# Python Programming

LECTURES BY

DR. AVINASH GULVE

# Expressions

You can produce new data (numeric or text) values in your program using *expressions*

- This is an expression that uses the *addition operator*

```
>>> x = 2 + 3
>>> print(x)
5
>>> print(5 * 7)
35
>>> print("5" + "7")
57
```

# Expressions

You can produce new data (numeric or text) values in your program using *expressions*

- This is an expression that uses the *addition operator*
- This is another expression that uses the *multiplication operator*

```
>>> x = 2 + 3
>>> print(x)
5
>>> print(5 * 7)
35
>>> print("5" + "7")
57
```

# Expressions

You can produce new data (numeric or text) values in your program using *expressions*

- This is an expression that uses the *addition operator*
- This is another expression that uses the *multiplication operator*
- This is yet another expression that uses the *addition operator* but to *concatenate* (or glue) strings together

```
>>> x = 2 + 3
>>> print(x)
5
>>> print(5 * 7)
35
>>> print("5" + "7")
57
```

## Expressions

You can produce new data (numeric or text) values in your program using *expressions*

*Another example…*

```
>>> x = 6
>>> y = 2
>>> print(x - y)
4
>>> print(x/y)
3.0
>>> print(x//y)
3
```

*Yet another example…*

```
>>> print(x*y)
12
>>> print(x**y)
36
>>> print(x%y)
0
>>> print(abs(-x))
6
```

## Expressions

When integers and reals are mixed, the result is a real number.

- Example: `1 / 2.0` is `0.5`

- The conversion occurs on a per-operator basis.

```
7 // 3 * 1.2 + 3 // 2
   2   * 1.2 + 3 / 2
     2.4     + 3 / 2
     2.4     +   1
         3.4
```

## Python Operators

- Operators in general are used to perform operations on values and variables in Python.
- These are standard symbols used for the purpose of logical and arithmetic operations.

## 1. Arithmetic Operators

Considering X=10 and Y =3

| Operator | Operation | Syntax | Result |
|---|---|---|---|
| + | Addition: adds two operands | X + Y | 13 |
| - | Subtraction: subtracts two operands | X – Y | 7 |
| * | Multiplication: multiplies two operands | X * Y | 50 |
| / | Division (float): divides the first operand by the second | X / Y | 3. 3333333333333335 |
| // | Division (floor): divides the first operand by the second | X // Y | 3 |
| % | Modulus: returns the remainder when first operand is divided by the second | X % Y | |
| ** | Power : Returns first raised to power second | X ** Y | 1000 |

## Division

- When we divide integers with `/` , the quotient is also an integer.

```
         3                       52
      ____                    ______
  4 ) 14                 27 ) 1425
      12                      135
       2                        75
                                54
                                21
```

  - More examples:
    - `35 / 5` is `7`
    - `84 / 10` is `8`
    - `156 / 100` is `1`

- The `%` operator computes the remainder from a division of integers.

```
         3                        43
      ____                     _____
  4 ) 14                   5 ) 218
      12                       20
       2                        18
                                15
                                 3
```

