Introduction to: Computers & Programming: Final Review

Adam Meyers
New York University
Summary

• Procedural Matters
• What you should know
• The Structure of the Final
• Sample Final Questions
Procedural Matters

• Day and Time Listed on the class Website  
  – June 30, 2011: 11:30 – 12:05

• Room will be the same room as the class

• I will take attendance, please bring your ID card.

• You will have 1 hour and 35 minutes

• The test booklet and the answer booklet  
  – Write the answers on either or both
  – Please write your name on both
    • otherwise, your answers could be lost
Grading

• As with the midterms, the final will be graded on a curve
• The Final counts for 42% of the total grade
  – Grade = 33% Midterm + 42% Final + 25% HW = 100%
• If you missed a midterm, the Final will count for 75%
• All grades converted to 4.0 scale before averaging, although it is possible to get up to 4.3 on Homework if you did the extra credit assignment.
Problem Solving Through Programming

• An *algorithm* is a plan for a program
  – Implementation of an algorithm is a computer program that basically follows that plan
  – Some questions may provide an algorithm for implementing the solution to a problem

• How do you plan a program top down?
  – Write code at the top most level, including functions you have not written yet. E.G:
    
    ```python
    def build_wall(wWidth, wHeight, bWidth, bHeight):
        for row in range(1, math.ceil(wHeight/bHeight)):
            build_brick_row(wWidth, bWidth)
            go_up(bHeight+1)
    ```

• The first step for writing a program that builds a 2D wall out of bricks:
  – the dimensions of the wall are wWidth X wHeight
  – the dimensions of each brick are bWidth X bHeight

• The next step would be to write build_brick_row
  
  ```python
  def build_brick_row(wWidth, bWidth)
      for brick_num in range(1, math.ceil(wWidth/bWidth)):
          build_brick(bWidth)
          go_forward(bWidth)
  ```
Classes of Python Objects

• While we have not really focused on *Object Oriented Programming*, we have encountered Python objects

• What are the properties of the classes of Python objects?
  – Which functions take which type of objects as arguments?
  – Which methods are used with which objects?
  
    • e.g., string.upper() is a string method

  – Which objects contain other objects?
  
    • How are these objects accessed?
    
    • How can we access a defined subset of these objects?

• Objects we have encountered: integer, float, boolean, none, range, string, tuple, list, dictionary, set, turtle.Turtle, turtle.Screen, function, stream (TextIOWrapper)
Information about Objects

• Numbers:
  – Functions: arithmetic operators, math functions, etc.
  – Subtypes:
    • integer: number with no decimal
      – // – operator that does integer division
      – int – converts compatible non-integers to integers
    • Float
      – / – division
      – float – maps compatible non-floats to floats

• Boolean: True or False
  – Expressions with Boolean operators are evaluated as True or False
  – Boolean Expressions are a crucial part of control structures (if, elif, while, etc.)
  – Other elements are function as True or False
    • Both 0 (zero) and None function as False
    • Most other items function as True
Information about more Objects

• None – Type is NoneType
  – The object that is returned by functions that don't return anything
  – Treated as \textbf{False} in a Boolean expression

• Objects that are defined in modules
  – Examples
    • `turtle.Turtle()` – creates a turtle object
    • `turtle.Screen()` – creates a screen object
    • `re.search object (aka, SRE_Match)`
  – These have methods for using these objects
    • `turtle.fd(10)`
    • `search_object.start()`
    • etc.
Sequences: ordered collections of objects

- All sequences
  - Functions: `len(sequence)`
  - Indices and Slices:
    - `sequence[0], sequence[1]...sequence[length-1]`
    - `sequence[-length]...sequence[-1]`
    - `sequence[X:Y], sequence[X:], sequence[:Y]`

- `range` – a sequence of consecutive integers
  - Useful in `for` loops

- set and tuple (not on final, similar to list, but in different ways)

- string and list – Next 2 slides
Strings

• Special Strings
  – Empty string: string of length 0
  – character = string of length 1
    • ord(char) → maps character to unicode value
    • chr(number) → maps unicode number to character
  – Special characters
    • \n – newline, \r – return, \t – tab, \\ – backslash,
      \’ – single quote, “” – double quote
• String = sequence formed by concatenating zero or more characters to the empty string.
• Special operator for strings: +
• String methods: string.lower(), string.upper(), string.split(separator), string.find(substring), string.strip(substring), ...
Strings: Slide 2

• Converting Characters:
  – `chr(ord(character) + number)` → can be used to convert characters, e.g., upper to lower case

• `re.search(pattern, string)`
  – Returns a search pattern if the pattern matches
  – `pattern.start()` and `pattern.end()` – return beginning/end positions in string
  – `pattern.group(0)` – returns the substring that matched the pattern

• regular expressions – next slide
Regular Expressions

- “|” used to represent “or”
  - 'Dog|dog' means 'dog' or 'Dog'

- [ ] are used to list alternative characters
  - '[Dd]og' means 'dog' or 'Dog'
  - Inside [], ^ means not
  - A hyphen (-) indicates a range of characters [a-z] means one lowercase character (a or b or c or d...)

