C200 Programming Assignment № 7

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

Indiana University, Bloomington, IN, USA

March 31, 2023

Please please start early on this HW. The HW is due on **Friday, April, 07 at 10:59 PM EST**. Please commit, push and submit your work to the Autograder before the deadline.

1. Make sure that you are **following the instructions** in the PDF, especially the format of output returned by the functions. For example, if a function is expected to return a numerical value, then make sure that a numerical value is returned (not a list or a dictionary). Similarly, if a list is expected to be returned then return a list (not a tuple, set or dictionary).

2. Test **debug** the code well (syntax, logical and implementation errors), before submitting to the Autograder. These errors can be easily fixed by running the code in VSC and watching for unexpected behavior such as, program failing with syntax error or not returning correct output.

3. Make sure that the **code does not have infinite loop (that never exits) or an endless recursion (that never completes)** before submitting to the Autograder. You can easily check for this by running in VSC and watching for program output, if it terminates timely or not.

4. Given that you already tried points 1-3, if you see that Autograder does not do anything (after you press 'submit') and waited for a while (30 seconds to 50 seconds), try refreshing the page or using a different browser.

5. Once you are done testing your code, comment out the tests i.e. the code under the __name__ == "__main__" section.

Problem 1: Recursion to Generator

You worked with these last homework. For this homework, you'll write the tail recursion, while, and generator. This is a reminder that in the starter code, we are providing you with the regular recursion code, and the functions you must implement must utilize tail recursion, a while loop, or a generator as specified. In your starter code, function names will end in _t, _w, or _g if the function needs to be implemented using tail recursion, a while loop, or a generator, respectively. Review the lecture slides to learn more about generators.

$$
\begin{aligned}
p(0) &= 10000 && (1)\\
p(n) &= p(n-1) + 0.02p(n-1) && (2)\\
\\
c(1) &= 9 && (3)\\
c(n) &= 9c(n-1) + 10^{n-1} - c(n-1) && (4)\\
\\
d(0) &= 1 && (5)\\
d(n) &= 3d(n-1) + 1 && (6)\\
& && (7)
\end{aligned}
$$

Programming Problem 1: Recursion to Generators

- For reach function you'll write the tail recursive form, while, and generator.
- We have added the signature to help you–in particular, c(n) requres two accumulators.

Problem 2: Recursion

While equation 13 seems to be very complex, you have all the tools you need: c_2( ), sigma as a range function. The c_2() function is provided in the starter code to assist you in this problem. You are explicitly allowed (and encouraged) to use both sum( ) and list comprehension.

$$
\begin{aligned}
B_0 &= 1 \qquad (8)\\
B_n &= \frac{-\sum_{k=0}^{n-1}\binom{n+1}{k}B_k}{n+1} \qquad (9)
\end{aligned}
$$

$$
\begin{aligned}
B(1) &= \frac{-\sum_{k=0}^{0}\binom{2}{k}B_0}{1+1} \qquad (10)\\
&= -\frac{\binom{2}{0}1}{2} \qquad (11)\\
&= -.5 \qquad (12)
\end{aligned}
$$

$$
\begin{aligned}
B(2) &= -\frac{[\sum_{k=0}^{1}\binom{3}{k}B_k]}{3} \qquad (13)\\
&= -\frac{[\binom{3}{0}B_0+\binom{3}{1}B_1]}{3} \qquad (14)\\
&= -\frac{[1(1)+3(-.5)]}{3} \qquad (15)\\
&= -\frac{[1-1.5]}{3} = .5/3 = .1\bar{6}\\
B(3) &= -\frac{[\sum_{k=0}^{2}\binom{4}{k}B_k]}{4} \qquad (16)\\
&= -\frac{[\binom{4}{0}B_0+\binom{4}{1}B1+\binom{4}{2}B2]}{4} \qquad (17)\\
&= -\frac{[1(1)+4(-.5)+6(1.\bar{6})]}{4} \qquad (18)\\
&= -\frac{[1-2+1]}{4} = 0 \qquad (19)
\end{aligned}
$$

Here are some outputs:

output

```
B(0) = 1
B(1) = -0.5
B(2) = 0.16666666666666666
B(3) = -0.0
B(4) = -0.033333333333333305
B(5) = -7.401486830834377e-17
```

Programming Problem 2: Recursion

- Complete the function B() using recursion.
- You can (and are encouraged to) use sum().
- List comprehension is useful too!

