1 Etude de cas : module turtle

Le module `turtle` permet de créer des images à l'aide d'une tortue qui se balade sur un canvas et qui dessine des lignes. Cette tortue se trouve initialement à la position (0, 0) au centre du canevas et peut se déplacer dans le plan x-y.

Référence: https://docs.python.org/3.7/library/turtle.html (https://docs.python.org/3.7/library/turtle.html)

1.1 Le module turtle

Tout d'abord il faut importer le module `turtle` pour pouvoir utiliser ses fonctions de dessin.

In [67]:
```
import turtle
```

Pour l'insertion d'images dans ce notebook, nous diminuons la taille du canvas.

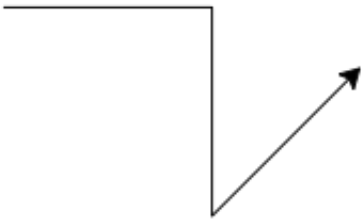
In [138]:
```
turtle.setup(width=600, height=300)
```

Les fonctions suivants permettent de déplacer la tortue:

- `forward` , `fd`
- `backward` , `bk`
- `right` , `rt`
- `left` , `lt`
- `goto` , `setpos` , `setposition`
- `setheading`

In [69]:
```
turtle.reset()
turtle.forward(100)
turtle.left(90)
turtle.backward(100)
turtle.right(45)
turtle.fd(100)
```

Voici le resultat:



Pour obtenir de l'aide utilisez la fonction `help` .

In [64]:
```
help(turtle.forward)
```

```
Help on function forward in module turtle:

forward(distance)
    Move the turtle forward by the specified distance.

    Aliases: forward | fd

    Argument:
    distance -- a number (integer or float)

    Move the turtle forward by the specified distance, in the direct
ion
    the turtle is headed.

    Example:
    >>> position()
    (0.00, 0.00)
    >>> forward(25)
    >>> position()
    (25.00,0.00)
    >>> forward(-75)
    >>> position()
    (-50.00,0.00)
```

1.2 Répétition simple

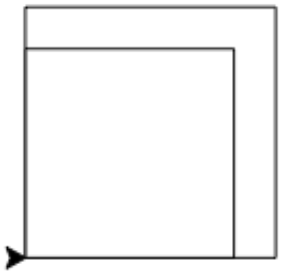
Pour dessiner un carré, on peut avancer de 100 pixels, tourner à gauche de 90 degrés, et répéter ces deux instructions 4 fois.

In [71]:
```
turtle.reset()

turtle.fd(100)
turtle.lt(90)

turtle.fd(100)
turtle.lt(90)

turtle.fd(100)
turtle.lt(90)

turtle.fd(100)
turtle.lt(90)
```

Un façon plus courte et plus efficace est d'utiliser une boucle.

In [73]:
```
for i in range(4):
    turtle.fd(120)
    turtle.lt(90)
```

Voici le resultat:



La vitesse de la tortue peux être lu et changée:

- 1 étant le plus lent
- 3 étant normal
- 10 le plus rapide
- 0 étant instantané (sans animation)

In [72]:
```
turtle.speed()
```

Out[72]: 3

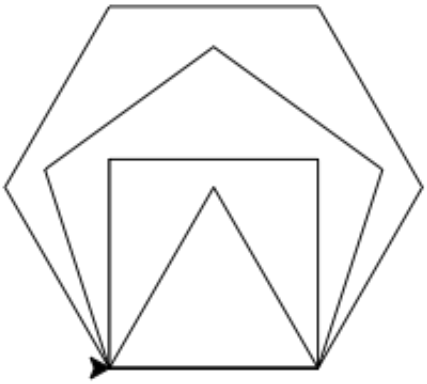
1.3 Dessiner un polygone

Définissons donc une fonction qui dessine un polygone avec `n` sommets et des côtés de longuer `a` ,

In [75]:
```
def polygon(n, a):
    for i in range(n):
        turtle.fd(a)
        turtle.lt(360/n)
```

In [79]:
```
turtle.reset()
turtle.setpos(0, -100)
polygon(3, 100)
polygon(4, 100)
polygon(5, 100)
polygon(6, 100)
```

