

# Cours n°4

mercredi 14 octobre

Python :

→ une autre boucle: while

→ le module **turtle**

# la boucle while

while permet de répéter des tâches “tant que”

- **toutes** les boucles écrites jusqu'à présent aurait pu être écrites avec while (***mais pas forcément plus simplement***)
- quelles boucles ne peuvent pas être écrites avec for de façon “propre”? Celles qui ne correspondent pas au parcours d'une séquence comme
  - jouer jusqu'à la fin de la partie
  - saisir jusqu'à un mot clé
  - calculer jusqu'à dépasser une valeur ou atteindre une limite

# syntaxe du while

```
while <test>:  
    <instructions1>  
else:  
    <instructions2>
```

le else et le deuxième bloc étant facultatifs.  
Ceci revient à:

```
while <test>:  
    <instructions1>  
<instructions2>
```

# while

premier exemple:

on veut afficher les nombres entiers dans l'ordre décroissant de 15 à 1

```
x= 15 # initialisation
```

```
while x>0:      # attention ne pas oublier le caractère :  
    print (x)    # attention à bien mettre ces deux  
    x=x-1        # instructions avec la même indentation  
                # dans le while
```

# exemple

Autre exemple:

on veut récupérer les indices des espaces dans une chaîne de caractères sous forme d'un tuple:

```
res= ()  
for i in range(len(ch)):  
    if ch[i]==' ':  
        res+=(i,)  
print (res)
```

# exemple

Ce qui s'écrit avec while:

```
res= ()  
i=0  
while i<len(ch):  
    if ch[i]==' ':  
        res+=(i,)  
    i+=1 # dans tous les cas il faut passer au suivant  
print (res)
```

## Quelles différences for/while ?

```
res= ()  
i=0      # il faut initialiser i  
while i<len(ch):  
    if ch[i]==' ':  
        res+=(i,)  
    i+=1  # il faut incrémenter i  
print (res))
```

Ces deux étapes sont faites dans le  
`for i in range(len(ch))`



## Autre comparaison de l'écriture d'une boucle

écrire une chaîne entrecoupée d'étoiles

```
def etoile1(ch):  
    i,res=0,"" # il faut initialiser i  
    while i<len(ch):  
        res+=ch[i]+ "*"   
        i=i+1 # et incrémenter i  
    return res
```

for et range permettent d'écrire la même boucle :

```
def etoile2(ch):  
    res=""  
    for i in range(len(ch)):  
        res+=ch[i]+ "*"   
    return res
```



## remarque

Mais avec for on peut aussi écrire dans ce cas un parcours **par caractères**  
*ce que l'on ne peut pas faire avec while:*

```
def etoile2(ch):  
    res=""  
    for c in ch:  
        res+=c+"*"  
    return res
```

## Autre exemple sur des chaînes de caractères

On veut par exemple compter les 'e' dans une chaîne:

```
def compte(x,ch):  
    cp,i,n= 0,0,len(ch) #initialisation compteur  
                        # et de l'index  
    while i<n: # les caractères sont numérotés  
                #de 0 à n-1  
        if ch[i]==x:  
            cp=cp+1  
            i=i+1  
    return cp
```

**!!! attention à la place (indentation) du  $i=i+1$ :**

Que penser du programme suivant?

```
def compte(x,ch):  
    cp,i,n= 0,0,len(ch)  
    while i<n:  
        if ch[i]==x:  
            cp=cp+1  
            i=i+1  
    return cp
```

On a une boucle infinie: car le compteur ne progresse que quand le caractère est celui qu'on compte

**!!! attention à la place (indentation) du  $i=i+1$ :**

Que penser du programme suivant?

```
def compte(x,ch):  
    cp,i,n= 0,0,len(ch)  
    while i<n:  
        if ch[i]==x:  
            cp=cp+1  
        i=i+1  
    return cp
```

On a une boucle infinie:

En effet le compteur ne progresse jamais

# Attention, risque de boucle infinie !!!

Vous allez probablement faire vos premières boucles infinies en TP, soit comme dans les exemples précédents à cause de la place de l'incrémantation soit avec des fautes de frappe par exemple:

```
x=1
while x< 100:
    print (x),
    x = x- 1    # faute de frappe – au lieu de + ...
```

Ce programme ne peut pas s'arrêter tout seul!

