

Exceptions, file Input & Output

講者：Isaac

Outline

- ▶ Exceptions
- ▶ File Input & output



Exceptions



Exceptions

- ▶ Triggered by errors or intercepted by your code.
- ▶ four exception handling statements(Try/Except, Try/Finally, Raise, Assert).
- ▶ Example:

```
1 def division(a, b):  
2     return a / b  
3  
4 print(division(3, 0))
```

```
-----  
ZeroDivisionError                                Traceback (most recent call last)  
<ipython-input-4-204a77054d07> in <module>  
      2     return a / b  
      3  
----> 4 print(division(3, 0))  
  
<ipython-input-4-204a77054d07> in division(a, b)  
      1 def division(a, b):  
----> 2     return a / b  
      3  
      4 print(division(3, 0))
```

```
ZeroDivisionError: division by zero
```

Exceptions

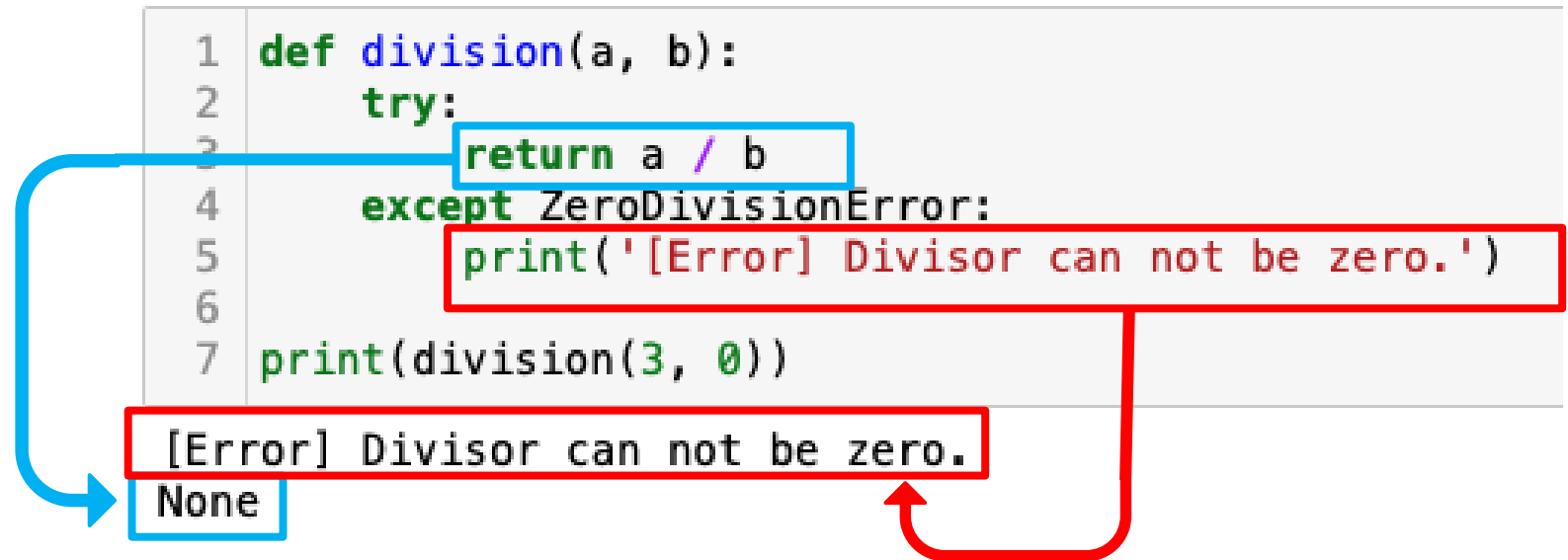
▶ Try/Except

- ▶ Catch and recover from exceptions raise by Python or developer.
- ▶ Syntax:

```
1  try:
2      <statements>           # run this main action first
3  except <name1>:
4      <statements>           # run if name1 is raised during try block
5  except (name2, name3):
6      <statements>           # run if any of these exceptions occur
7  except <name4> as <data>:
8      <statements>           # run if name4 is raised, and get instance raised
9  except:
10     <statements>           # run for all other exceptions raised
```

Exceptions

- ▶ Try/Except Example:
 - ▶ Catch one error.



Exceptions

► Common Python3 Built-in Exception:

Exception	Description
AssertionError	Raised when assert statement fails.
AttributeError	Raised when attribute assignment or reference fails.
EOFError	Raised when the input() functions hits end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raise when a generator's close() method is called.
ImportError	Raised when the imported module is not found.
IndexError	Raised when index of a sequence is out of range.
KeyError	Raised when a key is not found in a dictionary.
KeyboardInterrupt	Raised when the user hits interrupt key (Ctrl+c or delete).
MemoryError	Raised when an operation runs out of memory.

