**Lab Code:20ECL301**

# Data Structures using Python Lab Manual


**Department of Electronics & Communication Engineering**

## Bapatla Engineering College :: Bapatla

**(Autonomous)**

**G.B.C. Road, Mahatmajipuram, Bapatla-522102, Guntur (Dist.)**
**Andhra Pradesh, India.**
**E-Mail:bec.principal@becbapatla.ac.in**
**Web:www.becbapatla.ac.in**

# Contents

# Bapatla Engineering College :: Bapatla
**(Autonomous)**

## Vision

- To build centers of excellence, impart high quality education and instill high standards of ethics and professionalism through strategic efforts of our dedicated staff, which allows the college to effectively adapt to the ever changing aspects of education.

- To empower the faculty and students with the knowledge, skills and innovative thinking to facilitate discovery in numerous existing and yet to be discovered fields of engineering, technology and interdisciplinary endeavors.

## Mission

- Our Mission is to impart the quality education at par with global standards to the students from all over India and in particular those from the local and rural areas.

- We continuously try to maintain high standards so as to make them technologically competent and ethically strong individuals who shall be able to improve the quality of life and economy of our country.

# Bapatla Engineering College :: Bapatla

## (Autonomous)

### Department of Electronics and Communication Engineering

## Vision

To produce globally competitive and socially responsible Electronics and Communication Engineering graduates to cater the ever changing needs of the society.

## Mission

- To provide quality education in the domain of Electronics and Communication Engineering with advanced pedagogical methods.

- To provide self learning capabilities to enhance employability and entrepreneurial skills and to inculcate human values and ethics to make learners sensitive towards societal issues.

- To excel in the research and development activities related to Electronics and Communication Engineering.

# Bapatla Engineering College :: Bapatla

# (Autonomous)

## Department of Electronics and Communication Engineering

---

### Program Educational Objectives (PEO's)

**PEO-I:** Equip Graduates with a robust foundation in mathematics, science and Engineering Principles, enabling them to excel in research and higher education in Electronics and Communication Engineering and related fields.

**PEO-II:** Impart analytic and thinking skills in students to develop initiatives and innovative ideas for Start-ups, Industry and societal requirements.

**PEO-III:** Instill interpersonal skills, teamwork ability, communication skills, leadership, and a sense of social, ethical, and legal duties in order to promote lifelong learning and Professional growth of the students.

# Program Outcomes (PO's)

Engineering Graduates will be able to:

**PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

**PO2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

**PO3. Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**PO4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

**PO6. The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**PO7.Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**PO9. Individual and Teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**PO10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**PO11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**PO12. Life-long learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

# Bapatla Engineering College :: Bapatla

## (Autonomous)

### Department of Electronics and Communication Engineering

## Program Specific Outcomes (PSO's)

**PSO1:** Develop and implement modern Electronic Technologies using analytical methods to meet current as well as future industrial and societal needs.

**PSO2:** Analyze and develop VLSI, IoT and Embedded Systems for desired specifications to solve real world complex problems.

**PSO3:** Apply machine learning and deep learning techniques in communication and signal processing.

**Data Structures Using Python Lab**
**II B.Tech. – III Semester (Code: 20ECL301)**

| Lectures | 0 | Tutorial | 0 | Practical | 3 | Credits | 1.5 |
|---|---|---|---|---|---|---|---|
| Continuous Internal Evaluation | | | 30 | Semester End Examination (3 Hours) | | | 70 |

**Prerequisites:** Data Structures

**Course Objectives:** Students will be able to

- Implement various Searching and Sorting Techniques.
- Create different linear data structures like linked lists, stacks, and queues.
- Create non-linear data structures like trees and graphs.
- Understand the searching mechanism like depth first search and breadth first search.

**Course Outcomes:** After studying this course, the students will be able to

| | |
|---|---|
| **CO1** | Compose different sorting and searching algorithms |
| **CO2** | Implement linear data structures like linked list, stacks, and queues. |
| **CO3** | Develop non-linear data structures like trees and graphs. |
| **CO4** | Demonstrate traversal techniques on non-linear data structures. |

**Mapping of Course Outcomes with Program Outcomes & Program Specific Outcomes**

| | PO's | | | | | | | | | | | | PSO's | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **CO** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **1** | **2** | **3** |
| **CO1** | 2 | 3 | | | 2 | | | | 3 | | | | | 2 | 2 |
| **CO2** | 2 | 3 | | | 2 | | | | 3 | | | | | 2 | 2 |
| **CO3** | 2 | 3 | | | 2 | | | | 3 | | | | | 2 | 2 |
| **CO4** | 2 | 3 | | | 2 | | | | 3 | | | | | 2 | 2 |
| **AVG** | 2 | 3 | | | 2 | | | | 3 | | | | | 2 | 2 |

## LIST OF LAB PROGRAMS

1. Python program to implement bubble sort, selection sort and insertion sort.
2. Python program to implement merge sort, quick sort
3. Python program on linear search and binary search.
4. Python program to implement Singly Linked List
5. Python program to implement Doubly Linked List
6. Python program to implement Circular Linked List

7. Python programs to implement stacks using arrays and linked lists.
8. Python programs to implement queues using arrays and linked lists.
9. Python programs to implement double ended queues.
10. Python program to perform Binary Tree traversal operations.
11. Python programs to perform Binary search tree operations.
12. Python program to create diff. types of graphs using graph ADT.
13. Python program to Travers in a graph using Depth first search.
14. Python program to Travers in a graph using breadth first search.

**NOTE:** A minimum of 10 (Ten) experiments have to be Performed and recorded by the candidate to attain eligibility for Semester End Examination.

# 1. SORTING TECHNIQUES
# (Bubble Sort, Selection Sort and Insertion Sort)

## Aim:

To Write a Python program to implement bubble sort, selection sort and insertion sort.

**Software Required:** IDLE (Python 3.11)

## Theory:

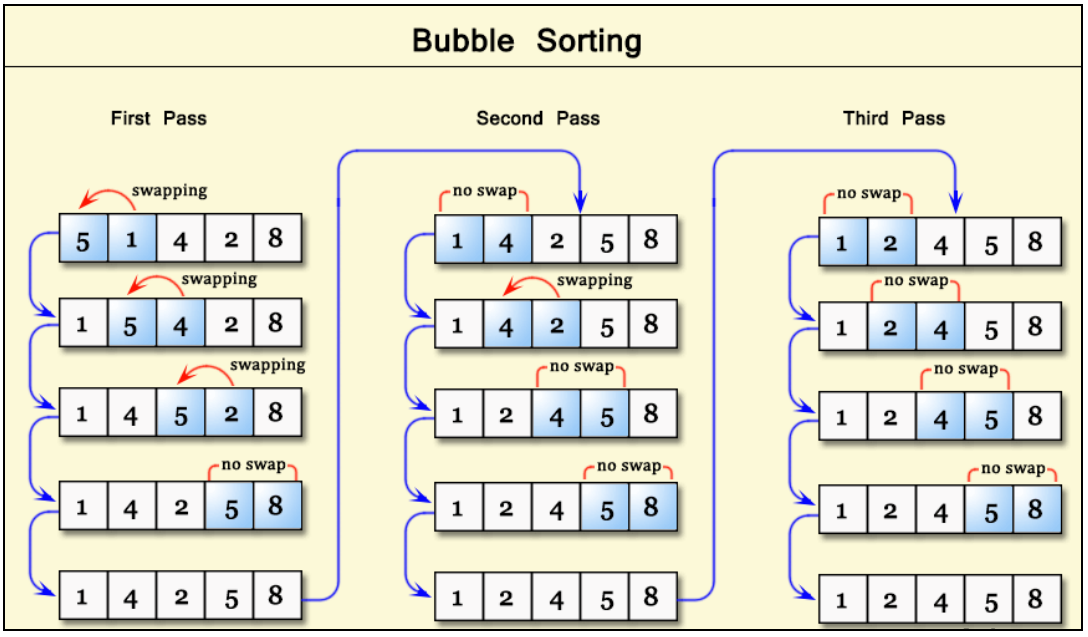
Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

**Characteristics of Bubble Sort:**

Time Complexity: $O(n^2)$ in the worst and average cases, O(n) in the best case (when the array is already sorted).

Space Complexity: O(1) because it is an in-place sorting algorithm.

Stability: Bubble sort is a stable sort; it maintains the relative order of equal elements.




## Source Code:

## Bubble Sort:

```
def bubble_sort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
```

```
        # Last i elements are already sorted

        for j in range(0, n-i-1):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
array = [64, 34, 25, 12, 22, 11, 90]
print("Unsorted array:",array)
bubble_sort(array)
print("Sorted array:", array)
```

## Optimized Bubble Sort:

```
def bubble_sort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # Last i elements are already sorted
        swapped = False
        for j in range(0, n-i-1):
 # Traverse the array from 0 to n-i-1
 # Swap if the element found is greater than the next element
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
 # If no two elements were swapped by inner loop, then break
        if not swapped:
            break

array = [64, 34, 25, 12, 22, 11, 90]
print("Unsorted array:",array)
bubble_sort(array)
print("Sorted array:", array)
```

## Output:

**Unsorted array: [64, 34, 25, 12, 22, 11, 90]**
**Sorted array: [11, 12, 22, 25, 34, 64, 90]**

## Selection Sort:

## Theory:

Selection Sort is a simple comparison-based sorting algorithm. It works by repeatedly selecting the smallest (or largest, depending on the sorting order) element from the unsorted portion of the array and moving it to the beginning. This process is repeated for each position in the array until it is fully sorted.

