C200 Programming Assignment №8
File I/O, Sorting, & Numpy

Professor M.M. Dalkilic

Computer Science

School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

April 4, 2025

Introduction

Due Date: 5:00 PM, Friday, April 11, 2025. Please note that the assignment will need to be submitted to Gradescope and **must be done before the deadline. We do not accept late submissions**. Any form of cheating or plagiarism will be dealt with seriously. If this results in a grade change, we are compelled by the University to send a formal notification to the Dean, please review the policy on academic misconduct in the syllabus.

As always, all the work should be with you and your partner; but *both* of you should contribute. Please remember:

- You will be submitting on Gradescope
- Your highest score of **10 submissions** will be used for your final score.
- Any function that is defined requires a docstring, please refer to lab 03 for what a docstring should have.
- Do NOT change the function names, or the parameters.
- Finally, manual grading will be done, so if you receive a 50 out of 100 on the autograder, it means the lowest grade you can receive is 50. The rest will come from manual grading of the problems.

If your timestamp is 5:01PM or later, the homework will not be graded. So **do not wait until 4:59 PM** to submit, make sure to also plan accordingly. If you have questions or problems with Gradescope, please visit office hours or make a post on Inscribe ahead of time. Since you are working in pairs, your paired partner is shown on canvas. Esure that you are using only features that we have discussed in lectures and labs.

Problem 1: Family Tree

This problem focuses on implementing functions with list comprehensions. The data is in a file called family.txt. The data is in two columns as shown in the table below, and also is given as a visualization. It is a family tree. The names are simply characters, so no type conversion is necessary.

Parent	Child
0	1
0	4
0	5
1	2
1	3
5	6
6	7
6	A
6	g

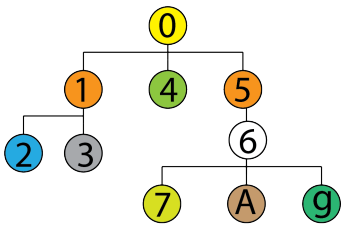


Table 1: Table (left) from family.txt file, and it's visualization (right) as a tree

You will implement a series of queries based on the data that you read from the file. The queries are formed as Python functions. A few of the functions required on this problem are already completed here,

```
1
2  # children of name
3  def get_child(name,data):
4      return [child for parent,child in data if parent == name]
5
6  #does name have any children
7  def has_children(name,data):
8      return bool(get_child(name,data))
9
10 #get all values in data (cannot help duplicates)
11 def get_all(data):
12     return [k for k in [i for i,j in data] + [j for i,j in data]]
13
14 #data_1 has file contents...
15
16 print(has_children('0',data_1)) #true
17 print(has_children('7',data_1)) #false
18 print(get_child('6',data_1))
19 print(get_all(data_1))
```

has outputs:

```
True
False
['7', 'A', 'g']
['0', '0', '0', '1', '1', '5', '6', '6', '6', '1', '4', '5', '2', '3',↩
    '6', '7', 'A', 'g']
```

Programming Problem 1: Family Tree

- Complete the code to read the file. Do not change any string–these are the names.
- Complete each of the functions only using comprehension or bool on a comprehension
- Include a docstring for each function you implement, and one for *get_child*() as well

Problem 2: Central Dogma, DNA to RNA to Protein

The central dogma in biology is that DNA → RNA → protein. If you want, you can read more about it at `https://en.wikipedia.org/wiki/Central_dogma_of_molecular_biology`. In this problem you will read in a two files: The first file (amino_acids.txt) contains mapping of codons (a triplet of three DNA bases is called as **codon**) , by mapping we mean a rule that tells which specific codons will convert into a specific amino acid, and the second file (DNA.txt) contains a DNA sequence. The first file will help you to understand the mappings, and then by using the mappings, we will convert the sequence from the second file (DNA.txt) into amino acids. In essence, we will write a program to convert the DNA into protein as living organisms do! (a protein is a sequence of amino acids)

To start with, the contents of amino_acids.txt are shown below in the table.

