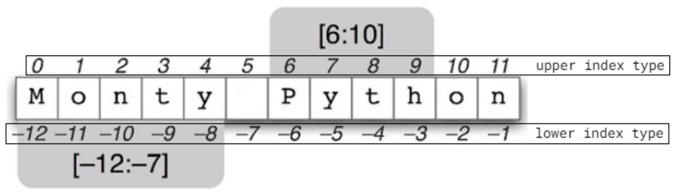
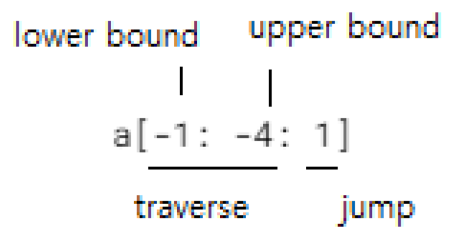
String

String slicing & subset

 [6:10] 0 1 2 3 4 5 6 7 8 9 10 11 upper index type M o n t y P y t h o n –12 –11 –10 –9 –8 –7 –6 –5 –4 –3 –2 –1 lower index type [–12:–7] Scale concept	 lower bound upper bound a[-1: -4: 1] traverse jump

a= “Monty Python”

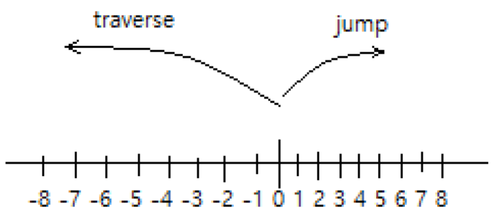
special problems have to be memorized.

1. **Special prob01**

   a[-1: -4: 1]

   = ‘’

   Because there is a conflict of direction between the traverse and jump, we will neither get any value nor error.



2. a[0:300:1]

   == Monty Python
3. a[-2:]

   == on
4. a[::-1] or a[-1::-1] or a[-1:-300:-1]

   == ‘nohtyP ytnoM’
5. A[-2:5:1]

   == ‘’
6. a[-2:5:-1]

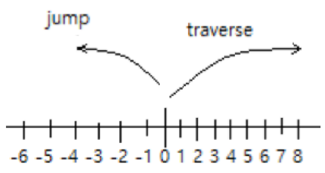
   == ‘ohtyP ytnoM’

7. **Special prob02**

   A[0::-1]

   == ‘M’

The traverse is upper index type where we want to traverse from 0 but doesn’t have an upper bound.
On the other hand the jump is negative, which



makes **conflict of direction** between traverse and jump. Conflict of direction creates out of scope indexes, thus after index 0 it won't be able to parse any data from the **list.**

8. **Special prob03**
   A[0:4:-1]
     == []

other methods on string

1. a.find('o')
   Will return the index value of the first o in the string
2. a.count('o')
   Will return the number of o in the string

3. a.split()
   Will return a list of words in the string.
   ['Monty','python']

4. a.split('o')
   Will return a list of words in the string splitted where there is 'o'.
   ['M','nty pyth','n']

5. a.swapcase()
   Will return the string where lowerCase will be upperCase
   And upperCase will be lowerCase.
   'mONTY pYTHON'

6. a.title() or a.capitalize()
   Returns the same string but the first letter as upperCase.

7. ' '.join("test")
   = 't e s t'

8.

```
for i reversed("mubin"):
           print(i)
```

```
= n
  i
  b
  u
  m
```

9. s= ‘ mubin no ‘

s.rstrip()	‘🟥mubin🟥no‘
s.strip()	‘mubin🟥no🟥‘
s.lstrip()	‘mubin🟥no‘

10. s.replace(‘no’,’no1’)
    =‘ mubin no1 ‘

11. The method that starts with is return boolean

s= ‘mubin’	s.isupper()	False	
s= ‘mubin’	s.isalnum()	True	Check whether **number** and **alphabet** or not
s= ‘mubin’	s.isalpha()	True	Check whether **alphabet** or not
s= ‘ß’	s.isascii()	False	Unicodes are not: ex: õ Å ,ß Ascii values:#, &,$,’’,\n,abc,ABC,0123
s= ‘mubin’	s.isdigit()		Not digit: ‘100$’, ‘100 ‘,’10.5’ Digit:100, '\u0038','۶'
s= ‘mubin’	s.isidentifier()		
s= ‘mubin’	s.islower()		
s= ‘mubin’	s.isnumeric()		
s= ‘mubin’	s.isprintable()		
s= ‘mu bin’	s.isspace()	True	Returns true if space exists in the string.

