

Fondements informatiques I

Cours 7: Types et typage

Sorina Ionica sorina.ionica@uvsq.fr

Sandrine Vial sandrine.vial@uvsq.fr

Les séquences

À retenir : Une séquence est ensemble fini et ordonné d'éléments indicés de 0 à $n - 1$ (si la séquence comporte n éléments).

- les listes : modifiables
- les tuples : non-modifiables
- les chaînes de caractères : non-modifiables
- les "range" (intervalles)

x in s x not in s s + t s * n ou n * s s[i] s[i:j] s[i:j:k] len(s)	True si un élément de s est égal à x, sinon False False si un élément de s est égal à x, sinon True la concaténation de s et t ajouter s n fois à lui-même i -ème élément de s en commençant par 0 tranche (slice) de s de i à $j-1$ tranche (slice) de s de i à $j-1$ avec un pas de k longueur de s
---	--

Les séquences

À retenir : Les séquences sont des objets **itérables**.

On peut boucler sur tous éléments grâce à une boucle **for**.

```
ma_chaine = "Abracadabra"
```

```
for carac in ma_chaine:  
    print(carac)
```

```
l = [55, 23, 103, 258]  
t = (128, 0, 255)
```

```
for nb in l:  
    print(nb)
```

```
for nb in t:  
    print(nb)
```

Chaînes de caractères

- On peut notamment utiliser les *tranches*.

```
chaine = 'Abracadabra!'
print(chaine[2:5])
print(chaine[-1])
```

- La différence principale entre les listes et les chaînes de caractères est que les chaînes de caractères sont des objets **non-modifiables** après création.

```
chaine[2] = 'd' #TypeError: 'str' object does not support item assignment
liste_chaines=["Paris", "londres", "Rome", "Budapest"]
liste_chaines[1][0]='L'
```

Les encodages

En Python, un caractère est une chaîne de longueur 1.

L'encodage ASCII (American Standard Code for Information Interchange)

ASCII représente 128 caractères anglais sous forme de nombres, chaque lettre étant associée à un numéro spécifique compris entre 0 et 127.

- Par exemple, le code ASCII pour la lettre majuscule A est 65, pour la majuscule B c'est 66 etc.
- La fonction `ord` convertit un caractère dans sa représentation numérique.
- La fonction `chr` convertit un nombre dans un caractère ASCII.

Les encodages

En Python, un caractère est une chaîne de longueur 1.

L'encodage ASCII (American Standard Code for Information Interchange)

ASCII représente 128 caractères anglais sous forme de nombres, chaque lettre étant associée à un numéro spécifique compris entre 0 et 127.

- Par exemple, le code ASCII pour la lettre majuscule A est 65, pour la majuscule B c'est 66 etc.
- La fonction `ord` convertit un caractère dans sa représentation numérique.
- La fonction `chr` convertit un nombre dans un caractère ASCII.

En fait, Python repose sur l'encodage Unicode, qui comprend beaucoup plus de caractères :

```
>>> chr(245)  
'ñ'
```

Les ensembles

Un **ensemble (set)** est une collection d'éléments ne contenant pas de doublons.

Exemple : $A = \{1, 3, 10, 5, 2\}$ est un ensemble, tandis que
 $B = \{1, 10, 3, 3, 5, 2, 1\}$ n'en est pas un.

Python permet de créer des objets de type set.

- Un set n'est ni **ordonné**, ni **modifiable**.
- On utilise des {} pour créer un nouveau set.

```
s = {1, 2, 3, 4, 5}  
print(s)  
print(type(s))
```

Créer des ensembles

La fonction `set()` permet de générer un ensemble à partir de n'importe quel objet itérable.

```
# À partir d'une liste  
s = set([1, 3, 4, 5, 4, 3])
```

```
# À partir de range()  
s = set(range(10))
```

```
# À partir d'une chaîne de caractères  
s = set("Versailles")
```

Accéder aux éléments d'un ensemble

- Un set est **itérable**.
- Un set n'est pas **ordonné**, donc il n'est pas une séquence.
 - Il est impossible de récupérer un élément par sa position.

```
s = {0, 5, 10, 15}
for element in s :
    print(element)

print(s[2])      #TypeError: 'set' object is not subscriptable
```

Doublons

Les sets sont très pratiques pour *supprimer des doublons*.

Exercice : Écrire une fonction qui prend en paramètre une liste et renvoie une nouvelle liste, égale à celle passée en paramètre mais sans les doublons.

Doublons

Les sets sont très pratiques pour *supprimer des doublons*.

