Computational Thinking and Programming - 1

Python - Operators

XI

Python - Operators

An operator is a special symbol in Python that is used to to perform specific mathematical or logical computation on values/variables. The values that the operators work on are called operands.

For example, in the expression num1 + num2, the variable num1 and num2 are operands and the + (plus) sign is an operator.

Python supports several kinds of operators whose categorisation will be covered in the next slides.

// Arithmetic Operators

Python uses arithmetic operators upon different type of data depending on the compatibility of the operator with the literal which is it used:

SNo.	Name	Symbol	Example	Result
1.	Exponentiation	**	3**2	9
	(R to L)		2 ** 3 ** 2	2 ** 9 = 512
			4 ** 0.5	2.0
2.	Multiplication	*	3 * 4	12
			2.0 * 3	6.0
			“Python” * 2	PythonPython

Arithmetic Operators

3.	Addition	+	3 + 2	5
			5 + 2.0	7.0
			“Python” + “World”	PythonWorld
4.	Subtraction	-	3 - 2	1
			5.0 - 2	3.0
5.	Division	/	6/2	3.0
			5.0/2	2.5
6.	Floor Division	//	7//2	3
			7//2.0	3.0
7.	Modulo/Remainder	%	5%2	1
			7.0% 3	1.0

Precedence of Arithmetic Operators

Order of Precedence	Operators
1	()
2	**
3	+ - (Unary)
4	* / % //

Order of Precedence	Operators
5	+ - (Binary)
6	<= < > >=
7	== !=
8	= %= /= //= -= += *= **=

Evaluation of Arithmetic Expressions

The order of precedence of operators determines the result of any arithmetic expression. However the precedence of the operators may be altered by writing an expression in parentheses. For example,

Expression	Result
$2 \overset{\textcircled{2}}{+} 3 \overset{\textcircled{1}}{*} 5$	2 + 15 = 17
$2 \overset{\textcircled{3}}{-} 4 \overset{\textcircled{1}}{*} 9 \overset{\textcircled{4}}{+} 6 \overset{\textcircled{2}}{/} 2$	2 - 36 + 3.0 = -31.0
$2 \overset{\textcircled{3}}{+} 4 \overset{\textcircled{2}}{*} (9 \overset{\textcircled{1}}{-} 6)$	2 + 4 * 3 = 2 + 12 = 14

Facts about Arithmetic Operators

All arithmetic operators are left bound, the exponential operator ** is always right hand i.e. consecutive occurrences of ** operators in an expression is always evaluated starting with the 2 operands in the right.

Example:

$2 \overset{\textcircled{2}}{**} 3 \overset{\textcircled{1}}{**} 2 = 2 ** 9 = 512$

$(2 \overset{\textcircled{1}}{**} 3) \overset{\textcircled{2}}{**} 2 = 8 ** 2 = 64$

$13 \overset{\textcircled{1}}{\%} 5 \overset{\textcircled{2}}{\%} 2 = 3 \% 2 = 1$

$13 \overset{\textcircled{2}}{\%} (5 \overset{\textcircled{1}}{\%} 3) = 13 \% 2 = 1$

Some more facts about Arithmetic Operators

The division operator (/) always gives the result of the division as a float value.

The floor division operator (//) gives the nearest smaller integer result of the division of the operands. **(In case any of the operands is a float, the final result is a float equivalent of the int result)**

A modulo operator returns the remainder of the division operation of the two operands. Numerator % Denominator is calculated as

Numerator = Quotient * Denominator + Remainder or

Remainder = Numerator - Quotient * Denominator where

Quotient = Numerator // Denominator

x % y = x - (x//y) * y

XI

Quick Exercise:

```
7/2 =

7.0/2 =

17//4 =

17.0 // 4 =

-17 // 2 =

-17 / 2 =
```

```
-17 % -2 =

17 % -2 =

-9.5 % 2 =

 9.5 % 2 =

-17 % -5=

8 // 5 =
```

Quick Exercise:

7/2 =   3.5

7.0/2 =3.5

17//4 =4

17.0 // 4 =4.0

-17 // 2 =-9

-17 / 2 =- 8.5

-17 % -2 = -1

17 % -2 = -1

-9.5 % 2 = 0.5

9.5 % 2 =1.5

-17 % -5= -2

8 // 5 = 1

Relational Operators

Relational operators compares two values and returns either **True** or **False** according to the condition.

