Introduction to:
Computers & Programming:
Review for Midterm 1

Adam Meyers
New York University
Summary

• Procedural Matters
• What you need to Know
• Practice Test and Answers
Procedural Matters Regarding the Midterm

• The test date: February 27, 2017
  – Section 004: FA 110 at 11AM
  – Section 009: FA C10 at 8AM
  – Classrooms are crowded – switches should be pre-arranged

• 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
  – Low grades discounted if higher on Midterm2 and Final
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 variable?
    - Look for the last assignment statement (with =)
  - Open ended questions:
    - What does a function called by the main program do?
    - What are the conditions that will cause the program to end?
      - look at if and while statements to figure out answer
    - Give some plausible input and output for a program (trace what the program does).

- Section II: Write functions that solve stated problems
  - Read questions carefully
  - Common errors:
    - Functions that solve problem not stated in the question
    - Functions without input parameters that only solve problem for sample input
      - Example: function adds only 500 and 3457, but it really should add any 2 numbers
What you should know

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

• Know how to:
  – Define functions
  – Use input parameters to functions
  – Use input statements
  – 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 (or cast) 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
Algorithms

– May be asked to implement an algorithm in Python

– Pseudo Code: a series of ordered statements
  • Structured using line numbers, indents, bullets, etc.
  • Connected by logical and temporal connectors
    – if, else, unless, not, until, when
  • Includes Loops using various terms, including
    – repeat action until condition is satisfied
    – do action X times

– Flow Chart: Connected Series of Boxes
  • circles/ovals = start/end
  • rectangles = steps in processing
  • diamonds = decisions
  • arrows = sequence of steps
Functions

- Programming language Functions have 3 optional features:
  - Input (via parameters)
  - Output (return statements)
  - Side effects (print statements and other effects)
- Using the “input” function to solicit interactive input from a user
  - Do NOT reset parameters of the function with an input statement
- *print* versus *return*
  - *print* is significant for its side effect—printing to computer screen
  - *return*
    - Exits 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
Data Types

- Floats and Integers
- Strings
- Boolean (True or False)
- Nonetype (Output of void functions, like `print`)
- Ranges
String Components

• Two identical delimiters: ““, ', ""

• The Characters between delimiters including:
  – digits (0-9)
  – letters(a,b,c,..z,A,B,C,...,Z, space)
  – escape characters: \n, \t, '\n
• Sample Strings:
  – '!@##$^&*()'
  – “This is an apostrophe: ' “
  – 'This is a double quote: “ ' 
  – ""Triple quotes can include ' or “ or between them"

More on Strings

• Special characters are often represented with the backslash + a character, together these are called “escape characters”
  – 'The quote \ and the newline \n are useful'

• When printed, a string will be displayed in a way that interprets these components.
  – The quote ' and the newline are useful
  – The delimiters are eliminated and escape characters are interpreted.
Numbers

- **Integers:**
  - No Decimal Place

- **Float**
  - Limited in Length
  - Used for numbers with decimals
  - Approximations using Scientific Notation

- **Normal Division with Integer Input**
  - Output is a float

- **Integer Division** (\(/\))
  - Input/Output are integers (output is floor of answer)

- **Import Math library for many special functions/variables**
Type Conversion Functions (Numbers)

• Float
  – Converts Integers and compatible strings to floats

• Int
  – Converts floats (by truncation) to integer
  – Converts compatible string to integer

• Converted Strings can participate in math operations
  – 5 * int('5')
  – 20 / float('5.5')
Converting Non-Strings to Strings

- `str(5.55)`
  - '5.55'

- Makes a string out of any type of object (using definition of that object)

- Once converted, non-strings can be combined with strings through concatenation
  - 'The number is '+str(5)
  - output = 5+100
  - 'The sum of 5 and 100 is '+ str(output)
Arithmetic Operators & the Assignment Operator

• Know all the mathematical operators and what they do: +, -, *, **, /, //, %

• Be familiar with the two equal signs
  – The assignment operator =
  – The test for equality operator ==

• Understand how most of the operators can be restated as functions
Boolean Operators

