Midterm Exam for V22.0002-001
June 13, 2011

Name: _____

There are 3 sections, each worth 35 points. The maximum score on the test will be 105.

It is essential that you **PUT YOUR NAME 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: Each example consists of Python code, followed by a question and a place to fill in an answer. 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:

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

Question: What does output equal?

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:

```
addition = 1 + 1
concatenation = '1'+'1'
concat_addition = concatenation * addition
```

Question: What is the value of the variable **concat_addition** after the code above executes?

Answer:

Question 2:

```
output = 0
for number in range(10):
    output = output + (number % 3)
```

Question: What is the value of the variable **output** after the code above executes?

Answer:

Question 3

```
def asterisk_pattern(spaces,asterisks,spaces_first):
    if spaces_first:
        print(spaces*' ',asterisks*'*',sep='',end='')
    else:
        print(asterisks*'*',spaces*' ',sep='',end='')

def main():
    total = 6
    for number in range(total):
        asterisk_pattern(number,total-number,True)
        print()

main()
```

Question: Draw (approximately) the configuration that is printed out when the code above executes.

Answer:

Question 4:

```
def yes_or_no(question):
    answer = input(question+' Please answer \'yes\' or \'no\': ')
    if (answer == 'yes') or (answer == 'Yes') \
       or (answer == 'Y') or (answer == 'y'):
        return(True)
    else:
        return(False)

def classify_the_beast ():
    print('Use this program to classify a beast.')
    backbone = yes_or_no('Does the beast have a backbone?')
    if backbone:
        feathers = yes_or_no('Does your beast have feathers?')
        if feathers:
            print('Your beast is a bird!')
            return('bird')
        else:
            gills = yes_or_no('Does your beast have gills (at any age)?')
            if gills:
                limbs=yes_or_no('Does your beast have limbs with fingers or toes?')
                if limbs:
                    print('Your beast is an amphibian!')
                    return('amphibian')
                else:
                    print('Your beast is a fish!')
                    return('fish')
            else:
                scaly_skin = yes_or_no('Is its skin scaly or shell-covered?')
                if scaly_skin:
                    print('Your beast is a reptile!')
                    return('reptile')
                else:
                    ear_bones = yes_or_no('Does your beast have middle ear bones?')
                    mammary = yes_or_no('Does the female have mammary glands?')
                    hair = yes_or_no('Does your beast have hair?')
                    if ear_bones or mammary or hair:
                        print('Your beast is a mammal!')
                        return('mammal')
                    else:
                        print('Congratulations! You found a new type of vertebrate!')
                        return('Unclassified vertibrate specimen')
    else:
        print('Your beast is a slithering slimey crawley thing. ')
        return('Invertebrate')
```

Question: Fill out the table on the next page, indicating the classification returned by the above function given the listed answers to the yes/no questions.

Question 4 (continued): Please put answers in table.

beast	**yes/no answers**	**Classification Returned**
dragon	yes, no, no, yes	
unicorn	yes, no, no, no, yes, yes, yes	
mermaid	yes, no, yes, yes	
dust bunny	no (to all questions)	

Question 5:

```
import turtle
my_screen = turtle.Screen()
my_turtle = turtle.Turtle()

def multi_turtle_circle(number):
    distance = 50
    for num in range(number):
        my_turtle.pd()
        my_turtle.circle(distance)
        my_turtle.left(360/number)
    my_turtle.pu()
    my_turtle.fd(distance)
    my_turtle.left(90)
    my_turtle.pd()
    my_turtle.circle(distance)

multi_turtle_circle(3)
```

Question: Draw the shapes (approximately) that the turtle draws when the above code is executed. The circle method (my_turtle.circle(distance)), draws a circle where *distance* is the radius. It begins and ends at the starting position. The radius of the circle is 90 degrees to the left of the starting point. For example, a circle drawn at the turtle's starting position would be above that position and the starting position would be the bottom of the circle.

Answer:

Section 2: This section consists of 2 sample pieces of code containing syntax errors. On the appropriate line numbers in the table below or next to the function, please list the errors (if any), along with possible corrections. Sample output is provided to indicate what the program is intended to do.

A minimum number of errors is listed. You do not need to find more errors than that number. For example, if the description says to find three errors, you only need to find three, even if there are actually four or five errors. If you identify more errors than the minimum, you will be graded on the total number of errors that you list. For example if you list 3 actual errors and you are mistaken about a fourth one, you will get 3/4 of the total number of points available.

There is assumed to be at most one error per line. If there are two errors on a line, they count as part of the same error.

The comments are meant to be instructive. Please do not look in the comments for errors. Assume that the CONTENTS of comments are error-free (but of course syntactically incorrect comments should be marked as incorrect).

Sample Question B: The function minus subtracts the decrement from the total. **There is only one error.**

Line	**Code**
1. 2 3.	def minus(total,decrement) output = total - decrement return(output)
Line	**Correction of Error**
1.	need to add colon (:) at end of line
2.	
3.	

Question 6: This function polls people to rate the quality of their lunch at a particular sandwich shop. Find four errors.

Line	Code
1.	def number_one_to_five (attribute):
2.	answer = input('Please rate '+attribute+' on a scale from 1 (bad) to 5 (great) :')
3.	return(answer)
4.	def rate_your_sandwich_experience():
5.	score = '0'
6.	print('Print Thank you for agreeing to complete questionnaire about your sandwich experience.")
7.	print('Your results will be Entered into our database, so we can serve you better in the future.')
8.	score = score + number_one_to_five('freshness of bread')
9.	score = score + float(number_one_to_five('freshness of filling'))
10.	score = score + .5(float number_one_to_five('tastiness of condiments'))
11.	score = score + (float(number_one_to_five('color of sandwich'))*.25)
12.	return(score)

Line	Correction of Error
1.	
2.	
3.	
4.	
5.	
6.	
7.	
8.	
9.	
10.	
11.	
12.	

