



Informatique 1

Introduction à la programmation

Mission 6 : restructuration

Kim Mens Siegfried Nijssen Charles Pecheur

Lire Fichiers de Texte

approche simple

(1) Ouverture

```
file = open ( "file.txt", "r" )
```

```
s = file.read ()
```

```
file.close ()
```

(2) Traitement

(3) Fermeture

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

Lire /
Read

Stack / Pile

s

file

Heap / Tas d'Objets

"Bruxelles\nMons\
nLouvain-la-Neuve\n"

(détails cachés)

Mémoire de l'ordinateur

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

Stack / Pile

Heap / Tas d'Objets

s

"Bruxelle"

file

(détails cachés)

Mémoire de l'ordinateur

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 /
Read

Stack / Pile

s

s2

file

Heap / Tas d'Objets

"Bruxelle"

"s\nMons\nL"

(détails cachés)

Mémoire de l'ordinateur

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 ()
```



Like

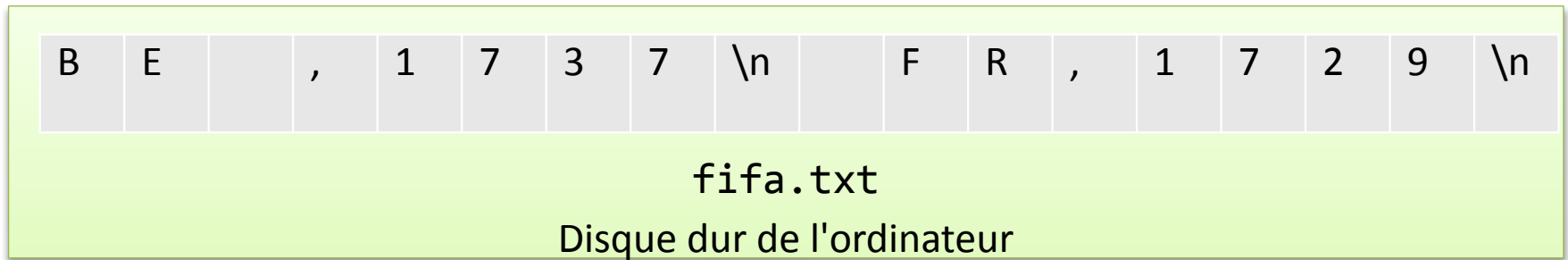
Lire Fichiers de Texte

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
```

