* numpy (NUMeric Python): matrices and linear algebra
* scipy (SCIentific Python): many numerical routines
* matplotlib: (PLOTting LIBrary) creating plots of data

for . py to .exe “https://www.youtube.com/watch?v=lOIJIk\_maO4”

Computers calculations are of two types:

1. Embedded into the language like addition subtraction etc
2. The one said by the programmer.

Types of knowledge: declarative (statement of fact) and imperative (sequence of simple steps, flow control and a way to stop - recipe)

Way to find square root: take a guess no if that is not a close take the guess as number over guess and continue

Computers are machines that has either a fixed program type to store the recipe like calculators or the stored program type like central processing units which can take decisions and feature based usage where user can edit few lines and produce a different functionality.

Computer basic architecture: Control unit has a program counter that reads the series of codes (sored in memory) from step1 and send the info to ALU to perform required operations. The ALU keeps the track with memory in storing data into required variables and increments the program counter by +1 and the control unit reads the next line in memory and final output into the output device.

Allen Turing (computer scientist) said that 6 primitives: move-left, move-right, read, write, scan, do-nothing can make any computation possible. This makes the origin for a programming language and the high level has more ease of variable primitives that makes computation simple.

A computation done in one language can be equally possible in other languages too.

Set of primitives gives an expression in programming. Getting back to English, the words being primitives, in programming bool operators floats etc comes as primitives. An only one meaning has to persist with the code the user writes.

Types of errors:

1. syntactic errors: where syntax go wrong
2. Static sematic error: where the phrase is correct with different values. Ex: 3 + “HI” where we have two operands and an operator, but the data type is wrong.
3. Ambiguity error: where more than one possible outcome is observed.

PYTHON:

Python deals with the objects and each object is defined for a type that tells what kind of operations can be performed on it. For ex: 5 can be used as + - \* / as it is a number.

Objects are of two types: scalar (basic like int etc) which cannot be subdivided. And non-scalar like lists which can be sub-divided. Type() to find the type of object. We can convert any datatypes: float (3) = 3.0; int(3.9) = 3;

Print(3+2) : prints the value into console.

Operators: % (reminder) \*\* (power) and assignment binds value to a variable (=)

Never use floats in loop assertions as that leads to infinite loop as 3.0 never gets into system it goes to 3.000000 or more as per no. of bits

Lecture2:

Concatinaion of strings: “hi” + ” ” + “abhilash” = hi Abhilash

Multiplication: (hi)\*3 = hi hi hi

Print(“hi”,”Abhilash”) = hi Abhilash (space is automatically generated for comma)

Print(“hi”+”abhilash”) = hiabhilash (concatenate if you don’t want space)

Input(“type anything”) = waits user to give a value and press enter. It automatically takes the input as string always and cast to get into int. ex: int(input(“type number”))

If <condition>:

Statement1

Statement2

Else/Elif <condition>:

statements

the intrication of 4 spaces makes a code block to be in one set

while <condition>: // use while when you don’t know the no of times to execute a loop

statement

statement // out of loop statements

**ctrl+c stop from execution of infinite loop**

for n in range (5) // range implies getting from 0 to 4 (stop number-1) numbers and range is only for integers

range(start,stop,step)

range(start,stop) //default step1

range (stop) //start=0, step=1

Any for loop can be written with while but while cannot always be replaced with for.

For loop can only be used when we know actual number of times we need the loop to continue

We need not increment in the for loop and is automatic but for while loop one has to include the step in its body

Lectue3:

Strings: len(string\_name) tells the length of string

If s = “abc” len(s) = 3

Indexing: positioning the characters in the string. In above, s[0] =a s[2] = c ; s[-1] =c // negative implies from back starting at first. If input number in index Is beyond it gives an error.

To slice the strings into substrings: s= “abcdefgh” s[1:4:1] = bcd (other prop same as range) if no end is given then it takes length of string (s[ : : ] = string itself) s[ : : -1] = reverse the string;

Strings are immutable // cannot be modified

S = “hello” if s[0] = “y” is an error; but can replace the whole string

**Alogorithms:**

**Guess and check**

Ex: cube root of 9:

M=9

For guess in range(abs(m)+1)

If guess\*\*3 >= m

Break;

If guess\*\*3 != m

Print(“it is not perfect cube”)

Else

If(m<0)

Guess = -guess

Print(“cube root of”,m,”is”,guess)

**Approximates:**

Here we take a guess and input a increment value like 0.1 and keep checking that

(Guess\*\*3 – m) <= epsilon where epsilon is my tolerance error that can be

And finally print the number.

Note: but there can be an issue where the increment. Ex: if m=100 I may get 99.8 with one cube and error 0.2 which is beyond tolerance and other is 100.6 for the next increment cube with error 0.6 which is also beyond tolerance and later with the increments that goes on increasing error thereby going to an infinite loop.

Hence also constrain the loop condition with (Guess\*\*3 – m) <= epsilon and guess\*\*3 <m

**Bisection Search:**

M = 56 //number to find cube root

Lower =0;

Higher =m

Epsilon =0.1

Guess = (higher+lower)/2

while guess\*\*3 -m <= epsilon

If guess\*\*3 >m

Higher =guess

If guess\*\*3 <m

Lower= guess

Guess = (higher+ lower)/2.0

Print (“cube root is ”, guess)

