Informatique 1
Introduction à la programmation

Mission 6 : restructuration

Kim Mens Siegfried Nijssen Charles Pecheur

Lire Fichiers de Texte

approche simple

B	r	u	x	e	l
l	e	s	\n	M	o
n	s	\n	L	o	u
v	a	i	n	-	l
a	-	N	e	u	v
e	\n	EOF			

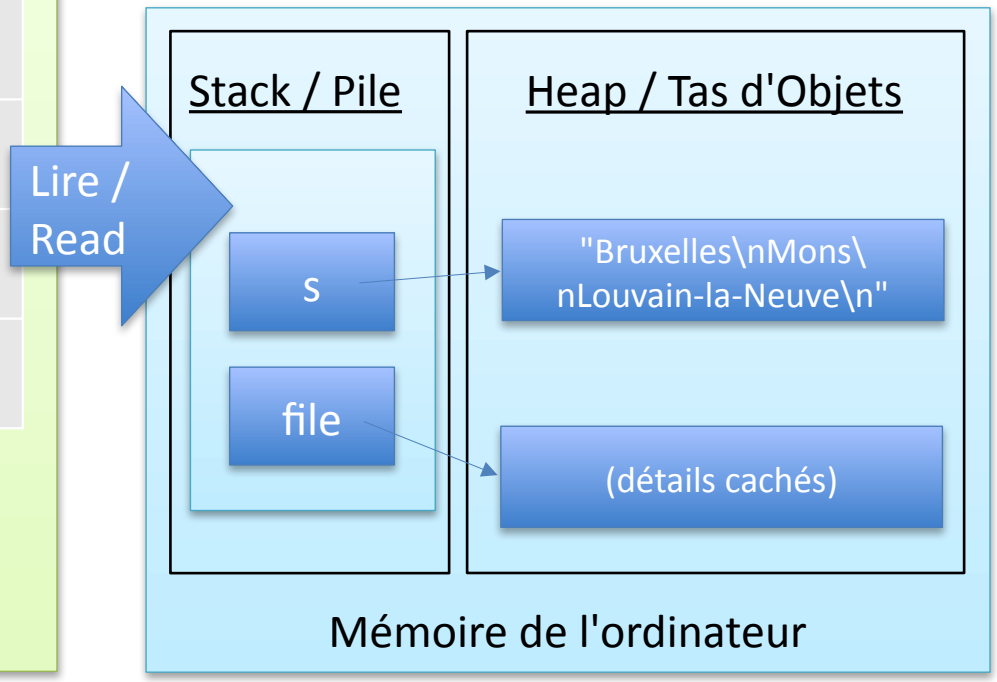
`file.txt`

Disque dur de l'ordinateur

(1) Ouverture

```
file = open ( "file.txt", "r" )
s = file.read ()  (2) Traitement
file.close ()  (3) Fermeture
```

Lire / Read



Lire Fichiers de Texte

```
file = open ( "file.txt", "r" )
s = file.read ()
file.close ()
```

```
file = open ( "file.txt", "r" )
l = file.readlines ()
file.close ()
```

```
file = open ( "file.txt", "r" )
for line in file:
  print ( line )
file.close ()
```

Lire Fichiers de Texte

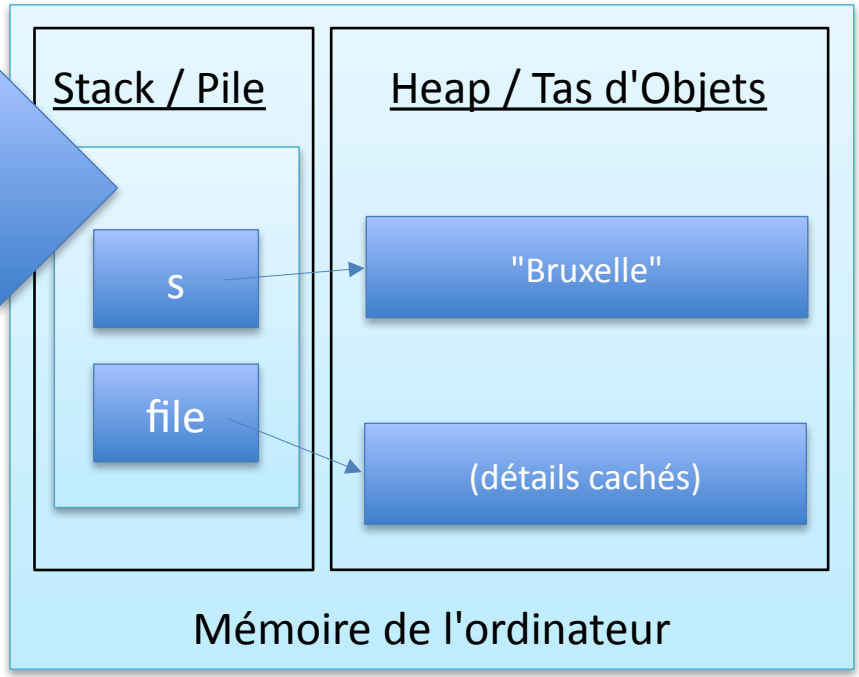
B	r	u	x	e	l
l	e	s	\n	M	o
n	s	\n	L	o	u
v	a	i	n	-	l
a	-	N	e	u	v
e	\n	EOF			

