|  |  |  |
| --- | --- | --- |
| Numérique et Sciences Informatiques | | |
| 30min | **Programmer sous forme de fonctions** |  |
| **Objectif** : savoir découper un programme en fonctions | | |
| **Matériel**: pythontutor puis Python | | |

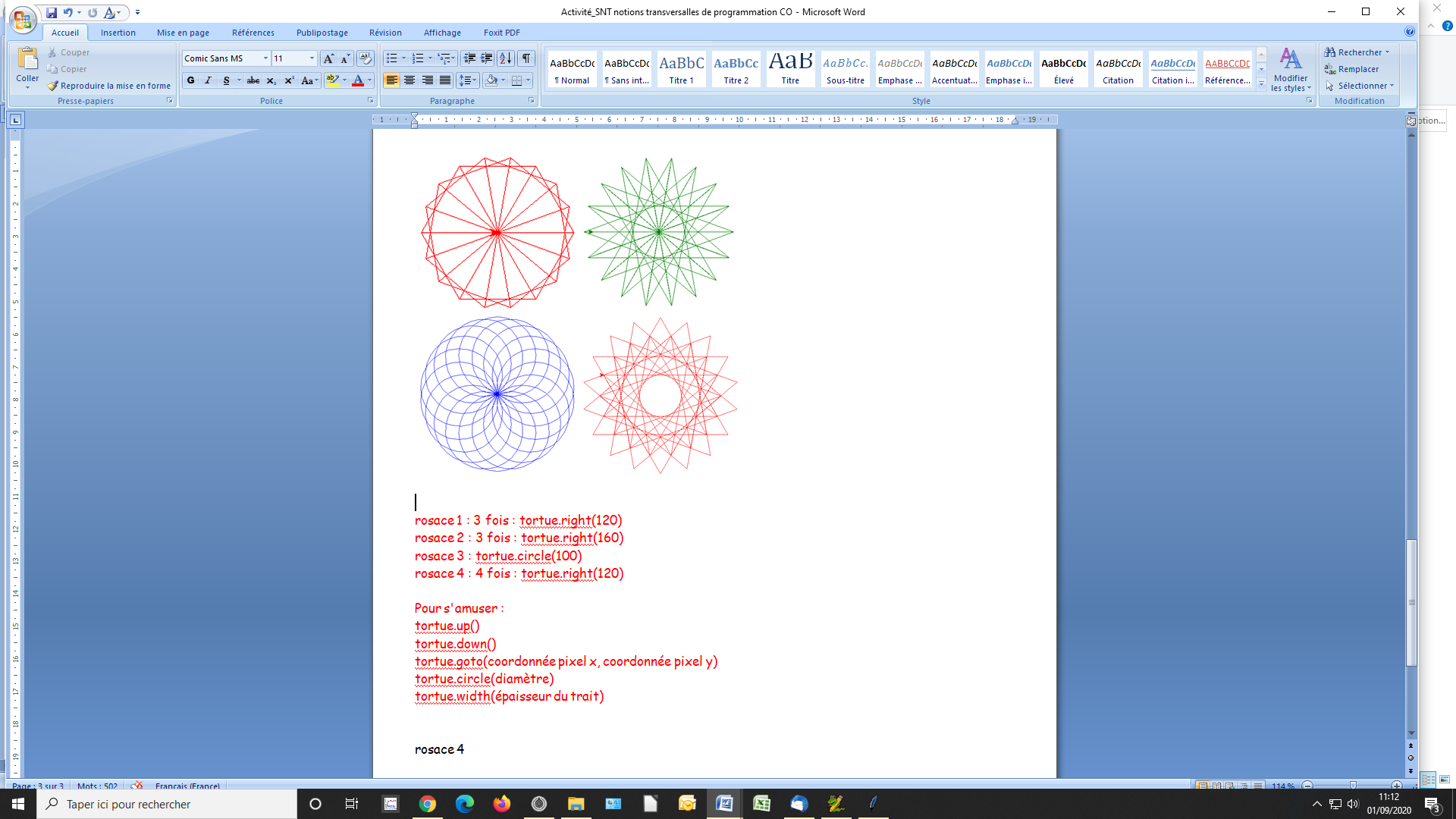
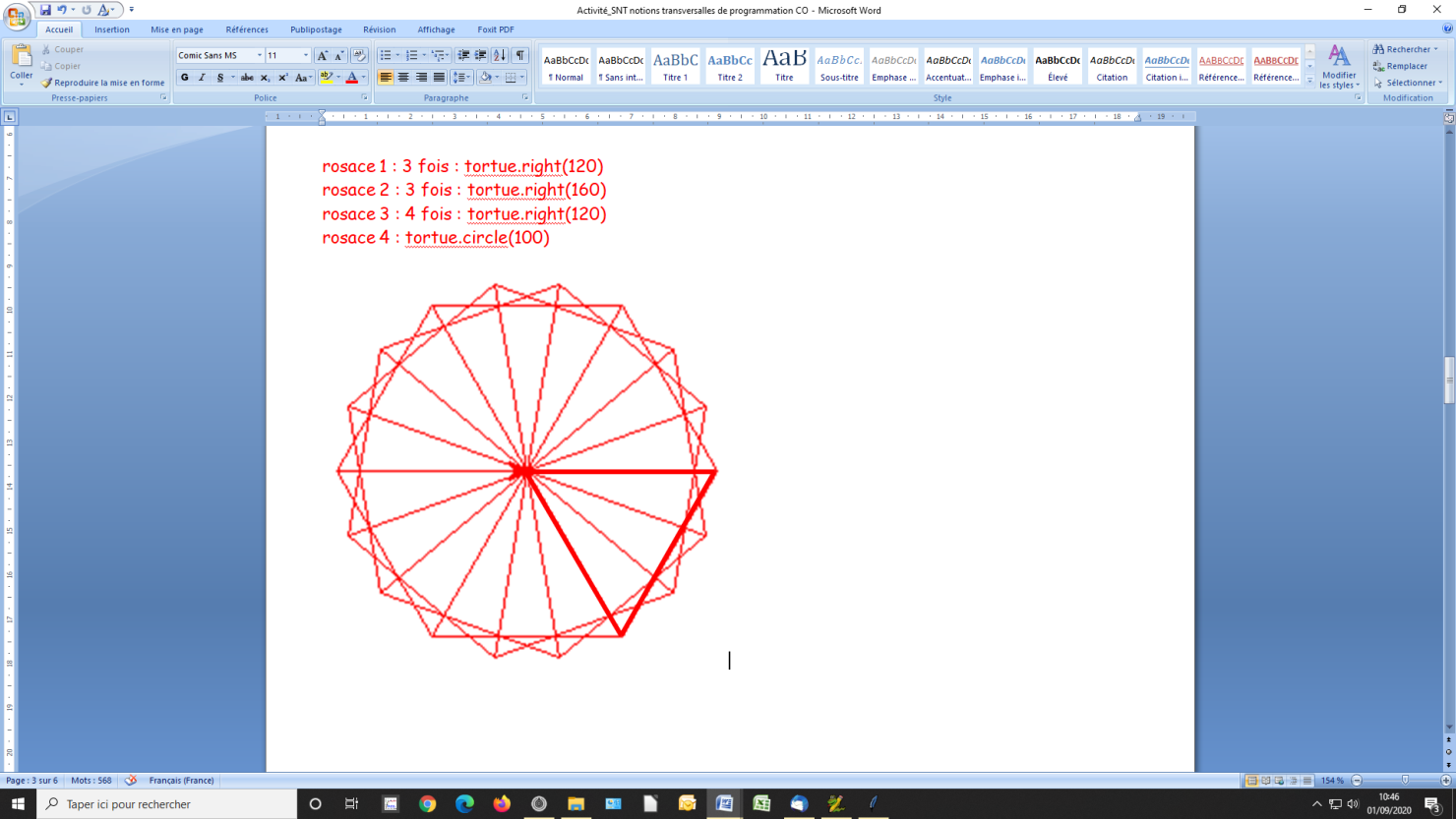
L'intérêt de programmer sous forme de fonctions est divers :

* Si on appelle un code identique plusieurs fois autant mettre ce code dans une fonction.
* Si un programme est long, il est plus facile de le comprendre si le programme principal appelle des fonctions ayant un nom explicite.

