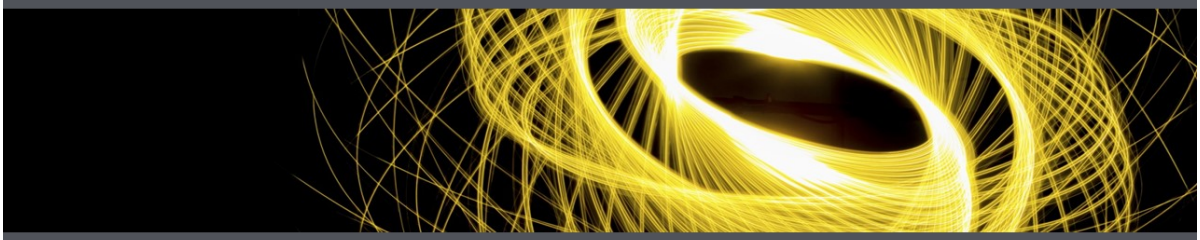
Department of Computer Science

University of Reading

CSMAD21 – Applied Data Science with Python



Exceptions

University of Reading

Lecture Objectives

- Understand the concept of exception in Python.
- Comprehend and apply different strategies to handle exceptions.
- Acknowledge the different kind of exceptions that can be encounter.
- Understand the purposes and benefit of handling exceptions.

University of Reading

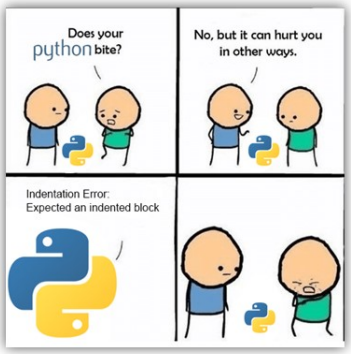
Outline

- Exceptions definition
- Why use exceptions?
- Exception roles
- Possible exceptions
- Ways to raise catch and handle exceptions

University of Reading

Exceptions and Errors

- Exceptions are events that can modify the flow of control through a program. In Python, Exceptions are triggered automatically on errors, and they can be triggered and intercepted by your code.



https://pics.me.me/does-your-python-bite-no-but-it-can-hurt-you-63085382.png

University of Reading

Why use exceptions?

- In a nutshell, exceptions let us jump out of arbitrary large chunks of a program. Let's consider a pizza making process. To make a pizza we need to go through several steps or stages of the process that goes from taking the order, prepare the dough, add toppings, bake the pie and so on.

https://media.istockphoto.com/vectors/pizza-process-set-vector-id1201112404?k=20&m=1201112404&s=170667a&w=0&h=bOzHrbhxrInPlEhFcSS0J_L-Q0S-xjO1Ea9QBNYBCKg=

University of Reading

Why use exceptions?

- Imagine that there is a problem in the “prepare the dough” sub process, in this scenario we want to be able to jump to code that handles such states.

https://media.istockphoto.com/vectors/pizza-process-set-vector-id1201112404?k=20&m=1201112404&s=170667a&w=0&h=bOzHrbhxrInPlEhFcSS0J_L-Q0S-xjO1Ea9QBNYBCKg=

University of Reading

Exception Roles

Error Handling

- Python raises exceptions whenever it detects errors in programs at runtime. You can catch and respond to the errors in your code or ignore the exceptions that are raised. If an error is ignored, Python's default exception-handling behaviour kicks in: it stops the program and prints an error message. If you don't want this default behaviour, code a ***try*** statement to catch and recover from the exception— Python will jump to your try handler when the error is detected, and your program will resume execution after the ***try***.

Event notification

- Exceptions can also be used to signal valid conditions without you having to pass result flags around a program or test them explicitly. For instance, a search routine might raise an exception on failure, rather than returning an integer result code— and hoping that the code will never be a valid result.

University of Reading

Exception Roles

Special-case handling

- Sometimes a condition may occur so rarely that it's hard to justify convoluting your code to handle it in multiple places. You can often eliminate special-case code by handling unusual cases in exception handlers in higher levels of your program. An assert can similarly be used to check that conditions are as expected during development.

Termination actions

- As you'll see, the ***try / finally*** statement allows you to guarantee that required closing-time operations will be performed, regardless of the presence or absence of exceptions in your programs.

Unusual control flows

- Finally, because exceptions are a sort of high-level and structured "go to," you can use them as the basis for implementing exotic control flows. There is no "go to" statement in Python, but exceptions can sometimes serve similar roles; a ***raise***, for instance, can be used to jump out of multiple loops.