Problem 3: Approximating the Derivative

Calculus is used in virtually every field and is fundamental to understanding AI. In this problem we approximate the derivate which is instantaneous change. We'll focus on univariate functions initially:

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon} \tag{20}$$

This is really a λ function, since we're given a function f and a value h and returning a function f' that has x as an argument. We can approximate $f'(x)$ shown here:

$$f'(x) \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{\epsilon} \tag{21}$$

We will write a function called derivative that takes a function and epsilon and returns a lambda function. for example,

$$\begin{aligned} f(x) &= x^2 - 3x && (22) \\ f'(x) &= 2x - 3 && (23) \\ f'(2) &= 4 - 3 = 1 && (24) \end{aligned}$$

```
def derivative(f,epsilon):
    pass

def f(x):
    return x**2 - 3*x

data = 2
epsilon = 10e-8
print((derivative(f,epsilon)(data)))
f_prime = derivative((lambda x:x**2-3*x),epsilon)
print(f_prime(data))
```

has outputs:

```
0.9999999961429751
0.9999999961429751
```

Deliverables for Programming Problem 3

- Complete the derivate function

Problem 4: Assessing the Quality of a Model

One typical approach to assessing the quality of a model is to determine the sum of squares. Given data $D = [(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})]$, we build a model $f(x) \leftarrow model(X, Y)$. We find this sum:

$$\text{SSE} = \sum_{i=0}^{n-1}(y_i - f(x_i))^2 \tag{25}$$

SSE is the (S)um of (S)quare (E)rror. If it's zero, then there isn't any error. Numpy's polyfit will calculate the SSE error for you, but you'll be implementing it from scratch and then compare values. The critical line is:

```
z,SSE,*_ = np.polyfit(x,y,degree,full=True)
```

by setting full we can get the SSE. As a toy example, assume $D = [(1,2),(2,3),(3,9)]$ and the model $f(x) = x^2$. Then the SSE is

$$\begin{aligned}\text{SSE} &= \sum_{i=0}^{2}(y_i - f(x_i))^2 && (26)\\ &= (2 - f(1))^2 + (3 - f(2))^2 + (9 - f(3))^2 && (27)\\ &= (2-1)^2 + (3-4)^2 + (9-9)^2 = 2 && (28)\end{aligned}$$

When complete the plot you'll see is shown in Fig. 1. Note that the error is automatically computed and is shown at the top of the graph in the title.

Hint: Reading the file is easy and you don't necessarily need to use csv_reader (however we won't penalize if you use it). You can read all lines at once using the readlines() function and then split the lines by "," to get each field (meaning the numbers) in each line and then can store them in a list. You can also revise the lab7 on file reading to understand how to read the files. Again, if you want to use csv_reader then you won't be penalized.

Deliverables for Programming Problem 4

- Complete the get_data and error functions
- The data is from a file that contains scores for happiness as a function of income named `income_data.csv`
- Keep in mind that after you have tested your code, you must comment out the 'import matplotlib' and the plotting code given under the __main__ before submitting to the Autograder. Autograder can not draw graphical plots on the web browser so it will return an error if you do not comment out the plotting code/import matplotlib.

Figure 1: Plots of happiness data D as a function of income and the model (blue). There are 498 data points. The average SSE is ≈ 0.514

.

Problem 5: Candlesticks

Given a sequence of values $[v_0, v_1, \ldots, v_n]$ where $0, 1, \ldots, n$ represents time, a candlestick is the most popular visualization that is used to both summarize and inform financial analysts. For a given sequence, a box and lines are drawn as shown in Fig. 2 (right)

For any sequence we can quickly get the four values we need:

```
>>> import random as rn
>>> prices = [rn.randint(2,50) for _ in range(200)]
>>> open,close,max_p,min_p = prices[0],prices[len(prices)-1],max(prices),min(prices)
>>> open,close,max_p,min_p
(37, 2, 50, 2)
```

Using matplotlib's patches https://matplotlib.org/stable/api/_as_gen/matplotlib.patches.Rectangle.html we can translate the four values into a candlestick, Fig. 2. (left). To plot a rectangle, we'll need the lower-lefthand corner (x,y), width, and height.

