



Lecture 17: Exceptions & Unit Testing

IN628: Programming 4

Semester One, 2020

Kaiako: Grayson Orr

Te Kura Matatini ki Otago, Ōtepoti, Aotearoa

SYNTAX ERRORS

► Parsing errors

```
while True print('John_Doe')
```

```
File "<ipython-input-1-2b688bc740d7>", line 1
```

```
    while True print('John_Doe')
```

```
            ^
```

```
SyntaxError: invalid syntax
```

EXCEPTIONS

- ▶ Errors detected during execution
- ▶ Most exceptions aren't handled by the programmer

INDEXERROR

- Raised when a sequence index is out of range

```
nums = [1, 2, 3, 4, 5]  
nums[5]
```

```
IndexError                                Traceback (most recent call last)  
<ipython-input-16-725b0b18bdb2> in <module>  
      1 nums = [1, 2, 3, 4, 5]  
----> 2 nums[5]
```

```
IndexError: list index out of range
```

KEYERROR

- Raised when a dictionary key isn't found in the set of existing keys

```
person = { 'first_name': 'John', 'last_name': 'Doe' }  
person['first']
```

```
KeyError                                Traceback (most recent call last)  
<ipython-input-18-c227410ba879> in <module>  
      1 person = { 'first_name': 'John', 'last_name': 'Doe' }  
----> 2 person['first']  
  
KeyError: 'first'
```

NAMEERROR

- Raised when a local or global name isn't found

```
x
```

```
NameError                                Traceback (most recent call last)
<ipython-input-11-6fcf9dfbd479> in <module>
----> 1 x
```

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

TypeError

- Raised when an operation or function is applied to an object isn't supported

```
'1' + 1
```

```
TypeError                                Traceback (most recent call last)
<ipython-input-12-cc892b1f57d5> in <module>
----> 1 '1' + 1
```

```
TypeError: can only concatenate str (not "int") to str
```

ZERODIVISIONERROR

- Raised when the second argument of a division or modulo operation is zero

```
10/0
```

```
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-13-e574edb36883> in <module>
----> 1 10/0
```

```
ZeroDivisionError: division by zero
```


RAISING EXCEPTIONS

- ▶ Allows the programmer to force a specified exception to occur
- ▶ The sole argument to raise must be either an exception instance or exception class

```
raise NameError
```

```
NameError                                Traceback (most recent call last)  
<ipython-input-1-42b67b2fc75d> in <module>  
----> 1 raise NameError
```

```
NameError:
```

HANDLING EXCEPTIONS

- ▶ Try
- ▶ Except

```
while True:
    try:
        x = int(input('Please enter a number: '))
        break
    except ValueError:
        print('Oops! That was an invalid number. Please try again ...')
```

```
Please enter a number: !
Oops! That was an invalid number. Please try again...
Please enter a number: 1
```

CLEAN-UP ACTIONS

► Finally

```
try:
    raise KeyboardInterrupt
finally:
    print('exit(0)')
```

```
exit(0)
```

```
KeyboardInterrupt                                Traceback (most recent call last)
<ipython-input-1-cf37e4784c69> in <module>
      1 try:
----> 2     raise KeyboardInterrupt
      3 finally:
      4     print('exit(0)')
```

```
KeyboardInterrupt:
```

AUTOMATION TESTING

- ▶ Technique used to test & compare the actual outcome with the expected outcome
- ▶ Writing test scripts or using automation testing tools
- ▶ Use of software to control the execution of tests
- ▶ Automate tasks which are difficult to perform manually

UNIT TESTING

- ▶ Individual units/components are tested
- ▶ Smallest testable part of any software
- ▶ One or two inputs & one output
 - ▶ In OOP, the smallest unit is a method
- ▶ Each unit performs as designed
- ▶ unittest module
 - ▶ Originally inspired by JUnit

UNIT TESTING

```
from unittest import TestCase, main

class Person:
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    def is_legal(self):
        return True if self.age >= 18 else False

class TestPerson(TestCase):
    def setUp(self):
        self.person.1 = Person('John', 'Doe', 25)
        self.person.2 = Person('Jane', 'Doe', 5)

    def test_is_legal(self):
        self.assertEqual(True, self.person.1.is_legal())

    def test_is_not_legal(self):
        self.assertEqual(False, self.person.2.is_legal())

    def tearDown(self):
        self.person.1 = None
        self.person.2 = None

if __name__ == '__main__':
    main(argv=[''], verbosity=2, exit=False)
```

