**Armstrong Number**

**Definition-**

An Armstrong number of three digits is an integer such that the sum of the cubes of its digits is equal to the number itself. For example, 371 is an Armstrong number since 3\*\*3 + 7\*\*3 + 1\*\*3 = 371.

Procedure-

**Armstrong Number Algorithm**

We need two parameters to implement armstrong numbers. One is the number of digits in the number, and second the sum of the digits raised to the power of a number of digits. Let us look at the algorithm to gain a better understanding:

**Algorithm:**

* The number of digits in num is found out
* The individual digits are obtained by performing num mod 10, where the mod is the remainder operation.
* The digit is raised to the power of the number of digits and stored
* Then the number is divided by 10 to obtain the second digit.
* Steps 2,3 and 4 are repeated until the value of num is greater than 0
* Once num is less than 0, end the while loop
* Check if the sum obtained is same as the original number
* If yes, then the number is an armstrong number

**Methods to Find an Armstrong Number using Python**

There are different Python programs associated with finding an Armstrong number. You can check whether a given number is an Armstrong number or not. Alternatively, you can find all the Armstrong numbers within a specified range of numbers. We will go with both these approaches related to the identification of Armstrong numbers, in Python.

*To Check Whether the Given Number is Armstrong*

*# Python program to check if the number is an Armstrong number or not*

*# take input from the user*

*num = int(input("Enter a number: "))*

*# initialize sum*

*sum = 0*

*# find the sum of the cube of each digit*

*temp = num*

*while temp > 0:*

*digit = temp % 10*

*sum += digit \*\* 3*

*temp //= 10*

*# display the result*

*if num == sum:*

*print(num,"is an Armstrong number")*

*else:*

*print(num,"is not an Armstrong number")*

**Palindrome Number**

A **palindrome number** is *a number that is same after reverse*. For example 121, 34543, 343, 131, 48984 are the palindrome numbers.

Palindrome number algorithm

* Get the number from user
* Hold the number in temporary variable
* Reverse the number
* Compare the temporary number with reversed number
* If both numbers are same, print palindrome number
* Else print not palindrome number

*Palindrome Program( String) using inbuilt Method*

*string=input(("Enter a string:"))*

*if(string==string[::-1]):*

*print("The string is a palindrome")*

*else:*

*print("Not a palindrome")*

# Stack in Python

A **stack** is a linear data structure that stores items in a *Last***-***In***/***First***-***Out* **(***LIFO***)** or First-In/Last-Out (FILO) manner. In stack, a new element is added at one end and an element is removed from that end only. The insert and delete operations are often called push and pop.



**The functions associated with stack are:**

* **empty()** – Returns whether the stack is empty – Time Complexity: O(1)
* **size()** – Returns the size of the stack – Time Complexity: O(1)
* **top() / peek()**– Returns a reference to the topmost element of the stack – Time Complexity: O(1)
* **push(a)** – Inserts the element ‘a’ at the top of the stack – Time Complexity: O(1)
* **pop()** – Deletes the topmost element of the stack – Time Complexity: O(1)

### Implementation:

There are various ways from which a stack can be implemented in Python. This article covers the implementation of a stack using data structures and modules from the Python library.   
Stack in Python can be implemented using the following ways:

* list
* Collections.deque
* queue.LifoQueue

### Implementation using list:

Python’s built-in data structure list can be used as a stack. Instead of push(), append() is used to add elements to the top of the stack while pop() removes the element in LIFO order.   
Unfortunately, the list has a few shortcomings. The biggest issue is that it can run into speed issues as it grows. The items in the list are stored next to each other in memory, if the stack grows bigger than the block of memory that currently holds it, then Python needs to do some memory allocations. This can lead to some append() calls taking much longer than other ones.

