H2 Computing Notes

Basic Knowledge on Python assumed, including but not limited to

- if-else, loops, functions
- try-except

This is a series of common algorithms/ data structures/ pieces of code to refer to, which could be useful in examinations. This includes basic example/ boilerplate code as well as template code for easy memorisation and regurgitation.

Contents

1. Common Algorithms/ Libraries

1.1 Basics

In [1]:

```
# Simulate a REPEAT-UNTIL loop
untilCondition = False
while not untilCondition:
    untilCondition = True
```

In [2]:

```
# While loop to simulate a for loop
output = ""

counter = 0
while counter < 10:
    output += f"{counter} "
    counter += 1
print(output)
```

```
0 1 2 3 4 5 6 7 8 9
```

1.2 Input Control Structures

In [3]:

```
# Sentinel value termination
def thingsInput():
    things = []

    isRunning = True
    while isRunning:
        option = input("Enter thing ('Q' to quit): ").strip()

        if option == 'Q':
            isRunning = False
        else:
            things.append(option)

    return things

thingsInput()
```

```
Enter thing ('Q' to quit): thing
Enter thing ('Q' to quit): Q
```

Out[3]:

```
['thing']
```

In [4]:

```
MENU_TEXT = """\
1. Option 1
2. Option 2
3. Option 3
4. Exit
"""
def menu():
    isRunning = True
    while isRunning:
        print(MENU_TEXT)
        option = input("Enter option: ").strip()

        # options
        if option == '1':
            print("Option 1 chosen")
        elif option == '2':
            print("Option 2 chosen")
        elif option == '3':
            print("Option 3 chosen")
        elif option == '4':
            isRunning = False

menu()
```

```
1. Option 1
2. Option 2
3. Option 3
4. Exit

Enter option: 1
Option 1 chosen
1. Option 1
2. Option 2
3. Option 3
4. Exit

Enter option: 4
```

1.3 Validation

In [5]:

```
# Basic Loop
def validInput(isValid, prompt="Enter string: ", error="The input is not valid"):
    inputString = input(prompt)
    while not isValid(inputString):
        print(error)
        inputString = input(prompt)
```

In [6]:

```
# Presence Check
def isPresent(inputString) :
    return inputString != ""

validInput(isPresent)
```

```
Enter string:
The input is not valid
Enter string: There's a string here
```

In [7]:

```
# Length Check
def isLength(inputString):
    return len(inputString) == 1

validInput(isLength)
```

```
Enter string: notLength1
The input is not valid
Enter string: 1
```

In [8]:

```
# Format Check
def isDigit(inputString):
    return inputString.isdigit()

def isAlpha(inputString):
    return inputString.isalpha()

validInput(isAlpha)
```

```
Enter string: Has spaces
The input is not valid
Enter string: OnlyAlphabets
```

In [9]:

```
# Range Check
def inRange(inputString):
    number = int(inputString)
    return -100 < number < 0

validInput(inRange)
```

```
Enter string: 0
The input is not valid
Enter string: -100
The input is not valid
Enter string: -99
```

1.4 Number Handling

In [10]:

```
# Input Sentinel
def numbersSentinel():
    total = 0
    maximum = 0
    minimum = 999999999999999999999 # Set to large value
    count = 0

    isRunning = True
    while isRunning:
        ### Input ################################
        option = input("Enter number ('Q' to quit): ").strip()
        if option == 'Q':
            isRunning = False
        ### Processing ###########################
        else:
            number = int(option)
            count += 1
            total += number
            if number > maximum: # or maximum == None # Invalid value
                maximum = number
            if number < minimum: # or minimum == None # Invalid value
                minimum = number


    average = total / count
    return count, total, average, maximum, minimum

numbersSentinel()
```

```
Enter number ('Q' to quit): 1
Enter number ('Q' to quit): 2
Enter number ('Q' to quit): 3
Enter number ('Q' to quit): Q
```

Out[10]:

```
(3, 6, 2.0, 3, 1)
```

In [11]:

```
# Input Fixed Loop
def numbersLoop():
    total = 0
    maximum = 0
    minimum = 999999999999999999999 # Set to large value
    count = int(input("Enter no. of numbers to input: "))

    for i in range(count):
        option = input(f"Enter the number {i + 1} number :").strip()
        number = int(option)
        total += number
        if number > maximum: # or maximum == None # Invalid value
            maximum = number
        if number < minimum: # or minimum == None # Invalid value
            minimum = number


    average = total / count
    return count, total, average, maximum, minimum

numbersLoop()
```

```
Enter no. of numbers to input: 3
Enter the number 1 number :1
Enter the number 2 number :2
Enter the number 3 number :3
```

Out[11]:

```
(3, 6, 2.0, 3, 1)
```

In [12]:

```
# Generic Calculations
def numbersCalculations(numList):
    total = 0
    maximum = 0
    minimum = 999999999999999999999 # Set to large value
    count = len(numList)

    for number in numList:
        total += number
        if number > maximum: # or maximum == None # Invalid value
            maximum = number
        if number < minimum: # or minimum == None # Invalid value
            minimum = number

    average = total / count
    return count, total, average, maximum, minimum

numbersCalculations([9, 8, 7, 6, 5, 4, 3, 2, 1])
```

Out[12]:

```
(9, 45, 5.0, 9, 1)
```

In [13]:

```
# Multiple Minimum or Maximum
def multipleMinMax(numList, stringList):
    minVal = None
    minItems = []
    maxVal = None
    maxItems = []

    for index in range(len(numList)):
        if minVal == None or minVal > numList[index]:
            minVal = numList[index]
            minItems = [stringList[index]]
        elif minVal == numList[index]:
            minItems.append(stringList[index])

        if maxVal == None or maxVal < numList[index]:
            maxVal = numList[index]
            maxItems = [stringList[index]]
        elif maxVal == numList[index]:
            maxItems.append(stringList[index])

    return minVal, minItems, maxVal, maxItems

multipleMinMax([1, 1, 3, 2, 3], ["a", "b", "c", "d", "e"])
```

Out[13]:

```
(1, ['a', 'b'], 3, ['c', 'e'])
```

In [14]:

```
# Other Calculations
import math
dp = 2
print(round(math.pi, dp))
print(math.ceil(math.pi))
print(math.floor(math.pi))
```

```
3.14
4
3
```

In [15]:

```
# Random
import random
random.randint(0, 10)
#random.randrange(0, 10) # Excludes 1
```

Out[15]:

```
9
```

1.5 File I/O

Python's built-in libraries

In [16]:

```
## Write mode
file = open('TEST.txt', 'w')
file.write('This is some test data\nAnd this is another line')
file.close()
```

In [17]:

```
## Read mode
'''
Other functions
file.read() # Reads all items
file.readlines() # Reads lines into list (with newline character behind each line)
file.readline() # Reads 1 line at a time
'''
file = open('TEST.txt') #,'r')
for line in file:
    print(line.strip())
file.close()
```

```
This is some test data
And this is another line
```

In [18]:

```
## Append mode
file = open('TEST.txt', 'a')
file.write('\nAnd this is another appended line')
file.close()
```

In [19]:

```
## Read again
file = open('TEST.txt')
print(file.read())
file.close()
```

```
This is some test data
And this is another line
And this is another appended line
```

CSV

In [20]:

```
## CSV Writer
data = [
    ['Alice', 'H1GP', 'H2MATH', 'H2ECONS', 'H2PHY', 'H2COMP'],
    ['Bobby', 'H1GP', 'H2MATH', 'H2ECONS', 'H2PHY', 'H2CLL'],
    ['Charlie', 'H1GP', 'H2MATH', 'H2ECONS', 'H2CHEM', 'H2COMP'],
    ['Danny', 'H1GP', 'H2MATH', 'H2PHY', 'H2CHEM', 'H2COMP'],
    ['Ernest', 'H1GP', 'H2MATH', 'H2PHY', 'H2COMP', 'H2ELL']
]

import csv
file = open('studentsubjects.csv','w', newline='') # Newline
writer = csv.writer(file)# , delimiter=' ',quotechar='|', quoting=csv.QUOTE_MINIMAL)

#for record in data: writer.writerow(record)
writer.writerows(data)
file.close()
```

In [21]:

```
## CSV Reader
import csv
file = open('studentsubjects.csv') #,'r')
reader = csv.reader(file)
for record in reader:
    print(record)
file.close()
```

```
['Alice', 'H1GP', 'H2MATH', 'H2ECONS', 'H2PHY', 'H2COMP']
['Bobby', 'H1GP', 'H2MATH', 'H2ECONS', 'H2PHY', 'H2CLL']
['Charlie', 'H1GP', 'H2MATH', 'H2ECONS', 'H2CHEM', 'H2COMP']
['Danny', 'H1GP', 'H2MATH', 'H2PHY', 'H2CHEM', 'H2COMP']
['Ernest', 'H1GP', 'H2MATH', 'H2PHY', 'H2COMP', 'H2ELL']
```

In [22]:

```
! rm studentsubjects.csv | rm TEST.txt
```

1.6 Base Conversion

In [23]:

```
def baseToDenary(number, base):
    mapping = {}

    for digit in range(0, 9+1):
        mapping[str(digit)] = digit

    for value in range(0, 26):
        mapping[chr(65+value)] = 10+value
        mapping[chr(97+value)] = 10+value

    value = 0
    for i in range(len(number)):
        currDigit = number[-1-i]
        value += mapping[currDigit] * (base**i)

    return value

def denaryToBase(number, base):
    reference = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"

    ans = ""
    remaining = number
    while remaining // base > 0:
        currPlaceValue = remaining % base
        currDigit = reference[currPlaceValue]
        ans = currDigit + ans
        remaining = remaining // base
    # For the last place
    currPlaceValue = remaining % base
    if currPlaceValue != 0:
        currDigit = reference[currPlaceValue]
        ans = currDigit + ans

    return ans
```

In [24]:

```
baseToDenary('AB',16)
```

Out[24]:

```
171
```

In [25]:

```
denaryToBase(10, 16)
```

Out[25]:

```
'A'
```

1.7 Datetime

In [26]:

```
from datetime import datetime
now = datetime.now()

## Accessing attributes
print("Object now: \n", now, "\n")
print("Get Current date, time and weekday: \n", now.date(), now.time(), now.weekday(),
"\n")
print("Get min and max date and time: \n", now.min, now.max, "\n")
print("Get date attributes: \n", now.year, now.month, now.day, "\n")
print("Get time attributes: \n", now.hour, now.minute, now.second, now.microsecond, "\n
")

# Display Date
print("Display with formatting: \n", now.strftime("%Y/%m/%d %H:%M:%S"))
print("Display in ISO Format: \n", now.isoformat(), "\n")
```

```
Object now:
 2020-09-20 12:11:13.239108

Get Current date, time and weekday:
 2020-09-20 12:11:13.239108 6

Get min and max date and time:
 0001-01-01 00:00:00 9999-12-31 23:59:59.999999

Get date attributes:
 2020 9 20

Get time attributes:
 12 11 13 239108

Display with formatting:
 2020/09/20 12:11:13
Display in ISO Format:
 2020-09-20T12:11:13.239108
```

In [27]:

```
## Create your own dates

#datetime(2017, 11, 28, 23, 55, 59, 342380)
datetime.strptime("2020/10/31 15:10:34", "%Y/%m/%d %H:%M:%S")
```

Out[27]:

```
datetime.datetime(2020, 10, 31, 15, 10, 34)
```

In [28]:

```
## DateTime Arithmetic

from datetime import timedelta
diff = timedelta( weeks=1, days=0, hours=1, minutes=0, seconds=0, microseconds=0)
print("Past: \n",now-diff, "\n")
print("Future: \n",now+diff, "\n")
```

```
Past:
 2020-09-13 11:11:13.239108

Future:
 2020-09-27 13:11:13.239108
```

