

Formation au langage de programmation Python

Partie II interpréteurs – types prédéfinis – exercices

Formateur : IBRAHIM M. S.



Global Knowledge®

du 30/05 au 02/06 2017

Chapitre : interpréteurs – types prédéfinis – exercices

- 1 Version installée/prioritaire
- 2 Python en mode interactif
- 3 Types numériques prédéfinis
- 4 Les chaînes de caractères : type string
- 5 Typage – conversion – évaluation
- 6 Mots-clés et mots réservés
- 7 Conteneurs : types composés
- 8 Opérations sur les conteneurs
- 9 Exercices
- 10 Résumé — questions

Identifier la version de Python installée

```
# python  
>>> print " La configuration installée est opérationnelle."
```

Identifier la version de Python installée

```
# python  
>>> print " La configuration installée est opérationnelle."
```

Messages d'erreur possibles

- 1 Python non trouvé : il faut alors mettre à jour la variable système `$PATH`

Identifier la version de Python installée

```
# python  
>>> print " La configuration installée est opérationnelle."
```

Messages d'erreur possibles

- 1 Python non trouvé : il faut alors mettre à jour la variable système `$PATH`
- 2 Code non interprété (parenthèses) – c'est normal ! bonne version installée

Identifier la version de Python installée

```
# python  
>>> print " La configuration installée est opérationnelle."
```

Messages d'erreur possibles

- 1 Python non trouvé : il faut alors mettre à jour la variable système `$PATH`
- 2 Code non interprété (parenthèses) – c'est normal ! bonne version installée
- 3 pas de message d'erreur : Python 2.7.xy installé ou prioritaire

Identifier la version de Python installée

```
# python  
>>> print " La configuration installée est opérationnelle."
```

Messages d'erreur possibles

- 1 Python non trouvé : il faut alors mettre à jour la variable système `$PATH`
- 2 Code non interprété (parenthèses) – c'est normal ! bonne version installée
- 3 pas de message d'erreur : Python 2.7.xy installé ou prioritaire

La version installée ou prioritaire est :

```
# python  
>>> from sys import version as v ;print(v)  
3.6.0 |Anaconda 4.3.0 (x86_64)| (default, Dec 23 2016, 13 :19 :00)  
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)]
```

Identifier la version de Python installée

```
# python  
>>> print " La configuration installée est opérationnelle."
```

Messages d'erreur possibles

- 1 Python non trouvé : il faut alors mettre à jour la variable système `$PATH`
- 2 Code non interprété (parenthèses) – c'est normal ! bonne version installée
- 3 pas de message d'erreur : Python 2.7.xy installé ou prioritaire

La version installée ou prioritaire est :

```
# python  
>>> from sys import version as v ;print(v)  
3.6.0 |Anaconda 4.3.0 (x86_64)| (default, Dec 23 2016, 13 :19 :00)  
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)]
```

- dans notre cas, c'est bien une version 3.x.y qui est exécutée

Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python

Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes

Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat

Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

 instructions sur 1 ligne



Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences

Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

- `;` instructions sur 1 ligne
- `,` constructeur de séquences
- `:` syntaxe `for if while def class`






Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences
-  syntaxe **for if while def class**
-   **blocs du programme**





Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences
-  syntaxe **for if while def class**
-  blocs du programme

Indentation





Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences
-  syntaxe **for if while def class**
-  blocs du programme

Indentation

- **délimiteur de fin de bloc**





Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences
-  syntaxe **for if while def class**
-  blocs du programme

Indentation

- délimiteur de fin de bloc
- **constitutif du langage**





Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences
-  syntaxe **for if while def class**
-  blocs du programme

Indentation

- délimiteur de fin de bloc
- constitutif du langage
- **structure le programme**





Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences
-  syntaxe **for if while def class**
-  blocs du programme

Indentation

- délimiteur de fin de bloc
- constitutif du langage
- structure le programme
- **nombre d'espaces fixe**





Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences
-  syntaxe **for if while def class**
-  blocs du programme

Indentation

- délimiteur de fin de bloc
- constitutif du langage
- structure le programme
- nombre d'espaces fixe
- **défini les scopes (des variables)**





Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences
-  syntaxe **for if while def class**
-  blocs du programme

Indentation

- délimiteur de fin de bloc
- constitutif du langage
- structure le programme
- nombre d'espaces fixe
- définit les scopes (des variables)
- **source d'erreurs syntaxiques**





Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences
-  syntaxe **for if while def class**
-  blocs du programme

Indentation

- délimiteur de fin de bloc
- constitutif du langage
- structure le programme
- nombre d'espaces fixe
- définit les scopes (des variables)
- source d'erreurs syntaxiques
- **source d'erreurs de logiques**





Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences
-  syntaxe **for if while def class**
-  blocs du programme

Indentation

- délimiteur de fin de bloc
- constitutif du langage
- structure le programme
- nombre d'espaces fixe
- définit les scopes (des variables)
- source d'erreurs syntaxiques
- source d'erreurs de logiques
- **aide de l'éditeur de texte**






Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences
-  syntaxe **for if while def class**
-   blocs du programme

Indentation

- délimiteur de fin de bloc
- constitutif du langage
- structure le programme
- nombre d'espaces fixe
- définit les scopes (des variables)
- source d'erreurs syntaxiques
- source d'erreurs de logiques
- aide de l'éditeur de texte
- **facilite (re)-lecture de code**






Python en mode interactif : interpréteurs

entre autres : python, iPython, bPython, notebook, idle, Jupiter

Boucle d'interaction

- on lance python
- on saisi les commandes
- on récupère le résultat
- on dispose d'un historique

Séparateurs

-  instructions sur 1 ligne
-  constructeur de séquences
-  syntaxe **for if while def class**
-   blocs du programme

Indentation

- délimiteur de fin de bloc
- constitutif du langage
- structure le programme
- nombre d'espaces fixe
- définit les scopes (des variables)
- source d'erreurs syntaxiques
- source d'erreurs de logiques
- aide de l'éditeur de texte
- facilite (re)-lecture de code
- **choisir : tabulations ou espaces**

Types de nombres prédéfinis

Entiers longs : int

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- `x**y` puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Flottants : float

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- `x**y` puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Flottants : float

- approximation des nombres réels

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- `x**y` puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Flottants : float

- approximation des nombres réels
- conversions implicites

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$
- $a**b = e^{b \cdot \ln a}$

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$
- $a**b = e^{b \cdot \ln a}$

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Complexes : complex

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$
- $a**b = e^{b \cdot \ln a}$

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Complexes : complex

- `a = complex(x,y)`

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$
- $a**b = e^{b \cdot \ln a}$

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Complexes : complex

- `a = complex(x,y)`
- `a = x + y * 1j`

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$
- $a**b = e^{b \cdot \ln a}$

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Complexes : complex

- `a = complex(x,y)`
- `a = x + y * 1j`
- `a.real`, `a.imag`, `abs(a)`

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$
- $a**b = e^{b \cdot \ln a}$

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Complexes : complex

- `a = complex(x,y)`
- `a = x + y * 1j`
- `a.real`, `a.imag`, `abs(a)`
- `a.conjugate()`

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$
- $a**b = e^{b \cdot \ln a}$

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Complexes : complex

- `a = complex(x,y)`
- `a = x + y * 1j`
- `a.real`, `a.imag`, `abs(a)`
- `a.conjugate()`

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$
- $a**b = e^{b \cdot \ln a}$

Booléens : bool

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Complexes : complex

- `a = complex(x,y)`
- `a = x + y * 1j`
- `a.real`, `a.imag`, `abs(a)`
- `a.conjugate()`

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$
- $a**b = e^{b \cdot \ln a}$

Booléens : bool

- `True` / `False`

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Complexes : complex

- `a = complex(x,y)`
- `a = x + y * 1j`
- `a.real`, `a.imag`, `abs(a)`
- `a.conjugate()`

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$
- $a**b = e^{b \cdot \ln a}$

Booléens : bool

- `True` / `False`
- `b == a`, `a != b`, `not c`

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- `x**y` puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Complexes : complex

- `a = complex(x,y)`
- `a = x + y * 1j`
- `a.real`, `a.imag`, `abs(a)`
- `a.conjugate()`

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$
- $a**b = e^{b \cdot \ln a}$

Booléens : bool

- `True` / `False`
- `b == a`, `a != b`, `not c`
- `1 < 4`, `z is s`

Types de nombres prédéfinis

Entiers longs : int

- pas de limite de taille
- $x**y$ puissance
- `//` quotient, `%` reste
- `divmod(a,b) -> (q,r)`

Complexes : complex

- `a = complex(x,y)`
- `a = x + y * 1j`
- `a.real`, `a.imag`, `abs(a)`
- `a.conjugate()`

Flottants : float

- approximation des nombres réels
- conversions implicites
- $a/b \neq a//b$
- $a**b = e^{b \cdot \ln a}$

Booléens : bool

- `True` / `False`
- `b == a`, `a != b`, `not c`
- `1 < 4`, `z is s`
- `x in R`, `a is not c`

Le type string : chaîne de caractères

Chaînes de caractères : string

Le type string : chaîne de caractères

Chaînes de caractères : string

- 'caracteres' ou "caracteres"

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0], w[-1], w[7 :11]`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

- `w.upper()`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

- `w.upper()`
- `w.lower()`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

- `w.upper()`
- `w.lower()`
- `w.isdigit()`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

- `w.upper()`
- `w.lower()`
- `w.isdigit()`
- `w.isalpha()`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

- `w.upper()`
- `w.lower()`
- `w.isdigit()`
- `w.isalpha()`
- `w.strip(" ,'-?!")`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

- `w.upper()`
- `w.lower()`
- `w.isdigit()`
- `w.isalpha()`
- `w.strip(" ,'-?!")`
- `w.count("e")`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

- `w.upper()`
- `w.lower()`
- `w.isdigit()`
- `w.isalpha()`
- `w.strip(" ,'-?!")`
- `w.count("e")`
- `w.split(" ")`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

- `w.upper()`
- `w.lower()`
- `w.isdigit()`
- `w.isalpha()`
- `w.strip(" ,'-?!")`
- `w.count("e")`
- `w.split(" ")`
- `w.format()`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

- `w.upper()`
- `w.lower()`
- `w.isdigit()`
- `w.isalpha()`
- `w.strip(" ,'-?!")`
- `w.count("e")`
- `w.split(" ")`
- `w.format()`
- `w.startswith("a")`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

- `w.upper()`
- `w.lower()`
- `w.isdigit()`
- `w.isalpha()`
- `w.strip(" ,'-?!")`
- `w.count("e")`
- `w.split(" ")`
- `w.format()`
- `w.startswith("a")`
- `w.endswith("a")`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

- `w.upper()`
- `w.lower()`
- `w.isdigit()`
- `w.isalpha()`
- `w.strip(" ,'-?!")`
- `w.count("e")`
- `w.split(" ")`
- `w.format()`
- `w.startswith("a")`
- `w.endswith("a")`
- `w.join('adding words')`

Le type string : chaîne de caractères

Chaînes de caractères : string

- `'caracteres'` ou `"caracteres"`
- `w = "ceci n'est pas un mot"`
- `nb = len(w)`
- `B = 'non, '`
- `w is B`
- `B+w` (concaténation)
- `w[0]`, `w[-1]`, `w[7 :11]`
- `w[9 :-2]`, `w[-2 :3]`, `w[:-1]`
- `w[-9 :-2]`, `w[4 :2]`
- `for i in w : print(i)`
- pas de caractère simple (char)
- Unicode pris en charge

Opérations possibles

- `w.upper()`
- `w.lower()`
- `w.isdigit()`
- `w.isalpha()`
- `w.strip(" ,'-?!")`
- `w.count("e")`
- `w.split(" ")`
- `w.format()`
- `w.startswith("a")`
- `w.endswith("a")`
- `w.join('adding words')`
- ...

