



RV Educational Institutions[®]
RV College of Engineering[®]

Autonomous
Institution Affiliated
to Visvesvaraya
Technological
University, Belagavi

Approved by AICTE,
New Delhi

UNIT - II

CONTENT OUTLINE

Branching, While Loops, and Program Planning

1. Using the If statement
2. Using the else Clause
3. Using the elif clause
4. Creating while Loops
5. Avoiding Infinite Loops
6. Creating Intentional infinite Loops
7. Using Compound Conditions



Conditional Statements

What are Conditional Statements in Python?

Conditional Statement in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false.

Conditional statements are handled by IF statements in Python.

Python if Statement is used for decision-making operations. **It contains a body of code which runs only when the condition given in the if statement is true.** If the condition is false, then the optional else statement runs which contains some code for the else condition



1. Using the If statement

- Introducing the Password Program
- Examining the if Statement
- Creating Conditions
- Understanding Comparison Operators
- Using Indentation to Create Blocks
- Building Your Own if Statement

IF Statement

The general Python syntax for a simple if statement is
if condition :

Eg.

```
weight = 60
```

```
if weight > 50:
```

```
    print("There is a $25 charge for luggage that heavy.")
```

```
    print("Thank you for your business.")
```



Introducing the Password Program

Password

Demonstrates the if statement

```
print("Welcome to System Security Inc.")
```

```
print("-- where security is our middle name\n")
```

```
password = input("Enter your password: ")
```

```
    if password == "secret":
```

```
        print("Access Granted")
```

```
input("\n\nPress the enter key to exit.")
```



Examining the if Statement

The key to program Password is the if statement:

```
if password == "secret":
```

```
    print("Access Granted")
```

The if statement is pretty straightforward. You can probably figure out what's happening just by reading the code.



Creating Conditions

Creating the if Statement

In Python, there are three forms of the if...else statement.

1.if statement

2.if...else statement

3.if...elif...else statement



Understanding Comparison Operators in if Statement

Python supports the usual logical conditions from mathematics:

Equals: $a == b$

Not Equals: $a != b$

Less than: $a < b$

Less than or equal to: $a <= b$

Greater than: $a > b$

Greater than or equal to: $a >= b$



Using Indentation to Create Blocks

Using Indentation to Create Blocks

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important. Python uses indentation to indicate a block of code.

Using Indentation to Create Blocks

Python indentation is a way of telling the Python interpreter that a series of statements belong to a particular block of code.

In languages like C, C++, Java, we use curly braces { } to indicate the start and end of a block of code. In Python, we use space/tab as indentation to indicate the same to the compiler.

■ -> Indicates 1 Space Indentation

Statement 1
Statement 2
■ Statement 3
■ ■ Statement 4
■ Statement 5
■ Statement 6
Statement 7

How the interpreter visualises

Code Block 1 begins
Code Block 1 continues
Code Block 2 begins
Code Block 3
Code Block 2 continues
Code Block 2 continues
Code Block 1 continues

Here:

Statements 1, 2, 7 belong to code block 1 as they are at the same distance to the right.
Statements 3, 5, 6 belong to code block 2
Statement 4 belongs to code block 3

Execution happens in the same order.

all the statements with the same distance (space) to the right, belong to the same block

Using Indentation to Create Blocks

In C

```
#include <stdio.h>
int main()
{ //Beginning of Block 1
int a;
printf("Enter the easiest programming language: ");
scanf("%d", &a);
if (a == "python")
{ //Beginning of Code Block 2
printf("Yes! You are right");
} //End of Code Block 2
else
{ //Beginning of Code Block 3
printf("Nope! You are wrong");
} //End of Code Block 3
return 0;
} //End of Block 1
```

In Python

```
a = input("Enter the easiest programming language: ")

if a == 'python':
    print("Yes! You are right")
else:
    print("Nope! You are wrong")
```

Additional space before print is used to indicate a new block of code. Here, both the print are indented 2 spaces.

