Introduction to Python (Day 1)

Presenter: Dr. Sourav Saha

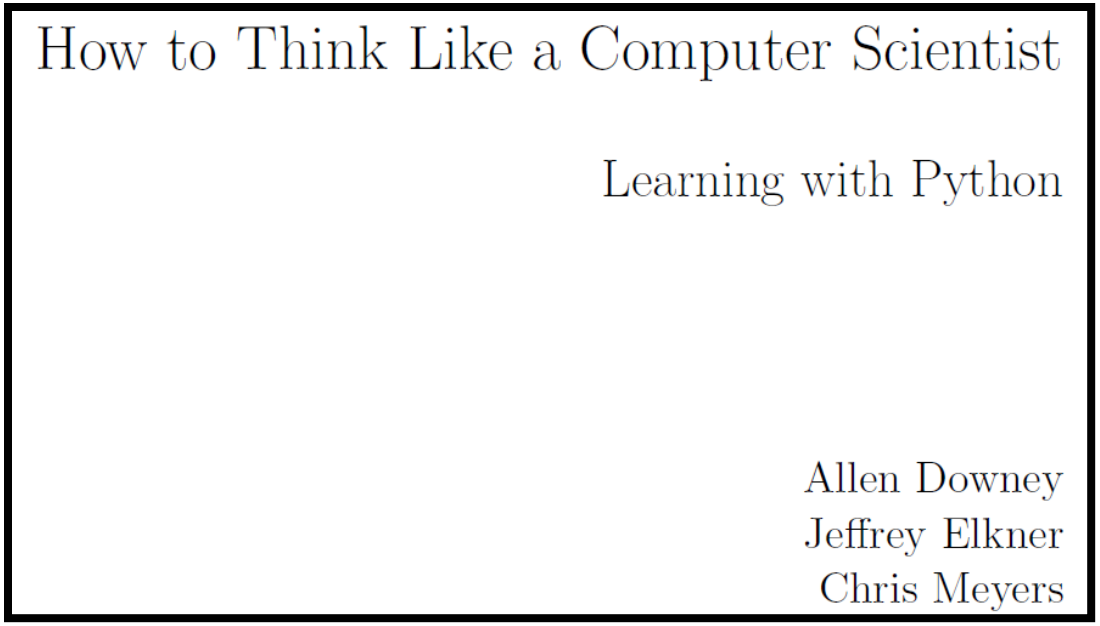
This course is designed for students who…

… **have never written a Python program before**

… **like solving problems and building solutions using regular sized or very large data sets**

… **are curious about how Machine Learning programs can be written**

Acknowledgements ...



How to Think Like a Computer Scientist

Learning with Python

Allen Downey
Jeffrey Elkner
Chris Meyers

What is programming language?

program: A set of instructions to be carried out by a computer

program execution: The act of carrying out the instructions contained in a program

programming language: A set of rules used to describe computations in a format that is readable by humans

TYPES OF PROGRAMMING LANGUAGE

procedural languages: programs are a series of commands

- **Pascal** (1970): designed for education
- **C** (1972): low-level operating systems and device drivers

functional programming: functions map inputs to outputs

- **Lisp** (1958) / **Scheme** (1975), **ML** (1973), **Haskell** (1990)

object-oriented languages: programs use interacting "objects"

- **Smalltalk** (1980): first major object-oriented language
- **C++** (1985): "object-oriented" improvements to C
  - successful in industry; used to build major OSes such as Windows
- **Python** (1991):
  - The language taught in this course

Why Python?

-- Expressive language

- ✓ expresses complex ideas in a simple way
- ✓ well-designed

-- Object-oriented

-- Pre-written software

-- Widely used especially for solving problems using machine learning tools and techniques

Structure of programming language

- Writing output
- Storing values
- Operators and Expressions
- Input Operation
- Decision Control
    - If-else
    - Loop
- Special Data Structures: List, Tuple, Set, Dictionary, Numpy-Array
- Function
- File Handling

Getting started

First **install Python.** Then **install an IDE** (or integrated development environment) for Python – **IDLE** being the default IDE which has been bundled with the default implementation of Python since 1.5.2b1. Another option is to install **Anaconda*,** which is a scientific Python distribution. The default IDE bundled with Anaconda is **Spyder**.

After the IDE is installed, **install basic libraries** of Python. To start with, install the library "os" for using operating system dependent functions

Command: pip install

Syntax: pip install 'SomePackage'

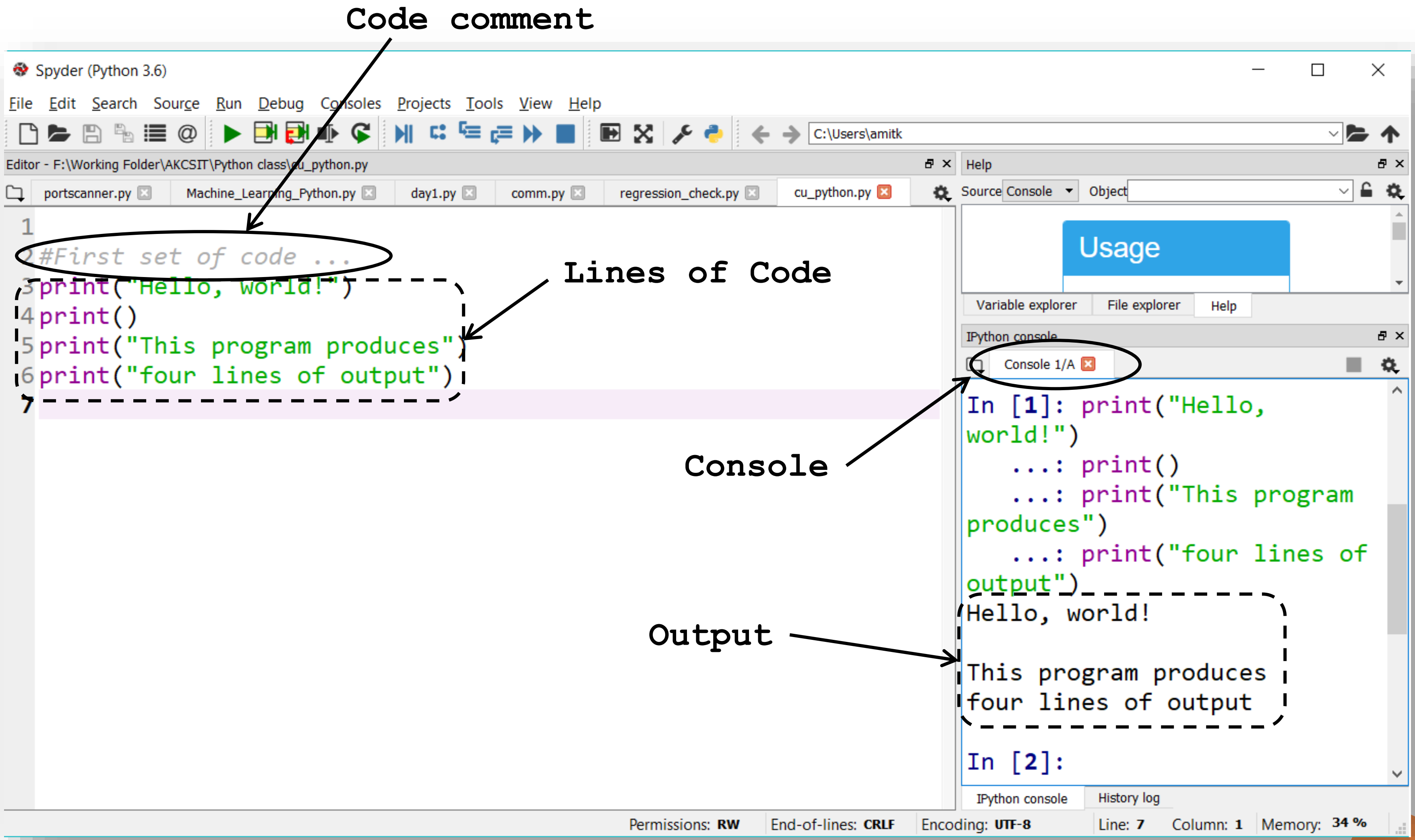
After installing, libraries need to be loaded by invoking the command:

Command: import <<library-name>>

Syntax: import os

*** There is a minimal Anaconda Python without all the packages, called Miniconda. After installing miniconda, you can use conda and install only those scientific packages that you wish and avoid a bloated installation.**

A Python program



```
#First set of code ...
print("Hello, world!")
print()
print("This program produces")
print("four lines of output")
```

```
In [1]: print("Hello,
world!")
   ...: print()
   ...: print("This program
produces")
   ...: print("four lines of
output")
Hello, world!

This program produces
four lines of output

In [2]:
```

Command: print( )

Used to print a line of output on the console

Three ways to use print :

- `print("…`*`string`*`…")`

  Prints the given string as output

- `print(var)`

  Prints a variable value as output

- `print()`

  Prints a blank line of output

STRING

```
string: A sequence of characters
-- Starts and ends with a " quote " character or a '
quote ' character
-- The quotes do not appear in the output when printed

    Examples:

    "hello"
    "This is a string.  It's very long!"
    'Here is "another" with quotes in "
    '''This is a paragraph. It is
    made up of multiple lines and sentences. '''
```

String (contd.)

Syntax Rules:

```
    Strings surrounded by " " or ' ' may not span
    multiple lines

    "This is not
    a legal String."


    Strings surrounded by " " may not contain a "
    character.

    "This is not a "legal" String either."


    Strings surrounded by ' ' may not contain a '
    character.

    'This is not a 'legal' String either.’
```

So, how to handle these problems?

ESCAPE SEQUENCES

A sequence of characters used to represent certain special characters in a string.

```
    \t   tab character
    \n   new line character
    \"   quotation mark character
    \\   backslash character
```

```
“Now, this is \n a legal String."

"This is also a \"legal\" String."

'This is another \'legal\' String.’
```

Operations on Strings

String concatenation: + operator

```
print("Hello," + " world!")
```

Use + str(value) to print a string and a variable's value on one line.

```
>>> marks = (95.1 + 71.9 + 82.6) / 3.0
>>> print("Your marks is " + str(marks))
>>> print("Your marks is ", marks)

apples = 13 + 17 + 5
Print("There are " , apples, " apples in the
basket.")
```

Output:

```
Your grade was 83.2
There are 35 apples in the basket.
```

LET'S TAKE A PAUSE: QUIZ TIME …

>>> What print statements will generate this output?

```
    A "quoted" String is
    'much' better if you learn
    the rules of "escape sequences."

    Also, "" represents an empty String.
    Don't forget: use \" instead of " !
    '' is not same as "
```

>>> Write a program to print the following figures.

```
      ______
     /      \
    /        \
    \        /
     \______/

    \        /
     \______/
    +--------+


      ______
     /      \
    /        \
    |  STOP  |
    \        /
     \______/


      ______
     /      \
    /        \
    +--------+
```

Let's look at the solutions

```
>>> print("A \"quoted\" String is")
>>> print("'much' better if you learn")
>>> print("the rules of \"escape sequences.\"")
>>> print()
>>> print("Also, \"\" represents an empty String.")
>>> print("Don't forget: use \\\" instead of \" !")
>>> print("'' is not the same as \"")
```

OR

```
>>> print("A \"quoted\" String is \n 'much' better
if you learn \n the rules of \"escape sequences.\"")
>>> print("\n Also, \"\" represents an empty String.
\n Don't forget: use \\\" instead of \" !\n '' is
not the same as \"")
```

Let's look at the solutions (contd.)

```
>>> print("  ______")
>>> print(" /      \\")
>>> print("/        \\")
>>> print("\\        /")
>>> print(" \\______/")
>>> print()
>>> print("\\        /")
>>> print(" \\______/")
>>> print("+--------+")
>>> print()
>>> print("  ______")
>>> print(" /      \\")
>>> print("/        \\")
>>> print("|  STOP  |")
>>> print("\\        /")
>>> print(" \\______/")
>>> print()
>>> print("  ______")
>>> print(" /      \\")
>>> print("/        \\")
>>> print("+--------+")
```

Comments in Python

- `# First comment`
- `print ("Hello, Python! ") # second comment`

Assigning Values to Variables

- `counter = 100 # An integer assignment`
- `miles = 1000.0 # A floating point`
- `name = "John" # A string`

