

CS 1110

Prelim 1 Review
Fall 2014

Exam Info

- Prelim 1: 7:30–9:00PM, Thursday, October 16th
 - Last name **A – Gr** in Ives 305
 - Last name **Gu – z** in Statler Auditorium
 - SDS Students will get an e-mail
- To help you study:
 - Study guides, review slides are online
 - Solutions to Assignment 2 are in CMS
- Arrive early! Helps reducing stress

Grading

- We will announce *approximate* letter grades
 - We adjust letter grades based on all exams
 - But no hard guidelines (e.g. mean = grade X)
 - May adjust borderline grades again at final grades
- Use this to determine whether you want to drop
 - **Drop deadline** is next day, October 17th
 - **Goal:** Have everyone graded by noon of that day
 - Will definitely notify you if you made less than C

What is on the Exam?

- **Five** Questions out of Six Topics:
 - String slicing functions (A1)
 - Call frames and the call stack (A2)
 - Functions on mutable objects (A3)
 - Testing and debugging (A1, Lab 3, Lec. 10)
 - Lists and For-Loops (Lab 6 and 7)
 - Short Answer (Terminology)
- + 2 pts for writing your name and net-id

What is on the Exam?

- String slicing functions (A1)
 - Will be given a function specification
 - Implement it using string methods, slicing
- Call frames and the call stack (A2)
- Functions on mutable objects (A3)
- Testing and debugging (A1, Lab 3, Lecture 10)
- Lists and For-Loops (Lab 6 and 7)
- Short Answer (Terminology)

String Slicing

```
def make_netid(name,n):
```

```
    """Returns a netid for name with suffix n
```

```
    Netid is either two letters and a number (if the student has no  
    middle name) or three letters and a number (if the student has  
    a middle name). Letters in netid are lowercase.
```

```
    Example: make_netid('Walker McMillan White',2) is 'wmw2'
```

```
    Example: make_netid('Walker White',4) is 'ww4'
```

```
    Precondition: name is a string either with format '<first-name>  
<last-name>' or '<first-name> <middle-name> <last-name>';  
    names are separated by spaces. n > 0 is an int."""
```

Useful String Methods

Method	Result
<code>s.find(s1)</code>	Returns first position of <code>s1</code> in <code>s</code> ; -1 if not there.
<code>s.rfind(s1)</code>	Returns LAST position of <code>s1</code> in <code>s</code> ; -1 if not there.
<code>s.lower()</code>	Returns copy of <code>s</code> with all letters lower case
<code>s.upper()</code>	Returns copy of <code>s</code> with all letters upper case

- We will give you any methods you need
- But you must know how to slice strings!

String Slicing

```
def make_netid(name,n):
    """Returns a netid for name with suffix n."""
    name = name.lower() # switch to lower case
    fpos = name.find(' ') # find first space
    first = name[:fpos]
    last = name[fpos+1:]
    mpos = last.find(' ') # see if there is another space
    if mpos == -1:
        | return first[0]+last[0]+str(n) # remember, n is not a string
    else:
        | middle = last[:mpos]
        | last = last[mpos+1:]
        | return first[0]+middle[0]+last[0]+str(n)
```


What is on the Exam?

- String slicing functions (A1)
- Call frames and the call stack (A2)
 - **Very similar to A2 (see solution in CMS)**
 - **May have to draw a full call stack**
 - **See lectures 4 and 9 (slide typos corrected)**
- Functions on mutable objects (A3)
- Testing and debugging (A1, Lab 3, Lecture 10)
- Lists and For-Loops (Lab 6 and 7)
- Short Answer (Terminology)

Call Stack Example

- Given functions to right
 - Function `fname()` is not important for problem
 - Use the numbers given
- Execute the call:
`lname_first('John Doe')`
- Draw **entire** call stack when helper function `lname` completes line 1
 - Draw nothing else

```
def lname_first(s):
```

```
    """Precondition: s in the form  
    <first-name> <last-name>"""
```

```
1   first = fname(s)
```

```
2   last = lname(s)
```

```
3   return last + ',' + first
```