1.8 F-Strings

In [29]:

```
## Justification and spacing
print("|", f"{'hello':<10}", "|")
print("|", f"{'hello':^10}", "|")
print("|", f"{'hello':>10}", "|")
print("|", f"{'hello':->10}", "|") # Fill empty space with extra stuff

spacing = 30
print("|", f"{'Variable spacing':>{spacing}}", "|")
```

```
| hello      |
|   hello    |
|      hello |
| -----hello |
|               Variable spacing |
```

In [30]:

```
## Number Formatting
num = 1 # Set as you like

# Significant figures
print("2 Decimal places:", f"{num:.2f}")
print("2 Signicant figures:", f"{num:.2g}")

# Formats
print()
print("Binary form:", f"{num:2b}")

# Signs
print()
print("Show sign:", f"{num: 2}")# Leading space in front of positive numbers only
print("Show sign:", f"{num:+2}") # Always Show sign
print("Show sign:", f"{num:+2}")# Only for negative numbers
```

```
2 Decimal places: 1.00
2 Signicant figures: 1

Binary form:  1

Show sign:  1
Show sign: +1
Show sign: +1
```

In [31]:

```
## Combination
print("|", f"{1010:<+10.2f}", "|")
```

```
| +1010.00   |
```

In [32]:

```
#help('FORMATTING') # Run this command for more information about f-strings
```

In [33]:

```
## Generating a table

def table(header, spacings, data):
    output = ''
    for index in range(len(header)):
        output += f'{header[index]:<{spacings[index]}}'

    for record in data:
        output += '\n'
        for index in range(len(record)):
            item = record[index]
            output += f'{item:<{spacings[index]}}'

    print(output)

header = ['A','B','C','D']
spacings = [10,10,10,10]
data = [[j for j in range(i,i+4)] for i in range(10)]
table(header, spacings, data)
```

```
A         B         C         D
0         1         2         3
1         2         3         4
2         3         4         5
3         4         5         6
4         5         6         7
5         6         7         8
6         7         8         9
7         8         9         10
8         9         10        11
9         10        11        12
```

2. Sorting Algorithms

2.1 Bubble Sort

In [34]:

```
## Iterative
def bubbleSort(arr):
    noswap = False
    while noswap == False:
        noswap = True
        for i in range(1,len(arr)):
            if not(arr[i-1] <= arr[i]):
                # Swapping
                temp = arr[i]
                arr[i] = arr[i-1]
                arr[i-1] = temp
                noswap = False

arr = [1,0,3,6,3,45,2]
bubbleSort(arr)
arr
```

Out[34]:

```
[0, 1, 2, 3, 3, 6, 45]
```

In [35]:

```
## Recursive

def recursiveBubbleSort(arr, low, high):
    noswap = True
    for i in range(low+1, high):
        if not(arr[i-1] <= arr[i]):
            # Swapping
            temp = arr[i]
            arr[i] = arr[i-1]
            arr[i-1] = temp
            noswap = False
    if noswap == False:
        recursiveBubbleSort(arr, low+1, high)

arr = [1,0,3,6,3,45,2]
recursiveBubbleSort(arr,0,len(arr))
arr
```

Out[35]:

```
[0, 1, 3, 2, 3, 6, 45]
```

2.2 Insertion Sort

In [36]:

```
## Iterative
def insertionSort(arr):
    for key_pos in range(len(arr)):
        ## Shifting
        key = arr[key_pos]
        shift_pos = key_pos - 1
        while shift_pos >= 0 and arr[shift_pos] >= key:
            arr[shift_pos+1] = arr[shift_pos]
            shift_pos -= 1

        ## Insertion
        arr[shift_pos+1] = key


arr = [1,0,3,6,3,45,2]
insertionSort(arr)
arr
```

Out[36]:

```
[0, 1, 2, 3, 3, 6, 45]
```

In [37]:

```
## Recursive

def recursiveInsertionSort(arr, curr_pass=1):
    ### Base Case ############################
    if curr_pass >= len(arr):
        return

    ### Recursive Process ##################
    key_pos = curr_pass

    ## Shifting
    key = arr[key_pos]
    shift_pos = key_pos - 1
    while shift_pos >= 0 and arr[shift_pos] >= key:
        arr[shift_pos+1] = arr[shift_pos]
        shift_pos -= 1

    ## Insertion
    arr[shift_pos+1] = key

    ### Recursive Case #####################
    recursiveInsertionSort(arr, curr_pass+1)

arr = [1,0,3,6,3,45,2]
recursiveInsertionSort(arr)
arr
```

Out[37]:

```
[0, 1, 2, 3, 3, 6, 45]
```

2.3 Merge Sort

In [38]:

```
## General Steps
def cmp(a, b):
    return a < b # Ascending

def merge(left, right):
    result = []
    l = 0 # Index of left array
    r = 0 # Index of right array
    while l + r < len(left) + len(right):
        bothIndexesInRange = l < len(left) and r < len(right)
        if (bothIndexesInRange and cmp(left[l],right[r])) or \
            r >= len(right):
            result.append(left[l])
            l += 1
        else:
            result.append(right[r])
            r += 1
    return result
```

In [39]:

```
## Recursive

def MergeSort(array):
    # Base case
    if len(array) <= 1: # Already sorted
        return array
    else:
        # Divide
        mid = len(array) // 2
        left = array[:mid]
        right = array[mid:]

        # Sort
        left = MergeSort(left)
        right = MergeSort(right)

        # Merge
        return merge(left, right)
```

In [40]:

```
## Iterative

def MergeSortPass(Array, sz, SortOrder): # Working
    # Sort the smaller subarrays
    index = 0
    while index < len(Array):
        low = index
        mid = index + sz
        high = index + 2 * sz
        left = Array[low:mid]
        right = Array[mid:high]

        result = merge(left, right, SortOrder)
        for index in range(len(result)):
            Array[low + index] = result[index]

        index = high # Sort the next subarray

def MergeSortLoop(Array, SortOrder):
    sz = 1 # Size of subarray to sort
    while sz < len(Array):
        MergeSortPass(Array, sz, SortOrder)
        sz *= 2 # Double the array size
    return Array
```

2.4 Quick Sort

In [41]:

```
# Recursive, in place
def quicksort(arr, s, e):
    if not s<e:
        return
    pivot,l,r = arr[s],s+1,e
    while l <= r:
        while l <= r and arr[l] <= pivot:
            l+=1
        while l <= r and arr[r] >= pivot:
            r-=1
        if l<=r:
            temp = arr[l]
            arr[l] = arr[r]
            arr[r] = temp

    temp = arr[s]
    arr[s] = arr[r]
    arr[r] = temp

    print(arr,s,l,r,e)
    quicksort(arr,s,r-1)
    quicksort(arr,r+1,e)

import random
arr = [random.randint(1,10) for i in range(10)]
og = arr

quicksort(arr, 0, len(arr)-1)
arr == sorted(og)
```

```
[1, 1, 5, 4, 2, 7, 8, 10, 8, 7] 0 6 5 9
[1, 1, 5, 4, 2, 7, 8, 10, 8, 7] 0 2 1 4
[1, 1, 2, 4, 5, 7, 8, 10, 8, 7] 2 5 4 4
[1, 1, 2, 4, 5, 7, 8, 10, 8, 7] 2 3 2 3
[1, 1, 2, 4, 5, 7, 8, 7, 8, 10] 6 9 8 9
[1, 1, 2, 4, 5, 7, 7, 8, 8, 10] 6 8 7 7
```

Out[41]:

```
True
```

In [42]:

```
# Recursive, out of place
def quicksort(arr):
    ## Base Case: Already sorted ##########
    if len(arr) <= 1:
        return arr
    ## Recursive Case ######################
    else:
        # Partitioning
        pivot = arr[0]
        left = []
        right = []
        for item in arr[1:]: # Exclude the first element
            if item < pivot:
                left.append(item)
            else: #if item >= pivot:
                right.append(item)
        ## Recursion Call #################
        return quicksort(left) + [pivot] + quicksort(right)

import random
arr = [random.randint(1,20) for i in range(10)]
quicksort(arr)
```

Out[42]:

```
[1, 2, 2, 3, 3, 11, 12, 12, 15, 18]
```

3. Searching Algorithms

3.1 Linear Search

In [43]:

```
array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
item = 4

def linearSearch(array, item):
    for index in range(len(array)):
        if array[index] == item:
            return item
    return -1 # Cannot find

linearSearch(array, item)
```

Out[43]:

```
4
```

3.2 Binary Search

In [44]:

```
## Iterative
array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
def binarySearch(array, low, high, find):
    while low <= high:
        mid = int((low+high)/2)
        if array[mid] == find:
            return mid
        elif array[mid] > find:
            high = mid - 1
        elif array[mid] < find:
            low = mid + 1
    return -1

binarySearch(array, 0, 9, 2)
```

Out[44]:

```
1
```

In [45]:

```
## Recursive
def recursiveBinarySearch(array, low, high, find):
    ### Base Cases ############################
    if low > high: # Not in array
        return -1

    mid = int((low + high)/2) # Trucate
    if array[mid] == find:
        return mid

    ### Recursive cases #######################
    if find < array[mid]:
        return recursiveBinarySearch(array, low, mid-1, find)
    elif find > array[mid]:
        return recursiveBinarySearch(array, mid+1, high, find)

recursiveBinarySearch(array, 0, 9, 2)
```

Out[45]:

```
1
```

3.3 Hash Table

In [46]:

```
class Node:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None

    def __str__(self):
        return f"{self.key},{self.value}"

    ### For Separate Chaining ###############
    def insert(self, key, value):
        if self.next == None:
            self.next = Node(key, value)
        else:
            self.next.insert(key, value)

    def find(self, key):
        if self.key == key:
            return self
        elif self.next != None:
            return self.next.find(key)
        else:
            return None
```

In [47]:

```
## Open Addressing

class HashTable:
    def __init__(self, max_size=53):
        self.size = max_size
        # Use a python list to simulate an array
        self.array = []
        for i in range(max_size):
            self.array.append(None)

    def get_hash(self, key):
        '''
        # Weighted ordinal hash
        key = str(key)
        hash = 0
        for i in range(len(key)):
            hash += ord(key[i]) * i
        '''
        hash = key
        hash %= self.size
        return hash

    def insert(self, key, data):
        pos = self.get_hash(key)

        # If at empty value
        if self.array[pos] == None:
            self.array[pos] = Node(key,data)
            return 0

        # Open addressing
        original = pos
        pos = original + 1
        while pos != original:
            if self.array[pos] == None:
                self.array[pos] = Node(key,data)
                return 1
            pos = (pos + 1) % self.size # Wrap around

        return -1 # Failed to insert

    def find(self, key):
        pos = self.get_hash(key)

        # If at empty value
        if self.array[pos] != None and self.array[pos].key == key:
            return self.array[pos]

        # Open addressing
        original = pos
        pos = original + 1
        while pos != original:
            if self.array[pos] != None and self.array[pos].key == key:
                return self.array[pos]
            pos = (pos + 1) % self.size # Wrap around

        return -1 # Failed to find
```

```
ht = HashTable()
ht.insert(1, 3)
ht.insert(54, 2)
print(ht.array)
print(ht.find(54))
```

```
[None, <__main__.Node object at 0x00000184E10A16A0>, <__main__.Node object
at 0x00000184E10A1780>, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, Non
e, None, None, None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None]
54,2
```