*# Python program to demonstrate stack implementation using list*

*stack = [ ]*

*# append() function to push*

*# element in the stack*

*stack.append('a')*

*stack.append('b')*

*stack.append('c')*

*print('Initial stack')*

*print(stack)*

*# pop() function to pop*

*# element from stack in*

*# LIFO order*

*print('\nElements popped from stack:')*

*print(stack.pop())*

*print(stack.pop())*

*print(stack.pop())*

*print('\nStack after elements are popped:')*

*print(stack)*

*# uncommenting print(stack.pop())*

*# will cause an IndexError*

*# as the stack is now empty*

**Output**

Initial stack

['a', 'b', 'c']

Elements popped from stack:

c

b

a

Stack after elements are popped [ ]

# Queue in Python

Like stack, queue is a linear data structure that stores items in First In First Out (FIFO) manner. With a queue the least recently added item is removed first. A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.



Operations associated with queue are: 

* **Enqueue:** Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition – Time Complexity : O(1)
* **Dequeue:** Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition – Time Complexity : O(1)
* **Front:** Get the front item from queue – Time Complexity : O(1)
* **Rear:** Get the last item from queue – Time Complexity : O(1)

## Implementation

There are various ways to implement a queue in Python. This article covers the implementation of queue using data structures and modules from Python library.  
Queue in Python can be implemented by the following ways:

* list
* collections.deque
* queue.Queue

#### Implementation using list

List is a Python’s built-in data structure that can be used as a queue. Instead of enqueue() and dequeue(), append() and pop() function is used. However, lists are quite slow for this purpose because inserting or deleting an element at the beginning requires shifting all of the other elements by one, requiring O(n) time.

# Python program to

# demonstrate queue implementation

# using list

# Initializing a queue

queue = []

# Adding elements to the queue

queue.append('a')

queue.append('b')

queue.append('c')

print("Initial queue")

print(queue)

# Removing elements from the queue

print("\nElements dequeued from queue")

print(queue.pop(0))

print(queue.pop(0))

print(queue.pop(0))

print("\nQueue after removing elements")

print(queue)

# Uncommenting print(queue.pop(0))

# will raise and IndexError

# as the queue is now empty

**Output:** 

Initial queue

['a', 'b', 'c']

Elements dequeued from queue

a

b

c

Queue after removing elements

[]

Traceback (most recent call last):

File "/home/ef51acf025182ccd69d906e58f17b6de.py", line 25, in

print(queue.pop(0))

IndexError: pop from empty list

**Duck number**

A Duck number is a positive number which has zeroes present in it, For example 3210, 8050896, 70709 are all Duck numbers. Please note that a numbers with only leading 0s is not considered as Duck Number. For example, numbers like 035 or 0012 are not considered as Duck Numbers. A number like 01203 is considered as Duck because there is a non-leading 0 present in it.

**Examples :**

*Input : 707069   
Output : It is a duck number.   
Explanation: 707069 does not contains zeros at the beginning.*

*Input : 02364   
Output : It is not a duck number.   
Explanation: in 02364 there is a zero at the beginning of the number.*

***# Python program to check whether a number is Duck Number or not.***

*# Function to check whether*

*# the given number is duck number or not.*

*def check\_duck(num) :*

*# Length of the number(number of digits)*

*n = len(num)*

*# Ignore leading 0s*

*i = 0*

*while (i < n and num[i] == '0') :*

*i = i + 1*

*# Check remaining digits*

*while (i < n) :*

*if (num[i] == "0") :*

*return True*

*i = i + 1*

*return False*

*# Driver Method*

*num1 = "1023"*

*if(check\_duck(num1)) :*

*print "It is a duck number"*

*else :*

*print "It is not a duck number"*

**Output**

It is a duck number

**Time Complexity:** **O(n)**where n is length of string.  
**Auxiliary Space: O(1)**

**Magic number**

# Check if a number is magic (Recursive sum of digits is 1)

A number is said to be a magic number, if the sum of its digits are calculated till a single digit recursively by adding the sum of the digits after every addition. If the single digit comes out to be 1,then the number is a magic number.

**For example-**   
Number= 50113   
=> 5+0+1+1+3=10   
=> 1+0=1   
This is a**Magic Number**

**For example-**   
Number= 1234   
=> 1+2+3+4=10   
=> 1+0=1   
This is a **Magic Number**

**Examples :**

Input : 1234

Output : Magic Number

Input : 12345

Output : Not a magic Number

The approach used brute force. The function keeps adding digits until a single digit sum is reached. To understand how i am calculating the sum up to a single digit .

