Computing in Context

An Introduction to Computer Science & Programming for Nonmajors

Document Contents

Chapter 0: Introduction and Review

Chapter 1: What is Computer Science?

Chapter 2: Writing Algorithms

Chapter 3: Welcome to Python

Chapter 4: Python Essentials I

Chapter 5: Python Essentials II

Chapter 6: Python Essentials III

Chapter 7: Files and JSON

Chapter 8: Introducing Python Libraries

Chapter 9: Pandas

Chapter 10: Numpy

Chapter 11: MatPlotLib

Chapter 12: Conclusion

Chapter 0: Introduction and Review

Hello everyone and welcome to COMS 1002 Computing in Context, I will be one of your Teaching Assistants for the duration of the semester. My name is Griffin and I will be writing this guide today! Please note that this should be used as a supplement to the lectures, while I will ideally cover everything relevant in this document, I am going off memory of assisting last fall so some things I will miss. There are plenty of contexts so I will not be covering this course from a context specific manner, in fact this document will be more like if you took ENGL 1006 Intro to Computing for Engineers and Applied Scientists, but more toned down to what would be expected of you.

That being said, there is some things you should be comfortable with and aware of as you begin this course, this especially true for those who intend to do the economics track so I am just going to list those topics and whenever you have the free time you may go and review these topics if you feel it necessary:

1. Solving Linear Equations
2. Properties of Exponents (Specifically relating to powers of 2)
3. The Powers of 2: 1,2,4,8,16,32,64,128,256,512,1024
4. Summations (Σ) (economic specific I believe)
5. Basic Probability Theory (you will be taught what you need to know if/when it is necessary but just in case you want a head start)
6. Time Management Skills: chances are the skills you had in high school if you are a first year student won't cut it in college (believe me you'll get swarmed real quick) be sure to start assignments early

Be sure to attend lectures and ask any questions you have on ED!

Chapter 1: What is Computer Science?

I'll start with a synopsis and then go more in depth. Computer Science is the study of algorithms and their properties. Think of it like this, what is the process for eating food everyday, well to put it in simplest terms possible there are really three steps to eating food:

1. Acquiring the food
2. Eating the food
3. Cleaning up your mess

For now I will refer to this as a routine, but it really is just synonymous with an algorithm. It is possible that within a routine there are subroutines. For example there may be a subroutine in your quest to acquire food. When we consider the properties we primarily consider its time and space complexities as its primary properties, along with the inputs and outputs. We will only focus on the inputs,outputs, and the time complexity. Discussing the inputs and outputs is easy, if we consider our routine a giant black box then we get the following diagram:

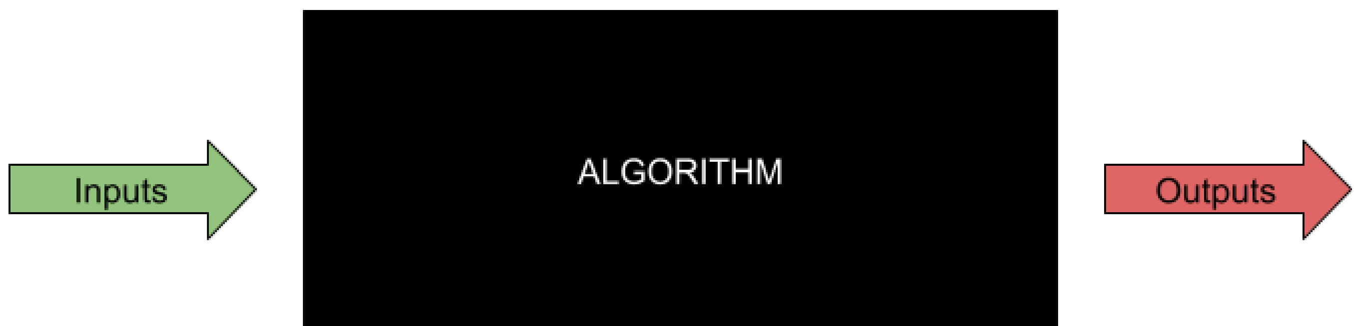

Image 1: A diagram representing the inputs and outputs of an algorithm

This will become more clear later but keep this image in mind when we discuss writing algorithms in the next chapter. The time complexity

essentially means approximately how long does it take to run? We typically use Big-O notation to represent this kind of information, if we wanted to demonstrate something that runs in linear time then we would write O(n) where n is the size of our input. The smallest possible complexity is O(1) this means that regardless of the size of the input the algorithm runs in the same amount of time. In a perfect world all algorithms would be O(1) but this is not the case and actually there are some problems that do not even have polynomial time solutions either. We won't concern ourselves with that concept for now but we will now transition ourselves to writing some introductory algorithms, first in pseudocode and later on in python.

Chapter 2: Writing Algorithms

Let’s start with the simplest algorithm possible, say we have a collection of items, let's call this collection of items A, say that the following is true:

$$A = \{3,2,6,1,9,4\}$$

Now say we wanted to determine if a particular integer with the value x was present in our collection. Well we have a pretty simple solution to this problem, we can just search through the collection, A, item by item and see if our item x is present, if it is we can either return true to denote the item is in the collection or its position in the collection. I will choose the latter approach for this. This algorithm is known as linear search. We can write this algorithm which is a healthy mix of actual code and English, we will follow a python-like pseudocode with the exception that in pseudocode, indexing starts at the value 1.

```
def linear_search(A, x):
    idx = 1
    list_size = length(A)

    while(idx <= list_size):
        if(A[idx] is x) then :
            return idx
        otherwise:
            idx = idx + 1

    return -1
```

The traditional choice is to return -1 whenever the item is not present in the collection. As we can see we start a variable at the value of one and go until we are out of elements to look through that is what the loop is doing inside the loop we check to see if the item at the current position of the collection is the element we want then we return the position we found the element at, otherwise we increase the position by 1.