Name	Abr.	Codons
Isoleucine	I	ATT, ATC, ATA
Leucine	L	CTT, CTC, CTA, CTG, TTA, TTG
Valine	V	GTT, GTC, GTA, GTG
Phenylalanine	F	TTT, TTC
Methionine	M	ATG
Cysteine	C	TGT, TGC
Alanine	A	GCT, GCC, GCA, GCG
Glycine	G	GGT, GGC, GGA, GGG
Proline	P	CCT, CCC, CCA, CCG
Threonine	T	ACT, ACC, ACA, ACG
Serine	S	TCT, TCC, TCA, TCG, AGT, AGC
Tyrosine	Y	TAT, TAC
Tryptophan	W	TGG
Glutamine	Q	CAA, CAG
Asparagine	N	AAT, AAC
Histidine	H	CAT, CAC
Glutamic acid	E	GAA, GAG
Aspartic acid	D	GAT, GAC
Lysine	K	AAA, AAG
Arginine	R	CGT, CGC, CGA, CGG, AGA, AGG
Stop codons	-	TAA, TAG, TGA

The first column is the full name of the amino acid. The second column is the abbreviation as one letter initial (for the same amino acid). For the Stop_codons, we use a dash. The rightmost column are what three letters of DNA are used to make the amino acid. The amino acid Arginine has an abbreviation R. There are six codon (three bases of DNA) that code for Arginine: CGT, CGC, CGA, CGG, AGA, AGG, as can be seen in the table.

About DNA.txt: It's basically a FASTA file. Please don't be confused by the name-it's just a text file that folllows certain rules, hence the special name. It has two parts: a header (information about the DNA sequence that it contains) and the DNA sequence itself. Here's the one you'll be using (don't copy them from here, we have already pushed both files to your repositories).

```
>HSGLTH1 Human theta 1-globin gene
CCACTGCACTCACCGCACCCGGCCAATTTTTGTGTT
TTTAGTAGAGACTAAATACCATATAGTGAACACCTA
AGACGGGGGGCCTTGGATCCAGGGCGATTCAGAGG
GCCCCGGTCGGAGCTGTCGGAGATTGAGCGCGCGC
GGTCCCGGGATCTCCGACGAGGCCCTGGACCCCCG
GGCGGCGAAGCTGCGGCGCGGCGCCCCCTGGAGGC
CGCGGGACCCCTGGCCGGTCCGCGCAGGCGCAGCG
GGGTCGCAGGGCGCGGCGGGTTCCAGCGCGGGGAT
GGCGCTGTCCGCGGAGGACCGGGCGCTGGTGCGCG
CCCTGTGGAAGAAGCTGGGCAGCAACGTCGGCGTCT
ACACGACAGAGGCCCTGGAAAGGTGCGGCAGGCTG
GGCGCCCCCGCCCCCAGGGGCCCTCCCTCCCCAAG
CCCCCCGGACGCGCCTCACCCACGTTCCTCTCGCAG
GACCTTCCTGGCTTTCCCCGCCACGAAGACCTACTT
CTCCCACCTGGACCTGAGCCCCGGCTCCTCACAAGT
CAGAGCCCACGGCCAGAAGGTGGCGGACGCGCTGA
GCCTCGCCGTGGAGCGCCTGGACGACCTACCCCAC
GCGCTGTCCGCGCTGAGCCACCTGCACGCGTGCCA
GCTGCGAGTGGACCCGGCCAGCTTCCAGGTGAGCG
GCTGCCGTGCTGGGCCCCTGTCCCCGGGAGGGCCC
CGGCGGGGTGGGTGCGGGGGGCGTGCGGGGCGGG
TGCAGGCGAGTGAGCCTTGAGCGCTCGCCGCAGCT
CCTGGGCCACTGCCTGCTGGTAACCCTCGCCCGGCA
CTACCCCGGAGACTTCAGCCCCGCGCTGCAGGCGTC
GCTGGACAAGTTCCTGAGCCACGTTATCTCGGCGCT
GGTTTCCGAGTACCGCTGAACTGTGGGTGGGTGGCC
GCGGGATCCCCAGGCGACCTTCCCCGTGTTTGAGTA
AAGCCTCTCCCAGGAGCAGCCTTCTTGCCGTGCTCT
CTCGAGGTCAGGACGCGAGAGGAAGGCGC
```

Though not necessary but if you want, you can read about this gene here: `https://pubmed.`

ncbi.nlm.nih.gov/3422341/. In the file, the first line describes the sequence providing the name and other attributes. The remaining lines is the DNA sequence (ignore all whitespace), all lines after the header are part of the same sequence so there are no line breaks (ignore whitespaces) in the sequence.

