• Midterm 2 Review/Q/A

• Testing and Debugging, cont’d

• Python Lists

“When you express your understanding in code, you debug your brain.”

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
'''
```
Tests for any_upper

>>> any_upper('A')  
True
>>> any_upper('a')  
False
>>> any_upper('Abc')  
True
>>> any_upper('abc')  
False
>>> any_upper('aBc')  
True
>>> any_upper('abc')  
False
>>> any_upper('abC')  
True
>>> any_upper('122')  
False

What is the result when the example code is executed? (be careful)

def any_upper(astring):
    '''
    for ch in astring:
        if ch.isupper():
            return True
        else:
            return False
    '''

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

def any_upper(astring):
    for ch in astring:
        if ch.isupper():
            return True
    return False

>>> any_upper('aBc')
>>> any_upper('abc')
```python
>>> is_anagram('hello', 'goodbye')
False
>>> is_anagram('mite', 'emit')
True
>>> is_anagram('good', 'goodbye')
False
>>> is_anagram('gogo', 'gooo')
False
```

def is_anagram(word1, word2):
    '''(str, str) -> Boolean'''
    if len(word1) != len(word2):
        return False
    for ch in word1:
        if ch not in word2:
            return False
        else:
            # remove found char from word2
            pos = word2.find(ch)
            word2 = word2[:pos] + word2[pos+1:]
    return True
```
Recall - Errors

Syntax

Runtime

Semantic/logical

Debugging Step 0 – design good code to avoid/minimize bugs!

- Short description of function
- Examples/Well-chosen test cases

Function design recipe:
- Inputs/outputs (parameter types, returned value)

Algorithm [be clear]

Python toolkit/Code [test sub/units/use Shell]

Testing (and Debugging) [use automated testing]
CIS 122 Introduction to Programming and Problem Solving  Spring 2017

Debugging Recipe

Step 0 – Careful design of algorithm and code, including well chosen test cases

Step 1 – Determine WHAT is happening, i.e., repeatable, isolated, unexpected result

Step 2 – Determine WHY this is happening. (Concentrate on finding why the program is doing what it is doing - not why it isn’t doing what you want it to.)

Step 3 – Change one thing at a time, for a reason. Keep a backup copy of the program prior to making this change!

Step 4 – Once you’ve determined that your change has fixed the particular bug, re-run the entire test suite (easy with automated testing).

CIS 122 Introduction to Programming and Problem Solving  Spring 2017

From Testing to Debugging

(Novice) programming errors $\rightarrow$ fixes

(1) Disengage from the task when trouble occurs
   $\rightarrow$ expect bugs; leave time for debugging

(2) Neglect to track closely what programs do
   $\rightarrow$ know what output you are expecting
From Testing to Debugging
( Novice ) programming errors

( 3 ) Try to repair bugs by haphazardly tinkering with code
  → ( sub - ) unit testing helps contain the bug
  → keep a copy of last working version

( 4 ) Have difficulty breaking problems down into parts suitable for separate chunks of code
  → good program design / keep functions small

Lists – Why

For example,

to check whether a character is a vowel ... can use a string of vowels ...
Lists – Why

For example,

check passwords against frequently used passwords

??

Lists – What

>>> mypass = ‘hello’
>>> frequent_psw = [‘abc123’, ‘password’, …]
Lists – What

>>> mypass = ‘hello’
>>> frequent_psw = [‘abc123’, ‘password’, …]

>>> for item in frequent_psw:
    if mypass == item:
        return False
    else:
        return True  # be careful
Lists – What

```python
>>> mypass = ‘hello’
>>> frequent_psw = [‘abc123’, ‘password’, …]

>>> mypass in frequent_psw
```

Python lists can be heterogeneous

```python
>>> mixitup = [“hello”, “goodbye”, True, 99, 99.0, [‘a’]]

>>> mixitup
??
>>> [2, 4, 6, 8]
??
```
Lists are sequences

so the sequence operators and built in functions we’ve seen for strings work for lists, too –

\[
[ ], +, *, [ :], \text{len, max, min, in}
\]

>>> example = [5, True, ‘string’, 99.0]
>>> example[1:]
>>> example[:3]
>>> example[1:3]

also - sorted, sum, del
Lists are sequences

so the sequence operators and built in functions we’ve seen for strings work for lists, too – 

[ ], +, *, [:], len, max, min, in

also - del, sorted, sum

and – another string method, split -

>>> 'C2O H2O'.split()

['C2O', 'H2O']

mixitup = [“hello”, “goodbye”, True, 99, 99.0]

len(mixitup)

“goodbye” in mixitup

“good” in mixitup

“good” in “goodbye”

mixitup[1:3]

mixitup[:]

mixitup + mixitup

mixitup * 2
>>> stuff = [2, 25, 80, 12]

>>> sorted(stuff)

>>> sum(stuff)

>>> stuff.count(12)

Lists are mutable – can update one item at a time

mixitup = ['hello', 'goodbye', True, 99, 99.0]

mixitup[0] = 'bonjour'
mixitup[1] = 'adios'
del mixitup[-1]

compare: check = 'abc'
check[0] = 'z'
Lists are mutable!

>>> stuff = [2, 25, 80, 12]

>>> stuff[1] = 99

>>> stuff[stuff[0]] = stuff[3]

>>> stuff
[2, 99, 12, 12]

>>> stuff.count(12)
3

>>> del stuff[0]

>>> stuff
[2, 99, 12]
Lists are mutable – be careful – list methods

```python
>>> list1 = ['a', 'b', 99]
>>> list1 = list1.reverse()
>>> list1
??
```

```
Lists are mutable – be careful - list methods

```python
>>> list1 = ['a', 'b', 99]
>>> list1 = list1.reverse()
>>> list1
None

list is updated in place, as a side effect

None value is returned
```
Lists are mutable – be careful

```python
>>> list1 = ['a', 'b', 99]    >>> w1 = 'ducks'
>>> list1 = list1.reverse()  >>> w1 = w1.capitalize()
>>> list1    # unexpected    >>> w1
??          ??
```

```python
>>> list1.reverse()
>>> list1    # expected
??
```

Lists are mutable

be very aware of what list methods do and what values they return
```python
CIS 122 Introduction to Programming and Problem Solving  Spring 2017

>>> stuff = [5, 4, 3]
>>> sorted(stuff)
??
>>> stuff
??
>>> stuff.sort()
??
>>> stuff
??
>>> stuff.reverse()
??
>>> stuff
??
>>> stuff = stuff.sort()
>>> stuff
??

CIS 122 Introduction to Programming and Problem Solving  Spring 2017

>>> stuff = [5, 4, 3]
>>> sorted(stuff)
[3, 4, 5]
>>> stuff
[5, 4, 3]
>>> stuff.sort()
>>> stuff
[3, 4, 5]
>>> stuff.reverse()
>>> stuff
[5, 4, 3]
>>> stuff = stuff.sort()
>>> stuff
>>> print(stuff)
None
```