Voici le resultat:



1.4 Sauvegarder une capteure d'écran

La fonction `getcanvas()` retourne l'objet `ScrolledCanvas`

In [87]:
```
cv = turtle.getcanvas()
cv
```

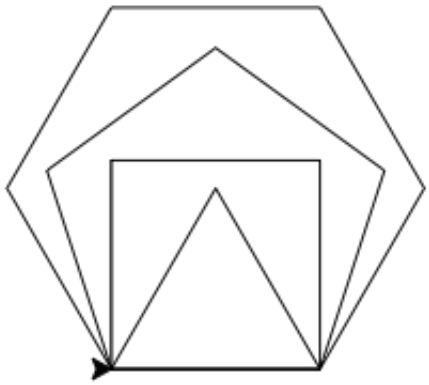
Out[87]:
```
<turtle.ScrolledCanvas object .!scrolledcanvas>
```

L'objet canvas possède une méthode `postscript` qui permet de sauvegarder image.

In [135]:
```
help(cv.postscript)
```

```
Help on method postscript:

postscript(*args, **kw) method of turtle.ScrolledCanvas instance
```

In [88]:
```
turtle.getcanvas().postscript(file="img.eps");
```



Définissons une fonctions pour enregistrer des captures d'écran en format eps.

In [89]:
```
img_index = 0
def img():
    global img_index
    cv = turtle.getcanvas()
    img_name = 'img' + str(img_index) + '.eps'
    cv.postscript(file=img_name, colormode='color');
    img_index +=1
```

La fonction ci-dessous sauvegarde l'actuel image dans le dossier courant, en incrémentant automatiquement l'index de l'image.

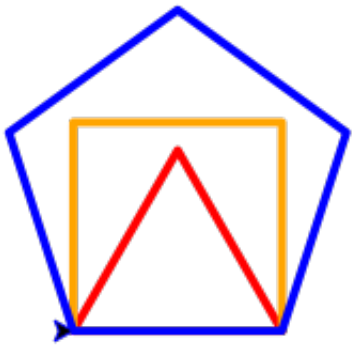
In [92]:
```
img()
```

Déssinons les polygones triangle, carré et pentagone en différentes couleurs.

In [98]:
```
turtle.reset()
turtle.setpos(0, -100)
turtle.pensize(4)
turtle.pencolor('red');    polygon(3, 100)
turtle.pencolor('orange'); polygon(4, 100)
turtle.pencolor('blue');   polygon(5, 100)
```

In [96]:
```
img()
```

Voici le résultat:



1.5 Contrôle du stylo

Le stylo peut être controlé avec les fonctions:

- `pendown , pd , down`
- `pendup , pu , up`
- `pensize , width`

L'état du stylo est obtenu avec la fonction `pen`

In [99]:
```
turtle.pen()
```

Out[99]:
```
{'shown': True,
 'pendown': True,
 'pencolor': 'red',
 'fillcolor': 'red',
 'pensize': 1,
 'speed': 3,
 'resizemode': 'noresize',
 'stretchfactor': (1.0, 1.0),
 'shearfactor': 0.0,
 'outline': 1,
 'tilt': 0.0}
```

1.6 Les couleurs

Les couleurs pour le stylo et le remplissage peuvent être lu et écrits pare les fonctions:

- `pencolor`
- `fillcolor`

Les couleurs peuvent être indiqué par

- un string avec un nom: 'red', 'violet', 'pink')
- un tuple RGB : (255, 0, 0), (255, 0, 255)
- un string avec une valeur RGB hexadécimale: `'#ff00ff'`

