Day 4 of 5°

Magic method in Python: Review

Comparative operators

Operator	Description
>=	
<=	
!=	
==	

Boolean operators (True vs False)

AND (&)

1 True	0 False	output Operation
1	1	1
1	0	0
0	1	0
0	0	0

OR (+)

0	1	1
1	0	1
1	1	1
0	0	0

NOT

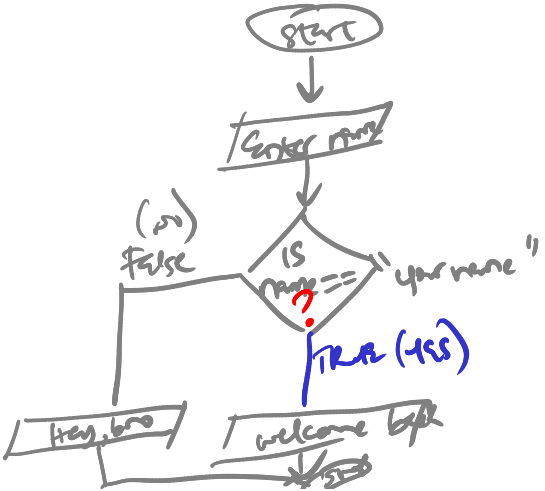
T != 3 ⇒ NOT TRUE ⇒ False

Is `Victor == victor` ⇒ false

len(victor) = 6

len(Victor) = 6

wap ask user to enter a name & check if the name = "yourname"



Looping / iteration

Assign...no. Wap that checks username & password, if it matched grant access else deny access

Prefix: username & password

username == "class1"

password == "1234" → int

Enter your name: → (input())

" " password:

If username == "class1"

If password == (1234)

print(access grant)

else print (access denied)

pass = int(input("enter your password"));

If username == "class1" and password == 1234

If=)

Comparing two nos & print the largest:

num1 = 10

num2 = 20

20

20

largest = num2

max(3,2)

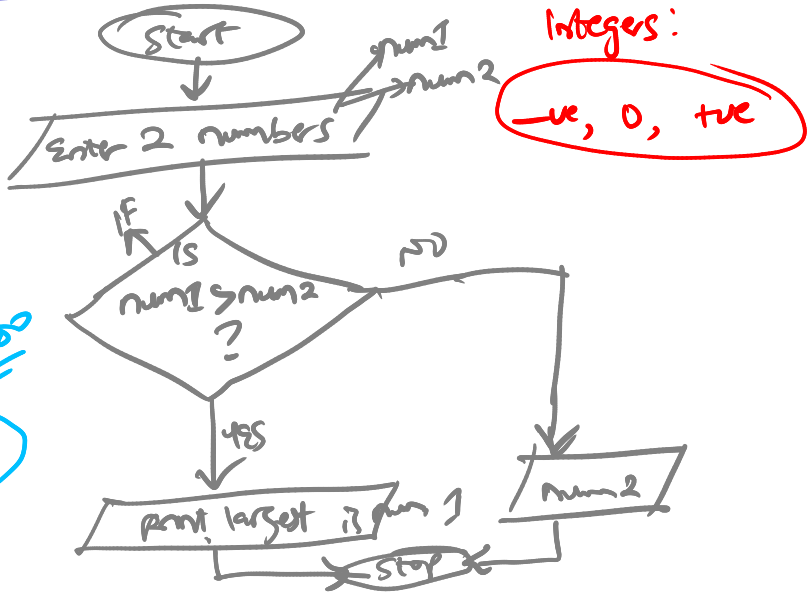
both are equal

arbitrary: -300, 200

-0.013e55

Integers:

-ve, 0, +ve



num = 20
num = 20

else if:

```
If (num1 > num2):
    print ("num1 is larger")
        num1, "is larger
        ↓
elif (num1 == num2)
    print ("Both are equal")
else
```

Days of 50 Days to New Challenge ⇒ 31st Dec ← Medical / DS / ML] 40 days

x-rays, CT scans, MRI's

CONTROL STMTS in PYTHON

1) If Stmt
2) If-else
3) Nested-If
4) If-elif ladder
5) Short Hand if Stmt

Shorthand operator

i = i + 1

i += 1

6) Short Hand if-else

When do we use Nesting?

```
If Condition:
    <print stmt>
        If Condition
            <print stmt>

else
```

Recall: tax = 0.18, qty = 5

price = 30

amount = price * qty + tax

If amount > 150:

    If amount > 500:

        print ("the amount is greater than 500")

    else:

        If

Short hand:

List Comprehension → lambda operation

num = 12

If num > 12 : print (num, "is greater than 12");

else.

Python Support ternary Operators → If < > - else <

→ Not Java, C++

num1 = 3

num2 = 6

If (num1 > num2):

    print (num1 is greater")

equality.

else:

    print ("num2 is greater")

print ("num1 is greater" If num1 > num2 else print ("num 2 is greater")

Q. Can we have multiple else stmts on same line?

Ans: Yes:

print("num1 is >") If num1 > num2 else print("Equal") If num1 == num2 else print("

a = 3

b = 2

c = 5

AND OR NOT

If a > b and a > c

True and False = False

print ...