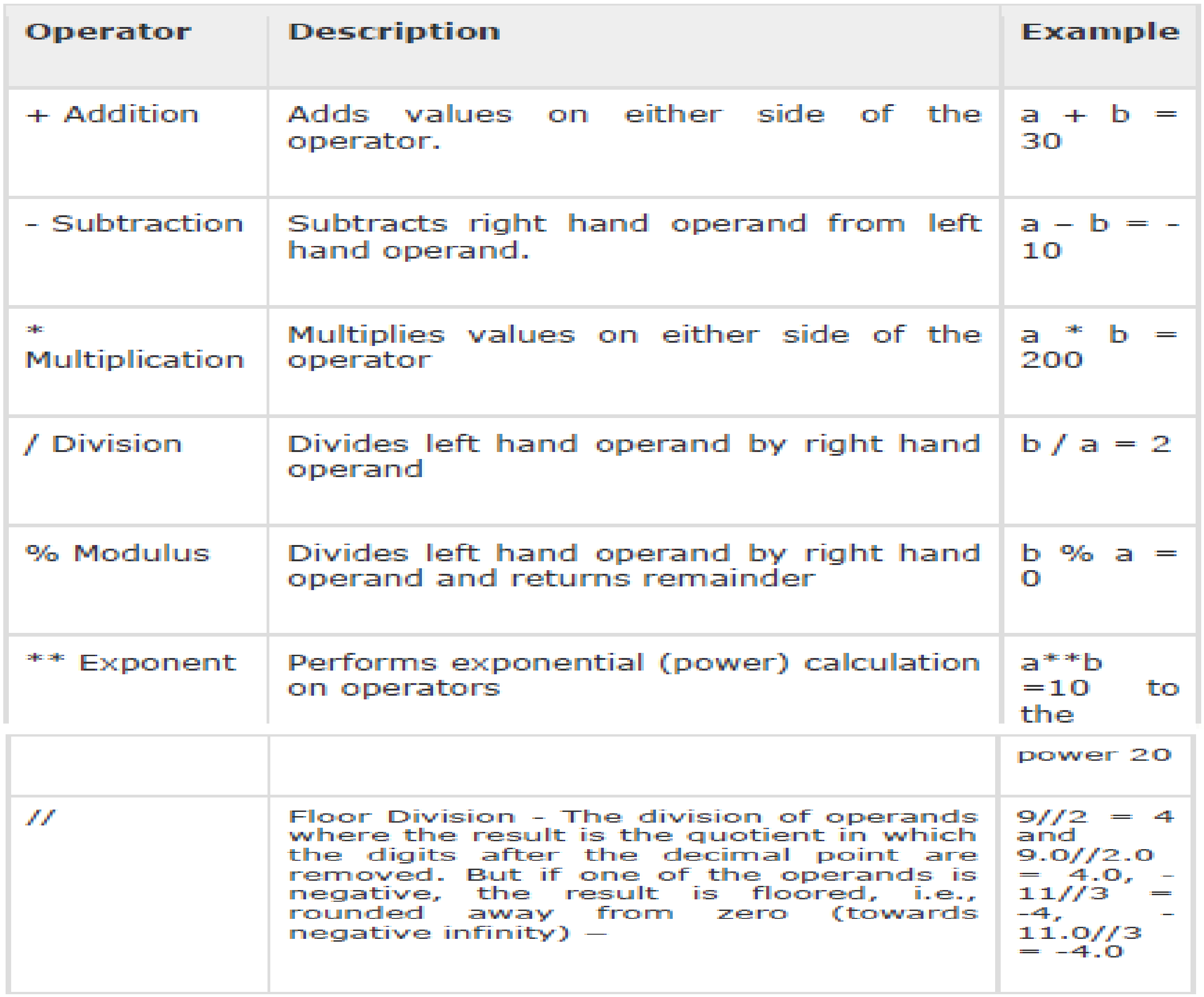
Multiple Assignment

- `a = b = c = 1`
- `a, b, c = 1, 2, "john"`

Types of Operator

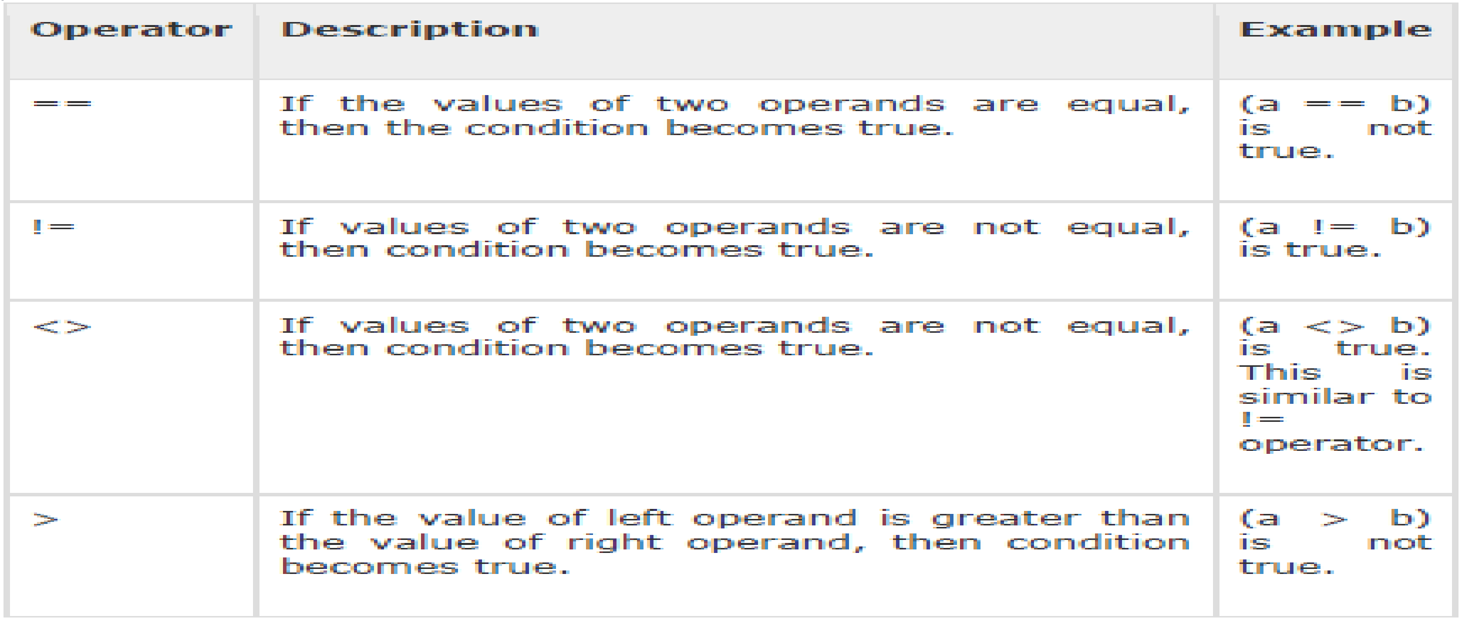
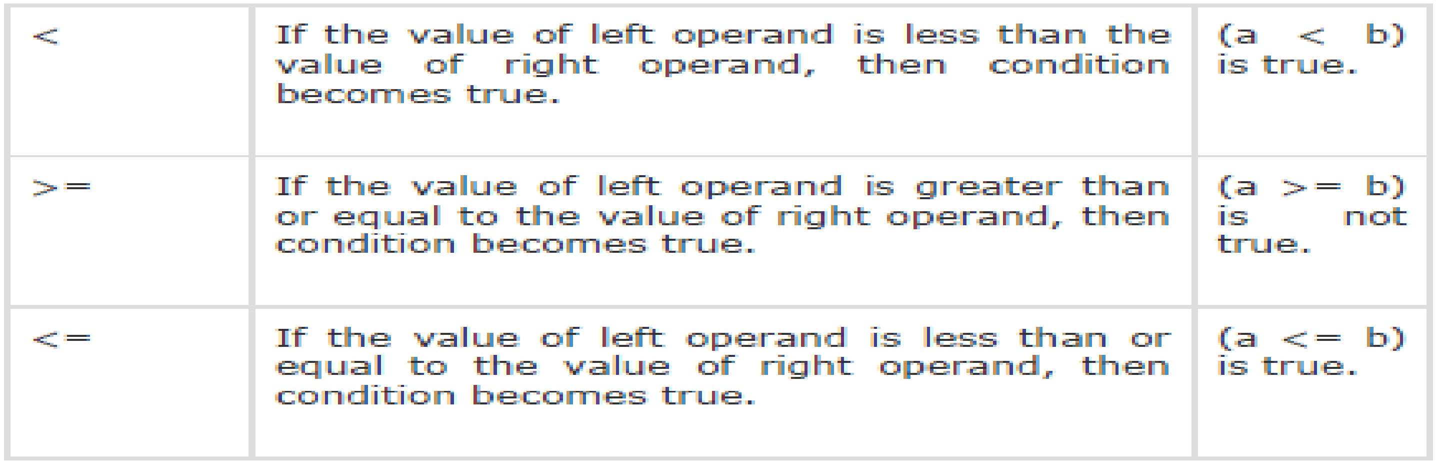
- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

