Types complexes

demos
Learning is changing®

Présentation des listes Python

Jusqu'à présent, nous n'avons stocké qu'une seule valeur à la fois dans nos variables. Les listes sont un type de données très particulier au sens où elles représentent des données composées ou combinées.

Pour définir une nouvelle liste en Python, on va devoir utiliser une paire de crochets [ ]. Nous allons placer les différents éléments de notre liste dans ces crochets en les séparant par des virgules.

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Affichage

```
['apple', 'banana', 'cherry']
```

De plus, vous devez savoir que tous les éléments d'une liste n'ont pas à être du même type,

```
thislist = ["Un",2,3.00]
print(thislist)
```

```
['Un', 2, 3.0]
```

Récuperation d'une ou plusieurs valeurs dans une liste

Les listes Python sont par défaut indexées ou indicées. Cela signifie que chaque valeur d'une liste est lié à un indice qu'on va pouvoir utiliser pour récupérer cette valeur en particulier.

Les listes possèdent des indices numériques qui commencent à 0.
Pour récupérer une valeur en particulier dans une liste, on va devoir préciser le nom de la liste suivi de l'indice de cette valeur entre crochets

```
mylist = ["apple", "banana", "cherry"]
print(mylist[1])
```

Le Silcing

On va également pouvoir récupérer une tranche de valeurs dans une liste, c'est-à-dire un ensemble de valeurs qui se suivent. Pour cela, on utilisera le symbole : mylist[debut:fin]
Ex :

```
mylist = ["apple", "banana", "cherry"]
print(mylist[0:2])
```

```
['apple', 'banana']
```

Si on utilise : sans indice, alors une copie superficielle de la liste sera renvoyée. Si on mentionne un indice avant : mais pas d'indice après, alors une copie superficielle partielle de la liste de départ sera renvoyée, en commençant à copier à partir de l'indice donné. Si au contraire on mentionne un indice après « : » mais pas d'indice avant, une copie superficielle partielle de la liste de départ sera renvoyée qui commence au début de la liste et jusqu'à l'indice donné.

```
mylist = ["apple", "banana", "cherry"]
print(mylist[1:])
print(mylist[:1])
print(mylist[:])
```

```
['banana', 'cherry']
['apple']
['apple', 'banana', 'cherry']
```

Gestion des éléments d'une liste

A la différence des types de données simples comme les chaînes qui sont immuables, les listes sont un type de données altérable ce qui signifie qu'on va pouvoir altérer leur structure ou modifier leur contenu en ajoutant, supprimant ou remplaçant des valeurs.

```
mylist = ["apple", "banana", "cherry","strawberry"]
mylist[1]="Banana2"
print(mylist)
mylist[2:]=[]
print(mylist)
```

```
['apple', 'Banana2', 'cherry', 'strawberry']
['apple', 'Banana2']
```

```
l = [0, 10, 20, 30, 40, 50]
del l[0]
print(l)
# [10, 20, 30, 40, 50]
del l[3]
print(l)
# [10, 20, 30, 50]
```

Concaténation de listes

```
l = [0, 10, 20]
y = [30,40,50]
ly=l+y
print(ly)
```

```
[0, 10, 20, 30, 40, 50]
```

Gestion des éléments d'une liste

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Les tuples

Les tuples ressemblent aux listes : un tuple consiste en différentes valeurs entourées par des virgules.
La grande différence entre un tuple et une liste est qu'un tuple est une donnée immuable à la différence d'une liste qui est altérable. Cela signifie qu'on ne va pas pouvoir modifier les valeurs d'un tuple après sa création.

```
mytuple = ("apple", "banana", "cherry")
print(mytuple)
print(mytuple[0])
```

```
('apple', 'banana', 'cherry')
apple
```

Il va donc être intéressant d'utiliser des tuples plutôt que des listes dans les cas où on veut s'assurer que les données ne soient pas modifiées dans un programme

Les dictionnaires en Python

Les dictionnaires sont un type de données intégré à Python qui permet de stocker plusieurs valeurs indexées par des clés uniques, de manière similaire aux données séquentielles.
La principale distinction entre les données séquentielles et les dictionnaires réside dans la méthode d'indexation des valeurs et la nature de l'index utilisé.
Les séquences utilisent des indices numériques commençant à 0 pour associer les différentes valeurs.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
print(thisdict["brand"])
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
Ford
```

Les dictionnaires en Python

Les dictionnaires sont des structures de données modifiables, permettant d'ajouter ou de modifier des valeurs après leur création.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["model"]="Fiesta"
del thisdict["year"]
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Fiesta'}
```

Les dictionnaires en Python

```
a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}
```

```
for key in a_dict:
    print(key, '->', a_dict[key])
```

```
for key, value in a_dict.items():
    print(key, '->', value)
```

```
>>> keys = a_dict.keys()
>>> keys
dict_keys(['color', 'fruit', 'pet'])
```

Les dictionnaires en Python

clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary