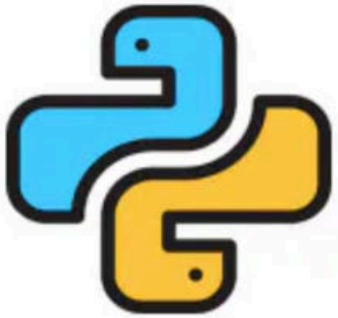
unacademy

Python as OOPs Part 1

Course on Data Structure and Algorithms Using Python

Ankush Saklecha • Lesson 7 • Dec 14, 2023



Python Programming

Python Collections (List and Set)

Content

- List
- List Indexing and Slicing
- Nested List
- List Methods
- Set
- Set Methods
- Sample Problems

List

List used to store multiple items in a single variable. It is one of 4 built-in data types in Python used to store collections of data. other three are Set, Tuple, Dictionary. It is defined as:

Syntax:

List_name=[item-1, item-2, item-3, item-4,……, item-n]

Example:

Designation=['Assistant', 'Supervisor', 'Manager', 'Technician']

List

List items are ordered, changeable or mutable, and allow duplicate data items.

Ordered:

```
Age=[20,24,17,26,22,28,32,25,36,31]
print(Age)
```

Age[2]=35

Mutable:

0 1 2

```
Age=[20,24,17,26,22,28,32,25,36,31]
Age.append(45)
print(Age)
```

Duplicate:

```
Age=[20,24,17,26,22,28,32,25,36,31,32,42]
print(Age)
```

List Indexing & Slicing

- Like a string, we can also define indexes to data items of list.
- Python support two types of indexing:

  --Positive Indexing

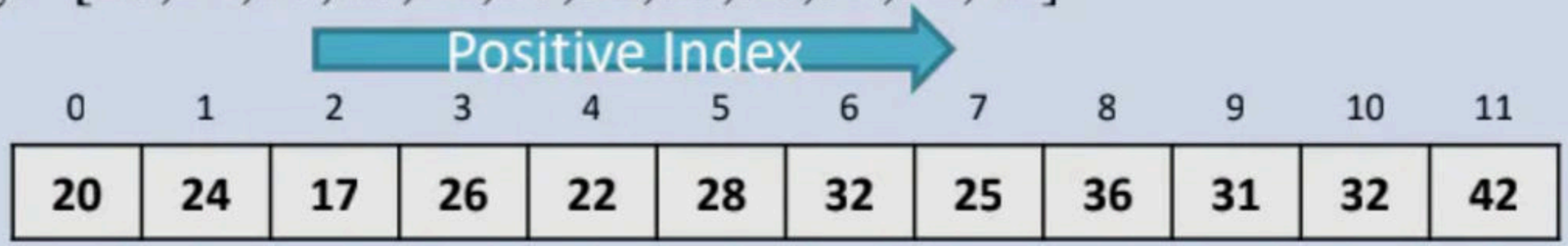
  --Negative Indexing

Positive Indexing

- It begin with zero and end with list length-1 from left hand side.

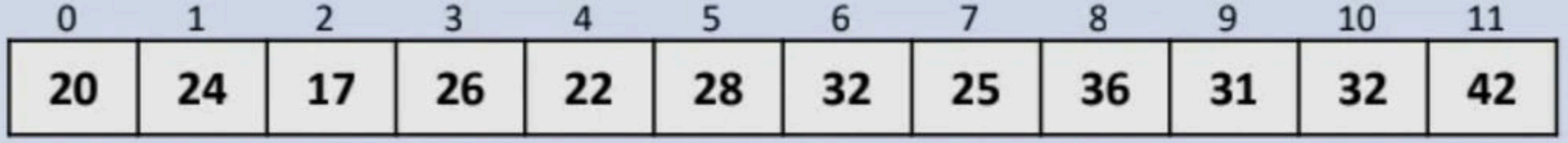
List Positive Indexing Examples:

Age=[20,24,17,26,22,28,32,25,36,31,32,42]



Access List Data Items via Index:

Age=[20,24,17,26,22,28,32,25,36,31,32,42]




```
print(Age[1])
```

Let's Try…

Guess Output?

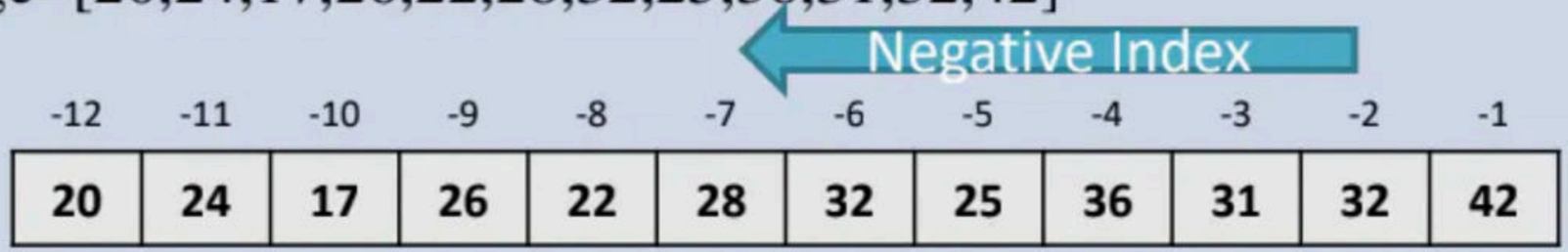
```
Age=[20,24,17,26,22,28,32,25,36,31,32,42]

print(Age[2*3])

print(Age[10//3])

print(Age[10%3])

print(Age[2**2])

print(Age[8>>2])

print(Age[2&4])

print(Age[6^3])
```

Negative Indexing

- It begin with -1 and end with list length in negative from right hand

String Negative Indexing Examples:

Age=[20,24,17,26,22,28,32,25,36,31,32,42]



Access String Character via Negative Index:

Age=[20,24,17,26,22,28,32,25,36,31,32,42]


```
print(Age[-8])
```

Let's Try…

Guess Output?

```
Age=[20,24,17,26,22,28,32,25,36,31,32,42]

print(Age[-5])

print(Age[-6-2])

print(Age[-9+11])

print(Age[14-10])

print(Age[-8-3])

print(Age[-4+4])

print(Age[-12+6])
```

List Slicing

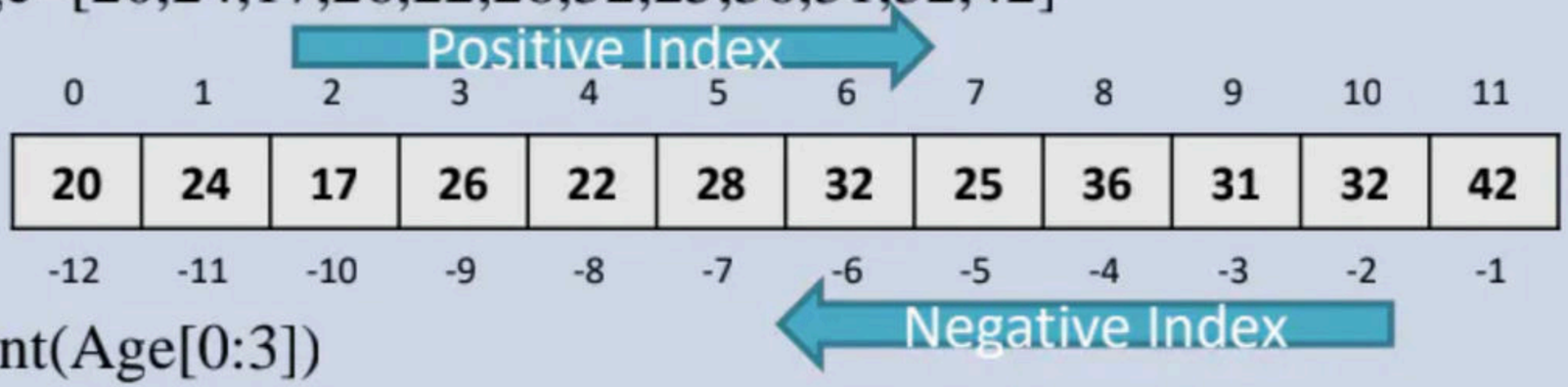
- It is way to access set of data items from the given list. It is denotes as:

  list_name[startindex : endindex]

- It slice list from startindex and end with endindex-1 i.e. it excludes endindex value. Slice index may be positive or negative range.

String Slicing Examples:

Age=[20,24,17,26,22,28,32,25,36,31,32,42]




```
print(Age[0:3])

print(Age[-11:-7])
```

Let's Try…

Guess Output?

```
Age=[20,24,17,26,22,28,32,25,36,31,32,42]
print(Age[1+1:4+0])

print(Age[0:2**3])

print(Age[-12:-7+3])

print(Age[-1:-7)

print(Age[:10])

print(Age[:-6])

print(Age[2:-2])

print(Age[-8:14])
```

Change List Data Items

Example:

```
EmpName=['Jhon','Bob','Marry','Cherry']
EmpName[1]='Romy'
print(EmpName)
EmpName[0:2]=['Romy','Jack']
print(EmpName)
```

Insert New Data Items in List

Example:

```
EmpName=['Jhon','Bob','Marry','Cherry']

EmpName.insert(0,'Jam')

print(EmpName)
```

Append New Data Items at End of the List

Example:

```
EmpName=['Jhon','Bob','Marry','Cherry']
EmpName.append('Jam')
print(EmpName)
```

Remove Specified Data Items from the List

Example:

```
EmpName=['Jhon','Bob','Marry','Cherry']
EmpName.remove('Bob')
print(EmpName)
EmpName.pop(0)
print(EmpName)
del EmpName[1]
print(EmpName)
del EmpName
print(EmpName)
EmpName.clear()
print(EmpName)
```

Sort

- sort() is a method of list class and can only be used with lists.

- ***Syntax:*** *List_name.sort(key, reverse=False)*
  Parameters:
  key: *A function that serves as a key for the sort comparison.*
  reverse: *If true, the list is sorted in descending order.*
  Return type: *None*

Sort List Data Items

Example:

```
EmpName=['Jhon','Bob','Marry','Cherry']
EmpName.Sort()                          # Ascending Order
print(EmpName)
EmpName.Sort(reverse=True)                          # Descending Order
print(EmpName)
```

```
def myFunc(e):
  return len(e)

cars = ['Tata', 'creta', 'BMW', 'Ford']

cars.sort(key=myFunc)
print(cars)
```

```
 def myFunc(e):
   return len(e)

 cars = ['Tata', 'creta', 'BMW', 'Ford']

 cars.sort(key=myFunc, reverse = True)
 print(cars)
```

Copy Lists

Example:

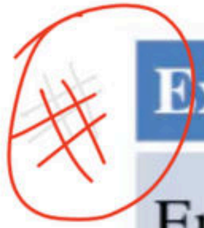
```
EmpName=['Jhon','Bob','Marry','Cherry']
Name=EmpName.copy()                    # Copy method
print(Name)
Name=list(EmpName)                     # List Method
print(Name)
```

Join Lists

Example:

```
EmpName=["Jhon","Bob","Marry","Cherry"]
Age=[24,22,28,19]
NameAge=EmpName+Age                    #  + Operator
print(NameAge)
for x in Age:
    EmpName.append(x)
print(EmpName)
EmpName.extend(Age)                    #  Extend() Method
print(EmpName)
```

List Comprehension



Example:

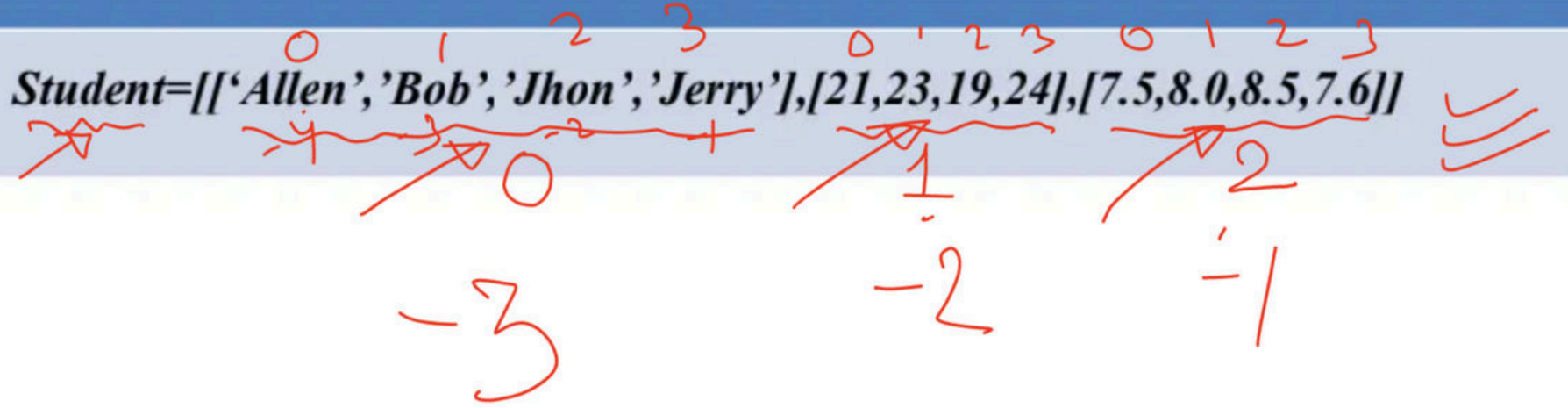
```
EmpName=['Jhon','Bob','Marry','Cherry']

newlist = [x for x in EmpName ]

print(EmpName)
```

Nested Lists

Nested Lists are used to store multiple sublists in a single list..\

It is defined as:

Syntax:
List_name=[[item-1, item-2], [item-3, item-4],[……, item-n]]
Example:
Student=[['Allen','Bob','Jhon','Jerry'],[21,23,19,24],[7.5,8.0,8.5,7.6]]



Nested Lists Indexing & Slicing

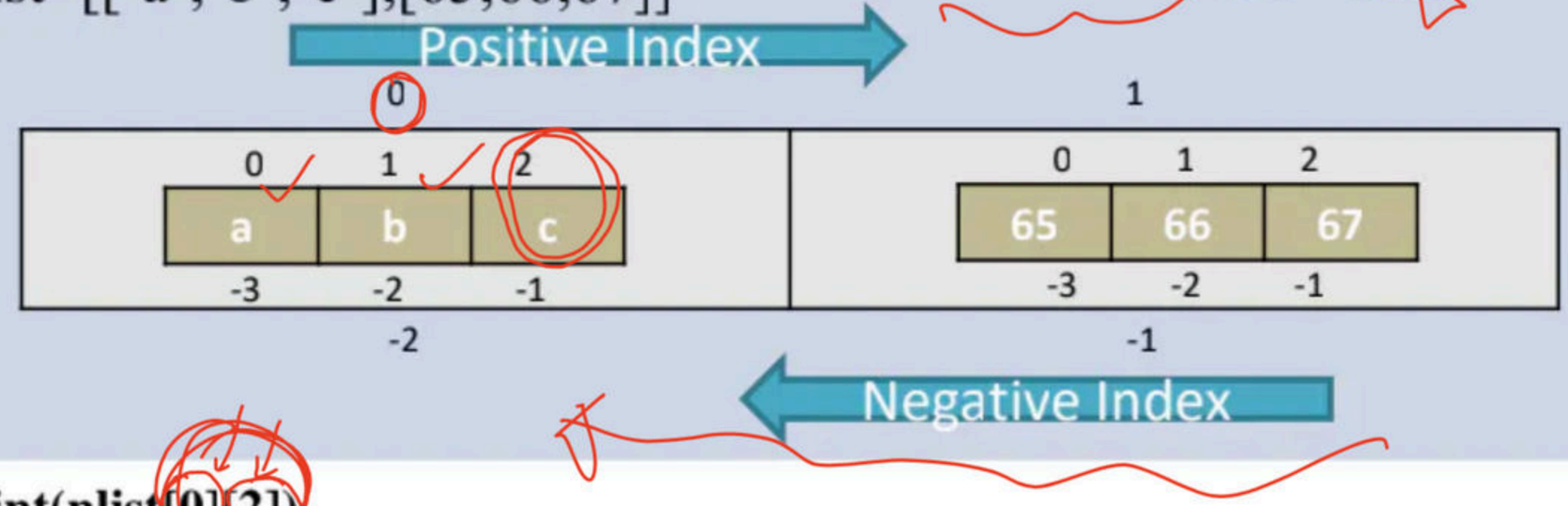
- It is way to access set of data items from the given list. It is denotes as:

list_name[[startindex : endindex],[],[]]

- It slice list from startindex and end with endindex-1 i.e. it excludes endindex value. Slice index may be positive or negative range.

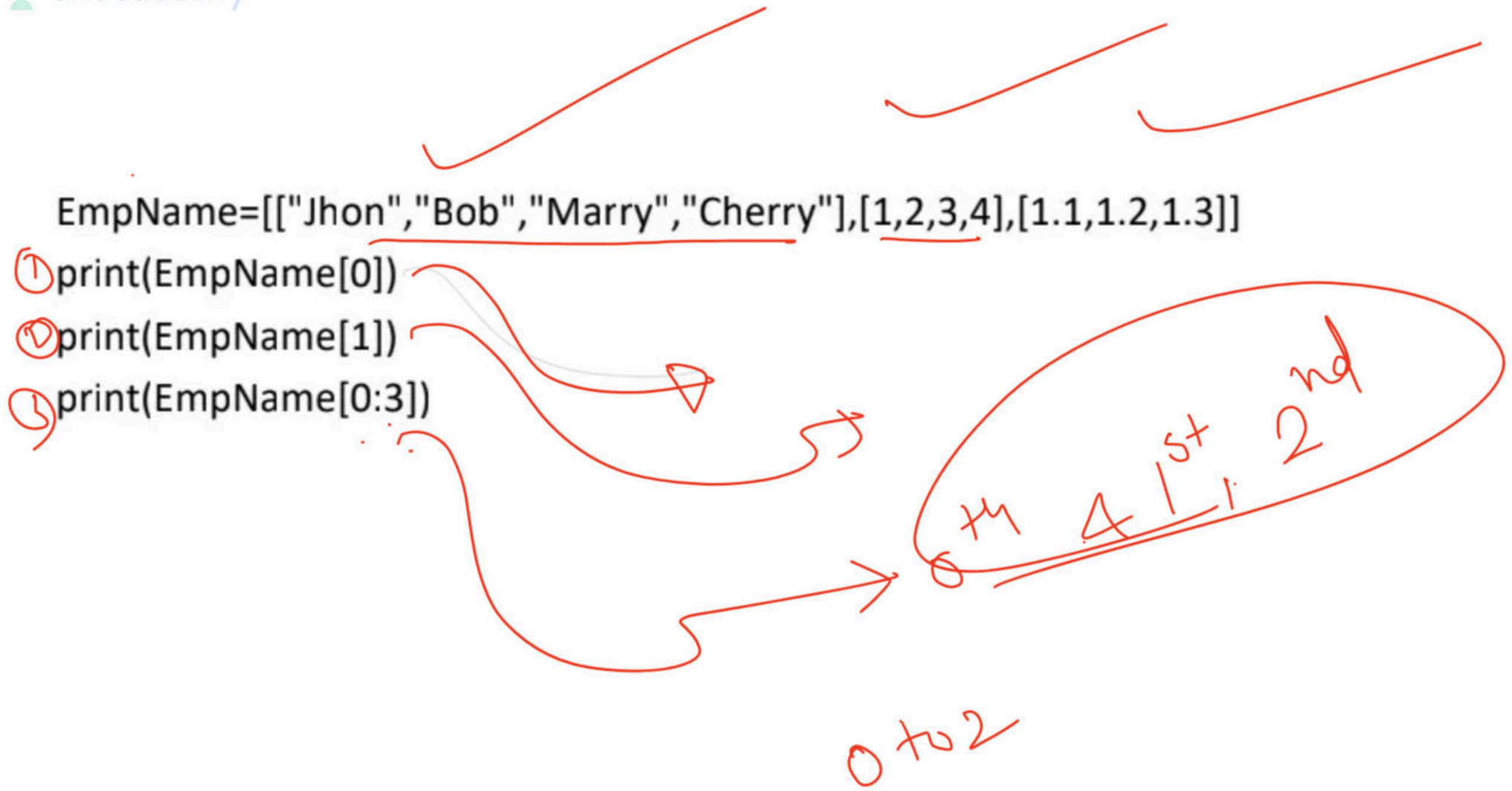
String Slicing Examples:

nlist=[['a','b','c'],[65,66,67]]

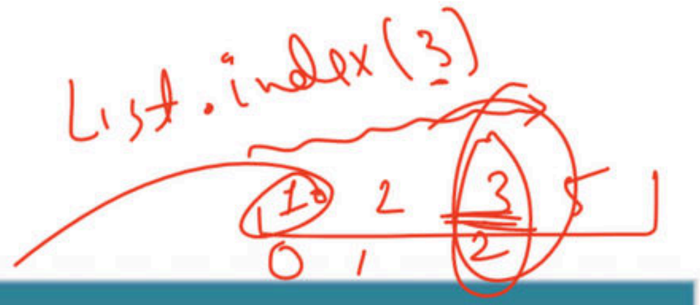


print(nlist[0][2])

```
EmpName=[["Jhon","Bob","Marry","Cherry"],[1,2,3,4],[1.1,1.2,1.3]]
print(EmpName[0])
print(EmpName[1])
print(EmpName[0:3])
```



List Methods



In python, several built-in methods are available.

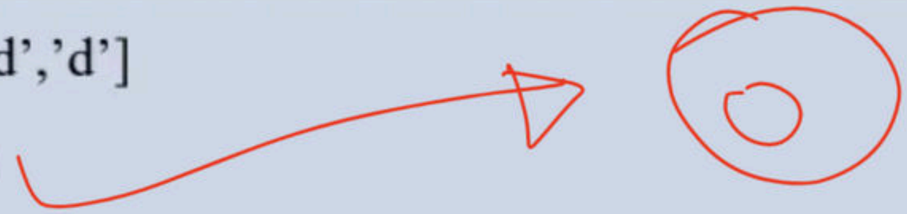
Method Name	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

pop(0)

List Methods

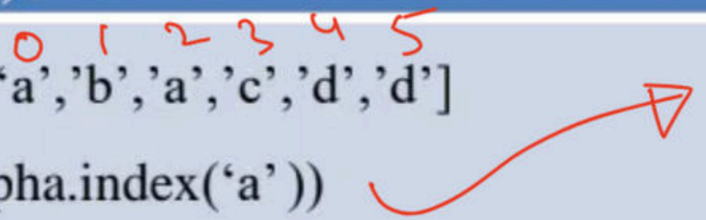
count( ):

```
Alpha=['a','b','a','c','d','d']
print(Alpha.count('e'))
```



index( ):

```
Alpha=['a','b','a','c','d','d']
print(Alpha.index('a' ))
```

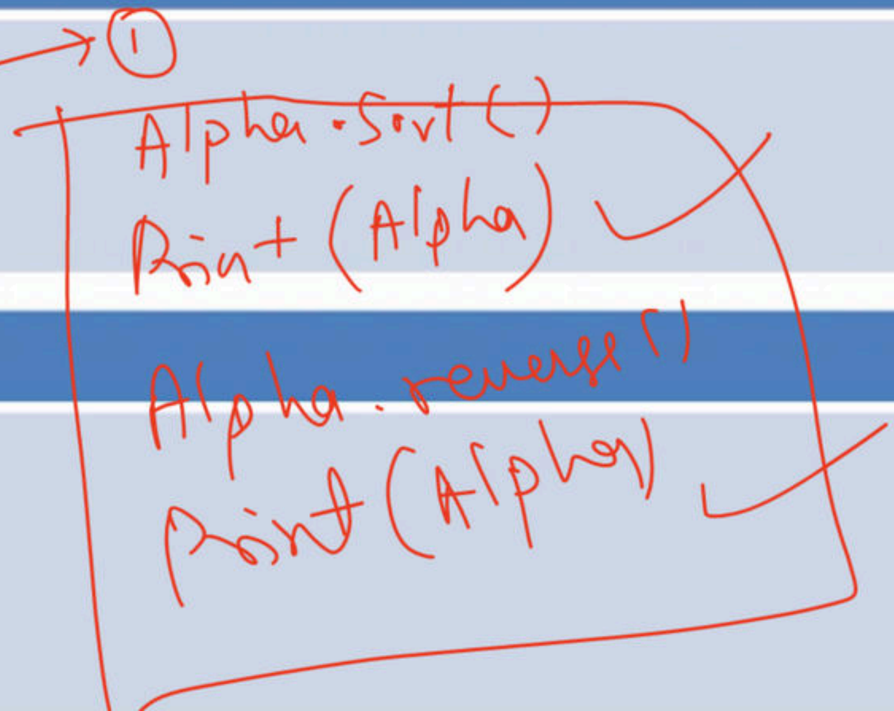


sort & reverse( ):

```
Alpha=['a','b','a','c','d','d']
print(Alpha.sort( ))
print(Alpha.reverse( ))
```

None

None



Set

Set is also used to store multiple items in a single variable. It is defined as:

Syntax:
Set_name={item-1, item-2, item-3, item-4,……, item-n}
Example:
Designation={'Assistant', 'Supervisor', 'Manager', 'Technician'}

Set

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered:

```
Age={20,24,17,26,22,28,32,25,36,31}
print(Age)
```

Immutable or Unchangeable:

```
Age={20,24,17,26,22,28,32,25,36,31}
Age[2]=35
print(Age)
```

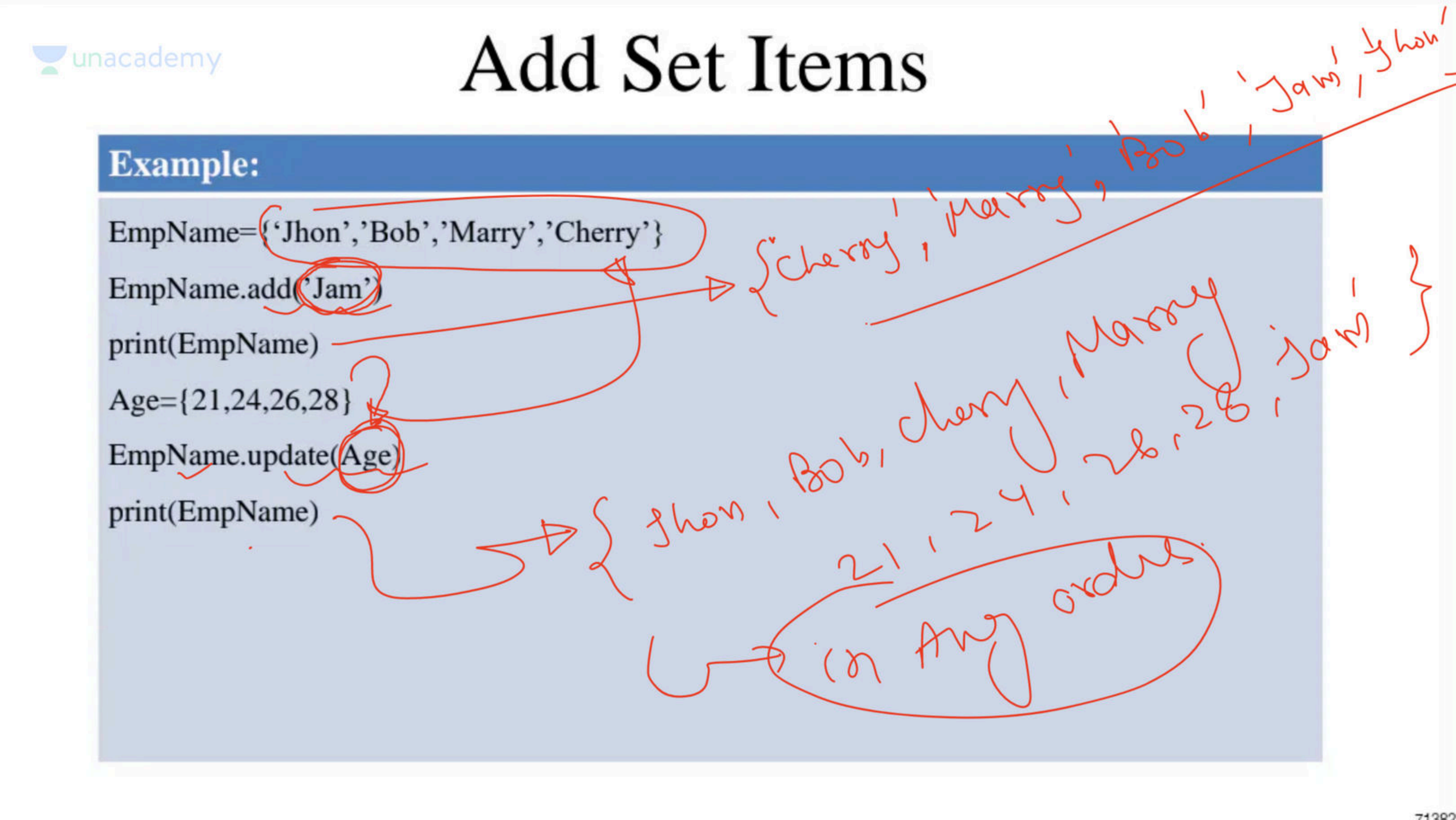
Duplicate Not Allowed:

```
Age={20,24,17,26,22,28,32,25,36,31,32,42}
print(Age)
```

Add Set Items

Example:

```
EmpName={'Jhon','Bob','Marry','Cherry'}
EmpName.add('Jam')
print(EmpName)
Age={21,24,26,28}
EmpName.update(Age)
print(EmpName)
```



Remove Set Items

Example:

```
EmpName={'Jhon','Bob','Marry','Cherry'}
EmpName.remove('Bob')          #remove()  method that cause errors if items not found
print(EmpName)

EmpName.discard('Jam')         #discard()  method does not cause errors if items not found
print(EmpName)

print(EmpName.pop())           #pop() method remove random element from the set

print(EmpName.clear())         # clear() method remove all elements from the set

print(EmpName)                 # del keyword will delete set automatically
```

Join Sets

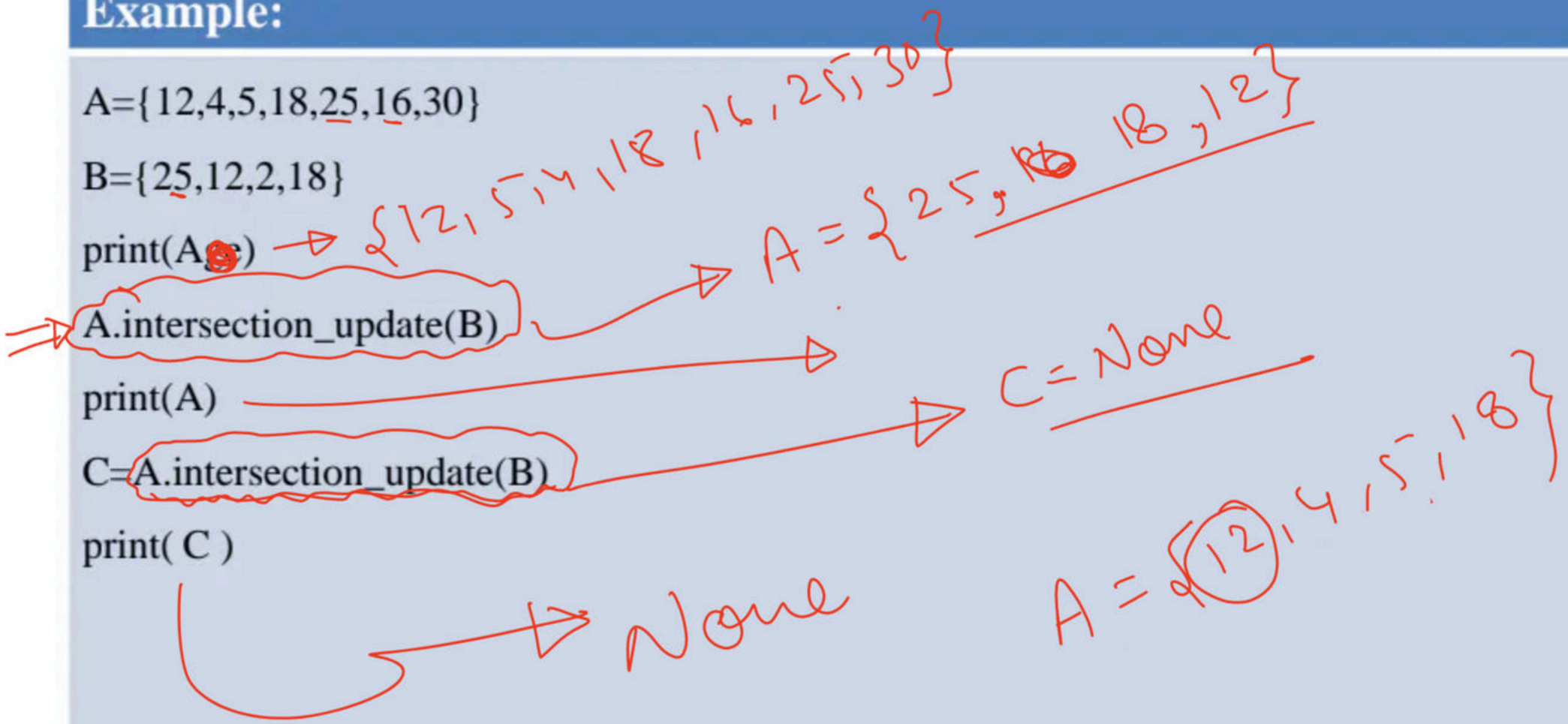
Example:

```
EmpName={'Jhon','Bob','Marry','Cherry'}
print(EmpName)
Age={21,24,26,28}
print(Age)
EmpAge=EmpName.union(Age)
print(EmpAge)
```

Keep Duplicates

Example:

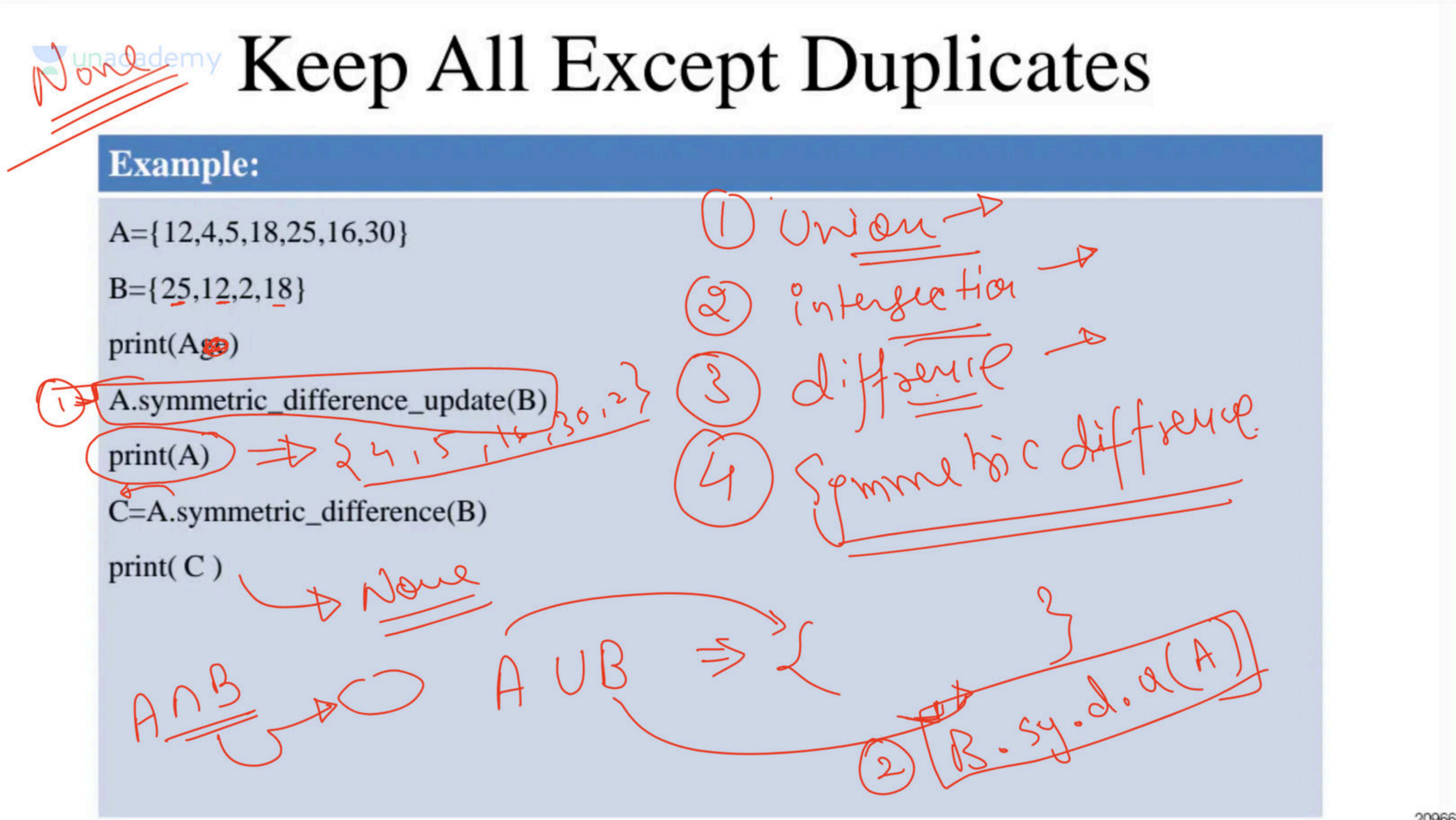
```
A={12,4,5,18,25,16,30}
B={25,12,2,18}
print(A)
A.intersection_update(B)
print(A)
C=A.intersection_update(B)
print( C )
```

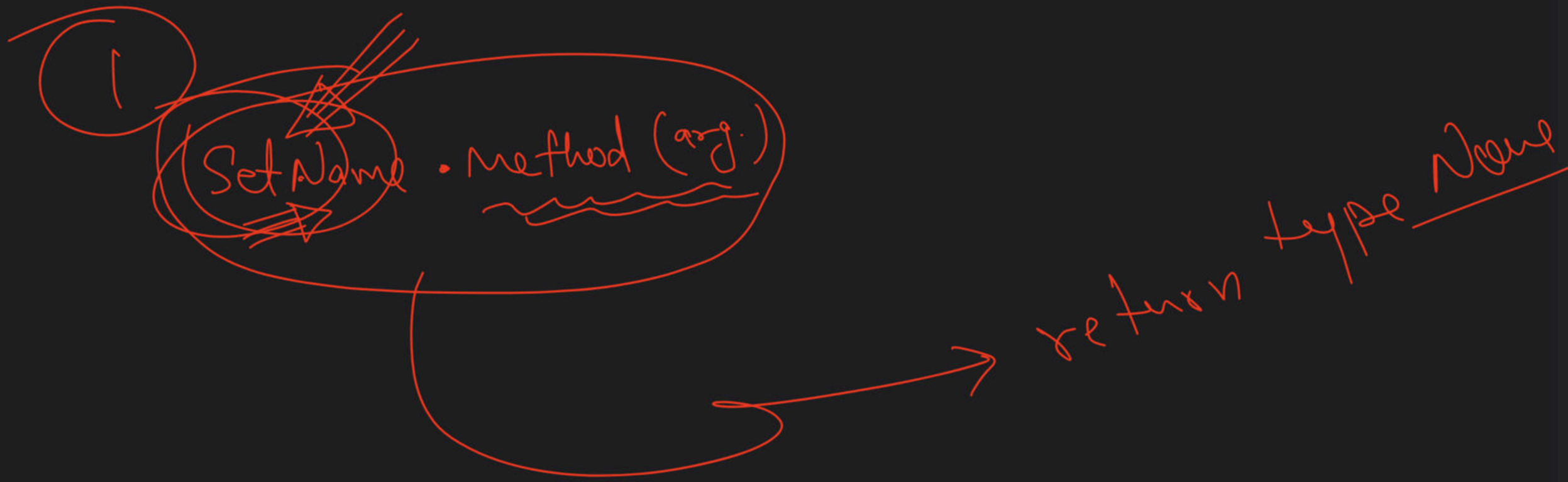


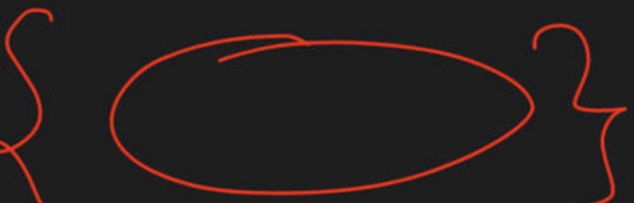
Keep All Except Duplicates

Example:

```
A={12,4,5,18,25,16,30}

B={25,12,2,18}

print(Age)

A.symmetric_difference_update(B)

print(A)

C=A.symmetric_difference(B)

print( C )
```





$A - B \Rightarrow \{$  $\}$

$B - A \Rightarrow \{ \quad \}$

$A - B \neq B - A$

$A = \{1, 2, 3, 4\}$

$B = \{3, 4, 5, 6\}$

$A - B = \{1, 2\}$

$B - A = \{5, 6\}$

unacademy

Symmetric difference $\Rightarrow$

$A = \{1, 2, 3, 4\}$

$B = \{3, 4, 5, 6\}$

$\Rightarrow \{1, 2, 3, 4, 5, 6\}$

$A \Delta B \Rightarrow \{1, 2, 5, 6\}$

$a \cup b$

$a \cap b$

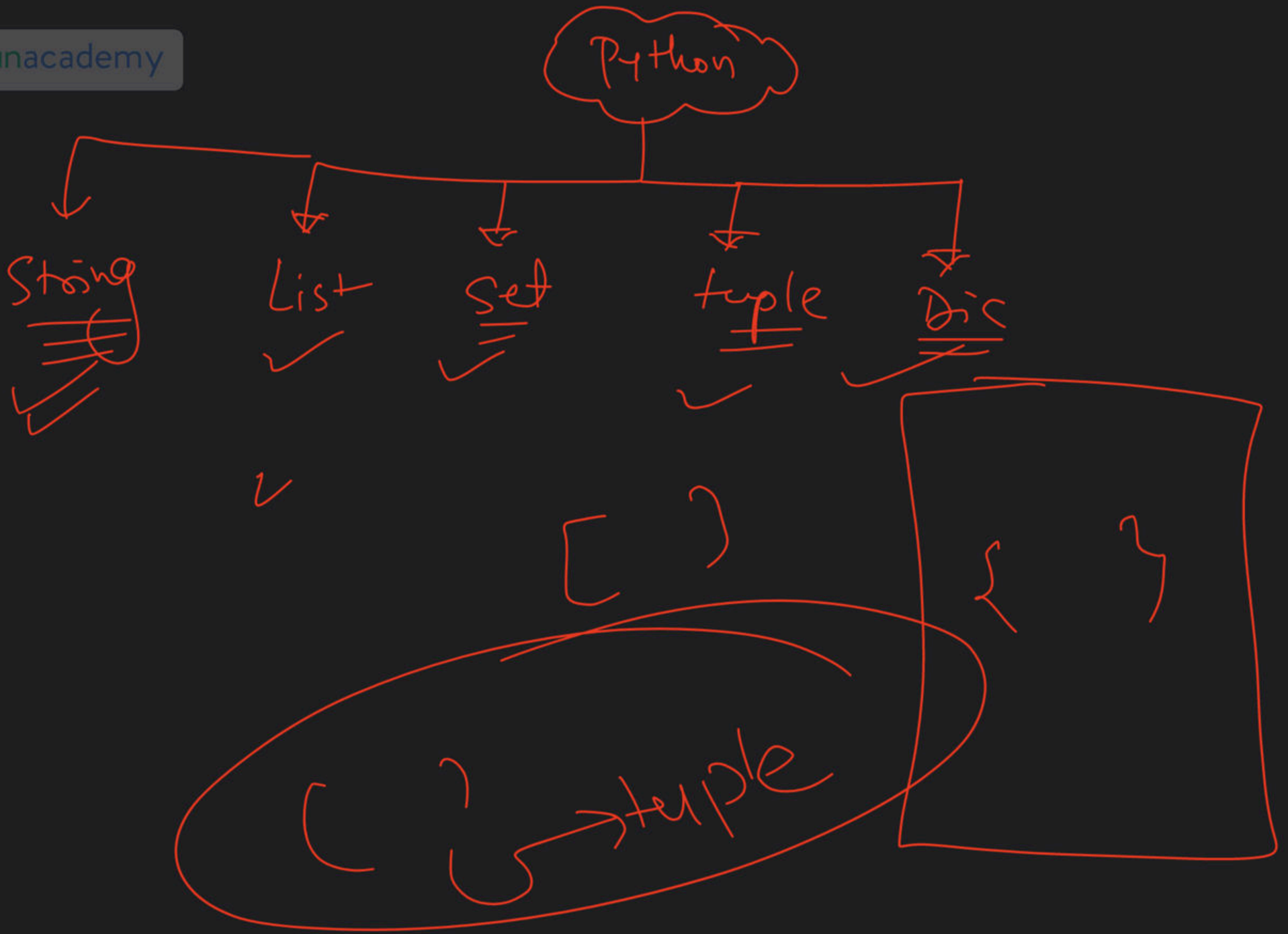
$a - b$

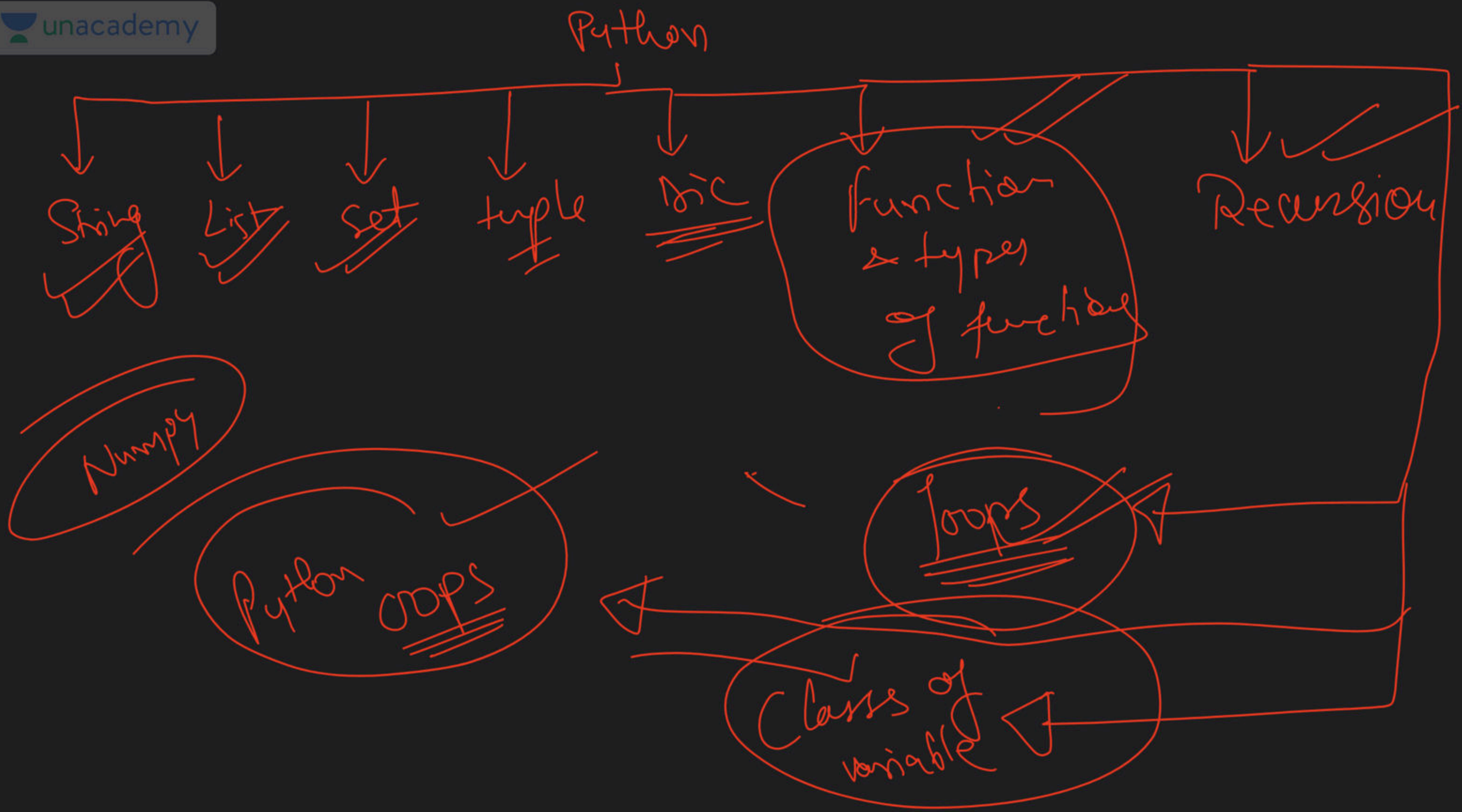
$A \Delta B$

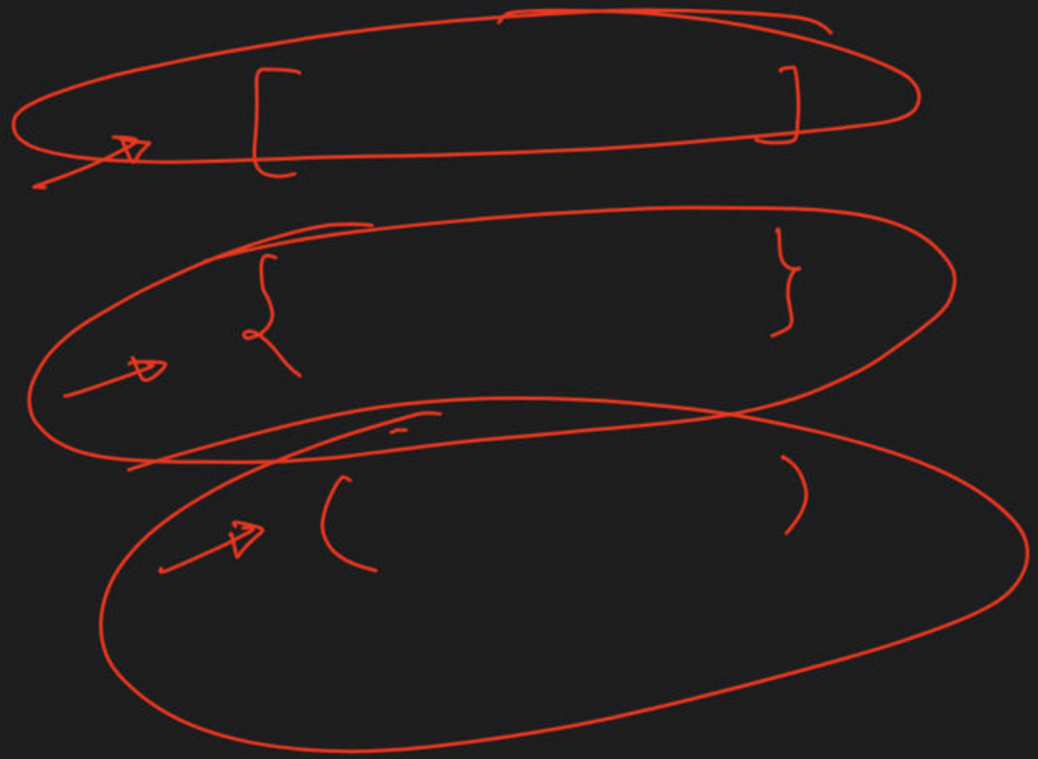
Set Methods

In python, several built-in methods are available.

Method Name	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not




Python
String
List
set
tuple
Dic
function & types of functions
Recursion
Numpy
Python OOPS
loops
Class of variable



Set Methods

In python, several built-in methods are available.

Method Name	Description
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Set Methods

isdisjoint( ):

```
A={12,4,5,18,25,16,30}
B={25,12,2,18}
z = A.isdisjoint(B)
print(z)
```

false

A B → {4,5,6}

↳ {1,2,3} ∩ = ∅

issubset( ):

```
A={12,4,5,18,25,16,30}
B={25,12,2,18}
z = A.isdisjoint(B)
print(z)
```

false

z = B.isdisjoint(A)

A = {1,2,3}

B = {3,4,5}

issuperset( ):

```
A={12,4,5,18,25,16,30}
B={25,12,2,18}
z = A.issuperset(B)
print(z)
```

false

A ⊉ B

B ⊉ A

A = {1, 2, 3, 4}

B = {2, 3, 4, 5, 1, 6}

Z = A.issuperset(B)

① Print(z) → False

Z = B.issuperset(A)

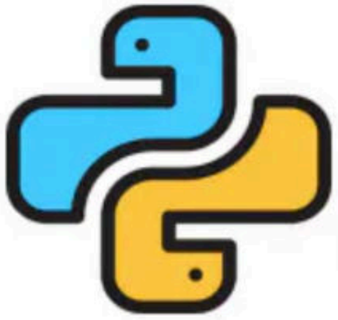
② Print(z) True

③ Z = A.issubset(B)

Print(z) True

④ Z = B.issubset(A)

Print(z) False



Python Programming

Python Collections (Tuple and Dictionary)

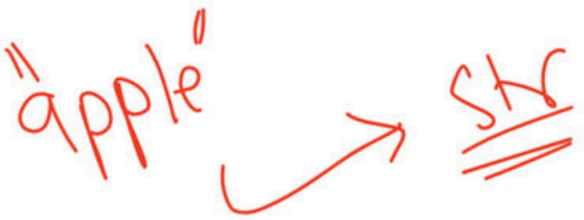
12-09-2023

- Tuples are used to store multiple items in a single variable.
- Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.
- A tuple is a collection which is ordered and **unchangeable**.
- Tuple items are ordered, unchangeable, and allow duplicate values.
- Tuples are written with round brackets.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

To determine how many items a tuple has, use the `len()` function:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```



➢To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
thistuple = ("apple",)
print(type(thistuple))
```


```
#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

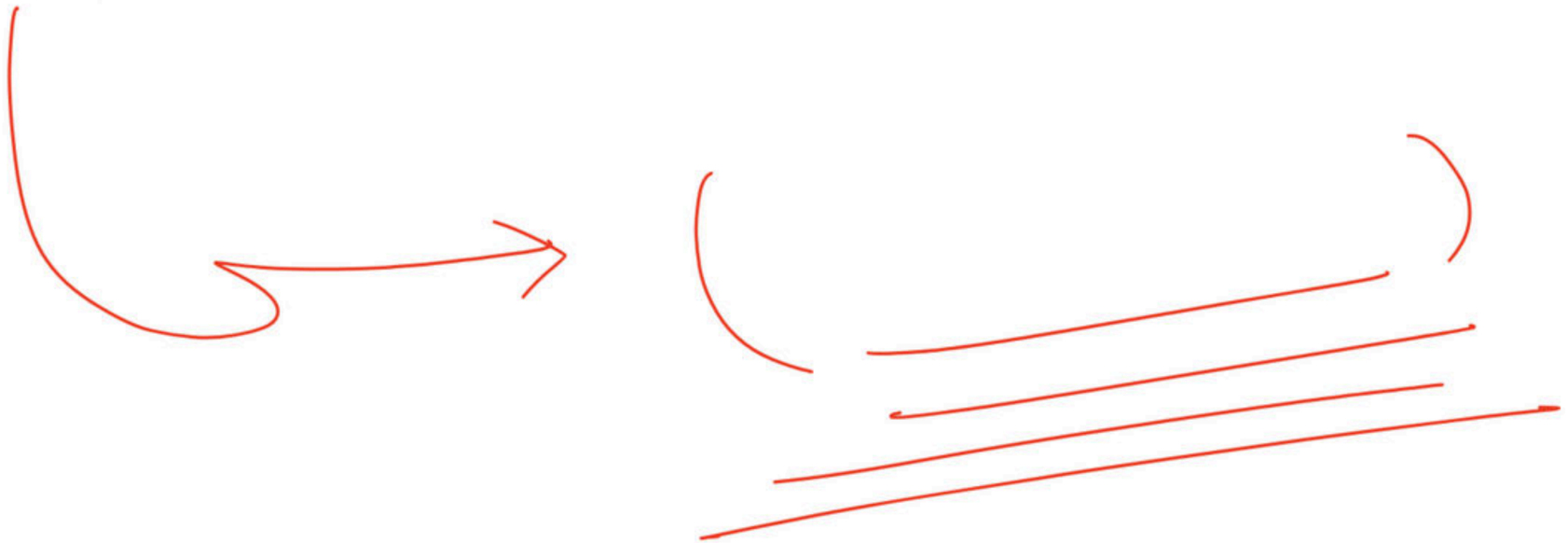
```
tuple1 = ("apple", “mango", "cherry")
tuple2 = (1, 2, 3,4, 5)
tuple3 = (True, False, False)
tuple4 = (“xyz", 34, True, 40, “female")
```

From Python's perspective, tuples are defined as objects with the data type 'tuple':

```
print(type(tuple1))
print(type(tuple2))
print(type(tuple3))
print(type(tuple4))
```

tuple() Constructor

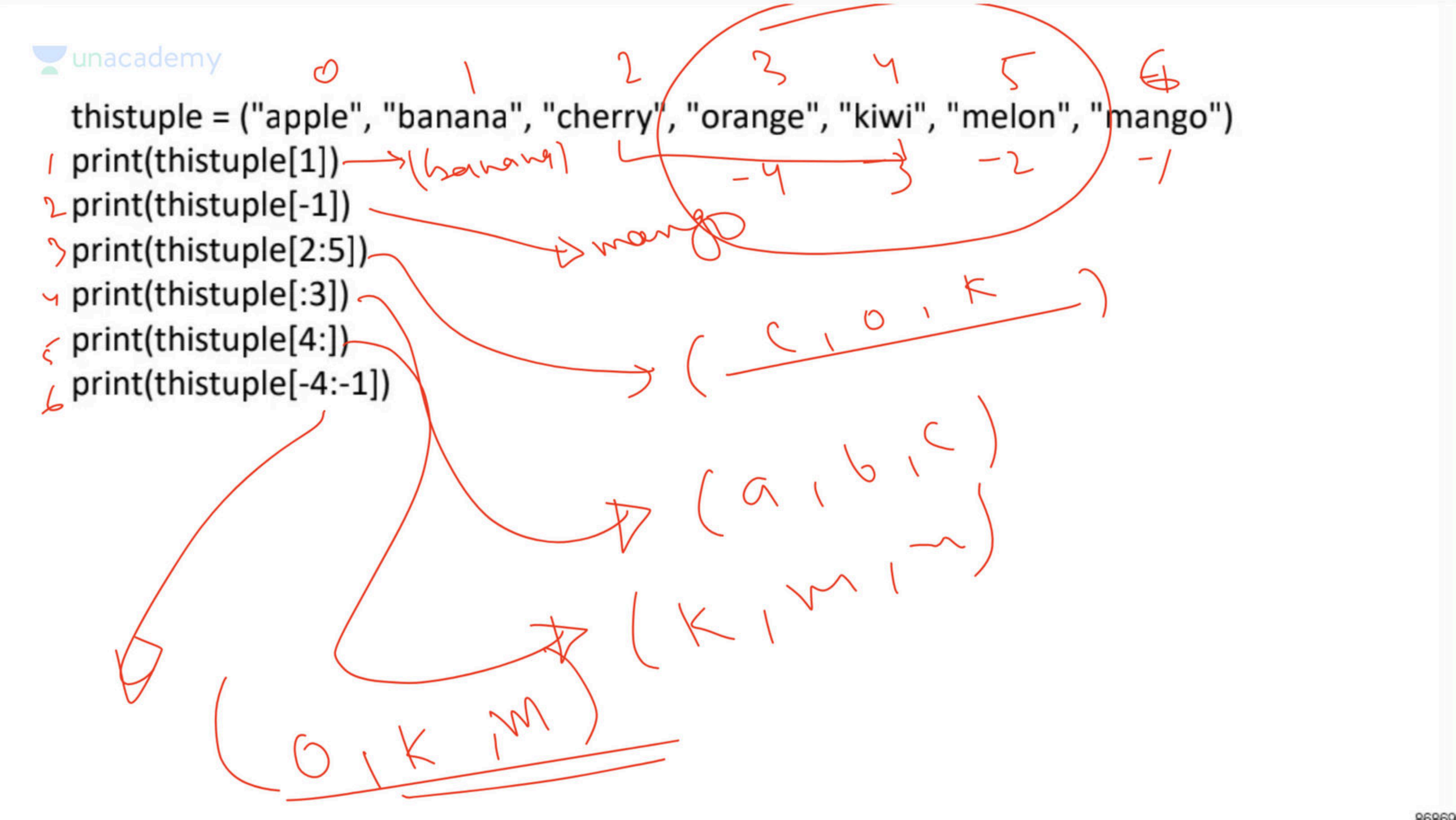
```
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
print(thistuple)
```



t = (. . . . .)

t = tuple (( ))

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[1])
print(thistuple[-1])
print(thistuple[2:5])
print(thistuple[:3])
print(thistuple[4:])
print(thistuple[-4:-1])
```



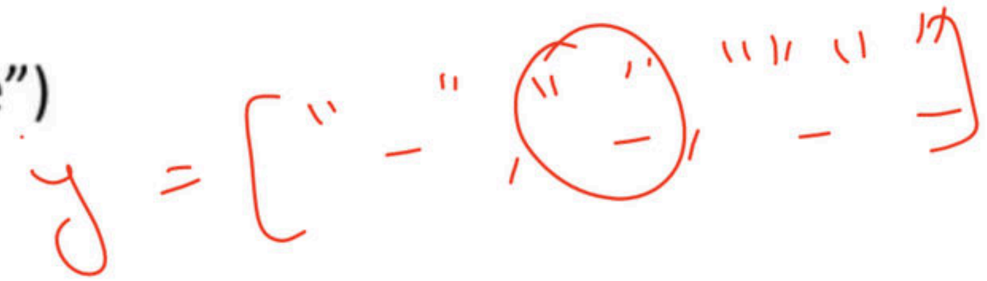
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

thistuple[1] = "x42"

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.
But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
x =  ("apple", "banana", "cherry", "orange")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```



Add Items

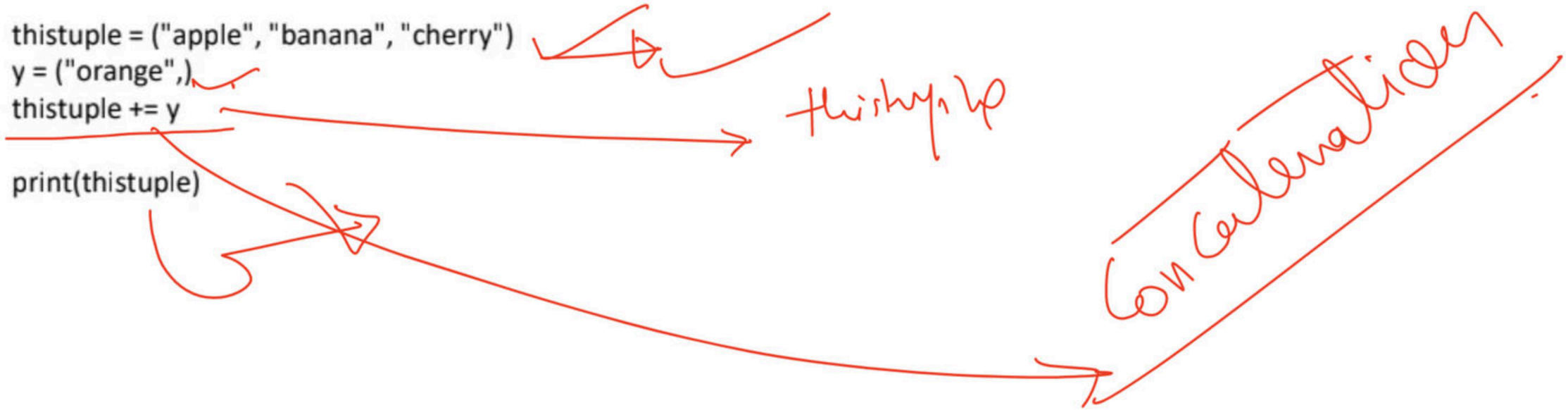
Since tuples are immutable, they do not have a built-in `append()` method, but there are other ways to add items to a tuple.

1. **Convert into a list**: Just like the workaround for *changing* a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
```

2. **Add tuple to a tuple**. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```



class List

List.append()

List()

Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

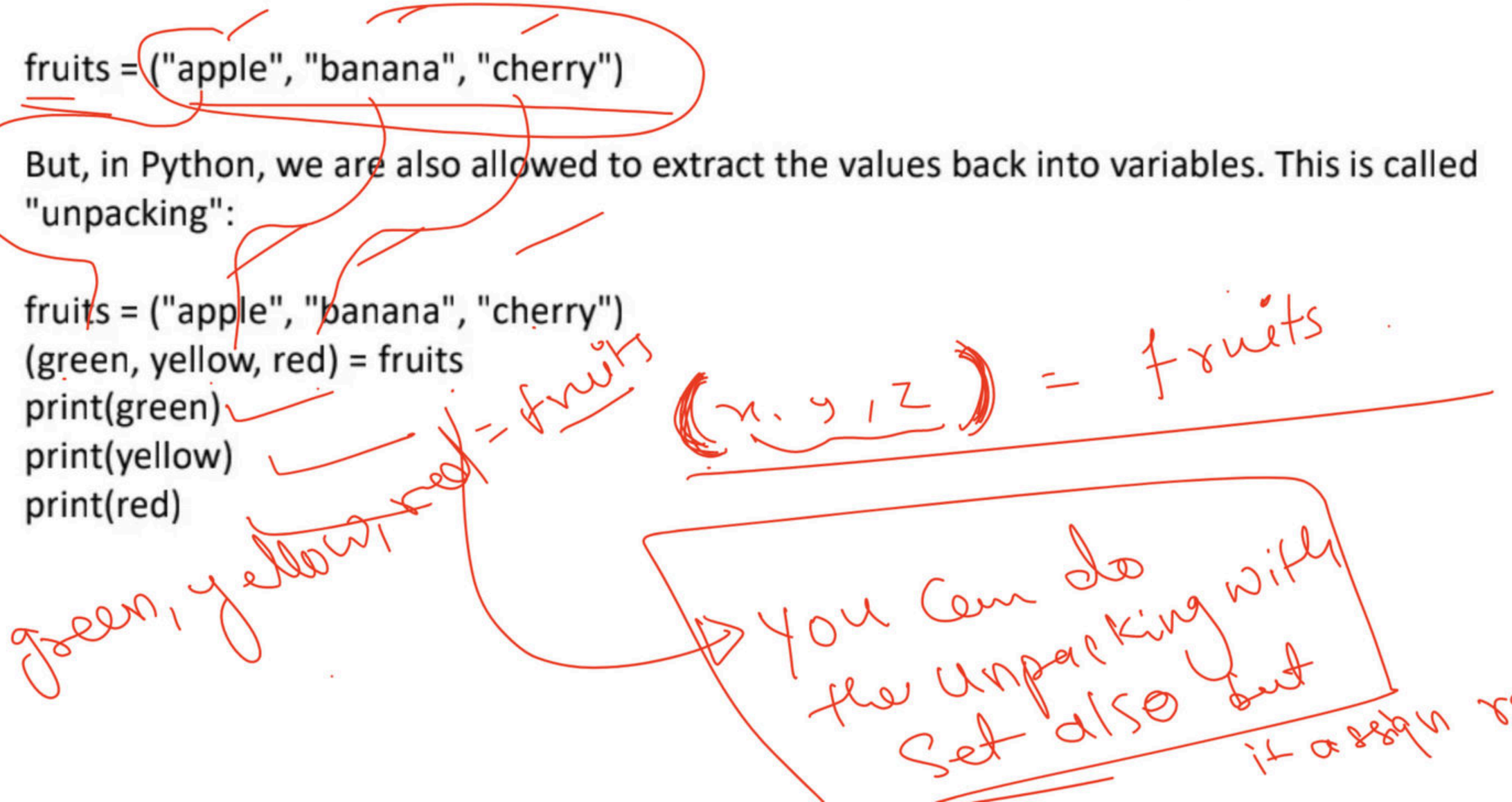
Unpacking a Tuple

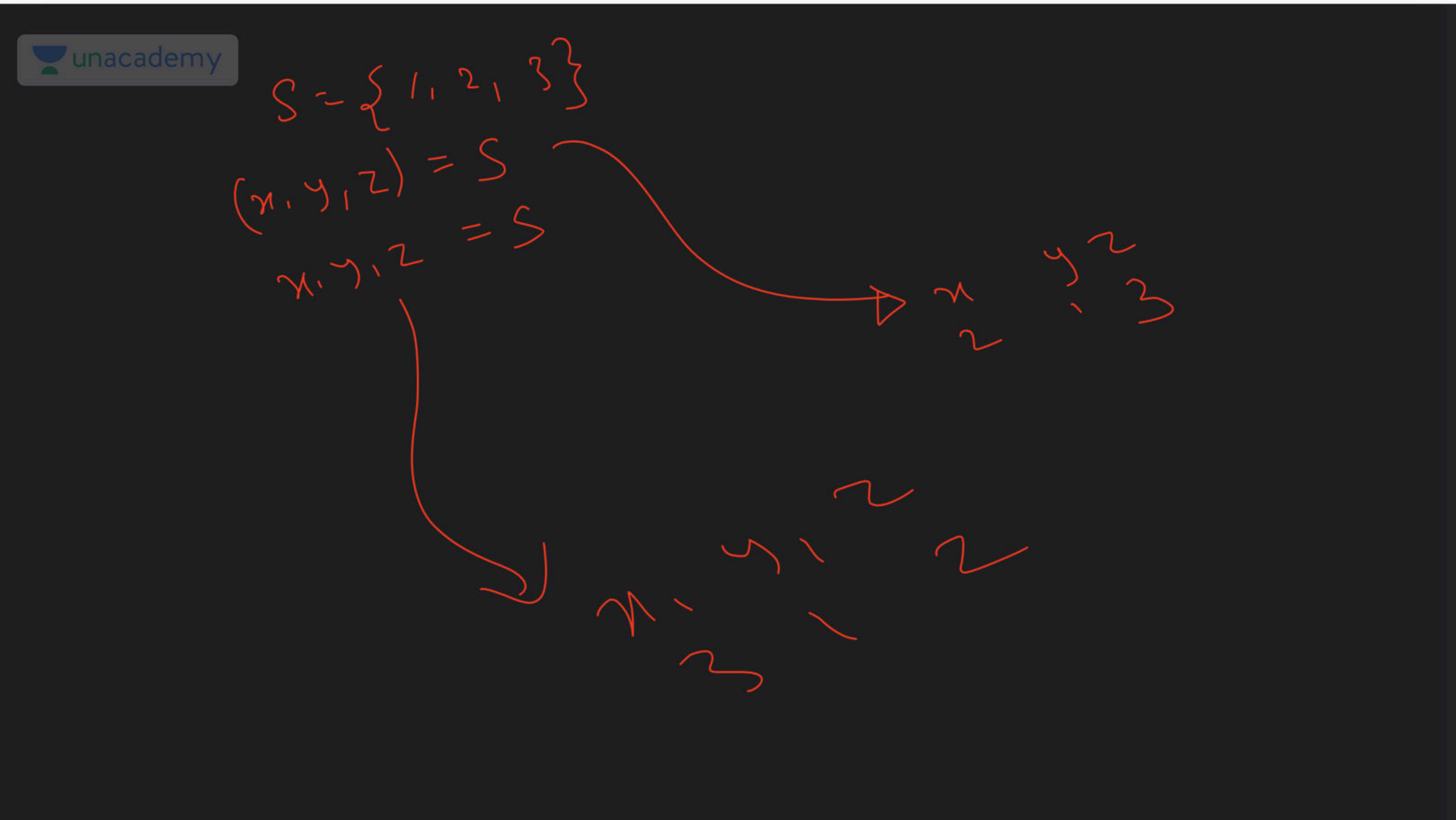
When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

```
fruits = ("apple", "banana", "cherry")
```

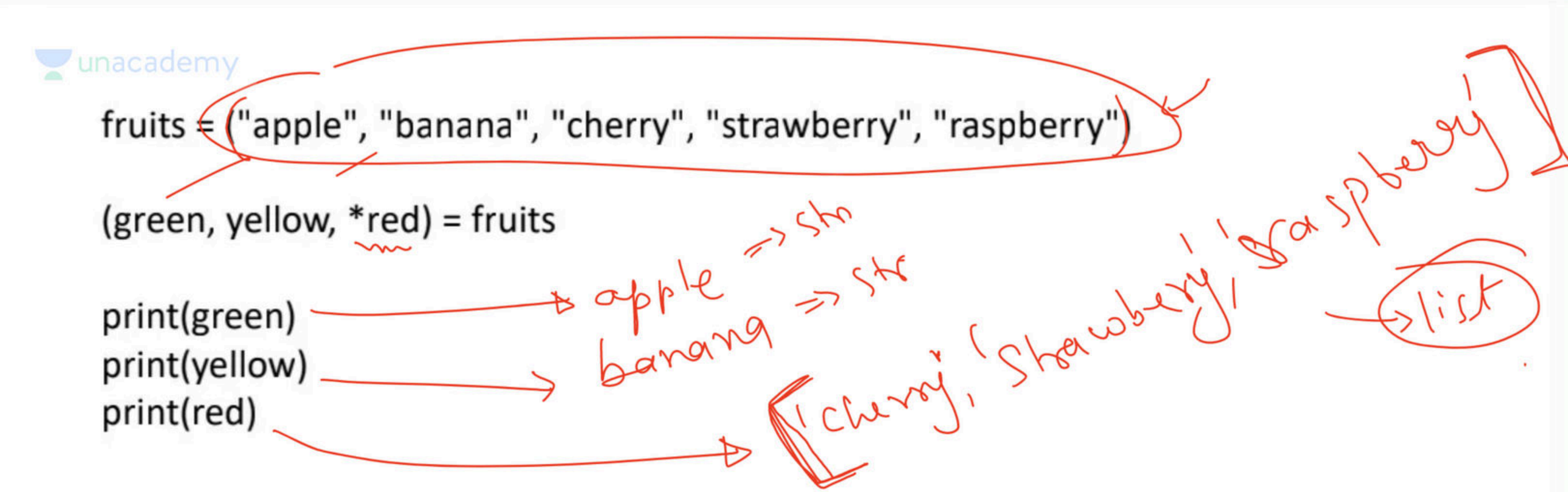
But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)
```




S = {1, 2, 3}
(x, y, z) = S
x, y, z = S
x ≠ y ≠ z
x, y, z ≠ 2
3 1

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```



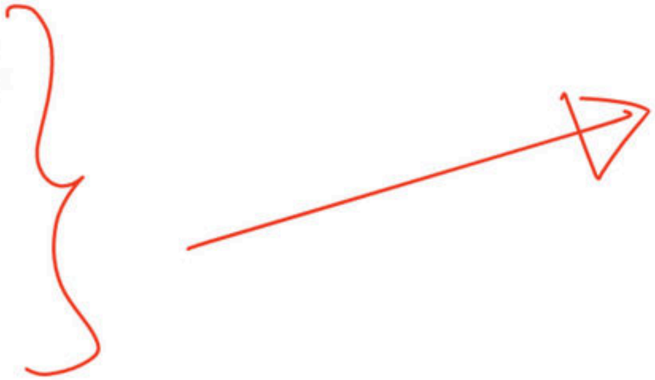

```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

print(green)
print(tropic)
print(red)
```

```
t= ("apple", "banana", "cherry")
for x in t:
  print(x)
```

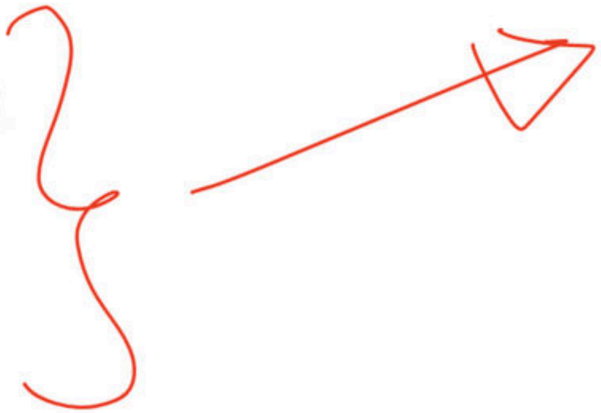
```
t = ("apple", "banana", "cherry")
for i in range(len(t)):
  print(t[i])
```

```
t = ("apple", "banana", "cherry")
i = 0
while i < len(t):
  print(t[i])
  i = i + 1
```

```
t1 = ("a", "b" , "c")
t2 = (1, 2, 3)
t3 = t1 + t2
print(t3)
```



```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2
print(mytuple)
```

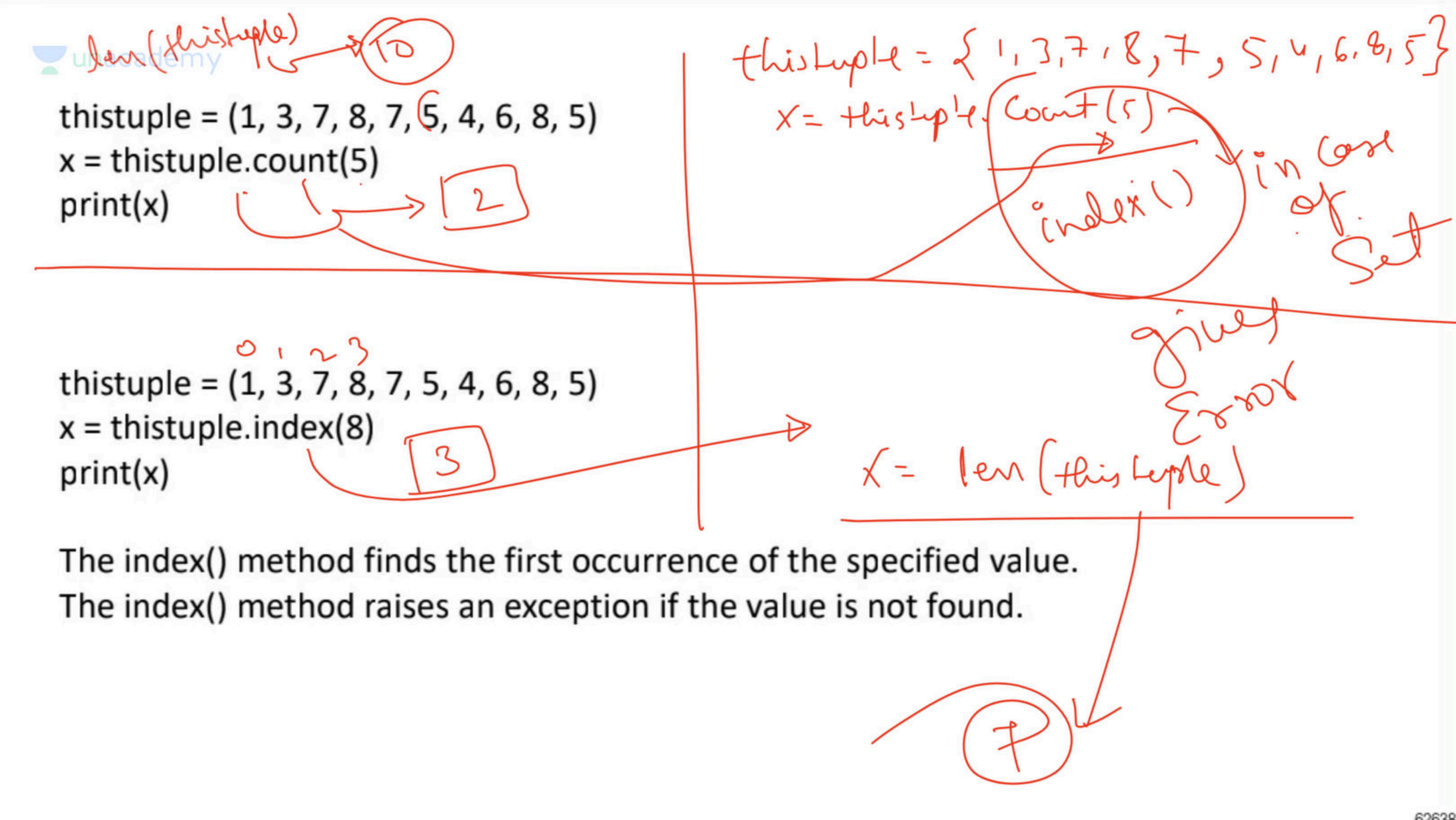


```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.count(5)
print(x)
```

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.index(8)
print(x)
```

The index() method finds the first occurrence of the specified value.

The index() method raises an exception if the value is not found.



There are four collection data types in the Python programming language:

List is a collection which is ordered and changeable. Allows duplicate members.

Tuple is a collection which is ordered and unchangeable. Allows duplicate members.

Set is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.

Dictionary is a collection which is ordered** and changeable. No duplicate members.

*Set *items* are unchangeable, but you can remove and/or add items whenever you like.

**As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

Thank You…