URBAN EDGE ___/___/___

Data Structures & Algorithms by CodeWithHarry

This course will get you prepared for placements and will teach you how to create efficient and fast algorithms.

Data structures and algorithms are two different things.

Data Structures: Arrangement of data so that they can be used efficiently in memory (data items)

Algorithms: Sequence of steps on data using efficient data structures to solve a given problem.

Other Terminology

Database :- Collection of information in permanent storage for faster retrieval and updation.

Data warehousing :- Management of huge amount of legacy data for better analysis.

Big data :- Analysis of too large or complex data which cannot be dealt with traditional data processing application.

Data Structures and Algorithms are nothing new. If you have done programming in any language like C you must have used Arrays → A data structure and some sequence of processing steps to solve a problem → Algorithm

URBAN EDGE

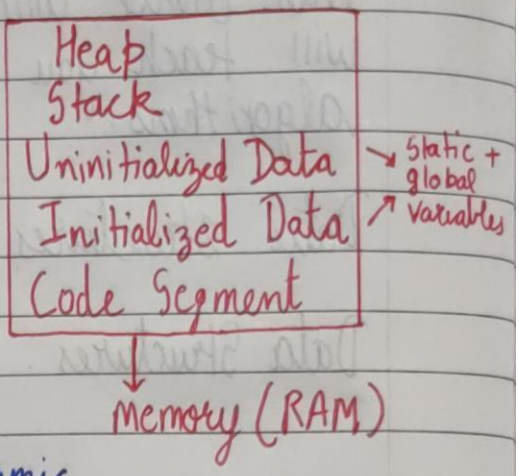
Memory layout of C programs

When the program starts, its Code is copied to the main memory.

Stack holds the memory occupied by the functions.

Heap contains the data which is requested by the program as dynamic memory.

Initialized and uninitialized data segments hold initialized and uninitialized global variables respectively



URBAN EDGE

Time Complexity & Big O notation

This morning I wanted to eat some pizzas; So I asked my brother to get me some from Dominos (3 km far)
He got me the pizza and I was happy only to realize it was too less for 29 friends who came to my house for a surprize visit!

My brother can get 2 pizzas for me on his bike but pizza for 29 friends is too huge of an input for him which he cannot handle.

2 pizzas ⟶  okay! not a big deal!

68 pizzas ⟶  Not possible! in short time

What is Time Complexity?
Time Complexity is the study of efficiency of algorithms.

(i) Time Complexity = How time taken to execute an algorithm grows with the size of the input!

Consider two developers who created an algorithm to sort n numbers. Shubham and Rohan did this independently.

When ran for input size n, following results were recorded:

no of elements (n)	Shubham's Algo	Rohan's Algo
10 elements	90 ms	122 ms
70 elements	110 ms	124 ms
110 elements	180 ms	131 ms
1000 elements	2 s	800 ms

We can see that initially Shubham's algorithm was shining for smaller input but as the number of elements increases Rohan's algorithm looks good!

Quick Quiz: Who's Algorithm is better?

Time Complexity: Sending GTA V to a friend
Let us say you have a friend living 5 kms away from your place. You want to send him a game.
Final exams are over and you want him to get this 60 GB file from you. How will you send it to him?
Note that both of you are using (JIO) 4G with 1 GB/day data limit.

The best way to send him the game is by delivering it to his house.
Copy the game to a Hard disk and send it!

Will yo do the same thing for sending a game like minesweeper which is in kbs of size?
No because you cand send it via internet.

As the file size grows, time taken by online sending increases linearly → $O(n^1)$
As the file size grows, time taken by physical sending remains constant. $O(n^0)$ or $O(1)$

Calculating Order in terms of Input size
In order to calculate the order, most impactful term containing n is taken into account.
↳ Size of input

Let us assume that formula of an algorithm in terms of input size n looks like this:

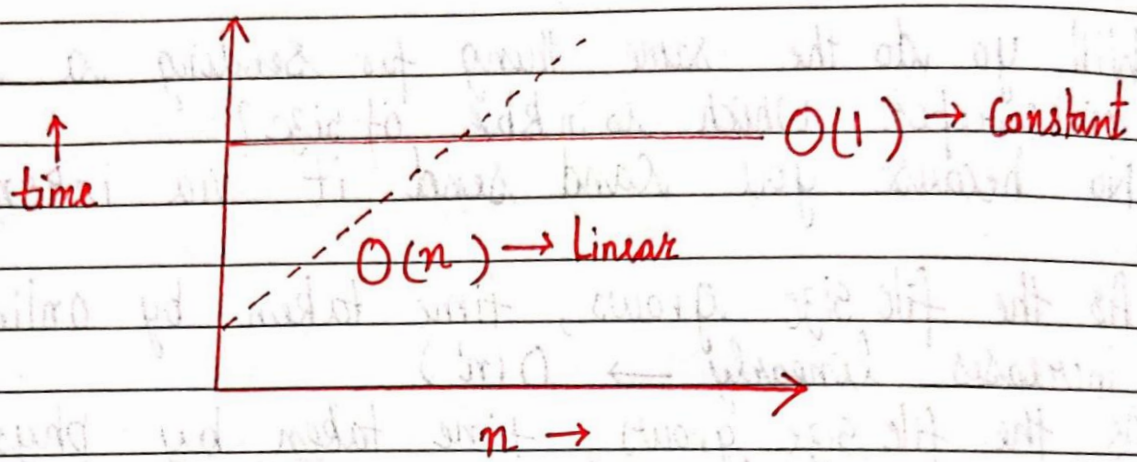
Algo 1 → $k_1 n^2 + k_2 n + 36$ ⇒ $O(n^2)$
($k_1 n^2$: Highest order term; $k_2 n + 36$: can ignore lower order terms)

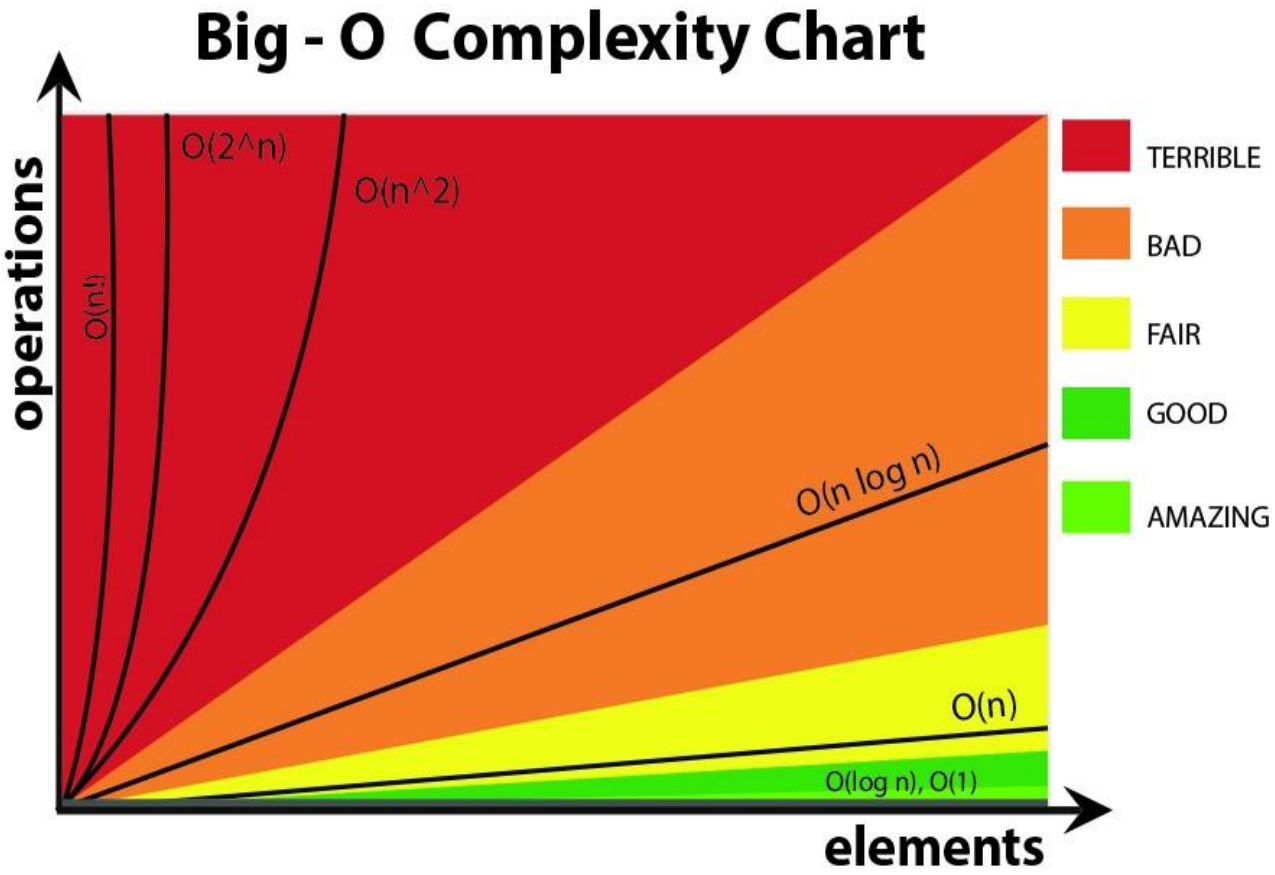
Algo 2 → $k_1 k_2^2 + k_3 k_2 + 8$
⇓
$k_1 k_2^2 n^0 + k_3 k_2 + 8$ ⇒ $O(n^0)$ or $O(1)$

Note that these are the formulas for time taken by them.

Visualising Big O

If we were to plot O(1) and O(n) on a graph, they will look something like this :





Source: https://stackoverflow.com/questions/3255/big-o-how-do-you-calculate-approximate-it

Asymptotic Notations

Asymptotic notations give us an idea about how good a given algorithm is compared to some other algorithm
Let us see the mathematical definition of "order of" now.

Primarily there are three types of widely used asymptotic notations.

1) Big oh notation (O)
2) Big omega notation (Ω)
3) Big theta notation (Θ) ⟶ Widely used one!

