String Slicing

```
name = "Logic First"
```

Using Indexing

str[start:stop:step]

```
print(name[3])
```

```
i
```

```
print(name[0:4])
```

```
Logi
```

```
print(name[:4])
```

```
Logi
```

```
print(name[2:4])
```

```
gi
```

```
print(name[2:])
```

```
gic First
```

```
print(name[2:10:3])
```

```
g r
```

```
print(name[-2])
```

```
s
```

```
print(name[-5:-2])
```

```
Fir
```

```
print(name[-2:-5:-1])
```

```
sri
```

```
print(name[::-1])
```

```
tsriF cigoL
```

```
print(name[:-3:-1])
```

```
ts
```

```
print(name[2:-2])
```

```
gic Fir
```

Using Slice Method

```
x = slice(2,-2) # parameters similar to indexing
slice(start,stop,step)
print(name[x])
```

```
gic Fir
```

Lists

```
cities = ["Chennai","Madurai","Trichy","Coimbatore","Salem"]
val = [3,5,6,3,2,9]
list1 = ["chennai",3,"Salem"]
```

Accessing List with Indexing

```
print(cities[0])
print(val[2])
print(cities[:3])
print(cities[-2])
print(cities[::2])
print(val)
```

```
Chennai
6
['Chennai', 'Madurai', 'Trichy']
Coimbatore
['Chennai', 'Trichy', 'Salem']
[3, 5, 6, 3, 2, 9]
```

Modify

```
cities[2] = "Tiruchy"
print(cities)
```

```
['Chennai', 'Madurai', 'Tiruchy', 'Coimbatore', 'Salem']
```

Append

```
cities.append("Karur")
print(cities)
```

```
['Chennai', 'Madurai', 'Tiruchy', 'Coimbatore', 'Salem', 'Karur']
```

Insert

```
cities.insert(3,"Thanjavur")
print(cities)
```

```
['Chennai', 'Madurai', 'Tiruchy', 'Thanjavur', 'Coimbatore', 'Salem',
'Karur']
```

Remove using del

```
del cities[3]
print(cities)
```

```
['Chennai', 'Madurai', 'Tiruchy', 'Coimbatore', 'Salem', 'Karur']
```

Remove using pop()

```
deleted = cities.pop(2)
print(deleted + " has been deleted")
print(cities)
```

```
Tiruchy has been deleted
['Chennai', 'Madurai', 'Coimbatore', 'Salem', 'Karur']
```

Remove by value

```
city_del = "Coimbatore"
cities.remove(city_del)
print(cities)
```

```
['Chennai', 'Madurai', 'Salem', 'Karur']
```

Permanent Sort

```
cities.sort()
print(cities)
```

```
['Chennai', 'Karur', 'Madurai', 'Salem']
```

Temoporary Sort

```
print(sorted(cities))
print(sorted(val))
print(cities)
```

```
['Chennai', 'Karur', 'Madurai', 'Salem']
[2, 3, 3, 5, 6, 9]
['Chennai', 'Karur', 'Madurai', 'Salem']
```

Reverse

```
cities.reverse()
print(cities)
```

```
['Salem', 'Madurai', 'Karur', 'Chennai']
```

Length of a List

```
print(len(cities))
```

```
4
```

```
for city in cities:
    print(city.upper())
```

```
SALEM
MADURAI
KARUR
CHENNAI
```

```
cities2 = cities
print(cities2)
```

```
['Salem', 'Madurai', 'Karur', 'Chennai']
```

```
TN = cities
Karnataka = ['Bangalore','Mysore','Udupi']
AP = ['Tirupathi','Nellore','Vijayawada','Guntur']
India = [TN,Karnataka,AP]
print(India[0][1])
```

```
Madurai
```

```
int_2d = [[2,3,4],[3,4,5],[4,5,6]]
```

```
cities.remove('Karur')
print(TN)
```

```
['Salem', 'Madurai', 'Chennai']
```

```
TN = cities[:]  #   TN = cities.copy()
cities.append('Tanjavur')
print(TN)
print(cities)
```

```
['Salem', 'Madurai', 'Chennai']
['Salem', 'Madurai', 'Chennai', 'Tanjavur']
```

```
Indian_States = India[:] #shollow
India[0][0] = 'kadalur'
print(Indian_States[0][0])
```

```
kadalur
```

```
import copy
Indian_States = copy.deepcopy(India)
```