Question 7: This function calculates a person's hat size based on the circumference of their head. Hat sizes are derived by finding the diameter of the head from the circumference (measured 1/8 of an inch above the ears) and rounding the result to the nearest eighth of an inch. Note that the function *round*, rounds decimals below .5 to the previous integer and decimals .5 or greater up to the next integer. Find four errors.

Line	Code
1.	def hat_size(head_circumference)
2.	import math
3.	diameter = head_circumference / pi
4.	hat size is the diameter rounded to the nearest eighth
5.	whole_number = int(diameter // 1)
6.	eighths = round((diameter % 1)*8)
7.	if eighths == 8:
8.	output = str(whole_number + 1)
9.	elif eighths == 0:
10.	output = str(whole_number)
11.	elif eighths == 2:
12.	output = whole_number + ' 1/4'
13.	elif eighths == 4:
14.	output = str(whole_number)+' 1/2'
15.	elif eighths == 6:
16.	output = str(hole_number)+ ' 3/4'
17.	else:
18.	output = str(whole_number)+ ' '+str(eighths)+'/8'
19.	print('Your hat size is',output)
20.	return(whole_number+(eighths/8))

Line	Correction of Error
1.	
2.	
3.	
4.	
5.	
6.	
7.	
8.	
9.	
10.	
11.	
12.	
13.	
14.	
15.	
16.	
17.	
18.	
19.	
20.	

Section 3: Write Programs as specified.
Question 8: Write a program that creates a box of asterisks around a set of three sentences as described below. The three sentences can either be parameters of the function or they can be entered by the user using input statements. Your program should do the following:

1. Set a variable *maximum_length* to six more than the length of longest of the three sentences. Remember that the function **len(sequence)** returns the length of the sequence.
2. Print a line consisting of *maximum_length* asterisks.
3. Print each of the sentences centered within asterisks such that the total length of the string equals *maximum_length*. The asterisks preceding the sentence should be either the same as or one more than the asterisks following the sentence. For example, if the sentence is 45 characters and *maximum_length* is 61 characters, the line should consist of 8 asterisks, followed by the sentence, followed by 8 more asterisks. If the sentence is 8 characters long with the same *maximum_length*, the program should print 27 asterisks, the sentence, then 26 asterisks.
4. Finally print another line consisting of *maximum_length* asterisks.

Some sample input and output follows:

```
embed_3_sentences_in_asterisks('Hello World!','This is a Python print statement',\
'You can recognize it by its syntax')

**************************************
**************Hello World!**************
****This is a Python print statement****
***You can recognize it by its syntax***
**************************************
```

EXTRA CREDIT (DO ONLY IF YOU HAVE TIME): Alter the above program so all spaces in sentences are replaced with asterisks. If you do the extra credit, the output above would be the following instead:

```
embed_3_sentences_in_asterisks('Hello World!','This is a Python print statement',\
'You can recognize it by its syntax')

**************************************
**************Hello*World!**************
****This*is*a*Python*print*statement****
***You*can*recognize*it*by*its*syntax***
**************************************
```

Question 9: Write a game program that tries to guess a number from 1 to 1000 picked by the player in less than 10 tries. Use a *for* loop to ensure that the computer has at most 10 tries. If the computer guesses correctly in less than 10 tries, a return will exit the *for* loop.

The player is instructed to pick a number from 1 to 1000. The computer will guess numbers between 1 and 1000 and the player will answer either: *High* if the computer's guess is too high; *Low* if the computer's guess is too low or *Correct* if the computer's guess is correct. The game ends when either: (1) the for loop ends and the computer has not guessed the right answer; or (2) the computer guesses correctly.

The program should always guess the midpoint between the highest and lowest possible numbers (rounded to the nearest whole number). Thus the computer's first guess will always be 500 or 501 (depending on how you are rounding). When the player tells the program that a guess is *high*, the program should reset its high to one less than its current guess. When the player tells the program that a guess is *low*, the program should reset its low to one more than the current guess. Thus guessing the average between high and low will always yield a new guess in the right direction. Below is a sample game output in which the program won in 10 rounds.

```
>>> high_low()
I will guess 500. Please answer 'correct', 'high' or 'low' high
I will guess 250. Please answer 'correct', 'high' or 'low' high
I will guess 125. Please answer 'correct', 'high' or 'low' high
I will guess 62. Please answer 'correct', 'high' or 'low' high
I will guess 31. Please answer 'correct', 'high' or 'low' low
I will guess 46. Please answer 'correct', 'high' or 'low' low
I will guess 54. Please answer 'correct', 'high' or 'low' low
I will guess 58. Please answer 'correct', 'high' or 'low' high
I will guess 56. Please answer 'correct', 'high' or 'low' low
I will guess 57. Please answer 'correct', 'high' or 'low' correct
I won in 10 turns.
>>>
```

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**.
   - **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.
7. **Turtles**
    - The turtle is initially in the center of the screen facing rightward.
    - *my_turtle.left(degrees)* – rotates the *my_turtle degrees* left (from its perspective).
    - *my_turtle.fd(distance)* – moves the *my_turtle distance* units forward.
    - *my_turtle.pu()* – picks the pen up
    - *my_turtle.pd()* – puts the pen down (ready to write)
    - *my_turtle.circle(radius)* – creates a circle with radius *radius*. The circle will be above the direction the turtle was facing when it started drawing. The turtle will move left and up in a circle and end up in the same place as before.