Operator	Meaning	Example
==	Equal to	a == b
!=	Not equal to	a != b
<	Less than	a < b
>	Greater than	a > b
<=	Less than or equal to	a <= b
>=	Greater than or equal to	a >= b

Logical Operators

Logical Operators operate upon Boolean values and return a result either **True** or **False**

Operator	Meaning	Example
not	Reverses the result, returns True if the condition evaluates to False, False otherwise	**not** (a < b and b>10)
or	Returns True if one of the condition evaluates to True	a < b **or** b < c
and	Returns True only if both or all the conditions evaluate to True	a < b **and** b < c

Logical Operators

Logical operators (unary) - not

Condition	not(Condition)
True	False
False	True

Logical operators (binary) - and & or

Condition 1	Condition 2	and	or
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

Precedence of Logical Operators

When multiple logical operators exist in an expression, their execution sequence is decided by their precedence order, which is not > and > or. It means that not has higher priority than and which has higher priority than or. This precedence order can be overridden by using parentheses.

Let us understand :

We assume that **x=3, y=4, and z=6.**

Expression	Evaluation
x>y and y>z or x<z	False and False or True (False and False) or True False or True True
x>y or y>z and x <z	False or False and True False or False False

Expression	Evaluation
not x>y or y<z x=3, y=4, and z=6	not False or True True or True True
not (x>y or y<z)	not (False or True) not True False
not (x < y and y < z)	not (True and True) not True False

NOTE :

- The AND operator checks for the second condition only when the first is true,otherwise ignores it.

  a=4 and 0 will store 0

- The OR operator will test the second condition only if the first operand is false,otherwise it ignores it.

  b=30 or 0 will store 30

Example:

```
print(23==23,45==34)
print("Hello"=="Bye","Bye"=="Bye")
print(43>23,43<23)
print("AB">"AC","AB"<"AC")
print(51<=51,51<=54)
print("51"<"533")
print(50>34 and 52<90)
print(50>34 and 52>90)
print(65>34 or 52<90,50<34 or 52>90)
print(not("Morning"=="Evening"))
print(not(25>35))
```

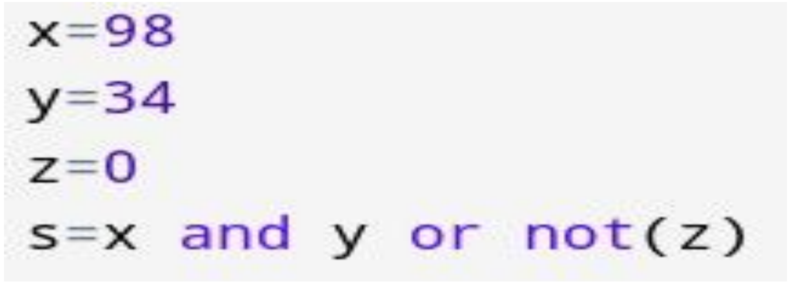
Example:

```
print(23==23,45==34)
print("Hello"=="Bye","Bye"=="Bye")
print(43>23,43<23)
print("AB">"AC","AB"<"AC")
print(51<=51,51<=54)
print("51"<"533")
print(50>34 and 52<90)
print(50>34 and 52>90)
print(65>34 or 52<90,50<34 or 52>90)
print(not("Morning"=="Evening"))
print(not(25>35))
```

```
True False
False True
True False
False True
True True
True
True
False
True False
True
True
```

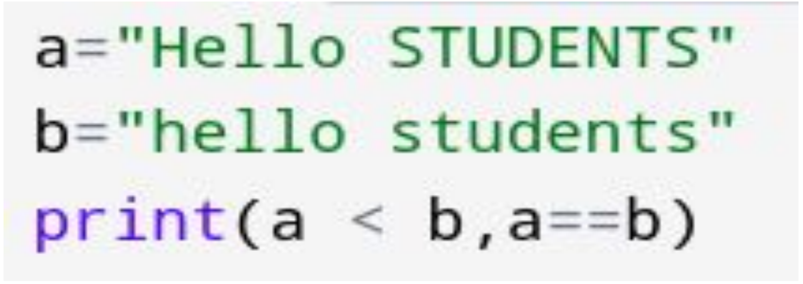
Quick Exercise:

Q1. What will be the output of following snippet :



```
x=98
y=34
z=0
s=x and y or not(z)
```

Q2. What will be the output of following snippet :



```
a="Hello STUDENTS"
b="hello students"
print(a < b,a==b)
```

Quick Exercise:

Q1.

