## .py - file is a python program

## Python interpreter- reads through the program and interprets what each word means. Each word as in , for example what a function like print means.

1. **Print function** : prints
2. **Syntax highlighting** : highlights words as it (python interpreter) recognises a function, method, string, class, etc.
3. Variable: each variable holds a value .Value of a variable can be changed at any time and Python will keep track of its current value.
4. Variable names :

i) Can start with a letter/underscore , but not with a number.

ii) Spaces are not allowed in variable names.

iii) Variable names should be short and descriptive.

iv) Conventionally: variables are written in lower cases and classes in upper cases.

v) Do not end a variable name with a full stop.

1. Traceback: record of where the interpreter ran into trouble while executing the code.
2. NameError:

i) forgot to set variable’s value before using it.

ii) we made a spelling mistake

# 1.hello\_world.py 1-8.

#2-1 , 2-2

#name.py 9 -12

1. **Strings:** simple series of characters , can be in single,double, triple quotes.

When applying methods to strings: stringname.methodname()

**Changing the cases of the strings**

1. .title(): capitalises the first letter of the string
2. .upper(): capitalises all letters in the string
3. .lower(): lower cases all letters in the string

# 2.name.py 9 -12

**Combing or concatenating strings:** #3.Combining\_or\_concatenating\_strings.py-13-14

1. Python uses (+) symbol to combine strings.
2. Concatenation: The process of combing strings is known as concatenation.

**Adding whitespaces to strings with Tabs or Newlines :**

#4.Adding\_whitespace\_to\_strings 15-17

1. Whitespace: spaces, tabs, end of line symbols.
2. \t- adds a tab to your text(within a string)
3. \n- adds a new line to your string(within a string)

**Stripping white spaces-** #5.strip.py 18- 20.

For example, one important instance might involve checking people’s usernames when they log in to a website. Extra whitespace can be confusing in much simpler situations as well. Fortunately, Python makes it easy to eliminate extraneous whitespace from data that people enter.

1. .rstrip()- eliminates whitespace at the right end of the string.
2. .lstrip()- eliminates whitespace at the left end of the string.
3. .strip()- eliminate whitespace from both sides of the string

Stripping whitespace using these methods is temporary solution.

Rename the variable to make it permanent.

These methods are usually used to clean up the inputs before they are stored in a program.

Changing a variable’s value and then storing the new value in the original variable is often done in programming to make some methods from temporary to permanent.

**Avoiding Syntax Errors with Strings :** 6.apostrohphe.py 21-21.

1. SyntaxError: When python does not recognise a part of your code. Usually an open string is spotted by a colored higlighted line.

**#2-3 --- 2-7**

**Numbers:** 7.numbers.py

1. + - add
2. - - subtract
3. \* - multiply
4. / - divide
5. \*\* - to the power of/exponents

python supports order of operations, you can also use parentheses to create your own order.

1. .0 – called a float , when dividing to get accurate decimal point answer. Python3 will give exact answer without using a float. You need to add a float in Python 2.

**Avoiding Type errors with str() Function:** 8.birthday.py 28-31

1. TypeError: it means Python cannot understand what kind of information you are using. Cannot concatenate str and int objects.
2. Str() : when concatenating string and a number: specifying that you want to use the number as a string of characters is important.

Age=23

Message=”happy”+str(age)+”rd birthday”

1. Strings can be added to variables whose value is a string ONLY. In other words,

strings can only be added to strings.

When you use integers within strings like this , you need to specify explicitly that you want Python to use the integer as a string of characters. You can do this by wrapping the variable in the str() function , which tells Python to represent non-string values as strings.

1. Python does not accept plain text – NameError.
2. # - a comment or comment enclosed in triple quotes is also a comment.
3. **The Zen of Python:**
4. Code can be and should be beautiful.
5. A simple solution is always better than a complex one. Easier to understand and easier to build on.
6. Complex code is better than complicated code, meaning- it can be complex as long as it is easiest solution available… Don’t connect too many parts and create a ‘kichdi’.. That no one will understand… Simple code is easy to explain. Complex code is difficult to explain. Complicated code means you don’t seem to have a proper idea about how you got the correct result and youhave o revisit your own code again n again.
7. Write informative comments.
8. There should be one – preferable only one – obvious way to do it. Programming consists of using small, common approaches to simple situations within a larger, more creative project. The nuts and bolts of your program should make sense to other python programmers.
9. Don’t try to write perfect code ; write code that works and then try and improve your code.

33. **.find() method:** 9.find.py

it determines if the *str* occurs in string , or in a substring of string if starting index *beg* and ending index *end* are given.

Parameters:

str : This specifies the string to be searched.

beg : This is the starting index. by default its 0.

end: This is the ending index , by default its equal to the length of the string.

Return Value:

Starting index if found , and -1 otherwise.

Format:

str.find(str, beg=0, end=len(string))

34. **.replace() method:** 10.replace.py

Returns a copy of the string in which the occurrence of *old* have been replaced with *new* , optionally restricting the number of replacements to *max.*

IS TEMPORARY, overwrite to make permanent.

Format:

str.replace(old, new[, max])

Parameters:

old – This is the old substring to be replaced.

new : The is the new substring which would replace the old substring.

max : If this optional argument max is given , only the first count occurences are replaced. max here refers to the number of replacements you want to make ***.***

ReturnValue:

This method returns a copy of the string with all the occurrences of substring old replaced by new.

If the optional argument max is given, only the first count occurrences are replaced.

**For lists : Use index method to find a certain element.**

35. randint() function:

The randint() function is a part of the random module.

random.randint(0,10) will give any number from 0 to 10.

**Introducing Lists: VIEW 3-9.EveryFunction. 1-13.**

1. List: collection of items in a particular order. It should have plural name. Such as : names , numbers , animals. Square brackets indicate a list and individual elements in a list are separated by commas.
2. To access an element of a list: name\_of\_list[index\_number]- Python returns the element without any square brackets or quotations.
3. ALL methods of chapter 1 can be used in any element of the lists. (Simple logic- all methods in the previous chapter were for strings , SO if we access an element from a list, which is a string- we can use those methods on these elements of the list.
4. From the left side of a string- index position starts from 0.

From the right side, index position starts from -1.

**Changing ,Adding and Removing elements:**

1. Changing an element: name\_of\_list[index\_position]=new\_element\_name
2. Adding an element to a list:

Append: adds the new element at the end of the list.

List\_name.append(new\_element\_name)

Append has a permanent affect on the list.

1. Inserting elements into a list:

Insert() method: opens up space at the index position and stores the new\_element\_name at that location. This method Then, shifts every element one position to the right.

List\_name.insert(index\_position , new\_element\_name)

The insert method takes in two arguments – index\_position and new\_element\_name.

1. Remove elements from a list:
2. Removing an item using the **del** statement:

del statement can only be used if you know the index\_position of the element you want to remove.

del list\_name[index\_position]

del statement does not let you access the value that was removed from the list after using the del statement.

1. Removing an item using the **pop()** method:

The pop() method lets you remove the last item of the list but its lets you work with the items after removing them.

Variable\_name=list\_name.pop(index\_position)

The pop method returns the last element of the list and needs a variable name to be stored in.

The index position refers to the position of the element you want to pop and use later. If no index position is entered: pop removes the last element of the list and saves it in the variable\_name to be used later.

-->**use pop when you want to save the element you are removing for later use.**

**Use del when you have no future use of the element you are removing.**

**del and pop are used when we know the index\_position of the element of the list.**

1. Removing an item by value :

list\_name.remove(element\_name)

Remove is only used when we know the value of the element we want to remove from the list.

Remove only removes the first occurrence of the element\_name. If we want to remove all occurrences of that element. We have to use a loop.

**Organizing a list:**

1. Sort() method: organizes the list into alphabetical order , **permanently.**

For reverse alphabetical order: put in the argument: reverse = True

list\_name.sort()

1. Sorted() function :

sorted(list\_name)

This organises the list in alphabetical order **temporarily.**

Sorted function accepts argument – reverse=True for reverse alphabetical order.

1. Reverse() method : to print a list in reverse order , use the reverse method :

list\_name.reverse() – reverses the order of the list permanently. But you can revert to original by entering the same command again.

1. Finding the length of a list:

len(list\_name) : gives the number of elements in the list OR the last index\_position + 1.

**Avoiding Index Errors when working with a list:**

1. Index Error : python cannot figure out the index you requested.

Seeing the actual list or the exact number of elements in the list using the the len() function can really be helpful in solving index errors.

**Working with Lists:**

1. Looping: allows you to take a set of actions with each item of a list.

When you want to do the same action with each item in the list we use the

1. For loop: avoids repetition and code changing.

Each time a for loop enters: Python stores the current element in the variable for eg: magician ( for magician in magicians) its start with the first element in the list, performs the blocks of code and keeps on looping till its finished performing the same action(s) on each item of the list.

Format: for list\_name(singular) in list\_name(plural):

OR

for item in list\_of\_items:

<code blocks>

The code blocks are executed once for each item in the list

Every indented line within the for loop is block of code / or set of action(s) o perform for each item in the list.

Any lines that are not indented are executed once without repetition.

🡪If looping though an empty list/string/dictionary: Python won’t enter the for loop.

**The for loop does not enter empty data structures**.

**Avoiding Indentation Errors:**

1. Python uses indentation to determine which line of code is linked to the line above it.
2. Forgetting to indent:

Indenting lines within a loop, function, class etc. Is important. Beware of these errors. Python will point out: expected an indented block

1. Forgetting to indent additional lines:

These types of errors are logical errors, cause they work and don’t point out an error. But you code does not work the way it is supposed to. Meaning: the lines indented within a for loop will get executed once for each item in the list. But? The line you forget to indent will be executed only once after the loop is finished.

