Types de données

In [ ]:

```
#Exécuter le code ci-dessous :
a,b,c,d,e = -42,3.14,'hello there', [2,4,6,8], 3<4
print(a,type(a))
print(b,type(b))
print(c,type(c))
print(d,type(d))
print(e, type(e))
```

Nous avons déjà manipulé des données de différents types : des nombres entiers ou réels, des chaînes de caractères et des listes ou encore des booléens. Examinons d'un peu plus près ces types de données.

1. Les données numériques

Le type integer (entier)

In [ ]:

```
n = 5
a,b,c = 3,2,1
while c < n :
    print( c , ": ", b)
    a , b = b , a * b
    c = c + 1
```

Exercice 1 :

1. Exécuter le code ci-dessus. Que fait ce programme ?
2. Exécuter le code avec $n = 15$, puis $n = 20$, puis $n = 30$, puis $n = 45$. Que constate-t-on ?

Réponses :

A retenir :

On peut considérer que Python est capable de traiter des entiers de taille illimitée, la limite est celle de la taille de la mémoire disponible dans l'ordinateur. Plus précisément, les entiers sont codés sur 32 bits lorsque leur taille le permet. Les très grands entiers occupent une place variable, en fonction de leur taille.

Le type float (nombre réel)

In [ ]:

```
n = 45
a,b,c = 3.0 , 2.0 , 1
while c < n :
    print( c , ": ", b)
    a , b = b , a * b
    c = c + 1
```

Exercice 2:

1. Qu'a-t-on modifié dans le code ci-dessus par rapport au programme précédent ?
2. Exécuter le code. Que remarque-t-on ?
3. Remplacer `a*b` par `a/b` . Que constate-t-on ?

Réponses :

A retenir :

Le type de valeur `float` permet le calcul sur de très grands nombres ou de très petits nombres avec un degré de précision constant. Mais ce degré de précision indique donc que l'on travaille parfois avec des valeurs approchées, ce qui peut poser des problèmes dans les programmes. Plus précisément, ces nombres sont codés sur 64 bits (8 octets) dans la mémoire de la machine : une partie du code correspond aux chiffres significatifs, l'autre à l'ordre de grandeur (puissance de 10)

2. Les chaînes de caractères

La plupart des langages de programmation sont aussi capables de manipuler des caractères alphabétiques, des mots, des phrases ou des suites de symboles quelconques. Il existe pour ce type de valeurs des structures de données appelés les chaînes de caractères, en anglais `string` .

Le type string

Dans un script python, on peut délimiter suite de caractères, soit par des apostrophes (simple quotes), soit par des guillemets (double quotes). Exemples :

In [ ]:

```
#Exécuter le code ci-dessous :
phrase1 = 'la musique.'
phrase2 = '"Oui", dit-il,'
phrase3 = "j'adore"

print(phrase2, phrase3, phrase1)
print(type(phrase1),type(phrase2),type(phrase3))
```

Remarques :

- On peut utiliser les apostrophes pour délimiter une chaîne qui contient des guillemets (phrase 2)
- On peut utiliser les guillemets pour délimiter une chaîne qui contient des apostrophes (phrase 3)

Accéder au caractères individuels

Python considère qu'une chaîne de caractères est une collection ordonnée d'éléments. Cela signifie que les caractères d'une chaîne sont toujours disposés dans un certain ordre. Par conséquent, chaque caractère de la chaîne peut être désigné par sa place dans la séquence, à l'aide d'un index. Pour accéder à un caractère bien déterminé, on utilise le nom de la variable qui contient la chaîne et on lui accole, entre deux crochets, l'index numérique qui correspond à la position du caractère dans la chaîne.

Exercice 3 :

In [ ]:

```
#Exécuter le code ci-dessous :
nom = 'Hendrix'
print(nom[1])
```

Modifier la deuxième ligne du programme ci-dessus pour afficher :

1. La lettre 'H'.
2. La lettre 'x'.

In [ ]:

```
# Réponses :
```

A retenir :

Les données informatiques sont presque toujours numérotées à partir de zéro (et non à partir de un) ! C'est le cas pour les caractères d'une chaîne.

Opérations élémentaires sur les chaînes.

Concaténation

Exercice 4 : Exécuter ce code. Que fait l'opérateur + ?

In [ ]:

```
a = 'MAE'
b = 'STRO'
c = a + b
print(c)
d = b + a
print(d)
```

Réponse :

Longueur d'une chaîne

Exercice 5 : Exécuter ce code. Que fait la fonction `Len` ?