file.txt

Disque dur de l'ordinateur

```
file = open ( "file.txt", "r" )
s = file.read ( 8 )
file.close ()
```

Lire / Read



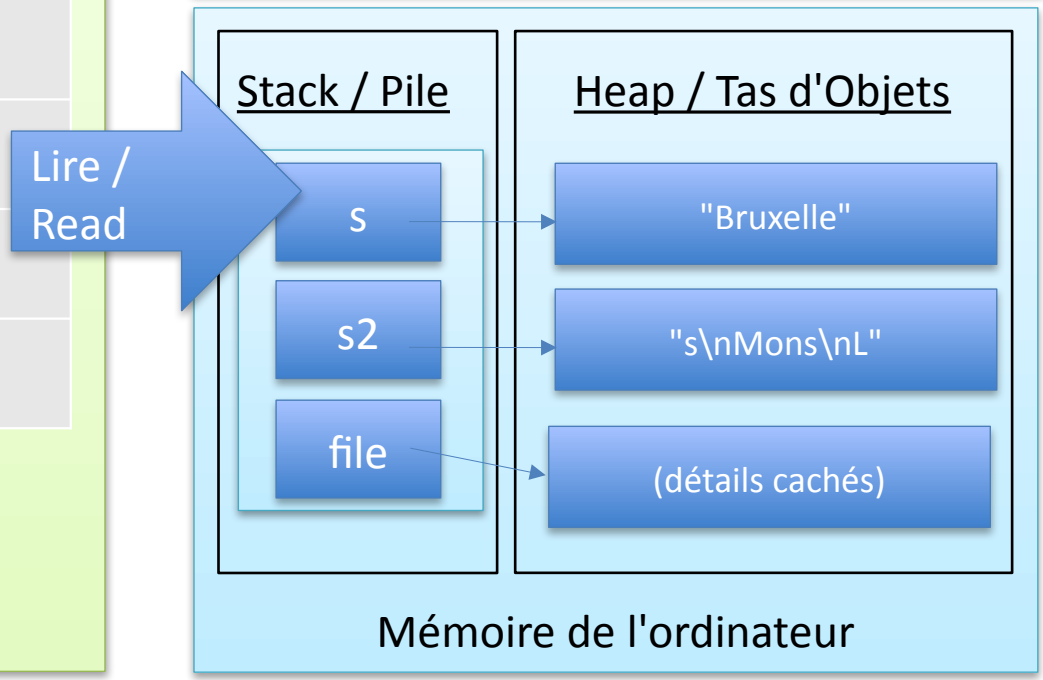
Lire Fichiers de Texte

B	r	u	x	e	l
l	e	s	\n	M	o
n	s	\n	L	o	u
v	a	i	n	-	l
a	-	N	e	u	v
e	\n	EOF			

file.txt

Disque dur de l'ordinateur

```
file = open ( "file.txt", "r" )
s = file.read ( 8 )
s2 = file.read ( 8 )
file.close ()
```



Lire Fichiers de Texte

```
file = open ( "file.txt", "r" )
s = file.read ()
file.close ()
```

```
file = open ( "file.txt", "r" )
l = file.readlines ()
file.close ()
```

```
file = open ( "file.txt", "r" )
s = file.read ( 8 )
file.close ()
```

```
file = open ( "file.txt", "r" )
for line in file:
  print ( line )
file.close ()
```

Lire Fichiers de Texte

Aucune distinction entre les lignes

```
file = open ( "file.txt", "r" )
s = file.read ()
file.close ()
```

```
file = open ( "file.txt", "r" )
s = file.read ( 8 )
file.close ()
```

Distinguer les lignes

```
file = open ( "file.txt", "r" )
l = file.readlines ()
file.close ()
```

```
file = open ( "file.txt", "r" )
for line in file:
  print ( line )
file.close ()
```

Lire Fichiers de Texte

Lire tout le fichier en mémoire

```
file = open ( "file.txt", "r" )
s = file.read ()
file.close ()
```

```
file = open ( "file.txt", "r" )
l = file.readlines ()
file.close ()
```

```
file = open ( "file.txt", "r" )
s = file.read ( 8 )
file.close ()
```

```
file = open ( "file.txt", "r" )
for line in file:
  print ( line )
file.close ()
```

Lire seulement une partie en mémoire

Lire Fichiers de Texte

Fermer un fichier est nécessaire pour assurer que le fichier est modifié

Fermer le fichier explicitement

```
file = open ( "file.txt", "w" )
file.write ( s )
file.close ()
```

Lire Fichiers de Texte

Mauvais exemple: pourquoi?

Fermer le fichier explicitement

```
def has_louvain ( filename ):
  file = open ( filename, "r" )
  for line in file:
    if line == "Louvain\n":
      return True
  file.close ()
  return False
```

Lire Fichiers de Texte

Corrigé

Fermer le fichier explicitement

```
def has_louvain ( filename ):
  file = open ( filename, "r" )
  for line in file:
    if line == "Louvain\n":
      file.close ()
      return True
  file.close ()
  return False
```

Lire Fichiers de Texte

Fermer le fichier explicitement

```
file = open ( "file.txt", "r" )
s = file.read ()
file.close ()
```

Fermer le fichier implicitement

```
with open ( "file.txt", "r" ) as file:
  s = file.read ()
```

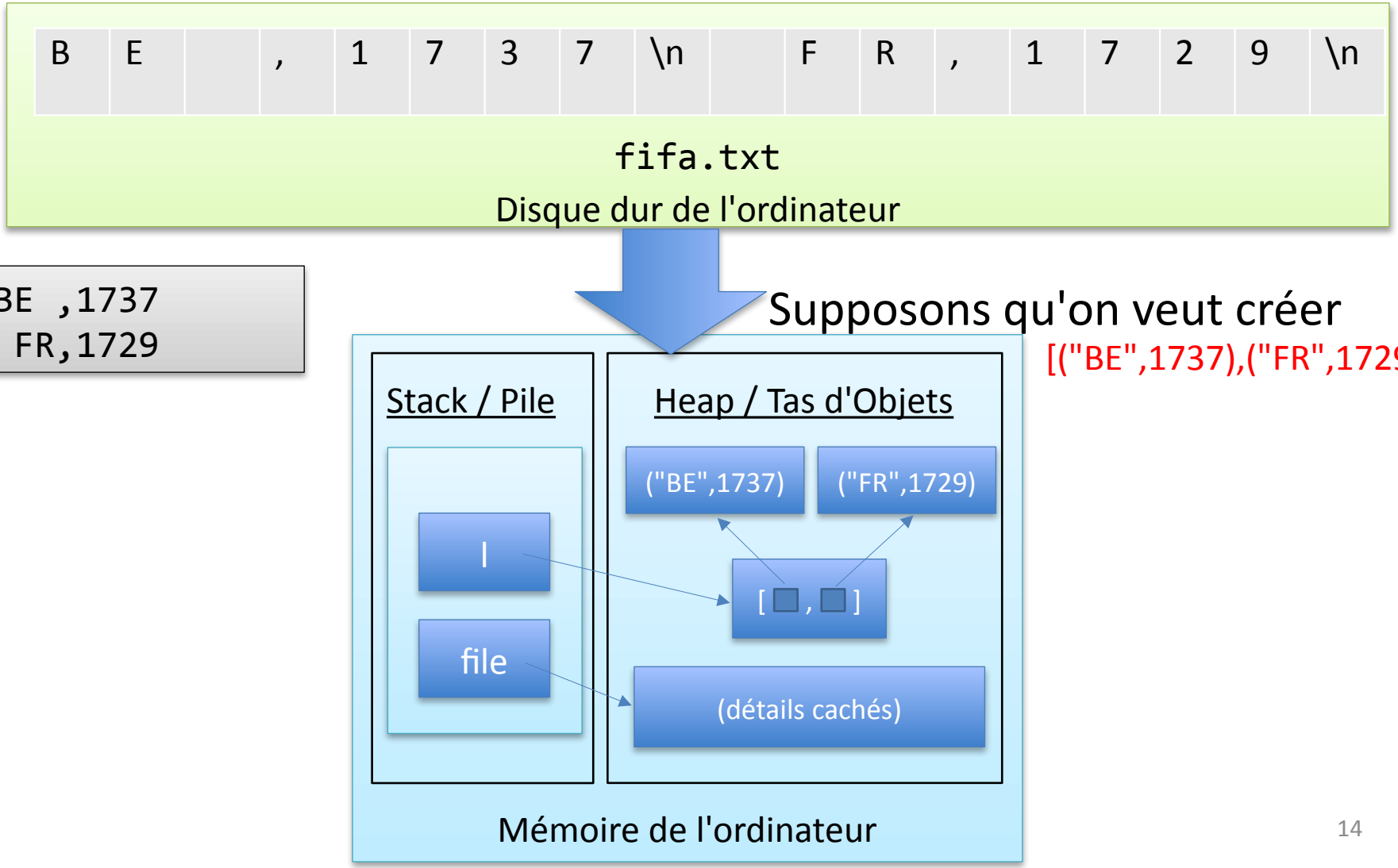
Lire Fichiers de Texte

Corrigé

Fermer le fichier implicitement

```
def has_louvain ( filename ):
  with open ( filename, "r" )\
       as file:
    for line in file:
      if line == "Louvain\n":
        return True
  return False
```

Lire Fichiers de Texte

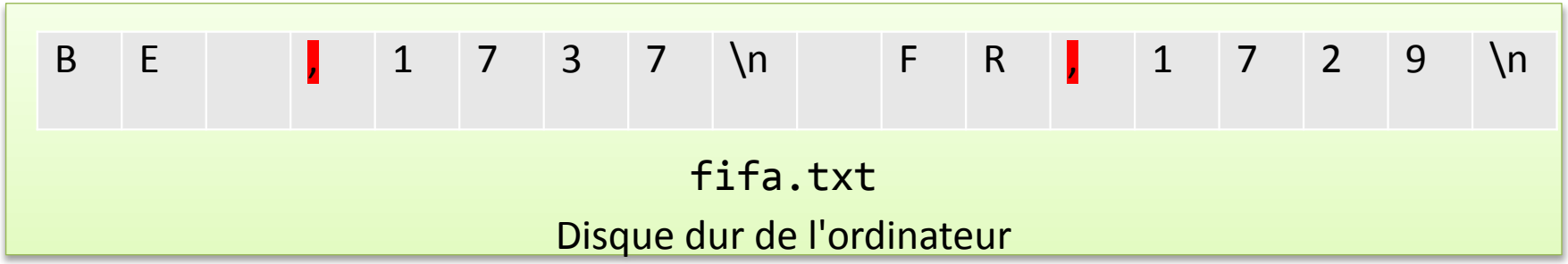


Lire Fichiers de Texte

readlines ()

[("BE",1737),("FR",1729)]

Lire Fichiers de Texte

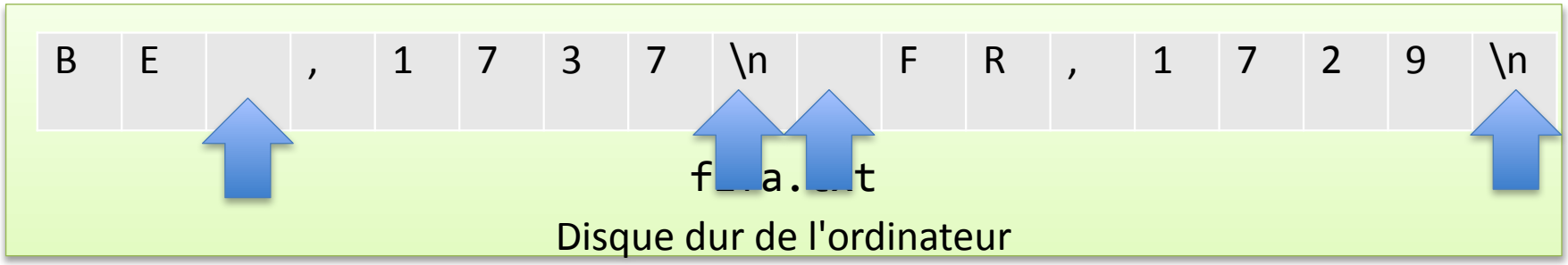


readlines ()

split(",")

[("BE",1737),("FR",1729)]

