This is a self-scheduled exam to be completed during one of the final exam periods.

Please write all your work on these pages (front and back is okay, but do not use a blue book or any other pages).

The exam is closed notes, closed Internet, and closed technology, but you may use two “cheat sheets”.

Your cheat sheets must be hand-written, created by you, 8.5” × 11”, and can be double-sided (up to 4 sides total).

Submit both cheat sheets with your exam.

Do not discuss the exam with other students and respect the honor code of doing your own work.

If you are unable to make progress on any part of the exam, tell me what you tried; describe your thought process.

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

| Part 1 | /25 |
| Part 2 | /10 |
| Part 3 | /25 |
| Part 4 | /15 |
| Part 5 | /25 |
| Total  | /100 |
Part 1: Vocabulary and Short Answer

(a) Match each line of code on the left with the most appropriate description on the right (choose one description for each line).

```
dot = Circle(Point(x,y), radius)  # defining a method
p = dot.getCenter()               # calling/invoking a method

c = color_rgb(0, 255, 0)          # defining a function

def move(self, width):            # defining an instance variable
    ...

def shuffle_sort(lst):            # using a constructor
    ...

self.parts_lst = [tail, body, eye]    # calling/invoking a function
```

(b) For each assigned variable below, fill in the type in the blank on the right (note: we should never use variable names like this, variable names should help convey the type).

```
a = (5 > 10)                      # type of a: ____________
b = math.pi                       # type of b: ____________
c = "summer"                      # type of c: ____________
d = {"A": 5, "B": 3}             # type of d: ____________
e = "smith college".split()      # type of e: ____________
f = Fish(x,y,color)               # type of f: ____________
g = 42                             # type of g: ____________
h = dot.getCenter()               # type of h: ____________
```

(c) For each of the types in (b) above, put an “M” next to your answer if the type is mutable.
(d) The following code shows an attempt to count the number of lines of a file that contain the word "spam".

```python
words = open("my_file.txt","r")
x = 0
for text in words:
    if text.count("spam") > 0:
        x += 1
words.close()
```

What is wrong with this code? Explain below. To the right of the code, rewrite this code to fix the error and modify the variable names so the code is more clear.

(e) **Mystery while loops**: analyze the function below and answer the following questions.

```python
def mystery_function(item, sorted_lst):
    i = 0
    while item > sorted_lst[i]:
        i += 1
    return sorted_lst[i]
```

i. Explain what this function is doing in words, being specific about the input (parameters) and output (return value).

ii. Say we test this function with the following list of strings:

```
sorted_lst = ['bam','clam','dam','ham','jam','lamb','ram','scam','spam','tram','yam']
```

For each test case below, write what i is equal to at the end of the function call. Then write what is returned.

- item = "am", i = ________, return value: ________
- item = "slam" i = ________, return value: ________
- item = "zam" i = ________, return value: ________

iii. How does the last test case expose an error in the code? Edit the code above to fix the problem.
Part 2: Dictionaries

Looking back on the dice roller example, we could have instead using a dictionary to keep track of the pips that should be “on” when each value is rolled. For example, in the image shown below, we rolled a 2 and a 5. For the value 2, the 0\textsuperscript{th} and 6\textsuperscript{th} pips should be “on”.

![Dice Roller Image]

Here is an example of how we could define such a dictionary (perhaps as a global variable).

\[
pips\_dict = \{1: [3], 2: [0, 6], 3: [0, 3, 6], 4: [0, 2, 4, 6], \\
5: [0, 2, 3, 4, 6], 6: [0, 1, 2, 4, 5, 6]\%
\]

(a) Modify the \texttt{set\_value}(...) method to use this dictionary instead of a bunch of cases. By adding one line to the code below, define the \texttt{pips\_on} variable using the \texttt{pips\_dict}.

```python
    def set_value(self, value):
        """Change which pips are the foreground color to simulate
        rolling a different number (value can be 1,2,3,4,5,6)."
        
        # first set ALL the pips back to the background color
        for pip in self.pip_lst:
            pip.setFill(self.background)

        # TODO: based on the value, define a list of pips to be "on"
        # for each of the "on" pips, change its color to the foreground
        for i in pips_on:
            self.pip_lst[i].setFill(self.foreground)
```

