Aide à l'utilisation de Python

Lycée

Pierre Martin

SOMMAIRE

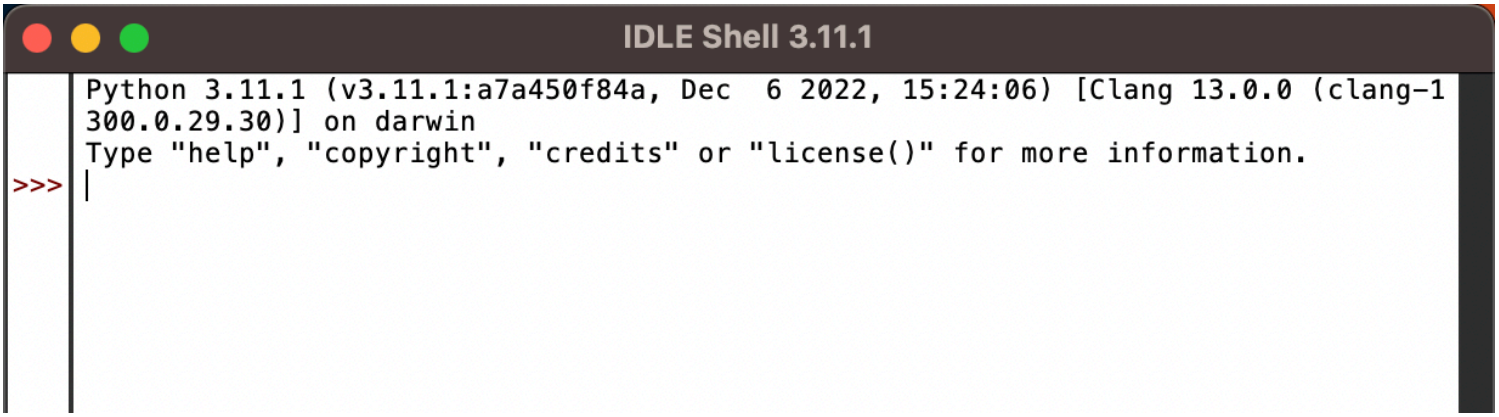
(Interactif : cliquer sur la partie pour y accéder directement)

Fonctionnement général 3
Les opérations mathématiques 4
Déclarer une variable 4
Un premier programme 5
Courbe représentative d'une fonction 6
1. Relier des points, utiliser les coordonnées 6
2. Tracer la courbe représentative d'une fonction 7
3. Plusieurs courbes représentatives sur un même repère 9
Réaliser des diagrammes 10
1. Diagramme en barres 10
2. Histogrammes 11
3. Diagramme circulaire 12
Appeler une fonction 14
Générer un tableau de valeurs 14
Les suites numériques 15
1. Suite définie par une formule explicite 15
2. Suite définie par une relation de récurrence 16
3. Suite de Fibonacci 17
4. Recherche de seuil 17
L'aléatoire 18

Fonctionnement général

Python est un logiciel de programmation, dont le téléchargement et l'utilisation sont gratuites. Pour commencer, télécharger le logiciel ici. A titre d'information, ce langage de programmation a été inventé par Guido van Rossum en 1990.

Une fois le téléchargement terminé, lancer l'application IDLE.
La fenêtre suivante s'affiche alors :



C'est ici que l'on va écrire les lignes de codes. Essayons de taper la ligne de code suivante :

```
print("Salut les Maths de Jean-Kevin !")
```

On remarque que certains caractères ont changé de couleur de façon automatique. Le texte s'est alors affiché en bleu sous notre ligne de code : c'est le résultat. A retenir, donc : le résultat sera toujours affiché en bleu sous la dernière ligne de notre code.

Sur la ligne suivante, on retrouve les caractères suivants : >>>. Cela signifie que le logiciel attend une nouvelle ligne de code. Si on souhaite lancer un nouveau programme ou un programme déjà réalisé, il faudra expliquer au logiciel que l'on souhaite le lancer. On verra comment dans une prochaine partie.

Il est possible de changer la police, le fond et plein d'autres paramètres de l'application. Pour cela, il faut ouvrir le menu préférences de l'application.

Enfin, pour enregistrer un fichier, le raccourcit « **CTRL + S** » ou « **CMD + S** » fonctionne. On peut aussi passer par « **Fichier** » puis « **enregistrer sous** ».

Les opérations mathématiques

Python est très rapide quand il s'agit d'effectuer un calcul. Pour lui faire effectuer les opérations, il faut cependant faire attention à utiliser le bon opérateur dont voici la liste :

Opération	Symbole à utiliser	Exemple	Résultat
Addition	+	`>>> 2+2`	**4**
Soustraction	–	`>>> 7-10`	**– 3**
Multiplication	*	`>>> 6*7`	**42**
Division	/	`>>> 8/10`	**0.8**
Quotient de la division euclidienne	//	`>>> 17//3`	**5**
Reste de la division euclidienne	%	`>>> 17%3`	**2**
Puissance	**	`>>> 2**5`	**32**
Racine carrée	sqrt	`>>> sqrt(49)`	**7**

Déclarer une variable

Une variable est, en quelque sorte, une boîte vide. On remplie ensuite cette boîte de ce que l'on veut : un nombre réel, une caractère, une chaîne de caractères (un mot donc) ou même une liste de nombre. On peut même modifier le contenue de cette variable.