In [48]:

```
## Separate Chaining

# Uses the previous node class
class HashTable:
    def __init__(self, max_size=53):
        self.size = max_size
        # Use a python list to simulate an array
        self.array = []
        for i in range(max_size):
            self.array.append(None)

    def get_hash(self, key):
        '''
        # Weighted ordinal hash
        key = str(key)
        hash = 0
        for i in range(len(key)):
            hash += ord(key[i]) * i
        '''
        hash = key
        hash %= self.size
        return hash

    def insert(self, key, data):
        pos = self.get_hash(key)

        # If at empty value
        if self.array[pos] == None:
            self.array[pos] = Node(key,data)
        # Open addressing
        else:
            self.array[pos].insert(key, data)

    def find(self, key):
        pos = self.get_hash(key)

        # If at empty value
        if self.array[pos] != None and self.array[pos].key == key:
            return self.array[pos]

        # Separate Chaining
        return self.array[pos].find(key)


ht = HashTable()
ht.insert(1, 3)
ht.insert(54, 2)
print(ht.array)
print(ht.find(54))
```

```
[None, <__main__.Node object at 0x00000184E10A1D30>, None, None, None, Non
e, None, None, None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, Non
e, None, None, None, None, None, None, None, None, None]
54,2
```

4. Data Structures

4.1 Linked Lists

There are a few main approaches

1. Pointer implementation
2. Array Implementation

In [49]:

```
# Pointer Implementation
class Node:
    def __init__(self, value=None,next=None):
        self.value = value
        self.next = next

class LinkedList:
    def __init__(self):
        self.start = None

    def isEmpty(self):
        return self.start == None

    def display(self):
        output = ""
        # Traversal
        curr = self.start
        while curr != None:
            output += str(curr.value)
            curr = curr.next
            if curr != None:
                output += ", "
        print(output)

    ### General Traversal Methods ###################################
    def travelToNode(self, reachedFunc):
        counter = 0
        prev = None
        curr = self.start
        while curr != None and (not reachedFunc(counter, curr)):
            prev = curr
            curr = curr.next
            counter += 1

        return prev, curr, counter

    def get(self, reachedFunc):
        prev, curr, counter = self.travelToNode(reachedFunc)
        return curr

    def insert(self, reachedFunc, value):
        prev, curr, counter = self.travelToNode(reachedFunc)

        # Insertion
        newNode = Node(value,curr)
        if prev == None: # Link at start
            self.start = newNode
        else: # Link in between
            prev.next = newNode

    def remove(self, reachedFunc):
        prev, curr, counter = self.travelToNode(reachedFunc)
        if prev == None: # Link at start
            self.start = curr.next
        else: # Remove in between nodes/ at the end
            prev.next = curr.next

    ### Specific types ##############################################
    def getPosition(self, pos):
```

```
        def reachedFunc(currPos, currNode):
            return currPos == pos
        return self.get(reachedFunc)

    def insertPosition(self, pos, val):
        def reachedFunc(currPos, currNode):
            return currPos == pos
        return self.insert(reachedFunc, val)

    def removePosition(self, pos):
        def reachedFunc(currPos, currNode):
            return currPos == pos
        return self.remove(reachedFunc)

    def insertSorted(self, val):
        def reachedFunc(currPos, currNode):
            return currNode.value >= val
        return self.insert(reachedFunc, val)

def driver():
    l = LinkedList()
    print(l.isEmpty())
    l.insertPosition(10,2)
    l.insertSorted(1)
    l.insertPosition(10,3)
    l.insertPosition(10,4)
    l.insertPosition(10,5)
    l.display()
    print(l.isEmpty())
    print(l.getPosition(3).value)
    l.removePosition(2)
    l.display()

if __name__ == "__main__":
    driver()
```

```
True
1, 2, 3, 4, 5
False
4
1, 2, 4, 5
```

In [50]:

```
# Array Implementation
class ListNode:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

    def isEmpty(self):
        return self.data == None

    def __str__(self):
        return f"Node({self.data}, {self.next})"

class ArrayLinkedList:
    def __init__(self, size=10):
        self.initialise(size)

    def initialise(self, size=10):
        # Use a python list to simulate an array of Nodes
        self.size = size
        self.array = []
        for i in range(self.size + 1): # 1-indexed. 0-indexed goes unused
            next = (i + 1) % (self.size + 1)
            node = ListNode(None, next)
            self.array.append(node)

        # Others
        self.start = 0 # Pointer to First element in linked list
        self.end = 1 # Pointer to Last element in linked list
        self.nextFree = 1 # Pointer to First element in free list

    ### Auxilary Functions ###########################################################
    def isEmpty(self):
        return self.start == 0

    def isFull(self):
        # Check for unused nodes
        return self.nextFree == 0

    def printStructure(self):
        output = ""
        # Pointers
        output += f"Start Pointer = {self.start}\n"
        output += f"End Pointer = {self.end}\n"
        output += f"nextFree Pointer = {self.nextFree}\n"
        # Array Contents
        output += f"{'Index':5} {'Content':20} {'Next':4}\n"
        for index in range(1,len(self.array)):
            content = str(self.array[index].data)
            nextPointer = self.array[index].next
            output += f"{index:5} {content:20} {nextPointer:4}" + "\n"
        print(output[:-1])

    ### Generic Traversal ############################################################
    def display(self):
        output = "Items in order:"
        prevPos = None
        currPos = self.start
        currNode = self.array[currPos]
        while not currNode.isEmpty():
```

```
            output += f'\n{currNode.data}'
            prevPos = currPos
            currPos = currNode.next
            currNode = self.array[currPos]

        print(output)

    def travelToNode(self, reachedFunc):
        counter = 0
        prevPos = None
        currPos = self.start
        currNode = self.array[currPos]
        while not currNode.isEmpty() and not reachedFunc(counter, currPos, currNode):
            prevPos = currPos
            currPos = currNode.next
            currNode = self.array[currPos]
            counter += 1

        if prevPos != None:
            prevNode = self.array[prevPos]
        else:
            prevNode = None

        return prevPos, prevNode, currPos, currNode, counter

    ### Generic Traversal Methods ###############################################
    def get(self, reachedFunc):
        prevPos, prevNode, currPos, currNode, counter = self.travelToNode(reachedFunc)
        return currNode

    def insert(self, reachedFunc, item):
        ### Mandatory checks ####################################
        if self.isFull():
            raise Exception("No space left to add new node!")

        ### Traversal ###########################################
        prevPos, prevNode, currPos, currNode, counter = self.travelToNode(reachedFunc)

        ### Create new node #####################################
        newNodePos = self.nextFree
        newNode = self.array[newNodePos]
        newNode.data = item
        self.nextFree = newNode.next # Get nextFree

        ### Linking #############################################
        ### ---[prev]---[new]---[curr]--- ###################
        # Inserting at the front
        if prevPos == None:
            self.start = newNodePos
        # Linking from previous node
        else:
            prevNode.next = newNodePos

        # Inserting at the back
        if currNode.isEmpty():
            self.end = newNodePos
        # Linking to next node
        newNode.next = currPos

    def remove(self, reachedFunc):
        ### Mandatory checks ####################################
```

```
        if self.isEmpty():
            raise Exception("Nothing to remove!")

        ### Traversal ########################################
        prevPos, prevNode, currPos, currNode, counter = self.travelToNode(reachedFunc)
        nextPos = currNode.next
        nextNode = self.array[nextPos]

        ### Freeing up curr node #############################
        currNode.next = self.nextFree
        currNode.data = None
        self.nextFree = currPos

        ### Unlinking ########################################
        ## ---[prev]---[curr]---[next]--- ####################
        if prevNode == None:
            self.start = nextPos
        else:
            prevNode.next = nextPos

        # Linking to end (if remove to end)
        if currNode.isEmpty():
            self.end = prevPos

    ### Specific types ########################################
    def getPosition(self, pos):
        def reachedFunc(currPos, arrayPos, currNode):
            return currPos == pos
        return self.get(reachedFunc)

    def insertPosition(self, pos, val):
        def reachedFunc(currPos, arrayPos, currNode):
            return currPos == pos
        return self.insert(reachedFunc, val)

    def removePosition(self, pos):
        def reachedFunc(currPos, arrayPos, currNode):
            return currPos == pos
        return self.remove(reachedFunc)

    def insertSorted(self, val):
        def reachedFunc(currPos, arrayPos, currNode):
            return currNode.data >= val
        return self.insert(reachedFunc, val)

### Driver ###############################################
def driver():
    l = ArrayLinkedList(5)
    print("Is Empty:", l.isEmpty())
    print("### Addition of Items ##############################")
    l.insertPosition(10,2)
    l.insertSorted(1)
    l.insertPosition(10, 5)
    l.insertSorted(3)
    l.insertSorted(4)
    l.printStructure()
    l.display()
    print()
    print("Is Empty:", l.isEmpty())
    print("Is Full:", l.isFull())
    print('Data at position 3:', l.getPosition(3).data)
```

```
    print("### Removal of Items ##############################")
    l.removePosition(2)
    l.removePosition(1)
    #l.removePosition(0)
    #l.removePosition(0)
    #l.removePosition(0)
    l.printStructure()
    l.display()
    print("Is Empty:", l.isEmpty())
    print("Is Full:", l.isFull())

if __name__ == "__main__":
    driver()
```

```
Is Empty: True
### Addition of Items ##############################
Start Pointer = 2
End Pointer = 3
nextFree Pointer = 0
Index Content               Next
    1 2                        4
    2 1                        1
    3 5                        0
    4 3                        5
    5 4                        3
Items in order:
1
2
3
4
5

Is Empty: False
Is Full: True
Data at position 3: 4
### Removal of Items ##############################
Start Pointer = 2
End Pointer = 2
nextFree Pointer = 1
Index Content               Next
    1 None                     4
    2 1                        5
    3 5                        0
    4 None                     0
    5 4                        3
Items in order:
1
4
5
Is Empty: False
Is Full: False
```

4.2 Stacks

There are a few main approaches

1. Linked List
2. Array
3. Python List

In [51]:

```
# Driver
def driver(stack, size):
    print("TRUE SHOWS THAT IT WORKS!!!")

    print("---Initialised Stack---")
    print("size()", stack.size() == 0)
    print("isEmptyStack()",stack.isEmpty() == True)
    print("isFullStack()",stack.isFull() == False)

    print("---Appending Elements---")
    for i in range(size):
        stack.push(i)
    print("isEmptyStack()",stack.isEmpty() == False)
    print("isFullStack()",stack.isFull() == True)
    for i in range(size - 1, 0 - 1, -1):
          print("peek()", stack.peek() == i,
                "isEmptyStack()",stack.isEmpty() == False,
                "isFullStack()",stack.isFull() ==  (i == size-1),
                "pop()", stack.pop() == i)

    print("---Emptied Elements---")
    print("isEmptyStack()",stack.isEmpty() == True)
    print("isFullStack()",stack.isFull() == False)
```

In [52]:

```
# Pointer Implementation
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class PointerStack:
    def __init__(self):
        self.head = None
        self.current_size = 0

    def push(self, data):
        self.current_size += 1
        if self.head is None:
            self.head = Node(data)
        else:
            new_node = Node(data)
            new_node.next = self.head
            self.head = new_node

    def pop(self):
        if self.head is None:
            return None
        else:
            self.current_size -= 1
            popped = self.head.data
            self.head = self.head.next
            return popped

    def peek(self):
        if self.head is None:
            return None
        else:
            popped = self.head.data
            return popped

    ### Auxilary Functions ##############################################
    def size(self):
        return self.current_size

    def isEmpty(self):
        return self.size() == 0

    def isFull(self):
        return False # Not implemented

driver(PointerStack(), 5)
```

```
TRUE SHOWS THAT IT WORKS!!!
---Initialised Stack---
size() True
isEmptyStack() True
isFullStack() True
---Appending Elements---
isEmptyStack() True
isFullStack() False
peek() True isEmptyStack() True isFullStack() False pop() True
peek() True isEmptyStack() True isFullStack() True pop() True
peek() True isEmptyStack() True isFullStack() True pop() True
peek() True isEmptyStack() True isFullStack() True pop() True
peek() True isEmptyStack() True isFullStack() True pop() True
---Emptied Elements---
isEmptyStack() True
isFullStack() True
```

