# UNIT - II

# TYPES, OPERATORS AND EXPRESSIONS

## Data Types:

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has some standard data types:

- Numbers
- String
- Boolean
- List
- Tuple
- Set
- Dictionary

## Python Numbers:

Number data types store numeric values. Number objects are created when you assign a value to them.

Python supports four different numerical types:

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Python allows you to use a lowercase L with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.

A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x is the real part and b is the imaginary part of the complex number.

For example:

**Example:**

```
a = 3
b = 2.65
c = 98657412345L
d = 2+5j
print "int is",a
print "float is",b
print "long is",c
print "complex is",d
```

**Output:**

```
int is 3
float is 2.65
```

long is 98657412345
complex is (2+5j)

## Python Strings:

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. For example:

**Example:**

```
str ="WELCOME"
print str # Prints complete string
print str[0] # Prints first character of the string
print str[2:5] # Prints characters starting from 3rd to 5th
print str[2:] # Prints string starting from 3rd character
print str * 2 # Prints string two times
print str + "CSE" # Prints concatenated string
```

**Output:**

```
WELCOME
W
LCO
LCOME
WELCOMEWELCOME
WELCOMECSE
```

**Some Built-in String methods for Strings:**

| SNO | Method Name | Description |
|---|---|---|
| 1 | capitalize() | Capitalizes first letter of string. |
| 2 | center(width, fillchar) | Returns a space-padded string with the original string centered to a total of width columns. |
| 3 | count(str, beg= 0,end=len(string)) | Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given. |
| 4 | isalpha() | Returns true if string has at least 1 character and all characters are alphabetic and false otherwise. |
| 5 | isdigit() | Returns true if string contains only digits and false otherwise. |
| 6 | islower() | Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise. |
| 7 | isnumeric() | Returns true if a unicode string contains only numeric characters and false otherwise. |
| 8 | isspace() | Returns true if string contains only whitespace |

| | | |
|---|---|---|
| | | characters and false otherwise. |
| 9 | istitle() | Returns true if string is properly "titlecased" and false otherwise. |
| 10 | isupper() | Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise. |
| 11 | len(string) | Returns the length of the string. |
| 12 | lower() | Converts all uppercase letters in string to lowercase. |

**Example:**

```
str1="welcome"
print "Capitalize function---",str1.capitalize()
print str1.center(15,"*")
print "length is",len(str1)
print "count function---",str1.count('e',0,len(str1))
print "endswith function---",str1.endswith('me',0,len(str1))
print "startswith function---",str1.startswith('me',0,len(str1))
print "find function---",str1.find('e',0,len(str1))
str2="welcome2017"
print "isalnum function---",str2.isalnum()
print "isalpha function---",str2.isalpha()
print "islower function---",str2.islower()
print "isupper function---",str2.isupper()
str5="welcome to java"
print "replace function---",str5.replace("java","python")
```

**Output:**

```
Capitalize function--- Welcome
****welcome****
length is 7
count function--- 2
endswith function--- True
startswith function--- False
find function--- 1
isalnum function--- True
isalpha function--- False
islower function--- True
isupper function--- False
replace function--- welcome to python
```

## Python Boolean:

Booleans are identified by True or False.

Example:

**Example:**

```
a = True
b = False
print a
```

```
print b
```

**Output:**

```
True
False
```

## Operators:

## Types of Operators:

Python language supports the following types of operators.

