Admin

• Reminder: Lecture Mar 9th cancelled
  – Have posted two exercises on Lists for you to do on your own time – recommend working through with a partner.
  – Racquel will hold office hours during the lecture slot (9:30 – 11am) in X341

• JD office hours March 9\textsuperscript{th} also cancelled
  – Replacement office hours today 3 – 4:30 pm

• Midterm grades posted
  – See Piazza post for details
Do you want to “do better” in this course?

Suppose you aren’t satisfied with your learning or your marks in this course to date.

What are you going to do differently from now on to improve your learning and/or your grade?

Adapted from slides provided by
Beth Simon
CWSEI Science Teaching and Learning Fellow
What the research says

• In a study of Biology students in a genetics course, students who had specific plans for improving after the first midterm did better in the class than those with “general” plans.

• NOT GOOD:
  – “I’m going to turn things around.”
  – “I’m going to dig myself out of this bad mark.”

• BETTER:
  – “I’m going to revise my notes for at least 10 minutes after each class and before the next class.”
  – “I’m going to read the textbook pages before class.”
  – “I’m going to ask questions of the professor or TA.”

Problems with General Goals

1. You still have to figure out what you should actually be doing to accomplish that goal.
   – What will you do each day or week to “turn it around”?

2. It’s hard to get feedback to let you know if you are making any progress on that.
   – What kind of thing that you control will “tell you” if you are successfully “turning it around”?
     • The final exam is a bit late . . .
Benefits of Specific Goals

1. The goal itself tells you exactly what you should be doing to accomplish it.

2. You can get feedback on your own to determine if you are meeting your goal.
   - The goal should be measurable or countable.
   - You can make a log (on paper, on the wall, in email, whatever) that shows the work that you did to accomplish the goal.
Specific, measurable goals

• Are more likely to produce actual results.
  – Sports teams have highly specialized, specific training programs.

• Describe something you can do
  – you think addresses a deficiency in your learning or understanding

• Identify how you will know if you have done it
  – What the “evidence” will be

• When you are going to do it

• Keep a record (maybe email a friend)
Possible Concrete Goals

• I’m going to take notes as I read the book before class, and write down two questions I have about the reading.
• I am going to trace and then summarize each of the example codes from the textbook.
• I am going to work through an exercise from the textbook every day.
• After each class, I am going to discuss for 20 minutes with another student what I thought was important from class and what I am confused about.
• I am going to review and paraphrase my notes after each class.
• I am going to go back and solve all of the problems from the first midterm.
• I am going to look for and solve additional problems from other introductory Python courses.
CPSC 301: Computing in the Life Sciences
Lecture Notes 11:
Lists

2017-03-07

Jessica Dawson
jqdawson@cs.ubc.ca
http://www.cs.ubc.ca/~jqdawson

University of British Columbia
Department of Computer Science

2016 W2
Slicing

• Given a list, we can extract an entire sublist, called a slice of the list, using

\[ \text{list}[i:j] \]

which returns a new list containing the elements from the old list that are in positions \( i, i+1, \ldots, j-1 \)

• For instance:

```python
groceries = ['bread', 'milk', 'cheese', 'tomatoes']
dairy = groceries[1:3] \text{ then } dairy \rightarrow ['milk', 'cheese']
```

• \( \text{list}[:j] \) is the same as \( \text{list}[0:j] \)

\( \text{list}[i:] \) is the same as \( \text{list}[i:\text{len(list)}] \)

\( \text{list}[:] \) creates a new copy of \( \text{list} \)

\( \text{list}[:\text{len} - 1] \) creates a new copy of the original list without the last element
Common Patterns for Using Lists (and Sequences)

• **Reduce** a list:
  
  Ex 1: prod = 1  
  for item in old_list:  
      prod *= item  
  
  Ex 2: sum = 0  
  for item in old_list:  
      sum += item  

• **Map** an operation to a list:
  
  for i in range(len(old_list)):
      old_list[i] = old_list[i]**2  

• **Filter** a list:
  
  new_list = []  
  for item in old_list:
      if (item >= 0):
          new_list.append(item)  

