Fondements informatiques I

Cours 5: Types mutables et immutables
Fonctions (suite et fin)

Sorina Ionica sorina.ionica@uvsq.fr

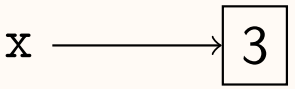
Sandrine Vial sandrine.vial@uvsq.fr

Types mutables et immutables

En Python, on distingue deux types d'objets:

1. objets immutables : qu'on ne peut pas changer
   - entiers, flottants
   - tuples
   - chaînes de caractères

```
x=3
x+=1
```

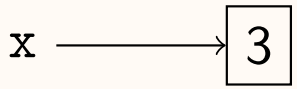


Types mutables et immutables

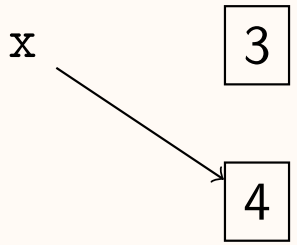
En Python, on distingue deux types d'objets:

1. objets immutables : qu'on ne peut pas changer
   - entiers, flottants
   - tuples
   - chaînes de caractères

```
x=3
x+=1
```



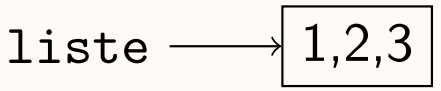
Après incrémentation:



Variables mutables et immutables

2. objets mutables: dont on peut modifier le contenu
   - les listes
   - les ensembles (sets)
   - les dictionnaires (qu'on verra plus tard)

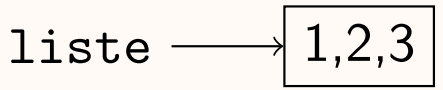
```
liste=[1,2,3]
liste[0]=5
```



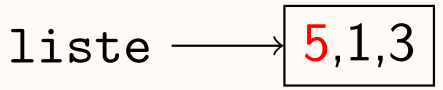
Variables mutables et immutables

2. objets mutables: dont on peut modifier le contenu
   - les listes
   - les ensembles (sets)
   - les dictionnaires (qu'on verra plus tard)

```
liste=[1,2,3]
liste[0]=5
```



Après modification:



Questions

Que se passe-t-il à l'exécution?

```
mon_tuple=(1,2,3)
mon_tuple[0]=5
```

```
chaine="hello world"
chaine[0]='H'
```

Questions

Ecrivez un programme qui prend une chaîne de caractères et affiche la chaîne avec la première lettre de chaque mot en majuscule.

Questions

Ecrivez un programme qui prend une chaîne de caractères et affiche la chaîne avec la première lettre de chaque mot en majuscule.

```
resultat=""
debut_mot = True  # Indique si on est au début d'un mot

for c in chaine:
        if debut_mot and c != " ":      # Si c'est le début d'un mot
            resultat += c.upper()       # Mettre en majuscule
            debut_mot = False
        else:
            resultat += c
        if c == " ":
            debut_mot = True

print(resultat)
```

Les variables d'une fonction

- Les variables définies dans une fonction sont appelées **variables locales**.
- La portée d'une variable locale est limitée à la fonction où elle a été définie, on ne peut pas y faire référence en dehors.

Exemple

```
def incremente(x):
    x +=  1
    z = 2

incremente(2)
print(x)
#print(z)
```

Lors de l'exécution, l'affichage produit un NameError (variable pas définie).

Évaluation de la fonction

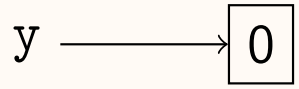
- Les arguments sont des variables locales à la fonction.
- Les arguments sont passés par la copie d'une référence sur l'objet donné en argument.

```
def incremente(x):
    x += 1

y = 0
incremente(y)
print(y)
```

Évaluation de la fonction

- Les arguments sont des variables locales à la fonction.
- Les arguments sont passés par la copie d'une référence sur l'objet donné en argument.

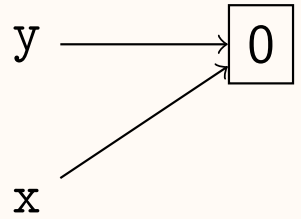
```
def incremente(x):
    x += 1

y = 0
incremente(y)
print(y)
```



Évaluation de la fonction

- Les arguments sont des variables locales à la fonction.
- Les arguments sont passés par la copie d'une référence sur l'objet donné en argument.

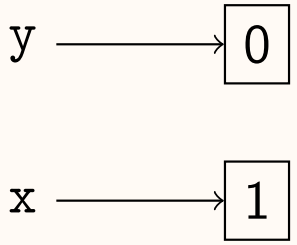
```
def incremente(x):
    x += 1

y = 0
incremente(y)
print(y)
```



Évaluation de la fonction

- Les arguments sont des variables locales à la fonction.
- Les arguments sont passés par la copie d'une référence sur l'objet donné en argument.

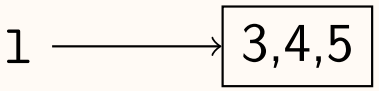
```
def incremente(x):
    x += 1

y = 0
incremente(y)
print(y)
```



Référence sur une liste

À retenir :

- Sur une liste, on peut modifier son contenu depuis la fonction, car objet mutable.



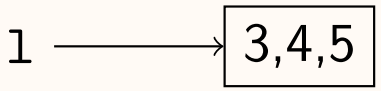

```
def incrementeListe(k):
    for i in range(len(k)):
        k[i] += 1

l = [3,4,5]
incrementeListe(l)
print(l)
```

Référence sur une liste

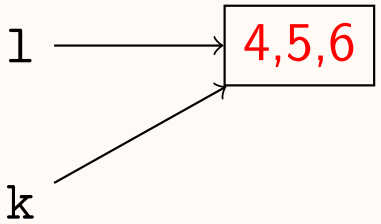
À retenir :

- Sur une liste, on peut modifier son contenu depuis la fonction, car objet mutable.

```
def incrementeListe(k):
    for i in range(len(k)):
        k[i] += 1

l = [3,4,5]
incrementeListe(l)
print(l)
```



Après appel `incrementeListe`

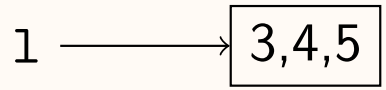


Référence sur une liste

À retenir :

- Cependant, on ne peut pas modifier la liste elle même!

```
def supprimeListe(k):
    k= []

l = [3,4,5]
supprimeListe(l)
print(l)
```

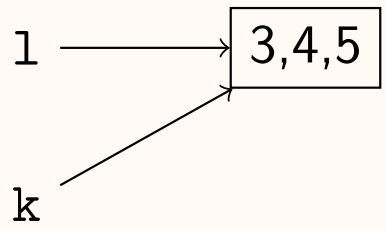


Référence sur une liste

À retenir :

- Cependant, on ne peut pas modifier la liste elle même!

```
def supprimeListe(k):
    k= []

l = [3,4,5]
supprimeListe(l)
print(l)
```

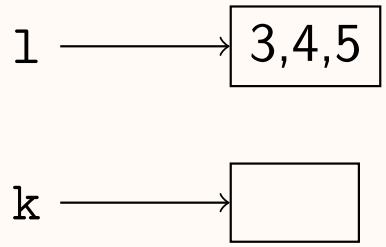


Référence sur une liste

À retenir :

- Cependant, on ne peut pas modifier la liste elle même!

```
def supprimeListe(k):
    k= []

l = [3,4,5]
supprimeListe(l)
print(l)
```



Utilisation de variables globales

- Les variables définies hors des fonctions sont **globales** par opposition aux variables **locales** des fonctions.
- On peut les utiliser dans toutes les fonctions.
- Attention: on ne peut pas changer la valeur d'une variable globale à l'intérieur d'une fonction.

```
x = 1
print(x)

def affiche(): #utilisation d'une variable globale
    print(x)

x += 3
affiche()

def ajoute(): #modification d'une variable globale
    x += 1

ajoute()
print(x)
```

Utilisation de variables globales

- Pour modifier une variable globale, il faut spécifier dans la fonction que la variable est globale par le mot clé `global`.
  - Il est *déconseillé* de faire usage de ce mot clé et des variables globales en général.

```
x=1

def ajoute(): #modif. d'une var. globale avec le mot clé "global"
    global x
    x += 1

ajoute()
print(x)
```

Utilisation de variables globales

Qu'affiche-t-on à l'exécution?

```
x=3

def ajoute():  #rédefinir une variable globale
    x=5
    x += 1

ajoute()
print(x)
```

Explication : On a rédefinit la variable `x`.

Représentation des arguments dans un appel de fonction

- On peut donner des valeurs par défaut aux arguments d'une fonction de la manière suivante:
  `def ma_fonction(pays, age = 1, nom = "toto").`
- Les variables ayant une valeur par défaut peuvent être omises.

```
def ma_fonction(pays, age = 1, nom = "toto"):
    print(nom," a ", age, "ans et vit en ", pays)

ma_fonction("france")
ma_fonction("allemagne", 18, "kurt")
ma_fonction("italie", 77)
```

Ordre des arguments dans un appel de fonction

- En règle générale, on respecte l'ordre des arguments donnes dans la signature.
- On peut également donner les arguments dans le désordre en spécifiant leur nom.

Exemple

```
def ma_fonction(pays, age = 1, nom = "toto"):
    print(nom," a ", age, "ans et vit en ", pays)

ma_fonction(nom = "kader", age = 18, pays = "algérie")
ma_fonction("france", nom = "sylvie")
```

Appel de fonction depuis une autre fonction

La fonction peut appeler une autre fonction.

Exemple

```
def doubler(x):
    return 2*x

def sommer(x,y):
    return x+doubler(y)

print(sommer(2,3))
```

Appel de fonction depuis une autre fonction

Que affiche-t-on à l'exécution? Au besoin, corriger afin de doubler les valeurs dans la liste.

```
def doubler(x):
    return 2*x


 def doubler_liste(l):
      for i in range(len(l)):
          doubler(l[i])
      return l

l=[2,3,4]
print(doubler_liste(l))
```

Expressions lambda

- Pour définir une fonction courte, il existe une syntaxe alternative utilisant l'opérateur **lambda**.
- La définition doit tenir sur une ligne.
  - On ne peut pas utiliser des instructions de contrôle

```
g = lambda x: x*2
print(g(2))
```

Expressions lambda

- Pour définir une fonction courte, il existe une syntaxe alternative utilisant l'opérateur **lambda**.
- La définition doit tenir sur une ligne.
  - On ne peut pas utiliser des instructions de contrôle

```
g = lambda x: x*2
print(g(2))
```

- La fonction définie par un lambda est anonyme, c'est à dire qu'elle n'a pas de nom.
  - C'est utile quand la fonction sert une seule fois

```
print((lambda x: x*2)(3))

#applique la fonction à chaque élément de range(10)
list(map(lambda x: x*2,range(10)))
```