# C200 Programming Assignment № 6

**Dr. M.M. Dalkilic**
Computer Science
School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

October 21, 2022

## Introduction

**Please start early on this HW.**
**Due Date: 10:59 PM, Friday, October 28, 2022**
Submit your work **using** the new autograder before the deadline and remember to add, commit and push to Github. Autograder Link: `http://c200.luddy.indiana.edu`.

For this HW, we are not giving explicit unit testing file, but print statements are given in the starter code to do initial testing. You are also encouraged to write some of your own test cases.

You can choose your programming partner (up-to one programming partner), but write their name in the a6.py file. We have created a comment for that at the top of the file.

**Note:** Please follow the instructions for Problem-10 carefully (both in the PDF and the starter code) to ensure that you don't face problem while submitting to Autograder and can run it successfully.

## Problem 1

Implement the recursive functions.

$$s(0) = 0 \quad (n = 0) \tag{1}$$
$$s(n) = s(n-1) + n \quad (n = 1, 2, 3, \ldots) \tag{2}$$

$$p(0) = 10000 \quad (n = 0) \tag{3}$$
$$p(n) = p(n-1) + 0.02p(n-1) \quad (n = 1, 2, 3, \ldots) \tag{4}$$

$$b(1) = 2 \quad (n = 1) \tag{5}$$
$$b(2) = 3 \quad (n = 2) \tag{6}$$
$$b(n) = b(n-1) + b(n-2) \quad (n = 3, 4, 5, \ldots) \tag{7}$$

$$c(1) = 9 \quad (n = 1) \tag{8}$$
$$c(n) = 9c(n-1) + 10^{n-1} - c(n-1) \quad (n = 2, 3, 4, 5, \ldots) \tag{9}$$

$$d(0) = 1 \quad (n = 0) \tag{10}$$
$$d(n) = 3d(n-1) + 1 \quad (n = 1, 2, 3, \ldots) \tag{11}$$

**Programming Problem 1: Recursion**

- Complete each of the functions.
- You should make sure you can find the values for small inputs by hand.
- There are no examples for this problem.

## Problem 2: Building Recursion Functions

In this problem, we'll build a simple recurrence and use it to build a more complicated recurrence. Here we want to write a function that finds the minimal value in a list. For this next problem, we'll implement this in several ways. It's always worthwhile to look at the core of the problem first. Let's write finding the minimum value of two numbers. We have two instances we'll face, when the two numbers are equal and when they're unequal

$$min(x, x) = x \tag{12}$$
$$min(x, y) = \begin{cases} x & \text{if } x < y \\ y & \text{if } y \leq x \end{cases} \tag{13}$$

Our goal was to write this for a list of numbers that has at least one number. We can have a list of one number or two numbers–this is really nothing more than what we had with the numbers

themselves.

$$MIN([x]) = min(x, x) \tag{14}$$
$$MIN([x, y]) = min(x, MIN[y]) \tag{15}$$

Prove to yourself that these both are equal to the min function above. Now, assume we have a list of numbers $[x, \ldots]$. The second MIN is an example of this. So, we form the following:

$$MIN([x]) = min(x, x) \tag{16}$$
$$MIN([x, y]) = min(x, MIN[y]) \tag{17}$$
$$MIN([x, y, \ldots, z]) = min(x, MIN[y, \ldots, z]) \tag{18}$$

Observe that $[x, y]$ and $[x, y, \ldots]$ is described more generally by $[x, \ldots]$. Rewriting the recurrence above yields:

$$min(x, y) = \begin{cases} x & \text{if } x < y \\ y & \text{if } y \leq x \end{cases} \tag{19}$$
$$MIN([x]) = x \tag{20}$$
$$MIN([x, \ldots]) = min(x, MIN[\ldots]) \tag{21}$$

**Deliverables for Programming Problem 2**

- Complete min(x,y) and MIN(lst) functions shown in Eq. 19-21.
- There are no examples for this problem.
- You can assume that the list is **non-empty**.

