Modules

- Module : (" boîtes à outils ")
- Programme Python contenant des **fonctionnalités** qu'on souhaite **utiliser-réutiliser**
- Appelé aussi bibliothèque ou libraire

- Importation : exemple

In [ ]:
```
import random
for i in range(3):
    print(random.randint(0, 10))
```

- **Quelques modules** :
  - math : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
  - random : génération de nombres aléatoires.
  - time : accès à l'heure de l'ordinateur et aux fonctions gérant le temps.
  - re : gestion des expressions régulières.
  - urllib : récupération de données sur internet depuis Python.
  - Tkinter : interface python avec Tk. Création d'objets graphiques
  - os : dialogue avec le système d'exploitation.

Module random

- génération de nombres aléatoires
- Exemples d'utilisations :

- **Permuter aléatoirement des listes**

In [ ]:
```
import random
x = [1, 2, 3, 4]
print(x)
for i in range(3):
    random.shuffle(x)
    print(x)
```

- **Tirer aléatoirement un ou plusieurs éléments dans une liste :**

In [ ]:
```
import random
bases = ["T", "C", "G", "A"]
print(random.choice(bases))
print(random.choice(bases))
print(random.choices(bases, k=5)) # choices
print(random.choices(bases, k=5))
print(random.choices(bases, k=10))
```

- **Génération des nombres pseudo-aléatoires**
  - Basée sur une graine aléatoire via la fonction seed() : (U_0)

In [ ]:
```
print("Première fois :")
s = 10
for i in range(3):
    random.seed(13*i + s) # Générer des nombres pseudo-aléatoires
    print(random.randint(0, 100))
print("Deuxième fois :")
s = 10
for i in range(3):
    random.seed(13*i + s)
    print(random.randint(0, 100))
```

Fonctions

Algorithme: calcul de la factorielle , de l'Arrangement et la Combinaison

```
Variables A,C,F,n,k,i : Entier
Variables NF,NKF,KF : Entier
Ecrire("Calcul de la factorielle:")
Ecrire("Saisir un entier n:")
Lire(n)
F ← 1
Pour i allant de 1 à n
    F ← F * i
FinPour
Ecrire("La factorielle de ",n," : ",F)

Ecrire("----------------------------")
Ecrire("Calcul de l'Arrangement:")
Ecrire("Saisir un entier n:")
Lire(n)
Ecrire("Saisir un entier k (k ≤ n): ")
Lire(k)

NF ← 1
Pour i allant de 1 à n
    NF ← NF * i
FinPour

NFK ← 1
Pour i allant de 1 à n-k
    NKF ← NKF * i
FinPour

A ← NF / NKF

Ecrire("L'arrangement de",k," à ",n")
Ecrire(" vaut :",A)

Ecrire("----------------------------")
Ecrire("Calcul de la Combinaion:")
```

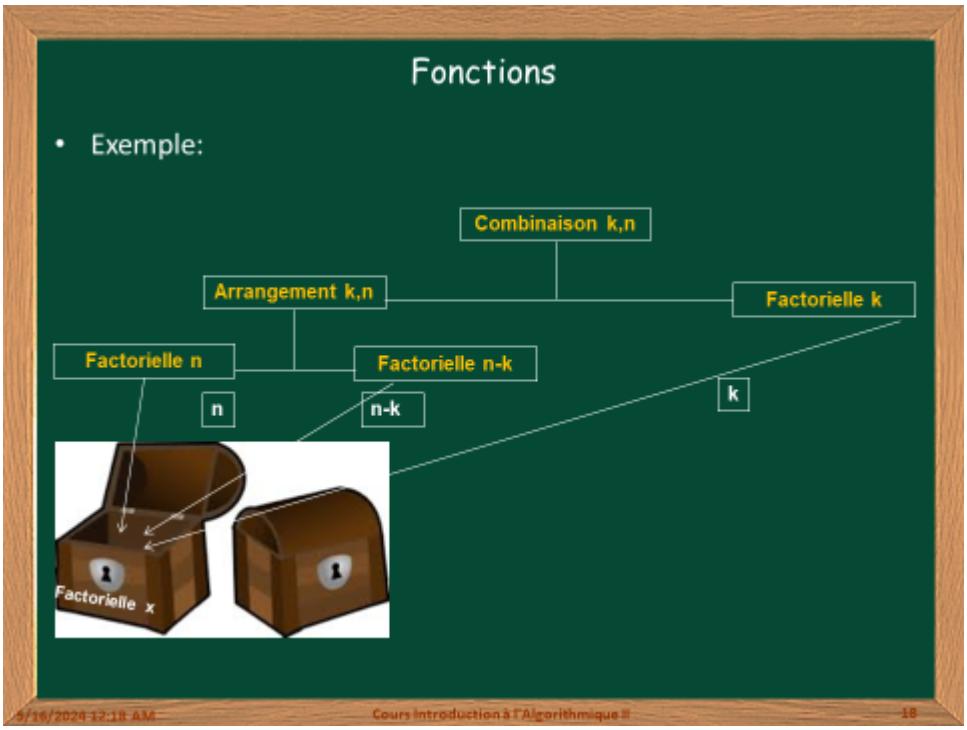
```
NF ← 1
Pour i allant de 1 à n
    NF ← NF * i
FinPour

NKF ← 1
Pour i allant de 1 à n-k
     NKF ← NKF * i
FinPour

FK ← 1
Pour i allant de 1 à k
    FK ← FK * i
FinPour
C ← NF / (KF * NKF)
Ecrire("La combinaison de ",k," parmi ",n," vaut :",C)
```