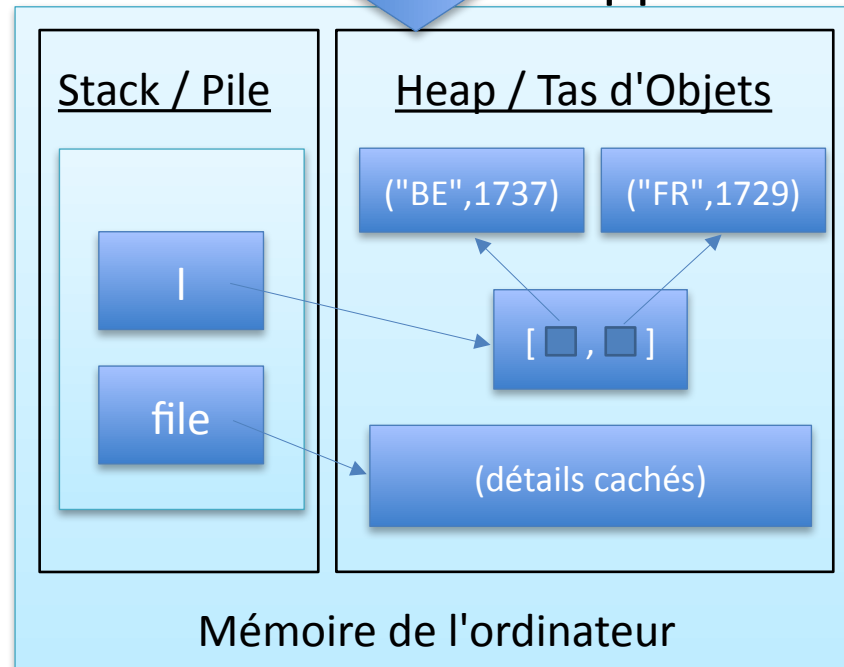
Lire Fichiers de Texte



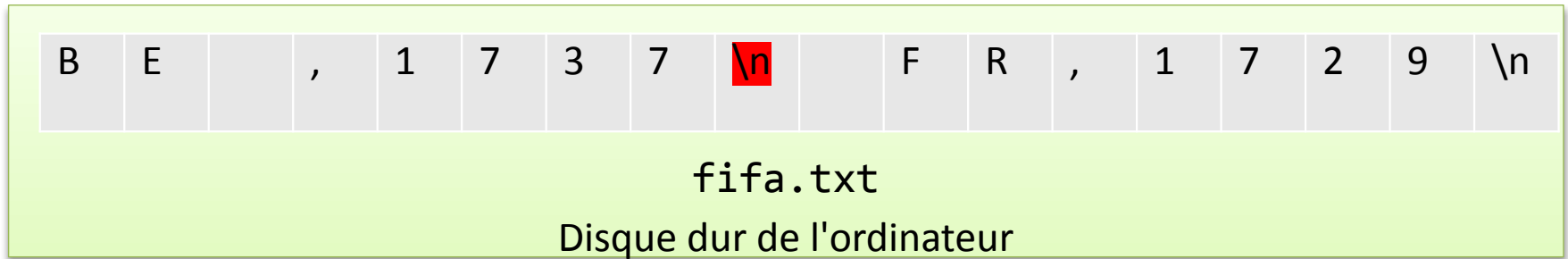
BE ,1737
FR,1729

Supposons qu'on veut créer

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



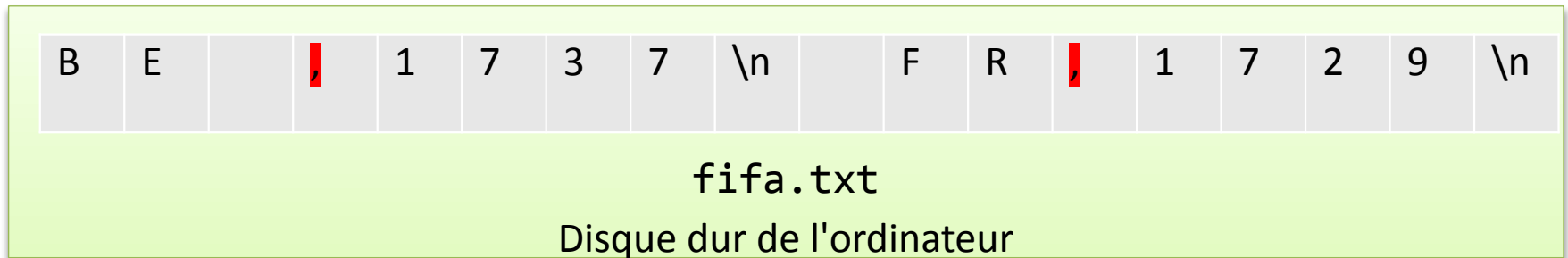
Lire Fichiers de Texte



`readlines ()`

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

Lire Fichiers de Texte

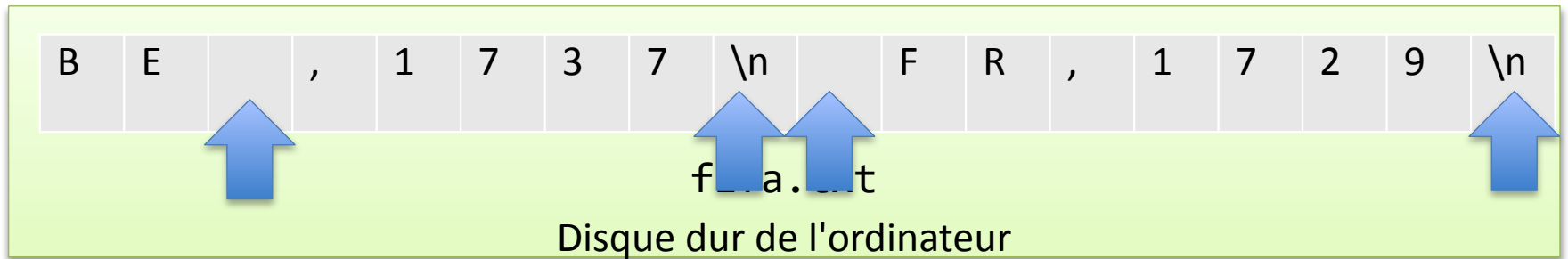


`readlines ()`

`split(",")`

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

Lire Fichiers de Texte



`readlines ()`

`split(",")`

