Program to implement various Data Structures in Python and their operations

Data Structure is used for organizing, managing and storing data. It is important as it enables easier access and efficient modifications. It allows us to organize data in such a way that enables us to store collections of data, relate them and perform operations on them accordingly.

Types of Data Structures in Python:

The Data Structure in Python is divided into two categories:

1. Built-In Data Structure

2. User-Defined Data Structures

1. Built-In Data Structure:

Built-In Data Structures are further divided into two sections:

A) Primitive Data Structure –

These are the most primitive or the basic data structures. They are the building blocks for data manipulation and contain pure, simple values of a data. Python has four primitive variable types:

- **Integers** - We can use an integer represent numeric data, and more specifically, whole numbers from negative infinity to infinity, like 4, 5, or -1.

```
In [10]: x = 20; y= 50

         print(y-x)

         print(type(x))
```

```
30
<class 'int'>
```

- **Float -** "Float" stands for 'floating point number'. We can use it for rational numbers, usually ending with a decimal figure, such as 1.11 or 3.14.

```
In [9]: x = 20.5 ; y = 33.33

        print(x+y)

        print(type(x))
```

```
53.83
<class 'float'>
```

- **Strings** - In Python, we can create strings by enclosing a sequence of characters within a pair of single or double quotes. We can also do slicing, indexing etc.

```
In [11]: x = 'Cake'
         y = 'Cookie'
         print(x + ' & ' + y)
```

```
Cake & Cookie
```

- **Boolean -** This built-in data type that can take up the values: 'True' and 'False', which often makes them interchangeable with the integers 1 and 0. Booleans are useful in conditional and comparison expressions.

```
In [14]: x = 4 ; y = 2

         x == y
```

```
Out[14]: False
```

Program to implement various Data Structures in Python and their operations

B) Non-Primitive Data Structures-

- **Lists -** Lists are used to store data of different data types in a sequential manner. There are addresses assigned to every element of the list, which is called as Index.

  1. Creating a list: To create a list, we use the square brackets and add elements into it accordingly.

```
In [17]: x = ["apple", "banana", "cherry"]

         print(x)

         x2 = [1,'apple',3]
         print(x2)
         print(x2[1]) #indexing

         print(type(x))
```

```
['apple', 'banana', 'cherry']
[1, 'apple', 3]
apple
<class 'list'>
```

  2. Adding Elements: List is mutable so we can easily add or remove elements.

```
In [19]: x = ["apple", "banana", "cherry"]
         x.append("grapes")
         print(x)
```

```
['apple', 'banana', 'cherry', 'grapes']
```

  3. Deleting Elements-

```
In [20]: x = ["apple", "banana", "cherry", "grapes"]
         x.remove("cherry")
         print(x)
```

```
['apple', 'banana', 'grapes']
```

- **Tuple-** Tuples are the same as lists are with the exception that the data once entered into the tuple cannot be changed no matter what. The only exception is when the data inside the tuple is mutable, only then the tuple data can be changed.

  1. Creating a Tuple: To create a tuple, you use parenthesis and add elements into it accordingly.

```
In [26]: x = ("apple", "banana", "cherry") ; print(x)
         x1 = (1,2,3,4,5) ; print(x1)
         y = ('c','a','k','e') ; print(y)
         print(x1_tuple[0]) #indexing
         print(type(x))
```

```
('apple', 'banana', 'cherry')
(1, 2, 3, 4, 5)
('c', 'a', 'k', 'e')
1
<class 'tuple'>
```

  2. Adding Elements: Tuple is immutable so we cannot add or remove elements.

```
In [28]: x1 = (1,2,3,4,5) ; print(x1)
         x[0] = 0 # We cannot change values inside a tuple
```

```
(1, 2, 3, 4, 5)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-c3c1d3b108bc> in <module>
      1 x1 = (1,2,3,4,5) ; print(x1)
----> 2 x[0] = 0 # We cannot change values inside a tuple

TypeError: 'tuple' object does not support item assignment
```

Program to implement various Data Structures in Python and their operations.

- **Dictionary** - Dictionaries is something similar to a telephone book. They are made up of key-value pairs. Key is used to identify the item and the value holds as the name suggests, the value of the item.

1. Creating a Dictionary:

```
In [29]: x_dict = {'Edward':1, 'Jorge':2, 'Prem':3, 'Joe':4}
         del x_dict['Joe']
         x_dict

Out[29]: {'Edward': 1, 'Jorge': 2, 'Prem': 3}
```

2. Changing and adding key, value pairs:

```
In [31]: x_dict = {'Edward':1, 'Jorge':2, 'Prem':3, 'Joe':4}
         x_dict['Noah'] = 5
         x_dict

Out[31]: {'Edward': 1, 'Jorge': 2, 'Prem': 3, 'Joe': 4, 'Noah': 5}
```

3. Accessing Elements:

```
In [34]: x_dict = {'Edward':1, 'Jorge':2, 'Prem':3, 'Joe':4}
         x_dict['Noah'] = 5
         x_dict
         x_dict.keys()

Out[34]: dict_keys(['Edward', 'Jorge', 'Prem', 'Joe', 'Noah'])

In [35]: x_dict.values()

Out[35]: dict_values([1, 2, 3, 4, 5])
```

4. Deleting key, value pairs-

```
In [38]: x_dict = {'Edward':1, 'Jorge':2, 'Prem':3, 'Joe':4}
         x_dict['Noah'] = 5
         a = x_dict.pop("Edward")
         print(x_dict)

         {'Jorge': 2, 'Prem': 3, 'Joe': 4, 'Noah': 5}
```

- **Sets -** Sets are a collection of unordered elements that are unique. It means that even if the data is repeated more than one time, it would be entered into the set only once.