Je veux, par exemple, déclarer une variable que l'on appellera « a ». Je vais initialiser cette variable à 2. Autrement dit, ma variable « a » vaut 2 au début de mon programme. On écrira donc :

```
a=2
```

avant de valider en appuyant sur la touche **entrée**. Pour le logiciel, dès qu'on utilisera cette variable « a », il utilisera la valeur 2 à la place. Par exemple, si on tape la ligne suivante :

```
a+40
```

le logiciel va effectuer le calcul 2 + 40 puisque la variable « a » vaut 2. Il affichera donc le résultat **42**. On peut également changer la valeur de cette variable. Si je souhaite mettre au carré ma variable, il me suffit d'écrire :

```
a=a**2
```

Pour ensuite affiche la valeur de la variable, on utilise la ligne suivante :

```
print(a)
```

la logiciel nous retourne alors la valeur 4.

Un premier programme

C'est le moment ! On va enfin créer notre premier programme !! Bon, après, ne t'emballe pas. On ne va pas non plus lancer une fusée…
Pour créer un programme, il faut lui donner un nom. Il faut aussi savoir à l'avance ce qu'il va faire. Ici, on va saisir notre année de naissance et le programme va nous donner notre âge. Pour cela, il faut qu'on lui donne deux informations : l'année actuelle et notre année de naissance. Il faut donc dire à l'avance au programme que l'utilisateur va donner ces deux informations. On appelle cela un **argument**. Les arguments seront stockés dans deux variables. Appelons notre programme naissance en utilisant la ligne suivante :

```
def naissance(a,b):
```

« def » c'est pour définir notre programme. Il est suivi du nom de celui-ci en bleu. Entre parenthèse, on donne les arguments dont on a besoin dans ce programme. C'est l'utilisateur qui donnera les valeurs par la suite. Le programme va être court. Il faut juste afficher la différence entre l'année actuelle et l'année en cours dans une phrase. Voici la syntaxe à utiliser :

```
def naissance(a,b):
    print('Vous avez', a-b, 'ans.')
```

Le « print » c'est pour afficher à l'écran (soit du texte, soit un résulat). On met ensuite entre parenthèse ce que l'on souhaite afficher. Si on souhaite afficher du texte, on doit le mettre entre les apostrophes. Pour ensuite afficher le résultat souhaité, on sépare le texte du résultat par une virgule.

Pour maintenant exécuter notre programme, on tape la ligne suivante si on est naît en 1994 et que l'année actuelle est 2023 :

```
naissance(2023,1994)
```

On appuie ensuite sur entrée et la magie opère.

Tu connais maintenant les bases pour utiliser ce logiciel. Dans les parties suivantes, je t'explique comment utiliser des commandes précises, liées au programme de mathématiques du lycée. Parce que oui, avec Python, on peut faire de l'analyse, de la géométrie, des statistiques, des probabilités, …

Courbe représentative d'une fonction

Oui, on peut tracer la courbe représentative d'une fonction sur Python. Il nous faut cependant ajouter une bibliothèque (une collection de modules contenant des paquets de codes réutilisables). Cette bibliothèque se nomme **Matplotlib**.

1. Relier des points, utiliser les coordonnées

Dans un premier temps, il faut créer une liste de points. Et pour cela, il faut créer deux listes de nombres : une qui contient toutes les abscisses de nos points et une seconde liste qui contient toutes les ordonnées correspondantes (donc dans le même ordre).
Dans cet exemple, on va relier les points A(1 ; 2), B(3 ; 3), C(4 ; 5) et D(6 , 1). On range donc nos coordonnées dans les deux listes suivantes en utilisant la commande `plt.plot`.

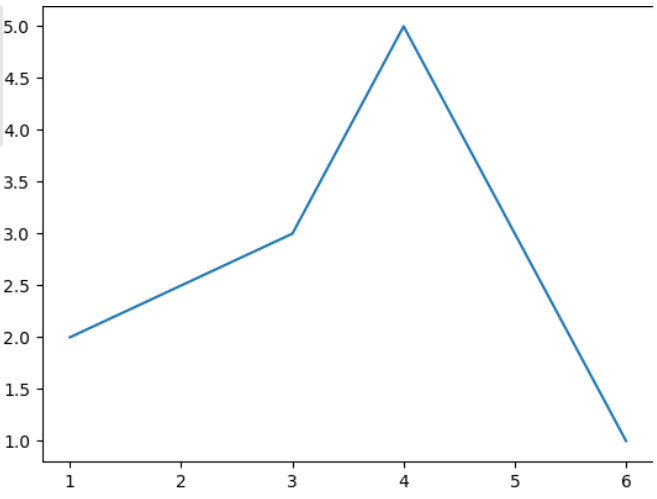
```
plt.plot([1,3,4,6],[2,3,5,1])
```

Les deux listes sont dans des crochets, elles mêmes séparées par une virgule. Pour l'instant, on a juste dit à Python que cette liste existe. On ne lui a pas dit de l'afficher. Pour afficher cette liste de point, il faut ajouter `plt.show`.

```
plt.plot([1,3,4,6],[2,3,5,1])
plt.show()
```