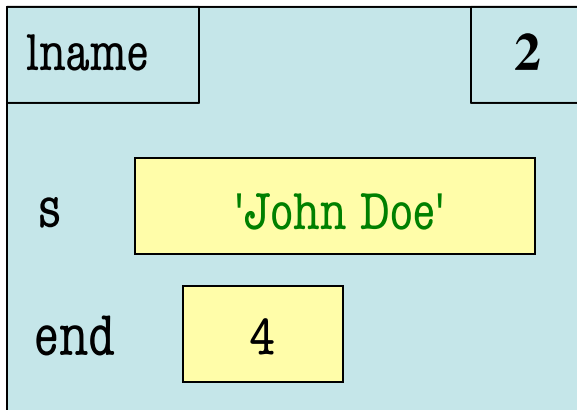
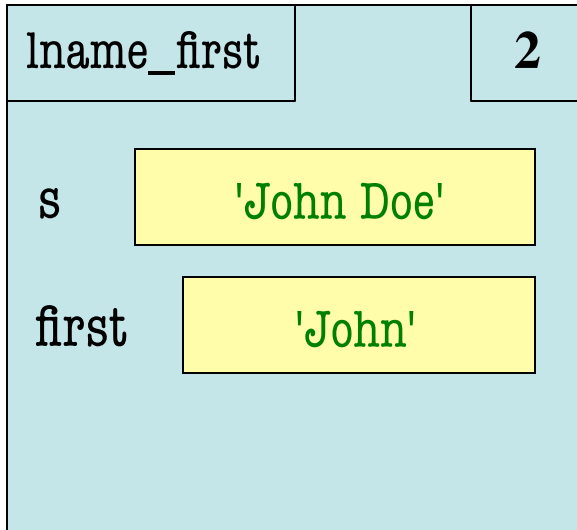
```
def lname(s):
```

```
    """Prec: see last_name_first"""
```

```
1   end = s.find(' ')
```

```
2   return s[end+1:]
```

Call Stack Example: lname_first('John Doe')



```
def lname_first(s):
```

```
    """Precondition: s in the form  
    <first-name> <last-name>"""
```

```
1 first = fname(s)
```

```
2 last = lname(s)
```

```
3 return last + ',' + first
```

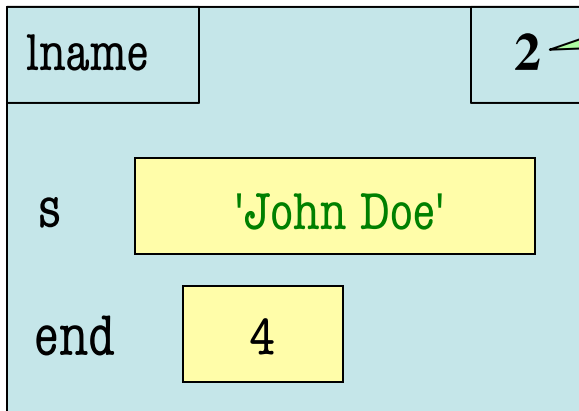
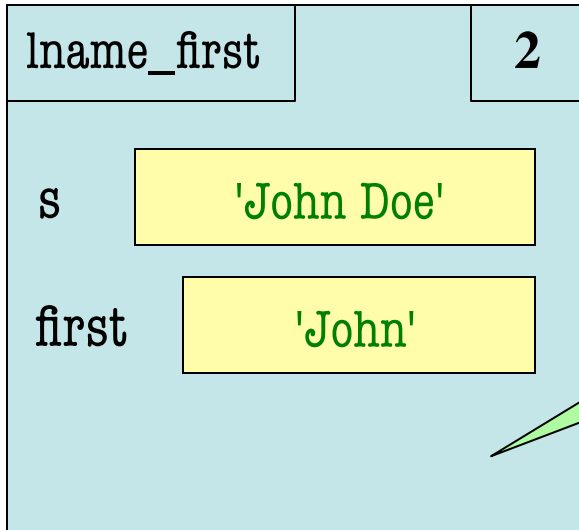
```
def lname(s):
```

```
    """Prec: see last_name_first"""
```

```
1 end = s.find(' ')
```

```
2 return s[end+1:]
```

Call Stack Example: lname_first('John Doe')



```
def lname_first(s):
```

No variable last.
Line 2 is not complete.

s in the form
<first-name>"""

```
1 | first = lname(s)
2 | last = lname(s)
```

Line 1 is **complete**.
Counter is **next line**.

```
1 | """Prec: see last_name_first"""
2 | end = s.find(' ')
   | return s[end+1:]
```

