There are three sections. Sections 1 is worth 20 points. Section 2 is worth 40 points. Section 3 is worth 50 points. Partial credit is possible for each question. The maximum score for the test is 110 points.

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

**Section 1**: Below will be one function definition containing errors. On the appropriate line numbers in the table below the function, please list the errors (if any), along with possible corrections. The page preceding the function will be the output that the corrected function produces in a Python command window (e.g., Idle’s Python Shell).

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.

The comments are meant to be instructive. Please do not look in the comments for errors. Assume that the comments are error-free with one exception: if they are incorrectly marked, e.g., if *Note:* is used at the beginning of a comment line instead of #.

**Sample 1**: The function minus takes two numbers as input and subtracts the second from the first. **There is only one error.**

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>def minus(total,decrement)</td>
</tr>
<tr>
<td>2.</td>
<td>output = total - decrement</td>
</tr>
<tr>
<td>3.</td>
<td>return(output)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line</th>
<th>Correction of Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>need to add colon (:) at end of line</td>
</tr>
<tr>
<td>2.</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td></td>
</tr>
</tbody>
</table>
**Question 1**: The corrected version of the function on the next page draws the 3 kinds of faces (as shown below). Please find at least 4 errors in the version on the next page.

```python
>>> draw_face('frown')

```

```plaintext

This is the corrected version.

```python
>>> draw_face('smile')

```

```plaintext

Please check for errors.

```python
>>> draw_face('blank')

```

```plaintext

This is the corrected version.

```
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>def draw_face(expression):</td>
</tr>
<tr>
<td>2.</td>
<td>for number in range(3):</td>
</tr>
<tr>
<td>3.</td>
<td>print(11*'%')</td>
</tr>
<tr>
<td>4.</td>
<td>for number in range(6):</td>
</tr>
<tr>
<td>5.</td>
<td>if (number==1):</td>
</tr>
<tr>
<td>6.</td>
<td>print((number*'%')+(2*'%')+'O'+(3*'%')+'O'+(2*'%')+number*'%')</td>
</tr>
<tr>
<td>7.</td>
<td>elif (number==2) and (expression == 'frown'):</td>
</tr>
<tr>
<td>8.</td>
<td>print(((number*'%')+(3*'%')+'\ '+(3*'%')+number*'%'))</td>
</tr>
<tr>
<td>9.</td>
<td>else (number==3):</td>
</tr>
<tr>
<td>10.</td>
<td>if expression == smile:</td>
</tr>
<tr>
<td>11.</td>
<td>print((number*'%')+(1*'%')+'\ /'+(1*'%')+number*'%')</td>
</tr>
<tr>
<td>12.</td>
<td>elif expression == frown:</td>
</tr>
<tr>
<td>13.</td>
<td>print((number*'%')+(1*'%')+'\ /\ '+(1*'%')+number*'%')</td>
</tr>
<tr>
<td>14.</td>
<td>else:</td>
</tr>
<tr>
<td>15.</td>
<td>print((number*'%')+(1*'%')+'\ '+(1*'%')+number*'%'))</td>
</tr>
<tr>
<td>16.</td>
<td>else:</td>
</tr>
<tr>
<td>17.</td>
<td>print((number*'%')+(11 - 2<em>number)</em>'%')+number*'%')</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line</th>
<th>Correction of Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td></td>
</tr>
<tr>
<td>16.</td>
<td></td>
</tr>
<tr>
<td>17.</td>
<td></td>
</tr>
</tbody>
</table>
Section 2: You will be asked to determine what a program does given a particular input and given particular outcomes for random number functions.

- You will be asked what print statements do, taking into account where in the program they occur. Thus if a `print` statement occurs within a loop, you will be expected to predict how many times the print statement will print (and what it will print each time). Similarly, if a `print` statement is in the body of an `if`, `else` or `elif` statement, you will be expected to determine if the `print` statement will do anything at all.

- You will be expected to determine what the function returns (if anything).

**Question 2:** Approximately draw the shapes produced by the command `draw_village()`, as defined below. Assuming that the turtle starts in the middle of the screen facing rightward, indicate the order in which the turtle draws the houses. You can either number the houses or draw arrows to indicate the order.

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

def draw_house():
    my_turtle.pd()
    my_turtle.fd(20)
    my_turtle.left(90)
    my_turtle.fd(20)
    my_turtle.left(30)
    my_turtle.fd(20)
    my_turtle.left(120)
    my_turtle.fd(20)
    my_turtle.left(30)
    my_turtle.fd(20)
    my_turtle.pu()