1. Indenting Unnecessarily: Indentation error: unexpected indent
2. Indenting unnecessarily after the loop: if you indent a code that needs to be run after the loop is finished, it will be taken in the loop as block of code and be executed once for each item in the list. This is another logical error. You code will run but not the way you want it to run.
3. Forgetting the Colon:

Don’t forget using the colon! You’ll get a Syntax error.

Such errors are difficult to find as we usually see what we expect to see. While reading through the code , you will probably see a colon after a for loop statement… Get in the habit of putting a colon after a loop, function or class.

**Making numerical lists**

**Using the range() function:**

1. Range(first\_number,last\_number,interval) : generates a series of numbers starting from the first number , all the way to the last number(-1). Python stops generating the numbers when it reaches the last number. So range(1,5) will be 1,2,3,4 … Will not include 5.

If no interval is added, interval is assumed to be 0.

If only one number is given in the arguments, Python will start counting from 0 till that number.

The range() function returns a list of numbers.

**Using range() to make a list of numbers.**

1. list(): the argument within the parentheses can be a string, range() , etc.

Converts the string/ range of numbers into a list of words/numbers respectively.

🡪This can also be done by putting square brackets around the data you want to convert into a string.

**Simple statistics with a list of numbers :**

1. Digits is a list of numbers from 0,9.

min(digits)=0

min(list\_name) tell you the smallest number in the list.

1. Similarly, max() tell you the biggest number in the list.
2. sum() tells you the sum of all the digits in the list.

**List comprehensions:**

1. This can be used when you want to do a single thing using a loop to create a list. In other words the executing blocks in the for loop do only one thing and require 1 line of code.

Eg: normal way to do it:

squares=[]

For value in range(1,11s:

Squares.append(value\*\*2)

List comprehension;

squares=[value\*\*2 for value in range(1,11]

Format for list comprehension:

List\_name=[expression(value) for value in data\_type]

When generating a loop to create a list, and it begins to feel repetitive, and unnecessarily long. Try and generate a list comprehension. That’s how you’ll learn.

**Slicing a list:**

1. Format: list\_name[index1:index2]

Index1: index\_position where you want to start slicing

Index2:index\_ position you want to stop slicing +1

If I sliced list[0:3] … It would return me : index 0, index1 and index2.

--if index1 is not entered, assumed 0.

---if index 2 is not entered, it will return all the elements of the list starting from index 1 till the last element of the list.

Using negative (number) without index2 will return the last (number) element in the list.

🡪**The number of elements that slicing returns is equal to difference of index2 and index1.**

**Looping through a slice :**

11. To loop through a slice, just write:

for element in list[index1:index2]

12. Copying a list: To copy a list :

list\_copy=original\_list[:]

This tells Python to start at the first item and copy all elements up till the last element.

By omitting index1 and index2 : we generate a copied list.

This is helpful cause if we just write:

list\_copy=original\_list

whatever changes we do to the copy\_list will affect the original list cause copy\_list gets assigned to original\_list. The copy\_list and original list point to the same list. No New list is created.

On the other hand: If we create an actual copy by slicing the original list from starting to end: We make a completely new list which is distinguished from the original list. Although, they both look the same.

13. Tuples: List of items that cannot change.

You can access individual elements of a tuple by using their index position, just like a list.

If you want the elements of list to never change , put them in a tuple. For eg: The dimensions of a rectangle. (200,100)

if you do try to change an element of the tuple:

tuple\_name[index\_position]=new\_element

#>>>TypeError: 'tuple' object does not support item assignment.

**Looping through all the values in a Tuple:**

14. When trying using a for loop on a tuple to print out each element: Python returns the elements in a tuple just as it did with the list.

**Writing Over a Tuple:**

15. There is only one way to write over a tuple:

tuple\_name=(new\_elemet1,new element2,……)

You cannot modify a tuple, but you can certainly write over it.

All immutable objects can be written over and not modified.

**Styling Your Code:**

* Helps in code-readability
* Helps in working with others

16. Adhere to this Styling code:

1. i) Indentation: four spaces per indentation
2. ii) Line Length: Less than 72 characters per line
3. use Blank Lines to differentiate between different parts of your program. Python interpreter uses horizontal lines to interpret indentations but disregards vertical blank lines.
4. Whenever you feel that your code is repeating itself, Try a simpler way and always AVOID REPETITION.
5. When using Comparison Operators or Boolean Expressions: Space Your Code among the comparison operator.

**IF STATEMENTS:**

1. If statements involve examining a set of condition and deciding which action to take according to these conditions.

2. Conditional Test: This is a test that either evaluates to True or False.

If the Conditional test evaluates to True: Python executes the code following the if statement.

If the test evaluates to False, Python ignores the code following the if statement.

**Checking for Equality :**

3. == : The equality operator , returns True if the values on the right side and left side match, False otherwise.

🡪It uses 2=’s cause 1 = is used for assignment.

**Ignoring Case when checking for equality:**

4. Testing for equality using equality operator is case sensitive.

If the case matters when you are checking for equality: Just check:

if this==that:

But if it does not matter:

Use if this.lower()= that.lower()

Converting both sides to lower() will tell us if both of them are the same string of words or list of words,etc.

You can use .upper() also.

*Websites enforce certain rules for the data that users enter in a manner similar to this. For example, a site might use a conditional test like this to ensure that every user has a truly unique username, not just a variation on the capitalization of another person’s username. When some- one submits a new username, that new username is converted to lowercase and compared to the lowercase versions of all existing usernames. During this check, a username like 'John' will be rejected if any variation of 'john' is already in use.*

**Checking for Inequality :**

5. != :means not equal to.

**Mathematical Comparisons:**

6. >-greater than

7. < Less than

8. >= Greater than equal to

9. <= Less than equal to

**Checking Multiple Conditions:**

10. and- The and operator works like this:

True and True : True

True and False: False

False and True : False

False and False: False

🡪Returns True only if both the conditions on the left and right are True.

11. or : The or operator works like this:

True or True: True

True or False: True

False or True: True

False and False

Returns true if either of the two conditions on the left and right are True.

**Checking Whether a Value Is in a List:**

12. in : To find out weather a particular value is in a list or any other data type( strings , dictionaries, etc. )

It checks for existence of the value in the data structure.

It’s a Boolean Expression. Returns Either True or False

**Checking whether a value is not in a List:**

13. not in: This is also a Boolean Expression.

Return True if the value is not in the data structure.

Returns False if the value is in the data structure.

**Boolean Expressions:**

14. I know 10 Boolean Expressions, They return either True or False.

== , != , > , < , >= , <= , and , or, in , not in

**if Statements:**

1. If statements involve examining a set of condition and deciding which action to take according to these conditions.

2. Conditional Test: This is a test that either evaluates to True or False.

If the Conditional test evaluates to True: Python executes the code following the if statement.

If the test evaluates to False, Python ignores the code following the if statement.

14. All indented statements following the if statement will be executed if the test passes or they will be ignored if the test does not pass.

**if-else Statements:**

15. Usually you will want to take once action when a conditional test passes and another, if the conditional test does not pass.

i) If the conditional test of ‘if’ passes, then the code blocks indented below the if statement are executed and ‘else’ is totally ignored.

ii) If the conditional test of ‘if’ fails, then the code blocks under the if statement are ignored and the code blocks of the else statement are executed.

🡪The if-else structure works well in situations in which you want Python to always execute one of the two possible actions.

**if-elif-else Chain:**

16. When you want to evaluate more than 2 possible conditions:

The if-elif-else chain is used.

It runs each conditional test in order ..until one passes.

When a test passes, the code following the test is executed and Python skips the rest of the test.

For Eg: An amusement park that charges different rates for different age groups.

Both if and elif follow a condition. else does not have a condition attached to it. This means that: When all test fail, Python executes the else block.

**Using multiple elif Blocks:**

17. Multiple elif statements can be used to evaluate all conditions that are necessary to evaluate.

**Omitting the else Block:**

18. The else block is not mandatory and an if statement would do nothing (i) if the if statement’s condition fails AND ii) there is no else block)

Sometimes, it is better to use an elif statement than an else statement as it shows the last possible condition. But this can only be done when you know all the possible cases/conditions possible.

**DICTIONARIES:**

1. i) A dictionary in Python is a collection of key-value pairs. Each key(info\_type) is connected to a value(info).

ii) You can use a key to access the values associated with that key.

In Python , A dictionary is Dictionaries allow you to connect pieces of information.

iii) A dictionary is wrapped in {}, with a series of key-value pairs inside the braces.

iv) A key value pair is a set of values associated with each other.

v) Every key is connected to its value by a colon.

vi) All key-value pairs are separated by a comma.

vii) You can store as many key-value pairs as you want.

🡪You can nest dictionaries inside lists.

🡪You can nest lists inside dictionaries.

🡪Dictionaries can store any two kinds of information that can be matched up.

Eg: List of words and their meanings.

**A simple Dictionary:**

2. Format of A Simple Dictionary: dictionary\_name={‘info\_type1’:’info1’ , ‘info\_type2’:’info’ , ……}

**OR**

dictionary\_name={key1:value1 , key2:value2, ……}

**Don’t Forget the comma after each key-value pair. You always forget the comma.**

**Accessing value in a dictionary:**

3. Give the name of the dictionary and then place the key inside the square brackets.

print(dictionary\_name[key\_name])

**OR**

print(dictionary\_name[‘info\_type]

**Adding New Key-Value Pairs :**

4. name\_of\_dictionary[new\_key\_name]=value\_of\_new\_key\_name

Here, we do not refer to any index number to position the new key-value pair. We used to append() a new element in a list it add it to the end of the list. We used to insert() a new element onto a list in any position we wanted to. But here, In a Dictionary: Python does not care about the order of the key-value pairs. Python only cares about the connection between each key and its value.

