15-112 F25 Practice Midterm2
Time: 80 Minutes

Notes:

- This practice exam is lightly adapted from S23-L3 Midterm1.
- Due to editing issues, there is no FR2 on this practice exam.
- Submit your completed version in person or online by lecture on 10/7.

This is the Practice Midterm2, part of Prep 7. In order to receive credit, you must simulate taking this as if it was an actual exam, and you must write your name on this paper and hand this back in during lecture on 10/7, or submit it online here. This Practice Midterm is graded based on completion and effort rather than correctness, but if we do not receive it immediately, or if you have not demonstrated apparent effort, you will receive a zero on the assessment.

Before and during the practice midterm, you may not view any other notes, prior work, websites or resources, including any form of AI. You may not communicate with anyone else except for current 112 TAs or faculty during the assessment. All syllabus policies apply.

Do not use sets, dictionaries, recursion, or anything else disallowed in units 1-4.

Do not open this or look inside (even briefly) before you are ready to begin. Do not spend more than 80 minutes on this assessment.

CT1: Code Tracing [7pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below.

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

print(ct1(5))
```

CT2: Code Tracing [7pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def ct2(s):
    f = ''
    u = f
    n = 0
    s = s[1:]
    for c in s:
        if c.isspace():
            n += 1
        elif c.isupper():
            f += c.lower() * n
        elif c.isalpha():
            u += c
    return f'{f}-{u}-{n}'

print(ct2('xyZ? A b c'))
```

CT3: Code Tracing [8pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
import copy
def ct3(L):
    A = copy.deepcopy(L)
    B = [A[0], L[1]]
    A[0].pop()
    B[1][0] = 3
    A = A[1] + A[0]
    B[1] = B[0][0] + B[0][1]
    print(f'A:{A}')
    print(f'B:{B}')

C = [[6, 4, 7], [9, 8]]
ct3(C)
print(f'C:{C}')
```

Fill in the Blank (FitB)

FitB1: encode [18 pts]
You are given the function encode(s) that solves the following problem, only with some parts removed. You need to fill in the blanks with the missing code.

Note: when you fill in blanks, you must not add any newlines and you must not use any semicolons.

Also note: there is an excellent chance that midterm2 will have one or more FitB problems on it.

With that, here is the writeup for encode:

Backgound: One way to encode a string s is to add extra letters. Here, we first add an 'A', then a 'B', and so on, before each letter in the original string. Thus, if the string is 'cat', our encoded string would be 'AcBaCt'. We do not add anything before non-letters, so we encode 'm333N4' as 'Am333BN4'. Also, we will only add the letters from 'A' to 'D'. If the original string is long enough, then after we add 'D', we wrap around so that the next letter we add is 'A' again. Thus, we encode 'qrstuv' as 'AqBrCsDtAuBv'.

With this in mind, the function encode(s) takes a string s and returns the encoded version of that string as just described. Here are some test cases:

```
def testEncode():
    assert(encode('cat') == 'AcBaCt')
    assert(encode('m333N4') == 'Am333BN4')
    assert(encode('qrstuv') == 'AqBrCsDtAuBv')
    assert(encode('x?') == 'Ax?')
    assert(encode('15-112') == ('15-112'))
    assert(encode('') == '')
```

Here is the FitB solution (fill in the missing blanks):

```
def encode(s):
    result = ''
    i = 0
    extras = 'ABCD'
    for c in s:
        if c._____________________________________():  # BLANK #1

            result += extras[i]

            i = (_________________________) % len(extras) # BLANK #2
        result += c
    return result
```

Free Response 1: isSummish + nthSummish [20pts]

Note: To receive credit on this problem, **you may not use strings, lists, or tuples.**

We will say that a positive integer x is "summish" (a coined term) if its largest digit occurs only once, and the ones digit of the sum of all the **other digits** in x equals the largest digit in x.
For example, consider x = 15765:

- Its largest digit is 7
- 7 occurs only once in x
- The sum of the other digits is 1+5+6+5 = 17
- The ones digit of the sum of the other digits is 7, which equals the largest digit in x

Thus, 15765 is summish.

Note: the first 10 summish numbers are: 112, 121, 123, 132, 134, 143, 145, 154, 156, 165

With that in mind, write the function **nthSummish(n)** that takes a non-negative integer n and returns the nth summish number, where the 0th summish number is 112.

You must **also** write the function **isSummish(x)** that takes a positive integer x and returns True if it is summish, and False otherwise.

Remember, **do not use strings, lists, or tuples.**

Begin your FR1 answer here or on the following page

You may begin or continue your FR1 answer here

You may continue your FR1 answer here

Free Response 3: makeBorderish [20pts]

Background: given a 2d list of integers L, we will say that a value is on the border of L if it is in the top or bottom row or in the left or right column. Also, values that are not on the border are on the interior of L.

We will also say that a rectangular 2d list of integers L is "borderish" (a coined term) if the sum of the values on the border equals the sum of the values on the interior.

Finally, we will say that L is "almost borderish" if it is not borderish but it can be made borderish by swapping two columns in L, and that there is **only one such column swap** that makes L borderish.

With that in mind, write the function makeBorderish(L) that takes a rectangular 2d list of integers L that is **almost** borderish, and mutatingly swaps two columns in L so that it is borderish.

For example, this list is almost borderish:

```
L = [ [ 1, 0, 2, 1 ],
      [ 2, 5, 0, 4 ],
      [ 1, 1, 1, 0 ]
    ]
```

First, we see that L is not borderish because:

- The sum of the border values is 1+0+2+1+2+4+1+1+1+0 == 13
- The sum of the interior values is 5+0 == 5

By swapping col 2 with col 3 we get this borderish list:

```
M = [ [ 1, 0, 1, 2 ],
      [ 2, 5, 4, 0 ],
      [ 1, 1, 0, 1 ]
    ]
```

We see that the result is borderish because:

- The sum of the border values is 1+0+1+2+2+0+1+1+0+1 == 9
- The sum of the interior values is 5+4 == 9

(Continued on next page)

Thus we get this test case:

```
L = [ [ 1, 0, 2, 1 ],
      [ 2, 5, 0, 4 ],
      [ 1, 1, 1, 0 ]
    ]
M = [ [ 1, 0, 1, 2 ],
      [ 2, 5, 4, 0 ],
      [ 1, 1, 0, 1 ]
    ]
assert(makeBorderish(L) == None) #This is a mutating function
assert(L == M)
```

Notes/hints:

- You are guaranteed that L will be almost borderish and will have at least 3 rows and 3 cols.
- One approach might involve mutatingly swapping each pair of columns, checking if the result is borderish, and undoing the swap if not.
- You may wish to write the helper functions isBorderish(L) and swapCols(L, col0, col1)

Begin your FR3 answer here or on the following page

You may begin or continue your FR3 answer here

You may continue your FR3 answer here

Free Response 4: Moon Animation [20pts]

Write an animation with the following features.

- When the app starts:
  - A large blue dot with radius 100 is drawn in the center of the canvas)
  - The number 0 is drawn in the center of the blue dot
  - A smaller gray dot with radius 25 is drawn with its left edge touching the left edge of the canvas, and its top edge touching the top edge of the canvas
- Clicking inside the gray dot with the mouse changes its color to red. When the dot is red, it is selected. If it is clicked again, the red dot returns to gray and is no longer selected.
- Every time the small dot becomes selected and turns red, the number inside the large blue dot increases by 1.
- If the mouse is moved while the small dot is selected, the small dot updates its position to be centered on the mouse cursor, **unless** the new position would cause it to intersect with the larger blue dot.
  - In the case described above where the new position for the small dot would intersect with the larger dot, the small dot does not move, and instead becomes unselected and becomes gray. The small dot must be clicked again in order to become red and selected.
- Pressing the 'r' key resets the position of the small dot, it becomes gray, and is no longer selected. The count inside the large dot does NOT reset, however.

Notes:

- You **may** assume a 400x400 canvas.
- You will be penalized if your code results in an MVC violation
- Make reasonable choices for anything not specified above (like font size and color, etc).
- Please write your code using the provided function headers for onAppStart, redrawAll, onMouseMove, onMousePress, and onKeyPress. You may use the provided distance function without rewriting it. As usual, you may write and use additional helper functions if you wish. (You may also add other event functions, but this is not recommended or necessary.)

Begin your FR4 answer on the following page

Begin your FR4 answer here:

```
from cmu_graphics import *

def distance(x1, y1, x2, y2):
    return ((x1-x2)**2 + (y1-y2)**2)**0.5

def onAppStart(app):









def redrawAll(app):
```

Continue your FR4 answer here

```
def onMousePress(app, mouseX, mouseY):
```

Continue your FR4 answer here

```
def onMouseMove(app, mouseX, mouseY):

def onKeyPress(app, key):

runApp()
```

bonusCT1: Code Tracing [1pt]

This question is optional, and intentionally quite difficult. Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def bonusCT1(k):
    return [i for i in range(k**2) if sum([i%j==0 for j in range(2,i)])==0][k]

print(bonusCT1(11))
```

bonusCT2: Code Tracing [1pt]

This question is optional, and intentionally quite difficult. Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def f(x):
    return x+1
def g(x):
    return 10*f(10*x)
def h(t):
    s = 'g(0)'
    while (eval(s) < 10**t):
        s = f'g({s})'
    return (chr(ord('A') + str(eval(s)).count('1')))
def bonusCt2():
    print(''.join([h(x) for x in [42, 27, 42]]))
bonusCt2()
```

bonusCT3: Code Tracing [1pt]

This question is optional, and intentionally quite difficult. Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
import math
def bonusCT3():
    a = [int(math.log(x,3)) for x in range(1,3)]
    for k in range(1234):
        a = [a[0] or k, a[1] and k]
    return int("".join([str(x)*x**y for x in a for y in a]))
print(bonusCT3())
```