IIIT-Srikakulam, RGUKT-AP, Dept. Of CSE

Unit 4. Python Operators

What are operators in python?

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.
For example:

```
>>>2+3
5
```

Here, `+` is the operator that performs addition. `2` and `3` are the operands and `5` is the output of the operation.

Arithmetic operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Arithmetic operators in Python

Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Example #1: Arithmetic operators in Python

```
x = 15
y = 4
                                        # Output: x + y = 19
print('x + y =',x+y)
                                        # Output: x - y = 11
print('x - y =',x-y)
                                        # Output: x * y = 60
print('x * y =',x*y)
                                        # Output: x / y = 3.75
print('x / y =',x/y)
                                        # Output: x // y = 3
print('x // y =',x//y)
                                        # Output: x ** y = 50625
print('x ** y =',x**y)
```

Out Put:

```
x + y = 19
x - y = 11
x * y = 60
x / y = 3.75
x // y = 3
x ** y = 50625
```

Comparison operators

Comparison operators are used to compare values. It either returns `True` or `False` according to the condition.

Comparison operators in Python

Operator	Meaning	Example
>	Greater that - True if left operand is greater than the right	x > y
<	Less that - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Example #2: Comparison operators in Python

```
x = 10
y = 12
                                # Output: x > y is False
print('x > y  is',x>y)
                                # Output: x < y is True
print('x < y  is',x<y)
                                # Output: x == y is False
print('x == y is',x==y)
                                        # Output: x != y is True
print('x != y is',x!=y)
                                        # Output: x >= y is False
print('x >= y is',x>=y)
                                        # Output: x <= y is True
print('x <= y is',x<=y)
```

Output:

```
x > y  is False
x < y  is True
x == y is False
x != y is True
x >= y is False
x <= y is True
```

Logical operators

Logical operators are the `and`, `or`, `not` operators.

Logical operators in Python

Operator	Meaning	Example
and	True if both the operands are true	x and y

or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Example #3: Logical Operators in Python

```
x = True
y = False
                                    # Output: x and y is False
print('x and y is',x and y)
                                    # Output: x or y is True
print('x or y is',x or y)
                                    # Output: not x is False
print('not x is',not x)
```

Output

```
x and y is False
x or y is True
not x is False
```

Bitwise operators

Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name.
For example, 2 is `10` in binary and 7 is `111`.
In the table below: Let `x` = 10 (`0000 1010` in binary) and `y` = 4 (`0000 0100` in binary)

Bitwise operators in Python

Operator	Meaning	Example

&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	X << 2 = 40 (0010 1000)

Assignment operators

Assignment operators are used in Python to assign values to variables.

`a = 5` is a simple assignment operator that assigns the value 5 on the right to the variable `a` on the left.
There are various compound operators in Python like `a += 5` that adds to the variable and later assigns the same. It is equivalent to `a = a + 5`.

Assignment operators in Python

Operator	Example	Equivatent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x – 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

Special operators

Python language offers some special type of operators like the identity operator or the membership operator. They are described below with examples.

Identity operators

is and is not are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Identity operators in Python

Operator	Meaning	Example
Is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Example #4: Identity operators in Python

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
                                        # Output: False

print(x1 is not y1)
                                        # Output: True

print(x2 is y2)
                                         # Output: False

print(x3 is y3)
```

Output:

```
False
True
False
```

- Here, we see that `x1` and `y1` are integers of same values, so they are equal as well as identical.
- Same is the case with `x2` and `y2` (strings).
- But `x3` and `y3` are list. They are equal but not identical.
- It is because interpreter locates them separately in memory although they are equal.

Membership operators

`in` and `not in` are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).
In a dictionary we can only test for presence of key, not the value.

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x

not in	True if value/variable is not found in the sequence	5 not in x

Example #5: Membership operators in Python

```
x = 'Hello world'
y = {1:'a',2:'b'}
                                        # Output: True
print('H' in x)
                                        # Output: True
print('hello' not in x)
                                        # Output: True
print(1 in y)
                                        # Output: False
print('a' in y)
True
True
True
False
```

Here, `'H'` is in `x` but `'hello'` is not present in `x` (remember, Python is case sensitive). Similary, `1` is key and `'a'` is the value in dictionary `y`. Hence, `'a' in y` returns `False`.

Python Operators Precedence Example

The following table lists all operators from highest precedence to lowest.

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)

* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'td>
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Operator precedence affects how an expression is evaluated.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first multiplies 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.

Example

