

# Turtle

Avancer et tourner

Des formes complexes

Crayon et couleurs


Boucles

Fonctions

Oeuvre d'art

## ^ < Avancer et tourner >

### ^ < Exercice 1 : Avancer, tourner >

Allez sur [cette page](#) dans un nouvel onglet ou une nouvelle fenêtre, écrivez le code suivant et cliquez sur  !

```
forward(50)
left(90)
forward(100)
right(45)
```

En français, on peut traduire *forward* par *avancer* et *left/right* par *gauche/droite*.

### ^ < Exercice 2 : raccourcis >

Écrire `forward`, c'est fatigant, il existe le raccourci `fd`, de même on a `lt` et `rt` qui veulent dire **L**eft **T**urn et **R**ight **T**urn, que l'on traduirait par *Tourner à Gauche* et *Tourner à Droite*.

Que fait ce code ?

```
fd(100)
lt(120)
fd(100)
lt(120)
fd(100)
```

Et celui-ci ?

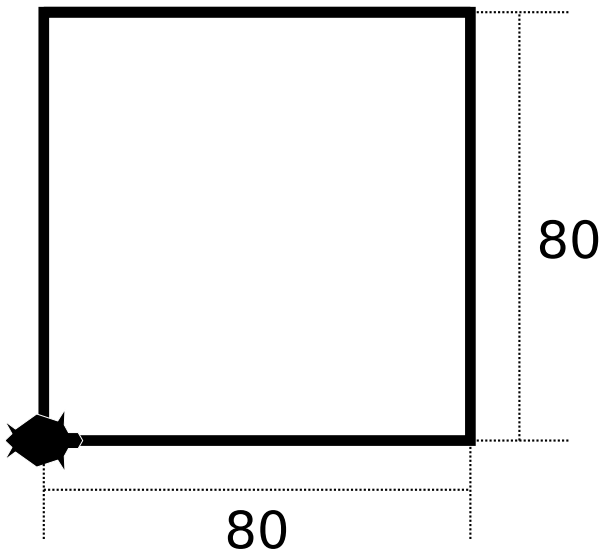
```
fd(100)
lt(180)
fd(200)
lt(180)
fd(100)
```

```
lt(90)
fd(100)
lt(180)
fd(200)
```