The value of s will be 34

s=98 and 34 or not(0)

=98 and 34 or 1

=34 or 1

=34

Q2.

True False

Assignment and Augmented Assignment Operators

Assignment and Augmented Assignment operators are used to assign values to variables.

Operator	Meaning	Example
=	Assigns value of right operand to left operand	a = 10
+=	Adds value of right operand to left and assigns to left operand	a+= b is similar to a=a+b
-=	subtracts value of right operand to left and assigns to left operand	a-= b is similar to a=a-b

Assignment and Augmented Assignment Operators

Operator	Meaning	Example
*=	multiplies value of right operand with left operand and assigns to left operand	a*= b is similar to a=a*b
/=	divides value of left operand to right and assigns to left operand	a/= b is similar to a=a/b
//=	Performs floor division and assigns value to left operand	a//= b is similar to a=a//b
**=	Performs exponential calculation and assigns value to left operand	a**= b is similar to a=a**b

Identity Operators

Identity operators are used to determine whether the value of a variable is of a certain type or not. Identity operators can also be used to determine whether two variables are referring to the same object or not. There are two identity operators.

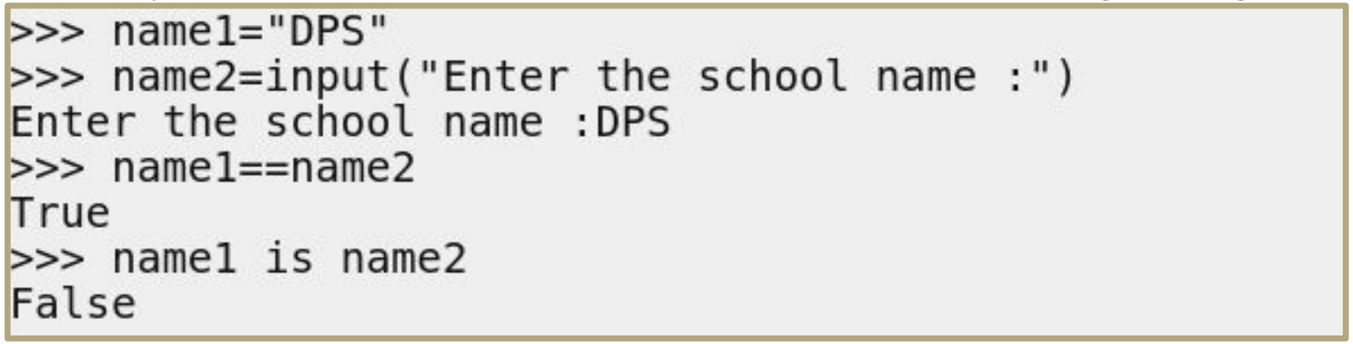
Operator	Meaning	Example
is	Evaluates to True if the variables on either side of the operator point to the same memory location and False otherwise. var1 is var2 results to True if id(var1) = id(var2)	`>>> num1=10` `>>> num2=10` `>>> id(num1)` `140658508700240` `>>> id(num2)` `140658508700240` `>>> num1 is num2` `True` `>>> num3=20` `>>> id(num3)` `140658508700560` `>>> num1 is not num3` `True` `>>> num1 is num3` `False`
is not	Evaluates to False if the variables on either side of the operator point to the same memory location and True otherwise. var1 is not var2 results to True if id(var1) != id(var2)	

Special Cases

If the expression a is b is True, it means that a==b will also be True. But it is not always true the other way round. That means in some cases if a==b, but a is b is False.

