MONASH University

MONASH
INFORMATION
TECHNOLOGY

FIT9133 Semester 2 2019
Programming Foundations in Python

Week 11:
Testing and Exception Handling

Chunyang Chen
Gavin Kroeger

GROUP OF EIGHT AUSTRALIA

Module 5 Synopsis

- Module 5 is aimed to introduce you with:
  - Concepts of testing
    - Testing strategies
    - Unit testing in Python
  - Types of programming error
    - Syntax errors
    - Run-time errors
    - Logic errors
  - Exception handling

Module 5 Learning Objectives

- Upon completing this module, you should be able to:
  - Deploy different strategies for testing your programs
  - Construct appropriate handling code in Python for specific types of error or exception

Basic Concepts of Testing

Why Testing?

- Accidents: https://en.wikipedia.org/wiki/List_of_software_bugs

ALEX DAVIES TRANSPORTATION 03.13.19 08:26 PM

BOEING PLANS TO FIX THE 737 MAX JET WITH A SOFTWARE UPDATE

Uber self-driving car accident: Who's to blame when there's no driver?

The Conversation By Raja Jurdak and Salil S. Kanhere

Posted 20 Mar 2018, 4:55pm

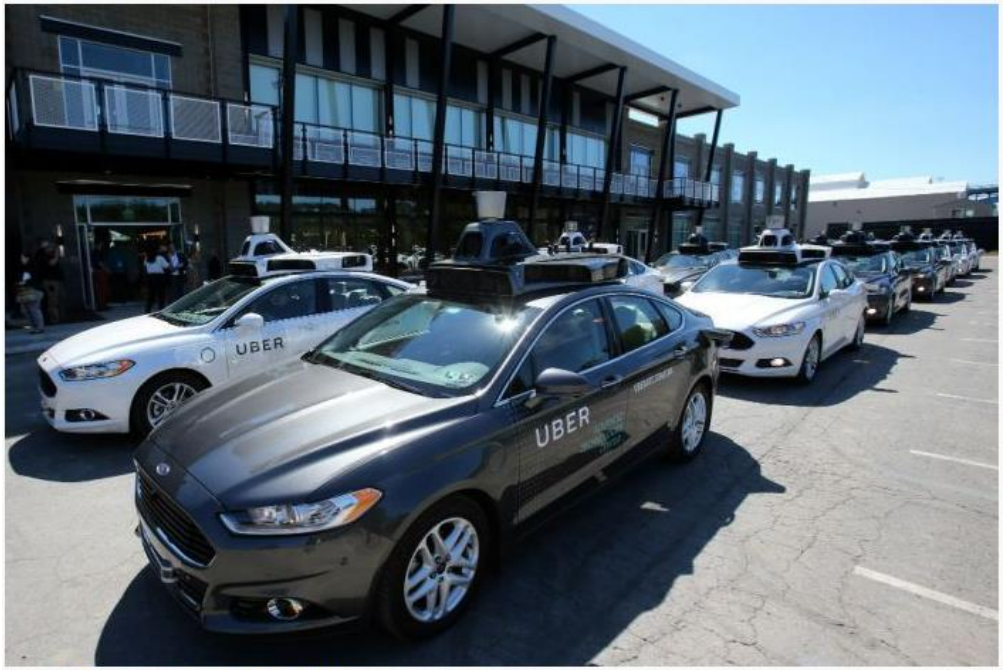
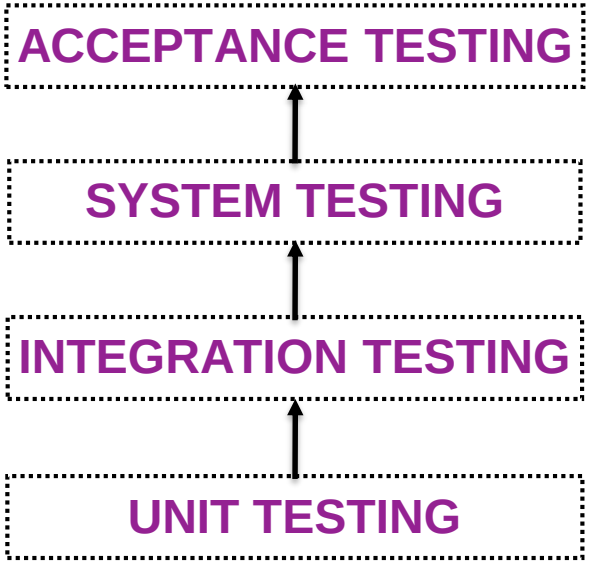


