De scratch à python : 4

La programmation par évènements :

Gérard Swinnen explique très concrêtement, dans son livre « Apprendre à programmer avec python 3 », la différence entre un programme en « mode texte » et un programme « piloté par les évènements ». Je reprends ici ses explications, en en modifiant quelques termes.

Tous les programmes d'ordinateur comportent *grosso-modo* trois phases principales : ***une phase d'initialisation***, laquelle contient les instructions qui préparent le travail à effectuer (appel des modules externes nécessaires, ouverture de fichiers, connexion à un serveur de bases de données ou à l'Internet, etc.), ***une phase centrale*** où l'on trouve la véritable fonctionnalité du programme (c'est-à-dire tout ce qu'il est censé faire : afficher des données à l'écran, effectuer des calculs, modifier le contenu d'un fichier, imprimer, etc.), et enfin ***une phase de terminaison*** qui sert à clôturer « proprement » les opérations (c'est-à-dire fermer les fichiers restés ouverts, couper les connexions externes, etc.).

Dans un programme « **en mode texte** », ces trois phases sont simplement organisées suivant un schéma *linéaire* comme dans l'illustration ci-dessous.

En conséquence, ces programmes se caractérisent par une *interactivité très limitée* avec l'utilisateur. Celui-ci ne dispose pratiquement d'aucune liberté : il lui est demandé de temps à autre d'entrer des données au clavier, mais toujours dansun ordre prédéterminé correspondant à la séquence d'instructions duprogramme.

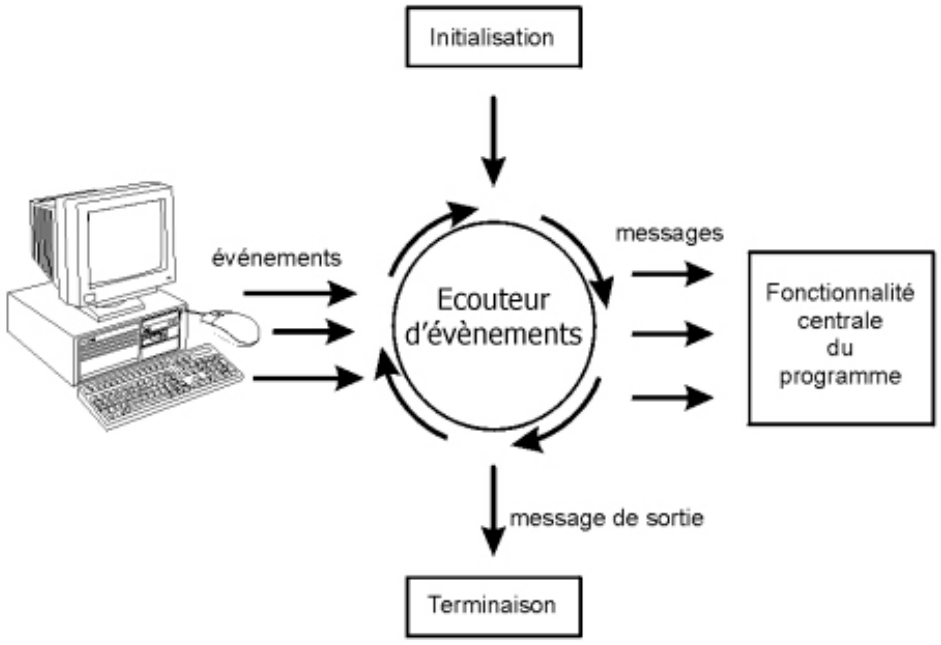
Dans le cas d'un programme qui utilise une interface graphique, par contre, l'organisation interne est différente. On dit d'un tel programme qu'il est ***piloté par les événements***. Après sa phase d'initialisation, un programme de ce type se met en quelque sorte « en attente », et passe la main à un autre logiciel, lequel est plus ou moins intimement intégré au système d'exploitation de l'ordinateur et « tourne » en permanence dans une « **boucle d'exécution** ».