Situations where this is violated (if a == b, a is b is False)

1. Input of strings from the console (These strings are bound to fresh memory even if they have value identical to some other existing string in memory)



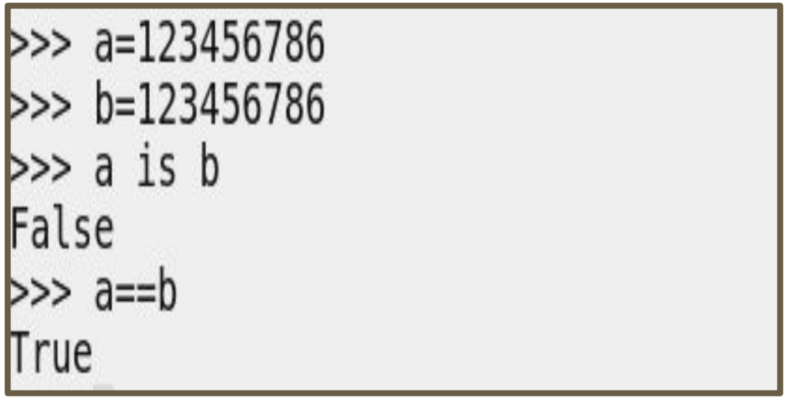
```
>>> name1="DPS"
>>> name2=input("Enter the school name :")
Enter the school name :DPS
>>> name1==name2
True
>>> name1 is name2
False
```

Special Cases

2. Writing integer literals with many digits (very big integers)

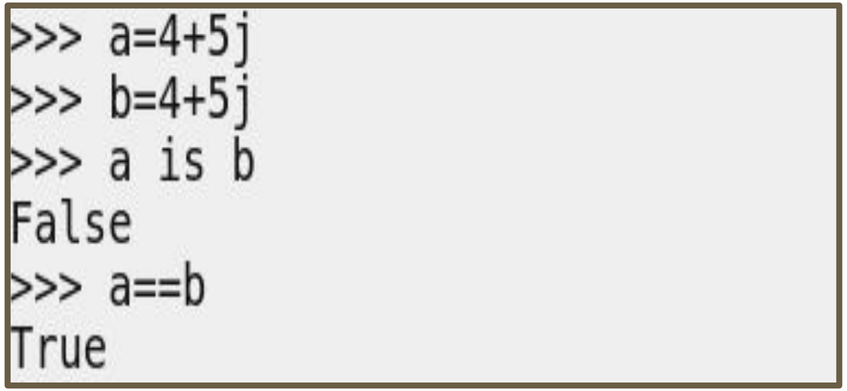
3. Writing floating point literals and complex literals

Big Integers



```
>>> a=123456786
>>> b=123456786
>>> a is b
False
>>> a==b
True
```

Complex Numbers



```
>>> a=4+5j
>>> b=4+5j
>>> a is b
False
>>> a==b
True
```

Membership Operators

Membership Operators are used to check if a value is a member of a given sequence or not.

Operator	Meaning	Example
in	Returns True if the variable/value is found in the specified sequence and False otherwise.	a = [1,2,3] 2 in a returns True '1' in a returns False
not in	Returns True if the variable/value is not found in the specified sequence and False otherwise.	a = [1,2,3] 10 not in a returns True 1 not in a returns False

Precedence of Operators

Parenthesis can be used to override the precedence. The expression within () is evaluated first. For operators with equal precedence, they are evaluated left to right

Order of Precedence	Operators
1	**
2	+,- (Unary)
3	*, /, %, //
4	+, - (Binary)
5	<=,<,>,>=