Exercice : Écrire une fonction qui prend en paramètre une liste et renvoie une nouvelle liste, égale à celle passée en paramètre mais sans les doublons.

```
def supprime_doublons(liste) :  
    return list(set(liste))  
  
nouvelle_liste = supprime_doublons([1, 3, 3, 5, 6, 7, 3, 1])  
print(nouvelle_liste)
```

Opérateurs sur les ensembles

Les ensembles permettent d'effectuer des opérations mathématiques telles que l'*union*, l'*intersection*, la *différence* ou la *différence symétrique*.

Exemple : $A = \{0, 1, 2, 3, 4\}$, $B = \{2, 3, 4, 5\}$

- $A \cup B := \{x \in A \text{ ou } x \in B\} = \{0, 1, 2, 3, 4, 5\}$
- $A \cap B := \{x \in A \text{ et } x \in B\} = \{2, 3, 4\}$
- $A \setminus B := \{x \in A \text{ et } x \notin B\} = \{0, 1\}$
- $A \Delta B := \{x \in A \cup B \text{ et } x \notin A \cap B\} = \{0, 1, 5\}$

Opérateurs sur les ensembles

- $a \mid b : A \cup B$
- $a \& b : A \cap B$
- $a - b : A \setminus B$
- $a \wedge b : A \Delta B$

```
chaine1 = 'chaussette'  
chaine2 = 'archiduchesse'  
  
a = set(chaine1)  
b = set(chaine2)  
  
print(a | b)  
print(a & b)  
print(a - b)  
print(a ^ b)
```

Dictionnaires

Un dictionnaire est une structure de données, dont les éléments sont accessibles grâce à une *clé*.

- Pour construire un dictionnaire, on utilise des accolades {}.
- Un élément d'un dictionnaire est un couple **clé**: **valeur**.
- On accède à une **valeur** du dictionnaire en utilisant la **clé**.

Exemple

```
dico_notes = {"Gaëtan" : 14, "Laure" : 9, "Kenza" : 17, "Gabriel" : 14}  
print(dico_notes["Laure"])
```

Dictionnaires

- Dans un dictionnaire, les clés doivent être *uniques*, mais pas les valeurs.
- Les clés peuvent être n'importe quel objet non-modifiable (immutable)
 - Par exemple des int ou des string.
- Il est facile de créer et modifier un dictionnaire.

```
dico_notes = {"Gaëtan" : 14, "Laure" : 9, "Kenza" : 17, "Gabriel" : 14}

# Changer une entrée existante
dico_notes["Gaëtan"] = 12

# Ajouter une nouvelle entrée

dico_notes["Anne"] = 15
print(dico_notes)

# Créer un dictionnaire vide.
dico_vide = {}
```

Parcourir un dictionnaire

On peut vérifier facilement qu'une valeur est dans le dictionnaire si on connaît sa clé.

```
cle="Kenza"  
if cle in dico_notes:  
    print(cle, "a obtenu la note", dico_notes[cle])
```

Les clés et leurs valeurs peuvent être récupérées en même temps en utilisant la méthode `items()`.

```
dico_notes = {"Gaëtan" : 14, "Laure" : 9, "Kenza" : 17, "Gabriel" : 14}  
  
for i in dico_notes.items():  
    print(i)
```

Typage en Python

Rappel : En Python le typage est dynamique.

```
a=5
b=7
print(a+b)  # int + int
x="salut"
y="hello"
print("salut" + "hello") # str + str
print(a + y)          #int + str  Erreur!

# Problème de type pour l'opération /
"hello" / 2
```

Annotation de type

- Depuis la version 3.5 on peut ajouter **des annotations de type** aux variables et fonctions.
- Ces annotations permettent d'apporter plus de structure aux programmes pour éviter certains bugs.
- Ces annotations sont facultatives et vous pouvez mélanger du code annoté ou non.

```
a: int = 5
b: int =7
print(a+b)  # int + int
x: str="salut"
y: str = "hello"

y=6
print(type(y))

liste: list[int] = [1, 2, 3]
```

Annotation de type pour les fonctions

On peut spécifier le type des arguments.

- Les annotations `a: int, b:int` indiquent que les paramètres doivent être des entiers.

Exemple

```
def monPrint(a : int, b: int):  
    print(a + b)
```

```
mon_print(5,7)  
mon_print("salut","hello")  
mon_print(5,"salut")
```

- Les informations de type ne changent rien à l'exécution.
- Il y a une erreur de type seulement quand on applique une fonction à un élément et qu'il n'est pas du type attendu par la fonction.

Annotation de type

On peut spécifier le type de retour.

- L'annotation `-> int` indique que la fonction renvoie un entier.

Exemple

```
def somme(a: int, b: int) -> int:  
    return a + b
```

```
print(somme(5,7))  
print(somme("salut","hello"))
```

- Pour détecter les erreurs, il faut utiliser des outils comme `mypy`.