tableau):
        trie=sorted(tableau)
        n=len(trie)
        return trie[n//4]
```

Parmi les 1000 premières décimales de $\sqrt{2}$, le premier quartile est le chiffre 2.

b) Troisième quartile

```
def Q3(tableau):
        trie=sorted(tableau)
        n=len(trie)
        return trie[3*n//4]
```

Parmi les 1000 premières décimales de $\sqrt{2}$, le troisième quartile est le chiffre 7.

IV/ Effectifs

1°) Comptage sous Python

a) Un effectif

Pour savoir combien de fois le 6 est sorti (l'effectif du 6), on peut faire

```
print(chiffres.count(6))
```

On apprend que le chiffre 6 figure 90 fois parmi les 1000 premières décimales de $\sqrt{2}$.

b) Effectifs

Donc la liste des effectifs peut s'obtenir par

```
effectifs=[chiffres.count(n) for n in range(9)]
print(effectifs)
```

Exercice d'algorithmique :
Écrire un algorithme qui, à partir des effectifs, donne les fréquences.

2°) Effectifs cumulés

a) Effectifs cumulés croissants

```
effcum=[sum(effectifs[0:n]) for n in range(9)]
print(effcum)
```

Les effectifs cumulés se représentent en général par un polygone, mais le diagramme en bâtons est aussi possible.

b) Généralisation

On définit également les effectifs cumulés décroissants, et les fréquences cumulées.

Les effectifs et fréquences cumulés peuvent aider à déterminer graphiquement les quartiles.

Chapitre 13

Fonctions homographiques

I/ Fonction "inverse"

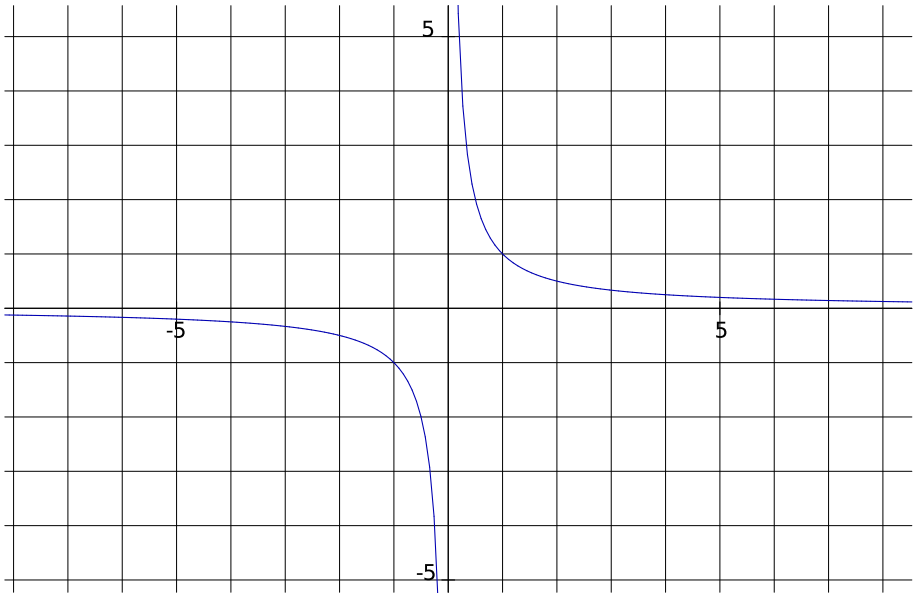
1°) Définition

L'inverse d'un réel x non nul est le quotient de 1 par x par lui-même : $x^{-1} = \dfrac{1}{x}$.

```
def inverse(x):
        return x**(-1)
```

La fonction "inverse" est définie sur $]-\infty;0[\cup]0;+\infty[$.

2°) Représentation graphique



3°) Variations

a) Tableau de variations

x	$-\infty$		0		$+\infty$
$\frac{1}{x}$		$\searrow$		$\searrow$	

II/ Homographies

1°) Définition

On dit qu'une fonction h est homographique s'il existe quatre réels a, b, c et d tels que $h(x) = \dfrac{ax+b}{cx+d}$.

2°) Valeur interdite

a) Remarque

Si $x = -\dfrac{d}{c}$, le dénominateur s'annule : $-\dfrac{d}{c}$ est la **valeur interdite** de la fonction h.

b) Ensemble de définition

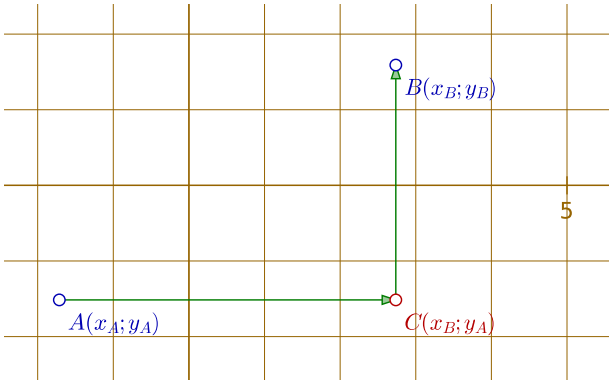
La fonction $h(x) = \dfrac{ax+b}{cx+d}$ est donc définie sur $\left]-\infty; -\dfrac{d}{c}\right[\cup \left]-\dfrac{d}{c}; +\infty\right[$.

Chapitre 14

Distances dans un repère orthonormé

On ne parle de mesures (distances ou angles) que dans un repère orthonormé.

I/ Pythagore



Dans un repère orthonormé, soient $A(x_A;y_A)$ et $B(x_B;y_B)$. Alors, en ajoutant le point $C(x_B;y_A)$, le vecteur $\overrightarrow{AC}$ a la même abscisse que le vecteur $\overrightarrow{AB}$ et le vecteur $\overrightarrow{CB}$ a la même ordonnée que le vecteur $\overrightarrow{AB}$. Par ailleurs, le triangle ABC est rectangle en C **puisque le repère est orthonormé**, donc, d'après le théorème de Pythagore, le carré de la longueur de $\overrightarrow{AB}$ est la somme des carrés de ses coordonnées.

II/ longueur d'un vecteur

La longueur du vecteur $\vec{u}$ est donc donnée par $\sqrt{x_{\vec{u}}^2 + y_{\vec{u}}^2}$.
On peut donc ajouter une propriété à l'objet *Vecteur* de *Python* :

```
from math import *

class Vecteur:
        def __init__(self,x,y):
                self.x=x
                self.y=y
        def __str__(self):
                return '('+str(self.x)+';'+str(self.y)+')'
        def __add__(self,v):
                return Vecteur(self.x+v.x,self.y+v.y)
        def longueur(self):
                return hypot(self.x,self.y)
```

1°) Distance entre deux points

```
from math import *

class Point:
        def __init__(self,x,y):
                self.x=x
                self.y=y
        def __str__(self):
                return '('+str(self.x)+';'+str(self.y)+')'
        def milieu(self,p):
                xM=(self.x+p.x)/2
                yM=(self.y+p.y)/2
                return Point(xM,yM)
        def vecteur(self,p):
                return Vecteur(p.x-self.x,p.y-self.y)
        def distance(self,p):
                return self.vecteur(p).longueur()
```

Dans un repère orthonormé, la distance entre deux points A et B est donnée par

$$AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

Mais seulement dans un repère orthonormé, hein !

III/ Exemple

Dans un repère orthonormé, soient $A(3;2)$, $B(6;1)$ et $C(4;5)$. Déterminer la nature du triangle ABC.

```
from points import *
from vecteurs import *
A=Point(3,2)
B=Point(6,1)
C=Point(4,5)
d1=A.distance(B)
d2=A.distance(C)
d3=B.distance(C)
print(d1)
print(d2)
print(d1**2+d2**2)
print(d3**2)
```

Chapitre 15

Fonctions trigonométriques

I/ Radians

1°) De degrés en radians

```
from math import *
print(radians(30))
print(pi/6)
```

2°) De radians en degrés

```
from math import *
print(degrees(pi/4))
```

II/ Fonctions trigonométriques

1°) Cosinus

```
from math import *
print(cos(pi/6))
print(sqrt(3)/2)
```

2°) Sinus

```
from math import *
print(sin(pi/4))
print(sqrt(2)/2)
```

Chapitre 16

Direction

I/ Multiplication

1°) Définition

Le produit du vecteur $\vec{u}$ par le réel k est défini comme le vecteur dont l'abscisse est $k \times x_{\vec{u}}$ et l'ordonnée est $k \times y_{\vec{u}}$

2°) Nouvelle méthode

```
class Vecteur:
        def __rmul__(self,r):
                return Vecteur(self.x*r,self.y*r)
```

Pour utiliser cette méthode de l'objet vecteur, on utilise le symbole de multiplication :