PHOTO: Autonomous cars are starting to hit the roads in pilot trials. (Reuters: Aaron Josefczyk)

Why Testing?

- Testing:
  - Determine the correctness and quality of your application (program)
  - Identify and rectify errors and defects in your application before its deployment by end users

- Levels of testing:



Levels of Testing

- Unit testing:
  - Individual units or components of a program are tested
  - Validate that each unit is fully functional without errors
- Integration testing:
  - Individual units are combined and tested as a group
  - Errors/defects might expose during interaction between integrated units
- System testing:
  - The complete integrated application is tested as a whole
  - Ensure that all the functionality and requirements are achieved
- Acceptance testing:
  - The complete application is tested by users before its deployment
  - Evaluate the system complies with all the business requirements

Our focus is on this

How Testing is Performed?

- Basic approach:
  - Define a test strategy with various test cases identified, which are important to ensure the correctness of your program

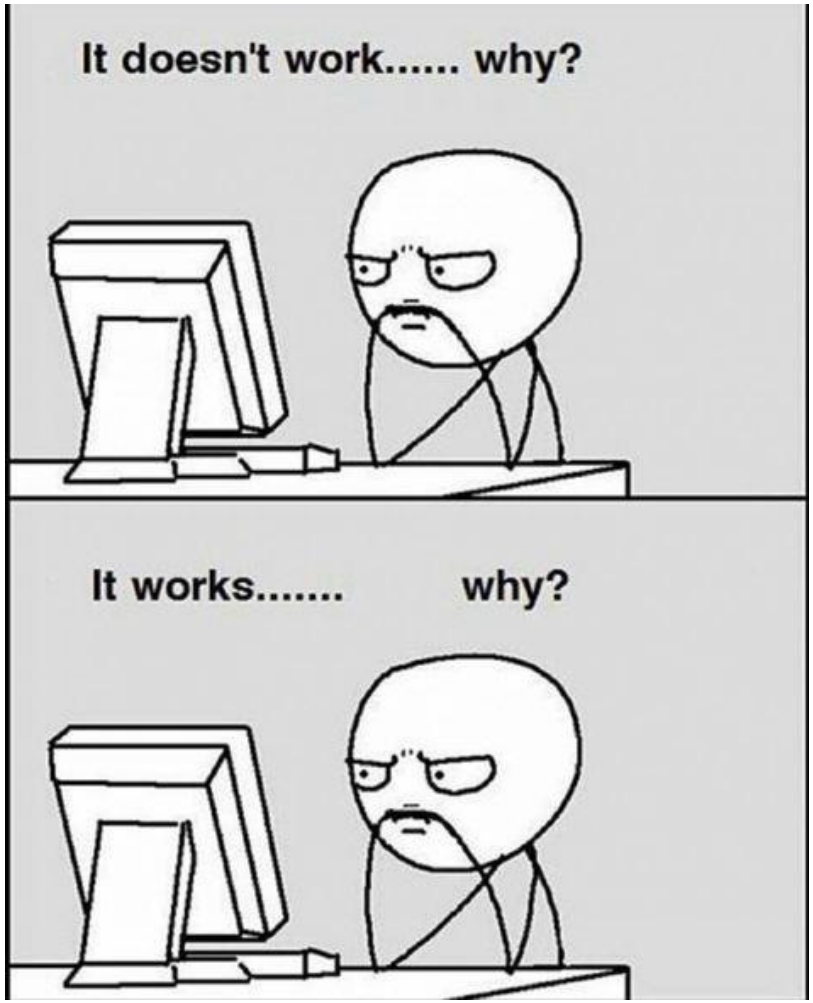
- "Good" testing strategy:
  - Make sure all the functionality can be covered/tested in a finite amount of testing time
  - Composed of reasonable and manageable number of test cases
  - Maximise the chance of detecting an error or a defect

