Introduction to:
Computers & Programming:
Review for Final

Adam Meyers
New York University
Summary

• Some Procedural Matters
• Summary of Topics Covered This Semester
  – Focus on a Subset of these topics
• The Practice Final
Procedural Matters Regarding the Midterm

- The 009 test is on Mon May 14 8:00 to 9:50AM
  - Room C10
- The 004 test is on Mon May 14 10:00 to 11:50AM
  - Room 110
  - **004 test starts 1 hour earlier than normal class time**
- Both tests are in the respective “normal” classrooms
- I will take attendance – please bring your School ID
- The test will be graded on a curve
- The final counts for 35% of the final grade
- Grades will be posted to NYUClasses
  - Graded Tests can be viewed by appointment: saved for 7 years :)
Type of Questions that Could Be on the Test

• Section I: Answer questions about Code
  – What is printed out?
    • Look for “print” commands (consider if statements and loops)
  – What value is returned by a function?
    • Look for “return” commands
  – What is the value of a certain variable
    • Look at assignment statements (with =)
  – Open ended questions:
    • What does a function called by the main program do?
      – what does it return or print and why?
    • What are the conditions that will cause the program to end?
      – look at if and while statements to figure out answer

• Section II: Write functions that solve stated problems
  – Read questions carefully
  – Common errors involve misreading questions and writing functions that solve problems other than those stated in the question
What you should know

- Basics: *algorithm, function, program, operator, input, output (return), side effects (print), variables (global, local, scope), comments*
- Data Types: *integer, float, boolean, range, character, string, list, stream, dictionary, turtle, none*
- Control Structures and Related Programming Approaches
  - *if/else/elif, decision trees, loops (for & while), try/except*
  - *Recursion & Object Oriented Programming (optional)*
- Sequences: indices, slices, for-loops, type-specific methods/functions
- Packages: math, os, random, time, turtle
- Graphics (turtle)
- File handling (renaming files, creating directories, pathnames)
Basics: Planning a Program

• Write a big function that solves the problem, with clear input and output names

• Break problem down into smaller parts (also functions), some of which repeat (using loops).

• Example:
  
  – `def color_wall(paint, paint_brush, wall):
    for section in wall:
      apply_paint(paint, paint_brush, section)
  `  

• Repeat process to define smaller parts of problem, e.g.,

  – `def dip_paint(brush, paint):
    ...`
  
  – `def brush_wall_section(brush, wall_section):
    ...`
  
  – `def apply_paint(brush, wall, paint):
    dip_paint(brush, paint)
    brush_wall(brush, wall)`:  

Functions

• Functions communicate with each other in programs using parameters and return statements

• `def calculate_average(inlist):
  
  total = 0 ## accumulator variable
  
  for num in inlist:
    
    total = total+num

  return(total/len(inlist))  ## return exits function, determines ## value of function call

• `def add_averages(list1,list2):
  
  output = calculate_average(list1)+calculate_average(list2)

  return(output)`
Variables

• Global Variables
  – Can be defined outside of functions
  – Should be declared “global” inside of functions

• Local Variables – internal to functions
  – parameters or variables initialized in functions
  – other functions cannot “see” local variables
  – values can be “passed” by setting local variables to values returned by other functions

• Variable assignment with =
  – variable1 = 5  ## assign a variable to a value
  – variable2 = str(5) ## assign a variable to the output returned from a function
  – variable1,variable2 = 5,10 ## assign multiple values to multiple variables
  – variable1,variable2 = [5,10]  ## same, but using a sequence
  – variable1,variable2 = func_returning_2_values(20) ## same with function
Interaction with (Human) User

• **print**
  – provide human beings with explanations that are printed on the screen
  – does **NOT** return output
  – Take multiple parameters separated by commas
  – Allow 2 key word parameters: sep and end
  – Example: `print(1,2,3,sep='*and*','end='--go')`

• **input**
  – way for program to get input from human users
  – takes one string parameter (a prompt)
  – returns string input by the human being (ignoring newline character)
  – Example: `number = int(input('Give me a number '))`
  – Note: Do **NOT** overuse input statements – parameters are more appropriate for functions called by other programs
Ending Loops