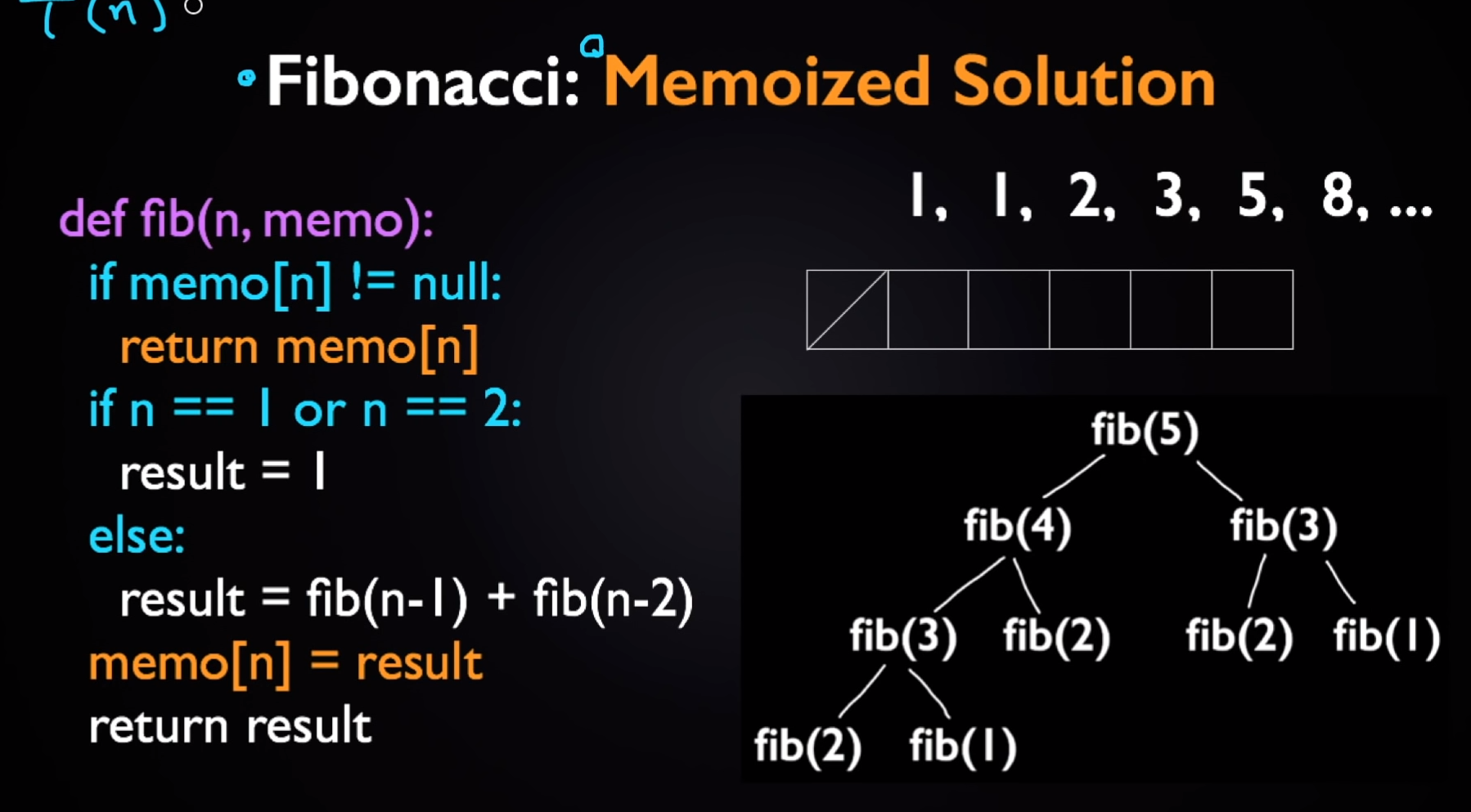
This bisection holds good for m>1 because for ex: m=0.5 cube root is 0.7 which fails for bisection

To find the number of guesses as for first guess m/2 gets into the space to find the solution. For the second guess m/4 is left.

m/(2^n) gives the solution as 1 says that n = log(m,2)

dynamic programming:

storing values intermittently to make use later to reduce computations. Ex:



Or we can also use bottom up or recursion here.

Bottom up is put array[0] =1 array [1]=1 and array [i] = array[i-1] + array[i-2] //i>2 till the recursion value needed as i.

Lecture 4:

Abstraction: relates to black box utility. Something like a projector which we donot know how the internals work but know how to use it. Can be achieved through docstrings or fn specifications. It is used where we need not write code/ hide code/ hide tedious lines of code

Decomposition: ability to divide the whole work into chunks and able to produce same output as whole. It is achieved through modules which can be reusable.

Def name function (i): // def tells that it is fn definition in python. I is called formal parameters as it does not carry any value unless it is called with some input

Statements

Return …… (expression)

Name function (3) // fn call here 3 is actual parameter which holds value.

For docstring “’ write lines in these triple quotes and will be visible to user who uses it “’

If no return is explicitly given then python returns none by default. Where none is a special type of data type NONE type which means absence of value.

In global perspective initially the compiler does not care about the code inside the def but just notices that there is something called a fn\_name that is defined.

We can also make functions as an arguments in functions. // proves everything in python is an object

Ex: def fun\_a():

Return1

Def fun\_c(z):

Return z()

Fun\_c(fun\_a) // goes to c then to a return 1 from a to c then returns 1 from c to call.

Three situations of scope:

* 1. When you declare same variable in def of function and in main both are different
  2. When you donot declare the variable in def its ok in python that it gets temporarily out of scope into main and takes up value from main.
  3. When you try to increment a value without declaring it in def like in case 2 it may bring up the value but doesn’t allow you to reinitialise the value. // unbound local error

Generally on doesn’t use global variable as that makes the program messy

Lecture 5:

Tuples and lists are compound data types as they make use of more than one data type sequence

Tuples: an ordered sequence of elements can be of different data types

Tu = () // tuple syntax

Tu = (2 , ‘h’, 4.0) // these are immutable like strings

Tu[1:2] // gives value ‘h’, // this comma represents a tuple object

(1,2,3,4) + (5,6) = (1,2,3,4,5,6) // len( tu ) = 6

We can return more than one value with tuples like return tu(2,3)

To make use of the index element : (tu[0]**,**) is the syntax where 0 can be replaced with number

Ex: 0+ (tu[0],)

We can have tuple of tuples

Lists: li = [1,2,3] // these are mutable

Operations on lists: li.append(element) // appends element to the end of list li

. operation is like performing a function on an object of specific category. Here we apply fn append to the mutable object type list.

L = l1 + l2 // adds two lists (concatenate) into l

Or use l.extend([0,6]) this extend method appends this list

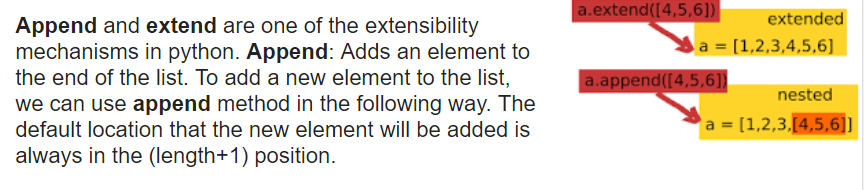
To remove elements :

l.remove (2) // removes if any of the element in list has 2 ( if there are multiple of value 2 it removes only one first)

l.pop() removes the last element of the list

del(l[1]) removes the element of index 1

// after removing the indexes of elements does not gets changed as they do not adjust themselves. And as all these are fn they return none. To make use of iteration on list manipulations iterate over indexes of clone on original list to manipulate.



