C200 Programming Assignment №5
Unboundled Loops

Professor M.M. Dalkilic
Edited by Geoffrey Brown
Computer Science
School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

September 28, 2021

Contents

Introduction **2**
All the Deliverables for Homework 2

Problem 1: Entropy **2**
Deliverables 3

Problem 2: Magic in Programming! **4**
Deliverables 5

Problem 3: Magic Square **6**
Deliverables 7

Problem 4: Caeser Cipher **9**
Deliverables 10

Problem 5: Radix **11**
Deliverables 12

Problem 6: Central Dogma **13**
Deliverables 16

Introduction

All the Deliverables for Homework

Complete the functions described below. You will be asked to rewrite some code and design and implement others. This homework is meant to help you improve your understanding of loops, bases, file I/O using slightly more complex problems. Remember, problem $\rightarrow$ conceptualization $\rightarrow$ logic $\rightarrow$ implementation with bugs $\rightarrow$ solution and interpretation. You will complete this before Tuesday, October 5 2021 11PM (Bloomington, IN). You will *not* turn anything in on canvas. If your timestamp is 11:00PM or greater, the homework cannot be graded. So do not wait until 10:58PM to turn it in. This homework asks you to start creating your own functions to implement the required ones. This allows you more creativity. Since we have an impending exam (next homework), we have decreased the number of problems.

We will not be accepting late homework.

Problem 1: Entropy

In this problem, we calculate what's called *Entropy*. This is related to the entropy you've seen in physics and chemistry. Entropy is used in AI and ML. Some time ago, Claude Shannon at IBM was working on making a mathematical model of communication. Imagine, you'd have a piece of text, and then his function would give a number indicating how important or significant the message was.

What Shannon observed was that importance was really the degree of how surprising the message was. So, in a sense, the message itself wasn't as important as how unexpected it was. He then realized that he could utilize probability as a measure of surprise:

Message	Entropy	Probability
not surprising	$\downarrow$	high
surprising	$\uparrow$	low

For this problem, we are only working with finite probabilities. We can think of probabilities as a list of numbers $p_0, p_1, \ldots, p_n$ such that

$$\begin{aligned} p_i &\geq 0, \quad i = 0, 1, \ldots, n && (1) \\ 1 &= p_0 + p_1 + p_2 + \cdots + p_n && (2) \\ &= \sum_{i=0}^{n} p_i && (3) \end{aligned}$$

Line (3) is usually how it's written using the $\sum$ as a shorthand for addition. We'll learn about this later, but I thought it might be interesting to see it now. Observe it looks a lot like range! You might think about what's similar and what's different. To say in words, they are a finite collection of numbers that are non-negative that sum to exactly one.

We can make a list (we'll assume the items are of the same type and immutable) into a probability. Consider $\mathrm{x} = [\text{"a"},\text{"b"},\text{"a"},\text{"c"},\text{"c"},\text{"a"}]$

1. gather the items uniquely
2. count each time the item occurs
3. find the total count
4. create a new list of the ratio of count to total

For x, we have:

1. {"a": 3, "b":1, "c": 2}
2. total count is 6
3. return [3/6,1/6,2/6] (**Please note: the fractions here are for understanding but not expecting output like that in your code**)

We still need to show you *how* to calculate entropy:

$$\begin{aligned} entropy &= -(p_0 \log_2(p_0) + p_1 \log_2(p_1) + \cdots + p_n \log_2(p_n) && (4)\\ \text{entropy}(\text{makeProbability}(\text{x})) &= -(\frac{3}{6}\log_2(3/6) + \frac{1}{6}\log_2(1/6) + \frac{2}{6}\log_2(2/6) && (5)\\ &= -(.5(-1.0) + 0.17(-2.58) + .33(-1.58)) && (6)\\ &= 1.46 && (7)\end{aligned}$$

Because of continuity arguments we treat $\log_2(0) = 0$. Python's math module will correctly state this math error, so you'll have to simply add 0 if the probability is 0.

Deliverables Problem 1

- Complete $\log\_2$ function that returns the $\log_2$ of the input
- Complete makeProbability which takes a list of immutable objects and returns a probability distribution: a list of values $[p_0, p_1, \ldots, p_n]$ such that:

  $p_i \geq 0$ for $i = 0, 1, \ldots, n$

  $\sum_{i=0}^{n} p_i = p_0 + p_1 + \cdots + p_n = 1$
- Complete entropy function
- Provide docstrings for all the functions

Problem 2: Magick

You've encountered a soothsayer named Soothy McSoothface. He will magically determine any whole, positive number you guess by asking you to perform a few operations and tell him the result. It's quite amazing. For example, he asks you your age (say you're 25). He then asks you to add fifteen, multiply by three, subtract nine, divide by 3 and tell him the number.
You say, "It's 37."
He replies, "Your age is...is twenty-five!"