## ^ < Formes complexes >

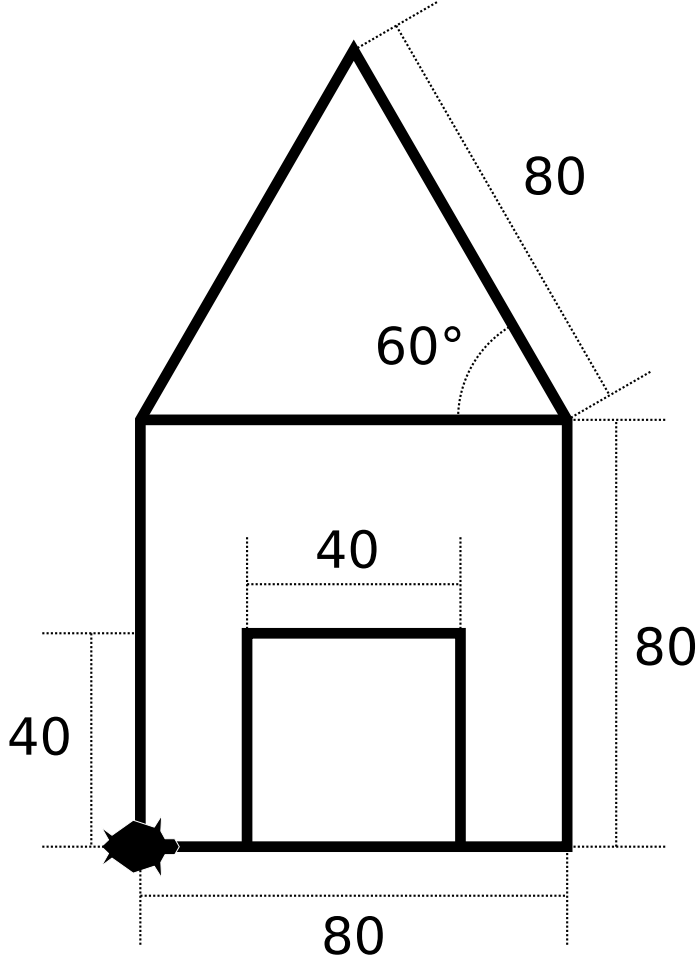
### ^ < Exercice 3 : Le carré >

Maintenant, tu vas écrire le code ! Essaie de dessiner un **carré** de côté 80.

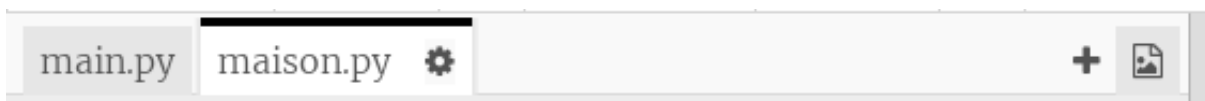


### ^ < Exercice 4 : La maison >

Plus difficile, une maison :



Sauve ce code dans un nouveau fichier pour pouvoir le ré-utiliser plus tard !



## ^ < Crayon et couleurs >

La nouvelle commande `penup()` — raccourci `pu()` — permet de lever le crayon, ainsi la tortue peut bouger sans dessiner. Ensuite, `pendown()` — raccourci `pd()` — repose le crayon pour plus de dessins.

Que fait ce code ?

```
fd(50)
penup()
fd(100)
pendown()
fd(150)
```

La nouvelle commande `color("red")` (attention aux guillemets dans "red" !) permet de changer la couleur du trait en rouge. D'autres couleurs existent par défaut comme `green`, `blue`, `yellow` et bien d'autres.

Pour pouvoir indiquer toutes les couleurs existantes, on regarde dans Paint :

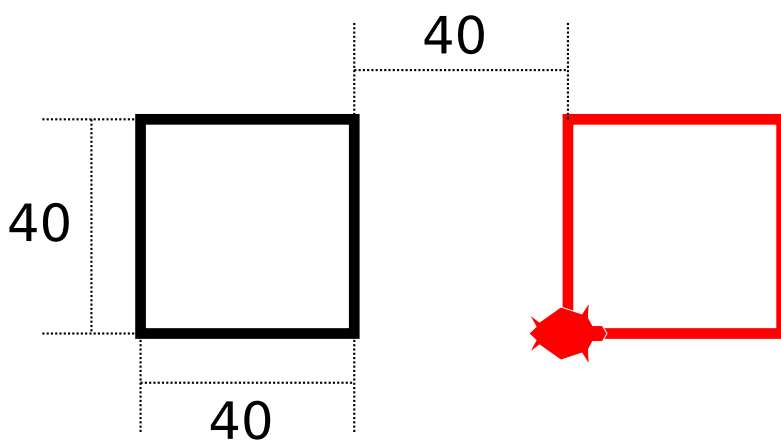


Pour ce magnifique orange, j'écrirai la commande `color(253, 126, 0)`.

```
fd(100)
color("yellow")
fd(100)
color("red")
fd(100)
color(253, 126, 0)
fd(100)
```

### ^ < Exercice 5 : Deux carrés colorés >

Essaie de faire ceci :



### ^ < Boucles >

Dans le code du carré, on avait **quatre fois** le même code :

```
fd(80)
lt(90)

fd(80)
lt(90)

fd(80)
lt(90)

fd(80)
lt(90)
```

Il existe une instruction qui permet de dire *Répète 4 fois ce qui suit*, c'est le **for** :

```
for i in range(4):
    fd(80)
    lt(90)
```

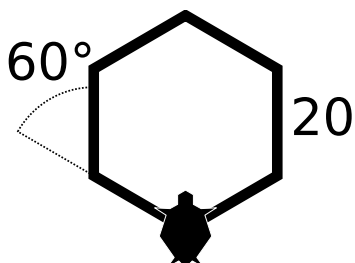
Attention à ne pas oublier le **:**, de passer à la ligne et d'écrire au moins une instruction.

