Topic 8

Testing, Debugging and Exception Handling

Murdoch
UNIVERSITY

Last Topic

Murdoch UNIVERSITY

- The concept of files and directories
- Read and write text files
- Open a CSV file
- Read and write CSV files
- Get operating system information
- Access the underlying file system

This Topic

Murdoch UNIVERSITY

- Testing
- Debugging
- Catch exceptions
- Raise exceptions
- Assertions
- Defensive programming

Definitions

Murdoch UNIVERSITY

- **Testing**: is an act of examining whether a given piece of program meets the program requirements
- **Debugging**: is the process of finding and fixing the program errors
- **Exception Handling**: is a mechanism to handle runtime errors or abnormal conditions in a program

Defensive Programming

Murdoch UNIVERSITY

- Write specifications for functions
- Modularise programs
- Check conditions on inputs/outputs

Testing/Validation

- Compare input/output pairs to specification
  - "it's not working"
  - "How can I break my program?"

Debugging

- Study events leading to an error
  - "Why is it not working?"
  - "How can I fix my program?"

What You Should Do

Murdoch UNIVERSITY

- From the start, design your code to ease the subsequent testing and debugging
- Break your program up into modules that can be tested and debugged individually
- Document the constraints of your modules
  - What do you expect the input to be?
  - What do you expect the output to be?
- Document assumptions behind the code design

When Are You Ready to Test?

Murdoch UNIVERSITY

- Ensure the code runs
  - Remove syntax errors
  - Remove static semantic errors
  - Python interpreter can usually find these for you
- Have a set of expected results
  - an input set
  - for each input, the expected output

Types of Tests

Murdoch UNIVERSITY

- Unit testing
  - validate each unit of the program, such as a function, a method in a class, a class, or a module
  - test each unit separately
- Regression testing
  - check whether previous changes or updates introduce new errors
  - catch reintroduced errors that were previously fixed
  - add test for bugs as you find them
- Integration testing
  - does overall program work?
  - tend to rush to do this

Testing Methods

- Intuition about the natural boundaries to the problem

```
def is_bigger(x,y):
    """ Assumes x and y are ints
    returns True if y is less than x , else False"""
```

- If no natural partitions, might do random testing
  - probability that code is correct increases with more tests
  - better options below
- Black box testing
  - explore paths through specification
- Glass box testing
  - explore paths through code

Black Box Testing

Murdoch UNIVERSITY

- Design the test cases without looking at the code
- This can be done by someone other than the implementer to avoid implementer biases
- The test can be reused if implementation changes
- Paths through specification
  - builds test cases in different natural space partitions
  - Also consider boundary conditions (empty lists, singleton list, large numbers, small numbers, etc)

Black Box Testing

Murdoch UNIVERSITY

```
def sqrt(x, eps):
    """ Assumes x, eps floats, x >= 0, eps > 0
    Returns res such that x-eps <= res*res <= x+eps """
```

CASE	x	eps
boundary	0	0.0001
perfect square	25	0.0001
less than 1	0.05	0.0001
irrational square root	2	0.0001
negative number	-4	0.0001
negative eps	35	-0.001
extremes	2	1.0/2.0**64.0
extremes	1.0/2.0**64.0	1.0/2.0**64.0
extremes	2.0**64.0	1.0/2.0**64.0
extremes	1.0/2.0**64.0	2.0**64.0
extremes	2.0**64.0	2.0**64.0

Glass Box Testing

Murdoch UNIVERSITY

- Use code directly to guide design of test cases
- Called path-complete if every potential path through code is tested at least once
- What are some drawbacks of this type of testing?
  - can go through loops arbitrarily many times
  - missing paths
- Guidelines
  - branches
    - exercise all branches of a conditional
  - for loops
    - loop not entered
    - body of loop executed exactly once
    - body of loop executed more than once
  - while loops
    - same as for loops, cases that catch all ways to exit loop

Glass Box Testing

```
def abs(x):
    """ Assumes x is an int
        Returns x if x>=0 and -x otherwise """
    if x < -1:
        return -x
    else:
        return x
```

- A path-complete test suite could miss a bug
- Path-complete test suite: 2 and -2
- But `abs(-1)` incorrectly returns -1
- Should still test boundary cases

Debugging

Murdoch UNIVERSITY

- Steep learning curve
- Goal is to have a bug-free program
- Tools
  - IDLE has a built-in debugger
  - Python tutor (https://pythontutor.com/), which can visualise the execution of your code
  - Several other IDEs which come with their integrated debugging tools, such as PyCharm, VS Code, Jupyter, Spyder, and Atom.
  - `print` statement
  - Use your brain, be systematic in your hunt