Vous avez déjà utilisé des fonctions sans le savoir, exemples de fonctions intégrée dans Python : **len("bonjour")**, **range(4)**, **input() ...**

**Exemple de fonction**

Dessin de la rosace suivante avec le module **Turtle** de Python :

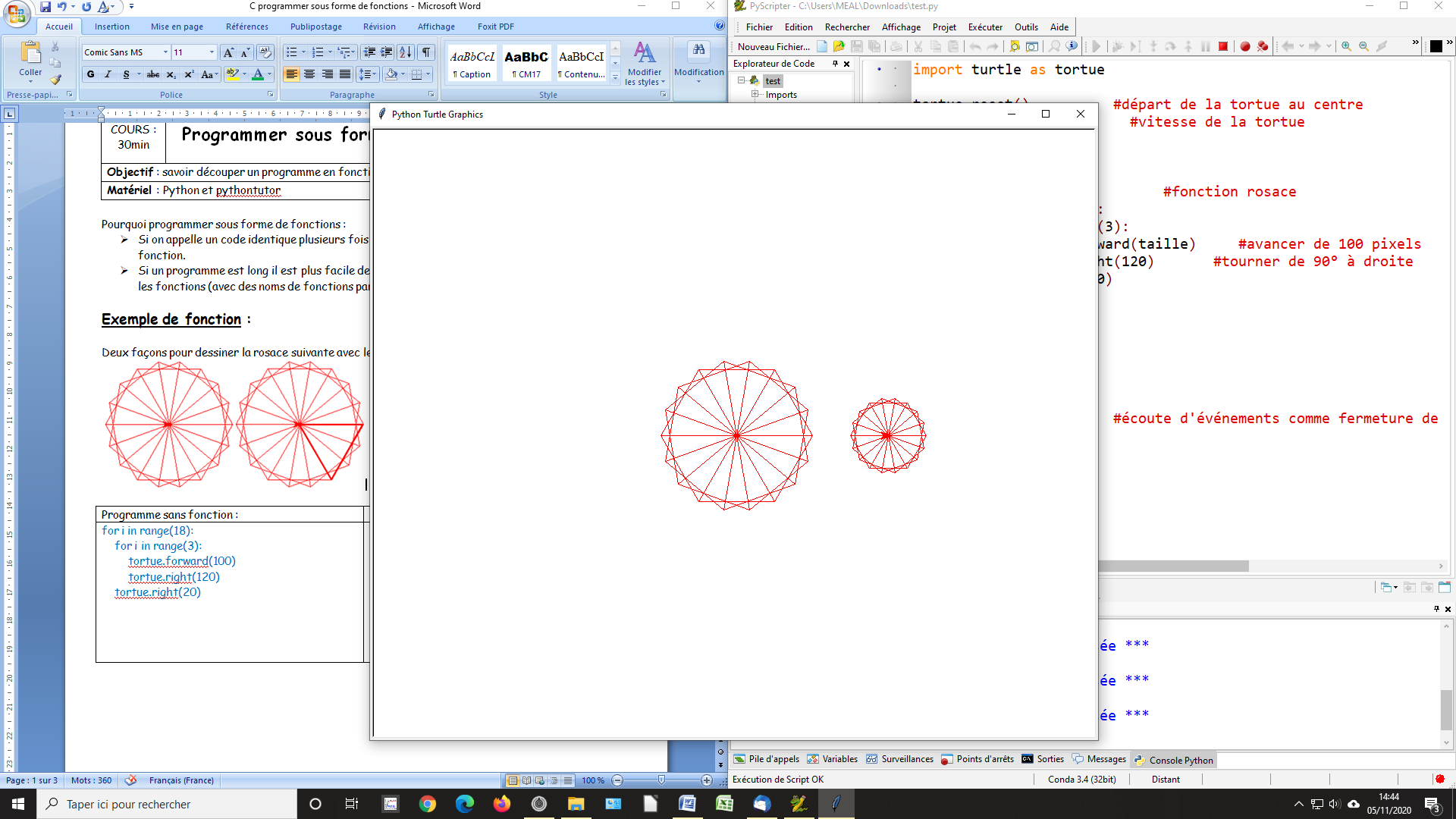


|  |  |
| --- | --- |
| Programme sans fonction : | Programme avec fonction : |
| for i in range(18):  for i in range(3):  turtle.forward(100)  turtle.right(120)  turtle.right(20) | # fonction  def triangle():  for i in range(3):  turtle.forward(100)  turtle.right(120)  # programme principal  for i in range(18):  triangle()  turtle.right(20) |

Remarque : **def** permet la définition d'une fonction en Python.

**Code 1\*\*** :Transformez le programme suivant pour dessiner deux rosaces de tailles différentes l'une à côté de l'autre :

|  |
| --- |
| import turtle #import du module turtle  turtle.reset() #départ de la tortue au centre  turtle.speed(100) #vitesse de la tortue  turtle.color("red") #couleur de la tortue en rouge  def rosace(taille): #fonction rosace  for i in range(18):  for j in range(3):  turtle.forward(taille) #avancer de 100 pixels  turtle.right(120) #tourner de 90° à droite  turtle.right(20)  rosace(100) #appel de la fonction rosace()  turtle.mainloop() #écoute d'événements comme fermeture de la fenêtre de dessin par exemple |



|  |  |
| --- | --- |
| **Aide :** | |
| turtle.up() | Arrêt du dessin du tracé de la tortue |
| turtle.goto(200,0) | Déplacement de la tortue selon l'axe x de 200 pixels |
| turtle.down() | Démarrage du dessin du tracé de la tortue |