Question: Et si on voulait commencer le calcul de la factorielle à partir du rang 2:

```
FK ← 1
Pour i allant de 2 à n
     FK ← FK * i
FinPour
```



- Moyen de **réutiliser** des lignes de code
- Syntaxe en python :

```
def  NomFct ( suite de parametres ) :
        …
        Corps de la fonction
        …
```

In [1]:
```python
# Algorithme: calcul de la factorielle , de l'arrangement et la combinaison.

def Fact(m):
    F = 1
    for i in range(2,m+1):
        F = F * i
    return F
print("\n----------------------------")
print("Calcul de la factorielle:")
n = 5
NF = Fact(n)
print("La factorielle de ",n," : ",NF)

print("\n----------------------------")
print("Calcul de l'Arrangement:")
n = 5
k = 3
NKF = Fact(n-k)

print("n!     :",NF)
print("(n-k)! :",NKF)

A = NF // NKF
print("L'arrangement de",k," à ",n, " vaut :",A)

print("\n----------------------------")
print("Calcul de la Combinaion:")
FK = Fact(k)
print("k!     :",FK)

C = NF // (FK * NKF)
print("La combinaison de ",k," parmi ",n," vaut :",C)
```

```
----------------------------
Calcul de la factorielle:
La factorielle de  5  :  120

----------------------------
Calcul de l'Arrangement:
n!     : 120
(n-k)! : 2
L'arrangement de 3  à  5  vaut : 60

----------------------------
Calcul de la Combinaion:
k!     : 6
La combinaison de  3  parmi  5  vaut : 10
```

In [6]:
```python
def TimeS(x , i):
```

```
    return x*i
```

In [7]:
```
x = 4
print(TimeS(x,2))
```

```
8
```

In [4]:
```
x = "Bon"
print(TimeS(x,2))
```

```
BonBon
```

In [5]:
```
x = [13, 7, 2]
print(TimeS(x,3))
```

```
[13, 7, 2, 13, 7, 2, 13, 7, 2]
```

- Attention : **return** arrête l'execution de la fonciton

In [ ]:
```

```

Passage des paramètres

Soit la fonction d'incrémentation suivante

In [8]:
```
def Increm(x , y):
    x += 1
    y *= 3
```

In [9]:
```
x = 5 ; y = 4
print("Avant incrémentation : ")
print("x :" , x, "- y:" , y)

Increm(x , y)

print("")

print("Après incrémentation : ")
print("x :" , x, "- y:" , y)
```

```
Avant incrémentation :
x : 5 - y: 4

Après incrémentation :
x : 5 - y: 4
```

Les paramètres de la fonction ne change pas de valeurs à l'intérieur de la fonction

- **Solution** :

In [10]:
```
def increm(x , y):
    x += 1
    y *= 3
    return x , y
```

In [11]:
```
x = 5 ; y = 4
print("Avant incrémentation : ")
print("x :" , x, "- y:" , y)

x,y = increm(x , y)

print("")

print("Après incrémentation : ")
print("x :" , x, "- y:" , y)
```

```
Avant incrémentation :
x : 5 - y: 4

Après incrémentation :
x : 6 - y: 12
```

