Database Basics

SQL is a program for storing data? You can take tables and join them in different way (inner joins, equi-joins, outer joins).

Mark: No, SQL (structured query language) is a language for interacting with the database. It allows convenient access to organize the data in an organized and consistent way.

Relational vs Non-relational databases are different places where data is stored and how it's stored and manipulated.

Mark: Relational databases (DB) are for storing carefully structured data in a consistent way. They have all sorts of requirements to ensure data can be stored and manipulated consistently.

Non-relational databases are more flexible (i.e. you can store whatever you want instead of storing only what the DB has been defined for) but are harder to work with since there is often no consistent way the data are structured.

Is the main thing I need to know about relational databases and non-relational databases is that they are 2 ways to store data, depending on what you need to do in your company?

- I don't really understand the difference because I don't really understand what it's applications are. They talk about transactions a lot for relational databases, so would they store credit cards because they're faster and more reliable but can't store variable types of information, which is where NoSQL databases come in? Is that all I should know?
    - *Mark: I would think of non-relational DBs as just loose structures of data. (Think of a Python dictionary and how that might work as a database.) Think of a relational database as a table with set columns that can only admit data that is included in each column (imagine a table of credit card info where one column is the customer name). Knowing how the data are structured beforehand is a help and improves the speed and reliability but suppose you also want to put a customer bio for some of your customers. What do you do then? Well you can't use that table. You have already defined that it has the columns it has. You must have something more flexible. Enter non-relational DBs.*
- Should I know how to code SQL? Or will I eventually learn how to interact with these databases?
    - *Mark: I would recommend learning SQL eventually. I would also master knowing when to use what language or software. Often this is more important than knowing every language.*

Data Frames and Series

Are data frames and series just ways to make tables in pandas?

Mark: Yes and no. A Data Frame can be thought of as a 2-D dictionary. They have key1, key2, value combinations (where key1 is the row row key and key2 is the column key). A series is just a data frame with only one column so the column part is unnecessary. This may seem like a trivial thing but there are all sorts of tools included with these data structures that make them powerful for data analysis. This also allows easy manipulation of data.

Extract, Transform, and Load

Is this just what happens when you pull data from a database so the end use can make a decision? Like facebook would extract and transform data about your profile and load it on your screen. Or amazon would extract and transform your credit card information from their database to load it on your screen so you can use it?

Mark: not sure exactly what you're referring to here. I would need more context to answer this question accurately.

JSON, XML, CSV

JSON (JavaScript Object Notation) stores data in text, using C-family language structure

- Usually a text representation of dictionaries/lists/etc.
- *Mark: except note that, in json, all keys in dictionaries must be strings where in Python the keys can be of any type*

XML (external markup language) is human-readable and machine-readable code for "marking up" text (e.g. HTML)

Mark: No. XML stands for "eXtensible Markup Language". It is a structured language that is human-readable (in the loosest sense of the word) and is easy for computer to parse and process. (HTML is a subset of XML intended for the web or sending "Hyper Text".) The whole spec revolves around open and close tags such as: `<h1 id="top"> content here </h1>`*. It is called extensible because it can be applied to any type of data.*

CSV (comma-separated values) is a file that stores data in tables using commas in the ~~language~~ *Mark: file type*

- ~~Usually~~ *always* text representation of rows and columns (csv files are not fundamentally different from a text file)

TXT

- Just text

Regular Expressions

Is this just code that searches for and finds text? Which you can replace if you want, or do something else with it?

Mark: Regular expressions (regex) is code that is used to match patterns in text. So if you wanted a line in a text file that matches any amount of whitespace followed by the word "cheese" followed by any number, the following text file:

```
cheese 1
  cheese 2342
      cheese 092

```

Searched with the following regex: `\scheese [0-9]`

would match thusly (matching indicated by the stuff between the curly braces):

```
3 hits:
    Line 1:  { cheese 1}
    Line 2: {   cheese 2}342
    Line 3:     {   cheese 0}92
# note that lines 2 and three contain tabs instead of just spaces.
```

regex can be combined and used to do all sorts of pattern matching in huge text files very efficiently. You can then do find and replace but the key is pattern matching.