**Selection Sort Working Process:**

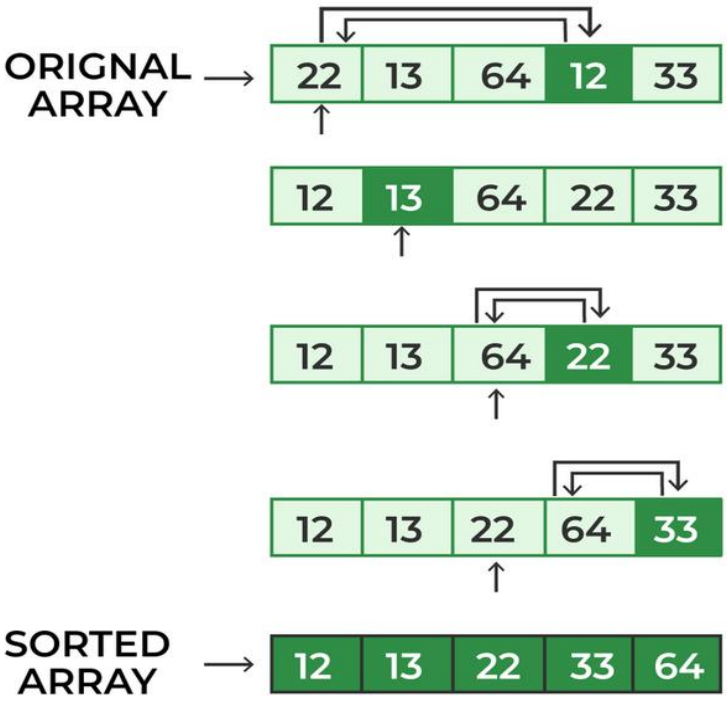
Initialization: Start with the first element as the minimum.

Iteration: For each position in the array:

Find the smallest element in the unsorted portion.

Swap it with the element at the current position.

Repeat: Continue this process for each element until the entire array is sorted.




## Source Code:

```
def selection_sort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
     # Assume the minimum is the first element of the   unsorted part
        min_index = i
    # Find the minimum element in remaining unsorted array
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
```

```
    # Swap the found minimum element with the first element of
    unsorted part
      arr[i], arr[min_index] = arr[min_index], arr[i]

array = [22, 13, 64, 12, 33]
print("Unsorted array:",array)
selection_sort(array)
print("Sorted array:", array)
```

## Output:

**Unsorted array: [22, 13, 64, 12, 33]**
**Sorted array: [12, 13, 22, 33, 64]**

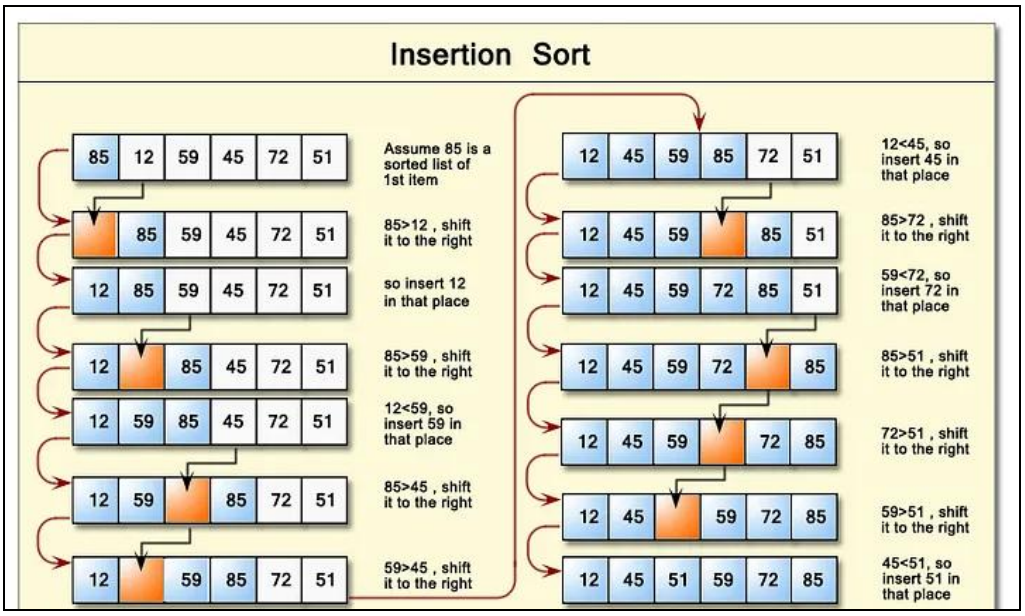
## Insertion Sort:

## Theory:

Insertion Sort is a straightforward sorting algorithm that builds a sorted array one element at a time.

It is efficient for small datasets and works well for nearly sorted arrays.

The algorithm divides the array into a sorted and an unsorted section, repeatedly moving elements from the unsorted section to their correct position in the sorted section.




## Source Code:

```
def insertion_sort(arr):
    n = len(arr)
    # Iterate over each element starting from the second
    for i in range(1, n):
        key = arr[i]  # The current element to be inserted
        j = i - 1
        # Move elements of arr[0..i-1] that are greater than key
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
             # Shift element to the right
            j -= 1
        arr[j + 1] = key  # Insert the key at its correct position
array = [64, 34, 25, 12, 22, 11, 90]
print("Unsorted array:",array)
insertion_sort(array)
print("Sorted array:", array)
```

## Output:

**Unsorted array: [64, 34, 25, 12, 22, 11, 90]**

**Sorted array: [11, 12, 22, 25, 34, 64, 90]**

# 2. SORTING TECHNIQUES (Merge Sort and Quick Sort)

## Aim:

To Write a Python program to implement Merge sort and Quick sort.

**Software Required:** IDLE (Python 3.11)

## Source Code:

### Merge Sort:

```
L1=[5,6,3,7]
L2=[8,1,2,4]
M=[0,0,0,0,0,0,0,0]
i=0
while(i<4):
  j=i+1
  while(j<4):
    if(L1[i]>L1[j]):
      L1[i],L1[j]=L1[j],L1[i]
    if(L2[i]>L2[j]):
      L2[i],L2[j]=L2[j],L2[i]
    j+=1
  i+=1
i=0
j=0
k=0
while(k<8):
  if(L1[i]<=L2[j]):
    M[k]=L1[i]
    i+=1
    k+=1
  else:
    M[k]=L2[j]
    j+=1
    k+=1
  if(i==4 or j==4):
   break
while(j<4):
  M[k]=L2[j]
  j+=1
  k+=1
while(i<4):
```

```
    M[k]=L1[i]
    i+=1
    k+=1
  print("Array after sorting:",M)
```

## Output:

Array after sorting: [1, 2, 3, 4, 5, 6, 7, 8]

## Quick Sort:

```
 def split(arr,lower,upper):
    l=lower+1
    u=upper p=arr[lower]
    while(u>=l):
      while(arr[l]<p):
        l=l+1
        if(l==len(arr)):
          break
      while(arr[u]>p):
        u=u-1if(u>l):
        arr[l],arr[u]=arr[u],arr[l]
      arr[lower],arr[u]=arr[u],arr[lower]
    return u
  def quicksort(arr,lower,upper):
    if(upper>lower):
      pivote=split(arr,lower,upper)
      quicksort(arr,lower,pivote-1)
      quicksort(arr,pivote+1,upper)
  print("Enter list elements:")
  arr=list(input())
  print("Before quicksort the array is:",arr)
  quicksort(arr,0,len(arr)-1)
  print("After quicksort the array is:",arr)
```

## Output:

**Enter list elements:654321**
**Before quicksort the array is: ['6', '5', '4', '3', '2', '1']**
**After quicksort the array is: ['1', '2', '3', '4', '5', '6']**

# 3. SEARCHING TECHNIQUES
# (Linear Search and Binary Search)

## Aim:

To Write a Python program to implement Linear and Binary Search.

**Software Required:** IDLE (Python 3.11)

## Source Code:

### Linear Search:

```
print("Enter the list values")
a=input()
print("Enter a value to find")
n=input()
for i in range(len(a)):
   if(n==a[i]):
      print("the given value",n,"is found in",i,"th location")
      break
if(i==len(a)):
   print("The given element is not found in the list")
```

## Output:

```
Enter the list values87654
Enter a value to find5
The given value 5 is found in 3 th location
```

### Binary Search:

```
print("Enter list elements")
arr=list(input())
print("The length of list is:",len(arr))
print("The list elements before sorting",arr)
i=0
while(i<len(arr)-1):
      j=0
      while(j<len(arr)-1):
      if arr[j]>arr[j+1]:
         arr[j],arr[j+1]=arr[j+1],arr[j]
      j=j+1
   i=i+1
print("The list elements after sorting are",arr)
print("Enter a value to search")
```

```
n=input()
flag=0
l=0
u=(len(arr)-1)
mid=int((l+u)/2)
while(l<=u):
    if(n==arr[mid]):
      flag=1
      print("The given number is found in the location",mid)
      if(n<arr[mid]):
       u=mid-1
      else:
       l=mid+1
    mid=int((l+u)/2)
if(flag==0):
   print("The element is not found in the list")
```

**Output:**

```
Enter list elements321654
The length of list is: 6
The list elements before sorting ['3', '2', '1', '6', '5', '4']
The list elements after sorting are ['1', '2', '3', '4', '5', '6']
Enter a value to search 3
The given number is found in the location 2
```

# 4. SINGLY LINKED LIST

**Aim:**
To Write a Python program to implement Singly Linked List.

**Software Required:** IDLE (Python 3.11)

**Source Code:**

```
class Node:
    def __init__(self, data):
        self.item = data
        self.ref = None
class LinkedList:
    def __init__(self):
        self.start_node = None
#display
    def traverse_list(self):
        if self.start_node is None:
            print("List has no element")
            return
        else:
            n = self.start_node
            while n is not None:
                print(n.item , "--> ",end='')
                n = n.ref
            print()
#insert at start
    def insert_at_start(self, data):
        new_node = Node(data)
        new_node.ref = self.start_node
        self.start_node= new_node
# insert at end
    def insert_at_end(self, data):
        new_node = Node(data)
        if self.start_node is None:
            self.start_node = new_node
            return
        n = self.start_node
        while n.ref is not None:
            n= n.ref
```

```
            n.ref = new_node
    # insert at position
        def insert_at_position (self, position, data):
            if position == 1:
                new_node = Node(data)
                new_node.ref = self.start_node
                self.start_node = new_node
            i = 1
            n = self.start_node
            while i < position-1 and n is not None:
                n = n.ref
                i = i+1
            if n is None:
                print("position out of bound")
            else:
                new_node = Node(data)
                new_node.ref = n.ref
                n.ref = new_node
    #del at start
        def delete_at_start(self):
            if self.start_node is None:
                print("The list has no element to delete")
                return
            self.start_node = self.start_node.ref
    #del at end
        def delete_at_end(self):
            if self.start_node is None:
                print("The list has no element to delete")
                return
            n = self.start_node
            while n.ref.ref is not None:
                n = n.ref
            n.ref = None
    # del element by value
        def delete_element_by_value(self, x):
            if self.start_node is None:
                print("The list has no element to delete")
                return
```

```
        # Deleting first node
            if self.start_node.item == x:
                self.start_node = self.start_node.ref
                return
            n = self.start_node
            while n.ref is not None:
                if n.ref.item == x:
                    break
                n = n.ref
            if n.ref is None:
                print("item not found in the list")
            else:
                n.ref = n.ref.ref
    sll=LinkedList()
    n1=Node(5)
    sll.start_node=n1
    n2=Node(10)
    n1.ref=n2
    sll.insert_at_start(15)
    sll.insert_at_end(20)
    sll.insert_at_end(30)
    sll.insert_at_end(40)
    sll.traverse_list()
    print('del starts here')
    sll.delete_at_start()
    sll.delete_at_end()
    sll.delete_element_by_value(30)
    sll.traverse_list()
```

**Output:**

```
5 --> 10 -->
15 --> 5 --> 10 -->
15 --> 5 --> 10 --> 20 -->
15 --> 5 --> 10 --> 20 --> 30 -->
15 --> 5 --> 10 --> 20 --> 30 --> 40 -->
deletion starts from here
5 --> 10 --> 20 --> 30 --> 40 -->
5 --> 10 --> 20 --> 30 -->
5 --> 10 --> 20 -->
```

# 5. DOUBLY LINKED LIST

## Aim:

To Write a Python program to implement Doubly Linked List.

