PYTHON:

**CHAPTER 2 BASICS**

* Reserved words
* Patterns of code: Sequential, Conditional, Repititive

**CHAPTER 3: OPERATORS**

* \*\* is power
* type(xyz) returns type of variable xyz
* Division converts to float
* , in print leads to space
* # is commenting
* 4 spaces and tab are different in python

**CHAPTER 4: FUNCTION**

* try and except can be used where user input may lead to error
* max returns max value from the given string, small alphabets have higher ascii value so they are returned
* parameter i.e variable of a function gains its value during a function call
* argument is something from parent code that is sent to functn
* argument to a function is what you send if x=5 and you send x then x is argument not 5

**CHAPTER 5: LOOPS**

* Continue breaks the current iteration and goes back to loop
* In for loops flag value can best be None
* is and is not are usually used for true, false and none

**CHAPTER 6: STRINGS**

* Strings are indexed list, immutable i.e. particular element cannot be changed whole string can be
* Input function always gives a string
* IN CAN BE USED AS == in strings
* a string can be used in lower case: string\_name.lower(). string\_name can also be a variable having string
* dir() gives the methods in class that is in ()
* find(‘’) tells where in string did it found only first instant; replace() replaces multiple word or letter with other
* strip, ;lstrip, rstrip removes the white spaces, startswith returns true or false if string starts with asked letter or word
* Slicing can be used to edit string, since they are immutable
* string.isalnum(), string.isalpha(), string.isdigit(), string.islower(), string.isupper() checks if given string has alpha-numeric or purely alpha or purely numeric or lower case alpha or upper case alpha.

use **+** to concenate, **join** converts list into a string by joining all values,

**Parsing:**

**Regular expression: findall, search, sub, split, finditer.**  RE are tiny specialised codes in python used for text matching. Metacharacters don’t work in class.

1. **[ ]** is used to match a set of characters in [ ] i.e. [abc] matches a or b or c

2.

**CHAPTER 7: FILE HANDLING**

* Intro to interaction with secondary memory
* open(‘file\_name’, mode) gives data in file to handle; mode is reading or writing. We can also use .read() just to read
* \n : newline character. However. read command reads the complete file as a single line string
* when we refer open to a for loop it considers number of lines
* Print creates its new line and rstrip is used to avoid \n of line coz we already have \n of print()
* quit() exits the code
* open is file format
* and line from ‘file\_name’ is text format

**CHAPTER 8: LISTS**

* List can have multiple data types; another list as well
* Strings(‘ ‘) aren’t mutable
* but lists([ ]) are mutable
* range( ) returns all numbers from 0 to range - 1
* when we use for a in b, a take every value of list b but if we want number at which that element of list is,we use for a in range(b)
* using for a in b; when b is a string, a takes value of each string element, for file it takes no of line
* Concenate, Slicing same as string
* in and not in are used in the list as boolean comparisons true or false
* list can be sorted using sort
* split() converts string into list, can also use any character
* double split can also be used. Like split and then split again

**CHAPTER 9: DICTIONARIES**

* A bag of values with labels
* Mutable
* dictionaryname.get(name,default\_value)
* list(dictionary\_name) gives individual keys of dict as list or print(dict\_name.keys())
* print(dict\_name.values()) gives values in the same order of keys
* print(dict\_name.items()) gives key value pairs in tuples
* for aaa,bbb in dict\_name.items(): lets aaa, bbb take key and value repectively
* no duplication of keys allowed

**CHAPTER 10: TUPPLES**

* Unmutable, use ( ) to define tupples
* when comparing two tupples, it starts with first and goes to last till it finds result unless anything is false i.e (3,4)>(2,5) is true but (2,5)>(1,6) is not
* dictionaries or tupples can be sorted but only on the basis of key
* we can convert a dictionary to tupples but reversing the key and value and then sort by value.
* Using sorted(tupplename, reverse = True) we can sort in descendin order

**Map** function returns an iterator object and not a list, it can be used to convert input straight into int

a=sorted(map(int, input().split(' ')), reverse=True)

**Basics of oops and characters, encoding**

* 128 characters: ASCII,
* ord(‘H’) represents the numeric value of H i.e. ordinal
* Every character stored as a byte of memory and UTF 8
* Unicode representation
* *OOPS*:
* Class: object constructor or a blue print, eg string.
* Object is the output of the class
* Methods dir(variable) eg uppercase
* Attribute
* *Object life cycle: Construction and destruction* methods i.e. object initiates class and destruction deletes it at the end of prog or may be re-initialise it.
* Inheritance: An ability to create a class which has capabilities of existing class plus some additional capabilities

