# Activité Prise en main python

## Partie 1 : variables, int, float, str, list

Un algorithme manipule des données. Ces données ne sont pas connues au moment où on écrit l'algorithme. Les variables servent à nommer ces données afin de pouvoir écrire cet algorithme. On procède la plupart du temps dans l'ordre suivant :

1. On écrit l'algorithme.
2. On affecte des valeurs aux variables.
3. On exécute l'algorithme.

Quelques exemples à essayer autour des variables :

```
i = 3                       # entier = type numérique (type int)
r = 3.3                     # réel   = type numérique (type float)
s = "exemple"               # chaîne de caractères = type str (exemple n'est
                            # pas une variable)
s = 'exemple'               # " et ' peuvent être utilisées pour définir une
                            # chaîne de caractères
sl = """ exemple sur
plusieurs lignes"""         # on peut définir une chaîne sur plusieurs lignes #avec """ ou '''
n = None                      # None signifie que la variable existe mais #qu'elle ne contient
rien
                            # elle est souvent utilisée pour signifier qu'il   #n'y a pas de
résultat
                            # car... une erreur s'est produite, il n'y a pas #de résultat
                           # (racine carrée de -1 par exemple)
print (i,r,s,n, sl)

v = "anything"          # affectation
print ( v )             # affichage
v1, v2 = 5, 6           # double affectation
v1,v2
print ( v1 ,v2 )
```

## Partie 2 : Tests

Les tests permettent de faire un choix : selon la valeur d'une condition, on fait soit une séquence d'instructions soit une autre.

```
v = 2
if v == 2 :
    print ("v est égal à 2")
else :
    print ("v n'est pas égal à 2")
```

résultat :

```
v est égal à 2
```

La clause `else` n'est obligatoire :

```
v = 2
if v == 2 :
    print ("v est égal à 2")
```

Résultat :

```
v est égal à 2
```

(Sinon rien ne s'affiche)

Plusieurs tests enchaînés :

```
v = 2
if v == 2 :
    print ("v est égal à 2")
elif v > 2 :
    print ("v est supérieur à 2")
else :
    print ("v est inférieur à 2")
```

Résultat :

```
v est égal à 2
```

Vous pouvez vous amuser à changer la valeur de v

## Partie 3 : boucles

Les boucles permettent de répéter un nombre fini ou infini de fois la même séquence d'instructions. Quelques exemples à essayer autour des boucles :

```
for i in range (0, 10) :    # on répète 10 fois
    print ("dedans",i)      # l'affichage de i
    # ici, on est dans la boucle
# ici, on n'est plus dans la boucle
print ("dehors",i)                  # on ne passe par 10
```

Résultat :

```
dedans 0
dedans 1
dedans 2
dedans 3
dedans 4
dedans 5
dedans 6
dedans 7
dedans 8
dedans 9
dehors 9
```

Boucle while :

```
i = 0
while i < 10 :
    print (i)
    i += 1
```

Résultat :

```
0
1
2
3
4
5
6
7
8
9
```

Interrompre une boucle :

```
for i in range (0, 10) :
    if i == 2 :
        continue            # on passe directement au suivant
    print (i)
    if i > 5 :
        break               # interruption définitive
```

Résultat :

```
0
1
3
4
5
6
```

**Parcours d'une liste :** observer les différences entre les trois écritures

1

```
l = [ 5, 3, 5, 7 ]
for i in range (0, len(l)) :
    print ("élément ",i, "=", l [ i ] )
```

Résultat1 :

```
élément  0 = 5
élément  1 = 3
élément  2 = 5
élément  3 = 7
```

2

```
l = [ 5, 3, 5, 7 ]
for v in l :
    print ("élément ", v )
```

Résultat2 :

```
élément  5
élément  3
élément  5
élément  7
```

3

```
l = [ 5, 3, 5, 7 ]
for i,v in enumerate(l) :
```

```
    print ("élément ",i, "=", v )
```

Résultat3 :

```
élément  0 = 5
élément  1 = 3
élément  2 = 5
élément  3 = 7
```

Que fait le programme suivant ?

```
l = [ 4, 3, 0, 2, 1 ]
i = 0
while l[i] != 0 :
    i = l[i]
    print (i)             # que vaut l[i] à la fin ?
```

Résultat : (ne pas tricher directement…)

```
4
1
3
2
```

Il est possible de créer des fonctions pour ne pas avoir à réécrire le même code plusieurs fois, un exemple, une fonction qui retourne le nombre le plus grand parmi 2 nombres :

```
def quiEstLePlusGrand(a,b) :
        if a<b :
                return b
        else :
                return a

print(quiEstLePlusGrand(2,3))
print(quiEstLePlusGrand(5,3))
print(quiEstLePlusGrand(8,3))
print(quiEstLePlusGrand(24,123))
```

Résultat :

```
3
5
8
123
```

Dès que la fonction est écrite, on peut l'appeler en écrivant son nom avec double parenthèse, et le code à l'intérieur s'exécute, ici c'est un exemple simple.