Typage dynamique faible

Typage dynamique faible

- pas de déclaration explicite du type

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Type d'une variable

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Type d'une variable

- `type(variable)`

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Type d'une variable

- `type(variable)`

Conversion – saisie au clavier

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Type d'une variable

- `type(variable)`

Conversion – saisie au clavier

- `float()` – `int()` – `str()`

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Type d'une variable

- `type(variable)`

Conversion – saisie au clavier

- `float()` – `int()` – `str()`
- `x = int(input("x=(entier)"))`

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Type d'une variable

- `type(variable)`

Conversion – saisie au clavier

- `float()` – `int()` – `str()`
- `x = int(input("x=(entier)"))`

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Opérations disponibles

Type d'une variable

- `type(variable)`

Conversion – saisie au clavier

- `float()` – `int()` – `str()`
- `x = int(input("x=(entier)"))`

Typage – conversion – évaluation

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Opérations disponibles

- `dir(variable)`

Type d'une variable

- `type(variable)`

Conversion – saisie au clavier

- `float()` – `int()` – `str()`
- `x = int(input("x=(entier)"))`

Typage – conversion – évaluation

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Opérations disponibles

- `dir(variable)`
- `dir(type)`

Type d'une variable

- `type(variable)`

Conversion – saisie au clavier

- `float()` – `int()` – `str()`
- `x = int(input("x=(entier)"))`

Typage – conversion – évaluation

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Type d'une variable

- `type(variable)`

Conversion – saisie au clavier

- `float()` – `int()` – `str()`
- `x = int(input("x=(entier)"))`

Opérations disponibles

- `dir(variable)`
- `dir(type)`

eval

- `eval("string")`

Typage – conversion – évaluation

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Type d'une variable

- `type(variable)`

Conversion – saisie au clavier

- `float()` – `int()` – `str()`
- `x = int(input("x=(entier)"))`

Opérations disponibles

- `dir(variable)`
- `dir(type)`

eval

- `eval("string")`

exec

- `exec("string")`

Typage – conversion – évaluation

Typage dynamique faible

- pas de déclaration explicite du type
- l'interpréteur déduit type approprié
- celui-ci est mis à jour si besoin
- pas forcément le plus approprié

Type d'une variable

- `type(variable)`

Conversion – saisie au clavier

- `float()` – `int()` – `str()`
- `x = int(input("x=(entier)"))`

Opérations disponibles

- `dir(variable)`
- `dir(type)`

eval

- `eval("string")`

exec

- `exec("string")`

Un type spécial

- `None`

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- **simulation de l'interpréteur**

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- simulation de l'interpréteur

Cas d'utilisation

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- simulation de l'interpréteur

Cas d'utilisation

- validation de formulaire

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- simulation de l'interpréteur

Cas d'utilisation

- validation de formulaire
- **mini-interpréteur en ligne**

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- simulation de l'interpréteur

Cas d'utilisation

- validation de formulaire
- mini-interpréteur en ligne

Exemple

```
y = eval('input("dict or list")')  
print(type(y)) <class 'dict'>  
print(y) {'aa' : 123, 12 : [1, 3, 5]}
```

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- simulation de l'interpréteur

Cas d'utilisation

- validation de formulaire
- mini-interpréteur en ligne

Exemple

```
y = eval('input("dict or list")')  
print(type(y)) <class 'dict'>  
print(y) {'aa' : 123, 12 : [1, 3, 5]}
```

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- simulation de l'interpréteur

La fonction exec

- **execute le contenu d'un string**

Cas d'utilisation

- validation de formulaire
- mini-interpréteur en ligne

Exemple

```
y = eval('input("dict or list")')
print(type(y)) <class 'dict'>
print(y) {'aa' : 123, 12 : [1, 3, 5]}
```

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- simulation de l'interpréteur

La fonction exec

- exécute le contenu d'un string
- **expression définition déclaration**

Cas d'utilisation

- validation de formulaire
- mini-interpréteur en ligne

Exemple

```
y = eval('input("dict or list")')
print(type(y)) <class 'dict'>
print(y) {'aa' : 123, 12 : [1, 3, 5]}
```

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- simulation de l'interpréteur

La fonction exec

- exécute le contenu d'un string
- expression définition déclaration
- **simulation d'un script**

Cas d'utilisation

- validation de formulaire
- mini-interpréteur en ligne

Exemple

```
y = eval('input("dict or list")')
print(type(y)) <class 'dict'>
print(y) {'aa' : 123, 12 : [1, 3, 5]}
```


Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- simulation de l'interpréteur

La fonction exec

- exécute le contenu d'un string
- expression définition déclaration
- simulation d'un script

Cas d'utilisation

- validation de formulaire
- mini-interpréteur en ligne

Cas d'utilisation

Exemple

```
y = eval('input("dict or list")')  
print(type(y)) <class 'dict'>  
print(y) {'aa' : 123, 12 : [1, 3, 5]}
```

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- simulation de l'interpréteur

La fonction exec

- exécute le contenu d'un string
- expression définition déclaration
- simulation d'un script

Cas d'utilisation

- validation de formulaire
- mini-interpréteur en ligne

Cas d'utilisation

- scripting coté client

Exemple

```
y = eval('input("dict or list")')  
print(type(y)) <class 'dict'>  
print(y) {'aa' : 123, 12 : [1, 3, 5]}
```

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- simulation de l'interpréteur

La fonction exec

- exécute le contenu d'un string
- expression définition déclaration
- simulation d'un script

Cas d'utilisation

- validation de formulaire
- mini-interpréteur en ligne

Cas d'utilisation

- scripting coté client
- **interpréteur dans un IDE**

Exemple

```
y = eval('input("dict or list")')
print(type(y)) <class 'dict'>
print(y) {'aa' : 123, 12 : [1, 3, 5]}
```

Interprétation du contenu d'une chaîne de caractères

La fonction eval

- évaluer le contenu d'un string
- pas de définition ou déclaration
- simulation de l'interpréteur

La fonction exec

- exécute le contenu d'un string
- expression définition déclaration
- simulation d'un script

Cas d'utilisation

- validation de formulaire
- mini-interpréteur en ligne

Cas d'utilisation

- scripting coté client
- interpréteur dans un IDE

Exemple

```
y = eval('input("dict or list")')
print(type(y)) <class 'dict'>
print(y) {'aa' : 123, 12 : [1, 3, 5]}
```

Exemple

```
y = exec('input("dict or list")')
print(type(y)) <class 'dict'>
print(y) {'aa' : 123, 12 : [1, 3, 5]}
```

Attention ! Seuls les mots clés ne peuvent être redéfinis !

