

Python 3 Exceptions & Handling Errors

Eddie Guo

October 2019

1 Introduction to Exceptions

1.1 Topics Covered

- (i) What are exceptions?
- (ii) Raising exceptions
- (iii) Catching exceptions
- (iv) Assertions

1.2 What are Exceptions?

- Input validation != exception handling
- Exception: event during execution of program that disrupts normal flow of program
- Exceptions allow us to handle errors/exceptional conditions
- In Python, exception is obj that reps an error

1.3 Exceptions in Python

- Python script raises an exception where error detected
 - $365 * (12/0) \rightarrow$ **ZeroDivisionError**: integer division or modulao by zero
- Python interpreter raises exception when it detects run-time error
- Can explicitly raise an exception
 - `'2013' + 1` \rightarrow **TypeError**: cannot concatenate 'str' and 'int' objects
 - `175 + cmput*13` \rightarrow **NameError**: name 'cmput' is not defined
- Accessing non-existent dictionary key will raise **KeyError** exception
- Searching list for non-existent value will raise **ValueError** exception
- Calling non-existent method will raise **AttributeError** exception
- [Python documentation for exceptions](#)

1.4 Why Use Exceptions?

- Separating error-handling code from regular code
- Deferring decisions about how to respond to exceptions
- Providing mech for specifying diff kinds of exceptions that can arise in program

1.5 Exception Handling Blocks

- If you have code that may raise exception, place code in **try**: followed by **except**:
- Don't catch exception? Entire program crashes.
- **except** w/o explicit exception will catch all remaining exceptions
- **except** may name none/one/multiple exceptions as parenthesized tuple

```
1 except (RuntimeError, TypeError,
2         NameError):
    [do something here]
```

1.6 Multiple Except Clauses

- try may have >1 except clause to specify handlers for diff exceptions
- At most, one handler will be executed
- Handlers only handle exceptions that occur in

corresponding try clause, not in other handlers of same try statement

- Go from specific exceptions to more general ones b/c Python reads top-down

1.7 The try Statement

- If no exception raised by code w/in try block (or methods called w/in try block), code executes normally & all except blocks skipped
- If exception arises in try block, execution of try block terminates execution immediately & except is sought to handle exception

1. If appropriate except clause found, it's executed
2. Elif exception propagated to method or outer try block
3. Elif no handler found → **unhandled exception** & execution stops w/ message

1.8 Propagating Exceptions

- An exception will bubble up call stack until it:
 - Reaches method w/ suitable handler or
 - Propagates thru main stack (1st method on call stack)

- If exception not caught by any method, exception treated like error: stack frames displayed & program terminates

```
1 try:
2     f = open('myfile.txt', 'r')
3     s = f.readline()
4     i = int(s.strip())
5 except IOError:
6     print('File does not exist or cannot be read.')
7 except ValueError:
8     print('Could not convert data to an integer')
9 except: # If this were first, no IOErrors or ValueErrors will be caught
10    print('Unexpected error')
11    raise # can explicitly propagate exceptions using raise
```

1.9 Raising Exceptions

- What can be raised as exception?
 - Any standard Python exception
 - New instance of exception w/ custom ar-

guments

- Instances of our own specialized exception classes

```
1 try:
2     print('Raising an exception')
3     raise Exception('CMPTUT', '274')
4 except Exception as inst: # the exception instance
5     print(inst.args) # arguments stored in .args
6     x, y = inst.args # unpacks args
7     print('x =', x, 'y =', 'y')
```

1.10 else & finally Clause

- else clause CAN'T come b4 try & except (i.e., must follow all except clauses)
- Code in else clause must be executed if clause does not raise exception

- finally will execute regardless if error was raised (executed under ALL circumstances)
- finally useful if you wanna perform “cleanup” operations b4 exiting method (ex: closing file) & avoids duplicating code in each except clause

```

1 def divide(x, y):
2     try:
3         result = x / y
4     except ZeroDivisionError:
5         print('division by zero!')
6     else:
7         print('the result is', result)
8     finally:
9         print('thanks for dividing!')

```

finally clause is executed in any event

- no exception
- division by 0
- type error

Type error is **re-raised** after the *finally* clause since no *except* exists for it.

```

>>> divide(2, 1)
result is 2.0
thanks for dividing
>>> divide(2, 0)
division by zero!
thanks for dividing
>>> divide("CMPUT", "175")
thanks for dividing
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 3, in divide
TypeError: unsupported operand
type(s) for /: 'str' and 'str'

```

2 Summary: Possible Execution Paths

- No exception occurs
 - Execute try block
 - Execute else & finally clauses
 - Execute rest of method
 - Execute finally clause
 - Execute rest of method
- Exception occurs & is caught
 - Execute try block until 1st exception occurs
 - Execute 1st except clause that matches exception
 - Execute finally clause
 - Propagate exception to calling method
- Exception occurs & is not caught
 - Execute try block until 1st exception occurs
 - Execute try block until 1st exception occurs
 - Execute finally clause
 - Propagate exception to calling method

3 Assertions

- Assertion is statement that raises **AssertionError** exception if condition not met
- `assert Expression[, Arguments]`
- If assertion fails, Python uses given arg as arg for **AssertionError**
- AssertionError** exceptions can be caught & handled like any other exception
- Good practice to place assertions at start of fn to check for valid input, & after fn call to check for valid output

```

1 def KelvintoFahrenheit(temperature):
2     assert (temperature >= 0), 'Colder than absolute zero!'
3     return ((temperature - 273) * 1.80 + 32)
4
5 if __name__ == '__main__':
6     try:
7         fahrenheit = KelvintoFahrenheit(-23)
8         print(fahrenheit)
9     except AssertionError as my_error:
10        print(my_error.args)
11
12 # Output
13 (Colder than absolute zero!, )

```