## Software Required: IDLE (Python 3.11)

## Source Code:

```
# Initialise the Node
class Node:
    def __init__(self, data):
        self.item = data
        self.next = None
        self.prev = None
# Class for doubly Linked List
class doublyLinkedList:
    def __init__(self):
        self.start_node = None
    # Insert Element to Empty list
    def InsertToEmptyList(self, data):
        if self.start_node is None:
            new_node = Node(data)
            self.start_node = new_node
        else:
            print("The list is not empty")
    # Insert element at the end
    def InsertToEnd(self, data):
        # Check if the list is empty
        if self.start_node is None:
            new_node = Node(data)
            self.start_node = new_node
            return
        n = self.start_node
        # Iterate till the next reaches NULL
        while n.next is not None:
            n = n.next
        new_node = Node(data)
        n.next = new_node
        new_node.prev = n
    def insertatstart(self,data):
        if self.start_node is None:
            new_node=Node(data)
            self.start_node=new_node
            return
```

```
        else:
            new_node=Node(data)
            new_node.next=self.start_node
            self.start_node.prev=new_node
            self.start_node=new_node
    # Delete the elements from the start
    def DeleteAtStart(self):
        if self.start_node is None:
            print("The Linked list is empty, no element to delete")
            return
        if self.start_node.next is None:
            self.start_node = None
            return
        self.start_node = self.start_node.next
        self.start_node.prev = None;
    # Delete the elements from the end
    def delete_at_end(self):
        # Check if the List is empty
        if self.start_node is None:
            print("The Linked list is empty, no element to delete")
            return
        if self.start_node.next is None:
            self.start_node = None
            return
        n = self.start_node
        while n.next is not None:
            n = n.next
        n.prev.next = None
    # Traversing and Displaying each element of the list
    def Display(self):
        if self.start_node is None:
            print("The list is empty")
            return
        else:
            n = self.start_node
            print('elements in doubly linkedd list')
            while n is not None:
                print(n.item,'--',end='')
                n = n.next
        print("\n")

# Create a new Doubly Linked List
NewDoublyLinkedList = doublyLinkedList()
# Insert the element to empty list
```

```
NewDoublyLinkedList.InsertToEmptyList(10)
# Insert the element at the end
NewDoublyLinkedList.InsertToEnd(20)
NewDoublyLinkedList.InsertToEnd(30)
NewDoublyLinkedList.InsertToEnd(40)
NewDoublyLinkedList.InsertToEnd(50)
NewDoublyLinkedList.InsertToEnd(60)
# Display Data
NewDoublyLinkedList.Display()
# Delete elements from start
NewDoublyLinkedList.DeleteAtStart()
# Delete elements from end
NewDoublyLinkedList.DeleteAtStart()
# Display Data
NewDoublyLinkedList.Display()
```

**Output:**

**Elements in doubly linked list**
**10 --20 --30 --40 --50 --60 --**

**Elements in doubly linked list**
**30 --40 --50 --60 --**

# 6. CIRCULAR LINKED LIST

## Aim:

To Write a Python program to implement Circular linked list.

## Software Required: IDLE (Python 3.11)

## Source Code:

```
# node structure
class Node:
  def __init__(self, data):
    self.item = data
    self.ref = None

#class Linked List
class LinkedList:
  def __init__(self):
    self.start_node = None

  #Add new element at the start of the list
  def insert_at_start(self, data):
    new_node = Node(data)
    if(self.start_node == None):
      self.start_node = new_node
      new_node.ref = self.start_node
      return
    else:
      n = self.start_node
      while(n.ref != self.start_node):
        n = n.ref
      n.ref = new_node
      new_node.ref = self.start_node
      self.start_node = new_node

  #display the content of the list
  def PrintList(self):
    n = self.start_node
    if(n != None):
      print("The list contains:", end=" ")
      while (True):
        print(n.item, end=" ")
        n = n.ref
        if(n == self.start_node):
```

```
            break
        print()
      else:
        print("The list is empty.")
    def insert_at_end(self, data):
      new_node = Node(data)
      if(self.start_node == None):
        self.start_node = new_node
        new_node.ref = self.start_node
        return
      else:
        n = self.start_node
        while(n.ref != self.start_node):
          n = n.ref
        n.ref = new_node
        new_node.ref = self.start_node
    #Delete first node of the list
    def delete_at_start(self):
      if(self.start_node != None):
        if(self.start_node.ref == self.start_node):
          self.start_node = None
        else:
          n = self.start_node
          while(n.ref != self.start_node):
            n = n.ref
          self.start_node = self.start_node.ref
          n.ref = self.start_node
      else:
        print("no elements to delete")
  #del at last node
    def delete_at_end(self):
        if(self.start_node != None):
            if(self.start_node.ref == self.start_node):
                self.start_node = None
            else:
                n = self.start_node
                while(n.ref.ref != self.start_node):
                    n = n.ref
                n.ref = self.start_node
        else:
          print("no elements to delete")

  # test the code
  MyList = LinkedList()
```

```
#Add three elements at the start of the list.
MyList.insert_at_end(10)
MyList.insert_at_end(20)
MyList.insert_at_end(30)
MyList.PrintList()
MyList.delete_at_end()
MyList.PrintList()
MyList.insert_at_end(40)
MyList.PrintList()
MyList.delete_at_start()
MyList.PrintList()
```

## Output:

**The list contains: 10 20 30**
**The list contains: 10 20**
**The list contains: 10 20 40**
**The list contains: 20 40**

# 7.STACKS

## Aim:

To Write a Python program to implement Stacks using Arrays and Linked list.

**Software Required:** IDLE (Python 3.11)

## Source Code:

**Stack using Arrays:**

```
class stack:
    def __init__(self):
        self.arr=[ ]
    def push(self,item):
        self.arr.append(item)
        print('stack elements after pushing',item,'is',self.arr)
    def popp(self):
        if len(self.arr)==0:
            return print('empty stack')
        removed=self.arr.pop()
        print('stack elements after pop',self.arr)
        return removed
s1=stack()
s1.push(5)
s1.push(10)
s1.push(15)
s1.push(20)
s1.popp()
s1.push(30)
s1.popp()
s1.popp()
s1.popp()
s1.popp()
s1.popp()
```

**Output:**

```
stack elements after pushing 5 is [5]
stack elements after pushing 10 is [5, 10]
stack elements after pushing 15 is [5, 10, 15]
stack elements after pushing 20 is [5, 10, 15, 20]
stack elements after pop [5, 10, 15]
stack elements after pushing 30 is [5, 10, 15, 30]
stack elements after pop [5, 10, 15]
stack elements after pop [5, 10]
stack elements after pop [5]
stack elements after pop []
empty stack
```

## Stack using Linked List:

```
class Node:
    # Class to create nodes of linked list
    # constructor initializes node automatically
    def __init__(self,data):
        self.data = data
        self.ref = None
class Stack:
    # top is default NULL
    def __init__(self):
        self.top = None
    # Method to add data to the stack
    # adds to the start of the stack
    def push(self,data):
        if self.top == None:
            newnode=Node(data)
            self.top=newnode
        else:
            newnode = Node(data)
            newnode.ref = self.top
            self.top = newnode
    # Remove element that is the current top (start of the stack)
    def pop(self):
        if self.top == None:
            print('stack is empty')
        else:
            poppednode = self.top
            self.top = self.top.ref
            poppednode.ref = None
            return poppednode.data
    # Returns the top node data
    def peak(self):
        if self.top == None:
            print('stack is empty')
        else:
            print('top element is', self.top.data)
         # Prints out the stack
    def display(self):
        if self.top == None:
            print("Stack Underflow")
        else:
            n = self.top
            print('\nstack elements are')
```

```
        while(n != None):
            print(n.data)
            n = n.ref
# Driver code
MyStack = Stack()
MyStack.display()
MyStack.push(11)
MyStack.display()
MyStack.push(22)
MyStack.display()
MyStack.push(33)
MyStack.display()
MyStack.push(44)
MyStack.display()
MyStack.pop()
MyStack.pop()
MyStack.display()
MyStack.peak()
```

**Output:**

```
Stack Underflow
stack elements are
11
stack elements are
22
11
stack elements are
33
22
11
stack elements are
44
33
22
11
stack elements are
22
11
top element is 22
```

# 8.QUEUES

## Aim:

To Write a Python program to implement Queues using Arrays and Linked list.