Note: Note that the input is a collection of values contained inside a list so each sublist represents a candelstick. You will iterate over the contents of the list one-by-one and process each sublist insde the make() function to plot the candlesticks. The plotting is taken care of by the matplotlib patches but you still need to finish the make() function to return the values that are needed by the matplotlib patches. For quickly understanding how it works, try playing with the starter code of problem 5 under "__main__".

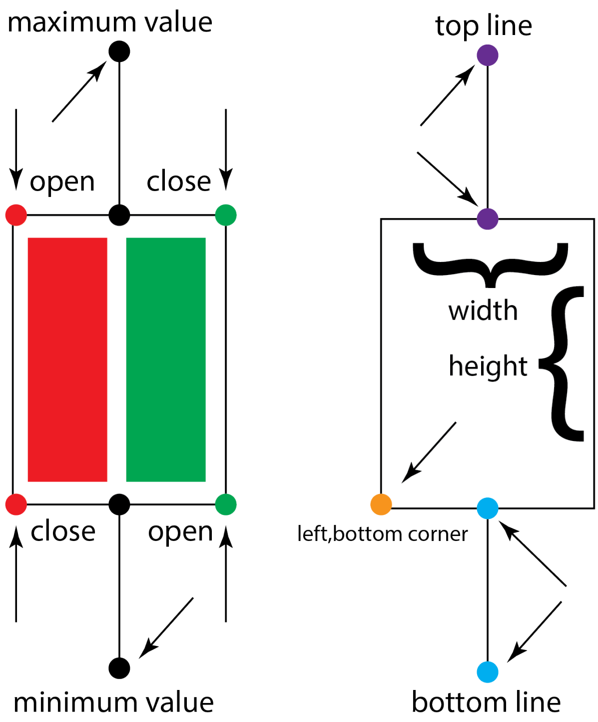

Figure 2: A candlestick image (right). The box is created by the opening and closing values. The top line is from the maximal value to the top middle of the box. The bottom line is from the minimal value to the bottom middle of the box. The color is determined by the open, close values. If open is greater than close, then the color is red; otherwise it's green. We use matplotlib patches. The candlestick image is determined the rectangle and two lines. The rectangle is determined by the lower-left point (x, y), width, and height. The two lines are determined by two points computed from max, min, open, and close.

Hint: As each sublist is passed inside the make() function, you will first find the correct values of open, close, max_p and min_p (from the sublist), and then use those to values to calculate the values required to plot the candlestick. How to find, open, close, max_p and min_p is shown in the code listing above. The open and close will be used as the y values (i.e. the values on the y axis) and you have to find them from the sublist.

Let's look at the code:

```
data = [[20,15,32,10],[10,14,15,9],
        [22,23,27,9],[15,16,16,15],
        [26,12,30,2],[5,30,40,4]]

# open,close,max_p,min_p = 20, 15, 32, 10

# INPUTS ith candle, starting value of x, default width, and the four critical values: open, close, max\_p, min\_p.
# RETURN three tuples: (point, width, height, color), topline,
```

```
bottomline
# point is the coordinates of the lower-left point x, y, width and height are numeric values and color will be a string of color ex. 'red' or 'green'
# topline ((xt0,yt0),(xt1,yt1)) line from max to top middle of box
# bottomline ((xb0,yb0),(xb1,yb1)) line from min to bottom middle of box

# When you see the code for testing Problem5 under __main__, you will see that the first three values of the first tuple i.e., point, width and height are passed as the first arguement of matplotlib.patches.Rectangle() function and the last value i.e. color is passed as the second arguement. Feel free to play around with the test code to get a feeling of how it is working. You will understand it much better with a bit of experimentation.

def make(i, start, width_default, d):
    pass

fig = plt.figure()
ax = fig.add_subplot(111)
start = 0
default_width = 10
for i in range(len(data)):

    candle_box, top_line, bottom_line = make(i,start,default_width, data[i])
    print(candle_box)
    ax.add_patch(matplotlib.patches.Rectangle(*candle_box[0:3],color =candle_box[3]))
    plt.plot([x for x,_ in top_line],[y for _,y in top_line],'black')
    plt.plot([x for x,_ in bottom_line],[y for _,y in bottom_line],'black')
    start += default_width

plt.xlabel("time (hour)")
plt.ylabel("Stock X price")
plt.title("Candlestick for Stock X mm/dd/yyyy")
plt.xlim([0, 60])
plt.ylim([0, 35])
plt.show()
```