```
from vecteurs import *
u=Vecteur(3,2)
print(2*u)
```

3°) Propriétés

Lorsqu'un vecteur est le produit d'un autre vecteur par un nombre, on dit qu'ils sont **colinéaires**.

Cela veut dire (en latin) qu'ils ont la même direction

a) Droites parallèles

Les droites (AB) et (CD) sont parallèles si et seulement si les vecteurs $\overrightarrow{AB}$ et $\overrightarrow{CD}$ sont colinéaires.

b) Points alignés

Les points A, B et C sont alignés si et seulement si les vecteurs $\overrightarrow{AB}$ et $\overrightarrow{AC}$ sont colinéaires.

II/ Test de colinéarité

1°) Produit en croix

Deux vecteurs $\vec{u}(x_{\vec{u}}, y_{\vec{u}})$ et $\vec{v}(x_{\vec{v}}, y_{\vec{v}})$ sont colinéaires si et seulement si leurs coordonnées forment un tableau de proportionnalité, soit

$$x_{\vec{u}} \times y_{\vec{v}} = y_{\vec{u}} \times x_{\vec{v}}$$

2°) Test en Python

À l'objet *Vecteur*, on peut ajouter une méthode de colinéarité (avec un autre vecteur) :

```
from math import *

class Vecteur:
        def __init__(self,x,y):
                self.x=x
                self.y=y
        def __str__(self):
                return '('+str(self.x)+';'+str(self.y)+')'
        def __add__(self,v):
                return Vecteur(self.x+v.x,self.y+v.y)
        def longueur(self):
                return hypot(self.x,self.y)
        def __rmul__(self,r):
                return Vecteur(self.x*r,self.y*r)
        def colin(self,v):
                return self.x*v.y==self.y*v.x
```

3°) Exemple

On voudrait savoir si les points $A(2,1;1,3)$, $B(5,5;3,4)$ et $C(14,4;8,9)$ sont alignés. Pour cela on peut vérifier si les vecteurs $\overrightarrow{AB}$ et $\overrightarrow{AC}$ sont colinéaires :

```
from points import *
from vecteurs import *
A=Point(2.1,1.3)
B=Point(5.5,3.4)
C=Point(14.4,8.9)
u=A.vecteur(B)
v=A.vecteur(C)
print(u.colin(v))
```

Chapitre 17

Échantillonnage

I/ Échantillon

1°) Définition

On constitue un **échantillon** en prélevant au hasard des éléments d'un ensemble. On note N la taille de l'échantillon.

En latin, *alea* veut dire "les dés" ; en arabe, *al zahr* veut dire "les dés" ; en anglais, "au hasard" se dit *at random*.

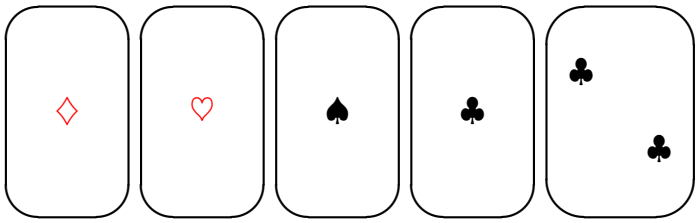
2°) Exemple

Au poker, une main de 5 cartes s'obtient en choisissant les cartes au hasard dans un jeu de 32 cartes :

```
from random import *
valeurs=['1','7','8','9','10','V','D','R']
couleurs=['\u2665','\u2666','\u2663','\u2660']
jeu=[v+'␣'+c for v in valeurs for c in couleurs]
print(jeu)

main=sample(jeu,5)
print(main)
```

En Anglais, "échantillon" se dit *sample*.



II/ Sondages

1°) Méthode

Pour effectuer un sondage, on prélève un échantillon au hasard (pour qu'il soit représentatif). On cherche à estimer la proportion réelle d'un caractère (par exemple, ceux qui sont pour un candidat) dans la population entière, à partir de la proportion mesurée dans l'échantillon. Pour cela on va inverser le problème, en estimant la probabilité que la proportion dans l'échantillon soit proche de la proportion réelle. Et on va estimer cette probabilité par une mesure de fréquence.

2°) Exemple

a) Énoncé

Dans une ville de 100000 habitants, 43000 ont l'intention de voter pour le maire sortant aux prochaines municipales, mais il ne le sait pas. Alors il va commanditer un sondage sur un échantillon de 100 personnes. 51 personnes parmi les 100 disent vouloir voter pour lui.

b) Simulation du sondage

On va simuler 1000 échantillons de 100 personnes chacun, et compter combien d'entre eux donnent l'impression que le maire sera réélu :