In [ ]:

```
mot = 'anticonstitutionnellement'
print(len(mot))
```

Réponse :

Conversion

Exercice 6 : Exécuter le code ci-dessous :

In [ ]:

```
mdp = '1234'
# mdp = int(mdp)
print(10*mdp)
print(10*mdp+5)
```

Questions :

1. Expliquer le message obtenu.
2. Décommenter la troisième ligne. Que fait la fonction `int` ?

Réponses :

3. Les tableaux indexés

Le type list

Sous Python, on peut définir une liste comme une collection d'éléments séparés par une virgule, l'ensemble étant délimité par des crochets.

Exemples :

In [ ]:

```
#Exécuter le code ci-dessous :
semaine = ['Lun','Mar','Mer','Jeu','Ven']
jours = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,
30,31]
mois = ['Jan','Fév','Mars','Avr','Mai','Juin','Juil','Août','Sep','Oct','Nov','Déc']
temperatures = [-30.0,-20.0,-10.0,0.0,10.0,20.0,30.0,40.0]

ajd = [['Dim',21,'Juin'],20.0]

print (semaine, type(semaine))
print (ajd, type(ajd))
```

Les éléments individuels qui constituent une liste peuvent être de types variés :

- Les éléments des listes `semaine` et `mois` sont des chaînes de caractères.
- Les éléments de la liste `jours` sont des entiers.
- Les éléments de la liste `temperatures` contient des nombres de type 'float'.
- La liste `ajd` contient...2 éléments : une liste (elle-même composée de chaînes et d'un entier) et un nombre de type 'float'.

Similitudes avec les chaînes

Accès aux éléments :

Comme les chaînes de caractères, les listes sont des collections ordonnées d'objets. On peut donc accéder à chaques élément individuellement si l'on connaît son index dans la liste . Comme pour les chaînes, l'index commence à partir de zéro.

Exemples:

In [ ]:

```
#Exécuter le code ci-dessous :
print(semaine[0])
print(jours[30])
print(mois[2])
```

Exercice 7 :

Modifier le programme ci-dessus pour qu'il affiche la date d'aujourd'hui.

In [ ]:

```
# Réponse : à adapter en fonction de la date
```

Longueur d'une liste

La fonction `len` s'utilise aussi avec les listes. Exemples :

In [ ]:

```
print(len(jours)) #affiche la longueur de la liste 'jours', c'est à dire son nombre d'éléments
print(len(ajd)) # la liste 'ajd' contient 2 éléments
```

Concaténation

Comme pour les chaînes, on peut utiliser l'opérateur `+` pour rassembler 2 listes. Exemples :

In [ ]:

```
#Exécuter le code ci-dessous :
L1=[1,2,3]
L2=[4,5,6]
print(L1+L2)
print(L2+L1)
```

Différences avec les chaînes

Modifier des éléments

À la différence de ce qui se passe pour les chaînes, qui constituent un type de données non-modifiables, il est possible de modifier les éléments individuels d'une liste :

In [ ]:

```
# Executer le code ci-dessous
semaine[0]='Dim'
print(semaine)
```

Exercice 8 :

Annuler la modification précédente

In [ ]:

```
#Réponse:
```

Supprimer des éléments

La fonction `del` permet de supprimer d'une liste un élément quelconque, à partir de son index.

In [ ]:

```
del(semaine[4])
print(semaine)
```

Ajouter des éléments

On peut ajouter un élément à une liste. Pour cela, il faut considérer que la liste est un objet, dont on va utiliser l'une des méthodes, ici la méthode `append` .

In [ ]:

```
#Exécuter le code ci-dessous
semaine.append('Ven')
print(semaine)
```

Remarques :

- 'append' en anglais veut dire 'ajouter'.
- l'élément est ajouté à la fin de la liste
- on applique une méthode à un objet en reliant les deux à l'aide d'un point.

Exercice 9 :

Ajouter le week-end à la liste 'semaine' et l'afficher

In [ ]:

```
# Réponse :
```

4. Booléens

In [ ]:

```
#Exécuter le code ci-dessous
a = (2<3)
b = (2==3)
print (a, type(a))
print(b, type(b))
```

Lorsqu'un programme contient des instructions du type `while` ou `if` , l'ordinateur doit évaluer la véracité d'une condition. Est-elle vraie ou fausse ?

- `True` est renvoyé si la condition est vraie.
- `False` est renvoyé si la condition est fausse.