Big oh notation

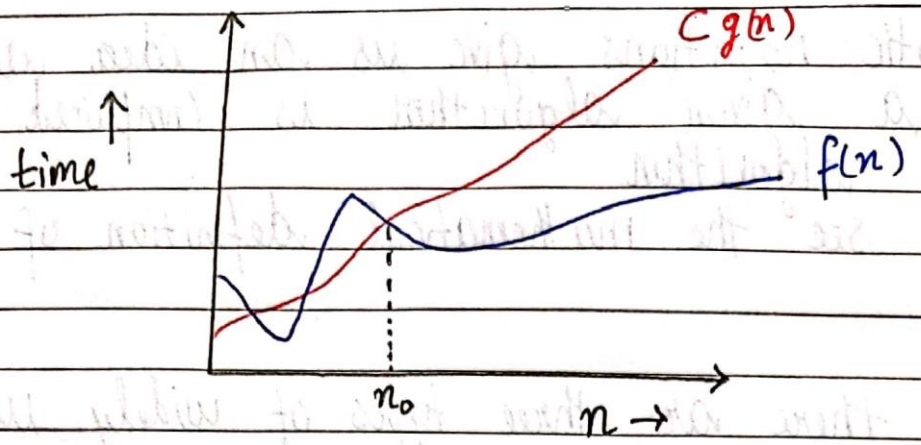
Big oh notation is used to describe asymptotic upper bound
Mathematically, if $f(n)$ describes running time of an algorithm; $f(n)$ is $O(g(n))$ iff there exist positive constants C and n_0 such that

$$0 \leq f(n) \leq C\,g(n) \quad \text{for all } n \geq n_0$$

⇓
used to give upper bound on a function.

if a function is $O(n)$, it is automatically $O(n^2)$ as well!

Graphic example for Big oh (O)



Big Omega notation

Just Like O notation provides an asymptotic upper bound, Ω notation provides asymptotic lower bound. Let $f(n)$ define running time of an algorithm;

$f(n)$ is said to be $\Omega(g n)$ if there exists positive constants C and n_0 such that

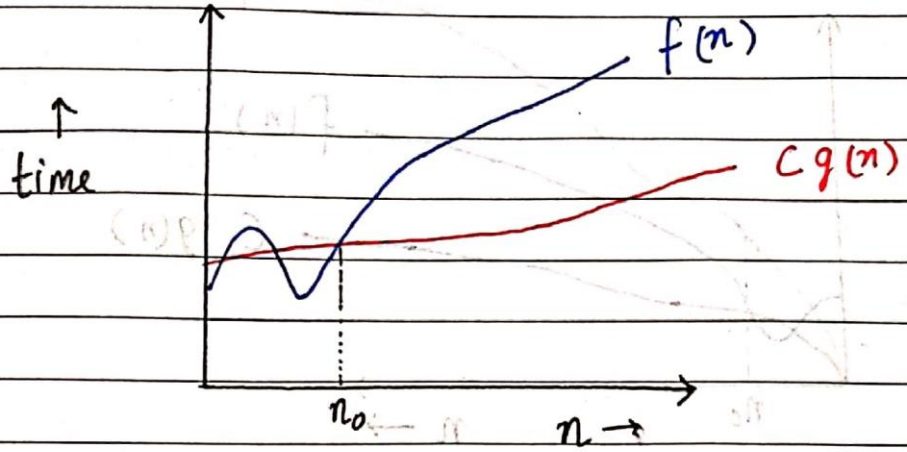
$$0 \leq c\,g(n) \leq f(n) \quad \text{for all } n \geq n_0$$

⇓ used to give lower bound on a function

if a function is $O(n^2)$ it is automatically $O(n)$ as well

URBAN EDGE

Graphic example for Big omega (Ω)



Big theta notation

Let $f(n)$ define running time of an algorithm

$f(n)$ is said to be $\Theta(gn)$ iff $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$

Mathematically,

$$0 \leq f(n) \leq C_1 g(n) \quad \forall\ n \geq n_0$$

$$0 \leq C_2 g(n) \leq f(n) \quad \forall\ n \geq n_0$$

Sufficiently large value of n

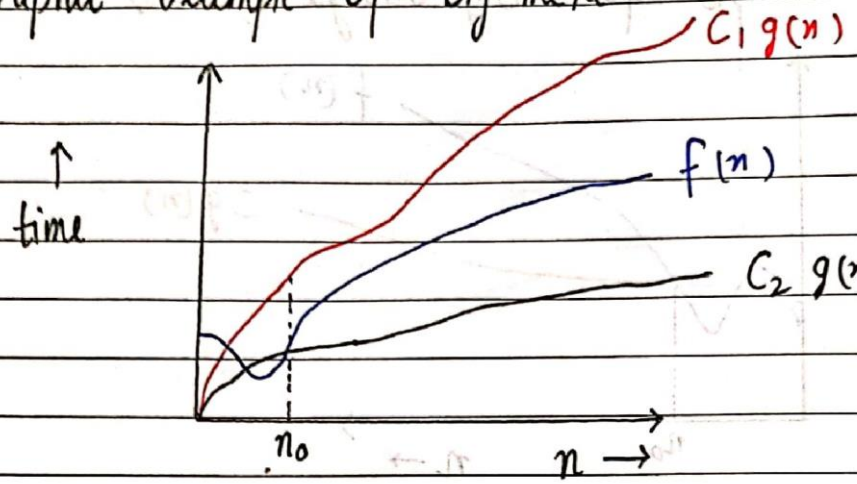
Merging both the equations, we get:

$$0 \leq C_2 g(n) \leq f(n) \leq C_1 g(n) \quad \forall\ n \geq n_0$$

The equation simply means there exist positive constants C_1 and C_2 such that $f(n)$ is sandwiched between $C_2 g(n)$ and $C_1 g(n)$

URBAN EDGE

Graphic example of Big theta



Which one of these to use ?
Since Big theta gives a better picture of runtime for a given algorithm, most of the interviewers expect you to provide an answer in terms of Big theta when they say "Order of".

Quick Quiz :- Prove that n^2+n+1 is $O(n^3)$, $\Omega(n^2)$ and $\Theta(n^2)$ using respective definitions.

Increasing order of common runtimes

$$1 < \log n < n < n\log n < n^2 < n^3 < 2^n < n^n$$

Better ⟵ ⇓ ⟶ Worse

Common runtimes from better to worse ⟶

URBAN EDGE

Best, worst and Expected Case

Sometimes we get lucky in life. Exams cancelled when you were not prepared, surprise test when you were prepared etc. ⟹ Best case

Some times we get unlucky. Questions you never prepared asked in exams, rain during sports period etc. ⇒ Worst case

But overall the life remains balance with the mixture of lucky and unlucky times. ⇒ Expected case.

Analysis of a search algorithm

Consider an array which is sorted in increasing order

1	7	18	28	50	180

We have to search a given number in this array and report whether its present in the array or not.

Algo 1 → Start from first element until an element greater than or equal to the number to be searched is found.

Algo 2 → Check whether the first or the last element is equal to the number. If not find the number between these two elements (center of the array). If the center element is greater than the number to be searched, repeat the process for first half else repeat for second half until the number is found.

Analyzing Algo 1

If we really get lucky, the first element of the array might turn out to be the element we are searching for. Hence we made just one comparison.

$$\text{Best case Complexity} = O(1)$$

If we are really unlucky, the element we are searching for might be the last one.

$$\text{Worst case Complexity} = O(n)$$

For calculating Average case time, we sum the list of all the possible case's runtime and divide it with the total number of cases.

⇓

Sometimes calculation of average case time gets very complicated

Analyzing Algo 2

If we get really lucky, the first element will be the only one which gets compared

$$\text{Best case Complexity} = O(1)$$

If we get unlucky we will have to keep dividing the array into halves until we get a single element (the array gets finished)

Worst case Complexity = $O(\log n)$

What log(n)? What is that

log(n) → Number of times you need to half the array of size n before it gets exhausted

$\log 8 = 3 \implies \frac{8}{2} \to \frac{4}{2} \to \frac{2}{2}$ → Cant break anymore.

$1 + 1 + 1$

$\log 4 = 2 \implies \frac{4}{2} \to \frac{2}{2}$ → Cant break anymore.

$1 + 1$

Log n simply means how many time I need to divide n units such that we cannot divide them (into halves) anymore.

Space Complexity

Time is not the only thing we worry about while analyzing algorithms. Space is equally important.

Creating an array of size n → O(n) space
↳ Size of input

If a function calls itself recursively n times its space complexity is O(n)

URBAN EDGE

Quick Quiz → Calculate Space Complexity of a function which calculates factorial of a given number n.

Why cant we calculate Complexity in seconds?

→ Not everyone's Computer is equally powerful

→ Asymptotic analysis is the measure of how time (runtime) grows with input

Techniques to Calculate Time Complexity

Once we are able to write the runtime in terms of size of the input (n), we can find the time complexity.

For example $T(n) = n^2 \Rightarrow O(n^2)$

$T(n) = \log n \Rightarrow O(\log n)$

Some tricks to calculate complexity

1. Drop the constants : Any thing you might think is $O(3n)$ is $O(n)$
   ↳ Better representation
2. Drop the non dominant terms : Any thing you represent as $O(n^2+n)$ can be written as $O(n^2)$
3. Consider all variables which are provided as input : $O(mn)$ & $O(mnq)$ might exist for some cases!

