List.Flatten

In [ ]:

```
# List.Flatten
Lsts = [[1,2,3], [4,5,6], [7], [8,9]]
        # [1, 2, 3, 4, 5, 6, 7, 8, 9]
# Funtion
def Flatten(Lsts):
    output = []
    for i in range(len(Lsts)):
        output += Lsts[i]
    return output

# print
Flatten(Lsts)
```

List.Chop

In [ ]:

```
# List.Chop
Lst = [1, 2, 3, 4, 5]

# Funtion
def ListChop(Lst, Length = 2):
    output = []
    for i in range(0, len(Lst), Length):
        output.append(Lst[i:i + Length])
    return output

# Print
ListChop(Lst, 1)
```

List.UniqueItems | Remove duplicate elements

In [ ]:

```
# List.UniqueItemss
Lst = [2, 4, 10, 20, 5, 2, 20, 4]

# Python code to remove duplicate elements
def UniqueItems(Lst):
    output = []
    for x in Lst:
        if x not in output:
            output.append(x)
    return output

# print
UniqueItems(Lst)
```

List.IndexOf

In [ ]:

```
# List.IndexOf
Lst = ["b", "a", "e", "c", "f"]  # [ 0,   1,   2,   3,   4 ]
Elemnt = ["a", "b", "c"]
# Funtion
def IndexOf(Lst, Elemnt):
    output = []
    for x in Elemnt:
        Locations = Lst.index(x)
        output.append(Locations)
    return output
# Print
IndexOf(Lst, Elemnt)
```

List.GetItemAtIndex

In [ ]:

```
# List.GetItemAtIndex
Lst = [10, 20, 30, 40, 50]
    # [ 0,  1,  2,  3,  4]
InDex = [0, 2, 4]
# Funtion
def GetItemAtIndex(Lst, InDex):
    output = []
    for i in InDex:
        output.append(Lst[i])
    return output
# Print
GetItemAtIndex(Lst, InDex)
```

List.Split.String | Setting the Specific Data Type

In [ ]:

```
# StringSplit, setting the Specific Data Type
Lst = ["CH = 2700", "CH = 2800"]
Separator1 = "="
# Funtion
def StringSplit(Lst):
    output = []
    for Str in Lst:
        for i in Str:
            if i == Separator1:
                k = Str.split(i)
                output.append(int(k[-1])) # String.ToNumber
    return output

# Print
StringSplit(Lst)
```

List.SecondLargest

In [ ]:

```
# Find the second largest number in an array
def second_largest(given_list):
    largest = None
    second_largest = None
    for i in given_list:
        if largest == None:
            largest = i
        elif i > largest:
            second_largest = largest
            largest = i
        elif second_largest == None:
            second_largest = i
        elif i > second_largest:
            second_largest = i
    return  largest

print(second_largest( [1, 3, 4, 5, 0, 2] ) )
print(second_largest( [1, 3, 4, 2] ) )
print(second_largest( [2, 2, 1] ) )
print(second_largest( [2] ) )
print(second_largest( [] ) )
```

List.String.Front_Back

In [ ]:

```
# Given a string, return a new string where the first
# and last chars have been exchanged

# Method 1:
def front_back(Str):
    if len(Str) <= 1:
        return Str
    mid = Str[1:len(Str)-1]  # can be written as str[1:-1]
  # last + mid + first
    return Str[len(Str)-1] + mid + Str[0]

# Method2:
def front_back(Str):
    out = []
    if len(Str) <= 1:
        return Str
    else:
        for a in Str:
            out.append(a)
        out[0] = Str[-1]
        out[-1]= Str[0]
        return "".join(out)

print(front_back('code'))
print(front_back(''))
print("a")
print("Ab")
```

List.not_string

In [ ]:

```
# Given a string, return a new string where "not " has been added to the front.
# However, if the string already begins with "not", return the string unchanged.

def not_string(Str):
    if Str[:3] != "not":
        return "not " + Str
    return Str
  # str[:3] goes from the start of the string up to but not
  # including index 3

not_string('candy')
```

List.String.makes10

In [ ]:

```
# Given 2 ints, a and b, return True if one if them is 10 or if their sum is 10.


# makes10(9, 10) → True
# makes10(9, 9) → False
# makes10(1, 9) → True

def makes10(a, b):
    return (a == 10 or b == 10 or a+b == 10)
```

