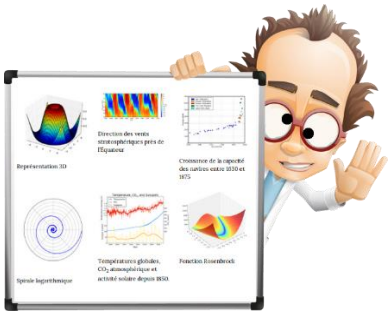
Créer des graphiques scientifiques avec python

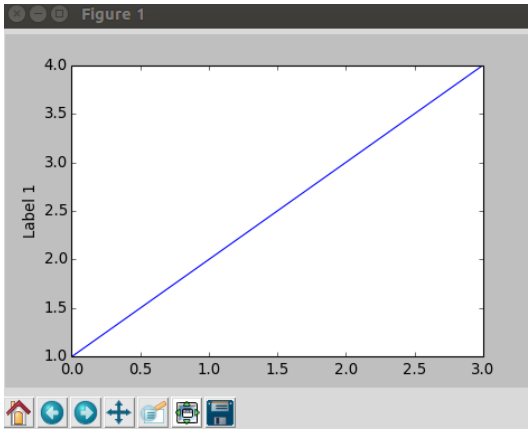


Créer son premier graphique

Voici un exemple simple de graphique :

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot([1,2,3,4])
plt.ylabel('Label 1')
plt.show()
```



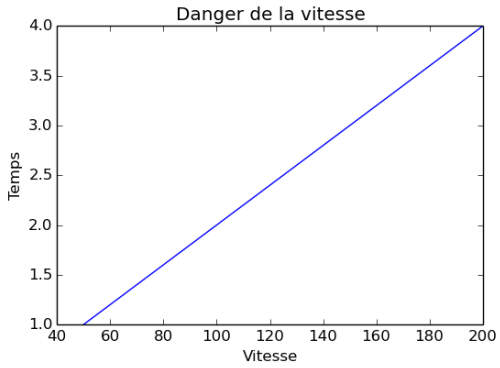
Dans cet exemple on voit que par défaut il n'y a pas qu'un graphique, il y a aussi des outils pour exploiter ce graphique : il est en effet possible de zoomer, de modifier la taille du rendu et bien sur de l'enregistrer.

Pour visualiser le gaphique en plein écran vous pouvez utiliser le raccurci Ctrl + F. Il est possible d'afficher une grille en appuyant sur la touche **g** de votre clavier.

Dans notre exemple nous avons renseigné uniquement les valeurs de l'abscisse, c'est à dire [1,2,3,4], la lib a donc remplit la valeur des ordonnées d'elle-même comme si nous avions donné comme valeur **range(4)** c'est à dire [0,1,2,3]

Il est possible d'associer une valeur x à une valeur y comme ceci :

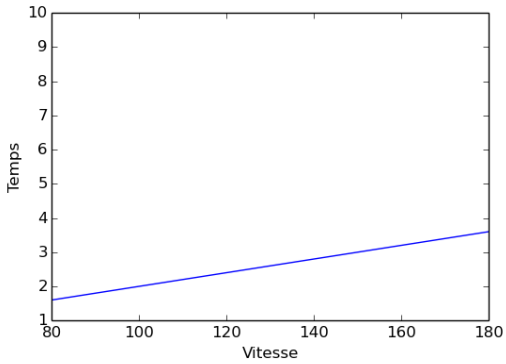
```
plt.figure()
plt.title("Danger de la vitesse")
plt.plot([50,100,150,200], [1,2,3,4])
plt.xlabel('Vitesse')
plt.ylabel('Temps')
plt.show()
```



axis(xmin, xmax, ymin, ymax)

La courbe est juste mais je ne souhaite délimiter moi même les limites du graphique. Seul les vitesses entre 80 et 180 m'intéressent et le temps de 1 secondes à 10 secondes. Il est possible d'encadrer son graphique à l'aide de la méthode **axis(xmin, xmax, ymin, ymax)**

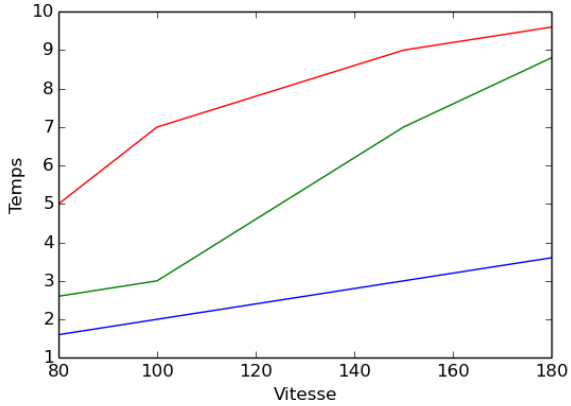
```
plt.figure()
plt.plot([50,100,150,200], [1,2,3,4])
plt.xlabel('Vitesse')
plt.ylabel('Temps')
plt.axis([80, 180, 1, 10])
plt.show()
```



Plusieurs courbes

Il est possible de superposer les courbes :

