Agenda

23 December 2024 22:37

Recap
Data types
- **int, float, Boolean**
- **String**
- **List**
- **Tuple**
- **Dictionary**

Recap

23 December 2024 23:22

→ python
→ Python version release structure (alpha, beta, final)
dev solving bugs Release

→ Google Colab (Gpu server free).

→ Overview of Datatypes

→ Keywords & Identifiers

→ Data types
→ what is data type in python
→ int, float, boolean, String

String :-

```
x = " data"
y = " dreamers"
print ( x + " - " + y)
x = input ("please enter your name : ")
y = input ("Please enter your age : ")
print ("My name is " + x + "and my age is" + y)
" ".format
```

Normal Data types(int, float, Boolean)

23 December 2024 23:22

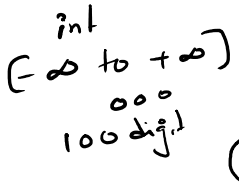
Int: It is one of the data type to store integer values (can be -ve & +ve)

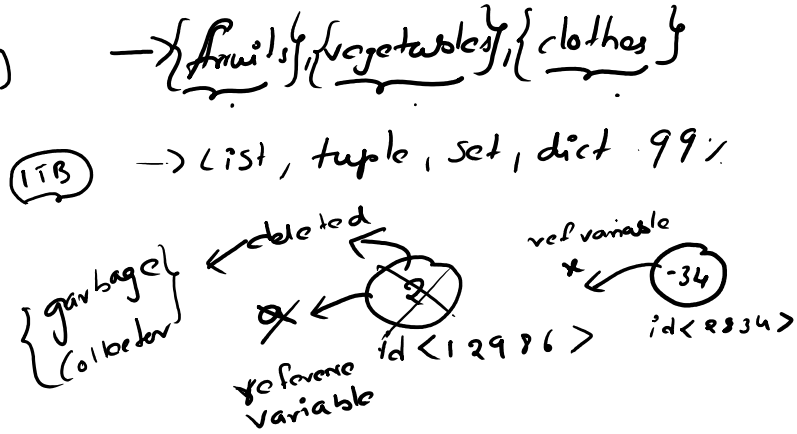
Declaration syntax:

a=2
x=-34
y=56565
z=0

What operations we can perform?

a=2, b=3
a+b
a-b
a%b
a/b





Float: It is another data type to store float values (can be -ve & +ve)

Declaration syntax:

a=2.3
x=-34.33
y=56565.45
z=0.0

What operations we can perform?

a=2.3, b=3.1
a+b
a-b
a%b
a/b

Boolean: It stores True and False

Declaration syntax:

_c = True
__f123=False

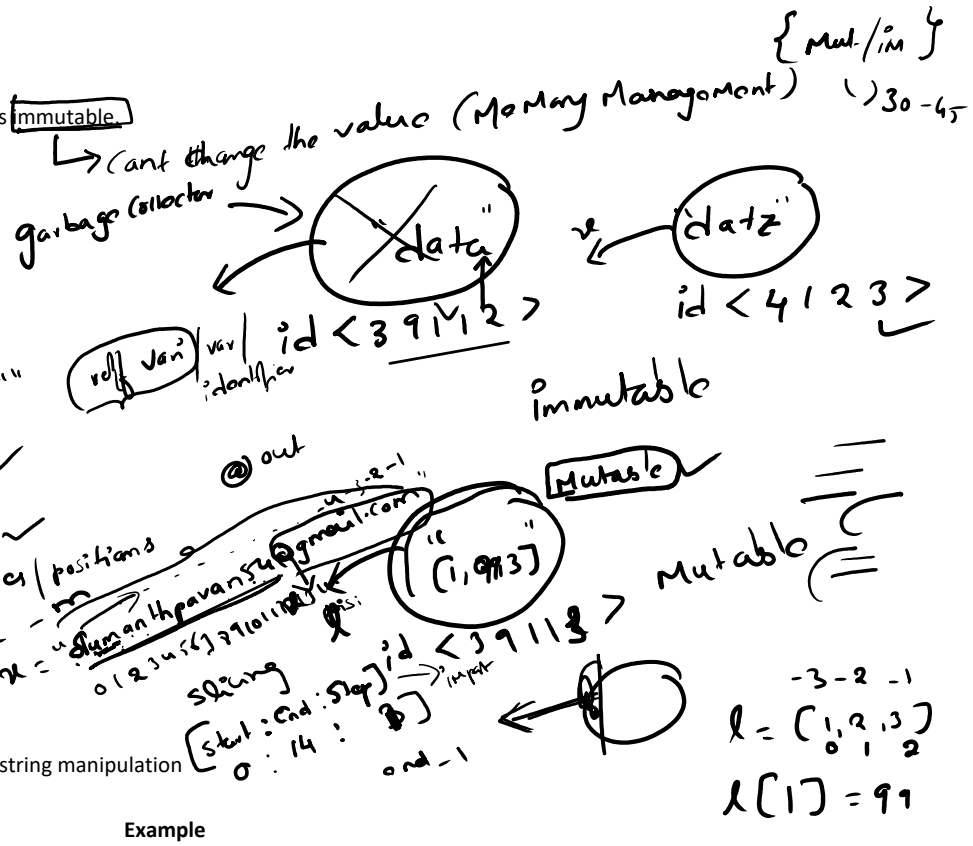
What operations we can perform?

X=True, Y=False
X and Y
X or Y
not X
not Y

String

23 December 2024 23:12

-It is used to store sequence of characters and it is immutable.



Declaration syntax:

```
x= 'data'
y="dreamers"
z='''I am good
      At python'''
r="""I am good
      At pyspark"""
```

What operations can we perform?

```
x= 'data'
y="dreamers"
```

Concatenation
Indexing
Slicing
length

String Methods

Python provides a variety of built-in methods for string manipulation

Method	Description	Example
lower()	Converts all characters to lowercase.	"Python".lower() → "python"
upper()	Converts all characters to uppercase.	"Python".upper() → "PYTHON"
strip()	Removes leading and trailing spaces.	" Hello ".strip() → "Hello"
replace(old, new)	Replaces a substring with another.	"Python".replace("Py", "My") → "Mython"
split(delimiter)	Splits a string into a list.	"a,b,c".split(",") → ['a', 'b', 'c']
join(iterable)	Joins elements of an iterable into a string.	" ".join(['Hello', 'World']) → "Hello World"
startswith(sub)	Checks if the string starts with a substring.	"Python".startswith("Py") → True
endswith(sub)	Checks if the string ends with a substring.	"Python".endswith("on") → True

String Formatting

It helps you to display the string outputs in readable, maintainable, and adaptable to changing values.

f-strings

str.format() Method

List

23 December 2024 23:23

-It is a collection of items and it is mutable

Declaration syntax:

```
a=[1,2,3,4,2.3,True,"apple"]
x=[1, "apple", True, 3.5,[2,3,4]]
y=["apple", "banana", "apple"]
```

What operations we can perform:

- **Concatenation**
- **Indexing**
- **Slicing**
- **length**

List methods:

Method	Description
append(item)	Adds an item to the end of the list.
extend(iterable)	Adds all the elements of an iterable (e.g., another list, tuple) to the end of the list.
insert(index, item)	Inserts an item at the specified index.
remove(item)	Removes the first occurrence of the specified item.
pop(index)	Removes and returns the item at the specified index. If no index is provided, removes the last item.
clear()	Removes all items from the list.
index(item)	Returns the index of the first occurrence of the specified item.
count(item)	Returns the number of times the specified item appears in the list.
sort(key=None, reverse=False)	Sorts the list in ascending order by default. Accepts optional key and reverse parameters.
reverse()	Reverses the order of the list in place.
copy()	Returns a shallow copy of the list.

Function:

Sorted(iterable,key=None, reverse=False) The sorted() function always returns a **new list** and does not modify the original

---

fun (VS) Methods (oops)

fun (vs) Methods (oops)

① functions —

inbuilt & user-defined

len( ) default/fix
min( )
max( )
mode( )
mean( )

def <fun_name>:
    ≡

(or)

def simple_intrest(p,t,r):
    s.i = (p*t*r)/100
    return s.i

y = 3 years
R = 8.02
P = 10000

S.I = PTR/100
= 10000×3×2/100
(or)
= 10000×3×0.02

def Validiction(age):
    ⟶ {
    return True/False

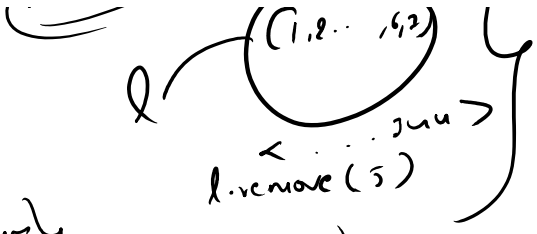
② method :-

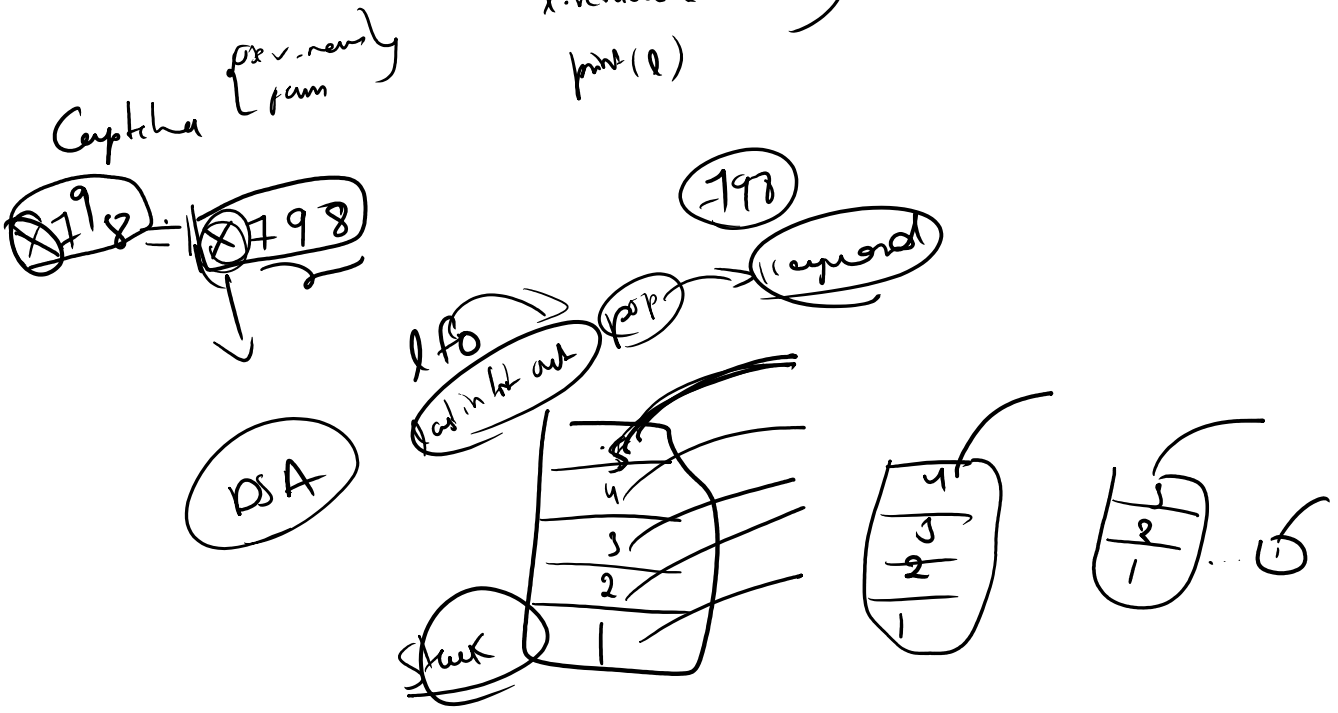
class Temp:
    def simple_intrest(self):
        s.i = (p*t*r)/100
        return s.i
    → Method

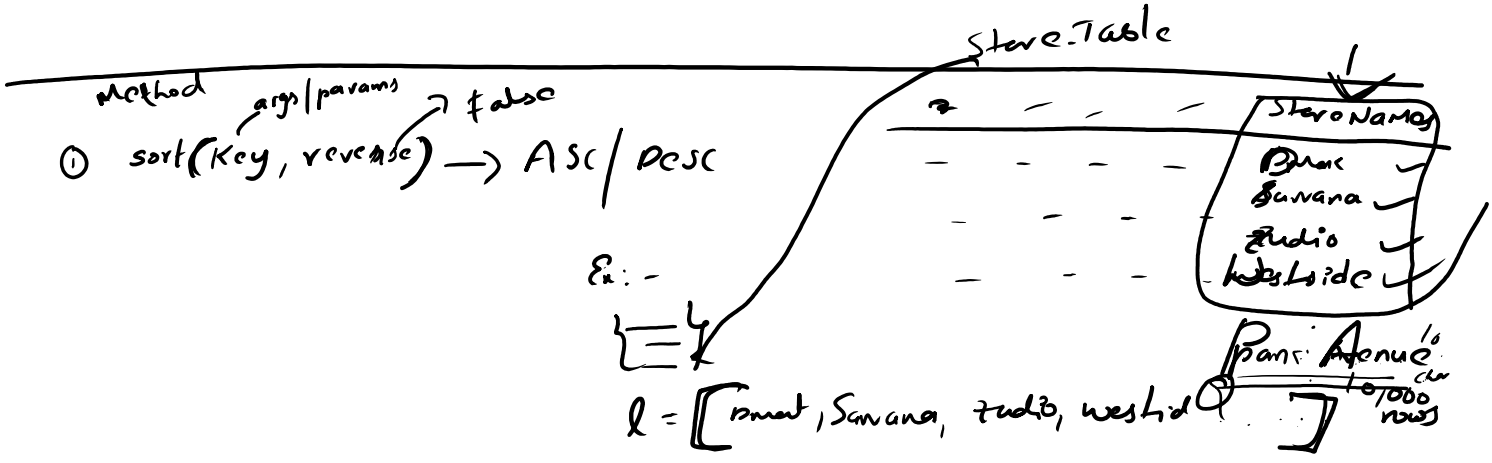
Qu:-

class List:
    def append( )
    def extend( )
    def insert( )
    def remove( )

datatype.Methodname( )







Scenario1:- l.sort() => Ascending

print(l)

[A, B . . . . C . . . Westside, Zudio]

Scenario2:- l.sort(reverse=True) => Descending

[Zudio, Westside . . . . . C . . . B A]

Key arg/param:-

Ex:- Sort as per length of storeName

Sort as per Capital letters

l.sort(key=len)reverse

len, Asc

print(l)

[A, a, —, Smart]

---

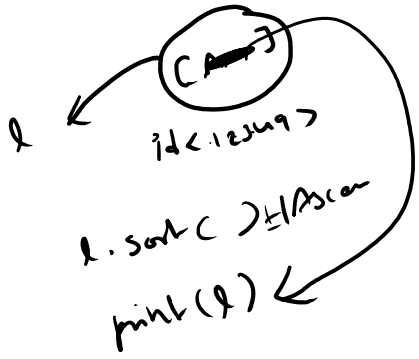
sorted (iterable, key, reverse)

{ Any datatype on which you can apply looping

sort, sorted

l = [ ]

l.sort( )

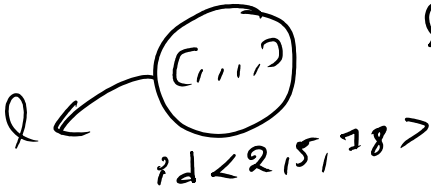


l.sort( ) ±1/Asc

print(l)

l = [,,,]

sorted(l)



l1 = sorted(l)

---

shallow copy ✓

l = [ ] ✓

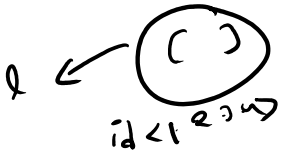
l1 = l.copy( ) ✓

aliasing ✓

l = [ ]

l1 = l

Deep (copy)



Tuple

23 December 2024 23:23

-It is a collection of items and it is **immutable**

Declaration syntax:

```
x = (1, 2, 3, "apple", True)
y=("apple", "banana", "apple")
z=(1, 2, 3, "apple", True,(1,2,3,4),[1,2,5,6])
r=(1,)
t=tuple()
```

What operations we can perform:

- **Concatenation**
- **Indexing**
- **Slicing**
- **length**

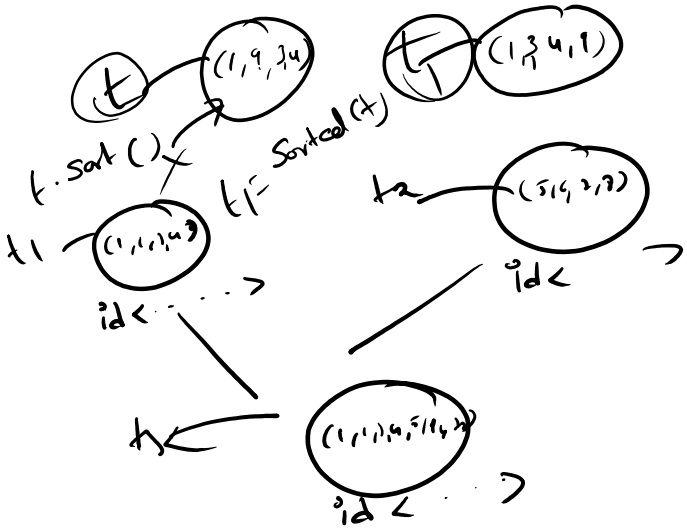
Methods:

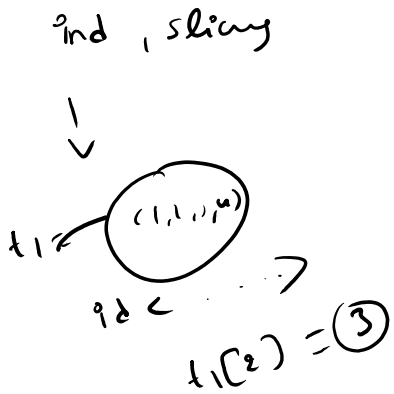
Method	**Description**
count(item)	Returns the number of times the specified item appears in the tuple.
index(item)	Returns the index of the first occurrence of an item. Raises ValueError if the item is not found.

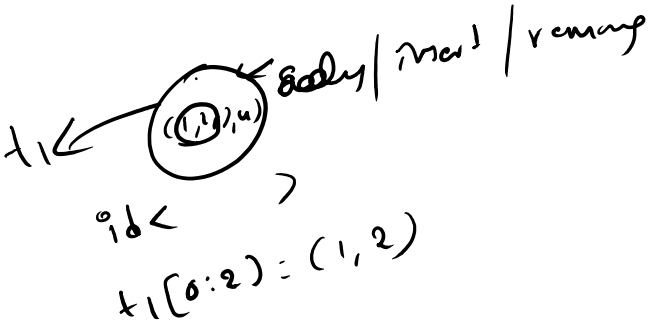
Topics:
When to use tuples
Why sorted() function can be applied but not sort()
Tuple unpacking

Ex:-

```
t1=(1,2,3,4)
t2=(5,6,7,8)
t3 = t1+t2
print(t3)
print(id(t1))
print(id(t2))
print(id(t3))
```

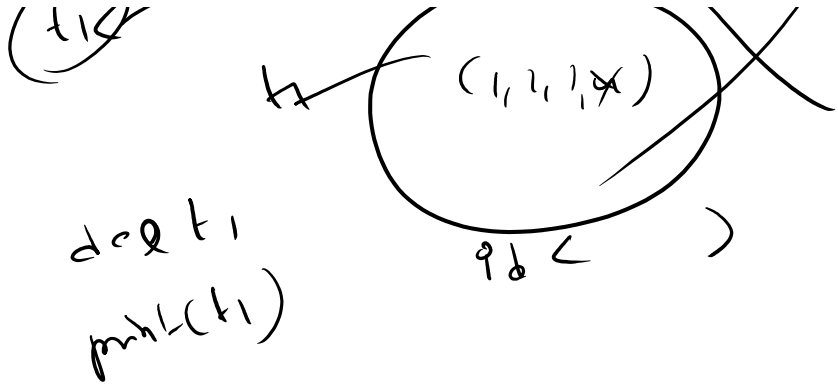




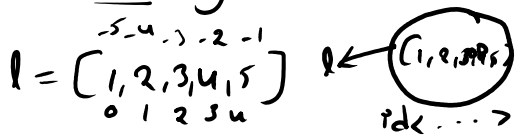


```
t1=(1,2,3)
del t1
print(t1)
```

```
t1=(1,2,3)
del t1
print(t1)
```

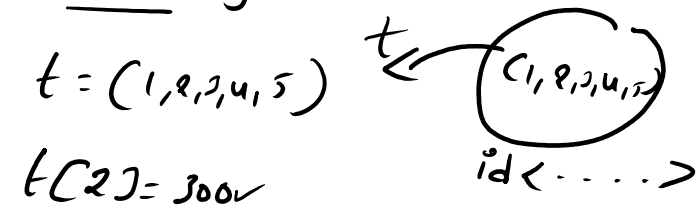


Mutability



```
l=[1,2,3,4,5]
l[2]=300
print(l) o/p [1,2,300,4,5]
```

Immutability



```
t=(1,2,3,4,5)
t[2]=300
print(t) o/p:- Error
```

When to use tuple?

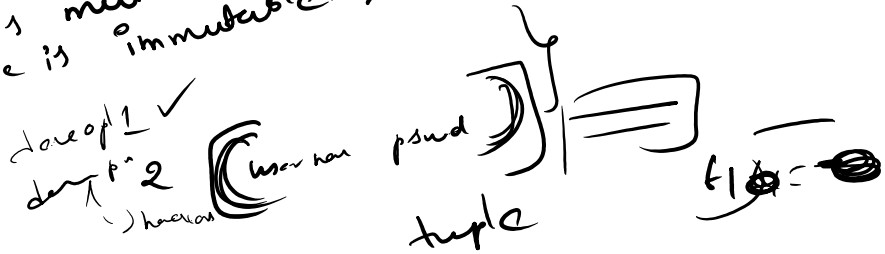
Ex: 1) t = (23.786, 21.796) l = [23.786, 21.796]

t[1] = 100 l[1] = 100

2) days_of_week = ("mon", "tue" ... "sunday") l = ["mon" ...]

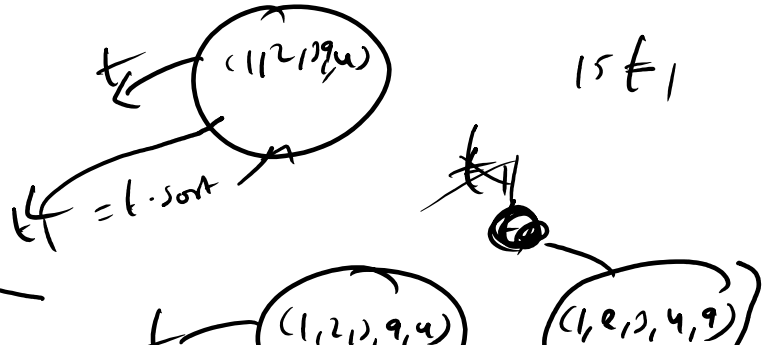
diff b/w list & tuple

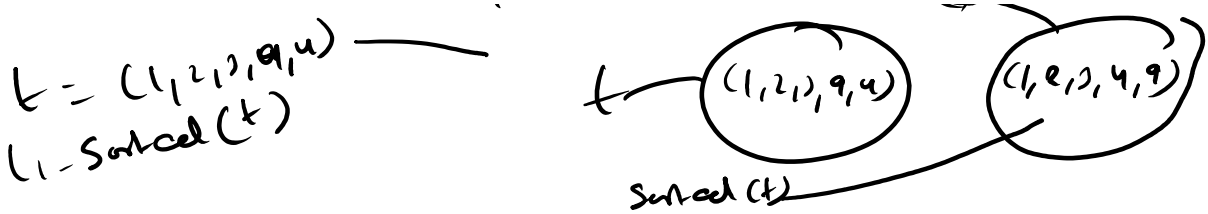
1) list is mutable → can change in same memory
2) tuple is immutable → cannot " "



Ex: 2 t = (1,2,9,4) →

t1 = t.sort()




t = (1,2,1,9,4)
l_sorted(t)
t
(1,2,1,9,4)
(1,2,1,4,9)
sorted(t)

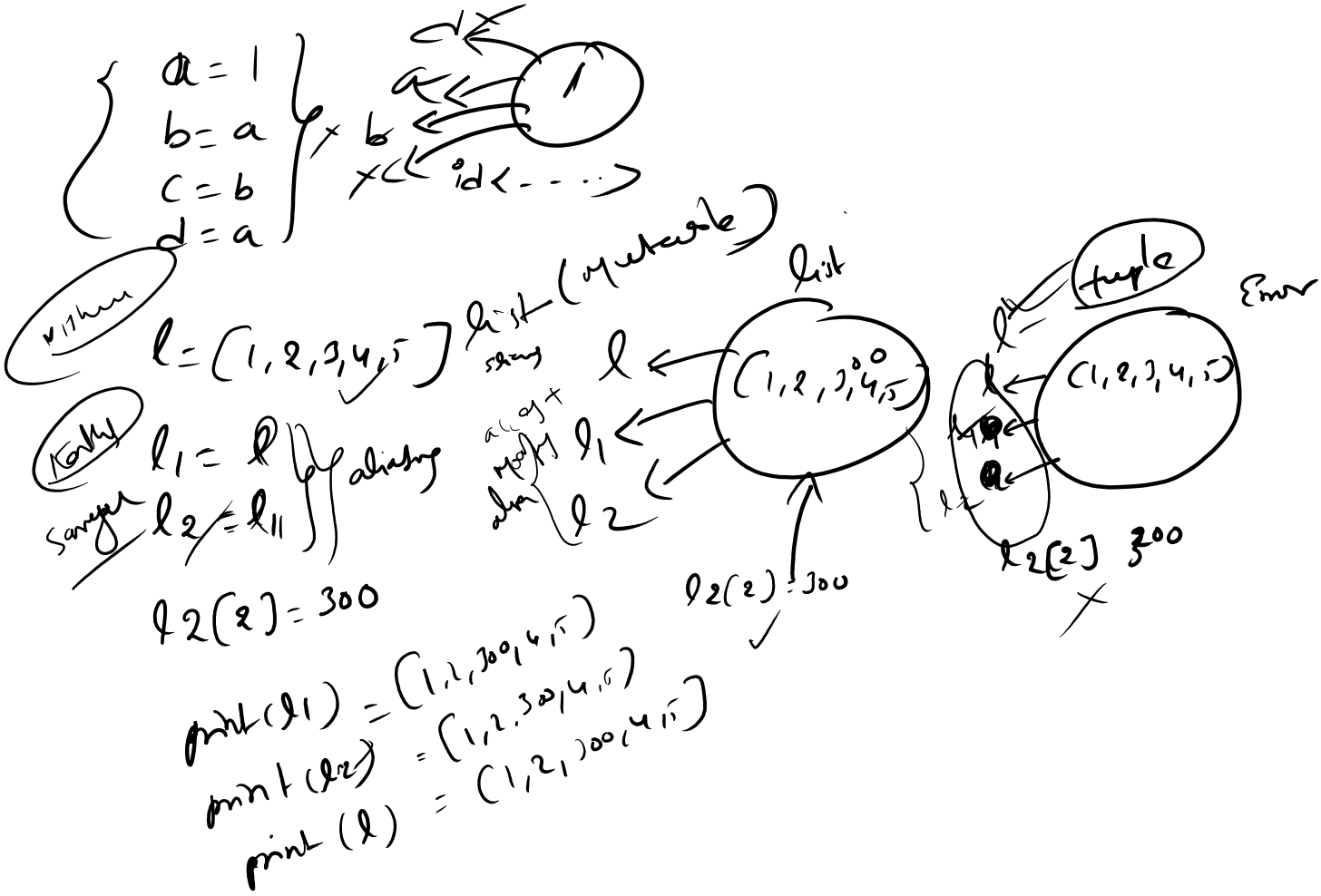
Miscellaneous Interview Questions

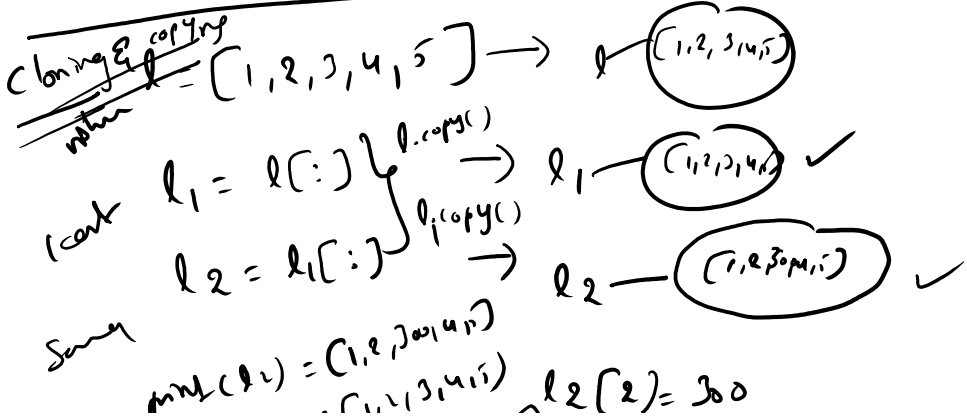
29 December 2024 22:47

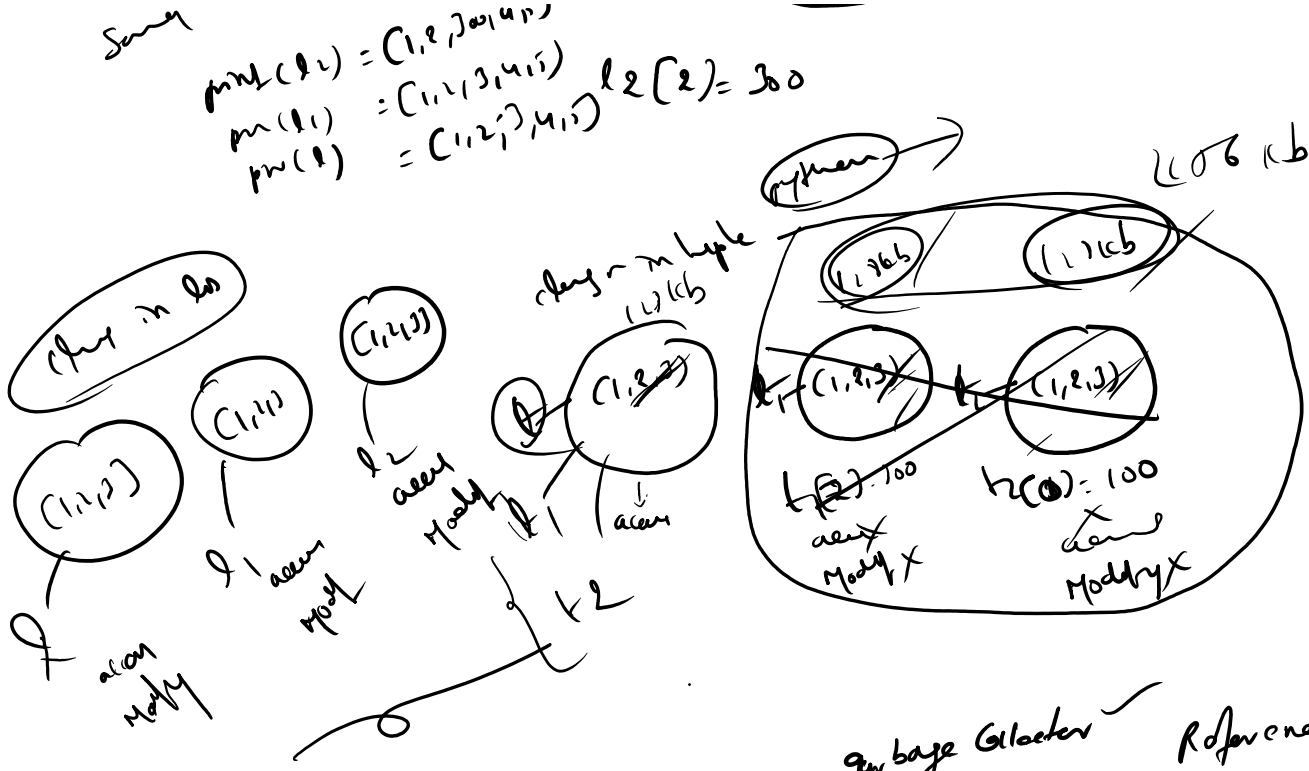
What is Aliasing? ✓
Cloning & Copying ✓ (alternatives to aliasing, so that the entity memory remains same)
What is the role of Garbage Collector ✓
How tuples and lists are stored? ✓
Why tuples takes less memory and time for execution? ✓
Difference between tuple and list ✓
What is mutability and how can we handle it (Cloning and Copying)
Shallow Copy vs Deep Copy

What is Aliasing?

→ pet name, original name, friendly name (same person)



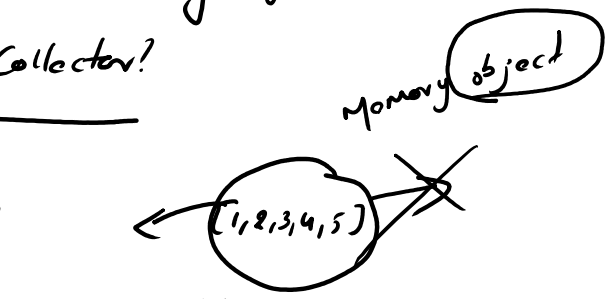


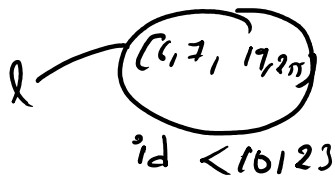


garbage Collector Reference counting

What is role of garbage Collector?

```
l = [1,2,3,4,5]
l = [6,7,19,255]
print(l)
[6,7,19,255]
```

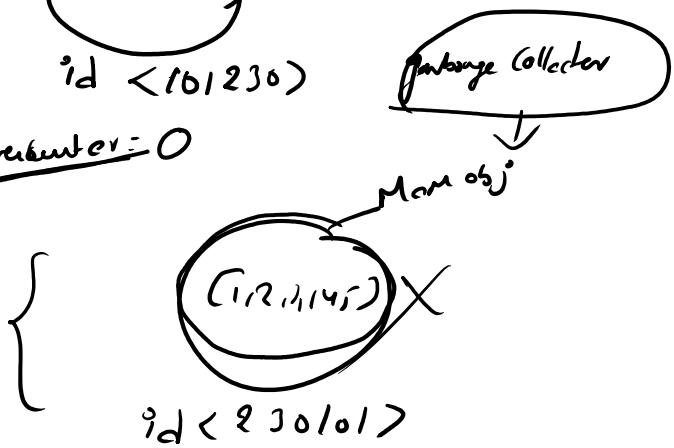


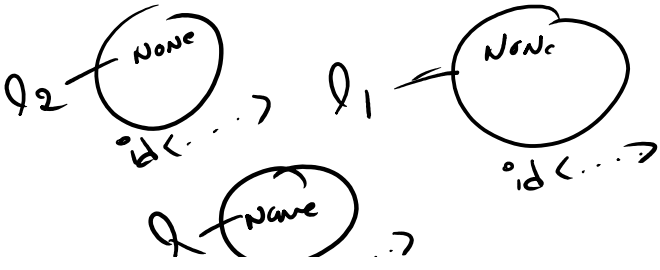


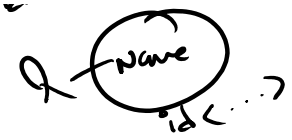
Reference Counting :-

```
l = [1,2,3,4,5]
l1 = l
l2 = l1
l1 = None
l2 = None
l = None
```

Reference counter = 0



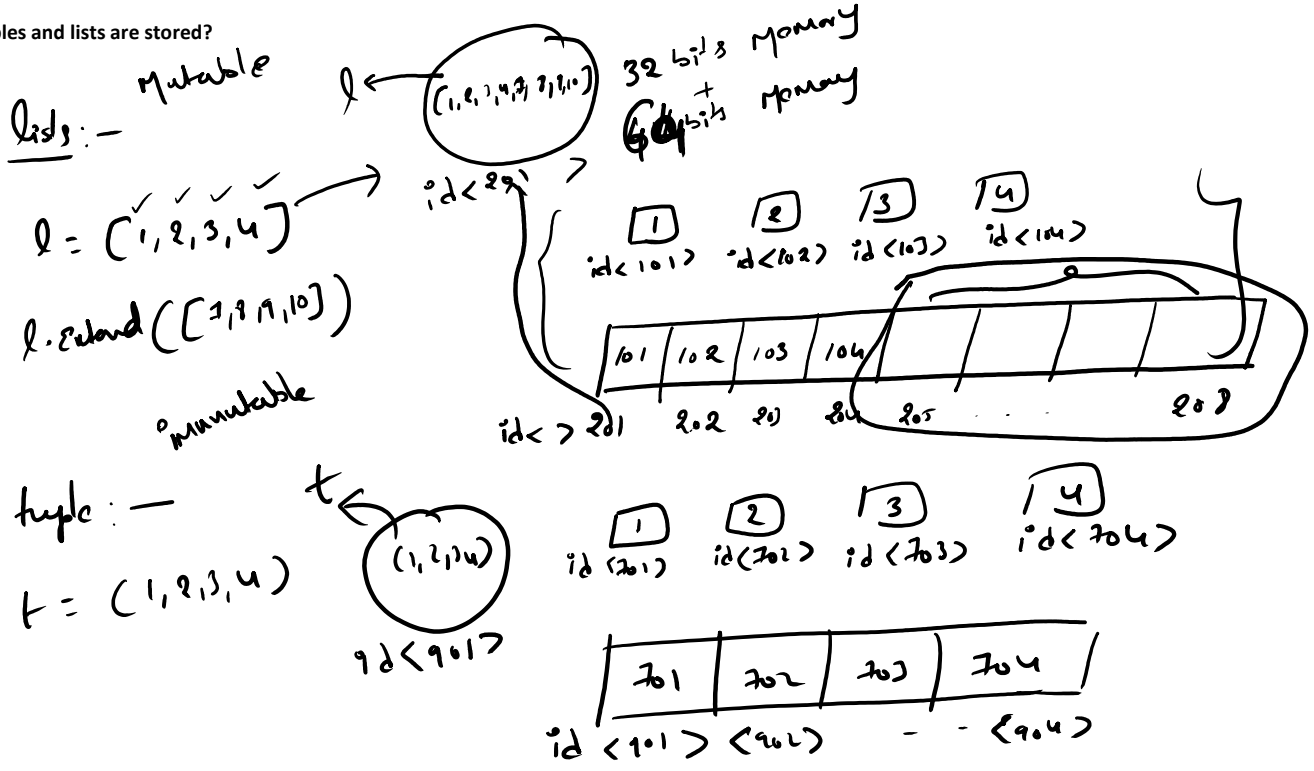




Note :- Python is a dynamic in allocating & deallocating the Memory

---

How tuples and lists are stored?



Why tuples takes less memory and time for execution?

-> list will take more memory than tuple (dynamic free space & mutability)

-> list will take more Execution time than tuple

Difference between tuple and list

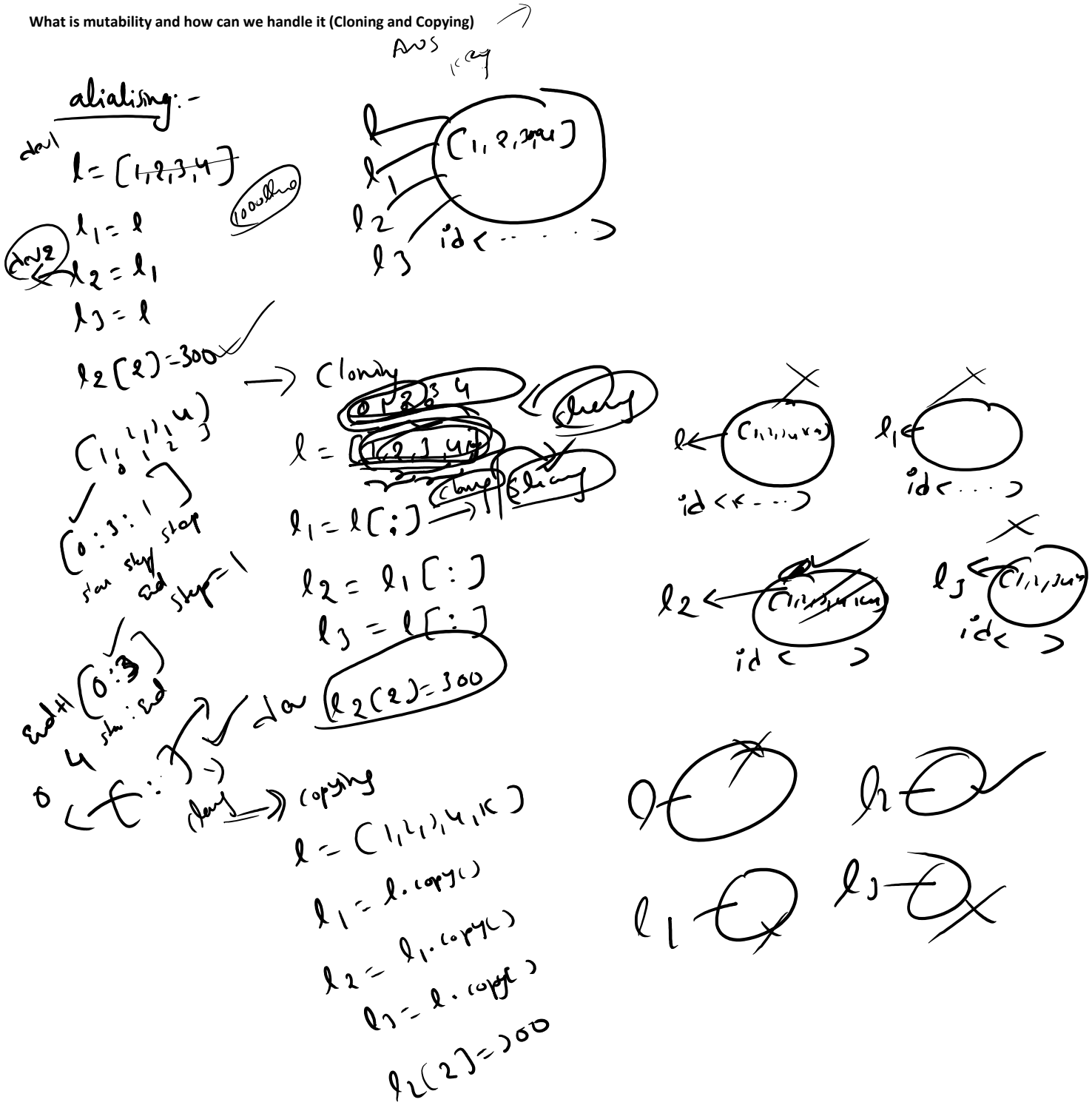
	list	tuple
1)	Mutability	immutability
2)	More Memoryspace	less Memory space
3)	More Execution time	less Execution time
		Syntax is diff ( )

3) More execution time

4) Syntax is different [ ]

Syntax is diff ( )

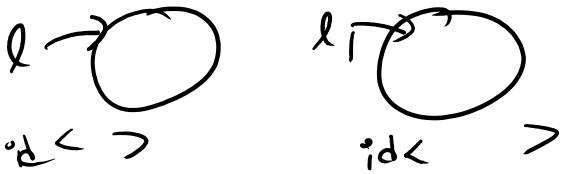
What is mutability and how can we handle it (Cloning and Copying)



Shallow Copy vs Deep Copy

l = [1, 2, 3, 4]

l1 = l.copy() ✓ shallow copy

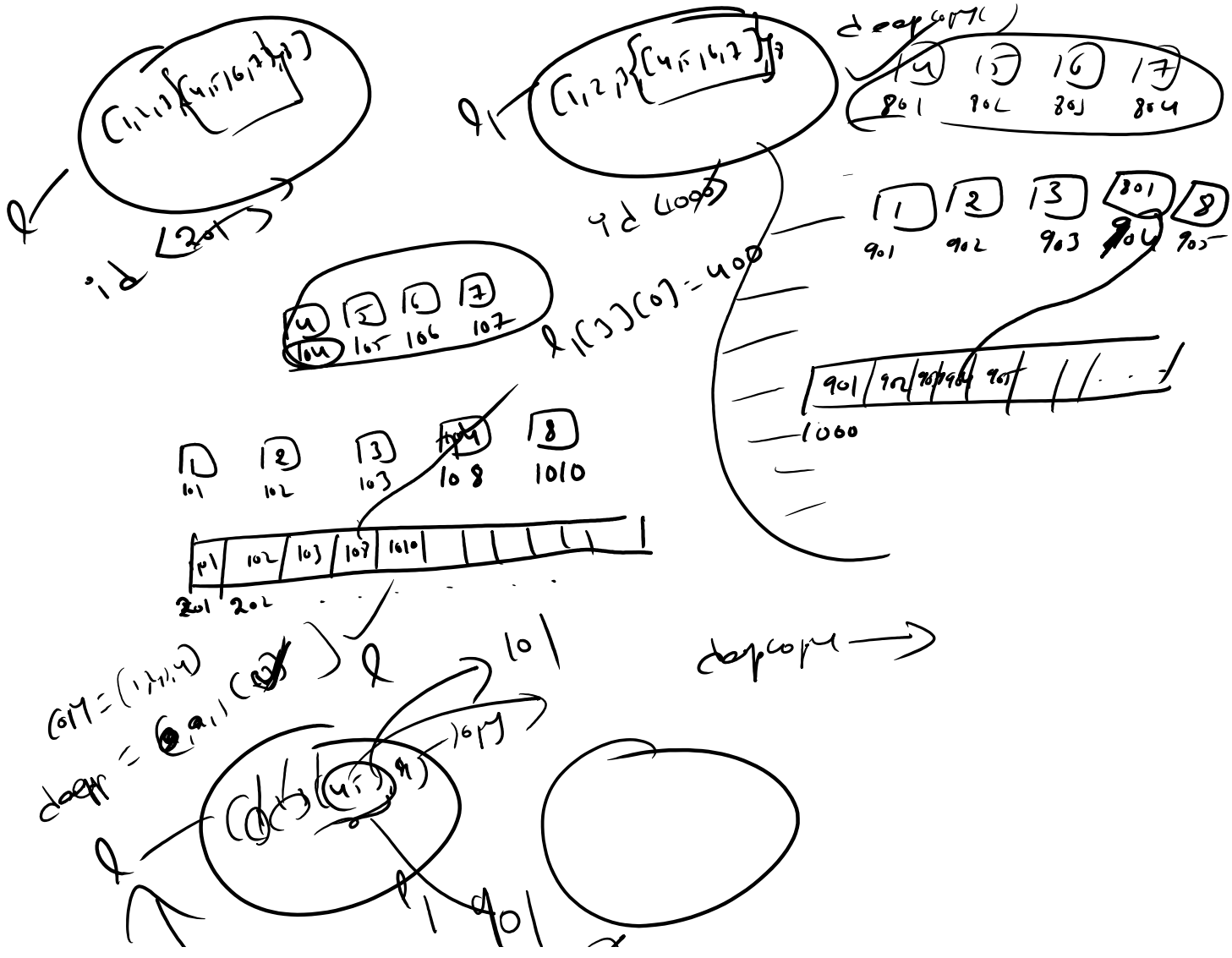


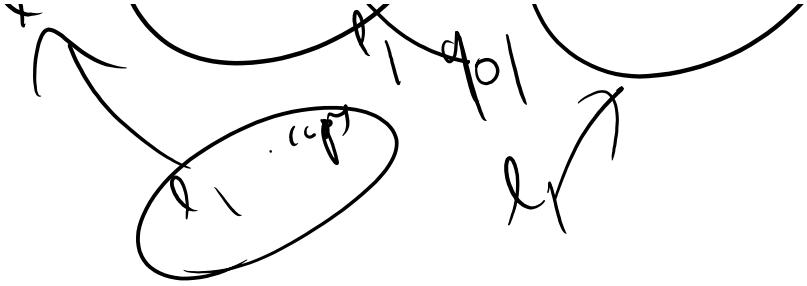
l = [1, 2, 3, [4, 5, 6, 7], 8]

1
2
3
[4, 5, 6, 7]
8

l1 = l.copy()

l1 = copy.deepcopy(l)





Set

25 December 2024 19:53

It is also collection of items

Remember 4 rules:

1. Set doesn't allow duplicates
2. Set is by default mutable
3. Set doesn't allow indexing and slicing
4. Set allows immutable datatypes inside it.

Creation of Sets:

==========

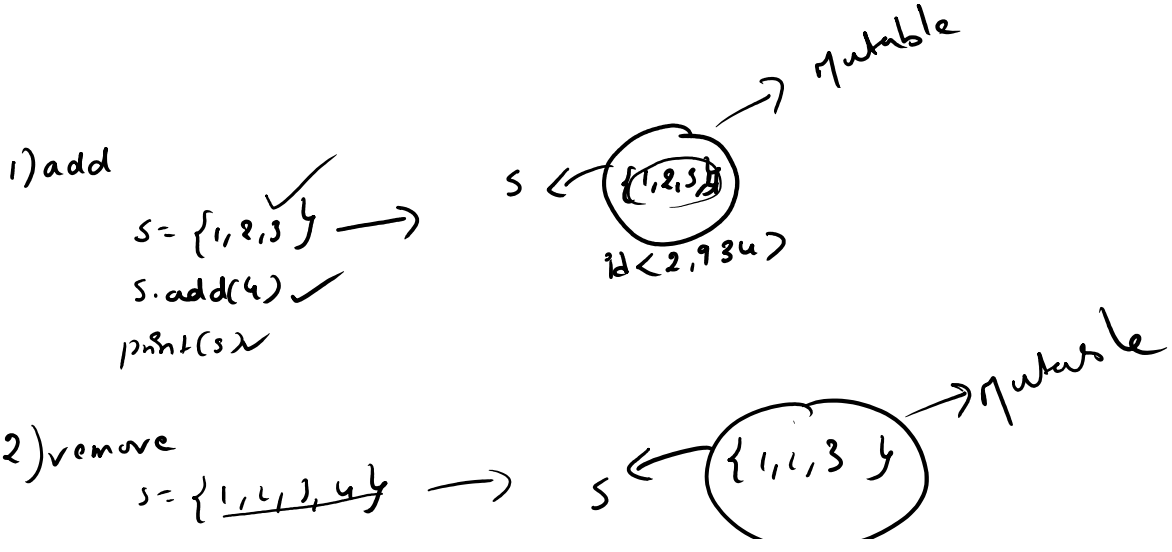
1. s={1,2,3,4}
2. s=set()
3. s={"ugiug","uhiug","yftyut"}
4. s={1,2.3,"juhg","True"}
5. s=set("god")

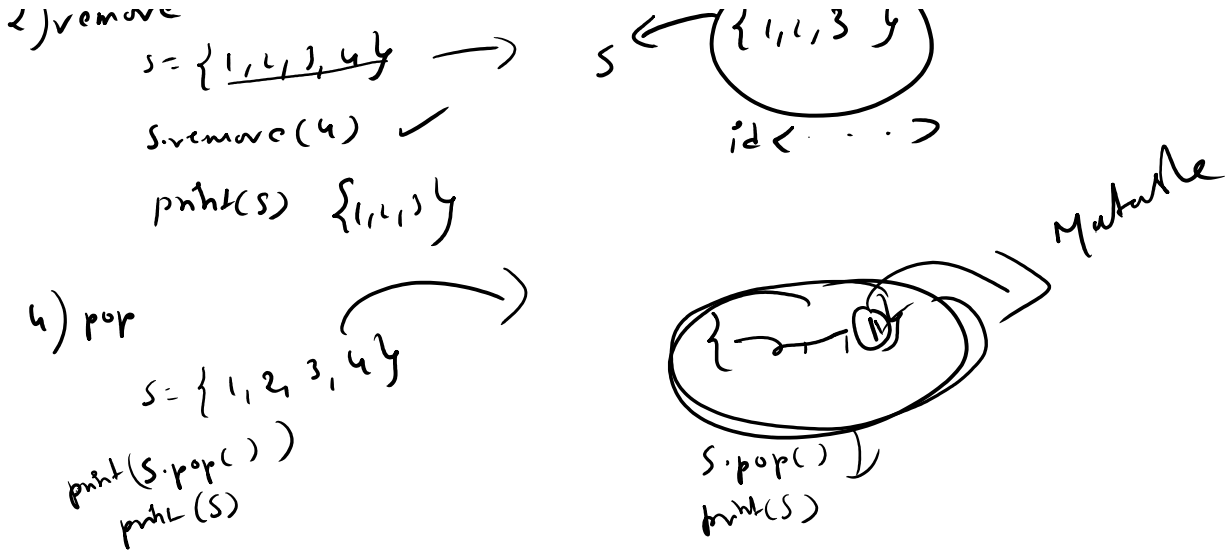
Operations:

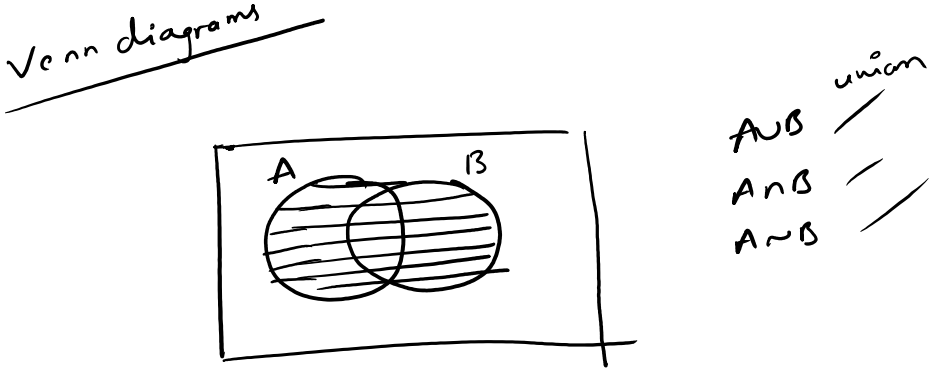
==========

Concatenation
Indexing
Slicing
Length
in

Method	Description
add(element)	Adds a **single** element to the set. If the element is already present, it has no effect.
remove(element)	Removes a specified element from the set. Raises a KeyError if the element is not found.
discard(element)	Removes a specified element from the set if it is present. Does nothing if the element is absent.
pop()	Removes and returns an arbitrary element from the set. Raises a KeyError if the set is empty.
clear()	Removes all elements from the set, making it empty.
union(*others)	Returns a new set containing all unique elements from the set and other sets (or iterables).
intersection(*others)	Returns a new set with elements common to the set and other sets (or iterables).
difference(*others)	Returns a new set with elements in the set **but not** in the other sets (or iterables).
symmetric_difference(other)	Returns a new set with elements in either the set or the other set, but not both.
update(*others)	Updates the set by adding elements from other sets (or iterables).
intersection_update(*others)	Updates the set with the intersection of itself and other sets (or iterables).
difference_update(*others)	Updates the set by removing elements found in other sets (or iterables).
symmetric_difference_update(other)	Updates the set with the symmetric difference of itself and another set.
isdisjoint(other)	Returns True if the set has no elements in common with another set.
issubset(other)	Returns True if the set is a subset of another set.
issuperset(other)	Returns True if the set is a superset of another set.
copy()	Returns a shallow copy of the set.