OPERATORS



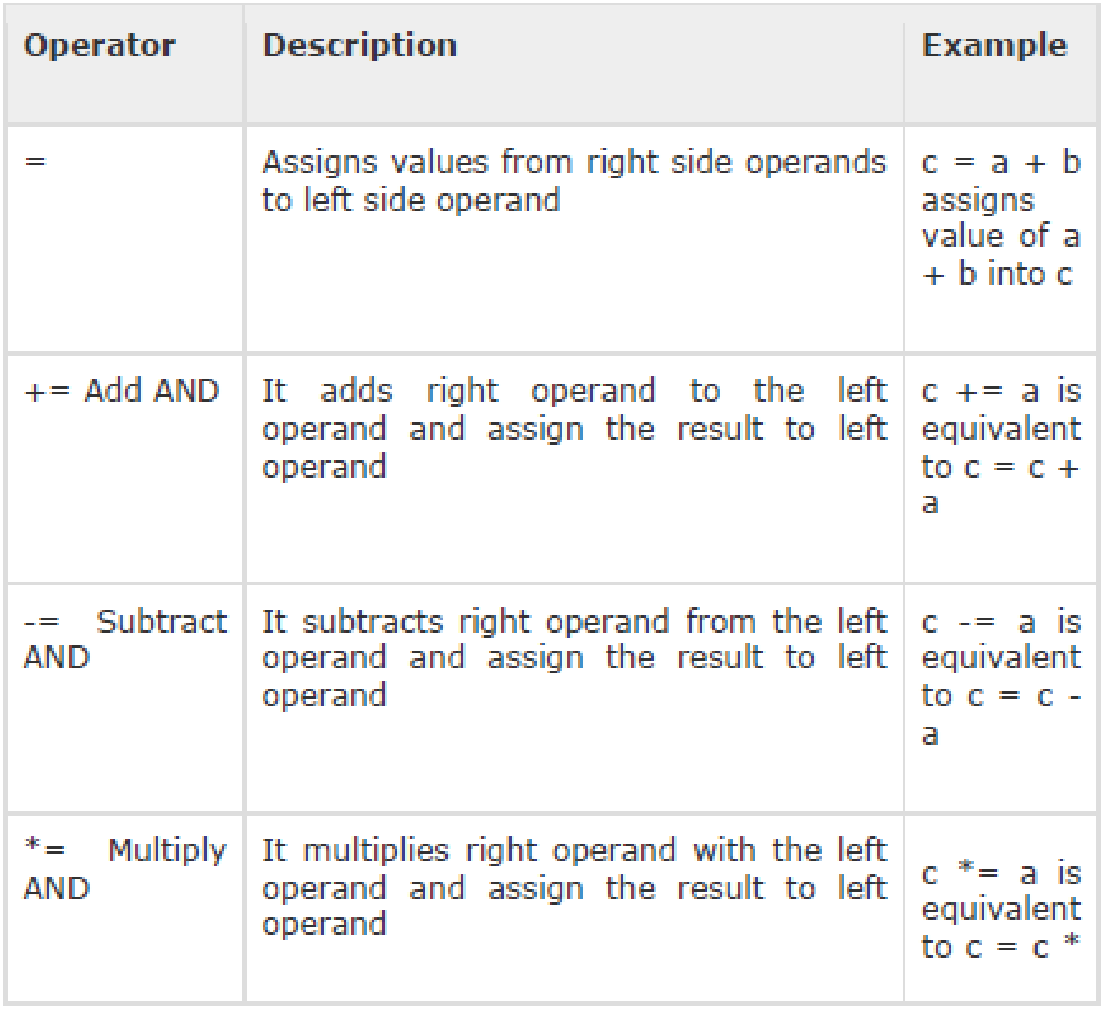
Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand 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. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) −	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

