ENSIAS

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

Langage Python

Lecture2: Types et structures de données

Abderrahim MESBAH
a.mesbah@um5r.ac.ma

Plan

- Types de données
- Nombres
- Séquences
- Dictionnaires
- Egalité et l'identité/Similitude

Types de données

Types de données

- **Python connaît différents types de données. Pour trouver le type d'une variable, utilisez la fonction type():**

```
In [5]: a = 45
In [6]: type (a)
Out[6]: int

In [7]: b = 'This is a string'
In [8]: type (b)
Out[8]: str

In [9]: c = 2 + 1j
In [10]: type (c)
Out[10]: complex

In [11]: d = [1, 3, 56]
In [12]: type (d)
Out[12]: list
```

Nombres

Nombres

- **Entiers**

  - Si nous devons convertir une chaîne contenant un nombre entier en un nombre entier, nous pouvons utiliser la fonction int() :

```
In [16]: a = '34 ' # a is a string containing the characters 3 and 4
In [17]: x = int(a) # x is in integer number
```

  - La fonction int() convertira également les nombres à virgule flottante en entiers :

```
In [18]: int (7.0)
Out[18]: 7

In [19]: int (7.9)
Out[19]: 7
```

Nombres

- **Entiers**

  - **int** tronquera toute partie non entière d'un nombre à virgule flottante. Pour **arrondir** un nombre de points à un entier, utilisez la commande **round (),** puis convertissez le **float** arrondi en un entier:

```
In [20]: round (7.9)
Out[20]: 8.0

In [21]: int(round (7.9))
Out[21]: 8
```

Nombres

- **Nombres à virgule flottante**
  - Une chaîne contenant un **nombre à virgule flottante** peut être convertie en un nombre à virgule flottante à l'aide de la commande **float ():**

```
In [27]: a = '35.342 '
In [28]: b = float (a)
In [29]: print b
35.342

In [30]: print type(b)
<type 'float'>
```

Nombres

- **Nombres complexes**

```
In [32]: x = (1+3j)
Out[32]: (1+3j)

In [33]: abs(x) # computes the absolute value
Out[33]: 3.1622776601683795

In [34]: x. imag
Out[34]: 3.0

In [35]: x. real
Out[35]: 1.0

In [36]: x * x
Out[36]: (-8+6j)

In [37]: x * x. conjugate ()
Out[37]: (10+0j)

In [38]: 3*x
Out[38]: (3+9j)
```

Nombres

- **Nombres complexes**

  - Notez que si vous souhaitez effectuer des opérations plus complexes (telles que la racine carrée, etc.), vous devez utiliser le module **cmath** (Complex Mathematics):

```
In [39]: import cmath

In [40]: cmath.sqrt(x)
Out[40]: (1.442615274452683+1.0397782600555705j)
```

ENSIAS

Séquences

Séquences

- **Séquence type 1: String**

  - Longueur d'une chaine de caractères: **Len()**

```
In [43]: a = "Hello Moon"
In [44]: len(a)
Out[44]: 10
```

  - Concaténation: **+**

```
In [45]: 'Hello ' + 'World '
Out[45]: 'Hello World '
```

Séquences

- **Séquence type 1: String**

  - Renvoie une chaîne en majuscule: **upper ()**

```
In [46]: a = "This is a test sentence."

In [47]: a.upper ()
Out[47]: 'THIS IS A TEST SENTENCE.'
```

  - Renvoie une chaîne en minuscule: **lower ()**

```
In [48]: b='HELLO WORLD'

In [49]: b.lower()
Out[49]: 'hello world'
```

Séquences

- **Séquence type 1: String**

  - Convertit une chaîne en une liste de chaînes: **split()**

```
In [50]: a = "This is a test sentence."

In [51]: a.split()
Out[51]: ['This', 'is', 'a', 'test', 'sentence.']
```

  - À base d'un séparateur: **split (** *séparateur***)**

```
In [52]: a = "The dog is hungry . The cat is bored . The snake is awake."

In [53]: a.split (".")
Out[53]: ['The dog is hungry ', ' The cat is bored ', ' The snake is awake', '']
```