**Software Required:** IDLE (Python 3.11)

## Source Code:

**Queue using Arrays:**

```
class queue:
    def __init__(self):
        self.arr=[ ]
    def enqueue(self,item):
        self.arr.append(item)
        print('queue elements after enqueue',item,'is',self.arr)
    def dequeue(self):
        if len(self.arr)==0:
            return print('empty queue')
        removed=self.arr.pop(0)
        print('queue elements after dequeue',self.arr)
        return removed
s1=queue()
s1.enqueue(1)
s1.enqueue(2)
s1.enqueue(3)
s1.enqueue(4)
s1.enqueue(5)
s1.dequeue()
s1.dequeue()
s1.dequeue()
s1.dequeue()
s1.dequeue()
s1.dequeue()
s1.dequeue()
s1.dequeue()
```

**Output**

queue elements after enqueue 1 is [1]
queue elements after enqueue 2 is [1, 2]
queue elements after enqueue 3 is [1, 2, 3]
queue elements after enqueue 4 is [1, 2, 3, 4]
queue elements after enqueue 5 is [1, 2, 3, 4, 5]
queue elements after dequeue [2, 3, 4, 5]
queue elements after dequeue [3, 4, 5]
queue elements after dequeue [4, 5]
queue elements after dequeue [5]
queue elements after dequeue []
empty queue
empty queue
empty queue

**Queue using Linked List:**

```
# Python3 program to demonstrate linked list
# based implementation of queue
# A linked list (LL) node to store a queue entry
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
# A class to represent a queue
# The queue, front stores the front node
# of LL and rear stores the last node of LL
class Queue:
    def __init__(self):
        self.front = self.rear = None
 # Method to add an item to the queue
    def EnQueue(self, item):
        new_node = Node(item)
        if self.rear == None:
            self.front = self.rear =new_node
            return
        self.rear.next = new_node
        self.rear = new_node

    # Method to remove an item from queue
    def DeQueue(self):
        if self.rear==None:
            print("empty queue")
            return
        temp = self.front
        self.front = temp.next
        temp.next=None
        if(self.front == None):
            self.rear = None
        return temp.data

q = Queue()
q.EnQueue(10)
q.EnQueue(20)
q.DeQueue()
q.DeQueue()
q.EnQueue(30)
q.EnQueue(40)
q.EnQueue(50)
```

```
q.DeQueue()
print("Queue Front " + str(q.front.data))
print("Queue Rear " + str(q.rear.data))
```

**Output:**

```
Queue Front 40
Queue Rear 50
```

# 9.Double Ended Queues

## Aim:

To Write a Python program to implement Double Ended Queues using Collections module and Double Linked list.

**Software Required:** IDLE (Python 3.11)

## Source Code:

**Double Ended Queues implementation by using Collections Module:**

```
# Python code to demonstrate working of
# append(), appendleft(), pop(), and popleft()
# importing "collections" for deque operations
import collections
# initializing deque
de = collections.deque([1,2,3])

# using append() to insert element at right end
# inserts 4 at the end of deque
de.append(4)

# printing modified deque
print ("The deque after appending at right is : ")
print (de)

# using appendleft() to insert element at left end
# inserts 6 at the beginning of deque
de.appendleft(6)

# printing modified deque
print ("The deque after appending at left is : ")
print (de)

# using pop() to delete element from right end
# deletes 4 from the right end of deque
de.pop()

# printing modified deque
print ("The deque after deleting from right is : ")
print (de)

# using popleft() to delete element from left end
# deletes 6 from the left end of deque
de.popleft()
```

```
# printing modified deque
print ("The deque after deleting from left is : ")
print (de)
```

**Output:**

```
The deque after appending at right is :
deque([1, 2, 3, 4])
The deque after appending at left is :
deque([6, 1, 2, 3, 4])
The deque after deleting from right is :
deque([6, 1, 2, 3])
The deque after deleting from left is :
deque([1, 2, 3])
```

**Double Ended Queue using Double Linked List:**

```
class Node:
    def __init__(self, data):
        self.item = data
        self.next = None
        self.prev = None
class DoubleEndedQueue:
    def __init__(self):
        self.front = None
        self.rear= None
#display
    def traverse_list(self):
        if self.front is None:
            print("List has no element")
            return
        else:
            n = self.front
            print('double ended queue elements')
            while n is not None:
                print(n.item , "  ",end='')
                n = n.next
            print()
#insert at start
    def insert_at_front(self, data):
        new_node=Node(data)
        if self.front==self.rear==None:
            self.front=self.rear=new_node
            return
        new_node.next = self.front
```

```
        self.front.prev=new_node
        self.front= new_node
    # insert at end
    def insert_at_rear(self, data):
        new_node=Node(data)
        if self.front==self.rear==None:
            self.front=self.rear=new_node
            return
        self.rear.next = new_node
        new_node.prev=self.rear
        self.rear= new_node
    #del at start
    def delete_at_front(self):
        #no elements
        if self.front is None:
            print("The list has no element to delete")
            return
        #single element
        if self.front==self.rear:
            self.front==self.rear==None
            return
        #more than one
        self.front = self.front.next
        self.front.prev=None
    #del at end
    def delete_at_rear(self):
        #no elements
        if self.rear is None:
            print("The list has no element to delete")
            return
        #single element
        if self.front==self.rear:
            self.front==self.rear==None
            return
        #more than one
        self.rear = self.rear.prev
        self.rear.next=None

s1=DoubleEndedQueue()
s1.delete_at_rear()
s1.traverse_list()
s1.insert_at_front(2)
s1.traverse_list()
s1.insert_at_rear(3)
```

```
s1.traverse_list()
s1.insert_at_front(20)
s1.traverse_list()
s1.insert_at_rear(30)
s1.traverse_list()
s1.insert_at_front(200)
s1.traverse_list()
s1.insert_at_rear(300)
s1.traverse_list()
s1.delete_at_rear()
s1.traverse_list()
s1.delete_at_front()
s1.traverse_list()
s1.delete_at_rear()
s1.traverse_list()
s1.delete_at_front()
s1.traverse_list()
```

**Output:**

```
The list has no element to delete
List has no element
double ended queue elements
2
double ended queue elements
2   3
double ended queue elements
20   2   3
double ended queue elements
20   2   3   30
double ended queue elements
200   20   2   3   30
double ended queue elements
200   20   2   3   30   300
double ended queue elements
200   20   2   3   30
double ended queue elements
20   2   3   30
double ended queue elements
20   2   3
double ended queue elements
2   3
```

# 10. BINARY TREE IMPLEMENTATION AND ITS TRAVERSAL

## Aim:

To Write a Python program to implement Binary tree and its traversal operations.

**Software Required:** IDLE (Python 3.11)

## Source Code:

### Binary Tree using Array:

```
# Python3 implementation of tree using array
# numbering starting from 0 to n-1.

tree = [None] * 10
def root(key):
    if tree[0] != None:
        print("Tree already had root")
    else:
        tree[0] = key
def set_left(key, parent):
    if tree[parent] == None:
        print("Can't set child at", (parent * 2) + 1, ", no parent found")
    else:
        tree[(parent * 2) + 1] = key
def set_right(key, parent):
    if tree[parent] == None:
        print("Can't set child at", (parent * 2) + 2, ", no parent found")
    else:
        tree[(parent * 2) + 2] = key
def print_tree():
    for i in range(10):
        if tree[i] != None:
            print(tree[i], end="")
        else:
            print("-", end="")
    print()
 # Driver Code
root('A')
set_right('C', 0)
set_left('D', 1)
set_right('E', 1)
```

```
set_right('F', 2)
print_tree()
```

## Output:

**Can't set child at 3 , no parent found**
**Can't set child at 4 , no parent found**
**A-C---F---**

## Binary Tree using Linked List and its Traversal:

```
class Node:
    def __init__(self):
        self._element = None
        self._left = None
        self._right = None

class dummynode:
    def __init__(self):
        self._root = None

class BinaryTree:
    def __init__(self):
        self._root = Node()

    def maketree(self,element,left, right):
        self._root._element = element
        self._root._left = left._root
        self._root._right = right._root

    def inorder(self,root):
        if root:
            self.inorder(root._left)
            print(root._element,end=' ')
            self.inorder(root._right)

    def preorder(self,root):
        if root:
            print(root._element,end=' ')
            self.preorder(root._left)
            self.preorder(root._right)

    def postorder(self,root):
        if root:
            self.postorder(root._left)
```

```
            self.postorder(root._right)
            print(root._element, end=' ')

n = dummynode()
a = BinaryTree()
b = BinaryTree()
c = BinaryTree()
a.maketree(10,b,c)
d = BinaryTree()
e = BinaryTree()
b.maketree(20,d,e)
f = BinaryTree()
g = BinaryTree()
c.maketree(30,f,g)
d.maketree(40,n,n)
e.maketree(50,n,n)
f.maketree(60,n,n)
g.maketree(70,n,n)
print('Inorder Traversal')
a.inorder(a._root)
print()
print('Preorder Traversal')
a.preorder(a._root)
print()
print('Postorder Traversal')
a.postorder(a._root)
```

**Output:**

```
Inorder Traversal
40 20 50 10 60 30 70
Preorder Traversal
10 20 40 50 30 60 70
Postorder Traversal
40 50 20 60 70 30 10
```

# 11. BINARY SEARCH TREE

## Aim:

To Write a Python program to implement Binary search tree operations.

