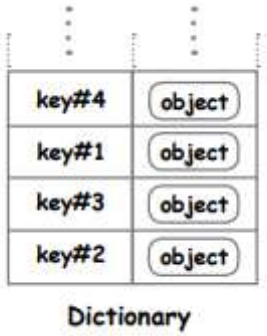
Les Dictionaires en Python

Semaine 10

Ali Assi, PhD

Le dictionnaire est un ensemble **non ordonné de paires clé-valeur** où les clés doivent être uniques.



Cette structure clé-valeur signifie qu'au lieu d'accéder aux éléments en utilisant un index numérique comme dans les listes ou les tuples, nous utilisons leur clé pour accéder à leur valeur.

Leurs principales caractéristiques sont les suivantes :

1. **Mutables** : Ils peuvent être modifiés une fois qu'ils ont été définis
2. **Indexables** : Nous pouvons accéder à des éléments spécifiques si nous connaissons leur position (leur clé) dans le dictionnaire.
3. **Aucun doublon de clé** : Ils ne peuvent pas contenir de doublons dans les valeurs des clés, mais ils peuvent en contenir dans les valeurs.

Comment définir un dictionnaire Python ?

Il existe de plusieurs façons pour créer un dictionnaire en Python :

```
1. Ajoutez des paires clé-valeur séparées par des virgules à l'intéri
eur d'accolades.
2. Convertir une liste de tuples en dictionnaire.
3. Définir un dictionnaire en utilisant des arguments de mots-clés.
4. Créer un dictionnaire de manière incrémentielle.
```

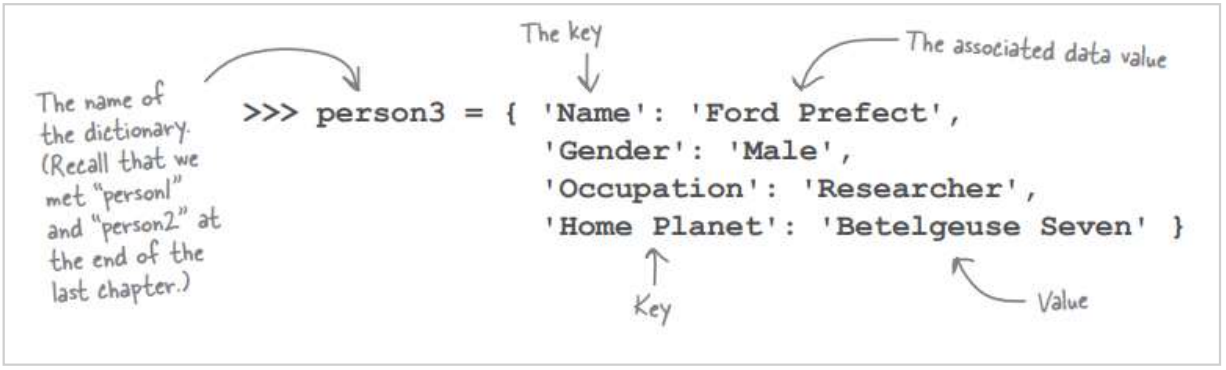
1. **Paires clé:valeur séparées par des virgules**

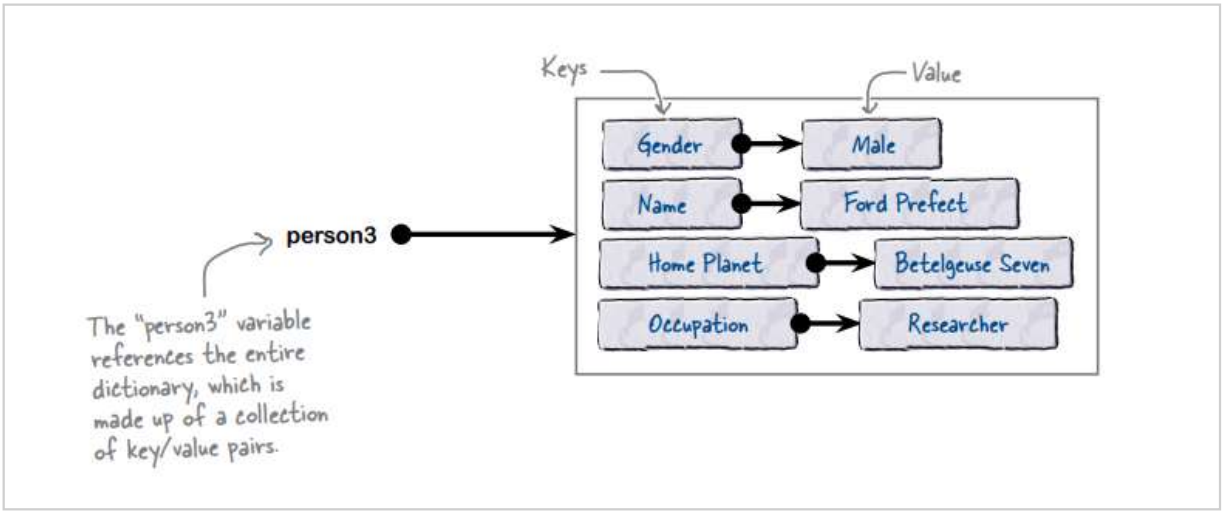
C'est la façon la plus simple de créer un dictionnaire en Python : ajoutez des clés et des valeurs associées à l'intérieur d'un `{}` . Chaque clé doit être suivie de deux points qui la séparent de la valeur.

In [3]:
```
#exemple 1
person3 = { 'Name': 'Ford Prefect',
            'Gender': 'Male',
            'Occupation': 'Researcher',
            'Home Planet': 'Betelgeuse Seven'
          }
```

In [4]:
```
print(person3)
```

```
{'Name': 'Ford Prefect', 'Gender': 'Male', 'Occupation': 'Researcher', 'Home
Planet': 'Betelgeuse Seven'}
```





Comment accéder aux valeurs du dictionnaire Python

L'accès aux éléments d'un dictionnaire est différent de l'accès aux éléments d'une liste ou d'un tuple car au lieu d'utiliser leur index numérique, nous avons des clés que nous pouvons utiliser pour accéder aux valeurs.

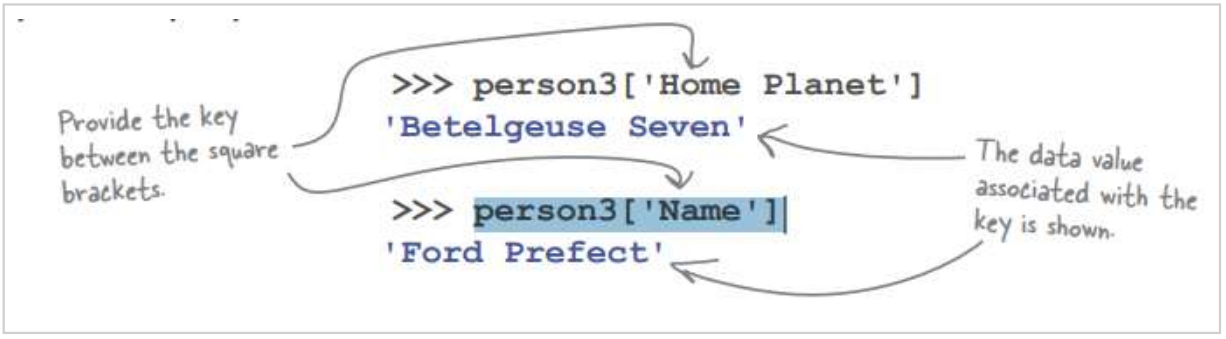
Il existe deux méthodes principales :

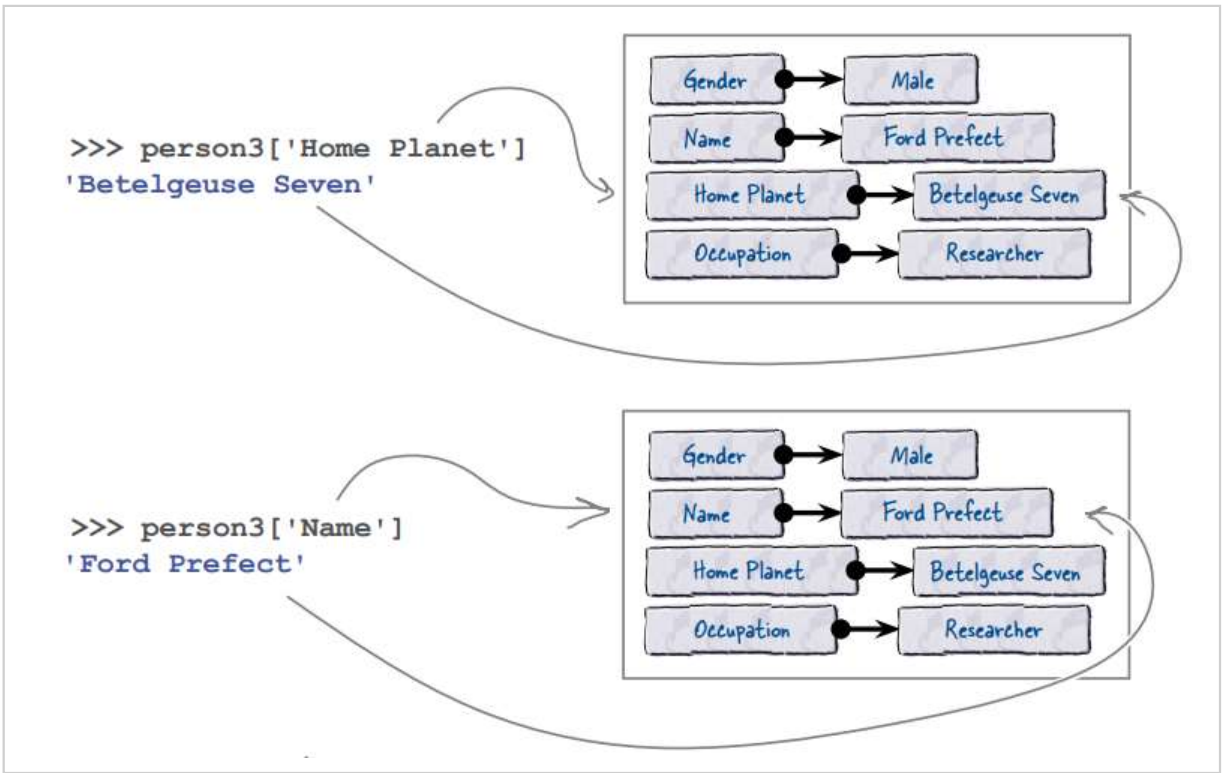
1. Utilisation de la notation `[]` pour spécifier la clé pour laquelle nous voulons accéder à la valeur.
2. Utilisation de la méthode `get()` pour spécifier la clé à laquelle nous voulons accéder à la valeur. La méthode `get` renvoie une valeur pour une clé donnée. Si une clé n'existe pas, le dictionnaire renvoie par défaut `None`.

In [1]:
```
#exemple 5
person3 = { 'Name': 'Ford Prefect',
            'Gender': 'Male',
            'Occupation': 'Researcher',
            'Home Planet': 'Betelgeuse Seven'
          }


print(person3["Home Planet"])
print(person3['Name'])
```

```
Betelgeuse Seven
Ford Prefect
```





In [35]:
```
print(person3.get('Home Planet'))
print(person3.get('Name'))
```

```
Betelgeuse Seven
Ford Prefect
```

la différence entre les deux est que pour `get` si la clé n'existe pas, une erreur ne sera pas déclenchée, alors que pour la première méthode, une erreur sera déclenchée. Par exemple:

In [2]:
```
#exemple 6
person3 = { 'Name': 'Ford Prefect',
            'Gender': 'Male',
            'Occupation': 'Researcher',
            'Home Planet': 'Betelgeuse Seven'
          }


print(person3.get("married")) ## None
print(person3["married"]) ## an exception
```

```
None

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[2], line 10
      2 person3 = { 'Name': 'Ford Prefect',
      3             'Gender': 'Male',
      4             'Occupation': 'Researcher',
      5             'Home Planet': 'Betelgeuse Seven'
      6           }
      9 print(person3.get("married")) ## None
---> 10 print(person3["married"]) ## an exception

KeyError: 'married'
```

Mutable

Comme les listes, et contrairement aux tuples, les dictionnaires sont mutables, ce qui signifie que nous pouvons modifier, ajouter ou supprimer des éléments une fois que le dictionnaire a été créé.

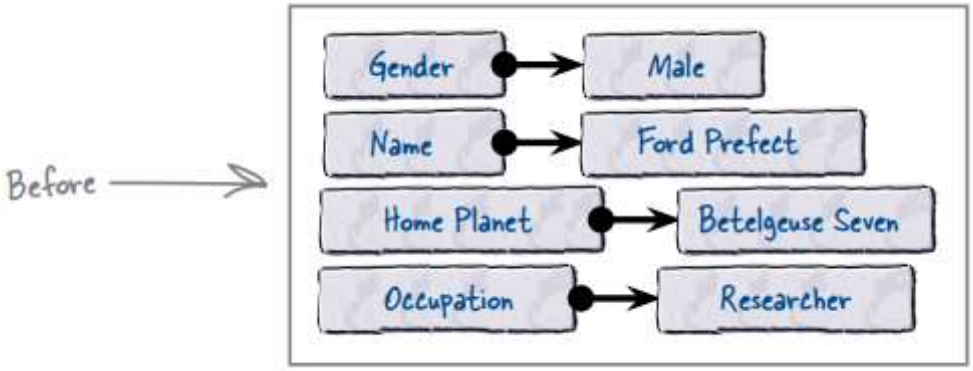
Nous pouvons le faire d'une manière similaire en accédant aux éléments individuels et en utilisant l'affectation de variable `( = )` pour modifier la valeur réelle.

Nous pouvons également ajouter de nouvelles paires clé-valeur en spécifiant simplement une nouvelle clé, puis en lui attribuant une valeur (ou même pas).

In [5]:
```
#exemple 8
#create the dictionary
person3 = {'Name': 'Ford Prefect',
           'Gender': 'Male',
           'Home Planet': 'Betelgeuse Seven',
           'Occupation': 'Researcher'}
```

In [6]:
```
person3
```

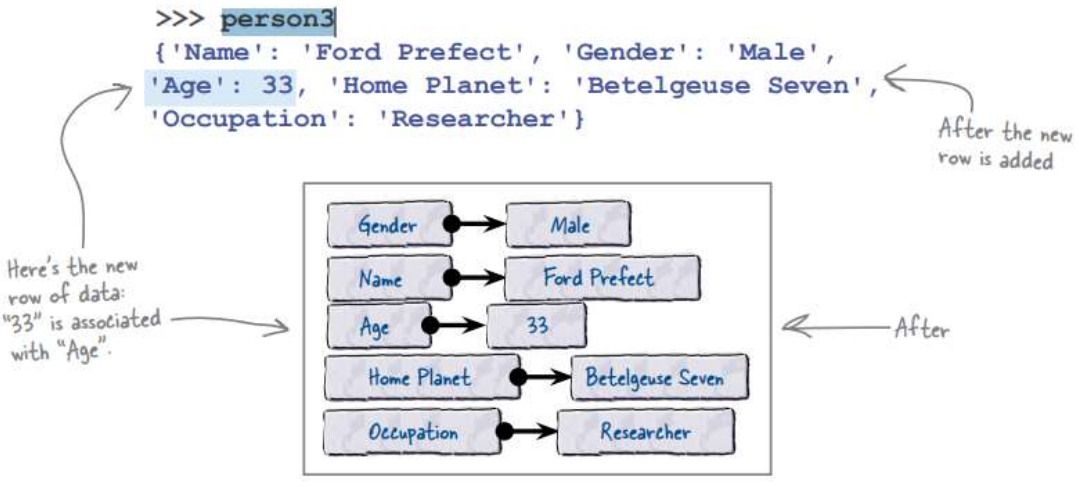
Out[6]:
```
{'Name': 'Ford Prefect',
 'Gender': 'Male',
 'Home Planet': 'Betelgeuse Seven',
 'Occupation': 'Researcher'}
```



In [38]:
```
# Assign an object (in this case, a number) to a new key to add a row of data
person3['Age'] = 33
```

In [39]:
```
print(person3)
```

```
{'Name': 'Ford Prefect', 'Gender': 'Male', 'Home Planet': 'Betelgeuse Seven',
'Occupation': 'Researcher', 'Age': 33}
```



Enfin, nous pouvons ajouter un nouveau **dictionnaire à un dictionnaire** existant en utilisant la méthode `update()` :

In [40]:
```
person3 = {'Name': 'Ford Prefect',
           'Gender': 'Male',
           'Home Planet': 'Betelgeuse Seven',
           'Occupation': 'Researcher'}

person3_age = {'Age': 33}
person3.update(person3_age)
```

In [41]:
```
print(person3)
```

```
{'Name': 'Ford Prefect', 'Gender': 'Male', 'Home Planet': 'Betelgeuse Seven',
'Occupation': 'Researcher', 'Age': 33}
```

Nous pouvons donc modifier les informations contenues dans un dictionnaire, bien que nous ne puissions pas modifier la **clé réelle**, à part la supprimer.

Donc une clé de dictionnaire est immuable. Il est donc possible d'utiliser par exemple un tuple comme clé de dictionnaire. Mais une liste ou un autre dictionnaire ne peuvent pas être utilisés comme clé.

Voici comment on supprime des éléments d'un dictionnaire:

In [7]:
```
#exemple 9
student = {
    "name": "Sofie",
    "graduated": False,
    "age": 23,
    "married": False,
    "courses": ["Java", "Maths", "Python"]
}

#we can use the del method
del student["married"]

print(student)
```

```
{'name': 'Sofie', 'graduated': False, 'age': 23, 'courses': ['Java', 'Maths',
'Python']}
```

Opérateurs de dictionnaire

Ensuite, nous allons apprendre les opérations utiles que nous pouvons effectuer sur les dictionnaires.

Instruction `in`

In [43]:
```
#exemple 10
person3 = {'Name': 'Ford Prefect',
           'Gender': 'Male',
           'Home Planet': 'Betelgeuse Seven',
           'Occupation': 'Researcher'}

# Par exemple, vérifions si un ```person3``` a une clé ```Age``` :
flag = "Age" in person3

print(flag)

# Par exemple, vérifions si un ```student``` a une clé ```married``` :
flag = "Gender" in person3

print(flag)
```

```
False
True
```

Fonction `len()`

Pour vérifier la longueur d'un dictionnaire, on utilise la méthode intégrée `len()` .

In [44]:
```
#exemple 12
person3 = {'Name': 'Ford Prefect',
           'Gender': 'Male',
           'Home Planet': 'Betelgeuse Seven',
           'Occupation': 'Researcher'}

print(len(person3))
```

```
4
```

Méthodes du dictionnaire Python

Le dictionnaire Python dispose d'un grand nombre de méthodes intégrées utiles. Celles-ci peuvent vous aider à gagner du temps lorsque vous travaillez avec des dictionnaires.

Ces méthodes sont :

1. `dict.clear()`
2. `dict.copy()`
3. `dict.items()`
4. `dict.keys()`
5. `dict.pop()`
6. `dict.values()`
7. Etc.

dict.clear()

Cette méthode efface tout le dictionnaire. Par exemple:

In [45]:
```
#exemple 13
person3 = {'Name': 'Ford Prefect',
           'Gender': 'Male',
           'Home Planet': 'Betelgeuse Seven',
           'Occupation': 'Researcher'}

print(person3)
print("====")
person3.clear()
print("====")
print(person3)
```

```
{'Name': 'Ford Prefect', 'Gender': 'Male', 'Home Planet': 'Betelgeuse Seven',
'Occupation': 'Researcher'}
====
====
{}
```

dict.copy()

La méthode `copy()` permet de créer une copie du dictionnaire. Par exemple:

In [11]:
```
#exemple 14
person3 = {'Name': 'Ford Prefect',
           'Gender': 'Male',
           'Home Planet': 'Betelgeuse Seven',
           'Occupation': 'Researcher'}

person4 = person3.copy()

person3 = None
print(person4)
print(person3)
```

```
{'Name': 'Ford Prefect', 'Gender': 'Male', 'Home Planet': 'Betelgeuse Seven',
'Occupation': 'Researcher'}
None
```

dict.items()

La méthode `items()` renvoie renvoie un objet de type liste de tuples où chaque tuple est de la forme `(clé, valeur)` .

In [15]:
```
#exemple 16
person3 = {'Name': 'Ford Prefect',
           'Gender': 'Male',
           'Home Planet': 'Betelgeuse Seven',
           'Occupation': 'Researcher'}

person3_object= person3.items()
print(person3_object)
```

```
dict_items([('Name', 'Ford Prefect'), ('Gender', 'Male'), ('Home Planet', 'Be
telgeuse Seven'), ('Occupation', 'Researcher')])
```

Si vous n'appréciez pas l'objet view, vous pouvez le convertir en liste avec la méthode `list()` . Par exemple:

In [28]:
```
#exemple 17
print(list(person3_object))
```

```
[('Name', 'Ford Prefect'), ('Gender', 'Male'), ('Home Planet', 'Betelgeuse Se
ven'), ('Occupation', 'Researcher')]
```

dict.keys()

La méthode `keys()` renvoie toutes les clés du dictionnaire sous forme de liste dans un objet de vue. Par exemple :

In [29]:
```
#exemple 18
person3 = {'Name': 'Ford Prefect',
           'Gender': 'Male',
           'Home Planet': 'Betelgeuse Seven',
           'Occupation': 'Researcher'}
person3_keys = list(person3.keys())

print(person3_keys)
```

```
['Name', 'Gender', 'Home Planet', 'Occupation']
```

dict.pop()

La méthode `pop()` permet de supprimer une clé et renvoie sa valeur.

In [18]:
```
#exemple 19
person3 = {'Name': 'Ford Prefect',
           'Gender': 'Male',
           'Home Planet': 'Betelgeuse Seven',
           'Occupation': 'Researcher'}

person3_pop = person3.pop("Occupation")

print(person3_pop)
print(person3)

person3["Name"]= "XXXX"
person3["Home Planet"]= "YYYYYY"
person3
```

```
Researcher
{'Name': 'Ford Prefect', 'Gender': 'Male', 'Home Planet': 'Betelgeuse Seven'}
```

Out[18]:
```
{'Name': 'XXXX', 'Gender': 'Male', 'Home Planet': 'YYYYYY'}
```

dict.values()

La méthode `values()` renvoie une liste des valeurs du dictionnaire sous la forme d'un objet de vue. Par exemple :

In [19]:
```
#exemple 22
person3 = {'Name': 'Ford Prefect',
           'Gender': 'Male',
           'Home Planet': 'Betelgeuse Seven',
           'Occupation': 'Researcher'}
values_dic = list(person3.values())
print(values_dic)
```

```
dict_values(['Ford Prefect', 'Male', 'Betelgeuse Seven', 'Researcher'])
```

Comment parcourir en boucle des dictionnaires Python

Vous pouvez parcourir un dictionnaire en boucle à l'aide d'une boucle `for` .

Pour pouvoir boucler le dictionnaire, vous devez utiliser la méthode `dict.items()` . Par exemple:

In [22]:
```
#exemple 23
person3 = {'Name': 'Ford Prefect',
           'Gender': 'Male',
           'Home Planet': 'Betelgeuse Seven',
           'Occupation': 'Researcher'}

for key, value in person3.items():
    print(key, ": ", value)
print("-----")
for k in person3.keys():
    print(k, ": " person3)
```

```
Name :  Ford Prefect
Gender :  Male
Home Planet :  Betelgeuse Seven
Occupation :  Researcher
-----
Name :
Gender :
Home Planet :
Occupation :
```

Comment Pretty-Print un dictionnaire Python

Au cours de ce chapitre, vous avez imprimé des dictionnaires dans la console. Mais le résultat est plutôt désordonné car il est sur une seule ligne.

Lorsque vous imprimez de plus gros dictionnaires, il devient impossible de comprendre le résultat. C'est là que l'impression jolie peut vous aider.

Python possède un module `pprint` intégré. Vous pouvez l'importer et l'utiliser pour imprimer un dictionnaire.

In [33]:
```
#exemple 26
from pprint import pprint

person3 = {'Name': 'Ford Prefect',
           'Gender': 'Male',
           'Home Planet': 'Betelgeuse Seven',
           'Occupation': 'Researcher'}
pprint(person3)
print("========================")
print(person3)
```

```
{'Gender': 'Male',
 'Home Planet': 'Betelgeuse Seven',
 'Name': 'Ford Prefect',
 'Occupation': 'Researcher'}
========================
{'Name': 'Ford Prefect', 'Gender': 'Male', 'Home Planet': 'Betelgeuse Seven',
'Occupation': 'Researcher'}
```

In [ ]: