Informatique 1
Introduction à la programmation

Mission 7 : restructuration

Kim Mens **Siegfried Nijssen** Charles Pecheur

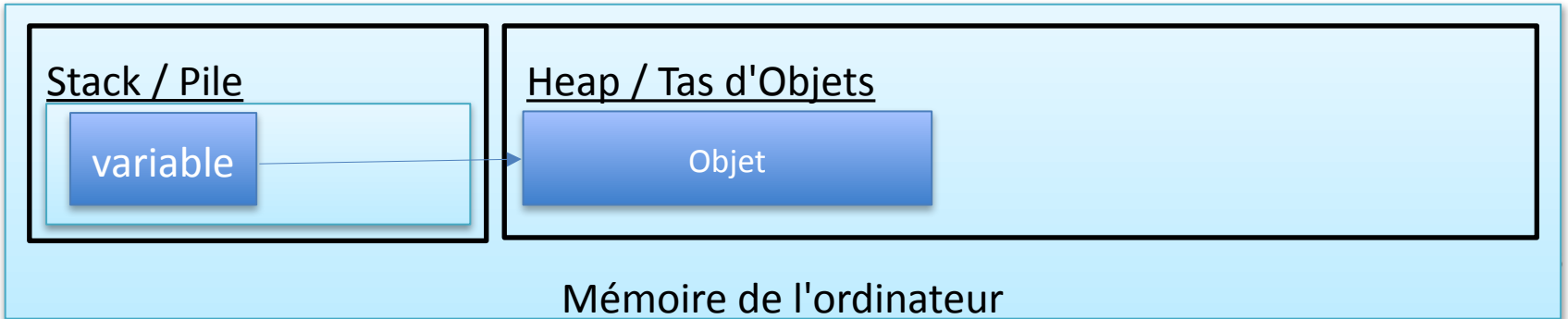
MISSION 7 : RESTRUCTURATION

Types en Python

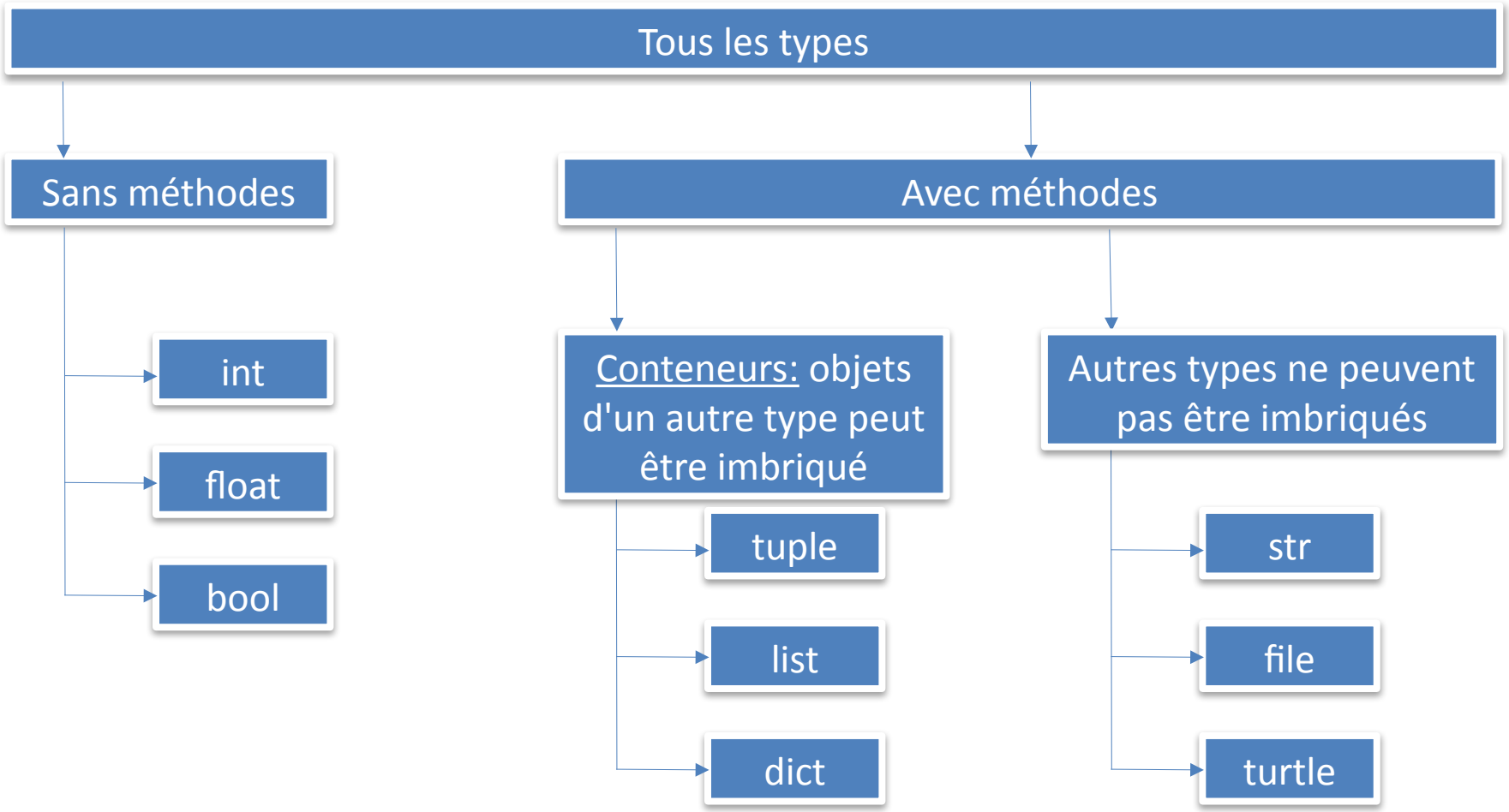
- En Python, chaque objet a un type

Int	String	List
Float	File	Tuple
Bool	Turtle	Dict

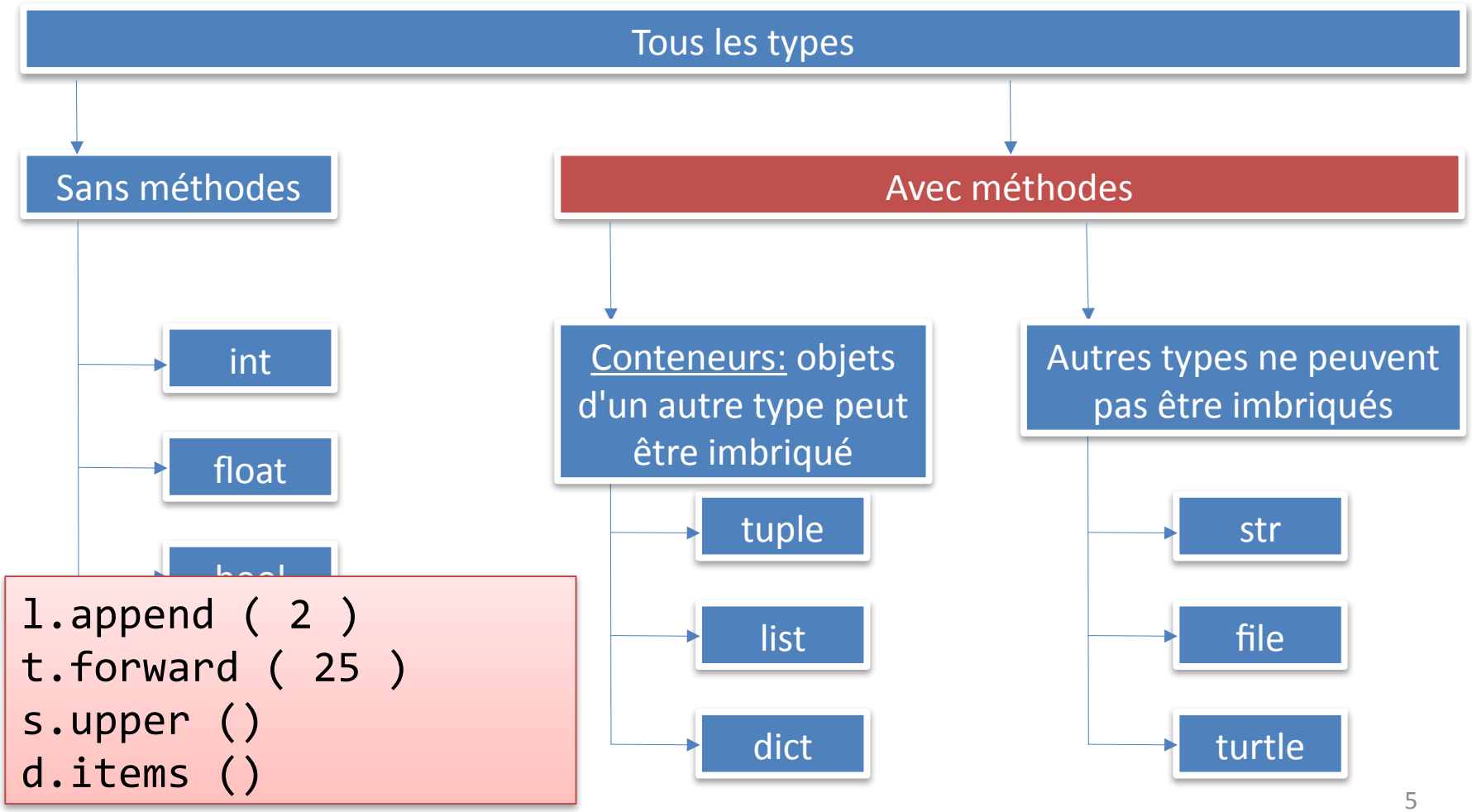
- Une variable est une référence vers un objet d'un certain type



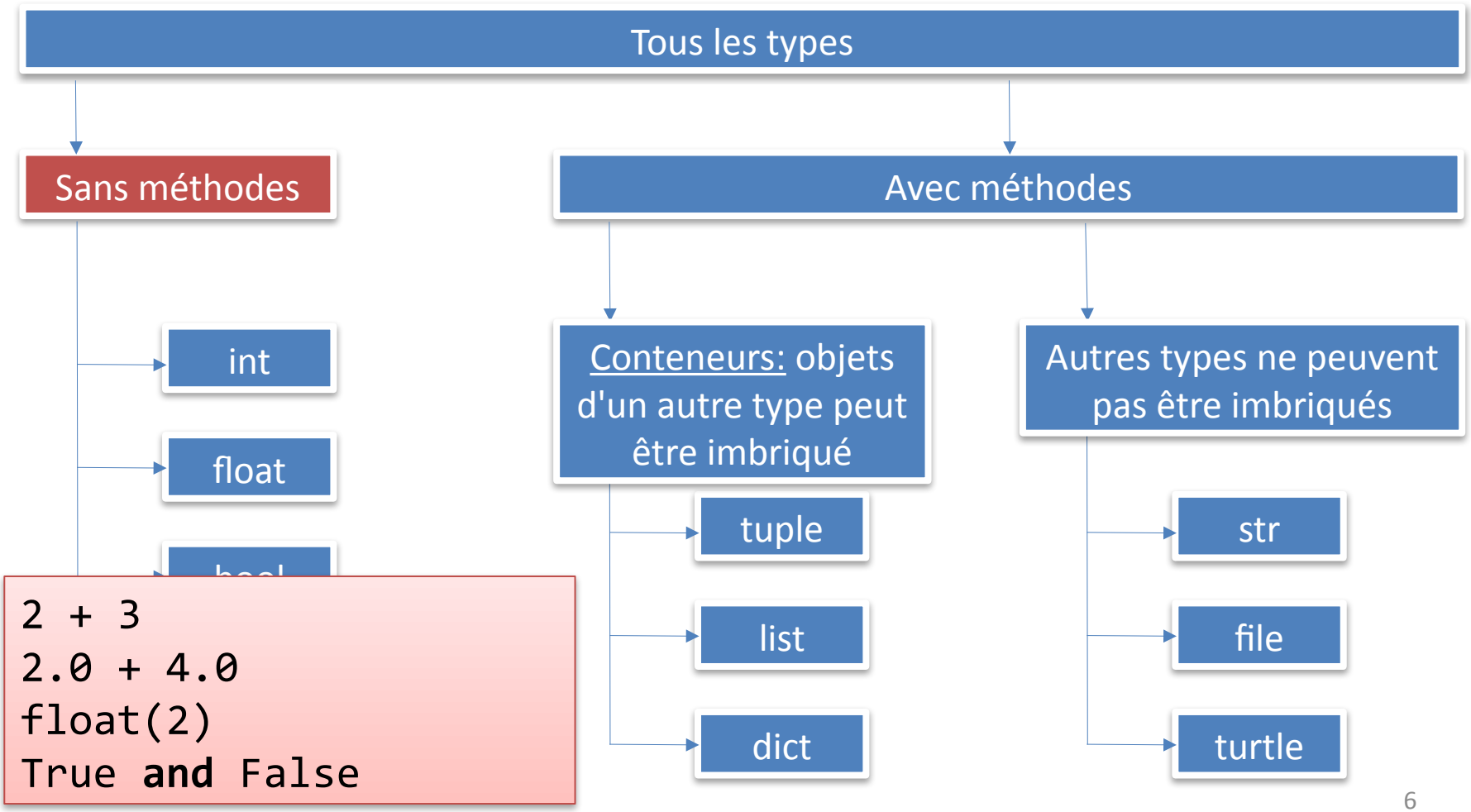
Types en Python
(jusque maintenant)