Il nous reste plus qu'à importer la bibliothèque et voici le résultat :

```
import matplotlib.pyplot as plt
plt.plot([1,3,4,6],[2,3,5,1])
plt.show()
```



Il est également possible d'ajouter un titre grâce à la commande `plt.title`. On mettra ensuite le titre entre parenthèses.

On verra, dans un prochain exemple, comment ajouter une légende mais aussi le nom de l'axe des abscisses et celui des ordonnées.

Enfin, il est possible d'afficher le quadrillage, la grille du repère. Pour réaliser cela, il faut utiliser la commande :

```
plt.grid()
```

2. Tracer la courbe représentative d'une fonction

Cette fois, nous avons besoin de la bibliothèque **Numpy** et plus précisément de la commande `np.linspace`. Elle fournit une liste de valeurs. On doit lui renseigner la première valeur, la dernière et le nombre de valeurs total souhaité. Si on souhaite faire une liste des entiers de 1 à 12 on doit utiliser :

```
import numpy as np
x=np.linspace(1,12,12)
print (x)
```

Le logiciel nous renvoie alors `[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12.]`.

Autre exemple :

```
import numpy as np
x=np.linspace(1,5,9)
print (x)
```

Le logiciel nous renvoie alors `[1.  1.5 2.  2.5 3.  3.5 4.  4.5 5. ]`

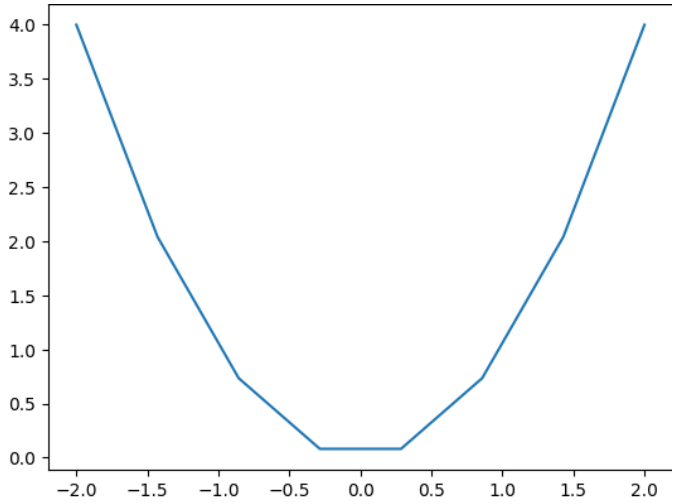
Et cela va nous être bien pratique pour gérer nos abscisses et donc nos antécédents quand on devra tracer la courbe représentative d'une fonction. Pour les ordonnées, il suffira juste de calculer les images de nos antécédents et de les stocker dans une autre liste.
Par exemple, si on veut calculer le carré de chacune des valeurs stockées dans la variable « `x` », on utilisera la commande :

```
import numpy as np
y=x**2
```

Pour résumer : on stocke nos antécédents dans une liste que l'on nomme « `x` » et nos images correspondantes dans une autre liste « `y` ».

On y est presque ! Maintenant que l'on a géré nos antécédents et nos images, il ne reste plus qu'à dire à Python d'afficher la courbe représentative correspondante. On utilise pour cela nos commandes `plt.plot` et `plt.show` de la partie précédente. Attention à ne pas oublier d'importer la bibliothèque **Matplotlib**.
Pour cet exemple, on va tracer la courbe représentative de la fonction carrée pour des valeurs d'antécédents allant de – 2 à 2 par pas de 0,5 (donc 8 valeurs.)

