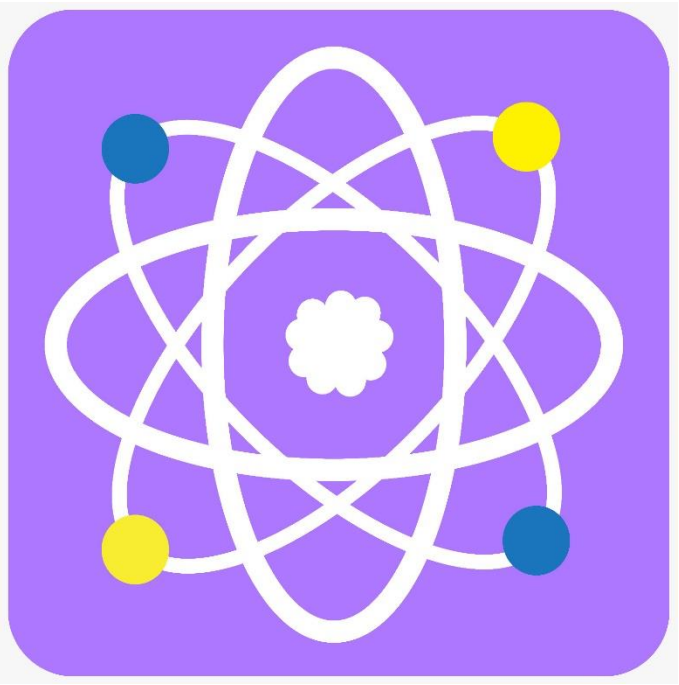


Ismail ELBOUKNIFY

- Python est un langage de programmation populaire. Il a été créé par Guido van Rossum et sorti en 1991.
- Il est utilisé pour :
    - développement web
    - développement de logiciels,
    - mathématiques,
    - script système,
    - Analyse de données et Intelligence artificielle …

Pourquoi Python ?

- Python fonctionne sur différentes plates-formes (Windows, Mac, Linux, Raspberry Pi, etc.).
- Python a une syntaxe simple similaire à la langue anglaise.
- Python a une syntaxe qui permet aux développeurs d'écrire des programmes avec moins de lignes que certains autres langages de programmation.
- Python peut être traité de manière procédurale, orientée objet ou fonctionnelle.

Les entrées/sorties standards

Dans l'algorithmique :

Ecrire("Entrer la valeur de la Note :")
Lire (Note)

Dans python :

print("Entrer la valeur de la Note :")
Note=**input**()

Ou bien :
Note=**input**("Entrer la valeur de la Note :")

Exemple

```
print("Entrer la valeur de la Note :")
Note=int(input())
Note+=2
print(Note)
```

Les entrées/sorties standards

La fonction **print**() a plusieurs paramètres :

- **sep** : les objets sont séparés par sep. Valeur par défaut: ' '.

```
a=input("Entrer une valeur :")
print("valeur ",a, sep =" = ",)
```

- Variable = **input**("Message à l'utilisateur ")
- Variable = **type(input**("Message à l'utilisateur "))

Commentaire Python

Les commentaires peuvent être utilisés pour expliquer le code Python.

- Les commentaires commencent par un #
- Commentaires multi-lignes :

```
"""
Ceci est un commentaire
écrit en plus que juste
une ligne
"""
```

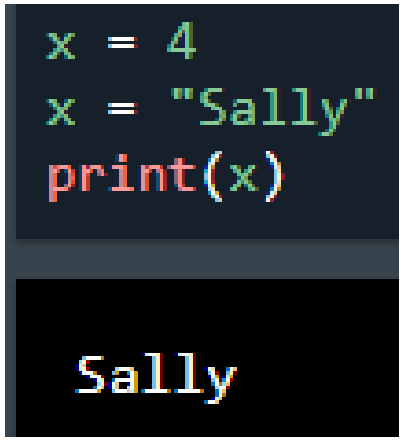
variables

- Le premier caractère de la variable doit être un alphabet ou underscore (_).
- Tous les caractères, à l'exception du premier caractère, peuvent être un alphabet composé de minuscules (a-z), de majuscules (A-Z), de traits de soulignement ou de chiffres (0-9).
- Le nom de l'identifiant ne doit pas contenir d'espace blanc, ou caractère spécial (!, , #, %, ^, &, *).
- Le nom de l'identifiant ne doit pas être similaire à un mot clé défini dans la langue(and , if , from ,True , Class ...
- Les noms d'identifiant sont sensibles à la casse
    - Exemples d'identifiants valides : **a123,_n, n_9, etc.**
    - Exemples d'identifiants invalides **:** **1a, n%4, n 9,** etc.