Perhaps from the name of the algorithm it can be implied that the algorithm grows linearly, when it comes to searching for elements we can do better if we make a certain assumption. Let's assume that the collection is already in ascending order. So take our initial collection A and sort it yielding the following collection:

$$A = \{1,2,3,4,6,9\}$$

Now we can take advantage of the collection by starting in the middle and effectively chop off half the collection with each iteration depending on the value x we are looking for. Consider the following algorithm:

[Continue To Next Page]

```
def binary_search(A, x):
    begin = 1
    end = length(A)
    while(begin <= end):
        mid = begin + (end - begin)//2
        if(A[mid] is x) then :
            return mid
        if(A[mid] > x) then:
            end = mid - 1
        otherwise:
            begin = mid + 1
    return -1
```

A lot of this looks similar to the linear search example, except this time we use three values to keep track of the effective state of our algorithm. We use "begin" and "end" as two markers that indicate the beginning and end of our effective array. Let's run through an example where the value we are looking for is not in the list to show off the entire algorithm. Say we want to find x = 10

```
A = {1,2,3,4,6,9}
     ^         ^
     b         e
```

We start by calculating the midpoint which would be 1 + 5//2 = 3:

```
A = {1,2,3,4,6,9}
     ^   ^     ^
     b   m     e
```

We see that x is greater than 3 so we move our b marker one position past the m pointer and continue and repeat

```
A = {1,2,3,4,6,9}
          ^ ^ ^
          b m e
```

Same thing again 10 is greater than 6 so we move the b marker one position past the m pointer and continue and repeat

```
A = {1,2,3,4,6,9}
              ^
            b/m/e
```

We see for a final time that 10 is greater than 9, at this point b > e meaning the loop ends and we return -1 as the result.

As we can see binary search chops the effective collection in half each time, meaning that the algorithmic complexity is logarithmic (since logn is the opposite of 2^n) in the realm of Computer Science all the following terms are equivalent: lgn = logn = $\log_2 n$

Those two algorithms - linear and binary search - were a subset of algorithms known as search algorithms, we will now move on to sorting algorithms in this case two: Selection Sort and Insertion Sort. To summarize these algorithms up quickly, selection sort *selects* the smallest element in the collection and swaps it with the current element you are on (starting from the first and ending on the last). Insertion sort effectively divides the collection into a sorted partition and an unsorted partition, it inserts the current element in the unsorted partition into its proper position in the sorted partition thus with each iteration the sorted part grows and the unsorted part shrinks.

Here we show off the pseudocode algorithms for the two sorting algorithms:

```
def selection_sort(A):
     for each item x in A:
          min = x
          for each item y in A:
               if(y < x) then:
                    min = y
          swap(x,min)

def insertion_sort(A):
     for i in range(2, length(A)+1):
          copy_elm = A[i]
          j = i - 1
          while(j >= 1 and copy_elm < A[j]):
               A[j+1]=A[j]
               j = j - 1
          A[j+1] = copy_elm
```

If you read the descriptions I provided above then this should make some nice sense to you but in case it doesn't let's quickly run them down.

Selection Sort: We iterate through each item in the collection one main time and then for each item we repeat that loop, doing this allows us to properly compare the elements in hopes of finding the smallest element, by the end of that inner loop we know what the smallest element is and then we perform a swap before moving on in the outer loop. For those who are still confused, here is a nice visualization of the algorithm (Selection Sort visualize | Algorithms | HackerEarth)

Insertion Sort: Starting with the second element in the collection we will continually swap with preceding elements until the collection is sorted, for each preceding element this occurs. If you would like to see a visualization of this algorithm follow this link (Insertion Sort visualize | Algorithms | HackerEarth)

Now that we have covered the basic searching and sorting algorithms we can now formally introduce Python. A lot of this will look quite reminiscent of the pseudocode we have been writing for the last little bit, and that was done intentionally to help ease into it!

Chapter 3: Welcome to Python

Now we can finally start some actual programming! We will start by writing a simple but an absolute classic program. After setting up your programming environment you should be able to open some IDE of your choice and type the following in your editor:

```
print("Hello World!")
```

This introduces us to two things in python:

1. Prebuilt Functions
2. Prebuilt Data Types

We will be writing user-defined functions and types later but for now we will stick to the prebuilt ones that help us get what we need done. In this case `print` is the function and `"Hello World!"` is the argument which is of type `str`. We can't continue programming by using hard coded values so now we will introduce variables take the original program we wrote above, we can add variables to it easily:

```
some_text = "Hello World!"
print(some_text)
```

The first line initalized the some_text variable to the string "Hello World!" and then we subsequently printed it out, well we are able to reassign variables and change their values like so:

```
some_text = "Hello World!"
print(some_text)
some_text = "Today is a wonderful day!"
print(some_text)
```

If you plug this into the editor and run the code you should see the statements appear on separate lines. There are plenty of different types of variables, we have seen Strings but there are also integers, floating point numbers, booleans and more here are some examples of these types:

```python
my_int = 4
my_float = 3.14
my_str = "1002"
my_bool = True
```

Those are the four basic types and the main types you will be working with throughout the semester. Now that we know about the built in types, how about we talk about some more built in functions.

We can convert between types with the proper associative function, for example we can turn my_int into a float with the following:

```python
my_second_float = float(my_int)
print(type(my_second_float))
```

We can also use functions like: int(x), str(x), bool(x) and others to do what you can probably guess they do by now. The type(x) function simply returns the type of x, in the case of the code above that will be float.