- éviter tout de même de redéfinir le reste : problèmes certains

Mots-clés et mots réservés — commentaires

Attention ! Seuls les mots clés ne peuvent être redéfinis !

- éviter tout de même de redéfinir le reste : problèmes certains

Mots clés : eux seuls sont essentiels

and assert break class continue def
del elif else except exec finally for
from global if import in is lambda
not or pass print raise return try
while yield
True False None

Attention ! Seuls les mots clés ne peuvent être redéfinis !

- éviter tout de même de redéfinir le reste : problèmes certains

Mots clés : eux seuls sont essentiels

and assert break class continue def
del elif else except exec finally for
from global if import in is lambda
not or pass print raise return try
while yield
True False None

Fonctions

help() dir() print() input()
raw_input() len() range() ord()
locals() globals() str() int()

Mots-clés et mots réservés — commentaires

Attention !

Seuls les mots clés ne peuvent être redéfinis !

- éviter tout de même de redéfinir le reste : problèmes certains

Mots clés : eux seuls sont essentiels

and assert break class continue def
del elif else except exec finally for
from global if import in is lambda
not or pass print raise return try
while yield
True False None

Fonctions

help() dir() print() input()
raw_input() len() range() ord()
locals() globals() str() int()

Modules : importés avant tout import

anydbm array atexit bisect calendar cmath codecs collections
commands ConfigParser copy ctypes datetime decimal dummy_thread
dummy_threading exceptions encodings.aliases formatter heapq
gettext locale linecache marshal math mmap operator os pickle
Queue re shelve shutil signal stat string StringIO struct subprocess sys
textwrap tempfile thread threading time timeit traceback unicodedata
xml.sax warnings whichdb _winreg

Mots-clés et mots réservés — commentaires

Attention !

Seuls les mots clés ne peuvent être redéfinis !

- éviter tout de même de redéfinir le reste : problèmes certains

Mots clés : eux seuls sont essentiels

and assert break class continue def
del elif else except exec finally for
from global if import in is lambda
not or pass print raise return try
while yield
True False None

Fonctions

help() dir() print() input()
raw_input() len() range() ord()
locals() globals() str() int()

Modules : importés avant tout import

anydbm array atexit bisect calendar cmath codecs collections
commands ConfigParser copy ctypes datetime decimal dummy_thread
dummy_threading exceptions encodings.aliases formatter heapq
gettext locale linecache marshal math mmap operator os pickle
Queue re shelve shutil signal stat string StringIO struct subprocess sys
textwrap tempfile thread threading time timeit traceback unicodedata
xml.sax warnings whichdb _winreg

Commentaires

Mots-clés et mots réservés — commentaires

Attention !

Seuls les mots clés ne peuvent être redéfinis !

- éviter tout de même de redéfinir le reste : problèmes certains

Mots clés : eux seuls sont essentiels

and assert break class continue def
del elif else except exec finally for
from global if import in is lambda
not or pass print raise return try
while yield
True False None

Fonctions

help() dir() print() input()
raw_input() len() range() ord()
locals() globals() str() int()

Modules : importés avant tout import

anydbm array atexit bisect calendar cmath codecs collections
commands ConfigParser copy ctypes datetime decimal dummy_thread
dummy_threading exceptions encodings.aliases formatter heapq
gettext locale linecache marshal math mmap operator os pickle
Queue re shelve shutil signal stat string StringIO struct subprocess sys
textwrap tempfile thread threading time timeit traceback unicodedata
xml.sax warnings whichdb _winreg

Commentaires

- # court : fin de ligne

Mots-clés et mots réservés — commentaires

Attention !

Seuls les mots clés ne peuvent être redéfinis !

- éviter tout de même de redéfinir le reste : problèmes certains

Mots clés : eux seuls sont essentiels

and assert break class continue def
del elif else except exec finally for
from global if import in is lambda
not or pass print raise return try
while yield
True False None

Fonctions

help() dir() print() input()
raw_input() len() range() ord()
locals() globals() str() int()

Modules : importés avant tout import

anydbm array atexit bisect calendar cmath codecs collections
commands ConfigParser copy ctypes datetime decimal dummy_thread
dummy_threading exceptions encodings.aliases formatter heapq
gettext locale linecache marshal math mmap operator os pickle
Queue re shelve shutil signal stat string StringIO struct subprocess sys
textwrap tempfile thread threading time timeit traceback unicodedata
xml.sax warnings whichdb _winreg

Commentaires

- # court : fin de ligne
- """ sur plusieurs lignes """

Mots-clés et mots réservés — commentaires

Attention !

Seuls les mots clés ne peuvent être redéfinis !

- éviter tout de même de redéfinir le reste : problèmes certains

Mots clés : eux seuls sont essentiels

and assert break class continue def
del elif else except exec finally for
from global if import in is lambda
not or pass print raise return try
while yield
True False None

Fonctions

help() dir() print() input()
raw_input() len() range() ord()
locals() globals() str() int()

Modules : importés avant tout import

anydbm array atexit bisect calendar cmath codecs collections
commands ConfigParser copy ctypes datetime decimal dummy_thread
dummy_threading exceptions encodings.aliases formatter heapq
gettext locale linecache marshal math mmap operator os pickle
Queue re shelve shutil signal stat string StringIO struct subprocess sys
textwrap tempfile thread threading time timeit traceback unicodedata
xml.sax warnings whichdb _winreg

Commentaires

- # court : fin de ligne
- """ sur plusieurs lignes """
- """ docstrings """

Mots-clés et mots réservés — commentaires

Attention !

Seuls les mots clés ne peuvent être redéfinis !