In [53]:

```
# Array Implementation
class ArrayStack:
    def __init__(self, size=30):
        self.MAX_SIZE = size
        # Simulate an array in Python
        self.array = []
        for index in range(self.MAX_SIZE):
            self.array.append(None)
        self.head = 0 # The first empty index

    def push(self, data):
        if self.isFull():
            print("Error: the stack is at full capacity!")
        else:
            self.array[self.head] = data
            self.head += 1

    def pop(self):
        if self.isEmpty():
            return None
        else:
            self.head -= 1
            return self.array[self.head]

    def peek(self):
        if self.isEmpty():
            return None
        else:
            return self.array[self.head - 1]

    ### Auxilary Functions ##############################################
    def size(self):
        return self.head

    def isEmpty(self):
        return self.size() == 0

    def isFull(self):
        return self.size() >= self.MAX_SIZE

driver(ArrayStack(5), 5)
```

```
TRUE SHOWS THAT IT WORKS!!!
---Initialised Stack---
size() True
isEmptyStack() True
isFullStack() True
---Appending Elements---
isEmptyStack() True
isFullStack() True
peek() True isEmptyStack() True isFullStack() True pop() True
peek() True isEmptyStack() True isFullStack() True pop() True
peek() True isEmptyStack() True isFullStack() True pop() True
peek() True isEmptyStack() True isFullStack() True pop() True
peek() True isEmptyStack() True isFullStack() True pop() True
---Emptied Elements---
isEmptyStack() True
isFullStack() True
```

In [54]:

```
# Python List Implementation
class PythonListStack:
    def __init__(self):
        self.items = []

    def push(self, data):
        self.items.append(data)

    def pop(self):
        if self.isEmpty():
            return None
        else:
            return self.items.pop()

    def peek(self):
        if self.isEmpty():
            return None
        else:
            return self.items[-1]

    ### Auxilary Functions ##################################
    def size(self):
        return len(self.items)

    def isEmpty(self):
        return self.size() == 0

    def isFull(self):
        return False

driver(PythonListStack(), 5)
```

```
TRUE SHOWS THAT IT WORKS!!!
---Initialised Stack---
size() True
isEmptyStack() True
isFullStack() True
---Appending Elements---
isEmptyStack() True
isFullStack() False
peek() True isEmptyStack() True isFullStack() False pop() True
peek() True isEmptyStack() True isFullStack() True pop() True
peek() True isEmptyStack() True isFullStack() True pop() True
peek() True isEmptyStack() True isFullStack() True pop() True
peek() True isEmptyStack() True isFullStack() True pop() True
---Emptied Elements---
isEmptyStack() True
isFullStack() True
```

4.2 Stack Application: Infix, Prefix and Postfix Conversion and Evaluation

Inspired from here (https://github.com/Hackin7/Programming-Crappy-Solutions/blob/master/Basic%20Exercises/NYJC%20A%20Level%20Computing%202019-2020/Practicals/Practical%2015/Q3_Chan%20Zun%20Mun%20Terence.ipynb)

In [55]:

```
Stack = ArrayStack
```

Convert Infix to Postfix (include order of precedence)

Using a stack, write a function make_postfix(expr) which takes in a math expression in infix form and returns the expression in postfix format.

Example, infix expression (1 + 2) *3 can be converted to the postfix expression 1 2 + 3*

Steps to convert from infix to postfix (include order of precedence):

Read in the tokens one at a time

1. If a token is an integer, write it into the output
2. If a token is a left parentheses '(', push it to the stack
3. If a token is an operator, push it to the stack. However, if the stack has operators with higher or equal priority, pop and write them to the output.
4. If a token is a right parentheses ')', you pop the stack and write entries to output until you meet '('. Do not write brackets to output as postfix has no brackets. When you finish reading the string, you pop out all tokens which are left in the stack and write them to output. Hint: Use the string methods split() and isdigit()

In [56]:

```
## WARNING: MAY NOT BE CORRECT ############################
def make_postfix_adv(infix):
    # Variable initialisation
    tokens = infix.split()
    output = ""
    stack = Stack(len(tokens))

    # Priority of operator in stack
    stackPriority = 3 #Represents no priority

    #Constants
    spacing = " "
    opLevel = {"+" : 2, "-" : 2, "*" : 1, "/" : 1} # Descending
    operators = opLevel.keys()

    # Read tokens one at a time
    for i in tokens:

        ### Number ###############################################
        if i.isdigit():
            output += i + spacing

        ### Paranthesis ##################################
        elif i == "(":
            stack.push(i)
        elif i == ")": # Empty all in brackets
            while not stack.isEmpty() and stack.peek() != "(":
                output += stack.pop() + spacing
            stack.pop() # Remove last paranthesis

        ### Operators without precedence
        #elif i in operators:
        #    stack.push(i)
        ### Operators and Precedence ####################
        elif i in operators:
            #### Higher Level Operators ##############
            '''
            I Suspect this code may have an issue, but its not really important for A L
evels yet
            '''
            while not stack.isEmpty() and \
                stack.peek() != "(" and stackPriority <= opLevel[i]: # Stack has higher
precedence
                    ### Evalutate higher priority first
                    output += stack.pop() + spacing
                    ### Get new stack priority
                    if opLevel.get(stack.peek()) != None:
                        stackPriority = opLevel.get(stack.peek())
            #### Current Operators ##################
            stack.push(i) # Use current operator
            stackPriority = opLevel[i] # Current Priority

    # When you finish reading the string, you pop out all tokens
    # which are left in the stack and write them to output.
    while not stack.isEmpty():
        output += stack.pop() + spacing

    if len(output) > 0:
```

```
        output = output[:-1] #Remove empty space
    return output
```

In [57]:

```
def test_case(case, expected):
  print(f"{case:<25} | {expected:<25} | {make_postfix_adv(case):<25} | \
{'True' if make_postfix_adv(case) == expected else 'False':<4}")

print(f"{'Case':<25} | {'Expected':<25} | {'Output':<25} | {'Correct? ':<4}")
print("--- Without Precedence " + "-" * 70)
test_case("1 + 2", "1 2 +")
test_case("( 1 + 2 ) * 3" ,"1 2 + 3 *")
test_case("( 1 + 2 ) * ( 3 + 4 )", "1 2 + 3 4 + *")
test_case("12 + 34", "12 34 +")
print("--- With Precedence " + "-" * 73)
test_case("5 * 6 + 7 * 8", "5 6 * 7 8 * +")
test_case("5 + 6 * 7 + 8" , "5 6 7 * + 8 +")
test_case("( 12 + 23 ) - 6 / 2 ", "12 23 + 6 2 / -")
test_case("( 5 * 6 + 7 * 8 ) + 8", "5 6 * 7 8 * + 8 +")
```

```
Case                      | Expected                  | Output
| Correct?
--- Without Precedence ----------------------------------------------------
------------------
1 + 2                     | 1 2 +                     | 1 2 +
| True
( 1 + 2 ) * 3             | 1 2 + 3 *                 | 1 2 + 3 *
| True
( 1 + 2 ) * ( 3 + 4 )     | 1 2 + 3 4 + *             | 1 2 + 3 4 + *
| True
12 + 34                   | 12 34 +                   | 12 34 +
| True
--- With Precedence -------------------------------------------------------
------------------
5 * 6 + 7 * 8             | 5 6 * 7 8 * +             | 5 6 * 7 8 * +
| True
5 + 6 * 7 + 8             | 5 6 7 * + 8 +             | 5 6 7 * + 8 +
| True
( 12 + 23 ) - 6 / 2       | 12 23 + 6 2 / -           | 12 23 + 6 2 / -
| True
( 5 * 6 + 7 * 8 ) + 8     | 5 6 * 7 8 * + 8 +         | 5 6 * 7 8 * + 8 +
| True
```

Evaluating a Postfix Expression

An application of Stack is to calculate postfix form expressions. The calculate function takes in a math expression in postfix form and returns the result of the expression as an integer (the expression is guaranteed to evaluate to an integer). The format of the input string is the same as the output in last question. For example, 1 2 + 3 *.

Steps to parse and evaluate a postfix expression.

1. We read the tokens in one at a time.
2. If it is an integer, push it on the stack
3. If it is a binary operator, pop the top two elements from the stack, apply the operator, and push the result back on the stack.

In [58]:

```
def calculate(postfix):
    # Variable initialisation
    tokens = postfix.split() # Split by spaces
    output = 0
    stack = Stack(len(tokens))

    # Constants
    operators = ["+", "-", "*", "/"]
    for i in tokens:
        if i.isdigit():
            stack.push(int(i))
        elif i in operators:
            val2 = stack.pop() # last element put in first
            val1 = stack.pop()
            ### Do The Math ##############################
            if i == "+":
                valTotal = val1 + val2
            elif i == "-":
                valTotal = val1 - val2
            elif i == "*":
                valTotal = val1 * val2
            elif i == "/":
                valTotal = val1 / val2
            stack.push(valTotal)
            ##############################################
    # Last Element is output
    output = stack.peek()
    return output
```

In [59]:

```
def test_case(case, expected):
  print(f"{case:<25} | {expected:<25} | {calculate(case):<25} | \
{'True' if calculate(case) == expected else 'False':<4}")

print(f"{'Case':<25} | {'Expected':<25} | {'Output':<25} | {'Correct? ':<4}")
print(f"{'-'*25:<25} | {'-'*25:<25} | {'-'*25:<25} | {'-'*4:<4}")

test_case("1 2 +", 3)
test_case("5 6 * 7 8 * +", 86)
test_case("5 6 7 * + 8 +", 55)
test_case("12 23 + 6 2 / -", 32)
```

```
Case                      | Expected                  | Output
| Correct?
------------------------- | ------------------------- | -----------------
------- | ----
1 2 +                     | 3                         | 3
| True
5 6 * 7 8 * +             | 86                        | 86
| True
5 6 7 * + 8 +             | 55                        | 55
| True
12 23 + 6 2 / -           | 32                        | 32.0
| True
```

4.3 Queues

There are a few main approaches

1. Pointer Implementation
2. Dual Stack Implementation
3. Array (Circular Queue) Implementation
4. Python List Implementation

In [60]:

```
# Driver
def driver(q):
    q.enqueue(1)
    q.enqueue(2)
    print("Is Empty:", q.isEmpty())
    print("Is Full:", q.isFull())
    print("Size:", q.size())

    print(q.dequeue())
    print(q.dequeue())
    print(q.dequeue())

    print("Is Empty:", q.isEmpty())
    print("Is Full:", q.isFull())
    print("Size:", q.size())

    q.enqueue(3)
    q.enqueue(4)
    print(q.dequeue())

    print("Is Empty:", q.isEmpty())
    print("Is Full:", q.isFull())
    print("Size:", q.size())
```

In [61]:

```
# Linked List Implementation
class Node:
    def __init__(self, data, next=None):
        self.data = data
        self.next = next

class PointerQueue:
    def __init__(self):
        self.head = None
        self.tail = None
        self.currSize = 0

    def enqueue(self, data):
        newNode = Node(data, None)
        # No node
        if self.isEmpty():
            self.head = newNode
            self.tail = newNode
        else:
            self.tail.next = newNode
        self.currSize += 1

    def dequeue(self):
        if self.isEmpty():
            return None
        else:
            self.currSize -= 1
            data = self.head.data
            self.head = self.head.next
            return data

    ### Auxilary Functions #############################################
    def size(self):
        return self.currSize

    def isEmpty(self):
        return self.head == None

    def isFull(self):
        return False


driver(PointerQueue())
```

```
Is Empty: False
Is Full: False
Size: 2
1
2
None
Is Empty: True
Is Full: False
Size: 0
3
Is Empty: False
Is Full: False
Size: 1
```

