Errors and Exceptions

UBCO Master of Data Science – DATA 533

Today's Class

Python modules and packages

- Python OOP (L1-2)
- Modules and Packages (L3)
- Collaborative version control (L4)
- Testing, CI/CD, **Errors and Exceptions** (L5-7)
- Publishing packages (L8)

Design an application in collaboration

Today's Class

Python Errors and Exceptions

- Syntax Errors
- Exceptions

Handling Exceptions

Raising Exceptions

User-defined Exceptions

Errors

Programs are error-prone.

A python program terminates as soon as it encounters an error.

There are (at least) two distinguishable kinds of errors:

- ***Syntax errors***
- ***Exceptions***

Syntax Errors

Syntax errors occur when the parser detects an incorrect statement

Syntax errors, also known as ***parsing errors***, are perhaps the most common kind of complaint.

```
def my_function()
    msg = "Hello world!"
    return msg
print(my_function())
```

Syntax Errors

```
def my_function()
    msg = "Hello world!"
    return msg
print(my_function())
```

Output:

```
Cell In[1], line 1
    def my_function()
                     ^
SyntaxError: expected ':'
```

The error is detected at the my_function(), as a colon (':') is missing before it.

An '***arrow***' pointing at the earliest point in the line where the error was detected.

A ***line number*** is printed indicating the error position.

Exceptions

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it.

When an error is detected, Python raises an exception.

```
10 * (1/0)
```

```
ZeroDivisionError Traceback (most
recent call last) Cell
In[2], line 1 ----> 1 10 * (1/0)
ZeroDivisionError: division by zero
```

Exceptions

An ***exception*** is an object that indicates an error or anomalous condition.

Exceptions can be handled by the program.

If the exception is not handled:

- Program terminates.
- Error message and traceback report is printed.

```
10 * (1/0)
```

```
ZeroDivisionError Traceback (most recent call
last) Cell In[2], line 1 ----> 1 10 * (1/0)
ZeroDivisionError: division by zero
```

Exception Examples (TypeError)

```
a = 2
b = 'Data533'
print(a + b)
```

```
TypeError  Traceback (most recent call last)
Cell In[3], line 3
      1 a = 2
      2 b = 'Data533'
----> 3 print(a + b)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Raised when an operation or function is applied to an object of ***inappropriate type***.

Exception Examples (NameError)

```
var2 = 4 + var1 * 3
print(var2)
```

```
NameError Traceback (most recent call last) Cell In[4], line 1
----> 1 var2 = 4 + var1 * 3
      2 print(var2)

NameError: name 'var1' is not defined
```

Raised when a ***name is not found***.

Exception Examples

```
NameError Traceback (most recent call last) Cell In[4], line 1
----> 1 var2 = 4 + var1 * 3
      2 print(var2)


NameError: name 'var1' is not defined
```

The last line of the error message indicates what happened.

The string printed as the exception type is the name of the built-in exception that occurred.

The rest of the line provides detail based on the type of exception and what caused it.

traceback

A `traceback` is a stack trace from the point of an exception handler down the call chain to the point where the exception was raised.

```
 1 def f1(x):
 2     assert x == 1
 3
 4 def f2(x):
 5     f1(x)
 6
 7 def f3(x):
 8     f2(x)
 9
10 f1(1)
11 f1(2)
```

```
AssertionError Traceback (most recent
call last) Cell In[5], line 11
       8 f2(x)
       10 f1(1)
  ---> 11 f1(2)

Cell In[5], line 2, in f1(x)
       1 def f1(x):
   ----> 2 assert x == 1

 AssertionError:
```

`traceback` example

```
 1 def f1(x):
 2     assert x == 1
 3
 4 def f2(x):
 5     f1(x)
 6
 7 def f3(x):
 8     f2(x)
 9
10 f2(1)
11 f2(2)
```

```
AssertionError
Traceback (most recent call last)
Cell In[6], line 11
      8     f2(x)
     10 f2(1)
---> 11 f2(2)