- éviter tout de même de redéfinir le reste : problèmes certains

Mots clés : eux seuls sont essentiels

and assert break class continue def
del elif else except exec finally for
from global if import in is lambda
not or pass print raise return try
while yield
True False None

Fonctions

help() dir() print() input()
raw_input() len() range() ord()
locals() globals() str() int()

Modules : importés avant tout import

anydbm array atexit bisect calendar cmath codecs collections
commands ConfigParser copy ctypes datetime decimal dummy_thread
dummy_threading exceptions encodings.aliases formatter heapq
gettext locale linecache marshal math mmap operator os pickle
Queue re shelve shutil signal stat string StringIO struct subprocess sys
textwrap tempfile thread threading time timeit traceback unicodedata
xml.sax warnings whichdb _winreg

Commentaires

- # court : fin de ligne
- """ sur plusieurs lignes """
- """ docstrings """
- **import proj ; proj.__doc__**

Conteneurs : types composés prédéfinis

Tuple

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- **immuable c-à-d non modifiable**

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Liste

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Liste

- éléments de types quelconques

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression
- accès par l'index : son rang

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Ensemble : set

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression
- accès par l'index : son rang

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Ensemble : set

- éléments de types quelconques

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression
- accès par l'index : son rang

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Ensemble : set

- éléments de types quelconques
- **non ordonnés, sans répétition**

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression
- accès par l'index : son rang

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Ensemble : set

- éléments de types quelconques
- non ordonnés, sans répétition
- entre `{}` — séparateur `,`

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression
- accès par l'index : son rang

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Ensemble : set

- éléments de types quelconques
- non ordonnés, sans répétition
- entre `{}` — séparateur `,`
- singleton `{a}` et vide `{ }`

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression
- accès par l'index : son rang

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Ensemble : set

- éléments de types quelconques
- non ordonnés, sans répétition
- entre `{}` — séparateur `,`
- singleton `{a}` et vide `{}`
- fonction `set()`

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression
- accès par l'index : son rang

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Ensemble : set

- éléments de types quelconques
- non ordonnés, sans répétition
- entre `{}` — séparateur `,`
- singleton `{a}` et vide `{}`
- fonction `set()`

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression
- accès par l'index : son rang

Dictionnaire

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Ensemble : set

- éléments de types quelconques
- non ordonnés, sans répétition
- entre `{}` — séparateur `,`
- singleton `{a}` et vide `{}`
- fonction `set()`

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression
- accès par l'index : son rang

Dictionnaire

- **structure associative non ordonnée : `key` → `value`**

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Ensemble : set

- éléments de types quelconques
- non ordonnés, sans répétition
- entre `{}` — séparateur `,`
- singleton `{a}` et vide `{}`
- fonction `set()`

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression
- accès par l'index : son rang

Dictionnaire

- structure associative non ordonnée : `key → value`
- `{k1 : v1, k2 : v2, k3 : v3, ...}`

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Ensemble : set

- éléments de types quelconques
- non ordonnés, sans répétition
- entre `{}` — séparateur `,`
- singleton `{a}` et vide `{}`
- fonction `set()`

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression
- accès par l'index : son rang

Dictionnaire

- structure associative non ordonnée : `key → value`
- `{k1 : v1, k2 : v2, k3 : v3, ...}`
- accès par la clé

Conteneurs : types composés prédéfinis

Tuple

- éléments de types quelconques
- entre `()` — séparateur `,`
- singleton `(a,)` et vide `()`
- immuable c-à-d non modifiable
- accès par l'index : son rang

Ensemble : set

- éléments de types quelconques
- non ordonnés, sans répétition
- entre `{ }` — séparateur `,`
- singleton `{a}` et vide `{ }`
- fonction `set()`

Liste

- éléments de types quelconques
- entre `[]` — séparateur `,`
- singleton `[a]` et vide `[]`
- ajout insertion suppression
- accès par l'index : son rang

Dictionnaire

- structure associative non ordonnée : `key → value`
- `{k1 : v1, k2 : v2, k3 : v3, ...}`
- accès par la clé
- unicité des clés : update valeur

Principales opérations sur les conteneurs

Tuple

Tuple

- `t[i:j:k] - len(t) - in`

Tuple

- `t[i:j:k] - len(t) - in`
- `tuple()` - conversion

Principales opérations sur les conteneurs

Tuple

- `t[i:j:k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible

Tuple

- `t[i:j:k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Principales opérations sur les conteneurs

Tuple

- `t[i:j:k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Liste

Principales opérations sur les conteneurs

Tuple

- `t[i:j:k] - len(t) - in`
- `tuple()` - conversion
- `t[0]` - lecture possible
- `t[0] = 2` - erreur en écriture

Liste

- `l[i:j:k] - del l[i:j:k] - in`

Principales opérations sur les conteneurs

Tuple

- `t[i:j:k] - len(t) - in`
- `tuple()` - conversion
- `t[0]` - lecture possible
- `t[0] = 2` - erreur en écriture

Liste

- `l[i:j:k] - del l[i:j:k] - in`
- `len(L) - sum(l) - l.copy()`

Principales opérations sur les conteneurs

Tuple

- `t[i:j:k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Liste

- `l[i:j:k]` – `del l[i:j:k]` – `in`
- `len(L)` – `sum(1)` – `l.copy()`
- `list()` – conversion

Principales opérations sur les conteneurs

Tuple

- `t[i:j:k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Liste

- `l[i:j:k]` – `del l[i:j:k]` – `in`
- `len(L)` – `sum(1)` – `l.copy()`
- `list()` – conversion
- `l[0]` – lecture possible

Principales opérations sur les conteneurs

Tuple

- `t[i : j : k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Liste

- `l[i : j : k]` – `del l[i : j : k]` – `in`
- `len(L)` – `sum(1)` – `l.copy()`
- `list()` – conversion
- `l[0]` – lecture possible
- `l[0] = 2` – écriture possible

Tuple

- `t[i : j : k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Ensemble : set

Liste

- `l[i : j : k]` – `del l[i : j : k]` – `in`
- `len(L)` – `sum(1)` – `l.copy()`
- `list()` – conversion
- `l[0]` – lecture possible
- `l[0] = 2` – écriture possible

Principales opérations sur les conteneurs

Tuple

- $t[i:j:k] - \text{len}(t) - \text{in}$
- `tuple()` – conversion
- $t[0]$ – lecture possible
- $t[0] = 2$ – erreur en écriture

Ensemble : set

- $| : \cup - \& : \cap - < : \subset$

Liste

- $l[i:j:k] - \text{del } l[i:j:k] - \text{in}$
- $\text{len}(L) - \text{sum}(1) - l.\text{copy}()$
- `list()` – conversion
- $l[0]$ – lecture possible
- $l[0] = 2$ – écriture possible

Principales opérations sur les conteneurs

Tuple

- $t[i:j:k] - \text{len}(t) - \text{in}$
- `tuple()` – conversion
- $t[0]$ – lecture possible
- $t[0] = 2$ – erreur en écriture