In [62]:

```
# Dual Stack Implementation
class DualStackQueue:
    def __init__(self):
        self.inbox = Stack()
        self.outbox = Stack()
        self.currSize = 0

    def enqueue(self, data):
        self.inbox.push(data)
        self.currSize += 1

    def dequeue(self):
        if self.outbox.isEmpty():
            while not self.inbox.isEmpty():
                popped = self.inbox.pop()
                self.outbox.push(popped)
        item = self.outbox.pop()
        if item != None:
            self.currSize -= 1
        return item

    ### Auxilary Functions ##############################################
    def size(self):
        return self.currSize

    def isEmpty(self):
        return (self.inbox.isEmpty() and self.outbox.isEmpty())

    def isFull(self):
        return False # not implemented

Stack = PythonListStack
driver(DualStackQueue())
```

```
Is Empty: False
Is Full: False
Size: 2
1
2
None
Is Empty: True
Is Full: False
Size: 0
3
Is Empty: False
Is Full: False
Size: 1
```

In [63]:

```
# Array Circular Queue Implementation
class CircularQueue:
    #Constructor
    def __init__(self):
        # Use a python list to simulate an array
        self.maxSize = 2
        self.array = []
        for i in range(self.maxSize): # 0-indexed
            self.array.append(None)
        # Other pointers
        self.head = 0
        self.tail = 0

        self.currSize = 0

    # Adding elements to the queue
    def enqueue(self, data):
        if self.isFull():
            return ("Queue Full!")
        self.array[self.tail] = data
        self.tail = (self.tail + 1) % (self.maxSize)# + 1)
        self.currSize += 1
        return True

    # Removing elements from the queue
    def dequeue(self):
        if self.isEmpty():
            return ("Queue Empty!")
        data = self.array[self.head]
        self.array[self.head] = None
        self.head = (self.head + 1) % (self.maxSize)# + 1)
        self.currSize -= 1
        return data

    ### Auxilary Functions #############################################
    # Calculating the size of the queue
    def size(self):
        return self.currSize
        '''
        if self.tail >= self.head:
            return (self.tail - self.head)
        return (self.maxSize - 1 - (self.head - self.tail))
        '''


    def isEmpty(self):
        return self.size() == 0

    def isFull(self):
        return self.size() == self.maxSize

q = CircularQueue()
driver(q)
print(q.array)
```

```
Is Empty: False
Is Full: True
Size: 2
1
2
Queue Empty!
Is Empty: True
Is Full: False
Size: 0
3
Is Empty: False
Is Full: False
Size: 1
[None, 4]
```

In [64]:

```
# Python List Implementation
class PythonListQueue:
    def __init__(self):
        self.items = []

    def enqueue(self, data):
        self.items.append(data)

    def dequeue(self):
        if self.isEmpty():
            return None
        else:
            return self.items.pop(0)

    def peek(self):
        if self.isEmpty():
            return None
        else:
            return self.items[0]

    ### Auxilary Functions ##############################################
    def size(self):
        return len(self.items)

    def isEmpty(self):
        return self.size() == 0

    def isFull(self):
        return False

driver(PythonListQueue())
```

```
Is Empty: False
Is Full: False
Size: 2
1
2
None
Is Empty: True
Is Full: False
Size: 0
3
Is Empty: False
Is Full: False
Size: 1
```

4.4 Binary Search Trees

There are a few main approaches

1. Pointer Implementation
2. Array Implementation

In [65]:

```
# Pointer Implementation
class BSTNode:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
        self.parent = None

    def insert(self, node):
        if self.key > node.key:
            if self.left is None:
                self.left = node
                node.parent = self
            else:
                self.left.insert(node)
        elif self.key < node.key:
            if self.right is None:
                self.right = node
                node.parent = self
            else:
                self.right.insert(node)

    ### Search ##################################
    def search(self, key):
        if self.key > key:
            if self.left is not None:
                return self.left.search(key)
            else:
                return None
        elif self.key < key:
            if self.right is not None:
                return self.right.search(key)
            else:
                return None
        return self
    ### Orderings ###############################
    def inorder(self):
        if self.left is not None:
            self.left.inorder()

        print(self.key, end=',')

        if self.right is not None:
            self.right.inorder()

    def preorder(self):
        print(self.key, end=',')

        if self.left is not None:
            self.left.inorder()

        if self.right is not None:
            self.right.inorder()

    def postorder(self):
        if self.left is not None:
            self.left.inorder()

        if self.right is not None:
```

```
            self.right.inorder()

        print(self.key, end=',')

    ### Other ############################
    def findKSmallest(self, K):
        remainingK = K

        ### Left Branch ######################
        if self.left is not None:
            remainingK, value = self.left.findKSmallest(remainingK)
            if remainingK <= 0:
                return remainingK, value

        ### Current Node #####################
        remainingK -= 1 # Current node
        if remainingK <= 0:
            return remainingK, self.key

        ### Right Branch #####################
        if self.right is not None:
            remainingK, value = self.right.findKSmallest(remainingK)
            if remainingK <= 1:
                return remainingK, value

        ### Still has K ######################
        return remainingK, None



class PointerBinarySearchTree:
    def __init__(self):
        self.root = None

    def inOrderTraversal(self):
        if self.root is not None:
            self.root.inorder()

    def findKSmallest(self, K):
        if self.root is not None:
            remainingK, value = self.root.findKSmallest(K)
            return value
        else:
            return None

    def add(self, key):
        new_node = BSTNode(key)
        if self.root is None:
            self.root = new_node
        else:
            self.root.insert(new_node)

    def search(self):
        if self.root is not None:
            return self.root.search()

    def isEmpty(self):
        return self.root == None

### Driver ###############################################################
#####
```

```python
def driver(addList=[]):
    bintree = PointerBinarySearchTree()

    print("isEmpty() =",bintree.isEmpty())
    print()

    # Add Item to List
    print("Adding:")
    for item in addList:
        print(item)
        bintree.add(item)

    # Output
    print()
    bintree.inOrderTraversal()
    print()
    print("isEmpty() =",bintree.isEmpty())
    for i in range(1, 7 + 1):
        print(f"findKSmallest({i}) =", bintree.findKSmallest(i))

driver(['INDIA', 'NEPAL', 'MALAYSIA', 'SINGAPORE', 'BURMA', 'CANADA', 'LATIVA'])
```

```
isEmpty() = True

Adding:
INDIA
NEPAL
MALAYSIA
SINGAPORE
BURMA
CANADA
LATIVA

BURMA,CANADA,INDIA,LATIVA,MALAYSIA,NEPAL,SINGAPORE,
isEmpty() = False
findKSmallest(1) = BURMA
findKSmallest(2) = CANADA
findKSmallest(3) = INDIA
findKSmallest(4) = LATIVA
findKSmallest(5) = MALAYSIA
findKSmallest(6) = NEPAL
findKSmallest(7) = SINGAPORE
```

In [66]:

```
# Array Implementation
class BSTNode:
    def __init__(self, left=0, Data='', right=0):
        self.left = int(left)
        self.data = Data
        self.right = int(right)

    def isEmpty(self):
        return self.data == ''

class ArrayBinarySearchTree:
    def __init__(self, size=10):
        self.initialise(size)

    def initialise(self, size=10):
        # Use a python list to simulate an array of Nodes
        self.size = size
        self.array = []
        for i in range(self.size + 1): # 1-indexed. 0-indexed goes unused
            nextPos = (i + 1) % (self.size + 1)
            node = BSTNode(nextPos, '', 0)
            self.array.append(node)

        # Others
        self.root = 0 # Pointer to root of BST
        self.nextFree = 1 # Pointer to First element in free list (Linked list of remai
ning nodes)

    ### Auxilary Functions ###################################################
    def isEmpty(self):
        return self.root == 0

    def isFull(self):
        # Check for unused nodes
        return self.nextFree == 0

    def printStructure(self):
        output = ""
        # Pointers
        output += f"root = {self.root}"
        output += f"\n"
        output += f"nextFreePosition = {self.nextFree}"

        # Array
        output += f"\n"
        output += f"array: "
        output += f"\n"
        output += f"{'Index':5} {'left':5} {'data':10} {'right':6}"
        for index in range(1, len(self.array)):
            node = self.array[index]
            output += f"\n"
            output += f"{index:5} {node.left:5} {node.data:10} {node.right:6}"
        print(output)

    ### Generic Traversal ####################################################
    def inOrderTraversal(self, currPos=None):
        if currPos == None:
            currPos = self.root
        ## Base Case: Empty
```

```
        if currPos == 0:
            return
        ## Case 2: Not empty
        node = self.array[currPos]
        # In order
        self.inOrderTraversal(node.left)
        print(node.data)
        self.inOrderTraversal(node.right)

        '''
        # Pre-order
        print(node.data)
        self.inOrderTraversal(node.left)
        self.inOrderTraversal(node.right)

        # Post-order
        self.inOrderTraversal(node.left)
        self.inOrderTraversal(node.right)
        print(node.data)
        '''

    ### Basic Functions ###################################################
    def addNodeOnly(self, item):
        ### Create new node ##################################
        newNodePos = self.nextFree
        newNode = self.array[newNodePos]
        newNode.data = item
        self.nextFree = newNode.left # Get nextFree
        newNode.left = 0
        newNode.right = 0
        return newNodePos

    def addRecursive(self, item, currPos=None):
        if currPos == None:
            currPos = self.root

        node = self.array[currPos]

        ### Case 1: Root does not have anything ##########################
        if currPos == 0:
            newNodePos = self.addNodeOnly(item) # Create new node
            self.root = newNodePos
            return

        ### Case 2: Root has element #####################################
        # Insert to left
        if item < node.data:
            if node.left == 0:
                newNodePos = self.addNodeOnly(item) # Create new node
                node.left = newNodePos # Insert Position
            else:
                self.addRecursive(item, node.left)
        elif item >= node.data:
            if node.right == 0: #Empty
                newNodePos = self.addNodeOnly(item) # Create new node
                node.right = newNodePos # Insert Position
            else:
                self.addRecursive(item, node.right)

    def addIterative(self, item):
        ### Create new node ##################################
```

```
        newNodePos = self.addNodeOnly(item)

        ### Root position ###################################
        if self.array[self.root].isEmpty():
            self.root = newNodePos
            return

        ### Binary Search for Position ######################
        prevPos = None
        prevNode = None
        currPos = self.root
        currNode = self.array[currPos]
        lastMove = 'X'
        while not currPos == 0: # and currNode.data != item:
            prevPos = currPos
            prevNode = currNode
            if item < currNode.data:
                currPos = currNode.left
                lastMove = 'L'
            elif item >= currNode.data:
                currPos = currNode.right
                lastMove = 'R'
            currNode = self.array[currPos]

        ### Linking #########################################
        if lastMove == 'L':
            prevNode.left = newNodePos
        else:
            prevNode.right = newNodePos


### Driver ###############################################################################
#####
def driver(addList=[]):
    bintree = ArrayBinarySearchTree(len(addList))

    print("isEmpty() =",bintree.isEmpty())
    print("isFull() =",bintree.isFull())
    print()

    # Add Item to list
    print("Adding:")
    for item in addList:
        print(item)
        bintree.addRecursive(item)
        #bintree.addIterative(item)

    # Output
    print()
    bintree.printStructure()
    bintree.inOrderTraversal()
    print()
    print("isEmpty() =",bintree.isEmpty())
    print("isFull() =",bintree.isFull())

driver(['INDIA', 'NEPAL', 'MALAYSIA', 'SINGAPORE', 'BURMA', 'CANADA', 'LATIVA'])
```

```
isEmpty() = True
isFull() = False

Adding:
INDIA
NEPAL
MALAYSIA
SINGAPORE
BURMA
CANADA
LATIVA

root = 1
nextFreePosition = 0
array:
Index left  data       right
    1     5 INDIA           2
    2     3 NEPAL           4
    3     7 MALAYSIA        0
    4     0 SINGAPORE       0
    5     0 BURMA           6
    6     0 CANADA          0
    7     0 LATIVA          0
BURMA
CANADA
INDIA
LATIVA
MALAYSIA
NEPAL
SINGAPORE

isEmpty() = False
isFull() = True
```