If a > b or a > c

True or False

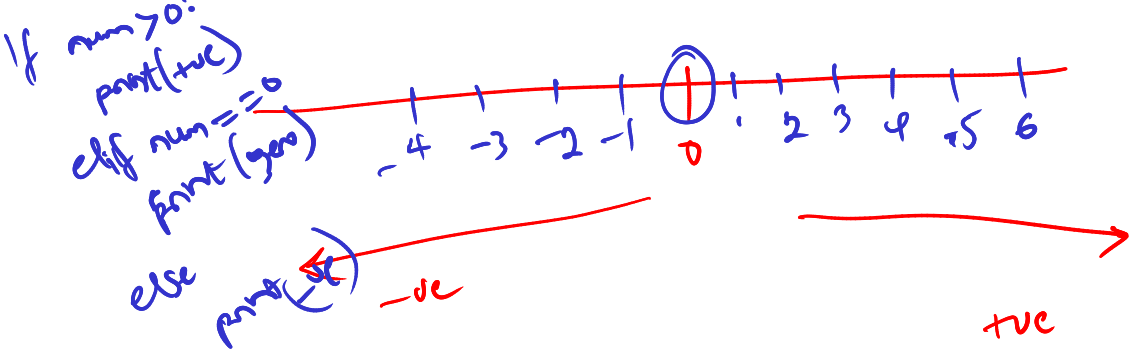
True

Why pass? to avoid getting error message

```
def sayHello
#prn
#
pass
```

Examples:

```
num1 = 27
num2 = 50
if a > b:
    pass
```



Structure of Switch case:

```
If ch == 'a' or ch == 'A'   print ("Vowel")
    elif ch == 'e' or ch == 'E'
    elif ch == 'i' or ch == 'I'
    elif ch == 'o' or ch == 'O'
    elif ch == 'u' or ch == 'U'
    else:
        print ("Consonant")
```

Starts	values
range(10)	0,1,2,3,4,5,6,7,8,9
range(1,7)	1,2,3,4,5,6
range(3,7)	3,4,5,6
range(3,7,1)	3,4,5,6
range(3,9,2)	3,5,7
range(2,15,3)	2,5,8,11,14
range(7,2,-1)	7,6,5,4,3

Concatenation: + repetition *

hello + 5

"hello" + "5" ⇒ hello5

"hi" + 4 + "you" ✓

In Operator:

for ch in name

If "a" in name ⇒ Search Algorithm

Indexing

len(5) = 5

s = input("enter the name") len(s) = 7

s = Amith

s[0] = A ✓

s[4] = h

s[-1] = h vs h vs i

Q what if the first
② " " " last
③ " " " middle
④ " " " second to the last char
⑤ print his output: it

s[-2]

s[:] ✓ returns the entire string

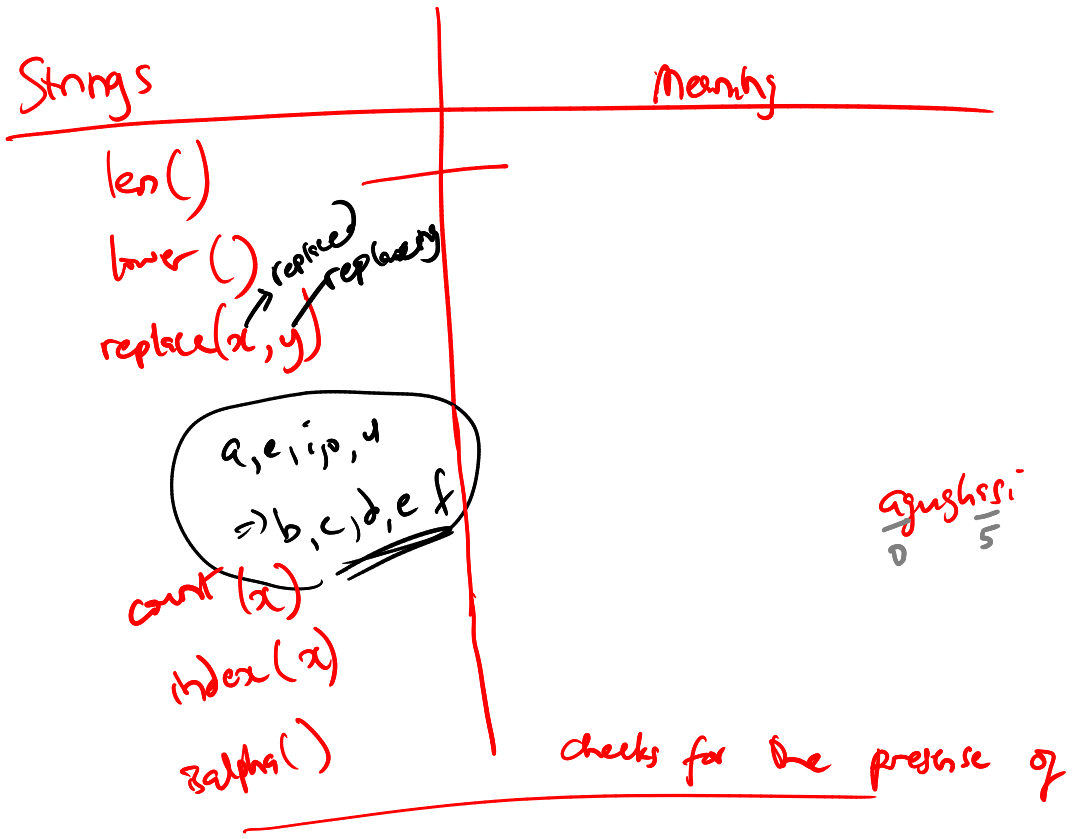
s[2:4] -1 2-3

s = technology 0 1 2 3 4 5 6 7 8 9

-10 -9 -8 -7 -6 -5 -4 -3 -2 -1

index = singular
indices = plural

Code Snippet	Yields	Description
s[2:5:1]	chn	characters at indices
s[:5]	Techn ✓	
s[:]	Technology	
s[1:8:2] (step)	TeNLG ✓	
s[::-1]	ygol... t	reverse the string.