Example with a Mutable Object

```
def cycle_left(p):
```

```
    """Cycle coords left
```

```
    Precondition: p a point"""
```

```
1    temp = p.x
```

```
2    p.x = p.y
```

```
3    p.y = p.z
```

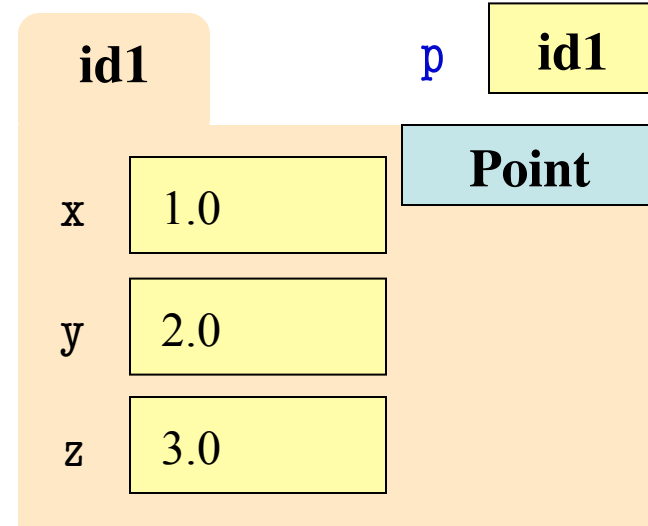
```
4    p.z = temp
```

- May get a function on a mutable object

```
>>> p = Point(1.0,2.0,3.0)
>>> cycle_left(p)
```
- You are not expected to come up w/ the “folder”
 - Will provide it for you
 - You just track changes
- **Diagram all steps**

Example with a Mutable Object

```
def cycle_left(p):  
    """Cycle coords left  
    Precondition: p a point"""  
1   temp = p.x  
2   p.x = p.y  
3   p.y = p.z  
4   p.z = temp
```



```
>>> p = Point(1.0,2.0,3.0)
```

```
>>> cycle_left(p) Function Call
```

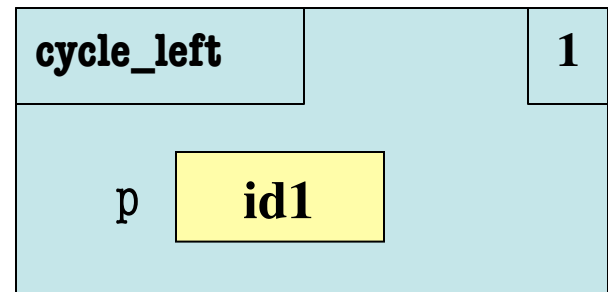
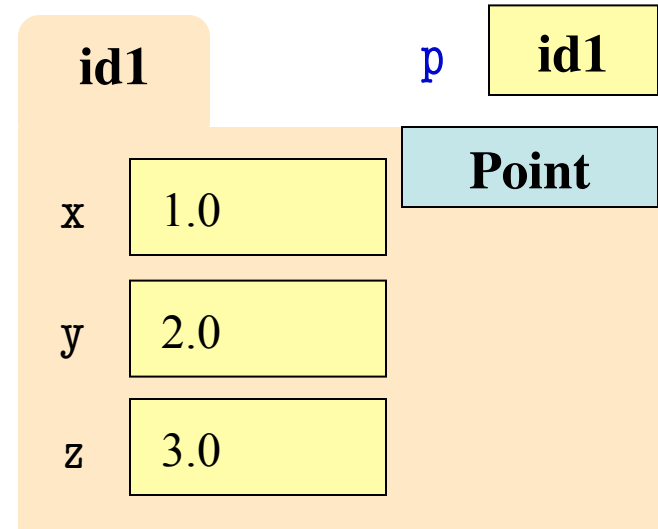
Example with a Mutable Object

```
def cycle_left(p):  
    """Cycle coords left  
    Precondition: p a point"""  
1   temp = p.x  
2   p.x = p.y  
3   p.y = p.z  
4   p.z = temp
```

```
>>> p = Point(1.0,2.0,3.0)
```

```
>>> cycle_left(p)
```

