1 Types simples et type composés

1.A Types simples

On utilisera trois types simples en Python : nombres entiers, flottants (décimaux), booléen (indiquant si une propriété est vraie ou fausse).

```
>>> type(45)
<class 'int'>
>>> type(45.1)
<class 'float'>
>>> type(4<8)
<class 'bool'>
```

Les opérations arithmétiques usuelles (+,-,/,*) peuvent être utilisées pour les entiers et flottants.

Pour les booléens, on pourra utiliser la disjonction (`or`) et la conjonction (`and`) :

```
>>> True and False
False
>>> True and True
True
>>> not True
False
>>> not (not True)
True
```

```
>>> 1<2 and 1==3
False
>>> 2<3 and 1+3==4
True
>>> not 1<2
False
>>> not (not 1==1)
True
```

1.B Types composés

1.B.1 Tuples

Un n-uplet, *tuple* en anglais, est une généralisation du concept de couple ou de triplet. On peut voir ce type comme la traduction informatique d'un produit cartésien d'ensembles. Comme en mathématiques, pour construire une expression n-uplet, il suffit de placer des expressions entre parenthèses séparées par des virgules.

Voici un couple et un triplet constitués d'entiers/flottants :

```
>>> (1,2.2)
(1, 2.2)
```

```
>>> (0,1,2)
(0, 1, 2)
```

Comme pour les autres types, il est possible de stocker un n-uplet dans une variable :

```
>>> t=(1,2.2)
```

Pour accéder aux composantes d'un n-uplet, c'est-à-dire aux sous-valeurs qu'il contient, on utilise l'expression `t[i]` où `i` est l'indice de la composante.

Attention. En Python, on commence à numéroter à partir de 0 et non de 1. Pour obtenir la première ou la seconde composante, on pourra évaluer :

```
>>> t[0]
1
```

```
>>> t[1]
2.2
```

Attention. Les n-uplets sont immuables. Cela signifie qu'il n'est pas possible d'affecter de nouvelles valeurs aux composantes. Ainsi l'instruction suivante produit une erreur.

```
>>> t[0]=5
TypeError: ....
```

Il est possible de coller un n-uplet et un p-uplet pour obtenir un $(n+p)$-uplet. On parle de *concaténation*. L'opérateur correspondant en Python est l'opérateur `+`.

```
>>> (1,2)+(3,4,5)
(1, 2, 3, 4, 5)
```

1.B.2 Test d'appartenance

Il est possible de tester si une valeur appartient à un n-uplet l'aide de l'opérateur `in`.

```
>>> 3 in (1,2,3)
True
```

1.B.3 Longueur d'un n-uplet

On obtient la longueur d'un n-uplet à l'aide de la fonction `len` :

```
>>> len((1,2))
2
>>> len(())
0
```

1.C Listes

Une *liste* est un n-uplet dont on peut changer les composantes.

Pour construire une liste, on remplace les parenthèses par des crochets. Toutes les opérations vues pour les n-uplets sont définies. Ici cependant, aucune erreur n'apparaît si on change la valeur d'un élément.

```
>>> [1,2,3]
[1, 2, 3]
```

```
>>> L=[1,2]
>>> L[1]
2
>>> L+[3,4]
[1, 2, 3, 4]
```

```
>>> L[0]=5486
```

En évaluant l'expression L, on se rend compte que cet élément a bien changé :

```
>>> L
[5486, 2]
```

1.D Chaînes de caractères

Il existe également un type chaines de caractères permettant de stocker des mots, des phrases.

```
s="bonjour"
t="␣classe␣de␣PTSI"
u=s+t # u contient "bonjour classe de PTSI"
```

Les propriétés des chaines de caractères sont les mêmes que celles des n-uplet (caractères immuable, accès aux caractères, contcaténation...)

Remarques

On peut accéder à une sous-liste, un sous-tuple, une sous chaîne à l'aide de ce qu'on appelle un slice :

```
L=[1,3,5,6,8]
L1=L[0:3] # L1 est la sous-liste [1,3,5]
L2=L[0:len(L)] # L2 est identique à L
L3=l[1:-1] # -1 remplace len(L)-1
```

2 Tests conditionnels

<u>Exercice :</u> Taper et exécuter les commandes suivantes. Constater.

```
# Les tests conditionnels
n=int(input("entrer␣un␣entier␣n␣:␣"))
m=int(input("entrer␣un␣entier␣m␣:"))

# Attention : input renvoie une chaine de caractères.
# La commande int permet de convertir en entier

# Test d'égalité (I)
if m==n:
    print("les␣entiers␣sont␣égaux")

# Test de comparaison (II)
if m<=n:
    print("l'entier␣m␣est␣inférieur␣ou␣égal␣à␣n")
else:
    print("lentier␣m␣est␣strictement␣supérieur␣à␣n")


# Test d'égalité et comparaison (III)
if m==n:
    print("les␣entiers␣sont␣égaux")
elif m<=n:
    print("l'entier␣m␣est␣inférieur␣ou␣égal␣à␣n")
else:
    print("l'entier␣n␣est␣strictement␣inférieur␣à␣m")

# Tests successifs d'égalité et comparaison (IV)
if m==n:
    print("les␣entiers␣m␣sont␣égaux")
if m<=n:
    print("l'entier␣m␣est␣inférieur␣ou␣égal␣à␣m")
if m>n:
    print("l'entier␣nest␣strictement␣ingérieur␣à␣m")

# Si m=n, le bloc (II) ne produit qu'un affichage
# Si m=n, le bloc (III) produit deux affichages

# Le bloc (IV) parcourt et exécute chaque ligne
# Cette façon de programmer n'est pas optimale,
# notamment lorsqu'on teste beaucoup d'hypothèses...
# ... ou que chaque hypothèse est difficile à tester
# (ce qui est couteux en temps de calcul)
```

3 Boucles énumeratives (boucles for)

La syntaxe d'une boucle `for` est très simple :

```
for element in iterable:
        instructions
#suite
```

On a noté **iterable**, un objet itérable, c'est-à-dire un objet de type composé dont on peut énumérer les éléments qui le constituent : *iterable* peut donc être un n-uplet, une chaîne de caractères, une liste...