In most of the cases, we try to represent the runtime in terms of the input which can be more than one in number. For example :

Painting a park of dimension $m \times n \Rightarrow O(mn)$

Time Complexity – Competitive Practice Sheet

1. Fine the time complexity of the func1 function in the program show in program1.c as follows:

```c
#include <stdio.h>

void func1(int array[], int length)
{
    int sum = 0;
    int product = 1;
    for (int i = 0; i < length; i++)
    {
        sum += array[i];
    }

    for (int i = 0; i < length; i++)
    {
        product *= array[i];
    }
}

int main()
{
    int arr[] = {3, 5, 66};
    func1(arr, 3);
    return 0;
}
```

2. Fine the time complexity of the func function in the program from program2.c as follows:

```c
void func(int n)
{
    int sum = 0;
    int product = 1;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("%d , %d\n", i, j);
        }
    }

}
```

3. Consider the recursive algorithm above, where the random(int n) spends one unit of time to return a random integer which is evenly distributed within the range [0,n][0,n]. If the average processing time is T(n), what is the value of T(6)?

```
int function(int n)
{
    int i;

    if (n <= 0)
    {
        return 0;
    }
    else
    {
        i = random(n - 1);
        printf("this\n");
        return function(i) + function(n - 1 - i);
    }
}
```

4. Which of the following are equivalent to O(N)? Why?
    a) O(N + P), where P < N/9
    b) 0(9N-k)
    c) O(N + 8log N)
    d) O(N + M^2)

5. The following simple code sums the values of all the nodes in a balanced binary search tree. What is its runtime?

```
int sum(Node node)
{
    if (node == NULL)
    {
        return 0;
    }
    return sum(node.left) + node.value + sum(node.right);
}
```

6. Find the complexity of the following code which tests whether a give number is prime or not?

```
int isPrime(int n){
    if (n == 1){
        return 0;
    }

    for (int i = 2; i * i < n; i++) {
        if (n % i == 0)
            return 0;
    }
```

```
    return 1;
}
```

7. What is the time complexity of the following snippet of code?

```
int isPrime(int n){

    for (int i = 2; i * i < 10000; i++) {
        if (n % i == 0)
            return 0;
    }

    return 1;
}
isPrime();
```

URBAN EDGE

Operations on an Array

Following operations are supported by an array.

Traversal
Insertion
Deletion
Search

⇒ There can be many other operations one can perform on arrays as well!
eg: Sorting asc., Sorting desc.

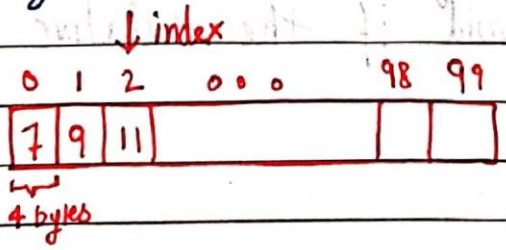
Traversal

Visiting every element of an array once → Traversal

Why traversal? → For use cases like:
→ Storing all elements → using scanf
→ Printing all elements → using printf

An important note about arrays
If we create an array of length 100 using a[100] in C language, we need not use all the elements. It is possible for a program to use just 60 elements out of these 100.
↳ But we cannot go beyond 100 elements.

An array can easily be traversed using a for loop in C language

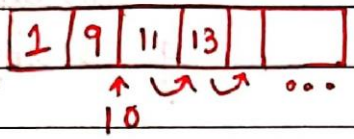


URBAN EDGE

Insertion

An element can be inserted in an array at a specified position.

In order for this operation to be successful, the array should have enough capacity

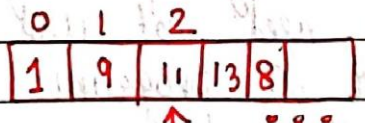


⇒ Elements need to be shifted to maintain relative order.

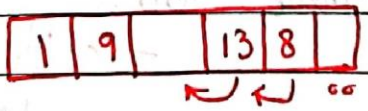
When no position is specified its best to insert the element at the end.

Deletion

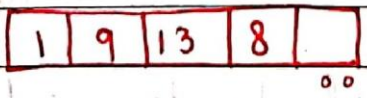
An element at specified position can be deleted creating a void which needs to be fixed by shifting all the elements to the left as follows:



Delete 11 at ind 2



Shift the elements

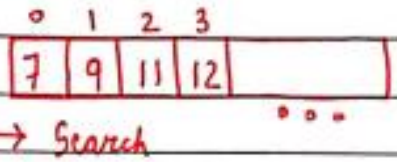


Deletion done!

We can also bring the last element of the array to fill the void if the relative ordering is not important

Searching

Searching can be done by traversing the array until the element to be searched is found



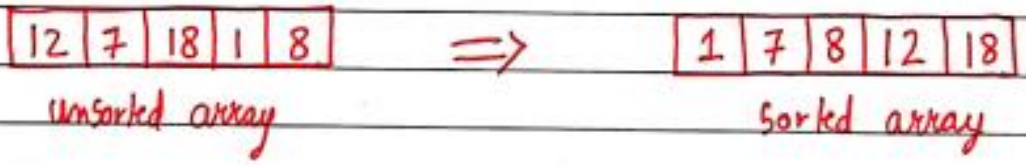
⇓

For sorted array time taken to search is much less than unsorted array!!

Sorting

Sorting means arranging an array in order (asc or desc)

We will see various sorting techniques later in the course.



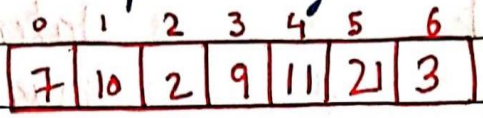
Unsorted array

Sorted array

Linear Vs Binary Search

Linear Search

Searches for an element by visiting all the elements sequentially until the element is found.



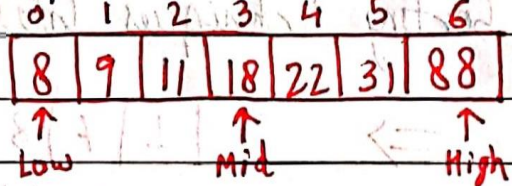
⇒ can be sorted or unsorted

Search '2' → Element found

WC Complexity : O(n)

Binary Search

Searches for an element by breaking the search space into half in a sorted array.



WC Complexity O(log n)

Search 18

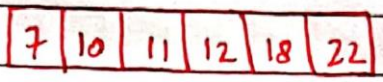
The search continues towards either side of mid based on whether the element to be searched is lesser or greater than mid.

	Linear Search	Binary Search
1.	Works on both sorted and unsorted arrays	Works only on sorted arrays
2.	Equality operations	inequality operations
3.	O(n) WC Complexity	O(log n) WC Complexity

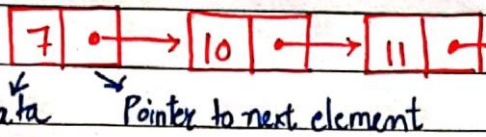
URBAN EDGE

Introduction to Linked Lists

Linked lists are similar to arrays (Linear data structures)



⇒ In Arrays elements are stored in contiguous memory locations



⇒ In linked lists, elements are stored in non contiguous memory locations

Why Linked Lists?

Memory and the capacity of an array remains fixed.
In case of linked lists, we can keep adding and removing elements without any capacity constraints

Drawbacks of Linked lists

→ Extra memory space for pointers is required (for every node 1 pointer is needed)
→ Random access not allowed as elements are not stored in contiguous memory locations.

Implementation

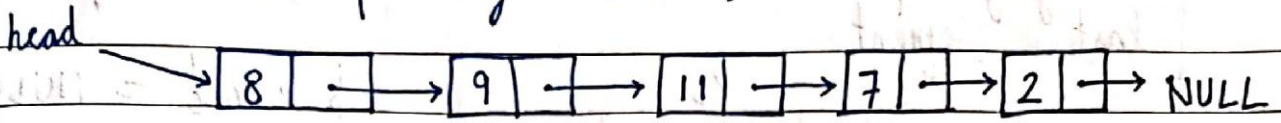
Linked list can be implemented using a structure in C language

```
Struct Node {
    int data;
    Struct Node * next;
};
```

⇒ Self refencing structure

Deletion in a Linked List

Consider the following Linked List

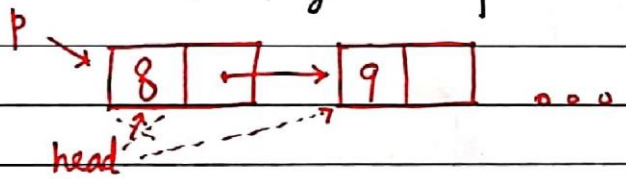


Deletion can be done for the following cases:

1. Deleting the first Node
2. Deleting the node at an index
3. Deleting the last Node
4. Deleting the first node with a given value.

The deletion just like insertion is done by rewiring the pointer connections, the only caveat being: we need to free the memory of the deleted node using free().

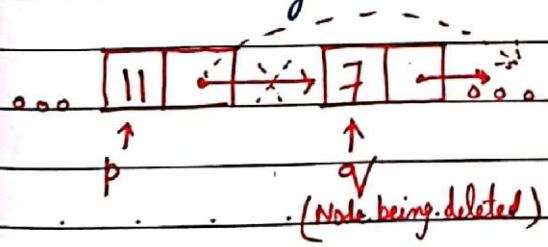
Case 1: Deleting the first node




```
Struct Node * p = head
head = head → next;
free(p);
```

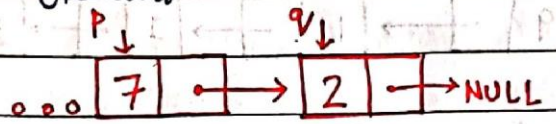
Case 2: Deleting the node at an index

for deleting a given node, we first bring a temporary pointer p before element to be deleted and q on the element being deleted



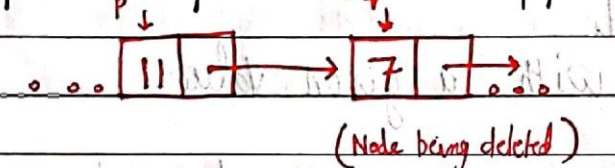

```
p → next = q → next;
free(q);
```

Case 3: Deleting the last Node
Last node can be deleted just like Case 2 by bringing p on second last element and q on last element.



p→next = NULL
free(q) → To free the memory!

Case 4: Delete the first node with a given value
This can be done exactly like Case 2 by bringing pointers p & q to appropriate positions

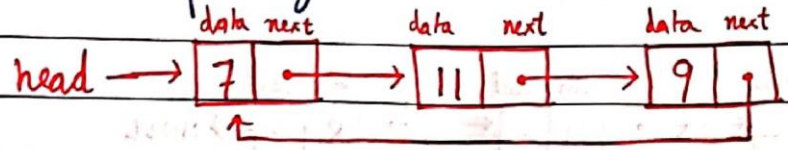


p→next = q→next;
free(q);

URBAN EDGE

Circular Linked List

A circular linked list is a linked list where the last element points to the first element (head) hence forming a circular chain



Operations on a circular linked List
operations on a circular linked lists can be performed exactly like a singly linked list.