will produce

```
((0, 15), 10, 5, 'red')
((10, 10), 10, 4, 'green')
```

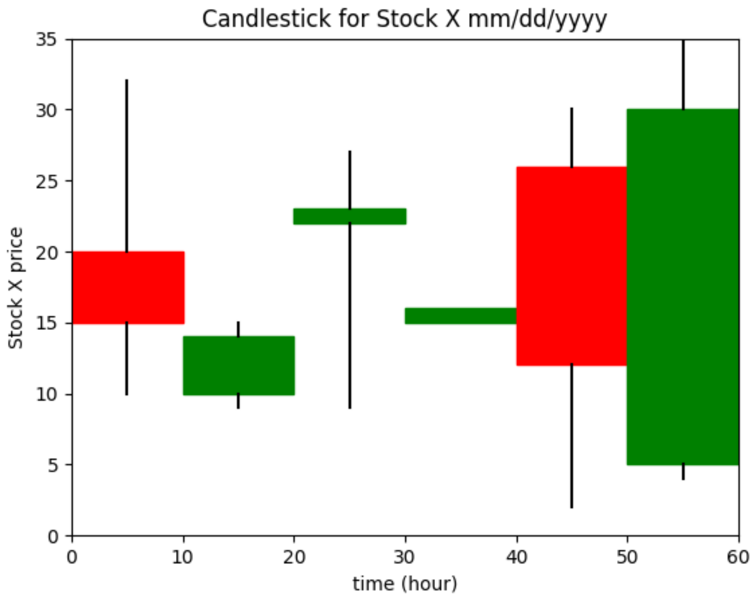

Figure 3: Six candlesticks.

```
((20, 22), 10, 1, 'green')
((30, 15), 10, 1, 'green')
((40, 12), 10, 14, 'red')
((50, 5), 10, 25, 'green')
```

and the plot.

Deliverables for Programming Problem 5

- Complete the function make that returns three things: the data for the rectangle, the top line, and the bottom line.
- **Hint:** If you see the output produced by printing candel_box, notice how the value of start changes for each candle_box-this is what makes each candlebox shift in the plot so that each candelbox start a specific distance after the previous one so that all of them looks seperated for easy visualization.
- Keep in mind that after you have tested your code, you must comment out the 'import matplotlib' and the plotting code given under the __main__ before submitting to the Autograder. Autograder can not draw graphical plots on the web browser so it will return an error if you do not comment out the plotting code/import matplotlib.