**Python3 program to check if a number is Magic number.**

def isMagic(n):

sum = 0;

# Note that the loop

# continues if n is 0

# and sum is non-zero.

# It stops when n becomes

# 0 and sum becomes single

# digit.

while (n > 0 or sum > 9):

if (n == 0):

n = sum;

sum = 0;

sum = sum + n % 10;

n = int(n / 10);

# Return true if

# sum becomes 1.

return True if (sum == 1) else False;

# Driver code

n = 1234;

if (isMagic(n)):

print("Magic Number");

else:

print("Not a magic Number");

**Calculator**

Create a simple calculator which can perform basic arithmetic operations like addition, subtraction, multiplication, or division depending upon the user input.

Approach :

* User chooses the desired operation. Options 1, 2, 3, and 4 are valid.
* Two numbers are taken and an if…elif…else branching is used to execute a particular section.
* Using functions add(), subtract(), multiply() and divide() evaluate respective operations.

Please select operation -

1. Add

2. Subtract

3. Multiply

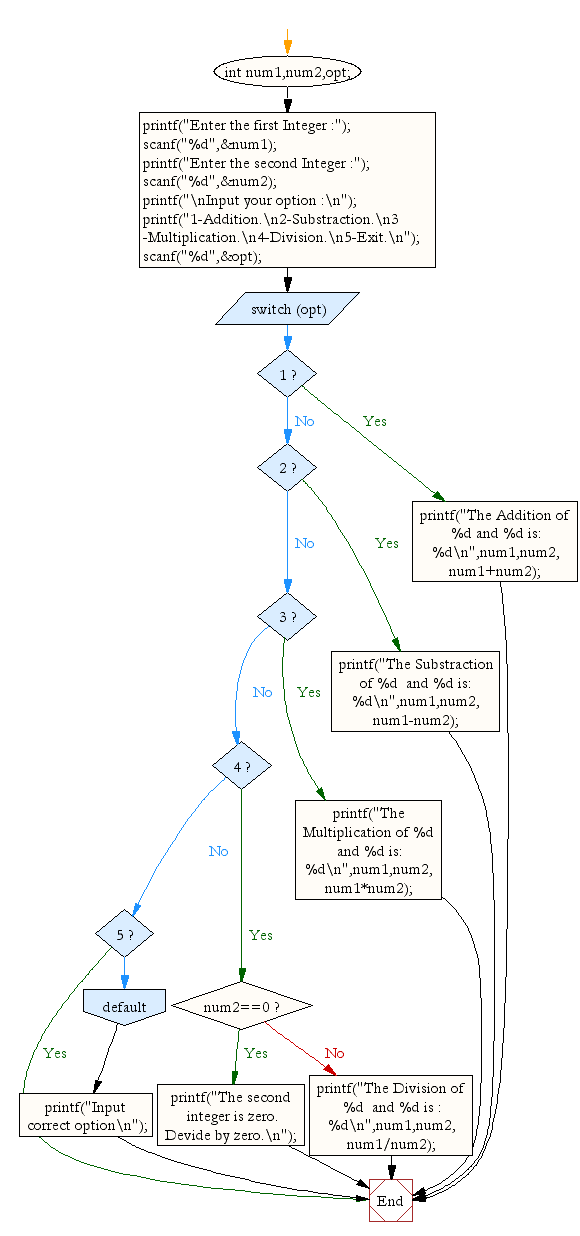
4. Divide

Select operations form 1, 2, 3, 4 : 1

Enter first number : 20

Enter second number : 13

20 + 13 = 33



**Python program for simple calculator**

# Function to add two numbers

def add(num1, num2):

return num1 + num2

# Function to subtract two numbers

def subtract(num1, num2):

return num1 - num2

# Function to multiply two numbers

def multiply(num1, num2):

return num1 \* num2

# Function to divide two numbers

def divide(num1, num2):

return num1 / num2

print("Please select operation -\n" \

"1. Add\n" \

"2. Subtract\n" \

"3. Multiply\n" \

"4. Divide\n")