**Starting with an Empty Dictionary:**

5. Here, We add key-value pairs starting with an empty dictionary.

dictionary\_name={}

dictionary\_name[key1]=value1

dictionary\_name[key2]=value2

print(dictionary\_name)

#>>>{key1:value1 , key2:value2}

**Modifying Values in a Dictionary:**

6. To modify a value associated with a key\_name:

dictionary\_name[key\_name]=new\_value

**Removing Key-Value Pairs:**

7. How did we completely delte an element from a list?

Using the del statement: We delete the element from a list permanently.

Other methods like remove() and pop() allowed us to work with the elements after removing them from the list.

Here we use the del statement.

del dictionary\_name[key\_name]

This line will delete the key-value pair associated with the key\_name permanently.

**A dictionary of Similar Objects:**

8. A dictionary can have different kinds of information about one object.

eg: Name, age, sex of a person

person={info\_type:info}

**OR** it can have 1 type of information about many objects.

**eg:** Polling a number of people and asking them about their favorite language.

favourite\_lang{name\_of\_person:lang\_name}

9. How to split a dictionary in several lines:

Press enter after the first brace

write one key-value pair and then add a comma to the end of the line.

Then press enter.

Once you’re done entering the last key-value pair along with the comma ,press enter and close the braces.

{

key-value,

key:value,

….

}

🡪You can do the same with lists too.

🡪You can break a print statement that is too long in a similar manner.

By closing the string once the line has reached 72 columns, and then adding a concatenation operator(+) .

Then press Enter and then press TAB ,

and then continue with a new string.

All further lines will be indented under the print statement automatically.

**Looping through a Dictionary: 2.i.**

10. You can loop through :

i) key-value pairs

ii) keys

iii)values

**Looping through all key-value pairs:**

11. Format :

for key, value in dictionary\_name.items():

print(“Key: “ + key)

print(“Value: “ + value)

key and value are just example, you can name the variables in the dictionary as you like.

.items() returns a list of key-value pairs with the key-value pairs in tuples.

🡪[(key1,value1),(key2,value2)….]

If we do not include .items() we get an error:

ValueError: to many values to unpack

This means when we try to loop through a dictionary using 2 variables, we just have to many values too fit inside 2 variables. When we use the .items() method. We create a list of tuples each having 2 values. One is the key and the other is the key’s value. Then Python Loops through that List.

the for loop then stores each of these pairs in the two variables provided.

When looping through a dictionary, Python does not care about the order in which the key-value pairs are stored. It tracks the connection between individual keys and their values.

**Looping through All the Keys in a Dictionary:**

12. .keys() method is useful when you want to loop through only the keys of the dictionary OR you don’t need to with all of the values in a dictionary.

print(“The keys are:”)

for key in dictionary\_name.keys():

print(key)

the for loop tells python to pull all the keys from the *dictionary\_name* and store them one at a time in the variable *key*.

Only looping thorugh the keys in a dictionary is actually the default behavior of Python when looping though a dictionary so this code will give the same output without the .keys() method:

for key in dictionary\_name:

print(key)

**Looping though a Dictionary’s Keys in order:**

13. Using the Sorted() function you can organize the keys in alphabetical order before looping through them.

Remember, the sorted() function is temporary.

The Sort() function will permanently sort the keys() in alphabetical order.

**Looping though all the Values in a Dictionary:**

14. If you are only interested in the values of a dictionary: Use the .values() method.

Format:

The values are:

for value in dictionary\_name.values():

print(value)

The for statement pull all the values in the *dictionary\_name* and stores them in the variable *value.*

15. set() function: The set function with in an input as a list , returns a list of all the values in the list, without repetition.In other words: When you wrap set() around a list that contains duplicate items, Python identifies the unique items in the list and builds a set from those items.

**Nesting:**

16. For storing set of dictionaries in a list.

OR

Storing a List of items as a value in a dictionary.

OR

A dictionary inside another dictionary.

17. If you just want the loop to repeat a specific number of times , use the range function:

for variable\_number in range(n)

n here stands for the number of times we want the loop to repeat.

18. It’s common to store a number of dictionaries in a list when each dic- tionary contains many kinds of information about one object. For example, you might create a dictionary for each user on a website and then store in in a list called users.

Remember: All the dictionaries in the list should have identical structures so you can loop through the list and work with each dictionary object in the same way.

19. We can make3 different dictionaries and store them in a List:

dictionary1={…}

dictionary2={…}

dictionary3={…}

List\_of\_Dictionary=[dictionary1, dictionary2, dictionary3]

But, when it comes to storing a dictionary within another dictionary:

Dictionary\_of\_Dictionaries={dictionary1, dictionary2, dictionary3} \_ DOES NOT WORK

#>>>TypeError: unhashable type: 'dict'

This happens as you cannot use a mutable object (such as a distionary,variable or lists) as a key in a dictionary.

**A List in a Dictionary :**

20. We can use a List as a value in a dictionary like for a dictionary: pizza:

pizza={

‘crust’:’thick’,

‘toppings’:[‘mushrooms’,’extra cheese’],

}

You can nest a list inside a dictionary anytime you want more than one value to be associated with a key.

favorite\_languages = { 'jen': ['python', 'ruby'],

'sarah': ['c'],

'edward': ['ruby', 'go'],

'phil': ['python', 'haskell'],

}

for name, languages in favorite\_languages.items():

print("\n" + name.title() + "'s favorite languages are:")

for language in languages: print("\t" + language.title())

#>>>

Jen's favorite languages are:

Python

Ruby

Sarah's favorite languages are:

C

Phil's favorite languages are:

Python

Haskell

Edward's favorite languages are:

Ruby

Go

**A dictionary inside a Dictionary:**

21. You can use a dictionary of key-value pairs to be a value of a key in a dictionary.

eg: Making alist of users of a website with the key: username and the value: A dictionary that has their first name, last name and location.

users = {

'aeinstein': {

'first': 'albert',

'last': 'einstein',

'location': 'princeton',

},

'mcurie': {

'first': 'marie',

'last': 'curie',

'location': 'paris',

},

}

for username, user\_info in users.items():

print("\nUsername: " + username)

full\_name = user\_info['first'] + " " + user\_info['last']

location = user\_info['location']

print("\tFull name: " + full\_name.title())

print("\tLocation: " + location.title())

Python stores each key(‘aenstein’, ‘mcurie’) in the variable username .

Python stores the dictionaries associated with each key(or username) in the variable user\_info.

Using the for loop, we first print the usrename(key)

and then we print what we know about the user:

full name

location

**Chapter7: User Input and While Loops:**

1. input() function : The input function pauses your program for the user to enter some text. Once Python receives the user’s input , it stores it in a variable to make it convenient to work with.

🡪The input function takes in one argument: the prompt , which tells the user what we want them to do.

once the prompt tells the users what to do, the users feed in their input.

Further Python lines utilize the input(acc. to the program)

🡪FORMAT:

variable\_name=input(prompt)

🡪Add a space at the end of your prompt to separate the prompt from the user input(in the terminal)

🡪The prompt can be saved in a variable and be passed onto the input() function.

This allows you to build your prompt over several lines and provide a clean input statement.

2. prompt+= : prompt = prompt +…..

**Using int() to accept numerical input :**

**3. Note: When you use the input function , Python interprets everything you enter as a string.**

When you try to use input() to do a numerical comparison:

Python produces an error because it cannot compare a string to an integer.

We can resolve this using the int() function.

-🡪Remember, when using input to do a numerical comparison or mathematical operation: After variable\_name=input(prompt)

, always add the line: *variable\_name=int(variable\_name)* to tell Python that the input the user has entered in is an integer.

eg: >>> age = input("How old are you? ")

How old are you? 21

‘’’Till here Python interprets 21 as a string ‘21’ ‘’’

>>> *age = int(age)*

But after this line, Python knows that the input entered is an integer. Not a string.

>>> age >= 18

True

Note that if you

i)do not enter the line : age = int(age)

ii) enter the age bigger than 18

🡺 You will still get True. Why? Cause Python understands that a string is bigger than a number. Even if you enter you age as 17 or 9! If it is a string , it will be bigger than your number : 18.

**The Modulo Operator:**

4. % : divides the first number by the second number and returns the remainder.

This operator only returns the remainder. Not the quotient.

For the quotient: we use / .

eg: 4%3

#>>>1

If the modulo of two numbers is 0, then the first number is divisible by the second number.

eg: 100%10

#>>>0

**Accepting Input in Python 2.7:**

5. Use raw\_input() function is used when prompting for input instead of input(). This function interprets all input as a string , just as input does in Python3.

**Introducing While Loops:**

6. While Loop: The for loop takes in blocks of code or actions , and executes each action once for each item in the data structure. The while loop runs as long as or *while* a certain condition is True.

**The while loop in action:**

7. current\_number = 1

while current\_number <= 5:

print(current\_number)

current\_number += 1

#>>>This will output the numbers from 1-5.

FORMAT:

while (some condition is True):

<Blocks of Code>

**Letting the Users choose when to quit.**

**# When we want ask question repeatedly until some condition (message !=’quit’),**

**# We use the raw\_input function INSIDE THE WHILE LOOP.**

8. The way to do this is: To create a *quit value* and then keep the program running as long as the user has not entered the *quit value.*

For eg:

prompt = "\nTell me something, and I will repeat it back to you:"

prompt += "\nEnter 'quit' to end the program. " Here, the quit value is : ‘quit’

message = "" This line is just added so that below when the while loop check the value of message, it should not be ‘quit’. This makes it enter the while loop and prompt the user for input.

while message != 'quit':

message = input(prompt)

print(message)

🡪Usually, the prompt includes the *quit value.* This lets the user know how to quit the program before even playing it.