## Software Required: IDLE (Python 3.11)

## Source Code:

## Binary Search Tree Insertion:

```
class _Node:
    def __init__(self, element, left=None, right=None):
        self._element = element
        self._left = left
        self._right = right

class BinarySearchTree:
    def __init__(self):
        self._root = None

    def insert(self,troot,e):
        temp = None
        while troot:
            temp = troot
            if e == troot._element:
                return
            elif e < troot._element:
                troot = troot._left
            elif e > troot._element:
                troot = troot._right
        n = _Node(e)
        if self._root:
            if e < temp._element:
                temp._left = n
            else:
                temp._right = n
        else:
            self._root = n
```

```
def inorder(self,troot):
        if troot:
            self.inorder(troot._left)
            print(troot._element,end=' ')
            self.inorder(troot._right)
B = BinarySearchTree()
B.insert(B._root,50)
B.insert(B._root,30)
B.insert(B._root,80)
B.insert(B._root,10)
B.insert(B._root,40)
B.insert(B._root,60)
B.insert(B._root,90)
B.insert(B._root,90)
B.insert(B._root,61)
B.inorder(B._root)
```

**Output:**

**10 30 40 50 60 61 80 90**

## Binary Search Tree Searching:

```
class _Node:
    def __init__(self, element, left=None, right=None):
        self._element = element
        self._left = left
        self._right = right
class BinarySearchTree:
    def __init__(self):
        self._root = None
    def insert(self,troot,e):
        temp = None
        while troot:
            temp = troot
            if e < troot._element:
                troot = troot._left
            elif e > troot._element:
                troot = troot._right
        n = _Node(e)
        if self._root:
            if e < temp._element:
```

```
                temp._left = n
            else:
                temp._right = n
        else:
            self._root = n
    def search(self,key):
        troot = self._root
        while troot:
            if key == troot._element:
                return True
            elif key < troot._element:
                troot = troot._left
            elif key > troot._element:
                troot = troot._right
        return False
    def inorder(self,troot):
        if troot:
            self.inorder(troot._left)
            print(troot._element,end=' ')
            self.inorder(troot._right)
B = BinarySearchTree()
B.insert(B._root,50)
B.insert(B._root,30)
B.insert(B._root,80)
B.insert(B._root,10)
B.insert(B._root,40)
B.insert(B._root,60)
B.insert(B._root,90)
B.inorder(B._root)
print()
print("is element 60 found?",B.search(60))
print(B.search(30))
print(B.search(70))
```

## Output:

**10 30 40 50 60 80 90**
**is element 60 found? True**
**True**
**False**

## Binary Search Tree Deletion:

```
class _Node:
    __slots__ = '_element', '_left', '_right'
    def __init__(self, element, left=None, right=None):
        self._element = element
        self._left = left
        self._right = right
class BinarySearchTree:
    def __init__(self):
        self._root = None
    def insert(self,troot,e):
        temp = None
        while troot:
            temp = troot
            if e == troot._element:
                return
            elif e < troot._element:
                troot = troot._left
            elif e > troot._element:
                troot = troot._right
        n = _Node(e)
        if self._root:
            if e < temp._element:
                temp._left = n
            else:
                temp._right = n
        else:
            self._root = n
    def delete(self,e):
        p = self._root
        pp = None
        while p and p._element != e:
            pp = p
            if e < p._element:
                p = p._left
            else:
                p = p._right
        if not p:
            return False
        if p._left and p._right:
```

```
            s = p._left
            ps = p
            while s._right:
                ps = s
                s = s._right
            p._element = s._element
            p = s
            pp = ps
        c = None
        if p._left:
            c = p._left
        else:
            c = p._right
        if p == self._root:
            self._root = None
        else:
            if p == pp._left:
                pp._left = c
            else:
                pp._right = c
    def inorder(self,troot):
        if troot:
            self.inorder(troot._left)
            print(troot._element,end=' ')
            self.inorder(troot._right)
B = BinarySearchTree()
B.insert(B._root,50)
B.insert(B._root,30)
B.insert(B._root,80)
B.insert(B._root,10)
B.insert(B._root,40)
B.insert(B._root,60)
B.insert(B._root,90)
B.inorder(B._root)
B.delete(50)
print()
B.inorder(B._root)
```

**Output:**

**10 30 40 50 60 80 90**

**10 30 40 60 80 90**

# 12. GRAPHS

## Aim:

To Write a Python program create different types graphs using Graph Abstract Data Type (ADT).

## Software Required: IDLE (Python 3.11)

## Source Code:

## Graph ADT:

```
import numpy as np
class Graph:
    def __init__(self, vertices):
        self._vertices = vertices
        self._adjMat = np.zeros((vertices, vertices))
    def insert_edge(self, u, v, x=1):
        self._adjMat[u][v] = x
    def remove_edge(self, u, v):
        self._adjMat[u][v] = 0
    def exist_edge(self, u, v):
        return self._adjMat[u][v] != 0
    def vertex_count(self):
        return self._vertices
    def edge_count(self):
        count = 0
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
                    count = count + 1
        return count
    def vertices(self):
        for i in range(self._vertices):
            print(i,end=' ')
        print()
    def edges(self):
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
```

```
                print(i,'--',j)
    def outdegree(self, vertex):
        count = 0
        for j in range(self._vertices):
            if self._adjMat[vertex][j] != 0:
                count = count + 1
        return count
    def indegree(self, vertex):
        count = 0
        for i in range(self._vertices):
            if self._adjMat[i][vertex] != 0:
                count = count + 1
        return count
    def display_adjMat(self):
        print(self._adjMat)
```

## Creation of Directed Graph:

```
import numpy as np
class Graph:
    def __init__(self, vertices):
        self._adjMat = np.zeros((vertices, vertices))
        self._vertices = vertices
    def insert_edge(self, u, v, x=1):
        self._adjMat[u][v] = x
    def remove_edge(self, u, v):
        self._adjMat[u][v] = 0
    def exist_edge(self, u, v):
        return self._adjMat[u][v] != 0
    def vertex_count(self):
        return self._vertices
    def edge_count(self):
        count = 0
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
                    count = count + 1
        return count
    def vertices(self):
        for i in range(self._vertices):
```

```
            print(i,end=' ')
        print()
    def edges(self):
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
                    print(i,'--',j)
    def outdegree(self, v):
        count = 0
        for j in range(self._vertices):
            if not self._adjMat[v][j] == 0:
                count = count + 1
        return count
    def indegree(self, v):
        count = 0
        for i in range(self._vertices):
            if not self._adjMat[i][v] == 0:
                count = count + 1
        return count
    def display_adjMat(self):
        print(self._adjMat)
G = Graph(4)
print('Graph Adjacency Matrix')
G.display_adjMat()
print('Vertices: ', G.vertex_count())
G.insert_edge(0, 1)
G.insert_edge(0, 2)
G.insert_edge(1, 2)
G.insert_edge(2, 3)
print('Graph Adjacency Matrix')
G.display_adjMat()
print('Edges Count:', G.edge_count())
print('Vertices:')
G.vertices()
print('Edges:')
G.edges()
print('Edges between 1--3:', G.exist_edge(1,3))
print('Edges between 1--2:', G.exist_edge(1,2))
print('Edges between 2--1:', G.exist_edge(2,1))
print('Degree of vertex 2:',G.indegree(2)+G.outdegree(2))
```

```
print('In-Degree of vertex 2:',G.indegree(2))
print('Out-Degree of vertex 2:',G.outdegree(2))
print('Graph Adjacency Matrix')
G.display_adjMat()
G.remove_edge(1,2)
print('Graph Adjacency Matrix')
G.display_adjMat()
print('Edges between 1--2:', G.exist_edge(1,2))
```

## Creation of Undirected Graph:

```
import numpy as np
class Graph:
    def __init__(self, vertices):
        self._vertices = vertices
        self._adjMat = np.zeros((vertices, vertices))
    def insert_edge(self, u, v, x=1):
        self._adjMat[u][v] = x
    def remove_edge(self, u, v):
        self._adjMat[u][v] = 0
    def exist_edge(self, u, v):
        return self._adjMat[u][v] != 0
    def vertex_count(self):
        return self._vertices
    def edge_count(self):
        count = 0
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
                    count = count + 1
        return count
    def vertices(self):
        for i in range(self._vertices):
            print(i,end=' ')
        print()
    def edges(self):
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
                    print(i,'--',j)
```

```
    def outdegree(self, v):
        count = 0
        for j in range(self._vertices):
            if self._adjMat[v][j] != 0:
                count = count + 1
        return count
    def indegree(self, v):
        count = 0
        for i in range(self._vertices):
            if self._adjMat[i][v] != 0:
                count = count + 1
        return count
    def display_adjMat(self):
        print(self._adjMat)

G = Graph(4)
print('Graph Adjacency Matrix')
G.display_adjMat()
print('Vertices: ', G.vertex_count())
print('Edges Count:', G.edge_count())
G.insert_edge(0, 1)
G.insert_edge(0, 2)
G.insert_edge(1, 0)
G.insert_edge(1, 2)
G.insert_edge(2, 0)
G.insert_edge(2, 1)
G.insert_edge(2, 3)
G.insert_edge(3, 2)
print('Graph Adjacency Matrix')
G.display_adjMat()
print('Edges Count:', G.edge_count())
print('Vertices:')
G.vertices()
print('Edges:')
G.edges()
print('Edges between 1--3:', G.exist_edge(1,3))
print('Edges between 1--2:', G.exist_edge(1,2))
print('Degree of vertex 2:',G.indegree(2))
print('Graph Adjacency Matrix')
G.display_adjMat()
```

```
G.remove_edge(1,2)
print('Graph Adjacency Matrix')
G.display_adjMat()
print('Edges between 1--2:', G.exist_edge(1,2))
```

## Creation of Weighted Directed Graph:

```
import numpy as np
class Graph:
    def __init__(self, vertices):
        self._adjMat = np.zeros((vertices, vertices))
        self._vertices = vertices
    def insert_edge(self, u, v, x=1):
        self._adjMat[u][v] = x
    def remove_edge(self, u, v):
        self._adjMat[u][v] = 0
    def exist_edge(self, u, v):
        return self._adjMat[u][v] != 0
    def vertex_count(self):
        return self._vertices
    def edge_count(self):
        count = 0
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
                    count = count + 1
        return count
    def vertices(self):
        for i in range(self._vertices):
            print(i,end=' ')
        print()
    def edges(self):
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
                    print(i,'--',j)
    def outdegree(self, v):
        count = 0
        for j in range(self._vertices):
            if not self._adjMat[v][j] == 0:
```

```
                count = count + 1
        return count
    def indegree(self, v):
        count = 0
        for i in range(self._vertices):
            if not self._adjMat[i][v] == 0:
                count = count + 1
        return count
    def display_adjMat(self):
        print(self._adjMat)

G = Graph(4)
print('Graph Adjacency Matrix')
G.display_adjMat()
print('Vertices: ', G.vertex_count())
G.insert_edge(0, 1, 26)
G.insert_edge(0, 2, 16)
G.insert_edge(1, 2, 12)
G.insert_edge(2, 3, 8)
print('Graph Adjacency Matrix')
G.display_adjMat()
print('Edges Count:', G.edge_count())
print('Vertices:')
G.vertices()
print('Edges:')
G.edges()
print('Edges between 1--3:', G.exist_edge(1,3))
print('Edges between 1--2:', G.exist_edge(1,2))
print('Edges between 2--1:', G.exist_edge(2,1))
print('Degree of vertex 2:',G.indegree(2)+G.outdegree(2))
print('In-Degree of vertex 2:',G.indegree(2))
print('Out-Degree of vertex 2:',G.outdegree(2))
print('Graph Adjacency Matrix')
G.display_adjMat()
G.remove_edge(1,2)
print('Graph Adjacency Matrix')
G.display_adjMat()
print('Edges between 1--2:', G.exist_edge(1,2))
```