Types of Test Cases

```
if mark >= 50 and mark <= 100:
    grade = "Passed"
else:
    grade = "Failed"
```

- Valid (positive) cases:
  - Based upon "correct" input data
  - Examples: 55, 60, 65, …, 85, 90, 95, …
- Invalid (negative) cases:
  - Based upon "incorrect" input data
  - Examples: -1, 0, 5, …, 45, 49, 101, 200, …
- Boundary cases:
  - Boundary values of the "equivalence class" for valid cases
  - Examples: (49, 50) and (100, 101)

Bugs in Your Program



Debugging in Python

- **Debugging** concept
  - The process of finding and resolving defects or problems within a computer program

- Basic debugging approaches:
  - `print` statements
  - `assert` statements

- `assert` statements:
  - Check a specific condition as specified by the program
  - Raise an AssertionError exception in Python if the condition fails
  - Syntax:
    ```
    assert (condition), "<error_message>"
    ```
  - Example:
    ```
    assert size <= 5, "size should not exceed 5"
    ```

- **The best debugging is to understand your program**

https://tenor.com/view/business-cat-working-cat-boss-angry-gif-13655998

Unit Testing in Python

- **`unittest`**:
  - Create a test class by *subclassing* from **`unittest.TestCase`**
  - Define various test methods (test cases) within the test class

```
def product_func(first_arg, second_arg):
    result = first_arg * second_arg
    return result
```

```
import unittest

class TestForProduct(unittest.TestCase):

    def test_product(self):
        self.assertEqual(product_func(2, 4), 8)

if __name__ == '__main__':
    unittest.main()
```

Three possible outcomes: OK, FAIL or ERROR.

- Running **`unittest`** on Jupyter Notebook:

```
def product_func(first_arg, second_arg):
    result = first_arg * second_arg
    return result
```

```
import unittest

class TestForProduct(unittest.TestCase):

    def test_product(self):
        self.assertEqual(product_func(2, 4), 8)
```

```
suite = unittest.TestLoader().loadTestsFromTestCase(TestForProduct)
unittest.TextTestRunner().run(suite)
```

The name of the test class

(More on) Unit Testing in Python

- **`assert`** methods provided by **`unittest.TestCase`**:

Method	Checks that
`assertEqual(a, b)`	`a == b`
`assertNotEqual(a, b)`	`a != b`
`assertTrue(x)`	`bool(x) is True`
`assertFalse(x)`	`bool(x) is False`
`assertIs(a, b)`	`a is b`
`assertIsNot(a, b)`	`a is not b`
`assertIsNone(x)`	`x is None`
`assertIsNotNone(x)`	`x is not None`
`assertIn(a, b)`	`a in b`
`assertNotIn(a, b)`	`a not in b`
`assertIsInstance(a, b)`	`isinstance(a, b)`
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`

https://docs.python.org/3/library/unittest.html

MONASH University

Review Exercise: Part 1

Question 1

Which of the following is an *valid* test case for the given condition?

```
the_value > 5 and the_value < 10
```

A. 5
B. 10
C. 6
D. ‘6’
E. All of them

Question 2

Which of the following is *valid* for testing the given function?

```
import unittest

def product_function(first_arg, second_arg):
    result = first_arg * second_arg
    return result

class TestForFunction(unittest.TestCase):
    def test_product_function(self):
        self.???(product_function(2,3),6)
```