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-2,2,8)
y=x**2
plt.plot(x,y)
plt.show()
```



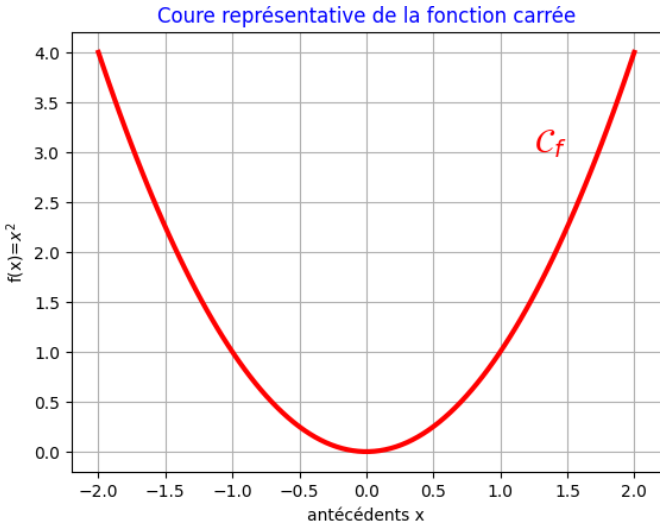
Quoi ?! Mais c'est dégueulasse !! Alors, ça manque de tact mais, oui, c'est perfectible. On peut déjà jouer sur le nombre de points à relier. Pour cela on peut augmenter le nombre de pas dans la liste « `x` ». Il existe ensuite plein de commande qui te permette de customizer l'affichage :

Commande	Utilité
`linewidth`	Gère l'épaisseur du trait. Par défaut, c'est linewidth=1.
`plt.xlabel`	Ajoute un libellé sur l'axe des abscisses.
`plt.ylabel`	Ajoute un libellé sur l'axe des ordonnées.
`plt.title`	Ajoute un titre.
`plt.grid`	Permet d'afficher la grille du repère.
`plt.text`	Permet d'ajouter du texte à un emplacement donné.

Pour chacune de ces commandes, il est possible de modifier la couleur (`color=…`), le type de trait, la taille du texte (`fontsize=…`) ou même le type de marqueur de point. Voici ce qu'il est alors possible d'obtenir en combinant tout cela :

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-2,2,100)
y=x**2
plt.plot(x,y,color='red',linewidth=3)
plt.grid()
plt.xlabel('antécédents')
plt.ylabel('f(x)=$x^2$')
plt.title('Courbe représentative de la fonction carrée',color='blue')
plt.text(1.25,3,$\mathcal{C}_f$,color='red',fontsize=20)
plt.show()
```

Le fait d'écrire entre « $ » permet d'écrire des formules mathématiques en suivant le langage Latex.
Les couleurs doivent être écrites en anglais.
Voici enfin un récapitulatif des paramètres ajustables lors des commandes du type « `plt.` » :

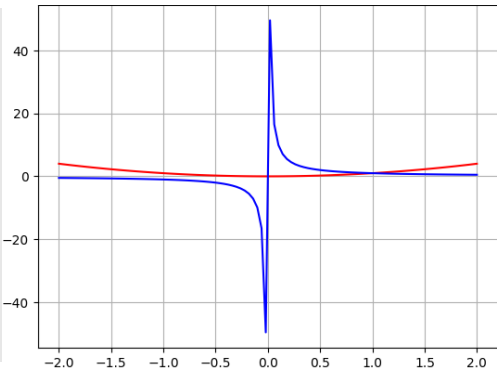


Commande	Utilité
`fontsize=42`	Pour modifier la taille de la police
`fontweight='. . .'`	Pour gérer la graisse (normal, bold, heavy, light, ultrabold, ultralight)
`style='. . .'`	Pour gérer le style de la police (normal, italic, oblique)
`rotation = '. . .'`	Pour tourner le texte (à exprimer en degré, dans le sens anti-horaire)
`color='. . .'`	Pour modifier la couleur du texte (en anglais)
`backgroundcolor='. . .'`	Pour modifier la couleur de fond (en anglais)

3. Plusieurs courbes représentatives sur un même repère

Pour afficher plusieurs courbes représentatives sur un même repère, il suffit d'ajouter une troisième liste de valeurs pour les images de la seconde fonction. On garde en revanche la première liste qui correspond aux antécédents. Voici l'exemple qui permet d'afficher la courbe représentative de la fonction carrée et celle de la fonction inverse :