You are skeptical and implement this as a Python program.

Magical Encantation
Pick a number x
Add fifteen to x
Multiply the sum by three
Subtract nine from the product
Divide the difference by three
Subtract 12 from the quotient
Hocus Pocus–that is your number

Table 1: Encantation (operations) on a secret number that yields the secret number at the end.

Ponder this a bit, and try your best. When you're done, and it works, take a look on the next page. To help you out, we'll remind you of some of the vocabulary:

Operation	Result Name
Division	Quotient
Addition	Sum
Multiplication	Product
Subtraction	Difference
Power	Exponetiation

Table 2: Vocabulary for mathematical operations.

Deliverables Problem 2

- Complete the function magick
- What does the function do?
- There is not any unit testing, since this would give the function away...
- Provide docstring for the function.

Problem 3: Magic Square

Please visit https://en.wikipedia.org/wiki/Magic_square. You'll learn about magic squares. In this problem, you'll write a Boolean function that returns True if the 3×3 square is magic, and False otherwise. For the problem here all squares are size three. In this problem, we are giving you the main function, but it is up to you how to add additional functions to make the problem easier to solve. The unit testing will only examine is_magic_square and generate_3_square.

Generating a Magic Square

We can use brute force to generate all magic squares of size three. If we look at the solutions as a simple list of numbers, we see they are permutations–unique reorderings. For example, the square below

8	1	6
3	5	7
4	9	2

can be thought of as the sequence. If we had a way to generate all the different permutations we could, if it was small enough, check for magic squares. As it turns out, for a list of n objects, there are $n!$ (read n factorial) permutations. We can find how many using Python:

```
>>> math.factorial(9)
362880
>>>
```

Writing our own permutation function is for homework in a few weeks. For now, we can use a module in Python, itertools.permutations, shown here:

```
>>> import itertools
>>> p = itertools.permutations([1,2,3])
>>> for i in p:
...     print(i)
...
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
>>> p = itertools.permutations("123")
>>> for i in p:
...     print(i)
...
```

```
('1', '2', '3')
('1', '3', '2')
('2', '1', '3')
('2', '3', '1')
('3', '1', '2')
('3', '2', '1')
>>>
```

As you can see, it takes an iterable and returns a list of tuples of all the permutations.

$$solution = 8, 1, 6, 3, 5, 7, 4, 9, 2$$

The function generate_3_squares returns a list of squares. When run, it should return:

```
[[[2, 7, 6], [9, 5, 1], [4, 3, 8]],
 [[2, 9, 4], [7, 5, 3], [6, 1, 8]],
 [[4, 3, 8], [9, 5, 1], [2, 7, 6]],
 [[4, 9, 2], [3, 5, 7], [8, 1, 6]],
 [[6, 1, 8], [7, 5, 3], [2, 9, 4]],
 [[6, 7, 2], [1, 5, 9], [8, 3, 4]],
 [[8, 1, 6], [3, 5, 7], [4, 9, 2]],
 [[8, 3, 4], [1, 5, 9], [6, 7, 2]]]
```

There are, evidently, only eight solutions (there are, one might argue, fewer–why?). As with the previous function, you might add other functions to make the solution easier.

Deliverables Problem 3

- You are allowed to use the function sum ($list$). Please learn about this function on python.org
- For the is_magic_square function, you are encouraged to develop and use other functions . For example, in a magic square, the columns must sum to the same number. Perhaps writing a get_column(c, square) where c is a number 0, 1, . . ., len(square)-1 might be useful
- use itertools.permutations to generate the solution space for magic squares that are from 1-9.
- If you run generate_3_square on 0-8, you'll find these are the solutions:

```
[[[1, 6, 5], [8, 4, 0], [3, 2, 7]],
 [[1, 8, 3], [6, 4, 2], [5, 0, 7]],
 [[3, 2, 7], [8, 4, 0], [1, 6, 5]],
 [[3, 8, 1], [2, 4, 6], [7, 0, 5]],
 [[5, 0, 7], [6, 4, 2], [1, 8, 3]],
 [[5, 6, 1], [0, 4, 8], [7, 2, 3]],
 [[7, 0, 5], [2, 4, 6], [3, 8, 1]],
 [[7, 2, 3], [0, 4, 8], [5, 6, 1]]]
```

  What does that tell you about the search space for squares of size three?
- Complete the generate_3_square function
- Provide docstrings for the functions

