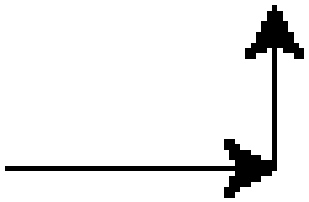
# Quelques perles

## I/ Coordonnées

L'idée du mini-exercice était de rappeler le mouvement des yeux qu'on fait lorsqu'on place un point de coordonnées données :



Seulement pour cela il faut que l'écran graphique de la tortue soit visible, ce qui nécessite que la fenêtre d'IDLE soit suffisamment réduite pour occuper environ la moitié gauche de l'écran, la moitié droite étant alors occupée par l'écran de la tortue. Attention : Une fois que l'écran de la tortue est créé, il attend des clics de souris, ce qui sous Windows XP, a tendance à la bloquer, voire à provoquer un plantage de Python. Le problème semble totalement inexistant sous Linux...

Un seul élève a mal lu les coordonnées (il a trouvé (0 ;30) donc l'ordonnée au moins était bonne...)

## II/ Carrés

La première surprise du TP : Voyant le nombre 50 entre parenthèses à la fin du script, la plupart des élèves ont inventé un théorème de situation : « Toute variable est égale à 50 puisque c'est avec 50 qu'on va tester ». Comme les angles de 50° ne donnent visiblement pas un carré, on « fait avec », et on voit la tortue tourner en rond autour d'un carré :

```
def square(x):
    for n in range(50    ):
        forward(x)
        left(90    )
```

Souvent, du moment qu'on a un carré, on s'en contente... Cependant

1. – *Le prof* : « Ça a combien de côtés, un carré ? »
   – *Les élèves* : « Quatre, voyons ! »
   – *Le prof* : « Chez vous, on dirait plutôt qu'il y en cinquante ! »
   – *Les élèves* : « Ha Ha ! »
2. Dans la question suivante, les élèves qui n'ont pas modifié la définition d'un carré ont été punis par la tortue Python, parce que ces 50 côtés, ils lui ont demandé de les faire souvent, ce qui à la longue prend un temps rédhibitoire...

Quelques élèves ont corrigé leur erreur sur la feuille :

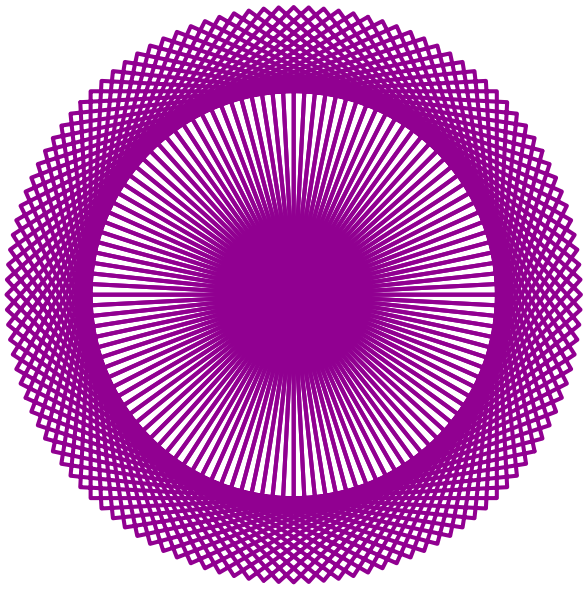
```
def square(x):
    for n in range(4):
        forward(x)
        left(90)
```

La méthode expérimentale a visiblement plus de succès que celle consistant à réfléchir d'abord à ce qu'on fait...

Le dessin de plusieurs carrés a visiblement motivé les élèves par le côté esthétique de la figure. Mais là encore, les élèves ont le plus souvent procédé par tâtonnements ; seuls les plus rapides ont pensé à compter les carrés de la figure. Avec parfois des erreurs d'arithmétique :

```
for n in range(120):
    square(50)
    left(3)
```

ce qui produit la figure que voici :



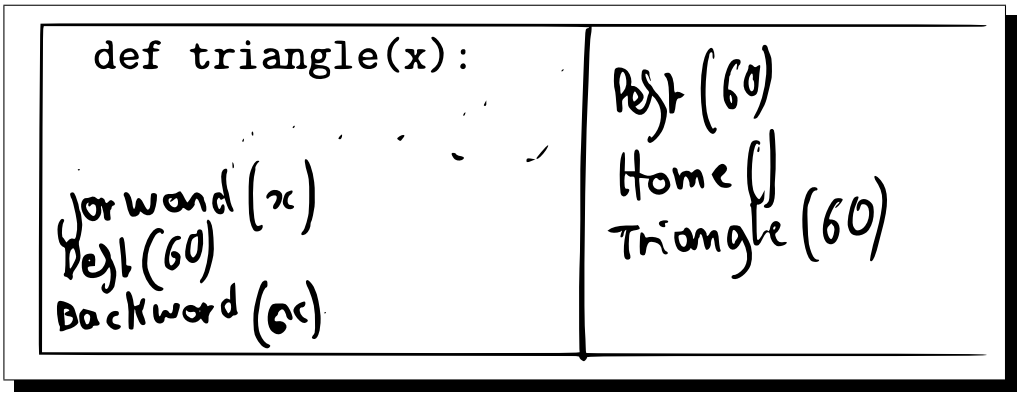
Comme disait Benoît Mandelbrot : « Le bug est un artiste ! »