Tuples

immutable - cannot be changed

```
tup = (2,3,4)
print(tup)
```

```
(2, 3, 4)
```

```
# tup[1] = 5   - cannot be done - error
```

```
tup = (3,5,6)
print(tup)
```

```
(3, 5, 6)
```

```
print(tup[1])
```

```
5
```

```
print(tup.index(6))
```

```
2
```

```
tup = (3,4,5,4,4)
print(tup.count(4))
```

```
3
```

```
for i in tup:
    print(i)
```

```
3
4
5
4
4
```

```
if 3 in tup:
    print('yes')
```

```
yes
```

```
if 3 not in tup:
    print('no')
```

```
if tup:
    print('tup is not empty')
```

```
tup is not empty
```

Dictionary

key value pair

```
user = {'name':'Ram','age':25,'gender':'male'}
print(user['gender'])
print(user)
```

```
male
{'name': 'Ram', 'age': 25, 'gender': 'male'}
```

Adding a New Key value pair

```
user['city'] = 'chennai'
print(user)
```

```
{'name': 'Ram', 'age': 25, 'gender': 'male', 'city': 'chennai'}
```

Modify

```
user['age'] = 26
print(user)
```

```
{'name': 'Ram', 'age': 26, 'gender': 'male', 'city': 'chennai'}
```

Delete

```
del user['gender']
print(user)
```

```
{'name': 'Ram', 'age': 26, 'city': 'chennai'}
```

Looping

```
for key,val in user.items():
    print("key: " + key)
    print("val: " + str(val))
```

```
key: name
val: Ram
key: age
val: 26
```

```
key: city
val: chennai

for key in user.keys():
    print(key)

name
age
city

for key in sorted(user.keys()):
    print(user[key])

26
chennai
Ram

job = {'Priya':'CTS','John':'Amazon','Vidhya':'CTS'}
for company in job.values():
    print(company)

CTS
Amazon
CTS
```

List of Dictionaries

```
users = []
user = {'name':'Ram','age':25,'gender':'male'}
users.append(user)
user = {'name':'Ramya','age':26,'gender':'Female'}
users.append(user)

print(users[1]['name'])

Ramya
```

List in Dicitonary

```
user['fav_food'] = ['poori','pizza','pasta']
print(user)
print(user['fav_food'][0])

{'name': 'Ramya', 'age': 26, 'gender': 'Female', 'fav_food': ['poori',
'pizza', 'pasta']}
poori
```

Set

- unique elements, not ordered

```
colors = {'blue','red','orange','red'}
print(colors)
```

```
{'red', 'blue', 'orange'}
```

```
color_list = list(colors)
print(color_list)
```

```
['red', 'blue', 'orange']
```

String Formatting

```
name = 'hari'
like1 = 'apples'
like2 = 'bananas'

print(name + ' likes ' + like1 + ' and ' + like2)
```

```
hari likes apples and bananas
```

```
text = '{} likes {} and {}'
print(text.format(name,like1,like2))
```

```
hari likes apples and bananas
```

```
print('{} likes {} and {}'.format(name,like1,like2))
```

```
hari likes apples and bananas
```

```
text = '{0} likes {2} and {1}'
print(text.format(name,like1,like2))
```

```
hari likes bananas and apples
```

```
text = '{name} likes {fruit1} and {fruit2}'
print(text.format(name='hari',fruit1='apples',fruit2='bananas'))
print(text.format(name='ravi',fruit2='grapes',fruit1='oranges'))
```

```
hari likes apples and bananas
ravi likes oranges and grapes
```

padding

```
print("*****{msg}******".format(msg="welcome"))
```

```
*****welcome******
```

```
print("*****{msg:20}******".format(msg="welcome"))
```

```
*****welcome              ******
```

```
print("*****{msg:<20}******".format(msg="welcome"))
```

```
*****welcome              ******
```

```
print("*****{msg:>20}******".format(msg="welcome"))
```

```
*****              welcome******
```

```
print("*****{msg:^20}******".format(msg="welcome"))
```

```
*****       welcome        ******
```

```
print("*****{:^20}******".format("welcome"))
```

```
*****       welcome        ******
```