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-2,2,100)
y1=x**2
y2=1/x
plt.plot(x,y1, color='red')
plt.plot(x,y2, color='blue')
plt.grid()
plt.show()
```



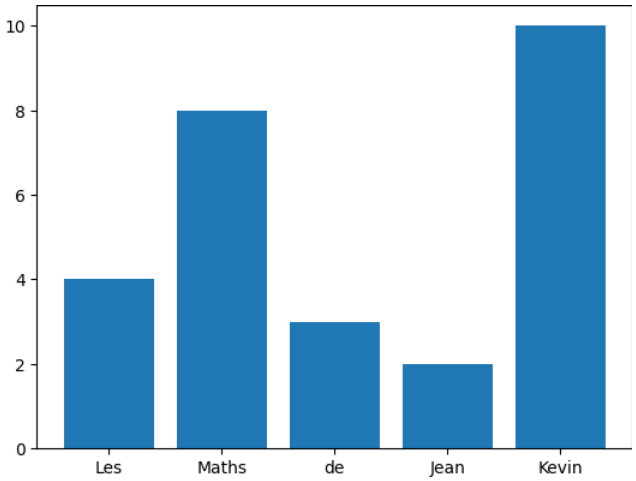
Réaliser des diagrammes

Il est possible, dans une fenêtre externe, d'afficher des diagrammes. Là encore, nous aurons besoin de la bibliothèque **Matplotlib**.

1. Diagramme en barres

Une fois la bibliothèque importée, il faut définir l'étiquette de chacune de nos barres ainsi que les valeurs associées. On demandera ensuite à Python de combiner ces données à l'aide de la commande « `plt.bar` ». Voici un premier exemple de diagramme en barres.

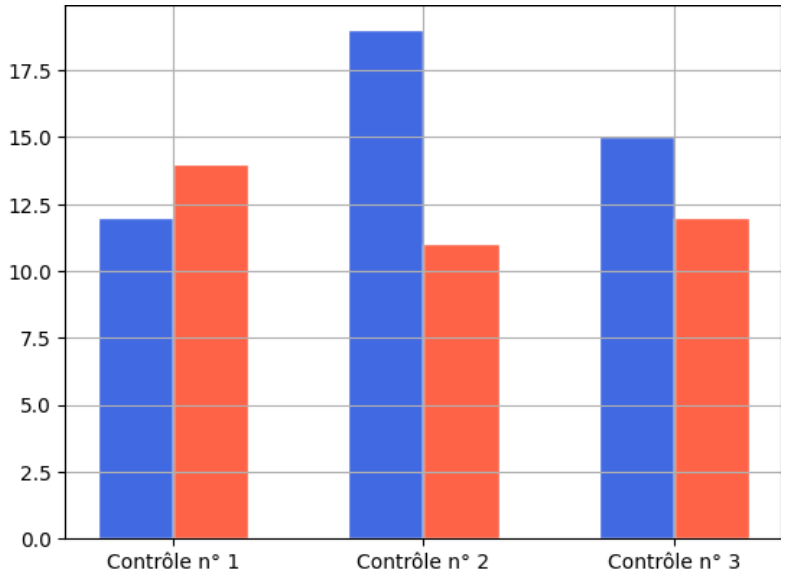
```
import matplotlib.pyplot as plt
names=['Les','Maths','de','Jean','Kevin']
values=[4,8,3,2,10]
plt.bar(names, values)
plt.show()
```



Là encore, il est possible de personnaliser ce diagramme à l'aide des commandes vues dans la partie 5 de ce document en ajoutant une grille, en changeant les couleurs, …
Il est également possible d'afficher des barres regroupées. Pour cela, il faut créer des listes de données supplémentaires. Il faudra également créer une variable qui va gérer la largeur des barres. Souvent, 0.3 est idéal pour un affichage propre.

Voici un code exemple pour, par exemple, comparer des notes entre deux trimestres en mathématiques. Les notes sont rangées dans les listes `y1` et `y2`. On utilise ensuite la commande `range(len(y1))` pour gérer la position des barres de la catégorie 1 que l'on place dans la liste `x1`. De même pour la liste `x2` en ajoutant la `largeur_barre` pour créer un décalage horizontal entre les barres. On utilise la commande `edgecolor` pour créer un contour de couleur blanc autour de nos barres (parce que c'est plus joli). Enfin, on utilise la commande `plt.xticks` pour afficher nos étiquettes et les placer au centre de nos deux barres. Bah ouais, on est en maths ou bien ? On fait en fait la moyenne des positions de deux barres collées.