Déclaration et affectation de variables

- Python ne nous oblige pas à déclarer une variable avant de l'utiliser dans l'application. Cela nous permet de créer des variables au moment requis.
- Les variables n'ont pas besoin d'être déclarées avec un type particulier et peuvent même changer de type après avoir été définies.
- L'opérateur égal (=) est utilisé pour attribuer une valeur à une variable.

```
>>> a=12#Affect 12 to variable a
>>> type(a)#return the type of the variable a
<class 'int'>
```



Affectations multiples de variables

- En Python, nous pouvons attribuer plusieurs valeurs à plusieurs variables.

```
x, y, z = 1, -4, "ismail"
print(x)
print(y)
print(z)
```

```
1
-4
ismail
```

```
x = y = z = 5
print(x)
print(y)
print(z)
```

```
5
5
5
```

Casting variables

- Si vous souhaitez spécifier le type de données d'une variable, vous pouvez le faire avec.

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
print(type(x))
print(type(y))
print(type(z))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
```

Types de données

Python a les types de données suivants :

- Le type **int** (Entier)
- le type **float** (Réel)
- Le type **bool** (Booléen)
- Le type **str** (Caractères)

Opération sur les variables

** : (puissance)

* : (multiplication)

/ : (division)

// : (division entière)

% : (modulo)

+ , - : (addition, soustraction)

= = : égal

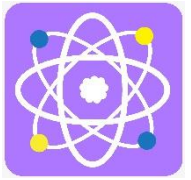
! = Différent

x+=3 ←→ x=x+3

x-=3 ←→ x=x-3

x*=3 ←→ x=x*3

x%=3 ←→ x=x%3



Exercice

1 / Ecrire un programme en python qui calcule la somme de deux Nombre entier saisies au clavier .

2 / Écrivez un programme qui demande à l'utilisateur un nombre, puis calcule et affiche le carré de ce nombre.

Solution

1:

```
Note1=int(input("Entrer le premier nombre :"))
Note2=int(input("Entrer la dexième nombre :"))
print("la somme est :",Note1+Note2)
```

```
Entrer le premier nombre :1

Entrer la dexième nombre :2
la somme est : 3
```

2:

```
x=int(input("donner un nombre    :"))
print("le carré de ",x,"est ",x*x)
```

```
donner un nombre    :5
le carré de  5 est  25
```

Schéma alternatif ou conditionnel

La sélection :Permet d'exécuter des instructions quand une condition est vérifiée.

Algorithme

```
Si condition Alors
    instructions
    ...
Fin Si
```

Python

```
if condition :
    instructions
    ...
```

Algorithme

```
Si condition Alors
    instructions
    ...
Fin Si
Sinon
    instructions
    ...
Fin Sinon
```

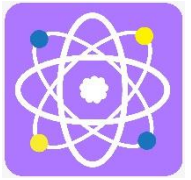
Python

```
if condition :
    instructions
    ...
else :
    instructions
    ...
```

Schéma alternatif ou conditionnel

The syntax of if-else statement

```
if expression 1:
        block of statements
elif expression 2:
        block of statements
elif expression 3:
        block of statements
else:
        block of statements
```



Exercices

1 / Entrez un nombre flottante. S'il est positif ou nul, affiche sa racine carrée, sinon affiche un message d'erreur. (utilise la fonction sqrt() de la bibliothèque **math)**

2/ Écrivez un programme qui lit un entier et détermine si cet entier est un nombre impair ou pair .

Solution

1:

```
from math import sqrt

x=float(input("donner un nombre "))
if x>=0:
    print("la racie de ",x,"est :",sqrt(x))
else:
    print("erreur")
```