Example: how to translate DNA into a protein

To convert from DNA to protein, we use a sequence of codons. We'll bold the protein when its translated.

Let's look at the first twelve bases: CCACTGCACTCA. Every three bases uniquely determine an amino acid.

1. Start with the first codon CCA, CCACTGCACTCA.
2. Looking at the first file we see: Proline, P, CCT, CCC, CCA, CCG. This means we can rewrite CCA as P.
3. Looking at the next codon CTG, **P**CTGCACTCA
4. We find it matches Leucine, L, CTT, CTC, CTA, CTG, TTA, TTG. So our protein is PL.
5. The next three are CAC **PL**CACTCA.
6. The table has Histidine, H, CAT, CAC.
7. The final three are TCA. **PLH**TCA
8. This matches Serine, S, TCT, TCC, TCA, TCG, AGT, AGC.
9. The protein is **PLHS**.

If you are at the end and only have two bases, you cannot match a protein, so you ignore them. Suppose we had CCAC. We know CCA is P. Then we only have C left. We ignore it.

How to approach this problem

Our main task is to take the DNA (by reading DNA.txt) and produce a string of single letters (as shown in above example, points 1-9) that reflect the encoding. Here is one way to solve this (you should implement how you feel most comfortably)

1. First you'll read in the table from amino_acids.txt, and create a dictionary whose entries are:

$$aa\_d \quad = \quad \{(c_0, c_1, \ldots, c_n) : [name, letter], \ldots\}$$

where c_i is a three letter codon, *name* is the full name of the amino acid, and *letter* is the single letter for the amino acid. Make sure you follow the format of keys and values exactly as shown here. The function get_amino_acid() reads the amino_acids.txt file and returns a dictionary.

The dictionary is also shown below in the code listing.

2. Read in the DNA sequence, the function get_DNA() reads the DNA.txt file and returns a faste data structure [header, DNA] (FASTA data structure) where header is the first line of the file DNA.txt and DNA is the DNA sequence (the sequence of A,T,G,C after the first line) (ignoring any whitespace). The read-in FASTA sequence is also shown below in the code listing.

3. The function translate() takes a FASTA data structure (that you created in step-2) and returns a string that is the translation using the dictionary (that you created in step-1). We can do a simple print to see whether our translation is the same as actual. An example of translate() function is also shown below in the code listing (you are encouraged to cross-check via pen and paper).

This code creates the dictionary and FASTA file (as a list), translates, and validates:

```
1  print("Dictionary")
2  print(aa_d)
3  print("FASTA file")
4  print(DNA_d)
5  print("Translations match:", str(protein == actual))
6
7  #should return "PLHS"
8  print(translate(["nothing", "CCACTGCACTCA"], aa_d))
9
10 #should return "D-"
11 print(translate(["nothing", "GACTAA"], aa_d))
```