• **and, or, not**
  - True and True $\rightarrow$ True, True and False $\rightarrow$ False, False and True $\rightarrow$ False, False and False $\rightarrow$ False
  - not(True) $\rightarrow$ False, not(False) $\rightarrow$ True
  - False or False $\rightarrow$ False, True or False $\rightarrow$ True, False or True $\rightarrow$ True, True or True $\rightarrow$ True

• **<, >, <=, >=**
  - Expected meanings from math

• **==, !=**
  - 'is equal to', 'is not equal to'

• **in** – 2 purposes
  - in *for* loops – relates item to sequence
    - for character in 'abcd':
      - print(character)
  - Boolean operator that tests for membership:
    - 'a' in 'abcdefg' $\rightarrow$ True
    - 5 in range(4) $\rightarrow$ False
Making Code Readable

• Comments
  – ## Know How to use comments
  – ## Know Why to use comments

• Naming Variables and Functions
  – Choose names that are self explanatory

• Use Parentheses with Operators
  – Do not rely on order of operations
  – Many operators are not covered by PEMDAS
  – You may not know what the defaults are
Identifiers

- Functions
  - How to define functions
  - Legal names for functions
  - Using colon, parentheses and indents

- Variables
  - Legal names of variables
  - The difference between local and global variables
If/elif/Else Statements

• Syntax (elif and else parts optional)
  
  If + boolean-expression + :
    body
  elif + boolean-expression + :
    body
  else:
    body

• Example:

  def classify_integer(integer):
    if integer==0:
      return('zero')
    elif (integer%2) == 0:
      return('even')
    else:
      return('odd')
for Loops

• The first line of a for loop:
  – for VARIABLE in SEQUENCE:
    • VARIABLE is name of variable
    • SEQUENCE is a sequence
      – a range is a sequence of numbers
      – a string is a sequence of characters, etc.

• The body of the loop is indented beneath the first line
  – The body repeats one time for each element in sequence
  – On each iteration, VARIABLE is set to the next item

• for letter in 'word':
  print(letter)
    • 1\textsuperscript{st} iteration: letter == 'w'; 2\textsuperscript{nd} iteration: letter = 'o', etc.
    • Thus each letter in 'word' is printed on its own line
Sample loops” try in IDLE

- Sample for loop
  - def diagonal_print(word):
    number = 0
    for letter in word:
      print(number*' ',letter,sep='')
      number=number+1

- Sample while loop
  - def guess_my_number ():
    import random
    my_number = str(random.randint(1,10))
    guess = '100'
    while guess !=my_number:
      guess = input('Guess my number: It is between 1 and 10: ') 
      if guess !=my_number: 
        print('You guessed wrong. Try again')
    print('You are correct. My number is',my_number)
Basics of Planning a Program

- Figure out the big steps first and “pretend” that you have already written most of the functions.
- Then write the functions that you assume that you need. Writing these functions may involve pretending that more functions exist, but that is OK.
- Repeat process until you don't have to pretend anymore because the functions are simple enough to just write and be done.
Reminder: A Test is a Game

- Tests are imperfect for measuring a expertise because, independent of expertise, some people know how to play the test game better than others.

- How to win the test game
  - Study sample tests (including 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
  - Budget Your time
    - 4 section 1 questions X 7.5 minutes = 30 minutes
    - 2 section 2 questions X 15 minutes = 30 minutes
    - 15 minutes left over to finish whatever you need to finish
The Midterm is Next Class

• Please feel free to ask me any questions
• We will go over the practice midterm in class
  – Longer than actual Midterm 1
  – Individual problems may be harder or easier than those on real midterm
  – good for preparation, but don't let it scare you
  – ask questions

• Suggested Studying Methods
  – Obvious
    • look over previous class lectures, notes, homeworks, etc
    • Practice problems of your choosing
  – Look at previous tests from Python classes I taught
    • My website: http://nlp.cs.nyu.edu/people/meyers.html
    • Old tests from both CSCI-UA.002 and V22.0002 websites may be helpful, although sometimes different material was covered.
    • Good luck!