# Operators on INT and Float

- ❑ a + b → sum
- ❑ a - b → difference
- ❑ a * b → product
- ❑ a / b → division

if both are ints, result is int

if either or both are floats, result is float

result is float

- ❑ a % b → Remainder
- ❑ a ** b → power

## 2. Relational Operators

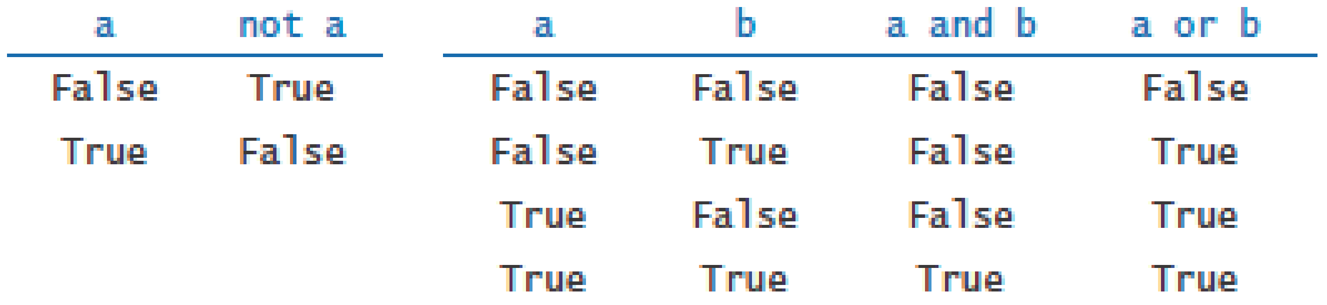
Considering X=10 and Y =3

| Operator | Operation | Syntax | Result |
|---|---|---|---|
| > | Greater than: True if left operand is greater than the right | x > y | True |
| < | Less than: True if left operand is less than the right | x < y | False |
| == | Equal to: True if both operands are equal | x == y | False |
| != | Not equal to - True if operands are not equal | x != y | True |
| >= | Greater than or equal to: True if left operand is greater than or equal to the right | x >= y | True |
| <= | Less than or equal to: True if left operand is less than or equal to the right | x <= | False |

# 3. Logical Operators

Considering X= True  and Y = False

| Operator | Operation | Syntax | Result |
|---|---|---|---|
| and | Logical AND: True if both the operands are true | x and y | False |
| or | Logical OR: True if either of the operands is true | x or y | True |
| not | Logical NOT: True if operand is false | not x | False |



| a | not a |
|---|---|
| False | True |
| True | False |

| a | b | a and b | a or b |
|---|---|---|---|
| False | False | False | False |
| False | True | False | True |
| True | False | False | True |
| True | True | True | True |

*Truth-table definitions of* `bool` *operations*

## 4. Bitwise Operators

Considering X=10 and Y =4

| Operator | Operation | Syntax | Result | |
|---|---|---|---|---|
| & | Bitwise AND | x & y | 0 | 1010 & 0100 -> 0000 -> 0 decimal |
| \| | Bitwise OR | x \| y | 14 | 1010 \| 0100 -> 1110 -> 14 decimal |
| ~ | Bitwise NOT | ~x | ~11 | ~1010 -> -(1010+1) -> -(1011) -> 11 decimal |
| ^ | Bitwise XOR | x ^ y | 14 | 1010 ^ 0100 -> 1110 -> 14 decimal |
| >> | Bitwise right shift | x>> 2 | 2 | 1010 >> 2 -> 0010 -> 2 decimal |
| << | Bitwise left shift | x<< 2 | 40 | 1010 << 2 -> 101000 -> 40 decimal |

## 5. Assignment Operators

- We assign a value to a variable using the assignment statement (=)
- An assignment statement consists of an expression on the right hand side and a variable to store the result

```
x = 3.9   *   x   *   (  1   -   x  )
```

## Simultaneous Assignment

- Python allows us also to assign multiple values to multiple variables all at the same time
- This form of assignment might seem strange at first, but it can prove remarkably useful (e.g., for swapping values)

```
>>> a, b = 2, 5
>>> print(a)
2
>>> print(b)
5
>>>
```

```
>>> a, b = 2, 5
>>> b, a = a, b
>>> print(a)
5
>>> print(b)
2
```

## 5. Assignment Operators

| Operator | | Operation |
|---|---|---|
| = | Assign value of right side of expression to left side operand | x = y + z |
| += | Add AND: Add right side operand with left side operand and then assign to left operand | a+=b  a=a+b |
| -= | Subtract AND: Subtract right operand from left operand and then assign to left operand | a-=b  a=a-b |
| *= | Multiply AND: Multiply right operand with left operand and then assign to left operand | a*=b  a=a*b |
| /= | Divide AND: Divide left operand with right operand and then assign to left operand | a/=b  a=a/b |
| %= | Modulus AND: Takes modulus using left and right operands and assign result to left operand | a%=b  a=a%b |
| //= | Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand | a//=b  a=a//b |

# 5. Assignment Operators

| Operator | | Operation |
|---|---|---|
| **= | Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand | a**=b a=a**b |
| &= | Performs Bitwise AND on operands and assign value to left operand | a&=b a=a&b |
| \|= | Performs Bitwise OR on operands and assign value to left operand | a\|=b a=a\|b |
| ^= | Performs Bitwise xOR on operands and assign value to left operand | a^=b a=a^b |
| >>= | Performs Bitwise right shift on operands and assign value to left operand | a>>=b a=a>>b |
| <<= | Performs Bitwise left shift on operands and assign value to left operand | a <<= b a= a << b |