A. assertEqual()
B. assertIs()
C. assertTrue()
D. None of the above

Programming Errors and Exceptions

Types of Programming Error

- Syntax errors:
  - Code is not syntactically well formed and cannot be understood by the compiler/interpreter
  - Examples in Python: SyntaxError
- Run-time errors:
  - Occur during execution; errors that are anticipated and can be dealt with appropriately
  - Examples in Python: ValueError, TypeError, NameError
- Logic errors:
  - Incorrect implementation of the program's logic
  - Program runs without errors but result in unexpected output

Errors and Exceptions in Python

- SyntaxError:
  - Errors in the syntax of your program (parsing errors in Python)
- NameError: (exception in Python)
  - Attempt to use a variable (or value) before initialising it; attempt to use a module or function without first importing it
- TypeError: (exception in Python)
  - Attempt to use incompatible data types within a single statement; attempt to pass an argument of the wrong type
- ValueError: (exception in Python)
  - Attempt to pass an argument with the correct type but with a wrong value
- RuntimeError: (exception in Python)
  - An error detected that doesn't fall into any of the pre-defined error types

Example: Python Exceptions

- SyntaxError:

```
if a_number > 2
    print(a_number, "is greater than 2")
```

- NameError:

```
a_number = random.random()
```

- TypeError:

```
if a_number > 2:
    print(a_number + "is greater than 2")
```

- ValueError:

```
sum_of_two = int('1') + int('b')
```

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

MONASH
University

Exception Handling in Python

Exception Handling

- **try** and **except**:

```
try:
    num1 = int(input("Enter first number: "))
    num2 = int(input("Enter second number: "))
    result = num1 // num2
    print("Result of division:", result)
except ValueError:
    print("Invalid input value")
except ZeroDivisionError:
    print("Cannot divide by zero")
```

  - Statements within the **try** block are executed; if no exceptions occur, the **except** blocks are skipped
  - If an exception is *thrown* (i.e. occurred) and it matches one of the given **except** blocks, the corresponding print statement executes
  - If an exception occurs which is not specified by any of the **except** blocks, it is considered as an "unhandled" exception and the program will stop abruptly (i.e. crashed)

(More on) Exception Handling

- **`else`**:

```
file_name = "input_file.txt"

try:
    file_handle = open(file_name, 'r')
except IOError:
    print("Cannot open", file_name)
except RuntimeError:
    print("A run-time error has occurred")
else:
    print(file_name, "has",
          len(file_handle.readlines()), "lines")
    file_handle.close()
```

  - Comes after all the **`except`** blocks (optional)
  - Useful where some code should be executed if the **`try`** block does not raise any exception.

(More on) Exception Handling

- **`finally`**:

```
file_name = "another_input_file.txt"

try:
    file_handle = open(file_name, 'r')
except IOError:
    print("Cannot open", file_name)
except RuntimeError:
    print("A run-time error has occurred")
else:
    print(file_name, "has",
          len(file_handle.readlines()), "lines")
    file_handle.close()
finally:
    print("Exiting file reading")
```

  - As a "clean-up" action
  - Execute under all circumstances whether an exception has occurred or not; or whether the exceptions have been handled

MONASH University

Review Exercise: Part 2

Question 3

Which of the following exception will be thrown for the given program?

```
>>> a_list = ['1', '2', '3']
>>> print(a_list[3])
```

A. NameError
B. TypeError
C. IndexError
D. RuntimeError
E. None of the above

Question 4

Which of the following exception will be thrown for the given program?

```
def test_function(first_arg, second_arg):
    result = first_arg + second_arg
    return result

>>> print(test_function('1',2))
```

A. NameError
B. TypeError
C. ValueError
D. RuntimeError
E. None of the above

Question 5

Which of the following exception will be thrown for the given program?

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
    def get_x(self):
        return self.x
    def get_y(self):
        return self.y

>>> a_point = Point(1,2)
>>> a_point.set_x(2)
```

A. NameError
B. AttributeError
C. RuntimeError
D. None of the above

Week 11 Summary

- We have discussed:
  - Testing strategy
  - Unit testing in Python
  - Types of programming error
  - Exception handling in Python

- Next week:
  - Concepts of recursion
  - Recursive sorting algorithms