Python

Style

- Surround top-level function and class definitions with two blank lines.
- Method definitions inside a class are surrounded by a single blank line.
- *Mark: These are not hard and fast rules. Use more if it helps your code readablity.*

Numpy is a wrapped library that connects to ~~ForTrans~~ *FORTRAN* and a number of other C libraries to make numerical calculations in python code RUN faster (matrices, etc.) *Mark: Basically it hands off the heavy lifting to a faster but less flexible code library.*

Pandas ~~("kinda replaces excel")~~ (*Mark: can replace most of excel*) allows you to analyze and manipulate data. E.g. Table of chemical ID (column), names (column), formula (column), export ait (column), etc. A Huge matrix of data, and what this allows me to do is graph, sort, call specific mathematical functions, etc.

SciPy gives you statistics, regression, optimization, etc.

Iterables

-Sequenced

Strings are made up of *characters*. ~~These are like lists for only characters.~~ *(More like tuples because they are not mutable inplace)*

Ranges are made up of *index_values*. ~~These are like lists for only numbers.~~ *(Actually a generator. That way they do not need to hold all the elements at once.)*

[Lists] are made up of *elements*. These iterables are the most useful. *(Debatable)*

(Tuples) are made up of *variables*. These are like lists that you cannot change.

-Not Sequenced

{Dictionaries} are made up of *items*, which consist of *keys* that are assigned to a corresponding *value*. Dictionaries are iterables but they are also ao mapping type. They take one type of data and connect it ("map it") to another. This is also called a "hash map" or a "hash table."

-Subscript and Slicing

Subscripting is when you print only 1 item from an iterable

Slicing is when you print a series of items from an iterable

Loops

While-loops use a truth expression as the constraint

For-loops usually use a ~~range expression~~ *iterable* as the constraint

If, elif, and else

These are not loops, but they are filtering code that determines which direction the code will go. They are only run once, or not at all (because they were skipped over).

Functions

You can make up your own functions, meaning, you can make your own structure of procedures to follow when you call this made-up function. That way it simplifies your code when you have to keep calling this function. (*Make your code resuable!*)

- **Parameters** - the general way the function is defined in terms of what it will accept as input (e.g. def factorial(number) *number is the parameter*)
- **Argument** - the actual, specific input to a function when you call it (e.g. factorial(4) *4 is the argument*)

Create a Matrix with List Comprehension

List Comprehension

Objects and Classes

Instantiation - calling an object into existence by a function

Shallow copy vs Deep copy. One is a new reference to the same object and the other creates a different object.

Questions

Logical Expressions

1. What's the difference between `1 and 2` and `bool(1 and 2)`?
    - *Mark: the `and` operator, the `or` operator and the `not` operator are more like the following functions (i.e. `and` and `or` do not evaluate to bools):*

```python
def or(x, y): # x or y
    if bool(x) is False:
        return y
    else:
        return x

def and(x, y): # x and y
    if bool(x) is False:
        return x
    else:
        return y

def not(x): # not x
    if bool(x) is False:
        return True
    else:
        return False
```

If, Elif, and else

1. Error = false in the if, elif, else section. Is error a special variable? What exactly does that do? Does python recognize error as something special to do if this variable is acted upon or something? What purpose does this serve at the beginning of the example? **Was it to set up lines 41-44 to manipulate the boolean logic?**
   - *Mark: `error` is just a variable that tells me, "Has an error occurred?" if an error occurs I do something different as seen in line 44. `error` is not a special variable, I just named it that so it was clear what it was for.*
2. If elif else: is the error the fact that the situation where the revenue equals 1000 then the message will still say congrats you're "above" the break even point? But that wouldn't cause an error...is it dividing an int by a float? Or does python know to convert that into a float already (that's what I think I heard before)?
   - *Mark: if you look at line 63, that is what happens if there is an error. I am trying to catch the case where the user misspells the farmer name. If there is no farmer to check then we do not want to do anything but let the user know something went wrong. Hence the print statement on line 64.*
3. The print on line 53, how come that print happens on both the if and the else statements and not just the else statement because it looks like it's nested in the else. Is it because it's not indented?
   - *Mark: That print statement is in the indent of `if not error:` on line 44. It is not part of the if or else blocks of line 48 and line 50.*

User_input.py file

1. Why does it spit out "Violas rule!" every time?
   - *Mark: Remember that the `bool` of a non-empty string always evaluates to `True` and the `or` operator is evaluated **after** all of the Boolean operators. This means that every instance of your string will always evaluate to `True` and the else block will never execute. You should use the `in` operator with a list of the possible combinations of words.*