formating numbers

```
pi = 3.14159
print("The val of pi is {:.3f}".format(pi))
```

```
The val of pi is 3.142
```

```
num = 1000000
print("The num is {:,}".format(num))
```

```
The num is 1,000,000
```

```
num = 101
print("The num is {:b}".format(num))
```

```
The num is 1100101
```

b for binary, o for octal, x(or X) for hexa, E for scientific notation

Lambda

lambda par1,par2..parN:expn - anonymous function

```
def add_ten(num):
    return num+10
print(add_ten(8))
```

```
18
```

```
add_10=lambda x:x+10
print(add_10(5))
```

```
15
```

```
product = lambda a,b,c:a*b*c
print(product(5,4,6))
```

```
120
```

```
tall_enough = lambda h:h>175
print(tall_enough(150))
```

```
False
```

```
strong_enough = lambda w : "yes" if w>70 else "no"
print(strong_enough(50))
```

```
no
```

Sorting with Key

```
# (itemcode,itemname,price)
items = [(3456,"shoe",780),(3566,"phone",25300),(2587,"book",450),
(5412,"pen",75)] # list of tuples
items.sort()
print(items)
items.sort(key=lambda item:item[1])
print(items)
items.sort(key=lambda item:item[2])
print(items)
items.sort(key=lambda item:item[2],reverse=True)
print(items)
# sort method cannot be used with tuples as it modifies orginal tuple.
Tuples are immutable i.e cannot be modified
```

```
[(2587, 'book', 450), (3456, 'shoe', 780), (3566, 'phone', 25300),
(5412, 'pen', 75)]
[(2587, 'book', 450), (5412, 'pen', 75), (3566, 'phone', 25300),
(3456, 'shoe', 780)]
[(5412, 'pen', 75), (2587, 'book', 450), (3456, 'shoe', 780), (3566,
'phone', 25300)]
[(3566, 'phone', 25300), (3456, 'shoe', 780), (2587, 'book', 450),
(5412, 'pen', 75)]
```

Assignment

(subject,studentname,marks)

students = [("maths","Anitha",80),("biology","Anand",82),("biology","Balaji",70),
("maths","Chandru",90)]

sort by marks, sort by subject, sort by name

```
# using sorted - temporary - can be used with tuples

items = ((3456,"shoe",780),(3566,"phone",25300),(2587,"book",450),
(5412,"pen",75)) #tuple of tuples
print(sorted(items,key=lambda item:item[1]))
print(items)
print(sorted(items,key=lambda item:item[2]))
print(sorted(items,key=lambda item:item[2],reverse=True))
```

```
[(2587, 'book', 450), (5412, 'pen', 75), (3566, 'phone', 25300),
(3456, 'shoe', 780)]
((3456, 'shoe', 780), (3566, 'phone', 25300), (2587, 'book', 450),
(5412, 'pen', 75))
[(5412, 'pen', 75), (2587, 'book', 450), (3456, 'shoe', 780), (3566,
'phone', 25300)]
[(3566, 'phone', 25300), (3456, 'shoe', 780), (2587, 'book', 450),
(5412, 'pen', 75)]
```

```
items = [(3456,"shoe",780),(3566,"phone",25300),(2587,"book",450),
(5412,"pen",75)]
```

Map

map(function,iterable)

```
items = [(3456,"shoe",780),(3566,"phone",25300),(2587,"book",450),
(5412,"pen",75)]
inr_usd = lambda item:(item[0],item[1],item[2]/74)
inr_usd2 = lambda item:
(item[0],item[1],float("{:.2f}".format(item[2]/74)))
inr_usd3 = lambda item:(item[1],float("{:.2f}".format(item[2]/74)))
items_usd = list(map(inr_usd3,items))
print(items_usd)
```

```
[('shoe', 10.54), ('phone', 341.89), ('book', 6.08), ('pen', 1.01)]
```

```
val = [4,5,1,9,7]
val_sq = list(map(lambda x:x*x,val))
print(val_sq)
```

```
[16, 25, 1, 81, 49]
```

```
def sq_fun(num):
    return num*num
val_sq = list(map(sq_fun,val))
print(val_sq)
```

```
[16, 25, 1, 81, 49]
```

Assignment

(subject,studentname,marks)

Ques 1

students = [("maths","Anitha",180),("biology","Anand",182),("biology","Balaji",170),("maths","Chandru",190)]

create another list with name and marks for 100.