Quelle va être la différence avec le code suivant ? Teste le et comprends pourquoi le résultat affiché est logique.

```
for i in range(4):
    fd(80)
lt(90)
```

## ^ < Exercice 6 : L'hexagone >

Essaie de faire un hexagone, n'hésites pas à utiliser le **for** pour simplifier ton code !



## ^ < Fonctions >

Afin de réutiliser du code, on peut le mettre dans une *fonction*. Par exemple, je vais faire une fonction

carre qui permet de dessiner un carré.

```
def carre():  
    for i in range(4):  
        fd(80)  
        lt(90)
```

Écris ce code en haut du fichier, et puis la nouvelle instruction `carre()` sera disponible !

```
fd(100)  
carre()  
lt(45)  
carre()  
fd(50)  
carre()
```

On peut également mettre des *paramètres* à la fonction, de telle sorte on pourra faire varier une certaine partie du code, comme par exemple, la taille du carré.

```
def carre(taille):  
    for i in range(4):  
        fd(taille)  
        lt(90)
```

Que l'on utilisera comme ceci :

```
fd(100)  
carre(80)  
lt(45)  
carre(40)  
fd(50)  
carre(60)
```

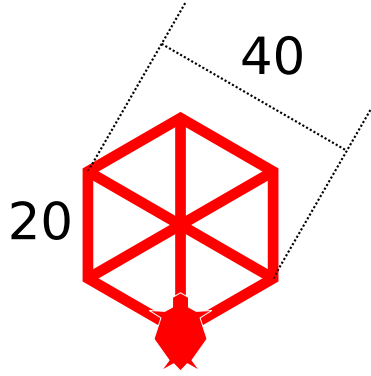
On peut mettre autant de paramètres que l'on veut, séparés par des virgules, comme `def carre(taille, couleur)`

On peut faire des opérations mathématiques comme `50 + taille * 2 - (taille - 1)`, attention à la priorité des opérations !

### ^ < Exercice 7 : Des triangles >

Faites une fonction `triangles`, qui va dessiner la forme suivante, mettez un paramètre `taille` et `couleur` afin de pouvoir l'appeler comme ceci :

```
triangles(20, "red")
```