File reading and writing

1. What does exit = "" mean?
   - *Mark: Perhaps you were referring to line 3 `ext = '.txt'` which is just the file extension I want to use for this section.*

6. Did you override the data variable in line 29?
   - *Mark: Overwrite would be a more accurate term. The data that was there before was deleted when I did that. I did this because it's just an example and ordinarily you would probably want unique names for the data.*
7. Do you always have to store a new file as a variable in the code if you want to manipulate it? Do you always store it as a variable even if you don't want to manipulate it? Line 19
   - *Mark: `f` in this code is a variable representing a file object (see here). This is different from the string contents of the file which is what I usually want. Therefore I have to bring the file into existence and then use the `read` method to get the string contents of the file. This must be done in the context block referred to by the with-as statement.*
8. Line 16, do you always end a row of data with \n?

   - *Mark: If I did not do that the file would contain one line of text with no visible separator between lines. The newline tells the file to go to the next line. Try this yourself and notice the difference in each file.*
9. Lines 14-16, why couldn't you use variableA + ",", variableB + ",", variableC + "\n" on one line of code?
   - *Mark: Yes. I only wrote it this way for readability.*

11. Line 20, if you manually feed it something to write, does that first manual feed always get put in as headers? How come it doesn't get put in as the data? Or is it just the first row, and the computer doesn't really interpret it as a header vs data but rather, row 1 vs row 2?
    - *Mark: When writing to a file, each call to `write` appends the argument to the contents of the file. Therefore, I am just writing text each time. It is entirely arbitrary what you write as a text file has no concept of "headers" or any other structure except text.*

Iterables

1. Why does print("Last from range:", example_range[-1]) spit out "4" when the range was 0 to 5? Is it because the 5 is exclusive in the range?
   - *Mark: All indices in Python work this way: if you give a range from 0 to n it will start at 0 and give n outputs which means the range does not include n but will include [0, n-1], hence range[-1] is the last value of the range and for n=5 that will be 4.*
2. And then why are the first two "(0, 2)"? That seems like 0, 1, and 2 to me. Unless the 2 is exclusive in this notation as well.
   - *Mark: See the above answer*

Functions

1. Why do you have to have that running=True variable at line 37?
   - *Mark: This is just a variable to keep my while loop (line 44) running, so I set it to `True` to start.*
2. Why does the main() function keep running even when the input is already given? Why doesn't the code just close out? I guess this is related to my previous question.
   - *Mark: Until `running` is `False` the code will loop infinitely in the while loop that starts on line 44. This is an example of a REPL loop.*
3. For the main() function, is there a way to say "for "make" in input(prompt)" instead of having to set the prompt's input equal to another variable and seeing if that variable matches your input? In other words, is there a way to bypass the need for a variable and just say, "if the input of the prompt was "make" do this"?
   - *Mark: If you put `input(prompt)` everywhere there is `choice` the code will continue prompting for every check of the if-else blocks. Doing it in line 45 lets us check once and then execute.*
4. I'm struggling to understand how recursions can be useful outside of factorials. Maybe an example would help. It kind of seems like a more complicated way to do a for-loop.
   - *Mark: Google the following "solve maze using recursion", and this example on adaptive quadrature*

More Functions

1. I'm trying to understand the use of parameters for functions. The parameters for a function seem like they can be written inside the function, (*Mark: yes that is true.*) like for the quadratic example, instead of using a, b, and c as parameters, we could have no parameters and ask for input from the user to give us an a, b, and c, and assign those inputs as variables which we would use to find the solutions. That would be valid too right? So what's the benefit of using parameters over simply assigning those variables inside the function?
   - *Mark: Well what if you don't want to use that function in the context of user input? What if you want to take a list of multiple sets of `a`, `b`, and `c` values and find the roots of their respective functions? User input would inhibit your ability to flexibly use that function. Often I use user input to make simple code easy to test but that is hardly the only use case.*
2. In lines 43-45, when you put an if statement, you don't always need to put the else?
   - *Mark: No but it is good practice to (almost) always have an else to your if statements.*
3. Could we go over more recursion examples? I could not come up with my own example without writing something without exceeding the max recursion cycles.
   - Try implementing the maze generator or solver (search "solve maze using recursion" on the internet). This will help you see how it works.