In [108]:
```
turtle.pencolor('#ff00ff')
turtle.fd(50)
```

In [102]:
```
turtle.pencolor('blue')
turtle.fd(50)
```

In [103]:
```
turtle.pensize(5)
```

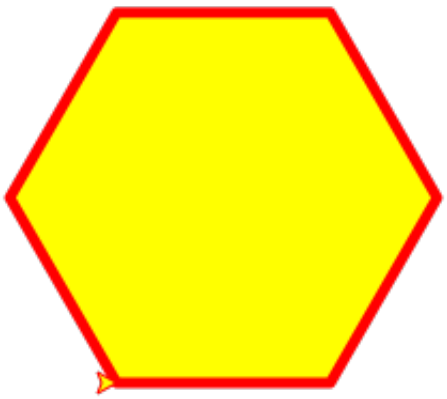
In [105]:
```
turtle.pencolor('pink')
turtle.fd(50)
```

In [109]:
```
turtle.pencolor()
```

Out[109]:
```
(1.0, 0.0, 1.0)
```

In [147]:
```
turtle.reset()
turtle.setpos(0, -100)
turtle.color('red', 'yellow')
turtle.pensize(5)
turtle.begin_fill()
polygon(6, 100)
turtle.end_fill()
```

Voici le résultat:



1.7 Remise à l'état initial

Les fonctions suivantes permettent de revenir:

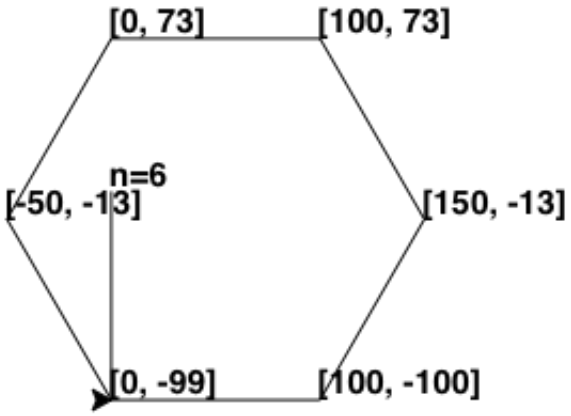
- `home` place la tortue à l'origine
- `clear` efface l'écran, sans bouger la tortue
- `reset` place la tortue à l'orine et efface l'écran

1.8 Ecrire un texte

La fonction `write` permet de placer du texte sur le canvas. Dans l'exemple ci-dessous nous dessinons un hexagone et nous plaçons les coordonnées à chaque sommet.

In [115]:
```
turtle.reset()
n = 6
turtle.write('n='+str(n), font=('Arial', 16, 'bold'))
turtle.setpos(0, -100)

for i in range(n):
    turtle.fd(100)
    turtle.lt(360/n)
    pos = [int(i) for i in turtle.pos()]
    turtle.write(pos, font=('Arial', 16, 'bold'))
```

Voici le résultat:



1.9 Dessiner des étoiles

Dessinons quelques étoiles, en forme d'asterisque.

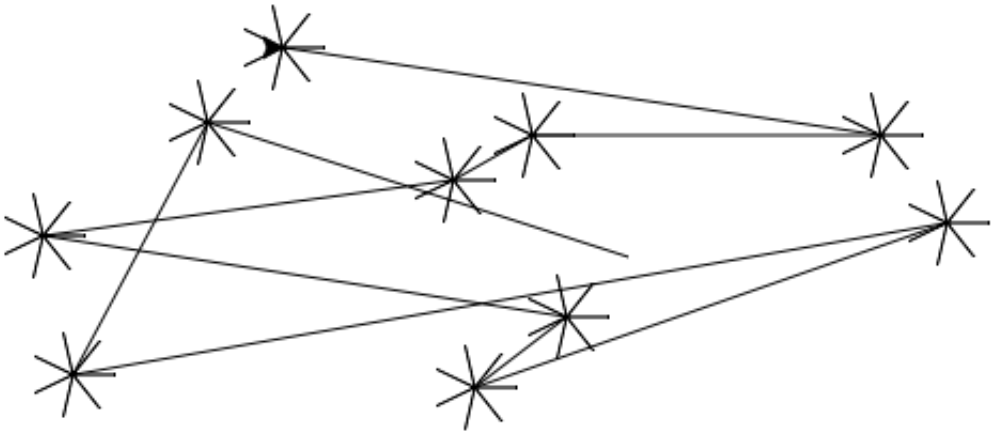
In [20]:
```
def star(n, a):
    for i in range(n):
        turtle.fd(a)
        turtle.bk(a)
        turtle.lt(360/n)
```

In [24]:
```
star(7, 50)
```

Le module `random` permet de choisir des valeurs aléatoires. `randint(a, b)` retourne une valeur entière aléatoire dans l'intervalle [a, b].