```
donner un nombre :9
la racie de  9.0 est : 3.0
```

2:

```
from math import sqrt

x=float(input("donner un nombre :"))
if x%2==0:
    print(x,"est un nombre pair")
else:
    print("est un nombre impair")
```

```
donner un nombre :5
est un nombre impair
```

Schéma itératif ou répétitif

Boucle : Pour

Algorithme

Pour variable ← valeur_initiale **à** valeur_finale **de pas** p **Faire**
    instructions
    ...
Fin Pour

Python

```
for variable in range(valeur_initiale, valeur_finale, p) :
    instructions
    ...
```

Boucle : Pour

Exemples :

-
```
for i in range(0, 10, 2) :
    print(i)
```
donne le résultat : 0, 2, 4, 6, 8
-
```
for i in range(0, 10) :
    print(i)
```
donne le résultat : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
-
```
for i in range(10) :
    print(i)
```
donne le résultat : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Schéma itératif ou répétitif

Boucle : Tant que

Algorithme

```
Tant que condition Faire
    instructions
    ...
Fin Tant que
```

Python

```
while condition :
    instructions
    ...
```

Exemple

```
i=0
while i<5 :
    print(i)
    i+=1
```

```
0
1
2
3
4
```

Schéma itératif ou répétitif

Sort immédiatement de la boucle **for** ou **while** en cours contenant l'instruction

- Break :

Exemple :

```
for i in range(10) :
    if i==5 :
        break
    print(i)
```

donne le résultat : 0, 1, 2, 3, 4

- continue :

Exemple :

```
for i in range(10) :
    if i==5 :
        continue
    print(i)
```

donne le résultat : 0, 1, 2, 3, 4, 6, 7, 8, 9

Exercices

1/ Ecrire une boucle affichant et incrémentant la valeur de **a** tant qu'elle reste inférieure à celle de **b**. (Initialiser deux entiers : a = 0 et b = 10)

2/ Utilisez l'instruction **continue** pour modifier une **boucle for** affichant tous les entiers de 1 à 10 inclus, sauf lorsque la variable de boucle est **5**.

Solution

1:

```
a, b = 0, 10
while a < b:
    print(a)
    a = a + 1
```

2:

```
for i in range(1, 11):
    if i == 5:
        continue
    print(i, end=" ")
```

```
1 2 3 4 6 7 8 9 10
```

Les chaînes de caractères

- Une chaîne de caractères est une suite finie de caractères consécutifs, qu'on note entre apostrophes ' ' ou guillemets " "
- Opération de concaténation +

```
chaine1 = "Tu sais programmer en python"
chaine2 = ' C'est facile'
chaine3=chaine1+chaine2
print(chaine3)
```

```
Tu sais programmer en python C'est facile
```

- Opération de multiplication *

```
ch = " python"
ch2=ch*3
print(ch2)
```

```
 python python python
```

Les chaînes de caractères

Accès à un caractère :

- Pour accéder à chacun des caractères de la chaine de caractères on utilise ([i]) : où i est un indice qui indique la position du caractère

```
ch = " python"

print(ch[2])
```

```
 y
```

- On ne peut pas modifier les caractères d'une chaine de caractères

Les chaînes de caractères

- Pour déterminer la longueur d'une chaîne, en appel à la fonction **len**()

```
ch = " python"

print(len(ch))
```

```
7
```

- Pour extraire une sous-chaîne de ch, on écrit ch[i :j] où i est l'indice du premier caractère de la sous-chaîne et j-1 est l'indice du dernier caractère :

```
ch = "python"
print(ch[1:3])
```

```
yt
```

Les chaînes de caractères

l'opérateur **in** sert à :

- afficher les caractères d'une chaîne de caractères

```
ch = "python"
for c in ch :
    print(c, end=" ")
```

```
p y t h o n
```

- tester l'appartenance d'un caractère à une chaîne de caractères

```
ch = "python"
if "e" in ch :
    print("exist")
else:
    print("n'exist pas")
```

```
n'exist pas
```

Exercices

1 / Écrivez un programme qui lit une chaine et Afficher chaque caractère de cette chaîne à l'aide d'une boucle for.

2 / Écrivez un programme qui lit une chaine et d'afficher l'inverse de cette chaine

Exemple : math ➔ tham

Solution

1:

```
c=input("donner une chaine : ")
for i in c :
    print(i)
```

```
donner une chaine : bonjour
b
o
n
j
o
u
r
```

2:

```
c=input("donner une chaine : ")
n=len(c)
while(n>0):
    print(c[n-1],end="")
    n=n-1
```

```
donner une chaine : math
htam
```

Liste

- Les listes sont utilisées pour stocker plusieurs éléments dans une seule variable.
- Créer une liste:

Liste = [item1 ,item2 , item3,....]

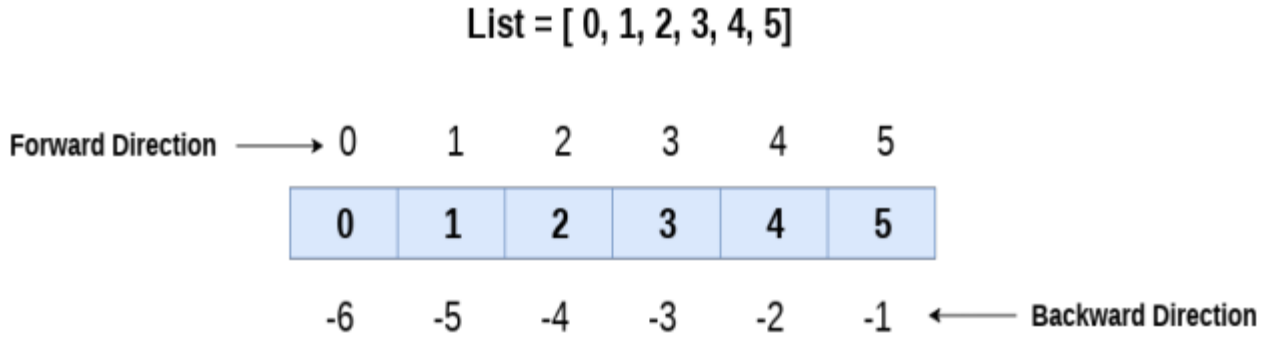
```
L=[1,2,3,4,5]
print(L)
```

```
[1, 2, 3, 4, 5]
```

- Indexation et fractionnement de liste

```
L=[1,2,3,4,5]
print(L[-2])
```

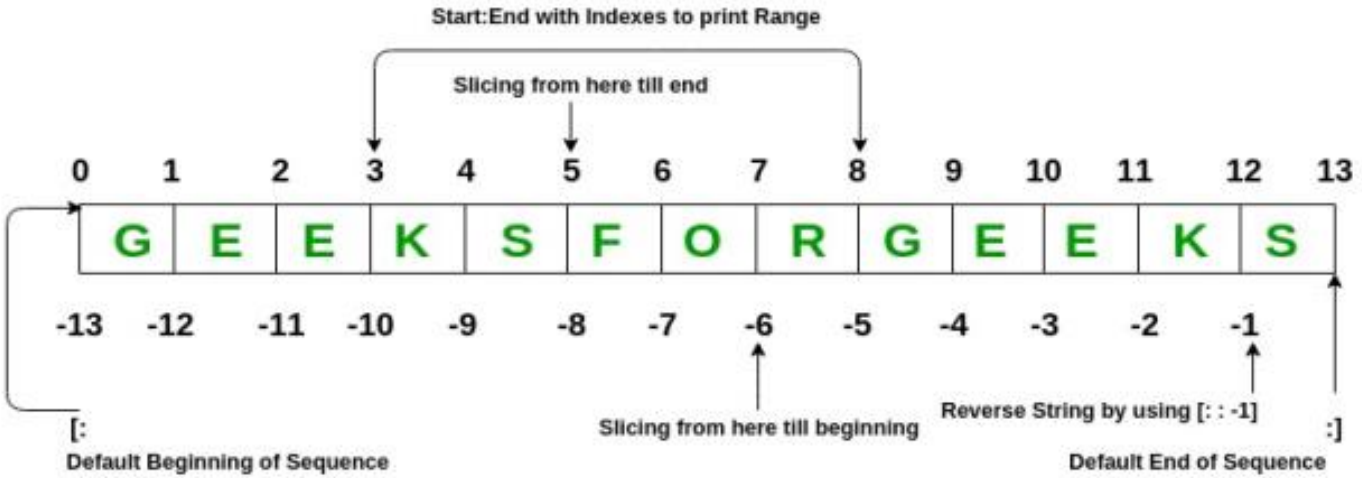
```
4
```



Liste