## III/ Triangles

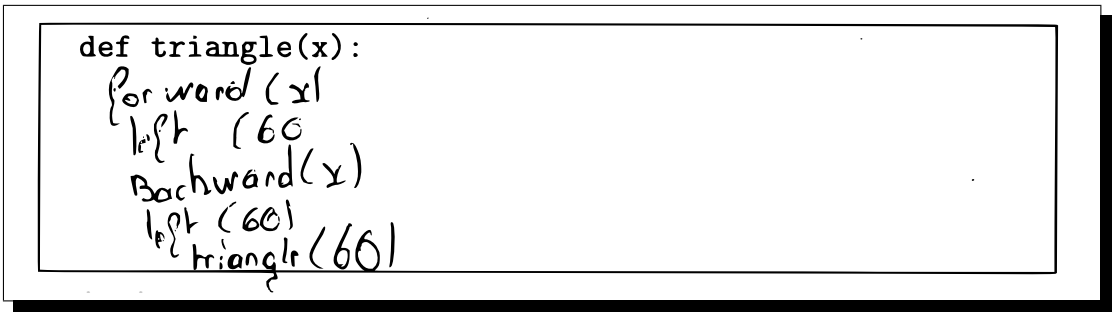
1. **Triangle équilatéral**

   L'erreur attendue était le remplacement de 90° par 60°, qui produit la moitié d'un hexagone au lieu du triangle[1]. Cette erreur a effectivement été fréquente, comme prévu. Ce qui n'a pas été prévu, c'est la manière de la contourner :

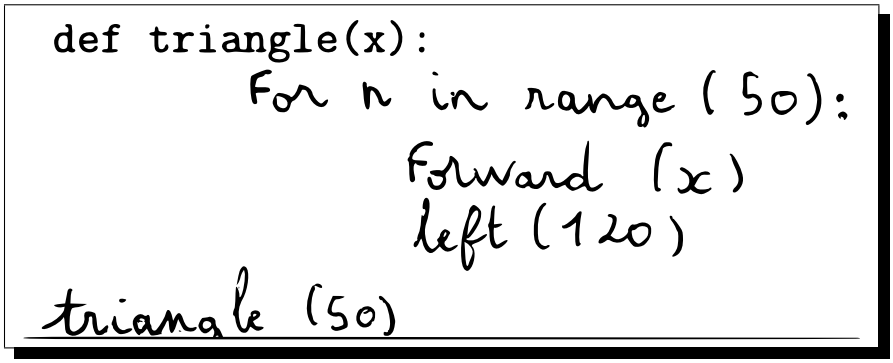
1. C'est une erreur d'orientation, liée au fait que la tortue évolue dans un repère mobile.



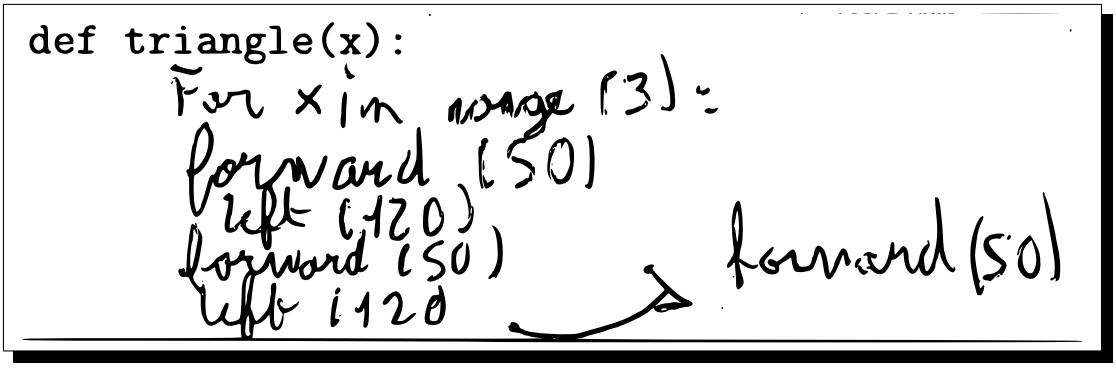

Parfois c'est incomplet (après le carré à cinquante côtés, le triangle à deux côtés !) :