Finally - for now at least - we need to talk about the input(x) function. Taking in user input is an important part of writing programs, especially in this *context* (hehe what a fun pun) the following program asks a user to input how many days are in a week:

```python
days = input("How many days are in a week? ")
print(days)
```

There is one important detail to mention here, the type of days is actually a string, if we attempt to perform any mathematical operation then we will receive an error from the Python Interpreter. If we wish for the type of days to be an integer for example then we can do so by adding on this tiny bit of code

```python
days = int(input("How many days are in a week? "))
print(days)
```

In the next chapter we will discuss these mathematical operations and others that allow us to manipulate our variables in new and fun ways!

Chapter 4: Python Essentials I

An essential part of Python is to perform mathematical operations. We have the basic cardinal operations (+ - * /) There is however more details than meets the eye, take the '/' symbol. If we perform the operation 5 / 2 we will get 2.5 however if we perform the operation 5 // 2 then we will get 2. The singular backslash represents floating point division while the double backslash represents integer division.

Other programming languages tend to make exponentiation a convoluted process to tackle. In Python we are fortunate to have the ** operator to perform exponential operations like 5 ** 2. Finally we have the % operator. To put simply a % b gives the user the remainder of a divided by b so 5 % 2 gives us 1. There are more in-built mathematical operations that you can learn more about on your own and some may even be mentioned in lecture but to get you started you can look here: (Python Math (w3schools.com))

Now we get to some of the more formal programming structures in Python. Prior to continuing it is important to learn about boolean expressions. Earlier we discussed briefly the data type bool in Python but now we go more in depth. A Boolean expression is any expression in Python that evaluates to True or False take the following expression into consideration:

```
x == 5
```

If we assume there is a variable named 'x' that is of an integer type then the expression is valid the expression either evaluates to True (when x equals 5) otherwise it evaluates to False. We will be using these boolean expressions for conditional statements on our conditionals and some of our loops which is why you need to know about them and how they work. We can combine multiple boolean expressions into more complex expressions using the 'and' and 'or' keywords. With 'and' the expression evaluates to True if and only if both expressions surrounding the 'and' are True. With 'or' the expression evaluates to True if at least one of the expressions is True.

Knowing this we can now discuss conditionals, to put it simply, a conditional is a block of code that will only execute if the

condition is True. Imagine we are looking to describe a student's letter grade in some course, we can use the following code to do this:

```
grade_average = 86
if(grade_average >= 90):
    print("A")
elif(grade_average >= 80):
    print("B")
elif(grade_average >= 70):
    print("C")
elif(grade_average >= 60):
    print("D")
else:
    print("F")
```

We can see that the basic conditional has an initial condition surrounded by the if keyword we then use the colon to indicate that we are opening a new code block and then inside that code block (which is indented by one level) we describe what we wish to do. If we want to add more conditionals and we desire only for one of the conditionals to execute we can wrap the other conditions in elif (also known as else if in other languages) if we wish to just have some action not be dependent on a specific condition and we wish to treat it is as a sort of final option we can use else like you see above. It is important to note that you are able to have some of these components on their own or without others the following are valid subsets of the above that you can legally do without having an interpreter error:

if -> elif -> else
if -> elif
if -> else
if

Any other configuration that you attempt to write will not be accepted. This is will be the end of my discussion of conditionals but Python does allow for match-case statements as well you can read about those here: (Python Match-Case Statement - GeeksforGeeks)

We continue onward to discuss loops. Loops allow us to repeat blocks of code for a certain number of iterations or until a predetermined condition is reached. For purposes of 1002 there are only two kinds of loops we should concern ourselves with, the while loop and the for loop. The while loop continues executing a block of code until the defining condition is reached. Say I wanted to print the even numbers in the range [1,100] I can do that in the following two ways:

```
number = 1
while(number <= 100):
     if(number % 2 == 0):
          print(number)
     number += 1 #logically equivalent to number = number + 1

number = 2
while(number <= 100):
     print(number)
     number += 2 #logically equivalent to number = number + 2
```

These code segments demonstrate that there are multiple ways to achieve a desired goal with your code along with the standard functionality of a while loop. Take note at the last line in each loop, please do not let that syntax confuse it, that is a kind of compound assignment operator which allows me to write the equivalent expression in the comment more succinctly. (If I seem to be more brief on a subject than you wish please do not worry as my weekly lecture notes will be more in depth since I will be covering these topics again anyway)

Now all that is left to discuss is the for loop, there are two different kinds of for loops in Python. I will touch on the first kind in this chapter but I will bring up the second one in the next chapter when we discuss lists as they are linked together quite nicely.

We are able to loop through a series of numbers in the following way, I will first give it to you generically and then following it up with an example:

```
for i in range(start, end, step):
    //some code
```

You are permitted to omit the start and step parts of the range; this will simply just default step to 1 and start to 0. It is important to note that the range is exclusive, meaning that i will be able to represent values in the range [start, end) rather than [start,end] which you may have expected. Consider the following example, say we wished to print all values x that are multiples of 3 in the range [1,100]. You could use a while loop like we discussed earlier or you can make use of the for loop which will allow us to write even less lines of code:

```
for x in range(3, 100, 3):
    print(x)
```

See in just two lines of code we did the task that was asked of us whereas with a while loop the same task would've taken twice as many lines of code. Here is a fun little challenge for you to try to see if you are understanding these loops, create a program that prints even numbers in the range [1,100] but you need to start at 100 and work backwards, do this with both a while loop and a for loop!

Chapter 5: Python Essentials II

Now that we have covered the basic control and looping structures in Python, we can now move on to some basic data structures in Python that are necessary to know moving forward. Say we wish to store a bunch of variables, well we could do that by declaring a bunch of variables and then assigning them values whenever we need to and then use them for future operations but this is inefficient and will simply clutter up your code. Here we will introduce lists, there are two ways to create lists in Python and they are shown below:

```
my_first_list = ["hey", 1002, "is", "cool", True]
my_second_list = list(("this","is","not", 1004))
```

The first method uses the standard bracket notation to contain the items of our list while the second uses the list constructor to create a list. In order to access elements of the list we use the list name along with the bracket notation sandwiching the index we want. In Python, indexing begins with 0. The following are all valid ways to get indices for the first list:

```
print(my_first_list[0]) #get the first element
print(my_first_list[len(my_first_list)-1]) #get the last element
print(my_first_list[-1]) #also gets the last element
```

We can append elements using the append(x) method and we can remove elements using the remove(x) method. As shown in the example above we can get the length of a list using the len(x) function there are plenty of other things you can do with lists in regards to these methods which you can explore more and read about here ([Python Lists (w3schools.com)]) to summarize lists though, list items are ordered, changeable, and allow duplicate values.

Now that we know about lists, we can revisit the for loop and discuss the second kind of for loop that Python has to offer, say we only desire to view the items within a list and we do not care for the position of that particular item. Well Python offers an easy way to do this, consider the following code segment:

```
my_watch_list = ["Suits", "The Wolf of Wall Street", "Silicon Valley",
"Demon Slayer"]

for x in my_watch_list:
      print(x)
```

This for loop simply iterates through the collection of items and prints them out if we use the previous type of for loop we would have done the following:

```
for x in range(len(my_watch_list)):
      print(my_watch_list[x])
```

See this way involves a lot more typing but the job gets done but just look at how neater the first approach is. This offers a new way of creating lists, consider our initial watch list from above and say we desire to have a new watch list containing entries in the original that have titles longer than 5 characters. Well we are able to do this using stuff we have already learned:

```
new_list = []
for x in my_watch_list:
  if len(x) > 5:
    new_list.append(x)
```

This is easy to understand but it requires a decent amount of code, here is an alternative approach using a concept known as list comprehension:

```
new_list = [x for x in my_watch_list if len(x) > 5]
```

This code contains 4x less lines as the previous example, is still reasonable enough to read and completes the same job as the previous code segment. You will most likely be asked to perform a task throughout the course that asks you to utilize list comprehension, the syntax varies slightly depending on the exact conditional you wish to use so I urge you to read more on the subject here: (Python - List Comprehension (w3schools.com)).

Diverting away from these concepts real quick we need to discuss how we organize our code, up to this point we have simply just written code on the console and executed it as we please without any extra effort. Well moving forward we will be writing user defined functions to contain the code we write, a major benefit of this is that if we need to run the same segment of code in multiple places we are able to do that without rewriting the code. To do this we will use the following format:

```
def my_function(param_1, param_2):
    #some code
    #example
    return param_1 + param_2
```

All of our functions will follow this format, the keyword 'def' then the function name and after an opening parenthesis we will have an parameter list, which could contain nothing but in the example above contains two parameters with each parameter separated by a comma. Once we finish the parameter list we put a closing parenthesis and then a colon and then indent the following lines we wish to associate with the function. Any variable we define within the function does not exist outside the function, this is known as scope. Specifically when we define a variable within a function that variable has local scope with respect to that function. Variables declared outside your

functions have global scope meaning you can reference them from anywhere.

Going back to actual Python we will wrap up this chapter with discussions on Tuples and Sets. Beginning with Tuples, they are essentially immutable lists, once we create a tuple we cannot add or remove elements from it but we are able to modify them if we so choose. Consider the following code segment that demonstrates Tuples:

```
def my_tuple_creator(a,b,c):
    my_tuple = (a,b,c) # = tuple(a,b,c) is equivalent
    return my_tuple
```

This code demonstrates how we can create tuples, we can modify the values in the same way as we do for lists since both are indexed, whenever we want to return multiple values from a function we are really returning a tuple. This leads to a concept known as unpacking, where we can automatically assign values from a tuple (or list but it is more relevant here) without doing the extra work, if it is not obvious from the example below then convince yourself that the two implementations are equivalent:

```
a,b,c = my_tuple_creator(x,y,z) #asssume x,y,z exist somewhere
#here is the alternative
t = my_tuple_creator(x,y,z)
a = t[0], b = t[1], c = t[2]
```

The first of these two options is much cleaner than the second. All operations that apply to lists apply to tuples as long as you aren't attempting to mutate the tuple. To read more about tuples view the following resources: (Python Tuples (w3schools.com))

Finally for the chapter we get to sets, which are unordered lists which can only contain distinct elements. You are able to add and remove items from sets but the items themselves cannot be changed once you add them. We can visualize sets with the code below:

```
def my_set_creator(a,b,c):
      my_set = {a,b,c} # = set(a,b,c) is equivalent
      return my_set
```

Sets are not indexable so you have no idea which element will come in what order as it is not guaranteed to be the same order as which you created it. An important thing to consider is that in Python what sets consider as duplicates may not be what you think, for example True and 1 would be considered duplicates despite being completely different types. For more reading on Python sets please see the following resource: (Python Sets (w3schools.com))

The following table shows important functions/methods and which data structure(s) they could be applied to

Function/Method	Description	Data Structure
len(x)	Returns the length of x	Lists, Tuples, Sets
append(x)	Appends x to the end of the collection	Lists
remove(x)	Removes x from the collection	Lists, Sets
add(x)	Adds x to the collection	Set

Next chapter we will discuss Strings, dictionaries and classes in Python.

Chapter 6: Python Essentials III

We briefly discussed Strings in an earlier chapter when discussing Python's built in data types. There is actually more to it than other data types which is they are getting their own little section.