5. Databases

5.1 SQL

Overall

In [67]:

```
import sqlite3
conn = sqlite3.connect("test.db")

### Deletion ##################################################################
try:
    print("Clearing DB with `DROP TABLE <TableName>`")
    conn.execute("DROP TABLE ReferenceTable;")
    conn.execute("DROP TABLE Data;")
except:
    print("DB had nothing to clear")
```

```
Clearing DB with `DROP TABLE <TableName>`
DB had nothing to clear
```

In [68]:

```
### Creation ##################################################################
creation = [
    "CREATE TABLE ReferenceTable (ID INTEGER PRIMARY KEY, Description text);",
    '''
    CREATE TABLE Data(
        ID INTEGER PRIMARY KEY ,
        Description TEXT NOT NULL,
        Number NUMBER,
        DataLink INTEGER );
    '''
]
print()
print("Creating Table with:")
for c in creation:
    print(c)
    conn.execute(c)
```

```
Creating Table with:
CREATE TABLE ReferenceTable (ID INTEGER PRIMARY KEY, Description text);

    CREATE TABLE Data(
        ID INTEGER PRIMARY KEY ,
        Description TEXT NOT NULL,
        Number NUMBER,
        DataLink INTEGER );
```

In [69]:

```
### Insertion ################################################################
print("Inserting values with `INSERT INTO ReferenceTable(ID, Description) VALUES(?,?)`
 and `INSERT INTO Data(ID, Description, Number, DataLink) VALUES(?,?,?,?)`")

reference_header = ["ID", "Description"]
reference = [
    (1, "ref 1"),
    (2, "ref 2")
]
for row in reference:
    conn.execute("INSERT INTO ReferenceTable(ID, Description) VALUES(?,?)",row)

data_header = ["ID", "Description", "Number"]
data = [
    (1, "hi", 3.1, 1),
    (2, "hello", 4, 2),
    (3, "Join test", 4, None)
]
for row in data:
    conn.execute("INSERT INTO Data(ID, Description, Number, DataLink) VALUES(?,?,?,?)",
row)
```

```
Inserting values with `INSERT INTO ReferenceTable(ID, Description) VALUES
(?,?)` and `INSERT INTO Data(ID, Description, Number, DataLink) VALUES
(?,?,?,?)`
```

In [70]:

```
### Query ############################################################
def nice_show_in_query(query=None, header=[0,1,2,3], colwidth=None):
    output = ""
    ### Header ###################################
    if colwidth == None:
        colwidth = [13 for i in range(len(header))]
    for j in range(len(header)):
        output+=f"{header[j]:<{colwidth[j]}} "
    output += "\n"

    ### Border ###################################
    border = ""
    for j in range(len(header)):
        border += "-" * (colwidth[j]+1)
    border += "\n"
    output = border+output+border

    ### GETTING VALUES ###########################################
    cursor = conn.execute(query)
    for item in cursor.fetchall():
        ### Per value ##########
        for index in range(len(header)):
            output += f"{str(item[index]):<{colwidth[index]}} "
        output += "\n"

    print(output+"\n")

#print(statement,cond_query)
def show_statement(statement, cond_query, header, colwidth=None):
    print(statement, cond_query)
    nice_show_in_query(cond_query, header, colwidth)

show_statement("Selecting all items:", "SELECT * FROM Data", data_header)
show_statement("Selecting all items:", "SELECT * FROM ReferenceTable", reference_header
)
show_statement("Inner Join Tables:", "SELECT * FROM Data INNER JOIN ReferenceTable ON
 (Data.DataLink == ReferenceTable.ID)", data_header + ["DataLink"] + reference_header)
show_statement("Left Outer Join Tables:", "SELECT * FROM Data LEFT OUTER JOIN Reference
Table ON (Data.DataLink == ReferenceTable.ID)", data_header + ["DataLink"] + reference_
header)


show_statement("Selecting specific rows:", "SELECT * FROM Data WHERE ID == 1;", data_he
ader)
show_statement("Selecting specific columns:", "SELECT Description FROM Data", ["Descrip
tion"])
show_statement("Selecting specific columns from joined tables:", "SELECT Data.DataLink,
Data.Description, ReferenceTable.Description FROM Data INNER JOIN ReferenceTable ON (Da
ta.DataLink == ReferenceTable.ID)", ["DataLink", "Data.Desc", "Ref.Desc"])

show_statement("Selecting SUM (You can also use COUNT, MAX, MIN)", "SELECT ID, SUM(Numb
er) FROM Data", ["ID", "SUM(Number)"])
```

```
Selecting all items: SELECT * FROM Data
----------------------------------------
ID            Description   Number
----------------------------------------
1             hi            3.1
2             hello         4
3             Join test     4


Selecting all items: SELECT * FROM ReferenceTable
---------------------------
ID            Description
---------------------------
1             ref 1
2             ref 2


Inner Join Tables: SELECT * FROM Data INNER JOIN ReferenceTable ON (Data.D
ataLink == ReferenceTable.ID)
-------------------------------------------------------------------------
----------
ID            Description   Number        DataLink      ID            Desc
ription
-------------------------------------------------------------------------
----------
1             hi            3.1           1             1             ref
1
2             hello         4             2             2             ref
2


Left Outer Join Tables: SELECT * FROM Data LEFT OUTER JOIN ReferenceTable
ON (Data.DataLink == ReferenceTable.ID)
-------------------------------------------------------------------------
----------
ID            Description   Number        DataLink      ID            Desc
ription
-------------------------------------------------------------------------
----------
1             hi            3.1           1             1             ref
1
2             hello         4             2             2             ref
2
3             Join test     4             None          None          None


Selecting specific rows: SELECT * FROM Data WHERE ID == 1;
----------------------------------------
ID            Description   Number
----------------------------------------
1             hi            3.1


Selecting specific columns: SELECT Description FROM Data
--------------
Description
--------------
hi
hello
Join test
```

```
Selecting specific columns from joined tables: SELECT Data.DataLink, Data.
Description, ReferenceTable.Description FROM Data INNER JOIN ReferenceTabl
e ON (Data.DataLink == ReferenceTable.ID)
----------------------------------------
DataLink      Data.Desc     Ref.Desc
----------------------------------------
1             hi            ref 1
2             hello         ref 2


Selecting SUM (You can also use COUNT, MAX, MIN) SELECT ID, SUM(Number) FR
OM Data
----------------------------
ID            SUM(Number)
----------------------------
3             11.1
```

In [71]:

```
### Updating Data ############################################################
update_statement = "UPDATE Data SET Description='Haha hacked' WHERE ID == 1;"
conn.execute(update_statement)
show_statement(f"Updated data with {update_statement}:", "SELECT * FROM Data", data_hea
der)
```

```
Updated data with UPDATE Data SET Description='Haha hacked' WHERE ID ==
1;: SELECT * FROM Data
----------------------------------------
ID            Description   Number
----------------------------------------
1             Haha hacked   3.1
2             hello         4
3             Join test     4
```

In [72]:

```
### Deleting Data ############################################################
update_statement = "DELETE FROM Data WHERE ID == 1;"
conn.execute(update_statement)
show_statement(f"Deleted data with {update_statement}:", "SELECT * FROM Data", data_hea
der)

### Finalisation #############################################################
conn.commit()
conn.close()
```

```
Deleted data with DELETE FROM Data WHERE ID == 1;: SELECT * FROM Data
----------------------------------------
ID            Description   Number
----------------------------------------
2             hello         4
3             Join test     4
```

In [73]:

```
!rm -rf test.db
```

Basic Insertion into Database

In [74]:

```
## Insertion into database
# Or, you know, use DB Browser?
# https://github.com/Hackin7/Programming-Crappy-Solutions/blob/master/Basic%20Exercises/NYJC%20A%20Level%20Computing%202019-2020/Tests%20and%20Exams/JC2/4%20Mid%20Year%20Common%20Test/TASK4_2_Chan%20Zun%20Mun%20Terence.py
# https://github.com/Hackin7/Programming-Crappy-Solutions/blob/master/Basic%20Exercises/NYJC%20A%20Level%20Computing%202019-2020/Other%20School%20Practices/RVHS%20JC2%20CT3%20P2%202020/Task_4_2.py

'''
import sqlite3
conn = sqlite3.connect('test.db')
'''
def refresh(filename): # filename is the .sql file with create statements
    # Clear
    tables = ['Result', 'Test', 'Student']
    for table in tables:
        try:
            conn.execute(f"DROP TABLE '{table}'")
        except Exception as e:
            print(e)

    # Run statements
    file = open(filename, 'r')
    statements = f.read().split("\n\n")
    file.close()
    for i in statements:
        conn.execute(i)

import csv
def read(filename, insert): #pass in function into `insert`
    file = open(filename, 'r') # Read mode
    reader = csv.reader(file)

    header = None # Change to a value if the file has no header
    for data in reader:
        if header == None:
            header = data
        else:
            insert(header, data)

    file.close()

def insertSpecific(header, record):
    # Data Conversion
    record[3] = int(record[3])
    # Insertion
    conn.execute("INSERT INTO Printer(SerialNumber, Toner, DateChanged)"
                 "VALUES (?,?,?)", record)
    # insert("Table", header, record) # alternative if you have time to code

################################################################################

# Code this if you are lazy to read the column headings
def insert(table, header, record):
    # For Columns
    columns = ""
    for h in header:
        if columns != "":
```

```
            columns += ","
        columns += h

    # Parametrised queries
    placeholder = ""
    for r in record:
        if placeholder != "":
            placeholder += ","
        placeholder += "?"

    statement = f"INSERT INTO {table}({columns}) VALUES({placeholder})"
    #print(statement, record)
    conn.execute(statement, tuple(record))


'''
refresh("TASK4_1.sql")
readCSV("students.csv", insertSpecific)
'''
```

Out[74]:

```
'\nrefresh("TASK4_1.sql")\nreadCSV("students.csv", insertSpecific)\n'
```

5.2 MongoDB

In [75]:

```
import pymongo

### Open Connection ######################################################
# Remember to open mongod in shell first
client = pymongo.MongoClient("127.0.0.1",27017) # Address, Port

### Create a database ####################################################
db_name = "database"
coll_name = "collection"

db = client.get_database(db_name) # Creates if doesn't exist
coll = db.get_collection(coll_name)

sample_data = [
    {
        "number":1,
        "text":"Item 1",
        "array": [1,2,3,4],
        "document": {"key":"value","n":1}
    },
    {
        "number":2,
        "text":"Item 2",
        "array": [3,4,5,6]
    },
    {
        "number":3,
        "text":"Item 3, No array",
    },
    {
        "number":3,
        "text":"Item 3 duplicate",
        "document":{"key":"I'm better","n":4}
    },
    {
        "number":5,
        "text":"Item 5",
        "array": [3,4,5,6]
    },
]

#coll.insert_one(sample_data[0])
coll.insert_many(sample_data)

### Search for items #####################################################

def show_in_query(query=None):
    for i in coll.find(query):
        print(i)

#### Shows the thing in a nice table format ############
def nice_show_in_query(query=None):
    output = ""
    ### Header ###################################
    header = ["number","text","array","document"]
    colwidth = [7,30,20,30]
    for j in range(len(header)):
        output+=f"{header[j]:<{colwidth[j]}} "
    output += "\n"
```

```
    ### Border #####################################
    border = ""
    for j in range(len(header)):
        border += "-" * (colwidth[j]+1)
    border += "\n"
    output = border+output+border

    ### Values #####################################
    for i in coll.find(query):
        ### Per value ##########
        for j in range(len(header)):
            output += f"{str(i.get(header[j])):<{colwidth[j]}} "
        output += "\n"

    print(output+"\n")

def showcase_query(statement,cond_query):
    print(statement,cond_query)
    nice_show_in_query(cond_query)
```

