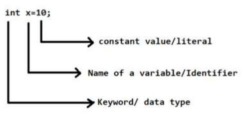
Literals:

Any constant value which can be assigned to a variable is called literal.

```
int x=10;
```



Integral Literals::

-> For the Integral data type[byte, short, int and long]we specify literals in the following ways.

1. Decimal literals(base-10):
   ->Allowed digits are 0-9.

   Example: int x=10;

2. Octal literals(base-8):
   ->Allowed digits are 0-7.

   Example: int x=010;

3. Hexadecimal literals(base-16):
   ->Allowed digits are 0-9,A(a),B(b),C(c),D(d),E(e),F(f)
   ->Literls should start with 0x or 0X.

   Example: int x=0x10;
   int x=0X10;

8
-
b
s
c
i
l
f
d

Q>Which of the folowing declartions are valid >

```
A. int x=0777;
B. int x=0786;
C. int x=0xFACE;
D. int x=0xbeef;
E. int x=0xBeer;
F. int x=0Xabb2cd;
```

Ans.

```
A. int x=0777;      [Valid]
B. int x=0786;      [Invalid]
C. int x=0xFACE;    [Valid]
D. int x=0xbeef;    [Valid]
E. int x=0xBeer;    [Invalid]
F. int x=0Xabb2cd;  [Valid]
```

-> Developers having the choice to specify either decimal or octal or hexadecimal.
But while printing JVM will always going to print decimal only.

Ex:

```
class Test{
  public static void main(String[] args){
       int i=10;
       int j=010;
       int k=0x10;
       System.out.println(i+"---"+j+"---"+k);
  }
}
```

octal to decimal conversiton

010[octal]-----> 1*(8^1)+0*(8^0)=8

Hexadecimal to decimal conversiton

0x10[Hexadecimal]-----> 1*(16^1)+0*(16^0)=16

O.P::10---8---16

->By default every integral literal is of int type but we can specify explicitly as long type by suffixing with lower-case "l" or upper case "L".

Ex:

```
int x=10;   [Valid]
long l=10l; [Valid]
long l=10;  [Valid]
int i=10l;  [Invalid]
```

->There is no direct way to specify byte and short literal explicitly.
But whenever we are assigning literal to byte variable and its value is within the range of byte, then compiler automatically treats it as byte literal.
Similarly for short literal.

Ex:

```
byte b=127;    [Valid]
short s=32767; [Valid]
```

Floating Point Literals:

-> Floating point literal is by default double type , but we can specify explicitly as float type by suffixing with 'f' or 'F'.

Example:

```
float f=123.45f;
double d=123.456;

float f=123.456;
CE:incompatible types: possible lossy conversion from double to float
              float f=123.456;
```

-> We can specify floating point literal as double type by suffixing with d or D.

Example:

```
double d=123.456d;
double d1=123.456D;
```

-> We can specify floating point literal only in decimal form and we cant specify in octal,hexadecimal and binary forms.

Example:

```
double d1=123.456;[Valid]
double d2=0123.45;[Valid][It is treated as decimal value but not as octal.]
double d3=0x123.45;[Invalid]

CE:malformed floating point literal
```

Q.Which of the following floating point declartions are valid ?

```
float f = 123.456;
float f = 123.456D;
double d = 0x123.456;
double d = 0xFace;
double d = 0xBeef;
```

Ans.

```
float f = 123.456;    [Invalid]
float f = 123.456D;   [Invalid]
double d = 0x123.456; [Invalid]
double d = 0xFace;    [Valid]
double d = 0xBeef;    [Valid]
```

->We can assign integral literal directly to the floating point data type and that integral literal can be specified in decimal, octal, and hexa-decimal form also.

Example:

```
double d=0xFace;
System.out.println(d); //64206

float f=100f;
system.out.println(f);//100
```

->We can't assign floating point literal directly to the integral types.

Example:

```
int x=10.0;
CE:incompatible types: possible lossy conversion from double to int
```

->We can specify floating point literal even in exponential form also.

