Instructions:

KEEP TEST BOOKLET CLOSED UNTIL YOU ARE INSTRUCTED TO BEGIN.

No calculators, notes, textbooks, or any other aids are allowed except a pen or pencil (or other writing utensils). If you need extra scratch paper, please pickup a blue book from the front of the class.

All answers must be submitted on this exam sheet. All answers must be legible.
Example True or False:
Instructions: Circle either True or False

1. Interpreted and/or Byte-Code languages add a level of abstraction from your traditional compiled languages that makes interpreted/byte-code languages more portable between platforms, but also causes a slight performance hit.

   True  False

[There are usually 5-10 of these types of questions]

Example Multiple Choice:
Instructions: Circle the letter of the best answer.

1. Given a string variable named 'foo' that contains a string “Where in the world is Carmen Sandiego?”, what would foo[:-1] return?
   A. “Where in the world is Carmen Sandiego”
   B. “Where in the world is Carmen”
   C. “Sandiego?”
   D. “?”
   E. “W”

[There are usually 5-10 of these types of questions]

Example Short Answer:
Instructions: In your own words, answer the questions as best as possible.

1. What are the main differences between a List and a Dict in Python.

[There are usually 5-10 of these types of questions]
**Example Entomology - Study of Bugs:**

Instructions: Find 3 bugs in the following program (there are more than three):
1) Specify the line number of the bug.
2) Check the box next to the type of error; either "Logic" for logic errors or "Other" for (syntax, type, etc.).
3) Write a short explanation or fix for the error.

Note: There can be more than one bug per line.

<table>
<thead>
<tr>
<th>Bug</th>
<th>Line #</th>
<th>Type of Error</th>
<th>Explanation/Fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[ ]</td>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[ ]</td>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>[ ]</td>
<td>Logic</td>
<td></td>
</tr>
</tbody>
</table>

```python
def median(numbers):
    ''' returns the median of the list of numbers passed in '''
    numbers.sort()
    if len(numbers)%2 == 1:  # if there are two medians
        median1 = numbers[int(len(numbers)/2) -1]  # the lower value
        median2 = numbers[int(len(numbers)/2) +1]  # the upper value
        median = (median1 + median2)/2  # get the average of them
    else:
        median_pos = len(numbers)//2  # round down for index position
        median = numbers[median_pos]
    return

def mode(numbers):
    ''' returns the mode of the list of numbers passed in '''
    numbers.sort()
    cur_mode = -1       # the mode for numbers
    cur_mode_freq = -1  # number of occurrences of the mode
    for n in numbers:
        cnt = numbers.count(n)  # number of occurrences
        if cnt > cur_mode_freq:
            cur_mode = n           # update most frequent number
            cur_mode_freq = n      # number of occurrences
    return n

def range(numbers):
    ''' returns the size of the range of the list of numbers '''
    numbers.reverse()
    return numbers[len(numbers)] - numbers[0]
```
Example What is the Output:

Instructions: What is the output if the following code is run?

```python
animals = {'cat': 'pad', 'dog': 'anna', 'pig': 'wilbur'}
print(animals.keys())

zoo = animals.copy()
zoo.pop('cat')
zoo['snake'] = 'balboa'
print(zoo.values())
print(animals.values())

mypets = ['dog', 'fish']
mypets.append('cat')
mypets.remove('fish')
mypets.insert(1, 'cat')
print('Cats =', mypets.count('cat'))

mypets = ['dog', 'cat', 'pig']
print(mypets)

yourpets = mypets
mypsots.pop()
print(yourpets)

pets = ['dog', 'cat', 'pig', 'cat']
pets = set(pets)
pets = list(pets)
print('Cats =', pets.count('cat'))

myletters = {'a', 'b', 'c', 'b', 'a', 'd'}
yourletters = {'a', 'b', 'c'}
yourletters.remove('b')
letters = myletters.intersection(yourletters)
print(len(letters))
```

Answer:
Example Explain the Code:

Instructions: The following code was written, but the documentation was left out. Analyze the code to determine what each function does and then what the program as a whole does. Fill in the missing comments to explain what this program does as a whole as well as what each function is does. Be as specific as possible.

```python
import turtle
import random
import math

times=10000     # how many things to draw
speed=10        # speed of drawing
height=600      # window height
width=600       # window width
min_length=100  # min size of item to draw

def draw(n=1000):
    '''
    for i in range(n)
        draw_something()

def draw_something():
    '''
        turtle.penup()
        # ______________________________________________________________________
        xpos = random.randint(-width/2, width/2)
ypos = random.randint(-height/2, height/2)
turtle.setpos(xpos, ypos)
        # ______________________________________________________________________
        heading = random.randint(1,360)
turtle.left(heading)
        # ______________________________________________________________________
        length = random.randint(min_length, math.sqrt(width*height))
turtle.pendown()
turtle.forward(length)
turtle.penup()

def main():
    turtle.setup(height, width)
turtle.speed(speed)
draw(times)

main()
```
import random

deck = [ 'A of Spades', 'A of Clubs', 'A of Diamonds', 'A of Hearts',
    'K of Spades', 'K of Clubs', 'K of Diamonds', 'K of Hearts' ]

def dealCards(count=______________________):
    '''
    returns a list comprised of 'count' number of string representations
    of cards from the deck (defaults to 2 cards), or returns None if
    there aren't enough cards left to deal 'count' number of cards.
    '''

    # return None if there isn't enough cards in the deck
    if count > ______________________:
        return None

    # Start with an empty list of cards to return
    dealingCards = ______________________

    # For each card needed to deal
    for i in range(______________________):
        # get a valid index into the deck
        index = random.randint(______________________)

        # remove that card from the deck
        drawnCard = deck.pop(______________________)

        # add it to the cards to return
        dealingCards.______________________(drawnCard)

    # return list of cards removed from the deck
    return ______________________

def main():
    # deal 3 cards from the deck
    print("Delt the following cards:", ______________________)

    # run the program
    ______________________()
Example Write the Code:

Instructions: In python code, write a program that will ask the user to enter as many numbers as he/she wants. When the user is finished they'll enter an empty string (they just hit return). At this point the program should tell the user:
* the median of all the numbers entered (if there is no single median, take the average of the two medians)
* the mode of all the numbers entered (if there is no single mode, simply choose one of the modes)

Example input/output:

   Enter a name (or 'return' to finish): 50
   Enter a name (or 'return' to finish): 50
   Enter a name (or 'return' to finish): 60
   Enter a name (or 'return' to finish): 80
   Enter a name (or 'return' to finish): 100
   Enter a name (or 'return' to finish):
      The median is: 60
      The mode is: 50

[ There are usually 2 – 3 of this type of problem ]