- Mes cours se trouvent à l'adresse https://mybinder.org/v2/gh/debimax/cours-debimax/master
- Éditeur python en ligne :https://repl.it/site/languages/python_turtle

1 Les bases de python3

Les types

Il y a différents types : entiers, flottants, complexes, chaînes de caractères, boléens, listes, tuples et dictionnaires.

- ***entier (integer)*** : 3
- ***flotant (float)*** : 2.3
- ***chaînes de caractères (string)*** : 'ISN'
- ***Boléen*** : True
- On aura toujours à l'esprit que $0.2 + 0.7 = 0.8999999999999999 \neq 0.9$
- ***int()*** permet de convertir, un nombre ou une chaîne de caractère en un entier.
- ***float()*** permet de convertir, un nombre ou une chaîne de caractère en un flottant.
- On utilise le caractère # pour écrire un commentaire

Les opérateurs

Opération	algorithme	python
Addition	2+3	2+3
Soustraction	12-5	12-5
Multiplication	3*6	3*6
Division	7/2	7/2
Quotient de la division euclidienne	7 div 2 ou div(7,2)	7//2
Reste de la division euclidienne	7 mod 2 ou mod(7,2)	7%2
puissance	7^2	7**2
racine carrée	$\sqrt{2}$ ou sqrt(2)	sqrt(2)

Variables

- Python utilise le symbole = pour affecter une valeur à une variable.
- Attention à ne pas confondre $a = 12$ et $a == 12$.

```
A=3; B=5
C='toto'
```

Entrées

- ***input("texte")*** permet de saisir du texte pour un programme.
- Il faudra éventuellement convertir ce texte dans le type voulu avec ***int()*** ou ***float()***

```
a=input("Saisir un texte a:  ")
b=int(input("Saisir un entier b:    "))
c=float(input("Saisir un réel c:    "))
```

Affichage

- On utilise de préférence la méthode ***format()*** pour afficher du texte et des variables.

```
print("La valeur de la variable a est", a ,".")
print("La valeur de la variable a est " + str(a) + ".") # autre forme
print("La valeur de la variable a est {}.".format(a)) # je préfére
print("Le produit de {} par {} est {}".format(a,b,a*b))
print("{0}*{1}={2} et {0}/{1}={3}".format(a,b,a*b,a/b))
```

Connecteurs logiques

algorithmique	python
a = b	a==b
a≠b	a!=b
a ⩽ b	a<=b

algorithmique	python
A et B	A and B
A ou B	A or B
non A	not(A)

Fonction

Syntaxe :

```
def nomfonction(parametres):
    instructions
    instructions
    return valeur
```

Créer la fonction affine $f : x \longmapsto 3x + 1$ et afficher les valeurs de $f(-5)$ à $f(5)$.

```
def f(x):
    return 3*x+1
for i in range(-5,6):
    print(f(i))
```

Condition SI

```
if x<0:
    print(']-inf;0[')
```

```
if x<0:
    print(']-inf;0[')
else:
    print('[0;+inf[')
```

```
if x<0:
    print(']-inf;0[')
elif 0<=x<=20:
    print('[0;20]')
else:
    print(']20;+inf[')
```

Exemple :

```
print("Saisissez deux valeurs numériques")
a=float(input("Saisir a: "))
b=float(input("Saisir b: "))
if a==b :
    print("Vous avez saisi deux fois la même valeur, à savoir {}.".format(a))
else :
    print("Vous avez saisi deux valeurs différentes {} et {}.".format(a,b))
```

Boucle pour

```
for i in range(7):      # pour i allant de de 0 à 6
    print(i)
for i in range(1,7):      # pour i allant de de 1 à 6
    print(i)
for i in range(1,6,2):     # pour i allant de de 1 à 6 par pas de 2 donc: 1 3 5
    print(i)
```

La syntaxe générale est ***for i in range(m,n,p)*** :
i prend alors toutes les valeurs de ***m à n-1 par pas de p***