Building Your Own if Statement Python if statement

The syntax of if statement in

Python is:

if condition:

body of if statement

Condition is True

```
number = 10
if number > 0:
    # code
# code after if
```

Condition is False

```
number = -5
if number > 0:
    # code
# code after if
```

```
number = 10
```

```
# check if number is greater than 0
```

```
if number > 0:
```

```
    print('Number is positive.')
```

```
    print('The if statement is easy')
```



UNIT - II

CONTENT OUTLINE

1. Using the If statement
- 2. *Using the else Clause***
3. Using the elif clause
4. Creating while Loops
5. Avoiding Infinite Loops
6. Creating Intentional infinite Loops
7. Using Compound Conditions



2. *Using the **else** Clause*

*Using the **else** Clause*

Introducing the Granted or Denied Program

Examining the else Clause

if condition:

 # block of code if condition is True

else:

 # block of code if condition is False



Introducing the Password Program

Password

Demonstrates the if statement

```
print("Welcome to System Security Inc.")  
print("-- where security is our middle name\n")  
password = input("Enter your password: ")  
    if password == "secret":  
print("Access Granted")  
    else:  
print("Access Denied : Enter the valid password ")  
input("\n\nPress the enter key to exit.")
```

Python if else statement

The syntax of if-else statement in Python is:

if condition;

block of code if condition is True

else:

block of code if condition is False

Condition is True

```
number = 10
if number > 0:
    # code

else:
    # code

# code after if
```

Condition is False

```
number = -5
if number > 0:
    # code

else:
    # code

# code after if
```

```
number = 10
```

```
if number > 0:
```

```
    print('Positive number')
```

```
else:
```

```
    print('Negative number')
```

```
print('This statement is always executed')
```




UNIT - II

CONTENT OUTLINE

1. Using the If statement
2. Using the else Clause
- 3. *Using the elif clause***
4. Creating while Loops
5. Avoiding Infinite Loops
6. Creating Intentional infinite Loops
7. Using Compound Conditions

3. Using the elif Clause

Using the *elif* Clause

Elif

The elif keyword is python's way of saying "if the previous conditions were not true, then try this condition".

if (condition):

#Set of statement to execute if condition is true

elif (condition):

#Set of statements to be executed when if condition is false and elif condition is true

else:

#Set of statement to be executed when both if and elif conditions are false

Python *if..elif..else* statement

The syntax of *if...elif...else* construct
statement in Python is:

if condition1:

code block 1

elif condition2:

code block 2

else:

code block 3

1st Condition is True

```
let number = 5
if number > 0 :
    # code
elif number < 0 :
    # code
else :
    # code
# code after if
```

2nd Condition is True

```
let number = -5
if number > 0 :
    # code
elif number < 0 :
    # code
else :
    # code
# code after if
```

All Conditions are False

```
let number = 0
if number > 0 :
    # code
elif number < 0 :
    # code
else :
    # code
# code after if
```

Python *if..elif..else* statement

*The syntax of if...elif...else construct
statement in Python is:*

if condition1:

code block 1

elif condition2:

code block 2

else:

code block 3

```
number = int(input('enter the number'))
```

```
if number > 0:
```

```
    print("Positive number")
```

```
elif number == 0:
```

```
    print('Zero')
```

```
else:
```

```
    print('Negative number')
```

```
print('This statement is always executed')
```



Python *nested. If..* statement

Nested “if-else” statements mean that an “if” statement or “if-else” statement is present inside another if or if-else block

```
if(condition):  
    #Statements to execute if condition is true  
    if(condition):  
        #Statements to execute if condition is true  
    #end of nested if  
#end of if
```

The above syntax clearly says that the if block will contain another if block in it and so on. If block can contain ‘n’ number of if block inside it.



Python *nested. If..* statement

```
num = 5
if(num >0):
    print("number is positive")

if(num<10):
    print("number is less than 10")
```



RV Educational Institutions[®]
RV College of Engineering[®]

Autonomous
Institution Affiliated
to Visvesvaraya
Technological
University, Belagavi

Approved by AICTE,
New Delhi

Python *nested. If..* statement

```
num = 7
if (num != 0):
    if (num > 0):
        print("Number is greater than Zero")
```



Nested if-else Syntax:

if(condition):

#Statements to execute if condition is true

if(condition):

#Statements to execute if condition is true

else:

#Statements to execute if condition is false

else:

#Statements to execute if condition is false

included the “if-else” block inside an if block, you can also include an “if-else” block inside “else” block.

Nested if-else Syntax:

```
num = -7
if (num != 0):
    if (num > 0):
        print("Number is positive")
    else:
        print("Number is negative")
else:
    print("Number is Zero")
```

Included the “if-else” block inside an if block, you can also include an “if-else” block inside “else” block.