## Problem 3: Hexadecimal

Complete the function `hex_dec` that takes a string in hexadecimal (can be lower or uppercase) and returns the decimal equivalent as an integer. A couple of runs are shown.

```
1 def hex_dec(hex):
2     pass
3
4 print(hex_dec("C1"))
5 print(hex_dec("7dE"))
```

has output

```
193
2014
```

Deliverables for Programming Problem 3

- Complete the function.
- You are encouraged to use upper() for strings–please read about it at python.org

## Problem 4: Number Systems

We saw that any number $n$ in base $b$ can be converted into another base $c$ by building a sequence of symbols $n\%c, (n//c)\%c, ((n//c)//c)\%c, \ldots \{0,1\}\%c$ which are reversed, but can easily be changed. For example to convert $13_{10}$ to base 2 we have:

```
>>> 13 % 2, (13 // 2) % 2, ((13 // 2)//2) % 2, (((13 // 2)//2)//2) % 2
(1, 0, 1, 1)
# bin() is a function in Python that returns the binary equivalent of a decimal number.
# For example, we already calculated that binary equivalent of 13 is 1101 (reading it from the back)
# We can use bin() to check if the result of our calculations match with the output of bin().
>>> bin(13)
'0b1101'
# It matches our result, just that we have to ensure that we reverse our results to get the output in the correct format.
```

Now, carefully observe that there's a recursion lurking in that pattern. You should ask yourself what is the base case and when exactly should we recurse (or inductive case).

**Hint:** Note that we are doing the same calculation i.e., (13//2) over and over again until the base case, each time the value of 13 actually decrease, the first time it is (13//2), the next time it is ((13//2)//2), did you see the pattern here!. So, we should think about decreasing 'dn' each time we call our function ('c_') and stop when we reach the base case. You would have to think about the base case, but it is not that difficult if you read the line 2 of paragraph 1 above, you can spot when we stop. In fact, using recursion we can reverse the symbols too, since our function return a string so you know how you can change the order of characters in a string for example, if a = '1' and b = 'b' then, I can write a + b = '1b', or b+a = 'b1', so it's just a question of how you return, or in what order you return in your function.

```
#INPUT decimal number, base
#RETURN string of symbols representing that base
def c_(dn, base):
    pass
for i in range(14):
```

```
    b2,b3,b4 = c_(i,2),c_(i,3),c_(i,4)
    print(f"{int(b2,2)} {b2}, {int(b3,3)} {b3}, {int(b4,4)} {b4}")
```

has output:

```
1 1, 1 1, 1 1
2 2 10, 2 2, 2 2
3 3 11, 3 10, 3 3
4 4 100, 4 11, 4 10
5 5 101, 5 12, 5 11
6 6 110, 6 20, 6 12
7 7 111, 7 21, 7 13
8 8 1000, 8 22, 8 20
9 9 1001, 9 100, 9 21
10 10 1010, 10 101, 10 22
11 11 1011, 11 102, 11 23
12 12 1100, 12 110, 12 30
13 13 1101, 13 111, 13 31
```

A call c_(i, 2) returns a binary string. I've shown converting decimal to base 2, 3, and 4. Don't be confused by the example above, our function only deals with one base at a time, and converting to more than one base i.e., 2, 3 and 4 is for the sake of explanation. We call our function 'c_()' with one base i.e., 2 or 3 or 4 and it will return the string representation in that base.

**Deliverables for Programming Problem 4**

- Complete the recursive function that takes a decimal number and base and returns a string representation of the number. For example, $13_{10}$ is $31_4$ since

$$31_4 \quad = \quad 3 \times 4^2 + 1 \times 4^0 = 12 + 1 = 13 \tag{22}$$

- You can only use integer division, modulus, and string operations.

## Problem 5: Practicing with Recurrence Equations

This first problem is a description of recursion to describe removing all occurences of an object x in a list. Using the definition below, implement this in Python:

$$rr(x, [\,]) \quad = \quad [\,] \tag{23}$$

$$rr(x, [y, \ldots]) \quad = \quad \begin{cases} rr([\ldots]) & x = y \\ [x] + rr(x, [\ldots]) & otherwise \end{cases} \tag{24}$$

Note that, the base case would occur when we have an empty list and therefore, we have nothing to remove so we just return the empty list, otherwisem we use recursion as shown above.

You can see how this is similar to how we solved the **only-int** functions during this week's lab. Accordingly, think about how to pass the list (during recursive function call) each time with reduced size.