- Arithmetic Operators **+, -, *, /, %, **, //**
- Comparison (Relational) Operators **= =, ! =, < >, <, >, <=, >=**
- Assignment Operators **=, +=, -=, *=, /=, %=, **=, //=**
- Logical Operators **and, or, not**
- Bitwise Operators **&, |, ^, ~,<<, >>**
- Membership Operators **in, not in**
- Identity Operators **is, is not**

## Arithmetic Operators:

Some basic arithmetic operators are +, -, *, /, %, **, and //. You can apply these operators on numbers as well as variables to perform corresponding operations.

| Operator | Description | Example |
|---|---|---|
| + Addition | Adds values on either side of the operator. | a + b = 30 |
| - Subtraction | Subtracts right hand operand from left hand operand. | a – b = -10 |
| * Multiplication | Multiplies values on either side of the operator | a * b = 200 |
| / Division | Divides left hand operand by right hand operand | b / a = 2 |
| % Modulus | Divides left hand operand by right hand operand and returns remainder | b % a = 0 |
| ** Exponent | Performs exponential (power) calculation on operators | a**b =10 to the power 20 |
| // Floor Division | The division of operands where the result is the quotient in which the digits after the decimal point are removed. | 9//2 = 4 and 9.0//2.0 = 4.0 |

**Example:**

```
a = 21
b = 10
print "Addition is", a + b
print "Subtraction is ", a - b
print "Multiplication is ", a * b
print "Division is ", a / b
print "Modulus is ", a % b
a = 2
```

```
b = 3
print "Power value is ", a ** b


a = 10
b = 4
print "Floor Division is ", a // b
```

**Output:**

```
Addition is 31
Subtraction is 11
Multiplication is 210
Division is 2
Modulus is 1
Power value is 8
Floor Division is 2
```

## Comparison (Relational) Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

| Operator | Description | Example |
|---|---|---|
| = = | If the values of two operands are equal, then the condition becomes true. | (a == b) is not true. |
| != | If values of two operands are not equal, then condition becomes true. | (a != b) is true. |
| < > | If values of two operands are not equal, then condition becomes true. | (a <> b) is true. This is similar to != operator. |
| > | If the value of left operand is greater than the value of right operand, then condition becomes true. | (a > b) is not true. |
| < | If the value of left operand is less than the value of right operand, then condition becomes true. | (a < b) is true. |
| > = | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. | (a >= b) is not true. |
| < = | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. | (a <= b) is true. |

**Example:**

```
a=20
b=30
if a < b:
    print "b is big"
elif a > b:
    print "a is big"
else:
print "Both are equal"
```

**Output:**

b is big

## Assignment Operators

| Operator | Description | Example |
|---|---|---|
| = | Assigns values from right side operands to left side operand | c = a + b assigns value of a + b into c |
| += Add AND | It adds right operand to the left operand and assign the result to left operand | c += a is equivalent to c = c + a |
| -= Subtract AND | It subtracts right operand from the left operand and assign the result to left operand | c -= a is equivalent to c = c - a |
| *= Multiply AND | It multiplies right operand with the left operand and assign the result to left operand | c *= a is equivalent to c = c * a |
| /= Divide AND | It divides left operand with the right operand and assign the result to left operand | c /= a is equivalent to c = c / a |
| %= Modulus AND | It takes modulus using two operands and assign the result to left operand | c %= a is equivalent to c = c % a |
| **= Exponent AND | Performs exponential (power) calculation on operators and assign value to the left operand | c **= a is equivalent to c = c ** a |
| //= Floor Division | It performs floor division on operators and assign value to the left operand | c //= a is equivalent to c = c // a |

**Example:**

```
a=82
b=27
a += b
print a
a=25
b=12
a -= b
print a
a=24
b=4
a *= b
print a
a=4
b=6
a **= b
print a
```

**Output:**

```
109
13
96
4096
```

## Logical Operators

| Operator | Description | Example |
|---|---|---|
| And<br>Logical AND | If both the operands are true then condition becomes true. | (a and b) is true. |
| Or<br>Logical OR | If any of the two operands are non-zero then condition becomes true. | (a or b) is true. |
| not<br>Logical NOT | Used to reverse the logical state of its operand. | Not (a and b) is false. |

**Example:**

```
a=20
b=10
c=30
if a >= b and a >= c:
   print "a is big"
elif b >= a and b >= c:
   print "b is big"
else:
   print "c is big"
```

