Remove duplicates in a list:

sam\_list = [11, 13, 15, 16, 13, 15, 16, 11, 11,12,13]

result = []

for i in sam\_list:

if i not in result:

result.append(i)

print((result))

Reverse a string

string = "greekforgreek"

sstr = ""

for i in string:

sstr = i+sstr

print(sstr)

print(string[::-1])

**Deep Copy & shallow copy**

* A shallow copy creates a new object which stores the reference of the original elements.
* Shallow copy doesn’t create a copy of nested objects, instead it just copies the reference of nested objects.

import copy

old\_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

new\_list = copy.copy(old\_list)

print("Old list:", old\_list)

print("New list:", new\_list)

* A deepcopy creates a new object and adds the copies of nested objects present in the original elements.

import copy

old\_list = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]

new\_list = copy.deepcopy(old\_list)

print("Old list:", old\_list)

print("New list:", new\_list)

**Reverse a number**

num = 1234

reversed\_num = 0

while num != 0:

digit = num % 10

reversed\_num = reversed\_num \* 10 + digit

num //= 10

print("Reversed Number: " + str(reversed\_num))

call the duplicates in a list

def Repeat(x):

#\_size = len(x)

repeated = []

for i in range(len(x)):

k = i + 1

for j in range(k, len(x)):

if x[i] == x[j] and x[i] not in repeated:

repeated.append(x[i])

return repeated

print(Repeat(x=[10, 20, 30, 20, 20, 30, 40,

50, -20, 60, 60, -20, -20]))

from collections import Counter

l1 = [1,2,1,2,3,4,5,1,1,2,5,6,7,8,9,9]

d = Counter(l1)

print(d)

new\_list = list([item for item in d if d[item]>1])

print(new\_list)

find the duplicates in a string

string = "greekforgreek"

sstr = ""

for i in string:

if i not in sstr:

sstr = sstr+i

print(sstr)

**count the duplicates**

MyList = ["b", "a", "a", "c", "b", "a", "c", 'a']

res = {}

for i in MyList:

res[i] = MyList.count(i)

print(res)

* tuple & list which is faster why?

1.Tuples are immutable python datatype. Tuples are stored single block of memory.it doesn’t required extra space for storing new object.

2.Lists are allocated in two blocks.

This is the reason for tuple is faster than list.

* Generators

1.Generators are a function. In generators in the place of return we use yield function.

2.generators returns a generator object that is an iterable. i.e can be used as an iterator.

3.generators gives you lazy evaluation.

4.generators returning them all at once they return one-by-one and the generator function paused until the next item required.

def fibon(n):

a = b = 1

for i in xrange(n):

yield a

a, b = b, a + b

for x in fibon(10):

print (x)

def fib(limit):

a,b = 0,1

while a<limit:

yield a

a,b = b,a+b

for i in fib(5):

print(i)

* **Decorator**

Decorator takes in a function, adds some functionality and returns it.

1.Decorators are a function, which allows to modify the behavior of the function.

2.you can store a function in a variable.

3.you can pass the function as a parameter.

4.return the function from another function.

def create\_adder(x):

def adder(y):

return x+y

return adder

add\_15 = create\_adder(15)

print(add\_15(10))

* **List Comprehension**

1.List comprehension are used to create a new list from existing list.

2.used to create a single line of code.

**Map():**map function is used to return a map object.

**1**.returns a list of the results after applying the given function to each item of a given iterable like list, tuple, set and dict.

**Filter():**

**1.**The filter method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.

**2.**returns an iterator that is already filtered.

Result = filter(lambda x:x%2 !=0, seq)

Print(list(result))

**Reduce():**

1.reduce function to reduce a list into a single value.

2.reduce a list to a single value and to calculate the sum of all elements in the list.

From functools import reduce

Score = [75, 65, 80, 95, 50]

Total = reduce(lambda a,b:a+b, score)

**Lambda functions**

An anonymous function is a function that is defined without a name.

Lambda functions are defined using the lambda keyword.

# Program to filter out only the even items from a list

my\_list = [1, 5, 4, 6, 8, 11, 3, 12]

new\_list = list(filter(lambda x: (x%2 == 0) , my\_list))

print(new\_list)

# Program to double each item in a list using map()

my\_list = [1, 5, 4, 6, 8, 11, 3, 12]

new\_list = list(map(lambda x: x \* 2 , my\_list))

print(new\_list)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Remove elements from an array**

list1 = [11, 5, 17, 18, 23, 50]

for elem in list1:

if elem%2 == 0:

list1.remove(elem)

print(list1)

------------------------------

array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

array.remove(10)

array.pop(1)

print(array) del array[1:2]

**remove():**the remove function takes an elements as an argument and removes it from a defined list.

**Pop():**the python pop function is used to return the removed elements from the given list.it takes the index value of an element as an argument.