```
plt.plot([50,100,150,200], [1,2,3,4])
plt.plot([50,100,150,200], [2,3,7,10])
plt.plot([50,100,150,200], [2,7,9,10])
```



Vous pouvez utiliser cette syntaxe également :

```
plt.plot([50,100,150,200], [1,2,3,4], [50,100,150,200], [2,3,7,10], [50,100,150,200],
[2,7,9,10])
```

Changer le visuel des courbes

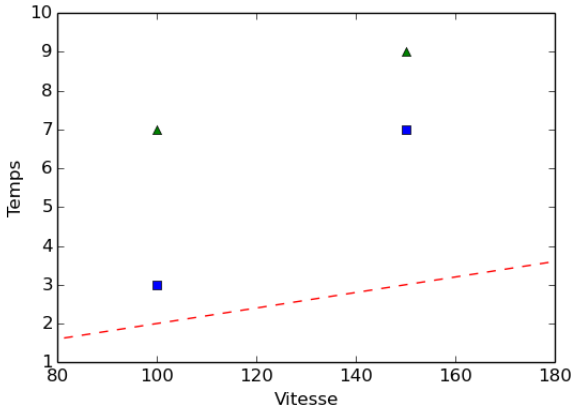
Il est possible de changer la couleur et la forme des courbes :

```
#Courbes possibles: - -- -. :
#Couleurs possibles: b g r c m y k w
plt.plot([50,100,150,200], [1,2,3,4], "r--")
```

```
plt.plot([50,100,150,200], [2,3,7,10], "bs")
plt.plot([50,100,150,200], [2,7,9,10], "g^")
```

Vous pouvez utiliser cette syntaxe également :

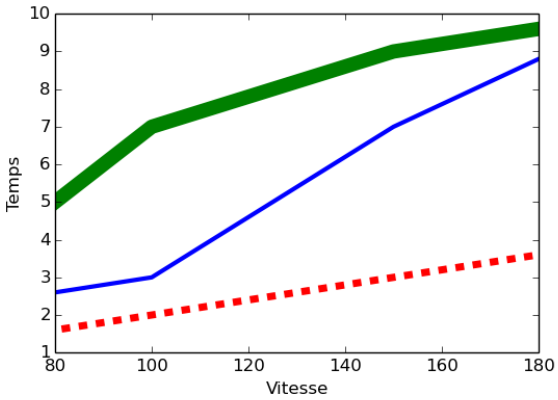
```
plt.plot([50,100,150,200], [1,2,3,4], "r--", [50,100,150,200], [2,3,7,10], "bs", [50,
100,150,200], [2,7,9,10], "g^")
```



Largeur des courbes

Vous pouvez changer la largeur des courbes comme ceci :