**Output:**

```
c is big
```

## Bitwise Operators

| Operator | Description | Example |
|---|---|---|
| &<br>Binary AND | Operator copies a bit to the result if it exists in both operands. | (a & b) = 12<br>(means 0000 1100) |
| \|<br>Binary OR | It copies a bit if it exists in either operand. | (a \| b) = 61<br>(means 0011 1101) |
| ^<br>Binary XOR | It copies the bit if it is set in one operand but not both. | (a ^ b) = 49<br>(means 0011 0001) |
| ~<br>Binary Ones Complement | It is unary and has the effect of 'flipping' bits. | (~a ) = -61 (means 1100 0011 in 2's complement form due to a signed binary number. |
| <<<br>Binary Left Shift | The left operands value is moved left by the number of bits specified by the right operand. | a << 2 = 240<br>(means 1111 0000) |
| >><br>Binary Right Shift | The left operands value is moved right by the number of bits specified by the right operand. | a >> 2 = 15<br>(means 0000 1111) |

## Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples.

| Operator | Description | Example |
|---|---|---|
| in | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | x in y, here in results in a 1 if x is a member of sequence y. |
| not in | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. | x not in y, here not in results in a 1 if x is not a member of sequence y. |

**Example:**

```
a = 3
list = [1, 2, 3, 4, 5 ];
if ( a in list ):
        print "available"
                                else:   print " not available"
```

**Output:**

available

## Identity Operators

Identity operators compare the memory locations of two objects.

| Operator | Description | Example |
|---|---|---|
| is | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. | x is y, here is results in 1 if id(x) equals id(y). |
| is not | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. | x is not y, here is not results in 1 if id(x) is not equal to id(y). |

**Example:**

```
a = 20
b = 20
if ( a is b ):
        print "Line 1 - a and b have same identity"
else:
        print "Line 1 - a and b do not have same identity"
if ( id(a) == id(b) ):
        print "Line 2 - a and b have same identity"
                                else:
```

**Output:**

```
print "Line 2 - a and b do not have same identity"
```

Line 1 - a and b have same identity
Line 2 - a and b have same identity

## Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

| Operator | Description |
|---|---|
| () | Parenthesis |
| ** | Exponentiation (raise to the power) |
| ~ x, +x, -x | Complement, unary plus and minus |
| * / % // | Multiply, divide, modulo and floor division |
| + - | Addition and subtraction |
| >> << | Right and left bitwise shift |
| & | Bitwise 'AND' |
| ^ \| | Bitwise exclusive `OR' and regular `OR' |
| <= < > >= | Comparison operators |
| <> == != | Equality operators |
| = %= /= //= -= += *= **= | Assignment operators |
| is is not | Identity operators |
| in not in | Membership operators |
| not or and | Logical operators |

## Expression:

An expression is a combination of variables constants and operators written according to the syntax of Python language. In Python every expression evaluates to a value i.e., every expression results in some value of a certain type that can be assigned to a variable. Some examples of Python expressions are shown in the table given below.

| Algebraic Expression | Python Expression |
|---|---|
| a x b – c | a * b – c |
| (m + n) (x + y) | (m + n) * (x + y) |
| (ab / c) | a * b / c |
| $3x^2$ +2x + 1 | 3*x*x+2*x+1 |
| (x / y) + c | x / y + c |

## Evaluation of Expressions

Expressions are evaluated using an assignment statement of the form

**Variable = expression**

Variable is any valid variable name. When the statement is encountered, the expression is evaluated first and then replaces the previous value of the variable on the left hand side. All variables used in the expression must be assigned values before evaluation is attempted.