We have seen about the “elif” statements but what is this elif ladder?

As the name itself suggests a program that contains a ladder of “elif” statements or “elif” statements are structured in the form of a ladder.

This statement is used to test multiple expressions.

Syntax:

```
#Set of statement to execute if condition is true
elif (condition):
    #Set of statements to be executed when if condition is false and elif
    condition is true
elif (condition):
    #Set of statements to be executed when both if and first elif condition is false
    and second elif condition is true
elif (condition):
    #Set of statements to be executed when if, first elif and second elif
    conditions are false and third elif statement is true
else:
    #Set of statement to be executed when all if and elif conditions are false
```



elif Ladder:

```
my_marks = 89
if (my_marks < 35):
    print("Sorry!, You failed the exam")
elif(my_marks > 60 and my_marks < 90):
    print("Passed in First class")
else:
    print("Passed in First class with distinction")
```



RV Educational Institutions[®]
RV College of Engineering[®]

Autonomous
Institution Affiliated
to Visvesvaraya
Technological
University, Belagavi

Approved by AICTE,
New Delhi

UNIT - II

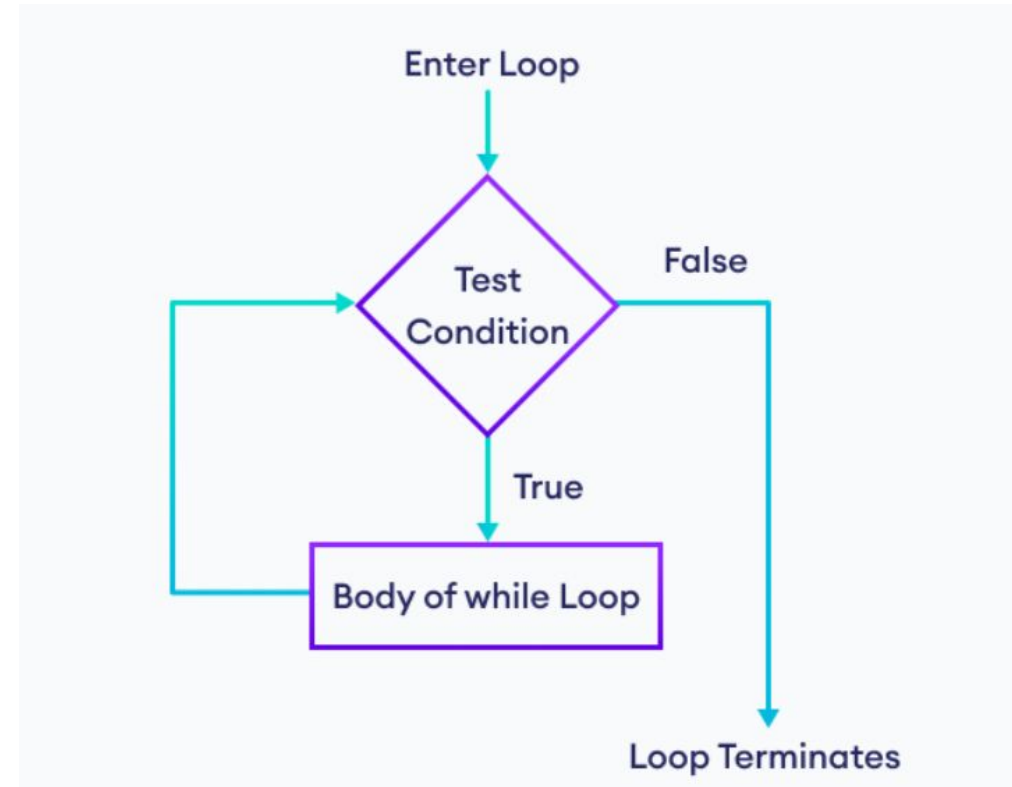
CONTENT OUTLINE

1. Using the If statement
2. Using the else Clause
3. Using the elif clause
- 4. *Creating while Loops***
5. Avoiding Infinite Loops
6. Creating Intentional infinite Loops
7. Using Compound Conditions

Python while Loop

Python while loop is used to run a specific code until a certain condition is met.