Un « **écouteur d'évènements** » est mis en place et il scrute sans cesse tous les périphériques (clavier, souris, horloge, modem, etc.) et réagit immédiatement lorsqu'un événement y est détecté. Un tel événement peut être une action quelconque de l'utilisateur : déplacement de la souris, appui sur une touche, etc., mais aussi un événement externe ou un automatisme (top d'horloge, par exemple).

Lorsqu'il détecte un événement, l'écouteur d'évènements envoie **un message spécifique** au programme, **lequel doit être conçu pour réagir en conséquence**.
La phase d'initialisation d'un programme piloté par les évènements comporte un ensemble d'instructions qui définissent les messages d'événements qui devront être pris en charge : on peut en effet décider que le programme ne réagira qu'à certains événements en ignorant tous les autres.

Alors que dans un programme « textuel », la phase centrale est constituée d'une suite d'instructions qui décrivent à l'avance l'ordre dans lequel la machine devra exécuter ses différentes tâches (même s'il est prévu des cheminements différents en réponse à certaines conditions rencontrées en cours de route), on trouve dans la phase centrale d'un programme piloté par les évènements un ensemble de fonctions indépendantes.

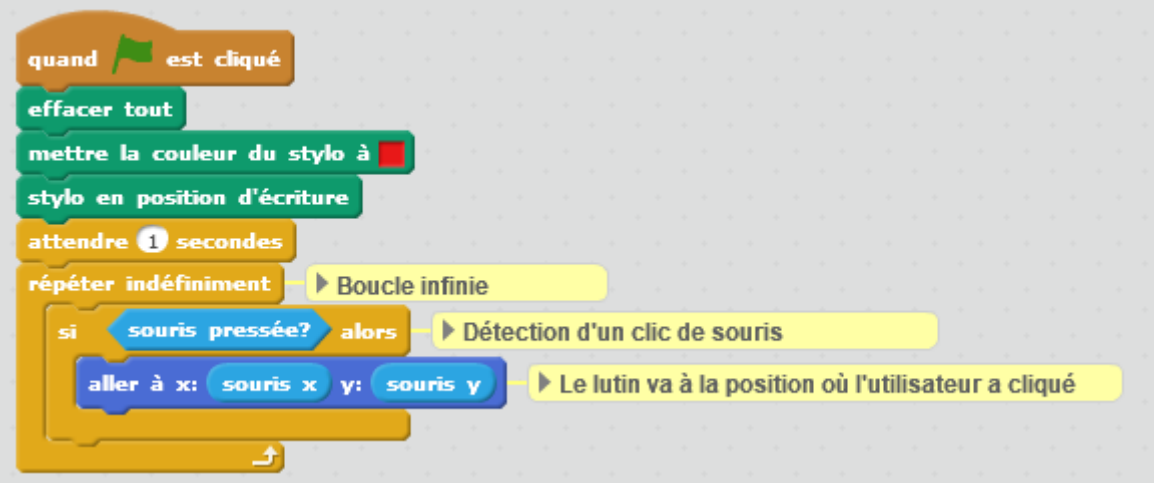
Chacune de ces fonctions est appelée spécifiquement lorsqu'un événement particulier est détecté par le système d'exploitation : elle effectue alors le travail que l'on attend du programme en réponse à cet événement, et rien d'autre.
Il est important de bien comprendre ici que pendant tout ce temps, l'écouteur d'évènements continue à « tourner » et à guetter l'apparition d'autres événements éventuels.
S'il arrive d'autres événements, il peut donc se faire qu'une deuxième fonction (ou une 3e, une 4e...) soit activée et commence à effectuer son travail « en parallèle » avec la première qui n'a pas encore terminé le sien. C'est ce qu'on appelle le multitâche.



Exercice 1 :

Créer un programme dans lequel le lutin se déplace là où l'utilisateur a cliqué.

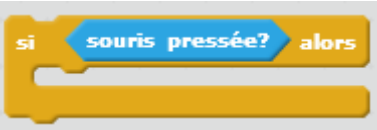
Il trace un trait entre sa position de départ et sa position d'arrivée.



Dans ce script, on remarquera l'utilisation d'une boucle infinie, qui englobe toutes les instructions du « jeu ».