Passage des Paramètres : Cas des listes

- Méthode 1

In [12]:
```
def ajoute_Un(liste):
    for indice in range(len(liste)):
        liste[indice] += 1

# Programme principal.
liste_notes = [100, 80, 160, 70, 150]
print(liste_notes)

# Execution Fct ajoute_Un :
ajoute_Un(liste_notes)

print(liste_notes)
```

```
[100, 80, 160, 70, 150]
[101, 81, 161, 71, 151]
```

N.B. : les listes sont modifiables à l'intérieure des fonctions

Passage des Paramètres : Cas des listes

- Méthode 2

In [13]:
```
def ajoute_un(Liste):
    for indice in range(len(Liste)):
        Liste[indice] += 1
    return Liste

# Programme principal.
liste_notes = [10, 8, 16, 7, 15]
print(liste_notes)
liste_notes = ajoute_un(liste_notes)
print(liste_notes)
```

```
[10, 8, 16, 7, 15]
[11, 9, 17, 8, 16]
```

Valeurs par défaut des paramètres

In [14]:
```
x = 5 ; y_ = 2
def Puiss(x , p):
    return x**p
print(Puiss(x,y_))
```

25

In [15]:
```
def PuisS(x , p = 1):
    return x**p

x = 9
print(PuisS(x))
```

9

In [18]:
```
def PuiSS(x = 4 , p = 2):
    return x**p

print(PuiSS())
```

16

In [20]:
```
print(PuiSS(p = 3))
# print(PuiSS(13))
```

64

In [23]:
```
q = 5
# print(PuiSS(q=5))
print(PuiSS(q))
```

25

Variables Locales et Globales

Variables Locales (variables automatiques):

- Définie à l'**intérieure** d'une fonction
- Reconnue **uniquement à l'intérieure** de cette fonction
- S'**autodétruisent** à la fin de la fonction

In [25]:
```
# définition d'une fonction carre()
def carre(n):
    y2 = n**2
    print("à l'interieure de la fonction carre, y2 =", y2)
    return y

m = carre(3)
```

```
à l'interieure de la fonction carre, y2 = 9
```

Variable Locale :

- Utilisation à l'exterieure de la fonction ??

In [26]:
```
print("y à l'exterieure de la fonction carre, y2 =",y2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[26], line 1
----> 1 print("y à l'exterieure de la fonction carre, y2 =",y2)

NameError: name 'y2' is not defined
```

Variable Globale :

- Définie à l'**exterieure** d'une fonction
- Reconnue partout dans le code qui suit sa définition (1ère apparition)

In [27]:
```
print("du code...")

w = 10

print("w var Globale : ", w)

def doubl(n):
    y = n*2
    return y

print("Fct double :", doubl(4))

w += 9
print("Nouvelle valeur de w:",w)
```

```
du code...
w var Globale :  10
Fct double : 8
Nouvelle valeur de w: 19
```

Variable Globale :

- **Reconnue aussi à l'interieure des fonctions**

In [28]:
```
u = 101
def tripl(n):
    y = n*3
    print("u var Globale in Fct tripl : ", u)
    return y

print(tripl(4))
u += 5
print("u var Globale ext Fct tripl : ", u)
```

```
u var Globale in Fct tripl :  101
12
u var Globale ext Fct tripl :  106
```

Variable Globale ou Locale ???

- Définie :
  - à l'**exterieure** d'une fonction

- à l'**intérieure** d'une boucle ??

In [29]:
```
# exemple : test de la variable b

i = 16
while(i):
    i //= 4
    b = i
    print("b in :",b)
print("Apres b:",b)
```

```
b in : 4
b in : 1
b in : 0
Apres b: 0
```

Conflit Variable Locale Vs Variable Globale :

In [30]:
```
z = 10

# définition d'une fonction cube()
def cube(x):
    z = x**3
    print("à l'interieure de la fonction cube ap, z = ", z)
    return z+1

# programme principal
t = 4

print("z avant :", z)
resultat = cube(t)
print("z après :", z)

print("Resultat retour fct cube :", resultat)
```

```
z avant : 10
à l'interieure de la fonction cube ap, z =  64
z après : 10
Resultat retour fct cube : 65
```