Si vous l'avez lancé..... utilisez control C

En principe cela arrête la boucle..... sinon il faudra “tuer” l'application Python (un “kill” dans un terminal....)

(heureusement, votre programme est normalement sauvegardé!)

# Saisie avec tests

On peut utiliser while pour faire un test sur une saisie et recommencer la saisie tant que la réponse n'est pas acceptable:

Si on reprend l'exemple des impôts sur Zorclub on veut vérifier que l'utilisateur tape bien homme ou femme (et sinon on veut le faire recommencer)

```
def zorclub():  
    s= input("homme ou femme")  
    while s!="homme" and s!="femme":  
        print("erreur de saisie")  
        s= input("homme ou femme")  
    age=input(" veuillez saisir votre âge ") # là aussi un test à  
mettre  
    if s=="homme" and age.... # suite du programme identique
```

Le while fait une boucle tant que la saisie n'est pas soit homme soit femme



Un cas ou le for n'est pas indiqué:  
une saisie jusqu'à .....

On veut faire une somme: l'utilisateur tape successivement tous les entiers qu'il veut additionner..... jusqu'à taper F pour finir. A ce moment là, on lui donne son résultat.

>>>

```
entrez un nombre ou F pour finir 5
entrez un nombre ou F pour finir 4
entrez un nombre ou F pour finir 6
entrez un nombre ou F pour finir F
votre total est de 15
```

Le soucis du for est qu'on ne sait pas à l'avance combien il faudra d'étapes



Avec un for, il faudrait lancer une boucle sur un grand nombre (mais combien?) et stopper la boucle (avec return ou break) pas toujours idéal

Avec le while:

```
def saisie():  
    somme=0  
    n=input("entrez un entier ou F pour finir ")  
    while n!="F":  
        somme= somme +int(n)  
        n=input("entrez un nombre ou F pour finir ")# important  
    print ("votre total est de ",somme)
```

**On peut rajouter un test pour vérifier les saisies...**

On peut aussi écrire

```
def saisie():  
    somme=0  
    n="0"  
    while n!="F":  
        somme= somme +int(n)  
        n=input("entrez un nombre ou F pour finir ")  
    print ("votre total est de ",somme)
```

**On peut rajouter un test pour vérifier les saisies...**

# Comment placer le test ?

```
def saisie():  
    somme=0  
    n=input("entrez un nombre ou F pour finir ")  
    while n!="F":  
        if correct(n): # n est une chaine  
            somme= somme +int(n) # c'est là que le programme planterait  
        else:  
            print( " mauvaise saisie")  
            n=input("entrez un nombre ou F pour finir ")  
    print ("votre total est de ", somme)
```

Il faut encore écrire la fonction correcte  
dans un premier temps pour tester le reste on peut toujours  
mettre:

```
def correct(ch):  
    return True
```

puis

```
def correct(ch):  
    return ch.isdigit()
```

## Une façon d'écrire le test:

```
def correcte(ch):  
    for x in ch:  
        if not (chiffre(x)):  
            return False  
    return True  
  
def chiffre(x):  
    return x in "0123456789"
```

Cela ne règle pas toutes les situations:  
par exemple ne convient pas à un nombre négatif ou à une saisie du genre 3+5

## Retour sur l'exo noté hier

```
def saisiePhrase():  
    phrase=""  
    for x in range(100):  
        a=input("Entrez un mot ou Stopfin ")  
        if a=="Stopfin":  
            return phrase  
        else:  
            phrase+= " " + a
```

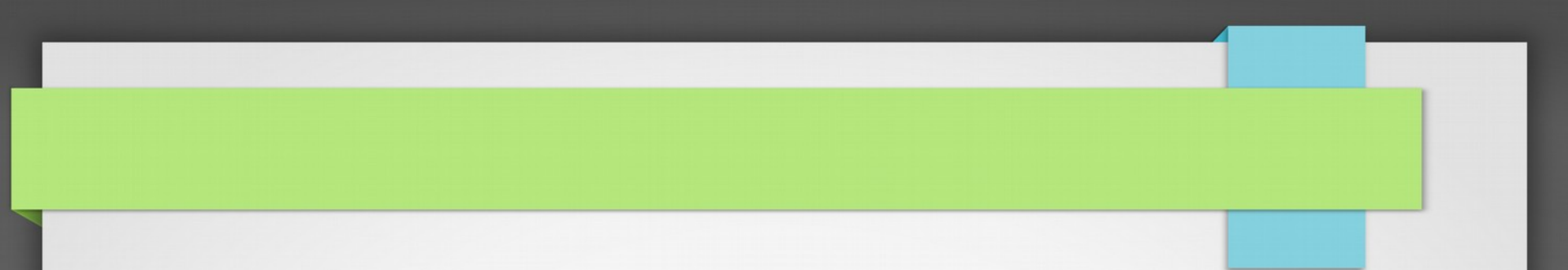
```
def saisiePhraseW():  
    phrase=""  
    a=input("Entrez un mot ou Stopfin ")  
    while a!="Stopfin":  
        phrase+= " " + a  
        a=input("Entrez un mot ou Stopfin ")  
    return phrase
```

## Avec des tests

```
def test(ch,carac):  
    return carac in ch
```

```
def saisieMot(): # saisie d'un seul mot  
    mot =input("Entrez un mot ou Stopfin pour finir")  
    if test(mot, " "):  
        print("un seul mot, recommencez")  
        return saisieMot()  
    else:  
        return mot
```

```
def saisieMotW():# saisie d'un seul mot  
    mot =input("Entrez un mot ou Stopfin pour finir")  
    while test(mot, " "):  
        print("un seul mot, recommencez")  
        mot =input("Entrez un mot ou Stopfin pour finir")  
    return mot
```



```
def saisiePhraseWW():  
    phrase=" "  
    a=saisieMotW()  
    while a!="Stopfin":  
        phrase+= " " + a  
        a=saisieMotW()  
  
    return phrase
```



```
def SaisiePhraseSansE():  
    phrase=" "  
    cp=0  
    a=saisieMotW()  
    while a!="Stopfin":  
        if test(a, "e"):  
            print( "perdu au bout de ", cp, " mots")  
            return cp  
        else:  
            phrase+= " " + a  
            cp+=1  
            a=saisieMotW()  
    return phrase
```

```
def SaisiePhraseSansE():  
    phrase=" "  
    cp=0  
    a=saisieMotW()  
    while a!="Stopfin":  
        if test(a, "e"):  
            print( "perdu au bout de ", cp, " mots")  
            a="Stopfin"  
        else:  
            phrase+= " " + a  
            cp+=1  
            a=saisieMotW()  
  
    return phrase
```

## Autre exemple: un calcul "jusqu'à dépasser..."

On s'intéresse à la somme des  $1/p$ . On sait que cette somme tend vers l'infini. On veut déterminer pour un nombre réel  $A$  quelconque l'entier  $n$  tel que la somme des  $1/p$  pour  $p$  variant entre 1 et  $n$  dépasse  $A$ .

```
>>> somme(5)
```

```
84
```

```
>>> somme(10)
```

```
12368
```

## Pour faire le calcul de la somme :

```
def som(n):  
    res=0  
    p=1  
    while p<=n:  
        res+=1/p  
        p+=1  
    return res
```

# attention ici il faut initialiser p et l'incrémenter

```
def som(n):  
    res=0  
    for p in range(1,n+1):  
        res+=1/p  
    return res
```

## Un calcul jusqu'à dépasser..

Pour faire la somme jusqu'à dépasser A : le schéma est le même **mais on n'utilise pas le programme précédent, on le modifie**

```
def somme(A):  
    res=0  
    p=1  
    while res<A:    # la boucle continue tant qu'on n'a pas atteint A  
        res+=1/p  
        p+=1  
    return p
```

# Important

## Si on utilise le programme précédent

```
def somme(A):  
    p=1  
    while A<som(p):  
        p+=1  
    return p
```

C'est **très inefficace** car on recommence le calcul de la somme au début à chaque fois!!!

Exemple: tirage de plusieurs objets jusqu'à en avoir 6 différents (exemple de 6 numéros au loto)

On va prendre un entier au hasard entre 1 et 49 jusqu'à en obtenir 6 différents:

```
def loto1():  
    res,n=(),0  
    while n<6:  
        x=randint(1,49)  
        if not (x in res):  
            res,n=res+(x,),n+1  
        # pas de else si on a déjà x on ne fait rien et  
        # surtout pas d'incrementation  
    return l
```



→ on peut aussi mettre le test d'arrêt sur la taille de la liste que l'on construit

```
def loto2():  
    res=()  
    while len(res)<6:  
        x=randint(1,49)  
        if not (x in res):  
            res+=(x,)   
    return res
```

# comparaison for / while

- while peut faire absolument **toutes les boucles** alors que for est prévu pour faire des **parcours de séquence (ou de dictionnaires, fichiers)**
- for est éventuellement plus vite écrit (gestion automatique de la variable de parcours: initialisation et  $i=i+step$ )
- for n'est pas pratique pour des boucles où l'incrémentatation ne se fait pas à chaque étape ou pas régulièrement
- while ne peut pas parcourir une chaîne caractères par caractères mais uniquement par les indices des caractères (idem pour les tuples, listes, etc.....).

Il ne sera pas très pratique pour parcourir un dictionnaire

# Un jeu: devinez un nombre

L'ordinateur a choisit un nombre, l'utilisateur doit le trouver.....

```
>>> mystere()  
entrez votre choix 50  
trop petit  
entrez votre choix 85  
trop petit  
entrez votre choix 100  
trop grand  
entrez votre choix 90  
trop petit  
entrez votre choix 95  
trop grand  
entrez votre choix 93  
trop grand  
entrez votre choix 92  
bravo
```

## Comment faire?

On doit avoir le schéma suivant

- 1. l'ordinateur choisit un entier qu'on appellera  $x$
- 2. on demande un entier à l'utilisateur qu'on appelle  $n$
- 3. si  $n=x$  l'utilisateur a gagné  
Sinon on doit afficher trop grand ou trop petit  
et recommencer à l'étape 2

Remarque: on fera une fonction pour lancer le jeu

→ 1. l'ordinateur choisit un entier qu'on appellera  $x$

On va utiliser une boucle "tant que l'on a pas gagné"  
donc avoir une variable *gagne* qui est à 0 tant que  
l'utilisateur n'a pas trouvé

→ 2. on demande un entier à l'utilisateur qu'on appelle  $n$   
C'est juste un input à utiliser (avec tests ensuite)

→ 3. si  $n=x$  l'utilisateur a gagné

Alors *gagne*=1

Sinon on doit afficher trop grand ou trop petit  
et recommencer à l'étape 2

Donc la boucle est sur les étapes 2 et 3

D'où le schéma

$x$  = entier choisi par l'ordinateur

gagne = 0

while gagne == 0

$n$  = entier proposé par l'utilisateur

    Si  $n == x$  alors gagne = 1

        Afficher que l'utilisateur a gagné

    Si  $n < x$  afficher trop petit

    Sinon afficher trop grand



# D'où le programme

```
from random import randint

def mystere():
    nmyst=randint(0,100)
    gagne=0
    while gagne==0:
        n=int(input ("entrez votre choix "))
        if n==nmyst:
            print("bravo")
            gagne=1
        elif n<nmyst:
            print("trop petit")
        else:
            print("trop grand")
```



## Améliorations:

### 1. compter les coups

```
>>> mystere1()  
entrez votre choix 50  
trop petit  
entrez votre choix 76  
trop grand  
entrez votre choix 65  
trop grand  
entrez votre choix 61  
trop grand  
entrez votre choix 59  
bravo gagné en 5 coups
```

## Compter les coups: il suffit de rajouter un compteur

```
def mystere1():  
    nmyst=randint(0,100)  
    gagne=0  
    cp=0  
    while gagne==0:  
        n=int(input ("entrez votre choix "))  
        cp+=1  
        if n==nmyst:  
            print("bravo gagné en ", cp, "coups")  
            gagne=1  
        elif n<nmyst:  
            print("trop petit")  
        else:  
            print("trop grand")
```

>>> mystere2()

entrez votre proposition ou help pour arrêter 45  
trop petit

entrez votre proposition ou help pour arrêter 87  
trop petit

entrez votre proposition ou help pour arrêter 98  
trop grand

entrez votre proposition ou help pour arrêter help  
paresseux il fallait trouver 94

```
def mystere2():
    nmyst=randint(0,100)
    gagne=0
    cp=0
    while gagne==0:
        n=input("entrez votre proposition ou help pour arrêter ")
        cp+=1
        if n=="help":
            print("paresseux il fallait trouver ",nmyst)
            gagne=1
        else:
            n=int(n)
            if n==nmyst:
                print("bravo gagné en ", cp, "coups")
                ..... fin inchangée
```

## Améliorations: 3. empêcher certains plantages

Pour l'instant le programme se plante si on saisie autre chose qu'un entier car c'est le int qui plante

D'où une saisie avec tests: on va déjà tester si une string peut correspondre à un entier;

On peut utiliser la fonction correcte vue tout à l'heure mais attention la saisie peut aussi être un mot clé help.

Puis on écrit une fonction de saisie:

Soit comme la semaine dernière

```
def saisie():  
    rep=input("entrez votre proposition ou help pour arrêter ")  
    if rep=="help":  
        return rep  
    elif corecte(rep):  
        return rep  
    else:  
        print ("erreur de saisie , recommencez ")  
        return saisie()
```

Cette fonction de saisie renvoie soit "help" soit une chaîne qui peut être transformée en entier: la fonction est ré-appelée si nécessaire

## On peut aussi utiliser while

```
def saisie():  
    ok=0  
    while not ok:  
        rep=input("entrez votre proposition ou help pour  
arrêter ")  
        if rep=="help" or correcte(rep):  
            ok=1  
        else:  
            print ("erreur de saisie , recommencez ")  
    return rep
```

Cette fonction de saisie renvoie soit "help" soit une chaîne qui peut être transformée en entier: on aura demandé tant que la réponse n'est pas acceptable.



```
def mystere3():
    nmyst=randint(0,100)
    gagne=0
    cp=0
    while gagne==0:
        n=saisie()
        cp+=1
        if n=="help":
            print("paresseux il fallait trouver ",nmyst)
            gagne=1
        else:
            n=int(n) # là cela ne peut pas planter
            if n==nmyst:
                print("bravo gagné en ", cp, "coups")
                gagne=1
            elif n<nmyst:
                print("trop petit")
            else:
                print("trop grand")
```