On arrêtera cette boucle infinie en cliquant sur le drapeau rouge.

Les éléments du panneau « Capteurs » de Scratch, permettent de « capturer » un évènement particulier que le programme doit traiter.



capture l'évènement « Souris pressée » et les instructions placées à l'intérieur du « Si » traite l'évènement.

Un évènement concernant la souris, s'accompagne toujours des coordonnées x et y de la souris au moment où l'évènement survient.

Avec Python

Les évènements que la « tortue » peut recevoir

Le module « turtle » permet de détecter l'appuie sur les touches du clavier ou un clic sur l'un des 3 boutons de la souris.

La boucle d'évènements :

Lorsqu'on utilise le module « turtle » et qu'on utilise une des fonctions spécifiques à la tortue, ou spécifique à l'écran sur lequel évolue la tortue, un objet Screen (écran) et/ou un objet Turtle (tortue) sont automatiquement créés.

Cet objet écran écoute les évènements tel qu'un clic de la souris avec l'un des 3 boutons,

Pour bien comprendre comment sont gérer les évènements avec le module « turtle », nous allons créer nous même un objet Screen sur lequel la tortue va évoluer et un objet Turtle.

Nous n'utiliserons pas les objets créés par défaut, comme nous l'avons fait jusqu'à présent.

Pour notre exercice, nous devons traiter l'évènement onclick reçu par l'écran.

Pour ce faire nous devons définir une fonction qui va gérer l'évènement, c'est-à-dire faire le travail en réponse à cet évènement.

Nous devons ensuite, lier l'évènement à son gestionnaire : lorsque l'évènement va se produire, le gestionnaire de cet évènement sera appelé pour le traiter.

ecran.onclick(nom du gestionnaire de l'évènement onclick, bouton de la souris utilisé)

Le programme

```
#Gestionnaire de l'évènement onclick
def h1(x, y):          #x, y sont les coordonnées de la tortue au moment où l'on clique
   marie.goto(x, y) #la tortue marie se déplace là où l'on a cliqué
#-----------------------------------------------
import turtle
#Création d'un objet Screen responsable de la détection et de la gestion
#de l'évènement "souris pressée": onclick
ecran=turtle.Screen()
ecran.title("Evènements souris")
ecran.bgcolor("lightgreen")
#-----------------------------------------------
#Création d'une tortue sur laquelle va agir le gestionnaire de clic
marie=turtle.Turtle()
marie.color("purple")
marie.pensize(3)
marie.shape("circle")
#-----------------------------------------------
```

```
#Liaison de l'objet ecran qui reçoit l'évènement onclick au gestionnaire
#chargé de traiter cet évènement.
ecran.onclick(h1,1)  # attache le gestionnaire h1 à un clic dans la fenêtre
                     #1 correspond au bouton gauche de la souris
#---------------------------------------------
#boucle infinie qui ne s'arrête que lorsqu'on clic sur le bouton
#de fermeture de la fenêtre
ecran.mainloop()
turtle.bye() #permet de libérer tous les objets créés: tortue, écran
```

Comment cela fonctionne t'il ?

Une fois que l'on a créé nos objets écran et tortue :

ecran.onclick(h1,1)

provoque la prise en compte automatique des évènements onclick, par la fonction h1.

A chaque clic à l'intérieur de la fenêtre, la fonction h1 est appelé et c'est elle qui traite l'évènement en déplaçant la tortue.

La boucle infinie mise en place automatiquement au démarrage du programme, s'arrête lorsque l'utilisateur clique sur le bouton de fermeture de la fenêtre.

L'instruction turtle.bye() qui suit ecran.mainloop() est exécutée dès que la boucle infinie s'arrête. Cette instruction permet de « libérer » tous les objets créés et d'arrêter proprement le programme.

Exercice 2 :

Créer un programme Scratch permettant de déplacer le lutin en utilisant les 4 flèches : haut, bas, vers la droite, vers la gauche.

Dans ce programme nous devons détecter et traiter 5 évènements : clic sur le drapeau vert, appui sur une des 4 flèches de direction.