## 6. Special Operators- Identity Operator

**is** and **is not** are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

**is** True if the operands are identical

**is not** True if the operands are not identical

```
>>> a1=3
>>> b1=3
>>> a1 is not b1
False
>>> a1 is b1
True
```

```
>>> a1='College'
>>> b1='College'
>>> a1 is b1
True
>>> a1 is not b1
False
```

```
>>> a1=[1,2,3]
>>> b1=[1,2,3,4]
>>> a1 is b1
False
>>> a1 is not b1
True
```

## 6. Special Operators-Membership Operator

**in** and **not in** are the membership operators; used to test whether a value or variable is in a sequence.

**in** True if value is found in the sequence

**not in** True if value is not found in the sequence

```
>>> x='Government'
>>> print('ve' in x)
True
>>> print('O' not in x)
True
```

```
>>> x=[1,2,3,4,5]
>>> print( 3 in x)
True
>>> print(7 in x)
False
>>> print(7 not in x)
True
```

## Operator Precedence

- When we string operators together - Python must know which one to do first
- This is called “operator precedence”
- Which operator “takes precedence” over the others

## Operator Precedence

- When we string operators together - Python must know which one to do first
- This is called "operator precedence"
- Which operator "takes precedence" over the others
- This is used in an expression with more than one operator with different precedence to determine which operation to perform first.

# Operator  Precedence

```
>>> x = 1 + 2 ** 3 // 4 * 5
>>> print x
11
>>>
```

$$\begin{tikzcd}
1 + 2 ** 3 // 4 * 5 \arrow[d] \\
1 + 8 // 4 * 5 \arrow[d] \\
1 + 2 * 5 \arrow[d] \\
1 + 10 \arrow[d] \\
11
\end{tikzcd}$$

$$\begin{tikzcd}
\text{Parenthesis} \arrow[d] \\
\text{Power} \arrow[d] \\
\text{Multiplication} \arrow[d] \\
\text{Addition} \arrow[d] \\
\text{Left to Right}
\end{tikzcd}$$

# Operator Precedence

Python operator precedence

1. Operations enclosed in parentheses
   - Forces operations to be performed before others
2. Exponentiation (**)
3. Multiplication (*), division (/ and //), and remainder (%)
4. Addition (+) and subtraction (-)

Higher precedence performed first

- Same precedence operators execute from left to right

## Operators Associativity

- If an expression contains two or more operators with the same precedence then Operator Associativity is used to determine. It can either be Left to Right or from Right to Left.

## Operators Associativity

| Operator | Description | Associativity |
|---|---|---|
| ( ) | Parentheses | left-to-right |
| ** | Exponent | right-to-left |
| * / % | Multiplication/division/modulus | left-to-right |
| + - | Addition/subtraction | left-to-right |
| << >> | Bitwise shift left, Bitwise shift right | left-to-right |
| < <=<br>> >= | Relational less than/less than or equal to<br>Relational greater than/greater than or equal to | left-to-right |
| == != | Relational is equal to/is not equal to | left-to-right |

## Assigning Input

So far, we have been using values specified by programmers and printed or assigned to variables

- How can we let users (not programmers) input values?

In Python, input is accomplished via an assignment statement combined with a built-in function called *input*

When Python encounters a call to *input*, it prints <prompt> (which is a string literal) then pauses and waits for the user to type some text and press the <Enter> key

**<variable> = input(<prompt>)**

The string can be converted by using the conversion methods int(string), float(string), etc.

## Assigning Input

Here is a sample interaction with the Python interpreter:

Notice that whatever the user types is then stored as a string

- What happens if the user inputs a number?

```
>>> name=input('Who are you')
Who are you I am Avinash
>>> name
' I am Avinash'
>>> type(name)
<class 'str'>
>>>
```

```
>>> num=eval(input('Enter a number\t'))
Enter a number      34
>>> num
34
>>> type(num)
<class 'int'>
>>>
```

## Assigning Input - Datatype Conversion

We can convert the string output of the *input* function into an integer or a float using the built-in ***int*** and ***float*** functions

```
>>> number = input("Enter a number: ")
Enter a number: 34
>>> number
'34'
>>> type(number)
<class 'str'>
```

```
>>> number = int(input("Enter a number: "))
Enter a number: 34
>>> number
34
>>> type(number)
<class 'int'>
```