**Variables locales et variables globales + passage des variables**

* A tester sous pythontutor : <http://pythontutor.com/visualize.html#mode=edit>

|  |  |
| --- | --- |
| La variable **nb3** n'est pas reconnue par le programme principal : | La variable **nb3** est une **variable locale** à la fonction : |
| def addition():  nb3=nb1+nb2  #programme principal  nb1=3  nb2=5  addition()  print(nb3) | def addition():  nb3=nb1+nb2  print(nb3)  #programme principal  nb1=3  nb2=5  addition() |

|  |  |
| --- | --- |
| Avec des variables **locales** et **globales** qui ont le même nom : | Avec des variables l**ocales** et **globales** qui ont des noms différents : |
| def addition(nb1,nb2):  nb3=nb1+nb2  return nb3  #programme principal  nb1=3  nb2=5  nb3=addition(nb1,nb2)  print(nb3)  Attention **nb1, nb2, nb3** dans **def** ne sont pas les mêmes variables que **nb1, nb2, nb3** dans le **programme principal**. | def addition(nb1\_loc,nb2\_loc):  nb3\_loc=nb1\_loc+nb2\_loc  return nb3\_loc  #programme principal  nb1=3  nb2=5  nb3=addition(nb1,nb2)  print(nb3) |

**On appelle le passage des variables entre un programme principal et une fonction ou entre fonctions, le passage de paramètres ou le passage d'arguments.**

**Quelques règles dans les fonctions**

On passe en argument la ou les variable(s) qu'on va modifier dans la fonction.

On retourne la ou les variable(s) créée(s) dans la fonction ou la ou les variable(s) modifiée(s).

Pour les tuples, listes et dictionnaires (on verra plus tard), c'est n'est pas la peine de les retourner car leur passage en argument se fait obligatoirement par référence (pointeur) et non copie en Python.