Lire Fichiers de Texte

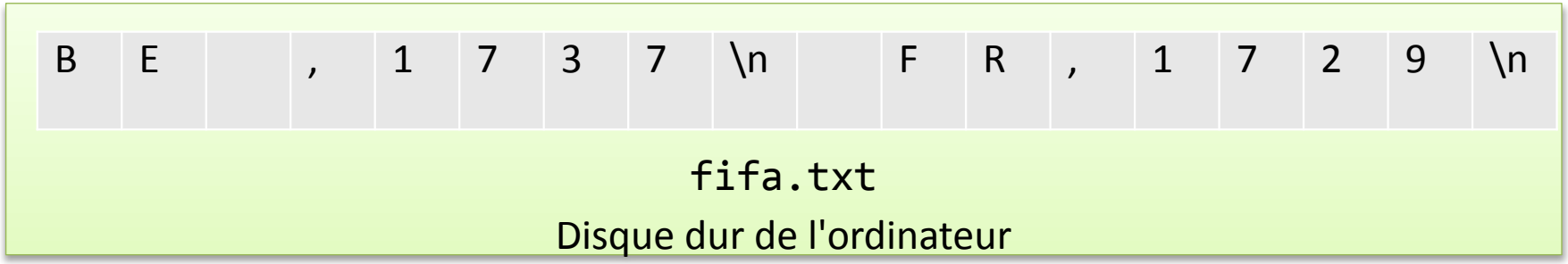


readlines ()

split(",") strip()

[("BE",1737),("FR",1729)]

Lire Fichiers de Texte



readlines ()

split(",") strip()

tuple() int()

[("BE",1737),("FR",1729)]

Lire Fichiers de Texte

```
l = []
with open ( "fifa.txt", "r" ) \
     as file:
  for line in file.readlines ():
     line = line.strip ()
     line = line.split (",")
     l.append((line[0],int(line[1])))
```

```
l = []
with open ( "fifa.txt", "r" ) \
     as file:
  for line in file.readlines ():
     line = line.split (",")
     line = line.strip ()
     l.append ((line[0],int(line[1])))
```

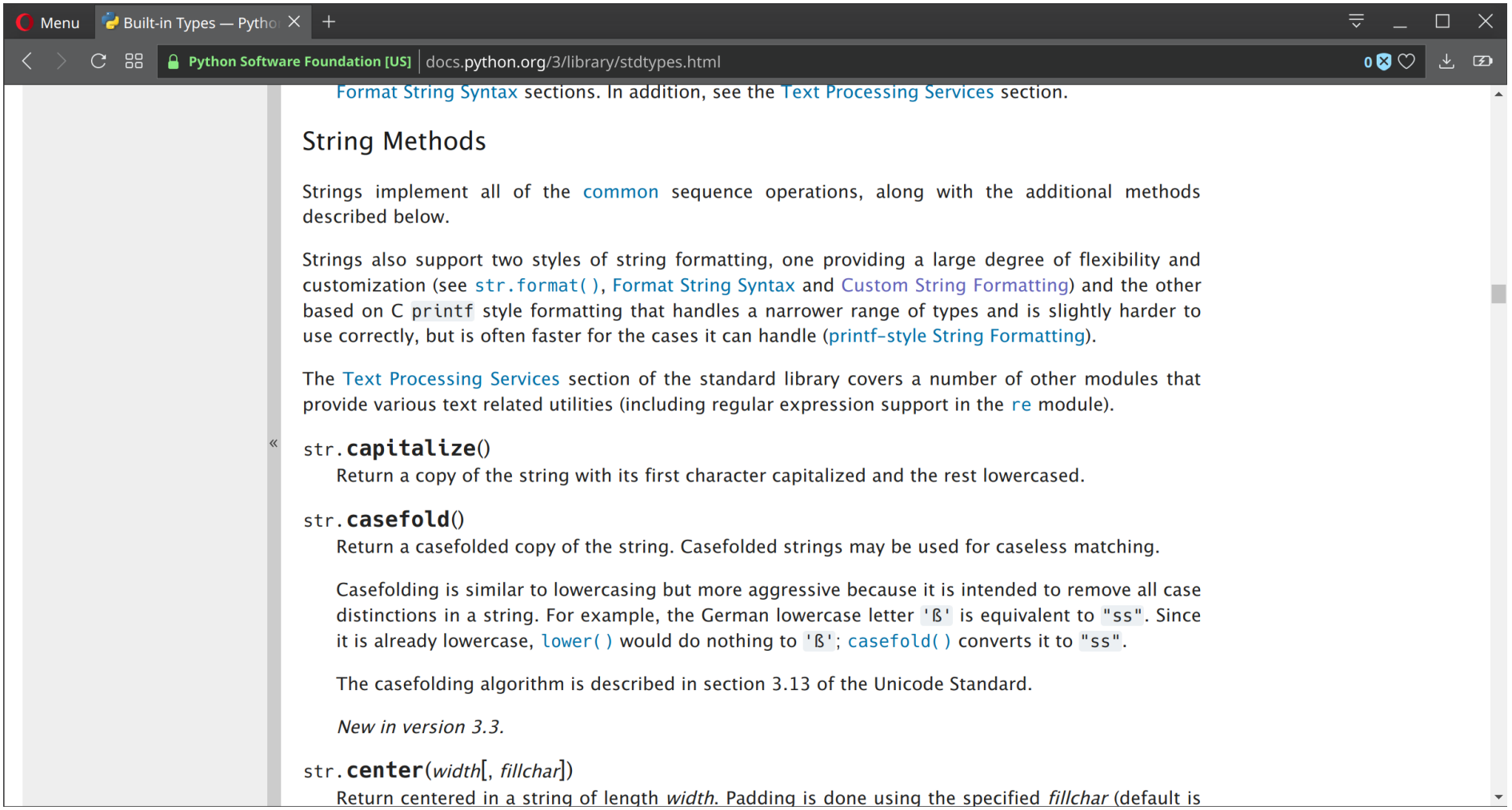
```
l = []
with open ( "fifa.txt", "r" ) \
     as file:
  for line in file.readlines ():
     line = line.split (",")
     l.append ((line[0].strip(), \
                int(line[1].strip())))
```

```
l = []
with open ( "fifa.txt", "r" ) \
     as file:
  for line in file.readlines ():
     line = line.split (",")
     line = line.strip ()
     l.append ((line[0].strip(), \
                int(line[1].strip())))
```