Input String Value	isdecimal()	isdigit()	isnumeric()
'123'	True	True	True
'$123'	False	False	False
'123.50'	False	False	False
'123a'	False	False	False
'¾'	False	False	True
'\u0034'	True	True	True

Advanced string

```
L = [ 2,432,431,11,"sudh", 3+4j, [32,51,13,"abc"]]
```

1. Spit word from a string into a list

```
listOfString=[]
for i in L:
    if type(i)==str:
        for j in i:
            listOfString.append(j)
print(listOfString)

['s', 'u', 'd', 'h']
```

2.

List vs String

The major difference between them is mutability.

- List is a mutable entity
- But string is not mutable
  - L1[0]=”k” is possible, but
  - “kuddus”[0]= “A” is not possible.

But can someone argue about the replace() method?

1. kuddus.replace(‘u’,’i’)

   This will return ‘kiddis’

While it does look like mutability, the main problem is we don’t know the actual definition of mutability.
Mutable means, being able to bring changes of a variable in its actual memory location.
And here in the replace() method what happens is, the replace() method creates a new reference or new space in memory.

Let's look at the code for more clarification:

```
i= "kuddus"
print(id(i))
print(id(i.replace("u","i")))
```

The output of the code will be like:

```
140154552143536
140154475638448
```

List

```
L1 = [‘kuddus’,’kumar’,2342]
L2 = [’xyz’,’pqr’,4134.41]
```

List concatenation

1. L1 + L2
   = [‘kuddus’,’kumar’,2342,’xyz’,’pqr’,4134.41]
2. L1*2
   =[‘kuddus’, ’kumar’, 2342, ‘kuddus’, ’kumar’, 2342]
3. L1+ [“amir”]
   = [‘kuddus’,’kumar’,2342, ‘amir’]
4. merging two list
   l1.extend(l2)
   == [‘kuddus’,’kumar’,2342,’xyz’,’pqr’,4134.41]
5.

List manipulation

1. 134 in L1
   Will check whether 134 available in L1 list or not
   = False
2. L2.append(“nana”)
   Will add “nana” add the last index of the list.
   =[’xyz’,’pqr’,4134.41,“nana”]
3. L2.pop()
   Will pop the the last index variable in the list
   =[’xyz’,’pqr’,4134.41]
4. L2.pop(1)
   = [’xyz’,4134.41]
5. index=1
   L2.insert(index,”**mubin**”)
   = [’xyz’,”**mubin**”,4134.41]
6. L2=[’xyz’,”**mubin**”,4134.41]
   L2.reverse()
   = [4134.41,”**mubin**”,’xyz’]
7. L2[1][2]
   =’b’

Method append() vs extend()

```
L3=[21312,123,12,312,312]
```

1. L3.append([**11,222,3333**])
   = [21312,123,12,312,312,[**1,222,3333**]]
2. L3.extend([11,222,3333])
   = [21312,123,12,312,312,**1,222,3333**]

List advanced

```
listADV = [1,2,3,3,4,,4,4,5,6,4,5,6,7,8,9,9,9,’k’,’k’]
```

1. set(list) returns set of unique values in the list
   Find unique values in list

```
        factors = [ 1,25,11,1,1,5,5,5 ]
        == set(factors)
        == { 1,25,5 }
```

2. The occurrence of an item in a list from a list of items

```
        Occurrence of unique=[1,2,3] in listADV
        == listADV.count(unique[0])
             == occurrence of 1 in listADV = 1
        == for i in unique:
              print(listADV.count(i))
             == 4
                2
                1
```

3.

Reverse a list or tuple or anything

```
L3=[21312,123,12,312,312]
```

There are two way

1. Slicing

```
L3[::-1]

== [12, 124, 123, 5, 5, 51, 10]
```

2. For loop

```
for i in range(len(dd)-1,-1,-1):
    print(dd[i])

== [12, 124, 123, 5, 5, 51, 10]
```

List comprehension

Tuple vs List

Tuple is immutable like string	List is mutable
Tpl = ()	Lst = []
Tpl. count()	Lst. append()
index()	clear()
	count()
	extend()
	index()
	insert()
	pop()
	remove()
	reverse

Tuple vs Int

Tuple is a **collection of data** thus a tuple with	Only one data inside will treat it as **int**

t1= (1,4,1,2,4,4,55,1233)
type(t1)
== **tuple**
	t1= (1)
	type(t1)
	== **int**

Tuple

```
T1 = (‘sudh’, 324, ‘adf’, 3123.11, True)
```

- Tuple is immutable
- Tuple is a collection of all type of data type

Tuple slicing

This is same as String slicing

Other methods on Tuple

1. T1.index(‘sudh’)
   = 0
2. Tuple is immutable so we can’t operate the following statement
   == T1[0]=’mubin’
3. Tuple is a collection of all type of data type
   Thus we can assign any type of data type in the tuple
4. To assign values of a tuple in multiple variable

   ```
   a,b,c,d,k= (35,1,561,"asdf",[123,155,54])
   K
   == [123, 155, 54]
   ```

5.

List to Tuple

1. To transform a list into tuple or tuple into list use
   = tuple() and list()
2.
3.

Set

```
S = [1,2,3,3,4,,4,4,5,6,4,5,6,7,8,9,9,9,’k’,’k’]
```

A Set is

- an **unordered collection data** type
-
- that is **iterable**,
- **mutable** and
- has **no duplicate** elements

S.	remove(4)	Remove 4 from the set, otherwise through error.
	discard(4)	Remove 4 if it's available in the set.

To find unique values inside a list use set()
set() return the unique value inside its parameter

1. Ss = set(s)

   ```
   = {1,2,3,4,5,6,7,8,9,k}
   ```
2. Set is a unordered collection of data type
   Thus trying to access its data through indexing will
   Through error

   ```
   S[0]
   TypeError: set object is not subscriptable.
   ```
   But to access data it has to be transformed into
   List, and access like we access data from list

   ```
   list(S)[0]
       == 1
   ```

3. Set can hold only an unordered collection of unique elements, primitive and **immutable** data types not in between collections[like: list,tuple...], thus it is able to Distinguish between duplicates inside its datas.
   a. Primitive data are,
      float,
      complex number,
      Int…

   So if we try to insert values other than primitive

data type it will throw an error.
{[1,23,4],3,51,2,51,5,}
TypeError: unhashable type: ‘list’

b. Immutable data types are
Tuple,
String,
In this case no error will be thrown.
{‘mubin’,3,51,2,51,5,}
{(1,23,4),3,51,2,51,5,}

4. Set is an unordered data type thus the output and the assignment in a set will not be shown same
Example:

```
s={13,1535,65,324236,675,74,2,3,51}

For i in s:
    print(i)
```

```
2
3
51
74
65
324236
13
1535
```

5. How to opened data in set:
You have to convert into a list() then again convert it back to set()
6.

Advanced set

```
set = {1,2,3,4,5,6,7,8,9,k}
```

1. Length of set
   == len(set)
2. Iterate through set values
   == test = list(set)
   == test[2]
   == 3
3.

Set vs dictionary

set	dict
Both of their **notation** is {}	
S = {1,2,4}	S = {}

Dictionary

```
d={ 33:"asdf", "key1":2131, "123/*":"mubin", {a:1,b:2,c:3} }
```

- Special character can't be a key in dictionary
  - Key has to be immutable element

Viable	Not viable
Immutable element	mutable element

Tuple	list
Double variable will work	_d won’t work
string	

- Multiple keys with the same name will result in older key values being replaced with newer ones.
- It is possible to store dict inside another dict

d.

get(“key1”)	Return the key’s **value** otherwise **blank**.
update(d2)	Updates d dict with d2 items.
fromkeys()	Help add key and value in existing dict.
values()	Returns **a list** of values of all key
items()	Returns **list** of tuples of **key and value in pairs**.
keys()	Returns all the keys of d
pop()	Delete last item
popitem()	
setdefault()	
updatev	

1. Item assignment operation
   d[“keyNew”]=”it's a new data”
   == { 33:”asdf”, **“key1”:2131**, “123/*”:”mubin”, {a:1,b:2,c:3},”keyNew”:”it's a new data” }
2. Update value of a key
   == d[33] = ”33isnew”
3. To delete a item
   == Del d[key1]

```
{ 33:”asdf”, “123/*”:”mubin”,{a:1,b:2,c:3},”keyNew”:”it's
a new data” }
```

4. To delete a whole dictionary

   == del d

5. Add multiple items in dict using tuples

```
key=(“name”,”mobile”,”email”)
value=(“mubin,”123123”,”afdsf@gmail.com”)
```

   == d=d1.fromkeys(key,value)

6.

Opend/concatenation operation in dictionary

```
d1={ 33:"asdf", "key1":1111, "key3":"mubin",
"multiple":{"a":1,"b":2,"c":3} }
d2={ 33:"afterUpdate", "key2222":222222, "key4":"didar",
"multiple":{"d":1,"e":2,"f":3} }
```

1. To update two dictionary into 1

   d1.update(d2)

   == replace value having same key name

   Add new item if key of d2 is not available in d1

```
{33: 'afterUpdate', 'key1': 1111, 'key2222':
222222, 'key3': 'mubin', 'key4': 'didar',
'multiple': {'d': 1, 'e': 2, 'f': 3}}
```

2.

Same operations in dictionary

d.get(“key100”) vs d[“key100”]	
Try accessing a not available key from a dictionary d[“key100”] It will through error	d.get(“key100”) It won’t

For loop in Dictionary

1. To find list of key with a condition (whose length is more than something)

```
d={"india":"IN",
   "canada":"CA",
   "bangal":"BD",
   "india122":"IN11",
   "canada12":"CA11",
   "bangal12":"BD11"
}
```

```
l=[]
L=[]
for i in d.keys():
    if len(i)>5:
        L.append(i)
    elif len(i)<=5:
        l.append(i)

print("l",l)
print("L",L)
```

```
l ['india']
L ['canada', 'bangal',
'india122', 'canada12',
'bangal12']
```

2.

Nested dictionary

```
d={"india":{
    "a":12,
    "b":331,
    "c":66

    },
   "BD":{
       "d":34,
       "e":62,
       "f":466

    }
}
```

```
d_1={"india":{
    "a":12,
    "b":331,
    "c":66
    },
   "BD":{
       "d":34,
       "e":62,
       "f":466
    },
   "Nepal":{
       "g":34,
       "h":"asd",
       "i":466
    },
    "j":34+4j,
   "hard":(1,2,3,4,6,33,5),
    "hard_2":[13,51,6,123,656],
     "hard_3":"sudh"
}
```

1. Find max of int values in every dictionary in nested dictionary

   This can be done in three approach

   i.

```
a=[]

for i in d.values():
    a.append(list(i.values()))

for i in a:
    print(max(i))
```

   ii.

```
for i in d.values():
    print(max(i.values()))
```

   iii.

```
for i in d.values():
    mx=0
    for j in i.values():
        if mx<j:
            mx=j
    print(mx)
```

2. Find max of only integer type values in every dictionary in nested dictionary

   i. This is better but

```
tt=[]
for i in d_1.values():
    if type(i) == dict:
        for k in i.values():
            if type(k)==int:
                tt.append(k)
    elif type(i)== int:
        tt.append(i)
    elif type(i) == tuple or type(i)==list or
type(i)== set:
        for k in i:
            if type(k)==int:
                tt.append(k)
```

ii. A bit Better version

```
l = []
for i in d_1.values():
    if type(i) != str and type(i)!= complex:
        if type(i) == list or type(i) == tuple or
type(i) == set:
            l.extend(list(i))
        elif type(i) == dict:
            for j in i.values():
                l.append(j)
print(max(l))
```

iii.