We can make Strings with either the double quotes like this: "hello" or with single quotes like this: 'world' if you need a String that spans multiple lines you will use three quotes take the following as an example:

```
message = """I hope you are finding this guide useful,
your feedback will help make this guide for future
iterations of 1002, congratulations for making it
this far, you are about halfway through!"""

print(message)
```

This is much neater and an overall better coding practice than simply putting the quote message in the print statement.

Another common thing with String formatting is the use of String interpolation commonly known as "f-strings" this allows us to create more modular and dynamic Strings in Python. Say we want to print a message to a user upon entering our system. Let's assume we have the user's name stored in a variable appropriately named 'name' we can use f-strings to format the string as follows:

```
name = "Griffin"
print(f"welcome to your dashboard {name}.")
```

The label you put inside the curly braces must match the variable name you wish to associate with. To learn more about "f-strings" please view this resource: (f-strings in Python - GeeksforGeeks)

Going back to regular details about Strings, we are able to index them like we can with lists since a String effectively is a list of characters, we are actually able to iterate through the individual characters using a for loop:

```python
for letter in "Computing in Context":
     print(letter)
```

Trivially, we can replace the String literal with a variable for convenience. Finally we can also use the len(x) function to determine the length of a String. To read more about Strings and their properties and operations you can perform please view the following resource: (Python Strings (w3schools.com))

The final data structure we will talk about is the dictionary (also referred to as a HashMap in other languages) besides this data structure being the literal god sent for technical interviews (when in doubt use a hashmap is the saying there) dictionaries are very useful for storing related pieces of information. Saying we needed a quick and efficient way to find the location of a particular product in a warehouse knowing just a product id (assuming the necessary backend already exists). Well we could use a list or other such data structure but the access time is on the order of O(n) so for a warehouse storing millions of items, this is costly. Dictionaries have O(1) access time assuming we know the key. This is their main benefit. The following demonstrates how we can use and modify a dictionary:

```python
my_list = [1,3,7,4,2,9]
target = 5

"""
Say our goal is to find a matching set of unique values
from the above list that is equal to the value of target.
For simplicity let's assume there is always a valid solution
in particular we want the indices of those values.
"""

my_dict = dict() #using {} also works too
for i,x in enumerate(my_list):
     if(x in my_dict):
          return (i, my_dict[x])
     else:
          my_dict[target-x] = i
```

The above code maps the difference between the target and the current value to the position of the current value. The reason we map the difference is because if we find the difference later in the list we know we have a matching value and thus a valid solution. To go further (and also to demonstrate the format of a dictionary) let's walk through the example I have hard coded above:

```
my_list = [1,3,7,4,2,9]
target = 5

my_dict = {}
```

We now loop through my_list and update the dictionary as follows:

```
my_dict = {4:0}
my_dict = {4:0,2:1}
my_dict = {4:0,2:1,-2:2}
```

We see that 4 is contained within the dictionary already so we return the current index along with the index stored with the key 4.

Also if you are curious, that is what enumerate does: it allows access to both the index and the element within a collection (the more you know).

For those of you who either have friends who major in Computer Science, or plan to major in it yourself you might be able to recognize this problem. This is the famous Two Sum problem that is the first problem many aspiring software engineers (including myself) solve on a site called leetcode, which is what enables us to practice for technical interviews. If you have no idea what I am talking about then bless you. While dictionaries are not traditionally indexed from 0 to len(dict)-1 they do allow for us to use the bracket notation we see with lists and the likes but instead of providing the index we provide the key. You see me use this notation for assignment and accessing in the code fragment above. If you wish to learn more about dictionaries then please view this resource: (Python Dictionaries (w3schools.com))

Python is an object oriented programming language, I personally do not ever use it as such but alas it is. What this means is that we are able to create our own data types! In reality more people call these objects but they are effectively custom data types. We use the class keyword to denote the beginning of a class:

```
class Coordinate:
      x = 10
      y = 02
```

We can now use this class to make objects as follows:

```
point_one = Coordinate()
print("The X position is: ", point_one.x)
print("The Y position is: ", point_one.y)
```

Of course this is quite specific; we want our Coordinate objects able to represent more than one coordinate pair. This is where we introduce constructors, denoted as __init__(x) it is always executed when we create objects and it is where we can define the object's attributes, consider a refined Coordinate class:

```
class Coordinate:
      def __init__(self, x, y):
            self.x = x
            self.y = y

      def __str__(self):
            return f"The Coordinate is: ({x}, {y})"

      def distanceFromOrigin(self):
            return ((self.x**2 + self.y**2)**0.5)

      def setCoordinateX(self, x):
            self.x = x

      def setCoordinateY(self, y):
            self.y = y
```

```python
    def getCoordinateX(self):
        return self.x

    def getCoordinateY(self):
        return self.y
```

This is looking much more complex now. I used the __init__(x) method above to initialize the attributes of our Coordinate object. We need to use the self keyword in any definition within the class as the object is also a parameter to its own functions and methods. The __str__(x) method returns a string that represents the object, if we attempt to print any instance of the object this function will implicitly be called.