`strip()`

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

Lire Fichiers de Texte

B	E		,	1	7	3	7	\n		F	R	,	1	7	2	9	\n
---	---	--	---	---	---	---	---	----	--	---	---	---	---	---	---	---	----

fifa.txt

Disque dur de l'ordinateur

`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])))
```

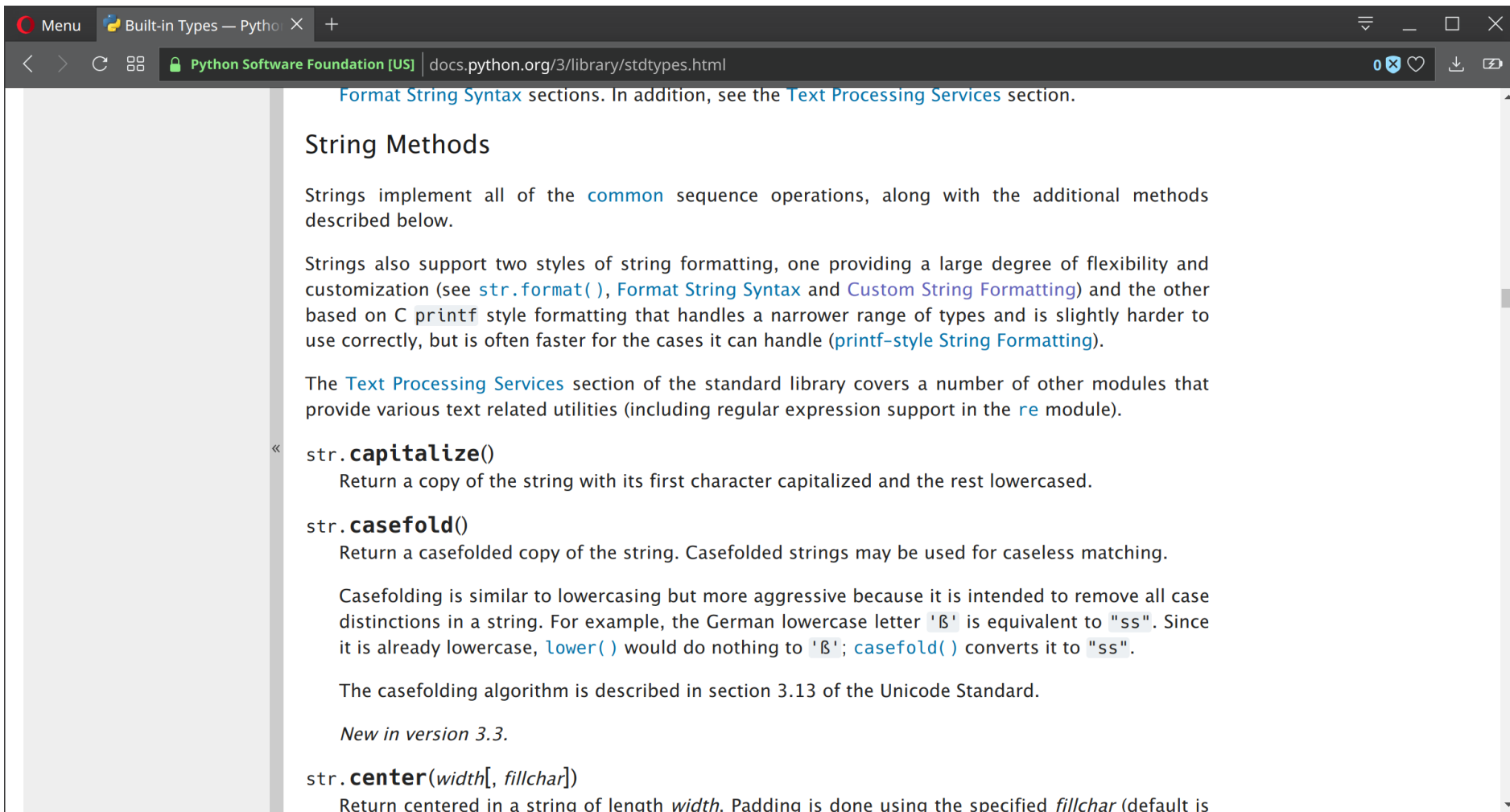
```
l = []  
with open ( "fifa.txt", "r" ) \  
    as file:  
    for line in file.readlines ():  
        line = line.split (",")  
        l.append ((line[0].strip(), \  
                    int(line[1].strip())))
```