Tant que

```
i=1
while i<=5:
    print(i)
    i=i+1 #où en plus concis i+=1
# Affichage: 1 2 3 4 5
# À la sortie de la boucle i=6
```

2 Les listes

2.1 Le slicing

```
In [1]: l=[1,2,3,4,5]
In [2]: l[0]            # 1° élément de la liste
Out[2]: 1
In [3]: l[1]            # 2° élément de la liste
Out[3]: 2
In [4]: l[-1]           # dernier élément de la liste
Out[4]: 5
In [5]: l[1:4]          # Du 2° élément jusqu'au 4°.
Out[5]: [2, 3, 4]
In [6]: l[1:]          # Du 2° élément jusqu'à la fin
Out[6]: [2, 3, 4, 5]
```

Plus généralement, la syntaxe $S[i:j]$ désigne la séquence, de même nature que S, de $S[i]$ compris à $S[j]$ non compris.

2.2 Les principales méthodes de listes

- ***l.append(x)*** **:** Ajoute x à la fin de la liste.

```
In [1]: l=[1,2,3,4,5]
In [2]: l.append(6)
In [3]: l
Out[3]: [1, 2, 3, 4, 5, 6]
```

- ***l.clear()*** Efface la liste l *(depuis python3.3)*.

```
In [4]: l.clear()
In [5]: l
Out[5]: []
```

- ***l.extend (t)*** **:** concatène la liste l avec le contenu de la liste t sans réaffectation.

```
In [6]: t=(2,3,4,2)
In [7]: l1=[1,2,3]
In [8]: l2=[4,5,6]
In [9]: l1.extend(l2)
```

- ***l.insert(n,x)*** **:** insert x à la liste l à l'indice n.

```
In [10]: l=[1,2,3,4]
In [11]: l.insert(1,8)
In [12]: l
Out[13]: [1, 8, 2, 3, 4]
```

- ***l.pop ([i])*** **:** affiche l'élément d'indice i et aussi supprime cet élément de l.

```
In [14]: l=[1,2,3,1]
In [15]: a=l.pop(2)
In [16]: a
Out[16]: 3
In [17]: l
Out[17]: [1, 2, 1]
```

- ***l.remove (x) :*** supprime le premier élément x de *s* et affiche une *ValueError* s'il ne le trouve pas.

```
In [9]: l=[1,2,3,1]
In [10]: l.remove(1)
In [11]: l
Out[11]: [2, 3, 1]
In [12]: l.remove(1)
In [13]: l
Out[13]: [2, 3]
In [14]: l.remove(1)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-14-ce3fdfde5d67> in <module>()
----> 1 l.remove(1)
ValueError: list.remove(x): x not in list
```

- ***l.reverse()*** inverse les éléments de l.

```
In [16]: l=[1,2,3]
In [17]: l.reverse()
In [18]: l
Out[18]: [3, 2, 1]
```

- ***l.sort()*** Trie les éléments de l.

```
In [19]: l=[4,6,2,1,0,5,3]
In [20]: l.sort()
In [21]: l
Out[21]: [0, 1, 2, 3, 4, 5, 6]
```

Turtle

reset()	On efface tout et on recommence
goto(x,y)	Aller à l'endroit de coordonnées x et y
forward(distance)	Avancer d'une distance donnée
backward(distance)	Reculer
up()	Relever le crayon (pour pouvoir avancer sans dessiner)
down()	Abaisser le crayon (pour pouvoir recommencer à dessiner)
color(couleur)	Couleur peut être une chaîne prédéfinie ('red', 'blue', 'green', etc.)
left(angle)	Tourner à gauche d'un angle donné (exprimé en degré)
right(angle)	Tourner à droite
width(épaisseur)	Choisir l'épaisseur du tracé
fill(1)	Remplir un contour fermé à l'aide de la couleur sélectionnée (on termine la construction par fill(0))
write(texte)	texte doit être une chaîne de caractères délimitée avec des " ou des '