Lists and strings:

S =”Hello hai”

List(s) implies [‘h’,’e’,’l’,’l’,’o’,’ ’,’h’,’a’,’i’]

s.split() implies [“hello”,”hai”] // if no parameter is given it splits by spaces

l = [‘h’,’e’,’l’,’l’,’o’]

‘\_’.join(l) implies “h\_e\_l\_l\_o” // use ‘’ instead of ‘\_’ to get hello

l.sort() sorts the list . if we use sorted(l) it does not mutate l so use l2 = sorted(l). l.reverse() to reverse list. <https://docs.python.org/3/tutorial/datastructures.html> for more

Aliasing and cloning: if we alias a list then both of them point to the same memory for example

a = [1,2,3]

b =a

b.append(4)

print (a) // we get [1,2,3,4] as both points to same memory

so we need to clone if we have to get a copy by b =a[:] // where first is 0 behind : and right part is length (a).

list name. index (element) to get index of element in list

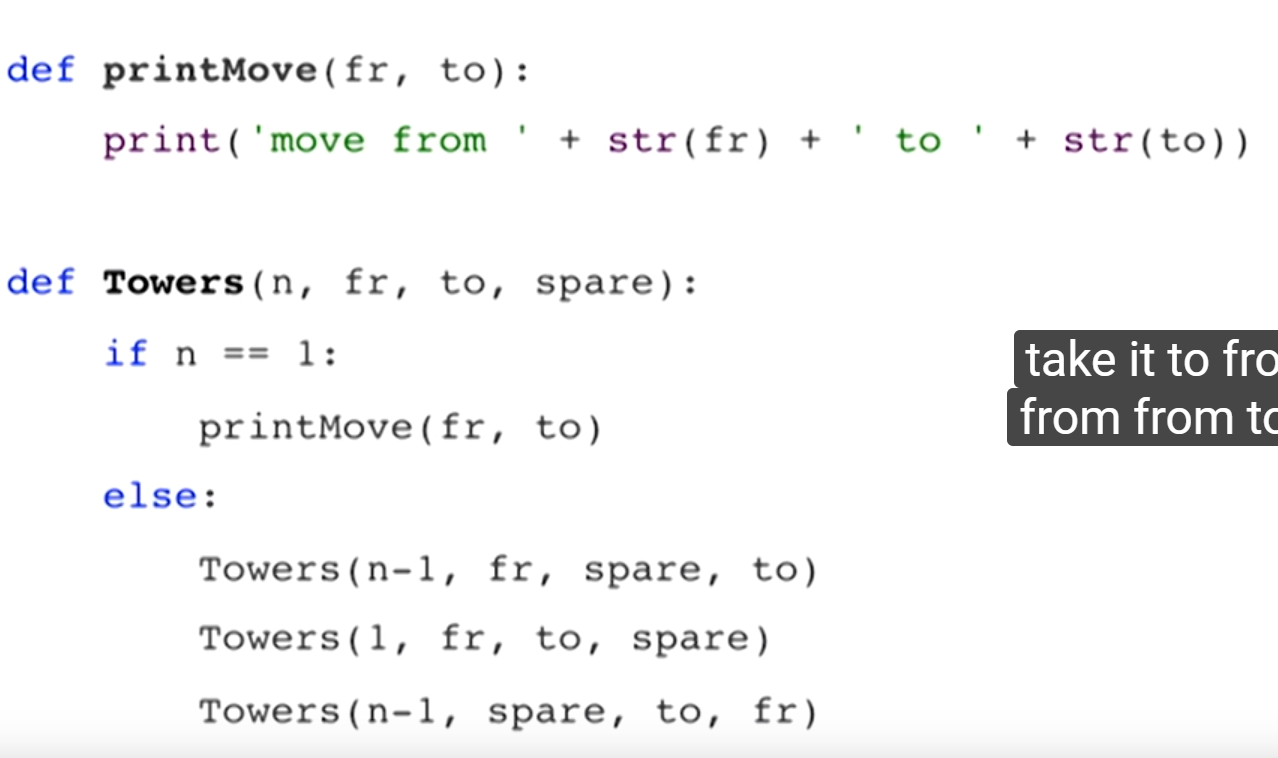
Lecture: 06

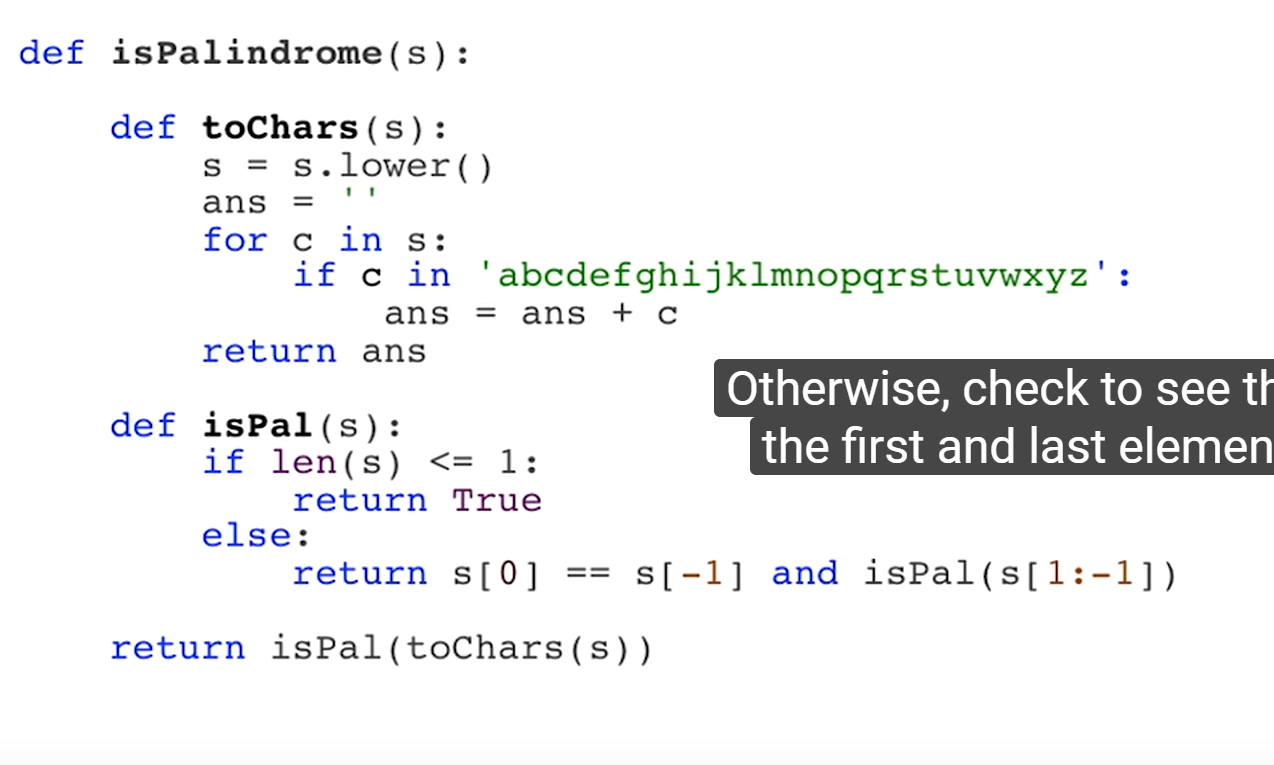
Recursion: iteratively calling same function again till a base condition is satisfied.

It helps in reducing a complex problem into a basic and iteratively computing for a larger solution. While if we use while loop it does not reduce the complexity of problem

Multiple scopes are created with natural flow and each time it creates local scope for computation

Mathematically the recursion Is an alter form of mathematical induction where when the statement gets a true value for basic inputs ( like gives correct value for n=0 , n=1 means it is true for k if we prove it is true for arbitrary k+1) here in recursion we make sure that it meets the basic condition and repeatedly adding up things together to make solution for the complex problems.





We can have two base conditions as of fibanocci series or two recursives and a base condition as in tower problem (set 1)

Dictionaries : if no dictionaries, if we need a register of name and grade of a class maintain names in a list and marks in same index of another list. To retrieve info get the index in first from the name and check marks in second.

List : index and value

Dict : key and value

My\_dict = {}

My\_dict = {‘Ana’: ‘50’ , ‘’:’’}

My\_dict[‘Ana’] : syntax to get value of key

To add: my\_dict[‘i’] =’10’

‘john’ in my\_dict gives false as it is not in dictionary implies testing of dictionary

Del(my\_dict[‘Ana’]) deletes the dictionary element

My\_dict . keys() gives all keys or My\_dict. Values() for values

Keys need to be unique and immutable and do not store the elements in sequence. They store randomly

Dictionaries used for memorization in recursion like in factorization to remember factorials in recursions and use the value if it is already there in dictionary rather than recurring again to find base condition. Ex: calculation fact(2) several times in recursion of finding fact (5) : in fact(4) fact(3) and fact(2)

Values are mutable in dictionaries

From ps2: # reading files and writing

Fo = open(“file.txt”,”r”) // fo is object and opening in read mode

Fo.close()

Fo.write(“type statement and /n to write into file”)

Fo.read(10) // 10bytes to read. Each char 1 byte

Fo.tell() to tell the position of reading // if it gives 11 it means the reading is at 11thchar

Fo.seek(offset,from) The *offset* argument indicates the number of bytes to be moved. The *from* argument specifies the reference position from where the bytes are to be moved.

Import os

os. Rename(“name1”,”name2”)

os.remove(“name2”) // del file

Lecture: 07:

It is good approach to make your program modular. Make as many functions as you could as that helps in debugging in later stages

Tests can be of unit testing: where we give inputs for outputs and check the functions accordingly

Regression testing: Testing the basic things and boundaries again and again making sure that core functionality is correct

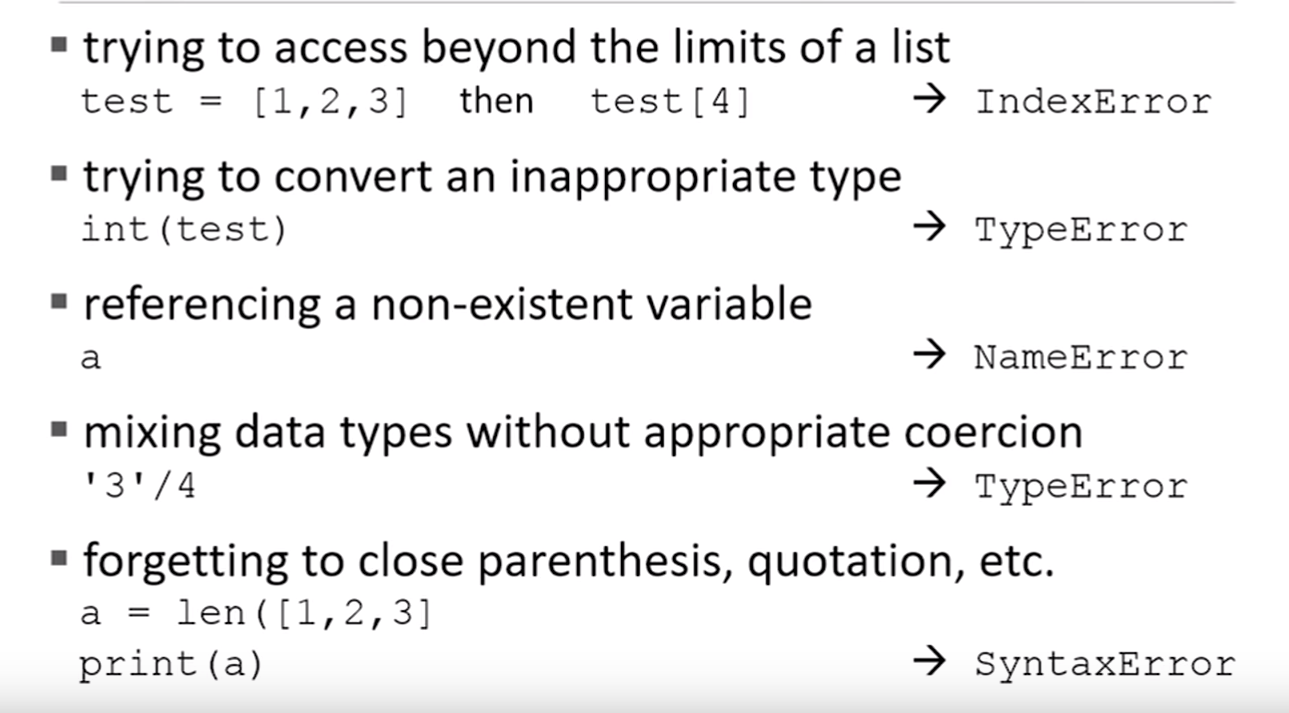
Integration testing: testing the whole program as one

Black box testing: know doc string and come with test cases

Glass box testing: come with test cases based on knowing the code

In glass box testing we call it path complete when it tests each path. For ex: in testing loop go for test case that doesn’t enter loop, enter once, enter more than once. In Branch testing get all part of conditional statements

Print statements are good helpers in debugging the code



“Rubber ducky” : explain to one who don’t understand anything to find logical error:

Try:

(If found problematic code)

Except value error:

Except div by zero : // exception for div by zero

Except: // exception of everything

Else: // executed when no exception

Finally: // executed completely

Assert condition, string // condition is to be checked as assertion string is the one that is printed if assertion goes wrong

When assertion goes wrong the function terminate and stops propagating wrong value