(b) Say now we wanted to be able to roll a 7, which will mean all seven pips are on. Write a line of code below to \textit{add} this pair to the \texttt{pips\_dict} dictionary (not redefine it from scratch).
Part 3: Writing Classes

The goal of this part is to write a `Customer` class representing customers at a bank. Each customer will be able to keep track of the amount of money they have (their `balance`). They should be able to deposit (add money to their account), as well as withdraw (take out money from their account).

```python
def main():
    kathy = Customer("Kathy McCartney", 500) # initial balance: $500
    print(kathy.get_balance())

    kathy.deposit(100)
    print(kathy.get_balance())

    kathy.withdraw(200)
    print(kathy.get_balance())

    kathy.withdraw(500)
    print(kathy.get_balance())

    print(kathy)

main()
```

(a) Analyze the code above which creates a `Customer` instance and performs various tests. Based on the goals of this class, what should be printed if I run `main`? Think about how to inform the user if they try to withdraw too much from their account.

(b) List the methods that need to be written inside the `Customer` class, including the constructor and a way for instances to be printed (*Hint: there should be 5 total.*
(c) Based on your analysis from (a) and (b), write the `Customer` class in the box below. Think about which instance variables to keep track of, and then write each method from (b). Note: there is more than one way to complete this class.

class Customer:
Part 4: Classes in Graphics

Analyze the code below. The goal is to write an animated graphics program where a ball (Circle) will bounce from the top of the window to the bottom of the window, then back to the top, then to the bottom, etc, indefinitely. It will continue to “bounce” up and down forever.

```python
from graphics import *

class BouncingBall:
    def __init__(self, x, y, speed):
        self.circle = Circle(Point(x,y), 40)
        self.y = y
        self.speed = speed

    def draw(self, window): # TODO
        pass

    def move(self, height): # TODO
        pass

def main(): # hint: draw a picture of the initial setup
    width = 600
    height = 600
    win = GraphWin("Bouncing Ball", width, height)

    ball = BouncingBall(width/2, 20, 5) # speed starts positive (moving down)
    ball.draw(win)
    while True:
        ball.move(height)
    main()
```

(a) How many instance variables does this class have? 
(b) Fill in the code for the draw method and the move method so that the ball will bounce up and down. It does not matter exactly when the ball starts going up or going down, just as long as it changes direction roughly at the top and bottom of the window.
Part 5: Recursion

The code below shows our recursive implementation of the Fibonacci numbers from class. Let the \( n \text{th} \) Fibonacci number be denoted \( F_n \). The first two Fibonacci numbers are \( F_0 = 1 \) and \( F_1 = 1 \). After that, the pattern is described recursively: \( F_n = F_{n-1} + F_{n-2} \).

```python
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

(a) What result is returned when I call `fib(4)`? i.e. what is \( F_4 \)?

(b) Say I call `fib(4)` in main. How many times is the `fib(\ldots)` function called/invoked, including the first call? Justify your answer.

(c) Using the same example of calling `fib(4)`, how many times is the base case reached? (i.e. how many times do we get to the line that says `return 1`?) Justify your answer.
(d) Going in a different direction, write a recursive function that will compute the sum of the first $n$ squares. For example, $S_0 = 0$, $S_1 = 1^2 = 1$, $S_2 = 1^2 + 2^2 = 5$, $S_3 = 1^2 + 2^2 + 3^2 = 14$, ...

To help complete this problem, first complete the table below. In the third column, use the symbol for the previous answer to develop a recursive pattern.

<table>
<thead>
<tr>
<th>symbol</th>
<th>write out the sum</th>
<th>use the previous answer</th>
<th>answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>$0^2$</td>
<td>base case</td>
<td>0</td>
</tr>
<tr>
<td>$S_1$</td>
<td>$0^2 + 1^2$</td>
<td>$S_0 + 1$</td>
<td>1</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$0^2 + 1^2 + 2^2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_4$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Now write a recursive function to accomplish this task. For example, when $n = 3$, this function should return 14.

```python
def squares_sum(n):
```

(e) Problems are typically not classified as recursive or iterative, but solutions are. Write a function that will accomplish the same goal as the recursive Fibonacci function, but iteratively (i.e. using a loop). Hint: think about how to keep track of the previous two Fibonacci numbers.

```python
def fib_loop(n):
```
More space (tell me which problem this is for), or draw me a picture.

Thank you for all your great work this semester – enjoy the summer break!