• Note: 1st and 3rd above do not modify the list. The 2nd modifies it.
  – You can write examples of all three general patterns to do either.
Copies, Aliases and Mutability

• An **alias** is another variable that references the same data in memory
• A **copy** is another variable that references data that looks the same, but is stored in a different memory location
• Previously our data was *immutable* (it could not change), so it did not matter whether a variable was a copy or an alias of another variable
• When a variable has an alias and references mutable data, its value can change without the variable appearing in an assignment
• Python provides a way of testing for copies and aliases
  – The boolean test `a == b` is **True** if `a` is a copy or an alias of `b`
  – The boolean test `a is b` is **True** if `a` is an alias of `b` but **False** if `a` is a copy of `b`
Copies and Aliases with Lists

• Lists are mutable
  – To create an alias of a we can use:
    \( b = a \) or use a as an argument to a function or method
  – Create a copy of a:
    \( b = a[:] \) or use a[:] as an argument to a function or method

• Example
  ```python
  a = [9, 99, 999]
b = [9, 99, 999]
c = a
d = b[:]
a[2] = 0
e = d.reverse()
  ```
  – What is the value of \( b \)? \( c \)? \( d \)?
  – What is the value of
    \( a == b? \) \( a \ is \ b? \) \( a == c? \) \( a \ is \ c? \)

Values in memory after execution:

- \( a \rightarrow [9, 99, 0] \)
- \( b \rightarrow [9, 99, 999] \)
- \( c \rightarrow [9, 99, 0] \)
- \( d \rightarrow [999, 99, 9] \)

*a and c are aliases of one another

Notes 11: Lists

- \( a == b? \) \( false \)
- \( a \ is \ b? \) \( false \)
- \( a == c? \) \( true \)
- \( a \ is \ c? \) \( true \)
Aliasing through Function / Method Calls

• Consider:

    def foo(x):
        for i in range(len(x)):
            x[i] += 1
    
    a = [9, 99, 999]
    b = a[:]
    c = foo(a)

    – What is the value of
      $a \rightarrow$
      $b \rightarrow$
      $c \rightarrow$

• Safe design principles for functions with list parameters:
  – Don't modify an input list; make a copy and modify that instead
  – If you really need to modify an input list, make sure to explicitly mention how you are modifying it in the docstring
  – If you modify an input list, do not return it (return None)
Nested Lists

- A list can contain anything, including another list
  - The sublists may contain anything (or nothing)
- Access a sublist's elements with repeated []
- Example: 
  
  ```
  x = [ [ 'a', 1 ], [ 'b', 2 ], [ 'c', 3 ], [ 'd', 4 ] ]
  ```

  - What is the value of `x[2]`? `['c', 3]`
  - What is the value of `x[1:]`? `[['b', 2], ['c', 3], ['d', 4]]`
  - What is the value of `x[3][0]`? `'d'`

  - What is the value of `x` after `x[1] = [['a', 1], 42, ['c', 3], ['d', 4]]`?
  - What is the value of `x` after `x[3][1] = 42`? `[['a', 1], ['b', 2], ['c', 3], ['d', 42]]`

(assuming `x` starts with values at top of slide)
Summary

• Python has a number of *collection* data types which can store multiple values
• Lists are mutable, ordered, indexed collections of heterogeneous data, and may contain other lists
• Brackets `[]` are used to create lists or to access or modify their elements or slices
  – `[]` are also used to access the characters in a string, but cannot modify the strings
• Python has many functions, methods and operators which can build, examine and modify lists
• Common design patterns for lists: reduce, map and filter
• Some common mistakes arise from the mutability of lists
  – List methods often modify the list and return `None`
  – Aliases can cause lists to change at surprising times
Change Log

• Design Patterns Slide:
  – Added an extra example of a reduction pattern
  – Added clarification about design patterns being used in ways that both modify and do not modify a list