**Del():**delete method to delete an item not to return any value. we can delete a range of value from the given list.

**\*args (Non Keyword Arguments)and \*\*kwargs(keyword Arguments)**

1.We define a function to take a reusable code that performs similar operation. We call a function with the specific values.

def adder(\*num):

sum = 0

for n in num:

sum = sum + n

print("Sum:",sum)

adder(3,5)

adder(4,5,6,7)

adder(1,2,3,5,6)

\*\*kwargs are passed as a dictionary and these arguments make a dictionary inside the function with name same as a parameter excluding double asterisk.

def intro(\*\*data):

print("\nData type of argument:",type(data))

for key, value in data.items():

print("{} is {}".format(key,value))

intro(Firstname="Sita", Lastname="Sharma", Age=22, Phone=1234567890)

intro(Firstname="John", Lastname="Wood", Email="johnwood@nomail.com", Country="Wakanda", Age=25, Phone=9876543210)

**Sort and sorted functions**

1.The difference between sort and sorted functions is that the sort function will modify the list.

2.The sorted function will create a new list containing a sorted version of the list.

3.The sorted function will not modify the list passed as a parameter.

4.the sorted function will return a list so you must assign the returned data to a new variable. The sort function modifies the list in-place and has no return value.

vegetables = ['squash', 'pea', 'carrot', 'potato']

new\_list = sorted(vegetables) print(new\_list)

vegetables = ['squash', 'pea', 'carrot', 'potato']

vegetables.sort()

print(vegetables)

**ArayList and Array**

1.Array can store elements of only one datatype but list can store the elements of different data types too.

2.Array stores homogeneous data values, and the list can store heterogeneous data values.

**Python swapping string characters**

def swap(c, i, j):

c=list(c)

c[i], c[j] = c[j],c[i]

return ''.join(c)

s='subhani'

print(swap(s,0,3))

**Tuple as a key**

test\_list = [(4, 5), (1, 3), (9, 4), (8, 2), (10, 1)]

start = 4

\_dict = {}

for i in test\_list:

\_dict[start] = i

start +=1

print(\_dict)

------------------------------------

A = {7: [21, 29, 43], 16: [20, 21, 37, 49], 21: [7, 16, 43], 29: [2, 7], 43: [7, 21], 46: [23, 36]}

a=[]

for key, values in A.items():

for value in values:

a.append((key,value))

print(a)

**List as a key in python**

a =[1, 2, 3, 4, 5]

d = {}

p = str(a)

d[p]=1

for i, j in d.items():

print(i ,":",j)

**DUCK TYPING**

Duck typing is related into dynamically typed python language and polymorphism.

The main reason for using duck typing is to provide support for dynamic typing in python. Don’t need to specify the variable’s data type and we can reassign the different data type values to same variables in further code.

x = 12000

print(type(x))

x = 'Dynamic Typing'

print(type(x))

x = [1, 2, 3, 4]

print(type(x))

**Rotating list using python**

1.Python has a built-in data type called list.it is like a collection of arrays with different methodology. Data inside the list can be any data type.

2.Rotating list means pushing and pulling elements from both ends of the list.

3.In the left rotation each element of the list is shifted to its left side by one position and the first element is added to the end.

4.Right rotation each element of the list is shifted to its right side by one position and the last element is added to the start of the list.

def right\_rotate(lists, n):

output\_list = []

x= len(lists)

for item in range(x - n, x):

output\_list.append(lists[item])

for item in range(0, x - n):

output\_list.append(lists[item])

return output\_list

rotate\_num = 3

list\_1 = [10, 25, 36, 43, 52, 60]

print("List:", list\_1)

print("Rotated list: ",right\_rotate(list\_1, rotate\_num))

**matched keys into two dict**

dict\_1 = {

'3471': ['AY219713', 'AJ504663'],

'3460': ['AJ621556', 'AJ575744'],

'0': ['AM158981', 'AM158980', 'AM158982']}

dict\_2 = {

'478': ['AY219713', 'AJ504663'],

'43': ['AJ575744', 'AJ621556'],

'2321': ['AM158979', 'L37584', 'AM158982']}

match1 = {}

match2 = {}

# sort once: makes a difference when you have lots of elements

for k in dict\_1:

dict\_1[k] = sorted(dict\_1[k])

for j in dict\_2:

dict\_2[j] = sorted(dict\_2[j])

for k in dict\_1:

for j in dict\_2:

if (dict\_1[k] == dict\_2[j]):

match1[k]=1

match2[j]=1

break;

print("matching keys:")

print (list(match1))

print (list(match2))

print ("\nnon matching keys:")

print (list(set(dict\_1.keys()) - set(match1)))

print (list(set(dict\_2.keys()) - set(match2)))

**find the keys in dict**

dic ={"geeks": "A","for":"B","geeks":"C"}

value = {i for i in dic if dic[i]=="B"}

print("key by value:",value)

**find the value in dict**

d = {'key1': 'val1', 'key2': 'val2', 'key3': 'val3'}

print(d['key1'])

**factorial**

def factorial(x):

if x == 1:

return 1

else:

return (x \* factorial(x-1))

num = 7

result = factorial(num)

print("The factorial of", num, "is", result)

**Prime number or not**

num = int(input("enter a number :"))

if num <1:

print("not a prime number")

else:

for i in range(2, num):

if num%2==0:

print("not a prime number")

else:

print("It is prime number")

**Palindrome or not**

s = input("enter a string")

def palindrom(string):

x = ""

for i in string:

x = i+string

return x

if s==palindrom(s):

print("It is palindrom")

else:

print("it is not palindrom")

**PANDAS**

Pandas loc and iloc it comes to selecting the rows and columns of a pandas dataframe.