```
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>



Format String Syntax sections. In addition, see the [Text Processing Services](#) section.

String Methods

Strings implement all of the [common](#) sequence operations, along with the additional methods described below.

Strings also support two styles of string formatting, one providing a large degree of flexibility and customization (see [str.format\(\)](#), [Format String Syntax](#) and [Custom String Formatting](#)) and the other based on C `printf` style formatting that handles a narrower range of types and is slightly harder to use correctly, but is often faster for the cases it can handle ([printf-style String Formatting](#)).

The [Text Processing Services](#) section of the standard library covers a number of other modules that provide various text related utilities (including regular expression support in the [re](#) module).

« **str.capitalize()**
Return a copy of the string with its first character capitalized and the rest lowercased.

str.casefold()
Return a casefolded copy of the string. Casefolded strings may be used for caseless matching.

Casefolding is similar to lowercasing but more aggressive because it is intended to remove all case distinctions in a string. For example, the German lowercase letter 'ß' is equivalent to "ss". Since it is already lowercase, `lower()` would do nothing to 'ß'; `casefold()` converts it to "ss".

The casefolding algorithm is described in section 3.13 of the Unicode Standard.

New in version 3.3.

str.center(width[, fillchar])
Return centered in a string of length *width*. Padding is done using the specified *fillchar* (default is

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
- (2) Traitement
- (3) Fermeture
- (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