## Améliorations

## indiquer des coups déjà joués

entrez votre proposition ou help pour arrêter 56  
trop petit

entrez votre proposition ou help pour arrêter 58  
trop petit

entrez votre proposition ou help pour arrêter 76  
trop grand

entrez votre proposition ou help pour arrêter 58  
déjà proposé, tant pis

entrez votre proposition ou help pour arrêter 67  
trop grand

entrez votre proposition ou help pour arrêter help  
paresseux il fallait trouver 64

## On utilise une “mémoire” : un tuple

```
def mystere4():
    nmyst=randint(0,100)
    gagne,cp=0,0
    vu=() # pour garder en mémoire les coups déjà joués
    while gagne==0:
        n=saisie()
        cp+=1
        if n=="help":
            print("paresseux il fallait trouver ",nmyst)
            gagne=1
        else:
            n=int(n) # là cela ne peut pas planter
            if n in vu:
                print ("déjà proposé, tant pis ")
            elif n==nmyst:
                print("bravo gagné en ", cp, "coups")
                gagne=1
            elif n<nmyst:
                print("trop petit")
            else:
                print("trop grand")
            vu+=(n,)
```

## Améliorations

## indiquer des coups stupides

>>> mystere5()

entrez votre proposition ou help pour arrêter 39  
trop petit

entrez votre proposition ou help pour arrêter 78  
trop petit

entrez votre proposition ou help pour arrêter 99  
trop grand

entrez votre proposition ou help pour arrêter 75  
stupide

trop petit # plus petit qu'un truc trop petit

entrez votre proposition ou help pour arrêter 100  
stupide

trop grand # plus grand qu'un truc trop grand

```
def mystere5():
    nmyst, gagne, cp, petits, grands = randint(0,100),0,0,(),()
    while gagne==0:
        n=saisie()
        cp+=1
        if n=="help":
            # pas de changement
        else:
            n=int(n)
            if n in grands+petits:
                print ("déjà proposé, tant pis ")
            elif n==nmyst:
                print("bravo gagné en ", cp, "coups")
                gagne=1
            elif n<nmyst:
                if petits and n< max(petits):
                    print ("stupide ")
                    petits+=(n,)
                print("trop petit")
            else:
                if grands and n> min(grands):
                    print ("stupide ")
                    grands+=(n,)
                print("trop grand")
```

# module de dessin

Nous allons dessiner avec “une tortue logo”: on donne des instructions du style “avance de 10”, “tourne à gauche de 50°..... la tortue porte un petit crayon, s'il est baissé, un trait est tracé pendant le déplacement.....

Il faudra importer le module turtle et surtout ne pas appeler son programme turtle



## Les fonctions de base

up () : lève le crayon

down (): baisse le crayon

forward (d): fait avancer la tortue de d, le trait est dessiné si le crayon est baissé

backward (d): fait reculer la tortue de d, le trait est dessiné si le crayon est baissé

left(a): fait pivoter la tortue d'un angle de a degrés vers la gauche

right (a): fait pivoter la tortue d'un angle de a degrés vers la droite

reset (): nettoie la fenêtre de dessin, réinitialise la tortue: elle est au centre de l'écran de dessin tournée vers la droite.

home(): remet la tortue au centre avec son cap (attention si le crayon est baissé il y a un tracé), n'efface rien



color(c): la couleur par défaut est le noir. On peut changer en mettant une couleur: c doit être une chaîne prédéfinie "red" ,"green" ,"blue" ,"yellow" (ou système rgb)...

bgcolor(c): modifie la couleur de fond (blanc par défaut).

dot(d,c): dessine un disque de diamètre d et de couleur c là où est la tortue.

circle(r): trace un cercle de rayon r, le point de départ de la tortue appartient au cercle (attention il n'est pas centré sur la position de la tortue)

circle(r,s): trace une portion du cercle correspondant à s degrés.

shape(f): où f est une forme permet de changer la forme du curseur ;

f peut prendre les valeurs "turtle", "classic" (valeur par défaut), "circle", "square", "triangle", "arrow".