Print Statement

Murdoch UNIVERSITY

- Good way to test hypothesis
- When to print
  - enter function
  - parameters
  - function results
- Use bisection method
  - put print halfway in code
  - decide where bug may be depending on values

Debugging Steps

Murdoch UNIVERSITY

- Study program code
  - don’t ask what is wrong
  - ask how did I get the unexpected results
- Scientific method
  - study available data
  - form hypothesis
  - repeatable experiments
  - pick simplest input to test with

Error Message - Easy

- Trying to access beyond the limit of a list

```
test = [1,2,3] then test[4]                    -> IndexError
```

- Trying to convert an inappropriate type

```
int(test)                                      -> TypeError
```

- Referencing a non-existent variable

```
a                                              -> NameError
```

- Mixing data types without appropriate coercion

```
'3'/4                                          -> TypeError
```

- Forgetting to close parenthesis, quotation, etc

```
a = len([1,2,3)
print(a)                                       -> SyntaxError
```

Logic Errors - Hard

Murdoch UNIVERSITY

- Think before writing new code
- Draw pictures, take a break
- Explain the code to
  - Someone else
  - A rubber ducky (explain you code to a toy duck)

Logic Errors - Hard

Murdoch UNIVERSITY

DON'T

- Write entire program
- Test entire program
- Debug entire program

- Change code
- Remember where bug was
- Test code
- Forget where bug was or what change you made
- Panic

DO

- Write a function
- Test the function, debug the function
- Write a function
- Test the function, debug the function
- *** Do integration testing ***

- Backup code
- Change code
- Write down potential bug in a comment
- Test code
- Compare new version with old version

Exceptions

Murdoch UNIVERSITY

- What happens when program execution hits an unexpected condition?
- Python interpreter throws you an exception (to what was expected)
  - trying to access beyond list limits

```
test = [1,7,4]
x = test[4]                    -> IndexError
```

  - trying to convert an inappropriate type

```
x = int(test)                  -> TypeError
```

  - trying to open a non-existent file

```
file = open("foo")             -> FileNotFoundError
```

  - trying to open a read-only file for writing

```
file = open("test.txt", "w")   -> PermissionError
```

More Examples of Exceptions

Murdoch UNIVERSITY

- Some of the common exceptions:
  - `SyntaxError`: Python can't parse the program
  - `NameError`: local or global name not found
  - `TypeError`: operand doesn't have correct type
  - `ValueError`: operand type ok, but value is illegal (eg, `int("hello")`
  - `IOError`: IO system reports malfunction (eg, file not found)

Why Exception Handling?

Murdoch UNIVERSITY

- When something that is not what you usually expected occurs, there are two ways to handle it.
- Take the example of reading student records from a file and print out the names of those who are enrolled in MIT course, the main thrust of the logic is

```
1. students = open( the file )
2. for st in students:
3.     if st is an MIT student:
4.         print the name of the student
5. close the file
6. print "bye-bye"
```

- The above program logic is simple, clear and easy to follow.

Why Exception Handling?

Murdoch UNIVERSITY

- What would happen if the file doesn't exist, or there is no permission to read the file?
  - to deal with these types of "abnormal" conditions, we need to add more steps

```
1. students = open( the file )
2. if the file doesn’t exits:
3.     print the file doesn’t exist
4.     exit
5. elif no permission to read the file:
6.     print cannot read the file
7.     exit
8. for st in students:
9.     if st is an MIT student:
10.        print the name of the student
11. close the file
12. print "bye-bye"
```

Why Exception Handling?

Murdoch UNIVERSITY

- The new version is obviously more correct than the first version, but it also
  - obscure the primary task of the program – to read a student file and print out the names of MIT students!
  - distract our attention from the main task of the program
    - your attention is equally shared between dealing an abnormal conditions (file does not exist, no read permission to the file) and getting the names of MIT students.
- This is the first approach.

Why Exception Handling?

Murdoch UNIVERSITY

- The second approach is treating the no file or no read permission as exceptional conditions, and they are handled elsewhere, not allowing the handling of these abnormal conditions to distract you from the main task of your program!
- Most modern programming languages support this approach with the *try–except* statement (Python syntax):

```
try:
     the main task
except abnormal condition 1:
     handling abnormal condition 1
except abnormal condition 2:
     handle abnormal condition 2
except:
     handle all other abnormal conditions
```

How Does It Work?

Murdoch UNIVERSITY

- When something abnormal occurs, an `Exception` object is thrown (or raised). This object contains the details of the abnormal condition.
- The program may choose to catch this exception object or ignore it.
  - However, if you ignore it, your program will crash!
- For example, when you call function `open` to open a file, and the system find the file doesn't exist, the `open` function would raise an `FileNotFoundError` exception.
  - You may catch this exception and handle it in a separate place (an `except` clause)
  - or you may ignore it – your program will crash!