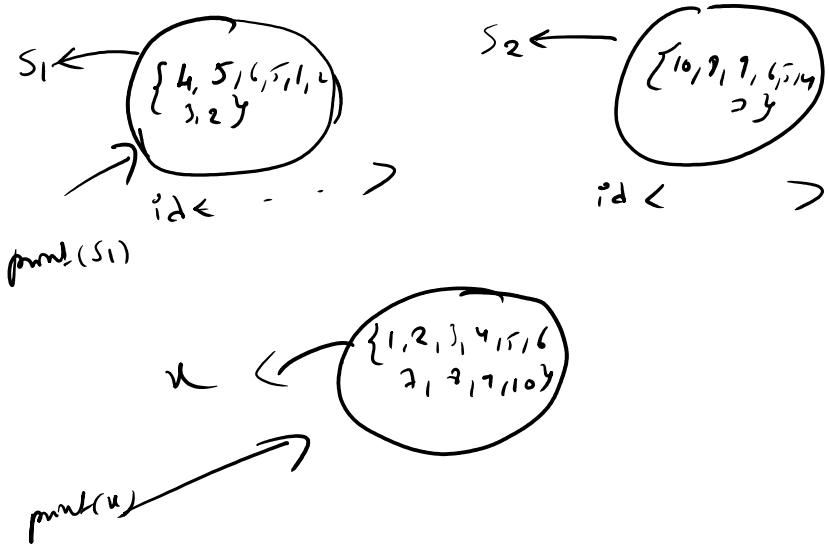




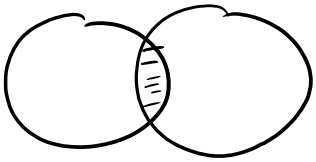


union(*others)

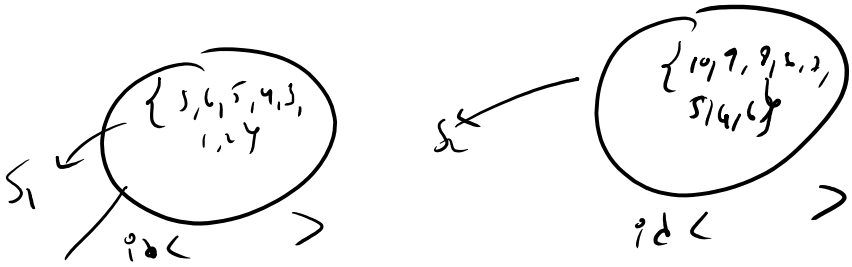
```
s1={1,2,3,4,5,6,5}
s2={4,5,6,7,8,9,10}
print(s1.union(s2))
print(s1)
```

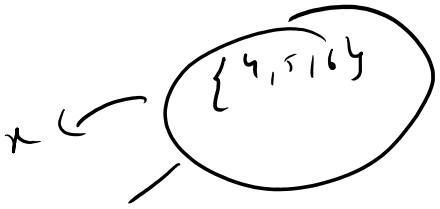


intersection(*others)

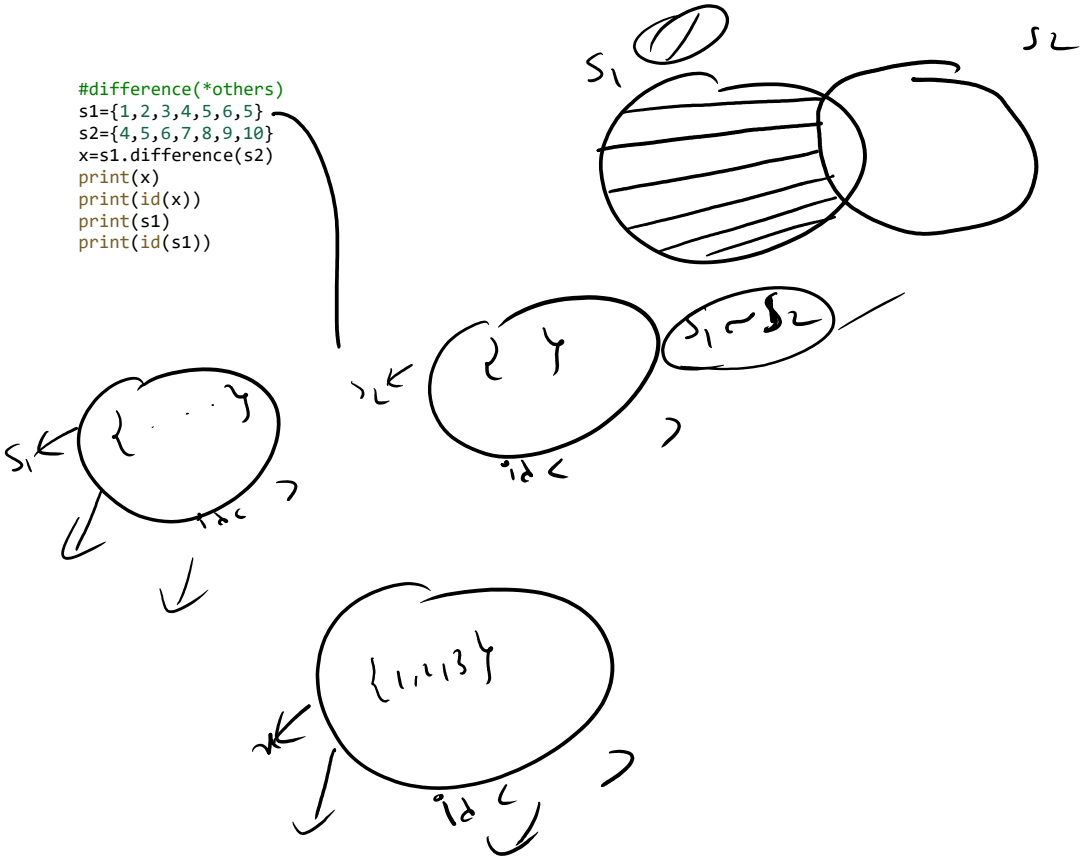



```
#intersection(*others)
s1={1,2,3,4,5,6,5}
s2={4,5,6,7,8,9,10}
x=s1.intersection(s2)
print(x)
print(id(x))
print(s1)
print(id(s1))
```

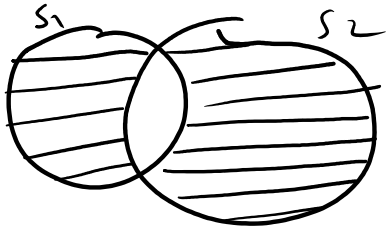




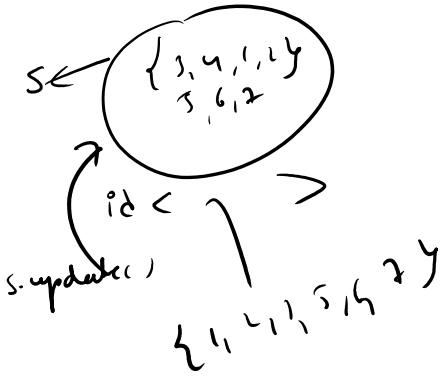
```
#difference(*others)
s1={1,2,3,4,5,6,5}
s2={4,5,6,7,8,9,10}
x=s1.difference(s2)
print(x)
print(id(x))
print(s1)
print(id(s1))
```



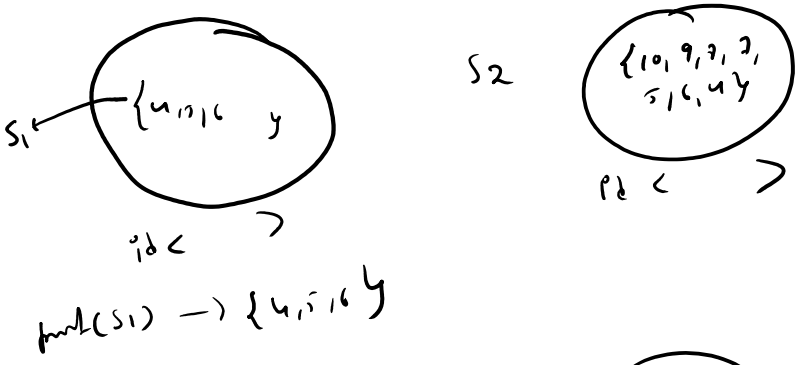
symmetric_difference(other)



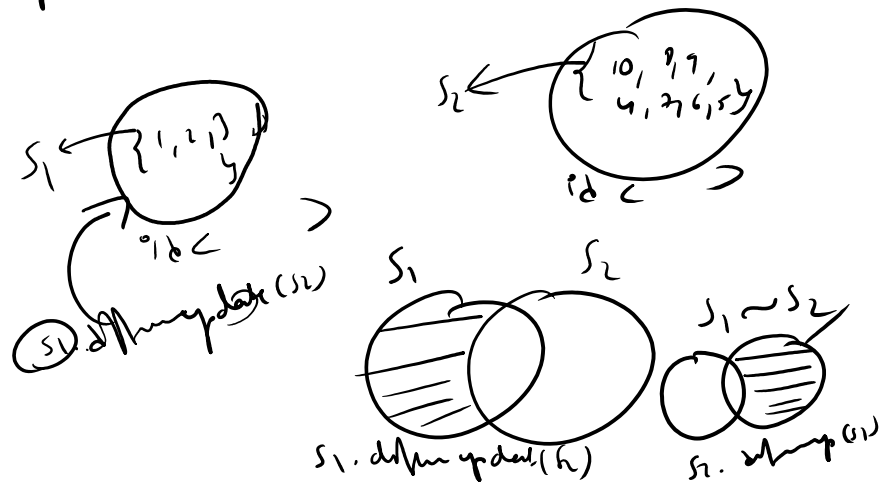
```
#symmetric_difference(*others)
s1={1,2,3,4,5,6,5}
s2={4,5,6,7,8,9,10}
x=s1.symmetric_difference(s2)
print(x)
print(id(x))
print(s1)
print(id(s1))
```



```
#update(*others)
s={1,2,3,4}
print(id(s))
s.update({5,6,6,7})
print(s)
print(id(s))
```



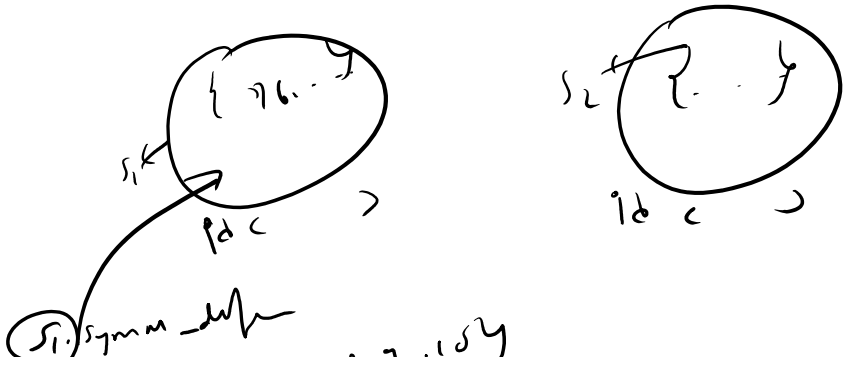
```
#intersection_update(*others)
s1={1,2,3,4,5,6,5}
s2={4,5,6,7,8,9,10}
s1.intersection_update(s2)
print(s1)
```

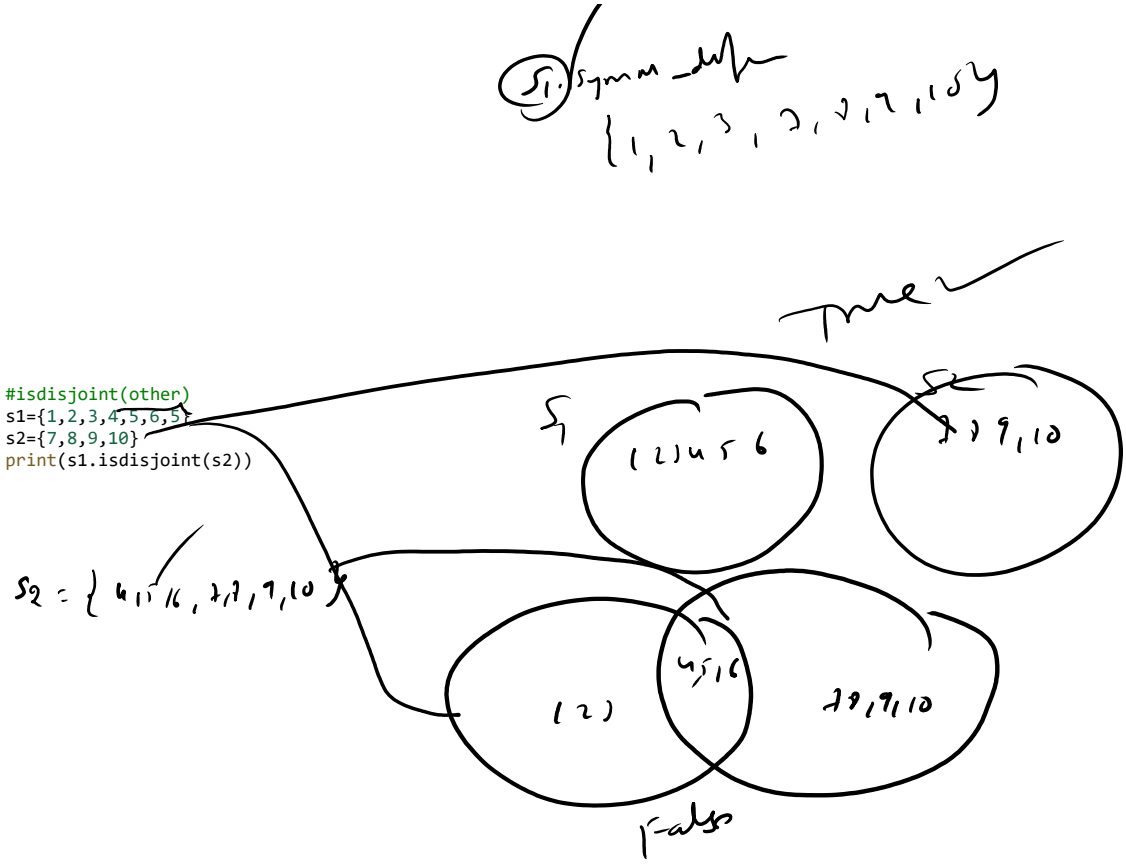


```
#difference_update(*others)
s1={1,2,3,4,5,6,5}
s2={4,5,6,7,8,9,10}
s1.difference_update(s2)
print(s1)
```

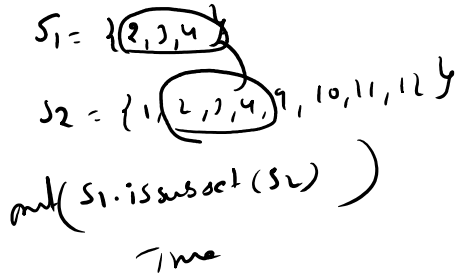


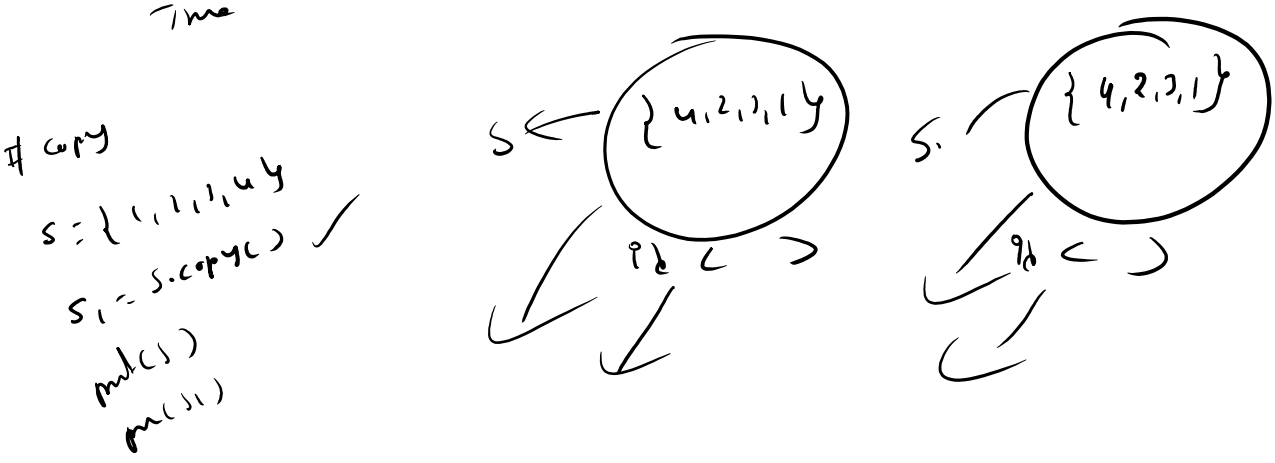
```
#symmetric_difference_update(*others)
s1={1,2,3,4,5,6,5}
s2={4,5,6,7,8,9,10}
s1.symmetric_difference_update(s2)
print(s1)
```





issubset(other)





Dictionary

23 December 2024 23:23

It is a collection of key-value pairs

Remember 4 rules:
==========

1. Dictionary is mutable
2. Dictionary keys should be immutable but values can be mutable or immutable
3. Dictionary doesn't support indexing and slicing like set
4. Dictionary keys should be unique

Creation of Dictionary:
==========

```
_ = {"Name":"data","age":25}
_x_ = {}
__123 = dict()
c__gpa =  {"Name":"data","age":25,"marks":["maths":96,"english":87]}
```

Operations:
==========
Concatenation
Indexing
Slicing
Length
in

Method	Description
dict.clear()	Removes all items from the dictionary.
dict.copy()	Returns a shallow copy of the dictionary.
dict.fromkeys(iterable, value=None)	Creates a new dictionary with keys from the iterable and values set to value.
dict.get(key, default=None)	Returns the value associated with the key if it exists; otherwise, returns the default value.
dict.items()	Returns a view object displaying a list of dictionary's key-value tuple pairs.
dict.keys()	Returns a view object displaying a list of all the keys in the dictionary.
dict.values()	Returns a view object displaying a list of all the values in the dictionary.
dict.pop(key, default=None)	Removes the specified key and returns its value. If the key is not found, returns the default value.
dict.popitem()	Removes and returns the last key-value pair in the dictionary as a tuple.
dict.setdefault(key, default=None)	Returns the value of the key. If the key does not exist, inserts the key with the specified default value.
dict.update([other])	Updates the dictionary with the key-value pairs from another dictionary or an iterable of key-value pairs.