- **Lors d'un conflit de nom, c'est la variable locale qui l'emporte**

(https://pythontutor.com/render.html#mode=display)

Récursivité

- Proche de l'esprit humain
- Analogie ~ mathématiques:
  - soit la suite :
    - U_0 = constante
    - $U_n = f(U_{n-1})$

Exemple : fonction Factorielle

In [ ]:
```
def Fact(n):
    if(n == 0):
        return 1
    return n*Fact(n-1)

print(Fact(0))
```

```
    Souvent la récursivité dans les fonctions est ainsi:
    fonction Fonc_Rec(paramètres) : Type_retoure
    début
    Si (cond arret 1 vérifiée )
        retourner valeur_1
    FinSi
    appel(s) récursivité Fonc_Rec(paramètres'>/b>)
    Fin

    - Attention, une mauvaise condition d'arrêt → Solution fausse.
```

- Une fonction récursive peut avoir plusieurs conditions d'arrêt .
  - Exemple: Fonction de Fibonacci

In [31]:
```
def Fibo_1(n):
    if n == 0 or n == 1:
        return n
    return Fibo_1(n-1) + Fibo_1(n-2)

m = 7
for i in range(m):
    print("Fibo", i, " :",Fibo_1(i))
```

```
Fibo 0  : 0
Fibo 1  : 1
Fibo 2  : 1
Fibo 3  : 2
Fibo 4  : 3
Fibo 5  : 5
Fibo 6  : 8
```

In [ ]:
```
LFB = [0 , 1]

# for j in range(len(LFB)):
#     print(LFB[j])

m = 7
for i in range(2,m):
    LFB.append(LFB[i-1] + LFB[i-2])

for i in range(m):
    print("Fibo 2 :", i, "::",LFB[i])
```

In [ ]:
```
import time
def Fibo_1(n):
    if n == 0 or n == 1:
        return n
    return Fibo_1(n-1) + Fibo_1(n-2)
```

```
nbf = 36

tic = time.time()
FBR = Fibo_1(nbf)
toc = time.time()

tps = toc-tic

T = list(range(nbf+1))
T[0] = 0; T[1] = 1

tic_it = time.time()
for i in range(2,len(T)):
    T[i] = T[i-1] + T[i-2]
toc_it = time.time()

tps_it = toc_it - tic_it

print("Fibo Récursive :",FBR)
print("Fibo Itérative :",T[len(T)-1])

print("")

print(f"Tps écoulé Récursive  :  {tps:.2f}   (sec)")

tps_it = (10**6) * tps_it
print(f"Tps écoulé Itérative  : {tps_it:.2f} (µ-sec)")
```

Récursivité : Exemple :: Recherche Dichotomique dans un tableau Trié

In [35]:
```
def RechDichRec(L, deb ,fin,e):
    if(fin < deb):
        return 0
    m = (deb+fin)//2
    if L[m] == e :
        return 1
    if L[m] < e :
        return RechDichRec(L, m+1 ,fin,e)
    else:
        return RechDichRec(L, deb, m-1,e)

L = [5, 7, 8, 11, 13, 16, 23, 31, 39]

el = 8
print(RechDichRec(L, 0 ,len(L)-1,el))
```

```
1
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Compléments chaine de caractères

In [ ]:
```
ch = "BonjourToutLeMonde"
L = len(ch)
print("La longueur :",L)
```

slicing :

In [ ]:
```
ch = "BonjourToutLeMonde"

print(ch[0:4])
print(ch[5:])
print(ch[:-2])
print(ch[1:-2:3])
```

' ' et / ou " "

In [ ]:
```
print("Un retour à la ligne\npuis une tabulation\t puis un guillemet\"")
print('J\'affiche un guillemet simple')
```

In [ ]:
```
print('Un brin d'ADN')
```

In [ ]:
```
print("Un brin d'ADN")
print('Cours "Python" pour API2')
```

In [ ]:
```
x = """
souris
chat
abeille
"""

x
```

In [ ]:
```
x = """souris
... chat
... abeille"""

print(x)
```

In [ ]:
```
s = "Voici un retour à la ligne\nEt là une autre ligne"
s
```

In [ ]:
```
print(s)
```

In [ ]:
```
s = r"Voici un retour à la ligne\nEt là une autre ligne"
s
```

In [ ]:
```
print(s)
```

Quelques méthodes pour chaines de caractères

In [ ]:
```
ch = "méthode"

