Introduction to: Computers & Programming: Review for Midterm 2

Adam Meyers
New York University
Summary

• Some Procedural Matters
• Summary of what you need to Know
  – For the Test and To Go Further in the Class
• The Practice Midterm (and Last Semester's Midterms)
Procedural Matters Regarding the Midterm

• The test is next class
• I will take attendance – please bring your School ID
• The test will be graded on a curve
• The midterm counts for 20% of the final grade
Type of Questions that Could Be on the Test

• Section I: Answer questions about Code
  – What is printed out?
    • Look for “print” commands (sometimes include several rounds of interaction for a complex program)
  – 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, e.g., checks if input is incorrect form
    • 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

- **Old Topics:** algorithm, function, program, operator, input, output, side effect, variable, data types, if statements, else, decision trees, loops

- **New Topics:** Turtles, Strings, Lists and sequence operations

- **Know old stuff from last midterm:** define functions, assign values to variables; return values from functions and operators; use print statements, including the sep and end keywords; import modules; use data types and coerce one data type to another; write simple functions that work and are easy to understand, due to comments and variable/function names; if/elif/else statements and basic decision trees; use for loops and while loops; random numbers (just random.randint)

- **New Stuff:** turtle graphics (drawing lines, circles, turning left and right, putting the pen up and down, etc.); problems involving manipulating strings (slices, identifying characters by indices, building new strings using loops, etc.); lists (slices, identifying items by indices, changing parts of lists, using list.append, list.pop, list.reverse and various other list methods); using max, min and sort with lists, strings, characters, etc.
Functions

- Programming language Functions have 3 optional features:
  - Input
  - Output
  - Side effects
- Input to a function via its parameters
- Using the “input” function to solicit interactive input from a user
- `print` versus `return`
  - `print` is significant for its side effect—printing to the computer screen
  - `return`
    - Exits block (function)
    - Provides a value to a function call – Example:
      - If: function1(a) returns 5 and function2(b) returns 10
      - Then: function(a) + function(b) = 15
Turtle Graphics

- `my_screen = turtle.Screen()`
- `my_turtle = turtle.Turtle()`
- `my_turtle.pu()`
- `my_turtle.pd()`
- `my_turtle.fd()`
- `my_turtle.left()`
- `my_turtle.circle()`
- `my_turtle.setposition(X,Y)`
Strings and Characters

- Characters: `chr(number)`, `ord(char)`, `\n`, `\t`, etc.
  - Comparison with `>`, `>=`, `<=`, `>=`, e.g., `'Z'<'a`

- Indices and Slicing
  - `my_string = 'big bad wolf'`

- String operators, functions, methods
  - `len(string)`, `string.reverse()`, `'abc'.upper()`, `'AbC'.lower()`, `string1+string2`
  - Comparison with `>`, `>=`, `<`, `<=`, e.g., `'Abc'>'ABc`

- Loops (for or while) with accumulator variables, e.g.,
  ```python
def some_function(input_string):
    output = ''
    number = 0
    for char in input_string:
      number = change_number_in_some_way(number)
      output = output+change_char_in_some_way(char)
    return(output)
  ```
Lists 1

- Indices and Slicing
  - List1 = [1, 2, 3, 4, 5]
  - list1[1:3] → [2, 3]
  - list1[:3] → [1, 2]
  - list1[1:] → [2, 3, 4, 5]
  - list1[:] → [1, 2, 3, 4, 5] (a copy of the original list)

- Functions/methods/operations that change list (mutability)
  - list1.append(6)
  - list1.extend([7, 8, 10])
  - list1.reverse()
  - List1[0] = 9  ## list1 → [9, 8, 7, 6, 5, 4, 3, 2, 1]
  - list1.sort()  ## list1 → [1, 2, 3, 4, 5, 6, 7, 8, 9]
  - list2 = [[5, 'sandwich'], [2, 'egg'], [10, 'enchilada']]  
    - list2.sort()  ## list2 → [[2, 'egg'], [5, 'sandwich'], [10, 'enchilada']]
Lists 2

- Comparisons with $>,\geq,\leq,=\text{ Max, Min, sort}$
  - `list2 = [[5,'sandwich'],[2,'egg'],[10,'enchilada']]`
  - `list2.sort()` $\Rightarrow$ `list2 → [[2, 'egg'], [5, 'sandwich'], [10, 'enchilada']]`
  - `max(list2) → [10,'enchilada']`
  - `[5,'sandwich']> [2,'egg']`

- loops with accumulator variables
  ```python
def make_derived_list(inlist):
    output = []
    for item in in_list:
        output.append(derive_something_from(item))
    return(output)
```
Practice Midterms

• I will go over this in detail today (see website)
  – practice midterm and practice midterm answers

• One of last term's midterm 2s are also online:
  – old midterm and old midterm answers

• Other midterms available online
  – On my webpage, find previous Intro to Programming Classes
  – There may be some overlap with practice midterm
  – You can look at both sections of CSCI-UA.0002 and V22.0002 (the previous number for this class)
    • However, the material covered may have changed a little, especially if you go back more than a year

• Structure is the same as the midterm:
  – 4 Part 1 questions
  – 2 out of 3 Part 2 questions
Timing

• There are 6 questions do be completed in 1:15
  – I suggest budgeting your time, e.g.,
    • 7 min X 4 part 1 questions = 28 min
      – Skip part 1 questions that are too hard and go back to them later
    • 15 min X 2 part 2 question = 30 min
    • Extra time for going over budget: 17 minutes

• The practice test may be longer and harder than the actual test, so don't let the practice test make you nervous.

• Previous year's tests are better for estimating timing.
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 fifteen 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 Midterm is Next Class

• Please feel free to ask me any questions
• We will go over the practice midterm in class: ask questions
• Suggested Studying Methods
  – Obvious
    • look over previous class lectures, notes, homeworks, etc
    • Practice problems of your choosing
  – Previous tests in other classes
  – Midterm1 for this year (both sections that I taught)
• Good luck!