Using Databases

* Development of database languages with increase of computational capabilities and memory.
* Connect is used to check access to the file and cursor is a communication medium that is we send our instructions to cursor and then cursor send output
* cursor\_name.fetchone() gives the top entry of the table. Cursor sends commands to sql but when we say select cursor doesn’t show. The data retrieved is not in cursor.
* Data injection is a web security technique where when you ask user for a name or id, it gives an SQL command that unknowingly changes your database

Data model

* Arrange data as per desired user interface but have efficient and optimised data storage. Importance of data model and their connections to storage and user interface.
* Data model is made from outwards to inwards.
* Data is case sensitive, but sql commands are not, column name may or may not be.
* Creating theoretical and practical data model.
* Using numbers instead of strings reduces memory occupied and amount of scanning.
* If ON is not specified, cross join is formed.
* We can join multiple tables using on \_\_\_ and \_\_ and \_\_\_
* Many to Many: can’t have foreign key, so create additional table to form relation
* Python 3 was about extracting data from file using modules to extract appropriate data from that and then create the table

Databases visualisation and geocoding

* Gathering data from database has many problems which makes the process blow away in between. So we gather the raw data and then use python to get a clean data.
* So we will write multiple python program, each for one step. This is some Data mining but Data mining is much more complex.
* We will deal with google’s geocoding API, pull data into database, and then visualise

**REGULAR EXPRESSIONS:**

[**https://towardsdatascience.com/a-gentle-introduction-to-regular-expressions-with-python-4f3fce46dcb4**](https://towardsdatascience.com/a-gentle-introduction-to-regular-expressions-with-python-4f3fce46dcb4)

[**https://towardsdatascience.com/anchors-away-more-python-regular-expressions-you-wish-you-knew-8a7780ac54e9**](https://towardsdatascience.com/anchors-away-more-python-regular-expressions-you-wish-you-knew-8a7780ac54e9)

1. re.findall(pattern, string): This function returns a list containing all instances of pattern in string
2. re.sub(pattern, repl, string): This function returns string with instances of pattern in string replaced with repl

Concepts driving regex:

1. **Character sets:** Represent options inside the brackets

char\_sets\_01 = re.findall(**r'[aeiou]**', basic\_string): **Finds all vowels in string**

char\_sets\_02 = re.findall(**r'[A-Z]**', basic\_string): **Matches range of characters**

char\_sets\_03 = re.findall(**r'[0-9]**', basic\_string): **Matches range of numbers**

[A-C1-4t-z]: **Matches everything in all ranges**

1. **Metacharacters**: Represent type of characters.

***\s. \n, \t*** *: space, newline and tab*

***\w****: Alphanumeric characters*

***\d****: Numerics*

In characters and metacharacters, it reads 12 as ‘1’, ‘2’. This is because it analyses each and every character individually.

1. **Quantifiers:** Quantify number of characters expected. Come after the character:

***+:*** *Quantifies one or matches. I.e. one or more characters following the given character unless a or . or space occurs*

***{}****: Specifies number of matches and {2, 4} specifies range of number of matches i.e. 2 to 4*

***\*:*** *zero or more matches*

1. **Capture groups:** Capture groups are regex grouped in parentheses.

| *capture\_group2 = re.findall(r'****(****[A-Z][a-z]+****)****\s\w+\s****(****\d{1,2}****)****\s****(****\w+\s\*\w\*****)****', basic\_string)*  This returns only things in round parentheses   1. **Text anchors:** Text anchors says to look either at the beginning or the end of the string.   ^: Matches the following regex at the beginning of the string  E.g. re.findall('^[Tt]he', string\_name)  $: Matches the preceding regex at the end of the string.  E.g. re.findall('ocean$', string\_name)  **Matching everything except:**  Capital Meta characters: Unlike metacharacters capital metacharacters match everything except what metacharacter would other wise.  ^ and character sets: Use ^ in conjunction with characters:  re.findall('[^AEIOUaeiou]', string\_name) : Finds everything except vowels.  re.findall('[^AEIOUaeiou\s\d\.]', negation): Also removes space, digits and fullstop   1. **Look around:**   **Look ahead:** "match this(?=if followed by this)"  re.findall('\w+\s(?=costs)', lookaround)  **Look behind:** "(?<=if preceded by this)match\_this"  E.g. re.findall('(?<=costs)\s\d\.\d{2}', lookaround) |  |
| --- | --- |
|  |  |

**Advanced :**