Next I defined a method that calculates the distance between the coordinate and the origin. I then followed this by some mutator methods (methods that alter the object's attributes) and finally some accessor methods (methods that return the object's attributes to the user) now we can use this updated class:

```python
point_two = Coordinate(10, 4)

print(point_two)
print(point_two.distanceFromOrigin())
print(point_two.getCoordinateX())
print(point_two.getCoordinateY())

point_two.setCoordinateX(20)
point_two.setCoordinateY(2)

print(point_two)
print(point_two.distanceFromOrigin())
print(point_two.getCoordinateX())
print(point_two.getCoordinateY())
```

You may have noticed that I have been using the terms 'function' and 'method' but these terms are not interchangeable. A method is any function that belongs to a class, while a function doesn't belong to

a class. A better way to distinguish between them is when we call a function it is of the form function(x) while when we call a method it is of the form someObject.method(x)

This distinction is not really a big deal, no one will penalize you for using them interchangeably unless it is on an exam or during a technical interview and you are directly asked. Knowing how to build classes is useful for designing scalable code, like I said earlier I personally do not use Python as an object oriented language (in the sense that I do not make my own custom data types). I reserve that right to Java, my personal favorite language and if you end up wanting to major in Computer Science you'll need to take COMS 1004 which is a course kind of like this but in Java.

There is much more to learn in regards to Object Oriented principles in Python, some of which is out of scope for this course, to be honest I do not even remember if classes and objects were covered in 1002 last Fall. If they were not then you do not need to concern yourself with this unless you want to, if you want to learn even more than I have just described here please view the following resource: (Python Classes (w3schools.com))

That wraps up the essentials of Python too, using this guide and a combination of the resources I have been leaving along the way (Thanks w3 and GeekForGeek) at this point you have a solid enough footing in Python to do any of the contexts you wish, while the next chapter is also relevant to all sections - I believe it is the last thing covered prior to becoming context specific for the rest of the course - it only expands on the essentials and just opens a new pandora's box of things to discover in Python. I recommend people from any context to read through Chapter 8 and after that you can pick and choose what remaining pieces of this text you wish to consume, as I mentioned in the beginning I am modeling this guide after the content from 1006 since that is what I have access to directly (as well as the only introductory Python course to keep such records available) and that course still has a few more important topics to cover.

At this point though, you should definitely be equipped to handle the first set of homeworks and a significant amount of content needed for the midterm exam in October.

Chapter 7: Files and JSON

Up until now we have been working solely within the confines of our own file space, now we are going beyond that and working with other files within our own file space.

First it is important to make sure that the file you wish to open is contained within the same directory as your current file, for example if you have a folder with all your Python files you'll need to put any other file you wish to open in that folder as well. We can use the following code to read from a file:

```
my_file = open("somefile.txt", "rt")

for line in my_file:
      print(line)
```

Whatever the contents of that file were, they should now be displayed where you have been seeing output this entire time. The second argument we passed to the open function means "read text" which is the default so if this is all we desired to do we could've omitted it. Once we are done with a file we can close it use the close() method.

If we wish to append to a file, we would use "a" or "at" if we wish to overwrite the file we would use "w" or "wt" (you can pretty much omit the second character unless you wanna work with binary in that case use "b") We can use the write method on a file in order to write to a file so let's assume the file I use is blank but exists and is within the directory.

```
my_file = open("someotherfile.txt" "a")
for i in range(1,101):
      my_file.write(str(i) + "\n")
my_file.close()
```

If we opened the file again for reading we should see the numbers 1 to 100 each on their own line within the file.

For the purposes of 1002, that pretty much sums up what you'll need

to know in regards to file handling, below is a chart of all the different string arguments you can use with the open function and what they do as a helpful tooltip:

Argument	Description
“r”	Allows for you to read the file
“w”	Allows for you to write to the file but overwrites the file in the process
“a”	Allows for you to write to the file and appends whatever you write to the end of the file
“x”	Allows for you to create a file of the given file name

If you want to learn more about Files in Python please view the following resource: (Python File Open (w3schools.com))

Now we will introduce you to your first Python library. JSON is a syntax for storing and exchanging data and is text, written with JavaScript object notation. Some of you may go on to use JSON in your day to day functions down the road so while it doesn’t come up often in your coursework (in fact I believe it is only relevant for like a single lab) it is still useful to learn.

To start please import the json module into your code, imports are done as the first lines of code in a python script, we will learn more about this in the next chapter so if it is unclear now don’t fret, but there will be an example down below anyway. Once this is done we can open the .json file like we normally do we can load our file’s contents into a Python dictionary using the loads(x) method (or you can use load(x) without having already parsed the file) we can then make modifications like we would with a normal dictionary and then finally using json.dump(x,y) we can return the modified content to the json file.

See below for a proper example of this:

```python
import json
my_file = open("somefile.json", "rt")
contents = my_file.read()

json_content = json.loads(contents)
json_content["new"] = "old"
json.dump(json_content, some_other_file)
```

The following chart shows the different methods and what they do:

Method	Description
json.loads(x)	Takes a string x and returns an object which is a Python dictionary
json.load(x)	Takes a file object and return an object which is a Python dictionary
json.dumps(x)	Takes a Python dictionary x and return a string formatted to the JSON standard
json.dump(x,y)	Takes a Python dictionary x and a file object y and writes the contents of x to y in JSON format

That wraps up JSON and as a whole files, if you wish to learn more about JSON in Python you can view these wonderful resources: (Python JSON (w3schools.com)); (Python - Difference Between json.load() and json.loads() - GeeksforGeeks)

The next chapter focuses on importing different libraries to expand the functionality of your programs, this includes your own files that you write and wish to use in another file.

After the next chapter it becomes more content specific so please view only those which you find relevant or interesting :)

Chapter 8: Introducing Python Libraries

While there is so much that is able to be done within Python, it is often not necessary to reinvent the wheel. People have already taken the time and energy to do certain things that we may find ourselves also wanting to do, it may also be true that we have some code in another file that we may wish to execute in our current file without rewriting the code you wish to run.