Visit www.codewithharry.com for practice sets / code / more

URBAN EDGE

Doubly Linked List

In a doubly linked list, each node contains a data part along with the two addresses, one for the previous node and the other one for the next node

Implementation

A doubly linked list can be implemented in C language as follows:

```
struct Node {
    int data;
    struct Node * next;
    struct Node * prev;
};
```

Operations on a Doubly Linked List

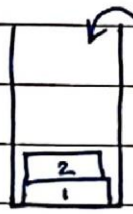
The insertion and deletion on a Doubly linked list can be performed by rewiring pointer connections just like we saw in a Singly linked list.

The difference here lies in the fact that we need to adjust two pointers (prev & next) instead of one (next) in the case of a Doubly linked list.

URBAN EDGE

Introduction to Stack Data Structure

Stack is a linear data structure. Operations on stack are performed in LIFO (Last in first out) order.



⇒ Item 2 which entered the basket last will be the first one to come out

LIFO (Last in first out)

Applications of Stack

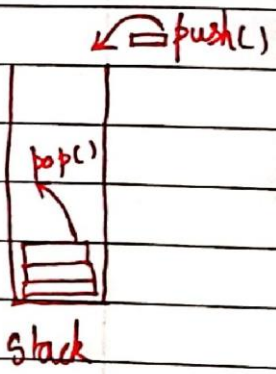
1. Used in function calls
2. Infix to postfix conversion (and other similar conversions)
3. Parenthesis matching & more...

Stack ADT

In order to create a stack we need a pointer to the topmost element along with other elements which are stored inside the stack.

Some of the operations of stack ADT are:

1. push() → push an element into the stack
2. pop() → remove the topmost element from the stack
3. peek(index) → Value at a given position is returned
4. isEmpty() / isFull() → Determine whether the stack is empty or full.



Implementation

A stack is a collection of elements with certain operations following LIFO (Last in First out) dicipline.

A Stack can be implemented using an array or a linked list

URBAN EDGE

Python programming notes by Harry

What is Programming?

Just like we use Hindi or English to communicate with each other, we use a programming language like Python to communicate with the computer.
Programming is a way to instruct the computer to perform various tasks.

What is Python?

Python is a simple and easy to understand language which feels like reading simple english. This Pseudo Code nature of python makes it easy to learn and understandable by beginners

Features of Python

Easy to understand = Less development time
Free and open source
High level language
Portable → Works on Linux / Windows / Mac.
+ Fun to work with

Installation

Python can be easily installed from python.org
When you click on the download button, python can be installed right after you complete the setup by executing the file for your platform

URBAN EDGE

Chapter 1 : Modules, Comments & pip

Lets write our very first python program. Create a file called hello.py and paste the below code in it

```
print ("Hello World")
```
→ print is a function (More later)

Execute this file (.py file) by typing python hello.py and you will see Hello World printed on the screen.

Modules

A module is a file containing code written by somebody else (usually) which can be imported and used in our programs

Pip

Pip is the package manager for python
You can use pip to install a module on your system

↳ pip install flask installs flask module.

Types of modules

There are two types of modules in Python

1) Built in modules → Pre installed in python
2) External modules → Need to install using pip.

Some examples of built in modules are os, abc, etc.
Some examples of external modules are tensorflow, flask etc.

Using Python as a Calculator

We can use python as a calculator by typing "python" + ↵ on the terminal

↳ This opens REPL or Read Evaluate Print Loop

Comments

Comments are used to write something which the programmer does not want to execute.

↳ Can be used to mark author name, date etc.

Types of Comments

There are two types of comments in Python

1. Single line Comments → Written using #
2. Multi line Comments → Written using ''' Comment '''

URBAN EDGE

Chapter 1 - Practice Set

1. Write a program to print Twinkle twinkle little star poem in python.

2. Use REPL and print the table of 5 using it

3. Install an external module and use it to perform an operation of your interest

4. Write a python program to print the contents of a directory using os module. Search online for the function which does that.

5. Label the program written in Problem 4 with Comments.

URBAN EDGE

Chapter 2 - Variables and Datatypes

A variable is the name given to a memory location in a program. for example

```
a = 30
b = "Harry"
c = 71.22
```

→ Variables = Containers to store a value.
→ Keywords = Reserved words in Python
Identifiers = class/function/variable name

Data Types

Primarily there are following data types in Python

1. Integers
2. Floating point numbers
3. Strings
4. Booleans
5. None

Python is a fantastic language that automatically identifies the type of data for us.

```
a = 71          ⇒ Identifies a as class<int>
b = 88.44       ⇒ Identifies b as class <float>
name = "Harry"  ⇒ Identifies name as class <str>
```

Rules for defining a Variable name → Also applies to other Identifiers

→ A variable name can contain alphabets, digits and underscores.
→ A variable name can only start with an alphabet and underscore.
→ A variable name cant start with a digit
→ No white space is allowed to be used inside a variable name.

URBAN EDGE

Examples of a few variable names are :-
harry, one8, seven_, _seven etc.

Operators in Python
Following are some common operators in Python:
1. Arithmetic operators ⇒ +, -, *, / etc.
2. Assignment operators ⇒ =, +=, -= etc.
3. Comparison operators ⇒ ==, >, >=, <, != etc.
4. Logical operators ⇒ and, or, not

type() function and Typecasting
type function is used to find the data type of a given variable in Python.

```
a = 31
type(a) => class <int>

b = "31"
type(b) => class <str>
```

A number can be converted into a string and vice versa (if possible)
There are many functions to convert one data type into another

```
str(31) => "31"      => Integer to String Conversion
int("32") => 32      => String to Integer Conversion
float(32) => 32.0    => Integer to Float Conversion

... and so on
```

Here "31" is a string literal and 31 a numeric literal.

URBAN EDGE

input() function
This function allows the user to take input from the keyboard as a string

a = input("Enter name") ⇒ If a is "harry", the user entered harry

It is important to note that the output of input is always a string (even if the number is entered) ⇒ If a is "34" user entered 34

URBAN EDGE

Chapter 2 - Practice Set

1. Write a Python program to add two numbers

2. Write a Python program to find remainder when a number is divided by 2.

3. Check the type of the variable assigned using input() function.

4. Use Comparison operators to find out whether a given variable 'a' is greater than 'b' or not.
   Take a = 34 and b = 80

5. Write a python program to find average of two numbers entered by the user.

6. Write a python program to calculate square of a number entered by the user.

URBAN EDGE

Chapter 3 - Strings

String is a data type in Python.
String is a sequence of characters enclosed in quotes.
We can primarily, write a string in these three ways

1. Single quoted strings → a = 'Harry'
2. Double quoted strings → b = "Harry"
3. Triple quoted strings → c = '''Harry'''

String Slicing

A String in Python can be sliced for getting a part of the string.
Consider the following string:

name = "H a r r y" ⇒ length = 5

	H	a	r	r	y
	0	1	2	3	4
	(-5)	(-4)	(-3)	(-2)	(-1)

The index in a string starts from 0 to (length-1) in Python. In order to slice a string, we use the following syntax:

sl = name [ind_start : ind_end]
first index included — last index is not included

sl [0:3] returns "Har" → characters from 0 to 3
sl [1:3] returns "ar" → characters from 1 to 3

Negative Indices: Negative Indices can also be used as shown in the figure above. -1 corresponds to the (length-1) index, -2 to (length-2)

URBAN EDGE

Slicing with skip value
We can provide a skip value as a part of our slice like this :

```
word = "amazing"
word [1:6:2] → 'mzn'
```

Other advanced slicing techniques

```
word = "amazing"
word [:7] → word [0:7] → 'amazing'
word [0:] → word [0:7] → 'amazing'
```

String functions
Some of the mostly used functions to perform operations on or manipulate strings are :

1> len() function → This function returns the length of the string

len ("Harry") → returns 5

2> string.endswith ("xxy") → This function tells whether the variable string ends with the string "xxy" or not. if string is "Harry", it returns true for "xxy" since Harry ends with xxy

3> string.count ("c") → Counts the total number of occurence of any character

4> string.capitalize() → This function capitalizes the first character of a given string.

URBAN EDGE

5. String.find(word) - This function finds a word and returns the index of first occurence of that word in the string.

6. String.replace(oldword, newword) - This function replaces the oldword with newword in the entire string.

Escape Sequence Characters
Sequence of characters after backslash '\' → Escape Seq characters

Escape sequence character comprises of more than one characters but represents one character when used within the strings.

Examples \n, \t, \', \\ etc.
(\n → newline, \t → Tab, \' → Single quote, \\ → backslash.)

URBAN EDGE

Chapter 3 - Practice Set

1. Write a Python program to display a user entered name followed by Good Afternoon using input() function

2. Write a program to fill in a letter template given below with name and date.

```
letter = ''' Dear <|NAME|>,
                You are selected!
<|DATE|> '''
```

3. Write a program to detect double spaces in a string

4. Replace the double spaces from Problem 3 with single spaces.

5. Write a program to format the following letter using escape sequence characters.

letter = "Dear Harry, This Python course is nice. Thanks!"

Chapter 4 - Lists and Tuples

Python Lists are containers to store a set of values of any data type

```
friends = ["Apple", "Akash", "Rohan", 7, False]
```

"Apple" → str()
"Akash" → can store value of any datatype
7 → int()
False → bool()

List Indexing

A List can be indexed just like a String

```
L1 = [7, 9, "Harry"]
```

```
L1[0]  => 7
L1[1]  => 9
L1[70] => Error
L1[0:2] => [7, 9]   => List Slicing
```

List Methods

Consider the following list:

```
L1 = [1, 8, 7, 2, 21, 15]
```

1. L1.sort(): updates the list to [1, 2, 7, 8, 15, 21]
2. L1.reverse(): updates the list to [15, 21, 2, 7, 8, 1]
3. L1.append(8): adds 8 at the end of the list
4. L1.insert(3,8): This will add 8 at 3 index

URBAN EDGE

5> L1.pop(2): Will delete element at index 2 and return its value

6> L1.remove(21): Will remove 21 from the list.

Tuples in Python

A tuple is an immutable data type in python.
↳ Cannot change

a = ( ) => Empty tuple
a = (1,) => Tuple with only one element needs a comma
a = (1, 7, 2) => Tuple with more than one element

Once defined a tuple's elements cant be altered or manipulated.

Tuple methods

Consider the following tuple
a = (1, 7, 2)

1> a.count(1): a.count(1) will return number of times 1 occurs in a.