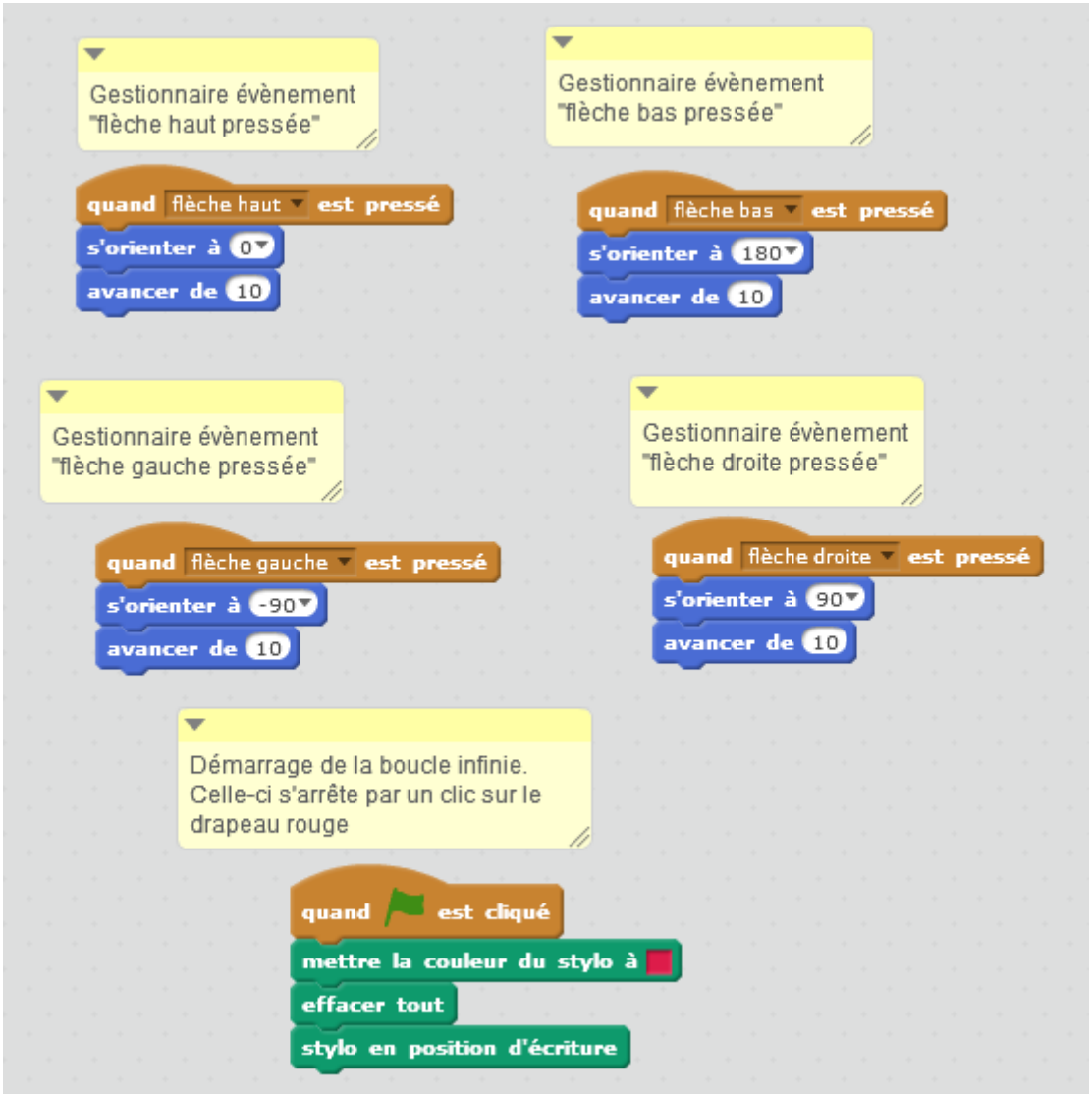
Un **clic sur le drapeau vert** démarre la boucle infinie puis le programme met en place les paramètres du crayon. Cette boucle infinie se terminera par un clic sur le drapeau rouge.