## Creation of Weighted Undirected Graph:

```
import numpy as np
class Graph:
    def __init__(self, vertices):
        self._adjMat = np.zeros((vertices, vertices))
        self._vertices = vertices
    def insert_edge(self, u, v, x=1):
        self._adjMat[u][v] = x
    def remove_edge(self, u, v):
        self._adjMat[u][v] = 0
    def exist_edge(self, u, v):
        return self._adjMat[u][v] != 0
    def vertex_count(self):
        return self._vertices
    def edge_count(self):
        count = 0
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
                    count = count + 1
        return count
    def vertices(self):
        for i in range(self._vertices):
            print(i,end=' ')
        print()
    def edges(self):
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
                    print(i,'--',j)
    def outdegree(self, v):
        count = 0
        for j in range(self._vertices):
            if not self._adjMat[v][j] == 0:
                count = count + 1
        return count
    def indegree(self, v):
        count = 0
        for i in range(self._vertices):
```

```
            if not self._adjMat[i][v] == 0:
                count = count + 1
        return count
    def display_adjMat(self):
        print(self._adjMat)

G = Graph(4)
print('Graph Adjacency Matrix')
G.display_adjMat()
print('Vertices: ', G.vertex_count())
G.insert_edge(0, 1, 26)
G.insert_edge(0, 2, 16)
G.insert_edge(1, 0, 26)
G.insert_edge(1, 2, 12)
G.insert_edge(2, 0, 16)
G.insert_edge(2, 1, 12)
G.insert_edge(2, 3, 8)
G.insert_edge(3, 2, 8)
print('Graph Adjacency Matrix')
G.display_adjMat()
print('Edges Count:', G.edge_count())
print('Vertices:')
G.vertices()
print('Edges:')
G.edges()
print('Edges between 1--3:', G.exist_edge(1,3))
print('Edges between 1--2:', G.exist_edge(1,2))
print('Degree of vertex 2:',G.indegree(2))
print('Graph Adjacency Matrix')
G.display_adjMat()
G.remove_edge(1,2)
print('Graph Adjacency Matrix')
G.display_adjMat()
print('Edges between 1--2:', G.exist_edge(1,2))
```

**Output:**

# 13. GRAPH TRAVERSAL USING DEPTH FIRST SEARCH

## Aim:

To Write a Python program to Travers in a graph using depth first search.

**Software Required:** IDLE (Python 3.11)

## Source Code:

```
import numpy as np
class Graph:
    def __init__(self, vertices):
        self._adjMat = np.zeros((vertices, vertices))
        self._vertices = vertices
        self._visited = [0] * vertices
    def insert_edge(self, u, v, x=1):
        self._adjMat[u][v] = x
    def remove_edge(self, u, v):
        self._adjMat[u][v] = 0
    def exist_edge(self, u, v):
        return self._adjMat[u][v] != 0
    def vertex_count(self):
        return self._vertices
    def edge_count(self):
        count = 0
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
                    count += 1
        return count
    def vertices(self):
        for i in range(self._vertices):
            print(i,end=' ')
        print()
    def edges(self):
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
```

```
                print(i,'--',j)
    def outdegree(self, v):
        count = 0
        for j in range(self._vertices):
            if not self._adjMat[v][j] == 0:
                count += 1
        return count
    def indegree(self, v):
        count = 0
        for j in range(self._vertices):
            if not self._adjMat[j][v] == 0:
                count += 1
        return count
    def display_adjMat(self):
        print(self._adjMat)
    def DFS(self,s):
        if self._visited[s]==0:
            print(s,end=' ')
            self._visited[s]=1
            for j in range(self._vertices):
                if self._adjMat[s][j]==1 and self._visited[j]==0:
                    self.DFS(j)

G = Graph(7)
G.insert_edge(0, 1)
G.insert_edge(0, 5)
G.insert_edge(0, 6)
G.insert_edge(1, 0)
G.insert_edge(1, 2)
G.insert_edge(1, 5)
G.insert_edge(1, 6)
G.insert_edge(2, 3)
G.insert_edge(2, 4)
G.insert_edge(2, 6)
G.insert_edge(3, 4)
G.insert_edge(4, 2)
G.insert_edge(4, 5)
G.insert_edge(5, 2)
G.insert_edge(5, 3)
G.insert_edge(6, 3)
```

```
print('Edges:')
G.edges()
print('DFS:')
G.DFS(0)
```

**Output:**

# 14.GRPAH TRAVERSAL USING BREADTH FIRST SEARCH

## Aim:

To Write a Python program to Travers in a graph using breadth first search.

## Software Required: IDLE (Python 3.11)

## Source Code:

```
from QueuesLinked import QueuesLinked
import numpy as np
class Graph:
    def __init__(self, vertices):
        self._adjMat = np.zeros((vertices, vertices))
        self._vertices = vertices
    def insert_edge(self, u, v, x=1):
        self._adjMat[u][v] = x
    def remove_edge(self, u, v):
        self._adjMat[u][v] = 0
    def exist_edge(self, u, v):
        return self._adjMat[u][v] != 0
    def vertex_count(self):
        return self._vertices
    def edge_count(self):
        count = 0
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
                    count += 1
        return count
    def vertices(self):
        for i in range(self._vertices):
            print(i,end=' ')
        print()

    def edges(self):
        for i in range(self._vertices):
            for j in range(self._vertices):
                if self._adjMat[i][j] != 0:
                    print(i,'--',j)
```

```
    def outdegree(self, v):
        count = 0
        for j in range(self._vertices):
            if not self._adjMat[v][j] == 0:
                count += 1
        return count
    def indegree(self, v):
        count = 0
        for j in range(self._vertices):
            if not self._adjMat[j][v] == 0:
                count += 1
        return count
    def display_adjMat(self):
        print(self._adjMat)
    def BFS(self, s):
        i = s
        q = QueuesLinked()
        visited = [0] * self._vertices
        print(i,end=' ')
        visited[i] = 1
        q.enqueue(i)
        while not q.isempty():
            i = q.dequeue()
            for j in range(self._vertices):
                if self._adjMat[i][j] == 1 and visited[j] == 0:
                    print(j,end=' ')
                     visited[j] = 1
                    q.enqueue(j)

G = Graph(7)
G.insert_edge(0, 1)
G.insert_edge(0, 5)
G.insert_edge(0, 6)
G.insert_edge(1, 0)
G.insert_edge(1, 2)
G.insert_edge(1, 5)
G.insert_edge(1, 6)
G.insert_edge(2, 3)
G.insert_edge(2, 4)
G.insert_edge(2, 6)
```

```
G.insert_edge(3, 4)
G.insert_edge(4, 2)
G.insert_edge(4, 5)
G.insert_edge(5, 2)
G.insert_edge(5, 3)
G.insert_edge(6, 3)
print('Edges:')
G.edges()
print('BFS:')
G.BFS(0)
```

# Appendix

## Data Structures Cheat Sheet

**Python - Data Structure**

### Data Types

It is a way of organizing data that contains the items stored and their relationship to each other

The areas in which Data Structures are applied:

- Compiler design
- Operating system
- Database Management System
- Statistical Analysis Package
- Numerical Analysis
- Graphics
- Artificial Intelligence
- Simulations

Data structures can be used in the following areas:

- RDBMS: Array ( Array of structure)
- Network data model: Graph
- Hierarchical Data model: Trees

### Types of Data Structures

**Primitive Data Structures:**

- Integer: It is used to represent numeric data, more specifically whole numbers from negative infinity to infinity. Eg: 4, 5, -1 etc
- Float: It stands for floating point number. Eg: 1.1,2.3,9.3 etc
- String: It is a collection of Alphabets, words or other characters. In python it can be created by using a pair of single or double quotes for the sequence.

```
Eg: x = 'Cake'
y = "Cookie"
```

**Certain operations can be performed on a string:**

- We can use * to repeat the string for a specific number of times.

```
Eg: x*2
```

- String can be sliced, that is to select parts of the string.

```
Eg: Coke
z1 = x[2:]
print(z1)
# Slicing
z2 = y[0] + y[1]
print(z2)
Output: ke
        Co
```

- To capitalize the strings

```
Eg: str.capitalize('cookie')
```

- To retrieve the length of the strings

```
Eg:
str1 = "Cake 4 U"
str2 = "404"
len(str1)
```

- To replace parts of a string with another string
  - Eg: str1.replace('4 U',str2)
  - Boolean: It is a built-in data type that can take the values TRUE or FALSE

**Non- Primitive Data Structures:**

- Array: It is a compact way of collecting data types where all entries must be of the same data type.

```
Syntax of writing an array in python:
import array as arr
a = arr.array('i',[3,6,9])
type(a)
```

- Linked list: List in Python is used to store collection of heterogeneous items. It is described using the square brackets [] and hold elements separated by comma

```
Eg: x = [] # Empty list
type(x)
```

- The list can be classified into linear and non-linear data structures
- Linear data structures contain Stacks and queues
- Non-linear data structures contains Graphs and Trees

- Stack: It is a container of objects that can be inserted or removed according to LIFO(Last In First Out) concept. pop() method is used during disposal in Python

```
Eg: stack.pop() # Bottom -> 1 -> 2 -> 3 -> 4 -> 5 (Top)
    stack.pop() # Bottom -> 1 -> 2 -> 3 -> 4 (Top)
    print(stack)
```

- Queue: It is a container of objects that can be inserted or removed according to FIFO(First In First Out) concept.
- Graph: It is a data structure that consists of a finite set of vertices called nodes, and a finite set of ordered pair (u,v) called edges. It can be classified as direction and weight
- Binary Tree: Tree is a hierarchical data structure. Here each node has at most two children
- Binary Search Tree: It provides moderate access/ search and moderate insertion/deletion
- Heap: It is a complete tree and is suitable to be stored in an array, It is either MIN or Max
- Hashing: Collection of items that are stored in a way that it becomes easy to find them is hashing

### Lists and Tuples in Python

Ordered sequence of values indexed by integer numbers.