2> a.index(1): a.index(1) will return the index of first occurence of 1 in a.

URBAN EDGE

Chapter 4 - Practice Set

1. Write a program to store seven fruits in a list entered by the user.

2. Write a program to accept marks of 6 students and display them in a sorted manner

3. Check that a tuple cannot be changed in python

4. Write a program to sum a list with 4 numbers.

5. Write a program to count the number of zeros in the following tuple:

$$a = (7, 0, 8, 0, 0, 9)$$

Chapter 5 : Dictionary & Sets

Dictionary is a collection of key-value pairs

Syntax :

```
a = { "key" : "value",
      "harry" : "code",
      "marks" : "100",
      "list" : [1,2,9] }
```

(keys → "key", "harry", "marks", "list"; values → "value", "code", "100", [1,2,9])

a["key"] => Prints "value"
a["list"] => Prints [1,2,9]

Properties of a Python Dictionaries

1. It is unordered
2. It is mutable
3. It is indexed
4. Cannot contain duplicate keys

Dictionary Methodes

Consider the following dictionary

```
a = { "name" : "Harry",
      "from" : "India",
      "marks" : [92,98,96] }
```

1. a.items() : returns a list of (key, value) tuples
2. a.keys() : returns a list containing dictionary's keys
3. a.update({"friend" : "Sam"}) : updates the dictionary with supplied key-value pairs

URBAN EDGE

4. a.get("name"): returns the value of the specified keys (and value is returned eg. "Harry" is returned here)

More methods are available on docs.python.org.

Sets in Python

Set is a collection of non repetitive elements

```
S = Set()                    ⇒ No repetition allowed!
S.add(1)
S.add(2)                     ⇒ or Set = {1, 2}
```

If you are a programming beginner without much knowledge of mathematical operations on sets, you can simply look at sets in python as data types containing unique values.

Properties of Sets

1. Sets are unordered ⇒ Element's order doesn't matter
2. Sets are unindexed ⇒ Cannot access elements by index
3. There is no way to change items in sets.
4. Sets cannot contain duplicate values

Operations on Sets

Consider the following Set:

S = {1, 8, 2, 3}

1. len(s): Returns 4, the length of the set
2. S.remove(8): Updates the set S and removes 8 from S.

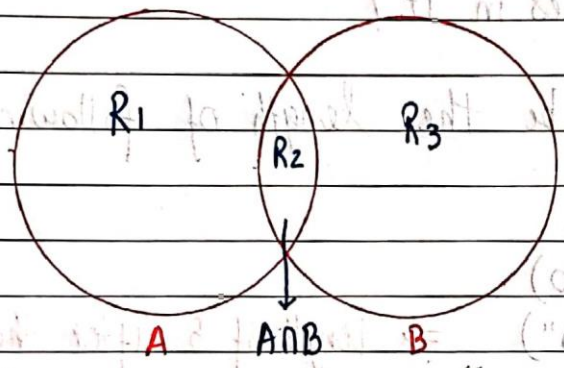
URBAN EDGE

3. S.pop() : Removes an arbitrary element from the set and returns the element removed

4. S.clear() : Empties the set S

5. S.union({8,11}) : Returns a new set with all items from both sets. => {1,8,2,3,11}

6. S.intersection({8,11}) : Returns a set which contains only items in both sets. => {8}



$R_2 \Rightarrow A \cap B$

$R_1 + R_2 + R_3 \Rightarrow A \cup B$

$R_1 + R_3 \Rightarrow A \Delta B$

$R_1 \Rightarrow A - B$

$R_3 \Rightarrow B - A$

URBAN EDGE

Chapter 5: Practice Set

1. Write a program to create a dictionary of Hindi words with values as their english translation. Provide user with an option to look it up!

2. Write a program to input eight numbers from the user and display all the unique numbers (once).

3. Can we have a set with 18 (int) and "18" (str) as a values in it?

4. What will be the length of following set S:
```
S = Set()
S.add(20)
S.add(20.0)
S.add("20")
```
=> Length of S after these operations?

5. S = {}
what is the type of S?

6. Create an empty dictionary. Allow 4 friends to enter their favourite language as values and use keys as their names. Assume that the names are unique

7. If names of 2 friends are same; what will to the program in Problem 6?

8. If languages of two friends are same; what will happen to the program in Problem 6?

9 Can you change the values inside a list which is contained in Set S

$S = \{8, 7, 12, \text{"Harry"}, [1, 2]\}$

Chapter 6 - Conditional Expressions

Sometimes we want to play pubG on our phone if the day is Sunday.
Sometimes we order Icecream online if the day is Sunny.
Sometimes we go hiking if our parents allow.

All these are decisions which depends on a condition being met.
In Python programming too, we must be able to execute instructions on a condition(s) being met. This is what conditionals are for!

If else and elif in Python

If else and elif statements are a multiway decision taken by our program due to certain conditions in our code.

Syntax:

```
if (condition1):                    => if condition 1 is true
    print ("yes")
elif (condition2):                  => if condition 2 is true
    print ("No")
else:                               => otherwise
    print ("May be")
```

Indentation ←

Code example:

```
a = 22
if(a>9):
    print("Greater")
else:
    print("lesser")
```

Quick Quiz: Write a program to print yes when the age entered by the user is greater than or equal to 18.

Relational Operators

Relational operators are used to evaluate conditions inside the if statements. Some examples of relational operators are:

- == → equals
- >= → greater than/equal to
- <=, etc.

Logical operators

In python logical operators operate on conditional statements. Example:

- and → true if both operands are true else false
- or → true if at least one operand is true else false
- not → inverts true to false & false to true

elif clause

elif in python means [else if]. An if statement can be chained together with a lot of these elif statements followed by an else statement

```
if (Condition1):
    # code
elif (condition 2):
    # code
elif (condition 3):
    # code
:
:
else:
    # code
```

⇒ This ladder will stop once a condition in an if or elif is met.

Important notes:

1. There can be any number of elif statements.
2. last else is executed only if all the conditions inside elifs fail.

URBAN EDGE

Chapter 6 - Practice Set

1. Write a program to find greatest of four numbers entered by the user.

2. Write a program to find out whether a student is pass or fail, if it requires total 40% and at least 33% in each subject to pass. Assume 3 subjects and take marks as an input from the user.

3. A spam comment is defined as a text containing following keywords:
"make a lot of money", "buy now", "subscribe this", "click this". Write a program to detect these spams.

4. Write a program to find whether a given username contains less than 10 characters or not.

5. Write a program which finds out whether a given name is present in a list or not.

6. Write a program to calculate the grade of a student from his marks from the following scheme:

90 - 100	→	Ex
80 - 90	→	A
70 - 80	→	B
60 - 70	→	C
50 - 60	-	D
<50	-	F

URBAN EDGE

7. Write a program to find out whether a given post is talking about "Harry" or not.

URBAN EDGE

Chapter 7 - Loops in Python

Sometimes we want to repeat a set of statements in our program. For instance: Print 1 to 1000

Loops make it easy for a programmer to tell the computer, which set of instructions to repeat and how!

Types of loops in Python

Primarily there are two types of loops in Python

1. While loop
2. For loop

We will look into these one by one!

While loop

The syntax of a while loop looks like this:

```
While Condition:
    #Body of the loop
```

⇒ The block keeps executing until the condition is true

In while loops, the condition is checked first. If it evaluates to true, the Body of the loop is executed, otherwise not!
If the loop is entered, the process of [Condition check & execution] is continued until the condition becomes false.

Quick Quiz: Write a program to print 1 to 50 using a while loop.

URBAN EDGE

An Example

```
i = 0
while i < 5:
    print("Harry")
    i = i + 1
```

⇒ Prints "Harry" - 5 times!

Note: If the condition never becomes False, the loop keeps getting executed.

Quick Quiz: Write a program to print the content of a list using while loops.

For loop

A for loop is used to iterate through a sequence like list, tuple or string [Iterables]

The syntax of a for loop looks like this:

```
l = [1, 7, 8]
for item in l:
    print(item)
```

→ print 1, 7 and 8

Range function in Python

The range function in python is used to generate a sequence of numbers.
We can also specify the start, stop and step-size as follows:

```
range(start, stop, step_size)
```

↳ step size is usually not used with range()

An Example demonstrating range() function

```
for i in range(0, 7):
    print(i)
```

→ range(7) can also be used
→ prints 0 to 6

For loop with else

An optional else can be used with a for loop if the code is to be executed when the loop exhausts

Example :

```
l = [1, 7, 8]
for item in l:
    print(item)
else:
    print("Done")
```

→ This is printed when the loop exhausts!

Output :

```
1
7
8
Done
```

The break statement

'break' is used to come out of the loop when encountered
It instructs the program to - Exit the loop now

Example :

```
for i in range(0,80):
    print(i)
    if i==3:
        break
```

→ This will print 0, 1, 2 and 3

URBAN EDGE

The Continue Statement

'Continue' is used to stop the current iteration of the loop and continue with the next one. It instructs the program to "skip this iteration"

Example:

```
for i in range(4):
    print("printing")
    if i == 2:
        continue
    print(i)
```

⇒ if i is 2, the iteration is skipped.

pass statement

pass is a null statement in python. It instructs to "Do nothing"

Example:

```
l = [1, 7, 8]
for item in l:
    pass
```

→ Without pass, the program will throw an error

URBAN EDGE

Chapter 7 - Practice Set

1. Write a program to print multiplication table of a given number using for loop.

2. Write a program to greet all the person names stored in a list l1 and which starts with S

$l_1 = [$"Harry", "Soham", "Sachin", "Rahul"$]$

3. Attempt problem 1 using while loop.

4. Write a program to find whether a given number is prime or not

5. Write a program to find the Sum of first n natural numbers using while loop.

6. Write a program to calculate the factorial of a given number using for loop.

7. Write a program to print the following star pattern

```
    *
  * * *
* * * * *
```
for n = 3

8. Write a program to print the following star pattern:

```
*
* *
* * *
```
for n = 3

URBAN EDGE

9 Write a program to print the following star pattern

```
* * *
*   *
* * *
```

for $n = 3$

10 Write a program to print multiplication table of n using for loop in reversed order.

URBAN EDGE

Chapter 8 - Functions & Recursions

A function is a group of statements performing a specific task.

When a program gets bigger in size and its complexity grows, it gets difficult for a programmer to keep track on which piece of code is doing what!

A function can be reused by the programmer in a given program any number of

Example and Syntax of a function
The syntax of a function looks as follows:

```
def func1():
    print("Hello")
```

This function can be called any number of times, anywhere in the program.

Function Call
Whenever we want to call a function, we put the name of the function followed by parenthesis as follows:

```
func1()
```

→ This is called function call

function definition
The part containing the exact set of instructions which are executed during the function call.

Quick Quiz: Write a program to greet a user with "Good day" using functions.

Types of functions in Python
There are two types of functions in Python:
1. Built in functions → Already present in Python
2. User defined functions → Defined by the user

Examples of built in function includes len(), print(), range() etc.

The funcI() function we defined is an example of user defined function

Functions with arguments
A function can accept some values it can work with. We can put these values in the parentheses
A function can also return values as shown below:

```
def greet(name):
    gr = "Hello " + name
    return gr

a = greet("Harry")
```

→ "Harry" is passed to greet in name

↳ a will now contain "Hello Harry"

Default Parameter Value
We can have a value as default argument in a function.
If we specify name = 'stranger' in the line containing def, this value is used when no argument is passed.

URBAN EDGE

For example:

```
def greet(name = "Stranger"):
    # function body
```

greet() → Name will be "Stranger" in function body (default)
greet("Harry") → Name will be "Harry" in function body (passed)

Recursion

Recursion is a function which calls itself
It is used to directly use a mathematical formula as a function. For example:

factorial(n) = n × factorial(n-1)

This function can be defined as follows:

```
def factorial(n):
    if i==0 or i==1: → Base condition which doesn't call the function any further
        return 1
    else:
        return n * factorial(n-1) → function calling itself
```

This works as follows:

Factorial(4)
↓ ↳ [Function called]
4 × factorial(3)
4 × [3 ↓ × factorial(2)]
4 × 3 × [2 ↓ × factorial(1)]
4 × 3 × 2 × [1] [Function returned]

The programmer need to be extremely careful while working with recursion to ensure that the function doesnt Infinitely keep calling itself.
Recursion is sometimes the most direct way to code an algorithm

Chapter 8 - Practice Set

1. Write a program using function to find greatest of three numbers

2. Write a python program using function to convert Celsius to fahrenheit.

3. How do you prevent a python print() function to print a new line at the end.

4. Write a recursive function to calculate the sum of first n natural numbers.

5. Write a python function to print first n lines of the following pattern:

```
* * *
* *        → For n = 3
*
```

6. Write a python function which converts inches to cms

7. Write a python function to remove a given word from a list and strip it at the same time.

8. Write a python function to print multiplication table of a given number.

URBAN EDGE

Project 1: Snake, Water, Gun Game

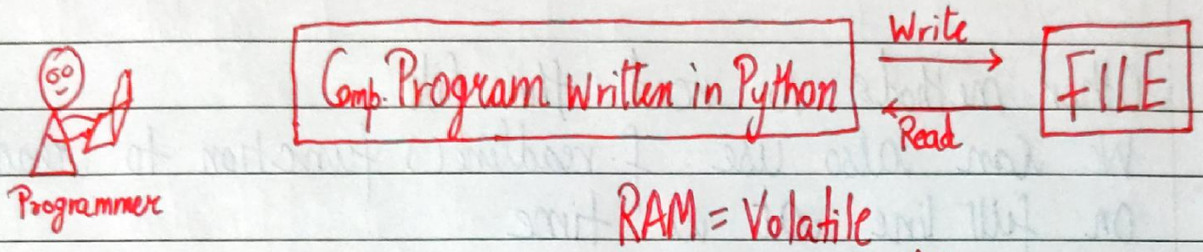
We all have played snake, water gun game in our childhood. If you haven't, google the rules of this game and write a Python program capable of playing this game with the user.

URBAN EDGE

Chapter 9 - File I/O

The random Access memory is volatile and all its contents are lost once a program terminates
In order to persist the data forever, we use files

A file is data stored in a storage device. A Python program can talk to the File by reading content from it and writing content to it



RAM = Volatile
HDD = Non Volatile

Types of Files

There are 2 types of files:

1. Text files (.txt, .c etc)
2. Binary files (.jpg, .dat, etc)

Python has a lot of functions for reading, updating and deleting files.

Opening a file

Python has an open() function for opening files. It takes 2 parameters: filename and mode.

```
open("this.txt", "r")
```

- open is a built-in function
- "this.txt" → Filename
- "r" → mode of opening (read mode)

URBAN EDGE

Reading a file in python

```
f = open("this.txt", "r")  → open the file in r mode
text = f.read()            → Read its contents
print(text)                → Print its contents
f.close()                  → Close the file
```

We can also specify the number of characters in read() function : f.read(2)
↳ Reads first 2 characters

Other methods to read the file
We can also use f.readline() function to read on full line at a time

f.readline() → Reads one line from the file

Modes of opening a file

r → open for reading
w → open for writing
a → open for appending
+ → open for updating

'rb' will open for read in binary mode
'rt' will open for read in text mode

Writing files in Python
In order to write to a file, we first open it in write or append mode after which, we use the python's f.write() method to write to the file!

```
f = open("this.txt", "w")
f.write("This is nice")
f.close()
```

→ Can be called multiple times

With statement

The best way to open and close the file automatically is the with statement

```
with open("this.txt") as f:
    f.read()
```

↳ Dont need to write f.close() as it is done automatically.

URBAN EDGE

Chapter 9 - Practice Set

1. Write a program to read the text from a given file 'poems.txt' and find out whether it contains the word 'twinkle'.

2. The game() function in a program lets a user play a game and returns the score as an integer. You need to read a file 'History.txt' which is either blank or contains the previous Hi-score. You need to write a program to update the Hi-Score whenever game() breaks the Hi-Score.

3. Write a program to generate multiplication tables from 2 to 20 and write it to the different files. Place these files in a folder for a 13-year old.

4. A file contains a word "Donkey" multiple times. You need to write a program which replaces this word with ###### by updating the same file.

5. Repeat program 4 for a list of such words to be censored.

6. Write a program to mine a log file and find out whether it contains 'python'.

7. Write a program to find out the line number where python is present from Ques 6

URBAN EDGE

8 Write a program to make a copy of a text file "this.txt"

9 Write a program to find out whether a file is identical & matches the content of another file.

10 Write a program to wipe out the contents of a file using python

11 Write a python program to rename a file to "renamed_by_python.txt"

URBAN EDGE

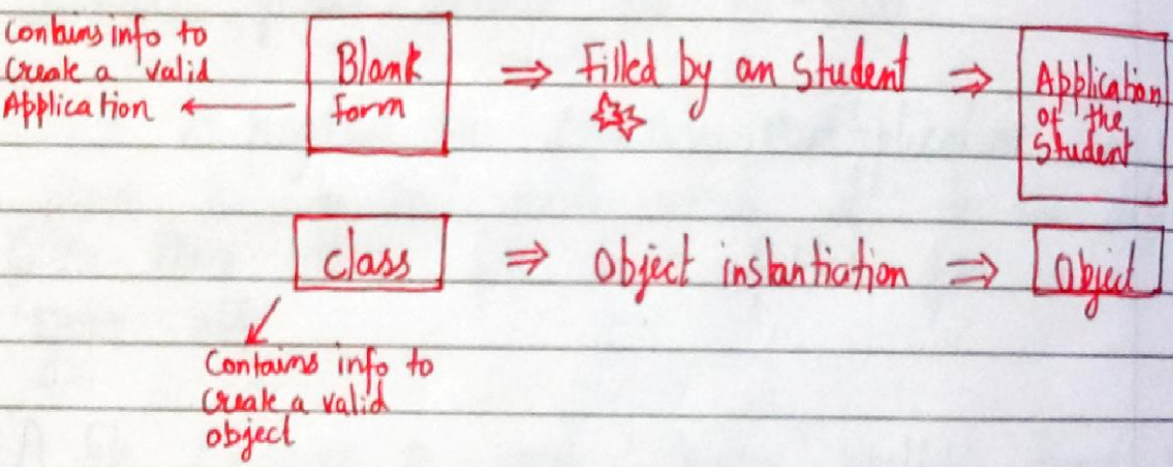
Chapter 10 - Object Oriented Programming

Solving a problem by creating objects is one of the most popular approaches in programming. This is called Object oriented programming.

This concept focuses on using reusable code.
↳ Implements DRY principle

Class

A class is a blueprint for creating objects.



The syntax of a class looks like this:

```
Class Employee:
    # methods & variables
```

[Classname is written in PascalCase]

Object

An Object is an instantiation of a class. When class is defined, a template (info) is defined. Memory is allocated only after object instantiation.

Objects of a given class can invoke the methods available to it without revealing the implementation details to the user. → Abstraction & Encapsulation!

URBAN EDGE

Modelling a problem in OOPs

We identify the following in our problem

Noun → Class → Employee
Adjective → Attributes → name, age, salary
Verbs → Methods → getSalary(), increment()

Class Attributes

An attribute that belongs to the class rather than a particular object.

Example :

```
Class Employee :
    company = "Google"      → [Specific to each class]

harry = Employee()          → object instantiation
harry.company
Employee.company = "YouTube" → changing class attribute
```

Instance Attributes

An attribute that belongs to the Instance (object)
Assuming the class from the previous example :

```
harry.name = "Harry"
harry.salary = "30K"        ⇒ Adding instance attributes
```

Note : Instance attributes take preference over class attributes during assignment & retrieval

harry.attribute1 → ① Is attribute1 present in object ?
② Is attribute1 present in class ?

'self' parameter
self refers to the instance of the class
It is automatically passed with a function call from an object

```
harry.getSalary()
```
→ here self is harry
↳ equivalent to Employee.getSalary(harry)

The function getsalary is defined as:

```
class Employee:
    Company = "Google"
    def getSalary(self):
        print("Salary is not there")
```

Static method
Sometimes we need a function that doesnt use the self parameter. we can define a static method like this:

```
@staticmethod
def greet():
    print("Hello user")
```
→ decorator to mark greet as a static method

__init__() Constructor
__init__() is a special method which is first run as soon as the object is created.
__init__() method is also known as constructor

It takes self argument and can also take further arguments

URBAN EDGE ___/___/___

For Example:

```
Class Employee:
    def __init__(self, name):
        self.name = name

    def getSalary(self):
        ...

harry = Employee("Harry")
```

↳ Object can be instantiated using constructor like this!