**Example:**

```
a=10
b=22
c=34
x=a*b+c
 y=a-b*c
 z=a+b+c*c-a
 print "x=",x
 print "y=",y
   print "z=",z
```

**Output:**

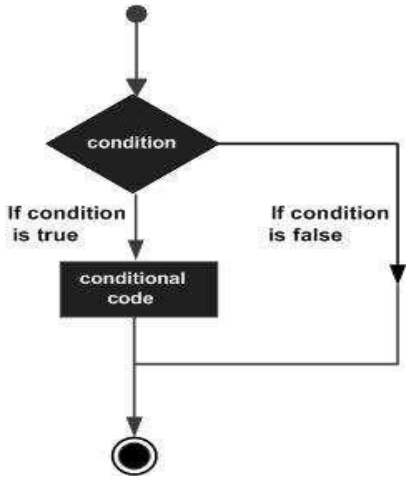
```
x= 254
y= -738
z= 1178
```

# Control Flow:

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce True or False as outcome. You need to determine which action to take and which statements to execute if outcome is True or False otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages:



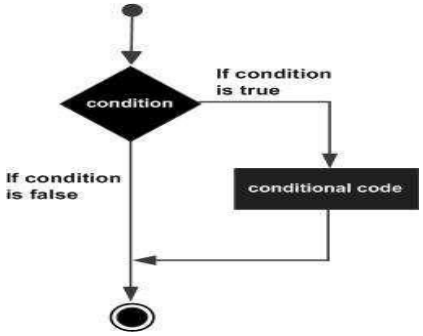

Python programming language assumes any non-zero and non-null values as True, and if it is either zero or null, then it is assumed as False value.

| Statement | Description |
|---|---|
| if statements | **if statement** consists of a boolean expression followed by one or more statements. |
| if...else statements | **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE. |

| | |
|---|---|
| nested if statements | You can use one **if** or **else if** statement inside another **if** or **else if** statement(s). |

## The if Statement

It is similar to that of other languages. The **if** statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.




**Syntax:**

if condition:
    statements

First, the condition is tested. If the condition is True, then the statements given after colon (:) are executed. We can write one or more statements after colon (:).

**Example:**

```
a=
 10
b
=
 15
if a < b:
print "B is big"
    print "B value
    is",b
```

**Output:**

B is big
B value is 15

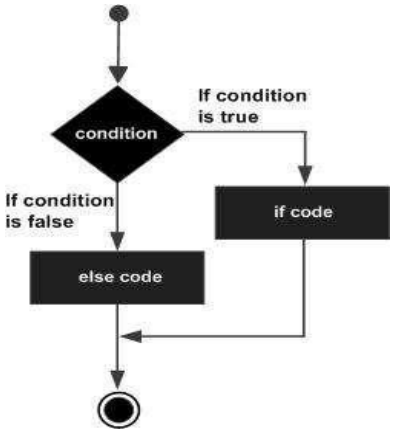
## The if ... else statement

An **else** statement can be combined with an **if** statement. An **else** statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

The else statement is an optional statement and there could be at most only one **else** statement following **if**.

**Syntax:**

```
if condition:
    statement(s
    )
else:
statement(s)
```




**Example:**

```
a=
48
b
=
34
if a < b:
print “B is big”
    print “B value
    is”, b
else:
print “A is big”
    print “A value
    is”, a
print “END”
```

**Output:**

A is big
A value is 48
END

**Q) Write a program for checking whether the given number is even or not.**

**Program:**

```
a=input("Enter a value: ")
if a%2==0:
   print "a is EVEN number"
else:
   print "a is NOT EVEN Number"
```

**Output-1:**

Enter a value: 56
a is EVEN Number

**Output-2:**

Enter a value: 27
a is NOT EVEN Number

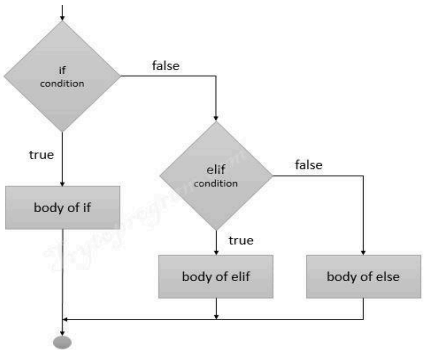
## The elif Statement

The **elif** statement allows you to check multiple expressions for True and execute a block of code as soon as one of the conditions evaluates to True.

