How to use popular data structures and algorithms in Python ?

List

```
numbers = []
numbers.append(1)
numbers.append(2)
numbers.append(3)

for num in numbers:
   print(num)

# Output
'''
1
2
3
'''
```

```
numbers = [1, 2, 3]
length = len(numbers)
for i in range(length):
   print(numbers[i])

# Output
'''
1
2
3
'''
```

Strings

```
string = 'Hello world'

# Iterate over each character in the string
for c in string:
   print(c)

# This also iterates over each character in the string
length = len(string)
for i in range(length):
   print(string[i])
```

Sorting a list

```
a = [2, 4, 1, 9, 8]
a.sort()
print(a)
# Output: [1, 2, 4, 8, 9]
```

Sorting a string

```
s = 'shivam'
sorted_s = ''.join(sorted(s))
print(sorted_s)
```

Dictionary

```
d = {'1': 'Dhoni', '2': 'Sachin', '3': 'Dravid'}

# Iterating over key and values
for k,v in d.items():
   print(k,v)

# Output
'''
1 Dhoni
2 Sachin
3 Dravid
'''
```

```
d = {}

# Store square of numbers from 1 to 5
for i in range(1, 6):
   d[i] = i*i

# Print the square
for k,v in d.items():
   print(k,v)

# Output
'''
1 1
2 4
3 9
4 16
5 25
'''
```

Converting a char into integer and vice-versa

```
print(ord('a')) # prints 97 - ord converts a character into an integer(unicode code)
print(chr(97)) # prints a -  chr converts an integer(unicode code) to its equivalent character
```

Using heap in Python

```
import heapq

L = []
heapq.heapify(L) # Min heap
heapq.heappush(L, 3) # Pushing 3 into heap
heapq.heappush(L, 2) # Pushing 2 into heap
print(heapq.heappop(L)) # Popping the minimum element
```

How to use max heap in Python - Multiply each number with -1 while pushing the element in the heap.

Using queue in Python

```
from collections import deque

q = deque([])
q.append(1)
q.append(2)
q.append(3)
print(q.popleft()) # 1 will be popped out and printed
print(q.popleft()) #2 will be popped out and printed
```

Using stack in Python

```
stack = []
stack.append(1)
stack.append(2)
stack.append(3)
print(stack.pop()) # Will remove 3 from stack and print
it
print(stack.pop()) # Will remove 2 from stack and print
it
```

```
from collections import deque

stack = deque([])
stack.append(1)
stack.append(2)
stack.append(3)

print(stack.pop()) # Will remove 3 from stack and print
it
print(stack.pop()) # Will remove 2 from stack and print
it
```

Create a 2D array/list

```
N = 5 # Number of rows
M = 5 # Number of columns
matrix = [[0]*M]*N

# Print the matrix
for i in range(N):
   for j in range(M):
       print(matrix[i][j], end='')
   print()

# Output:
'''
00000
00000
00000
00000
00000
'''
```

Using set in Python

A set is an unordered data type which doesn't contain any duplicate elements.

```
s = set()
s.add(1)
s.add(4)
s.add(3)
s.add(1)
print(s) # Prints {1, 3, 4}
```

Ordered dictionary in Python

Ordered dictionary preserves the order of insertion.

```
from collections import OrderedDict

d = OrderedDict()
d[1] = 2
d[2] = 3
d[3] = 4

# If we print the dictionary now, the keys will be
printed in order of insertion.
for k,v in d.items():
   print(k,v)

# output
# 1 2
# 2 3
# 3 4

print(d.popitem(last=False)) # (1,2) Removes the first
key value and prints it
print(d.popitem()) # (3,4) Removes the last key value and
prints it
```

Binary Search

Check if an elements is present or not and return the index if element is present

```
from bisect import bisect_left
a = [1, 2, 4, 4, 5]
x = 4
idx = bisect_left(a, x)
if idx != len(a) and a[idx] == x:
 print(f'{x} is present at index {idx}')
else:
 print('Element not found')
```

Find index of first element greater than or equal to target(x)

```
from bisect import bisect_left
a = [1, 2, 4, 4, 5]
x = 6
idx = bisect_left(a, x)
if idx == len(a):
  print('All elements are smaller than the target
element')
else:
  print(idx)
```

Find index of first element greater than target(x)

```
from bisect import bisect_right
a = [1, 2, 4, 4, 5]
x = 4
idx = bisect_right(a, x)
if idx == len(a):
  print('All elements are smaller than the target
element')
else:
  print(idx)
```

Graph

```
from collections import defaultdict

graph = defaultdict(list)
graph[1].append(2) # 1 -> 2
graph[2].append(3) # 1 -> 2 -> 3
graph[4].append(1) # 4 -> 1 -> 2 -> 3

visited = set()

def dfs(node, graph, visited):
   if node not in visited:
       print(node)
       visited.add(node)
       for v in graph[node]:
           dfs(v, graph, visited)

dfs(4, graph, visited)
```

Linkedlist

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

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

    def insert_at_end(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            temp = self.head
            while(temp.next):
                temp = temp.next
            temp.next = new_node

    def traverse(self):
        temp = self.head
        while(temp):
            print(temp.data)
            temp = temp.next

ll = LinkedList()
ll.insert_at_end(1)
ll.insert_at_end(2)
ll.insert_at_end(3)
ll.traverse() # 1 -> 2 -> 3
```

Binary Tree

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.right.left = Node(5)

# Binary tree created from the above code
'''
              1
            / \
           2   3
          /     \
         4       5
'''

def inorder(root):
    if not root:
        return
    print(root.data)
    inorder(root.left)
    inorder(root.right)

inorder(root)
```

There is also a binary tree module in Python but it doesn't come pre-installed in the standard modules. You can read about it here.