has output:

```
1  Dictionary
2  {('ATT', 'ATC', 'ATA'): ['Isoleucine', 'I'], ('CTT', 'CTC', 'CTA', '↩
       CTG', 'TTA', 'TTG'): ['Leucine', 'L'], ('GTT', 'GTC', 'GTA', 'GTG')↩
       : ['Valine', 'V'], ('TTT', 'TTC'): ['Phenylalanine', 'F'], ('ATG',)↩
       : ['Methionine', 'M'], ('TGT', 'TGC'): ['CYSteine', 'C'], ('GCT', '↩
       GCC', 'GCA', 'GCG'): ['Alanine', 'A'], ('GGT', 'GGC', 'GGA', 'GGG')↩
       : ['Glycine', 'G'], ('CCT', 'CCC', 'CCA', 'CCG'): ['Proline', 'P'],↩
        ('ACT', 'ACC', 'ACA', 'ACG'): ['Threonine', 'T'], ('TCT', 'TCC', '↩
       TCA', 'TCG', 'AGT', 'AGC'): ['Serine', 'S'], ('TAT', 'TAC'): ['↩
       Tyrosine', 'Y'], ('TGG',): ['Tryptophan', 'W'], ('CAA', 'CAG'): ['↩
       Glutamine', 'Q'], ('AAT', 'AAC'): ['Asparagine', 'N'], ('CAT', 'CAC↩
       '): ['Histidine', 'H'], ('GAA', 'GAG'): ['Glutamic_acid', 'E'], ('↩
       GAT', 'GAC'): ['AsparTic acid', 'D'], ('AAA', 'AAG'): ['Lysine', 'K↩
       '], ('CGT', 'CGC', 'CGA', 'CGG', 'AGA', 'AGG'): ['Arginine', 'R'], ↩
       ('TAA', 'TAG', 'TGA'): ['Stop_codons', '-']}
3  FASTA file
```

```
['>HSGLTH1 Human theta 1-globin gene', '↩
    CCACTGCACTCACCGCACCCGGCCAATTTTTGTGTTTTTAGT
AGAGACTAAATACCATATAGTGAACACCTAAGACGGGGGGC
CTTGGATCCAGGGCGATTCAGAGGGCCCCGGTCGGAGCTGT
CGGAGATTGAGCGCGCGCGGTCCCGGGATCTCCGACGAGGC
CCTGGACCCCCGGGCGGCGAAGCTGCGGCGCGGCGCCCCCT
GGAGGCCGCGGGACCCCTGGCCGGTCCGCGCAGGCGCAGCG
GGGTCGCAGGGCGCGGCGGGTTCCAGCGCGGGGATGGCGCT
GTCCGCGGAGGACCGGGCGCTGGTGCGCGCCCTGTGGAAGA
AGCTGGGCAGCAACGTCGGCGTCTACACGACAGAGGCCCTG
GAAAGGTGCGGCAGGCTGGGCGCCCCCGCCCCCAGGGGCCC
TCCCTCCCCAAGCCCCCCGGACGCGCCTCACCCACGTTCCTC
TCGCAGGACCTTCCTGGCTTTCCCCGCCACGAAGACCTACTT
CTCCCACCTGGACCTGAGCCCCGGCTCCTCACAAGTCAGAGC
CCACGGCCAGAAGGTGGCGGACGCGCTGAGCCTCGCCGTGG
AGCGCCTGGACGACCTACCCCACGCGCTGTCCGCGCTGAGC
CACCTGCACGCGTGCCAGCTGCGAGTGGACCCGGCCAGCTT
CCAGGTGAGCGGCTGCCGTGCTGGGCCCCTGTCCCCGGGAG
GGCCCCGGCGGGGTGGGTGCGGGGGGCGTGCGGGGCGGGT
GCAGGCGAGTGAGCCTTGAGCGCTCGCCGCAGCTCCTGGGC
CACTGCCTGCTGGTAACCCTCGCCCGGCACTACCCCGGAGAC
TTCAGCCCCGCGCTGCAGGCGTCGCTGGACAAGTTCCTGAGC
CACGTTATCTCGGCGCTGGTTTCCGAGTACCGCTGAACTGTG
GGTGGGTGGCCGCGGGATCCCCAGGCGACCTTCCCCGTGTTTG
AGTAAAGCCTCTCCCAGGAGCAGCCTTCTTGCCGTGCTCTCTC
GAGGTCAGGACGCGAGAGGAAGGCGC']
Translations match: True
PLHS
D-
```

Programming Problem 2: Central Dogma, DNA to RNA to Protein

- Complete the functions get_DNA(), get_amino_acids() and translate() as per the specifications.
- You are allowed to use replace for space and "{''. Other uses of it will actually be more difficult.
- Include a docstring for each function you implement