L’appui sur une des 4 flèches de direction sera traité par un gestionnaire particulier.

Les éléments du panneau « **Evènements** » permettent de mettre en place un certain nombre de gestionnaires capables de gérer différents évènements.

Pour les évènements en provenance du clavier, on utilise :

Il suffit de choisir dans le menu la touche du clavier à prendre en compte.



Avec Python

Pour que l'écran sur lequel la tortue évolue puisse écouter les évènements venant du clavier, on doit utiliser une instruction spéciale : listen() (écouter)

Nous devons mettre en place 4 gestionnaires pour traiter les 4 évènements en provenance des flèches.

Pour lier l'écran au gestionnaire d'une touche on utilise :

ecran.onkeypress(Nom du gestionnaire, key="Nom touche")

Pour flèche haut : Up, flèche bas : Down, flèche à droite : Right, flèche à gauche : Left

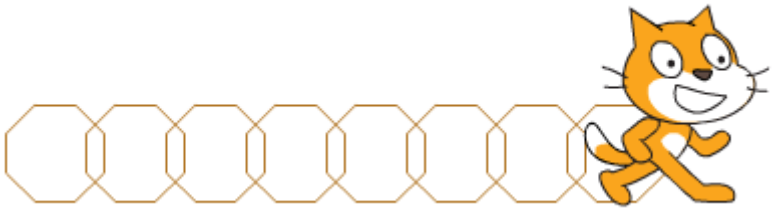
```
# Les 4 fonctions qui suivent sont des gestionnaires d'évènement".
# h1 gère l'évènement "appui sur la flèche vers le haut
def h1():
    marie.setheading(90)
    marie.forward(10)
#-----------------------------------------------------
# h2 gère l'évènement "appui sur la flèche vers la gauche
def h2():
    marie.setheading(180)
    marie.forward(10)
#-----------------------------------------------------
# h3 gère l'évènement "appui sur la flèche vers la droite
def h3():
    marie.setheading(0)
    marie.forward(10)
#-----------------------------------------------------
# h4 gère l'évènement "appui sur la flèche vers le bas
def h4():
    marie.setheading(270)
    marie.forward(10)
#-----------------------------------------------------
```

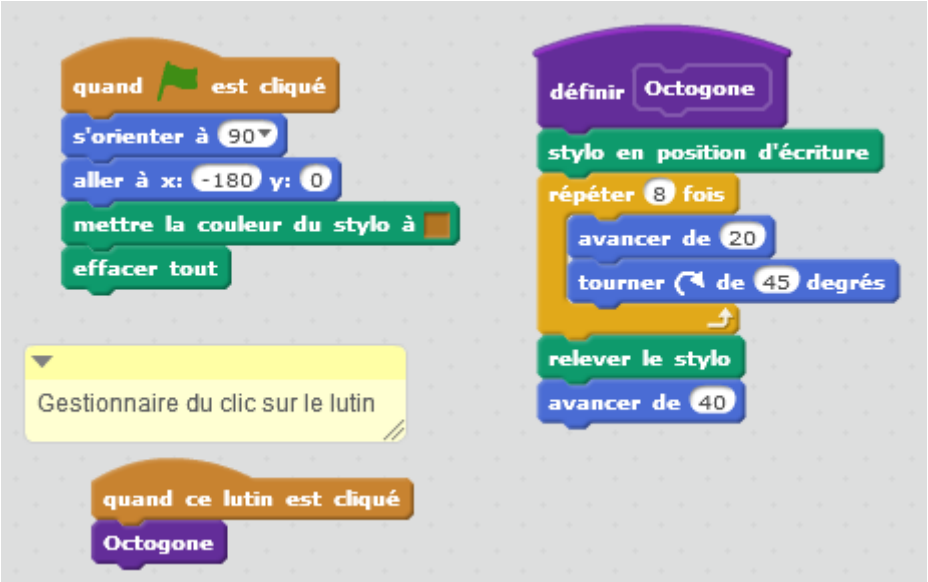
```
import turtle as tu
ecran = tu.Screen()              # Créer un objet écran : une fenêtre
ecran.title("Prise en compte du clavier!")     # Changer le titre de la fenêtre
ecran.bgcolor("lightgreen")           # Colorier le fond de la fenêtre
marie = tu.Turtle()             # Créer la tortue marie
marie.shape("turtle")

# Ces lignes "attachent" (bind en anglais) les évènements "touches enfoncées"
# aux différents gestionnaires que nous avons définis au dessus.
ecran.onkeypress(h1, "Up")
ecran.onkeypress(h2, "Left")
ecran.onkeypress(h3, "Right")
ecran.onkeypress(h4, "Down")

# Nous devons maintenant demander à l'objet fenêtre de commencer à écouter
# les évènements en provenance du clavier.
# Si une touche est enfoncée, l'objet fenêtre recherche le gestionnaire lié à cette touche.
# Si ce gestionnaire existe, alors l'objet fenêtre l'appelle afin qu'il exécute son travail.
ecran.listen()
#---------------------------------
ecran.mainloop()
#On ferme la fenêtre si on a cliqué sur le bouton de fermeture.
tu.bye()
```