readlines () split(",") strip() int() tuple()

Méthodes sur Chaînes de Caractères

https://docs.python.org/3/library/stdtypes.html



Erreurs

- Comment éviter que le programme s'arrête à cause d'un problème de lecture du fichier?

```
name = input ( "Provide a file name: " )
file = open ( name, "r" )
for line in file:
  print(line)
file.close ()
```

```
Provide a file name: nexistepas.txt
Traceback (most recent call last):
  File "test.py", line 2, in <module>
    file = open ( name, "r" )
FileNotFoundError: [Errno 2] No such
file or directory: 'nexistepas.txt'
```

Fichiers

Approche recommandée

```
name = input ( "Provide a file name: " )
try:
   with open( name, "r" ) as file:
        for line in file:
             print(line)

except:
   print ( "Error reading file" )
```

(1) Ouverture
(3) Fermeture
(2) Traitement
(4)Exceptions

Exceptions

On peut utiliser **`except`** pour intercepter beaucoup d'erreurs différentes

```
num = input ( "Provide a number: " )
try:
  print ( 1 / int(num) )
except:
  print ( "Error doing calculation" )
```

Gestion des Exceptions

- Python **lève** une **exception** s'il y a une erreur
- On peut **intercepter** plus exceptions différentes en utilisant un bloc `try… except`

```
name = input ( "Provide a file name: " )
try:
   file = open ( name, "r" )
   for line in file:
     print(line)
   file.close ()
except:
   print ( "Error reading file" )
```