Problem 3: Using Radix Sort

In lecture I provided a less succint radix sort so we could more easily observe each step. Here I write a simpler, though more succint, version:

```
1  def radix (lst,digit_index = 0):
2      if lst:
3          cluster = [[] for _ in range(10)]
4          for number in lst:
5              index = int(number[-(digit_index + 1)])
6              cluster[index] += [number]
7
8          sorted, unsorted = [],[]
9          for block in cluster:
10             for number in block:
11                 if len(number) > digit_index + 1:
12                     unsorted.append(number)
13                 else:
14                     sorted.append(number)
15         return sorted + radix(unsorted, digit_index + 1)
16     else:
17         return []
18
19 data21 = ["101","10","12","1000","99","1","5", '100', '120', '990', '↩
       310', '0', '301', '102', '654']
20 print(radix(data21))
```

producing

```
1  ['0', '1', '5', '10', '12', '99', '100', '101', '102', '120', '301', '↩
       310', '654', '990', '1000']
```

Using the string type helps a lot. That the output remains strings isn't problematic, since we can easily convert to an integer. Observe that the location in the list is the integer version of the digit we're examining. The `digit_index = 0` means we're starting with the last character of the string. Remember, we can iterate through a string using -1,-2,-3,... that starts from the right. These are lines 3-6. Once they've been partitioned, we separated strings of larger length (to be radix sorted again looking at the next left digit) from current length.

We can use radix to sort decimals as well:

```
1  def radix_decimal (lst):
2      pass #calls radix
3
4  data22 = [".301",".101",".20",".1",".12",".654",".99",".31",".309",]
5
```

```
print(radix_decimal(data22))
d_22 = data22[::]
d_22.sort()
print(d_22)
```

produces:

```
['.1', '.101', '.12', '.2', '.301', '.309', '.31', '.654', '.99']
['.1', '.101', '.12', '.20', '.301', '.309', '.31', '.654', '.99']
```

How can we use radix in its original form? Observe that the decimal maintains order when we remove the decimal and pad zeros to the right. For example, in this data, the longest decimal has three digits; therefore, we add zeros to make the length of all the numbers length three, *e.g.*, .1 $\rightarrow$ 100, .99 $\rightarrow$ 990, and .20 $\rightarrow$ 200. After padding zeros to the right, radix sort handles ordering by character and we reinsert the decimal to return correct numeric values. The ordering can be now done correctly with radix. After sorting we have to prepend the decimal and remove trailing zeros. We may, as a result, produce .2 instead of the original .20, but this doesn't matter. Lastly, we assume we will never had .0.

Programming Problem 3: Radix Sort

- Complete the radix_decimal() function that *must* use the radix function
- You are not allowed to change the radix function, you will receive a zero if you do so.
- You must return strings with the decimal point and without trailing zeros
- Tip: To remove the trailing zeros, we can use the strip() function that we learned in the FileIO lab and in lecture. However, you will have to call it differently because the default behavior of strip() is to remove the white spaces.
- Include a docstring for the function

Problem 4: Ksumsort

In this problem you'll be given a non-empty list of numbers `lst` and a non-negative integer $k = 0, 1, 2, \ldots$. You'll implement a function `ksumsort(lst,k) that first builds a list of tuples [t1, t2, ..., tn]` where `n = 2^len(lst)`. Each tuple is a pair `(s, [x0, ... ])` such that `s = sum([x0,...])` representing all the possible sums. Second, the result is sorted on the absolute difference between k and the sum. The function should generate all non-empty subsets of a list, and construct the tuples of the subset sums and subsets. These tuples should then be sorted based on their absolute difference from k. The sorted list of tuples should then be returned. As example is shown below,

```
1 lst = [1,2,3]
2 print(kns(lst,0))
3 print(kns(lst,3))
4 print(kns(lst,sum(lst)))
```

produces:

```
1 [(1, [1]), (2, [2]), (3, [3]), (3, [1, 2]), (4, [1, 3]), (5, [2, 3]), ↩
    (6, [1, 2, 3])]
2 [(3, [3]), (3, [1, 2]), (2, [2]), (4, [1, 3]), (5, [2, 3]), (1, [1]), ↩
    (6, [1, 2, 3])]
3 [(6, [1, 2, 3]), (5, [2, 3]), (4, [1, 3]), (3, [3]), (3, [1, 2]), (2, ↩
    [2]), (1, [1])]
```