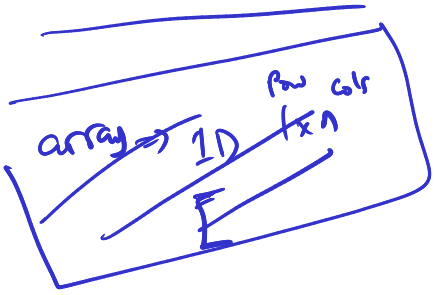


Day 7 of 50 : List

variable values

L = [1, 2, 3]

What is a List? used to store multiple items in a single variable.



⇒ It's among the datatypes in python

List, Tuples, Sets and Dictionary (ies)

S = 'Hello' vs "Hello" ⇒ len(S) = 5

L = ["Hello" 'H' 'E', L

~~[H,E] + [L,L,O] = HELLO~~

≡ 'HE' + 'LLO' (+3)

where L = List

L = [ ] ⇒ empty list

1D lists

rows col

Long list ≠ 3 x 4

LL = [1, 2, 3, 4
5, 6, 7, 8 ✓
9, 10, 11, 12]

L[0] ⇒

Similarities (to) with Strings

① len()

② in-operator → L = [1, 2, 3]

if 2 in L
    print ("the number is present")

③ not

If 7 not in L
    print ("the number is not found")

④ Indexing vs Slicing
L[0:3] L[:3] L[0:4]

⑤ Index vs Count

Some Egs

Ex	Value/output
[1,2,3] + [4,5]	[1,2,3,4,5]
[1,2,3] * 2	[1,1, 2,2,3,3] vs [1,2,3,1,2,3]
[#&%] * 10	#&%#&%... ... #&%

for item in range(len(L))
    1

Adam

Sep = " "

A D A M

D A M

Consider test scores of Students:

Queue & Stack

LIFO
FIFO

L = [9, 15, 12, 3, 14]

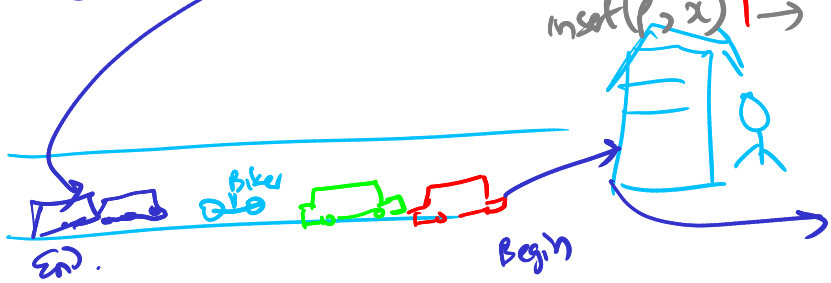
max(L) ⇒ 15

min(L) ⇒ 3

methods	meaning
sort()	sorts in ascending
append(a)	adds "a" at the end of the list
Pop(a)	
insert(p, x)	

Queue →

[9, 9, 4, 9]
↑



test scores

Ls = [30, 99, 98, 76, 56, 10, 97, 38]

Worst Scores: 10 and 30 ⇒ 30, 10 vs 10, 30

Best Scores: 99 and 98 99, 98 98, 99

Steps: sort() in ascending order

New lst = [10, 30, 38, 56, 76, 97, 98, 99]
indexes: 0, 1, 2, 3, 4, 5, 6, 7
negative indexes: -8, -7, -6, -5, -4, -3, -2, -1

NLS = [ ]

Least / Smallest: NLS[0], [1]

NLS[-1], [-2]

NLS[-2]] flexibility

In python, array doesn't exist ⇒ [accepts only data type]

char a = [ ] ⇒ lst in python is flexible

Python is Dynamically typed

Datatypes in Python

- String ✓
- Numbers → int (1, 2, 3) → float (1.23, 0.45) ✓
- List (1D vs 2D) ✓
- Tuples ×
- Dictionary ×

Functions

What is join() method?

