Introduction to Unit Testing. Exceptions

Lect. PhD. Arthur Molnar

Babes-Bolyai University

Overview

Testing

What is testing?

Observing the behavior of a program over many executions.

- We execute the program for some input data and compare the result we obtain with the known correct result.
- **Questions**:
  - How do we choose input data?
  - How do we know we have run enough tests?
  - How do we know the program worked correctly for a given test? (known as the oracle problem)

Testing

- Testing cannot prove program correctness, and cannot identify all defects in software. However, what it **can** prove is incorrectness, if at least one test case gives wrong results.
- **Problems with testing**
  - We cannot cover a function's input space
  - We have to design an oracle as complex as the program under test
  - Certain things are practically outside of our control (e.g. platform, operating system and library versions, possible hardware faults)

Unit Testing

- Tests that verify the functionality of a specific section of code, usually at function level.
- Testing is done in isolation. Test small parts of the program independently

How to test a function

1. Write the function's specification
2. Create a test function called *test_* $<$ *function_name* $>$ that has no input parameters, does no return anything and calls no functions except the one under test (e.g. *test_add_student*())
3. Add test cases to the test function using Python's assert[1] keyword
4. Run the test function. It should fail with an *AssertionError*
5. Write the code for the function under test
6. Test functions that do not raise *AssertionError* must complete quietly

[1] `https://docs.python.org/3/reference/simple_stmts.html#the-assert-statement`

Exceptions

An **exception** is an event that disrupts normal program flow

- Exceptions are present and used in many programming languages
- They are raised by code to signal an exceptional situation
- Your code will both raise (create) exceptions as well as "treat" them

NB!

The presence of an exception does not automatically mean that there's an error in the code

Exceptions

Most programming languages that support exceptions[2] use a common terminology and syntax

- Raising or throwing exceptions
- Catching or treating an exception
- Exception propagation
- **try** / **raise** (throw) / **except** (catch) keywords

[2] `https://docs.python.org/3/tutorial/errors.html#exceptions`

Exception handling

Exception handling is the process of handling error conditions in a program systematically by taking the necessary action.

```
try:
    # code that may raise exceptions
except ValueError:
    # code that handles the situation
```

Exception handling

A few points from the Python syntax above

- If you want to catch exceptions, the code has to be in a **try** - **except** block
- Exceptions are caught according to type
- A try block can catch **one**, **several** or **all** exception types
- Creating exceptions in your code is done using the **raise** keyword
- You can provide additional arguments (such as an error message) to exceptions you raise

Exception handling

Where is an exception handlded:

1. The function where the exception was raised
2. Any function that called the raising function (transitively)
3. The Python runtime, in which case program execution stops

Discussion

If the phrase *"unhandled exception has occurred in you application..."* sounds familiar, now you understand what happened!

Exceptions

Demo

Exceptions example, **ex08_Exceptions.py**

Exception handling

When to use exceptions?

- Signal an exceptional situation - the function is unable to do its work (e.g. function preconditions are violated, or the function encountered a situation in which it cannot make progress - a required file was not found, was not accessible, etc.)
- Enforce function preconditions
- Generally speaking, you should **not use** exceptions to control program flow!

Function specification

Is a way to abstract **functions** that only works if we provide:

- Meaningful function name
- Short description (the problem solved by the function)
- Type and meaning of each input parameter
- Conditions over the input parameters (**preconditions**)
- Type and meaning of each output parameter
- Relation between input and output parameters (**post condition**)
- **Exceptions** raised by the function

Exceptions and function specification

- **Precondition** - condition that must be true prior to running a section of code
- **Post condition** - condition that must be true after running a section of code

```
def gcd(a, b):
    '''
    Return the greatest common divisor of two
        positive integers
    a,b - integers
    Return the greatest common divisor of a and b
    Raise ValueError if a <= 0 or b <= 0
    '''
```