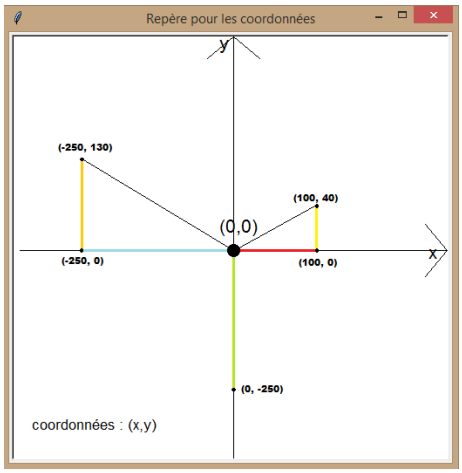
3 Turtle

- Importer le module **turtle**

```
import turtle
```

- Création d'une fenètre de largeur (**width**) 600px, de hauteur (**height**) 400px,

```
turtle.setup(640, 480)
```



3.1 Mouvements de la tortue

- **forward(d)** : fait avancer la tortue de d (en pixel), le trait est dessiné si le crayon est baissé ;
- **backward(d)** : fait reculer la tortue de d (en pixel), 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 ;
- **setheading(a)** : oriente la tortue dans la direction a (toujours mesurée par rapport à Ox ;
- **goto(x,y)** : la tortue va se positionner au point de coordonnées (x ; y) ;
- **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 ;
- **delay(d)** permet de choisir le temps écoulé (en millisecondes) entre deux mises à jour du dessin, et donc indirectement de choisir la vitesse du tracé ;

3.2 Contrôle du stylo

- **up()** : lève le crayon ;
- **down()** : baisse le crayon ;
- **pensize()** ou **width()** : fixe la largeur du trait (en pixel) ;
- **reset()** : nettoie la fenêtre de dessin, réinitialise la tortue ; elle est située alors au centre de l'écran de dessin tournée vers la droite.
- **pencolor(c)** : la couleur par défaut est le noir, on peut la changer en mettant une couleur prédéfinie "red" , "green" , "blue" , "yellow" , . . . ;
- **color(c1,c2)** : modifie la couleur du trait c1 et la couleur du remplissage c2 . On peut aussi les modifier séparement avec pencolor(c) et fillcolor(c) .
- **begin_fill()** et **end_fill()** permettent de commencer et de terminer le remplissage d'une figure géométrique.

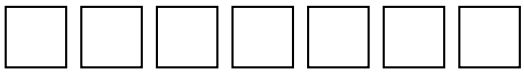
3.3 TP

Exercice 1

1. Dessiner un triangle équilatéral de coté 100 pixel.
2. Écrire la fonction ***triangle1(cote)*** qui dessine un triangle équilatéral dont les côtés sont de longueur *cote* et qui a la pointe vers le haut.
3. Écrire la fonction ***triangle2(cote)*** qui dessine un triangle équilatéral dont les côtés sont de longueur *cote* et qui a la pointe vers le bas.
4. Écrire la fonction ***triangle3(cote,angle)*** qui dessine un triangle équilatéral dont les côtés sont de longueur *cote* et d'une orientation bien déterminée.

Exercice 2

1. Écrire la fonction ***carre(cote)*** qui trace un carre de côté cote . Il est préférable que la tortue termine son dessin là où elle a démarré et avec la même orientation.
2. En déduire la fonction **ligne_de_carres(n,cote)** qui trace n carrés sur une ligne chaque carré étant de coté **cote** (on utilisera la fonction carre).



3. Écrire la fonction ***carres_croissants(n,cote)*** qui trace une ligne de n carrés, le premier carré étant de côté cote , le suivant de côté 1,25 fois la taille du côté du carré qui le précède ; les carrés seront espacés la première fois de cote/4 puis cette distance sera multipliée aussi par 1,25 à chaque fois.
   Vous utiliserez la fonction carre mais pas ***ligne_de_carres*** .