# Cours Info - 6

## Types composés

D.Malka

MPSI 2017-2018

# Sommaire

# Sommaire

# Les types composés

## Les types composés

Sont de type composé, les valeurs formées de valeurs de types plus simples.

Nous nous intéressons, ici, aux *séquences* que sont :

1. les n-uplets,
2. les chaînes de caractères,
3. les listes.

# Sommaire

# Construction des n-uplets

La construction d'un *n-uplets* correspond au produit cartésien en mathématiques :

$$(a, b, c) \in \mathbb{E}_{\not\in} \times \mathbb{E}_{\not\in} \times \mathbb{E}_{\not\in}$$

Construction :

CODE PYTHON

```
In [1]: t=(1,2,3,8)
In [2]: t2=('a','c','toto')
In [3]: t3=('nom','prenom', age)

In [4]: type(t)
Out[1]: tuple
```

# Stockage en mémoire

Soit un n-uplet :

CODE PYTHON

```
In [5]: t=(3,6,...,10,4)
```

Stockage en mémoire (première approche) :

| t | *0* | *1* | *i* | *n-2* | *n-1* |
|---|---|---|---|---|---|
| | 3 | 6 | ... | 10 | 4 |

Les composantes d'un n-uplet sont repérées par des indices **i commençant à 0.**

# Accès aux composantes

Accès à la composante d'indice *i* du n-uplet `t` : `t[i]`.

CODE PYTHON

```
In [6]: t=(1,2,3,8)
In [7]: t[2]
Out[2]: 3
```

# Immuabilité d'un n-uplet

Essayons de modifier la valeur d'une composante du n-uplet `t` :

CODE PYTHON

```
In [8]: t[2]=0

---------------------------------------------
TypeError        Traceback (most recent call last)

<ipython-input-25-8892b184d456> in <module>()

----> 1 t[2]=0

TypeError: 'tuple' object does not support item assignment
```

Les composantes d'un n-uplet ne sont pas modifiables.

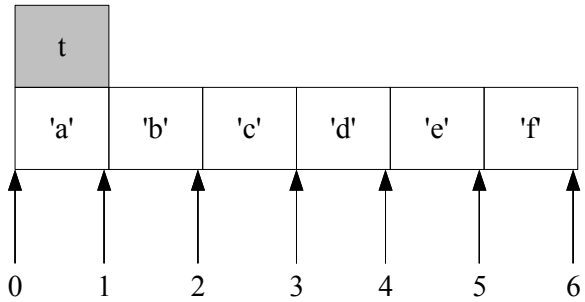
# Accès à une partie du n-uplet (slicing)

Slicing : `t[i:j]` renvoie un sous n-uplet de `t` constitués des composantes i à j(exclu)

CODE PYTHON

```
In [9]: t=('a','b','c','d','e','f)
In [10]: t[2:5]
Out[3]: ('c','d','e')
```

Représentation mentale du slicing :



# Concaténation

Opérateur de concaténation + : permet d'obtenir un n-uplet constitué de plusieurs n-uplets.

CODE PYTHON

```
In [11]: t=(1,2,3)
In [12]: t1=(3,2,1)
In [13]: t1+t
Out[4]: (3,2,1,1,2,3)
```

# Test d'appartenance

Comment savoir si un élément appartient à un n-uplet ? Opérateur `in`

CODE PYTHON

```
In [14]: t=(1,2,3,14)
In [15]: 3 in t
Out[5]: True

In [16]: 'Sabo' in t
Out[6]: False
```

# Longueur

Fonction `len(n-uplet)` :

CODE PYTHON

```
In [17]: t=(1,2,3,14,5)
In [18]: len(t)
Out[7]: 5

In [19]: type(len(t))
Out[8]: int
```

Renvoie le nombre d'éléments dans le n-uplet sous la forme d'un entier.

# Sommaire

# Les chaînes de caractères

Les chaînes de caractères ou (string) correspondent aux caractères, mots et phrases.

Codage binaire des symboles de base : table ASCII.