This is where we can properly introduce the import statement, at the top of the files we write we are able to write import statements that allow us to effectively bring in outside code to use within our own file. The following are all possible ways to import some module or file in Python:

```
import my_module
import my_module as mm
from my_module import some_function # the wildcard, * ,is also acceptable
```

The first way imports all functions from my_module but when you need to execute the function you'll need to preface the function call with my_module, for example:

```
my_module.some_function()
```

The second way does the exact same as the first except this time we provided an alias, meaning when we wish to execute the function from that module we can use the alias rather than the whole module name, for example:

```
mm.some_function()
```

Finally, the third way is used to import specific functions from a module, or if you use the wildcard you can import all of them, since you have already specified the module specifically using the from keyword you do not need to preface the function call with anything.

Personally, the third option is my go to whenever I am dealing with my own code that I wrote, since I don't need any namespace identifier to tell me what the code is associated with. If I am already pretty familiar with the module I am using code from I will use the second

option, I will also use this option if the module name is too long and I want to save the keystrokes. The first option is reserved for modules that I am relatively new to.

You are expected to import your own code from another file at least once throughout the course, it may be in your best interest to do it even if it is not required based on the circumstances.

The following are some examples of popular modules that people tend to import and what their primary use cases are:

Module	Description
pandas	Used primarily for processing and parsing large data sets efficiently
numpy	Used to provide more mathematical and scientific computing options as well as new object types
matplotlib	Used to display simple graphical information, such as images or plots of data

These are three of the more well known libraries/modules that are often used in Python programs, these will all be covered in their own individual chapters, in fact unless I decide to add more content in the future, this will make up all the remaining content in the document. Pandas is the only module that has been explicitly used in assignments in the past to my knowledge so I will go more in depth on that module compared to the others but all of these were regularly used in ENGL 1006 which is why I chose them to be included here. To save space in the future chapters these are the common ways people import these libraries:

```
import numpy as np
import pandas as pd
import matplotlib as plt
```

Chapter 9: Pandas

I won't even try to sugar coat anything here, I really do not like pandas, I am not someone who deals with data crunching that much so I tend to avoid it but for purposes of this document and probably my labs/office hours I have to use my brain and teach it :)

To start, in the file you wish to work in you'll need to import pandas and load up your .csv file (this is the file format we will be working with) you can see an example below:

```
import pandas as pd
data_frame = pd.read_csv('data.csv')
```

This is one of the two main data structures within Pandas, the other being a series, you'll be using data frames a lot so it is best we get used to loading them up like the above.

It is important to note that you may have to install pandas before it will let you do anything, open up the command prompt or terminal and run one of the two commands:

```
conda install -c conda-forge pandas
pip install pandas
```

If none of these work, please consult with a member of the teaching staff!

Moving on from that, to describe it briefly, a series is like a column in a table, it is a 1-dimensional array that holds a specific type of data. Accessing these elements depends on the index:

```
a = [1, 7, 2]
my_var = pd.Series(a)
print(my_var[0])
```

OR

```
my_var = pd.Series(a, index = ["x", "y", "z"])
print(my_var["y"])
```

For our purposes, you'll most likely end up doing it the normal way (0-based indexing) unless specifically asked otherwise. From this we can gather that a data frame is effectively an array of series! So now you can picture it like any other table you've seen in your entire life.

By default a data frame will use 0 based indexing, to get a specific row of the data frame the method we choose can depend, if we wish to get it strictly by 0 based indexing, we use .iloc(x) if we wish to go off a specific label (which may be 0 based indexing) we use .loc(x)

Overall .loc(x) is more powerful than .iloc(x) so use it when you can but for purposes of 1002, you will not be penalized for using your preference assuming they return the same correct result. To learn more about .loc(x) vs .iloc(x) please view the following resource: (Difference between loc() and iloc() in Pandas DataFrame - GeeksforGeeks)

We can use the .head(x) and .tail(x) methods to quickly view the first x or the last x rows within the data frame, we can also use methods like .info() and .describe() to retrieve more information about our data set as a whole.

If you do not like the default indexing provided you can change the indexing to index by a column within your data we can use the set_index(x, inplace=True) method. With x being the name of your column as a string. We use inplace=true because we wish to modify our existing dataframe rather than make a new one.

We are also able to reduce our dataframe using some criteria, say in our csv file from earlier we have an column detailing the literacy rates of countries and we only want to deal with countries with a literacy rate of 80 or higher than we would use the following line of code to store that:

```
high_lit_rate_df = data_frame[data_frame["literacy rate"] >= 80]
```

We also have tools that enable us to remove data that may not be complete by using the .dropna() method on the data frame to remove any rows that are missing some data. If we wish to do this for only a

specific row, we use a similar technique as the line of code above but rather than >= 80 we will do the following:

```
full_lit_df = data_frame[data_frame["literacy rate"].isnull() == False]
```

If the situation comes up in which you may consider using either of these, please make sure you won't be losing valuable data by asking a member of the teaching staff to double check and they will give you the go ahead on that matter.

That wraps up my discussion of pandas, this should definitely be enough information to at least get you started with pandas, if you are seriously considering doing anything related to data science then you will want to read the chapter on matplotlib as data visualization is a very important component to consider as well.

Below is a table discussing the important methods that I have mentioned and what they do:

Method	Description
.head(x)	Retrieve the first x rows of the dataframe
.tail(x)	Retrieve the last x rows of the dataframe
.loc[x]	Retrieves the row by its label
.iloc[x]	Retrieves the row by its position
.isnull()	Returns True if there are some missing values in the row, otherwise False
.dropna()	Returns a data frame that contains only fully filled out rows

Pandas is a very verbose module and has wayyyy more than I could possibly discuss, you could write a whole textbook on pandas itself, here is a link to the documentation: (pandas documentation — pandas 2.0.3 documentation (pydata.org))

Chapter 10: Numpy

Numpy is the most used computational library that python has to offer, while there are others that are similar like sympy or scipy we will focus on Numpy simply because it is used the most, has the most to offer you at this stage, and in the event you become a CS major or intend to minor you'll need to take Discrete Math and/or Computational Linear Algebra which make abundant use of numpy, while also going more in depth on it as well.