- A period . is used to mean any character

- $ = end of string and ^ = beginning of string (^ means not inside [])

- pattern* – means 0 or more instances of pattern

- pattern+ – means 1 or more instances of pattern

- Example patterns:
  - '^A.*[:]' – a line that begins with a capital A and contains a colon (:)
  - '[a-z]+ [a-z]' – One or more lowercase letters followed by a space and another lowercase letter.
Lists

• A sequence of zero or more objects of any type
  – [] empty list
  – [item, item, ....] – non-empty list

• Lists are mutable:
  – list[2] = 'abc'
    ## puts 'abc' in position 2 of list
  – They can be changed by functions
  – They can contain themselves

• List methods
  – list.index(item), list.count(item), list.append(item), list.extend(list), list.sort()
Dictionaries

- `{key1: value1, key2: value2, key3: value3...}
- A hash table, an efficient structure for associating keywords with values, typically representing a particular attribute
- Mutable, like lists
  - `dictionary[key] = value`
    ```
    # assigns value to key in the dictionary
    ```
- Unordered (unlike lists)
- Like sequences in some ways:
  - The `in` operator is used with key words
    - `key2 in {key1: value1, key2: value2} → True`
    - `for key in dictionary: ....`
- Dictionary methods
  - `dictionary.clear()`, `dictionary.copy()`
Streams

- Streams (we focused on text streams)
  - \textit{Instream} = \texttt{open(file, 'r')}  
    - creates an input stream called instream
  - \textit{Outstream} = \texttt{open(file, 'w')}  
    - creates an output stream called outstream
  - \texttt{stream.close()}  
    - closes a stream

- Input stream should be treated as lists of lines
  - \texttt{line} = string ending in \texttt{os.linesep}
  - \texttt{os.linesep (os module) = \r\n (Windows) or \n (others)}

- Output streams should be written to:
  - \texttt{stream.write(string+os.linesep)}
Operators and Functions

• Operators
  – The assignment Operator (=)
  – Arithmetic Operators (+,-,*,/,//,**,%)
  – Boolean Operators (and, or, not, <, >, <=, >=, ==, !=, in)
  – Precedence: Rules of Precedence or ( )
    • PE (M | D) (A | S)

• Functions
  – return – exits function and returns value
  – parameters – variables associated with arguments of function
    • keyword parameters – name parameters which use an = and which have defaults (like the end='\n' in the print function)
  – other variables can be set within function, including those using the input function
Variables

• Names associated with objects
• Global variables – variables available everywhere
• Local variables – variables with specific scopes, typically a function (but can be smaller blocks)
  – locally assigned variables and parameters
• By default, a name references the most local variable when there are two variables with the same name.
• If declared global (with the global keyword), a variable within assigned/modified within a function can be global
The Flow of Control

• A block of text:
  – A sequence of commands indented the same amount
  – Executed in Order of occurrence
    • print(1)
    • print(2)
  – If one block of text is contained in another
    • The sub-block begins where it occurs and ends before the higher block resumes

• A function:
  – A named block of text that is indented beneath `def`:
  – A function call executes this block
  – It is possible to call a function within itself (recursion)
    • Optional for final, but useful
The Flow of Control: Conditionals

- Conditional blocks execute if boolean is True or False
  - if (boolean):
    block
  - elif (boolean):
    block
  - else:
    block
- Decision Tree
  - Series of nested conditions used to solve a problem
  - Typically True/False decisions forming binary branching tree
Flow of Control: For Loops

• For Loops contain blocks that execute once for each item in a sequence
• Examples:
  – for character in string:
    block
  – for item in list:
    block
  – for number in range(5):
    block
• A variable (character/item/number) is set/reset to each item in a sequence.
• For loops can also use indices to manipulate sequences, e.g.,
  – for index in len(sequence):
    block
The Flow of Control: While Loops

• while (boolean):
  block

• Block repeats while boolean expression is true

• The block in the loop either:
  – Never executes
  – Executes repeatedly forever
  – Or the boolean expression contains a variable such that changing the variable can change the value of the boolean expression from True to False

• It is often helpful to initialize variables outside of a while loop that are updated inside the while loop and effect the value of the boolean expression.
The Flow of Control: All Loops

• Accumulators and Counters
  – Before a loop
    • Initialize accumulators: string = " or list = []
    • Initialize counters: number = 0
  – Update inside loop
    • string = string + character
    • list.append(item)
    • number = number + 1

• Nested Loops:
  – A loop within a loop is repeated along with the rest of the block
  – For example,
    • for num1 in range(5):
      for num2 in range(4):
        block
    • The inner loop is repeated 5 times and the block is repeated 4 X 5 = 20 times
The Print Function

- Prints objects according to the way each object is defined.
  - Strings
    - print without delimiters (quotes at beginning and end)
    - special (escape) characters are interpreted, e.g., \n → a new line, \' → ', \" → ", etc.
  - Numbers are printed the same as they are typed
  - Infinitely nested lists are represented as [...]
  - A turtle.Turtle object is printed as:
    - <turtle.Turtle object at 0x14d3410>