Try Except

1. Is running=True a variable?
   - *Mark: `running` is the variable, `True` is its value.*
2. I couldn't get my raise to work.
   - *Mark: I would have to see your code to know why.*

Imports

1. *Matrix question (LOVE THIS PROBLEM, cuz it's relevant to what I want to get into).*
   1. *What exactly is happening?*
      - *Mark: It is producing a list of lists with an equal number of rows and columns (e.g `[[2, 3],[1, 5]]`). Functionally this can be considered a matrix with dimensions `size` rows and `size` columns where size is a positive integer.*
   2. *The randomizer is putting a random element into each row, and then each row is being iterated on for size "size"?*
      - *Line 7 makes an empty list (or matrix).*
      - *Line 8 iterates through the number of rows.*
      - *Line 9 makes an empty list to be the row to be added.*
      - *Line 10 iterates of the number of columns (which is the same as number of rows)*
      - *Line 11 appends a random number to the current row*
      - *line 12 adds the completed row to the matrix*
      - *line 13 returns the matrix*
   3. *Which ones are the rows and which ones are the columns?*
      - *By convention, the first index is the row and the second index is the column. (This can be reversed in some conventions.)*
   4. *Can we go over this process one element at a time in the whole matrix?*
      - *You can iterate over the whole matrix this way:*

```python
for row in matrix:
    for element in row:
        print(element, end=" ") # just end with a space
   print() # just print the default newline char
```

- *You can also build this matrix with the list comprehension this way:*

```python
matrix = [[random() for i in range(size)] for j in range (size)]
```

Which is much easier don't you think?

Feedback

- Make vocab words explicit. Have its own box. Bolded, with a formal definition that we can refer back to when we forget what that word means.
- I like that you use the formal language of the python code instead of dumbing it down.
- Scaffold the "Hone Your Skills" with example problems,
- provide some example code to read for chapter 24,
- provide some riddles to solve when defining functions.
- You do this sometimes, like for the List Comprehension example, and that was gold. I loved that I had a specific resource to refer to, and a specific challenge to do according to the skill level I'm at, because I tried looking for some challenges or problems online but sometimes the answers implied that I knew some other function or method or thing that I didn't actually know yet.
    - *Mark: I will look into these. Good feedback! I would love any more you can give.*

Objects and Classes

1. Why is the second example "how all objects behave" but that's not how the first example behaved?
    - *This is a weird quirk of Python. When you assign to an integer, for example. Python makes that integer object (`int(2)`) and points `x` to it. When you say `y = x` Python points y to the same object. However, when you do the `+=` part, Python does the math, makes a new object (`int(5)`) and points `y` to the new object. The trick is to know this and understand when Python will do what thing.*
2. Can you give some examples of why you might want to use classes? I can't conceive of one, so I don't think I understand its utility, other than vaguely understanding that it groups a bunch of characteristics for something you want to keep track of, or store.
    - *Mark: Research encapsulation, polymorphism, and inheritance. These are the reasons Object-oriented programming (OOP) is used.*
3. Do classes always have a function like `def __init__(self, name, color, favorite_food):`?
    - *Mark: Yes, but it is defined in the parent class. Writing your own `__init__` allows you to override the parent behavior.*
4. Can we go over the Cat() example and go over what is a method (the functions right?), a member (the species variable right?), and what are the attributes (anything inside the function that belongs to it? Is it also the string "felis cactus" inside the species variable, since the string belongs to the variable? And the functions are, in turn, attributes to the class?

    - *Mark: Look up what a C struct is and what its for. A class lets us group both data and functionality together in the form of attributes (i.e. variables or data inside the class) and methods (i.e. functions inside the class).*

5. Could you explain this part to me? self is just a variable that could be named anything but is used by convention in Python. It is a reference to the **instance** of the object that is referring to the attribute.

    - *Mark: I need to know what part you need.*

6. So what exactly is object-oriented programming? As opposed to what?

    - *Mark: Look up the various types of programming (e.g. functional, procedural, imperative, OOP, etc.). See if you can understand the differences.*

7. So I looked up operator overloading, and I found this page on Real Python and I saw how you can specify how operators behave with your particular class, like adding to a shopping cart, however, I didn't understand how some of the other examples could be really useful. Could we go over a couple of their examples? Particularly the len() and the str() ones?

    - *Mark: This is a bit advanced but we can go over that stuff in a future class. (Please remind me.)*