Example:

```
double d=10e2;//[10*(10^2)]
float f=10e2f;
float f=10e2;
incompatible types: possible lossy conversion from double to float
```

Boolean literals:

->The only allowed values for boolean data types are true or false where case is important.i.e lower case only.

Example :

```
boolean b=true;      [Valid]
booelan b=0;         [Invalid] CE: incompatible types: int cannot be converted to boolean
boolean b=True;      [Invalid] CE: cannot find symbol
boolean b="true";    [Invlaid] CE: incompatible types: String cannot be converted to boolean
```

Char literals:

->A char literal can be represented as single character within single quotes.

Example:

```
char ch='a';
char ch=a;     [Invalaid] CE: cannot find symbol
char ch="a";   [Invalid]  CE: incompatible types: String cannot be converted to char
char ch='ab';  [Invalid]  CE: unclosed character literal
                              not a statement
```

->We can specify a char literal as integral literal which represents Unicode of that character.
We can specify that integral literal either in decimal or octal or hexadecimal or in binary form but allowed values ranage is 0 to 65,535.

Example:

```
char ch=97;      [Valid]
char ch='a';     [Valid]
char ch=0xFace;  [Valid]
cahr ch=65536;
incompatible types: possible lossy conversion from int to char
```

-> We can represent a char literal by Unicode representation which is nothing but '\uxxxx'[4 digit hexa decinal number.]

```
char ch='\u0061';   [Valid] O.P. a
char ch=\u0062;    [Invalid] cannot find symbol
char ch='\ubeef';  [Valid]
char ch='\ubeer';  [Invalid]    illegal unicode escape
char ch='\iface];  [Invalid]    illegal escape character
                                unclosed character literal
```

-> Every escape character in Java acts as a char literal.

Example:

```
char ch='\n';
char ch='\t';

char ch='\l';
```

Escape Character	Description
\n	New Line
\t	Horizontal Tab
\r	Carriage return
\b	Back Space
\f	Form feed
\'	Single Quote
\"	Double Quote
\\	Back Slash

Q> Which of the following char declarations are valid ?

```
char ch=a;
char ch='ab';
char ch=65536;
char ch=\uface;
char ch='/n';
```

Ans.

```
char ch=a;       [Invalid]  CE: cannot find symbol
char ch='ab';    [Invalid]  CE: unclosed character literal
char ch=65536;   [Invalid]  CE: incompatible types: possible lossy conversion from int to char
char ch=\uface;  [Invalid]  CE:illegal character
char ch='/n';    [Invalid]  CE:unclosed character literal
                               not a statement
```

String literals:

Any sequence of characters within double quotes is treated as String literal.

Example:

```
        String s="Java";
```

Note::null is the default value for any object reference.We cannot use null for primitive types.

1.7 Version enhancements with respect to Literals.

The following are tow two enhancements.

->Binary Literals

->Usage of '_' in numeric literals.

Binary Literals:

For the integral data type until 1.6V we can specify literal values in the following ways.
1. Decimal
2. Octal
3. Hexadecimal

But from 1.7V onwards we can specify literal vaue in Binary form also.
The allowed digits are 0 and 1.
Literal value should be prefixed with 0b or 0B.

Example::

```
class Test
{
     public static void main(String[] args)
     {
          int x=0b1011;
          System.out.println(x);
     }
}
```

O.P:11

Usage of _(UnderScore)symbol in numeric literals

From 1.7V onwards we use underscore(_) symbol in numeric literals.

```
double d1=1_23_456.7_8_9; [Valid]
double d2=123_456.7_8_9;  [Valid]
double d3=123456.789;     [Valid]
```

The main advantage of this approach is readability of the code will be improved.
At the time of compilation '_' symbol will be removed automatically, hence after compilation the above lines will become
double d=123456.789.

We can use more than one underscore symbol between the digits.

```
double d1=1_23__456.7_8_9;
```

We should use underscore symbol only between the literals i.e we can't use the underscore symbol at the beginning or end of the literal.

```
double d1=_1_23_456.7_8_9;   [Invalid]
double d2=1_23_456_.7_8_9;   [Invalid]
double d3=1_23_456.7_8_9_;   [Invalid]
```