Raise exception(“it encountered an exception!”) // raising an exception that breaks or halt program

Lecture : 08:

Objects represented by the type and how we can interact with the object

Lists are represented as | l[0] add(l[1]) | , | l[1] add([2]) | …. Internally but we need not know it to use the object type list

So these objects are used for data abstractions and generation of new data types

Class class\_name (object):

Attributes

//attributes has functions to tell us how to interact (procedures) and some variables (data representations)

Class class\_name (object): //self takes the instance of class object

Def \_init\_(self,x,y): // special type saying to go here first x and y are defined with self as they are parameters others you can define variables as you give in a function

Self.x =x

Self.y =y

P = class\_name(5,2) // self =p (automatic assignment by python), x=5 , y=2

Print(p.x)

Def fn\_name (self, other) // here we are using two objects of class class\_name

Return Self.x – other.x

O= class\_name(0,0)

P= class\_name(5,2)

We call call it as p.fn\_name(o) or class\_name.fn\_name(p,o)

Print(o) // gives the address and type of object

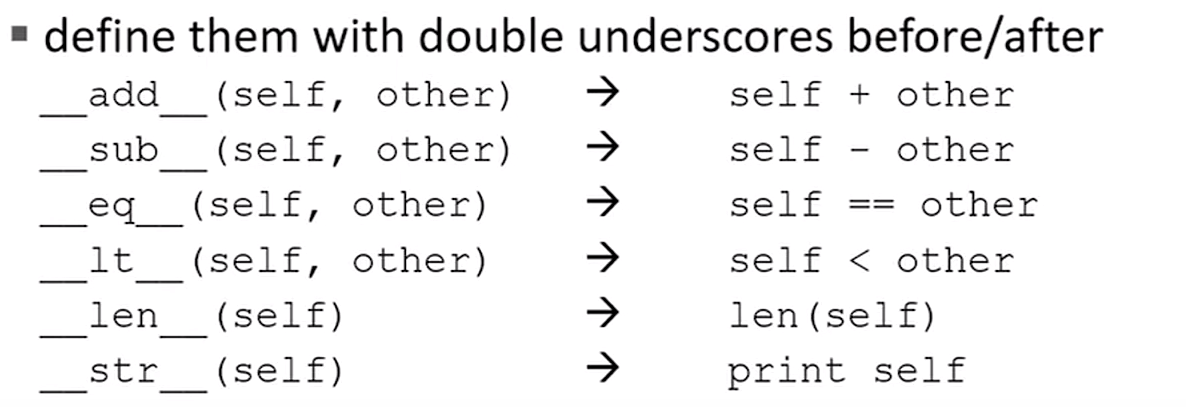
To get values of object or some details when we say print use special function

Def \_str\_ (self):

Return (str(self.x) +”and” + str(self.y))

Type(o) // \_main\_.class\_name

Self and other being two objects use the below special functions to make python understand what to do when operators are used between object types



Lecture:09

Getters and setters are frequently used in a class

Getters: returns the value from the classes which has data being accessed

Setters: sets values to the data attributes () = ()

You can also assign values to attributes or create attributes of class outside the class – but do not do that

Default arguments: fn\_name(self, name= “”)

So if I invoke function as object = fn\_name() // name = “”

Object = gn\_name(“Kate”) // name = “Kate”

We can have a parent class and a child class.

The child class has all the procedures and attributes of parent class and also can override the procedures and add extra too

Class parent\_name (object):

//defines parent class

Class child\_name (parent\_name):

// defines child class

Class variables are variable created outside of \_init\_ to get it accessed to all instances of class with modified numbers as required

Ex:

Class name(object)

tag =1

Def \_init\_ (self):

Self.x = name.tag

Name.Tag+=1

Here for the second instance tag value is 2 and is assigned

Lecture: 10

The efficiency of an algorithm depends on space (memory) and time

Eff = f(size of input, time) // helps in building algorithms and choice for algorithms

Ex: google search around 30 trillion search pages for a word, so it needs to be fast

This can be approached by following methods:

1. Import time and

Start clock() //t0 = time.clock()

Function­\_call()

Stop clock() //t1 = time.clock()

Time = t1-t0

But this is dependant on implementation and clock speed of system so it is not much good one to use

1. To count the number of operations that the function uses and number of digits in the input

But the problem here is sometimes retrieving data takes much more time than performing the operations so which is not counted here and not a clear definition of hat operations yo count

1. Order of growth: this method has input as length of input to calculate the efficiency

For ex: if I take a number, the length of it is its characteristic measure

For a list it is its length of list

For a function with multiple arguments take the one that has impact like if you have a list and a variable take the list

Likewise in the operations count get the expression and the decide the order

Like if we get the expression as 3n +2 it is of order 1. O(n) = 1

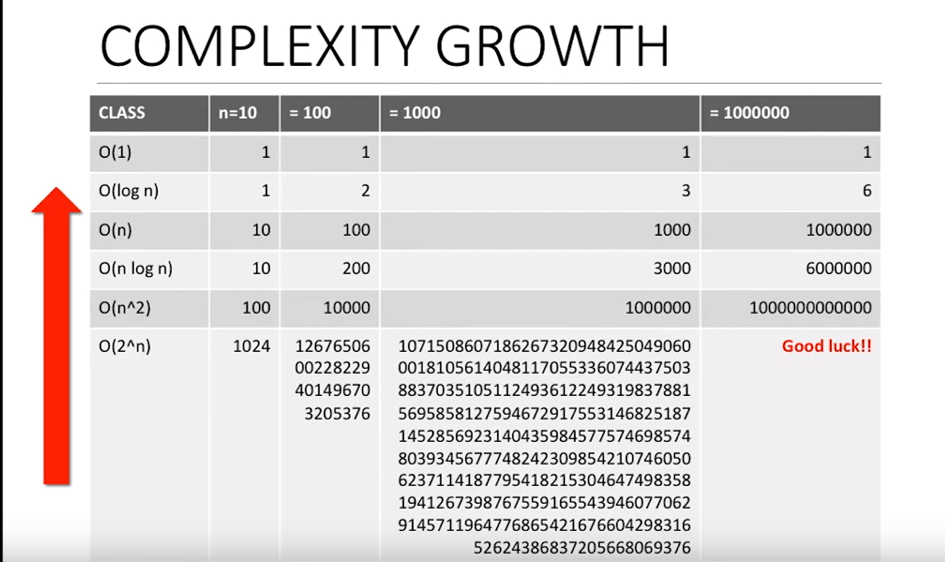
3N^2 +10000n+3^10000 here it is of 0(n) = 2

Always in case of power and exponentials we get exponential like n^1000 + 3^n it is 3^n as order