```
a = 20
b = 10
c = 15
d = 5
e = 0
e = (a + b) * c / d       #( 30 * 15 ) / 5
print ("Value of (a + b) * c / d is ",  e)
e = ((a + b) * c) / d     # (30 * 15 ) / 5
print ("Value of ((a + b) * c) / d is ",  e)
e = (a + b) * (c / d);    # (30) * (15/5)
print ("Value of (a + b) * (c / d) is ",  e)
e = a + (b * c) / d;      #  20 + (150/5)
print ("Value of a + (b * c) / d is ",  e)
```

Output:

```
Value of (a + b) * c / d is  90.0
Value of ((a + b) * c) / d is  90.0
Value of (a + b) * (c / d) is  90.0
Value of a + (b * c) / d is  50.0
```

Python Input and Output Functions

Python provides methods that can be used to read and write data. Python also provides supports of reading and writing data to Files.

Python "print" Statement

"print" statement is used to print the output on the screen.

print statement is used to take string as input and place that string to standard output.

Whatever you want to display on output place that expression inside the inverted commas. The expression whose value is to printed place it without inverted commas.

Syntax:

```
1. print "expression" or print expression.
```

Exampple

```
1. a=10
2. print "Welcome to the world of Python"
3. print a
```

Output:

```
1. >>>
2. Welcome to the world of Python
3. 10
4. >>>
```

Input from Keyboard:

Python offers two built-in functions for taking input from user, given below:

1) input()

2) raw_input()

1) input() function: input() function is used to take input from the user. Whatever expression is given by the user, it is evaluated and result is returned back.

Python input() Syntax:

```
1. input("Expression")
```

Python input() Function Example

```
1. n=input("Enter your expression ");
2. print "The evaluated expression is ", n
```

Output:

```
>>>
Enter your expression 10*2
The evaluated expression is  20
>>>
```

Python raw_input()

2) raw_input()raw_input() function is used to take input from the user. It takes the input from the Standard input in the form of a string and reads the data from a line at once.

Syntax:

```
raw_input(?statement?)
```

Python raw_input() Example

```
n=raw_input("Enter your name ");
print "Welcome ", n
```

Output:

```
>>>
Enter your name Rajat
Welcome  Rajat
>>>
```

Program to enter details of an user and print them.

```
name=raw_input("Enter your name ")
math=float(raw_input("Enter your marks in Math"))
physics=float(raw_input("Enter your marks in Physics"))
chemistry=float(raw_input("Enter your marks in Chemistry"))
rollno=int(raw_input("Enter your Roll no"))
print "Welcome ",name
print "Your Roll no is ",rollno
print "Marks in Maths is ",math
print "Marks in Physics is ",physics
print "Marks in Chemistry is ",chemistry
print "Average marks is ",(math+physics+chemistry)/3
```

Output:

```
>>>
Enter your name rajat
Enter your marks in Math76.8
Enter your marks in Physics71.4
Enter your marks in Chemistry88.4
Enter your Roll no0987645672
Welcome  rajat
Your Roll no is  987645672
Marks in Maths is  76.8
Marks in Physics is  71.4
Marks in Chemistry is  88.4
Average marks is  78.8666666667
>>>
```

Python Comments

Python supports two types of comments:

1) Single lined comment:

In case user wants to specify a single line comment, then comment must start with ?#?

Eg:

```
# This is single line comment.
```

2) Multi lined Comment:

Multi lined comment can be given inside triple quotes.

eg:

```
''' This
    Is
    Multipline comment'''
```

eg:

```
#single line comment
print "Hello Python"
'''This is
multiline comment'''
```

What is Debugging?

- Programming is a complex process, and because it is done by human beings, it often leads to errors.
- Programming errors are called **bugs** and the process of tracking them down and correcting them is called **debugging**.
- Three kinds of errors can occur in a program and It is useful to distinguish between them in order to track them down more quickly.
  - ✓ Syntax Errors.
  - ✓ Runtime Errors.
  - ✓ Semantic Errors.

Syntax Errors

- Python can only execute a program if the program is syntactically correct; otherwise, the process fails and returns an error message. Syntax refers to the structure of a program and the rules about that structure.

- If there is a single syntax error anywhere in your program, Python will print an error message and quit, and you will not be able to run your program.

Runtime Errors

- The second type of error is a runtime error, so called because the error does not appear until you run the program.
- These errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened.

Semantic Errors

- The third type of error is the semantic error.
- If there is a semantic error in your program, it will run successfully, in the sense that the computer will not generate any error messages, but it will not do the right thing. It will do something else.
- Specifically, it will do what you told it to do.
- The problem is that the program you wrote is not the program you wanted to write.
- The meaning of the program (its semantics) is wrong.
- Identifying semantic errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing.

Prepared by
Ramesh Yajjala, Assistant Professor(c)
Dept. of CSE, IIIT-SKLM, RGUKT-AP

Thank you