More on working with Data
• Questions on project?
Review Questions

• Given the python code
  - question1a = “go to the store”
  - question1b = “sufferin' succotash!”
  - Use the variable question1b and what you've learned to print out Sylvester's catchphrase “Thufferin' Thuccotash!”
  - Use question1a to print out the 'little kid' version “go to the tore”
  - How would we use slicing to get the string uffer from question1b?
recall from before

- every piece of data in the computer has a type
  - interpretation given to the 1s and 0s in the computer
- in python types often taken care of for you
  - still need to have an idea about the type of things stored in your variables though.
types we need to know

- in python still need to know a little about types
  - divide into
    - strings/text
    - whole numbers/integers
    - fractional/decimal numbers.(floating point)
  - two types of numbers stored very differently in computer.
String concatenation

• fancy term for putting two strings together to make a bigger string
  - this = “stringer”
  - that = “second “
  - result = that + this
  - print (result)
  - # prints out: second stringer
  - if thing on left of + is string, then string concatenation. If thing on left of + Is number - then addition.
Efficiency

• Concatenation works
  - But not too efficient
  - Pythonic way is 'string interpolation'
String interpolation

- Thus far print different items
  - name = input(“what is your name”)
  - print (“pleasure to meet you”, name)
  - Print puts in its own spacing
- how about if we put it all in one variable
  - output = “pleasure to meet you %s” % name
  - print(output)
String interpolation

- %s is special character sequence
  - when python sees it, will replace with something else
  - looks for % by itself after the string
  - should find string after the solitary % that string replaces %s
• Some string interpolation operators for starters
  - %s : string or text
  - %d : integer (in decimal/base10 notation)
  - %f : floating point/fractional number
  - %% just put a % symbol into the string.
now let's see if that made sense

• What will this code do?
  - grade = 92
  - print( “your grade is %d” % grade)

• how would you read in the name of the month and then print out that month at the end of the sentence “you are talking about” eg “you are talking about January” (tell me the python code)
multiple interpolation

- It's very possible to do string interpolation with multiple variables.
  - name = input("Name Please:")
  - age = 397

- output = "wow %s you are %d that's really old!" % (name, age)
  - print(output)
• consider the following code
  - note that the – bullet indicates the beginning of a line
  - numMeals = input(“how many meals did you eat yesterday?:”)
  - numSnacks = input(“how many snacks did you have yesterday”)
  - total = numMeals + numSnacks
  - print (total)
  - what gets printed?
  - now that you know - why?
converting

• if you know a string is just the text for a number
  - convert to number
    • str = "13"
    • number = int(str)
    • Or what else?
  - floating point number
    • str = "2.5"
    • number = float(str)
working with lots of data

• sometimes working with one piece of data at a time is fine
  - current temperature
  - name

• often want to work with lots of related data.
  - names of all students in class.
  - all grades for student x this semester

• easy way to do this in python: list
Lists

• python representation of well
  - ordered list of stuff
  - list can hold values of any data type
    • though often good to have all data in list of same type.
  - literal list
    • mylist = [“comp101”, “comp442”, “comp460”, “comp470”]
    • note list set off in []
    • items in list separated by commas.
accessing items in the list

• access list items using the [] operation
• starts counting from 0.
  - mylist = [“comp101”, “comp442”, “comp460”, “comp470”]
  - print (mylist[0])
  - prints out what?
• update same way
appending to lists

• can overwrite an item in a list
  - myList[0] = “new stuff”

• might also want to add to the end of the list.
  - lists also objects
  - append method for list objects.
  - myList.append(“even newer Stuff”)
  - puts that string as new last item in list
slicing lists

• can slice lists same as strings.
  - print (mylist[:2])
    • prints from beginning of list to item in slot 2 (third item in list.)
  - how will we get the 2\textsuperscript{nd} and 3\textsuperscript{rd} items only?
examine each list element once

• sometimes you'll want to do something with each element in the list
  - can't change the list itself, but can do something for each element.
  - called 'looping through the list'
  - do the same thing again and again for each list item.
looping through the list

• syntax
  - for <var_name> in <list>:
    • statement
    - var_name will have a different value each time statement is called
  - example
  - for item in mylist:
    • print(" teach %s this semester" % item)
  - note: at end of first line
  - note that statement is indented.
  - more examples?
Now let's look at project

- Your first out of class programming project.
Ranges

- Can create special list of numbers (usually consecutive)
  - Use range function:
    - range(10)
      - creates a list of 10 numbers: 0-9
    - range (3, 7)
      - creates a list of 4 numbers from 3 through 6
    - range (0,100,10)
      - create a list of 10 numbers, 0 -90 counting by ten each time
    - range (10, 0, -1)
      - create a list on numbers from 10 down to 1
  - Show using print(list(range(x,y))))
We've seen this before, but range looping is useful this way.

- print a count down
- countdown = range(10, 0, -1)
- for second in countdown:
  - print second
- print(“blastoff”)

Range looping
String Formatting

- Can format the items we put into strings
  - So that they look nice.
  - Format:
    - "string text {:<width>}".format(<value>)
    - Stuff in the {} is replaced by the formatted value
  - Eg
    - str = "a string with width 4 number {:4}"
    - str.format(1)
    - Yields:
      - a string with width 4 number    1
String Formatting 2

• Format floating point numbers
  - " {:7.5}".format(3.3333333)
  - Puts a width 7 number with 5 digits of precision
  - " {:7.5f}".format(3.3333333)
  - Puts a width 7 number with 5 digits after the decimal point
Reading a File

• Two types of files on computer
  - Text
  - binary

• Python makes reading from text files really easy (open a file)
  - Open file
  - `<variable> = open(<filename>, mode)`
  - Eg
    • `Infile = open("file1.txt", 'r')`
Reading file

- Read file
  - Once you have a file object – easiest
  - Way
    - File.readlines()
    - Returns a list of strings
    - Each string is a line in the file.
Writing a file

- Outfile = open("myoutput.txt", 'w')
- Print("stuff to go in file", file=outfile)
putting it all together

- lets put it all together and write a python program.
  - write a program that will print out a word in reverse.
  - how will we do this?
  - What about read in a file and print every second line to screen
  - Use sillyrec.txt
idle debugger

• Debugger:
  – program that will let you look at the program as it is running
  – and slow the program down to the point that it is useful.

• Now lets take a look at what the debugger can show you
End chapter 5

- be sure to read it.