1. Creating a Set:

```
In [39]: x_set = set('CAKE&COKE')
         y_set = set('COOKIE')

         print(x_set)

         {'O', 'K', 'C', 'E', 'A', '&'}

In [ ]:
```

Program to implement various Data Structures in Python and their operations.

2. Adding elements:

```
In [40]: x_set = set('CAKE&COKE')
         y_set = set('COOKIE')
         y_set.add("is tasty")

         print(y_set)
```

```
{'is tasty', 'O', 'I', 'K', 'C', 'E'}
```

3. Operations in sets- Union (), Intersection (), Difference ():

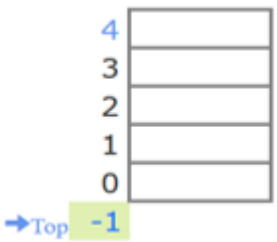
```
In [44]: set1 = {1,2,3,4,5}
         set2 = {2,4,6,8}
         print(set1.union(set2))
         print(set1.intersection(set2))
         print(set1.difference(set2))
```

```
{1, 2, 3, 4, 5, 6, 8}
{2, 4}
{1, 3, 5}
```

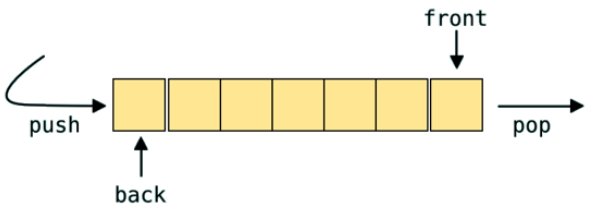
2. User-Defined Data Structures:

- **Arrays vs. Lists -** Arrays and lists are the same structure with one difference. Lists allow heterogeneous data element storage whereas Arrays allow only homogenous elements to be stored within them.

- **Stack -** Stacks are linear Data Structures which are based on the principle of Last-In-First-Out (LIFO) where data which is entered last will be the first to get accessed. It is built using the array structure and has operations namely, pushing (adding) elements, popping (deleting) elements and accessing elements.

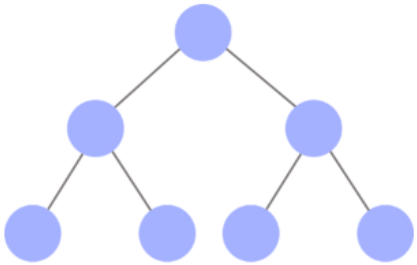


- **Queue -** A queue is also a linear data structure which is based on the principle of First-In-First-Out (FIFO) where the data entered first will be accessed first. It is built using the array structure and has operations which can be performed from both ends of the Queue, that is, head-tail or front-back. Operations such as adding and deleting elements are called En-Queue and De-Queue and accessing the elements can be performed.

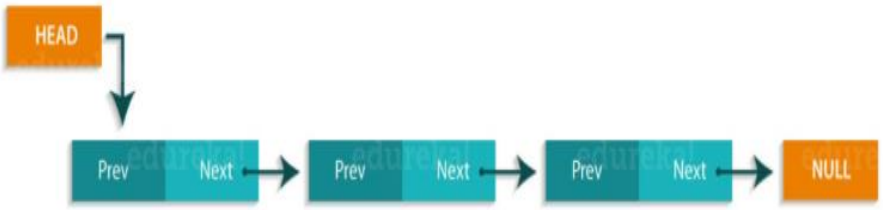


Program to implement various Data Structures in Python and their operations.

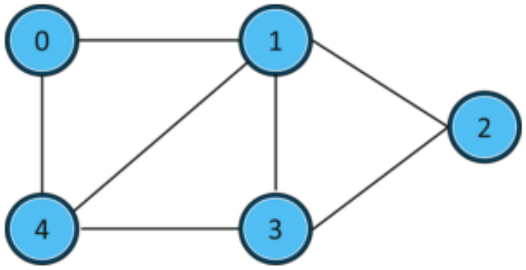
- **Tree -** Trees are non-linear Data Structures which have root and nodes. The root is the node from where the data originates and the nodes are the other data points that are available to us. The node that precedes is the parent and the node after is called the child. There are levels a tree has to show the depth of information. The last nodes are called the leaves.



- **Linked List-** Linked lists are linear Data Structures which are not stored consequently but are linked with each other using pointers. The node of a linked list is composed of data and a pointer called next. These structures are most widely used in image viewing applications, music player applications and so forth.



- **Graph-** Graphs are used to store data collection of points called vertices (nodes) and edges (edges). Graphs can be called as the most accurate representation of a real-world map. They are used to find the various cost-to-distances between the various data points called as the nodes and hence find the least path.



- **Hash Maps-** Hash Maps are the same as what dictionaries are in Python. They can be used to implement applications such as phonebooks, populate data according to the lists and much more.