Function Call

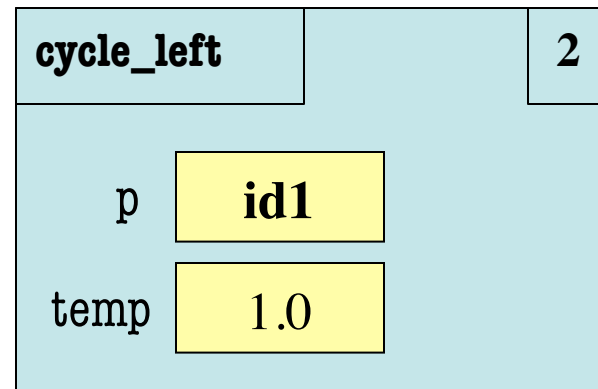
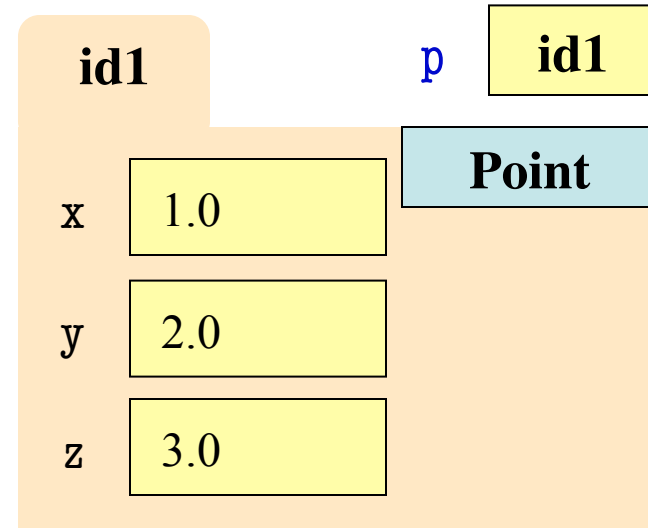


Example with a Mutable Object

```
def cycle_left(p):  
    """Cycle coords left  
    Precondition: p a point"""  
1    temp = p.x  
2    p.x = p.y  
3    p.y = p.z  
4    p.z = temp
```

```
>>> p = Point(1.0,2.0,3.0)
```

```
>>> cycle_left(p) Function Call
```

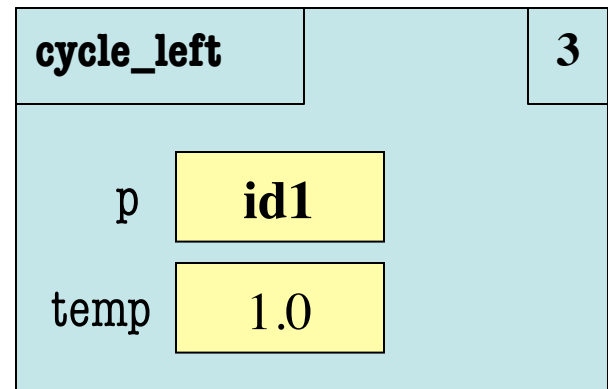
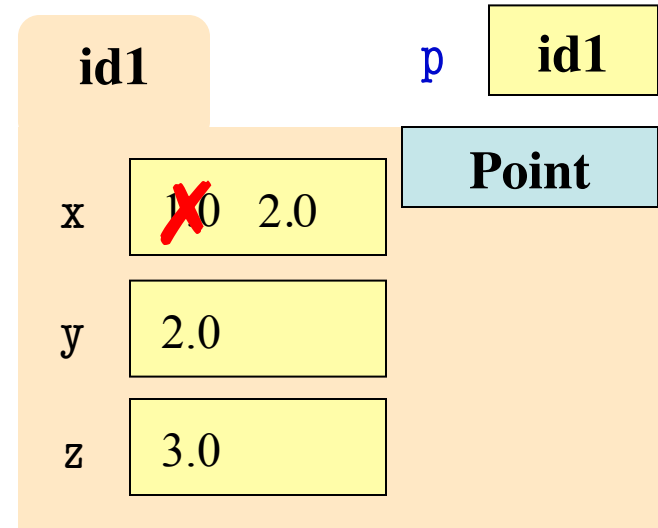


Example with a Mutable Object

```
def cycle_left(p):  
    """Cycle coords left  
    Precondition: p a point"""  
1   temp = p.x  
2   p.x = p.y  
3   p.y = p.z  
4   p.z = temp
```

```
>>> p = Point(1.0,2.0,3.0)
```

```
>>> cycle_left(p) Function Call
```



Example with a Mutable Object

```
def cycle_left(p):
```

```
    """Cycle coords left
```

```
    Precondition: p a point"""
```

```
1    temp = p.x
```