def draw_village():
    for row in range(4):
        for column in range (5):
            draw_house()
            my_turtle.left(90)
            my_turtle.fd(50)
            my_turtle.left(90)
            my_turtle.fd(40)
            my_turtle.left(90)
            my_turtle.fd(250)
            my_turtle.left(180)
```
Question 3: Ubbi Dubbi is a children’s language code where the string “ub” is added before the first vowel of each syllable. So the word professor becomes pruboffubessubor. This program turns normal sentences into Ubbi Dubbi sentences.

What will the command generate_ubbi_dubbi(’A short sentence’) do? What will be printed to the screen? What string will be returned by the function?

For extra credit, describe the minor error in what the program returns for the example ’A short sentence’.

def is_vowel (character):
    return(character in "aeiouAEIOU")

def is_alphabetic(character):
    code = ord(character)
    if ((97 <= code <= 122) or (65 <= code <= 90)):
        return(True)
    else:
        return(False)

def is_consonant(character):
    if is_alphabetic(character) and not(is_vowel(character)):
        return(True)
    else:
        return(False)

def generate_ubbi_dubbi(string):
    ## Ubbi Dubbi is a kid’s language code in which "ub" is added before
    ## the first vowel
    ## in every syllable.
    output = ''
    found_vowel = False
    last = 'A'
    for character in string:
        if found_vowel:
            if (not is_vowel(character)):
                found_vowel = False
            elif (is_vowel(character) or ((character in 'yY') and (is_consonant(last)))):
                output=output+'ub'
                found_vowel=True
                output=output+character
        print('Output so far: ',output,'character: ', character, 'Last: ',last)
        last = character
    return(output)
Section 3: Write Functions as specified.

Question 4: Write a function called `count_words` that takes a single argument called `sentence`, a string. The function should count the number of words in `sentence` and return this number as output.

For this function, a “word” is defined as any number of letters or digits in a row. For example, the command:

```python
count_words('I want 13 half-chicken/half-turkey sandwiches.')
```

should return 8 because we assume that the words in the sentence are: I, want, 13, half, chicken, half, turkey, sandwiches. Thus numbers count as words and any character (space, hyphen, asterisk, etc.) that is neither a digit, nor a letter is treated as something that separates words.

At your option, you can use the following functions in your code (this is to save time).

```python
def is_digit_character(character):
    return(48 <= ord(character) <= 57)

def is_lowercase_character (character):
    return(97 <= ord(character) <= 122)

def is_uppercase_character (character):
    return (65 <= ord(character) <= 90)

def is_number_or_letter(character):
    if is_digit_character(character) \
       or is_lowercase_character(character) \ 
       or is_uppercase_character(character):
        return(True)
    else:
        return(False)
```
**Question 5:** Write a function called *negotiator* that will negotiate a price on your behalf. Assume you are buying some item and the program is negotiating with the seller. (Optionally you can add the name of the item as an additional parameter and use it in some of your print statements. However, this is not necessary for the test.)

Given an initial low price and an initial high price, the negotiator will use your low price as its first bid and then negotiate with the seller to purchase the item at your high price or lower. The negotiator program should implement the algorithm in the figure on the next page.

You should use the functions listed below in your code as necessary. In order to simplify the problem for a testing environment, *set_offer* and *get_most_reasonable_offer* are specifically mentioned in the flowchart.

The function should be called with your initial high and low prices as parameters (arguments). Note that the function *get_most_reasonable_offer* uses the *input* function for getting input from the seller. You can either get all your bids from the seller using this function, or use the *input* function in your own way to get bids from the seller.

```python
def yes_or_no(input_string):
    output = ''
    output_string = ''
    while (output == ''):
        print ('Please answer yes or no to the following question.')
        output_string = input(input_string)
        if (output_string == 'yes' or output_string == 'Yes' \
            or output_string == 'YES'):
            output = True
        elif (output_string == 'no' or output_string == 'No' \
            or output_string == 'NO'):
            output = False
        else:
            print ('I don\'t understand your answer.')
    return(output)

def set_offer (high, low):
    ratio = high/low
    if ratio >= 3:
        return(2*low)
    elif ratio >= 2:
        return(1.25*low)
    elif ratio <= 1.1:
        return(high)
    else:
        return ((high + low)/2)

def get_most_reasonable_offer(high):
    offer = float(input('How much do you want for it?'))
    rejected=False
    question = 'I cannot accept your current offer. Do you still want to do business?: '
    new_offer = 99999999
    while(not rejected) and (offer > high) and yes_or_no(question):
        new_offer = float(input('How much do you want for it?'))
        if (new_offer > offer):
            rejected = True
        else:
            offer = new_offer
    return(offer)
```
Obtain an offer from the seller using get_most_reasonable_offer(current_high)

Is offer accepted?

Yes
Print 'We have a deal!' and return True

No

Is the price less than or equal to current_low?

Yes

Set current_high to seller’s price.
Set current_low to output of set_offer(current_high, current_low)

No

Print 'I am sorry, but I cannot afford to make this purchase.' and return False

Does the seller want to sell the item?

Yes

Offer to purchase for current low.

No

End

Set current_high and current low price at Buyer’s discretion.

Offer to purchase for current low.

Yes

Obtain an offer from the seller using get_most_reasonable_offer(current_high)

No

Is the price less than or equal to current_high?

Yes

Print 'I hope we can do business in the future' and return False

No

Figure 1: The Negotiator Algorithm