Séquences

- **Séquence type 1: String**

  - Fusionnement des chaînes par un séparateur : **join()**

```
In [56]: a = "The dog is hungry . The cat is bored . The snake is awake."

In [57]: s=a.split(".")
In [58]: s
Out[58]: ['The dog is hungry ', ' The cat is bored ', ' The snake is awake', '']

In [59]: ". join (s)
Out[59]: 'The dog is hungry . The cat is bored . The snake is awake.'

In [60]: " STOP". join (s)
Out[60]: 'The dog is hungry STOP The cat is bored STOP The snake is awake STOP'
```

Séquences

- **Séquence type 2: List**
  - Une liste est une séquence d'objets. Les objets peuvent être de tout type:
    - Type entiers:

```
In [61]: a = [34 , 12, 54]
```

    - Type chaîne de caractères:

```
In [62]: a = ['dog ', 'cat ', 'mouse']
```

    - Liste vide:

```
In [63]: a=[]

In [64]: type (a)
Out[64]: list
```

    - Type mixte:

```
In [65]: a = [123 , 'duck ', -42, 17, 0, 'elephant ']
In [66]: b = [1, 4, 56, [5, 3, 1], 300 , 400]
```

Séquences

- **Séquence type 2: List**
  - Longueur d'une liste: **len()**

```
In [67]: a = ['dog ', 'cat ', 'mouse ']

In [68]: len(a)
Out[68]: 3
```

  - Concaténation: **+**

```
In [69]: [3, 4, 5] + [34 , 35, 100]
Out[69]: [3, 4, 5, 34, 35, 100]
```

Séquences

- **Séquence type 2: List**
  - Ajout d'un objet à la fin d'une liste: **append ()**

```
In [81]: a = [34 , 56, 23]
In [82]: a. append (42)
In [83]: print a
[34, 56, 23, 42]
```

  - Ajout des éléments d'une liste à une autre liste: **extend ()**

```
In [84]: a = [1,2,3]
In [85]: b=[4,5,6,7]
In [86]: a.extend(b)
In [87]: a
Out[87]: [1, 2, 3, 4, 5, 6, 7]
```

  - Suppression d'un élément de la liste: **remove ()**

```
In [88]: a = [34,56,23,42,56,44]
In [89]: a.remove(56)
In [90]: a
Out[90]: [34, 23, 42, 56, 44]
```

Séquences

- **Séquence type 2: List**
  - Génération d'une liste d'entiers: **range (n)**

```
In [92]: range (10)
Out[92]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [93]: range (3, 10)
Out[93]: [3, 4, 5, 6, 7, 8, 9]

In [94]: range (3, 10, 2)
Out[94]: [3, 5, 7, 9]

In [95]: range (10 , 0, -1)
Out[95]: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Séquences

- **Séquence type 3: Tuple**
  - Un tuple est une séquence (immuable) d'objets. Le comportement des tuples est très similaire à celui des listes, à la différence qu'ils ne peuvent pas être modifiés.

```
In [97]: a = (12 , 13, 'dog ')

In [98]: a
Out[98]: (12, 13, 'dog ')

In [99]: a [0]
Out[99]: 12

In [100]: a [0] = 1
Traceback (most recent call last):
          File "<ipython-input-100-9b8ae8b981bf>", line 1, in <module>
          a [0] = 1
TypeError: 'tuple' object does not support item assignment
```

Séquences

- **Indexation: a[i]**

```
In [101]: a = ['dog ', 'cat ', 'mouse ']
In [102]: a [0]
Out[102]: 'dog '

In [103]: a [-1]
Out[103]: 'mouse '


In [104]: a = "Hello World!"
In [105]: a [0]
Out[105]: 'H'

In [106]: a [-1]
Out[106]: '!'
```

Séquences

- **Découpage des séquences: a[i,j] / a[:i] / a[i:] / a[:]**

```
In [109]: a = "Hello World!"
In [110]: a [0:3]
Out[110]: 'Hel'

In [111]: a [6:-1]
Out[111]: 'World'

In [112]: a [:5]
Out[112]: 'Hello'

In [113]: a [5:]
Out[113]: ' World!'