Problem 4: Caeser Cipher

Please visit https://en.wikipedia.org/wiki/Caesar_cipher. You will be writing code that encrpyts and decrypts using this method. Specifically, you'll be writing functions:

$$\begin{aligned} E_n(x) &= (x + n)\ mod\ 27 \\ D_n(x) &= (x - n)\ mod\ 27 \end{aligned}$$

On the Wiki page, the modulus (%) is 26, but we're using 27–why? We are adding an extra symbol { for space. Please visit https://en.wikipedia.org/wiki/ASCII. If you look at the printable ASCII characters, you'll notice that { (hex value 7A) follows z. Thus we can easily extend our cypher to include this symbol for space. Let's see how.

```
sentence = "this is a secret message about the class"
_sentence = sentence.replace(" ", "{")
print(_sentence)
es = ""
for i in _sentence:
    es += encrypt(i, 5)
print(es)

ds = ""
for i in es:
    ds += decrypt(i, 5)

o_sentence = ds.replace("{", " ")
print(o_sentence)
```

has output

```
this{is{a{secret{message{about{the{class
ymnxenxefexjhwjyerjxxfljefgtzyeymjehqfxx
this is a secret message about the class
```

In this cypher we are shifting by five. Look at the first letter 't'. Here is the shift in Python:

```
>>> ord('t')
116
>>> chr(ord('t') + 5)
'y'
>>> chr(ord('h') + 5)
'm'
```

Line one is our original sentence with { replacing space. Line two is the encrypted sentence. Line 3 the decrypted sentence. The shift is five, so we replace 't' with 'y' and 'h' with 'm'. What about { ? It starts at the beginning at ‘a’ and returns ‘e’ which is five spaces. You are free to use chr and ord in Python or you can make a dictionary. You have complete control on how this is implemented.

What the arguments to encrypt and decrypt? $E_n(x)$ and $D_n(x)$ take two parameters each. The letter and the amount of shift. Observe we retain the shift for all calls to both encrypt and decrypt.

Writing encrypt and decrypt might be easier if you use a dictionary–but that’s entirely up to you.

Deliverables Problem 4

- Complete $\mathrm{encrypt}$ and $\mathrm{decrypt}$. We use { to encode space
- You can use $\mathrm{replace}()$
- a dictionary will likely make the functions more easily implemented
- Provide docstrings for the functions

Problem 5: Radix

In this problem you use a data structure that allows extended functionality with bases. For any sequence of digits $d_n d_{n-1} \cdots d_0$ in a base b, we will use this structure:

$$\begin{aligned} \textit{wild number} &::= [\langle \textit{string} \rangle, b] \\ \langle \textit{string} \rangle &::= "d_n d_{n-1} \cdots d_0" \end{aligned}$$

where b is the base and the string is a string of digits. We will call our numbers wild numbers (WN). For example,

```
['101', 2]
['100', 2]
['5', 10]
```

are three WNs. The first is in binary (it's equivalent to 5_{10}), the second is binary (it's equivalent to 4_{10}) and the third is decimal 5. You will implement several functions whose use is shown here:

```
n1,n2 = 5,4
base2, base10 = 2,10

x1, y1 = make_number(n1,base2), make_number(n2,base2)
print(x1,y1)
print(convert(x1,base10))
print(add_(x1,y1,base10))
print(add_(x1,y1,base2))
print(convert(add_(x1,y1,base2), base10))
print(mul_(x1,y1,base2))
print(convert(mul_(x1,y1,base2),base10))
```

the output is

```
['101', 2] ['100', 2]
['5', 10]
['9', 10]
['1001', 2]
['9', 10]
['10100', 2]
['20', 10]
```

The function make_number(n,b) takes an actual integer n (Python) and base b (integer greater than one) and creates a WN. The function convert(WN,b) converts an existing WN into a new base. In this example, convert(['101', 2], 10) $\rightarrow$ ['5', 10]. This is true, since this is an encoding of |||||. The function add_ takes two WN and base and returns the sum as a WN in that (possibly new) base. The function mul_ takes two WN and a base and returns the product as a WN in that (possibly new) base. The only Python int function you are allowed to use is int(s,b) where s is a string of digits and b a base. Here is a quick session to remind you of its function:

```
>>> int('101',2)
5
>>> int('100',2)
4
>>> int('1001',3)
28
```

Deliverables Problem 5

- Complete the functions make_number, convert, add_ and mul_
- You will need to use aside from % and //
- The only Python function you can use is int(x,b)
- Provide docstrings for the functions

Problem 6: Central Dogma

The central dogma in biology is that DNA → RNA → protein. Please visit https://en.wikipedia.org/wiki/Central_dogma_of_molecular_biology. In this problem you will read in a two files: one that has how to translate three bases of DNA (codon) to an amino acid and one that has DNA.

