Midterm Exam for V22.0002 – Spring 2016 – Version 1

Name: ________________________________
Net ID ________________

There are 2 sections, the first section is worth 50 points and the second section is worth 50 points, for a total of 100.

It is essential that you PUT YOUR NAME AND NET ID ON ALL TEST MATERIALS. It can be difficult to identify the author of an unsigned test and it would be better to avoid this problem.

There is a GLOSSARY OF TERMS at the end of the test. Please feel free to look up some of the basics in this glossary. I will also answer any reasonable look-up style questions as I am more interested in your ability to reason than your ability to memorize.

Section 1: (50 points) Each example consists of Python code, followed by questions and places to fill in answers. Please read all questions carefully and answer them as precisely as possible.

Assume that there are no bugs in the code that will make the system crash, although the code may not solve a problem perfectly. If you find anything that you think is a bug, there is either a typo (and I should fix it for everyone) or you are mistaken.

Sample Question A:

```python
output = '1'+1'
```

**Question: What is the value of the variable output?**

**Answer: '11'**

**Note:** Attention to detail is important. The quotes indicate that it is a string. Partial credit is possible. For example, leaving out the quotes would have lost just a little bit, but answering 2, would have resulted in an incorrect answer.

**Question 1**

```python
def number_sandwich(filling,bread):
    output_string = str(bread)+str(filling)+str(bread)
    output = int(output_string)
    return(output)

answer = number_sandwich(57,1)
```

**Question 1a: What is the value of the global variable answer after the code executes?**

**Question 1b: Is this value a string, an integer or a float?**
Question 2

def print_pet():
    print('\_/\n  |\n  \__\n
Please note that \ represents the single character \ because the backslash is an escape character, a character used in two character combinations to represent special characters: \n represents a newline character, \t represents a tab and \ represents the backslash itself.

Question: Please draw (approximately) what would be printed out by executing the above code.
Question 3

def decorate_word(input_word, decoration):
    output = decoration
    for letter in input_word:
        output = output + letter + decoration
    return(output)

def make_pretty_word_pair(word1, word2, decoration):
    output1 = decorate_word(word1, decoration)  ## line 2
    output2 = decorate_word(word2, decoration)  ## line 3
    final_output = output1 + output2
    return(final_output)

answer = make_pretty_word_pair('Waffle', 'House', '*')

Question 3a. What is the value of answer after the above code is executed?

Question 3b. During the execution of the function make_pretty_word_pair, What is the values of the variable output1 after decorate_word is executed in the line ending in the comment ## line 2?

Question 3c. What is the value of the variable output2 after decorate_word is executed in the line ending in the comment ## line 3?
Question 4

def hot_or_cold(target, last_number, current_number):
    new_difference = abs(current_number-target)
    ## the function abs(number) returns the absolute value of the number
    if last_number=='nothing':
        if new_difference >= target:
            if new_difference < (2 * target):
                print('Hot')
            elif new_difference < (3 * target):
                print('Medium')
            else:
                print('Cold')
        elif new_difference < target:
            if new_difference <(.25 * target):
                print('Hot')
            elif new_difference < (.5 * target):
                print('Medium')
            else:
                print('Cold')
    elif (last_number > target) and (current_number<target):
        print('Cold')
    elif (last_number < target) and (current_number > target):
        print('Cold')
    else:
        old_difference = abs(last_number-target)
        if new_difference<old_difference:
            if new_difference <= (.5 * old_difference):
                print('Hot')
            else:
                print('Medium')
        elif new_difference==old_difference:
            print('That is the same guess as last time.')
        elif new_difference>old_difference:
            print('Cold')
    ## for numbers A and B, A == B, A < B or A>B (no else necessary)

def guess_my_number(my_number):
    current_guess = .0000001
    number_of_guesses = 0
    while my_number != current_guess:
        if number_of_guesses>1:
            hot_or_cold(my_number, last_guess, current_guess)
            print('Try again!'
        elif number_of_guesses>0:
            hot_or_cold(my_number, 'nothing', current_guess)
            print("Try Again!")
        last_guess = current_guess
        current_guess = int(input('Guess My Number: '))
        number_of_guesses = number_of_guesses + 1
        print('You guessed my number: ',my_number)
        print('It took you',number_of_guesses,'guesses.')
    return(number_of_guesses)

guess_my_number(57)
The above function call begins a guessing game in which the answer is 57. The system gives clues in the form of *Hot, Cold or Medium*, depending on the user’s guess. Initial guesses (when there is no previous guess) are rated numerically by how close they are to the right answer (see code above) and these ratings are converted into *Hot, Cold or Medium*. For subsequent guesses the system compares a current guess with a last guess, printing out either *Hot, Cold or Medium*, depending on whether the change in guess is in the “right” direction as per the code, and if the guess is closer to the right answer, how much closer it is. A warning is issued if the same number is guessed twice in a row.

Note: The function *abs* above takes the absolute value of a number, e.g., *abs(-5)* returns 5; and *abs 5* also returns 5.