On peut également utiliser des bibliothèques avec des imports, celle-ci ont été créées par d'autres personnes, et son remplies de fonction utile par tel que la bibliothèque random qui permet de générer des nombres aléatoires.

Source : http://www.xavierdupre.fr

# T.P. Utilisation du module Turtle

Le module turtle permet de tracer facilement des dessins en Python. Il s'agit de commander une tortue à l'aide d'instructions simples comme « avancer », « tourner ». . . C'est le même principe qu'avec Scratch, avec toutefois des différences : tu ne déplaces plus des blocs, mais tu écris les instructions.

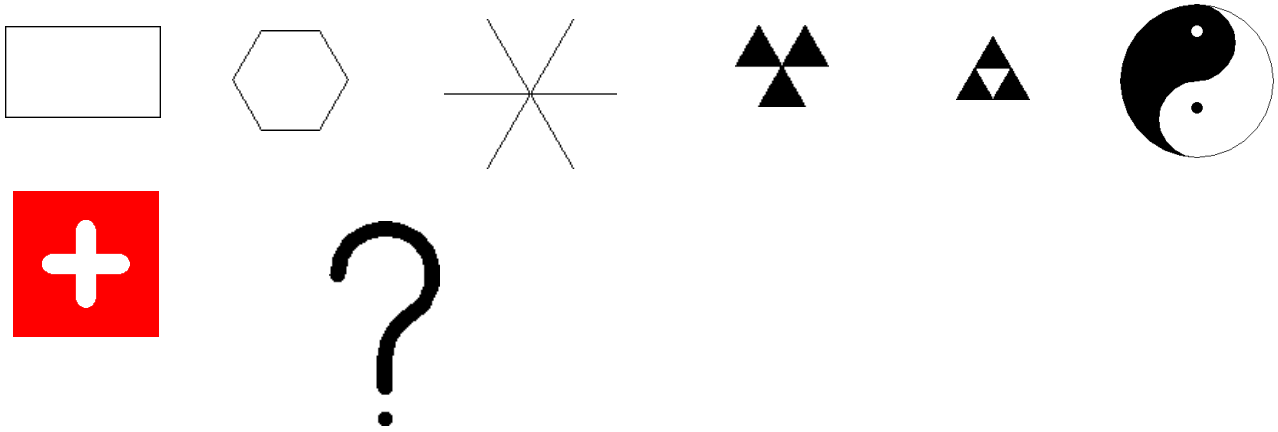
## Activité 1 : Premier pas.

Tester les programmes suivants :

```
from turtle import *
forward(100)
left(90)

forward(50)
width(5)

forward(100)
color('red')

right(90)
```

```
from turtle import *
width(6)

color(0.2,0.2,0.2)
goto(60,0)
goto(60,110)
goto(0,110)
goto(0,0)
up() #
goto(5,5) down()
width(1)
fillcolor("grey")
begin_fill()
goto(55,5)
goto(5,105)
goto(55,105)
goto(5,5)
end_fill()
```

## Activité 2 : Reproduire les formes suivantes (ou quelques-unes)



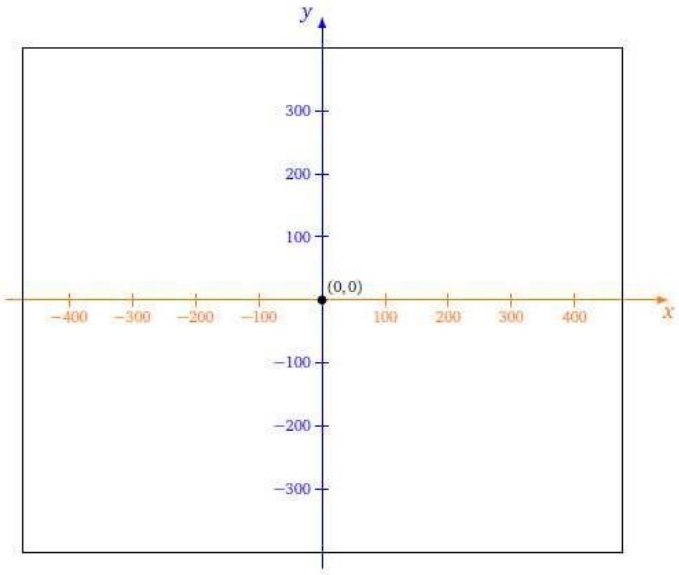
## Activité 3 : Dessiner le drapeau français