• Completing Loops
  – *for loop* ends when there are no more items in the sequence
    • for item in sequence:
      – ## executes once for each item in the sequence
      – ## sets the variable *item* to that item
  – *while loop* ends when the condition is false
    while condition
      ## condition contains variable that changes
      ## when that change causes condition to be False, loop ends

• Stopping/Limiting Loops via explicit commands
  – *break* – causes program to leave loop
  – *continue* – following statements skipped for current iteration
  – *pass* – do nothing (typically in *if* statement)
  – *return* – exits not only loop, but function as well (cannot be used outside of a function) – also can return values
# Example Sequences and Accumulator Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Initial Value</th>
<th>Incremental Change</th>
<th>Final Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>0</td>
<td>add next number from list of numbers</td>
<td>Total numbers in list</td>
<td>Need total for average</td>
</tr>
<tr>
<td>Integer</td>
<td>1</td>
<td>multiply next number in range</td>
<td>Product of numbers in range</td>
<td>Factorial</td>
</tr>
<tr>
<td>Character</td>
<td>empty string</td>
<td>concatenate code derived from next char in input string</td>
<td>Coded version of sentence</td>
<td>Convert string to code</td>
</tr>
<tr>
<td>list</td>
<td>Empty list</td>
<td>Convert next string in list of strings to other form and append it to accumulator</td>
<td>List of output values from list of strings</td>
<td>Collect a list of values for some function (length, lowercase, a code, etc.)</td>
</tr>
</tbody>
</table>
Lists of Lists

- A list of lists of lists of lists of …
  - Each “level” represents something
  - Example 1
    - student_list = list of students in a major
    - dept_list = list of student_list
    - school_list = list of dept_list
  - Example 2
    - product_description = list of traits in fixed order
    - Inventory = list of product_description

- A list of lists can represent dimensions
  - rows and columns in a table
  - X and Y in a Cartesian graph
  - Example
    - Tic_tac_toe_row = list of three positions filled by X, O or _ (underscore)
    - Tic_tac_toe_board = list of Tic_tac_toe_row
    - Tic_tac_toe_board[0][1] → item occupying middle position of first row
Mutability of Lists and Dictionaries

• Immutable types like strings and numbers
  – New values are created based on immutable types

• Mutable types like lists and Dictionaries
  – The original lists and dictionary can be changed
  – Variables point to these objects
    • Functions applied to variables can change these objects
      – Example:
        • my_dict = {1:'a',2:'b',3:'c'}
        • my_dict2 = my_dict
        • my_dict[4]='d'
        • my_dict2 → {1:'a',2:'b',3:'c',4:'d'}
def divide_10_by_an_integer ():
    while True: ## only exits if a value is returned
        try:
            number = int(input('Pick an integer: '))
            output = 10/number
            return(output)
        except:
            print('''Invalid entry: must choose integer and cannot choose zero. Try again!'"'')}
File Input/Output

- Simple reading lines in (using “with open...” syntax)
  - def do_something_to_input_lines(input_file):
    with open(input_file,'r') as instream:
      for line in instream:
        line = line.strip(os.linesp)
        do_something_to_line(line)

- Simple writing lines to a file
  - def write_out_lines_to_file(list_of_lines,output_file):
    with open(output_file,'w') as outstream:
      for line in list_of_lines:
        ## possibly add end of line character, e.g.,
        ## line = line+'\n'
        outstream.write(line)

- Opening and closing a stream (equivalent to first bullet)
  - def do_something_to_input_lines(input_file):
    instream = open(input_file,'r')
    for line in instream:
      line = line.strip(os.linesp)
      do_something_to_line(line)
    instream.close()
Reading & Writing Standard Data Formats