University of Reading

Exception Roles

- Error Handling
  - Python raises exceptions whenever it detects errors in programs at runtime. You can catch and respond to the errors in your code or ignore the exceptions that are raised. If an error is ignored, Python's default exception-handling behaviour kicks in: it stops the program and prints an error message. If you don't want this default behaviour, code a try statement to catch and recover from the exception— Python will jump to your try handler when the error is detected, and your program will resume execution after the try .
- Event notification
  - Exceptions can also be used to signal valid conditions without you having to pass result flags around a program or test them explicitly. For instance, a search routine might raise an exception on failure, rather than returning an integer result code— and hoping that the code will never be a valid result.

University of Reading

Exception List

Exception	Description	Exception	Description
AssertionError	Raised when the assert statement fails.	RuntimeError	Raised when an error does not fall under any other category.
AttributeError	Raised on the attribute assignment or reference fails.	StopIteration	Raised by the next() function to indicate that there is no further item to be returned by the iterator.
EOFError	Raised when the input() function hits the end-of-file condition.	SyntaxError	Raised by the parser when a syntax error is encountered.
FloatingPointError	Raised when a floating point operation fails.	IndentationError	Raised when there is an incorrect indentation.
GeneratorExit	Raised when a generator's close() method is called.	TabError	Raised when the indentation consists of inconsistent tabs and spaces.
ImportError	Raised when the imported module is not found.	SystemError	Raised when the interpreter detects internal error.
IndexError	Raised when the index of a sequence is out of range.	SystemExit	Raised by the sys.exit() function.
KeyError	Raised when a key is not found in a dictionary.	TypeError	Raised when a function or operation is applied to an object of an incorrect type.
KeyboardInterrupt	Raised when the user hits the interrupt key (Ctrl+c or delete).	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.
MemoryError	Raised when an operation runs out of memory.	UnicodeError	Raised when a Unicode-related encoding or decoding error occurs.
NameError	Raised when a variable is not found in the local or global scope.	UnicodeEncodeError	Raised when a Unicode-related error occurs during encoding.
NotImplementedError	Raised by abstract methods.	UnicodeDecodeError	Raised when a Unicode-related error occurs during decoding.
OSError	Raised when a system operation causes a system-related error.	UnicodeTranslateError	Raised when a Unicode-related error occurs during translation.
OverflowError	Raised when the result of an arithmetic operation is too large to be represented.	ValueError	Raised when a function gets an argument of correct type but improper value.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.	ZeroDivisionError	Raised when the second operand of a division or module operation is zero.

University of Reading

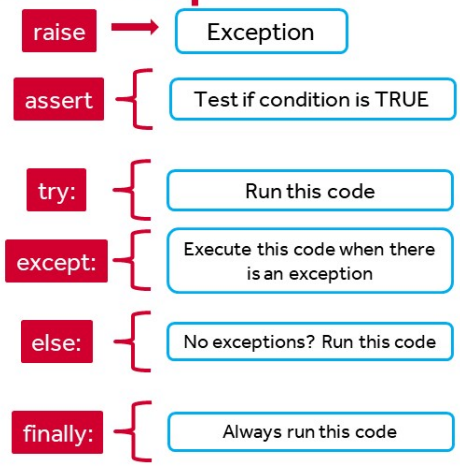
Ways to raise, catch and handle exceptions

- **Raise**: allows you to throw an exception at any time.
- **Assert**: enables you to verify if a certain condition is met and throw an exception if it isn't.
- **Try**: all statements are executed until an exception is encountered.
- **Except**: is used to catch and handle the exception(s) that are encountered in the try clause.
- **Else**: lets you code sections that should run only when no exceptions are encountered in the try clause.
- **Finally**: enables you to execute sections of code that should always run, with or without any previously encountered exceptions.

University of Reading

Ways to raise, catch and handle exceptions

- **Raise**: allows you to throw an exception at any time. Use to force an exception.
- **Assert**: enables you to verify if a certain condition is met and throw an exception if it isn't.
- **Try**: all statements are executed until an exception is encountered.
- **Except**: is used to catch and handle the exception(s) that are encountered in the try clause.
- **Else**: lets you code sections that should run only when no exceptions are encountered in the try clause.
- **Finally**: enables you to execute sections of code that should always run, with or without any previously encountered exceptions.