Ensemble : set

- $| : \cup - \& : \cap - < : \subset$
- $\wedge : \text{sym-diff} - \cup - - : \text{diff}$

Liste

- $l[i:j:k] - \text{del } l[i:j:k] - \text{in}$
- $\text{len}(L) - \text{sum}(1) - l.\text{copy}()$
- `list()` – conversion
- $l[0]$ – lecture possible
- $l[0] = 2$ – écriture possible

Principales opérations sur les conteneurs

Tuple

- `t[i : j : k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Ensemble : set

- `|` : \cup – `&` : \cap – `<` : \subset
- `^` : sym-diff – `U` – `-` : diff
- `set()` – conversion

Liste

- `l[i : j : k]` – `del l[i : j : k]` – `in`
- `len(L)` – `sum(1)` – `l.copy()`
- `list()` – conversion
- `l[0]` – lecture possible
- `l[0] = 2` – écriture possible

Principales opérations sur les conteneurs

Tuple

- $t[i:j:k] - \text{len}(t) - \text{in}$
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Ensemble : set

- $| : \cup - \& : \cap - < : \subset$
- \wedge : sym-diff $\cup - -$: diff
- `set()` – conversion
- `.add(x)` `.remove(x)` `len()`

Liste

- $l[i:j:k] - \text{del } l[i:j:k] - \text{in}$
- `len(L)` – `sum(1)` – `l.copy()`
- `list()` – conversion
- `l[0]` – lecture possible
- `l[0] = 2` – écriture possible

Principales opérations sur les conteneurs

Tuple

- `t[i : j : k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Ensemble : set

- `|` : \cup – `&` : \cap – `<` : \subset
- `^` : sym-diff – `U` – `-` : diff
- `set()` – conversion
- `.add(x)` `.remove(x)` `len()`

Liste

- `l[i : j : k]` – `del l[i : j : k]` – `in`
- `len(L)` – `sum(1)` – `l.copy()`
- `list()` – conversion
- `l[0]` – lecture possible
- `l[0] = 2` – écriture possible

Dictionnaire

Principales opérations sur les conteneurs

Tuple

- `t[i : j : k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Ensemble : set

- `|` : \cup — `&` : \cap — `<` : \subset
- `^` : sym-diff — `U` — `-` : diff
- `set()` — conversion
- `.add(x)` `.remove(x)` `len()`

Liste

- `l[i : j : k]` – `del l[i : j : k]` – `in`
- `len(L)` – `sum(1)` – `l.copy()`
- `list()` – conversion
- `l[0]` – lecture possible
- `l[0] = 2` – écriture possible

Dictionnaire

- `d[key] = value` – `len()`

Principales opérations sur les conteneurs

Tuple

- $t[i:j:k] - \text{len}(t) - \text{in}$
- `tuple()` – conversion
- $t[0]$ – lecture possible
- $t[0] = 2$ – erreur en écriture

Ensemble : set

- $| : \cup - \& : \cap - < : \subset$
- \wedge : sym-diff $\cup - -$: diff
- `set()` – conversion
- `.add(x)` `.remove(x)` `len()`

Liste

- $l[i:j:k] - \text{del } l[i:j:k] - \text{in}$
- `len(L)` – `sum(1)` – `l.copy()`
- `list()` – conversion
- $l[0]$ – lecture possible
- $l[0] = 2$ – écriture possible

Dictionnaire

- $d[\text{key}] = \text{value} - \text{len}()$
- `in` – `max()` : sur les clés

Principales opérations sur les conteneurs

Tuple

- `t[i : j : k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Ensemble : set

- `|` : \cup – `&` : \cap – `<` : \subset
- `^` : sym-diff – `U` – `-` : diff
- `set()` – conversion
- `.add(x)` `.remove(x)` `len()`

Liste

- `l[i : j : k]` – `del l[i : j : k]` – `in`
- `len(L)` – `sum(1)` – `l.copy()`
- `list()` – conversion
- `l[0]` – lecture possible
- `l[0] = 2` – écriture possible

Dictionnaire

- `d[key] = value` – `len()`
- `in` – `max()` : sur les clés
- `dict()` – conversion

Principales opérations sur les conteneurs

Tuple

- `t[i : j : k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Ensemble : set

- `|` : \cup – `&` : \cap – `<` : \subset
- `^` : sym-diff – `U` – `-` : diff
- `set()` – conversion
- `.add(x)` `.remove(x)` `len()`

Liste

- `l[i : j : k]` – `del l[i : j : k]` – `in`
- `len(L)` – `sum(1)` – `l.copy()`
- `list()` – conversion
- `l[0]` – lecture possible
- `l[0] = 2` – écriture possible

Dictionnaire

- `d[key] = value` – `len()`
- `in` – `max()` : sur les clés
- `dict()` – conversion
- `.keys().values().items()`

Principales opérations sur les conteneurs

Tuple

- `t[i:j:k]` – `len(t)` – `in`
- `tuple()` – conversion
- `t[0]` – lecture possible
- `t[0] = 2` – erreur en écriture

Ensemble : set

- `|` : \cup — `&` : \cap — `<` : \subset
- `^` : sym-diff — `U` — `-` : diff
- `set()` — conversion
- `.add(x)` `.remove(x)` `len()`

Liste

- `l[i:j:k]` – `del l[i:j:k]` – `in`
- `len(L)` – `sum(1)` – `l.copy()`
- `list()` – conversion
- `l[0]` – lecture possible
- `l[0] = 2` – écriture possible

Dictionnaire

- `d[key] = value` – `len()`
- `in` – `max()` : sur les clés
- `dict()` – conversion
- `.keys().values().items()`
- `iter_items|keys|values()`