**Using a Flag:**

9. Usually, there is more than one reason that stops a program from running.

If many possible events might occur to stop the program, trying to test all these conditions in on *while* statement becomes complicated.

For a program : that should run ONLY as long s as Some conditions remain True :

You can define *one variable* that determines whether or not the entire program is active.

This *variable* is called **flag.**

The flag acts as a signal to the program ,

We can write our program so the run as long as the flag is set to True and stop when several events set the flag to False.

🡪 As a result our whole program needs to check only one condition:

Whether or no the flag’s value is True.

🡪Then all the other events that will set the flag to false: Can be neatly organized in the rest of the program.

eg:

active = True

while active: Notice, no comparison is made in the while statement. The logic for the program to keep on running is taken care of in the other parts of the program.

As long as the active variable remains True, The program will keep on running.

message = input(prompt)

if message == 'quit':

active = False This changes the value of active to False so the while loop is not entered again.

else: print(message)

What was the whole point of changing the code, it does the same thing?

#>> NOW! we can add as many elif blocks that might cause the program to stop. All we will need to enter in the block of code will be active=False.

And the loop will stop running.

**Using break to exit the loop:**

10. break: To exit the while loop immediately without running any code in the loop. regardless of the result of any conditional test, use the break statement.

The break statement directs the flow of your program ;

you can use it to control which lines of code are executed and which aren’t.

This way: You control the code that is executed that you want to execute , when you want to execute it.

*Mainly break is useful in an infinite loop : when the user enters the quit value.*

As a loop that starts with while True will run forever till it reaches a break statement.

prompt = "\nPlease enter the name of a city you have visited."

prompt+= "\nEnter 'quit' when you are finished. "

while True:

city = input(prompt)

if city == 'quit':

break

else:

print("I'd love to go to " + city.title())

**Using continue in a Loop:**

11. continue: Rather than breaking out of the loop entirely without executoing the rest of its code, you can use the *continue* statement to return to the beginning of the loop based on the result of the conditional test.

PRINTING ODD NUMBERS:

print("Page126 counting.py")

current\_number = 0

while current\_number<10:

current\_number+=1

if current\_number % 2 == 0:

continue

else:

print(current\_number)

The continue statement tells Python to skip the rest of the loop and return to the first line of the loop. Start Again.

**Avoiding Infinite Loops:**

**🡺12.** Just make sure at least one part of your program can make the loop’s condition *False* or cause it to reach the *break* statement.

**13. Three Types of Ways To exit a While Loop:**

i) Break

ii) Use Flag , set it false where you would like to break the loop.

iii) Conditonal test :

Start with the basic conditional test attached to the while loop( if this is not True, Python will not enter the while loop)

Within the loop , using an if –else chain:

Set a condition that if the MAIN CONDITIONAL TEST becomes False at any point , continue (that means return to the first line of the loop , where the main conditional test will fail and the loop will break)

In the else block, put code that should be executed if the main conditional test does not fail.

print("\n7-6: Three Exits: ") # Very important , shows the differnce between the 3 types of way to exit a while loop

i) BREAK:

# print("Using break to exit the loop , when the user enters the 'quit' value.")

prompt = "Write a topping you would like to have on your pizza!"

prompt += "\nKeep writing till you're done."

prompt += "\nEnter 'quit' when full! "

toppings =[]

while True:

topping = raw\_input(prompt)

if topping != 'quit':

print("Ok, " + topping.title() + " is added to your pizza. ")

toppings.append(topping)

elif topping == 'quit':

break

print("Here are the final list of topping on your pizza: ")

for topping in toppings:

print(topping)

ii) FLAG: set it false where you would like to break the loop.

print("\nUsing an active variable to control how long the loop runs. ")

active = True

while active == True:

topping = raw\_input(prompt)

if topping != 'quit':

print("Ok, " + topping.title() + " is added to your pizza. ")

toppings.append(topping)

elif topping == 'quit':

active = False # when quit is entered , avtive is set to false instead of breaking the loop.

print("Here are the final list of topping on your pizza: ")

for topping in toppings:

print(topping)

iii) Conditonal test :

Start with the basic conditional test attached to the while loop( if this is not True, Python will not enter the while loop)

Within the loop , using an if –else chain:

Set a condition that if the MAIN CONDITIONAL TEST becomes False at any point , continue (that means return to the first line of the loop , where the main conditional test will fail and the loop will break)

In the else block, put code that should be executed if the main conditional test does not fail.

print("\nUsing a conditional test to control how long the loop runs. ")

topping = "" # Just so that the loop is entered

while topping != 'quit': #Starts really checking when after first topping

topping = raw\_input(prompt)

if topping == 'quit':

continue # This line of code will make the loop go back to the first line of the

# while loop WHERE the conditional test will fail and the loop will BREAK.

else:

print("Ok, " + topping.title() + " is added to your pizza. ")

toppings.append(topping)

print("Here are the final list of topping on your pizza: ")

for topping in toppings:

print(topping)

14. A for loop is effective for looping through a list , but you shouldn’t modify a list inside a for loop because Python will have trouble keeping track of the items in the list.

To modify a list as you work through it, use a whole loop.

Using a while loop with lists and dictionaries allows you to:

collect,

store,

and organize lots of input to examine and report on later.

15. while *data\_type:* This line means: while the data\_type is not empty, enter the loop.

**Functions:**

**greet\_user()**

1. Functions: named blocks of code are designed to do one specific job.

2. Call : When you want to perform a particular task that you’ve defined in a function, you *call* the name of the function responsible for it.

A function call tells python to execute the function.

3 . def : Used to define a function.

4. *parantheses* after the function is defined hold the information required by the function to do its job.

5. definition line of a function ends with a colon , don’t forget the colon : SyntaxError

6. After the definition line, comes the *docstring* which explains what the function will do. 🡪 Enclosed in triple quotes , Python looks for these docstrings when it tries to generate documentation for the functions in your programs.

7. *body:* Any indented lines within the function makes for the *body* of the function.

**Passing Information to a Function:**

8. **greet\_user(username)**

**Arguments and Parameters:**

9. parameter: refers to the variable(s) used within the parentheses when defining the function.

10. argument : refers to the data(string , list, dic, etc.) used to replace the variable withing the paratheses. The arguments are the thing that is actually utilized, worked upon by the function.

**Passing Arguments: *pets.py***

11. *positional arguments :* These arguments need to entered in the same order as the parameters.

You can use as many positional arguments as you want. Python works through the arguments you provide when calling the function and matches each one with the corresponding parameter in the function’s definition.

Just note that **order matters in the positional arguments. Don’t mix it up.** Or else you will see output you don’t want to.

12. *keyword arguments:* specifying the parameter\_name = value

These type of arguments are a name-value pair that you pass to your function.

This way we tell Python explicitly , which parameter needs to be matched with what argument/value.

🡺Just be sure to use the exact names as used in the parameters.

13. Default Values: When defining a function , you can define a *default value* for any parameter.

If the argument for a parameter is provided in the function call , Python uses the argument value. Else , it will use the default value.

SO , i if you define a default value for a parameter , you don’t necessarily need to include an argument for that parameter in the function call.

*When you use default values, any parameter with a default value needs to be listed after all the parameters that don’t have default values. This allows Python to continue interpreting positional arguments correctly. OR else, you can directly use key\_word arguments if you don’t put the default values at the end of the parameters.*

**Equivalent Function Calls:**

14. *t doesn’t really matter which calling style you use. As long as your function calls pro- duce the output you want, just use the style you find easiest to understand.*

*Note: When all arguments are default , use only keyword arguments ( just t be descriptive) You can use positional also.*

**Avoiding Argument Errors:**

15. Unmatched arguments: occur when you provide fewer or more arguments than a function needs to work with.

**Return Values:**

16. return : A function does not need to always have to display its output directly.

Instead, it can process some data and then return a value or a set of values.

The value a function returns is known as *return value.*

🡺The return statement takes a value from inside a function and sends it back to the line that called the function.

🡺Return values allow you to move much of your program’s grunt work into functions, which can simplify the body of your program.

🡺You can just save the return value of the function in a variable and access / use it later.

**Returning a Simple Value:**

**get\_formatted\_name.py**

**17. How to make an argument optional:**

set it’s default value as a blank string. So this way, if the user enters the argument for the optional parameter (*like middle name in get\_formatted\_name.py*) . It can be used , and if not entered : can be ignored.

Optional values allow functions to handle a wide range of use cases while letting function calls remain as simple as possible.

**18. Returning a Dictionary:**

A function can return any kind of value you need it to , including more complicated data structures like lists and dictionaries.

person.py , def build\_person(first\_name,last\_name):

**19. Passing a list**

**greet\_users.py**

**20. Modifying a list in a Function:**

When you pass a list to a function, the function can modify the list.

The changes made to the list within the body of the function are permanent, allowing you to work efficiently even when you’re dealing with large amounts of data.

🡺 An advantage of function is that the code of the function needs to be modified: We can make changes in the body of the function andthose changes will affect everywhere where the function has been / will be called.

**21. Preventing a function from Modifying a List:**

If you don’t want your function to alter the original list : You can send ion a copy of the original list.

*function\_name(list\_name[:])*

***Its usually better to pass in the original list. Why? Cause when you’re working with large lists : IT takes more time and memory to create a separate copy list.***

**22. Passing an Arbitary number of arguments:**

Sometimes you don’t know ahead of time how many arguments a function needs to accept.

Fortunately, Python allows a function to collect arbitrary number of argument from the calling statement.

For eg: A function that builds a pizza.

Now, How do you make a function accept arbitrary number of arguments?

🡺 **By entering \* before parameter\_name in the function definition.**

This asterix tells Python to create a tuple and pass whatever values it receives into this tuple.