Exercice 3 :

Dans un programme scratch lorsqu'on clique sur le lutin, celui-ci dessine un ortogone de 20 pas de côté, et avance de 40 pas à la fin de son tracé.

Faites dessiner au lutin la figure suivante en cliquant 8 fois dessus.

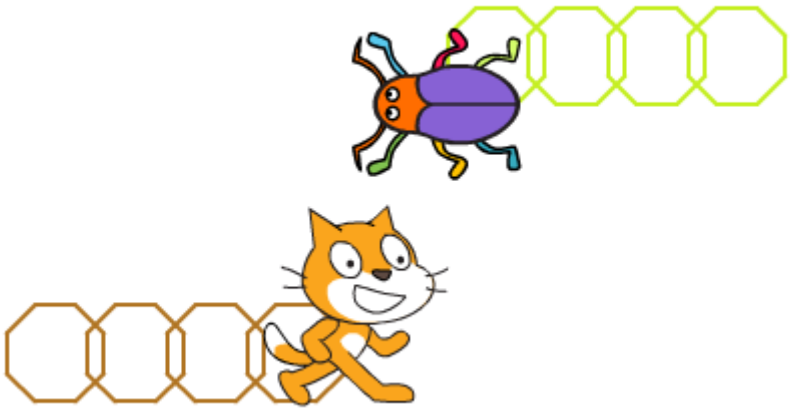




Ici c'est le lutin lui-même qui reçoit l'évènement : on lui a cliqué dessus.

Lorsque cet évènement arrive, le gestionnaire fait dessiner l'ortogone au lutin.

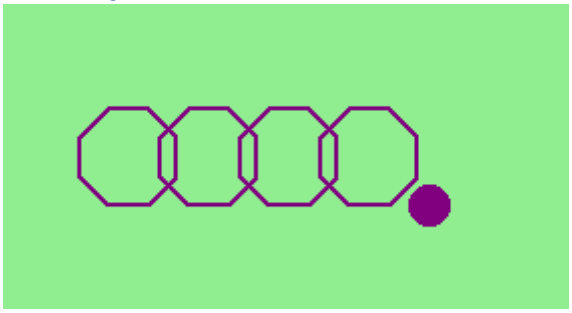
Et avec 2 lutins :



Il suffit de rajouter un deuxième lutin et de faire glisser sur l'image du deuxième lutin, les différents blocs d'instructions du premier. Ces blocs sont dupliqués dans l'environnement du deuxième lutin. Il suffit ensuite de modifier la position de départ et la couleur du stylo.

Ensuite on clique sur l'un ou l'autre des deux lutins.