Ques 2

val = [72,78,99,12,56] expected result =['even,'even','odd','even','even']

Ques 3

temp = [102,99,89,70,103] covert temp in fahrenheit to celsius

Filter

filter(function,iterable) function should return true or false. If true, the item will be added to result, if false omitted.

```
items = [(3456,"shoe",780),(3566,"phone",25300),(2587,"book",450),
(5412,"pen",75)]
less_than_500 = lambda item:item[2]<500
filtered = list(filter(less_than_500,items))
print(filtered)
filtered = list(filter(lambda item:item[1][0]=='p',items))
print(filtered)
```

```
[(2587, 'book', 450), (5412, 'pen', 75)]
[(3566, 'phone', 25300), (5412, 'pen', 75)]
```

Assignment

Ques1

In the items list above, filter item numbers between 3000 and 4000

Ques 2

students = [("maths","Anitha",180),("biology","Anand",182),("biology","Balaji",170),("maths","Chandru",190)]

In students list, filter 1. Name starts with "A" . 2. Marks>=180

Reduce

reduce(function,iterable) - performs function on first two elements and repeats it until one value remains on the iterable.

```
import functools
vals = [4,7,8,4,3]
sum = functools.reduce(lambda x,y:x+y,vals)
print(sum)
```

```
26
```

```
chars = ['pyt','h','o','n']
word = functools.reduce(lambda x,y:x+y,chars)
print(word)
```

```
python
```

List Comprehension

To create a new list from existing list

list = [exprn for item in iterable]

```
# List of square of first 10 natural numbers
sq_list = list(map(lambda x:x*x,range(1,11)))
print(sq_list)
sq_list = [x*x for x in range(1,11) ]
print(sq_list)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

list = [exprn for item in iterable if cond]

```
temp = [28,30,25,38,22,31]
temp_filtered = [i for i in temp if  i<30]
print(temp_filtered)
```

```
[28, 25, 22]
```

list = [exprn if-else for item in iterable]

```
temp = [28,30,25,38,22,31]
temp_filtered = [i if i<30 else 0 for i in temp]
print(temp_filtered)
```

```
[28, 0, 25, 0, 22, 0]
```

Dictionary Comprehension

dictionary = {key : exprn for (key,value) in iterable}

```
cart =
{'phone':25000.00,'lamp':2560.60,'table':5499.99,'pen':20.50,'bag':650
.20,'kettle':1500.00}
cart_rounded = {k[0]:round(v) for (k,v) in cart.items()}
print(cart_rounded)
```

```
{'p': 20, 'l': 2561, 't': 5500, 'b': 650, 'k': 1500}
```

dictionary = {key : exprn for (key,value) in iterable if cond}

```
cart2 = {k:round(v) for (k,v) in cart.items() if v>1000}
print(cart2)
```

```
{'phone': 25000, 'lamp': 2561, 'table': 5500, 'kettle': 1500}
```

dictionary = {key : if/else for (key,value) in iterable}

```
cart3 = {key:val*.9 if val>20000 else val for (key,val) in
cart.items()}
print(cart3)
```

```
{'phone': 22500.0, 'lamp': 2560.6, 'table': 5499.99, 'pen': 20.5,
'bag': 650.2, 'kettle': 1500.0}
```

dictionary = {key : func for (key,value) in iterable}

```
def furn_disc(k,v):
    if(k=='table' or k=='lamp'):
        v = round(.95*v)
    return v

cart4 = {k:furn_disc(k,v) for (k,v) in cart.items()}
print(cart4)
```

```
{'phone': 25000.0, 'lamp': 2433, 'table': 5225, 'pen': 20.5, 'bag':
650.2, 'kettle': 1500.0}
```

Zip

zips two or more iterables into a single iterable

```
items = ['phone','lamp','table','pen']
prices = [25000,2500,5200,15]
stocks = [40,33,10]
zipped = list(zip(items,prices,stocks))
print(zipped)
```

```
[('phone', 25000, 40), ('lamp', 2500, 33), ('table', 5200, 10)]
```