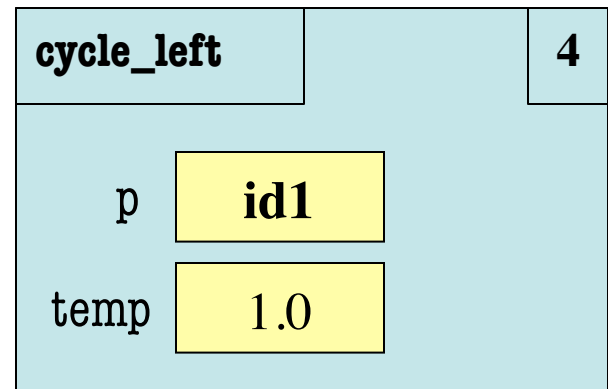
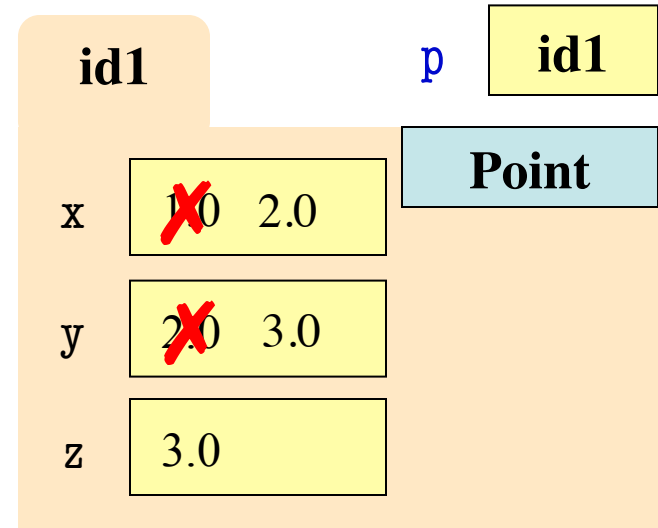
```
2    p.x = p.y
```

```
3    p.y = p.z
```

```
4    p.z = temp
```

```
>>> p = Point(1.0,2.0,3.0)
```

```
>>> cycle_left(p) Function Call
```

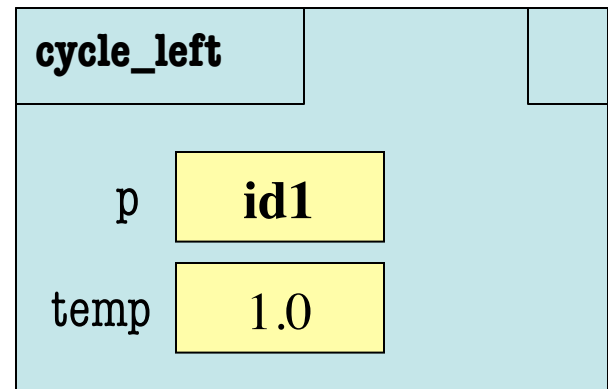
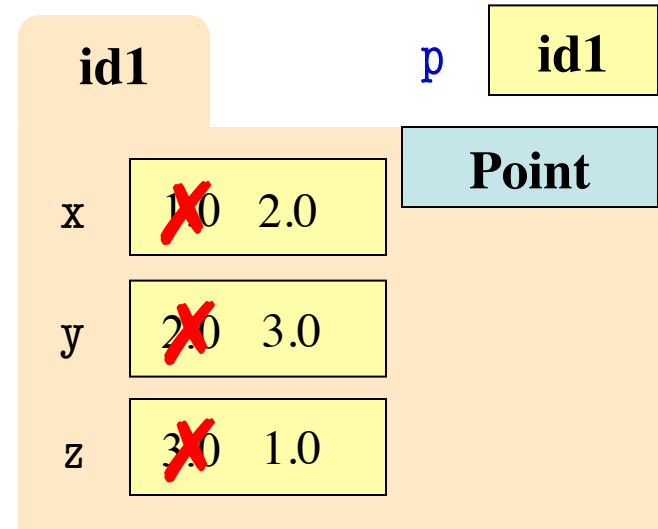


Example with a Mutable Object

```
def cycle_left(p):  
    """Cycle coords left  
    Precondition: p a point"""  
1    temp = p.x  
2    p.x = p.y  
3    p.y = p.z  
4    p.z = temp
```

```
>>> p = Point(1.0,2.0,3.0)
```

```
>>> cycle_left(p) Function Call
```