Opposite of split (breaks apart)

∴ Join (brings together)

print(',' .join(Li))

A, M, A, R

For Eg:

Li = ['A', 'M', 'A', 'R']

print(' ' .join(Li))

Output: A M A R

Anagram Generator

Input: Enter any word? "welcome"

output: elocwem ✓
wemolc ✓ Randomly

import random, shuffle

List

Step 1: Accept as it is (any)
Step 2: Store in a List
Step 3: Shuffle the List
finally: Join the shuffled list

List Comprehension

Eg1: Create a list of 10 integer numbers

Li = [0,1,2,3,4,5,6,7,8,9] ✗

Li = [i for i in range(10)] gives [0,1,2,...9]

Li = [0,1,2,3,4,5,6,7,8,9]
0 10 20 30 90

Examples: [i*2 for i in Li] → [0,1,4,9,16 ....

i*x2
[i*10 for i in Li] ⇒ [0,10,20 ... 90]

[i for i in Li if (i<7)] ⇒ [0,1,2,3,4,5,6]

Alp = ['one','two','three','four','five'] ≡ [Alp[0] for i in Alp if len==3]

Q: Return/print those items whose length == 3

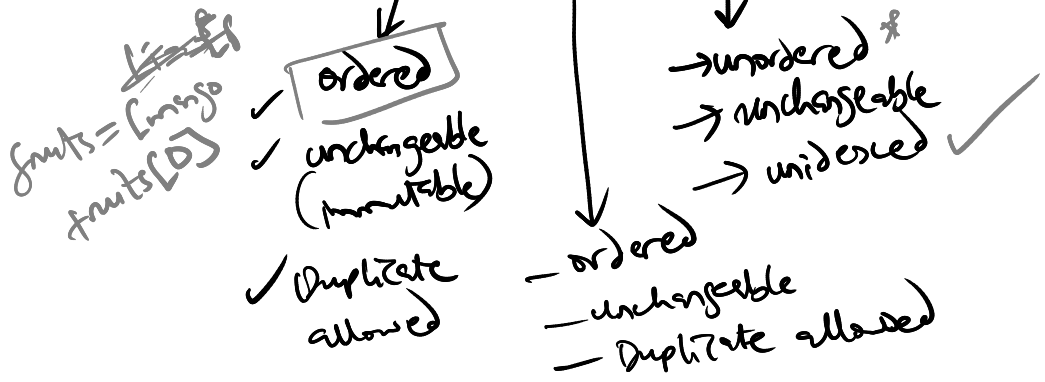
Soln ['0','two']

Len(Alp) = 5

Alp[0] = one
Alp[1] = two

Mon → 22/11/21

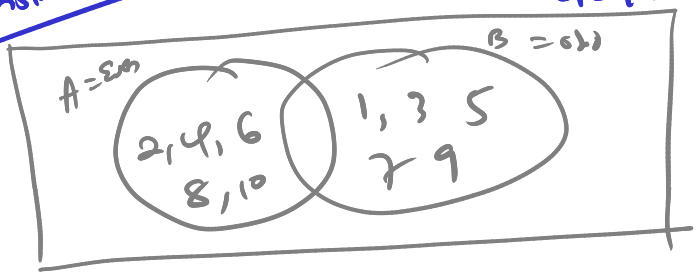
Review: Lists, Tuples, Sets, Dictionary → ordered | indexed – changeable – No duplicate item allowed



Review Set

Set Constructor

Set()



Set operations

A ∪ B ⇒ union = E ⇒ Universal {1,2,3,4,5, ... 10}

A ∩ B ⇒ intersect ⇒ disjoint = Null {/} or { } ✓ Empty

A' or A^c ⇒ Complement ⇒ Set B

B' or B^c ⇒ ⇒ Set A

Using Set Constructor (Set() : use to a create a Set

{"mangoes ... }

fruits = Set(("mangoes", "bananas", "apples"))

print (fruits) = {mangoes ... }

II) Accessing Items in a Set

```
for item in fruits:
    print(item)
```

(III) Adding items in a Set using add() method

```
fruits.add("Blueberry")
```

(IV) Adding Sets in a Set : Using update() method

✓ fruits = {"apples", "bananas", "oranges"}
vegetable = {"tomatoes", "carrots", "cucumber"}

print(fruits.update(vegetable)) ✓

A U B: Union operation on Sets.

(V) Can list(s) be added into a Set? Yes or No: use update()

ingredients = ["sause", "chilli", "some magic"]
fruits.update(ingredients)

(VI) Remove item from a Set:

pop() → removes the last item
remove()
discard() **

fruits.remove("apples")
fruits.remove("apples")

(VII) To empty a Set:.. how

remove()
clear()

(VIII) Using union():

What are Set methods

1) add().
2) clear().
3) cmp

Tuesday → 23/11/21 Day 12 of 50

Working with text files

.doc vs .docx

.txt .csv

Example:

Heya, what's up.\n

Do you know who I am\n

Obviously, you are a student

① Reading a file

hint.txt

Step1: Create the file (don't forget to save in some location)

Step2: lines = [line.strip() for line in open('hint.txt')]

strip: String output without \n

lines = open('hint.txt').read()

② writing to a file(s)

hint.txt → 'w' →

```
line = open('hint.txt', 'w')
print("Heya, This is me", file = f1)
print("I do you know me", file = f1)
f1.close()
```

hint.txt

write

Note: Be careful not to overwrite

Cricket.txt

Day | Club | Score | Club2 | Score Run

23/11/2021, RCB, 167, CSK, 165

167 165

0 1 2 0

-2

scores[2] → 167

Labs

Function:

```
def say_Hello():
    print("Hello world")
```
] function Definition

```
say_Hello()
```
] function call

✓ for reusability (inheritance)

✓ Readability

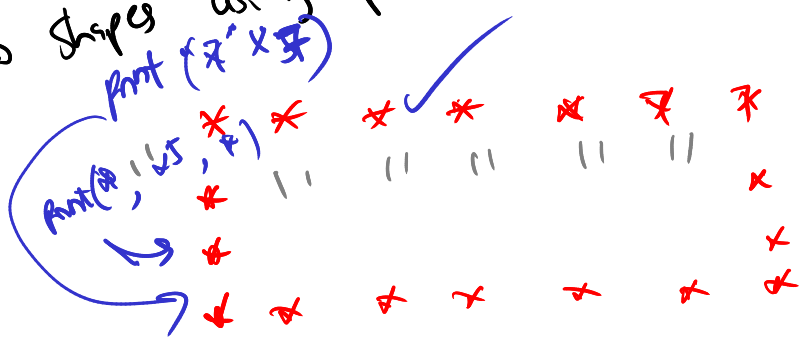
✓ easy to understand & follow

Assignment:

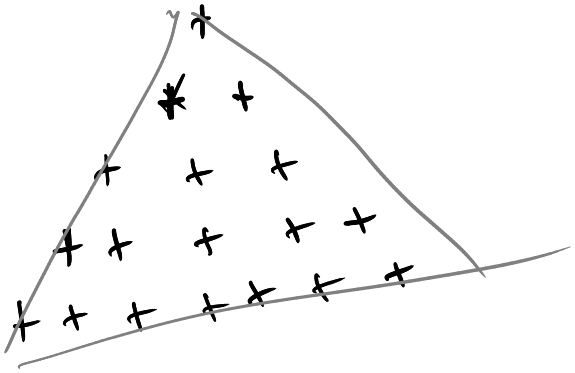
(1) Re-write at least 10 Assignment using functions.

(2) Can you draw shapes using functions?

```
print('*' * 5)
```



(3)



Day 13 of 50'

- ✓ Functions
- ✓ Lambda Expressions

What is a function? A piece of code (block of code) that executes when it's been called.

Two (2) aspects

- ✓ (I) Function definition: use of "def"
- ✓ (II) Function Call : call with function_name

Eg:

```
def sayHello():
    print("Hello Mysuru")
```

Step I

```
sayHello()
```

Step 2

Q2: What are arguments? Whatever we pass into a function (aka argument)

```
def sayHello(fname):
    print("Hello" + fname)
```

with one?

methods vs function()

```
sayHello("Anwar")
sayHello()
```

Q3: Arguments or parameters ≡ What are parameters?

They are same, just that we use both interchangeably.

Eg: Function with two (2) arguments

```
def sayHello(fname, lname):
    print("Hello" + fname + " " + lname)
sayHello("Anwar ","nagi")
```

Arbitrary Arguments, *args

[vs **args]

Q: Why do we need those?

```
public class sayHello {
    public static void main(String[] args){
        System.out.print("Say Hello");
    }
}
```

```
def myfavfood(*food):
    print ("My favourite meal is "+ food[3])
myfavfood("Biryani", "Veg fried Rice", "chicken momo", "Schezwan rice")
```

Keyword Arguments:

≡ key = meaning } Dictionary
✓ = value

why do we need those?

```
def myff(f1, f2, f3, f4):
    print ("my favorite meal is "+ f3)
myff(f1="Biryani", f2="veg fried rice", f3= "chicken momo", f4= "shezwan")
```

myff = {"f1": "Biryani"
"Biryani": f1 ...

Arbitrary Keword Argugments: **Kwargs

For eg:

```
def ff(**food):
    print ("my fav meal is "+ food[f4])
ff(f1= ... , f2 ... , f3 -- f4 ... )
```

Lambda Functions: Are anonymous (nameless) functions.

What are they? why do we need them?

Syntax: lambda argument: expression

(Ternary operator)

For Example: Add 20 to an argument

```
res = lambda a: a+20
print(res(10))
```

→ Result/output: 30

How about multiple arguments:

```
res = lambda a,b: a+b
print(res(10,20))
```

Day 14 of 50: 25/11/2021

Time 7:00pm (IST)

Q1: What is Recursion? ① → A process of defining smth in terms of itself

Eg: You look like your dad

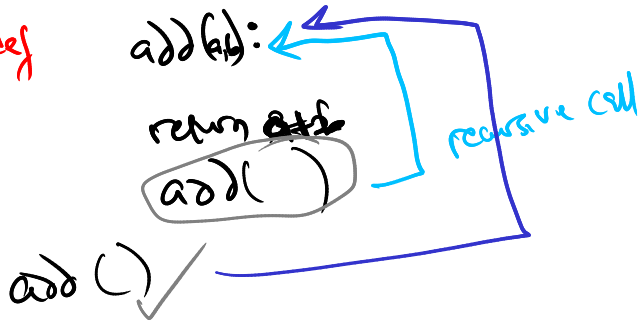
Your ancestor = dad's ancestor

② Recursion in Python: function/code block that repeats itself

Your ancestors = (your parents) + (your parents ancestors)

Eg:

```
def add(a,b):
    return a+b
    add()
add()
```



Advantages of Recursion?