List.String.Front3

In [ ]:

```
# Given a string, we'll say that the front is the first 3 chars of the string.
# If the string length is less than 3, the front is whatever is there.
# Return a new string which is 3 copies of the front.

# front3('Java') → 'JavJavJav'
# front3('Chocolate') → 'ChoChoCho'
# front3('abc') → 'abcabcabc'

def front3(str):
  # Figure the end of the front
    front_end = 3
    if len(str) < front_end:
        front_end = len(str)
    front = str[:front_end]
    return front + front + front


# Could omit the if logic, and write simply front = str[:3]
# since the slice is silent about out-of-bounds conditions.
```

List.string_times

In [ ]:

```
#Given a string and a non-negative int n,
# return a larger string that is n copies of the original string.

# string_times('Hi', 2) → 'HiHi'
# string_times('Hi', 3) → 'HiHiHi'
# string_times('Hi', 1) → 'Hi'

def string_times(Str, n):   # return n*Str
    result = ""
    for i in range(n):  # range(n) is [0, 1, 2, .... n-1]
        result = result + Str
        # could use += here
    return result
```

List | Return the number of 9's in the array

In [ ]:

```
# Given an array of ints, return the number of 9's in the array.

# array_count9([1, 9, 9]) → 2
# array_count9([1, 9, 9, 3, 9]) → 3

def array_count9(nums):
    count = 0
# Standard loop to look at each value
    for num in nums:
        if num == 9:
            count = count + 1
    return count
```

List.array123

In [ ]:

```
#  Given an array of ints,
# return True if the sequence of numbers 1, 2, 3 appears in the array somewhere.
# array123([1, 1, 2, 3, 1]) → True
# array123([1, 1, 2, 4, 1]) → False

def array123(nums):
    # Note: iterate with length-2, so can use i+1 and i+2 in the loop
    for i in range(len(nums)-2):
        if nums[i]==1 and nums[i+1]==2 and nums[i+2]==3:
            return True
    return False
```

sleep_in

In [ ]:

```
# The parameter weekday is True if it is a weekday,
# and the parameter vacation is True if we are on vacation.
# We sleep in if it is not a weekday or we're on vacation.
# Return True if we sleep in.
# sleep_in(False, False) → True  True    OK
# sleep_in(True, False) → False  False   OK
# sleep_in(False, True) → True   True    OK
# sleep_in(True, True) → True    True    OK

def sleep_in(weekday, vacation):
    if not weekday or vacation:
        return True
    else:
        return False
# This can be shortened to: return(not weekday or vacation)
```

monkey_trouble

In [ ]:

```
# We have two monkeys, a and b, and the parameters a_smile and b_smile indicate
# if each is smiling.
# We are in trouble if they are both smiling or if neither of them is smiling.
# Return True if we are in trouble.

# monkey_trouble(True, True) → True
# monkey_trouble(False, False) → True
# monkey_trouble(True, False) → False

def monkey_trouble(a_smile, b_smile):
    if a_smile and b_smile:
        return True
    if not a_smile and not b_smile:
        return True
    return False
## Or this very short version (think about how this is the same as the above)
##   return (a_smile == b_smile)
```

Absolute difference between n and 21

In [ ]:

```
# Given an int n, return the absolute difference between n and 21,
# except return double the absolute difference if n is over 21.
# diff21(19) → 2

def diff21(n):
    if n <= 21:
        return 21 - n
    else:
        return 2*(n-21)
```

Sum_double

In [ ]:

```
# Given two int values, return their sum.
# Unless the two values are the same, then return double their sum.

# sum_double(1, 2) → 3
# sum_double(3, 2) → 5
# sum_double(2, 2) → 8

def sum_double(a, b):
  # Store the sum in a local variable
    Sum = a + b
  # Double it if a and b are the same
    if a == b:
        Sum = Sum * 2
    return Sum
```