Similar to the **else**, the **elif** statement is optional. However, unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if**.

**Syntax:**

```
if condition1:
   statement(s)
elif condition2:
   statement
 (s) else:
statement(s)
```




**Example:**

```
a=
20
b
=
10
c=
30
if a >= b and a
   >= c: print "a
   is big"
elif b >= a and b
   >= c: print "b is
   big"
else:
print "c is big"
```

**Output:**

c is big

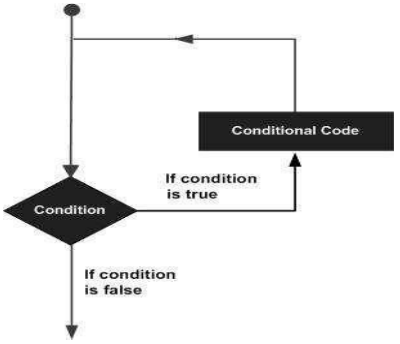
# Decision Loops

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple

times. The following diagram illustrates a loop statement:




Python programming language provides following types of loops to handle looping requirements.

| Loop Type | Description |
|---|---|
| while loop | Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body. |
| for loop | Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| nested loops | You can use one or more loop inside any another while, for loop. |

## The while Loop

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is True.

**Syntax**

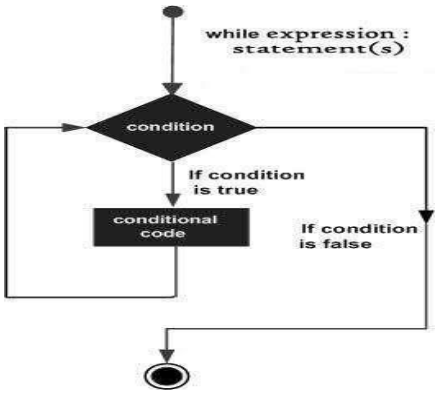
The syntax of a **while** loop in Python programming language is:

```
while expression:
    statement(s)
```

Here, **statement(s)** may be a single statement or a block of statements.

The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.


while expression :
statement(s)
condition
If condition is true
conditional code
If condition is false

**Example-1:**

```
i=1
while i < 4:
print i i+=1
print "END"
```

**Example-2:**

```
i=1
while i < 4:
print i i+=1
print "END"
```

**Output-1:**

```
1
END
2
END
3
END
```

**Output-2:**

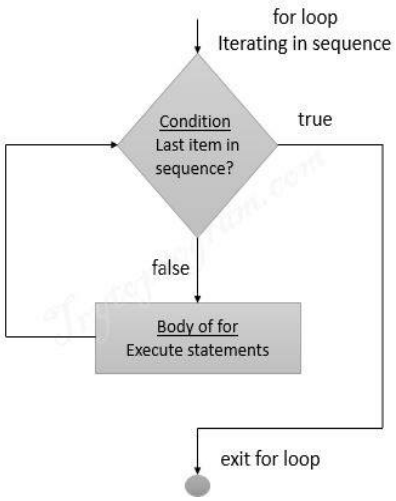
```
1
2
3
END
```

## The for loop:

The **for** loop is useful to iterate over the elements of a sequence. It means, the **for** loop can be used to execute a group of statements repeatedly depending upon the number of elements in the sequence. The **for** loop can work with sequence like string, list, tuple, range etc.

The syntax of the for loop is given below:

```
for var in sequence:
    statement (s)
```




The first element of the sequence is assigned to the variable written after „for" and then the statements are executed. Next, the second element of the

sequence is assigned to the variable and then the statements are executed second time. In this way, for each element of the sequence, the statements are executed once. So, the **for** loop is executed as many times as there are number of elements in the sequence.

**Example-1:**

```
for i range(1,5):
   print i
print "END"
```

```
1
END
2
END
3
END
```

**Output-1:**

**Example-2:**

```
for i range(1,5):
      pri
nt i print
"END"
```

```
1
2
3
END
```

**Output-2**

**Break:**

It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C.