Note: Python will pack the arguments in a tuple only even if it receives only one argument.

**build\_pizza.py**

**23. Mixing Positional and Arbitary Arguments:**

Just like default value/optional parameters are placed at the end…

If you want to mix positional and arbitrary arguments : position the arbitrary argument at the end.

Python matches Positional and Keyword Arguments First !

And then goes to the default parameters / arbitrary parameters.

**make\_pizza.py**

Python first takes in the Positional and Keyword arguments : matches them.

The rest of the arguments : are put into the tuple(arbitrary\_number of argument’s parameter).

**24. Using Arbitary Keyword Arguments**

You cannot always know the type/kind of information to put in a function.

In this case , you can write functions that accept many key-value pairs as the calling statement provides.

How to tell python that we want to take in artibatary number of keyword arguments ?

\*\*

The double asterisks before the *parameter\_name* cause Python to create an empty dictionary called *parameter\_name :* Just like a tuple is created with a single asterisk with the respective parameter name. So this dictionary packs in the keyword arguments. Within this function , you can access the name-calue pairs in the user\_info just as you would for any other dictionary.

To access the elements of the dictionary , think of it like a normal dictionary only.

**build\_person()**

**25. Storing Your functions in Modules:**

Advantage of Python : The way it seperates blocks of code from your main program.

By using descriptive name for your functions , your main program will be much easier to follow.

26 . Module: You can go further by storing your function in a separate file called a *module.*

*🡪 And then importing that module in your main program.*

27. *import:* An import statement tells Python to make the code in a module available in the currently running program file.

Basically, storing a function in a module , allows you to hide the details of your programs code and focus on its higher level logic.

It allows you to reuse functions in many other programs.

Knowing how to import function allows you to use libraries of functions that other programmer have written.

**Importing an Entire Module:**

1. Make a function in a .py file.

2. In the same directory , make another python file and write

28. 🡺 import name\_of\_module (without .py)

🡪All import statements are supposed to be at the beginning of the file.

The import statement (when used to import an entire module) import Or makes available All The Function is that module to the current file.

See functions.py , you have imported pizza.py in it.

Any function in pizza.py will be available to functions.py

One we import the entire module , Every function in the module is available to the current file with this syntax:

*29. module\_name.function\_name()*

***30. Importing Specific Functions:***

*🡺 From module\_name import function\_name*

***31. Importing multiple functions from a module:***

*🡺 From module\_name import function\_name1, function\_name2, function\_name3*

With 30, 31, we do not need to use a dot notation as we explicitly tell Python which function to import.

**32. Using as to Give a Funtion an Alias:**

If the name of the function you’re importing might conflict with an existing name in your program , or any reason you want that function to have a nickname :

You can use *as.*

*🡺 From module\_name import function\_name as f\_n*

***33.* Using as to Give a Module an Alias:**

**🡺** *import module\_name as m\_n*

**34. Importing All Functions in a Module:**

**🡺** *from pizza import \**

*\* tells Python to import all the functions from that module.*

Altough this is not suggested as , it is least descriptive: It does not require the dot notation to use the functions.

Ideally , :

Import the functions you want

Use the dot notation if you want to import the entire module

Being descriptive is key.

**35. Styling Functions:**

i) lowercase

ii) use underscores

iii) descriptive names

iv) comments /docstrings to aid what the fucntion’s doing or will do.

v) Use the docstring to tell what the function is used for

vi) Use comments to explain how its doing it.

vii) In a default value parameter: no spaces should be used.

viii) No spaces while entering keyword arguments.

**CLASSES:**

1. Object Oriented Programming:

You write *classes* : represent real world things and situations

You create *objects(instances)* based on these classes.

When you write a *class* you define the general behavior that a whole category of *objects(instances)* can have.

2. Instantiation: Making an object from a class.

3. The *class* tells python how to make an object(instance). After our class is written, we make individual instances , each having the common behavior described in the class and some unique characteristics for each instance.

4. FORMAT OF A CLASS:

class Dog():

"""A simple attempt to model a dog."""

def \_\_init\_\_(self,name,age):

"""Initialize name and age attribute"""

self.name=name

self.age=age

def sit(self):

"""Simulate a dog sitting in response to a command"""

print(self.name.title()+" is now sitting.")

def roll\_over(self):

"""Simulate rolling over in response to a command"""

print(self.name.title()+" rolled over!")

i) By convention, Capitalised names refer to classes in Python.

ii) Then we write a doc string describing what the class will do.

5. The \_\_init\_\_ function:

A function that is a part of a class is called a method .

The init() method is a special method that python runs whenever we make a new instance of the class.

This method has 2 underscores before and after init to prevent Python’s default method names from conflicting with your method names.

6. self : The self parameter is defined in the method definition, and it must come first before any other parameters. It must be included in the definition cause when Python calls the init method later(to create an instance of the class) , the method call will automatically pass the self argument : WHICH IS A REFERENCE TO THE INSTANCE ITSELF.

IT GIVES THE INSTANCE ACCESS TO THE METHODS AND ATTRIBUTES OF THE CLASS.

Whenever we need to make an instance from a class: We will only provide values for the rest of the parameters , self is passed in automatically so we do not need to pass it in.

7*.* attributes*: self.name=name*

*self.age=age :* These two variables defined each have the prefix self.

🡪 Any variable prefixed with self is available to any method of the class.

🡪We’’ll also be able to access these variables through any instance created from the class.

-🡪listing this down is important to work with the attributes later. As every instance will have different values attached to the parameters.

in one instance, self.name may be equal to ‘willie’ whereas in another instance self.name may be equal to ‘lexus’.

*self.name=name* takes the values stored in the parameter *name* and stores it in the variable *name*  which is then attatched to the instance created.

Variables, which are accessible through instances, are called *attributes.*

8. methods: The methods we define within a class are available to the instances we define.

9. Making classes in Python 2.7: class Dog(object)

Write *object* within parentheses when creating a class.

**Making an Instance from a Class :**

10. Think of a class as a *set of instructions* for making an instance.

*my\_dog=Dog('willie',6)*

When Python reads this line, It calls the \_\_init\_\_ function within the class Dog with the arguments ‘willie’ and 6.

The \_\_init() function creates an instance(my\_dog) and sets the name and age attributes using the values we provided.

🡪Now,

name=’willie’

age=6

The \_\_init\_\_ method does not have a return statement but Python returns an instance.

🡪Naming convention is helpful here as we can usually assume that capitalized name like *Dog* refers to a class and lowercase *my\_dog­* refers to a single instance created from a class.

**Accessing Attributes:**

11. To access the attributes within an instance, we use the dot notation.

instance\_name.attribute\_name **OR** my\_dog.name

my\_dog.age

When writing the class: We wrote:

self.name= name

self.age=age

These lines help in accessing attributes later when we create an instance.

🡪These line are also important for accessing these values For the methods within a class.

Like sit and roll\_over need these attributes.

self here is the instance name(‘my\_dog).

name(‘willie’) is the value we put in for the parameter *name.*

age(‘age’) is the value we put in for the parameter *age.*

So when we write :

my\_dog.name , what we are writing is :

self.name

which is assigned to the value of *name.*

**Calling Methods:**

12. After we create an instance from the class (*Dog)* , we can use the dot notation to call any method defined in the class(*Dog).*

Format: instance\_name.method\_name(‘arguments’,’if’,’needed’)

Too call a method, give the name of the instance and the method you want top call, separated by a dot.

🡪my\_dog.sit()

🡪my\_dog.roll\_over()

**Creating Multiple Instances:**

13. You can create as many instance as you like as long as it has a unique variable name(*self)* **Or** it occupies a unique spot in a list or dictionary.

**Working with Classes and Instances:**

14. There are three ways in which you can modify the attributes associated with a particular instance.

i) modify the attribute directly through an instance.

ii) set the value through a method.

iii) write methods that update/increment attributes in a specific way.

**The Car Class:**

**Setting a Default Value for an Attribute:**

15. Every attribute(self.) in a class needs an initial value, even id that value is 0 or an empty string.

In some cases , such as setting a default value: it makes sense to specify this initial value in the body of the \_\_init\_\_() method ;

If you do this for an attribute: You do not need to include a parameter for that attribute.

MEANING: If you set a **default** value: such as 0: There is no use for a parameter for that attribute as the person making the instance might set the parameter to something other than 0. Since the attribute is intended to be a **default** value, it just needs to be included in \_\_init\_\_() .

self.**default\_variable\_name**=value(0/empty string)

**Modifying Attribute Values:**

**Modifying Attribute’s Value Directly:**

16. -🡪 instance\_name.attribute\_name = new value

my\_new\_car.odometer\_reading = 23

my\_new\_car.read\_odometer()

#>>> This car has 23 miles on it.

**Modifying an Attribute’s Value through a Method :**

17.

🡪 def update\_attribute\_name(self, new\_value):

“””Set attribute\_name to given value”””

self.attribute\_name = new\_value

**Incrementing an Attribute’s Value Through a Method:**

**🡪** def increment\_attribute\_name(self, increment):

attribute\_name += increment

**Inheritance:**

18.

**🡪** We always do not need to write a class from scratch. If the class you’re writing is a specialized version of another class we wrote, we can use *inheritance*.

🡪When one class *inherits* from another, it automatically takes on all the attributes and methods of the first class.

🡪The original class is called the *parent class.* And the new class is the *child class.*

🡪 The child class inherits every attribute and method from the parent class. But is also free to define new attributes and methods of its own.