Rules of order of growth:

1. Addition: for complex algorithms we split up the cod into multiple chunks and add up to get the dominant order. For ex: O (n^2) + O(n) = O (n^2 + n) = O(n^2) // for j in (n\*n): print ‘b’ = O(n^2)
2. Multiplication: when we have nested loops, we multiply the orders of each loop to above loop like for i in n: for j in n: print ‘k’ = O(n)\*O(n) == O(n^2)

Order of growth always catches the worst case behaviour it may end up early if you are lucky!! :D



Notes on pointers: for ex when we want to go from one place to other in the memory we can go it by a for loop and check if there is element as such and notice the element memory

This takes O(n) = 1

This can be made constant by taking the base address and incrementing its value to a constant like if we need 5 th element of integer go for base address+ 4(size of int)

When case we do not know the data type go for the pointers case to make it a constant order

Lecture:11

Get into binary search: each step we cut the size of list by ½

1st we get into n/2 elements

2nd we get into n/4 elements … until we reach 1 element

If the number is not In the list we stop

Last case: 1 = n/ (2^i)

i = log(n) so O(n) = log(n)

# len[list]//2 (splits list to half) see list[half: ] or list [: half] to see half part left or right

Use the above sentence we make a copy of half of list and returning it. Here we do two things we make the selection to half and copy things so getting into order:

O(log n) + O(n) //copy takes linear which makes order n

So instead of this copy take two variables low and high and manipulate the values independent of n like low + high /2 etc which makes O(logn) since that variation is of constant order

So logarithmic comes into play when we cut down the problem in the further steps

Like when we put up a number into a string: we cut it by 10 times get the reminder and append it to a string. When we cut it by 10 times it means getting it into log form

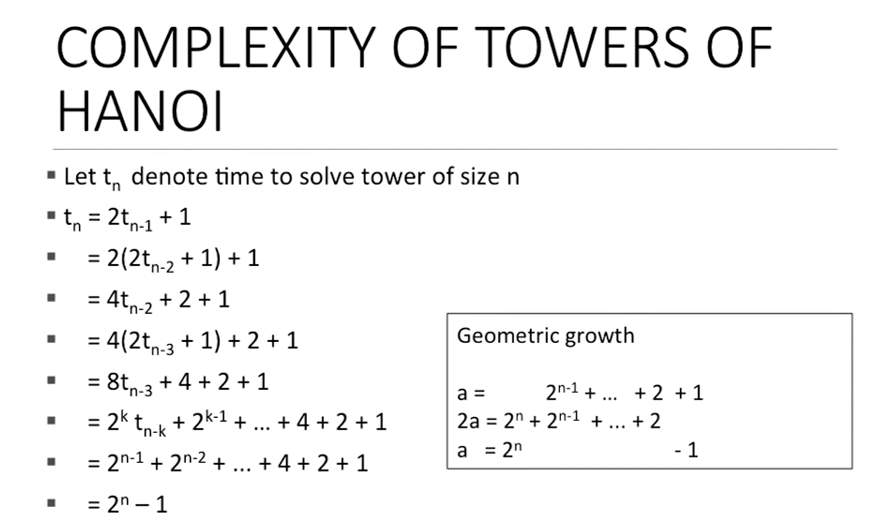
Recursive is not same as nested loops it doesn’t take the order in multiplicative way it goes to addition law

Log linear type: Ex: Merge sort // later class

Polynomial type: Nested loops.

Exponential types:

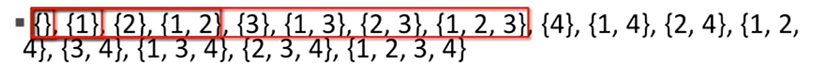
// 2 tn-1 + 1 == 2 is tn-1 put to 3rd 1 to 1st and n-1 from 3rd to 2nd



Here we run two recursive loops in one so it causes much

In actual Hanoi n= 64 to reach nirvana :D

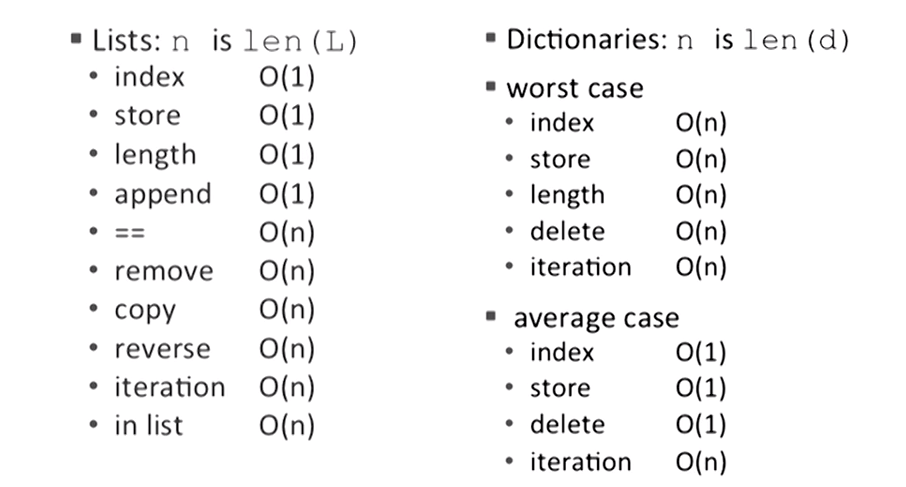
To generate power sets:



Get an empty set

Add element to all present till then and continue

1. 0+1 0+2 1+2 0+3 1+3 2+3 1+2+3 …..



Lecture:12

Search is of two types implicit and explicit search.

Explicit means searching with the help of keywords

Implicit is based on the data provided to it

Ex: Google uses both in Search Engines like explicit for the keywords you type and implicit with the location, search history, time and other stuff

Explicit search Algorithms:

Linear

Binary search: sort and check for mid half

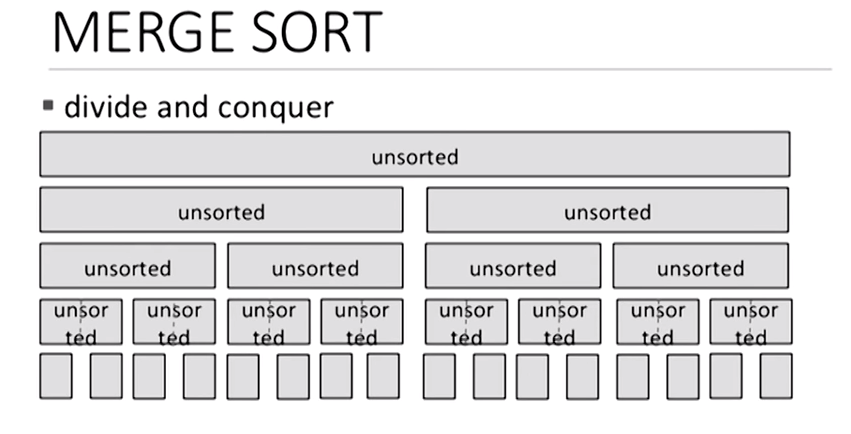
Here in binary for sorting is of O(n) and search is of O(log n) so overall algorithm again goes to O(n)

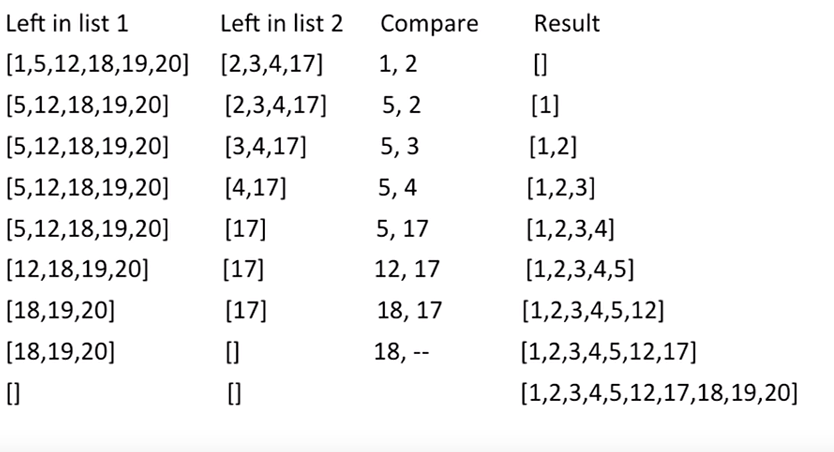
Monkey sort / bogo sort/ permutation sort: pick random and arrange them based on previous list. Worst case there is no order as such for efficiency calculation.

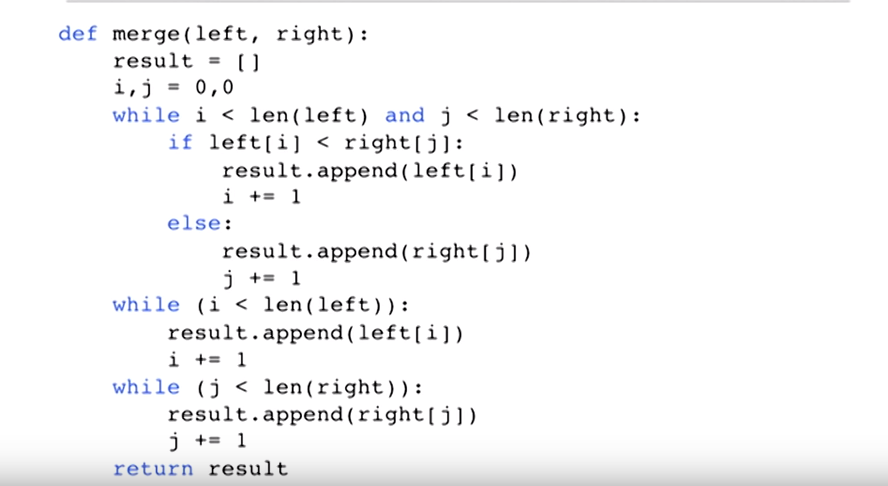
Bubble sort: check adjacent and swap based on criteria: O(n^2) // one n is to get it done to all elements and other n is for base condition to satisfy for halting sort

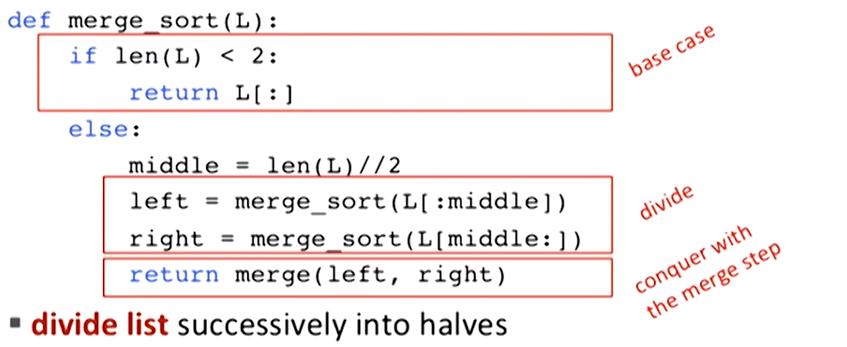
Selective Sort : select the first element and compare with each element of list. Get the smallest of all and put it up in the first position of list2. Do it again to find the second smallest and put it up in the second element of list 2 …. It is O(n^2) like bubble