## ASCII TABLE

| Decimal | Hexadecimal | Binary | Octal | Char |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | [NULL] |
| 1 | 1 | 1 | 1 | [START OF HEADING] |
| 2 | 2 | 10 | 2 | [START OF TEXT] |
| 3 | 3 | 11 | 3 | [END OF TEXT] |
| 4 | 4 | 100 | 4 | [END OF TRANSMISSION] |
| 5 | 5 | 101 | 5 | [ENQUIRY] |
| 6 | 6 | 110 | 6 | [ACKNOWLEDGE] |
| 7 | 7 | 111 | 7 | [BELL] |
| 8 | 8 | 1000 | 10 | [BACKSPACE] |
| 9 | 9 | 1001 | 11 | [HORIZONTAL TAB] |
| 10 | A | 1010 | 12 | [LINE FEED] |
| 11 | B | 1011 | 13 | [VERTICAL TAB] |
| 12 | C | 1100 | 14 | [FORM FEED] |
| 13 | D | 1101 | 15 | [CARRIAGE RETURN] |
| 14 | E | 1110 | 16 | [SHIFT OUT] |
| 15 | F | 1111 | 17 | [SHIFT IN] |
| 16 | 10 | 10000 | 20 | [DATA LINK ESCAPE] |
| 17 | 11 | 10001 | 21 | [DEVICE CONTROL 1] |
| 18 | 12 | 10010 | 22 | [DEVICE CONTROL 2] |
| 19 | 13 | 10011 | 23 | [DEVICE CONTROL 3] |
| 20 | 14 | 10100 | 24 | [DEVICE CONTROL 4] |
| 21 | 15 | 10101 | 25 | [NEGATIVE ACKNOWLEDGE] |
| 22 | 16 | 10110 | 26 | [SYNCHRONOUS IDLE] |
| 23 | 17 | 10111 | 27 | [ENG OF TRANS. BLOCK] |
| 24 | 18 | 11000 | 30 | [CANCEL] |
| 25 | 19 | 11001 | 31 | [END OF MEDIUM] |
| 26 | 1A | 11010 | 32 | [SUBSTITUTE] |
| 27 | 1B | 11011 | 33 | [ESCAPE] |
| 28 | 1C | 11100 | 34 | [FILE SEPARATOR] |
| 29 | 1D | 11101 | 35 | [GROUP SEPARATOR] |
| 30 | 1E | 11110 | 36 | [RECORD SEPARATOR] |
| 31 | 1F | 11111 | 37 | [UNIT SEPARATOR] |
| 32 | 20 | 100000 | 40 | [SPACE] |
| 33 | 21 | 100001 | 41 | ! |
| 34 | 22 | 100010 | 42 | " |
| 35 | 23 | 100011 | 43 | # |
| 36 | 24 | 100100 | 44 | $ |
| 37 | 25 | 100101 | 45 | % |
| 38 | 26 | 100110 | 46 | & |
| 39 | 27 | 100111 | 47 | ' |
| 40 | 28 | 101000 | 50 | ( |
| 41 | 29 | 101001 | 51 | ) |
| 42 | 2A | 101010 | 52 | * |
| 43 | 2B | 101011 | 53 | + |
| 44 | 2C | 101100 | 54 | , |
| 45 | 2D | 101101 | 55 | - |
| 46 | 2E | 101110 | 56 | . |
| 47 | 2F | 101111 | 57 | / |
| 48 | 30 | 110000 | 60 | 0 |
| 49 | 31 | 110001 | 61 | 1 |
| 50 | 32 | 110010 | 62 | 2 |
| 51 | 33 | 110011 | 63 | 3 |
| 52 | 34 | 110100 | 64 | 4 |
| 53 | 35 | 110101 | 65 | 5 |
| 54 | 36 | 110110 | 66 | 6 |
| 55 | 37 | 110111 | 67 | 7 |
| 56 | 38 | 111000 | 70 | 8 |
| 57 | 39 | 111001 | 71 | 9 |
| 58 | 3A | 111010 | 72 | : |
| 59 | 3B | 111011 | 73 | ; |
| 60 | 3C | 111100 | 74 | < |
| 61 | 3D | 111101 | 75 | = |
| 62 | 3E | 111110 | 76 | > |
| 63 | 3F | 111111 | 77 | ? |
| 64 | 40 | 1000000 | 100 | @ |
| 65 | 41 | 1000001 | 101 | A |
| 66 | 42 | 1000010 | 102 | B |
| 67 | 43 | 1000011 | 103 | C |
| 68 | 44 | 1000100 | 104 | D |
| 69 | 45 | 1000101 | 105 | E |
| 70 | 46 | 1000110 | 106 | F |
| 71 | 47 | 1000111 | 107 | G |
| 72 | 48 | 1001000 | 110 | H |
| 73 | 49 | 1001001 | 111 | I |
| 74 | 4A | 1001010 | 112 | J |
| 75 | 4B | 1001011 | 113 | K |
| 76 | 4C | 1001100 | 114 | L |
| 77 | 4D | 1001101 | 115 | M |
| 78 | 4E | 1001110 | 116 | N |
| 79 | 4F | 1001111 | 117 | O |
| 80 | 50 | 1010000 | 120 | P |
| 81 | 51 | 1010001 | 121 | Q |
| 82 | 52 | 1010010 | 122 | R |
| 83 | 53 | 1010011 | 123 | S |
| 84 | 54 | 1010100 | 124 | T |
| 85 | 55 | 1010101 | 125 | U |
| 86 | 56 | 1010110 | 126 | V |
| 87 | 57 | 1010111 | 127 | W |
| 88 | 58 | 1011000 | 130 | X |
| 89 | 59 | 1011001 | 131 | Y |
| 90 | 5A | 1011010 | 132 | Z |
| 91 | 5B | 1011011 | 133 | [ |
| 92 | 5C | 1011100 | 134 | \ |
| 93 | 5D | 1011101 | 135 | ] |
| 94 | 5E | 1011110 | 136 | ^ |
| 95 | 5F | 1011111 | 137 | _ |
| 96 | 60 | 1100000 | 140 | ` |
| 97 | 61 | 1100001 | 141 | a |
| 98 | 62 | 1100010 | 142 | b |
| 99 | 63 | 1100011 | 143 | c |
| 100 | 64 | 1100100 | 144 | d |
| 101 | 65 | 1100101 | 145 | e |
| 102 | 66 | 1100110 | 146 | f |
| 103 | 67 | 1100111 | 147 | g |
| 104 | 68 | 1101000 | 150 | h |
| 105 | 69 | 1101001 | 151 | i |
| 106 | 6A | 1101010 | 152 | j |
| 107 | 6B | 1101011 | 153 | k |
| 108 | 6C | 1101100 | 154 | l |
| 109 | 6D | 1101101 | 155 | m |
| 110 | 6E | 1101110 | 156 | n |
| 111 | 6F | 1101111 | 157 | o |
| 112 | 70 | 1110000 | 160 | p |
| 113 | 71 | 1110001 | 161 | q |
| 114 | 72 | 1110010 | 162 | r |
| 115 | 73 | 1110011 | 163 | s |
| 116 | 74 | 1110100 | 164 | t |
| 117 | 75 | 1110101 | 165 | u |
| 118 | 76 | 1110110 | 166 | v |
| 119 | 77 | 1110111 | 167 | w |
| 120 | 78 | 1111000 | 170 | x |
| 121 | 79 | 1111001 | 171 | y |
| 122 | 7A | 1111010 | 172 | z |
| 123 | 7B | 1111011 | 173 | { |
| 124 | 7C | 1111100 | 174 | \| |
| 125 | 7D | 1111101 | 175 | } |
| 126 | 7E | 1111110 | 176 | ~ |
| 127 | 7F | 1111111 | 177 | [DEL] |