URBAN EDGE

Chapter 10 - Practice Set

1. Create a class programmer for storing information of few programmers working at microsoft.

2. Write a class Calculator capable of finding square, cube and squareroot of a number.

3. Create a class with a class attribute a; create an object from it and set a directly using object.a = 0. Does this change the class attribute?

4. Add a static method in problem 2 to greet the user with hello.

5. Write a class Train which has methods to book a ticket, get status (no of seats) and get fare information of trains running under Indian Railways.

6. Can you change the self parameter inside a class to something else (say 'harry'). Try changing self to 'slf' or 'harry' and see the effects.

URBAN EDGE

Chapter 11 - Inheritance & more on OOPs

Inheritance is a way of creating a new class from an existing class

Syntax:

```
Class Employee:                    → Base Class
    # Code
    ...

Class Programmer (Employee):       → Derived or child class
    # Code
```

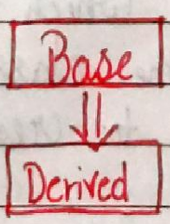
We can use the methods and attributes of Employee in Programmer object.
Also, we can overwrite or add new attributes and methods in Programmer class.

Types of Inheritance

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance

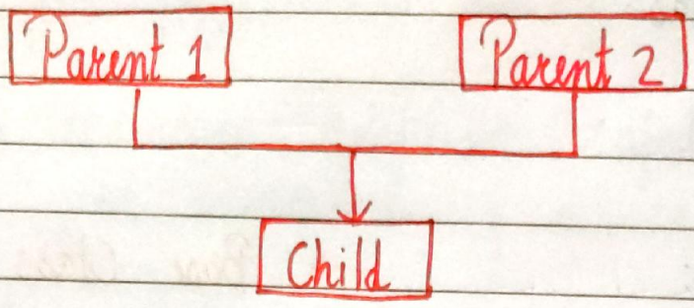
Single Inheritance

Single inheritance occurs when child class inherits only a single parent class



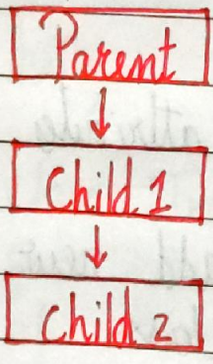
Multiple Inheritance

Multiple inheritance occurs when the child class inherits from more than one parent class.



Multilevel Inheritance

When a child class becomes a parent for another child class



Super() method

Super method is used to access the methods of a super class in the derived class

```
super().__init__()
```

↳ Calls constructor of the base class

Class methods

A class method is a method which is bound to the class and not the object of the class.

@classmethod decorator is used to create a class method

Syntax to create a class method:

```
@classmethod
def (cls, p1, p2):
    ...
```

@property decorators
Consider the following class

```
class Employee:
    @property
    def name(self):
        return self.ename
```

if e = Employee() is an object of class employee, we can print(e.name) to print the ename/call name() function.

@.getters and @.setters
The method name with @property decorator is called getter method
We can define a function + @name.setter decorator like below:

```
@name.setter
def name(self, value):
    self.ename = value
```

Operator overloading in Python
Operators in python can be overloaded using dunder methods.
These methods are called when a given operator is used on the objects.

Operators in python can be overloaded using the following methods:

$p_1 + p_2 \rightarrow p_1.\_\_add\_\_(p_2)$

$p_1 - p_2 \rightarrow p_1.\_\_sub\_\_(p_2)$

$p_1 * p_2 \rightarrow p_1.\_\_mul\_\_(p_2)$

$p_1 / p_2 \rightarrow p_1.\_\_truediv\_\_(p_2)$

$p_1 // p_2 \rightarrow p_1.\_\_floordiv\_\_(p_2)$

Other dunder/magic methods in python

`__str__()` → used to set what gets displayed upon calling `str(obj)`

`__len__()` → used to set what gets displayed upon calling `.__len__()` or `len(obj)`

URBAN EDGE

Chapter 11 - Practice Set

1. Create a class C2dVector and use it to create another class representing a 3-d vector.

2. Create a class pets from a class Animals and further create class Dog from Pets. Add a method bark to class Dog.

3. Create a class Employee and add salary and increment properties to it.
Write a method salaryAfterIncrement method with a @property decorator with a setter which changes the value of increment based on the salary.

4. Write a class Complex to represent complex numbers, along with overloaded operators + and * which adds and multiplies them

5. Write a class vector representing a vector of n dimension. Overload the + and * operator which calculates the sum and the dot product of them.

6. Write __str__() method to print the vector as follows:

$$7\hat{i} + 8\hat{j} + 10\hat{k}$$

Assume vector of dimension 3 for this problem.

URBAN EDGE

7 Override the __len__() method on Vector of problem 5 to display the dimension of the Vector.

URBAN EDGE

Project 2 - The Perfect Guess

We are going to write a program that generates a random number and asks the user to guess it.
If the player's guess is higher than the actual number, the program displays "Lower number please".
Similarly if the user's guess is too low, the program prints "higher number please"
When the user guesses the correct number, the program displays the number of guesses the player used to arrive at the number.

Hint: Use the random module

Chapter 12 - Advanced Python 1

Exception Handling in Python

There are many built-in exceptions which are raised in Python when something goes wrong.
Exceptions in Python can be handled using a try statement. The code that handles the exception is written in the except clause.

```
try:
    # Code                          → Code which might throw Exception
except Exception as e:
    print(e)
```

When the exception is handled, the code flow continues without program interruption.

We can also specify the exceptions to catch like below:

```
try:
    #Code
except ZeroDivisionError:
    # Code
except TypeError:
    # code
except:
    # code          → All other exceptions are handled here.
```

Raising Exceptions

We can raise custom exceptions using the raise keyword in python.

try with else clause
Sometimes we want to run a piece of code when try was successful.

```
try:
    # Some code
except:
    # Some code
else:
    # Code        → This is executed only if the try was successful
```

try with finally
Python offers a finally clause which ensures execution of a piece of code irrespective of the exception.

```
try:
    # Some code
except:
    # Some code
finally:
    # Some code    → executed regardless of error!
```

if __name__ == '__main__' in Python
__name__ evaluates to the name of the module in Python from where the program is ran

If the module is being run directly from the command line, the __name__ is set to string "__main__"
Thus this behaviour is used to check whether the module is run directly or imported to another file.

URBAN EDGE

The global keyword

global keyword is used to modify the variable outside of the current scope.

Enumerate function in Python

The enumerate function adds counter to an iterable and returns it

```
for i, item in list1:
    print(i, item)
```

↳ Prints the items of list 1 with index!

List Comprehensions

list comprehension is an elegant way to create lists based on existing lists

```
list1 = [1, 7, 12, 11, 22]

list2 = [i for item in list1 if item > 8]
```

URBAN EDGE

Chapter 12 - Practice Set

1. Write a program to open three files 1.txt, 2.txt and 3.txt. If any of these files are not present, a message without exiting the program must be printed prompting the same.

2. Write a program to print third, fifth and seventh element from a list using enumerate function

3. Write a list comprehension to print a list which contains the multiplication table of a user entered number.

4. Write a program to display a/b where a and b are integers. If b=0, display Infinite by handling the ZeroDivisionError.

5. Store the multiplication tables generated in Problem 3 in a file named Tables.txt.

URBAN EDGE

Chapter 13 - Advanced Python 2

Virtual Environment

An environment which is same as the System interpretor but is isolated from the other python environments on the system.

Installation

To use virtual environments, we write

```
pip install virtualenv → Install the package
```

We create a new environment using:

```
virtualenv myprojectenv → Creates a new venv
```

The next step after creating the virtual environment is to activate it.
We can now use this virtual environment as a separate python installation.

pip freeze command

pip freeze returns all the packages installed in a given python environment along with the versions

```
" pip freeze > requirements.txt "
```

The above command creats a file named requirements.txt in the same directory containing the output of pip freeze.
We can distribute this file to other users and they can recreate the same environment using:

URBAN EDGE

```
pip install -r requirements.txt
```

Lambda functions

functions created using an expression using lambda keyword

Syntax:

```
lambda arguments : expressions
```

↳ can be used as a normal function

Example:

```
Square = lambda x : x*x
Square(6)   → returns 36

Sum = lambda a,b,c : a+b+c
Sum(1,2,3)  → returns 6
```

.join method (strings)

creates a string from iterable objects

```
l = ["apple", "mango", "banana"]

",and,".join(l)
```

The above line will return "apple,and,mango,and,banana"

format method (strings)

formats the values inside the string into a desired output

```
template.format(p1, p2 ...)
```

↳ arguments

Syntax for format looks like:

"{} is a good {}".format("Harry", "boy") —①

"{1} is a good {0}".format("Harry", "boy") —②

Output for ①
Harry is a good boy

Output for ②
boy is a good Harry

Map, Filter & Reduce
Map applies a function to all the items in an input_list

Syntax:

map(function, input_list)
↳ Can be lambda function

Filter creates a list of items for which the function returns true.

list(filter(function))
↳ Can be a lambda function

Reduce applies a rolling computation to sequential pair of elements

```
from functools import reduce
val = reduce(function, list1)
```
↳ Can be a lambda function

If the function computes sum of two numbers and the

list is [1, 2, 3, 4]

1 2 3 4

3 3 4

6 4

10

=> Sequential Computation

URBAN EDGE

Chapter 13 - Practice Set

1. Create two virtual environments, install few packages in the first one. How do you create a similar environment in the second one?

2. Write a program to input name, marks and phone number of a student and format it using the format function like below:

"The name of the student is Harry, his marks are 72 and phone number is 99999888"

3. A list contains the multiplication table of 7. Write a program to convert it to a vertical string of same numbers (7 / 14 / ⋮)

4. Write a program to filter a list of numbers which are divisible by 5

5. Write a program to find the maximum of the numbers in a list using the reduce function.

6. Run pip freeze for the system interpreter. Take the contents and create a similar virtualenv.

7. Explore the Flask module and create a web server using Flask & Python.

URBAN EDGE

Project 3 - Student Library

Implement a student library system using OOPs where students can borrow a book from the list of books.
Create a separate Library and Student class.
Your program must be menu driven.
You are free to choose methods and attributes of your choice to implement this functionality.