This second problem, given an object x, a non-negative integer n, and a list ($\ell$), return true if x occurs at least n times in ($\ell$). We will implement this with a helper function that we can create inside our 'oal' function. To be particular, our helper function (let's call it 'o_') is as shown in the recursion below.

$$o\_([\,], cnt) = cnt \geq n \tag{25}$$

$$o\_(\ell, cnt) = \begin{cases} o\_(\ell[1:], cnt+1) & x = \ell[0] \\ o\_(\ell[1:], cnt) & otherwise \end{cases} \tag{26}$$

The implemented function shown below. Note the base and the inductive cases: our 'o_()' function, would return when the list ($\ell$) is empty i.e., no more values to check, and it would return True or False based on if cnt >= n, otherwise it would recurse as shown above. So, If we implement our 'o_()' function correctly, then we can just use it's output as the return value in 'oal()'. This is so because in 'o_()', we have already solved the problem, and if it returns True or False, then we can just return True or False from the 'oal()' as well.

```
1 def oal(x,n,lst):
2    # Create your local function here o_() and call it with required arguments.
3    # You can use the value returned by o_(), as the final return value of oal() as well.
4    pass
5
6 for i in [4,3,1]:
7     print(oal(1,i,lst))
```

with output:

```
1 4 [1, 1, 1, 2, 2, 0]
2 False
3 3 [1, 1, 1, 2, 2, 0]
4 True
5 1 [1, 1, 1, 2, 2, 0]
6 True
```

Take a note of the fact that our oal() function above does not include the cnt variable–so you must create a local function (call it 'o_()') **within** the function oal as explained earlier.

**Deliverables for Programming Problem 5**

- Complete the recursive functions.

## Problem 6: A mystery

Observe the input/output of a recursive function mystic:

```
1 def mystic(xstr):
2     return len(xstr) == 1 or (len(xstr) == 2 and xstr[0] == xstr[1]) or (↩
          xstr[0] == xstr[len(xstr)-1] and mystic(xstr[1:len(xstr)-1]))
3
4 data = ["ABBa","ratsliveonnoevilstar", "ATOYOTA","ccc","cc","ccedc",]
5
6 for d in data:
7     print(mystic(d))
```

```
1 False
2 True
3 True
4 True
5 True
6 False
```

**Deliverables for Programming Problem 6**

- Rewrite the recursive function mystic so that it's not so arcane. It must be recursive!

## Problem 7: Recursion

Here is a function for all non-negative integers $m, n$:

$$A(0,n) = n+1 \tag{27}$$

$$A(m+1,0) = A(m,1) \tag{28}$$

$$A(m+1,n+1) = A(m,A(m+1,n)) \tag{29}$$

and an implementation:

```
1 def A(m, n):
2     pass
3
4 for i in range(4):
```

```
    for j in range(4):
        print(f"A({i,j}) = {A(i,j)},  ", end="")
```

with output:

```
A((0, 0)) = 1
A((0, 1)) = 2
A((0, 2)) = 3
A((0, 3)) = 4
A((1, 0)) = 2
A((1, 1)) = 3
A((1, 2)) = 4
A((1, 3)) = 5
A((2, 0)) = 3
A((2, 1)) = 5
A((2, 2)) = 7
A((2, 3)) = 9
A((3, 0)) = 5
A((3, 1)) = 13
A((3, 2)) = 29
A((3, 3)) = 61
```

**Deliverables for Programming Problem 7**

- Implement A.
- Visit this site (`https://gfredericks.com/things/arith/ackermann`) to visualize the calls for A(4,1).

## Problem 8: Loops

The definition of $\sin(x)$ is:

$$\sin(x) = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!} \tag{30}$$

We can approximate this value using only six terms for $\pi/4$

$$\sin(\pi/4) \approx \sum_{i=0}^{5} (-1)^i \frac{(\pi/4)^{2i+1}}{(2i+1)!} \tag{31}$$

Complete the function ii(x,stop) that takes an argument x and upper-limit of the sum to approximate $\sin(x)$. The code

```
import math

```

```
def ii(x,stop):
    pass

x,stop = math.pi/4,5
print(ii(x,stop))
print(math.sin(x))
```

has output:

```
0.7071067811796194
0.7071067811865476
```

Consider the following equations.

$$\sum_{i=1}^{n}(3i-2)^2 = (\frac{1}{2})n(6n^2-3n-1) \quad (32)$$

$$\sum_{i=1}^{n} i^3 = (\frac{1}{4})n^2(n+1)^2 \quad (33)$$

$$\cos(x)\cdot\cos(2x)\cdots\cos(2^{n-1}x) = \frac{\sin(2^n x)}{2^n \sin(x)} \quad (34)$$

for the last expression $n = 1, 2, 3, \ldots$ and $\sin(x) \neq 0$.