COMPARISON OPERATORS

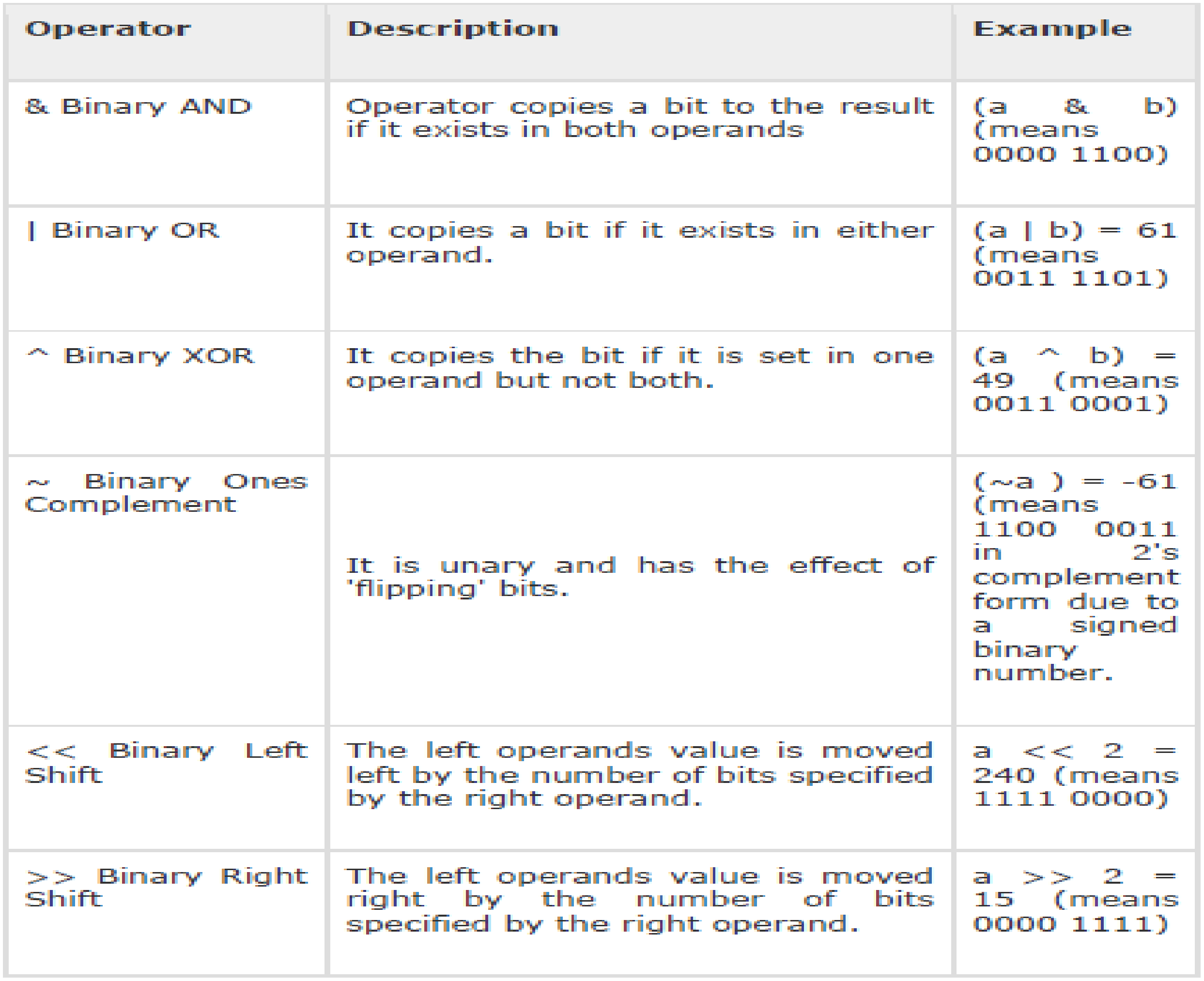
Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) 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.

ASSIGNMENT OPERATORS



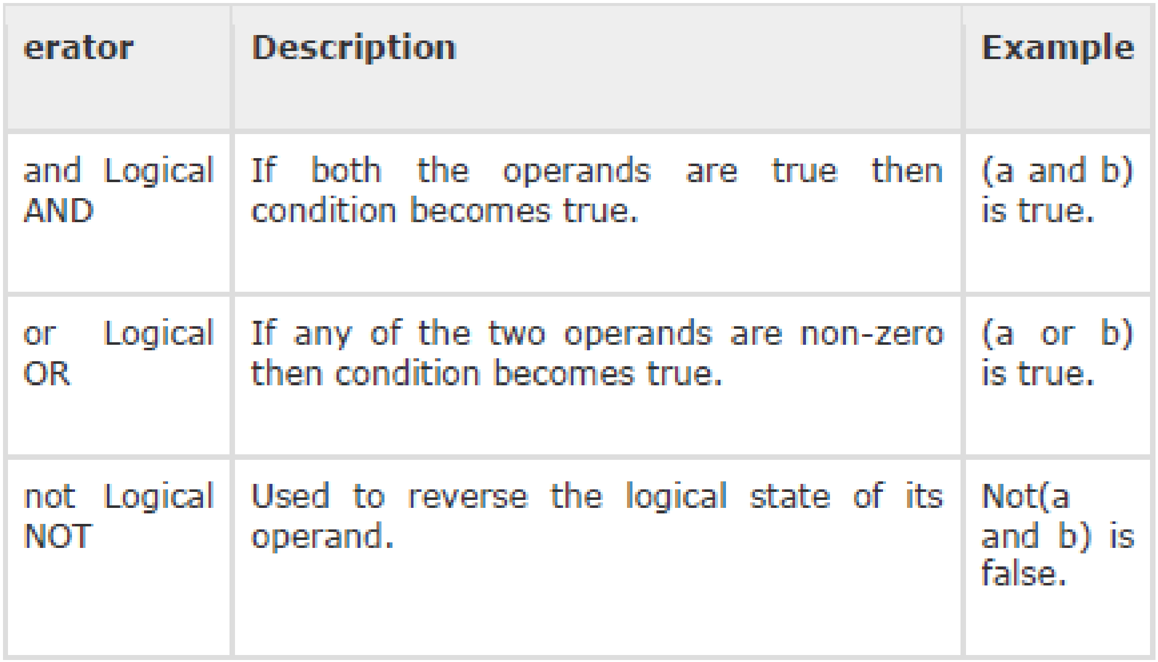
Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from 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 *

Bitwise operators



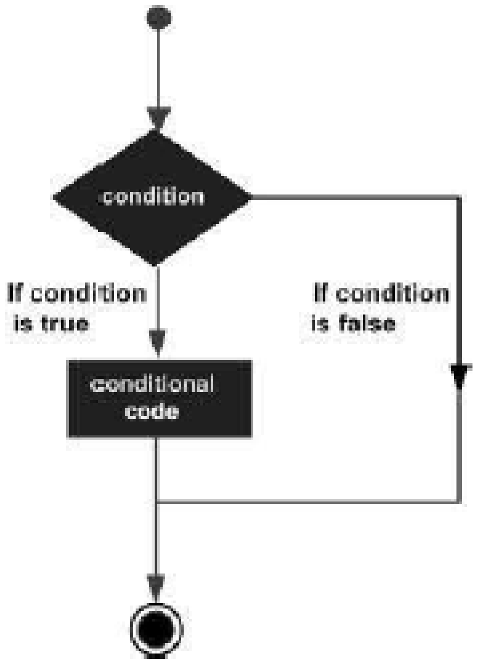
Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ 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.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Logical operators