In [76]:

```
print("Selecting 1 item")
print("    ",coll.find_one({"number":1}))# Select 1 item

print("Selecting all Items in Collection")
show_in_query()
print()
nice_show_in_query()

### Matching Fields ###################
cond_query = {"number":1, "text":"Item 1"}
showcase_query("Selecting by matching fields:",cond_query)

### Comparisons ########################
cond_query = {"number":{"$gt":1}}
# Has $gt, $gte, $lt, $lte, $ne, $eq, $ne, $exists:True/False
showcase_query("Selecting by comparions:",cond_query)


### $in/$nin ###################
cond_query = {"number":{"$in":[1,2]}}
showcase_query("Selecting by comparing in array:",cond_query)

### Boolean ###################
cond_query =  { "$and": # Has $and, $or
    [
        {"number": {"$not":{"$gte":5} } },
        {"array":{"$gt":4}}
    ]
}
showcase_query("Selecting with Boolean Operators:",cond_query)

### $not ########################
#$not selects those that don't meet condition OR the field doesn't exist
cond_query =  {"array":{"$not":{"$gt":4}}}
showcase_query("Selecting with $not Operators:",cond_query)

### With items in an array ################
# All normal operators should work
cond_query = {"array":{"$gte":5}}
showcase_query("Selecting by comparing items in documents' array:",cond_query)

### Nested Documents ########################
# All normal operators should work
cond_query = {"document.n":{"$gt":1}}
showcase_query("Selecting by comparing items in nested documents:",cond_query)
```

```
Selecting 1 item
    {'_id': ObjectId('5f66d665ae3958ac09725747'), 'number': 1, 'text':
'Item 1', 'array': [1, 2, 3, 4], 'document': {'key': 'value', 'n': 1}}
Selecting all Items in Collection
{'_id': ObjectId('5f66d665ae3958ac09725747'), 'number': 1, 'text': 'Item
1', 'array': [1, 2, 3, 4], 'document': {'key': 'value', 'n': 1}}
{'_id': ObjectId('5f66d665ae3958ac09725748'), 'number': 2, 'text': 'Item
2', 'array': [3, 4, 5, 6]}
{'_id': ObjectId('5f66d665ae3958ac09725749'), 'number': 3, 'text': 'Item
3, No array'}
{'_id': ObjectId('5f66d665ae3958ac0972574a'), 'number': 3, 'text': 'Item
3 duplicate', 'document': {'key': "I'm better", 'n': 4}}
{'_id': ObjectId('5f66d665ae3958ac0972574b'), 'number': 5, 'text': 'Item
5', 'array': [3, 4, 5, 6]}

--------------------------------------------------------------------------
------------------
number  text                            array               document
--------------------------------------------------------------------------
------------------
1       Item 1                          [1, 2, 3, 4]        {'key': 'val
ue', 'n': 1}
2       Item 2                          [3, 4, 5, 6]        None
3       Item 3, No array                None                None
3       Item 3 duplicate                None                {'key': "I'm
better", 'n': 4}
5       Item 5                          [3, 4, 5, 6]        None


Selecting by matching fields: {'number': 1, 'text': 'Item 1'}
--------------------------------------------------------------------------
------------------
number  text                            array               document
--------------------------------------------------------------------------
------------------
1       Item 1                          [1, 2, 3, 4]        {'key': 'val
ue', 'n': 1}


Selecting by comparions: {'number': {'$gt': 1}}
--------------------------------------------------------------------------
------------------
number  text                            array               document
--------------------------------------------------------------------------
------------------
2       Item 2                          [3, 4, 5, 6]        None
3       Item 3, No array                None                None
3       Item 3 duplicate                None                {'key': "I'm
better", 'n': 4}
5       Item 5                          [3, 4, 5, 6]        None


Selecting by comparing in array: {'number': {'$in': [1, 2]}}
--------------------------------------------------------------------------
------------------
number  text                            array               document
--------------------------------------------------------------------------
------------------
1       Item 1                          [1, 2, 3, 4]        {'key': 'val
ue', 'n': 1}
2       Item 2                          [3, 4, 5, 6]        None
```

```
Selecting with Boolean Operators: {'$and': [{'number': {'$not': {'$gte':
5}}}, {'array': {'$gt': 4}}]}
------------------------------------------------------------------------
------------------
number  text                           array                document
------------------------------------------------------------------------
------------------
2       Item 2                         [3, 4, 5, 6]         None


Selecting with $not Operators: {'array': {'$not': {'$gt': 4}}}
------------------------------------------------------------------------
------------------
number  text                           array                document
------------------------------------------------------------------------
------------------
1       Item 1                         [1, 2, 3, 4]         {'key': 'val
ue', 'n': 1}
3       Item 3, No array               None                 None
3       Item 3 duplicate               None                 {'key': "I'm
better", 'n': 4}


Selecting by comparing items in documents' array: {'array': {'$gte': 5}}
------------------------------------------------------------------------
------------------
number  text                           array                document
------------------------------------------------------------------------
------------------
2       Item 2                         [3, 4, 5, 6]         None
5       Item 5                         [3, 4, 5, 6]         None


Selecting by comparing items in nested documents: {'document.n': {'$gt':
1}}
------------------------------------------------------------------------
------------------
number  text                           array                document
------------------------------------------------------------------------
------------------
3       Item 3 duplicate               None                 {'key': "I'm
better", 'n': 4}
```

In [77]:

```
### Update items ###########################################################
'''
Many other operators such as
{"$rename" : {field_name: new_field_name}}
{"$min" : {field_name: value_if_curr_value_lower}}
{"$max" : {field_name: value_if_curr_value_higher}}

https://docs.mongodb.com/manual/reference/operator/update-array/
'''

cond_update = {"number":3}
modifier = {"$set":{"document":{"key":"yes"}}}
print("Updating first item that matches condition:",cond_update,modifier)
coll.update_one(cond_update,modifier)
nice_show_in_query()

cond_update = {"number":3}
modifier = {"$set":{"text":"erosion of local culture"}}
print("Updating some items:",cond_update,modifier)
coll.update_many(cond_update,modifier)
nice_show_in_query()

cond_update = {}
modifier = {"$set":{"array":[1,2,3,4]}}
print("Updating and creating new properties for all items:",cond_update,modifier)
coll.update_many(cond_update,modifier)
nice_show_in_query()

cond_update = {}
modifier = {"$unset":{"array":"value"}} # The "value" doesn't matter
print("$unset proporties of items:",cond_update,modifier)
coll.update_many(cond_update,modifier)
nice_show_in_query()


cond_update = {}
modifier = {"$inc":{"number":1}} # Also has $mul
print("$inc of items:",cond_update,modifier)
coll.update_many(cond_update,modifier)
nice_show_in_query()
```

```
Updating first item that matches condition: {'number': 3} {'$set': {'doc
ument': {'key': 'yes'}}}
------------------------------------------------------------------------
------------------
number  text                            array                document
------------------------------------------------------------------------
------------------
1       Item 1                          [1, 2, 3, 4]         {'key': 'val
ue', 'n': 1}
2       Item 2                          [3, 4, 5, 6]         None
3       Item 3, No array                None                 {'key': 'ye
s'}
3       Item 3 duplicate                None                 {'key': "I'm
better", 'n': 4}
5       Item 5                          [3, 4, 5, 6]         None


Updating some items: {'number': 3} {'$set': {'text': 'erosion of local c
ulture'}}
------------------------------------------------------------------------
------------------
number  text                            array                document
------------------------------------------------------------------------
------------------
1       Item 1                          [1, 2, 3, 4]         {'key': 'val
ue', 'n': 1}
2       Item 2                          [3, 4, 5, 6]         None
3       erosion of local culture        None                 {'key': 'ye
s'}
3       erosion of local culture        None                 {'key': "I'm
better", 'n': 4}
5       Item 5                          [3, 4, 5, 6]         None


Updating and creating new properties for all items: {} {'$set': {'arra
y': [1, 2, 3, 4]}}
------------------------------------------------------------------------
------------------
number  text                            array                document
------------------------------------------------------------------------
------------------
1       Item 1                          [1, 2, 3, 4]         {'key': 'val
ue', 'n': 1}
2       Item 2                          [1, 2, 3, 4]         None
3       erosion of local culture        [1, 2, 3, 4]         {'key': 'ye
s'}
3       erosion of local culture        [1, 2, 3, 4]         {'key': "I'm
better", 'n': 4}
5       Item 5                          [1, 2, 3, 4]         None


$unset proporties of items: {} {'$unset': {'array': 'value'}}
------------------------------------------------------------------------
------------------
number  text                            array                document
------------------------------------------------------------------------
------------------
1       Item 1                          None                 {'key': 'val
ue', 'n': 1}
2       Item 2                          None                 None
3       erosion of local culture        None                 {'key': 'ye
```

```
s'}
3       erosion of local culture       None                {'key': "I'm
better", 'n': 4}
5       Item 5                         None                None


$inc of items: {} {'$inc': {'number': 1}}
------------------------------------------------------------------------
------------------
number  text                           array               document
------------------------------------------------------------------------
------------------
2       Item 1                         None                {'key': 'val
ue', 'n': 1}
3       Item 2                         None                None
4       erosion of local culture       None                {'key': 'ye
s'}
4       erosion of local culture       None                {'key': "I'm
better", 'n': 4}
6       Item 5                         None                None
```

In [78]:

```
### Delete items ########################################################
print('Deleting first item that matches query {"number":3}')
coll.delete_one({"number":3})
nice_show_in_query()

print('Deleting many items with query {"document":{"$exists":True}}')
coll.delete_many({"document":{"$exists":True}})
nice_show_in_query()

print("Deleting rest of collection")
db.drop_collection(coll_name)
client.drop_database(db_name)
```

```
Deleting first item that matches query {"number":3}
------------------------------------------------------------------------------
-----------------
number  text                           array                document
------------------------------------------------------------------------------
-----------------
2       Item 1                         None                 {'key': 'valu
e', 'n': 1}
4       erosion of local culture       None                 {'key': 'yes'}
4       erosion of local culture       None                 {'key': "I'm b
etter", 'n': 4}
6       Item 5                         None                 None


Deleting many items with query {"document":{"$exists":True}}
------------------------------------------------------------------------------
-----------------
number  text                           array                document
------------------------------------------------------------------------------
-----------------
6       Item 5                         None                 None


Deleting rest of collection
```

In [79]:

```
### Close Connection ####################################################
client.close()
```

6. Socket Programming

In [80]:

```
%%writefile socketTester.py
import socket

# An overengineered socket class for A Levels
class Connection:
    def __init__(self, ip, port, server=False):
        self.net = (ip, port)
        self.isServer = server
        if server == True:
            self.set_server()
        else:
            self.set_client()

    ### Important Initialisation steps ################################
    def set_server(self):
        self.socket = socket.socket()
        self.socket.bind(self.net)
        self.socket.listen()
        print('Listening on',self.net)

        self.conn, self.conn_addr = self.socket.accept()
        print('Accepted connection from', self.conn_addr)

    def set_client(self):
        print("Connecting to",self.net)
        self.conn = socket.socket()
        self.conn.connect(self.net)
        print("Connected")

    ### Basic I/O #############################################################
    def send(self, message):
        self.conn.sendall(message.encode('ascii'))

    def receive(self):
        return self.conn.recv(1024).decode('ascii') # 1024 is max size

    def close(self):
        return self.conn.close()

    ### Unnecessary but useful ################################################
    def interactive(self):
        # Loop
        connected = True
        while connected:
            output = self.receive()
            if output == 'Exit':
                connected = False
            else:
                print(output)
                self.send(input())

        self.close()


option = input("Type something to launch as client: ")
if option == "":
    server = Connection('127.0.0.1',1234, True)
    for i in range(10):
        server.send('Send a message:') #Use like a print function
```

```
        print(server.receive())
    server.send('Exit')

    server.close()
else:
    client = Connection('127.0.0.1', 1234)
    client.interactive()
```

```
Writing socketTester.py
```