Example with a Mutable Object

```
def cycle_left(p):
```

```
    """Cycle coords left
```

```
    Precondition: p a point"""
```

```
1    temp = p.x
```

```
2    p.x = p.y
```

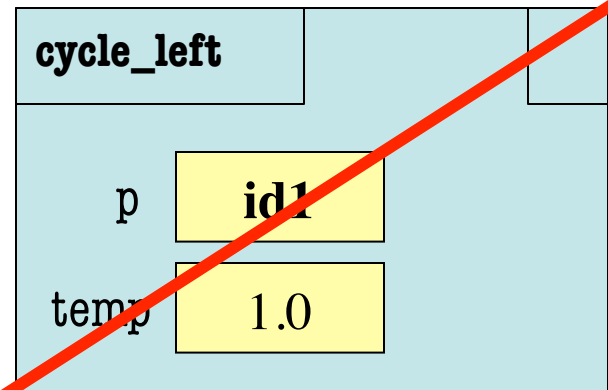
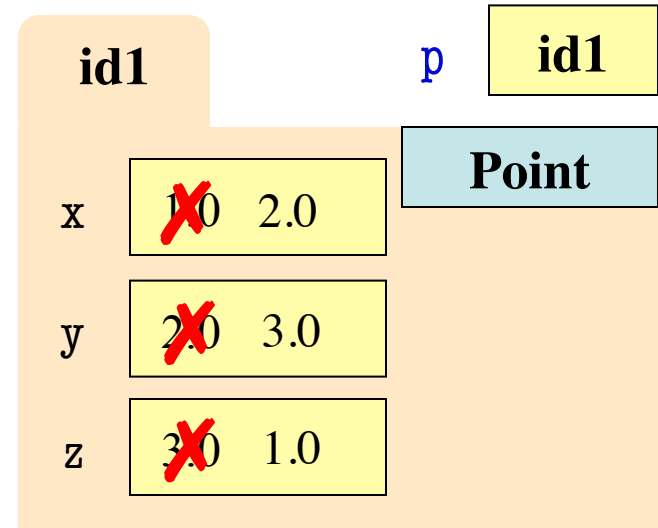
```
3    p.y = p.z
```

```
4    p.z = temp
```

```
>>> p = Point(1.0,2.0,3.0)
```

```
>>> cycle_left(p)
```

Function Call



What is on the Exam?

- String slicing functions (A1)
- Call frames and the call stack (A2)
- Functions on mutable objects (A3)
 - Given an object type (e.g. class)
 - Attributes will have invariants
 - Write a function respecting invariants
- Testing and debugging (A1, Lab 3, Lecture 10)
- Lists and For-Loops (Lab 6 and 7)
- Short Answer (Terminology)

Example from Assignment 3

- Class: RGB
 - Constructor function: RGB(r,g,b)
 - Remember constructor is just a function that gives us back a mutable object of that type
 - Attributes:

Attribute	Invariant
red	int, within range 0..255
green	int, within range 0..255
blue	int, within range 0..255

Function that Modifies Object

```
def lighten(rgb):
```

```
    """Lighten each attribute by 10%
```

```
    Attributes get lighter when they increase.
```

```
    Precondition: rgb an RGB object"""
```

```
    pass # implement me
```

Function that Modifies Object

```
def lighten(rgb):
```

Procedure:
no return

```
    """Lighten each attribute by 10%"""
```

```
    red = rgb.red # puts red attribute in local var
```

```
    red = 1.1*red # increase by 10%
```

```
    red = int(round(red,0)) # convert to closest int
```

```
    rgb.red = min(255,red) # cannot go over 255
```

```
    # Do the others in one line
```

```
    rgb.green = min(255,int(round(1.1*rgb.green,0)))
```

```
    rgb.blue = min(255,int(round(1.1*rgb.blue,0)))
```


Another Example

- Class: Length
 - Constructor function: Length(ft,in)
 - Remember constructor is just a function that gives us back a mutable object of that type
 - Attributes:

Attribute	Invariant
feet	int, non-negative, = 12 in
inches	int, within range 0..11

Function that Does Not Modify Object

```
def difference(len1,len2):
```

```
    """Returns: Difference between len1 and len2
```

```
    Result is returned in inches
```

```
    Precondition: len1 and len2 are length objects
```

```
    len1 is longer than len2"""
```

```
    pass # implement me
```

Function that Does Not Modify Object

```
def difference(len1,len2):
```

```
    """Returns: Difference between len1 and len2
```

```
    Result is returned in inches
```

```
    Precondition: len1 and len2 are length objects
```

```
    len1 is longer than len2"""
```

```
    feetdif = (len1.feet-len2.feet)* 12
```

```
    inchdif = len1.inches-len2.inches # may be negative
```

```
    return feetdif+inchdif
```

What is on the Exam?