**Question 4a. Assuming that the first guess by the user is 100, what clue would the program print out?**

**Question 4b. If the user’s second guess was 150, what would the program print out?**

**Question 4c. If the user’s third guess was 70, what would the program print out?**

**Question 4d. Provide at least 3 more incorrect guesses, along with what the program would print in response. Then have the user guess correctly (57) and provide the program response to that as well.**
Section 2 (50 points): Write 2 of the 3 questions in this section. For each question, you do, write a Python program as specified. If you choose to answer all 3 questions, please indicate which ones you would like to count for the test.

Question 5: Write a program that will accept one word at a time from a user and compute the average length of the words entered.

- The program should query the user with an input statement. The user should choose between: entering a period and entering a word. The input statement should be inside a while loop.
- If the user enters a period, this should cause the while loop to terminate. Otherwise, the length (len) of the word should be used to compute the average.
- The main function of your program should return the average.
**Question 6:** Write a program that asks questions to a user about a flying object that they observe and classifies these objects based on the user’s answers. Implement your program based on the questions and the decision tree provided below. You can also assume that the text of each question is encoded as a variable, e.g., you can use the variable name `question_1` to represent the string ‘Does it have feathers’ and `question_2` to represent the string ‘Is it alive’, etc.

Note: A shuttlecock is a ball-like object that has (typically plastic) feathers. It is hit back and forth in a tennis-like game called badminton.

1. Does it have feathers?
2. Is it alive?
3. Is it a weapon?
4. Does it have two legs?
5. Does it make a buzzing noise?
6. Is it made of paper?
7. Does it have wings?
**Question 7:** Write a program that will print a rectangle made up of alternating horizontal stripes of ‘X’ and ’O’ characters. The specifications are as follows:

- The program should consist of a single function that takes two parameters: height and width.
- *Height* should indicate the number of rows that will be printed.
- *Width* should indicate the total number of characters in each row.
- Rows should alternate between ‘X’ characters and ’O’ characters, such that the first row should be all ’X’ characters, the second should be all ’O’ characters, and so on with each odd row being all ’X’ and all even row being all ’O’.

For example the function call striped_rectangle(5,10) should produce the following output:

```
XXXXXXXXXX
OOOOOOOOOO
XXXXXXXXXX
OOOOOOOOOO
XXXXXXXXXX
```
Basic Stuff to Look Up for the Test

1. Some Basics

- **return**(X) causes the current function to exit and cause the expression represented by the function call to evaluate as X. For example, given the following steps, the value of *output* would be 5:

```python
def add(num1, num2):
    return(num1 + num2)
output = add(2, 3)
```

- **print**(X) prints X to the screen. This is only for the benefit of the user. It is not useful for having programs interact.

- The parameters of a function are the local variables inside of the parentheses in the function definition. They are useful when you have functions call functions.

- **input**(prompt) is used to ask a human being a question so that a program can interact with a human being. This is useful when you want a human being to enter information interactively. **input** statements should be used only when human interaction is appropriate. **input** statements return a string corresponding to what the user typed in. It may be necessary to convert this string to some other data type, e.g., an integer (with **int**) or a float (with **float**).

- The operator + will add two numbers or concatenate two strings

- The operator * will multiple two numbers or print a string some number of times.

2. sequences

- object made up of other objects in an order
- the function len(sequence) returns the number of items in the sequence
- the operator in tests for membership in sequence, e.g., ('a' in 'abc') would have the value True.
- sequences are used in **for** loops (see below)
- ranges
  - **range**(5) is approximately equivalent to [0,1,2,3,4]
  - **range**(1,5) is approximately equivalent to [1,2,3,4]
- Strings
  - an empty string has zero characters ""
  - strings are sequences of characters, e.g., 'Hello World!' consists of the items ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!']

3. Division and Modulus

- 5 // 2 == 2
- 5/2 == 2.5
- 5%2 == 1

4. print

- **sep** – separator between items
- **end** – printed at the end of print statement

5. **for** loops

- First Line: for **VARIABLE** in **SEQUENCE**:
- **VARIABLE** is set to each item in the sequence one at a time
• The indented body repeats once for each item in sequence (for each setting of VARIABLE).
• It is common to exit a loop of any kind by using a return to exit the function.
• It is common to initialize a variable outside a loop (called an accumulator) that then gets incremented inside the loop.

6. if statements

• the first line of an if statement consists of if BOOLEAN-EXPRESSION:
• the body of text indented under the first line is executed if the BOOLEAN-EXPRESSION evaluates to True
• the if statement can be followed by optional elif statements of the same form, except that the first line begins with elif. Each elif statement is only evaluated if the BOOLEAN expressions in the if and elif statements leading up to this one are False.
• The block of if and optional elif statements can end with an optional else statement. The first line is simply else:. The body of text under else executes if the Boolean expressions for all previous if and elif statements in the sequence evaluate to False.