width(e): fixe la largeur du trait (en pixel)

write (ch): permet d'écrire la chaîne ch là où est la tortue

goto(x,y): la tortue va se positionner au point au point de coordonnées (x,y) (le dessin est tracé si le crayon est baissé).

setheading(a): où a est en degré permet de fixer un cap absolu à la tortue

speed (vitesse): permet de fixer une vitesse à la souris. L'argument peut être 'fastest' 'fast' 'normal' 'slow' ou 'slowest'.

hideturtle() et showturtle(): permettent de supprimer le curseur ou de le remettre

begin\_fill() et end\_fill(): permettent de "remplir" une figure géométrique: on écrira par exemple begin\_fill() avant le dessin que l'on veut "remplir" et end\_fill() à la fin

## Exemple d'un dessin

```
def triangle(n,c): # dessin d'un triangle équilatéral de couleur c
```

```
    color(c)
```

```
    for i in range(3):
```

```
        forward(n)
```

```
        left(120)
```

```
couleur=("red","blue","green","black","yellow","purple",  
"orange","pink", "brown","grey")
```

```
def choisir(t):
```

```
    return choice(t)
```

```
def dessin (n):  
    for i in range(n):  
        c= choice(couleur)  
        t=randint (10,55)  
        bool=randint(0,1) # ce sera plein ou non aléatoirement  
        if bool:  
            begin_fill()  
            triangle(t,c)  
            end_fill()  
        up()  
        left (randint (0,360))  
        forward (randint (0,50))  
        down ()
```



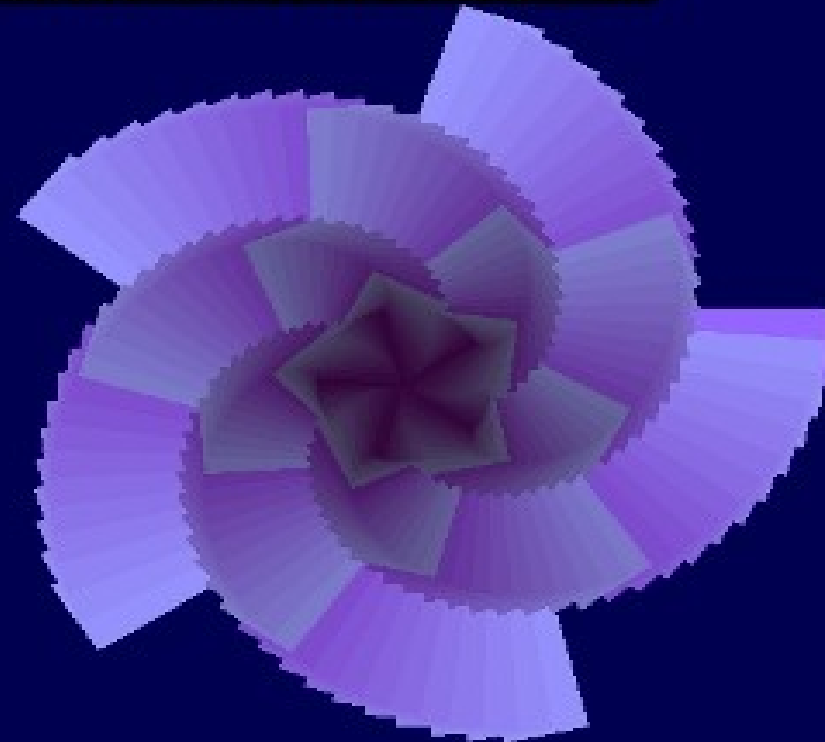
Nous dessinerons pendant un TP..... essentiellement  
des dessins assez simples pour vérifier nos boucles....

Mais on peut faire des choses sophistiquées comme ces  
travaux de lycéens de seconde  
(tous avec turtle et à base de figures géométriques de  
base)

Par Honorine et Mathilde



ssin réalisé avec le module turtle de Python



Dessin réalisé avec le module