Exécuter et interpréter les résultats

```
liste=['P','6','—','U', 'P', 'M', 'C', 'u', 'U', 'n', 'i', 'v', 'e', 'r', 's', 'i', 't', 'e', 'u', 'P', 'a', 'r', 'i', 's', 'u', 'V', 'I']
print(len(liste))
tuple(liste)
str(liste)
dict((x,0) for x in liste)
len(liste)

chaine='P6-UPMC Université Paris VI'
list(chaine)
tuple(chaine)
str(chaine)
dict((x,0) for x in chaine)
set(chaine)
len(chaine)

ensemble={'P','6','—','U', 'P', 'M', 'C', 'u', 'U', 'n', 'i', 'v', 'e', 'r', 's', 'i', 't', 'e', 'u', 'P', 'a', 'r', 'i', 's', 'u', 'V', 'I'}
print(ensemble)
list(ensemble)
tuple(ensemble)
str(ensemble)
D = dict((x,0) for x in ensemble)
set(ensemble)
len(ensemble)

nuplet=('P', '6', '—', 'U', 'P', 'M', 'C', 'u', 'U', 'n', 'i', 'v', 'e', 'r', 's', 'i', 't', 'e', 'u', 'P', 'a', 'r', 'i', 's', 'u', 'V', 'I')
list(nuplet)
tuple(nuplet)
str(nuplet)
dict((x,0) for x in nuplet)
set(nuplet)
liste[0] = 't'
print(liste)
len(nuplet)
```

Exercice 01 – maxint

- `maxint = ...`

Exercice 02

- $1+2+\dots+9999999 = \dots$

Exercice 03

- $7+14+21+\dots+2199113 =$
- $7+14+21+\dots+2199113 =$

Exercice 01 – maxint

- `maxint = ...`

Exercice 01 – solution

- `il n'y en a pas`

Exercice 02

- `1+2+...+9999999 = ...`

Exercice 03

- `7+14+21+..+2199113 =`
- `7+14+21+..+2199113 =`

Exercice 01 – maxint

- `maxint = ...`

Exercice 01 – solution

- il n'y en a pas

Exercice 02

- $1+2+\dots+9999999 = \dots$

Exercice 02 – solution

- `sum(range(9999999+1))`

Exercice 03

- $7+14+21+\dots+2199113 =$
- $7+14+21+\dots+2199113 =$

Exercice 01 – maxint

- `maxint = ...`

Exercice 01 – solution

- il n'y en a pas

Exercice 02

- $1+2+\dots+9999999 = \dots$

Exercice 02 – solution

- `sum(range(9999999+1))`

Exercice 03

- $7+14+21+\dots+2199113 =$
- $7+14+21+\dots+2199113 =$

Exercice 03 – solution

- `sum(range(7, 2199113+1, 7))`

Exercice 01 – maxint

- `maxint = ...`

Exercice 01 – solution

- il n'y en a pas

Exercice 02

- `1+2+...+9999999 = ...`

Exercice 02 – solution

- `sum(range(9999999+1))`

Exercice 03

- `7+14+21+..+2199113 =`
- `7+14+21+..+2199113 =`

Exercice 03 – solution

- `sum(range(7,2199113+1,7))`
- `sum([i for i in range(1, 2199113+1) if i%7==0])`


```
KasparovKarpov = "1. e4 e5 2. Nf3 Nc6 ... .. 41. Nf7"
```

```
KasparovKarpov = "1. e4 e5 2. Nf3 Nc6 ... .. 41. Nf7"
```

- Kasparov : e4 Nf3 Nf7

KasparovKarpov = "1. e4 e5 2. Nf3 Nc6 41. Nf7"

- Kasparov : e4 Nf3 Nf7
- Karpov : e5 Nc6 Rb8

KasparovKarpov = "1. e4 e5 2. Nf3 Nc6 41. Nf7"

- Kasparov : e4 Nf3 Nf7
- Karpov : e5 Nc6 Rb8
- liste inversée des déplacements : Nf7 Rb8 e5 e4

KasparovKarpov = "1. e4 e5 2. Nf3 Nc6 41. Nf7"

- Kasparov : e4 Nf3 Nf7
- Karpov : e5 Nc6 Rb8
- liste inversée des déplacements : Nf7 Rb8 e5 e4

Proportion de triplets $(a,b,c) \in \{a, \dots, z, A, \dots, Z\}$

KasparovKarpov = "1. e4 e5 2. Nf3 Nc6 41. Nf7"

- Kasparov : e4 Nf3 Nf7
- Karpov : e5 Nc6 Rb8
- liste inversée des déplacements : Nf7 Rb8 e5 e4

Proportion de triplets $(a,b,c) \in \{a, \dots, z, A, \dots, Z\}$

- distincts deux à deux, puis strictement croissants

KasparovKarpov = "1. e4 e5 2. Nf3 Nc6 41. Nf7"

- Kasparov : e4 Nf3 Nf7
- Karpov : e5 Nc6 Rb8
- liste inversée des déplacements : Nf7 Rb8 e5 e4

Proportion de triplets $(a,b,c) \in \{a, \dots, z, A, \dots, Z\}$

- distincts deux à deux, puis strictement croissants

gnu = "GNU GENERAL PUBLIC LIC ... lgpl.html>."

KasparovKarpov = "1. e4 e5 2. Nf3 Nc6 41. Nf7"

- Kasparov : e4 Nf3 Nf7
- Karpov : e5 Nc6 Rb8
- liste inversée des déplacements : Nf7 Rb8 e5 e4

Proportion de triplets $(a,b,c) \in \{a, \dots, z, A, \dots, Z\}$

- distincts deux à deux, puis strictement croissants

gnu = "GNU GENERAL PUBLIC LIC ... lgpl.html>."

- nombre de caractères, nombre de symboles différents

KasparovKarpov = "1. e4 e5 2. Nf3 Nc6 41. Nf7"

- Kasparov : e4 Nf3 Nf7
- Karpov : e5 Nc6 Rb8
- liste inversée des déplacements : Nf7 Rb8 e5 e4

Proportion de triplets $(a,b,c) \in \{a, \dots, z, A, \dots, Z\}$

- distincts deux à deux, puis strictement croissants

gnu = "GNU GENERAL PUBLIC LIC ... lgpl.html>."

- nombre de caractères, nombre de symboles différents
- dictionnaire des nombres d'occurrences par symboles

KasparovKarpov = "1. e4 e5 2. Nf3 Nc6 41. Nf7"

- Kasparov : e4 Nf3 Nf7
- Karpov : e5 Nc6 Rb8
- liste inversée des déplacements : Nf7 Rb8 e5 e4

Proportion de triplets $(a,b,c) \in \{a, \dots, z, A, \dots, Z\}$

- distincts deux à deux, puis strictement croissants

gnu = "GNU GENERAL PUBLIC LIC ... lgpl.html>."