(1) Ouverture
(2) Traitement
(3) Fermeture
(4) Exceptions

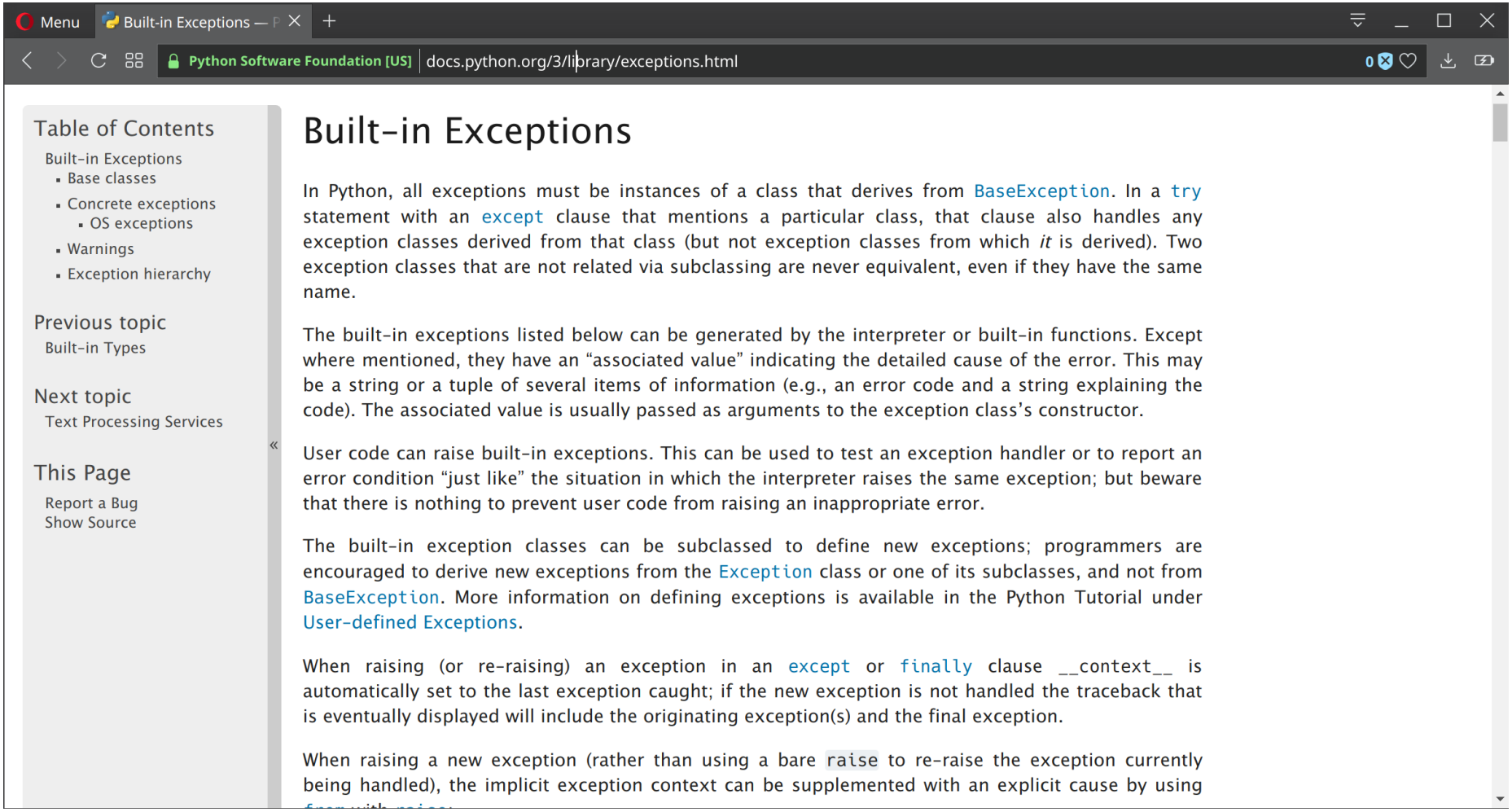
Gestion des Exceptions

- Intercepter des erreurs spécifiques :

```
num = input ( "Provide a number: " )
try:
  print ( 1 / int(num) )
except ValueError:
  print ( "No integer provided" )
except ZeroDivisionError:
  print ( "Division by zero" )
```

Autres Exceptions

https://docs.python.org/3/library/exceptions.html



Autres Exceptions

- *exception* **`ValueError`**

  Raised when an operation or function receives an argument that has the right type but an inappropriate value, and the situation is not described by a more precise exception such as `IndexError`.

- *exception* **`IndexError`**

  Raised when a sequence subscript is out of range. (Slice indices are silently truncated to fall in the allowed range; if an index is not an integer, `TypeError` is raised.)

- *exception* **`OSError`**([*arg*])

  This exception is raised when a system function returns a system-related error, including I/O failures such as “file not found” or “disk full” (not for illegal argument types or other incidental errors).

Gestion des Exceptions

Combiner des exceptions spécifiques avec un except défaut

```
num = input ( "Provide a number: " )
try:
  print ( 1 / int(num) )
except ValueError:
  print ( "No integer provided" )
except:
  print ( "Another error occurred" )
```

```
Provide a number: age
No integer provided
```

Gestion des Exceptions

Combiner des exceptions spécifiques avec un except défaut

```
num = input ( "Provide a number: " )
try:
  print ( 1 / int(num) )
except:
  print ( "Another error occurred" )
except ValueError:
  print ( "No integer provided" )
```

```
SyntaxError: default 'except:' must
be last
```

Exploiter les Exceptions

```
def is_number ( text, n ):
    try:
        value = int(text)
        return value >= 1 and value <= n
    except ValueError:
        return False
```

retourne True si *text* représente un nombre entier *i* tel que *1 <= i <= n*

Éviter les exceptions?

```
def is_number ( text, n ):
    if text.isdigit ():
        value = int(text)
        return value >= 1 and value <= n
    else:
        return False
```

Programmation défensive

Pas nécessaire en Python, les exceptions sont plus performantes que dans d'autres langages

Flux d'exécution?

B	r	u	x	e	l	l	e	s	\n

names.txt

Disque dur de l'ordinateur

```
def numbers(filename):
  try:
    with open ( filename, "r" ) as file:
      numbers = []
      for line in file:
        numbers.append ( int(line) )
      return numbers
  except ValueError:
    return []

try:
  v = numbers ( "names.txt" )
  print ( "Read numbers: ", v )
except:
  print ( "Error" )
```

Flux d'exécution?