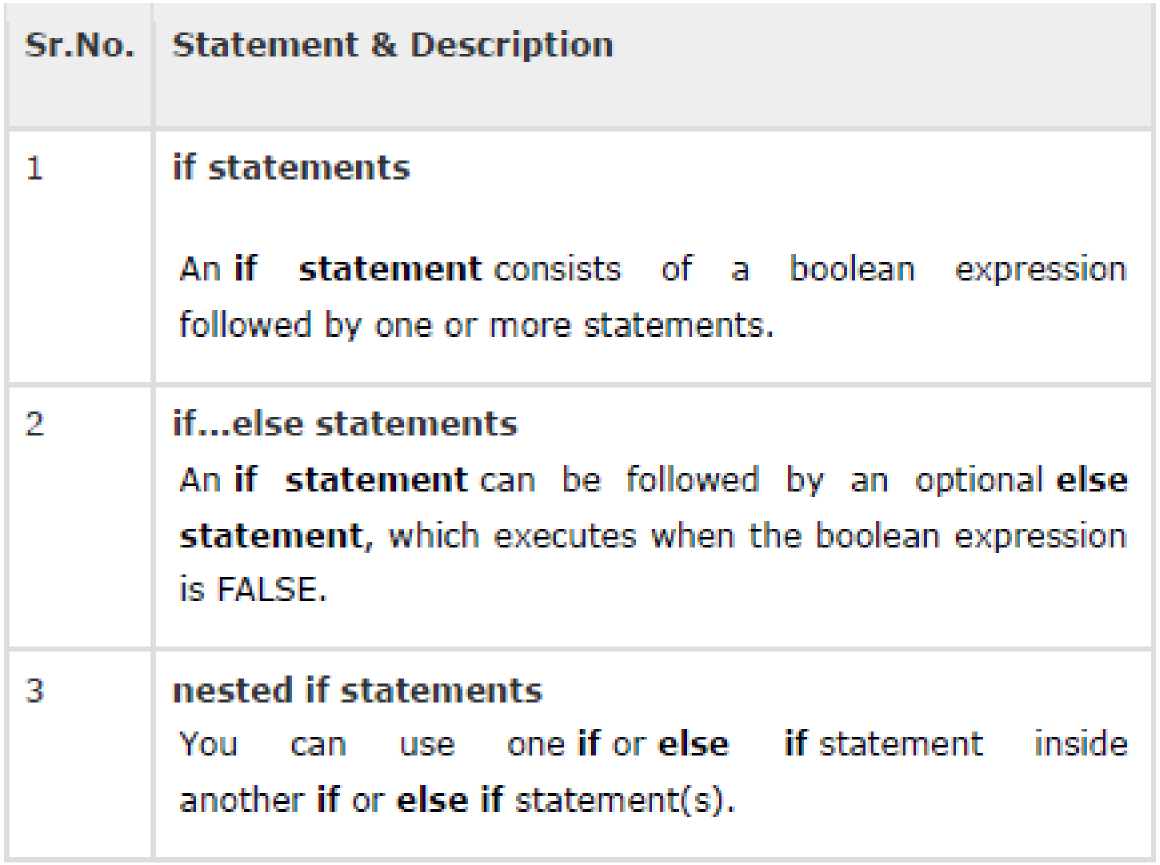


erator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b)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

Decision Making



Decision Making



Sr.No.	Statement & Description
1	**if statements** An **if statement** consists of a boolean expression followed by one or more statements.
2	**if...else statements** An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
3	**nested if statements** You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

Decision Making

```
var1 = 100
if var1:
    print ("1 - Got a true expression
value“)
print (var1)
```

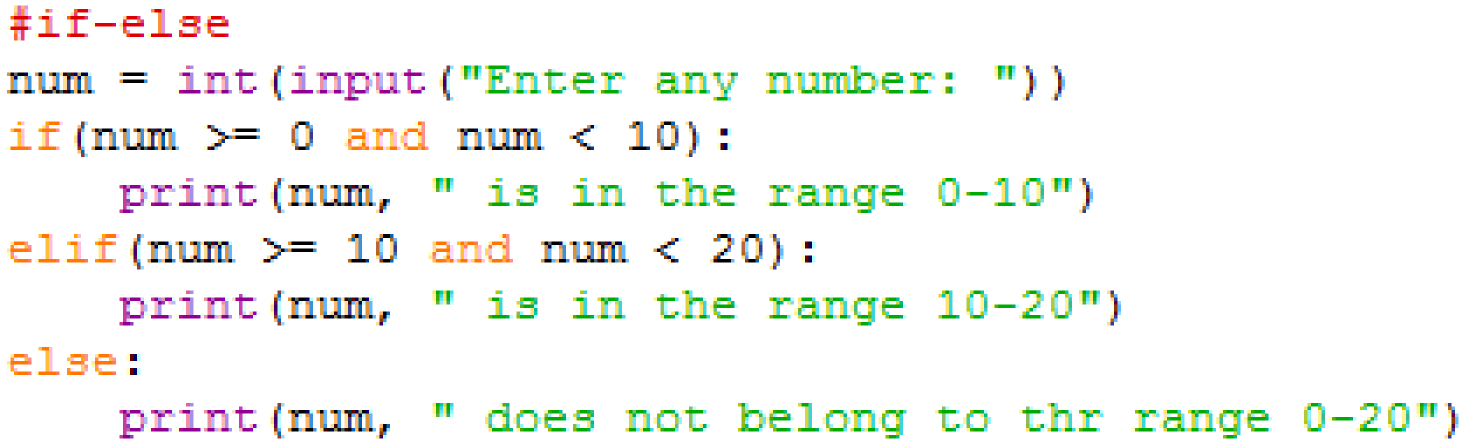
Decision Making

```
var1 = 100
if var1 > 10:
    print ("1 - Got a true expression
value“)
    print (var1)
else:
    print ("1 - Got a false expression
value")
    print var1
```

Decision Making

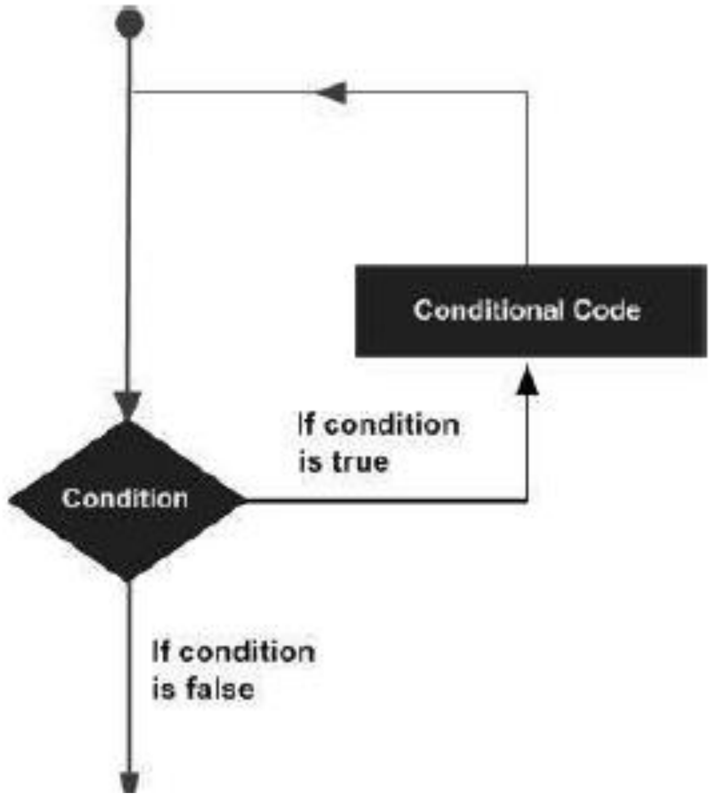
```
var = 100
if var == 200:
    print ("1 - Got a true expression value")
    print (var)
elif var == 150:
    print ("2 - Got a true expression value")
    print (var)
elif var == 100:
    print ("3 - Got a true expression value")
    print (var)
else:
    print ("4 - Got a false expression value")
    print (var)
```