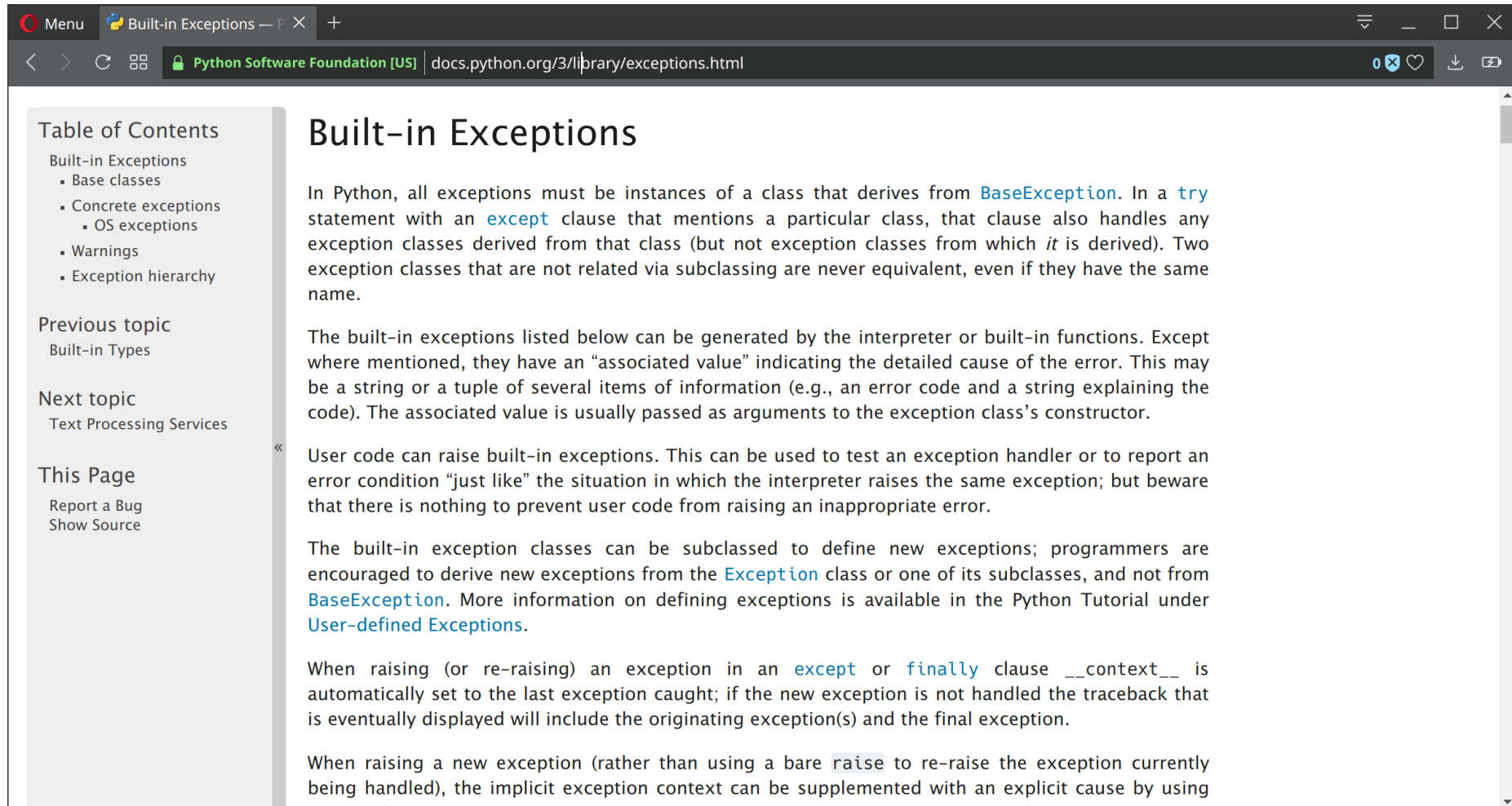
Gestion des Exceptions

- 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>



The image shows a screenshot of a web browser displaying the Python documentation page for Built-in Exceptions. The browser's address bar shows the URL `docs.python.org/3/library/exceptions.html`. The page content includes a table of contents on the left, a main heading "Built-in Exceptions", and several paragraphs of text explaining exception handling in Python.

Table of Contents

- Built-in Exceptions
 - Base classes
 - Concrete exceptions
 - OS exceptions
 - Warnings
 - Exception hierarchy

Previous topic
Built-in Types

Next topic
Text Processing Services

This Page
Report a Bug
Show Source

Built-in Exceptions

In Python, all exceptions must be instances of a class that derives from `BaseException`. In a `try` statement with an `except` clause that mentions a particular class, that clause also handles any exception classes derived from that class (but not exception classes from which *it* is derived). Two exception classes that are not related via subclassing are never equivalent, even if they have the same name.

The built-in exceptions listed below can be generated by the interpreter or built-in functions. Except where mentioned, they have an "associated value" indicating the detailed cause of the error. This may be a string or a tuple of several items of information (e.g., an error code and a string explaining the code). The associated value is usually passed as arguments to the exception class's constructor.

User code can raise built-in exceptions. This can be used to test an exception handler or to report an error condition "just like" the situation in which the interpreter raises the same exception; but beware that there is nothing to prevent user code from raising an inappropriate error.

The built-in exception classes can be subclassed to define new exceptions; programmers are encouraged to derive new exceptions from the `Exception` class or one of its subclasses, and not from `BaseException`. More information on defining exceptions is available in the Python Tutorial under [User-defined Exceptions](#).

When raising (or re-raising) an exception in an `except` or `finally` clause `__context__` is automatically set to the last exception caught; if the new exception is not handled the traceback that is eventually displayed will include the originating exception(s) and the final exception.

When raising a new exception (rather than using a bare `raise` to re-raise the exception currently being handled), the implicit exception context can be supplemented with an explicit cause by using `raise with origin`.

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 $1 \leq i \leq n$

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" )
```