**An integer (no single quotes)!**

## Assigning multiple Inputs

```
x, y = input("Enter two values\t").split()
print("Number of boys: ", x)
print("Number of girls: ", y)
x=int(x)
y=int(y)
print(x+y)
```

```
x, y, z = input("Enter three values \t").split()
x=float(x)
y=float(y)
z=float(z)
print(x)
print(y)
print(z)
print(x+y+z)
```

## Assigning multiple Inputs

```
 x, y, z = input("Enter three values \t").split()
print("The entered numbers are {},{} and {}".format(x,y,z))
print("The entered numbers are {0},{1} and {2}".format(x,y,z))
x=float(x)
y=float(y)
z=float(z)
print("and addition is ", x+y+z)
```

## Printing

When writing scripts, your outcomes aren't printed on the terminal.

Thus, you must print them yourself with the print() function.

```
>>> print("Python is powerful")
Python is powerful
>>> x="Python is powerful "
>>> y="and versatile"
>>> print(x+y)
Python is powerful and versatile
```

## Printing

```
>>> x="Python is powerful "
>>> y=76
>>> str1="Python is powerful "
>>> str2=52
>>> str3="and versatile"
>>> print(str1+str2+str3)
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    print(str1+str2+str3)
TypeError: can only concatenate str (not "int") to str
>>>
```

Beware to not mix up the different type of variables!

## Printing

- The print() function prints the given object to the standard output device (screen) or to the text stream file.

*Syntax: print(value(s), sep= ' ', end = '\n', file=file, flush=flush)*
**Parameters:**
**value(s)** : Any value, and as many as you like. Will be converted to string before printed
**sep**='separator' : (Optional) Specify how to separate the objects, if there is more than one. Default :' '
**end**='end': (Optional) Specify what to print at the end. Default : '\n'
**file** : (Optional) An object with a write method. Default :sys.stdout
**flush** : (Optional) A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False

**Returns**: It returns output to the screen.

## Printing – end parameter

By default, python's print() function ends with a newline. This function comes with a parameter called 'end.' The default value of this parameter is '\n,' i.e., the new line character. You can end a print statement with any character or string using this parameter.

```
a=7
print("a= ",a,   sep='00000',    end='\n\n\n')
print("a= ",a,   sep='0',   end = '‘ )
print("a= ",a,  sep='**')

Output:
a= 000007



a= 07a= **7
```

```
a=7
print("a= ",a,sep='00000',end =  4* ‘\n‘ )
print("a= ",a,sep='**')

Output
a= 000007



a= **7
```

## Printing – sep parameter

The print() function can accept any number of positional arguments. These arguments can be separated from each other using a **“,” separator**. These are primarily used for formatting multiple statements in a single print() function.

```
s="of"
print("Best",s,"Luck")
print("Best","Luck",sep=' of ')
print("Best","Luck",sep=' ')
```

Output

```
Best of Luck
Best of Luck
Best Luck
```

## Formatting output

Formatting output using String

1. modulo operator(%)

```
a=10
b=3.1412
print("a= %2d and b = %5.2f"
%(a,b))
```

output

```
a= 10 and b =  3.14
```

2. format method

```
a=10
b=205
print("a= {0} b = {1} and c= {other}".format(a,b,other=4567))
print("b= {1:3d} a = {0:2d} and c= {other:4d}".format(a,b,other=4567))
```

Output

```
a= 10 b = 205 and c= 4567
b= 205 a = 10 and c= 4567
```

..\..\..\..\..\Python Programs\myprogs\print_format3.py

## Formatting output

3. String method

```
dept = "MCA Department"
print ("Center aligned string with fillchr: ")
print (dept.center(40, '#'))
print ("The left aligned string is : ")
print (dept.ljust(40, '!'))
print ("The right aligned string is : ")
print (dept.rjust(40, '@'))
```

Output

```
Center aligned string with fillchr:
#############MCA Department#############
The left aligned string is :
MCA Department!!!!!!!!!!!!!!!!!!!!!!!!!!
The right aligned string is :
@@@@@@@@@@@@@@@@@@@@@@@@@@MCA Department
```

## Printing - Formatting

```
for x in range(1, 11):
   print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
Output
  1    1       1
  2    4       8
  3    9     27
  4  16     64
  5  25   125
  6  36   216
  7  49   343
  8  64   512
  9  81   729
10 100 1000
```

Argument number