FINAL EXAM – May 2017

CSC 111 01/02: Introduction to Computer Science

Instructor: Sara Mathieson

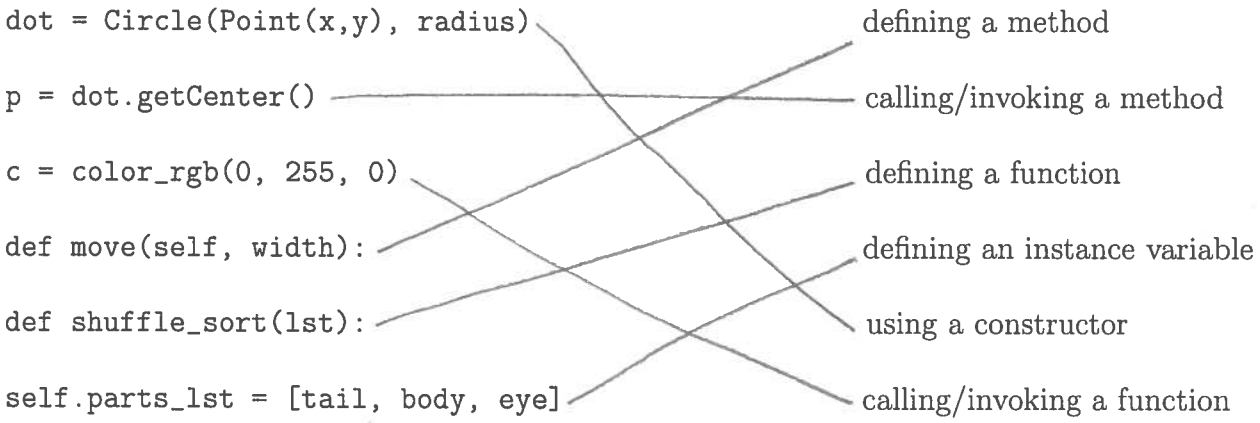
- This is a self-scheduled exam to be completed during one of the final exam periods.
- Please write all your work on these pages (front and back is okay, but do not use a blue book or any other pages).
- The exam is closed notes, closed Internet, and closed technology, but you may use two "cheat sheets".
- 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.

Name	Solutions (sketches)

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).



(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: bool
b = math.pi                     type of b: float
c = "summer"                    type of c: str
d = {"A": 5, "B": 3}            type of d: dict    M
e = "smith college".split()     type of e: list    M
f = Fish(x,y,color)             type of f: Fish    M
g = 42                          type of g: int
h = dot.getCenter()             type of h: Point   M
```

(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".

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

```
file = open("my_file.txt","r")
x = 0
for line in file:
    if line.count("spam") > 0:
        x += 1
file.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.

words is a file object, not a string, so we cannot call count(···) on words. Should be text.count("spam").

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

```
def mystery_function(item, sorted_lst):
    i = 0
    while item > sorted_lst[i]: and i < len(sorted_lst) - 1:
        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).

This function is performing linear search, to find the element in sorted_lst closest to item. It returns the closest element (could be off by one).

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

```
                0     1     2     3     4     5     6     7      8      9      10
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.

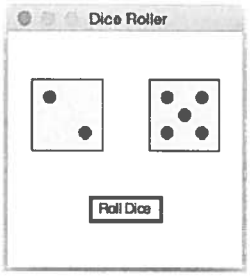
- `item = "am"`, i = 0, return value: "bam"
- `item = "slam"` i = 8, return value: "spam"
- `item = "zam"` i = 11, return value: error!

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

The last query produces an "index out of range error," since we are trying to access an element outside the list.

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^{th} and 6^{th} pips should be "on".



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 `set_value(..)` method to use this dictionary instead of a bunch of cases. By adding *one* line to the code below, define the `pips_on` variable using the `pips_dict`.

```
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"
        pips_on = pips_dict[value]

        # 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 *add* this pair to the `pips_dict` dictionary (not redefine it from scratch).

```
pips_dict[7] = [0, 1, 2, 3, 4, 5, 6]
```

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).

```
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.

500
600
400
Error! Cannot withdraw more than you have.
400
Kathy McCartney: $400 ← other variations okay

(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.*)

1. `__init__(self, name, balance)`
2. `get_balance(self)`
3. `deposit(self, amount)`
4. `withdraw(self, amount)`
5. `__str__(self)`

(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:

    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

    def get_balance(self):
        return self.balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount > self.balance:
            print("Error! Cannot withdraw more
                  than you have.")
        else:
            self.balance -= amount

    def __str__(self):
        return self.name + ": $" + str(self.balance)
```

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.

```
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
        self.circle.draw(window)


    def move(self, height): # TODO
        if self.y < 20 or self.y > height - 20:
            self.speed = -self.speed

        self.circle.move(0, self.speed)
        self.y += self.speed


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? 3

(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^{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: $F_n = F_{n-1} + F_{n-2}$.

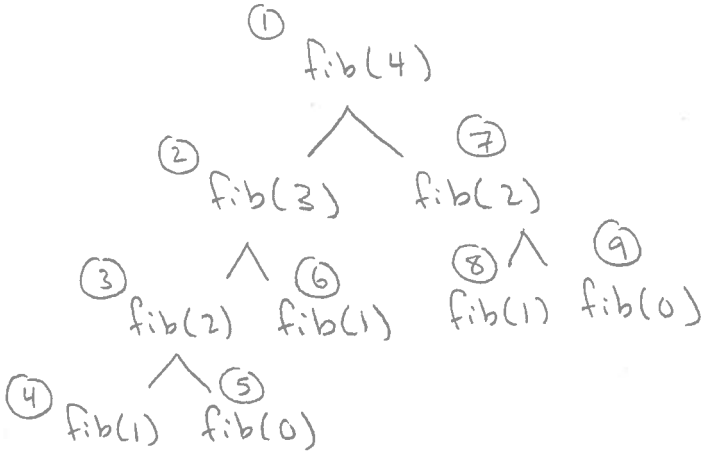
```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

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

n	0	1	2	3	4
F_n	1	1	2	3	5

$\Rightarrow$ $F_4 = 5$

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



9 times

(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.

When we call fib(1) or fib(0), we hit the base case. This happens 5 times as shown in the tree above.

(d) Going in a different direction, write a recursive function that will compute the sum of the first n squares. For example, $S_0 = 0$, $S_1 = 1^2 = 1$, $S_2 = 1 + 2^2 = 5$, $S_3 = 1 + 2^2 + 3^2 = 14$, ... 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.

symbol	write out the sum	use the previous answer	answer
S_0	0^2	base case	0
S_1	$0^2 + 1^2$	$S_0 + 1$	1
S_2	$0^2 + 1^2 + 2^2$	$S_1 + 2^2$	5
S_3	$0^2 + 1^2 + 2^2 + 3^2$	$S_2 + 3^2$	14
S_4	$0^2 + 1^2 + 2^2 + 3^2 + 4^2$	$S_3 + 4^2$	30

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

```
def squares_sum(n):
    if n == 0:
        return 0
    else:
        return squares_sum(n-1) + n*n
                                  (n^2)
```

(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.*

```
def fib_loop(n):
    s1 = 1
    s2 = 1
    for i in range(n-1):
        temp = s2
        s2 = s1 + s2
        s1 = temp
    return s2
```

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!