**The \_\_init\_\_() method for a child class:**

19.

class ElecticCar(Car):

"""Represents aspects of a car, specific to electric vehicles"""

def \_\_init\_\_(self, make, model, year):

"""Initialize attributes of the parent class."""

super().\_\_init\_\_(make, model, year)

i) When writing a child class: The parent class should be in the current file and must appear before the child class in the file.

ii) The name of the parent class should be included in the parentheses in the definition of the child class.

eg: class ElecticCar(Car):

iii) The \_\_intit\_\_ method takes in the information required to make a *Car*(parent class) instance. That’s all it does, just takes in information. This line has nothing to do with connecting to the parent class except the fact that it takes in the same arguments.

The next line, super().\_\_init\_\_(make, model, year) forms the connection between the child class and parent class.

def \_\_init\_\_(self, make, model, year):

iv) The *super()* function is a special function which helps in making connections between the parent class and the child class.

🡪 super().\_\_init\_\_(make, model, year)

This line tells Python to call the \_\_init\_\_() method from the *ElectricCar’s* (child class’s) parent class : which gives an instance of *ElectricCar* acces to all the attributes of its parent class.

🡪 In other words, This line tells Python to call the \_\_init\_\_() method from the child class’s parent class which will give an instance of the child class ,access to all the attributes and methods of the parent class.

🡪Remember, in this line *self* is not included in the parentheses.(FIND OUT WHY)

So, when we make an instance of the child class(*ElectricCar)* , it calls the \_\_init\_\_() method inthe child class which, in turn: calls the \_\_init\_\_() method defined in parent class(*Car).*

The child class’s(*ElectricCar’s) instance* works just like an instance of parent class (*Car).*

**Inheritance in Python 2.7:**

20. The super() function needs two arguments: a reference to the child class and the self object. These arguments are necessary to help Python make proper connections between the parent and child classes. When you use inheritance in Python 2.7, make sure you define the parent class using the object syntax as well.

🡪Till now, we have just made sure that the child class behaves the same way a normal car does. It is time to add attributes and methods unique to ElectricCar.

**Defining Attributes and Methods for the Child Class:**

21.

When you define attributes and methods specific to a child class, the attributes and methods are accessible to all instances of the child class but are not accessible to instances of the parent class.

**Overriding Methods from the Parent Class:**

22. When you use inheritance, you can make your child classes retain what you need and override anything that you don’t need from the parent class.

To override any method from the parent class that doesn’t fit what you’re trying to model with the child class : Just overwrite that method with another method that does fit into your child class.

Python will then disregard the method of the parent class, and only focus on the method provided in the child class.

For eg: *Car(*parent class) has a fill\_gas\_tank method, which does not make any sense in the *ElectricCar(*child class) .

**Instances as Attributes:**

23. You can break your classes into smaller classes that work together.

Modelling something from the real world can lead to a large number of attributes and methods . This makes the code lengthy.

eg: In *ElectricCar ,* we may lead to adding to many attributes and methods for the battery. When we see this happening, we can stop and move those attributes to a separate class called *Battery.*

*-🡪*THEN WE USE A BATTERY INSTANCE AS AN ATTRIBUTE IN THE *ElectricCar CLASS.*

So, first we define a class battery.

->give it an \_\_init\_\_ method

->Include out methods in it.

Then , we use the whole class as an attribute of the bigger class(*ElectricCar):*

self.battery = Battery()

🡪self.sub\_class = Sub\_Class()

This line tells Python to create a new instance of *Battery* and store that instance in the attribute self.battery …

Now, whenever an \_\_init\_\_ () method of *ElectricCar* is called, it will in turn call the \_\_init\_\_ method of the parent class AND THEN any *ElectricCar* instance will now have *Battery* instance created automatically.

🡪Usually, we just access an attribute of a class by

instance\_name.attribute\_name

But in this case, out attribute is a class which has further methods/attributes in it.

So to access that:

instance\_name.attribute\_name.(method/attribute\_name)

🡪my\_tesla.battery.describe\_battery()

🡪Creating a a subclass that is part of a bigger class is good so you can separate everything that is a part of subclass from the bigger class. It helps in keeping code clean and uncluttered.

**Modelling Real-World Objects:**

As you begin to model more complicated items like electric cars, you’ll wrestle with interesting questions. Is the range of an electric car a property of the battery or of the car? If we’re only describing one car, it’s probably fine to maintain the association of the method get\_range() with the Battery class. But if we’re describing a manufacturer’s entire line of cars, we proba- bly want to move get\_range() to the ElectricCar class. The get\_range() method would still check the battery size before determining the range, but it would report a range specific to the kind of car it’s associated with. Alternatively, we could maintain the association of the get\_range() method with the bat- tery but pass it a parameter such as car\_model. The get\_range() method would then report a range based on the battery size and car model.

This brings you to an interesting point in your growth as a program- mer. When you wrestle with questions like these, you’re thinking at a higher logical level rather than a syntax-focused level. You’re thinking not about Python, but about how to represent the real world in code. When you reach this point, you’ll realize there are often no right or wrong approaches to modeling real-world situations. Some approaches are more efficient than others, but it takes practice to find the most efficient representations. If your code is working as you want it to, you’re doing well! Don’t be discour- aged if you find you’re ripping apart your classes and rewriting them several times using different approaches. In the quest to write accurate, efficient code, everyone goes through this process.

**Importing Classes:**

24. Python lets you store classes in modules and then import the classes you need into your main program.

**Importing a Single Class: car.py / my\_car.py**

25. You should write a docstring for each module that you create. This module-level docstring briefly describes the contents of the module. So you can quickly know what is the module.

Module meaning a file containing a single class, that can be imported into another file/module.

🡪Simple logic, when we create a class in a different module and then, in a new module:

from module\_name(without .py) import Class\_Name

🡪from car import Car

🡪The whole class gets imported into the new module, although it does not show in the new module🡪Keeps code clean.

🡪Once your classes are made, store them in different modules and focus on the higher-level logic of your main program.

**Storing Multiple Classes in a Module: car.py/my\_electric\_car.py/**

26. You can store as many classes as you need in a single module, although each class in a module should be related somehow.

For eg: In car.py: You can store the class Car, ElectricCar, Battery…etc.

But don’t store Car and Restaurant in the same module.

**Importing Multiple Classes from a Module: car.py/importing\_multiple\_classes.py**

27. You can import multiple classes from a module by separating each class with a comma. Once you’ve imported the necessary classes, you’re free to make as many instances of each class as you need.

**Importing an Entire Module: importing\_an\_entire\_module.py**

28. You can import an entire class and then access the classes you need using dot notation.

This is simple and results in easy readability.

Every instance you create , it will include the module name in its definition.

instance\_name = module\_name.Class\_Name()

**Importing All Classes from a Module:**

29. You can import every class from a module using the following syntax:

from module\_name import \*

🡪This method is NOT recommended for two reason:

i) Its helpful to read which Classes the following program will be using.

🡪If you need to import all the Classes form a module , the better way is to use the *module\_name.Class\_Name syntax.*

ii) It may cause some naming conflicts which are hard to diagnose.

**Import a Module into a Module:**

30. Sometimes, you’ll want to spread out your classes over several modules to keep any one file from growing too large and avoid storing unrelated classes in the same module.

When you store your classes in several modules, you may find that a class in one module depends on a class in another module. When this happens, you can import the required class into the first module.

**Finding Your Own WorkFlow:**

31. When You’re starting off:

Try doing everything in one long file and then structure your code withing different modules.

**The Python Standard Library: collections\_Ordered\_Dict.py**

32. You can use a function or a class in the standard library by including a simple import statement at the top of your file. Let’s look at one class, OrderedDict from the module collections.

If you want to keep a track of the order in which the key-value pairs are added, use OrderedDict.

🡪How to use OrderedDict:

1. from collections import OrderedDict()

2. dictionary\_name = OrderedDict()

When we write this code, we create an instance of OrderedDict() and store it in the variable dictionary\_name.

3. dictionary\_name[key] = value

Repeat step 3 till you add all elements of the dictionary.

Note: This will not work if you simple define the dictionary after

step 2.

You have to add elements to the dictionary one by one or else,

dictionary\_name = OrderedDict() gets overwritten by the simply written Dictionary.

**Left: 1.5 hrs.**

**9-8 to 9-15 : 8 questions.**

**Styling Classes**

**Files And Exceptions:**

You’ll learn:

i) How to handle errors so your programs don’t crash whey they encounter unexpected situations.

ii) About *exceptions*, which are special objects Python creates to manage errors that arise while a program is running.

iii) about *json* module , while allows you to save user data so it isn’t lost when your program stops running.

iv) how to make your programs more applicable, usable and stable.

**Reading from a File:**

1. Reading from a file is particularly useful in data analysis applications, but it’s also applicable to any situation in which you want to analyze or modify information stored in a file.

🡪When you want to work with information in a text file, the first step is to read the file into memory. You can read the entire contents of the file , or you can work through the file one line at a time.

**Reading an Entire File: 1. pi\_digits.txt**

-🡪with open('1.pi\_digits.txt') as file\_object:

contents = file\_object.read()

print(contents.rstrip())

2. open() function: To do anything with a file, even if its just printing its contents , you first need to *open* the file to acess it.

The open() function needs one argument: the name of the file you want to open.

Python looks for the file in the directory where the program that’s currently being executed is stored.

In this example, *file\_reader.py* is currently running, so Python looks for *pi\_digits.txt* in the directory where *file\_reader.py* is stored

The open() function returns an object representing the file.

Here, open('pi\_digits.txt') returns an object representing *pi\_digits.txt*.

3. *file\_object:* Python stores this object(the file you open) in whatever variable we name after the keyword ‘as’ ..Here, its *file\_object.*

4. *with:* The keyword ‘with’ closes the file once access to it is no longer needed. It’s not always easy to know when you’re done using a file. Python does that for you.

5. *close():* Normally, we should always *close()* a program that we have o*pen()* ,but sometimes a bug may stop the program from closing. That is why we use the keyword ‘with’ .

This may seem trivial, but not closing a file may corrupt it/ or lead to data loss.