- String slicing functions (A1)
- Call frames and the call stack (A2)
- Functions on mutable objects (A3)
- Testing and debugging (A1, Lab 3, Lecture 10)
 - Coming up with test cases
 - Tracing program flow
 - Understanding assert statements
- Lists and For-Loops (Lab 6 and 7)
- Short Answer (Terminology)

Picking Test Cases

def pigify(w):

"""Returns: copy of w converted to Pig Latin

'y' is a vowel if it is not the first letter

If word begins with a vowel, append 'hay'

If word starts with 'q', assume followed by 'u';
move 'qu' to the end, and append 'ay'

If word begins with a consonant, move all
consonants up to first vowel to end and add 'ay'

Precondition: w contains only (lowercase) letters"""

Picking Test Cases

```
def pigify(w):
```

```
    """Returns: copy of w converted to Pig Latin"""
```

```
    ...
```

- Test Cases (Determined by the rules):
 - are => arehay (Starts with vowel)
 - quiet => ietquay (Starts with qu)
 - ship => ipshay (Starts with consonant(s))
 - bzzz => bzzzay (All consonants)
 - yield => ieldyay (y as consonant)
 - byline => ylinebay (y as vowel)

Tracing Control Flow

```
def first(n):  
    print 'Starting first'  
    try:  
        second(n)  
        print 'Done first try'  
    except:  
        print 'In first except'  
    print 'Ending first'
```

```
def second(n):  
    print 'Starting second'  
    try:  
        assert n <= 0, 'is not <= 0'  
        print 'Done second try'  
    except:  
        print 'In second except'  
    assert n >= 0, 'not >= 0'  
    print 'Ending second'
```

What is printed during the call `first(-1)`?

Not guaranteed to have a try-except.
Might have an if or a for-loop instead.
But this example is the hardest type.

Tracing Control Flow

```
def first(n):  
    print 'Starting first'  
    try:  
        second(n)  
        print 'Done first try'  
    except:  
        print 'In first except'  
    print 'Ending first'
```

Recovers;
no error msg

```
def second(n):  
    print 'Starting second'  
    try:  
        assert n <= 0, 'is not <= 0'  
        print 'Done second try'  
    except:  
        print 'In second except'  
    assert n >= 0, 'not >= 0'  
    print 'Ending second'
```

No error

What is printed during the call first(-1)?

Error, but not
inside try

'Starting first'
'Starting second'
'Done second try'
'In first except'
'Ending first'

Error messages do not
show up if an except
forces a recovery

Tracing Control Flow

```
def first(n):  
    print 'Starting first'  
    try:  
        second(n)  
        print 'Done first try'  
    except:  
        print 'In first except'  
    print 'Ending first'
```

```
def second(n):  
    print 'Starting second'  
    try:  
        assert n <= 0, 'is not <= 0'  
        print 'Done second try'  
    except:  
        print 'In second except'  
    assert n >= 0, 'not >= 0'  
    print 'Ending second'
```

What is printed during the call first(1)?

Tracing Control Flow

```
def first(n):  
    print 'Starting first'  
    try:  
        second(n)  
        print 'Done first try'  
    except:  
        print 'In first except'  
    print 'Ending first'
```

```
def second(n):  
    print 'Starting second'  
    try:  
        assert n <= 0, 'is not <= 0'  
        print 'Done second try'  
    except:  
        print 'In second except'  
    assert n >= 0, 'not >= 0'  
    print 'Ending second'
```

Error

What is printed during the call first(1)?