Programming partners

akundur@iu.edu, cartwolf@iu.edu
jl335@iu.edu, saseiber@iu.edu
mgorals@iu.edu, tshore@iu.edu
sijatto@iu.edu, snuthala@iu.edu
brakin@iu.edu, yijwei@iu.edu
qugriff@iu.edu, chnico@iu.edu
georliu@iu.edu, alexschu@iu.edu
lgflynn@iu.edu, bpoddut@iu.edu
moesan@iu.edu, greymonr@iu.edu
joshbrin@iu.edu, shahneh@iu.edu
anderblm@iu.edu, yz145@iu.edu
dombish@iu.edu, mszczas@iu.edu
mdgamble@iu.edu, imalhan@iu.edu
cheng47@iu.edu, sabola@iu.edu
eddykim@iu.edu, davthorn@iu.edu
bradhutc@iu.edu, lenonti@iu.edu
sfawaz@iu.edu, peschulz@iu.edu
bcarl@iu.edu, isarmoss@iu.edu
sl92@iu.edu, jcpilche@iu.edu
bgabbert@iu.edu, aw149@iu.edu
klongfie@iu.edu, sousingh@iu.edu
jl263@iu.edu, kalomart@iu.edu
aketcha@iu.edu, patel88@iu.edu
sbehman@iu.edu, omilden@iu.edu
jonhick@iu.edu, pvinod@iu.edu
abolad@iu.edu, criekki@iu.edu
achimeba@iu.edu, rpmahesh@iu.edu
aalesh@iu.edu, dpepping@iu.edu
arjbhar@iu.edu, cmulgrew@iu.edu
stfashir@iu.edu, jmissey@iu.edu
ljianghe@iu.edu, nmonberg@iu.edu
neybrito@iu.edu, btpfeil@iu.edu
jegillar@iu.edu, dy11@iu.edu
jacklian@iu.edu, kvanever@iu.edu
abdufall@iu.edu, arnpate@iu.edu
aaberma@iu.edu, nireilly@iu.edu
megofolz@iu.edu, jorzhang@iu.edu
nbakken@iu.edu, johwarre@iu.edu
jadbenav@iu.edu, mostrodt@iu.edu

sakinolu@iu.edu, dannwint@iu.edu
tifhuang@iu.edu, jwescott@iu.edu
joecool@iu.edu, yasmpate@iu.edu
nokebark@iu.edu, hnichin@iu.edu
egillig@iu.edu, leplata@iu.edu
davgourl@iu.edu, edshipp@iu.edu
jaybaity@iu.edu, dwinger@iu.edu
wdoub@iu.edu, rmatejcz@iu.edu
anthoang@iu.edu, almoelle@iu.edu
agoldswo@iu.edu, elyperry@iu.edu
siqidong@iu.edu, caitreye@iu.edu
tyfeldm@iu.edu, esmmcder@iu.edu
actonm@iu.edu, bashih@iu.edu
aadidogr@iu.edu, chrinayl@iu.edu
arykota@iu.edu, imaychru@iu.edu
katzjor@iu.edu, bensokol@iu.edu
nwbarret@iu.edu, hvelidi@iu.edu
mgambett@iu.edu, psaggar@iu.edu
nbulgare@iu.edu, webejack@iu.edu
coldjone@iu.edu, lmamidip@iu.edu
jasoluca@iu.edu, vthakka@iu.edu
ecastano@iu.edu, rair@iu.edu
jackssar@iu.edu, trewoo@iu.edu, zhaozhe@iu.edu
dud@iu.edu, cartmull@iu.edu
sjkallub@iu.edu, alescarb@iu.edu
qhodgman@iu.edu, zisun@iu.edu
hc51@iu.edu, mw154@iu.edu
pkasarla@iu.edu, ss126@iu.edu
ghyatt@iu.edu, grschenc@iu.edu
dk80@iu.edu, parisbel@iu.edu
sihamza@iu.edu, jonvance@iu.edu
aladkhan@iu.edu, danwils@iu.edu
swcolson@iu.edu, shethsu@iu.edu
lsherbst@iu.edu, svaidhy@iu.edu
vilokale@iu.edu, schwani@iu.edu
aibitner@iu.edu, cjromine@iu.edu
ivycai@iu.edu, jyamarti@iu.edu
ekkumar@iu.edu, sjxavier@iu.edu
dilkang@iu.edu, jarymeln@iu.edu
kbchiu@iu.edu, sowmo@iu.edu
jolawre@iu.edu, macsvobo@iu.edu

jacbooth@iu.edu, ornash@iu.edu
moalnass@iu.edu, jvalleci@iu.edu
hollidaa@iu.edu, nkmeade@iu.edu
jacuau@iu.edu, roelreye@iu.edu
ogift@iu.edu, crfmarti@iu.edu
chnbalta@iu.edu, sjvaleo@iu.edu
marbelli@iu.edu, ajneel@iu.edu
rythudso@iu.edu, wuyul@iu.edu
joracobb@iu.edu, jmom@iu.edu
sgcolett@iu.edu, avpeda@iu.edu