The syntax of while loop is:
while condition:
body of while loop





program to display numbers from 1 to 5

initialize the variable

i = 1

n = 5

while loop from i = 1 to 5

while i <= n:

print(i)

i = i + 1

Python while Loop

Variable Condition: $i \leq n$ Action

$i = 1$

$n = 5$ True 1 is printed. i is increased to 2.

$i = 2$

$n = 5$ True 2 is printed. i is increased to 3.

$i = 3$

$n = 5$ True 3 is printed. i is increased to 4.

$i = 4$

$n = 5$ True 4 is printed. i is increased to 5.

$i = 5$

$n = 5$ True 5 is printed. i is increased to 6.

$i = 6$

$n = 5$ False The loop is terminated.



4. Creating while Loops

- Creating while Loops
- Introducing the Three-Year-Old Simulator Program
- Examining the while Loop
- Initializing the Sentry Variable
- Checking the Sentry Variable
- Updating the Sentry Variable

Introducing the Three-Year-Old Simulator Program

Three Year-Old Simulator

Demonstrates the while loop

```
print("\t Welcome to the 'Three-Year-Old Simulator'\n")
```

```
print("This program simulates a conversation with a three-year-old child.")
```

```
print("Try to stop the madness.\n")
```

```
response = ""
```

```
while response != "Because":
```

```
    response = input("But Why?\n")
```

```
print("Oh. Okay.")
```

as long as response is not the word "Because" Loop will keep executing and say "But why?"

Initializing the Sentry Variable

- Often, while loops are controlled by a sentry variable, a variable used in the condition and compared to some other value or values.
- Like a human sentry, you can think of your sentry variable as a **guard**, helping form a barrier around the while loop's block.
- In the Three-YearOld Simulator program, the sentry variable is **response**.
- It's used in the condition and is compared to the string "Because." before the block is executed each time.
- It's important to initialize your sentry variable. Most of the time, sentry variables are initialized right before the loop itself.
- That's what I did with: `response = ""`
- If the sentry variable doesn't have a value when the condition is evaluated, your program will generate an error



Checking the Sentry Variable

Make sure that it's possible for the while condition to evaluate to True at some point;

Otherwise, the block will never run.

Take, for example, one minor change to the loop you've been working with

```
response = "Because."
```

```
while response != "Because":
```

```
response = input("Why?\n")
```

Since response is equal to "Because." right before the loop, the block will never run.

Infinite while Loop in Python

If the condition of a loop is always True, the loop runs for infinite times (until the memory is full).

For example,

while True:

```
num = int(input("Enter an integer: "))  
print("The double of",num,"is",2 * num)
```



RV Educational Institutions[®]
RV College of Engineering[®]

Autonomous
Institution Affiliated
to Visvesvaraya
Technological
University, Belagavi

Approved by AICTE,
New Delhi

UNIT - II

CONTENT OUTLINE

1. Using the If statement
2. Using the else Clause
3. Using the elif clause
4. Creating while Loops
- 5. *Avoiding Infinite Loops***
6. Creating Intentional infinite Loops
7. Using Compound Conditions



RV Educational Institutions[®]
RV College of Engineering[®]

Autonomous
Institution Affiliated
to Visvesvaraya
Technological
University, Belagavi

Approved by AICTE,
New Delhi

5. Avoiding Infinite Loops

- Introducing the Losing Battle Program
- Tracing the Program
- Creating Conditions That Can Become False



Introducing the Losing Battle Program

Losing Battle # Demonstrates the dreaded infinite loop

```
print("Your hero unsheathes his sword for the last fight of his life.\n")
```

```
health = 10
```

```
trolls = 0
```

```
damage = 3
```

```
while health != 0:
```

```
    trolls += 1
```

```
    health -= damage
```

```
print("Your hero swings and defeats an evil troll, " \ "but takes", damage, "damage points.\n")
```

```
print("Your hero fought valiantly and defeated", trolls, "trolls.")
```



Tracing the Program

Demonstrates the dreaded infinite loop

health	trolls	damage	health != 0
--------	--------	--------	-------------

10	0	3	True
----	---	---	------

7	1	3	True
---	---	---	------

4	2	3	True
---	---	---	------

1	3	3	True
---	---	---	------

-2	4	3	True
----	---	---	------

-5	5	3	True
----	---	---	------

-7	6	3	True
----	---	---	------

Creating Conditions That Can Become False

The line with the condition just needs to become

while health > 0:

Now, if health becomes 0 or negative, the condition evaluates to False and the loop ends. To be sure, you can trace the program using this new condition:

health	trolls	damage	health != 0
10	0	3	True
7	1	3	True
4	2	3	True
1	3	3	True
-2	4	3	False



RV Educational Institutions[®]
RV College of Engineering[®]

Autonomous
Institution Affiliated
to Visvesvaraya
Technological
University, Belagavi

Approved by AICTE,
New Delhi

Treating Values as Conditions

- Introducing the Maitre D' Program
- Interpreting Any Value as True or False



RV Educational Institutions[®]
RV College of Engineering[®]

Autonomous
Institution Affiliated
to Visvesvaraya
Technological
University, Belagavi

Approved by AICTE,
New Delhi

Introducing the Maitre D' Program

Evaluating condition

```
print("Welcome to the MTR")
print("It seems we are quite full this evening.\n")

money = int(input("How many Rupees you need to pay to reserve table'? "))

if money:
    print("Ah, I am reminded of a table. Right this way.")
else:
    print("Please, sit. It may be a while.")

input("\n\nPress the enter key to exit.")
```



Interpreting Any Value as True or False

The new concept is demonstrated in the line:

if money:

Notice that money is not compared to any other value.

money is the condition.

When it comes to evaluating a number as a condition, **0 is False** and everything **else is True**.

So, the above line is equivalent to

if money != 0:

The first version is simpler, more elegant, and more intuitive. It reads more naturally and could be translated to “if there is money.”



6. Creating Intentional Infinite Loops

- Introducing the Finicky Counter Program
- Using the break Statement to Exit a Loop
- Using the continue Statement to Jump Back to the Top of a Loop
- Understanding When to Use break and continue

Break and continue


- **Python break Statement**

- *The break statement is used to terminate the loop immediately when it is encountered.*

- *The syntax of the break statement is:*

- **break**
- **for i in range(5):**
 - **if i == 3:**
 - **break**
 - **print(i)**

```
while condition:  
    # code  
    if condition:  
        break  
    # code
```

A diagram illustrating the effect of a 'break' statement. A teal line starts from the 'break' statement, moves left, then down, and finally right as an arrow pointing to the end of the loop structure, indicating that the loop is terminated immediately.



RV Educational Institutions[®]
RV College of Engineering[®]

Autonomous
Institution Affiliated
to Visvesvaraya
Technological
University, Belagavi

Approved by AICTE,
New Delhi

Break and continue

i = 1

while (i<=10):

print('6 * ',(i), '=',6 * i)

if i >= 5:

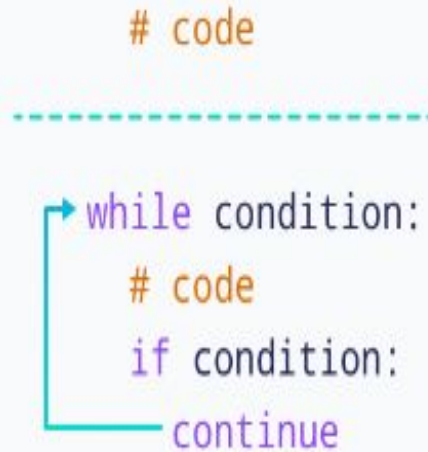
break

i = i + 1

Continue Statement

In Python, we can also skip the current iteration of the while loop using the continue statement

. For example

A diagram illustrating the execution of a while loop. It shows a loop structure with a 'while condition:' block. Inside the loop, there is a '# code' line followed by an 'if condition:' block. The 'if condition:' block contains a 'continue' statement. A dashed line represents the loop's path, and a blue arrow indicates that the loop jumps back to the start of the 'while condition:' block after the 'continue' statement is executed, skipping the rest of the loop body.

```
# code
-----
while condition:
    # code
    if condition:
        continue
```

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

It will skip printing three

Continue Statement

In Python, we can also skip the current iteration of the while loop using the continue statement

. For example

program to print odd numbers from 1 to 10

num = 0

while num < 10:

num += 1

if (num % 2) == 0:

continue

print(num)

Introducing the Finicky Counter Program

The Finicky Counter program counts from 1 to 10 using an intentional infinite loop. It's finicky because it doesn't like the number 5 and skips it. # Finicky Counter # Demonstrates the break and continue statements

```
count = 0
```

```
while True:
```

```
    count += 1
```

```
    if count > 10:
```

```
        break
```

```
    if count == 5: (skips if 5)
```

```
        continue
```

```
    print(count)
```

```
    input("\n\nPress the enter key to exit.")
```

Using the break Statement to Exit a Loop

I set up the loop with:

while True:

This technically means that the loop will continue forever, unless there is an exit condition in the loop body. Luckily, I put one in:

end loop if count greater than 10

if count > 10:

break

Since count is increased by 1 each time the loop body begins, it will eventually reach 11.