- The list [1,2,3] has partial sums [1],[2],[3],[1,2],[1,3],[2,3],[1,2,3].
- The tuples are [(1, [1]), (2, [2]), (3, [3]), (3, [1, 2]), (4, [1, 3]), (5, [2, 3]), (6, [1, 2, 3])].
  - if sorted by k=0, then the order is [(1, [1]), (2, [2]), (3, [3]), (3, [1, 2]), (4, [1, 3]), (5, [2, 3]), (6, [1, 2, 3])].
  - if sorted by k = 3, then you see that [(3, [3]), (3, [1, 2]) come first (the order doesn't matter, because $|3 - 3| = 0$; (2, [2]) ,(4, [1, 3]) comes next because |3-2| = |3-4| = 1; then (5,[2,3]), because $|3 - 5| = 2$ then (6,[1,2,3]) last.
  - If k = 6, then the list is reversed
- For any two or more tuples with the same sum, their relative order doesn't matter.

Here is anothe run:

```
1 print(kns([1,2,1],2))
```

giving

```
[(2, [2]), (2, [1, 1]), (1, [1]), (3, [2, 1]), (1, [1]), (3, [1, 2]), (4, [1, 2, 1])]
```

Programming Problem 4: Ksumsort

- Implement the function (assume at least one number)
- You cannot use any modules for this; however, you are free to use the `list.sort()` method from the lecture only.
- A hint: counting in binary might prove useful – you can use `bin()` for that.
- Include a docstring for the function

Problem 5: Polynomial Regression with NumPy

In this problem, you'll be modeling rock bass otolith data: the size as a function of years. The

Age (year)	Length (inches)
1	4.8
2	8.8
2	8.0
3	7.9
4	11.9
5	14.4
6	14.1
6	15.8
7	15.6
8	17.8
9	18.2
9	17.1
10	18.8
10	19.5
11	18.9
12	21.7
12	21.9
13	23.8
14	26.9
14	25.1

data is in a csv fish_data.txt. When plotted, the data appears polynomial–size three (see Fig. 1).

For this we'll use numpy's polyfit function found here:
`https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html`
You're responsible for reading this document to understand how to use it. To be specific, we have data D and a model

$$f(x) \quad = \quad a_0 + a_1 x + a_2 x^2 + a_3 x^3 \tag{1}$$

From D, we compute the "best" values of a_0, a_1, a_2, a_3. We saw in the previous problem we can use bounded loops to find these values, but there are a myriad of packages and modules available for us to use. Numpy has a polyfit function that takes the data $D = [(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)]$ as a list of x values and y values and returns an array of a_0, a_1, a_2, a_3. We can use these with numpy to build a function called poly1d( ). In this problem you'll need to:

1. Read the data from the file
2. Read about the package from the url provided and build the graph

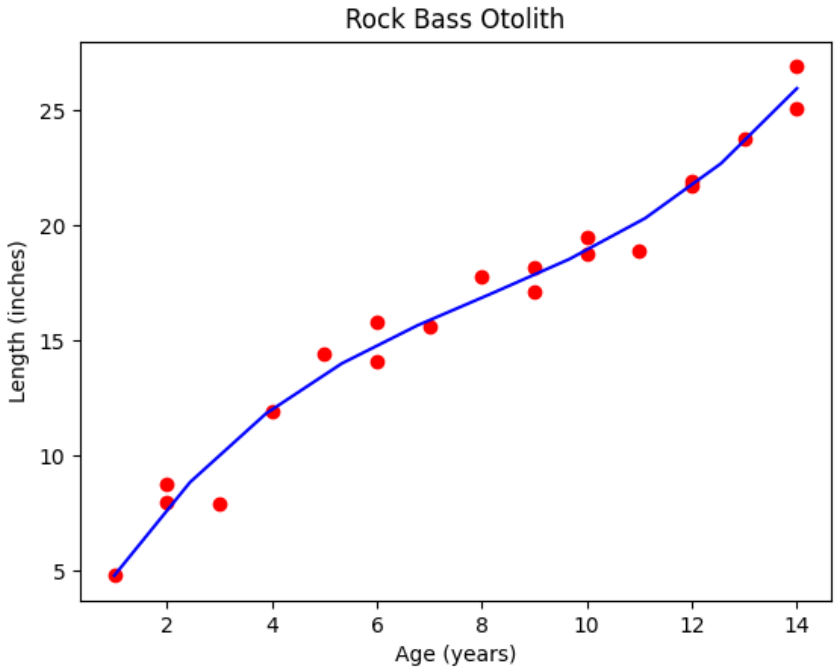


Figure 1: Plots of rock bass data (red) and model (blue). We use numpy's polyfit to do a regression on a polynomial of degree three.

```
1  def get_fish_data(path,name):
2      pass
3
4  #INPUT two lists
5  #RETURN a polynomial function degree 3
6  def make_function(X,Y, degree):
7      pass
8
9  X7, Y7 = get_fish_data(path,name)
10 data4 = [[i,j] for i,j in zip(X7, Y7)]
11 print(data4)
12
13 plt.plot(X7, Y7,'ro')                #plots data D in red
14
15 xp7 = np.linspace(1, 14, 10) #makes input for model
16 p3 = make_function(X7, Y7)  #builds model from data
17 plt.plot(xp7, p3(xp),'b')  #plots  model in bule
18
19 plt.xlabel("Age (years)")  #visualization elements
20 plt.ylabel("Length (inches)")
21 plt.title("Rock Bass Otolith")
22 plt.show()
```

gives output:

```
[[1, 4.8], [2, 8.8], [2, 8.0], [3, 7.9], [4, 11.9], [5, 14.4], [6, 14.1], [6, 15.8], [7, 15.6], [8, 17.8], [9,
18.2], [9, 17.1], [10, 18.8], [10, 19.5], [11, 18.9], [12, 21.7], [12, 21.9], [13, 23.8], [14, 26.9], [14, 25.1]]
```

and the plot.

Programming Problem 5: Polynomial Regression with NumPy

- Please read the documentation–please do the small examples shown.
- Complete the functions.
- You should plot the data first (red dots) without plotting the function (blue line) to see if the data is correct.
- 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.
- Unlike the previous file, this has a so-called *header*. Headers are useful, but in the simplest settings can be discarded. Please refer to:
  `https://docs.python.org/3/library/csv.html`
- Include a docstring for each function you implement.

Problem 6: 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:

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{2\epsilon} \tag{2}$$

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

$$f'(x) \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon} \tag{3}$$

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

$$f(x) = x^2 - 3x \tag{4}$$
$$f'(x) = 2x - 3 \tag{5}$$
$$f'(2) = 4 - 3 = 1 \tag{6}$$

```
1  def derivative(f,epsilon):
2      pass
3
4  def f(x):
5      return x**2 - 3*x
6
7  data = 2
8  epsilon = 10e-8
9  print((derivative(f,epsilon)(data)))
10 f_prime = derivative((lambda x:x**2-3*x),epsilon)
11 print(f_prime(data))
```

has outputs:

```
1 2.9999999906493713
2 2.9999999906493713
```

Assume you have a real-valued function $f(x)$ and you want to find an optimal value; namely, you want to find for $x \in [a, b]$ the value that makes $f(x)$ the smallest. While we can't develope the full explanation here, we show that if you have an approximation x_i, then this rule will usually work:

$$x_{i+1} \leftarrow x_i - \eta f'(x_i) \tag{7}$$

The $0 < \eta < 1$ is the *learning rate*, and this is multiplied by the derivative at that point. Intuitively, you're moving down toward the smaller value, the derivative indicates by how much.