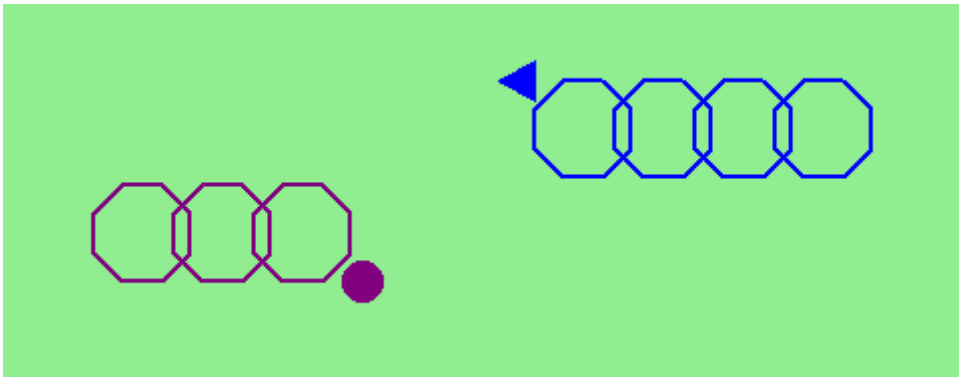
Avec Python



```
#Fonction qui fait tracer un octogone à la tortue marie
def octogone():
    marie.down()
    for i in range(8):
        marie.forward(20)
        marie.left(45)
    marie.up()
    marie.forward(40)
#-----------------------------------------------------
#Gestionnaire appelé lorsque la tortue marie est cliquée.
#Il reçoit obligatoirement deux paramètres correspondant aux coordonnées de la souris
#au moment du clic. Ce gestionnaire fait tracer à marie un octogone
def gestionnaire_marie(x, y):
    octogone()
 #-----------------------------------------------------
import turtle as tu
ecran = tu.Screen()
ecran.bgcolor("lightgreen")
#-----------------------------------------------------
marie = tu.Turtle()
marie.color("purple")
marie.shape("circle")
```

```
marie.pensize(2)
marie.up()
marie.goto(-180,-50)
marie.setheading(0)
#-----------------------------------------------------
#On attache le gestionnaire_marie à l'évènement onclick sur marie
marie.onclick(gestionnaire_marie)
#-----------------------------------------------------
ecran.mainloop()
tu.bye()
```

Et avec deux tortues :



Ici se pose la question : faut-il faire deux fonctions octogones comme dans Scratch (une pour chaque tortue) ou bien peut'on utiliser la même fonction pour les deux tortues ?

Nous allons utiliser la même fonction octogone, en lui passant en paramètre une référence à la tortue à faire bouger. Une référence est un pointeur vers un objet. (On peut imaginer ici, que la fonction octogone pilote les tortue avec une télécommande : la référence est alors la fréquence à utiliser pour piloter une tortue donnée.)

Suivant quelle référence de tortue on lui passe au moment de l'appel, la fonction octogone fera dessiner l'une ou l'autre tortue.

```
#octogone reçoit une référence à l'une des deux tortues
def octogone(tortue):
    tortue.down()
    for i in range(8):
        tortue.forward(20)
        tortue.left(45)
    tortue.up()
    tortue.forward(40)
#-------------------------------------
def gestionnaire_marie(x, y):
    #appel de la fonction octogone en lui passant la référence à la tortue marie
    octogone(marie)

def gestionnaire_alex(x, y):
    #appel de la fonction octogone en lui passant la référence à la tortue alex
    octogone(alex)
#-------------------------------------
import turtle as tu
ecran = tu.Screen()
ecran.bgcolor("lightgreen")
#----------On crée la tortue marie----------
marie = tu.Turtle()
marie.color("purple")
marie.shape("circle")
marie.pensize(2)
marie.up()
marie.goto(-180,-50)
marie.setheading(0)
```

```
#----------On crée la tortue alex----------
alex = tu.Turtle()
alex.color("blue")
alex.shape("triangle")
alex.pensize(2)
alex.up()
alex.goto(180,50)
alex.setheading(180)
#-------------------------------------
#On attache l'évènement onclick sur marie au gestionnaire de marie
marie.onclick(gestionnaire_marie)
#On attache l'évènement onclick sur alex au gestionnaire d'alex
alex.onclick(gestionnaire_alex)
#-------------------------------------
ecran.mainloop()
tu.bye()
```