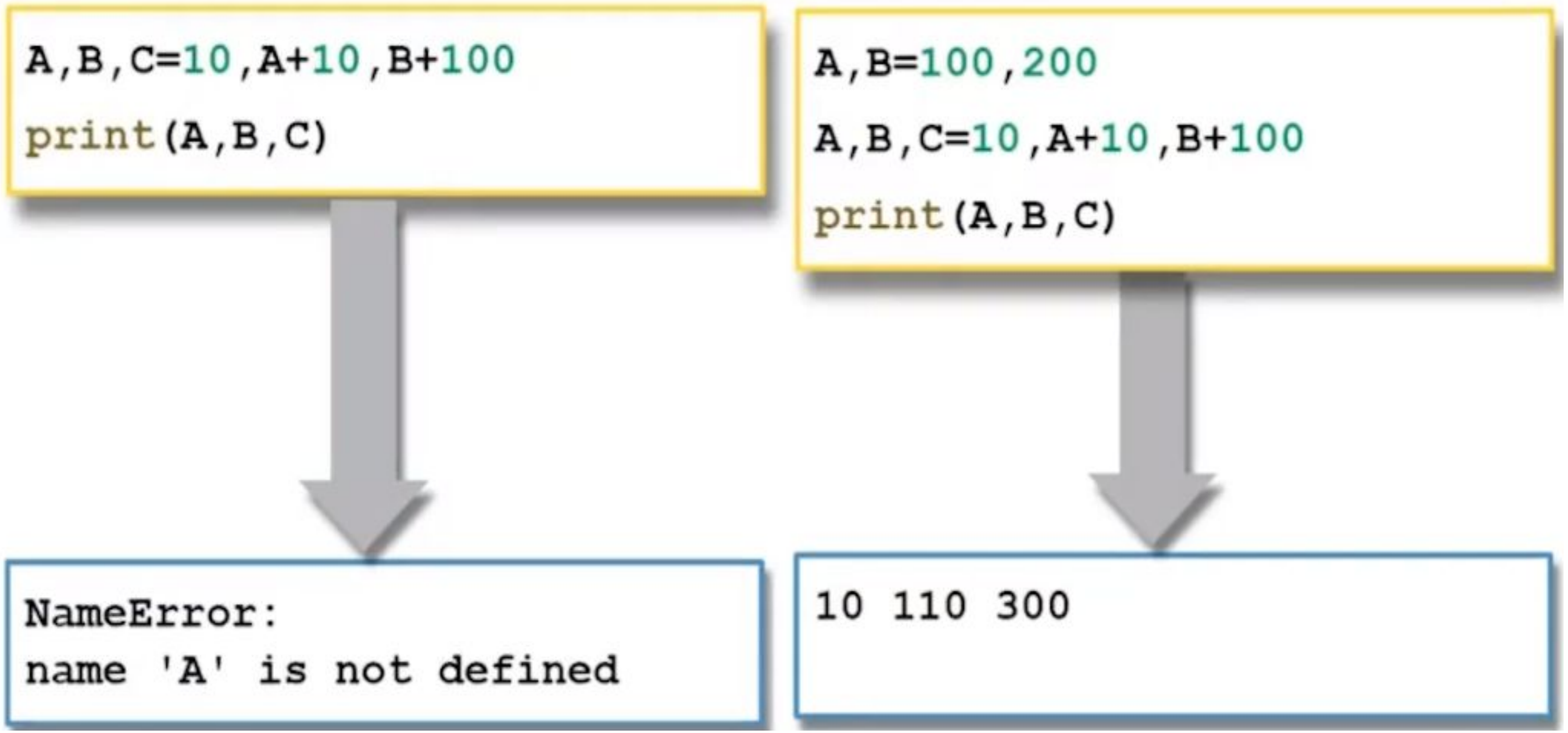
Order of Precedence	Operators
6	== , !=
7	=,*=,/=,%=,//=,+=,-=,**=
8	is, is not
9	in, not in
10	not, or, and

Quick Exercise:

What will be the output :

```
1.  print((4**2)*3**3)
2.  a=30

    b=40

    c=a*b//3

    print(c)

3.  x=2 *((3*12)-8/10)

    print("Value of x :",x)
```

Assignments in Python



```
A,B,C=10,A+10,B+100
print(A,B,C)
```

```
NameError:
name 'A' is not defined
```

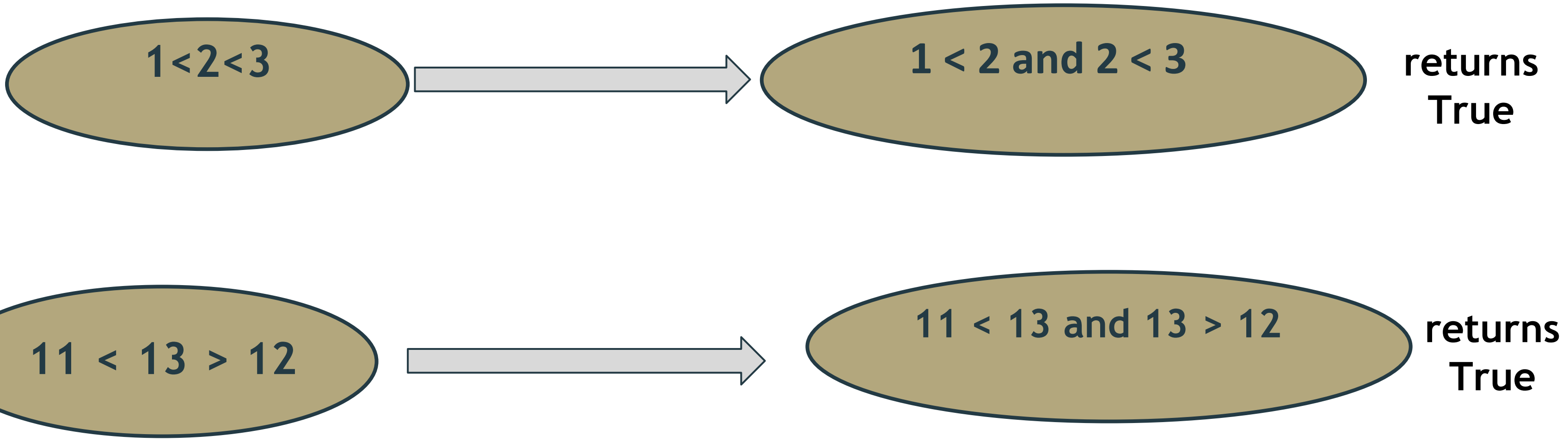
```
A,B=100,200
A,B,C=10,A+10,B+100
print(A,B,C)
```

```
10 110 300
```