In [150]:
```
import random

turtle.reset()
for i in range(10):
    x = random.randint(-300, 300)
    y = random.randint(-100, 100)
    turtle.goto(x, y)
    star(7, 20)
```

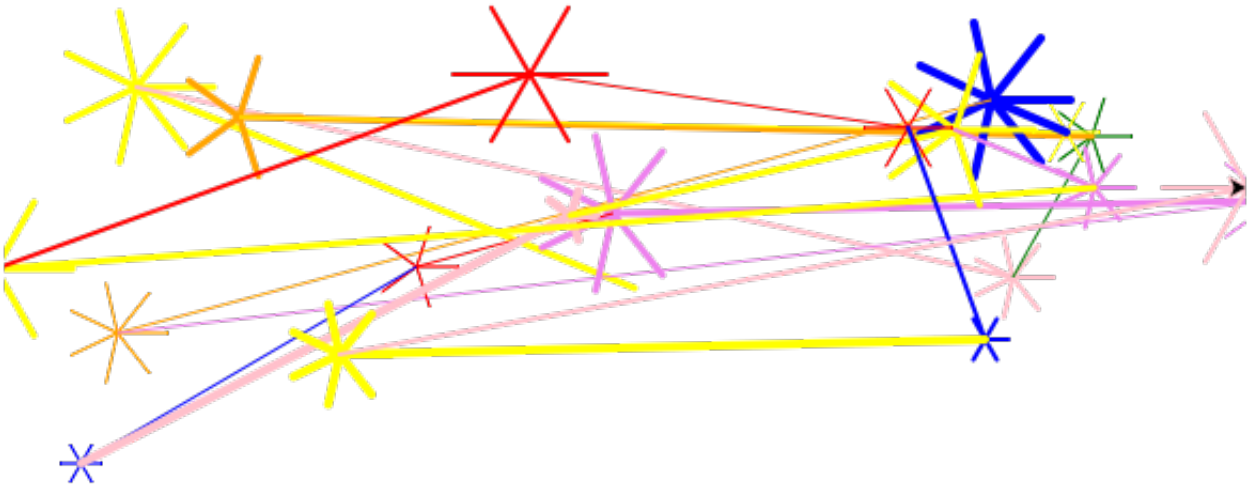
Voici le résultat:



Nous allons rendre le script plus intéressant en choisissant:
* des couleurs aléatoires
* des tailles d'étoile aléatoires
* un nombre de branches aléatoire

In [153]:
```
import random

colors = ('red', 'blue', 'pink', 'orange', 'yellow', 'green', 'violet'
turtle.reset()
for i in range(20):
    x = random.randint(-300, 300)
    y = random.randint(-100, 100)
    n = random.randint(5, 7)
    a = random.randint(10, 40)
    s = random.randint(1, 4)
    color = colors[random.randint(0, 6)]

    turtle.pencolor(color)
    turtle.pensize(s)
    turtle.goto(x, y)
    star(n, a)
```

Voici le résultat:



Le module turtle permet aussi d'afficher une fenêtre de dialogue pour demander une valeur textuelle ou numérique.

In [39]:
```
p1 = turtle.textinput('Player 1', 'Name of first player')
p2 = turtle.textinput('Player 2', 'Name of second player')
```

In [40]:
```
p1, p2
```

Out[40]:
```
('Raphael', 'Eleonore')
```

1.10 Le package turtledemo

En suivant le lien ci-dessous vous trouvez une série d'exemples faits avec le module `turtle`

https://docs.python.org/3.7/library/turtle.html#module-turtledemo
(https://docs.python.org/3.7/library/turtle.html#module-turtledemo)

Il suffit d'ouvrir un terminal et y entrer

```
python -m turtledemo.bytedesign
```