# Construction d'une chaine de caractères

Construction d'une chaîne de caractères :

CODE PYTHON

```
In [20]: mot="Bonjour"
In [21]: phrase='Hello Ortiz!'
In [22]: phrase
Out[9]: 'Hello Ortiz!'

In [23]: type(phrase)
Out[10]: str
```

# Stockage en mémoire (1ère approche)

CODE PYTHON

```
In [24]: bonjour="Bonjour"
```

Stockage en mémoire (première approche) :

| bonjour | *0* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|---|
| | "B" | "o" | "n" | "j" | "o" | "u" | "r" |

Les caractères sont repérés par des indices **i commençant à 0.**

# Accès à un caractère

Accès au caractère d'indice *i* de la chaîne `chaine` : `chaine[i]`.

CODE PYTHON

```
In [25]: salutation='Hello World!'
In [26]: salutation[0]
Out[11]: 'H'

In [27]: salutation[5]
Out[12]: ' '
```

# Accès à une partie de la chaîne de caractère

Slicing : `chaine[i:j]` renvoie une sous-chaîne de `chaine` constitués des caractères i à j(exclu).

CODE PYTHON

```
In [28]: salutation='Coucou!'
In [29]: salutation[0:4]
Out[13]: 'Couc'

In [30]:  salutation[0:8]
Out[14]: 'Coucou!'
```

Représentation mentale : voir n-uplet.