① Easy to read

② less no of code blocks

Demerits/Disadvantages

① Expensive (computationally) as it depends on memory & time

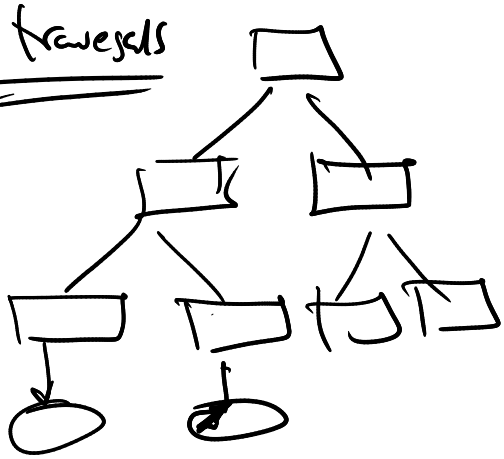
② Very difficult to debug

Why use Recursion?

- Cleaner
- Easy to understand

Use cases:

Tree traversals



What happens when we type.

① def Exp():
a = 10
exp()

First: namespace will be created
Secondly: assignment (value 10 to the variable a)

Factorial $3! = 3 \times 2 \times 1$

Rules.

a $1! = 1$
$0! = 1$

② $3 \times (3-1) \times (3-2) \times (3-3)$
$3 \times 2 \times 1 \times 0 = 6$

$n == 1$

③ Counting down program

```
def countdown(n):
    print(n)
    if n == 1:
        return  #terminate
    else:
        countdown(n-1) #Recursive call
countdown(5):
```

5 4 3 2 1

Exp. - 1

```
while n >= 0
    print(n)
    n = n-1
    Countdown(5)
```

Readable

Confusing

vs

Factorial:

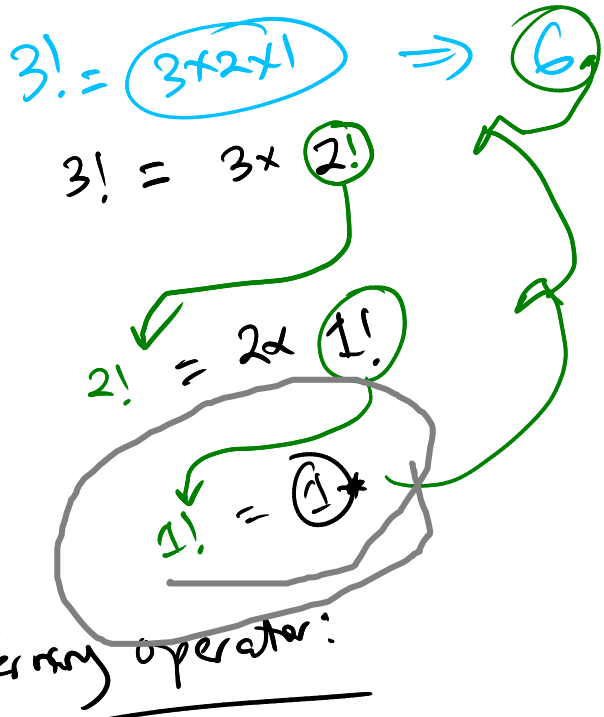
$$n! = n \times (n-1)(n-2) \times \cdots (n-n)$$

$0! = 1$

$1! = 1$

$\forall n = 0 \text{ or } n = 1$

$$\equiv \quad n! = \begin{cases} 1 \\ n(n-1)! \end{cases} \quad \forall n \geq 2$$



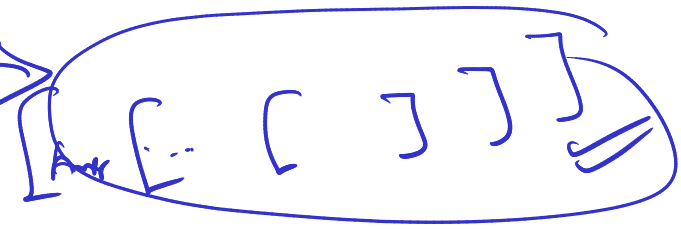
$1 \times 0 = 0$

Ternary operator:

def fact

Recursive function:

Traversing a Nested List:



names = ["Amar", ["Sukhi", ["Dhanush", "Raj"], "Ruchi"],]

index

names[1]

names[1][1]

len(names) ⇒ 3

(an we implement recursively?

Step 1: Define the function

Assignment / Homework:

1) Implement recursion on the list below: → Try to complete this

names = ["Akshita", ["Sachin", ["Sameer", "Tejas"], "Rithin", "Angeli"], "Lavish"]

Welcome to Day 15 of 50 Days

26/11/2021

Object Oriented Programming (OOPs)

Background: ⇒ Case Study: Design a program to solve differential equations

Add function:

1) Object orientation in Python:

- everything is an object consisting of two things → data, → functions (methods)

- methods such as len(), replace(), upper(), lower(), split(), join()

2) What is a Class?

- A class is a template for an object.

Shopping Cart → wordpress

3) How are classes created in Python?

- Using keyword "class" followed by ClassName or className

camelCase

~~variables~~

For eg: Add two numbers & print the result.

①
```
num1 = int(input("enter ---"))
num2 = int(input("enter num2"))
print(num3 = num+num2)
```

②
```
def addition():
```

① Using classes:

```
class Addition:
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def add(self):
        return self.a + self.b
c = Addition(3,2)
print(c.add())
```

output 5

Jargon Busting:

- class
- __init__
- self

~~init~~ ~~__init__~~

Eg2: A more practical example:

for instance: "This is who I am" (It's)

Expectation: ① No. of words ⇒ returns 5
② Exactly 2 character ⇒ 2
③ Starts with t ⇒ 1
④ Ends with m = 1

Hints: (import functions from string) (Data pre-processing) (Data Science/ML)
① Remove the list → input the text (sentence)
② Replace all punctuations with Empty String('')
③ Convert all to lowercase and use split method

Pseudocode + flowchart

④ Create respective function (eg starts_with, ends_with)

⑤ print the result

Concept of Inheritance:

Victor Ikechukwu Agughasi → family → Ancestors (fore fathers)

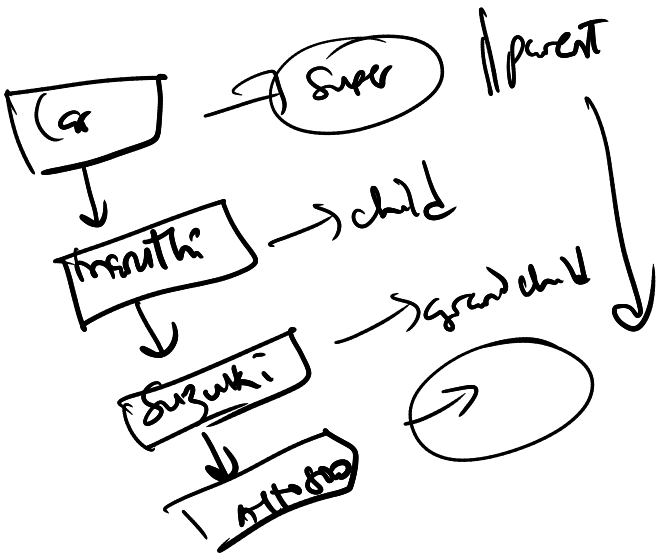
Victor is the son of Agughasi

Victor → Son (child class)

Victor
↑
Stephen
↑
Nwokeabia
↑
Nnejiorah
↑
Agughasi → Super class

(Great Great Great Grand Pa)

Victor inherited traits from Stephen inherited traits from




```
class parent:
    def __init__(self, a)
        self.a = a
    def method(self):
        return self.a*2

class child(parent):
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def method1(self)
        return self.a*7
```

```
def method2(self)
    return self.a + self.b

#create object
pr = parent("hello")
ch = child("Hi Jay", "welcome")

#Call the method
print(pr.method1())
print(pr.method2())
print(ch.method1())
print(ch.method(2))
```

Game Development: Using OOPs Concept:

Logic: A case study of Tic-tac-toe

What are the rules? Google (Thanks you!)

What are functions needed to prototype the game:

Step 1: check_open_spots(): Should return list of positions that have not been selected.

Step 2: check_valid_move(): return True if move (row, col, direction) else false

Step 3: make_a_move(): only in the valid directions (repeat until Condition → Draw / Winner)

Step 4: check_winner(): first to place before the opponent else (Draw)

Sat: 10-11 am

→ Exception handling ✓
→ Some more projects ✓
→ GitHub Repo ✓

fork

Day 16 of 50 days of Python:

Sat → 27/11/2021
Sun → 28/11/2021

CLASSES: OOP Concept.

Python Scope & Namespaces:

A particular (Gocblock) where the namespace is directly ✓ (JS)

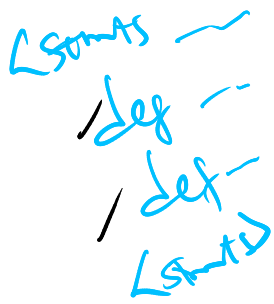
✓ mapping from name to objects.
✓ implemented using py dict.
✓ Note: No relation between names in different namespa

For Example:

```
def G_Scope():
    def L_Scope():
        a = "Hello"
    def nScope():
        b = "non Hello"

G_Scope()
print("Access the global scope", a)
L_Scope()
print("--")
nScope()
print("--")
```

Class SuperEg:



cl = SuperEg:

Notes on Instantiating Object:

Class Variables vs Instance Variables

For Eg:

```
kind = "feline" #global
class Cats:
    kind = "felines"  # class variable; shared by all instances
    def __init__(self, name):
        self.name = name  # instance variable; unique to each instance
ct = Cat("Persian")
cb = Cat("African Cat")
ct.kind
output = feline
```

ct.name output: Persian
cb.name out: African Cat

Python using Classes.

```
def StuGrade:
    purpose = "Student's grade"
    sem = "7th Sem"

st = StuGrade()
print(st.sem, st.purpose)
```

Can we reassign values to them

```
st2 = StuGrade()
st2.sem = "8th Sem"
print(st2.sem, st2.purpose)
```

st1.purpose

What are Iterators? Used as container objects for looping through list item by the iter(), subsequently using __next__()

list[12]

list = [1, 2, 3, 4, 5, 6]

Loop through using for loop

__init__ through StopIteration Exception

Index out of Bound Exception

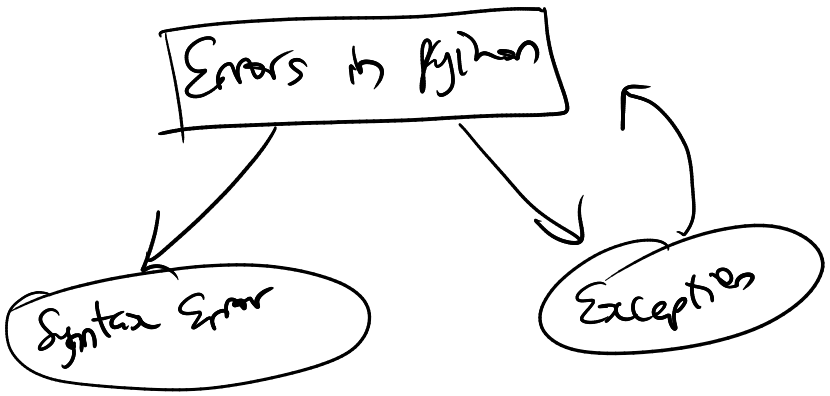
What are Generators? Used to optimize codes.

- ✓ Similar to tuples but efficient
- ✓ Versatile and flexible to use than tuples.
- ✓ ~~No double~~, we can also use List Comprehension.

Errors vs Exception

Day 17th of 50

Python Exceptions:



Syntax Error: (caused by wrong syntax in code)

for Eg: print("Hello")

✓ Easy to handle

✓ always lead to ~~poss~~ program termination

Exception: Errors raised when the program is syntactically correct, but the code resulted in an error (logically wrong)

Eg: 10/0, 3-2//0

x = [1,2,3,4]

x[4] => IndexError

Remainder after division

3 mod 2 = 1

num1 num2

even vs odd

num1.num2 == 0

✓ Various types of Error in Python? about 20+ (ask Google)

✓ Exception hierarchy: Thanks to google (parent-child relation)

base vs sub-class relationship

Exception handling:

Try and Except Statement

For eg: Consider the code below.

```
x = [1,2,3,4]        (len(x) = 4)

try:
    print ("Second element is %d", %a[2])
    print ("fourth  "  %d.", [a[4]])
except:
    print ("Index Error occured")
```

output = 3 — Error-free

error prone

Catching Specific Exceptions:

Syntax:

```
try:
    #Statement
    ---
except: # IndexError:
    #Statements
except ValueError:
except ZeroDivisionError:
```

An Example:

```
def hi(a):
    if a<3:
        b = a/a-3
    print("Value is ...", b)

try:
    hi(5) ; no error (2nd)
    hi(3) ; Error.
except ZeroDivisionError:
    print ("Can't divide by zero")
except NameError:
    print ("Name error has occured")
```

Handling Exception in Else Statts

```
def summ(a,b):
    try:
        c = ((a+b)/(a-b))
    except ZeroDivisionError:
        print("Results in error type")
    else:
        print("The actual result is ",c)
```

summ(4,6) = 10/-2 = -5

(4,4) 8/0 = 0

Use of finally keyword:

(Similar to default in Switch case)

Syntax:

```
try:
    //some statts
except:
    //statt
else:
    //statt
finally:
    //statt.
```

Can Exceptions be raised?

Regular Expression: Introduction

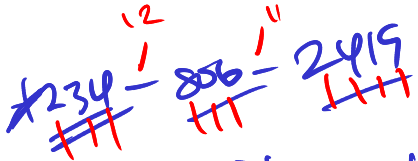
Ctrl+F and Replace

"Sevlet" ⇒ Exact match aka pattern → Similarity → length

234-555-1234 ✓

234-563-9274 ✗

Regular Expressions extends the function of searching in word processor.



→ Is a time saver

The knowledge of regEx gives rises to productivity

import re;

re.~

Qsc: How can we validate a phone no without regEx?

Ans: write methods:

```
def isValid(text):
    if len(text) != 12
    return false
```

Day 20 of 50 days of Python:

RE: Missing Notes: ✓ Done

X X X X

Introduction to GUI Programming using

Tkinter → 2.7
tkinter → 3.0

GUI vs CUI

Container based app (Console base)

↓

to beautify our apps
using external functionalities
(libraries)

Graphical User Interface: A form of user interface that allows you to interact with the computer using icons, menus, windows etc.

Advantages/merits over CUI (Character User Interface)

- Ease of access
- flexibility

What is tkinter?

Ans: An inbuilt python module that is used to create GUI apps.

- most commonly used modules for GUI apps in python - why?
  - Its easy to use
  - Lots of widgets added to the tk GUI toolkit

Other modules / python libraries are:

- ~~Kivy~~. Kivy
- Pyqt or wxPython

What are Widgets?

Widgets in tkinter refers to element of GUI apps providing various control such as Label, Buttons, ComboBoxes, checkboxes menubars, radio button etc.

Note:

Structure (Basic) of a tkinter program

Step1: Import tkinter modules

Step2: Create the main window for GUI

Step3: (optional) Adding widgets to the app

Step4: calling/create the main event "Loop"

Tkinter Widgets

Widgets	Description
1) Label	Used to display text or images/graphics on the screen
2) Button	" " add buttons to your application
3) Canvas	Canvas : to draw pictures and other graphics
4) ComboBox	

Demo: first tkinter program

WIDGETS CLASSES

Widgets	Description
Frame:	A Container to hold and organize the widgets