```
import matplotlib.pyplot as plt
largeur_barre = 0.3
plt.grid()
y1 = [12, 19, 15]
y2 = [14, 11, 12]
x1 = range(len(y1))
x2 = [i + largeur_barre for i in x1]
plt.bar(x1, y1, width = largeur_barre, color = 'RoyalBlue',edgecolor =
'White')
plt.bar(x2, y2, width = largeur_barre, color = 'Tomato', edgecolor = 'White')
plt.xticks([r + largeur_barre / 2 for r in range(len(y1))],['Contrôle n° 1',
'Contrôle n° 2', 'Contrôle n° 3'])
plt.show()
```



Tadaa ! C'est beau, hein ? Pour ajouter une troisième série de barres, il suffit de diminuer un peu la largeur de celles-ci et de changer le calcul de la moyenne dans la commande `plt.xticks`.

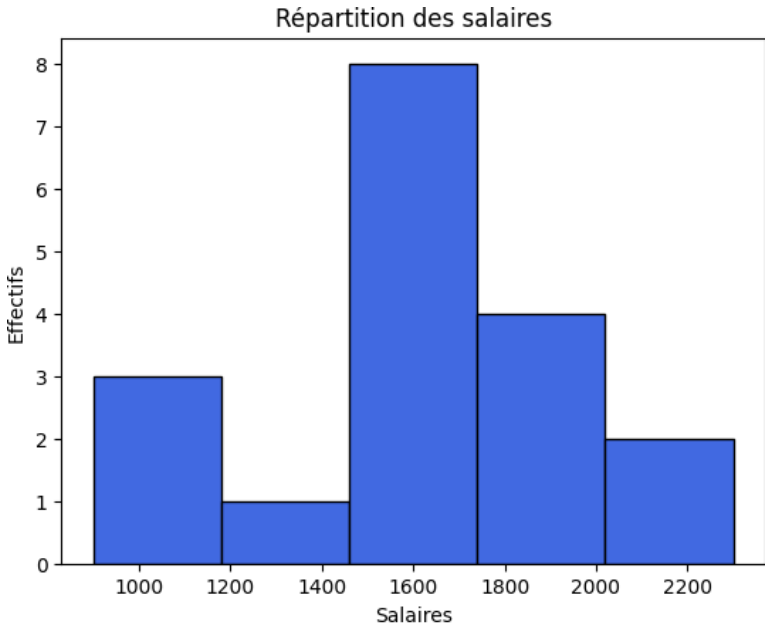
2. Histogrammes

Ici nous avons besoin de renseigner les données suivantes : une liste avec nos données, les bornes de notre intervalle souhaité et le nombre d'intervalle. Là encore, il est possible de modifier les couleurs, l'épaisseur, d'ajouter des titres, … L'ensemble de ces commandes est visible dans cette partie.

La commande principale se nomme « `pyplot.hist` ». Dans l'ordre, on lui donne la liste des valeurs, les bornes de notre intervalle à l'aide de « `range` » et le nombre d'intervalle souhaité dans « `bins` ».

Dans l'exemple suivant, on a représenté la répartition des salaires dans une entreprise.

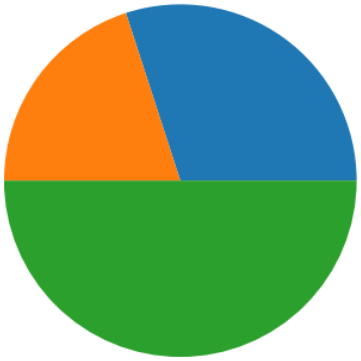
```
import matplotlib.pyplot as plt
x =
[900,900,1500,1500,1600,1100,1700,1800,2200,2300,1600,1500,1700,1900,1700,195
0,1750,1240]
plt.hist(x, range = (900, 2300), bins = 5, color = 'RoyalBlue',edgecolor =
'black')
plt.xlabel('Salaires')
plt.ylabel('Effectifs')
plt.title('Répartition des salaires')
plt.show()
```



3. Diagramme circulaire

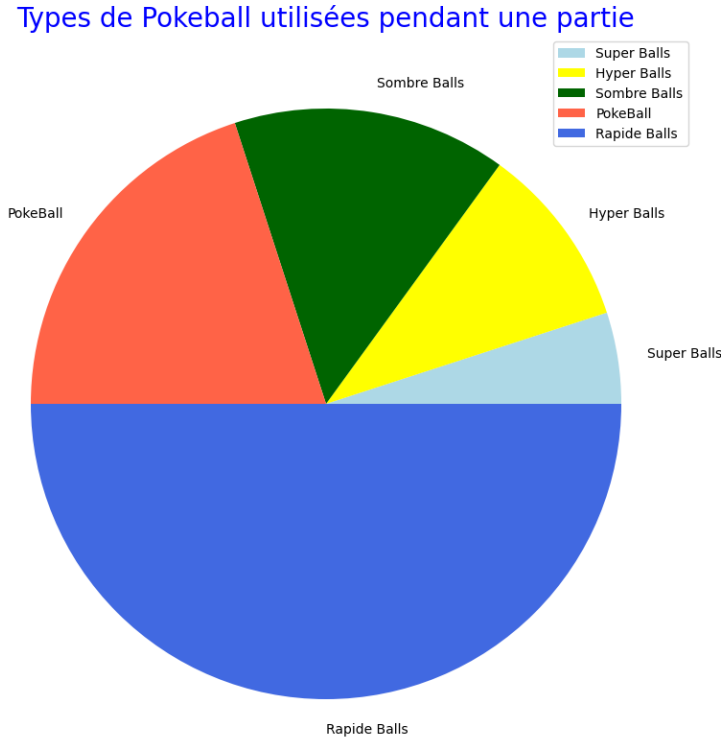
On va apprendre ici à réaliser un diagramme circulaire, ou un camembert comme disent les analphabètes. La commande principale est « `plt.pie` » dont voici un exemple d'utilisation :

```
import matplotlib.pyplot as plt
plt.pie([3,2,5])
plt.show()
```



En principal argument, on renseigne les données statistiques et Python s'occupe du reste. Il reste ensuite à personnaliser ce diagramme. On peut ajouter un titre à l'aide de « `plt.title` », une légende à l'aide de « `plt.legend` » et même de modifier la taille du diagramme grâce à « `plt.figure(figsize=…)` ». Je vous conseille enfin de modifier les couleurs des secteurs et d'ajouter des étiquettes dans la commande principale « `plt.pie` ». Cela rendra le résultat plus professionnel. Tout combiné, voici ce que cela peut donner :