Cell In[6], line 5, in f2(x)
      4 def f2(x):
----> 5     f1(x)

Cell In[6], line 2, in f1(x)
      1 def f1(x):
----> 2     assert x == 1

AssertionError:
```

More Exceptions 😟

All exceptions are ***objects***.

The classes that define the objects are organized in a ***hierarchy***

```
BaseException
+-- Exception
      +-- ArithmeticError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- NameError
      +-- TypeError
      +-- .........
```

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

Exceptions

Exceptions	Description
`BaseException`	Base class for most of the built-in exceptions.
`IndentationError`	Raised when indentation is not specified properly.
`NameError`	Raised when an identifier is not found in the local or global namespace.
`TypeError`	Raised when an operation or function is attempted that is invalid for the specified data type.

`IndexError`	Raised when an index is not found in a sequence.
	`a = ['a', 'b']`
	`print (a[2])`

Exceptions

Exceptions	Description

`KeyError`	Raised when the specified key is not found in the dictionary.
	`ages = {'Jim': 30, 'Pam': 28, 'Kevin': 33}`
	`ages['Michael']`

`ValueError`	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified. `int("value")`

`RecursionError`	Raised when the maximum recursion depth has been exceeded.
	`def recursion():`
	`    return recursion()`
	`recursion()`
	`import sys`
	`sys.getrecursionlimit()`

Exceptions

Exceptions	Description
`ArithmeticError`	**Base class** for all errors that occur for numeric calculation.

`OverflowError`	Raised when a calculation exceeds maximum limit for a numeric type.
	`import math`
	`print(math.exp(1000))`

`ZeroDivisonError`	Raised when division or modulo by zero takes place for all numeric types.

Exceptions

Exceptions	Description
`FileNotFoundError`	Raised when a file or directory doesn't exist.
`ImportError`	Raised when a module, or member of a module, cannot be imported.

`MemoryError`	Raised when a operation runs out of memory.
	`s = []`
	`for i in range(1000):`
	`    for j in range(1000):`
	`        for k in range(1000):`
	`            s.append("DATA533")`

`AssertionError`	When an assert statement is failed `assert "a" == "b"`

Exception Question

Question: The follow program will show a ________ exception

```
mydictionary = {'a':'1','b':'2'}

print(mydictionary['1'])
```

A) `KeyError` **B)** `ValueError` **C)** `MemoryError`

D) `IOError` **E)** `FileNotFoundError`

Exception Question

Question: The follow program will show a ________ exception

```
a = 100

b = "Data533"

assert a == b
```

A) `AssertionError` **B)** `NameError` **C)** `TypeError`

D) `IndentationError` **E)** `IndexError`

Exception Question

Question: The follow program will show a ________ exception

```
def my_function(a):

return a

print(my_function(10))
```

A) `ImportError` **B)** `NameError` **C)** `IndentationError`

D) `TypeError` **E)** `IndexError`

The try Statement

The `try` statement provides Python’s exception-handling mechanism.

A `try` statement may have more than one `except` clause, to specify handlers for different exceptions.

At most one handler will be executed.

Handlers only handle exceptions that occur in the corresponding `try` clause, not in other handlers of the same `try` statement.

The `try-except` Statement

```
try:
	#Normal code goes here
except <ExceptionType1>:
	# If ExceptionType1 was raised, then execute this block.
except <ExceptionTypeN>:
	# If ExceptionTypeN was raised, then execute this block.
except:
	# For all other exceptions, execute this block.
else:
	# If there was no exception then execute this block.
finally:
	# This block of code will always execute, even if there are
exceptions raised
```

Python Exceptions Example

```
try:
   num = int(input("Enter a number:"))
   print("You entered:",num)
except ValueError:
    print("Please write a valid number")
else:
    print("Thank you for the number")
finally:
    print("Always do finally block")
```

try block exit if error

execute if exception

only execute if no exception

Always execute

The try Statement

The `try` statement provides Python's exception-handling mechanism.

The try statement works as follows:

- First, the try clause is executed
- If no exception occurs, the except clause is skipped and execution of the try statement is finished
- If an exception occurs during execution of the try clause, the rest of the clause is skipped.
    - Then if its type matches the exception named after the except keyword, the except clause is executed, and then execution continues after the try statement.
    - If an exception occurs which does not match the exception named in the except clause, it is passed on to outer try statements
    - if no handler is found, it is an unhandled exception and execution stops with a message

Question: Exceptions

Question: What is the output of the following code?