UNIT TESTING: TEST SUITE

► A collection of test cases

```
from unittest import TestCase, TestSuite, TextTestRunner, main

class Person:
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    def is_legal(self):
        return True if self.age >= 18 else False

class TestPerson(TestCase):
    def setUp(self):
        self.person.1 = Person('John', 'Doe', 25)
        self.person.2 = Person('Jane', 'Doe', 5)

    def test_is_legal(self):
        self.assertEqual(True, self.person.1.is_legal())

    def test_is_not_legal(self):
        self.assertEqual(False, self.person.2.is_legal())

    def tearDown(self):
        self.person.1 = None
        self.person.2 = None

    def suite():
        test_suite = TestSuite()
        test_suite.addTest(TestPerson('test.is.legal'))
        return test_suite

if __name__ == '__main__':
    runner = TextTestRunner(stream=None, descriptions=True, verbosity=2)
    runner.run(suite())
```

INTEGRATION TESTING

- ▶ Group of individual units/components are tested
- ▶ Expose defects in the interaction between integrated units

INTEGRATION TESTING

```
from unittest import TestCase, main
from requests import get

class TestAPI(TestCase):
    def setUp(self):
        self.base_url = 'https://oosd-flask-api.herokuapp.com'
        self.api_url = '/api/videogames/'

    def test_url_is_ok(self):
        req = get(self.base_url)
        self.assertEqual(req.status_code, 200)

    def test_developer_is_atari(self):
        req = get(f'{self.base_url}{self.api_url}?id=0')
        self.assertEqual(req.json()[0].get('developer'), 'Atari')

    def test_title_is_donkey_kong(self):
        req = get(f'{self.base_url}{self.api_url}?id=1')
        self.assertEqual(req.json()[0].get('title'), 'Donkey_Kong')

    def test_year_release_is_1972(self):
        req = get(f'{self.base_url}{self.api_url}?id=2')
        self.assertEqual(req.json()[0].get('year.release'), 1972)

    def tearDown(self):
        self.base_url = None
        self.api_url = None

if __name__ == '__main__':
    main(argv=[''], verbosity=2, exit=False)
```

END-TO-END TESTING

- ▶ User interface or browser testing
- ▶ Testing the flow of an application from start to end
- ▶ Simulates a real user scenario
- ▶ Validates a system or systems under test & its components for integration & data integrity

END-TO-END TESTING: SELENIUM WEBDRIVER

- ▶ A collection of open source APIs
- ▶ Supports the automation of web browsers

END-TO-END TESTING

```
from unittest import TestCase, main
from selenium import webdriver

class TestGoogleSearch(TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome(
            '../chromedriver/chromedriver.mac')
        self.driver.get('https://google.com/')

    def test_search_in_google(self):
        self.assertEqual(True, 'Google' in self.driver.title)
        search_input = self.driver.find_element_by_xpath(
            '//*[@id="tsf"]/div[2]/div[1]/div[1]/div/div[2]/input')
        search_input.send_keys('Larry Page')
        search_btn = self.driver.find_element_by_xpath(
            '//*[@id="tsf"]/div[2]/div[1]/div[3]/center/input[1]')
        search_btn.click()
        self.assertEqual(
            True, 'Larry Page - Wikipedia' in self.driver.page_source)

    def tearDown(self):
        self.driver.close()

if __name__ == '__main__':
    main(argv=[''], verbosity=2, exit=False)
```

USER ACCEPTANCE TESTING

- ▶ Beta or end-user testing
- ▶ System is tested for acceptability
- ▶ Evaluating the system's compliance with the business requirements
- ▶ Assessing whether the system is acceptable for delivery

REGRESSION TESTING

- ▶ Ensures that changes to the application haven't adversely affected it
- ▶ New test cases aren't created
- ▶ Previously created test cases are re-executed

TEST-DRIVEN DEVELOPMENT

- ▶ Relies on the repetition of a short development cycle
- ▶ Requirements are turned into specific test cases
- ▶ The software is improved so that the tests pass

TEST-DRIVEN DEVELOPMENT CYCLE

- ▶ Add a new test - each new feature begins with writing a test
- ▶ Run the tests & see if the new test fails
- ▶ Write the code
- ▶ Run the tests & see if the new test passes
- ▶ Refactored the code
- ▶ Repeat