6. *read():* The second line contains the read() method. This line :

contents = file\_object.read()

stores the contents of the file\_object in one long string – contents.

🡪 read() returns an empty string when it reaches the end of the file. This empty string shows up as a blank line and is added to the output. That is why we use the *.rstrip* method when printing the contents.

When we print(contents) : we get the entire file back.

**File Paths:**

To get Python to open files form a directory other than the one where program file is stored , you need to provide a file path , which tells Python to look in a specific location in your system.

7. Realative file Path:

Tells Python to look for a given location relative to the directory where the currently running program file is stored.

Assume You are looking for a .txt file in a text\_files folder :

The current directory is python\_work : which contains the folder text\_files.

*with open(‘text\_files/file\_name.txt’) as file\_object*

In windows: use a backslash instead of a front slash.

FORMAT:

*with open(‘folder\_name’/’file\_name’) as file\_object*

8. Absolute file path:

You can also tell Python where the file is on your computer irrespective of where the program that’s being executed is stored.

Absolute paths are generally longer than relative paths so its helpful to store them in a vairbable and pass it in open().

🡪Take the file path from getting info about the folder in the GUI.

🡪Use terminal

**Reading Line by Line:**

**🡪**Examine each line of the file

🡪look for certain info

🡪modify text in file

For example, you might want to read through a file of weather data and work with any line that includes the word *sunny* in the description of that day’s weather. In a news report, you might look for any line with the tag <headline> and rewrite that line with a specific kind of formatting.

*file\_reader.py* 

filename = 'pi\_digits.txt'

with open(filename) as file\_object:

for line in file\_object:

print(line)

9.

🡪This time , we get even more of the blank lines , when we use a for loop.

These blank lines appear because an invisible newline character is at the end of each line the text\_file.

The *print* statement adds its own newline each time we call it, so we end up with two newline character at the end of each line.

One from file and one from *print* statement.

Using the rstrip() on each line the print statement eliminates these extra blank lines.

**Making a List of Lines from a File:**

When you use *with*, the file object returned by *open()* is only available inside the *with* block that contains it.

If you want to retain access to a file’s contents outside the *with* block ,

Block here means , everything indented within the *with open as* line will have access to the file:

you can store the file’s lines in a list inside the block and then work with that list.

You can process parts of the file immediately and postpone some processing for later in the program.

filename = 'pi\_digits.txt'

with open(filename) as file\_object:

lines = file\_object.readlines()

for line in lines:

print(line.rstrip())

10. readlines() method :

taken each line from the file and stores it in a list.

This list is stores in a *list\_name* which we can continue to work with after the with block ends.

Now each item in the list will correspond to a line in the file.

**Working with a File’s Contents: pi\_string.py**

Once you;ve read a file into memory :

You can do whatever you want with the data.

*When Python reads from a text file, it interprets all text in the file as a string. If you read in a number and want to work with that value in a numerical context, you’ll have to convert it to an integer using the int() function or convert it to a float using the float() function.*

No new concept: Just using the readlines() method and then appending each line to pi\_string to create a string with pi’s value to 30 decimal places.

11. float() function:

Return a floating point number constructed from a number or string *x*.

Basically , works like the int() function , used specially for decimal numbers.

Taken in a string and return the integer value of the string.

**Large Files: One Million Digits:**

12. Python has no inherent limit to how much data you can work with, you can work with as much data as your system’s memory can handle.

See pi\_string( You stored pi’s value to a million decimal places and then promp[ted the user for his/her birthdate in ddmmyy form)

See 10-1,10-2

13. **Writing to a File:**

One of the simplest ways to save data is to write it to a file.

When you write text To a file , the output will still be available after you close the terminal containing your program’s output.

You can examine output after a program finished running and you can share the output files with others as well.

You can also write programs that read the text back into memory and work with it again later.

**Writing to an Empty File:**

14. To write text to a file:

You need to call *open()* with a second argument telling Python that you want to write to a THE file.

**write\_messages.py**

file\_name = 'programming.txt'

with open(file\_name , 'w') as file\_object:

file\_object.write("I love programming.")

*15. write mode: with open(file\_name , 'w') as file\_object: 🡺*This line tells python to open file\_name in the write mode.

*🡪write mode* erases the contents of the file before returning the file\_object.

16. You can open the file in 4 modes:

1. write mode (‘w’)

2. read mode(‘r’)

3. append mode (‘a’)

4. read and write (‘r+’)

If you omit the mode argument: Python opens the file in read-only mode by default.

17. This file will have no terminal output: All it does is , writes the text you enter in the file\_name.

You can :

Open it()

write new text in it

copy from it

paste to it

and so forth (Basically , basic text functions you can perform)

*Python can only write strings to a text file. If you want to store numerical data in a text file, you’ll have to convert the data to string format first using the str() function.*

**Writing Multiple Lines:**

18. The *write()* function does not add newlines to the text you write.

So if you write more than one line without including newline characters , your file may not look the way you want it to.

🡪Just use \n and \t to create white spaces when writing into a file.

**Appending to a file:**

19. If you want to add content to a file instead of wrting over the existing content, you can open the file in *append mode.*

When you open a file in *append mode:*

Python DOES NOT ERASE the file before returning the file\_object.

Any lines you write , will be added to the existing text : towards the end of the file.

If the file does not exist, Python will create a new file.

with open(file\_name , 'a') as file\_object:

file\_object.write("I also love finding meaning in large datasets.\n")

file\_object.write("I love creating apps that can run in the browser.\n")

See 10-3: Add a guest\_name to a text file.

See 10-4: Prompt the user for a series of guest names: Greet and add guest name to the guestbook. Define a quit value.

See 10-5: Prompt the user for a reason the like programming , define a quit value , add reason to text file.

**Exceptions:**

20.

Python uses special objects called *exceptions* to manage errors that arise during a program’s execution.

Whenever an error occurs that makes Python unsure what to do next, it creates an exception object.

If you write code that handles the exception: Your code will start to work just fine.

If you don’t handle the exception , the program will halt and show a traceback, which includes a report of the exception that was raised.

21.

*try-except blocks:*

A try-except block asks Python to do something, but it also tells Python what to do if an exception is raised.

When you use try-except blocks : Your programs will continue running even if an exception is raised OR in other words, if something goes wrong.

Instead od tracebacks, which are usually confusing to read, users will see friendly error messages that you write.

**Handline the ZeroDivisionError Exception:**

22. What is the zerodivisionerror message? If you try and divide a number by 0, it shows this error message.

ZeroDivisionError: division by zero

Now , using exception we will tell Python what to do if such an error occurs. So then when , such an error occurs:

We can e prepared.

**WHY is there no program that self-debugs the whole program?**

**Spot errors using exceptions and solve them automatically. Find out!**

**Using try-except Blocks:**

23.

When you think an error may occur , you can write a *try-except* block to handle the exception that might be raised.

You tell Python to try running some code , AND you tell it what to do if the code results in a particular kind of exception.

*try:*

*print(5/0)*

*except ZeroDivisionError:*

*print("You can't divide a number by zero!")*

Simple Logic: Now, how does this work?

If the code in try block works: Then, except block is skipped.

If the code in the try block does not work: Python shifts to the except block.

**Using Exceptions to Prevent Crashes: division\_....py**

24. Handling errors becomes exceptionally important when the program has more work to do after the error occurs.

🡪This happens often in programs that prompt the user for input.

If the program responds to invalid input appropriately , it can prompt for more valid input instead of crashing.

It’s bad that the program crashed, but it’s also not a good idea to let users see tracebacks. Nontechnical users will be confused by them, and in a malicious setting, attackers will learn more than you want them to know from a traceback.

For example, they’ll know the name of your program file, and they’ll see a part of your code that isn’t working properly. A skilled attacker can sometimes use this information to determine which kind of attacks to use against your code.

**The else Block:**

25.

We can make divisions.py more error resistant by wrapping the line that might produce the errors in *try-except* block.

The error occurs in the line that performs the division , so that’s where we’ll put the *try-except* block.

Note: Always do this: Whenever you want to place a try-except block: Place it at the point where its actually needed.

while True:

first\_number = raw\_input("\nFirst number: ")

if first\_number == 'q':

break

second\_number = raw\_input("\nSecond number: ")

if second\_number == 'q':

break

try:

answer = int(first\_number)/int(second\_number)

except ZeroDivisionError:

print("You can't divide by 0!")

else:

print(answer)

🡪The thing/expression/operation that is likely to fail is put in the *try block,*

which included only the code that might cause an error.

🡪The except block displays a friendly message instead of traceback If the error does occur.

🡪Any code that depends of the try block succeeding is added to the else block.

26.

By figuring all the types of erros that may arise :

You can write robust programs that continue to run even when they encounter invalid data and missing resources.

Your code will become:

i) resistant to user mistakes

ii) resistant to malicious attacks

**Handling the FileNotFoundError Exception: alice.py , alice.txt**

27.

i) file may not exist at all

ii) you maybe finding it in the wrong location

iii) you may have misspelled the entire file\_name

See alice.py : file\_name is incorrect : The correct file name is alice.txt.

file\_name = 'alice\_in\_wonderland.txt'

with open(file\_name) as file\_object:

contents = file\_object.read()

print(contents)

🡺 IOError: [Errno 2] No such file or directory: 'alice\_in\_wonderland.txt'

In this program: The error is created at the open() function:

So when we put the try-except block: It would be placed just above the open() function.

FileNotFoundError returns a NameError whenused with except:

So instead of that:

I used IOError:

file\_name = 'alice\_in\_wonderland.txt'

try:

with open(file\_name) as f\_obj:

contents = f\_obj.read()

except IOError:

msg = "Sorry, the file " + file\_name + " does not exist."

print(msg)

**Analyzing Text :**

28. split() method:

It taken in a string and returns a list of words in order.