```
import sys
randomList = [2]
for entry in randomList:
    try:
        r = 1/int(entry)
    except ValueError:
        print("Value Error")
    except ZeroDivisionError:
        print("Divide by 0")
    except:
        print(sys.exc_info()[0],"occured.")
    else:
        print("Answer:",r)
    finally:
        print("Finally")
```

A) Finally

B) Divide by 0
Finally

C) Answer: 0.5
Finally

D) Value Error
Finally

E) None of the Above

Question: Exceptions

Question: What is the output of the following code?

```
import sys
randomList = ['p']
for entry in randomList:
    try:
        r = 1/int(entry)
    except ValueError:
        print("Value Error")
    except ZeroDivisionError:
        print("Divide by 0")
    except:
        print(sys.exc_info()[0],"occured.")
    else:
        print("Answer:",r)
    finally:
        print("Finally")
```

A) `Finally`

B) `Divide by 0`
`Finally`

C) `Answer: 0.5`
`Finally`

D) `Value Error`
`Finally`

E) `None of the Above`

Question: Exceptions

Question: What is the output of the following code?

```
def division_handle(x):
     try:
         print(20/x)
     except ZeroDivisionError:
         print("Can't divide by zero.")
     print("Complete running")

division_handle(0)
```

A) `Can't divide by zero`

B) `Complete running`

C) `Can't divide by zero`
`Complete running`

D) `Divide by 0 Error`

E) `None of the Above`

Try it: Python Exceptions

Question: Write a Python program that reads two comma separated numbers (use `num1, num2 = eval(input("Enter two numbers"))`). Then divide the first number by the second number.

- If the program gets a `ZeroDivisionError`, print `Cannot divide by 0!`
- If the program gets a `SyntaxError`, print `Comma missing`
- For all other errors, print `Wrong input`
- If the program runs successfully for the given inputs, print `Successful`
- Whether an exception occurs or not, the program prints `Finish` at the end

Now test the program with the following inputs:

10, 0

10, 2

10 2

Using the Exception Object

We can assign the object to a variable that we can use inside the except clause like this:

```
def division_handle(x):
     try:
          print(20/x)
     except ZeroDivisionError as ex:
          print(ex)
     except TypeError as ex:
          print(ex)
```