Ces valeurs sont appelées 'booléens'. Exemples :

In [ ]:

```
#Exécuter le code ci-dessous

#Tout ceci est vrai :
print(2 != 3)
print(2 == 2.0)
print(2 <= 1+1)
print('alpha' < 'alphabet')
print(2 in [0,2,4,6,8])
print(True == 1)
print(False == 0)
print(True + 1 == 2)

#Tout cela est faux :
print(-1 > 0)
print ((-4)**2 == -16)
print(1 in [0,2,4,6,8])
print(True == False)
print (0.1 + 0.1 + 0.1 == 0.3) # Eh oui...
print('Z' < 'A')
print(1 != 1)

# Ceci n'est pas evaluable
print('ma_note' < 20)
```

Remarques :

- Comme toutes les données, un booléen est une valeur numérique, True vaut 1 et False vaut 0
- Les chaînes de caractères peuvent être comparées entre elles.
- Il n'est pas possible d'évaluer des données de types différents.

5. Exercices

Exercice 10 :

In [ ]:

```
#Exécuter le code ci-dessous :
heures = 5.5
h = int(heures)
minutes = (heures -h)*60
mn = int(minutes)
print (h,'h',mn,'mn')
```

1. Expliquez ce fait ce programme
2. Quel est le type des données contenues dans :
   - `heures` ?
   - `minutes` ?
   - `mn` ?
3. Modifier ce programme pour qu'il affiche aussi les secondes et le vérifier en changeant la valeur de `heures` .

Réponses :

In [ ]:

```
# Réponse à la question 3
```

Exercice 11 :

En 2019, Kylian M., soucieux de son avenir, a placé 1000 € sur un compte qui lui rapporte chaque année, 4,5 % d'intérêts.

In [ ]:

```
# Exécuter le code ci-dessous
capital = 1000
taux = 4.5
annee = 2019

capital = capital + (taux/100)*capital
annee = annee + 1
print (annee ,':', capital,'€')
```

1. Expliquer ce que fait ce programme.
2. En utilisant une boucle `while` , modifier ce programme pour qu'il affiche la somme qui sera sur le compte en 2050
3. Modifier le programme pour qu'il affiche la somme disponible chaque année jusqu'en 2050

Réponses :

In [ ]:

```
#2.
```

In [ ]:

```
#3.
```

Exercice 12:

In [ ]:

```
# Exécuter le code ci-dessous
mot = 'super'
n = 0
for lettre in mot:
    if lettre == 'e':
        n = 1

if n == 0 :
    print('Pas de "e" dans', mot)
else :
    print('Il y a au moins un "e" dans',mot)
```

1. Expliquer comment fonctionne ce programme.
2. Le modifier pour qu'il affiche le nombre de 'e' dans la chaîne contenue dans la variable `mot` . Verifier en testant plusieurs mots.

Réponses :

In [ ]:

```
#2.
```

Exercice 13:

Compléter ce programme pour que la liste `pairs` contienne les entiers pairs de la liste `nombres` et pour que la liste `impairs` contienne les entiers impairs de la liste `nombres` .

In [ ]:

```
nombres = [2,4,8,1,3,5,7,9,6,0]
pairs = []
impairs = []




print(pairs)
print(impairs)
```

Réponse :

In [ ]:

```

```

Exercice 14

In [ ]:

```
# Exécuter le code ci-dessous
mot = 'devine'
inverse = ''
i = 0
while i < len(mot):
    inverse = inverse + mot[len(mot)-1-i]
    i = i + 1

print(inverse)
```

1. Que fait ce programme ?
2. le compléter pour qu'il détermine si le mot de départ est un palindrome, c'est à dire un mot qui peut se lire dans les deux sens (radar, kayak, s.o.s, bob...). Vérifier en testant plusieurs mots.

Réponse :

In [ ]:

```
#2.
```

Exercice 15 :

Écrivez un programme qui recherche le plus grand élément présent dans une liste donnée. Vérifier en modifiant les valeurs dans la liste.

In [ ]:

```
nombres = [-32, 5, 80, 8, 3, 75, 2, 15]
```

Réponse :

In [ ]:

```

```

Exercice 16 :

Ecrire un programme qui détermine si une année est bissextile ou non. Vérifier avec plusieurs années.

Attention : Une année A est bissextile si A est divisible par 4. Elle ne l'est cependant pas si A est un multiple de 100, à moins que A ne soit multiple de 400.

Réponse :

In [ ]:

```

```

FIN