# Immuabilité

Essayons de modifier un caractère de la chaîne :

CODE PYTHON

```
In [31]: immuable='Je suis immuable!'
In [32]: immuable[0]='Z'

---------------------------------------------
TypeError       Traceback (most recent call last)

<ipython-input-38-efbee75cb102> in <module>()

----> 1 immuabilite[0]='Z'


TypeError: 'str' object does not support item assignment
```

Les chaînes de caractères ne sont pas modifiables.

# Concaténation

Concaténation de chaînes de caractères : l'opérateur + permet d'obtenir une chaîne de caractères constituée de plusieurs chaînes de caractères.

CODE PYTHON

```
In [33]: moit='Hello'
In [34]: _moit=' Nathan!'
In [35]: salutation = moit + _moit
In [36]: salutation
Out[15]: 'Hello Nathan!'
```

# Test d'appartenance

Rechercher si un caractère ou un mot est présent dans une chaine de caractère : opérateur `in`.

CODE PYTHON

```
In [37]: salut="Hello Léa"
In [38]: 'o' in salut
In [39]: True
In [40]: 'Lea' in salut
Out[16]: True

In [41]: 'lea' in salut # sensibilité à la casse!
Out[17]: False
```

# Longueur

Fonction `len(chaine_de_caracteres)` :

CODE PYTHON

```
In [42]: jubilation="Vive la MPSI de Saint Ex'!"
In [43]: len(jubilation)
Out[18]: 26
```

Renvoie le nombre de caractères dans la chaîne sous la forme d'un entier.

# Conversion vers types simples

Fonction `float`, `int` ... :

CODE PYTHON

```
In [44]: float('1.2')
Out[19]: 1.2

In [45]: int('3')
Out[20]: 3
```

# Sommaire

# Qu'est-ce qu'une liste ?

## Qu'est-ce qu'une liste ?

La liste est une séquence de valeurs **modifiable**.

Les éléments peuvent être de types différents, simples ou composés.

Nous nous restreignons, ici, à des listes dont les éléments sont de même type. Dans ce cas, on appellera *tableau*, cette structure de données.

# Construction

Construction d'une liste :

CODE PYTHON

```
In [46]: l1=[1,2,3,4]
In [47]: l2=['a','b','c','d']
In [48]: l1
Out[21]: [1,2,3,4]

In [49]: l2
Out[22]: ['a','b','c','d']

In [50]: type(l2)
Out[23]: list
```

# Stockage en mémoire

CODE PYTHON

```
In [51]: l1=[1,2,3,4]
```

Stockage en mémoire (première approche) :

| l1 | *0* | *1* | *2* | *3* |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |

Les composantes sont repérées par des indices **i commençant à 0.**

# Accès aux composantes

Accès à la composante d'indice *i* de la liste `l` : `l[i]`.

CODE PYTHON

```
In [52]: l1=[1,2,3,4]
In [53]: l1[3]
Out[24]: 4
```

# Modification de la valeur d'une composante

Comment modifier la valeur d'une composante ?

CODE PYTHON

```
In [54]: l1=[2,48,3,5,6]
In [55]: l1[2]=0
In [56]: l1
Out[25]: [2,48,0,5,6]
```

# Accès à une partie de la liste (slicing)

Comme pour n'importe quelle séquence : `l[i:j]` renvoie une liste constituées des composantes i à j(exclu).

CODE PYTHON

```
In [57]: l1=['b','o','n','j','o','u','r']
In [58]: l1[3:7]
Out[26]: ['j','o','u','r']
```

# Concaténation

Opérateur de concaténation : +

CODE PYTHON

```
In [59]: l1=['b','o','n']
In [60]: l2=['j','o','u','r']
In [61]: l3=l1+l
In [62]: l3
Out[27]: ['b','o','n','j','o','u','r']
```

# Concaténation

En mémoire :

| l1 | *0* | *1* | *2* |
|---|---|---|---|
| | "b" | "o" | "n" |

| l2 | *0* | *1* | *2* | *3* |
|---|---|---|---|---|
| | "j" | "o" | "u" | "r" |

| bonjour | *0* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|---|
| | "b" | "o" | "n" | "j" | "o" | "u" | "r" |

# Ajout d'un élément

Ajout en queue de liste : méthode `append(elt)`.