B	r	u	x	e	l	l	e	s	\n

names.txt

Disque dur de l'ordinateur

```
def numbers(filename):
  try:
    with open ( filename, "r" ) as file:
      numbers = []
      for line in file:
        numbers.append ( int(line) )
      return numbers
  except ValueError:
    return []

try:
  v = numbers ( "names.txt" )
  print ( "Read numbers: ", v )
except:
  print ( "Error" )
```

Flux d'exécution?

B	r	u	x	e	l	l	e	s	\n

names.txt

Disque dur de l'ordinateur

```
def numbers(filename):
  try:
    with open ( filename, "r" ) as file:
      numbers = []
      for line in file:
        numbers.append ( int(line) )
      return numbers
  except ValueError:
    return []

try:
  v = numbers ( "names.txt" )
  print ( "Read numbers: ", v )
except:
  print ( "Error" )
```

Flux d'exécution?

B	r	u	x	e	l	l	e	s	\n

names.txt

Disque dur de l'ordinateur

```
def numbers(filename):
  try:
    with open ( filename, "r" ) as file:
      numbers = []
      for line in file:
        numbers.append ( int(line) )
      return numbers
  except ValueError:
    return []

try:
  v = numbers ( "names.txt" )
  print ( "Read numbers: ", v )
except:
  print ( "Error" )
```

Flux d'exécution?

B	r	u	x	e	l	l	e	s	\n

names.txt

Disque dur de l'ordinateur

```
def numbers(filename):
  try:
    with open ( filename, "r" ) as file:
      numbers = []
      for line in file:
        numbers.append ( int(line) )
      return numbers
  except ValueError:
    return []

try:
  v = numbers ( "names.txt" )
  print ( "Read numbers: ", v )
except:
  print ( "Error" )
```

Flux d'exécution?

B	r	u	x	e	l	l	e	s	\n

names.txt

Disque dur de l'ordinateur

```
def numbers(filename):
  try:
    with open ( filename, "r" ) as file:
      numbers = []
      for line in file:
        numbers.append ( int(line) )
      return numbers
  except ValueError:
    return []

try:
  v = numbers ( "names.txt" )
  print ( "Read numbers: ", v )
except:
  print ( "Error" )
```

Flux d'exécution?

B	r	u	x	e	l	l	e	s	\n

names.txt

Disque dur de l'ordinateur

```
def numbers(filename):
  try:
    with open ( filename, "r" ) as file:
      numbers = []
      for line in file:
        numbers.append ( int(line) )
      return numbers
  except ValueError:
    return []

try:
  v = numbers ( "names.txt" )
  print ( "Read numbers: ", v )
except:
  print ( "Error" )
```

Flux d'exécution?

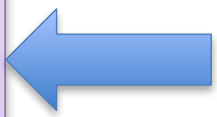
B	r	u	x	e	l	l	e	s	\n

names.txt

Disque dur de l'ordinateur

```
def numbers(filename):
  try:
    with open ( filename, "r" ) as file:
      numbers = []
      for line in file:
        numbers.append ( int(line) )
      return numbers
  except ValueError:
    return []

try:
  v = numbers ( "names.txt" )
  print ( "Read numbers: ", v )
except:
  print ( "Error" )
```



Flux d'exécution?

`names.txt` n'existe pas

Disque dur de l'ordinateur

```
def numbers(filename):
  try:
    with open ( filename, "r" ) as file:
      numbers = []
      for line in file:
        numbers.append ( int(line) )
      return numbers
  except ValueError:
    return []

try:
  v = numbers ( "names.txt" )
  print ( "Read numbers: ", v )
except:
  print ( "Error" )
```

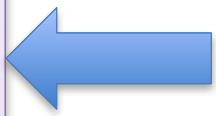
Flux d'exécution?

`names.txt` n'existe pas

Disque dur de l'ordinateur

```
def numbers(filename):
  try:
    with open ( filename, "r" ) as file:
      numbers = []
      for line in file:
        numbers.append ( int(line) )
      return numbers
  except ValueError:
    return []

try:
  v = numbers ( "names.txt" )
  print ( "Read numbers: ", v )
except:
  print ( "Error" )
```



Flux d'exécution?

`names.txt` n'existe pas

Disque dur de l'ordinateur

```
def numbers(filename):
  try:
    with open ( filename, "r" ) as file:
      numbers = []
      for line in file:
        numbers.append ( int(line) )
      return numbers
  except ValueError:
    return []

try:
  v = numbers ( "names.txt" )
  print ( "Read numbers: ", v )
except:
  print ( "Error" )
```