Tuples are immutable

- To initialize empty list /tuple:

```
Syntax: Lists: myList = []
Tuples: myTuple = ()
```

- To get an element in position x in list/tuple:

```
Syntax: "x" in myListOrTuple
```

- Index of element 'X' of list/tuple:

```
Syntax: myListOrTuple.index("x") -
- If not found, throws a ValueError exception
```

- Number of occurance of X in list/tuple:

```
Syntax: myListOrTuple.count("x")
```

- Update an item of List/tuple:

```
Syntax: Lists: myList[x] = "x"
Tuples: tuples are immutable!
```

- Remove element in position X of list/tuple:

```
Syntax: Lists: del myList[x]
Tuples: tuples are immutable!
```

- To specify size of tuple/list:

```
Syntax: len(myListOrTuple)
```

- Remove element in position X of list/tuple:

```
Syntax: Lists: del myList[x]
Tuples: tuples are immutable!
```

- Concatenate two lists/tuples:

```
Lists: myList1 + myList2
Tuples: myTuple1 + myTuple2
Concatenating a List and a Tuple will produce a TypeError exception
```

- Insert element in position x of a list/tuple:

```
Syntax: Lists: myList.insert(x,"value")
Tuples: tuples are immutable!
```

- Append "x" to a list/tuple:

```
Syntax: Lists: myList.append("x")
Tuples: tuples are immutable!
```

- Convert a list/tuple to tuple/list:

```
Syntax: List to Tuple: tuple(myList)
Tuple to List: list(myTuple)
```

### Sets

It is an unordered collection with no duplicate elements. It supports mathematical operations like union, intersection, difference and symmetric difference.

- To initialize an empty set:

```
Syntax: mySet = set()
```

- Initialize a non empty set:

```
Syntax: mySet = set(element1,element2...)
```

- To add element X to the set:

```
Syntax: mySet.add("x")
```

- Remove element "x" from a set:

```
Syntax:
Method 1: mySet.remove("x") --
If "x" is not present, raises a KeyErorr
Method 2: mySet.discard("x") --
Removes the element, if present
```

- Remove every element from the set:

```
Syntax: mySet.clear()
```

- Check if "x" is in the set:

```
Syntax: "x" in mySet
```

- Size of the sets:

```
Syntax: len(mySet)
```

- Check if the dictionary has key "k"

```
Syntax: "k" in myDict
```

- Get the list of keys

```
Syntax: myDict.keys()
```

- Get the size of the dictionary

```
Syntax: len(myDict)
```

- Delete element with key "k" from the dictionary

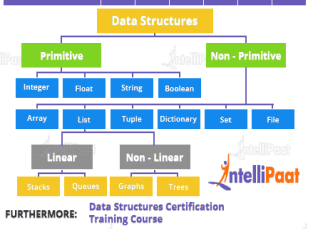
```
Syntax: del myDict["k"]
```

- Delete all the elements in the dictionary

```
Syntax: myDict.clear()
```

| Algorithm | Best case | Average case | Worst case | Remarks |
|---|---|---|---|---|
| Selection sort | $\frac{1}{2} n^2$ | $\frac{1}{2} n^2$ | $\frac{1}{2} n^2$ | n exchanges, quadratic is the best case |
| Bubble sort | n | $\frac{1}{2} n^2$ | $\frac{1}{2} n^2$ | Rarely useful, Insertion sort can be used instead |
| Insertion sort | n | $\frac{1}{4} n^2$ | $\frac{1}{2} n^2$ | Used for small or partialsorted arrays |
| Shell sort | $n \log_3 n$ | unknown | $c\, n^{3/2}$ | Tight code, Sub quadratic |
| Merge sort | $\frac{1}{2} n \lg n$ | $n \lg n$ | $n \lg n$ | n log n guarantee; stable |
| Quick sort | $n \lg n$ | $2\, n \ln n$ | $\frac{1}{2} n^2$ | n log n probabilistic guarantee; fastest in practice |
| Heap sort | $n^1$ | $2\, n \lg n$ | $2\, n \lg n$ | n log n guarantee; in place |

| Data Structure | Worst case | | | Average case | | |
|---|---|---|---|---|---|---|
| | Search | Insert | Delete | Search | Insert | Delete |
| Sequential search | n | n | n | n | n | n |
| Binary search | log n | n | n | log n | n | n |
| Binary search tree | n | n | n | log n | log n | sqrt(n) |
| Red-black BST | log n | log n | log n | log n | log n | log n |
| Hash table | n | n | n | $1^{\dagger}$ | $1^{\dagger}$ | $1^{\dagger}$ |




FURTHERMORE: Data Structures Certification Training Course

## References


1. Michael T. Good Rich, Roberto Tamassia, Michael H. Gold wasser."Data Structures & Algorithms", John Wiley & sons 2013.
2. Y. Daniel Liang, "Introduction to programming using python", Pearson, 2013.
3. Bill Lubanovic," Introducing Python Modern Computing in Simple Packages", O Reilly Publication, 1st Edition, 2015.
4. R. Nageswara Rao, "Core python programming", Dream tech, 2017
5. Mark Summer field, "Programming in Python3", Pearson Education, 2nd Edition.
6. Magnus Lie Hetland "Beginning Python – From Novice to Professional", A Press Publication, 3rd Edition, 2017.
7. https://www.javatpoint.com/python-tutorial
8. https://www.javatpoint.com/data-structure-tutorial
9. https://www.geeksforgeeks.org/python-programming-language-tutorial/
10. https://www.udemy.com/course/learning-data-structures-algorithms-in-python-from-scratch/learn/lecture/21429286?start=0#overview

# Python 3 Cheat Sheet

*Latest version on :*
*https://perso.limsi.fr/pointal/python:memento*

## Base Types

*integer, float, boolean, string, bytes*

```
int 783  0 -192  0b010 0o642 0xF3
          zero    binary  octal  hexa
float 9.23 0.0  -1.7e-6
                    ×10^-6
bool True  False
str "One\nTwo"   escaped new line
'I\'m'   escaped '
"""X\tY\tZ
1\t2\t3"""   escaped tab
bytes b"toto\xfe\775"
        hexadecimal  octal
```

*Multiline string:*

*immutables*

## Container Types

- **ordered sequences**, fast index access, repeatable values

```
list [1,5,9]   ["x",11,8.9]   ["mot"]   []
tuple (1,5,9)   11,"y",7.4   ("mot",)   ()
```

*Non modifiable values (immutables)* — *expression with only comas →* `tuple`

```
str bytes   (ordered sequences of chars / bytes)
""   b""
```

- **key containers**, no *a priori* order, fast key access, each key is unique

```
dictionary dict {"key":"value"}   dict(a=3,b=4,k="v")   {}
(key/value associations) {1:"one",3:"three",2:"two",3.14:"π"}
collection set {"key1","key2"}   {1,9,3,0}   set()
```

*keys=hashable values (base types, immutables…)*  `frozenset` *immutable set*  *empty*

## Identifiers

*for variables, functions, modules, classes… names*

`a…zA…Z_` followed by `a…zA…Z_0…9`

- diacritics allowed but should be avoided
- language keywords forbidden
- lower/UPPER case discrimination

```
☺ a toto x7 y_max BigOne
☹ 8y and for
```

## Variables assignment

`=`

assignment ⇔ **binding** of a *name* with a *value*

*1) evaluation of right side expression value*
*2) assignment in order with left side names*

```
x=1.2+8+sin(y)
a=b=c=0   assignment to same value
y,z,r=9.2,-7.6,0   multiple assignments
a,b=b,a   values swap
a,*b=seq
*a,b=seq   unpacking of sequence in item and list
x+=3   increment ⇔ x=x+3
x-=2   decrement ⇔ x=x-2
x=None   « undefined » constant value
del x   remove name x
```

and `*=` `/=` `%=` …

## Conversions

`type(`*expression*`)`

```
int("15") → 15
int("3f",16) → 63   can specify integer number base in 2nd parameter
int(15.56) → 15   truncate decimal part
float("-11.24e8") → -1124000000.0
round(15.56,1)→15.6   rounding to 1 decimal (0 decimal → integer number)
bool(x)  False for null x, empty container x , None or False x ; True for other x
str(x)→"…"   representation string of x for display (cf. formatting on the back)
chr(64)→'@'   ord('@')→64   code ↔ char
repr(x)→"…"   literal representation string of x
bytes([72,9,64]) → b'H\t@'
list("abc") → ['a','b','c']
dict([(3,"three"),(1,"one")]) → {1:'one',3:'three'}
set(["one","two"]) → {'one','two'}
separator str and sequence of str → assembled str
   ':'.join(['toto','12','pswd']) → 'toto:12:pswd'
str splitted on whitespaces → list of str
   "words with  spaces".split() → ['words','with','spaces']
str splitted on separator str → list of str
   "1,4,8,2".split(",") → ['1','4','8','2']
sequence of one type → list of another type (via list comprehension)
   [int(x) for x in ('1','29','-3')] → [1,29,-3]
```

## Sequence Containers Indexing

*for lists, tuples, strings, bytes…*

```
negative index   -5  -4  -3  -2  -1
positive index    0   1   2   3   4
lst=[10, 20, 30, 40, 50]
positive slice    0   1   2   3   4   5
negative slice   -5  -4  -3  -2  -1
```

**Items count**

`len(lst)→5`

**index from 0**
(here from 0 to 4)

Individual access to **items** via `lst[`*index*`]`

```
lst[0]→10   ⇒ first one   lst[1]→20
lst[-1]→50   ⇒ last one   lst[-2]→40
```

*On mutable sequences (`list`), remove with* `del lst[3]` *and modify with assignment* `lst[4]=25`

Access to **sub-sequences** via `lst[`*start slice*`:`*end slice*`:`*step*`]`

```
lst[:-1]→[10,20,30,40]   lst[::-1]→[50,40,30,20,10]   lst[1:3]→[20,30]   lst[:3]→[10,20,30]
lst[1:-1]→[20,30,40]   lst[::-2]→[50,30,10]   lst[-3:-1]→[30,40]   lst[3:]→[40,50]
lst[::2]→[10,30,50]   lst[:]→[10,20,30,40,50]   shallow copy of sequence
```

*Missing slice indication → from start / up to end.*
*On mutable sequences (`list`), remove with* `del lst[3:5]` *and modify with assignment* `lst[1:4]=[15,25]`

## Boolean Logic

Comparisons : `<` `>` `<=` `>=` `==` `!=` (≤ ≥ = ≠)
*(boolean results)*

`a and b` logical and — *both simulta-neously*

`a or b` logical or — *one or other or both*

pitfall : `and` *and* `or` *return* *value* *of* `a` *or of* `b` *(under shortcut evaluation).*
⇒ *ensure that* `a` *and* `b` *are booleans.*

`not a` logical not

`True` `False` — True and False constants

## Statements Blocks

```
parent statement:
    statement block 1…
    ⋮
parent statement:
    statement block2…
    ⋮
next statement after block 1
```

indentation !