CODE PYTHON

```
In [63]: l1=[1,2,3]
In [64]: l1
Out[28]: [1,2,3]
```

| l1 | *0* | *1* | *2* |
|---|---|---|---|
| | 1 | 2 | 3 |

# Ajout d'un élément

CODE PYTHON

```
In [65]: l1.append(7)
In [66]: l1
Out[29]: [1,2,3,7]
```

| l1 | *0* | *1* | *2* | *3* |
|---|---|---|---|---|
| | 1 | 2 | 3 | 7 |

La liste contenue dans la variable `l1` a été modifiée.

Les listes sont des objets $\Rightarrow$ « méthodes ».
Notation pointée : `l.append(elt)`.

# Ajout d'un élément

Insertion : méthode `insert(index,elt)`.

CODE PYTHON

```
In [67]: l1=[1,2,3]
In [68]: l1
Out[30]: [1,2,3]
```

| l1 | *0* | *1* | *2* |
|---|---|---|---|
| | 1 | 2 | 3 |

# Ajout d'un élément

CODE PYTHON

```
In [69]: l1.insert(1,7)
In [70]: l1
Out[31]: [1,7,2,3]
```

| l1 | *0* | *1* | *2* | *3* |
|---|---|---|---|---|
| | 1 | 7 | 2 | 3 |

La liste contenue dans la variable `l1` a été modifiée : les listes sont des objets.

# Retrait d'un élément

Fonction `del(liste[indice])` :

CODE PYTHON

```
In [71]: l1=[1,2,3,20,15]
In [72]: l1
Out[32]: [1,2,3,20,15]
```

| l1 | *0* | *1* | *2* | *3* | *4* |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 20 | 15 |

# Retrait d'un élément

CODE PYTHON

```
In [73]: del(l1[3])
In [74]: l1
Out[33]: [1,2,3,15]
```

| l1 | *0* | *1* | *2* | *3* |
|---|---|---|---|---|
| | 1 | 2 | 3 | 15 |

La liste contenue dans la variable `l1` a été modifiée.

# Test d'appartenance

Opérateur `in` :

CODE PYTHON

```
In [75]: l1=[6,7,8,15]
In [76]: 8 in l1
Out[34]: True
```

# Longueur

Fonction `len(liste)` :

CODE PYTHON

```
In [77]: l1=[6,7,8,15]
In [78]: len(l1)
Out[35]: 4
```

# Sommaire

# Que contient la variable stockant une séquence ?

Quel est ce résultat du code suivant ?

CODE PYTHON

```
In [79]: l1=[6,7,8,15]
In [80]: l2=l1
In [81]: l2.append(104)
In [82]: print(l2)
In [83]: print(l1)
```

# Que contient la variable stockant une séquence ?

Réponse :

CODE PYTHON

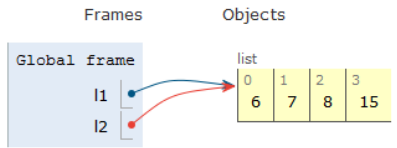
```
In [84]: l1=[6,7,8,15]
In [85]: l2=l1
In [86]: l2.append(104)
In [87]: l2
Out[36]: [6,7,8,15,104]

In [88]: l1
Out[37]: [6,7,8,15,104]
```

Pourquoi ce résultat surprenant ?

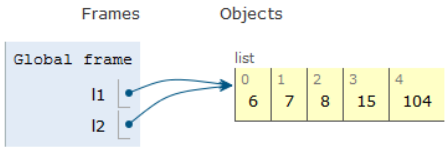
# Que contient la variable stockant une séquence ?

Tester le code sur : `http://www.pythontutor.com/`

```
1  l1=[6,7,8,15]
2  l2=l1
3  l2.append(104)
4
5  print(l1)
6  print(l2)
```




```
1  l1=[6,7,8,15]
2  l2=l1
3  l2.append(104)
4
5  print(l1)
6  print(l2)
```



# Que contient la variable stockant une séquence ?

Une séquence est un *Objet*.

Une variable ne contient pas l'objet mais la *référence* vers l'objet !

L'instruction `l1=l2` ne réalise pas une copie de l'objet mais une copie de la référence vers l'objet.

# Sommaire

# Conversion de types composés

Fonction `list(arg)`, `tuple(arg)`, `str(arg)`.

Exemples :

CODE PYTHON

```
In [89]: [6,7,8,15]
In [90]: tuple(l1)
Out[38]: 6,7,8,15)

In [91]: t1=(6,7,8,15)
In [92]: list(t1)
Out[39]: [6,7,8,15]

In [93]: list("bonjour")
Out[40]: ['b','o','n','j','o','u','r']
```