

15-112 Fall 2019 Midterm #1A
80 minutes

Name: _____

Andrew ID: _____@andrew.cmu.edu

Section: _____

- You may not use any books, notes, or electronic devices during this exam.
- You may not ask questions about the exam except for language clarifications.
- Show your work on the exam (not scratch paper) to receive credit.
- If you use scratch paper, you must submit it with your andrew id on it, and we will ignore it.
- All code samples run without crashing unless we state otherwise. Assume any imports are already included as required.
- Do not use these post-midterm1 topics: sets/dictionaries, recursion.
- You may use `almostEqual()` and `roundHalfUp()` without writing them. You must write everything else.

Do not write below here

Question	Points	Score
1. Free Response #1	25	
2. Free Response #2	20	
3. Free Response #3	20	
4. Short Answers	5	
5. Code Tracing #1	5	
6. Code Tracing #2	5	
7. Graphics Code Tracing	10	
8. Reasoning over code #1	5	
9. Reasoning over code #2	5	
10. Bonus	5 (bonus)	
TOTAL	100	

1. **Free Response: nthBalancedPrime** [25 pts]

Write the function `nthBalancedPrime(n)` that takes a non-negative integer `n` and returns the `n`th integer which is a balanced prime.

A number is a balanced prime if the following conditions are true:

1. The number itself is prime, and
2. It is equal to the average of the nearest prime below it and the nearest prime above it.

For example, `nthBalancedPrime(0)` should return 5 (the smallest balanced prime). This is because 5 is prime, and the nearest lower and upper primes are 3 and 7, whose average is 5. In other words, $5 == (3 + 7) / 2$.

The first few balanced primes are: 5, 53, 157, 173, and 211.

Note: you must write all the required helper functions here, including `isPrime`, but you do not need to write `fasterIsPrime`.

This page is blank (for your nthBalancedPrime solution, if needed).

2. Free Response: Book and Chapter Classes [20 pts]

Write the classes Book and Chapter so that the following test code passes. Do not hardcode the test cases, but you may assume that the parameters are always legal (so, for example, chapter indexes are always in bounds).

```
chapterA = Chapter('I love CS!', 30) # chapter title, # of pages
chapterB = Chapter('So do I!', 15)
book1 = Book('CS is Fun!', [chapterA, chapterB]) # book title, chapters
book2 = Book('The Short Book', [ Chapter('Quick Read!', 5) ])

assert(book1.chapterCount == 2)
assert(book1.getPageCount() == 45)
assert(book2.chapterCount == 1)
assert(book2.getPageCount() == 5)
assert(book1.getChapter(0).title == 'I love CS!')
assert(book1.getChapter(1).title == 'So do I!')
assert(book2.getChapter(0).title == 'Quick Read!')

# Move chapter 0 from book1 to the end of book2
# so moveChapter always moves to the end of the target book.
book1.moveChapter(0, book2)

assert(book1.chapterCount == 1)
assert(book1.getPageCount() == 15)
assert(book1.getChapter(0).title == 'So do I!')
assert(book2.chapterCount == 2)
assert(book2.getPageCount() == 35)
assert(book2.getChapter(0).title == 'Quick Read!')
assert(book2.getChapter(1).title == 'I love CS!')
```

This page is blank (for your Book and Chapter solutions, if needed).

3. **Free Response: modified wordSearchFromCellInDirection** [20 pts]

Write a modified version of `wordSearchFromCellInDirection`, the innermost of the three functions we wrote in the `wordSearch` case study. This function takes a board and a word, along with a starting location and a direction (specified using `drow` and `dcol`), and checks if the word is in the board starting from that location heading in that direction. Here, we will make two changes from the version in the notes:

1) Boolean return value

Your function should return `True` if the word is found and `False` if not.

2) Wildcards

Both the board and the word can contain '?', which is a so-called wildcard that matches any letter, so 'a?c' matches each of 'abc', 'acc', 'adc', and so on. However, a match can only use up to one wildcard, so 'a??' does not match 'abc', and 'a?c' does not match 'ab?', since each would use two wildcards.

Note: if you provide a correct solution except without wildcards, you will receive all but 5 points here.

we have provided the 'def' line for you:

```
def wordSearchFromCellInDirection(board, word, startRow, startCol, drow, dcol):
```

4. **Short Answers** [5 pts]

1. Fill in the bubble for ALL of the following expressions that demonstrate short-circuit evaluation:

- A) $(9 > 5)$ and $(5 > 0)$
- B) $(9 > 5)$ or $(5 > 0)$
- C) $(9 < 5)$ and $(5 < 0)$
- D) $(9 < 5)$ or $(5 < 0)$
- E) None of these

2. Fill in the True or False bubble for each of the following:

- True False Statements and expressions are two names for the same thing.
- True False When you write a new class in Python, you create a new type.
- True False Floating-point numbers in Python are sometimes exact, but other times only approximate, depending on the value.
- True False Technically Python does not need 'for' loops because any 'for' loop could be rewritten as a 'while' loop.
- True False To enforce top-down design, Python will not run any function that has more than 30 lines.