**Some helpful background:** Before we move ahead, note that equation-32, 33 and 34 are examples of analytical (closed from) solution. What it means is that, you can either solve it via the formula as shown in equations 32-34, or you can calculate each term individually, but the two solutions will match. Isn't that amazing that some people figured out that rather than calculating and adding each term in a loop, you can directly get the final result just by plugging the values in the formula :). Fasinating as it is to some of us, the analytical solutions are not available for all problems. We give them here for you to check that both loop based (going over each term) and the closed form solutions mathch. Equation-35 is the loop based calculation (for n = 5), and equation-37 is the direct result by plugging the value in the formula of equation-32.

$$\sum_{i=1}^{5}(3i-2)^2 = (3-2)^2+(6-2)^2+(9-2)^2+(12-2)^2+(15-2)^2 \quad (35)$$

$$= 1^2+4^2+7^2+10^2+13^2 = 1+16+49+100+169 = 335 \quad (36)$$

$$= (1/2)5(6(5^2)-(3(5))-1) = (5/2)(150-16) = 335 \quad (37)$$

Complete the code below that uses a loop to calculate the sum (the closed-form is included to help you validate your code)

```
import math

def iii(stop):
```

```
    def closed(n):
        return (1/4)*(n**2)*(n+1)**2
    sum = 0
    #complete bounded loop here
    return [sum,closed(stop)]

def iv(stop):
    def closed(n):
        return (1/2)*n*(6*(n**2)-(3*n)-1)
    sum = 0
    #complete bounded loop here
    return [sum, closed(stop)]

def vi(x,stop):
    def closed(x,n):
        return math.sin(x*(2**n))/((2**n)*math.sin(x))
    prod = 1
    #complete bounded loop here
    return [prod,closed(x,stop)]

print(iii(5))
print(iv(5))
print(vi(math.pi,5))
```

has output:

```
[225, 225.0]
[335, 335.0]
[-1.0, -1.0]
```

**Deliverables for Programming Problem 8**

- Complete the functions.
- **Round** your answers (for both loop based and closed form solution) to two decimal places.

## Problem 9: Environmental Quality

A small country–a very popular tourist destination–began to measure the environment quality over a span of a year. Three different groups produced three different models:

$$I(t) = \frac{5t^2 + t + 400}{2t^2 + 2t + 90} \tag{38}$$

$$J(t) = data[t] \tag{39}$$

$$data = [60.0, 59.00, 58.00, 55.26, 53.85, 52.8, \tag{40}$$

$$= 52.17, 51.55, 49.00, 46.26, 43.52, 50.76] \tag{41}$$

$$K(t) = \frac{50t^2 + 600}{t^2 + 10} \tag{42}$$

for $t = 0, 1, 2, \ldots 11$ where 0 means January. A committee assigned to work on this problem has decided to first examine how quality has decreased using each model. Implement a function env(f) that takes a model and determines the pairs of months with the **lowest** quality index. If multiple pairs have the same index, include them. Here is a run:

```
def I(t):
    pass

def J(t):
    pass

def K(t):
    pass


def env(f):
    months = ["Jan","Feb","Mar","Apr","May",
              "Jun","Jul","Aug","Sep","Oct","Nov","Dec"]
    pass

print(f"Model I\n {env(I)}")
print(f"Model J\n {env(J)}")
print(f"Model K\n {env(K)}")
```

produces

```
    Model I:
    [['Mar', 'Apr', 'Apr', 'May'], -0.21]
   Model J:
    [['Mar', 'Apr', 'Sep', 'Oct', 'Oct', 'Nov'], -2.74]
   Model K:
    [['Feb', 'Mar'], -1.95]
```

Evidiently there is some consensus in the early part of the year across all three models (Mar, Apr).

**Note on rounding:** To get accurate results, you should round the results to 2 decimal places but inside the env() function. For example, to get the quality index for a given month, we call I, J and K with index for that month i.e, (0, 1, 2, 3, ...), and once we have the quality index for that month, we can get the difference in quality between months as f(1) - f(0), where f can be any model. You should round this result, round(f(1)-f(0), 2), rather than rounding within the I, J or K functions.

