• Testing and Debugging

• Midterm 2 Q/A

Even more than the act of testing, the act of designing tests is one of the best bug preventers ...

Example

```python
def any_upper(astring):
    """(str) -> Bool

    Return True if any character in astr is uppercase, and there is at least one cased character in astring, else return False

    >>> any_upper('aBc')
    True
    ""
```
What is the result when the example code is executed?  
**(be careful)**

def any_upper(astring):
    for ch in astring:
        # Executing any_upper will
        if ch.isupper():
            # (a) always return correct res
            return True
            # (b) sometimes return ...
        else:
            # (c) never return correct res
            return False

Careful program design and testing is needed!

Testing starts at program design step.

Tests can show the absence of many bugs, but they can’t show a program is correct.
Develop and execute a good suite of test cases to increase confidence in the code. →

And re-execute every time a change is made. (automated testing/doctest).

Testing

Aspects of Testing

• program runs
• program does not have bugs
• program handles the unexpected gracefully
• program runs under extreme conditions
• program is usable
Testing

Unit testing
-- look at one isolated component (could be one line or part of a line of code)

Integrative testing
-- looks at behavior of the whole line/function/program/system

Start with unit testing and move to integrative.

Testing

Testing starts at program design time

Program/Function design checklist:
Inputs/outputs and types
Short description of function
Examples of how the function should work – these are also test cases
Testing

Testing starts at design time
Examples of how the function works

- **simple cases** - can be done by hand
- **general cases** – all paths
- **boundary cases** – 0, 1, large numbers, negative numbers, empty string, string len 1, etc.

Testing/Example – Problem Description

Given two days in a single year (numbers between 1 and 365), determine the number of full weeks that have elapsed between the two days.
Testing/Example – Problem Specification

Write a function `wks_between`, with two parameters, `day1` and `day2`, which are days in the year (numbers between 1 and 365). The function should return the number of full weeks that have elapsed between `day1` and `day2`.

```python
def wks_between(day1, day2):
    '''(int, int) -> int

    Return number of full weeks that have elapsed between the two days (in the same year).

    >>> wks_between(3, 20)
    2
    '''
    pass
    return #weeks
```
Testing – Beyond Examples

```python
>>> wks_between(3, 20)    # normal
2
>>> wks_between(8, 5)     # simple/boundary
0
>>> wks_between(40, 61)   # normal
3
>>> wks_between(1, 8)     # simple/boundary
1
>>> wks_between(365, 1)   # boundary
52
```

Develop an algorithm ...

1. Find number of days between day1 and day2

2. Convert days-difference to weeks-difference (divide by 7, ignore remainder)

... then review Python toolkit
import doctest

def wks_between(day1, day2):
    """(int, int) -> int

    Return number of full weeks that have elapsed between the two days (in the same year).
    ""

    >>> [all of the test cases]
    ""
    diff = day2 - day1
    weeks = diff // 7
    return wks

Python toolkit, 1.
Data types - operators and built in functions

- numeric -
  +, -, *, /, //, %, **, <, >, <=, >=, !=, abs, pow, round, int, float, min, max

- string –
  +, *, <, >, <=, >=, !=, len, input
  string methods - count, find, index, isupper, islower, isalnum, endswith, startswith, upper, lower, ...

- Boolean - not, and, or

- sequential data types (e.g., string): min, max, [], :, for

- multiple data types – help, id, print, int, float, str
CIS 122 Introduction to Programming Spring 2017   Week 6, 2.
Python toolkit, 2.
Other data types -
  • functions (executable)
  • None

variable assignment
expression evaluation

import
modules – math, turtle, random
automated testing – doctest module

function definition (def)
function docstrings (type contract, description, examples of use)
executing (calling) a function

conditionals/selection (if elif else)
loops (while and for)

CIS 122 Introduction to Programming and Problem Solving   Spring 2017

Summary of Topics for Midterm 2

• Python toolkit – tools and use
• User-defined functions:
  -- program/function design recipe
  -- function header and docstring
  -- parameters, arguments, local vars
  -- functions are expressions; they evaluate to a value
  -- return statement, returned values, None
  -- what happens when a function is called
  -- function namespace
• Conditionals/selection (if elif else)
• Loops (for while)
CIS 122 Introduction to Programming and Problem Solving  Spring 2017

• Programming = CT + Coding
• High level languages: Python, etc.
• Python primitive elements
  Objects (type/value(s)/id)
• Type – values and operations
• Primitive elements can be combined
• Expressions evaluate to a value
• Assignment statements (are not expressions) associate name/object
• Variables are expressions
• Order of operations

CIS 122 Introduction to Programming and Problem Solving  Spring 2017

• Syntax, Runtime, and Logic Errors
• Good programming style (whitespace, variable names, docstrings)
Errors

• **Syntax** - program won’t run, for example
  >>> def myfunc(a, b)
      print(‘hello, world’)

• **Runtime** - program runs until it can’t, for example
  >>> value = true

• **Logical** - program runs but result is incorrect
  >>> def add1(x):
      return x + 2
  >>> add1(99)

Other runtime errors, for example

NameError
TypeError
SyntaxError
IndexOutOfRangeError
DivisionbyZeroError
Be alert

• assignment statement v. expression evaluation
• **defining and calling functions**
• **parameters and arguments**
• function returned values and side effects
• functions always return a value (it may be None)
• variables v. strings v. Boolean values
• parameters v. local variables v. input
• flow of control – watch blocks of code

---

<table>
<thead>
<tr>
<th>Return</th>
<th>Print/Side Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. yes/None</td>
<td>yes   (side effect is the point)</td>
</tr>
<tr>
<td>2. yes/value</td>
<td>no     (returned value is the point)</td>
</tr>
<tr>
<td>3. yes/None</td>
<td>no   (odd)</td>
</tr>
<tr>
<td>4. yes/value</td>
<td>yes  (somewhat unusual)</td>
</tr>
</tbody>
</table>

```python
def check1():
    s = 'hello, world'
    print(s)
    return None

def check2():
    s = 'hello, world'
    return s

def check3():
    s = 'hello, world'
    return None

def check4():
    s = 'hello, world'
    print(s)
    return s
```
Thinking:

- give example(s) of use
- write function header
- write type contract
- write brief description
- write test examples (simple, normal, boundary)
- develop an algorithm

Turn it into Python code:

- write docstring (+ pass and return None)
- review Python toolkit
- write Python code

Test code using all examples (automated testing)
Revise and retest as needed