Decision Making

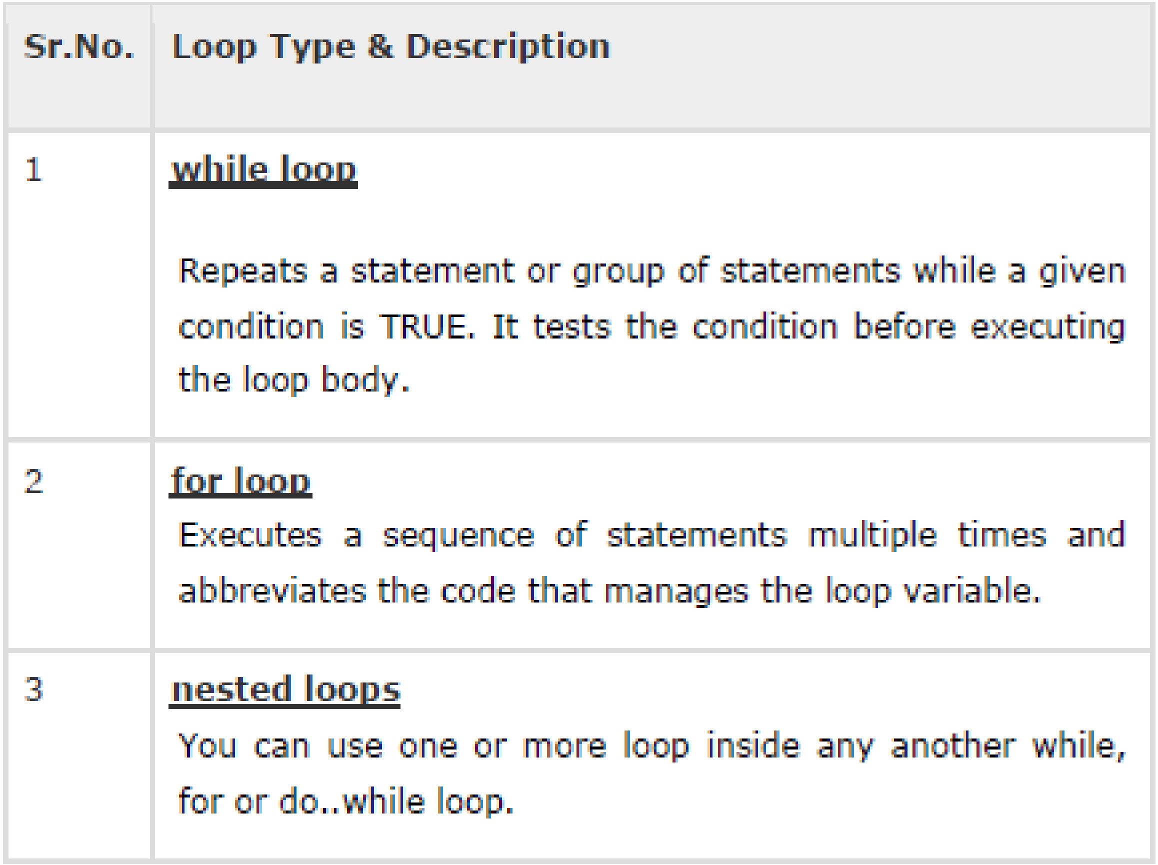


```
#if-else
num = int(input("Enter any number: "))
if(num >= 0 and num < 10):
    print(num, " is in the range 0-10")
elif(num >= 10 and num < 20):
    print(num, " is in the range 10-20")
else:
    print(num, " does not belong to thr range 0-20")
```

LOOPS

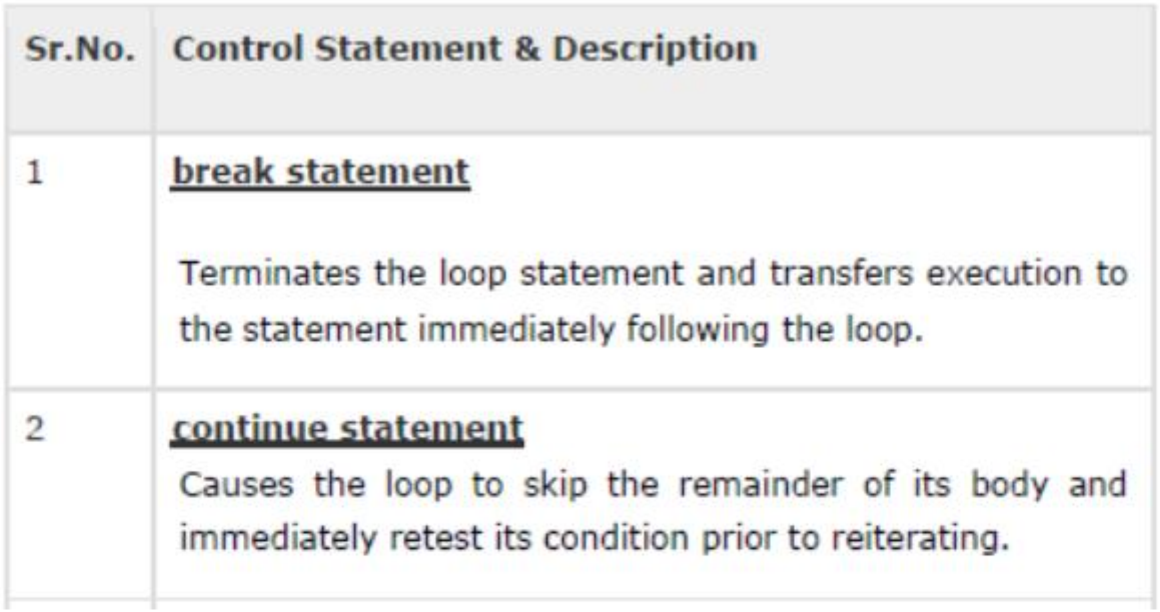

Conditional Code
If condition
is true
Condition
If condition
is false

LOOPS



Sr.No.	Loop Type & Description
1	**while loop** Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
2	**for loop** Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	**nested loops** You can use one or more loop inside any another while, for or do..while loop.