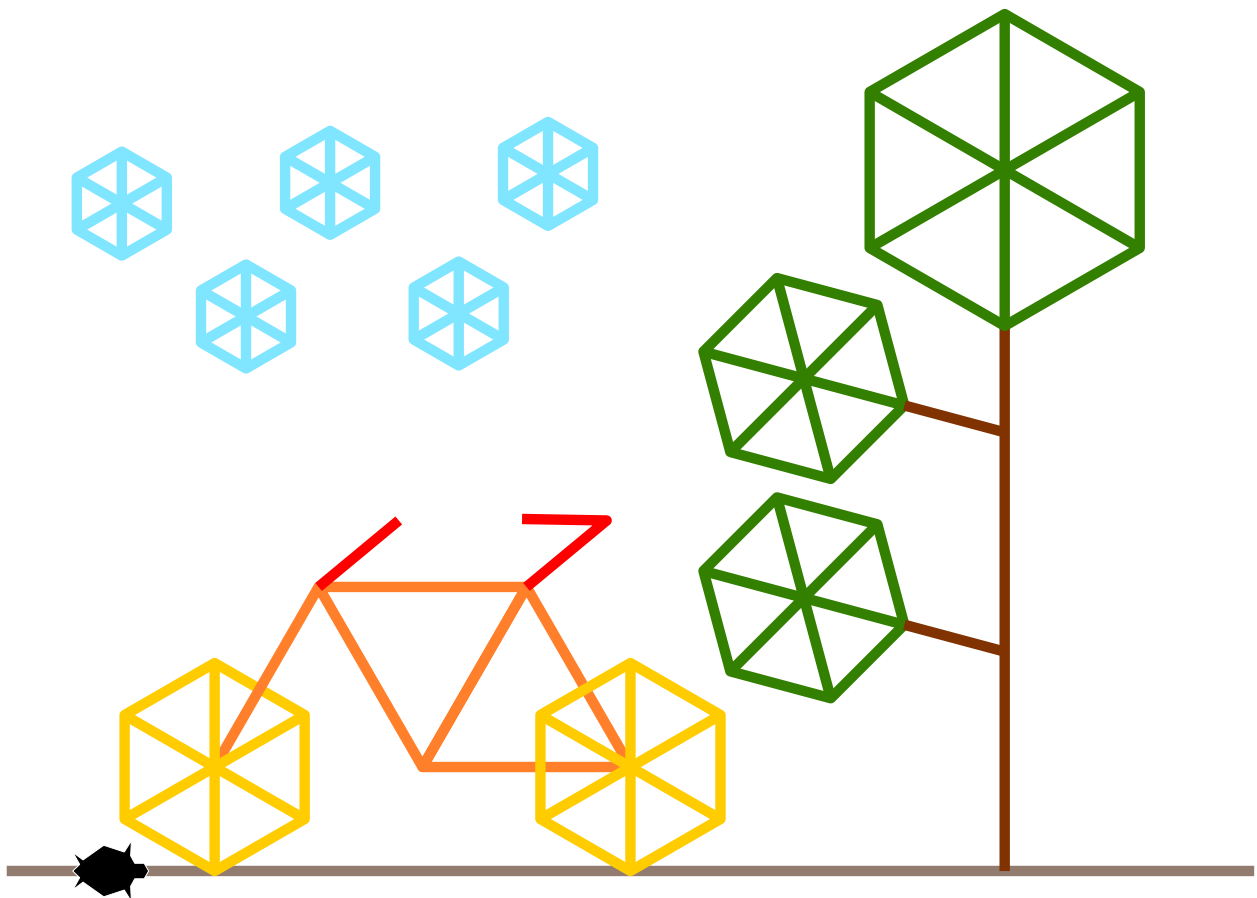


## ^ < Oeuvre d'art >

### ^ < Exercice 8 : Votre oeuvre d'art >

Dernier défi pour ce cours, laisse aller ton imagination pour créer un beau dessin !

Voici le mien, il utilise une fonction `triangles` plusieurs fois !



Voilà, nous avons appris à utiliser *turtle* en programmant en langage *python*. Il existe [d'autres commandes turtle](#).

## ^ < Programmation de jeux vidéos >

Dès la 3ème, le parascolaire de programmation de jeux vidéos vous apprendra à faire de [petits jeux 2D](#) !

## ^ < Pour en savoir plus

Voici un aperçu d'autres fonctionnalités turtle/python.

On peut créer plusieurs tortues (*objets* python) :

```
bob = Turtle()
bob.clor("blue")
bob.forward(50)
alice = Turtle()
alice.color("green")
alice.fd(50)
```

Et créer des variables en fonction de la tortue.

```
bob.deplacement = 5
alice.deplacement = 10

def avancer_en_traits(tortue):
    for i in range(3):
        tortue.penup()
        tortue.fd(tortue.deplacement)
        tortue.pendown()
        tortue.fd(tortue.deplacement)

avancer_en_traits(bob)
avancer_en_traits(alice)
```

Ou créer des listes de tortues :

```
liste = [bob, alice]
charles = Turtle()
charles.deplacement = 7
liste.append(charles) # ajouter à la fin

for tortue in liste: # pour chaque tortue dans la liste
    ancienne_couleur = tortue.color() # on sauve la couleur
    tortue.color("blue")
    avancer_en_traits(tortue)
    tortue.color(ancienne_couleur) # on réapplique l'ancienne couleur
```

Au revoir et pour toute question, je suis disponible par mail via [robertvandeneynde@hotmail.com](mailto:robertvandeneynde@hotmail.com)