Recovers;
no error msg

'Starting first'
'Starting second'
'In second except'
'Ending second'
'Done first try'
'Ending first'

Error messages do not
show up if an except
forces a recovery

Tracing Control Flow

```
def first(n):  
    print 'Starting first'  
    try:  
        second(n)  
        print 'Done first try'  
    except:  
        print 'In first except'  
    print 'Ending first'
```

```
def second(n):  
    print 'Starting second'  
    try:  
        assert n <= 0, 'is not <= 0'  
        print 'Done second try'  
    except:  
        print 'In second except'  
    assert n >= 0, 'not >= 0'  
    print 'Ending second'
```

What is printed during the call first(0)?

Tracing Control Flow

```
def first(n):  
    print 'Starting first'  
    try:  
        second(n)  
        print 'Done first try'  
    except:  
        print 'In first except'  
    print 'Ending first'
```

```
def second(n):  
    print 'Starting second'  
    try:  
        assert n <= 0, 'is not <= 0'  
        print 'Done second try'  
    except:  
        print 'In second except'  
    assert n >= 0, 'not >= 0'  
    print 'Ending second'
```

What is printed during the call first(0)?

'Starting first'
'Starting second'
'Done second try'
'Ending second'
'Done first try'
'Ending first'

Normal execution with
no errors happening

What is on the Exam?

- String slicing functions (A1)
- Call frames and the call stack (A2)
- Functions on mutable objects (A3)
- Testing and debugging (A1, Lab 3, Lecture 10)
- Lists and For-Loops (Lab 6 and 7)
 - Given a function specification
 - Implement it using a for-loop
 - Challenge is how to use accumulators
- Short Answer (Terminology)

Useful List Methods

Method	Result
<code>x.index(a)</code>	Returns first position of <code>a</code> in <code>x</code> ; error if not there
<code>x.append(a)</code>	Modify <code>x</code> to add element <code>a</code> to the end
<code>x.insert(a,k)</code>	Modify <code>x</code> to put <code>a</code> at position <code>k</code> (and move rest to right)
<code>x.remove(a)</code>	Modify <code>x</code> to remove first occurrence of <code>a</code>
<code>x.sort()</code>	Modify <code>x</code> so that elements are in sorted order

- We will give you any methods you need
- But you must know how to slice lists!

For-Loop in a Fruitful Function

```
def replace(thelist,a,b):
```

```
    """Returns: COPY of thelist with all occurrences of a replaced by b.
```

```
        Example: replace([1,2,3,1], 1, 4) = [4,2,3,4].
```

```
    Precondition: thelist is a list of ints; a and b are ints"""
```

```
    return [] # Stub return. IMPLEMENT ME
```

For-Loop in a Fruitful Function

```
def replace(thelist,a,b):
```

```
    """Returns: COPY of thelist with all occurrences of a replaced by b.
```

```
        Example: replace([1,2,3,1], 1, 4) = [4,2,3,4].
```

```
    Precondition: thelist is a list of ints; a and b are ints"""
```

```
    result = [] # Accumulator
```

```
    for x in thelist:
```

```
        if x == a:
```

```
            result.append(b)
```

```
        else:
```

```
            result.append(x)
```

```
    return result
```


For-Loop in a Procedure

```
def pairswap(seq):
```

```
    """MODIFIES thelist, swapping each two elements with each other
```

```
    Example: if a = [0,2,4,5], pairswap(a) makes a into [2,0,5,4]
```

```
           if a = [1,2], pairswap(a) turns a into [2,1]
```

```
    Precondition: thelist is a list with an even number of elements."""
```

```
    pass # implement me
```

For-Loop in a Procedure

```
def pairswap(thelist):
```

```
    """MODIFIES thelist, swapping each two elements with each other
```

```
    Example: if a = [0,2,4,5], pairswap(a) makes a into [2,0,5,4]
```

```
           if a = [1,2], pairswap(a) turns a into [2,1]
```

```
    Precondition: thelist is a list with an even number of elements."""
```

```
    for k in range(len(thelist)):
```

```
        if k % 2 == 0:
```

```
            tmp = thelist[k]          # Store old value
```

```
            thelist[k] = thelist[k+1] # Get next value
```

```
        else:
```

```
            thelist[k] = tmp          # Value stored in previous step
```

What is on the Exam?

- String slicing functions (A1)
 - Call frames and the call stack (A2)
 - Functions on mutable objects (A3)
 - Testing and debugging (A1, Lab 3, Lecture 10)
 - Lists and For-Loops (Lab 6 and 7)
 - **Short Answer (Terminology)**
 - See the study guide
 - Look at the lecture slides
 - Read relevant book chapters
- In that order

Any More Questions?