```
import matplotlib.pyplot as plt
plt.figure(figsize = (10,10))
x = [1, 2, 3, 4, 10]
plt.pie(x, labels = ['Super Balls', 'Hyper Balls', 'Sombre Balls',
'PokeBall', 'Rapide Balls'], colors=['LightBlue', 'Yellow','DarkGreen',
'Tomato','RoyalBlue'])
plt.legend()
plt.title('Types de Pokeball utilisées pendant une partie', color='Blue',
fontsize=20)
plt.show()
```



Les étiquettes doivent être rangées dans une liste (attention à l'ordre), idem pour les couleurs des secteurs. N'hésitez pas non plus à agrandir le titre et/ou les étiquettes. Il est possible de ne pas mettre d'étiquettes et d'afficher seulement la légende. Dans ce cas, il faut enlever la partie « `labels` » dans la commande principale « `plt.pie` ».

Appeler une fonction

Fonction ? Fonction ?! T'es là ?! Pour appeler une fonction dans un programme Python, il faut déjà la définir. Cas concret : je veux calculer l'image de 5 par la fonction définie sur IR par $f(x) = \frac{x^2}{1-x}$. On a bien envie de juste taper la commande « `f(5)` » mais Python ne va pas savoir à quoi correspond f. C'est pour cela qu'il faut d'abord définir notre fonction. On lui choisit un nom. Ici, ce sera « `f` » (#originalité). Cette fonction retournera l'image d'un réel renseigné par l'utilisateur par la fonction que l'on va définir. Voici à quoi cela ressemble :

```
def f(x):
    return x**2/(1-x)
```

La commande « `return` » permet de renvoyer le résultat du calcul. S'il n'y a pas cette commande, le programme ne retourne rien. Pour ensuite utiliser cette fonction, il suffit de taper « `f(5)` » et voici la réponse du programme :

```
def f(x):
    return x**2/(1-x)

print(f(5))
> -6.25
```

Parfois, on aura besoin de la bibliothèque **math** pour utiliser des commandes comme « `sqrt` » (pour la racine carrée). Il faudra ajouter, au début, la ligne « `from math import*` ».

Générer un tableau de valeurs

Il est possible de générer un tableau de valeurs. L'affichage n'est pas idéal mais la rapidité de calcul compense cela. On peut choisir d'afficher une première liste contenant les antécédents et une deuxième contenant les images correspondantes par une fonction, préalablement définie. Ici, on va afficher un tableau de valeurs de la fonction carrée pour des antécédents allant de – 2 à 2 par pas de 1.

```
def tableau_valeurs(a,b,p):
    K=int((b-a)/p+1)
    L1=K*[0]
    L2=K*[0]
    x=a
    n=0
    while x<=b :
        L1[n]=round(x,3)
        L2[n]=round(x**2,3)
        x=x+p
        n=n+1
    return(L1,L2)
```

La variable « a » correspond à la borne inférieure (dans notre exemple, – 2 ») et « b » à la borne supérieure (donc 2). Enfin, la variable « p » gère le pas.
L1 est notre liste d'antécédents dont les valeurs vont être – 2, – 1, 0, 1 et 2.
L2 est notre liste d'images correspondantes.
Ces deux listes doivent être initialisées, remplie de 0. K prend la valeur « int » pour lui attribuer un entier à partir des arguments renseignés.
Il faut également initialiser « x » à « a » et « n » à, n étant les rangs de nos listes.
Dans une boucle « while », il faut ensuite remplir chacun des termes de nos deux listes. On utilise « round, 3 » pour avoir trois décimales max dans nos résultats. On augmente la variable « x » du pas souhaité et on passe à l'indice suivant. On sort ensuite de notre boucle pour retourner les deux listes, ou une seule.

```
tableau_valeurs (-2,2-1)

> ([-2,-1,0,1,2],[4,1,0,1,4])
```

Les suites numériques

Là encore, Python va nous servir à gagner du temps dans nos calculs, notamment dans la recherche d'une limite.

1. Suite définie par une formule explicite

On souhaite afficher les termes de la suite (u_n) définie sur IN par $u_n = \dfrac{3n^2 - 7n + 5}{2n^2 - 5}$. On va utiliser un programme qui s'occupe de calculer les termes puis un autre qui va afficher proprement les résultats. Pour cela, on utilise la commande :

```
format()
```

à l'aide des accolades pour préciser que l'on veut un affichage du type « $u_1 = \ldots$ » afin d'éviter d'avoir les valeurs les unes à la suite des autres. On obtient alors quelques chose comme cela pour les 20 premiers termes de cette suite :

```
def u(n):
    return (3*n*n-7*n+5)/(2*n*n-5)

for n in range(20):
    print("u({})={}".format(n,u(n)))
```

À la place des accolades, il y aura les valeurs de « n » et les valeurs de « u(n) ».

On obtient alors le résultat suivant :

```
u(0) = -1.0
u(1) = -0.3333
u(2) = 1.0
u(3) = 0.8461
u(4) = 0.9259
u(5) = 1
u(6) = 1.0597
u(7) = 1.1075
u(8) = 1.1463
u(9) = 1.1783
u(10) = 1.2051
u(11) = 1.2278
u(12) = 1.2473
u(13) = 1.2642
u(14) = 1.2790
u(15) = 1.2921
u(16) = 1.3037
u(17) = 1.3141
u(18) = 1.3234
u(19) = 1.3319
```

La rapidité d'exécution du programme Python permet d'afficher très rapidement les premiers termes et d'effectuer une conjecture quant à la potentielle limite de cette suite.

2. Suite définie par une relation de récurrence

Cette fois-ci, on a besoin d'un premier terme pour pouvoir calculer les suivants. On va donc devoir ajouter une initialisation dans notre programme puis notre formule de récurrence.
On souhaite afficher les termes de la suite (u_n) définie par $u_0 = 4$ et pour tout entier naturel n par $u_{n+1} = 2u_n - 1$. En utilisant, une nouvelle fois, la commande « `format` » et si on souhaite afficher les 10 premiers termes, cela va donner :

```
u=4
print("u({}) = {}".format(0,u))

for n in range(1,10):
    u=2*u-1
    print("u({}) = {}".format(n,u))
```

pour obtenir ce résultat.