Generator vs Iterator vs Enumerate

- We will use iterator function just to understand the internal mechanism of for loop
- Generator is used for memory efficiency or yielding operation[sound was not clear]
-

Typecasting

1. If i want to type cast a int value inside a string
    a. I have to use try catch to check type casting to int is possible or not

User input

To take input from user we will use the method called input()

- As a default, this method typecasts any data type into a string.

    - Using the following examples, we can avoid such a situation:
        - To int: int(input())
        - To double: double(input())
- Using the parameter of input(), it can be displayed to the user while taking the input

If else statement

```
If … :
Elif… :
Else :
```

1. if/else condition returns a boolean value.
2.

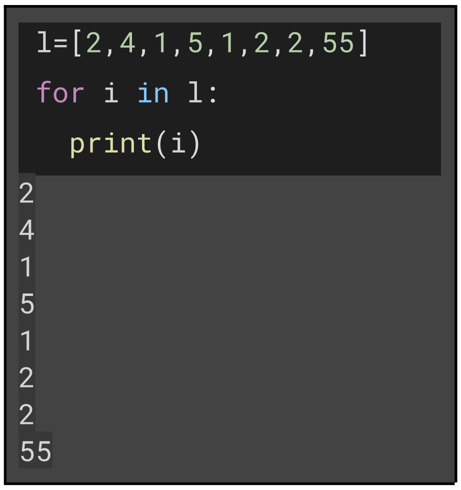
Bitwise operator vs Logical operator(& vs &&)

Bitwise & operator	Logical && operator
The bitwise & operator is used to compare two digits, resulting in a new digit.	The logical && operator compares two booleans, resulting in a boolean value

result= 5 & 6
print(result)
== 4
 	
0110 ==5
0101 ==6
---- and
0100 ==4
	result = True & False
	print(result)
	== False

For loop

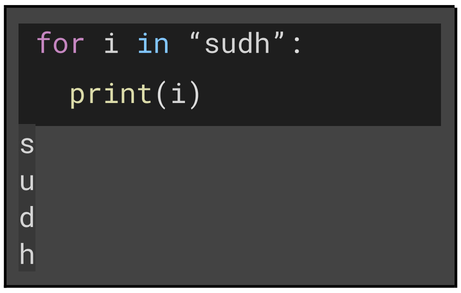
1. For loop in a list



```
l=[2,4,1,5,1,2,2,55]
for i in l:
  print(i)
```

```
2
4
1
5
1
2
2
55
```

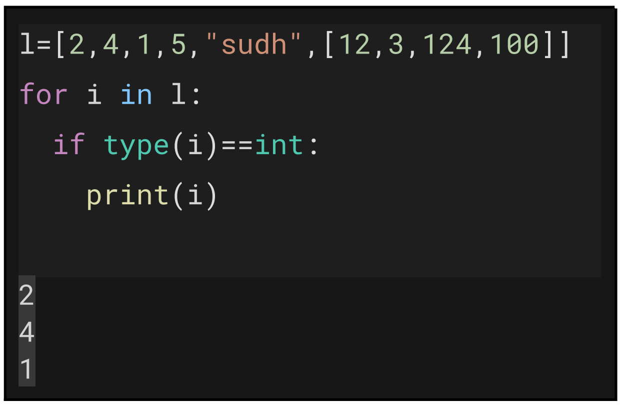
2. For loop in string



```
for i in “sudh”:
  print(i)
```

```
s
u
d
h
```

3. For loop with type cognition



```
l=[2,4,1,5,"sudh",[12,3,124,100]]
for i in l:
  if type(i)==int:
    print(i)
```

```
2
4
1
```

```
5
```

4. Find vowel in string, find word from string in string

```
s=”hi my name is nonw”
vowel=”aeiouAEIOU”
For i in s:
    If i in vowel:
        print(“vowel”,i)
    Else:
        print(“not vowel”,i)
```

5.

For loop sample problems:

```
L = [ 2,432,431,11,”sudh”, 3+4j, [32,51,13,”abc”]]
```

1. Print index of all the element
2. Extract all the list of char if element is string
3. Return a list after doing a square of all the int element

Ans 1:
In this case, you should avoid index() function since it returns the index of the first element every time for same element within the list
There are three principle way we can ans this question:
1. avoid index() function.
2. Using for loop with len() method
```
a. for i in range(len(L)):
b.   print(i)
```
3. Using for loop with enumerate
```
a. for i,j in enumerate(L):
b.   print(i)
```
4. Using list comprehension

Ans 2:

```
L = [ 2,432,431,11,"sudh", 3+4j, [32,51,13,"abc"]]
listOfString=[]
for i in L:
    if type(i)==str:
        for j in i:
            listOfString.append(j)
    elif type(i)==list:
        for k in i:
            if type(k)==str:
                for kk in k:
                    listOfString.append(kk)

print(listOfString)
```

For loop in Dictionary

```
d={ “a”: “fsdfsd”, ”b”:”fsdfsdf”, “c”:[2,4,1,4], “e”:”sudh”}
```

1.

```
For i in d:
    print(i)
```

```
==  a
    b
    c
    d
```

2.

```
For i in d:
    print(d[i])
```

```
== “fsdfsd”
   “fsdfsdf”
   [2,4,1,4]
   “sudh”
```

3.

```
For i in d.items():
```

```
== (“a”, “fsdfsd”)
   (”b”, ”fsdfsdf”)
   (“c”, [2,4,1,4],)
```

`print(i)`	

4.

While loop

continue vs **break** vs **pass**

These statements are used inside loops,

- “continue” continues the while loop without executing the next line.
- “break” on the other hand terminates the entire loop and executes the next line after loop.
- “pass” lets us run a loop without any definition inside it, to avoid error.
    - If we don’t want to write a statement inside a loop we use “pass”
    - Pass never through you in a loop
    - Example:

For i in l: pass Nothing shows	For i in l: == error

Map

```
l="141 2 4 12 4 124"
```

kk=l.split()
integer_map = map(int, kk)
integer_list = list(integer_map)

```
== [141, 2, 4, 12, 4, 124]
```

Function

- len()
- print()
- list()

These are all functions, some have created them, and we are able to use it, when required by passing our parameters...

Why use function

When you are providing a service to your client, or working in a professional environment, we have to follow a complete modular coding approach. Otherwise even if our code is working, in code audit no one will accept it even if it's working.

What consists in modular programming

An important part of **modular programming** is to separate the functionality of a program into independent, interchangeable modules.

- Function
- Classes
- Object
- Logging
- Monitoring
- scheduling

All of these are part of the modular coding approach.

And modular programming languages are, all the object-oriented programming languages like

C++, Java, c# etc.,

What we have to keep in our mind while writing function

- Function declaration will start with: def

-

```
1. To concatenate with function with other data types
        def test():
                Return “its a string”
        “I have concatenated with test() ” + test()
        == “I have concatenated with test() its a string”
2. If i return multiple data type/collection from a function,
        It will return a tuple of collection the return data
        example:
        Def test():
                Return 3,5,”sudh”,[12,4,125,1]
        test()
        ==(3,5,”sudh”,[12,4,125,1])
3.
```

Python Pattern Programs

New things

1.

```
s="hi my name is none"
vowel="aeiouAEIOU"
for i in s:
    for j in vowel:
        if i==j:
            print("vowel",i)
    else:
        print("not vowel",i)
print(type(s[2]))
```

2.
3.