COMP10001 Foundations of Computing
Testing and Debugging

Semester 2, 2018

Chris Leckie & Nic Geard

THE UNIVERSITY OF MELBOURNE

Reminders

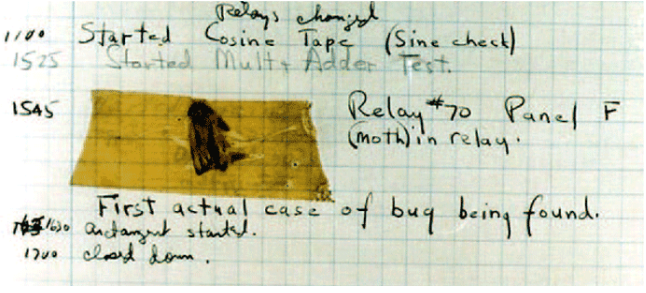
- Project 1 due this week
- No Grok worksheets due next week
- Mid-semester test will be 1pm Wednesday 5 September (venue details will be announced via the LMS)
- Practice tests are available on the MST and Exam page in the LMS

Lecture Agenda

- Last lecture:
  - File access
  - `defaultdict`
  - List Comprehensions
- This lecture:
  - Testing and debugging
  - Commenting
  - The call stack

Bugs

- A (software) "bug" is an error/flaw in a piece of code that leads to a malfunction
- The first attested computer "bug" (Grace Hopper, Harvard Mark II):



- So what's the big deal?

A Bug in Action: Mars Climate Orbiter

- Ideal: establish an orbit around Mars, and study the weather, climate, etc of Mars in tandem with the Mars Polar Lander
- Actuality: attempted to orbit too low and crashed as a result
- Cause: metric vs. Imperial conversion in calculations
- Cost: US$165m

Other Famous Bugs

- Y2K
- HAL 9000 (2001: A Space Odyssey)
- Estimate that software bugs cost the US economy 0.6% of the GDP
- Over 50% of the development cost of software is on testing and debugging
- No general way of "proving" that a given piece of software implements a given spec

Debugging

- Bugs are inevitable:
  - Fact: even the most carefully-engineered software will include at least 5 errors/1000 lines of code
  - Fact: Windows XP contained roughly 45m lines of code ...
- Bug/error types (revision):
  - **syntax errors** = incompatibility with the syntax of the programming language
  - **run-time errors** = errors at run-time, causing the code to crash
  - **logic errors** = design error, such that the code runs but doesn't do what it is supposed to do
- **Debugging** = the process of systematically finding and fixing bugs

Class Exercise: Spot and Fix the Bugs

Spot and fix the bug(s) in the following code, and classify each as a syntax, run-time or logic error:

```
1 def substrn(sup,sub)
2     sub_len = len(Sub)
3     for i in range(len(sup)-sub_len):
4         if sup[i:i+sub_len] == sub:
5             n += 1
6     print ("n")
```

Comments

Job much easier if there was a description of what the function should do.

```
1  def substrn(sup,sub)
2      '''
3      Return the number of times sub
4      occurs in sup.
5      '''
6      sub_len = len(Sub)
7      for i in range(len(sup)-sub_len):
8          if sup[i:i+sub_len] == sub:
9              n += 1
10     print ("n")
```

Functions and Docstring-style Commenting I

- A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the `__doc__` special attribute of that object.

```
def Celcius2Fahrenheit(n):
    """Return the (float)
    Fahrenheit equivalent
    of a temperature in Celcius"""
    return(9.0*n/5 + 32)
```

Functions and Docstring-style Commenting II

- It is possible to access the `__doc__` for a function via `help`, e.g. given:

```
def seconds_in_year(days=365):
    """Calculate seconds in a year"""
    return days*24*60*60
```

```
>>> help(seconds_in_year)
Help on function seconds_in_year
in module __main__:

seconds_in_year(days=365)
    Calculate seconds in a year
```

Comments

```
def f(x):
    '''This is a function of parameter x
    that calculates the length of x
    squared.'''
    return(len(x)**2)
```

Don't describe Python syntax; the reader knows Python.

```
def len_sqr(x):
    '''Return the square of the length of x.'''
    return(len(x)**2)
```

Comments

```
c = 0                   # a variable to count
for i in word:          # a loop
    if i in 'aeiou':    # is i in aeiou
        c += 1          # add one to count
```

Succint description of logic for a block in English.
Meaningful variable names help readability.

```
# Set count to the number of vowels in word
count = 0
for character in word:
    if character in 'aeiou':
        count += 1
```

Comments

```
# Set count to the number of vowels in word
count = 0
for character in word:
    if character in 'aeiou':
        count += 1
```

Use functions.

```
def count_vowels(word):
    '''Return the number of vowels in word.'''
    count = 0
    for character in word:
        if character in 'aeiou':
            count += 1

    return(count)
```

PEP8

- Programming Style Guide on the LMS under Projects
- No more than 79 characters per line. Yes! Really!
- 4 spaces per indent - preferred. Leave Grok default (2) for Project 1 if you are finished.

Prevention Rather than Cure: Defensive Programming

- Build up your code bit by bit, using functions copiously, testing as you go against known inputs/outputs
- "Log" each step of your progress
- Use comments to remind you about any assumptions made by the code/corners cut along the way
- Always be on the lookout for common gotchas
- Above all, remember that the program code must communicate with humans, not just machines

A General Approach to Debugging

- Reproduce the bug
- Determine exactly what the problem is
- Eliminate "obvious" causes (e.g. *Is it plugged in?*)
- Divide the process, separating out the parts that work from the part(s) that don't ("isolate" the problem)
- When you reach a dead end, reassess your information; then step through the process again
- As you proceed, make predictions about what should happen and verify the outcome

Common Python Gotchas

- Equality (==) vs. assignment (=)
- Printing vs. returning from functions
- Correct use of types (e.g. `False` vs. `"False"`)
- Incorrect use of function/method (e.g. `return list.sort()`)
- Spelling and capitalisation
- Loops and incrementing
- Conditionals and indentation
- Namespace problems

Tracing Functions: The Call Stack I

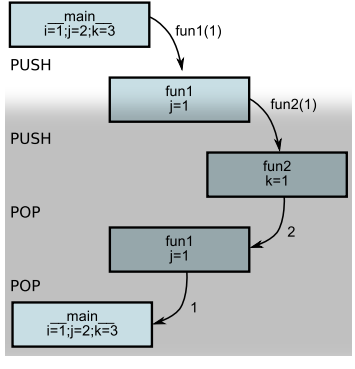
- We get some hints about how function "nesting" works from the python interpreter:

```
def plus_one(i):
    return(k + 1)
print(plus_one(2))
```

```
Traceback (most recent call last):
  File "program.py", line 13, in <module>
    print(plus_one(2))
  File "program.py", line 11, in plus_one
    return(k + 1)
NameError: name 'k' is not defined
```

Tracing Functions: The Call Stack II

- Functions are stored on the "call stack", facilitating function nesting, allowing functions to communicate with one another, and also preserving a function's local state/namespace



Lecture Summary

- What is a bug/debugging?
- Why do bugs occur?
- What different types of bugs are there?
- What is the basic procedure for identifying bugs?
- What are docstrings?
- What is the call stack?