# Take input from the user

select = int(input("Select operations form 1, 2, 3, 4 :"))

number\_1 = int(input("Enter first number: "))

number\_2 = int(input("Enter second number: "))

if select == 1:

print(number\_1, "+", number\_2, "=",

add(number\_1, number\_2))

elif select == 2:

print(number\_1, "-", number\_2, "=",

subtract(number\_1, number\_2))

elif select == 3:

print(number\_1, "\*", number\_2, "=",

multiply(number\_1, number\_2))

elif select == 4:

print(number\_1, "/", number\_2, "=",

divide(number\_1, number\_2))

else:

print("Invalid input")

**Output:**

Please select operation -

1. Add

2. Subtract

3. Multiply

4. Divide

Select operations form 1, 2, 3, 4 : 1

Enter first number : 15

Enter second number : 14

15 + 14 = 29

**Calculate perimeter/circumference and area of shapes such as  triangle, rectangle, square and circle.**

#We will make use of user defined python functions for this task.

#we will import math library to use sqrt(), this will used to #calculate square root.

import math  
def area\_square(a):  
    area1=float(a\*a);  
    print("Area of square is:",area1)  
def area\_circle(r):  
    area2=float(3.14\*r\*r);  
    print("Area of circle is:",area2)  
def area\_rectangle(a,b):  
    area3=float(a\*b);  
    print("Area of rectangle is:",area3)  
def area\_triangle(x,y):  
    area4=float((x\*y)/2);  
    print("Area of triangle is:",area4)  
def peri\_square(a):  
    peri1=float(4\*a);  
    print("Perimeter of square is:",peri1)  
def peri\_circle(r):  
    peri2=float(2\*3.14\*r);  
    print("Perimter of circle is:",peri2)  
def peri\_triangle(a,b):  
    hypotenuse=float(math.sqrt(a\*a+b\*b))  
    peri3=float(a+b+hypotenuse)  
    print("Perimter of right angled triangle is:",peri3)  
def peri\_rectangle(a,b):  
    peri4=float(2\*(a+b))  
    print("Perimter of rectangle is:",peri4)  
  
side=float(input("enter the side of square:"))  
area\_square(side)  
print()  
peri\_square(side)  
radius=float(input("enter the radius of circle:"))  
area\_circle(radius)  
peri\_circle(radius)  
length=float(input("enter the length of rectangle:"))  
breadth=float(input("enter the breadth of rectangle:"))  
area\_rectangle(length,breadth)  
peri\_rectangle(length,breadth)  
base=float(input("enter the base of right angled triangle:"))  
height=float(input("enter the height of right angled triangle:"))  
area\_triangle(base,height)  
peri\_triangle(base,height)

# Program to calculate Volume and Surface area of Hemisphere

import math

# Function to calculate volume

def volume(r):

volume = 2 \* math.pi \* math.pow(r, 3) / 3

print("Volume = ", '%.4f' %volume)

# Function to calculate surface area

def surface\_area(r):

s\_area = 2 \* math.pi \* math.pow(r, 2)

print("Surface Area = ", '%.4f' %s\_area)

# Driver code

r = 11

volume(r)

surface\_area(r)

**Output**

Volume = 2787.64

Surface Area = 760.265

**Time complexity:** O(1), as calculating square and cube using pow function takes constant operations.  
**Auxiliary space:** O(1)

**Python: Surface volume and area of a sphere**

pi=22/7

radian = float(input('Radius of sphere: '))

sur\_area = 4 \* pi \* radian \*\*2

volume = (4/3) \* (pi \* radian \*\* 3)

print("Surface Area is: ", sur\_area)

print("Volume is: ", volume)

Sample Output:

Radius of sphere: .75

Surface Area is: 7.071428571428571

Volume is: 1.7678571428571428

# Calculate Volume, Curved Surface Area and Total Surface Area Of Cylinder

# Importing Math Library For The Value Of PI

import math

pi = math.pi

# Function To Calculate Volume OF Cylinder

def volume(r, h):

vol = pi \* r \* r \* h

return vol

# Function To Calculate Total Surface Area

# of Cylinder

def totalsurfacearea(r, h):

tsurf\_ar = (2 \* pi \* r \* h) + (2 \* pi \* r \* r)

return tsurf\_ar

# Function To Calculate Curved Surface Area

# of Cylinder

def curvedsurfacearea(r, h):

cursurf\_ar = (2 \* pi \* r \* h)

return cursurf\_ar

# Driver Code

r = 5

h = 8

# Function Call And Printing Volume, TSA, CSA Of Cylinder

print("Volume Of Cylinder = ",volume(r, h))

print("Total Surface Area Of Cylinder = ",totalsurfacearea(r,h))

print("Curved Surface Area Of Cylinder = ",curvedsurfacearea(r,h))

**Output:**

Volume Of Cylinder = 628.3185307179587

Total Surface Area Of Cylinder = 408.4070449666731

Total Surface Area Of Cylinder = 251.32741228718345

**Time complexity**: O(1)   
**Auxiliary Space** : O(1)

# Python Program to find volume, surface area and space diagonal of a cuboid

# Python program to find the

# Surface area, volume and

# space diagonal of rectangular

# prism

import math

# function to calculate

# Surface area

def find\_surafce\_area(l, b, h):

    # formula of surface\_area = 2(lb + bh + hl)

    Surface\_area = 2 \* ( l \* b + b \* h + h \* l)

    # Display surface area

    print(Surface\_area)

# function to find the

# Volume of rectangular

# prism

def find\_volume(l, b, h):

    # formula to calculate

    # volume = (l \* b\*h)

    Volume = (l \* b \* h)

    # Display volume

    print(Volume)

    categories Most Used

 School Programming

 Aptitude

 Re

def find\_space\_diagonal(l, b, h):

    # formula to calculate

    # Space diagonal = square\_root(l\*\*2 + b\*\*2 + h\*\*2)

    Space\_diagonal = math.sqrt(l\*\*2 + b\*\*2 + h\*\*2)

    # display space diagonal

    print(Space\_diagonal)

# Driver Code

l = 9

b = 6

h = 10

# surface area

# function call

find\_surafce\_area(l, b, h)

# volume function call

find\_volume(l, b, h)

# Space diagonal function call

find\_space\_diagonal(l, b, h)

**Output:**

408

540

14.730919862656235

**Time Complexity:**O(logn)   
**Auxiliary Space:**O(1)

**Python program to convert decimal to binary, octal, hexadecimal**

## Source Code

# Python program to convert decimal into other number systems

dec = 344

print("The decimal value of", dec, "is:")

print(bin(dec), "in binary.")

print(oct(dec), "in octal.")

print(hex(dec), "in hexadecimal.")

**Output**

The decimal value of 344 is:

0b101011000 in binary.

0o530 in octal.

0x158 in hexadecimal.

### Program to print half pyramid a using numbers

rows = int(input("Enter number of rows: "))

for i in range(rows):

for j in range(i+1):

print(j+1, end=" ")

print("\n")

### Inverted half pyramid using numbers

rows = int(input("Enter number of rows: "))

for i in range(rows, 0, -1):

for j in range(1, i+1):

print(j, end=" ")

print("\n")

### Program to print full pyramid using \*

rows = int(input("Enter number of rows: "))

k = 0

for i in range(1, rows+1):

for space in range(1, (rows-i)+1):

print(end=" ")

while k!=(2\*i-1):

print("\* ", end="")

k += 1

k = 0

print()

### Inverted full pyramid of \*

rows = int(input("Enter number of rows: "))

for i in range(rows, 1, -1):

for space in range(0, rows-i):

print(" ", end="")

for j in range(i, 2\*i-1):

print("\* ", end="")

for j in range(1, i-1):

print("\* ", end="")

print()

### Floyd's Triangle

rows = int(input("Enter number of rows: "))

number = 1

for i in range(1, rows+1):

for j in range(1, i+1):

print(number, end=" ")

number += 1

print()

### Program to print half pyramid using alphabets

rows = int(input("Enter number of rows: "))

ascii\_value = 65

for i in range(rows):

for j in range(i+1):

alphabet = chr(ascii\_value)

print(alphabet, end=" ")

ascii\_value += 1

print("\n")