Basically it adds a new item to the list whenever a space occurs in the string.

See **analyzingtext.py** : tried to replicate the split function.

29. In analyzingtext.py , we found the number of words in a file.

By:

1. reading the file : contents stores the the text of the file by reading it.

2. we use the split function on contents cause the read() method returns a string.

3. Once we had the list of words in contents : we use the len function on the list to find out the number of words in the list.

🡺All the code that will run if the file is actually found is stored in the else block. As else block stores what should be executed after the try block succeeds.

file\_name = 'alice.txt'

try:

with open(file\_name) as f\_obj:

contents = f\_obj.read()

print(contents)

except IOError:

msg = "Sorry, the file " + file\_name + " does not exist."

print(msg)

else:

#Count the approximate number of words in the file.

words = contents.split()

print(contents.split())

num\_words = len(words)

print("The file "+ file\_name + " has about " + str(num\_words) + " words.")

**Working with Multiple files:**

**word\_count.py** : The above code is stored in a function called count\_words so that its easier to run analysis for multiple books.

In word\_count.py :

I downloaded 3 books: LittleWomen, MobyDick and AliceInWonderland

🡪put their file\_names in a list called books.

🡪Then i ran a for loop to count the number of words in each book.

Now, the advantage of having an except block was :

I did not download Siddhartha, but put Siddhartha.txt in the list of books.

If i did not have an except block: Python would shown an error and never continued to show me the number of words in MobyDick and LittleWomen.

But as i had an except block: It shows a friendly error message and then continues to show me the number of words in MobyDick and LittleWomen.

**Failing Silently : UsingPass\_missing\_filees.txt**

30. pass: Sometimes you’ll want the program to fail silently. In other words, using pass will : you will not show any error message and move on.

You explicitly tell Python to do nothing in the *except* block. Python has a *pass* statement that tells it to do nothing in a block.

No traceback is produced and there is no output in response to the error that was raised.

If we use the *pass* statement in word\_count.py , we will not see any indication of *Siddhartha.txt* failing.

Another thing we can do is , we create a missing\_files.txt : So we append the name of the file to missing\_text.txt.

def count\_words(file\_name):

'''Counts the approximate number of words in a file'''

try:

with open(file\_name) as f\_obj:

contents = f\_obj.read()

except IOError:

**pass**

# with open('missing\_text.txt' , 'a') as file\_object:

# file\_object.write(file\_name)

else:

#Count the approximate number of words in the file.

words = contents.split()

num\_words = len(words)

print("The file "+ file\_name + " has about " + str(num\_words) + " words.")

count\_words('alice.txt')

books = ['AliceInWonderland.txt' , 'Siddhartha.txt' , 'MobyDick.txt', 'LittleWomen.txt']

for book in books:

count\_words(book)

**31. Decide Which Errors to Report:**

How do you know when to report an error to the user and when to fail silently?

1. If users know which text are supposed to be analyzed , they might appreciate a message informing them why some texts were not analyzed.

2. If users expect to see some results but don’t know which books are supposed to be analyzed , they might not need to know which books are not available.

🡺 Giving users information that they aren’t looking for can decrease the usability of your program.

Python’s error-handling structures give you fine-grained control over how much to share with the users when things go wrong.

Its upto you to decide how much information to share.

Well – written , properly tested code is not very prone to internal errors such as syntax or logical errors.

🡺But everytime your program depends on something external:

i) user input

ii) network connection

iii) the existence of a file

Chances of errors are high.

32. continue: If a try block fails and the except block executes but you still want to go back and try again till the try block runs: Put everything in a while loop and use *continue* to go back to the stating of the loop and try again.

See 10-6:

while True:

try:

first\_number = raw\_input("Give me two numbers and i will add them." +

"\nEnter 'quit' to quit program anytime."+

"\nFirst Number: ")

if first\_number == 'quit':

break

else:

first\_number = int(first\_number)

second\_number = raw\_input("Second Number: ")

if second\_number == 'quit':

break

else:

second\_number = int(second\_number)

except ValueError:

print("You wrote in a word. Type in a number next time.")

continue

else:

print(first\_number + second\_number)

'''What the try block does:'''

# First it tells the user what the program will do.

# Next, it tell the use that he/she can quit at anytime.

# Next, It asks for the first number.

# Now if the user enters: 'quit'

# The program will exit/break.

# Till here , if the user has entered a non-integer value except 'quit'..

# The program will go to the first else block(first\_number = int(first\_number)

# Here, it will catch the error (ValueError) , if there is any.

# Similarly: The process is same for second\_number.

# Anytime a ValueError comes due to entering non-integer value(except 'quit'):

# The program shift to the except block.

'''What the except block does'''

# Anytime a ValueError comes due to entering non-integer value(except 'quit'):

# The program shift to the except block.

# 1.the except block displays a friendly message telling the user to enter a number instead

# of a word next time.

# 2. continue: Goes back to the starting of the loop and prompts for first number again.

This goes on till both inputs are correctly entered. OR

the user can ‘quit’ anytime.

# Starts the whole program again.

'''What the else block does:'''

# If no error occurs, it prints the sun of the 2 numbers.

# if first\_number != 'quit':

# second\_number = raw\_input("Second Number: ")

# if second\_number != 'quit':

# second\_number = int(second\_number)

# print(first\_number + second\_number)

**In 10-7: Modify 10-6**

They asked to use a while loop and a continue statement, which i had already done in 10-6.

**In 10-8:**

I wrote a simple function which takes in a file\_name ,read it and prints out the contents.

Now, once i used the function in try-except block:

The try block printed the contents of the file\_name and if IOerror occurred :

The except block was executed telling “The file was not found”

Now, the only problem with the whole program was that i wanted to know which file was not found.

So in the read\_file function: I entered a line of code that prints the name of the file (weather it exists or not) in title font. The only is the file opened and read..

So this way :

Dogs1.Txt:

The file was not found.

**In 10-9:** I modified 10-7 to fail silently using *pass.*

**In 10-10:**

I counted the number of occurrences of “is” in AliceInWonderland.txt using the

33. *count method:*

*Format: str.count(word , start , end)*

I remember making a function to count all the occurences of a word/substring.

The same is done by the *count* method.

word – This is the substring you are searching for.

start : index where you want to start searching. Default value: 0

end: search ends on this index. Default value: last index

**Storing Data:**

**34. JSON MODULE:**

Many of your programs will ask users to input certain kind of information.

You might allow users to store preferences for a game or provide data for a visualization.

You will store these information in data structures such as lists and dictionaries.

When users close a program, you’ll almost always want to save the information they entered.

For this: We store data using the *json* module.

🡺The *json* module allows you to dump simple Python data structures into a file and load the data from that file the next time program runs.

🡺*JSON* can be used to share data between different Python programs.

🡺Even better , the *json* data format is not fixed to Python, data can be stored and shared among many different languages.

*The JSON(JavaScript Object Notation) format was originally developed for JavaScript.*

*However, it has since become a common format used by many languages , including Python.*

**Using json.dump() and json.load()**

35. json.dump()

**🡺number\_writed.py**

stores a set of numbers

🡪uses json.dump() : stores the set of numbers.

🡪takes in 2 arguments: i) piece of data to store

ii) file\_object it can use to store the data.

import json

numbers = [2,3,5,7,11,13]

filename = 'numbers.json'

with open(filename , 'w') as file\_object:

json.dump(numbers, file\_object)

🡪Its customary to use .json file extention when using json module.

So that the file is saved in *JSON format.*

🡪The data is stored in the format that looks exactly like *numbers,*

🡪numbers is the piece of data stored.

🡪 file\_object is the file\_object it can use to store the data.

🡪 json.dump() is exactly like the write method instead json.dump is a function and taken in 2 arguments.

🡺 **number\_reader.py**

reads the numbers back into memory

🡪uses json.load() to load the numbers.

import json

filename = 'numbers.json'

with open(filename) as file\_object:

numbers = json.load(file\_object)

print(numbers)

🡪json.load() is just like the read() method except it’s a function and takes in the file\_object to read as the argument.

**Saving and Reading User-Genrated Data :**

**🡪**If you don’t save your user’s data , you’ll lose it when your programs stops running.

🡪**remember\_me.py**

**🡪greet\_user.py**

These 2 programs :

The first one promprts the user for their name

The second one rememebers their name when they run the program again.

**Refactoring:**

**VERRY IMPORTANT CONCEPT:**

**🡺** Often you will realize that eventhough your code works?

You could imporove the code by breaking it up into a series of functions that have specific jobs.

This process is called *refactoring*.

Refactoring makes your code :

i) Cleaner

ii) Easier to understand

iii) Easier to extend

🡺 We can refactor remember\_me.py by moving the bulk of its lgic into one or more functions.

🡺 The focus of remember\_me.py is greeting the user, so lets move all out existing code into function called greet\_user():

🡺 This is how you refactor:

Notice parts of your code that can be small parts to the bigger puzzle you are solving.

Make those small parts into functions which can be called in a single line of code.

Make sure to notice logical parts of your code , don’t try to divide your problem into too many small parts.

Refactor chronologically. Meaning : When you were solving the Birthday Problem , you created one function , then if it cause a problem or required another function, you created that.. All of the functions worked together to solve the problem.

That what you do when you refactor.

=🡺 Next , try to divide the small tasks into even smaller tasks.

🡺 Good Practice: A function should return the value you are expecting OR it should return None.

🡺 THE MOST IMPORTANT LEARNING IN PYTHON REFACTORING IS THAT WHWNEVER YOU SEE REPETITION IN YOUR PROGRAM/FUNCTION , make a function for that! And Call IT!

MAKES YOUR CODE MORE READABLE, EASY TO USE.