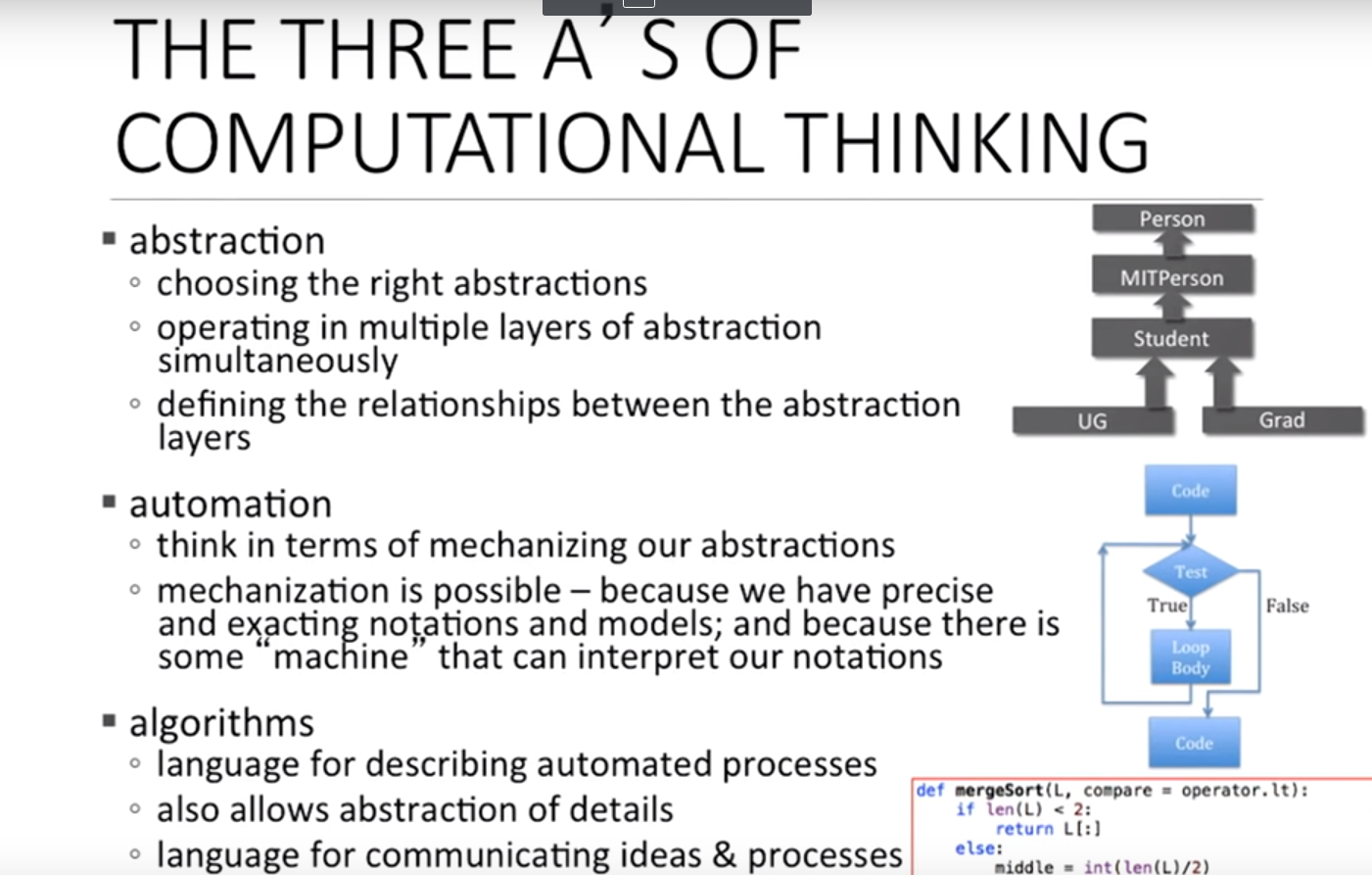
**Merge sort:**  break down to half till you reach either 0 or 1 element in it as it is fundamentally sorted. Now merge two blocks from the down adjacent to each other and make a sorted list on top and continue process while you merge we take first element of list 1 and compare with first element of list 2 and take the winner and go for next element in same way. **Order is O(n log n)**

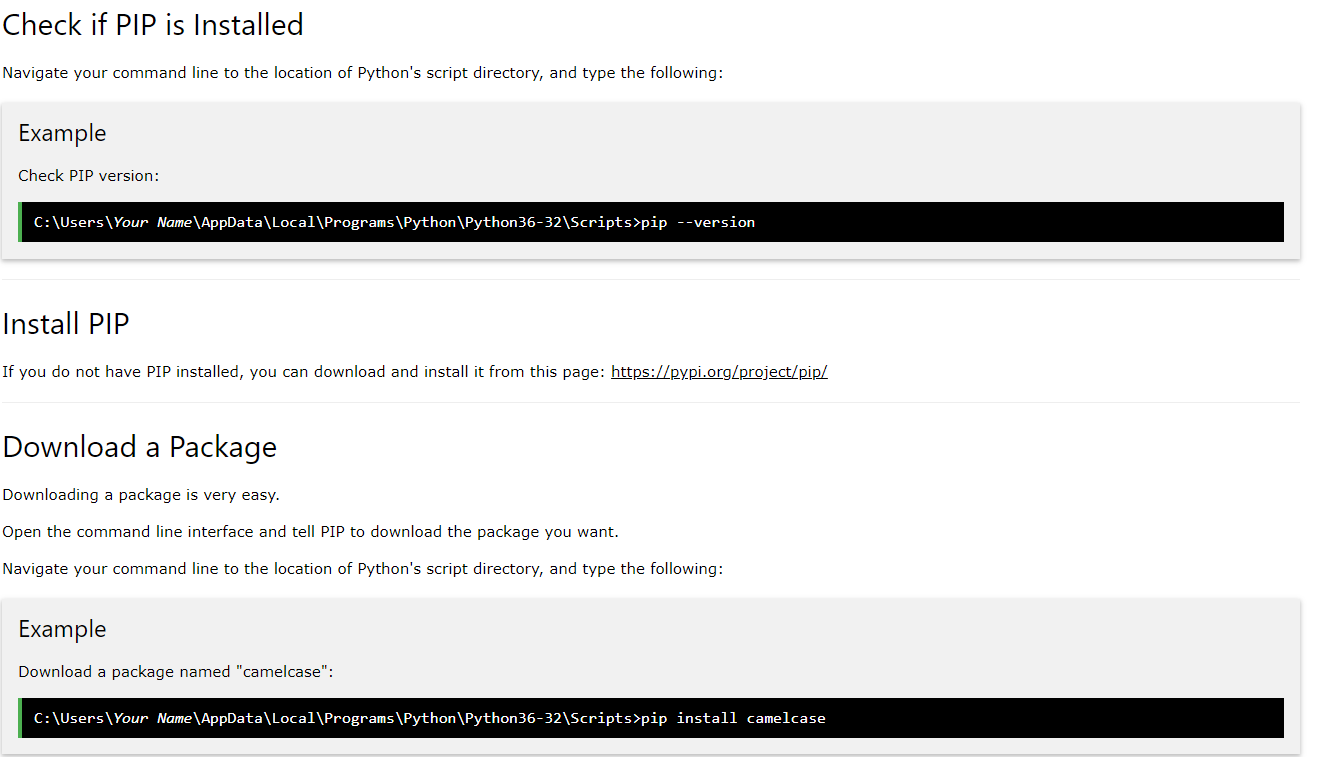












Go to above path in cmd and type pip

It gets executed and we get list of commands

To install any package type pip install package\_name

Pip list //gets list of installed packages

C:\Users\vakondur\AppData\Local\Programs\Python\Python37-32\Lib\site-packages to get the packages scripts

For getting access in spyder go to environment in anaconda navigator and install package and then launch IDE

Chrome driver is a tool that bridges the chrome browser and selenium package

from selenium import webdriver

driver = webdriver.Chrome("D:/Programming/HF\_Automation/Accessing\_Browser/chromedriver.exe")//give .exe compulsory other wise it throws an error on path and give path with / and not \

driver.get('https://towardsdatascience.com/controlling-the-web-with-python-6fceb22c5f08') //type URL

DateTime:

Import datetime

D = datetime.date(2019,1,15) // donot put 01. Output of d is 2019-01-15

Tday = datetime.date.today()

Tday.year like that will give particular year or month or weekday as asked

We can use time delta to use the difference of time. We can also use operators on time