Exceptions

► Common Python3 Built-in Exception:

Exception	Description
NameError	Raised when a variable is not found in local or global scope.
NotImplementedError	Raised by abstract methods.
OSError	Raised when system operation causes system related error.
OverflowError	Raised when result of an arithmetic operation is too large to be represented.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.
RuntimeError	Raised when an error does not fall under any other category.
StopIteration	Raised by next() function to indicate that there is no further item to be returned by iterator.
SyntaxError	Raised by parser when syntax error is encountered.
IndentationError	Raised when there is incorrect indentation.

Exceptions

► Common Python3 Built-in Exception:

Exception	Description
TabError	Raised when indentation consists of inconsistent tabs and spaces.
SystemError	Raised when interpreter detects internal error.
SystemExit	Raised by <code>sys.exit()</code> function.
TypeError	Raised when a function or operation is applied to an object of incorrect type.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
UnicodeError	Raised when a Unicode-related encoding or decoding error occurs.
UnicodeEncodeError	Raised when a Unicode-related error occurs during encoding.
UnicodeDecodeError	Raised when a Unicode-related error occurs during decoding.
UnicodeTranslateError	Raised when a Unicode-related error occurs during translating.

Exceptions

► Common Python3 Built-in Exception:


Exception	Description
ValueError	Raised when a function gets argument of correct type but improper value.
ZeroDivisionError	Raised when second operand of division or modulo operation is zero.

Exceptions

- ▶ Try/Except Example:
 - ▶ Use built-in exception as error message.

```
1 def division(a, b):  
2     try:  
3         return a / b  
4     except (ZeroDivisionError) as e:  
5         print('[Error]: Divisor can not be zero.')6         print(e)  
7  
8 print(division(3, 0))
```

```
[Error]: Divisor can not be zero.  
division by zero  
None
```



Exceptions

- ▶ Try/Except Example:
 - ▶ Catch all exceptions.

```
1 def division(a, b):  
2     try:  
3         return a / b  
4     except:  
5         print('[Error]: exception happened')  
6  
7 print(division(3, 0))  
8 print(division('a', 2))
```

[Error]: exception happened

None

[Error]: exception happened

None

Exceptions

► Try/Except/Else:

- Except: catch exceptions that happen while the try block is running
- Else: runs only if no exceptions happen while the try block runs.

```
1  ### Exception Syntax
2  try:
3      <statements> # run this main action first
4  except <name1>:
5      <statements> # run if name1 is raised during try block
6  except (name2, name3):
7      <statements> # run if any of these exceptions occur
8  except <name4> as <data>:
9      <statements> # run if name4 is raised, and get instance raised
10 except:
11     <statements> # run for all other exceptions raised
12 else:
13     <statements> # run if no exceptions was raised during try block
```

Exceptions

▶ Try/Except/Else:

▶ Example

```
1 def division(a, b):  
2     try:  
3         a / b  
4     except:  
5         print('[Error]: exception happened')  
6     else:  
7         print('if there no exceptions, this msg shows.')  
8         return a / b  
9  
10 print(division(3, 2))
```

if there no exceptions, this msg shows.
1.5

Exceptions

▶ Try/Finally

- ▶ Perform cleanup actions, whether exceptions occur or not.
- ▶ Syntax:

```
1  ### Exception Syntax
2  try:
3      <statements> → # run this action first
4  finally:
5      <statements> → # always run this code on the way out
```

▶ Example:

```
1  def division(a, b):
2      try:
3          return a / b
4      finally:
5          print('no matter how, this block runs')
6
7  print(division(3, 0))
8  print(division('a', 2))
```

no matter how, this block runs

Exceptions

► Try/Except/Else/Finally

► Syntax:

```
1  ### Exception Syntax
2  try:
3      <statements> — # run this main action first
4  except <name1>:
5      <statements> — # run if name1 is raised during try block
6  except (name2, name3):
7      <statements> — # run if any of these exceptions occur
8  except <name4> as <data>:
9      <statements> — # run if name4 is raised, and get instance raised
10 except:
11     <statements> — # run for all other exceptions raised
12 else:
13     <statements> — # run if no exceptions was raised during try block
14 finally:
15     <statements> — # always run this code on the way out
```


Exceptions

► Try/Except/Else/Finally

► Example:

```
1  # try/exception/else/finally/nested
2
3  def division(a, b):
4      try:
5          a / b
6      except Exception as e:
7          print(e)
8      else:
9          print('else block')
10     finally:
11         print('finally block')
12
13 print(division(3, 0))
```

division by zero
finally block
None

Exceptions

► Raise

- Trigger exception manually in your code.
- Syntax Example:

```
3 def chkPWD(pwd):
4     pwdlen = len(pwd)
5     if pwdlen > 6:
6         raise Exception('password is too long.')
7     if pwdlen < 4:
8         raise Exception('password is too short.')
9     else:
10        print('password correct!')
11
12 for pwd in ('abcdefg', 'abc', 'abcd'):
13     try:
14         chkPWD(pwd)
15     except Exception as err_msg:
16         print('Error occurs during password checking: ', str(err_msg))
17
```

```
Error occurs during password checking: password is too long.
Error occurs during password checking: password is too short.
password correct!
```

Exceptions

▶ Assert

- ▶ Conditionally trigger an exception in your code.
- ▶ as a *conditional* raise statement.
- ▶ Syntax:

assert <condition>, <string>

String is optional.