In [5]:
```
print(dir(locals()['__builtins__'])) ##built-in exceptions, functions, and attribute
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIO
Error', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'Conn
ectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetEr
ror', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception',
'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarnin
g', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError',
'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrup
t', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotAD
irectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError',
'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionErro
r', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyn
cIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'System
Exit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'Unicod
eDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'Unico
deWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionErr
or', '__IPYTHON__', '__build_class__', '__debug__', '__doc__', '__import__', '__load
er__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin',
```

```
'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compi
le', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmo
d', 'enumerate', 'eval', 'exec', 'filter', 'float', 'format', 'frozenset', 'get_ipyt
hon', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int',
'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'ma
x', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'pr
operty', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted',
'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

In [6]:

```
###Exceptions
##Let's define a function
def fetcher(obj, index):
    return obj[index]
```

In [11]:

```
x = 'Lupita'
##Lets explore the different kind of exceptions we can get while executing the funct
##No error:
#fetcher(x,5)
##IndexError:
#fetcher(x,9)
##ValueError:
fetcher(x,int('m'))
#fetcher(x,int('3'))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-a0f90059d8bf> in <module>
      6 #fetcher(x,9)
      7 ##ValueError:
----> 8 fetcher(x,int('m'))
      9 #fetcher(x,int('3'))

ValueError: invalid literal for int() with base 10: 'm'
```

In [13]:

```
##Defining a "catcher" to handle the exceptions
def catcher():
    word = str(input("Please enter the word to evaluate: "))
    indx = int(input("Please enter the possition of the letter to retrieve: "))
    print('The letter is: ', fetcher(word, indx))##This is the line that will trigge
```

In [14]:

```
##Simulating an index outside the limit
catcher()
```

```
Please enter the word to evaluate: hello
Please enter the possition of the letter to retrieve: 10
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-14-815c9bfb163f> in <module>
      1 ##Simulating an index outside the limit
----> 2 catcher()

<ipython-input-13-aa9fd55f71c1> in catcher()
      3     word = str(input("Please enter the word to evaluate: "))
      4     indx = int(input("Please enter the possition of the letter to retrieve:
 "))
----> 5     print('The letter is: ', fetcher(word, indx))##This is the line that wil
l trigger the exception

<ipython-input-6-cc8fbde68452> in fetcher(obj, index)
      2 ##Let's define a function
      3 def fetcher(obj, index):
----> 4     return obj[index]
```

```
IndexError: string index out of range
```

In [17]:
```
##Simulating a wrong index
catcher()
```

```
Please enter the word to evaluate: hello
Please enter the possition of the letter to retrieve: m
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-f68334d19e26> in <module>
      1 ##Simulating a wrong index
----> 2 catcher()

<ipython-input-13-aa9fd55f71c1> in catcher()
      2 def catcher():
      3     word = str(input("Please enter the word to evaluate: "))
----> 4     indx = int(input("Please enter the possition of the letter to retrieve:
 "))
      5     print('The letter is: ', fetcher(word, indx))##This is the line that wil
l trigger the exception

ValueError: invalid literal for int() with base 10: 'm'
```

Handling Exceptions

Simple exceptions, try and except:

In [18]:
```
##Simple exception
def catcher_exp():
    word = str(input("Please enter the word to evaluate: "))
    try:
        indx = int(input("Please enter the possition of the letter to retrieve: "))
        print('The letter is: ', fetcher(word, indx))##This is the line that will tr
    except:
        print('There was an exception!')
```

In [20]:
```
catcher_exp()
```

```
Please enter the word to evaluate: hello
Please enter the possition of the letter to retrieve: m
There was an exception!
```

In [21]:
```
##Exception with additional detail
def catcher_exp_det():
    word = str(input("Please enter the word to evaluate: "))
    try:
        indx = int(input("Please enter the possition of the letter to retrieve: "))
        print('The letter is: ', fetcher(word, indx))##This is the line that will tr
    except Exception as ex:
        print('There was an exception: ', ex)
```

In [23]:
```
catcher_exp_det()
```

```
Please enter the word to evaluate: hello
Please enter the possition of the letter to retrieve: m
There was an exception:  invalid literal for int() with base 10: 'm'
```

Handling particular exceptions (try and except):

In [24]:
```
##In this case we are going to handle the case when the index is out of range
def catcher_exp_indx():
    word = str(input("Please enter the word to evaluate: "))
    try:
        indx = int(input("Please enter the possition of the letter to retrieve: "))
        print('The letter is: ', fetcher(word, indx))##This is the line that will tr
    except IndexError:
        print('The index you provided is out of the limit')
```

In [25]:
```
catcher_exp_indx()
```