Here's a simple example of using AI to find the mean. Assume we have a list of numbers $[x_1, x_2, \ldots, x_N]$. We know how to analytically find the mean, but we can use AI and the rule above:

$$x_{i+1} \leftarrow x_i - \eta f'(x_i) \tag{8}$$

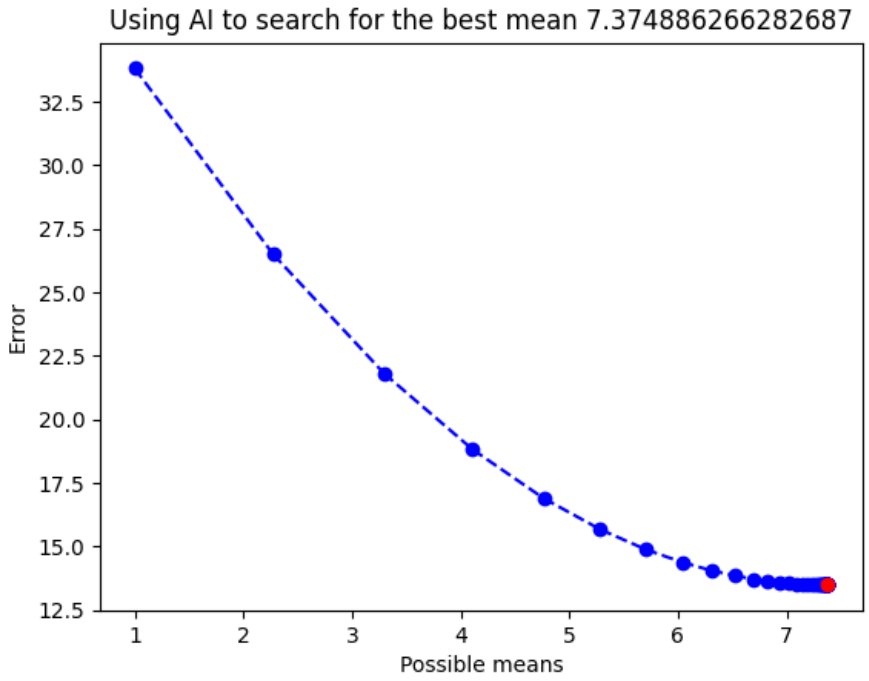
$$f(x) = (\tfrac{1}{2}) \sum_{i=1}^{n} (x_i - x)^2 \tag{9}$$

The (1/2) makes the problem easier and can be ignored. Using our computational approximation of derivative, we can write the rule as:

```
1    def update(w,data):
2         eta = .2
3         return w - eta*(derivative(lambda x:residuals(data,x),0.00001)↩
              (w))
```

We used λ to allow us to send the list of numbers as an additional argument. It can be ignored. But observe we're using the rule. The code below

```
1     N = 50
2     x = np.linspace(1,14,100)
3     gm = np.zeros(N)
4     r = np.zeros(N)
5     def mean(lst):
6         s_ = 0
7         N = len(lst)
8         for i in range(N):
9             s_ += lst[i]
10        m_ = round(s_/N,2)
11        return m_
12
13     #this is eq. 30...it's also gives error in our approximation
14    def residuals(lst,m):
15        s_ = 0
16        N = len(lst)
17        for i in range(N):
18            s_ += (lst[i] - m)**2
19        m_ = (1/2)*(s_/N)
20        return m_
21    data = [1,1,2,6,10,12,13,14]
22
23    #update rule
24    def update(w,data):
25       eta = .2
26       epsilon = 0.00001
27        return w - eta*(derivative(lambda x:residuals(data,x),epsilon)↩
              (w))
```




```
28
29      m_ = mean(data)
30
31      #successively applying the update rule
32      fmean = 1
33      for i in range(N):
34          gm[i] = fmean
35          r[i] = residuals(data,fmean)
36          #print(fmean,residuals(data,fmean))
37          fmean = update(fmean,data)
38
39      print(gm[-1])
40      print(m_)
41      plt.plot(gm,r,'bo')
42      for i in range(1,N):
43          plt.plot([gm[i-1],gm[i]],[r[i-1],r[i]],'b--')
44      plt.plot(m_,residuals(data,m_),'ro')
45      plt.xlabel("Possible means")
46      plt.ylabel("Error")
47      plt.title(f"Using AI to search for the best mean {gm[-1]}")
48      plt.show()
```

produces Fig. as successive approximations to the red value which is the mean.

Programming Problem 6: Approximating the Derivative

- Complete the derivate function
- You **must** return a lambda function
- Include a docstring for the function