In [81]:

```
!rm -rf socketTester.py
```

7. Web Application Programming

7.1 HTML

In [82]:

```
%%html
<!-- Basic HTML File Boilerplate -->
<!doctype html>
<html>
    <head>
        <title>Title</title>
    </head>
    <body>
        This is a basic boilerplate: Insert your Content here
    </body>
</html>
```

This is a basic boilerplate: Insert your Content here

In [83]:

```
%%html
<h3>Possible Form Options</h3>
<form action='' method='GET'>
    <label for="textfield">TextField</label>
    <input type='text' name='textfield'>

    <div>
        <input type="radio" name="radio" value="choice1">
        <label for="choice1">Choice 1</label>
        <input type="radio" name="radio" value="choice2">
        <label for="choice1">Choice 2</label>
    </div>

    <input type="checkbox">
    <br>

    <select name="dropdown">
        <option value="option1">Option 1</option>
    </select>
    <br><br>


    <input type='submit'>
</form>
```

Possible Form Options

TextField

Choice 1 Choice 2

Option 1

Submit

In [84]:

```
%%html
<form action="/upload" method="post" enctype="multipart/form-data">
    <input type="file" name="file" id="file">
    <input type="submit"/>
</form>
```

Choose File No file chosen

Submit

In [85]:

```
%%html
<style>
table,td,th{
    border:1px solid black;
    border-collapse: collapse;  /* Add to combine borders */
    background-color:gray;
}

th{
    color:white;
    background-color:black;
}
</style>

<h3>Table</h3>
<table border="10">
    <tr>
        <th>Header 1</th>
        <th>Header 2</th>
    </tr>
    <tr>
        <td>Value 1</td>
        <td>Value 2</td>
    </tr>
</table>
```

Table

Header 1	Header 2
Value 1	Value 2

In [86]:

```
%%html
<img src="https://www.google.com.sg/images/branding/googlelogo/1x/googlelogo_color_272x92dp.png" alt="Google image">
```

7.2 Flask

In [87]:

```
!mkdir templates | mkdir static
```

```
A subdirectory or file templates already exists.
```

Form to Table

In [88]:

```
%%writefile static/style.css
table,td,th{
    border-collapse: collapse;  /* Add to combine borders */
    border:1px solid black;
    background-color:gray;
}

/*
table{
    width:40vw;
    height:40vh;
}
*/
td,th{
    padding:10px;
}
th{
    color:white;
    background-color:black; /* Overriding Styles */
}
```

```
Writing static/style.css
```

In [89]:

```
%%writefile templates/table.html

<!doctype html>
<html>
    <head>
        <title>Table</title>
        <link rel="stylesheet" href="{{url_for('static', filename='style.css')}}"></lin
k>
    </head>
    <body>
        <h1>Table</h1>
        <table>
            <tr>
                {% for header in headers %}
                <th>{{header}}</th>
                {% endfor %}
            </tr>
            {% for record in data %}
            <tr>
                {% for i in record %}
                <td>{{i}}</td>
                {% endfor %}
            </tr>
            {% endfor %}
        </table>
    </body>
</html>
```

```
Writing templates/table.html
```

In [90]:

```
%%writefile templates/form.html
<!doctype html>
<html>
    <head>
        <title>Form</title>
        <link rel="stylesheet" href="{{url_for('static', filename='style.css')}}"></lin
k>
    </head>
    <body>

        <h1>Form</h1>
        <form action="{{url_for('table')}}" method='get' enctype='multipart/form-data'>
            <span>Text Field</span>
            <input type='text' name='textfield'>
            <input type='submit'>
        </form>

    </body>
</html>
```

```
Writing templates/form.html
```

In [91]:

```
%%writefile table.py
import flask
app = flask.Flask(__name__)

@app.route('/')
def main():
    return flask.render_template('form.html')

@app.route('/table', methods=['GET'])
def table():
    form_stuff = flask.request.args['textfield']
    # for POST use flask.request.form

    # Templating
    headers = ['Name','Value']
    data = [
        ['Text Field', form_stuff]
    ]
    return flask.render_template('table.html',
                                 headers=headers,
                                 data=data)

@app.route('/quit')
def quit():
    exit()

if __name__ == "__main__":
    app.run('0.0.0.0', 3000, debug=True)
```

```
Writing table.py
```

In [92]:

```
#!python table.py
```

Basic File Uploading

In [93]:

```
%%writefile file_upload_download.py

## File Uploading Template
import flask
app = flask.Flask(__name__)

@app.route("/", methods=["GET"])
def page():
    return """
    <form action="/upload" method="post" enctype="multipart/form-data">
        <input type="file" name="file" id="file">
        <input type="submit"/>
    </form>
    """

from werkzeug.utils import secure_filename
@app.route("/upload", methods=["POST"])
def upload():
    if flask.request.files:
        file = flask.request.files['file']
        filename = secure_filename(file.filename)
        file.save('./'+filename)
        return "Uploaded "+file.filename
    else:
        return "Nothing to upload"

@app.route("/download", methods=["GET"])
def download():
    return flask.send_from_directory('./','file_upload_download.py')

app.run('0.0.0.0', port=5000, debug=True)
```

```
Writing file_upload_download.py
```

Overall Summary

In [94]:

```
%%writefile templates/main.html
<!doctype html>
<html>
<head>
    <title>Flask Demo</title>
</head>
<body>
    <h1>Flask Demo</h1>

    <h2>HTTP Requests and Routing</h2>
    <ol>
        <li><a href="{{url_for('simple')}}">Simple Routing</a></li>
        <li><a href="{{url_for('simple_data', data=1.2)}}">Passing in Value as Path</a>
</li>
        <li><a href="{{url_for('simple_integer', integer=1)}}">Passing in Value of <b>S
pecific Data Type</b> as Path</a></li>
        <li><a href="{{url_for('func_name_paths')}}">Get Paths By Function name</a></li
>
    </ol>

    <h2>HTTP Responses and Status Code</h2>
    <ol>
        <li><a href="{{url_for('get_500')}}">Give Status Code 500</a></li>
        <li><a href="{{url_for('header')}}">Modifying header to show <code>text/plain</
code> </a></li>
        <li><a href="{{url_for('redirect')}}">Redirect back here lol</a></li>
    </ol>

    <h2>Templating and Rendering</h2>
    <ol>
        <li><a href="{{url_for('templating')}}">Templating sample</a></li>
    </ol>

    <h2>Form</h2>
    <h3>GET Request Form</h3>
    <form method="GET" action="/form">
        <input type="text" name="data">
        <input type="submit">
    </form>

    <h3>POST Request Form</h3>
    <form method="POST" action="/form">
        <input type="text" name="data">
        <input type="submit">
    </form>

    <h3>Upload Photos</h3>
    <form method="post" enctype="multipart/form-data" action="/upload">
        <input type="file" name="file">
        <input type="submit">
    </form>

    <h3>Download Source Code</h3>
    <a href="{{url_for('download')}}">Download app.py</a>
</body>
</html>
```

```
Writing templates/main.html
```

In [95]:

```
%%writefile templates/template.html
<!doctype html>
<html>
<head>
    <title>Jinja Template</title>
    <!-- Getting static files -->
    <link rel="stylesheet" href="{{url_for('static', filename='styles.css')}}"></link>
</head>
<body>
    <h1>Jinja Templating</h1>

    <h2>Passing in Values</h2>
    Values: {{value}}<br/>
    Length: {{value|length}}<br/>
    Safe HTML: {{safe_html|safe}}

    <h2>Jinja Statements</h2>

    <b>if-else statement</b>
    Current Choice value: {{choice}} <br/>
    {% if choice == 1 %}
        <span>1st class honors that no one cares about</span>
    {% elif choice == 2 %}
        <span>2nd class honors that you feel bad because it isn't 1st</span>
    {% else %}
        <span>Useless choice value here</span>
    {% endif%}
    <br><br>

    <b>for-in statement</b>
    Given Dictionary: {{dictionary}} <br/>
    <table>
    <tr><th>Key</th><th>Value</th></tr>
    {% for key, value in dictionary.items() %}
        <tr><td>{{key}}</td><td>{{value}}</td></tr>
    {% endfor %}
    </table>

</body>
</html>
```

```
Writing templates/template.html
```

In [96]:

```
%%writefile static/style.css
table{border-collapse:collapse;}
tr,td,th{border-style:solid;border-width:1px;padding:1em;}
```

```
Overwriting static/style.css
```

In [97]:

```
%%writefile app.py
import flask
app = flask.Flask(__name__)

@app.route('/')
def main():
    return flask.render_template('main.html')

### 3: HTTP Requests and Routing ###################################
base = '/simple'
@app.route(base + '/') # Path name
def simple():
    return 'Simple stuff. Try passing in longer paths next time?'

## Variable Routes
@app.route(base + '/greet/<data>') # More complex paths
def simple_data(data):
    return f'You gave the data {data}'

@app.route(base + '/<int:integer>') # Get Path <data_type:identifier>
def simple_integer(integer):
    return f'You gave the integer {integer}'

## Generate Paths from function names
@app.route(base + '/func_paths')
def func_name_paths():
    return '<b>Paths</b>:' + '<br/>' + \
           f'General path: {flask.url_for("simple")}' + '<br/>' + \
           f'Path and passing in arguments: {flask.url_for("simple_integer", integer=
1)}'

## Methods
@app.route('/data/', methods=['POST'])
def post_data():
    return 'You are using POST'

@app.route('/data/', methods=['GET'])
def get_data():
    return 'You are using GET'

### 4: HTTP Responses and Status Codes ##############################
base = '/responses'

## Changing Status Code
@app.route(base + '/get_500')
def get_500():
    return ('', 500)

@app.route(base + '/header')
def header():
    headers = {'Content-Type': 'text/plain'}
    return ('<b>This is not HTML!</b>', 200, headers)

@app.route(base + '/redirect')
def redirect():
    return flask.redirect(flask.url_for('main'))

### 5 & 6: Templating and Rendering ################################
@app.route('/templating') # Path name
```

```
def templating():
    dictionary = {'Key 1':'Value 1', 'Key 2':'Value 2'}
    return flask.render_template('template.html',
                                 value="sample_value",
                                 safe_html="<i>This is html stuff</i>",
                                 choice=1,
                                 dictionary=dictionary)

### 7 & 8: Forms
@app.route('/form', methods=['GET','POST'])
def form():
    if flask.request.method == "GET":
        return f" The data is given in <code>flask.request.args</code> like this: <code
>{flask.request.args}</code>"
    elif flask.request.method == "POST":
        print(flask.request.form)
        return f"The data is given in <code>flask.request.form</code> like this: <code>
{flask.request.form}</code>"

from werkzeug.utils import secure_filename
import os
@app.route('/upload', methods=['POST'])
def upload():
    if flask.request.files:
        print(flask.request.files)
        file = flask.request.files['file']
        filename = secure_filename(file.filename)
        path = './' + filename
        file.save(path)
        return f'<h1>Uploaded {file.filename}</h1>'
    else:
        return '<h1>Nothing uploaded</h1>'

@app.route('/download')
def download():
    return flask.send_from_directory("./","app.py")


### Running Code ########################################################
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5000, debug=True)
```

```
Writing app.py
```

Cleanup

In [98]:

```
! rm -rf templates | rm -rf static | rm -rf table.py | rm -rf file_upload_download.py |
rm -rf app.py
```