```
division_handle(0)
division by zero

division_handle('A')
unsupported operand
type(s) for /: 'int'
and 'str'
```

`ex` is not a string, but Python knows how to convert it into one – the string representation of an exception is the message.

Raising Exceptions

Exceptions can be raised and can be initiated explicitly with `raise`.

We can also provide additional information about the exception

- Use the exception class constructor and include the applicable error message to create the instance.
- Syntax: `raise ExceptionClass("Your argument")`

A plain `raise` statement re-raises the same exception object that the handler received.

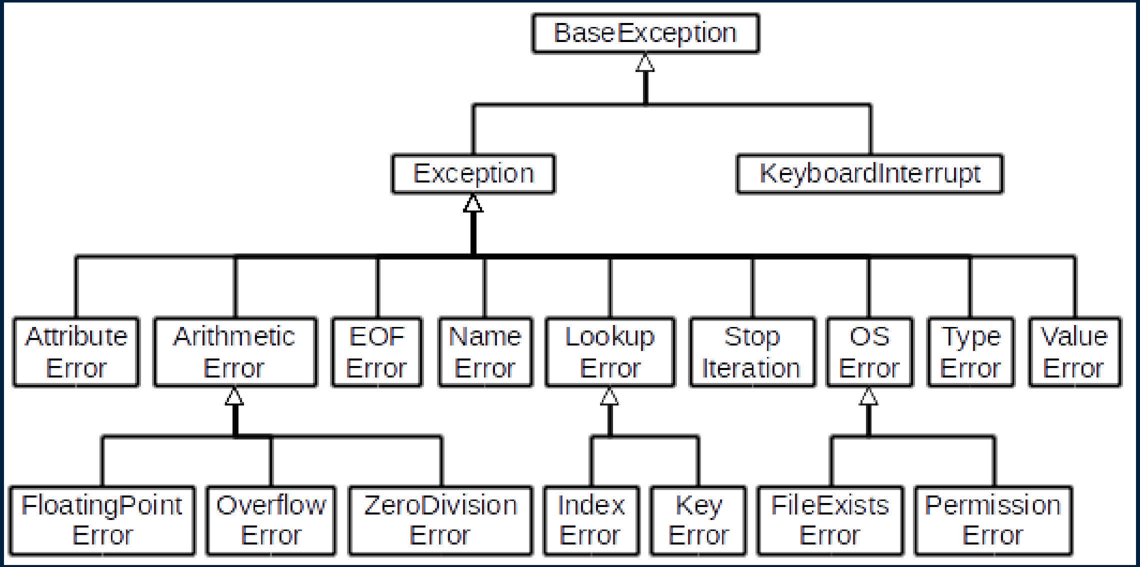
Raising Exceptions Example

```
age = int(input("Enter your age: "))
try:
    if age < 0:
        raise ValueError("Only positive integers are
allowed")
except ValueError as ex:
    print("Message:", ex)
except:
    print("something is wrong")
else:
    print("Your age is", age)
```

User-Defined Exceptions

Programs may name their own exceptions by creating a new exception class.

You can create a custom exception class by Extending `Exception` class



User-Defined Exceptions

```
class MyInputError(Exception):
    pass


a = int(input ("Enter a number >0 "))
try:
    if a <= 0:
        raise MyInputError()
except MyInputError:
    print ("Number should greater than 0")
```

User-Defined Exceptions

```
class Error(Exception):
    def __init__(self, value): # Constructor or Initializer
        self.value = value
    def __str__(self):           # __str__ to print() the value
        return(repr(self.value))
try:
    raise(Error(6*2))
except Error as ex:
    print('Exception raised:',ex.value)
```

Question: Exceptions

Question: What is the output of the following code?

```
class OverAge(Exception):
    def __init__(self):
        print("Overage")

def check_age(age):
    if age > 65:
        raise OverAge
    else:
        print('Age: ',age)

# main program
try:
    check_age(66)
except OverAge:
    print("Overage")
```

A) Overage

B) Overage
Overage

C) Overage
Overage
Overage

D) Age: 66

E) None of the Above

Exiting the system

If you want to *exit* the system at any point, call:

```
import sys
sys.exit()
```

This has an option:

```
sys.exit(arg)
```

If `arg` is an integer, zero is considered "successful termination"

Any nonzero value is considered "abnormal termination"

For more on this, see:

https://docs.python.org/3/library/sys.html#sys.exit

Try it: User-Defined Exceptions

Question: Write a Python program with the following three user-defined exceptions

- `Error`: It is using the `Exception` class as the parent.
- `PasswordTooSmallError` and `PasswordTooLargeError`, both derive from `Error`.
- All these classes have one `pass` statement.

Now prompt users to provide a password.

- If the password length is less than 6, it will raise `PasswordTooSmallError` `and` print `Password length is too small`
- If the password length is greater than 12, it will raise `PasswordTooLargeError` `and` print `Password length is too large`

Objectives

- Understand Python errors and exceptions
- Learn how to write code to handle exceptions
- Learn how to raise exceptions
- Be able to create user-defined exceptions

THE UNIVERSITY OF BRITISH COLUMBIA