Basics

Basic syntax from the python programming language

Showing Output To User

the print function is used to display or print output

```
print("Content that you wanna print on screen")
```

Taking Input From User

the input function is used to take input from the user

```
var1 = input("Enter your name: ")
```

Empty List

This method allows you to create an empty list

```
my_list = []
```

Empty Dictionary

By putting two curly braces, you can create a blank dictionary

```
my_dict = {}
```

Range Function

range function returns a sequence of numbers, eg, numbers starting from 0 to n-1 for range(0, n)

```
range(int_value)
```

Comments

Comments are used to make the code more understandable for programmers, and they are not executed by compiler or interpreter.

Single line comment

```
#This is a single line comment
```

Multi-line comment

```
'''This is a
multi-line
comment'''
```

Escape Sequence

An escape sequence is a sequence of characters; it doesn't represent itself when used inside string literal or character.

Newline

Newline Character

```
\n
```

Backslash

It adds a backslash

```
\\
```

Single Quote

It adds a single quotation mark

```
\'
```

Tab

It gives a tab space

```
\t
```

Backspace

It adds a backspace

```
\b
```

Octal value

It represents the value of an octal number

```
\ooo
```

Hex value

It represents the value of a hex number

```
\xhh
```

Carriage Return

Carriage return or \r is a unique feature of Python. \r will just work as you have shifted your cursor to the beginning of the string or line.

```
\r
```

Strings

Python string is a sequence of characters, and each character can be individually accessed. Using its index.

String

You can create Strings by enclosing text in both forms of quotes - single quotes or double-quotes.

```
variable_name = "String Data"
```

Slicing

Slicing refers to obtaining a sub-string from the given string.

```
var_name[n : m]
```

String Methods isalnum() method

Returns True if all characters in the string are alphanumeric

```
string_variable.isalnum()
```

isalpha() method

Returns True if all characters in the string are alphabet

```
string_variable.isalpha()
```

isdecimal() method

Returns True if all characters in the string are decimals

```
string_variable.isdecimal()
```

isdigit() method

Returns True if all characters in the string are digits

```
string_variable.isdigit()
```

islower() method

Returns True if all characters in the string are lower case

```
string_variable.islower()
```

isspace() method

Returns True if all characters in the string are whitespaces

```
string_variable.isspace()
```

isupper() method

Returns True if all characters in the string are upper case

```
string_variable.isupper()
```

lower() method

Converts a string into lower case

```
string_variable.lower()
```

upper() method

Converts a string into upper case

```
string_variable.upper()
```

strip() method

It removes leading and trailing spaces in the string

```
string_variable.strip()
```

List

A List in Python represents a list of comma-separated values of any data type between square brackets.

List

```
var_name = [element1, element2, and so on]
```

List Methods index method

Returns the index of the first element with the specified value

```
list.index(element)
```

append method

Adds an element at the end of the list

```
list.append(element)
```

extend method

Add the elements of a list (or any iterable) to the end of the current list

```
list.extend(iterable)
```

insert method

Adds an element at the specified position

```
list.insert(position, element)
```

pop method

Removes the element at the specified position and returns it

```
list.pop(position)
```

remove method

The remove( ) method removes the first occurrence of a given item from the list

```
list.remove(element)
```

clear method

Removes all the elements from the list

```
list.clear()
```

count method

Returns the number of elements with the specified value

```
list.count(value)
```

reverse method

Reverse the order of the list

```
list.reverse()
```

sort method

Sorts the list

```
list.sort(reverse=True|False)
```

Tuples

Tuples are represented as a list of comma-separated values of any data type within parentheses.

Tuple Creation

```
variable_name = (element1, element2, ...)
```

Tuple Methods count method

It returns the number of times a specified value occurs in a tuple

```
tuple.count(value)
```

index method

It searches the tuple for a specified value and returns the position.

```
tuple.index(value)
```

Sets

A set is a collection of multiple values which is both unordered and unindexed. It is written in curly brackets.

Set Creation: Way 1

```
var_name = {element1, element2, ...}
```

Set Creation: Way 2

```
var_name = set([element1, element2, ...])
```

Set Methods: add() method

Adds an element to a set

```
set.add(element)
```

clear() method

Remove all elements from a set

```
set.clear()
```

discard() method

Removes the specified item from the set

```
set.discard(value)
```

intersection() method

Returns intersection of two or more sets

```
set.intersection(set1, set2 ... etc)
```

issubset() method

Checks if a Set is Subset of Another Set

```
set.issubset(set)
```

pop() method

Removes an element from the set

```
set.pop()
```

remove() method

Removes the specified element from the Set

```
set.remove(item)
```

union() method

Returns the union of Sets

```
set.union(set1, set2...)
```

Dictionaries

The dictionary is an unordered set of comma-separated key: value pairs, within {}, with the requirement that within a dictionary, no two keys can be the same.

Dictionary

```
<dictionary-name> = {<key>: value, <key>: value ...}
```

Adding Element to a dictionary

By this method, one can add new elements to the dictionary

```
<dictionary>[<key>] = <value>
```

Updating Element in a dictionary

If the specified key already exists, then its value will get updated

```
<dictionary>[<key>] = <value>
```

Deleting Element from a dictionary

del let to delete specified key: value pair from the dictionary

```
del <dictionary>[<key>]
```

Dictionary Functions & Methods len() method

It returns the length of the dictionary, i.e., the count of elements (key: value pairs) in the dictionary

```
len(dictionary)
```

clear() method

Removes all the elements from the dictionary

```
dictionary.clear()
```

get() method

Returns the value of the specified key

```
dictionary.get(keyname)
```

items() method

Returns a list containing a tuple for each key-value pair

```
dictionary.items()
```

keys() method

Returns a list containing the dictionary's keys

```
dictionary.keys()
```

values() method

Returns a list of all the values in the dictionary

```
dictionary.values()
```

update() method

Updates the dictionary with the specified key-value pairs

```
dictionary.update(iterable)
```

Conditional Statements

The if statements are the conditional statements in Python, and these implement selection constructs (decision constructs).

if Statement

```
if(conditional expression):
statements
```

if-else Statement

```
if(conditional expression):
statements
else:
statements
```

if-elif Statement

```
if (conditional expression) :
statements
elif (conditional expression) :
statements
else :
statements
```

Nested if-else Statement

```
if (conditional expression):
if (conditional expression):
statements
else:
statements
else:
statements
```

Iterative Statements

An iteration statement, or loop, repeatedly executes a statement, known as the loop body, until the controlling expression is false (0).

For Loop

The for loop of Python is designed to process the items of any sequence, such as a list or a string, one by one.

```
for <variable> in <sequence>:
statements_to_repeat
```

While Loop

A while loop is a conditional loop that will repeat the instructions within itself as long as a conditional remains true.

```
while <logical-expression> :
loop-body
```

Break Statement

The break statement enables a program to skip over a part of the code. A break statement terminates the very loop it lies within.

```
for <var> in <sequence> :
statement1
if <condition> :
break
statement2
statement_after_loop
```

Continue Statement

The continue statement skips the rest of the loop statements and causes the next iteration to occur.

```
for <var> in <sequence> :
statement1
if <condition> :
continue
statement2
statement3
statement4
```

Functions

A function is a block of code that performs a specific task. You can pass parameters into a function. It helps us to make our code more organized and manageable.

Function Definition

```
def my_function(parameters):
# Statements
```

File Handling

File handling refers to reading or writing data from files. Python provides some functions that allow us to manipulate data in the files.

open() function

```
var_name = open("file name", "opening mode")
```

close() function

```
var_name.close()
```

Read () function

The read functions contains different methods, read(),readline() and readlines()

```
read() #return one big string
```

It returns a list of lines

```
read-lines
```

It returns one line at a time

```
readline
```

Write () function

This function writes a sequence of strings to the file.

```
write () #Used to write a fixed sequence of characters to a file
```

It is used to write a list of strings

```
writelines()
```

Append () function

The append function is used to append to the file instead of overwriting it. To append to an existing file, simply open the file in append mode (a):

```
file = open("Hello.txt", "a")
```

Exception Handling

An exception is an unusual condition that results in an interruption in the flow of the program.

try and except

A basic try-catch block in python. When the try block throws an error, the control goes to the except block.

```
try:
[Statement body block]
raise Exception()
```

```
except Exception as e:
[Error processing block]
```

OOPS

It is a programming approach that primarily focuses on using objects and classes. The objects can be any real-world entities.

class

The syntax for writing a class in python

```
class class_name:
#Statements
```

class with a constructor

The syntax for writing a class with the constructor in python

```
class CodeWithHarry:

# Default constructor
def __init__(self):
self.name = "CodeWithHarry"

# A method for printing data members
def print_me(self):
print(self.name)
```

object

Instantiating an object

```
<object-name> = <class-name>(<arguments>)
```

filter function

The filter function allows you to process an iterable and extract those items that satisfy a given condition

```
filter(function, iterable)
```

issubclass function

Used to find whether a class is a subclass of a given class (classinfo) or not

```
issubclass(class, classinfo)
```

Iterators and Generators

Here are some of the advanced topics of the Python programming language like iterators and generators

Iterator

Used to create an iterator over an iterable

```
iter_list = iter(['Harry', 'Aakash', 'Rohan'])
print(next(iter_list))
print(next(iter_list))
print(next(iter_list))
```

Generator

Used to generate values on the fly

```
# A simple generator function
def my_gen():
n = 1
print('This is printed first')
# Generator function contains yield statements
yield n
n += 1
print('This is printed second')
yield n
n += 1
print('This is printed at last')
yield n
```

Decorators

Decorators are used to modifying the behavior of function or class. They are usually called before the definition of a function you want to decorate.

property Decorator (getter)

```
@property
def name(self):
return self.__name
```

setter Decorator

It is used to set the property 'name'

```
@name.setter
def name(self, value):
self.__name=value
```

Deletor Decorator

It is used to delete the property 'name'

```
@name.deleter #property-name.deleter decorator
def name(self, value):
print('Deleting..')
del self.__name
```