In [114]: a [-2:]
Out[117]: 'd!'

In [115]: a [:]
Out[116]: 'Hello World!'
```

Dictionnaires

Dictionnaires

- **Dictionnaires: sont des structures de données flexibles et mutables qui permettent de stocker des paires clé-valeur**

```
In [118]: d = {}
In [119]: d['today '] = '22 deg C'
In [120]: d['yesterday '] = '19 deg C'

In [121]: d. keys ()
Out[121]: ['yesterday ', 'today ']

In [122]: d['today ']
Out[122]: '22 deg C'


In [123]: d2 = dict (a=1, b=2, c=3)
In [124]: d2
Out[124]: {'a': 1, 'b': 2, 'c': 3}

In [125]: d2['a']
Out[125]: 1
```

Dictionnaires

- **Dictionnaires:** fonctions
  - Récupération les valeurs d'un dictionnaire: **values ()**

```
In [133]: d=dict(today= '22 deg C',yesterday = '19 deg C')
In [134]: d
Out[134]: {'today': '22 deg C', 'yesterday': '19 deg C'}

In [135]: d.values()
Out[135]: ['19 deg C', '22 deg C']
```

  - Récupération les clés d'un dictionnaire: **keys ()**

```
In [136]: d.keys()
Out[136]: ['yesterday', 'today']
```

  - Récupération des paires clé/valeur dans un dictionnaire: **item()**

```
In [137]: d.items()
Out[137]: [('today', '22 deg C'), ('yesterday', '19 deg C')]
```

Dictionnaires

- **Dictionnaires:** fonctions
  - Vérification d'existence d'une clé dans un dictionnaire: **has_key ()**

```
In [137]: d. has_key ('today') # équivalent à 'today' in d
Out[137]: True

In [138]: d. has_key ('tomorrow')
Out[138]: False
```

  - Parcourir un dictionnaire:

```
In [140]: for temp in d.keys ():
                print d([temp])

19 deg C
22 deg C
```

Égalité et identité

Égalité et identité

- **Égalité**
  - Les opérateurs <,>, ==,> =, <= et! = Comparent les valeurs de deux objets. Les objets ne doivent pas nécessairement avoir le même type.

```
In [141]: a = 1.0; b = 1

In [142]: type (a)
Out[142]: float

In [143]: type (b)
Out[143]: int

In [144]: a == b
Out[144]: True
```

Égalité et identité

- **Identité / Similitude**
  - Pour vérifier si deux objets **a** et **b** sont **identiques** (c'est-à-dire que les références a et b sont **identiques au même endroit en mémoire**), nous pouvons utiliser l'opérateur **is:**

```
In [153]: a is b
Out[153]: False

In [154]: id(a)
Out[154]: 47403653032

In [155]: id(b)
Out[155]: 4299213848
```

Égalité et identité

- **Identité /Egalité**

```
In [156]: x = [0, 1, 2]
In [157]: y=x #affectation des références
In [158]: x==y
Out[158]: True

In [159]: id(x)
Out[159]: 47407233216

In [160]: id(y)
Out[160]: 47407233216
```

```
In [161]: x[0]=100 #modification la valeur de l'élément d'indice 0
In [162]: x
Out[162]: [100, 1, 2]

In [163]: y
Out[163]: [100, 1, 2] #le résultat est logique car x et y ont la même référence
```

Égalité et identité

- **Identité /Egalité**

```
In [164]: z = x [:] # affectation des valeurs
In [165]: z == x
Out[165]: True

In [166]: z is x
Out[166]: False

In [167]: id(z)
Out[167]: 4360360520

In [168]: id(x)
Out[168]: 47407233216

In [169]: x
Out[169]: [100, 1, 2]

In [170]: z
Out[170]: [100, 1, 2]
```