Mémento Python

types de variable

```
a = True # type bool, vaut True ou False
b = 2 # type int (entier)
c = 3.14 # type float (décimal)
d = "Bonjour" # type str (chaine de caractères)
e = [2,3,"quatre"] # type list (liste)
```

opérations sur int et float

`a + b` : addition
`a * b` : multiplication
`a/b` : division (décimale)
`a//b` : quotient de la division euclidienne
`a%b` : reste de la division euclidienne
`a**b` : « a puissance b »
`abs(a)` : valeur absolue
`round(a,n)` : valeur approchée de a à 10^{-n} près.

imports de libraries

Pour la librairie `math` : `from math import *`
Principales fonctions : `sqrt`, `cos`, `pi`...

Pour la librairie `random` : `from random import *`
Principales fonctions : `random`, `randint`, `choice`, `shuffle`...

Pour la librairie `turtle` : `from turtle import *`
Voir le document sur Turtle.

affectations

Affecter, c'est associer une *valeur* à un *nom* :

- évaluation du membre de droite (une ou plusieurs valeurs)
- affectation(s) au(x) nom(s) du membre de gauche.

```
x = 1.2 + 8*sin(y) # Python évalue d'abord y
a, b = 2, 3 # veut dire a = 2 et b = 3
a += 1 # veut dire a = a + 1
b /= 2 # veut dire b = b / 2
```

affichage à l'écran

```
a = 18 # a prend la valeur 18
print("Vous avez ", a, "ans.") # permet d'afficher
    plusieurs objets en une instruction
```

saisie des données

```
a = input("Entrez votre nom :")
# a est par défaut de type chaine de caractères
b = int(input("Entrez votre âge :"))
# pour que b soit un entier
c = float(input("hauteur de l'arbre :"))
# pour que c soit un décimal
```

à propos des str et des list

Soit `a` une variable de type `str` ou `list`.

- `len(a)` donne la longueur de `a`. Notons la `n`.
- On accède aux éléments de `a` par `a[0]`, ..., `a[n-1]`.
- `a[p:q]` retourne la sous-chaine (ou sous-liste) composée de `a[p]`, ..., `a[q-1]`.
- `a[p:]` retourne la sous-chaine (ou sous-liste) composée de `a[p]`, ..., `a[n-1]`.
- `a[:p]` retourne la sous-chaine (ou sous-liste) composée de `a[0]`, ..., `a[p-1]`.

modification des list

```
a = [] # ou a =list() ; crée une liste vide
a = a + [elt] # ou a.append(elt) ; y ajoute elt
del a[k] # supprime l'elt d'indice k de a
a.remove(elt) # supprime la première occurence de
    elt dans a
```

tests conditionnels

Python évalue si une affirmation est vraie ou fausse : il lui affecte la valeur booléenne `True` ou `False`.
On utilise les opérateurs suivants :

- <,>,<= et >= pour comparer
- == pour tester l'égalité
- != pour la différence
- `in` pour l'appartenance à un `list` ou un `str`.
- `and`, `or` et `not` pour les opérateurs logiques.

```
3<=4 # True
"Paul" == "Georges" # False
"e" in "Fred" # True
1 not in [1,2,4] # False
```

instructions conditionnelles

De la forme

- si *<condition>* alors *<instructions>*
- si *<condition>* alors *<instructions>* sinon *<instructions>*
  - — sinon si *<condition>* alors *<instructions>*
  - — sinon si *<condition>* alors *<instructions>*
  - — ...
  - — (éventuellement) sinon alors *<instructions>*

Si se note `if`, sinon se note `else` et sinon si se note `elif`.

```
prenom = input("Entrez un prénom : ")
if prenom == 'Robert':
    print("Robert, comme mon grand—père.")
elif prenom == 'Raoul':
    print("Mon oncle s'appelle Raoul.")
elif prenom == 'Médor':
    print("Médor, comme mon chien !")
else:
    print("Je ne connais personne de ce nom.")
print('Au revoir.')
```

Les blocs conditionnels doivent être **décalés** avec la touche →|. On dit qu'ils sont indentés ou tabulés.

boucle for

une variable parcourt une structure itérable. Le bloc à boucler est indenté.
Boucle « classique » :

```
somme = 0
for i in range(101): # i parcourt {0;...;100}
    somme = somme + i
print(somme) #5050
```

Boucle plus originale :

```
liste = [1,2,1,3,1,4]
compteur = 0
for nombre in liste: # nombre parcourt la liste
    if nombre == 1: # on teste si c'est un 1
        compteur = compteur + 1
print(compteur) # 3
```

boucle while

On répète un bloc d'instructions tant qu'une condition est vérifiée. Le bloc est indenté.

```
u = 5
n = 1
while u < 100000:
    n = n + 1
    u = 2 * u — 1
print(n) # détermine un seuil
```

définition de fonction

On utilise `def` *nom_de_fonction(variables)* :
La fonction est indentée, le résultat est retourné avec `return`.

```
def f(a, b):
    return a*b/(a*a + b*b+1)
```

On appelle la fonction naturellement :

```
print(f(1,2))
```

Soit dans le programme principal, soit dans la console.