3. Given an arbitrary destructive function $f(L)$ where L is a 2d list, write the function $g(L)$ that works the same as $f(L)$, except that it is non-destructive and its return value is the standard return value for most non-destructive functions. Note that $g(L)$ may call $f(L)$.

4. In our TwentyOne / BlackJack case study, recall that the `getHandString(self, hand)` takes a hand, which is a list of Card instances, and returns a comma-separated string of all the cards in that hand (so it returns a string like 'AC, 2D, KS'). With that in mind, fill in the blank with the missing expression:

```
def getHandString(self, hand):
```

```
    return _____
```

Hint: you may wish to call the `getString()` method of each card, and you may assume that method is already written.

5. In our Connect4 example, recall that the `checkForWin(board, player)` function takes a board and a player and returns True if that player won (has 4 in a row), and False otherwise. Also, recall that we made use of our `wordSearch` solution somehow here. With that in mind, write the (very short) function `checkForWin`. You may assume all its helper functions are already written.

```
def checkForWin(board, player):  
    # Hint: This is just a couple lines of code!
```


5. **Code Tracing** [5 pts]

Indicate what this prints. Place your answer (and nothing else) in the box.

```
def ct1(x):
    j = c = 0
    for i in range(x):
        while j < 2*i:
            j += i
            c += 1
            if (i**i == j): continue
            print(i, j)
    return c

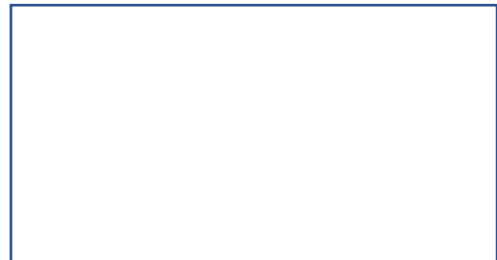
print(ct1(5))
```

6. **Code Tracing** [5 pts]

Indicate what this prints. Place your answers (and nothing else) in the box.

```
import copy
def ct2(L):
    a = L
    b = copy.copy(L)
    c = copy.deepcopy(L)
    a[1][0] += 2
    b[0] = a[1] * 2
    a[0][0] += a.pop()[0]
    b[1] = c[0]
    return b

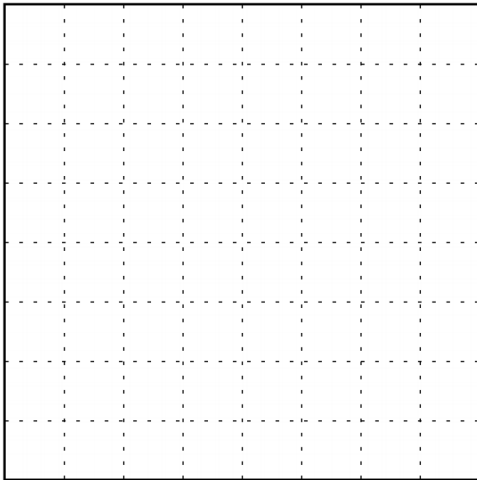
L = [[1],[2, 3]]
print(ct2(L))
print(L)
```



7. Graphics Code Tracing [10 pts]

Sketch what this will draw when run on a 400x400 canvas:

```
def draw(canvas, width, height):
    (x0,y0) = (50,50)
    h = 200
    count = 0
    dx = dy = 1
    for i in range(2):
        x1 = x0 + dx * h
        y1 = y0 + dy * h
        canvas.create_text(x1, y1, text=str(count))
        canvas.create_rectangle(x0, y0, x1, y1)
        h //= 2
        count += 1
        (x0,y0) = (x1,y1)
        dy = -dy
```



*this canvas is 400x400, boxes are 50x50

8. Reasoning Over Code [5 pts]

Find an argument for the following function that makes it return True. Place your answer (and nothing else) in the box beside the code:

```
def rc1(s):  
    n, count = 0, 1  
    for i in range(1, len(s)):  
        if (s[i] == s[i-1]):  
            count += 1  
        else:  
            n = 10*n + count  
            count = 1  
    return ((n == 213) and (s[0] == s[-1]))
```

9. Reasoning Over Code [5 pts]

Find an argument for the following function that makes it return True. Place your answer (and nothing else) in the box beside the code:

```
def rc2(L):  
    r = []  
    for M in L:  
        if len(M) > 2:  
            return False  
        for j in range(M[0]):  
            r.append(M[-1] + j)  
        r.append(str(len(r)))  
    return r == [5, 6, '2', 42, 43, 44, '6']
```

10. **Bonus** [5 pts]

Indicate what these print. Place your answers (and nothing else) in the box next to each block of code.

```
def f(x): return x+1
def g(x): return 10*f(10*x)
def h(t):
    global s
    s = 'g(0)'
    while (eval(s) < 10**t): s = f'g({s})'
    return (chr(ord('A') + str(eval(s)).count('1'))))
def bonusCt1():
    print(''.join([h(x) for x in [42, 27, 42]]))
bonusCt1()
```

```
def bonusCt2(x):
    k=P=1
    while k<=x:P%k and print(k, end='4');P*=k*k;k+=1
print(bonusCt2(12.3))
```