Operator Chaining

Chaining of relational operators is allowed in Python.

For example: x<y<=z is equivalent to x < y and y <=z



Operator Associativity

Associativity is the order in which an expression (having multiple operators of the same precedence) is evaluated. Almost all the operators have left-to-right associativity except exponentiation (**), which has right to left associativity

Expression	Evaluation
7 * 8 / 5 //2	56/5//2 11.2//2 5.0
3 ** 3 ** 2	3 ** 9 19863

Quick Exercise

What will be the output :

```
1.  x = 36 / 4 * (3 +  2) * 4 + 2
    print(x)

2.  var = "James" * 2  * 3
    print(var)

3.  var1 = 1
    var2 = 2
    var3 = "3"
    print(var1 + var2 + var3)
```

Quick Exercise:

What will be the output :

5.
```
X=10
X= X+10

X= X-5

print (X)

X, Y = X-2, 22

print (X, Y)
```

6.
```
first = 2

second =3

third = first * second

print (first, second, third)

first= first + second + third

third = second * first

print (first, second, third)
```

Quick Exercise:

What will be the output :

7.
```
x= 40
y=x+1
x,y=20, y +x
print (x, y)
```

8.
```
x, y =20, 60
y, x, y=x, y-10, x+10
print (x, y)
```

Quick Exercise:

What will be the output :

9.
```
a , b, c=10, 20, 30
p , q, r= c-5, a+3, b-4
print (‘a, b, c:’ , a, b, c, end = ‘ ’)
print (‘p, q, r:’ , p, q, r)
```

10.
```
a, b, c = 2, 3, 4
 a , b, c = a*a, a*b, a*c
print (a, b, c)
```

Type Conversion

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.

- Implicit Type Conversion
- Explicit Type Conversion

Implicit Type Conversion (Type Promotion)

It refers to the data type conversion performed by the interpreter without the user's intervention. It is generally applied whenever an expression having operands belonging to mixed data types is to be evaluated. In a mixed arithmetic expression, Python converts all operands up to the type of the largest operand (type promotion, also known as coercion)

If both operands of an expression are standard numeric types, the following coercions are applied :

Implicit Type Conversion - Rules

If either operand is a complex number, the other is converted to a complex	>>4+(2+2j) >>6+2j
if either argument is a floating point number, the other is converted to floating point	>>7+8.3 >>15.3
No conversion takes place if both operands are integers.	>> 6+5 >> 11

Explicit Type Conversion (Type Casting)

An explicit type conversion is user defined conversion that forces an expression to be of a specific data type.

It is performed by <datatype>(expression) function.

Some data type conversion functions:

Function	Converted from	Converted to	Example
int()	Any number convertible type	integer	int(7.8) will give 7 int('12') will give 12
float()	Any number convertible type	floating point number	float(7) will give 7.0 float('12') will give 12.0
complex()	numbers	Complex number	complex(7) will give 7+0j complex(1,2) will give 1+2j

Some data type conversion functions:

Function	Converted from	Converted to	Example
str()	Numbers, boolean	string	str(3) will give ‘3’ str(1.23) will give ‘1.23’ str(1+2j) will give ‘(1+2j)’ str(True) will give ‘True’
bool()	Any type	boolean	bool(0) will give False bool(1) will give True bool(‘’) will give False bool(‘hello’) will give True bool(45) will return True

Thank you!

Department of Computer Science

Mathura Road