- The print function takes the following parameters
  - 0 or more unnamed parameters divided by commas
  - Followed by 2 named parameters
    - sep= with default of '' (a space)
    - end= with default of '\n'
Modules

- **time** – `time.sleep(seconds)`
- **re** – `re.search(pattern, string)`
- **random** – `random.random()`
- **math**
  - `math.ceil(number)`, `math.floor(number)`
  - `math.pi`
- **turtle**
  - `turtle.Turtle()`, `turtle.Screen()`
  - `my_turtle.fd(number)`, `my_turtle.left(number)`, `my_turtle.circle(number)`, `my_turtle.pd()`, `my_turtle.pu()`
- **os**
  - `os.getcwd()`, `os.listdir(path)`, `os.path.isdir(path)`, `os.path.isfile(path)`
  - `os.linesep`, `os.sep`
Other Important Python Commands

- **input(prompt)**
  - The function prints prompt (a string) to the screen
  - and waits for a user to type in an output string and hit enter.
  - The output string is returned by the function

- **import module**
  - Makes available global variables, objects and functions found in module, provided that the names of these items are prefixed by the name of the module

- **Comments**
  - After #
  - Comment can follow code or be on its own line

- **Backslash (\)**
  - Used to divide lines that are too long
Files

• A file is a named collection of data on a storage device with a beginning and an end.
• A directory (or folder) is a named item that can contain other directories and/or files
• What is the file structure of a storage device?
  – A tree structure starting with the root (/ or C:)
  – A path is a sequence of identifiers, connected by a / or a \\, that uniquely identifies a file or directory in the tree structure, e.g., this file is:
    • /Users/adam/Desktop/python/Class Talks/Final Review.odp
  – The path is both the name and the address of the file
• The root is the directory that contains all other files and directories and is not part of another directory.
The Structure of the Final

• There will be two sections of the Final, each will be worth ½ of the total grade for the final, e.g., 50 + 50 or 60 + 60

• The first half will be similar to the midterm section 1.
  – Given some code, you will be expected to indicate what that code does (its output, what prints out, etc.)
  – You will have some choice of questions, e.g., choose 4 out of 6 questions to answer. If you answer all 6, I will count the first 4 (unless you cross one out or give some indication of your choice).
  – The level of difficulty will be mixed. This section will include some easy questions and some more difficult ones.

• The second half will be similar to the 3rd section on the midterm.
  – You will be expected to write programs.
  – I will give you some choice, e.g., choose 2 out of 3.
  – There will be at least one question that is easier than the others.
Sample 'Easy' Section 1 Questions

• Sample Question 1: After the following loop executes, what is the value of `multiplier`.

```python
multiplier = 1
while (multiplier < 20):
    multiplier = multiplier * 3
```

• Sample Question 2: What is returned if the user types 'yEs' after this function executes?

```python
def yes_or_no ():
    answer = ''
    while (not (answer in ['yes','no'])):
        answer = input('Type “yes” or “no” please ')
        answer = answer.lower()
    if answer == 'yes':
        return(True)
    else:
        return(False)
```
Sample 'Difficult' Section 1 Question

• Question: What would the following function call cause to be printed out, given the accompanying function definition?

```python
def sails(width, columns, rows):
    for row in range(rows):
        for number in range(1, width+1):
            for column in range(columns):
                print('~'*number + ' '*(width-number), end='')
        print()

sails(5,3,3)
```
'Easy' Sample Section 2 Question

Write a 'madlibs' program, given the text below. Your program should first ask the user for instances of various parts of speech or semantic classes (person name, verb, noun, ...). The program should replace the appropriate words in the text with the user's words. If a word in the original text is repeated, the replaced word should repeat as well. The text follows:

– Mary had a little lamb, little lamb, little lamb

   Mary had a little lamb, whose fleece was white as snow

For example, suppose you select: Mary, little, lamb, fleece, white, and snow as words that should be replaced. Then a user's choices may yield the following result:

– Herman had a blue pencil, blue pencil, blue pencil

   Herman had a blue pencil, whose lead was red as fire
Difficult Section 2 Question

• Write a program that generates 5 random playing cards (a poker hand).
  – There are 13 different faces:
    • 9 numbered cards from 2 to 10
    • Ace, Jack, Queen, King
  – There are 4 suits for each card:

• Make sure that your program does not generate the same card twice.

• Write an additional set of functions to evaluate the poker hand (See next Slide).
Difficult Problem 2

- If all of the cards in the hand have the same suit, then return 'Flush'

- Determine how many cards have the same face:
  - If four cards have the same face, return '4-of-a-kind'
  - If three cards have the same face, return '3-of-a-kind'
  - If two cards have the same face, return '2-of-a-kind'

- This is a minor simplification of the real problem. You do not need to identify: straights, straight-flushes, two-pairs, or a full-house.
Next for Preparation

• We can go over the sample text
• I can answer any questions about anything
• Good Luck