```
population=['contre']*57000+['pour']*43000
print(len(population))
print(sample(population,100))
```

Et pour compter les échantillons favorables au maire parmi les 1000 :

```
def favorables(ech):
        return [vote for vote in ech if vote=='pour']

def youpi():
        return len(favorables(sample(population,100)))>=50

p=len([n for n in range(1000) if youpi()])
print(p/1000)
```

III/ Intervalles de fluctuation

1°) Théorie

On admettra le résultat suivant, où p désigne la proportion de « pour » dans la population, et N la taille de l'échantillon :

La probabilité que la proportion de « pour » dans l'échantillon soit comprise entre $p - \frac{1}{\sqrt{N}}$ et $p + \frac{1}{\sqrt{N}}$ est 0,95.

2°) Intervalle de fluctuation

a) Définition

On dit que $\left[p - \frac{1}{\sqrt{N}}; p + \frac{1}{\sqrt{N}}\right]$ est un intervalle de fluctuation à 95 % pour la proportion de « pour » dans la population.

b) Exemple

Avec $N = 100$, on a $\frac{1}{\sqrt{N}} = \frac{1}{10} = 0,1$ donc l'intervalle de fluctuation allait de 0,43-0,1=0,33 à 0,43+0,1=0,53. On constate qu'une partie de cet intervalle donne un succès au maire.

c) Pratique

En pratique, on ne connaît pas p, alors on prend à la place, la proportion dans l'échantillon. On parle alors d'***intervalle de confiance*** à 95 % .

Exemple : Avec $p = 0,51$, on trouve comme intervalle de confiance

$$[0,41; 0,61]$$

Comme $0,43 \in [0,41; 0,61]$, le maire ne peut pas accuser l'intitut de sondage d'avoir triché.

Chapitre 18

Droites du plan

I/ Vecteur directeur

1°) Droite par deux points

```
import points
class Droite:
        def __init__(self,A,B):
                self.A=A
                self.B=B
```

Ici A et B sont des points.

2°) Vecteur

a) Définition

On dit que $\overrightarrow{AB}$ est un **vecteur directeur** de la droite (AB).

b) en Python

On peut ajouter une méthode à l'objet *Droite* :

```
import points
import vecteurs
class Droite:
        def __init__(self,A,B):
                self.A=A
                self.B=B
        def directeur(self):
                return self.A.vecteur(self.B)
```

Deux droites sont parallèles (ont la même direction) si et seulement si elles ont un vecteur directeur en commun.

II/ Équations cartésiennes

1°) Définition

On dit que l'équation $ax + by = c$ est une **équation cartésienne** de la droite d si $M(x_M, y_M) \in d \Leftrightarrow ax_M + by_M = c$.

2°) Méthode

```
        def __str__(self):
                a=-self.directeur().y
                b=self.directeur().x
                c=a*self.A.x-b*self.A.y
                eq='('+str(a)+')x+('+str(b)+')y='+str(c)
                return eq
```

III/ Équation réduite

1°) Cas parallèle à l'axe des ordonnées

Si d est parallèle à (OJ), une de ses équations cartésiennes peut s'écrire $x = c$: Cette équation est dite **réduite**.

2°) Autres cas

Sinon, d est la représentation graphique d'une fonction affine, et possède un coefficient directeur m et une ordonnée à l'origine p. Son équation réduite est alors $y = mx + p$.

a) Coefficient directeur

```
        def cd(self):
                return self.directeur().y/self.directeur().x
```

b) Ordonnée à l'origine

```
        def oalo(self):
                return self.A.y-self.cd()*self.A.x
```

c) Équation réduite

```
        def reduite(self):
                if self.directeur().x==0:
                        eq='x='+str(self.A.x)
                else:
                        eq='y='
                        eq=eq+str(self.cd())+'x'
                        eq=eq+'+('+str(self.oalo())+')'
                        return eq
```

3°) Parallélisme

a) Théorème

Deux droites sont parallèles si et seulement si elles ont le même coefficient directeur.

b) En Python

```
        def parallele(self,autre):
                return self.cd()==autre.cd()
```

On pouvait aussi faire

```
def parallele(self,autre):
        return self.directeur().colin(autre.directeur())
```