- File I/O works provided there are standard ways to read and write to your files
- Shared standards make programs easier for others to modify or integrate with their programs
- We discussed two standard file formats in class:
  - .tsv files, in which each line represents a set of fields, divided by tabs, e.g., 'Adam\tMeyers\tProfessor\t13'
    - I preferred this one because commas are sometimes part of data, but this is rarer for tabs
  - .csv files, in which each line represents a set of fields, divided by commas, e.g., 'Adam,Meyers,Professor,13'
- Optionally, for the first line of such files can list the categories, e.g., 'First_Name\tLast_Name\tJob\tshoe_size'
Modifying Lists

• my_list = [1,2,3,4]
• Add/Subtract one item:
  – my_list.append(5)
  – my_list → [1,2,3,4,5]
  – my_list.pop() returns last item and shortens list → [1,2,3,4]
• Combine 2 lists
  – my_list.extend([6,7]) or my_list += [6,7]
  – my_list → [1,2,3,4,6,7]
• Return the combination of 2 lists
  – my_list + [8,9] → [1,2,3,4,6,7,8,9]
  – my_list → [1,2,3,4,6,7]
• Change/Subtract item at a particular index
  – my_list[2] = '5.7' → [1,2,3,5.7,4,5,6,7]
  – my_list.pop(2) returns item at index 2 and shortens list → [1,2,5.7,4,5,6,7]
• **append & list[index]=item** add/change exactly one item
• **extend & += require that the additional argument be a list**
• **pop** removes item (see above) and returns it (so a variable can be set to removed item)
Practice Final

• I will go over this in detail today:
    • Related file: http://cs.nyu.edu/courses/spring18/CSCI-UA.0002-004/specimens.tsv
    • Printable test
  – http://cs.nyu.edu/courses/spring18/CSCI-UA.0002-004/practice_final.py
    • Python file
      – Part 1 programs to test
      – Sample Part 2 answers
  – Note there is an additional practice final online as well:
    • http://cs.nyu.edu/courses/spring18/CSCI-UA.0002-004/practice_final2.pdf
    • http://cs.nyu.edu/courses/spring18/CSCI-UA.0002-004/animals.tsv
    • http://cs.nyu.edu/courses/spring18/CSCI-UA.0002-004/paragraph_from_wikipedia.txt
    • http://cs.nyu.edu/courses/spring18/CSCI-UA.0002-004/character_dict.tsv
    • http://cs.nyu.edu/courses/spring18/CSCI-UA.0002-004/practice_final2.py

• Structure is similar to midterm:
  – 4 Part 1 questions
  – 2 out of 3 Part 2 questions
  – Time is 1:50 minutes (about 1.5 times the 1:15 allowed for midterm)
Timing

• There are 6 questions to be completed in 1:50. This is about 50% more time than the midterms (1:15), but some of the questions may take a little longer.

• I suggest budgeting your time, e.g.,
  – 10 min X 4 part 1 questions = 40 min
    • Skip part 1 questions that are difficult and go back to them later
  – 20 min X 2 part 2 question = 40 min
  – Extra time for going over budget: 30 minutes
Reminder: A Test is a Game

- Unfortunately, tests are imperfect for measuring a person's expertise because (independent of such expertise) some people know how to play the *test game* better than others

- How to win the test game
  - Study sample test instructions
  - Time is a crucial factor (you have 1 hour and fifty minutes)
  - Do easy problems before hard ones
  - Do not spend a lot of time on low-point problems
  - Do not get stuck on details that you don't need
    - Solving all problems is more important than doing 1 problem elegantly
  - Go for partial credit on program questions (most points)
    - If you cannot program some detail – write pseudo code
    - Basic solution strategy is more important than perfect syntax
The Final is Next Week

• Please feel free to ask me any questions
• We will go over the practice midterm in class: ask questions
• Suggested Studying Materials
  – Previous class lectures, notes, homeworks, etc.
  – The Practice Finals on the website
  – Practice problems of your choosing
  – Previous tests:
    • Midterms from this class
    • Tests from previous Intro to Programming classes listed on my website
      – http://nlp.cs.nyu.edu/people/meyers.html
      – Recent tests should be similar than older ones
• I am also available for office hours prior to the test
• Good luck!