Python

Quelques bases : Calculs, opérations

Dans le tableau ci-après, sont présentés les symboles utilisés pour les opérations de base.

Opérations	Symboles	Exemples
Addition	+	2 + 5 donne 7
Soustraction	-	8 - 2 donne 6
Multiplication	*	6 * 7 donne 42
Puissance	**	5 ** 3 donne 125
Division	/	7 / 2 donne 3.5
Reste de division entière	%	7 % 3 donne 1
Quotient de division entière	//	7 // 3 donne 2

Variables, affectations, calculs

Affecter une valeur à une variable :

```
a=3
```

Afficher la valeur de la variable, avec ou sans commentaire :

```
print(a)
print("a =",a)
```

Addition de 1 à la valeur de a :

```
a=a+1
print("a =",a)
```

Affectations multiples :

```
a=b=c=5
print("a =",a,"   b =",b,"   c =",c)
```

d,e=10,4.3 Attention : on met un point décimal dans 4.3 et non une virgule la virgule est un séparateur

```
print("d =",d,"   e =",e)
```

Echange de deux variables :

```
print("avant : a =",a,"   b =",b)
a,b=b,a
print("après : a =",a,"   b =",b)
```

Lire variable :

```
f=input("Valeur de f ? ")
print("2f =",2*f)
```

Attention : input renvoie une chaine de caractères ! Si f="3", 2*f est "33" (chaîne de caractères)

il faut préciser si notre variable est entière … :

```
f=int(input("Valeur de f (entier) ? "))
print("2f =",2*f)
```

… ou décimale :

```
f=float(input("Valeur de f (décimal) ? "))
print("2f =",2*f)
```

Les tests conditionnels if

Notion de bloc avec indentation (décalage vers la droite de certaines lignes)

Test simple (si... alors) :

```
if condition :
        instructions
```

Exemple :

```
a=demande("Donnez un nombre : ")
if a<10:
      print("ce nombre est inférieur à 10")
```

On note que la ligne avec "if" se termine par ":"
Tout le bloc correspondant au traitement est décalé c'est ce qu'on appelle l'indentation

Test plus complet (si... alors... sinon) structure générale puis exemple :

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

```
a=float(input("Donnez un autre nombre : "))
if a<10:
      print("ce nombre est inférieur à 10")
else:
      print("ce nombre est supérieur ou  égal à 10")
```

Les tests if … elif ... else

```
if condition 1 :
      instructions
 elif condition 2 :
      instructions
 else :
      instructions
```

remarque : elif est la contraction de "else if"

Exemple : Le tarif d'un spectacle est le suivant : 15€ par personne pour 1 ou 2 personnes
Pour 3 personnes, le tarif est réduit de 20 % et pour 4 personnes et plus, la réduction est de 30 %. Voilà un programme qui calcule le prix à payer.

```
nb_personne=int(input("Nombre de personnes ?"))
if nb_personne <=2:
        prix=nb_personne*15
elif nb_personne==3:
        prix=nb_personne*15*0.8
else:
        prix=nb_personne*15*0.7
print("prix à payer :",prix)
```

Les différentes comparaisons :
a==10 signifie a est égal à 10. Exemple d'utilisation : if a==10… Ne pas confondre avec a=10 qui affecte la valeur 10 à la variable a
a!=10 signifie a est différent de 10. Exemple d'utilisation : if a!=10...
a>=10 signifie a est supérieur ou égal à 10
a<=10 signifie a est inférieur ou égal à 10
a>10 signifie a est strictement supérieur à 10
a<10 signifie a est strictement inférieur à 10

Double inégalité
Exemple :
```
a=demande("Donnez un 3e nombre : ")
if 10<=a<100:
    print("ce nombre a 2 chiffres")
else:
    print("ce nombre est inférieur à 10 ou supérieur à 99")
```

Tests imbriqués
Exemple :
```
a=float(input("Donnez un 4e nombre : "))
if a<10:
    print("ce nombre a 1 chiffre")
else:
    if a<100:
        print("ce nombre a 2 chiffres")
    else:
        print("ce nombre a 3 chiffres ou plus")
```

On remarque l'indentation : le 2e "else" est dans le bloc du 1er "else"

Boucle conditionnelle : while (tant que)

Structure générale : c'est la même que celle de if :
while condition :
instructions
La boucle s'exécute tant que cette condition est vraie
Ne pas oublier d'initialiser la variable testée, ici i=100.

Exemple :
```
i=100
while i!=0:
    print(i)
    i=i-1
print("Boum !")
```

Boucle itérative : for (pour ...)

Tout d'abord, la fonction "range"
range(début, fin, incrément ou pas). Exemple : pour obtenir [2, 6, 10, 14, 18] :
```
a=range(2,22,4)
print(list(a))
```
Ici, on affiche tous les entiers à partir de 2, de 4 en 4, jusqu'à 22 EXCLUS
a est une variable spéciale : c'est une liste d'où la syntaxe pour l'afficher.

Si l'incrément vaut 1, il est inutile de le mettre :
```
b=range(2,10)
print(list(b))
```
On voit que 10 ne figure pas dans la liste

Si on part de 0, il est inutile de le marquer. Pour avoir [0, 1, 2, 3, 4], on met :
```
c=range(5)
print(list(c))
```
for i in range(n) fait aller i de 0 à n-1

Maintenant, une boucle itérative. Structure générale puis exemple :
for i in range(…) :
instructions
```
for i in range(5):
    print(i, "a pour carré",i**2)
```

On peut mettre un incrément négatif :
```
for i in range(10,5,-1):
    print(i, "a pour carré",i**2)
print()
```

Idem, le dernier terme, 5, n'est pas affiché

On peut se faire une liste sur mesure
```
for i in [2,5,6,8,4]:
    print(i,'-->',i**2)
```
la boucle est effectuée pour la liste des valeurs données

Les fonctions

On peut définir des fonctions et les utiliser ailleurs dans le programme

Exemple : un petit tableau de valeurs facile
```
def f(x):
    return 2*x**2+3

for i in range(11):
    print("f(",i,") = ",f(i))
```

Il faut mettre : à la fin de la ligne def f(x), et pas d'accent sur def.
Ensuite, il y a indentation et on termine en « retournant » f(x) avec return ….

Les modules

Qu'est-ce qu'un module ? Il s'agit d'une sorte de bibliothèque (un regroupement de fonctions prédéfinies) qui une fois importée permet d'accéder à de nouvelles fonctions. Il en existe beaucoup. On peut citer :
- le module **turtle** qui permet de réaliser des dessins géométriques,
- le module **numpy** qui permet de faire du calcul scientifique,
- le module **sympy** qui permet de faire du calcul formel,
- le module **matplotlib** qui permet de faire des graphiques en tout genre,

……..
- le module **math** permet d'avoir accès aux fonctions mathématiques comme le cosinus (cos), le sinus (sin), la racine carrée (sqrt), le nombre π(pi) et bien d'autres.
- le module **mpmath** qui permet de calculer avec des nombres réels ou complexes avec une précision arithmétique fixée.