```
Please enter the word to evaluate: hello
Please enter the possition of the letter to retrieve: 9
The index you provided is out of the limit
```

In [26]:
```
##Handle out of range and wrong index:
def catcher_exp_indx_val():
    word = str(input("Please enter the word to evaluate: "))
    try:
        indx = int(input("Please enter the possition of the letter to retrieve: "))
        print('The letter is: ', fetcher(word, indx))
    except IndexError:
        print('The index you provided is out of the limit')
    except ValueError:
        print('The index you provided is incorrect')
```

In [28]:
```
catcher_exp_indx_val()
```

```
Please enter the word to evaluate: hello
Please enter the possition of the letter to retrieve: m
The index you provided is incorrect
```

Handling particular exceptions (try, try except and finally):

In [32]:
```
def catcher_exp_indx_val_els():
    word = str(input("Please enter the word to evaluate: "))
    while True:
        try:
            indx = int(input("Please enter the possition of the letter to retrieve:
            ltr = fetcher(word, indx) ##This is the line that will trigger the excep
        except IndexError:
            print('The index you provided is out of the limit')
        except ValueError:
            print('The index you provided is incorrect')
        else:
            print('The letter is: ', ltr)
            break
```

In [33]:
```
catcher_exp_indx_val_els()
```

```
Please enter the word to evaluate: hello
```

```
Please enter the possition of the letter to retrieve: m
The index you provided is incorrect
Please enter the possition of the letter to retrieve: j
The index you provided is incorrect
Please enter the possition of the letter to retrieve: 2
The letter is:  l
```

In [34]:

```python
def catcher_exp_indx_val_els_fin():
    word = str(input("Please enter the word to evaluate: "))
    while True:
        try:
            indx = int(input("Please enter the possition of the letter to retrieve:
            ltr = fetcher(word, indx)##This is the line that will trigger the except
        except IndexError:
            print('The index you provided is out of the limit')
        except ValueError:
            print('The index you provided is incorrect')
        else:
            print('The letter is: ', ltr)
            break
        finally:
            print('Im always here')
```

In [35]:

```python
catcher_exp_indx_val_els_fin()
```

```
Please enter the word to evaluate: hello
Please enter the possition of the letter to retrieve: 9
The index you provided is out of the limit
Im always here
Please enter the possition of the letter to retrieve: m
The index you provided is incorrect
Im always here
Please enter the possition of the letter to retrieve: 2
The letter is:  l
Im always here
```

Raise and Assert:

In [36]:

```python
##Raise
def catcher_exp_indx_val_raise():
    word = str(input("Please enter the word to evaluate(just characters): "))
    if word.isalpha():
        while True:
            try:
                indx = int(input("Please enter the possition of the letter to retrie
                print('The letter is: ', fetcher(word, indx))
                break
            except IndexError:
                print('The index you provided is out of the limit')
            except ValueError:
                print('The index you provided is incorrect')
    else:
        raise TypeError('Only characters are allowed')
```

In [38]:

```python
catcher_exp_indx_val_raise()
```

```
Please enter the word to evaluate(just characters): hello9977
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
<ipython-input-38-b7df9bf79a1e> in <module>
----> 1 catcher_exp_indx_val_raise()

<ipython-input-36-f41cbf54d747> in catcher_exp_indx_val_raise()
     13                 print('The index you provided is incorrect')
     14     else:
---> 15         raise TypeError('Only characters are allowed')

TypeError: Only characters are allowed
```

In [40]:

```
##Assert
import sys
assert ('linux' in sys.platform), "This code runs on Linux only."
#assert ('win32' in sys.platform), "This code runs on Windows only."
catcher_exp_indx_val_raise()
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-40-a3ba91d4ff50> in <module>
      1 ##Assert
      2 import sys
----> 3 assert ('linux' in sys.platform), "This code runs on Linux only."
      4 #assert ('win32' in sys.platform), "This code runs on Windows only."
      5 catcher_exp_indx_val_raise()

AssertionError: This code runs on Linux only.
```

University of Reading

Summary

- Exceptions are simple but powerful tools. They are high-level control flow device.
- They may be rise by Python or by your own program.
- There are several exception roles such as error handling, event notification, termination actions, special case handling and unusual control flows.
- Raise and assert statements triggers exceptions on demand both built-ins and new exceptions.
- The statements try, except, else and finally are statements to handle exceptions.

University of Reading

Questions

University of Reading

References

- Mark Lutz, 'Learning Python: Powerful Object-Oriented Programming', Chapter 33: Exception Basics
- Python Documentation, 8. Error and Exceptions, https://docs.python.org/3/tutorial/errors.html
- Said van de Klundert, Python Exceptions: An Introduction, Real Python, https://realpython.com/python-exceptions/