*configure editor to insert 4 spaces in place of an indentation tab.*

## Modules/Names Imports

*module* `truc` ⇔ *file* `truc.py`

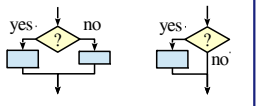
```
from monmod import nom1,nom2 as fct
   →direct access to names, renaming with as
import monmod   →access via monmod.nom1 …
```

modules and packages searched in *python path* (cf `sys.path`)

## Conditional Statement

*statement block executed only if a condition is true*

`if` *logical condition*`:`
→ *statements block*




Can go with several *elif*, *elif*... and only one final *else*. Only the block of first true condition is executed.

```
if age<=18:
    state="Kid"
elif age>65:
    state="Retired"
else:
    state="Active"
```

*with a var* `x`*:*
`if bool(x)==True:` ⇔ `if x:`
`if bool(x)==False:` ⇔ `if not x:`

## Maths

*floating numbers… approximated values*

Operators: `+` `-` `*` `/` `//` `%` `**`
Priority `(…)`
× ÷ ↑ ↑ $a^b$
integer ÷ ÷ remainder

`@` → matrix × *python3.5+*`numpy`

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
pow(4,3)→64.0
```

*usual order of operations*

*angles in radians*

```
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
sqrt(81)→9.0   √
log(e**2)→2.0
ceil(12.5)→13
floor(12.5)→12
```

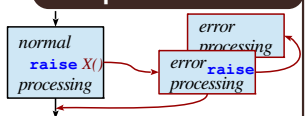
modules `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, *etc. (cf. doc)*

## Exceptions on Errors

Signaling an error:
`raise` *ExcClass(…)*

Errors processing:

```
try:
    →normal procesising block
except Exception as e:
    →error processing block
```




`finally` *block for final processing in all cases.*

## Conditional Loop Statement

*statements block executed **as long as** condition is true*

```
while logical condition:
    statements block
```


```
s = 0          # initializations before the loop
i = 1          # condition with a least one variable value (here i)
while i <= 100:
    s = s + i**2
    i = i + 1  # make condition variable change !
print("sum:",s)
```

*Algo:*

$$s = \sum_{i=1}^{i=100} i^2$$

*beware of infinite loops!*

### Loop Control

`break` *immediate exit*
`continue` *next iteration*
`else` *block for **normal** loop exit.*

## Display

```
print("v=",3,"cm :",x,",",y+4)
```

items to display : literal values, variables, expressions

`print` options:

- `sep=" "` items separator, default space
- `end="\n"` end of print, default new line
- `file=sys.stdout` print to file, default standard output

## Input

```
s = input("Instructions:")
```

`input` always returns a **string**, convert it to required type (cf. boxed *Conversions* on the other side).

## Generic Operations on Containers

`len(c)` → items count
`min(c)` `max(c)` `sum(c)`
`sorted(c)` → `list` sorted *copy*
`val in c` → boolean, membership operator `in` (absence `not in`)
`enumerate(c)` → *iterator* on (index, value)
`zip(c1,c2…)` → *iterator* on tuples containing $c_i$ items at same index
`all(c)` → `True` if **all** `c` items evaluated to true, else `False`
`any(c)` → `True` if **at least one** item of `c` evaluated true, else `False`

*Note: For dictionaries and sets, these operations use **keys**.*

*Specific to **ordered sequences containers** (lists, tuples, strings, bytes…)*

`reversed(c)` → *inversed iterator* `c*5`→ duplicate `c+c2`→ concatenate
`c.index(val)` → *position* `c.count(val)` → *events count*

```
import copy
copy.copy(c) → shallow copy of container
copy.deepcopy(c) → deep copy of container
```

## Operations on Lists

modify original list

| | |
|---|---|
| `lst.append(val)` | add item at end |
| `lst.extend(seq)` | add sequence of items at end |
| `lst.insert(idx,val)` | insert item at index |
| `lst.remove(val)` | remove first item with value *val* |
| `lst.pop([idx])` →*value* | remove & return item at index *idx* (default last) |
| `lst.sort()` `lst.reverse()` | sort / reverse liste *in place* |

## Operations on Dictionaries

`d[key]=value` `d.clear()`
`d[key]` → *value* `del d[key]`
`d.update(d2)` update/add associations
`d.keys()` `d.values()` `d.items()` →*iterable views on keys/values/associations*
`d.pop(key[,default])` → *value*
`d.popitem()` → *(key,value)*
`d.get(key[,default])` → *value*
`d.setdefault(key[,default])` →*value*

## Operations on Sets

Operators:
`|` → union (vertical bar char)
`&` → intersection
`-` `^` → difference/symmetric diff.
`<` `<=` `>` `>=` → inclusion relations
*Operators also exist as methods.*

`s.update(s2)` `s.copy()`
`s.add(key)` `s.remove(key)`
`s.discard(key)` `s.clear()`
`s.pop()`

## Files

*storing data on disk, and reading it back*

```
f = open("file.txt","w",encoding="utf8")
```

- file **variable** for operations
- **name** of file on disk (+path…) — cf. modules `os`, `os.path` and `pathlib`
- opening **mode**: `'r'` read, `'w'` write, `'a'` append, …`'+'` `'x'` `'b'` `'t'`
- **encoding** of chars for *text files*: `utf8` `ascii` `latin1` …

**writing**

`f.write("coucou")`
`f.writelines(list of lines)`

**reading**

*read empty string if end of file*
`f.read([n])` → *next chars* — *if n not specified, read up to end !*
`f.readlines([n])` → `list` *of next lines*
`f.readline()` → *next line*

*text mode `t` by default (read/write `str`), possible binary mode `b` (read/write `bytes`). **Convert from/to required type** !*

`f.close()` dont forget to **close the file** after use !

`f.flush()` write cache
`f.truncate([size])` resize

*reading/writing progress sequentially in the file, modifiable with:*
`f.tell()` →*position*
`f.seek(position[,origin])`

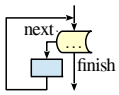
Very common: opening with a guarded block (automatic closing) and reading loop on lines of a text file:

```
with open(…) as f:
    for line in f :
        # processing of line
```

## Iterative Loop Statement

*statements block executed **for each** item of a container or iterator*

```
for var in sequence:
    statements block
```




Go over sequence's **values**

```
s = "Some text"   # initializations before the loop
cnt = 0
for c in s:       # loop variable, assignment managed by for statement
    if c == "e":
        cnt = cnt + 1
print("found",cnt,"'e'")
```

*Algo: count number of `e` in the string.*

loop on dict/set ⇔ loop on keys sequences
use *slices* to loop on a subset of a sequence

Go over sequence's **index**

- modify item at index
- access items around index (before / after)

```
lst = [11,18,9,12,23,4,17]
lost = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        lost.append(val)
        lst[idx] = 15
print("modif:",lst,"-lost:",lost)
```

*Algo: limit values greater than 15, memorizing of lost values.*

Go simultaneously over sequence's **index** and **values**:
`for idx,val in enumerate(lst):`

*good habit : don't modify loop variable*

## Integer Sequences

`range([start,] end [,step])`

*start* default 0, *end* not included in sequence, *step* signed, default 1

`range(5)` → 0 1 2 3 4 `range(2,12,3)` → 2 5 8 11
`range(3,8)` → 3 4 5 6 7 `range(20,5,-5)` → 20 15 10
`range(len(seq))` → *sequence of index of values in seq*

*range provides an immutable sequence of int constructed as needed*

## Function Definition

function name (identifier)
named parameters

```
def fct(x,y,z):
    """documentation"""
    # statements block, res computation, etc.
    return res
```

`fct`

result value of the call, if no computed result to return: `return None`

parameters and all variables of this block exist only *in* the block and *during* the function call (think of a "black box")

Advanced: `def fct(x,y,z,*args,a=3,b=5,**kwargs):`
**args variable positional arguments (→`tuple`), default values,*
***kwargs variable named arguments (→`dict`)*

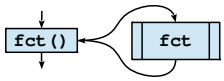
## Function Call

```
r = fct(3,i+2,2*i)
```

*storage/use of returned value* — *one argument per parameter*

this is the use of function name *with parentheses* which does the call

*Advanced: *sequence **dict*




## Operations on Strings

`s.startswith(prefix[,start[,end]])` `s.endswith(suffix[,start[,end]])` `s.strip([chars])`
`s.count(sub[,start[,end]])` `s.partition(sep)` → *(before,sep,after)*
`s.index(sub[,start[,end]])` `s.find(sub[,start[,end]])`
`s.is…()` *tests on chars categories (ex. `s.isalpha()`)*
`s.upper()` `s.lower()` `s.title()` `s.swapcase()`
`s.casefold()` `s.capitalize()` `s.center([width,fill])`
`s.ljust([width,fill])` `s.rjust([width,fill])` `s.zfill([width])`
`s.encode(encoding)` `s.split([sep])` `s.join(seq)`

## Formatting

formating directives — values to format

```
"modele{} {} {}".format(x,y,r) → str
```

`"{`*selection* `:` *formatting* `!` *conversion*`}"`

- **Selection** :
  `2`
  `nom`
  `0.nom`
  `4[key]`
  `0[2]`

Examples:

```
"{:+2.3f}".format(45.72793)
→'+45.728'
"{1:>10s}".format(8,"toto")
→'      toto'
"{x!r}".format(x="I'm")
→'"I\'m"'
```

- **Formatting** :

*fill char* *alignment* *sign* *mini width*.*precision~maxwidth* *type*

`< > ^ =` `+ - space` `0` at start for filling with 0

integer: `b` binary, `c` char, `d` decimal (default), `o` octal, `x` or `X` hexa…
float: `e` or `E` exponential, `f` or `F` fixed point, `g` or `G` appropriate (default),
string: `s` … `%` percent

- **Conversion** : `s` (readable text) or `r` (literal representation)