```
L=[1,2,3,4,5]
print(L[0:2])
```

```
[1, 2]
```




```
L=[1,2,3,4,5]
print(L[0:4:2])
```

```
[1, 3]
```

```
L=[1,2,3,4,5]
print(L[-1:0:-1])
```

```
[5, 4, 3, 2]
```

Liste

- Modifier un élément d'une liste : **Liste[index]=item1**
- Modifier une tranche d'une liste**: Liste[start:end]**=[elm1,elm2,....]

```
L=[1,2,3,4,5]
L[2:]=[0,0]
print(L)
```

```
[1, 2, 0, 0]
```

- Tester l'existence d'un élément :

```
L=[1,2,3,4,5]
if 2 in L:
    print("exist")
```

```
exist
```

Liste

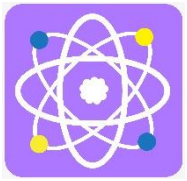
- Supprimer élément d'une liste : **del** Liste[index]
- Supprimer une tranche d'une liste : **del** Liste[start:end:step]
- Ajouter un élément à une liste : Liste.**append**(item)
- Parcourir une liste :

```
L=[1,2,3,4,5]
for i in L:
    print(i)
```

```
1
2
3
4
5
```

```
L=[1,2,3,4,5]
for i in range(len(L)):
    print(L[i])
```

```
1
2
3
4
5
```



Liste

Opérations de liste Python :

append() Ajoute un élément à la fin de la liste

clear() : Supprime tous les éléments de la liste

copy() : Renvoie une copie de la liste

count() : Renvoie le nombre d'éléments avec la valeur spécifiée

insert() : Ajoute un élément à la position spécifiée

pop() : Supprime l'élément à la position spécifiée

remove() : Supprime l'élément avec la valeur spécifiée

reverse() : Inverse l'ordre de la liste

sort() : Trie la liste

Exercice

1 / Écrivez un programme qui lit un nombre entier et de savoir si le nombre existe dans la liste L=[1,2,4,7]

2 / combien de fois le nombre saisie répète dans la liste

Solution

1:

```
L=[1,2,4,7]
x=int(input("donner un nombre : "))
if x in L:
    print(x," exist dans la liste")
else:
    print(x," n'exist pas dans la liste")
```

```
donner un nombre : 3
3  n'exist pas dans la liste
```

```
donner un nombre : 2
2  exist dans la liste
```

2:

```
L=[1,2,4,7,7,1,7]
x=int(input("donner un nombre : "))
a=0
for i in range(len(L)):
    if L[i]==x:
        a=a+1
print(x,"repet ",a,"fois dans la liste")
```

```
donner un nombre : 7
7 repet  3 fois dans la liste
```

Les dictionnaires

- Un dictionnaire est un tableau associatif, c.à.d. un type de données permettant de stocker des couples {**clé : valeur**}, avec un accès à la valeur à partir de la clé, la clé ne pouvant être présente qu'une seule fois dans le dictionnaire.
- Dict = *{*Key1: Value1,Key2: Value2,....*}*
- Pour accéder aux éléments contenus dans le dictionnaire, on utilise des **clés**

Les dictionnaires

- L'instruction **del** vous permet d'effacer des éléments d'un dictionnaire en fonction de leur clé
- Vous pouvez utiliser une boucle **for** et l'opérateur **in**

```
>>> D = {'nom':'El mohamadi', 'prenom':'Mohamed',
         1:[11,8,2012]  ,  'ville':'Tanger'}
>>> for elt in D:
        print(elt)

ville
nom
prenom
1
```

```
>>> for elt in D:
        print(D[elt])

