Structure générale d'un script Python

Correspondance python <-> block / scratch

```
import math
import random

if __name__ == '__main__':
    nbTentatives = 10100
    nbReussites = 0
    for i in range(1,nbTentatives+1):
        x = random.uniform(0,2)
        y = random.uniform(0,2)
        distance = math.sqrt((x-1)*(x-1)+(y-1)*(y-1))
        if distance <= 1:
            nbReussites = nbReussites + 1
    pi = 4*nbReussites/nbTentatives

```

Point d'entrée dans le script, parfois masqué par les outils
C'est une bonne pratique

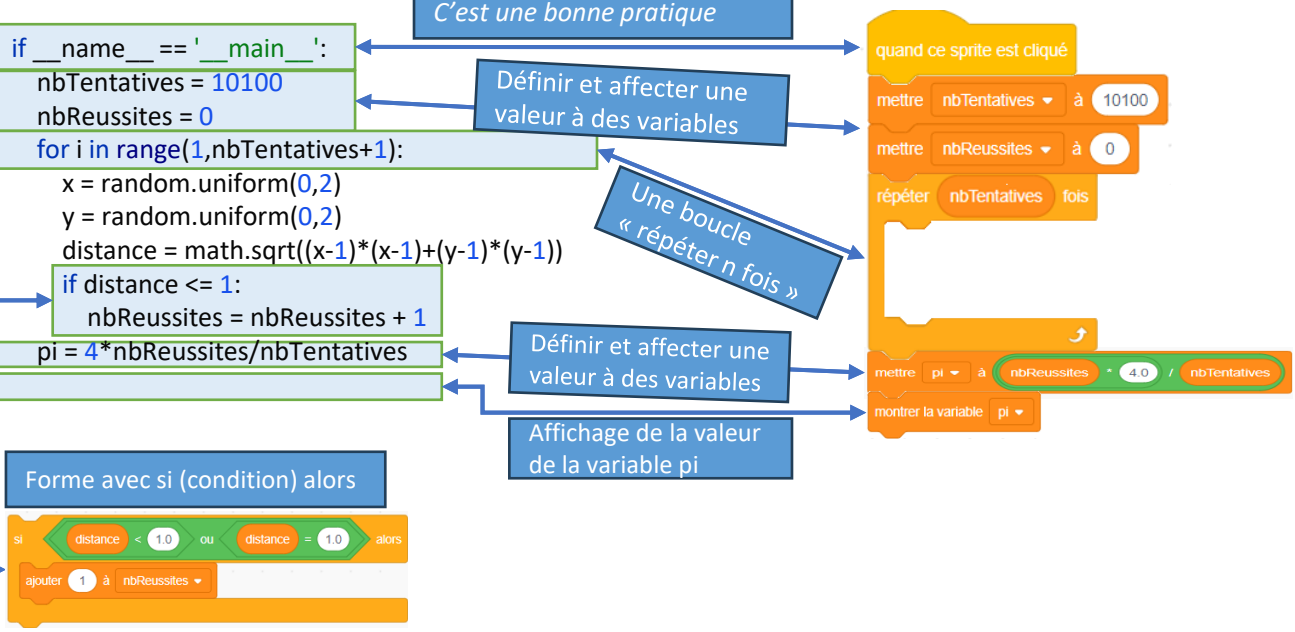
Définir et affecter une valeur à des variables

Une boucle « répéter n fois »

Définir et affecter une valeur à des variables

Affichage de la valeur de la variable pi

Forme avec si (condition) alors



Structuration du code avec les indentations

Niveau 0 : le « bord du fichier »

Niveau 1 : dans le point d'entrée

Niveau 2 : dans la boucle for

Niveau 3 : dans le « if »

```
import math
import random

if __name__ == '__main__':
    nbTentatives = 10100
    nbReussites = 0
    for i in range(1,nbTentatives+1):
        x = random.uniform(0,2)
        y = random.uniform(0,2)
        distance = math.sqrt((x-1)*(x-1)+(y-1)*(y-1))
        if distance <= 1:
            nbReussites = nbReussites + 1
    pi = 4*nbReussites/nbTentatives
    print(f"approximation de pi = {pi}")
```

Le retour à une indentation précédente met fin au « if »

Niveau 1 : dans le point d'entrée

```
nbTentatives = 10100
nbReussites = 0
for i in range(1,nbTentatives+1):
    # [...]
pi = 4*nbReussites/nbTentatives
print(f"approximation de pi = {pi}")
```

Niveau 2 : dans la boucle for

```
x = random.uniform(0,2)
y = random.uniform(0,2)
distance = math.sqrt((x-1)*(x-1)+(y-1)*(y-1))
if distance <= 1:
    # [...]
```

Memo Python Version 1

Instructions et éléments de Python

Affectation : nom_variable = valeur
Variable : « symbole » qui représente une valeur changeante
Nom d'une variable : explicite, pour savoir ce que c'est

Il existe des mots-clés réservés par Python. Exemple : « if », « else », « for », « while », etc.

Type « courant » des variables :

- Nombre entier (int) , Nombre « réel » (float)
- Expression booléenne (boolean, **True** / **False**)
- Chaine de caractères (string / str)

Conversion de type : int(), float(), str()

print : pour afficher du texte
input : pour lire un texte tapé au clavier

Instruction conditionnelle « if » et les conditions

L'instruction conditionnelle commence par « if » suivi d'une expression booléenne et d'un « : »

```
if (note >= 16):
    print("c'est une mention TB")
elif (note >= 14):
    print("c'est une mention B")
elif (note >= 12):
    print("c'est une mention AB")
elif (note >= 10):
    print("c'est une mention passable / sans mention")
else:
    print("malheureusement, le bac n'est pas obtenu")
```

elif permet d'enchainer des cas différents, si le 1er n'est pas vérifié.
ici on regarde si note est entre 14 et 16 (exclu). Ce test n'est fait que si le premier est faux. S'il est vrai, on sera bien entre 14 et 16

Ici on regarde si note est entre 12 et 14 (exclu)
Ce test n'est fait que si les deux premiers sont faux
S'il est vrai, on sera bien entre 12 et 14

Et ainsi de suite : si tous les tests précédents sont faux, ce test-là est effectué…

Finalement, il est possible de terminer le « if » avec un « else » (sinon) pour le cas où toutes les conditions précédentes sont fausses

des booléens : **True** (vrai) et **False** (faux) et d'autres expressions entre booléen :

- a **and** b : (et) expression qui est vrai uniquement si a vaut vrai **et** si b vaut vrai
- a **or** b : (ou) expression qui est vrai si a vaut vrai **ou** si b vaut vrai **ou** les deux valent vrais.
- **not** a : (not = négation, le contraire) expression qui vaut vrai si a est faux

Les égalités / inégalités mathématiques forment des expressions booléennes : (==, !=, <, >, <=, >=)

Boucle while :

Le mot-clef « while » définit une boucle « tant que »

la boucle continue tant que la condition est vraie, cela peut être une expression plus compliquée

```
while nbAllumettes > 0 :
    print(f"il reste {nbAllumettes} allumettes")
    coupJ = int(input("Combien d'allumettes prenez-vous ? 1, 2 ou 3 : "))
    nbAllumettes = nbAllumettes - coupJ
    print(f"Après votre tour, il reste {nbAllumettes} allumettes")
```

Les « : » sont obligatoires

Ce qui est répété est marqué par l'indentation
Le contenu de la boucle peut être n'importe quelles instructions (d'autres boucles, des « ifs », etc.)

La condition doit évoluer dans la boucle, dans le bon sens, sinon c'est une boucle sans fin / une boucle infinie qui bloque le script

fonction:

Mot-clé **def** qui annonce la définition d'une fonction

Le nom de la fonction qui détermine comme elle sera appelée. Le nom doit expliquer ce qu'elle fait

Paramètre de la fonction. Il peut ne pas y en avoir : ( )

Les « : » sont obligatoires

```
def formattage_nb_allumettes(nbAll) :
    formatage = ""
    if nbAll == 0:
        formatage = "zéro allumette"
    elif nbAll == 1:
        formatage = "une seule allumette"
    else:
        formatage = str(nbAll)+" allumettes"
    return formatage
```

« corps » de la fonction : c'est le code qu'elle contient. Ce peut être long, court, avec des boucles, des « if », des appels à d'autres fonctions, etc. C'est l'indentation qui détermine la fin de la fonction

La fonction peut retourner une valeur : print ne retourne rien, input retourne une string. C'est marqué par le mot-clé **return**

SNT