- nombre de caractères, nombre de symboles différents
- dictionnaire des nombres d'occurrences par symboles
- symboles par ordre décroissant du nombre d'occurrences

- `Kasparov = KasparovKarpov.split(' ')[1 : :3]`

- `Kasparov = KasparovKarpov.split(' ')[1 : :3]`
- `Karpov = KasparovKarpov.split(' ')[2 : :3]`

- `Kasparov = KasparovKarpov.split(' ')[1 : :3]`
- `Karpov = KasparovKarpov.split(' ')[2 : :3]`
- ▶ `cps = KasparovKarpov.split(' ')`

- `Kasparov = KasparovKarpov.split(' ')[1 : :3]`
- `Karpov = KasparovKarpov.split(' ')[2 : :3]`
- - ▶ `cps = KasparovKarpov.split(' ')`
 - ▶ `([cps[i] for i in range(len(cps)) if i%3 != 0])[: :-1]`

- `Kasparov = KasparovKarpov.split(' ')[1 : :3]`
- `Karpov = KasparovKarpov.split(' ')[2 : :3]`
- ▶ `cps = KasparovKarpov.split(' ')`
 ▶ `([cps[i] for i in range(len(cps)) if i%3 != 0])[: :-1]`

```
>>> a = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) ]
>>> b = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) if len(set((x,y,z)))==3]
>>> c = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) if x<y<z]

>>> len(b)/len(a)*100, len(c)/len(a)*100
(94.30473372781066, 15.717455621301776)
```

- `Kasparov = KasparovKarpov.split(' ')[1 : :3]`
- `Karpov = KasparovKarpov.split(' ')[2 : :3]`
- ▶ `cps = KasparovKarpov.split(' ')`
 ▶ `([cps[i] for i in range(len(cps)) if i%3 != 0])[: :-1]`

```
>>> a = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) ]
>>> b = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) if len(set((x,y,z)))==3]
>>> c = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) if x<y<z]

>>> len(b)/len(a)*100, len(c)/len(a)*100
(94.30473372781066, 15.717455621301776)
```

- `len(gnu), len(set(gnu))`

- `Kasparov = KasparovKarpov.split(' ')[1 : :3]`
- `Karpov = KasparovKarpov.split(' ')[2 : :3]`
- ▶ `cps = KasparovKarpov.split(' ')`
 ▶ `([cps[i] for i in range(len(cps)) if i%3 != 0])[: :-1]`

```
>>> a = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) ]
>>> b = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) if len(set((x,y,z)))==3]
>>> c = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) if x<y<z]

>>> len(b)/len(a)*100, len(c)/len(a)*100
(94.30473372781066, 15.717455621301776)
```

- `len(gnu), len(set(gnu))`
- ▶ `fr = dict((c,gnu.count(c)) for c in set(gnu))`

- `Kasparov = KasparovKarpov.split(' ')[1 : :3]`
- `Karpov = KasparovKarpov.split(' ')[2 : :3]`
- ▶ `cps = KasparovKarpov.split(' ')`
 ▶ `([cps[i] for i in range(len(cps)) if i%3 != 0])[: :-1]`

```
>>> a = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) ]
>>> b = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) if len(set((x,y,z)))==3]
>>> c = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) if x<y<z]

>>> len(b)/len(a)*100, len(c)/len(a)*100
(94.30473372781066, 15.717455621301776)
```

- `len(gnu), len(set(gnu))`
- ▶ `fr = dict((c,gnu.count(c)) for c in set(gnu))`
 ▶ `li_fr = list((x,fr[x]) for x in fr.keys())`

- `Kasparov = KasparovKarpov.split(' ')[1 : :3]`
- `Karpov = KasparovKarpov.split(' ')[2 : :3]`
- ▶ `cps = KasparovKarpov.split(' ')`
 ▶ `([cps[i] for i in range(len(cps)) if i%3 != 0])[: :-1]`

```
>>> a = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) ]
>>> b = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) if len(set((x,y,z)))==3]
>>> c = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) if x<y<z]

>>> len(b)/len(a)*100, len(c)/len(a)*100
(94.30473372781066, 15.717455621301776)
```

- `len(gnu), len(set(gnu))`
- ▶ `fr = dict((c,gnu.count(c)) for c in set(gnu))`
 ▶ `li_fr = list((x,fr[x]) for x in fr.keys())`
- ▶ `li_fr.sort(reverse=True)`

- `Kasparov = KasparovKarpov.split(' ')[1 : :3]`
- `Karpov = KasparovKarpov.split(' ')[2 : :3]`
- ▶ `cps = KasparovKarpov.split(' ')`
 ▶ `([cps[i] for i in range(len(cps)) if i%3 != 0])[: :-1]`

```
>>> a = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) ]
>>> b = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) if len(set((x,y,z)))==3]
>>> c = [(x, y, z) for x in range(52) for y in range(52) for z in range(52) if x<y<z]

>>> len(b)/len(a)*100, len(c)/len(a)*100
(94.30473372781066, 15.717455621301776)
```

now exiting Console...

- `len(gnu), len(set(gnu))`
- ▶ `fr = dict((c,gnu.count(c)) for c in set(gnu))`
 ▶ `li_fr = list((x,fr[x]) for x in fr.keys())`
- ▶ `li_fr.sort(reverse=True)`
 ▶ `li_fr.sort(key=lambda b :b[1] , reverse=True)`

Résumé

Résumé

- versions python

Résumé

- versions python
- types simples

Résumé

- versions python
- types simples
- types composés

Résumé

- versions python
 - types simples
 - types composés
- mots clés

Résumé

- versions python
- types simples
- types composés
- mots clés
- opérations de base

Résumé

- versions python
- types simples
- types composés
- mots clés
- opérations de base
- **slicing**

Résumé

- versions python
- types simples
- types composés
- mots clés
- opérations de base
- slicing
- **conversion**

Résumé

- versions python
- types simples
- types composés
- mots clés
- opérations de base
- slicing
- conversion
- fonction range

Résumé

- versions python
- types simples
- types composés
- mots clés
- opérations de base
- slicing
- conversion
- fonction range

Questions ?