The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both while and for loops.

If you are using nested loops, the break statement stops the execution of the innermost loop and start executing the next line of code after the block.

The syntax of the break loop is given below:

```
break
```

**Example 1:**

```
for letter in 'Python':
  if letter == 'h':
     break
  print 'Current Letter:', letter
```

**Output:**

```
Current Letter: P
Current Letter: y
Current Letter: t
```

**Continue:**

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

The syntax of the continue loop is given below:

```
continue
```

**Example - 1:**

```
for i in range(1,11):
  if i==5:
    continue
  print i
```

**Output:**

```
1
2
3
4
5
6
7
8
9
10
```

**Pass:**

The pass statement is a null operation since nothing happens when it is executed. It is used in the cases where a statement is syntactically needed but we don't want to use any executable statement at its place.

For example, it can be used while overriding a parent class method in the subclass but don't want to give its specific implementation in the subclass.

Pass is also used where the code will be written somewhere but not yet written in the program file.

The syntax of the pass statement is given below.

```
pass
```

**Example – 1:**

```
for letter in 'Python':
    if letter == 'h':
       pass
       print ' This is pass block'
    print 'Current Letter :', letter
```

**Output:**

```
Current Letter : P
Current Letter : y
Current Letter : t
```

```
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
```

**Write a program to display factorial of a number**

**Program:**

```
n=input("Enter the number: ")
f=1
while n>0:
    f=f*n
    n=n-1
print "Factorial is",f


n=input("Enter the number: ")
f=1
for i in range(1,n+1):
   f=f*i
print "Factorial is",f
```

**Output:**

```
Enter the number: 5
Factorial is 120
```

**Write a program for print given number is prime number or not**

**Program:**

```
n=input("Enter the n value")
count=0
for i in range(2,n):
   if n%i==0:
      count=count+1
      break
if count==0:
  print "Prime Number"
else:
  print "Not Prime Number"
```

**Output:**

```
Enter n value: 17
Prime Number
```

**Write a program print Fibonacci series and sum the even numbers. Fibonacci series is 1,2,3,5,8,13,21,34,55**

**Program:**

```
n=input("Enter n value ")
  f0=1
f1=2
sum=f1
print f0,f1,
for i in range(1,n-1):
  f2=f0+f1
  print f2,
  f0=f1
  f1=f2
  if f2%2==0:
     sum+=f2
     print "\nThe sum of even Fibonacci numbers is", sum
```

**Output:**

```
Enter n value 10
1 2 3 5 8 13 21 34 55 89
The sum of even fibonacci numbers is 44
```

**Write a program to print given number is Armstrong or not.**

**Program:**

```
n=input("Enter the number: ")
sum=0
t=n
while n>0:
    r=n%10
   sum+=r*r*r
   n=n/10
 if sum==t:
   print "ARMSTRONG"
else:
    print "NOT ARMSTRONG"
```

**Output:**

```
Enter the number:
```

153

ARMSTRONG

**Write a program to take input string from the user and print that string after removing ovals.**

**Program:**

```
st=input("Enter the string:")
st2=""
for i in st:
    if i not in "aeiouAEIOU":
        st2=st2+i
    print st2
```

**Output:**

```
Enter the string:"Welcome to
you"
Wlcm t y
```

**Write a program that takes input string user and display that string if string contains at least one Uppercase character, one Lowercase character and one digit.**

**Program:**

```
pwd=input("Enter the password:")
u=False
l=Fal
se
d=Fa
lse
for i in
   range(0,len(pwd)): if
   pwd[i].isupper():
      u=True
   elif
      pwd[i].islower(
      ): l=True
   elif
      pwd[i].isdigit(
      ): d=True
if u==True and l==True and d==True:
   print pwd.center(20,"*")
else:
   print "Invalid Password"
```

**Output-1:**

```
Enter the password:"Mothi556"
******Mothi556******
```

**Output-2:**

Enter the password:"mothilal"
Invalid Password