Cependant le pentacontagone a toujours du succès, même lorsqu'on pense bien à tourner de 120° :




5 élèves ont fait des marches arrières avec *home()* pour boucler la boucle ; 10 élèves ont tourné deux fois de 60°, contre 9 qui ont tourné directement de 120° ; il y en a même un qui n'a pas écrit de boucle (ce qui évidemment est parfaitement juste) :




La meilleure solution (on remarquera le soin, poussé jusqu'à la reproduction des *prompts* de Python) :

```
>> def triangle(x):
        for n in range(3):
            forward(x)
            left(60)
            left(60)
>> triangle(60)
```

Mais même avec un algorithme correct et rapidement trouvé, on retrouve le mélange des variables (ici le côté mesure lui aussi 120) :

```
def triangle(x):
for n in range(3):
forward(x)
left(120)
triangle(120)
```

2. **Triangle de mesures données**

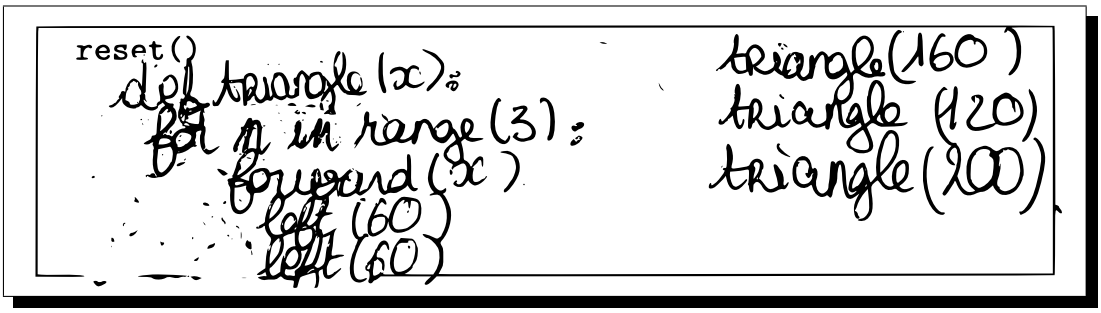
L'exercice était censé montrer que même pour écrire un programme en Python, la réciproque de Pythagore pouvait être utile, ainsi que la trigonométrie, qui permet de calculer de combien on doit tourner après avoir tracé l'hypoténuse. Mais comme là encore, il ne fallait pas tourner de 37° mais de son supplémentaire 143°, la recherche de cet angle par tâtonnements a eu plus de succès que la trigonométrie ; encore que le succès n'était pas toujours au rendez-vous :

```
reset()
forward(200)
left(150)
forward(160)
left(60)
forward(120)
```

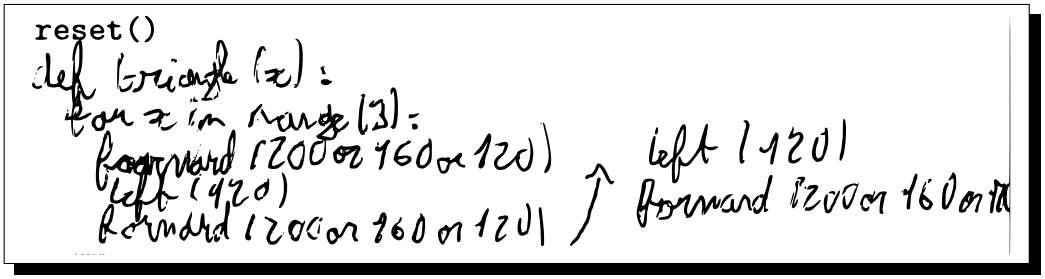
Le spécialiste du nombre 50 a encore essayé de mettre des 50 partout :

```
reset()
for n in range(50):
    forward(200)
    left(50)
    backward(160)
    left(70)
    home()
```

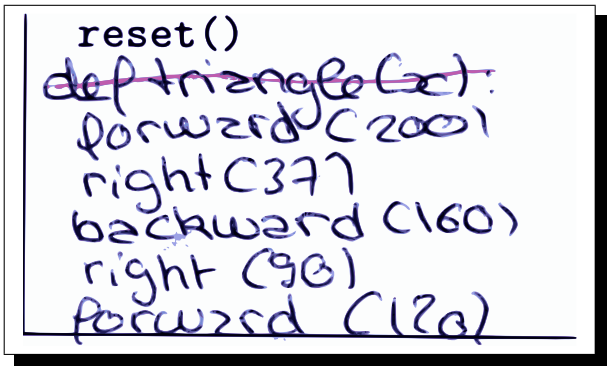
Comme signalé plus haut, lorsqu'on prend l'initiative de dessiner 50 pentacontagones, on a $50 \times 50 = 2500$ côtés à tracer, ce qui prend suffisamment de temps pour permettre à l'élève de calculer mentalement le nombre de côtés (avec une erreur fréquente : $50 \times 50 = 250$...). Ce qui oblige concrètement à quitter puis redémarrer Python et tout recommencer...
Une élève a essayé de réinvestir le triangle équilatéral :