Types en Python
(jusque maintenant)

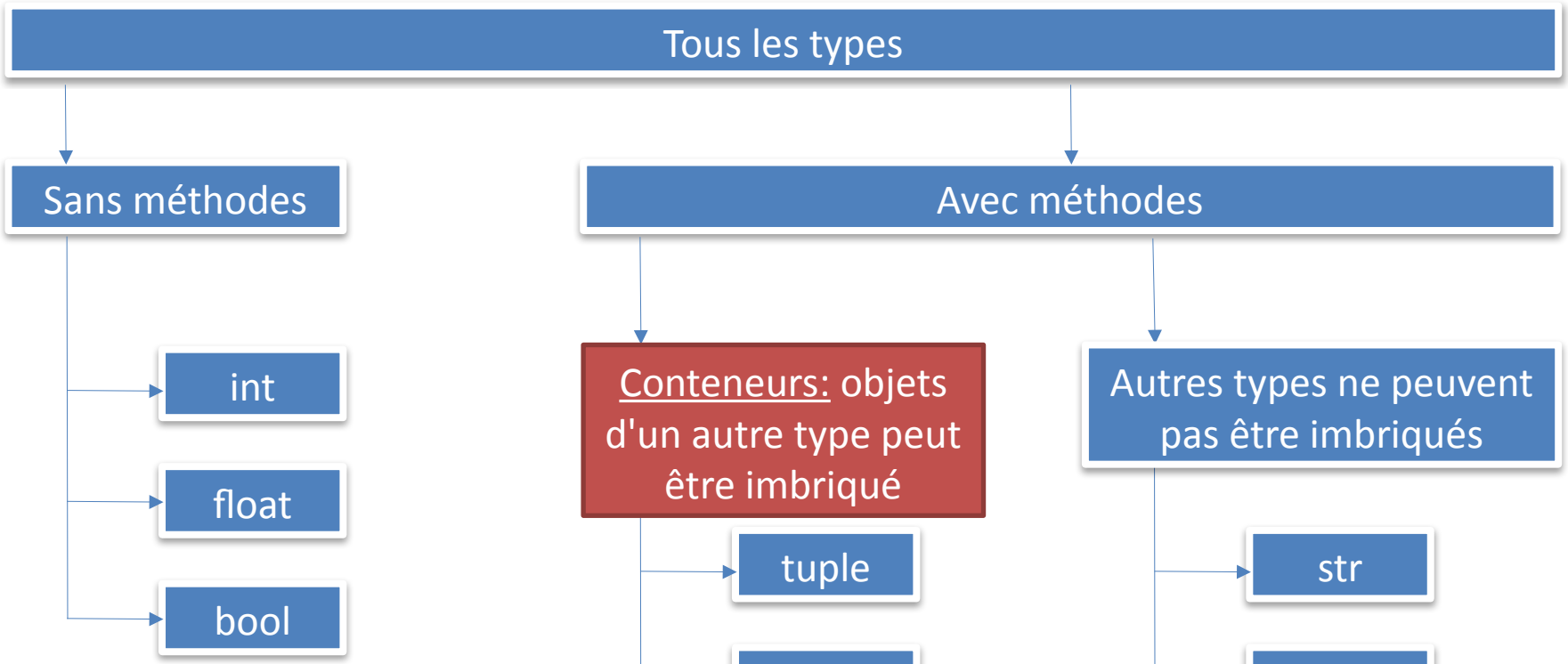


Types en Python