Parrot_trouble #Ctrl-/ comment

In [ ]:

```
# We have a loud talking parrot.
# The "hour" parameter is the current hour time in the range 0..23.
# We are in trouble if the parrot is talking and the hour is before 7 or after 20.
# Return True if we are in trouble.


# parrot_trouble(True, 6) → True
# parrot_trouble(True, 7) → False
# parrot_trouble(False, 6) → False

def parrot_trouble(talking, hour):
    return (talking and (hour < 7 or hour > 20))
# Need extra parenthesis around the or clause
# since and binds more tightly than or.
# and is like arithmetic *, or is like arithmetic +
```

pos_neg

In [ ]:

```
# Given 2 int values, return True if one is negative and one is positive.
# Except if the parameter "negative" is True, then return True only if both are negati

def pos_neg(a, b, negative):
    if negative:
        return (a < 0 and b < 0)
    else:
        return ((a < 0 and b > 0) or (a > 0 and b < 0))

# pos_neg(1, -1, False) → True
# pos_neg(-4, -5, True) → True
```

missing_char

In [ ]:

```
# Given a non-empty string and an int n,
# return a new string where the char at index n has been removed.
# The value of n will be a valid index of a char in the original string

# missing_char('kitten', 4) → 'kittn'
# Method1:
def missing_char(Str, n):
    Lst = Str.split(Str[n])   # for Str1 in Lst:
    New_Str = "".join(Lst)           # New_Str += Str1 ( New_Str = "")
    return New_Str
# Method2:
def missing_char2(Str, n):
    front = Str[:n]
    back  = Str[n+1:]
    return fronk + back
```

Enumerate() in Python

In [ ]:

```
element = ["Textnote1", "Textnote2", "Textnote3", "Textnote4"]
Type    = ["DVQ1",      "DVQ2",      "DV3",       "DVQ4"   ]
#
def Enum(element, Type):
    out = []
    for i,t in enumerate(Type):
        if "DVQ" in t:
            out.append(element[i])
        else:
            pass
    return out
#
print(Enum(element, Type))
```

Filter list by Boolean list | List.FilterByBoolMask

In [ ]:

```
Lst =  [6,    4,     8,     9,    10  ] # List to filter
Mask = [True, False, False, True, True] # List of booleans representing a mask.

def FilterByBoolMask(Lst, Mask):
    IN = []                             # Items whose mask index is True
    OUT = []                            # Items whose mask index is False
    for pos, ele in enumerate(Mask):
        if ele == True:
            IN.append(Lst[pos])
        elif ele == False:
            OUT.append(Lst[pos])
    return IN, OUT
#
print(list(FilterByBoolMask(Lst, Mask)))
```

List.Contains + List.Map

In [ ]:

```
Lst1 = ["b", "a", "e", "a","f"] # [True, True, False, True, False]
Lst2 = ["a", "b", "c"]

# Lst1[1] = False
def CheckContain(Lst1, Lst2):
    for x in range(len(Lst1)):
        for y in range(len(Lst2)):
            if Lst1[x] == Lst2[y]:
                Lst1[x] = True
        if Lst1[x] != True:
            Lst1[x] = False
    return Lst1

CheckContain(Lst1, Lst2)
```

List.Contains + List.Map + List.FilterByBoolMask

In [ ]:

```
# List.Contains + List.Map + List.FilterByBoolMask
Lst1 = ["b", "a", "e", "c","f"]
b = [10, 20, 30, 40, 50]
Lst2 = ["a", "b", "c"]
# Funtion
def CMF(Lst1, Lst2):
    out = []
    out1 = []
    for ele in Lst1:
        if ele not in Lst2:
            out.append(ele)
        elif ele in Lst2:
            out1.append(ele)
    return out1, out
# Print
print(list(CMF(Lst1, Lst2)))
```

near_hundred

List.MaximumItem

List.SetUnion

List.SetDifference

List.LastItem

List.Transpose

In [ ]:

```
# Python program to get transpose
def transpose(l1, l2):
    for i in range(len(l1[0])):
        row =[]
        for item in l1:
            row.append(item[i])
        l2.append(row)
    return l2

# Driver code
l1 = [[4, 5, 3, 9], [7, 1, 8, 2], [5, 6, 4, 7]]
l2 = []
print(transpose(l1, l2))

l = [[1, 2, 3], [4, 5], [7, 8, 9]]
list(map(list, zip(*l)))
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```