Exceptions

► Example:

```
3 def chkPWD(pwd):
4     pwdlen = len(pwd)
5     assert pwdlen > 6, 'password is too long.'
6     assert pwdlen < 4, 'password is too short.'
7     print('password correct!')
8
9 print(chkPWD('abc'))
```

```
-----
AssertionError                                Traceback (most recent call last)
<ipython-input-24-9c9d71e7e81b> in <module>
      7     print('password correct!')
      8
----> 9 print(chkPWD('abc'))

<ipython-input-24-9c9d71e7e81b> in chkPWD(pwd)
      3 def chkPWD(pwd):
      4     pwdlen = len(pwd)
----> 5     assert pwdlen > 6, 'password is too long.'
      6     assert pwdlen < 4, 'password is too short.'
      7     print('password correct!')
```

```
AssertionError: password is too long.
```

File Input & output



IO

▶ print() syntax:

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

- ▶ Value: output data, could be multiple data.
- ▶ Sep: a separator to separate multiple data, default null char.
- ▶ End: end char, default '\n'.
- ▶ File: output location, default sys.stdout(screen).
- ▶ Flush: flush the buffer of data stream.

▶ Input() syntax:

```
value = input('prompt: ')
```

String variable

IO

- ▶ `open()` function.
- ▶ Syntax:
 - ▶ `file_obj = open(file, mode = 'r', encoding = None)`
 - ▶ File: path
 - ▶ Mode: file opening mode.
 - ▶ Encoding: utf-8

Open() mode description

Mode	description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

Open() mode description

Mode	description
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

IO

- ▶ Open file and get various information related to that file.
- ▶ Here is a list of all attributes related to file object.

File object attributes	Description
file.close	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.

IO

- ▶ **Open() & Close() example:**
 - ▶ Open a file and check its information.

```
1  #!/usr/bin/python
2
3  # Open a file
4  f = open("fruit.txt", "wb")
5  print ("Name of the file: ", f.name)
6  print ("Closed or not : ", f.closed)
7  print ("Opening mode : ", f.mode)
8  f.close()
9  print ("Closed or not : ", f.closed)
```

```
Name of the file:  fruit.txt
Closed or not :   False
Opening mode :    wb
Closed or not :   True
```

IO

- ▶ Write() example:
 - ▶ Write txt to a file, close.

```
1 f = open('example.txt', 'w', encoding= 'utf-8')
2
3 txt = '''Python is powerful... and fast;
4 plays well with others;
5 runs everywhere;
6 is friendly & easy to learn;
7 is Open.'''
8
9 f.write(txt)
10 f.close()
```

IO

▶ Read() example:

- ▶ Read the example file, close.
- ▶ Content will be all print out .

```
1 f = open('example.txt', 'r', encoding= 'utf-8')
2 content = f.read()
3 print(content)
4 f.close()
```

Python is powerful... and fast;
plays well with others;
runs everywhere;
is friendly & easy to learn;
is Open.

IO

▶ Read() example:

- ▶ If the txt file is too big, we need to limit the read() to avoid memory runs out
- ▶ Read by given characters.

```
1 f = open('example.txt', 'r', encoding= 'utf-8')
2 content = f.read(10)
3 print(content)
4 f.close()
```

Python is

IO

- ▶ Readlines() example:
 - ▶ Read the file line by line.

```
1 f = open('example.txt', 'r', encoding= 'utf-8')
2 content = f.readlines()
3 print(content)
4 f.close()
```

```
['Python is powerful... and fast; \n', 'plays well with others  
; \n', 'runs everywhere; \n', 'is friendly & easy to learn; \n', 'is Open.']
```

IO

- ▶ With ...as example:
 - ▶ More elegant
 - ▶ Easy to deal exceptions
 - ▶ Modify the read() example the file line by line.

```
1 with open ('example.txt', 'r', encoding= 'utf-8') as f:  
2     content = f.read()  
3     print(content)  
4     f.close()
```

Python is powerful... and fast;
plays well with others;
runs everywhere;
is friendly & easy to learn;
is Open.

CSV file

- ▶ Comma Separated Values format is most common import and export format for spreadsheets and databases.
 - ▶ Use comma to separate data
- ▶ How to use it:
 - ▶ import csv
- ▶ Example:
 - ▶ Write CSV

```
1 import csv
2
3 with open('csv_example.csv', 'w', newline='') as csvfile:
4     writer = csv.writer(csvfile)
5     writer.writerow(['Name', 'score'])
6     writer.writerow(['May', 75])
7     writer.writerow(['Jean', 65])
```

CSV file

► Read CSV

```
1 import csv
2
3 with open ('csv_example.csv', newline='', encoding= 'utf-8') as csvfile:
4     rows = csv.reader(csvfile)
5     for row in rows:
6         print(row)
```

```
['Name', 'score']
['May', '75']
['Jean', '65']
```