(jusque maintenant)



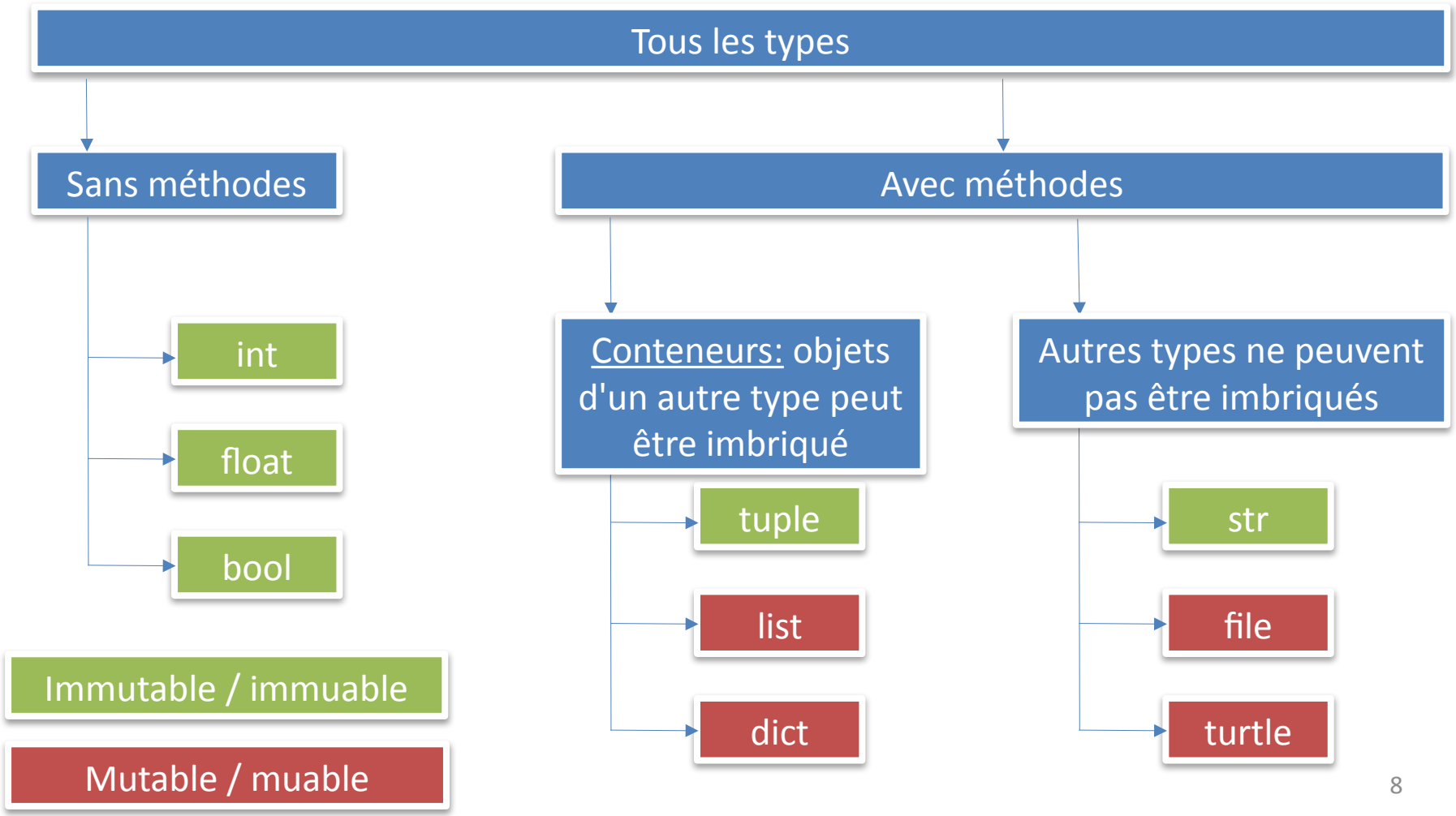

```
2 + 3
2.0 + 4.0
float(2)
True and False
```

Types en Python
(jusque maintenant)




```
l.append ( 2.0 )
t = ( 2.0, "Hi" )
d["Hi"] = "Bonjour"
```

Types en Python
(jusque maintenant)



Type Abstrait

- Un **type abstrait** est une spécification mathématique
  - d'un ensemble de données
  - de l'ensemble des opérations qu'on peut effectuer sur elles

Listes	Dictionnaires
Séquences ordonnées d'objets	Associations entre valeurs immuables et autres valeurs

`l.append ( 2.0 )`
`l[2] = 3.0`
~~`l["Hi"] = "Bonjour"`~~
~~`v= l.items ()`~~
	~~`l.append ( 2.0 )`~~
	`l[2] = 3.0`
	`l["Hi"] = "Bonjour"`
	`v = l.items ()`

Structure de Données

- Chaque type abstrait est implémenté par une **structure de données** dans la mémoire de l'ordinateur
- Le même type abstrait pourrait être implementé en utilisant différentes structures de données
  - Par exemple, Python pourrait implémenter un dictionnaire en utilisant une liste ordonnée ou une liste non ordonnée

Opérations sur Listes

- append
  - Ajoute un seul élément

```
l = [ 1, 2, 3 ]
l.append ( 4 )
```

  - N'ajoute pas plusieurs éléments

```
l = [ 1, 2, 3 ]
l.append ( 4, 5 )
```

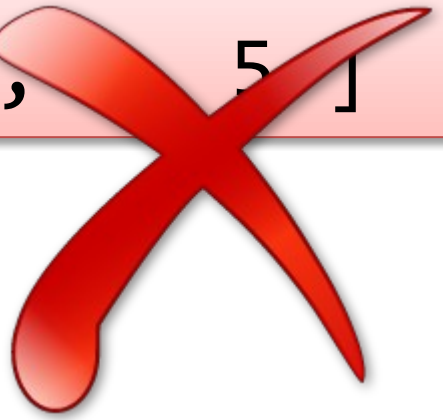
Opérations sur Listes

- append
  - On peut ajouter une liste!!

```
l = [ 1, 2, 3 ]
l.append ( [4, 5] )
```

  - Mais le résultat n'est pas

```
l == [ 1, 2, 3, 4, 5 ]
```



Opérations sur Listes

- append
  - On peut ajouter une liste!!

```
l = [ 1, 2, 3 ]
l.append ( [4, 5] )
```

  - Le résultat est

```
l == [ 1, 2, 3, [ 4, 5 ] ]
```

Liste imbriquée

Opérations sur Listes

- Affectations

```
l = [ 1, 2, 3 ]
l[1] = [ 4, 5, 6 ]
```

```
l == [ 1, [ 4, 5, 6 ], 3 ]
```

Opérations sur Listes

- Affectations

```
l = [ 1, 2, 3, 4 ]
len(l) = 2
```

```
l == [ 1, 2 ]
```

Opérations sur Listes

- En Python on n'écrit **jamais**

```
fonction(paramètres) = quelque chose
```

Créer des Dictionnaires Complexes

- Suppose que pour une liste de strings comme

```
lines = [“aabb”,”ab”,”bb”,”aa”]
```

```
aabb
ab
bb
aa
```

- On souhaite créer

```
letters = { "a": [0,0,1,3,3], "b": [0,0,1,2,2] }
```

Créer des Dictionnaires Complexes

- Une approache est d'ajouter les strings itérativement à un dictionnaire

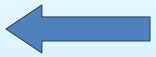
```
aabb
ab
bb
aa
```

```
letters = { }
```

Créer des Dictionnaires Complexes

- Une approache est d'ajouter les strings itérativement à un dictionnaire

```
aabb
ab
bb
aa
```



```
letters = { "a": [0,0], "b": [0,0] }
```

Créer des Dictionnaires Complexes

- Une approache est d'ajouter les strings itérativement à un dictionnaire

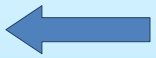
```
aabb
ab
bb
aa
```

```
letters = { "a": [0,0,1], "b": [0,0,1] }
```

Créer des Dictionnaires Complexes

- Une approache est d'ajouter les strings itérativement à un dictionnaire

```
aabb
ab
bb
aa
```

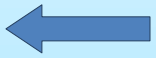


```
letters = { "a": [0,0,1], "b": [0,0,1,2,2] }
```

Créer des Dictionnaires Complexes

- Une approache est d'ajouter les strings itérativement à un dictionnaire

```
aabb
ab
bb
aa
```



```
letters = { "a": [0,0,1,3,3], "b": [0,0,1,2,2] }
```

Créer des Dictionnaires Complexes

- Remplir le dictionnaire ligne par ligne

```
def create_dict(lines) :
  letters = {}
  for i in range(len(lines)) :
    Ajoute ligne lines[i] avec index i à letters
  return letters
```

Créer des Dictionnaires Complexes

- Remplir le dictionnaire ligne par ligne

```
def create_dict(lines) :
  letters = {}
  for i in range(len(lines)) :
    add_line ( lines[i], i, letters )
  return letters
```

Créer des Dictionnaires Complexes

- Remplir le dictionnaire ligne par ligne

```
def create_dict(lines) :
  letters = {}
  for i in range(len(lines)) :
    add_line ( lines[i], i, letters )
  return letters
```

```
def add_line(line,number,letters) :
  for car in line :
    letters[car].append ( number )
```

Créer des Dictionnaires Complexes

- Remplir le dictionnaire ligne par ligne

```
def create_dict(lines) :
  letters = {}
  for i in range(len(lines)) :
    add_line ( lines[i], i, letters )
  return letters
```

```
def add_line(line,number,letters) :
  for car in line :
    letters[car].append ( number )
```

```
print(create_dict(["aabb"]))
```

Créer des Dictionnaires Complexes

- Remplir le dictionnaire ligne par ligne

```
def create_dict(lines) :
  letters = {}
  for i in range(len(lines)) :
    add_line ( lines[i], i, letters )
  return letters
```

```
def add_line(line,number,letters) :
  for car in line :
    if car not in letters:
      letters[car] = [number]
    else:
      letters[car].append ( number )
```

Créer des Dictionnaires Complexes

- Remplir le dictionnaire ligne par ligne

```
def create_dict(lines) :
  letters = {}
  for i in range(len(lines)) :
    add_line ( lines[i], i, letters )
  return letters
```

```
def add_line(line,number,letters) :
  for car in line :
    letters.get(car,[]).append ( number )
```

Créer des Dictionnaires Complexes

- Remplir le dictionnaire ligne par ligne

```
def create_dict(lines) :
  letters = {}
  for i in range(len(lines)) :
    add_line ( lines[i], i, letters )
  return letters
```

```
def add_line(line,number,letters) :
  for car in line :
    letters.get(car,[]).append ( number )
```

Créer des Dictionnaires Complexes

- Remplir le dictionnaire ligne par ligne

```
def create_dict(lines) :
  letters = {}
  for i in range(len(lines)) :
    add_line ( lines[i], i, letters )
  return letters
```

```
def add_line(line,number,letters) :
  for car in line :
    letters.setdefault(car,[]).append(number)
```

Copie vs Affectation

- Résultat d'Affectation ?

```
lines = [ “aabb” ]
letters = create_dict ( lines )
letters2 = letters
add_line(“ba”,1,letters)
print(letters)
print(letters2)
```

Copie vs Affectation

- Résultat d'une copie?

```
import copy

lines = [ “aabb” ]
letters = create_dict ( lines )
letters2 = copy.deepcopy(letters)
add_line(“ba”,1,letters)
print(letters)
print(letters2)
```