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 name of the amino acid. The second column is the one letter initial. For the Stop_codons, we use a dash. The remaining columns 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.

A FASTA file has two parts: a header (information about the sequence) and the sequence itself. Here's the one you'll be using:

>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
```

You can read about this gene here: https://pubmed.ncbi.nlm.nih.gov/3422341/. The first line describes the sequence providing the name and other attributes. The remaining lines are the DNA sequence (ignore all whitespace).

Translating DNA into a protein

To convert from DNA to protein, we use a sequence of codons.

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, CCACTGCACTCA
4. We find it matches Leucine, L, CTT, CTC, CTA, CTG, TTA, TTG. So our protein is PL.
5. The next three are CAC CCACTGCACTCA.
6. The table has Histidine, H, CAT, CAC. We extend our string to PLH.

7. The final three are TCA. CCACTGCAC<u>TCA</u>
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, so you ignore them. Suppose we had CCAC. We know CCA is P. Then we only have C left. We ignore it.

In this problem, you'll read in the first table 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. Your task is to take the DNA and produce a string of single letters that reflect the encoding.

The function get_amino_acid takes a file path and returns a dictionary. This is global, since all translations use the same code. The function get_DNA takes a file path and returns a list [header, DNA] (FASTA data structure) where header is the first line of the file and DNA is a string composed of all T,C,G,A characters (ignoring any whitespace). The function translate takes a FASTA data structure and returns a string that is the translation using the dictionary. In this problem we have (awkwardly) assigned the variable actual the string that is the correct translation. We can simply print to see whether our translation is the same as actual.

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

```
print("Dictionary")
print(aa_d)
print("FASTA file")
print(DNA_d)
print("Translations match:", str(protein == actual))
```

has output:

```
Dictionary
{('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', '-']}
FASTA file
['>HSGLTH1 Human theta 1-globin gene',
 'CCACTGCACTCACCGCACCCGGCCAATTTT
TGTGTTTTAGTAGAGACTAAATACCATATA
GTGAACACCTAAGACGGGGGGCCTTGGATC
CAGGGCGATTCAGAGGGCCCCGGTCGGAGC
TGTCGGAGATTGAGCGCGCGCGGTCCCGGG
ATCTCCGACGAGGCCCTGGACCCCCGGGCG
GCGAAGCTGCGGCGCGGCGCCCCCTGGAGG
CCGCGGGACCCCTGGCCGGTCCGCGCAGGC
GCAGCGGGGTCGCAGGGCGCGGCGGGTTCC
AGCGCGGGGATGGCGCTGTCCGCGGAGGAC
CGGGCGCTGGTGCGCGCCCTGTGGAAGAAG
CTGGGCAGCAACGTCGGCGTCTACACGACA
GAGGCCCTGGAAAGGTGCGGCAGGCTGGGC
GCCCCCGCCCCCAGGGGCCCTCCCTCCCCA
AGCCCCCCGGACGCGCCTCACCCACGTTCC
TCTCGCAGGACCTTCCTGGCTTTCCCCGCC
ACGAAGACCTACTTCTCCCACCTGGACCTG
AGCCCCGGCTCCTCACAAGTCAGAGCCCAC
GGCCAGAAGGTGGCGGACGCGCTGAGCCTC
GCCGTGGAGCGCCTGGACGACCTACCCCACG
CGCTGTCCGCGCTGAGCCACCTGCACGCGTG
CCAGCTGCGAGTGGACCCGGCCAGCTTCCAG
GTGAGCGGCTGCCGTGCTGGGCCCCTGTCCCC
GGGAGGGCCCCGGCGGGGTGGGTGCGGGGGG
CGTGCGGGGCGGGTGCAGGCGAGTGAGCCTTG
AGCGCTCGCCGCAGCTCCTGGGCCACTGCCTGC
TGGTAACCCTCGCCCGGCACTACCCCGGAGACT
TCAGCCCCGCGCTGCAGGCGTCGCTGGACAAGT
TCCTGAGCCACGTTATCTCGGCGCTGGTTTCCGA
GTACCGCTGAACTGTGGGTGGGTGGCCGCGGGA
TCCCCAGGCGACCTTCCCCGTGTTTGAGTAAAGC
CTCTCCCAGGAGCAGCCTTCTTGCCGTGCTCTCT
CGAGGTCAGGACGCGAGAGGAAGGCGC']
 Translations match: True
```

Deliverables Problem 6

- We have edited to the print output to make it more easily viewed
- Complete the functions
- Provide docstrings for the functions
- You are only allowed to use replace for space and “{‘. You cannot use it otherwise. This might seem strange, but unfortunately, since some of the abbreviations are C,G,T,A, this approach will become difficult.