Pour chaque élément de **iterable**, la suite des instructions `instructions` est réalisée. L'indentation marque le bloc exécuté pour chaque élément de **iterable**. Une fois que tous les éléments de **iterable** ont été passés en revue, Python passe à la suite.

Exercice : Taper et exécuter les commandes suivantes. Constater

```
# Itérables et boucles énumératives


for x in "une chaine de caractères est itérable":
    print(x)

for x in [4,8,9]: # une liste aussi
    print(2*x)

# range(start,end,step) créé un intervalle entier.
for i in range(10):
    print(i)
# par défaut range(n) correspond à range(0,n,1)

# On peut sélectionner un entier sur 2 (ou 3, etc.)
for i in range(0,21,2):
    print(i)
```

```
# Deux exemples d'utilisation des boucles for

# Table de multiplication de n
n=int(input("Entrer un entier positif : "))
for i in range(1,11):
    print(n,"*",i,"=",n*i)

# Calcul de la factorielle de n
n=int(input('Entrer un entier naturel  : '))
A=1
for i in range(1,n+1):
    A=A*i
print('La factorielle de ',n,' vaut:',A)
```

Exercice 1

Dans tout l'exercice on travaille avec des entiers naturels et leur représentation en base 10.

1. Écrire les instructions demandant un entier naturel n à l'utilisateur et stockant les chiffres le composant dans une **chaîne de caractères**.
   Cet entier sera composé d'au moins 3 chiffres dans la suite.
2. Compléter les instructions précédentes afin d'obtenir un booléen indiquant si le premier et le dernier chiffre composant l'entier n sont égaux.
3. Écrire les instructions permettant d'obtenir un booléen indiquant si le nombre de chiffre composant l'entier n est pair.
4. Écrire les instructions permettant d'échanger le premier et le dernier chiffre de n.
5. Écrire les instructions permettant d'obtenir un booléen indiquant si le dernier chiffre composant l'entier n appartient à la liste composée des $d-1$ premiers chiffres de n (on a noté d le nombre de chiffres composant n).

Exercice 2

1. Écrire les instructions demandant à l'utilisateur une chaîne de caractère composée de 3 lettres.
2. Compléter les instructions précédentes pour obtenir un booléen indiquant si la chaîne entrée est un palindrome.
3. Recommencer avec un mot de 4 lettres.

4. Créer un troisième mot obtenu comme la concaténation des deux précédents : on ajoutera un espace entre les deux mots.
5. Comment retrouver à partir de ce dernier mot, les deux premiers ?

Exercice 3

Écrire un programme qui calcule l'aire de différentes figures : disque, triangle et rectangle.
Le programme attend le choix de l'utilisateur suivant le code suivant :
- 1 pour disque
- 2 pour triangle
- 3 pour rectangle

Puis selon le choix, le programme attend les informations suivantes :
- rayon
- base et hauteur
- longueur et largeur

Exercice 4

Écrire un programme qui donne les racines (éventuellement complexes) d'un trinôme à coefficients réels $aX^2 + bX + c$ entré par l'utilisateur.
Les coefficients réels du trinôme seront demandés à l'utilisateur sous forme d'un triplet (a, b, c). Le i complexe est noté `1j` en Python.

Exercice 5

Écrire un script qui demande un entier `n` à l'utilisateur et qui affiche la liste de tous les entiers pairs inférieurs ou égaux à `n`.

Exercice 6

Écrire un script qui demande un entier `n` à l'utilisateur et qui affiche la somme $\sum_{k=0}^{n} k$ des entiers inférieurs ou égaux à `n`.

Exercice 7

Écrire un script qui demande un entier $n \in \mathbb{N}^*$ et qui affiche la liste de tous les diviseurs impairs de n.

Remarques

On rappelle que `L.append(x)` ajoute l'élément `x` à la fin de la liste `L`.

Exercice 8

La suite de Fibonacci est définie par $F_0 = F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n$ pour tout entier $n \geqslant 0$.

1. Ecrire un programme demandant un entier $n \in \mathbb{N}$ à l'utilisateur et renvoyant le nombre F_n.
2. Écrire un programme demandant un entier $n \in \mathbb{N}$ à l'utilisateur et renvoyant la liste $[F_0, F_1, ..., F_n]$.
3. Déterminer à la main les racines du trinôme $X^2 - X - 1$ et stocker la plus grande racine dans une variable que l'on appellera `phi`.
4. Stocker les quotients $\frac{F_{n+1}}{F_n}$ dans une liste `L` pour $n \in [\![0, 49]\!]$.
5. Afficher les 10 derniers termes de cette liste. Conjecture ?