```
plt.plot([50,100,150,200], [1,2,3,4], "r--", linewidth=5)
plt.plot([50,100,150,200], [2,3,7,10], "b", linewidth=3)
plt.plot([50,100,150,200], [2,7,9,10], "g", linewidth=10)
```



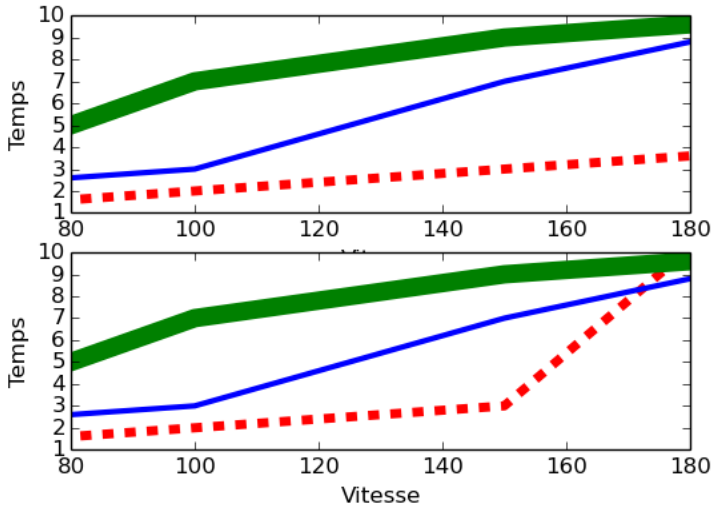
Plusieurs graphiques

Il est possible de mettre plusieurs graphiques sur une même image :

```
plt.subplot(2,1,1)
plt.plot([50,100,150,200], [1,2,3,4], "r--", linewidth=5)
plt.plot([50,100,150,200], [2,3,7,10], "b", linewidth=3)
plt.plot([50,100,150,200], [2,7,9,10], "g", linewidth=10)
plt.xlabel('Vitesse')
plt.ylabel('Temps')
plt.axis([80, 180, 1, 10])

plt.subplot(2,1,2)
plt.plot([50,100,150,200], [1,2,3,15], "r--", linewidth=5)
plt.plot([50,100,150,200], [2,3,7,10], "b", linewidth=3)
plt.plot([50,100,150,200], [2,7,9,10], "g", linewidth=10)
plt.xlabel('Vitesse')
plt.ylabel('Temps')
plt.axis([80, 180, 1, 10])

plt.show()
```

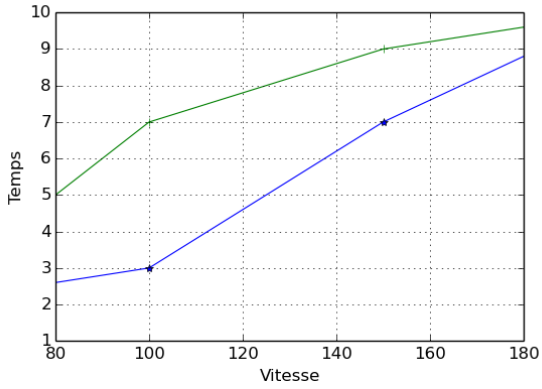


Grille

Vous pouvez également ajouter une grille (et des marqueurs) :

```
#marqueurs possibles : o + . x * ^
```

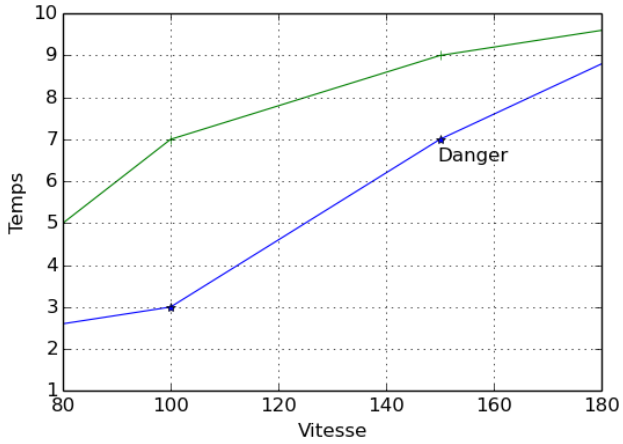
```
plt.grid(True)
plt.plot([50,100,150,200], [2,3,7,10], "b", linewidth=0.8, marker="*")
plt.plot([50,100,150,200], [2,7,9,10], "g", linewidth=0.8, marker="+")
```



Ecrire du texte dans votre graphique

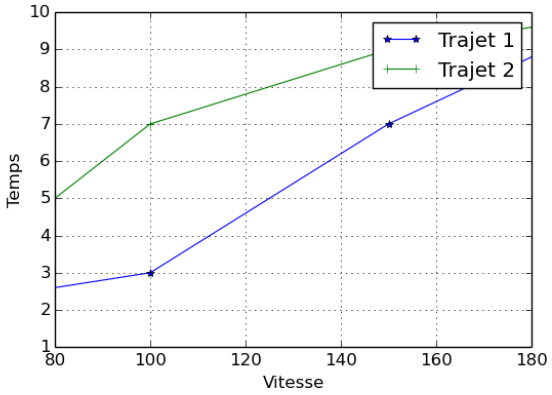
Il est possible d'ajouter du texte en indiquant les coordonnées :