LOOPS

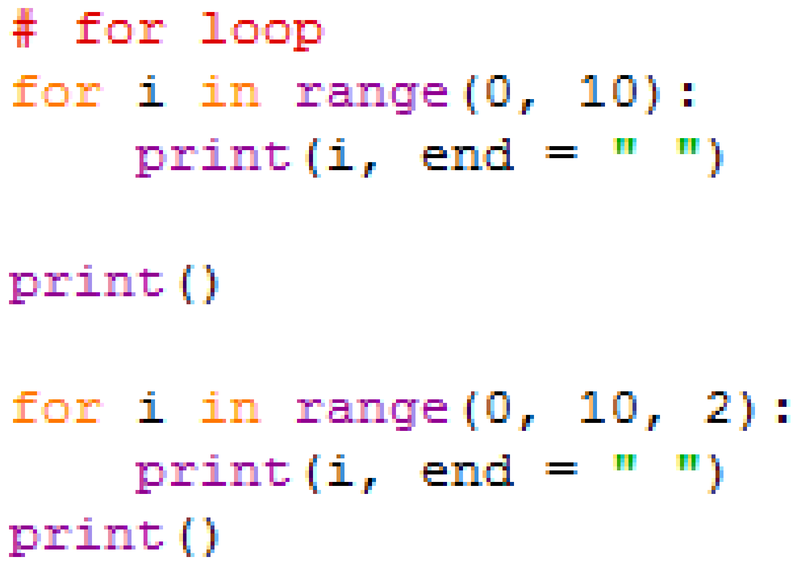


Sr.No.	Control Statement & Description
1	**break statement** Terminates the loop statement and transfers execution to the statement immediately following the loop.
2	**continue statement** Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Loops

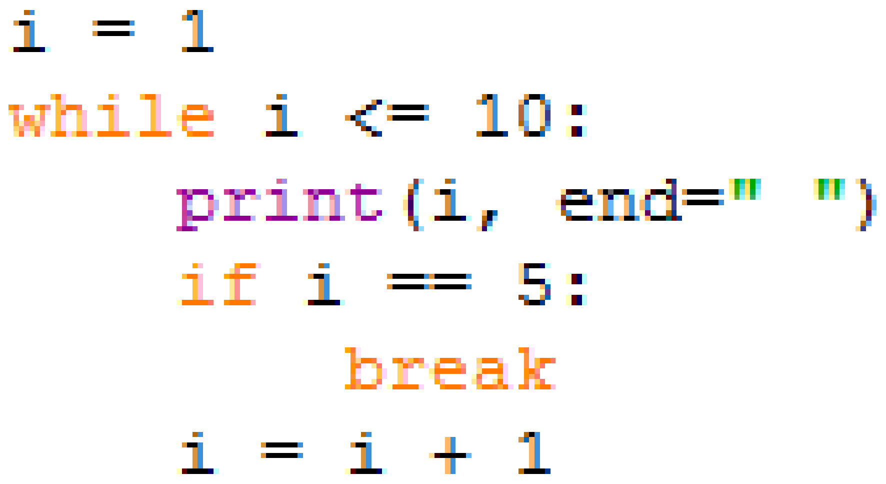
```
count = 0
while (count < 9):
    print ('The count is:', count)
    count = count + 1
print "Good bye!"
```

LOOPS



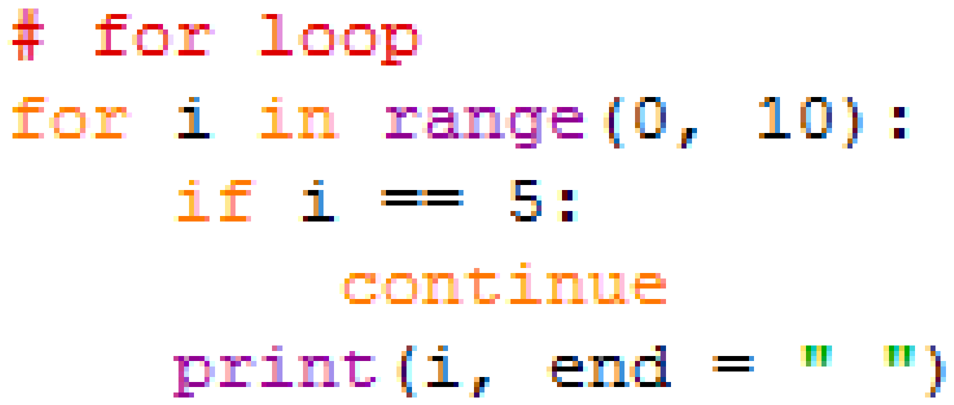
```
# for loop
for i in range(0, 10):
    print(i, end = " ")

print()

for i in range(0, 10, 2):
    print(i, end = " ")
print()
```

LOOPs



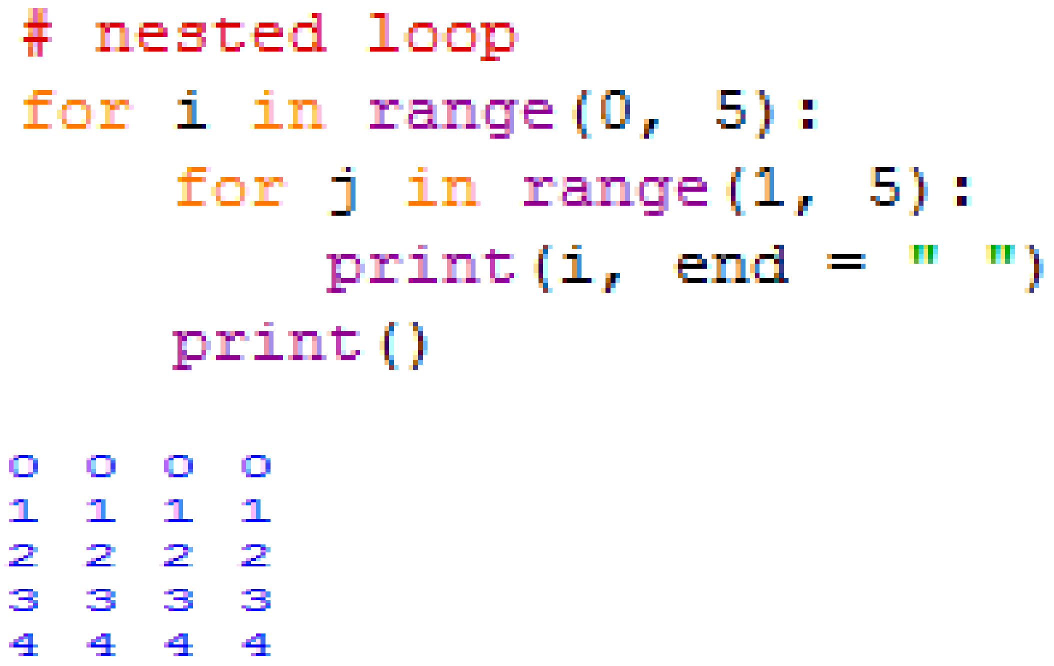
```python
i = 1
while i <= 10:
    print(i, end=" ")
    if i == 5:
        break
    i = i + 1
```

Loops



```python
# for loop
for i in range(0, 10):
    if i == 5:
        continue
    print(i, end = " ")
```

LOOPS



```
# nested loop
for i in range(0, 5):
    for j in range(1, 5):
        print(i, end = " ")
    print()
```

```
0 0 0 0
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
```

Thank You & Stay Tuned!