Un autre a essayé de réinvestir la logique vue précédemment [2] :




Là encore, la méthode à reculons a bien fonctionné (13 élèves) :




Au passage, on remarque que beaucoup d'élèves ont confondu un script (suite d'instructions en Python) avec une fonction (qui a une entrée et une sortie, et peut être réutilisée à volonté). Ce qui montre quand même un certain malaise face à une méthode purement expérimentale, et un réel besoin de faire de la programmation structurée. Ainsi, une élève a essayé de définir une fonction `triangle` sans variable avec

```
def triangle:
```

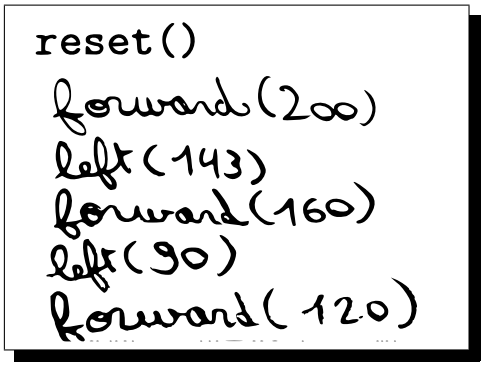
ce qui produit un message d'erreur, la bonne syntaxe étant

```
def triangle():
```

2. L'idée est très fine : Le "ou" logique correspond à un choix, et Python est effectivement capable de faire des choix, par exemple au hasard avec *choice*.

Cette erreur montre que, l'air de rien, le module tortue permet de manipuler des fonctions non numériques, l'antécédent de la fonction `square` étant un nombre, mais l'« image » étant une figure géométrique (carré ou petacontagone selon l'élève).

Finalement, un seul élève a trouvé comment tourner :

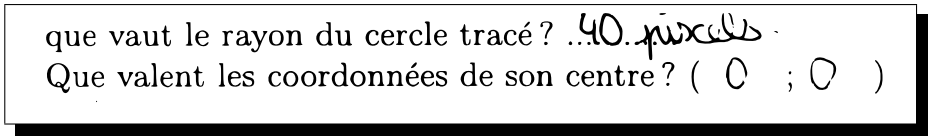



## IV/ Cercles

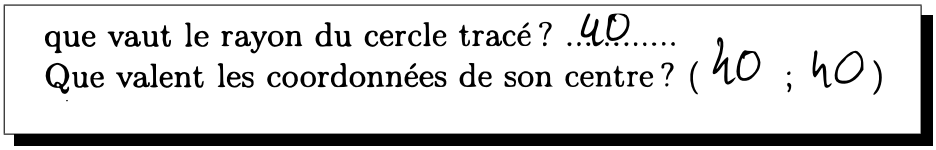
La manipulation attendue était (pour une fois, expérimentalement) de bouger la tortue jusqu'au centre du cercle puis de lire ses coordonnées, le rayon étant alors le déplacement nécessaire. Par exemple :

```
reset()
circle(80)
left(90)
forward(80)
position()
```

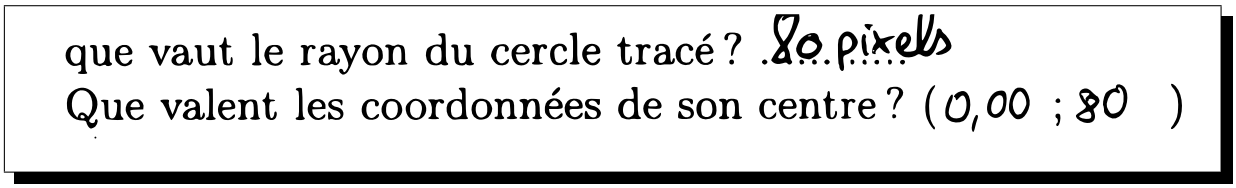
La confusion entre rayon et diamètre a amené la réponse 40 plusieurs fois, soit sans les coordonnées, soit avec le centre du cercle à l'origine :




Ou encore avec un centre fantaisiste :




La bonne réponse a visiblement été obtenue par lecture :




Ceci se confirme d'ailleurs avec une abscisse "négative" chez 6 élèves :

que vaut le rayon du cercle tracé ? ..80 pixels
Que valent les coordonnées de son centre ? (-0.00 ; 80.00)

Un élève a eu un polygone étoilé en guise de cercle, mais n'a pas réussi à se rappeler comment il a obtenu ce comportement étrange de Python...