We can make a numpy array, the primary data structure used with numpy operations, in a very simple way:

```
import numpy as np #reminder that we can assign alias to our modules

#we can also make multidimensional arrays if we so wished
my_np_array = np.array([1,2,3])

print(my_np_array)
```

Numpy arrays have the property that if you add them or do any sort of algebraic manipulation to them or between them you will see the effects as you would probably expect which is not a normal function of regular python lists:

```
print(2*my_np_array) #yields [2,4,6]
print(2*[1,2,3]) #yields [1,2,3,1,2,3]
```

Since most of the usefulness of numpy comes from its applications in mathematical and scientific fields I will simply provide a list of useful numpy methods and what they do and close the lid on numpy, here is the link to the documentation if you want to take a look for yourself: ([NumPy documentation — NumPy v1.25 Manual]())

Please note that you will most likely not be required to use numpy for any operation throughout the course but the option exists assuming it provides you with the correct result and does not heavily abstract the algorithmic complexity of the assignment.

Method	Description
np.array(x)	Converts a python list x into a numpy array
np.dot(x,y)	Returns the dot product of x and y
x.shape	Returns the dimensions of a numpy array x
x.max(), x.min(), x.mean(), x.std()	Returns the maximum, minimum, average, and standard deviation of a numpy array x
np.zeros((x,y))	Returns the 0 matrix of dimension x by y
np.ones((x,y))	Returns the 1 matrix of dimension x by y
np.eye((x,y))	Returns the identity matrix of dimension x by y
np.arange(x)	Returns a 1-dimensional containing the numbers in the range [0, x)
np.vstack([x,y])	Returns a new numpy array containing the elements of x on top of the elements of y
np.hstack([x,y])	Returns a new numpy array containing the elements of y to the right of the elements of x

This is only a fraction of the stuff that numpy can do too! These are just the ones I felt provide the best introduction to numpy, as I had said previously you may want to check out the lengthy documentation in order to find even more methods to use and how they operate and the kinds of arguments they take in!

The last library that we will be discussing in a bit of depth is matplotlib which allows us to plot graphs and images

Chapter 11: Matplotlib

Matplotlib is one of the oldest data visualization libraries that Python has to offer, and it is not the only one but it is often used in conjunction with numpy so that is what we will be doing here throughout the examples. In particular, I will demonstrate how to generate three types of figures, a line chart, a scatter plot and a bar graph.

Of course there are more of these available and the particular technique used to do them varies, at the end of the chapter I will provide a link to the documentation so you can read about all the different things that Matplotlib can do for you!

First we are gonna make the necessary imports:

```
import matplotlib.pyplot as plt
import numpy as np
```

Once we have imported these two we can make the data and then plot a line chart:

```
# make data
x = np.linspace(0, 10, 100)
y = 4 + 2 * np.sin(2 * x)

# plot
plt.plot(x,y)
plt.show()
```

This plots the graph of y = sin(2x) from 0 to 100 in increments of 10.

If we want to do a scatter plot instead, then we would change the following:

```
plt.plot(x,y) -> plt.scatter(x,y)
```

And if we wanted to make a bar graph instead we would make the following change:

```
plt.plot(x,y) -> plt.bar(x,y)
```

Those are the plain defaults but we are able to make other changes as well for example we can change the color from its default blue with the following addition:

```
plt.plot(x,y, color="red")
```

For the bar graph we can make adjustments to things like the width of the bars and their edge color with the following changes:

```
plt.bar(x, y, width=1, edgecolor="white", linewidth=0.7)
```

Finally with scatter plots we are able to change the sizes of each of the dots by adding the attribute like so:

```
np.random.seed(3)
x = 4 + np.random.normal(0, 2, 24)
y = 4 + np.random.normal(0, 2, len(x))
sizes = np.random.uniform(15, 80, len(x))
plt.scatter(x,y, s = sizes)
```

There are plenty of more examples provided along with the documentation to learn more about matplotlib available at the following link: (Matplotlib — Visualization with Python) Your use of matplotlib may vary depending on the context you are enrolled in, I would assume a context like art may make more use of visualizations than something like linguistics.

Matplotlib is used in plenty of other courses as well, if it has Python involved matplotlib is not far behind. ENGL 1006, COMS 3203, COMS 3251 and more all make use of matplotlib in coursework and other assignments in the event you choose to enroll in any of those courses.

Matplotlib is not limited to data visualization either, you can use your knowledge of file handling to display and modify images as well! A common use case is to show off encryption and decryption of images. That was one of my last assignments in ENGL 1006 and it was a festive picture that resulted from the decryption :)

Chapter 12: Conclusion

Well you made it to the end of the document, if you actually made it this far then congratulations I sincerely hope that you were able to learn something new along the way. While this document will be posted within the course files there will be some files that will not be there immediately to begin with and will be gradually introduced as the need arises or as I see fit.

Your reward is access to more materials that compliment much of what you have been working on as well as more concrete examples that you can just download and run within your coding environment of choice, in order to access this collection of files and other materials please send me, Griffin Newbold, an email to request access and I will send you the appropriate resource in response.

If you are reading this as a refresher after the course has concluded then you are still welcome to email me and request access.

Of course this document may change as time moves forward, I may add more chapters and other relevant information as I see fit, I will continually update the document as these changes occur but for now this is all I have for you, I appreciate your time in reading this and I really hope you found it somewhat useful. If you have any questions feel free to email me or stop by my office hours!