1.loc selects rows and columns with specific labels.

2.iloc selects rows and columns at specific integer positions.

import pandas as pd

df = pd.DataFrame({'team': ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B'],

'points': [5, 7, 7, 9, 12, 9, 9, 4],'assists': [11, 8, 10, 6, 6, 5, 9, 12]},

index=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'])

print(df)

df.loc[['E', 'F']]

df.loc[['E', 'F'], ['team', 'assists']]

df.loc['E': , :'assists']

df.iloc[4:6]

df.iloc[4:6, 0:2]

df.iloc[4: , :3]

print(df.index)

print(df.count())

**DATAFRAMES**

1.The pandas data frame is a structure that contains two-dimensional data and its corresponding labels. Data frames are widely used in data science, ML and many other data intensive fields.

2.Dataframes are similar to SQL tables, data frames are faster, easier to use and more powerful than tables.

3.Data organized in two dimensions, rows and columns.

4.Labels that corresponded to the rows and columns.

5.you can pass the data as a two-dimensional list, tuple or NumPy array, dictionary or pandas’ series.

**Assignment**

I would iterate through the letters keeping track of the vowels and consonants, and then use join at the end.

def encrypt\_sentence(sentence):

vowel\_set = set("aeiouAEIOU")

final\_list=[]

word=sentence.split()

for i in range(0,len(word)):

if((i%2)==0):

final\_list.append(word[i][::-1])

else: # do rearrangement

vowels = list()

consonants = list()

for letter in word[i]:

if letter in vowel\_set:

vowels.append(letter)

else:

consonants.append(letter)

new\_string = "".join(consonants) + "".join(vowels)

final\_list.append(new\_string)

return final\_list

**ACID(Atomicity, Consistency, Isolation, and Durability)**

* A transaction is a single logical unit of work that accesses and possibly modifiers the contents of database.
* Transactions access data using read and write operations.
* If a database operation has these ACID properties, it can be called ACID transaction and data storage system that apply these operations are called transaction systems.

**Packing & unpacking**

* The \*args and \*\*kwargs operators are use in packing and unpacking in python.

Def \_\_init\_\_(self, \*args, \*\*kwargs):

Pass

Packing as it pick all the arguments into one single variable that this method call receives into a tuple called args.

Unpacking :

* In python tuples are used to stores immutable object.
* A tuple is a sequence of immutable python object.
* tuples are sequences, the tuples cannot be changed and tuples use parentheses.
* In unpacking of tuple number of variables on LHS should be equal to number of values in given tuple. In packing we put values into a new tuple while in unpacking we extract those values into a single variable.

Tup\_1 = (“a”,12,”c”) #packing

(“d”,”e”,15) = tup\_2 #unpacking

**Pickling :**

* Pickling is used in Serializing and deserializing a python object structure.
* The process of converting a python object into a byte stream to store in a file or database.to transfer the data over the network.

Str = “abc”

Arr = bytes(str, ‘utf-8’)

Print(Arr)

**Unpickling:**

* Unpickling is a process by which original python objects are retrieved from the stored string representation from the pickle file. It converts the byte stream into a python object.

B = b”Lets grab a \xf0\x9f\x8d\x95!”

Type(B)

S = b.decode(‘UTF-8’)

Print(S)

**Delete a file in Python**

Import os

Os.remove(“filename”)

String operators in python: =,+,\*,[],&!=,in,not in

**Constructors in python**

* Constructors are used for instantiating an object.in python \_\_init\_\_() method is called the constructor and it is always called when an object is created.

1. Defaulf constructor:is a simple constructor which doesn’t accept any arguments.only one argument which reference to the instance being constructed.
2. Parameterized constructor: constructor with parameters is known as parameterized constructor.the parameterized constructor takes first argument as a reference to the instance being constructed know as self.

**Search index in an array python**

* L1 = [[1,2,3,4],[5,6,7],[8,9,10]]
* L1.index([1,2,3,4])

**Tuple Matching**

* Tuple matching in python is a method of grouping the tuples by matching the second element in the tuples.it Is used in dict by checking the second element in each tuple.

**Cache Memory in python**

* To store data in a temporary location instead of retrieving it from the source each time.
* Python doesn’t catche memory automatically,the call will be done twice.