Tanger
El mohamadi
Mohamed
[11, 8, 2012]
```

Les dictionnaires

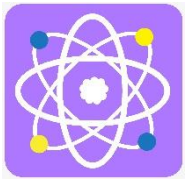
- Vous pouvez utiliser la méthode **keys()** pour afficher les clés
- Pour effectuer un traitement sur les valeurs, utiliser la méthode **values()**

```
>>> D = {'nom': 'El mohamadi', 'prenom': 'Mohamed', 1 :[11,8,2012], 'Ville'
:'Tanger'}
>>> for elt in D.keys():
        print(elt)


Ville
nom
prenom
1
```

```
>>> for elt in D.values():
        print(elt)


Tanger
El mohamadi
Mohamed
[11, 8, 2012]
```



Exercice

On considère le dictionnaire suivant dont les clés sont les noms des élèves et les valeurs des clés sont les moyennes générales obtenues en passant l'examen final:

etudiants = {"etudiant_1" : 13 , "etudiant_2" : 17 , "etudiant_3" : 9 , "etudiant_4" : 15 , "etudiant_5" : 8 , "etudiant_6" : 14}

Ecrire un programme Python qui partitionne ce dictionnaire en deux sous dictionnaires:

- **etudiantAdmis** dont les clés sont les étudiants admis et les valeurs des clés sont les moyennes obtenues (moyenne supérieurs ou égales à 10).
- **etudiantNonAdmis** dont les clés sont les étudiants non admis et les valeurs des clés sont les moyennes obtenues (moyenne inférieur ou égale à 10).

Solution

```
etudiants = {"etudiant_1" : 13 , "etudiant_2" : 17 , "etudiant_3" : 9

etudiantAdmis = {}
etudiantNonAdmis = {}

for key , valeur in etudiants.items():
    if(valeur < 10):
        etudiantNonAdmis[key] = valeur
    else:
        etudiantAdmis[key] = valeur

print(etudiantAdmis)
print(etudiantNonAdmis)
```

```
{'etudiant_1': 13, 'etudiant_2': 17, 'etudiant_4': 15,
'etudiant_6': 14}
{'etudiant_3': 9, 'etudiant_5': 8}
```

Les fonctions

Division du programme complexe en plusieurs sous-programmes :

- Moins complexes
- Indépendants
- Réutilisables

La syntaxe Python pour la définition d'une fonction est la suivante :

```
def Nom_Fonction(liste de paramètres) :
    instruction1
    instruction2
    ...
    return expression
```

Pour utiliser la fonction il suffit de l'appeler par son nom : **Nom_Fonction(liste de paramètres)**

Les variables globales et locales

Les variables locales :

- Les variables locales sont les variables qui n'existent que dans le corps de la fonction.
- Les paramètres sont aussi des variables locales.
- A la sortie de la fonction, les variables locales sont perdues

Les variables globales :

- Une variable globale définie en dehors de toute fonction.
- Une variable globale est connue dans toute la portion de code qui suit sa création.

Exercice

1/ Ecrire une fonction permet de remplir une liste et d'afficher cette liste

2/ Ecrire une fonction **puissance_recursive** prenant en entrée deux paramètres x et n et calculant récursivement x^n, en exploitant l'égalité : $x^n = x * x^{n-1}$

Solution

1:

```python
def creat_List(n):
    L=[]
    for i in range(n):
        x=int(input("donner une valeur : "))
        L.append(x)
    return L
n=int(input("donner la taille de la liste : "))
L=creat_List(n)
print(L)
```

```
donner la taille de la liste : 3

donner une valeur : 1

donner une valeur : 2

donner une valeur : 3
[1, 2, 3]
```

2:

```python
def puissance_recursive(x,n):
    if n==0 :
        return 1
    return x*puissance_recursive(x,n-1)

x=int(input("donner x  :"))
n=int(input("donner n :"))
print(x," à la puissance de ",n," est :", puissance_recursive(x,n))
```

```
donner x  :4

donner n :3
4  à la puissance de  3  est : 64
```

packages Python

Un package est une collection de fonctions existantes qui peut être utilisée dans votre code.

- **Data science : NumPy , SciPy , Pandas , Matplotlib, TensorFlow , Keras , SciKit-Learn ….**
- **développement web : Django, TurboGears …**
- **interface graphique : Tkinter …**
- **Applications Mobile : Kivy …**