Re-visit the Example

Murdoch UNIVERSITY

- Let's re-write the pseudo-code with the try-except statement:

```
1.  try:
2.      students = open( the file )
3.      for st in students:
4.         if st is an MIT student:
5.            print the name of the student
6.      close the file
7.      print "bye-bye"
8.  except (FileNotFoundError, PermissionError):
9.      print the error message
10. except:
11.     print error opening file
```

- The main task and the abnormal conditions are processed in different places, allowing you to focus on the primary task of the program!

Multiple `except` Clauses

Murdoch UNIVERSITY

- The try-except statement may have multiple except clause, each handling one type of error.

```
try:
    a = int(input("Tell me one number: "))
    b = int(input("Tell me another number: "))
    print("a/b = ", a/b)
    print("a+b = ", a+b)
except ValueError:
    print("Could not convert to a number.")
except ZeroDivisionError:
    print("Can't divide by zero!")
except:
    print("Something went wrong!")
```

Note: the code in an `except` clause is executed only if an exception is raised in the `try` block. If an exception is raised but it is not `ValueError` or `ZeroDivisionError`, the code in the last `except` clause is executed.

The `else` and `finally` Clause

Murdoch UNIVERSITY

- The *`try-except`* statement may have an optional `else` clause and optional `finally` clause:

```
try:
      statements
except:
      statements
else:
      statements (executed only if there is no exception)
finally:
      statements (always executed, used to clean-up
      code that should be run no matter what else
      happened, eg, close a file)
```

What to Do with an Error?

Murdoch UNIVERSITY

- What to do when encounter an error?
- Fail silently:
  - Substitute default values or just continue
  - Bad idea! User gets no warning
- Return an "error" value
  - What value to choose?
  - Complicates code having to check for a special value and handle the error at the same place
- Stop execution, signal an error condition
  - raise an exception

```
raise Exception("a descriptive message")
```

Raise an Exception

Murdoch UNIVERSITY

- When unable to produce a result consistent with the function's specification, raise an exception:

```
raise <exception>(<argument>)
```

keyword

name of error you want to raise

optional, but typically a string with a message

- Example:

```
raise ValueError("something is wrong")
```

Example: Raise an Exception

Murdoch UNIVERSITY

```
def get_ratios(L1, L2):
    """ Assumes: L1 and L2 are lists of equal length of numbers
        Returns: a list containing L1[i]/L2[i] """
    ratios = []
    for i in range(len(L1)):
        try:
            ratios.append(L1[i]/L2[i])
        except ZeroDivisionError:
            ratios.append(float('nan')) # nan = not a number
        except:
            raise ValueError("get_ratios called with a bad arg")
    return ratios

n1 = [3.9, 10.5, "pi", 23.1]
n2 = [4.5, 0.0,  7.9,  91.1]
try:
    n1_over_n2 = get_ratios(n1, n2)
except ValueError:
    print("There are bad numbers in the lists")
```

Assertions

Murdoch UNIVERSITY

- Want to be sure that assumptions on state of computation are as expected?
- Use an `assert` statement to raise an `AssertionError` exception if assumptions are not met:
  - This is an example of good defensive programming

```
def avg(marks):
    assert len(marks) != 0, 'no marks data'
    return sum(marks) / len(marks)
```

- Raise an `AssertionError` if the function is called with an empty list for marks.
- Otherwise runs ok

Assertions As Defensive Programming

Murdoch UNIVERSITY

- Assertions don’t allow a programmer to control response to unexpected conditions.
- Ensure that execution halts whenever an expected condition is not met.
- Typically used to check inputs to functions but can be used anywhere.
- Can be used to check outputs of a function to avoid propagating bad values.
- Can make it easier to locate a source of a bug!

Where to Use Assertions

Murdoch UNIVERSITY

- Goal is to spot bugs as soon as introduced and make clear where they happened.
- Use as a supplement to testing.
- Raise exceptions if users supplied bad data input.
- Use assertions to
  - Check types of arguments or values
  - Check that invariants on data structures are met (eg, in a sorted list of numbers, an earlier number should not be larger than a later number)
  - Check constraints on return values
  - Check for violations of constraints, eg, no duplicates in a list

Summary

Murdoch UNIVERSITY

- Testing
- Debugging
- Catch exceptions
- Raise exceptions
- Assertions
- Defensive programming

Acknowledgement

- geekforgeek
- Python Tutor
- MIT OCW