print(ch)
print(ch.upper())
```

In [ ]:
```
w = "LETTRES"

print(w)
print(w.lower())
```

In [ ]:
```
z = "phrase avec separateur espace asse"
print(z[0].upper() + z[1:])
```

In [ ]:
```
print(".capitalize      :",z.capitalize())
print(".title           :",z.title())
```

In [ ]:
```
import string
z = "phrase avec separateur espace asse"
print("z : ",z)

print("string.capwords  :",string.capwords(z) )
print("string.capwords  :",string.capwords(z, sep = 'a') )
print("string.capwords  :",string.capwords(z, sep = 'as') )
```

Traitement chaînes / Fichiers , BI ...

Méthode .split() :

In [ ]:
```
desObjets = "livre souris ecran smartphone"
Lw = desObjets.split()
print("Lw de type : ",type(Lw))
print("")
print("Lw :" ,Lw)
```

In [ ]:
```
for unObjet in desObjets.split():
    print(unObjet)
```

In [ ]:
```
desObjets = "Livre:Souris:Ecran:Smartphone:Tab"

print(desObjets.split(":"))
print(desObjets.split(":", maxsplit=1))
print(desObjets.split(":", maxsplit=2))
```

In [ ]:
```
chemin = "D:\data\info\prog\python"
print(chemin.split("\\"))
```

Application Méthode .siplit() pour multisaisie

In [ ]:
```
# x, y = input("Enter two values: ").split()
# x, y = input("Enter two values: ").split('-')
w = input("Enter two values : ")
x,y = w.split()

print("Val saisies * 10 :", int(x)*10,int(y)*10)
```

Méthode .find() :

In [ ]:
```
desObjets = "Livre:Souris:Ecran:Smartphone:Tab"
chemin = "D:\data\info\prog\python"

pos_Ob = desObjets.find(":")
print(pos_Ob)
print(desObjets[pos_Ob+1 : ])

pos_ch = chemin.find("\\") # find le caractère \
print(chemin[pos_ch+1:])

pos_ch = chemin.find(":\\") # find la chaine :\
print(chemin[pos_ch+1:])
print(chemin[pos_ch+len(":\\"):])
```

Méthode .replace() :

In [ ]:
```
desObjets = "Calcul:AppPhoto:Livre:TV:Montre"
print(desObjets.replace("Calcul:AppPhoto:Livre:TV","Smartphone"))
```

Méthode .count() :

In [ ]:
```
desObjets = "Calcul:AppPhoto:Livre:TV:Montre:Tab"
print("occ l :", desObjets.count("l"))
print("occ L :", desObjets.count("L"))

chMin = desObjets.lower()
print("occ l :", chMin.count("l"))
```

Méthode .startswith()

In [ ]:
```
chaine = "Bonjour monsieur !"

print("Commence par Bonjour :",chaine.startswith("Bonjour"))

print("Commence par Hello :", chaine.startswith("Hello"))
```

Méthode .strip()

- Supprimer les espaces au début et à la fin

In [ ]:
```
chs = "      Supprimer    les espaces au début et à la fin ?       "
chs.strip()
```

In [ ]:
```
ch = "  \tfonctionne avec tabulations et retours à la ligne\n"
print(ch.strip(), end = "*")
```

```
print("apres")
```

Méthode .join()

In [ ]:
```
seq = ["A", "T", "G", "A", "T"]
seq
```

In [ ]:
```
resL = "-".join(seq)
print(resL)
print("")
resCH = "QRSTUVWXYZ"
print("_".join(resCH))
```

In [ ]:
```
ch = " "
print(ch.join(resCH))
```

In [ ]:
```
print("".join(seq))
```

Complément sur les Listes

- .insert(pos , elem)

In [ ]:
```
# Rappel
a = [10, 20, 30]
a.append(9)
a
```

In [ ]:
```
a = [7, 5, 13]
pos = 0
elem = 2
a.insert(pos,elem)
print(a)
```

In [ ]:
```
a.insert(35 , 11)
print(a)
```

In [ ]:
```
a = [10, 20, 30]
a.insert(-1 , 333)
print(a)
```