```
plt.text(150, 6.5, r'Danger')
```



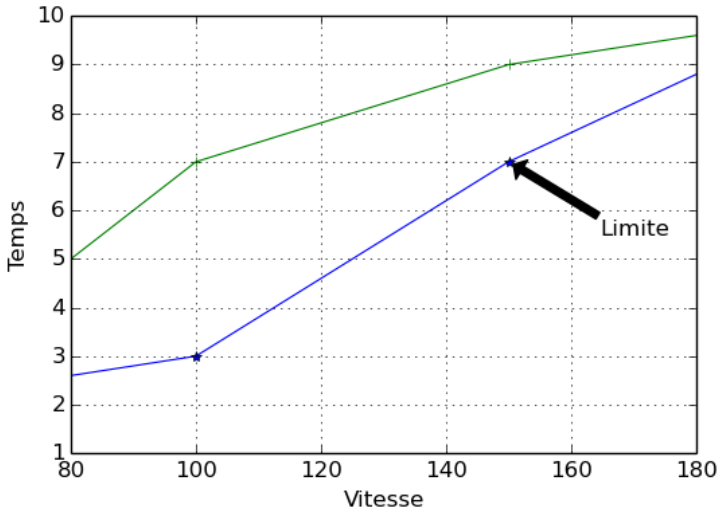
Ajouter une légende pour les courbes

```
plt.grid(True)
plt.plot([50,100,150,200], [2,3,7,10], "b", linewidth=0.8, marker="*", label="Trajet 1")
plt.plot([50,100,150,200], [2,7,9,10], "g", linewidth=0.8, marker="+", label="Trajet 2")
plt.axis([80, 180, 1, 10])
plt.xlabel('Vitesse')
plt.ylabel('Temps')
plt.legend()
plt.show()
```



Ajouter une flèche descriptive

```
plt.grid(True)
plt.plot([50,100,150,200], [2,3,7,10], "b", linewidth=0.8, marker="*")
plt.plot([50,100,150,200], [2,7,9,10], "g", linewidth=0.8, marker="+")
plt.axis([80, 180, 1, 10])
plt.annotate('Limite', xy=(150, 7), xytext=(165, 5.5),
arrowprops={'facecolor':'black', 'shrink':0.05} )
plt.xlabel('Vitesse')
plt.ylabel('Temps')
```



Les histogrammes

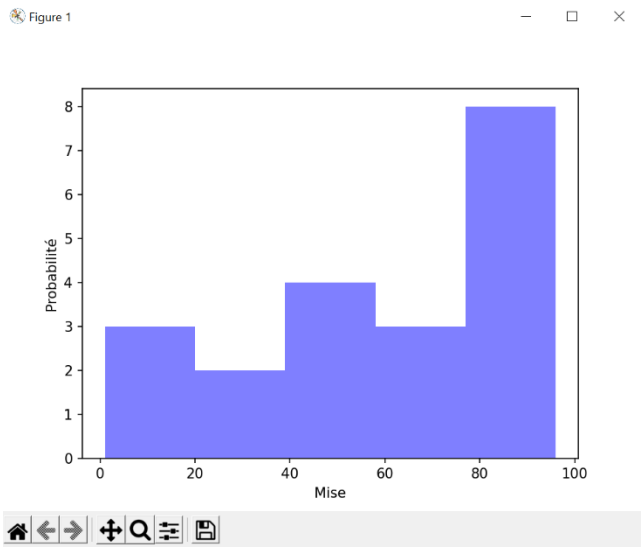
Pour créer un **histogramme** on utilise la méthode **hist** . On peut lui donner des données brutes et il s'occupera de faire les calcules nécessaires à la présentation du graphique.

On a prit ici l'exemple de 20 tirage au sort (random) de valeur entre 0 et 100 et voici le résultat:

```
import matplotlib.pyplot as plt
import random

# 20 tirages entre 0 et 100
x = [random.randint(0,100) for i in range(20)]
num_bins = 5
n, bins, patches = plt.hist(x, num_bins, normed=1, facecolor='b', alpha=0.5)

plt.show()
```



La valeur **50** donné en deuxième paramètre de la méthode **hist** indique le nombre de barres à afficher. Si vous lui passer une liste par exemple **[25,50]** une seule barre sera affichée comprenant l'intervalle 25;50

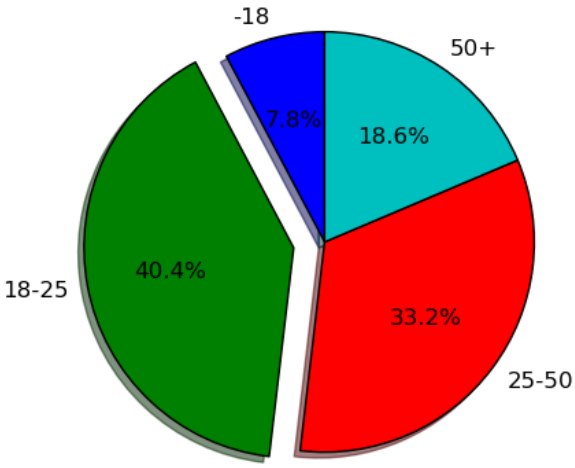
Les accents peuvent faire boguer votre script, n'oubliez pas de mettre votre string en unicode comme dans l'exemple ci-dessus.

pie / diagramme circulaire

Un des graphiques les plus utilisés doit être le **diagramme circulaire** (ou **camembert**) :

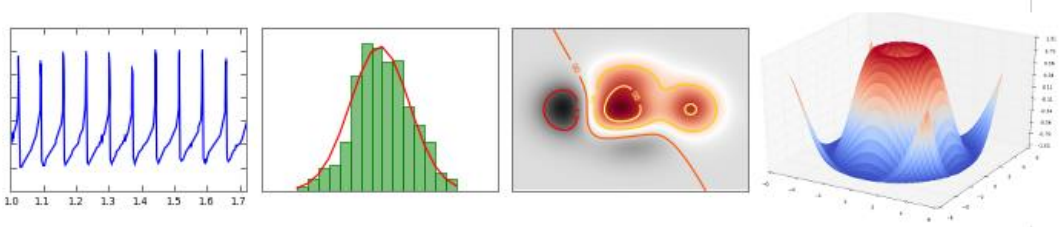
```
name = ['-18', '18-25', '25-50', '50+']
data = [5000, 26000, 21400, 12000]

explode=(0, 0.15, 0, 0)
plt.pie(data, explode=explode, labels=name, autopct='%1.1f%%', startangle=90, shadow=
True)
plt.axis('equal')
plt.show()
```



Les données **explode** indiquent les espaces entre les parts. Nous avons fait en sorte dans cet exemple de séparer la part **18-25** des autres part. Le paramètre **autopct** indique le pourcentage réel par rapport aux données qu'on lui a fournit. Enfin la méthode **axis("equal")** rend le diagramme parfaitement circulaire. En son absence lorsque vous redimensionnez le graphique celui-ci s'adapte au conteneur ce qui peut lui donner une forme élliptique.

Site officiel: matplotlib.org



Source: **https://python.doctor/page-creer-graphiques-scientifiques-python-apprendre**