When it does, the break statement, which here means “break out of the loop,” is executed and the loop ends



Using the continue Statement to Jump Back to the Top of a Loop

Just before count is printed,

I included the lines:

```
# skip 5
```

```
    if count == 5:
```

```
        continue
```

The continue statement means “jump back to the top of the loop.”



Understanding When to Use break and continue

- You can use break and continue in any loop you create.
- They aren't just restricted for use in intentional infinite loops.
- But they should be used sparingly.
- Both break and continue make Branching, while Loops, and Program Planning



RV Educational Institutions[®]
RV College of Engineering[®]

Autonomous
Institution Affiliated
to Visvesvaraya
Technological
University, Belagavi

Approved by AICTE,
New Delhi

7. Using Compound Conditions and Operators

[Arithmetic Operators](#)

[Assignment Operators](#)

[Comparison Operators](#)

[Identity Operators](#)

[Membership Operators](#)

[Bitwise Operators](#)



Arithmetic Operator

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Assignment Operator

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
 =	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operator

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y



Identity Operator

Operator	Description	Example
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y



Identity Operator

```
a = 20  
b = 20
```

```
if ( a is b ):  
    print "a and b have same identity"  
else:  
    print " a and b do not have same identity"
```

```
a = 20  
b = 40
```

```
if ( a is not b ):  
    print "a and b do not have same identity"  
else:  
    print "a and b have same identity"
```



Membership Operator

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y



RV Educational Institutions[®]
RV College of Engineering[®]

Autonomous
Institution Affiliated
to Visvesvaraya
Technological
University, Belagavi

Approved by AICTE,
New Delhi

Membership Operator

a = 1

b = 20

list = [1, 2, 3, 4, 5];

if (**a in list**):

 print ("a is available in the given list")

else:

 print ("a is not available in the given list")

Bitwise Operator

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off



Bitwise Operator

AND

$a = 10 = 1010$ (Binary)

$b = 4 = 0100$ (Binary)

$a \& b = 1010$

$\&$

0100

$= 0000$

$= 0$ (Decimal)

OR

$a = 10 = 1010$ (Binary)

$b = 4 = 0100$ (Binary)

$a \mid b = 1010$

\mid

0100

$= 1110$

$= 14$ (Decimal)

Bitwise not operator: Returns one's complement of the number.

Eg – $a = 10$

$b = \sim(a)$

$b = -(a+1) = -11$



RV Educational Institutions[®]
RV College of Engineering[®]

Autonomous
Institution Affiliated
to Visvesvaraya
Technological
University, Belagavi

Approved by AICTE,
New Delhi

Bitwise Operator

XOR

$a = 10 = 1010$ (Binary)

$b = 4 = 0100$ (Binary)

$a \wedge b = 1010$

\wedge

0100

= 1110

= 14 (Decimal)



Bitwise Operator

Bitwise right shift: Shifts the bits of the number to the right and fills 0 on voids left(fills 1 in the case of a negative number) as a result. Similar effect as of dividing the number with some power of two.

Example:

Example 1:

$a = 10 = 0000\ 1010$ (Binary)

$a \gg 1 = 0000\ 0101 = 5$

Example 2:

$a = -10 = 1111\ 0110$ (Binary)

$a \gg 1 = 1111\ 1011 = -5$



Bitwise Operator

Bitwise left shift: Shifts the bits of the number to the left and fills 0 on voids right as a result. Similar effect as of multiplying the number with some power of two.

Example:

Example 1:

$a = 5 = 0000\ 0101$ (Binary)

$a \ll 1 = 0000\ 1010 = 10$

$a \ll 2 = 0001\ 0100 = 20$

Example 2:

$b = -10 = 1111\ 0110$ (Binary)

$b \ll 1 = 1110\ 1100 = -20$

$b \ll 2 = 1101\ 1000 = -40$