**Deliverables for Programming Problem 9**

- Complete the functions.
- Round to 2 decimal places inside the env() function as explained above.

## Problem 10: Drawing a Triangle in Pygames

**This won't be graded**, but you'll need to complete it to finish it for the next homework. We're going to draw a recursive triangle–called the Sierpinski Triangle `https://en.wikipedia.org/wiki/Sierpi\%C5\%84ski_triangle`. It's a **fractal set** shown in Fig. 1. To do this, we need to first draw an equilateral triangle and pygames is a great tool for this. Please read about pygames at `https://www.pygame.org/news`.

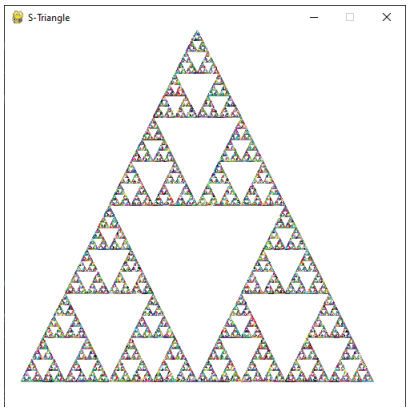


Figure 1: Recursive Triangle

We'll draw a triangle using the polygon method. The code we're providing has everthing you need to draw the triangle in Fig. 2 (uncomment that code to see how it works). Pygames, like games, uses quadrant IV to draw. The x-axis goes left to right, but the y-axis goes top to down. To draw an equilaterial triangle we'll use Fig. 3. The function you're implementing is

```
1 #INPUT takes a location loc = (x,y) pair of points top of the triangle and↩
      width
```

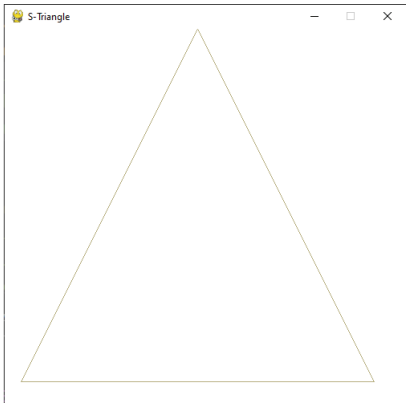


Figure 2: An equilateral triangle.

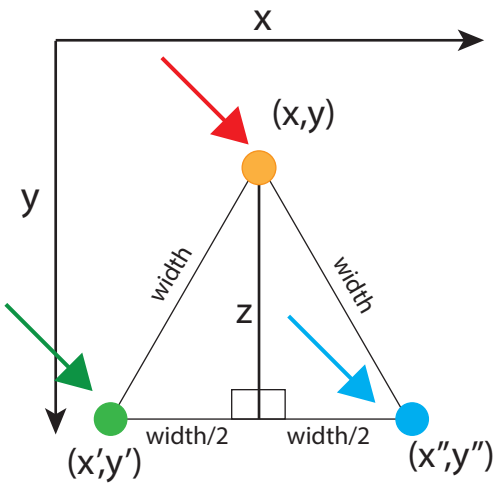


Figure 3: Drawning an equilaterial triangle in Pygames.

```
#RETURN 3 tuples of the equilateral triangle determined by loc and width
def triangle(loc,width):
    pass
```

This takes the top of the triangle as $(x, y)$ and a width $w$. Looking at Fig. 3, if we know the top and width, we can calculate the two other points $(x', y'), (x'', y'')$ that form the triangle using that there are two right triangles that form the equilateral triangle. The function should return $(x, y), (x', y'), (x'', y'')$.

**Important Note, REMEMBER TO COMMENT ALL CODE FOR PROBLEM 10 before submitting to AUOTOGRADER** (only for problem 10). Otherwise, it would return with an error. This is because the Autograder can't run graphical outputs for you as of now. Once you run the code for Problem-10, you may be puzzled that nothing is happening, look out for a small graphical

window, it may show behind your VSC window, so you may have to minimize VSC to look at it. Press the 'cross' icon to close that window.

**Deliverables for Programming Problem 10**

- Complete the triangle function.
- Try to draw a square using the left top point and a width.
- Try to draw an isosceles triangle using the top, side, and base.
- Remember to comment all code for this problem before submitting to Autograder.