# Fiche Méthode Turtle

Voici une liste des principales commandes,
accessibles après avoir écrit :
from turtle import *

• forward(longueur) avance d'un certain nombre de pas
• backward(longueur) recule
• right(angle) tourne vers la droite (sans avancer) selon un angle donné en degrés
• left(angle) tourne vers la gauche
• circle( $r, a$ ) traceun arc de cercle d'angle $a$ en degré et de rayon $r$ .
• dot( $r$ ) tracer un point de rayon $r$ .
• setheading(direction) s'oriente dans une direction (0 = droite, 90 = haut, -90 = bas, 180 =gauche)
• goto $(x, y)$ se déplace jusqu'au point $(x, y)$
• setx( $newx$ ) change la valeur de l'abscisse
• sety( $newy$ ) change la valeur de l'ordonnée
• down() abaisse le stylo
• up() relève le stylo
• width(epaisseur) change l'épaisseur du trait
• color(couleur) change la couleur : "red", "green", "blue", "orange", "purple". . .
On peut aussi définir une couleur en fpurnissant un triple de nombre compris entre 0 et 1 , indiquant le nombre réspectif de vert de rouge et de bleu. Par exemple $color(1,0,0)$ donne du rouge. $Color(0,5,0,0,6)$donne du violet.
• fillcolor(couleur) sélectionne une couleur de remplissage puis begin_fill() active le remplissage etend_fill() désactive le remplissage .
• position() renvoie la position ( $x, y$ ) de la tortue
• heading() renvoie la direction angle vers laquelle pointe la tortue
• towards( $x, y$ ) renvoie l'angle entre l'horizontale et le segment commençant à la tortue et finissantau point ( $x, y$ )
• speed(s) définie la vitesse s de déplacement de la tortue (un nombre entre 1 et 10)
• reset() tout effacer et recommencer à 0.
• ht( ) ne montre plus la tortue.
• exitonclick() termine le programme dès que l'on clique
• title(titre ) donne un titre au dessin.
Les coordonnées de l'écran par défaut vont de - 475 à $+475$ pour les $x$ et de - 400 à $+400$ pour les $y$ ;

$(0, 0)$ est au centre de l'écran.



# TP Turtle et fonction

## Exercice 1 :

Tapez le programme suivant, enregistrez le et exécutez le , puis commenter le :

```
from turtle import *
a=0
while a<12:
   a=a+1
   forward(150)
   left(150)
```

## Exercice n°2 :

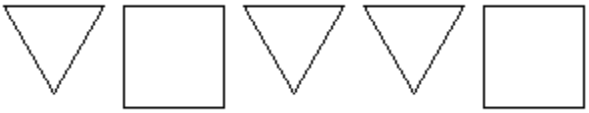
1. Écrire la fonction triangle1(n)qui dessine un triangle équilatéral dont les côtés sont delongueur n et qui a la pointe vers le haut.
2. Écrire la fonction triangle2(n)qui dessine un triangle équilatéral dont les côtés sont delongueur n et qui a la pointe vers le bas.

## Exercice n°3 :

1. Écrire la fonction carre(a) qui trace un carré de côté a. Il est préférable que la tortuetermine son dessin là où elle a démarré et avec la même orientation.
2. En déduire la fonction ligne_de_carres(a,n) qui trace n carrés sur une ligne chaquecarré étant de côté a (on utilisera la fonction carre)

## Exercice n°4 :

Réaliser le dessin suivant,



Le dessin ci-dessous correspond à l'appel de la fonction FiguresPleines(4,20), le premier paramètre est le nombre de répétitions et le deuxième correspond la longueur du côté du triangle et du carré.

Vous maitrisez les éléments de bases et pouvez passer sur la PI PICO