```
u(0) = 4        u(5) = 97
u(1) = 7        u(6) = 193
u(2) = 13       u(7) = 385
u(3) = 25       u(8) = 769
u(4) = 49       u(9) = 1537
```

3. Suite de Fibonacci

Elle a droit à sa propre partie, car elle est définie à partir de ses deux premiers termes (et puis parce que je l'aime bien, chacun son truc).
Au début de ce programme, on ajoute une petite commande très utile :

```
=int(input(""))
```

« int » c'est pour dire que la variable sera un nombre entier. « input » c'est pour faire rentrer cette valeur à l'utilisateur. On va en effet lui demander combien de termes il souhaite afficher.
La suite de Fibonacci est définie par u_0 et u_1 et pour tout entier naturel n par $u_{n+2} = u_{n+1} + u_n$. Il faut donc utiliser deux variables et les initialiser avant de créer la boucle qui va gérer la formule de récurrence.

```
Nombre_termes = int(input("Combien de termes voulez-vous ?"))

u0 = 0
u1 = 1

print("\n Voici le début de la suite de Fibonacci : ")
print(u0, " ; ", u1, end=" ; ")

for i in range(2, Nombre_termes):
     terme_suivant = u0 + u1
     print(terme_suivant, end=" ; ")
     u0 = u1
     u1 = terme suivant
```

Si on souhaite afficher les 10 premiers termes, on donne comme valeur de Nombre_termes « 10 » et on obtient l'affichage suivant :

```
Combien de termes voulez-vous ? 10

Voici le début de la suite de Fibonacci :
0 ; 1 ; 1 ; 2 ; 3 ; 5 ; 8 ; 13 ; 21 ; 34
```

4. Recherche de seuil

Le problème est le suivant : pour une suite (u_n) donnée, à partir de quel rang n a-t-on $u_n < a$ (ou bien $u_n > a$) où a est un réel donné.

Jean-Kevin possède un livret A qui lui rapporte 2% par an. Il place 1 000€ sur ce compte. Il veut savoir au bout de combien d'année il aura 3 000€ sur ce livret.
On considère la suite (u_n) définie par $u_0 = 1\,000$ et pour tout entier naturel n par $u_{n+1} = 1{,}02u_n$ où u_n représente la somme d'argent sur le compte de jean-Kevin au bout de n années.
On va demander à Python de calculer les termes de la suite et dès que $u_n > 3\,000$, le programme nous retourne la valeur de n correspondante.

Il faut pour cela utiliser une boucle « `while` » qui calcule les termes de la suite. Dès qu'un terme dépasse le seuil recherché, la boucle d'arrête et la valeur de n est affichée.

```
def argent():
     u = 1000
     n = 0
while u < 3000 :
     u = 1.02*u
     n = n + 1
print(" Le seuil recherché est de ", n, ".")
```

Visiblement, ce sera au bout de 56 ans… C'est très long.
On peut également modifier ce programme pour que la valeur du seuil soit saisie par l'utilisateur. Cela donne le programme suivant :

```
def argent(S):
     u = 1000
     n = 0
while u < S :
     u = 1.02*u
     n = n + 1
print(" Le seuil recherché est de ", n, ".")
```

L'aléatoire

Python possède une commande qui permet de donner une valeur aléatoire comprise entre deux bornes, à renseigner.

```
from random import*
a = randint(min ; max)
```

La fonction `randint` renvoie un entier aléatoire compris entre les valeurs « `min` » et « `max` », ces deux valeurs étant comprise.

Il est alors possible de simuler le lancé d'un dé en faisant deviner le résultat à l'utilisateur.

```
from random import*
a = randint(1 , 6)
test = int(input("On lance un dé. Quelle est la face obtenue ?"))
while test != a :
     print("Perdu, essaye encore.")
     test = int(print("Quel est ta nouvelle réponse ?"))
print("Bravo, la face est bien celle portant le numéro ", a, ".")
```

A l'aide d'un compteur (sous forme de variable) et d'une boucle, il est possible de réaliser plusieurs lancers et de compter le nombre de fois que l'on a obtenu une face, le 4 par exemple.

```
from random import*
compteur = 0
n = int(input("Combien de lancers voulez-vous réaliser ?"))
for i in range (n) :
     a = randint(1 , 6)
     if a == 4 :
          compteur = compteur + 1
print("On a obtenu  ", compteur, " fois la face 4 sur les ", n, " lancers.")
```