



Python Programming Language

Prepared by: Mohamed Ayman

Machine Learning Engineer

spring 2018



facebook.com/sw.eng.MohamedAyman



sw.eng.MohamedAyman@gmail.com



wuzzuf.net/me/engMohamedAyman



codeforces.com/profile/Mohamed_Ayman



python

Basic Operators

Outline

- 1- Types of Operator
- 2- Arithmetic Operators
- 3- Comparison Operators
- 4- Assignment Operators
- 5- Bitwise Operators
- 6- Logical Operators
- 7- Membership Operators
- 8- Identity Operators
- 9- Operators Precedence

Types of Operator



- Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.
- Python programming language supports the following of operators
 - Arithmetic Operators
 - Comparison (Relational) Operators
 - Logical Operators
 - Bitwise Operators
 - Membership Operators
 - Identity Operators

Arithmetic Operators



Operator	Description	Example
+ Addition	Adds values on either side of the operator.	<code>a + b = 31</code>
- Subtraction	Subtracts right hand operand from left hand operand.	<code>a - b = -11</code>
* Multiplication	Multiplies values on either side of the operator	<code>a * b = 210</code>
/ Division	Divides left hand operand by right hand operand	<code>b / a = 2.1</code>
% Modulus	Divides left hand operand by right hand operand and returns remainder	<code>b % a = 1</code>
** Exponent	Performs exponential (power) calculation on operators	<code>a**b = 10 to the power 20</code>
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	<code>9//2 = 4</code> and <code>9.0//2.0 = 4.0</code>

Arithmetic Operators



```
a, b = 21, 10
c = a + b
print('Line 1 : a + b =', c)

c = a - b
print('Line 2 : a - b =', c)

c = a * b
print('Line 3 : a * b =', c)

c = a / b
print('Line 4 : a / b =', c)

c = a % b
print('Line 5 : a % b =', c)

a, b = 2, 3
c = a ** b
print('Line 6 : a ** b =', c)

a, b = 10, 5
c = a // b
print('Line 7 : a // b =', c)
```

```
Line 1 : a + b = 31
Line 2 : a - b = 11
Line 3 : a * b = 210
Line 4 : a / b = 2.1
Line 5 : a % b = 1
Line 6 : a ** b = 8
Line 7 : a // b = 2
```

Comparison Operators



Operator	Description	Example
<code>==</code>	If the values of two operands are equal, then the condition becomes true.	<code>(a == b)</code> is not true.
<code>!=</code>	If values of two operands are not equal, then condition becomes true.	<code>(a != b)</code> is true.
<code>></code>	If the value of left operand is greater than the value of right operand, then condition becomes true.	<code>(a > b)</code> is not true.
<code><</code>	If the value of left operand is less than the value of right operand, then condition becomes true.	<code>(a < b)</code> is true.
<code>>=</code>	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	<code>(a >= b)</code> is not true.
<code><=</code>	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	<code>(a <= b)</code> is true.

Comparison Operators



```
a, b = 21, 10

c = (a == b)
print('Line 1 : (a == b) =', c)

c = (a != b)
print('Line 2 : (a != b) =', c)

c = (a > b)
print('Line 3 : (a > b) =', c)

c = (a < b)
print('Line 4 : (a < b) =', c)

c = (a >= b)
print('Line 5 : (a >= b) =', c)

c = (a <= b)
print('Line 6 : (a <= b) =', c)
```

```
Line 1 : (a == b) = False
Line 2 : (a != b) = True
Line 3 : (a > b) = True
Line 4 : (a < b) = False
Line 5 : (a >= b) = True
Line 6 : (a <= b) = False
```


Assignment Operators



Operator	Description	Example
=	Assigns values from right side operands to left side operand	<code>c = a + b</code> assigns value of <code>a + b</code> into <code>c</code>
<code>+=</code> Add AND	It adds right operand to the left operand and assign the result to left operand	<code>c += a</code> is equivalent to <code>c = c + a</code>
<code>-=</code> Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	<code>c -= a</code> is equivalent to <code>c = c - a</code>
<code>*=</code> Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	<code>c *= a</code> is equivalent to <code>c = c * a</code>
<code>/=</code> Divide AND	It divides left operand with the right operand and assign the result to left operand	<code>c /= a</code> is equivalent to <code>c = c / a</code> <code>c /= a</code> is equivalent to <code>c = c / a</code>
<code>%=</code> Modulus AND	It takes modulus using two operands and assign the result to left operand	<code>c %= a</code> is equivalent to <code>c = c % a</code>
<code>**=</code> Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	<code>c **= a</code> is equivalent to <code>c = c ** a</code>
<code>//=</code> Floor Division	It performs floor division on operators and assign value to the left operand	<code>c //= a</code> is equivalent to <code>c = c // a</code>

Assignment Operators



```
a, b, c = 21, 10, 0

c += a
print('Line 1 : value of c is ', c)

c -= b
print('Line 2 : value of c is ', c)

c *= a
print('Line 3 : value of c is ', c)

c //= b
print('Line 4 : value of c is ', c)

c %= a
print('Line 5 : value of c is ', c)

c **= b
print('Line 6 : value of c is ', c)

c /= b
print('Line 7 : value of c is ', c)
```

```
Line 1 : value of c is 21
Line 2 : value of c is 11
Line 3 : value of c is 231
Line 4 : value of c is 23
Line 5 : value of c is 2
Line 6 : value of c is 1024
Line 7 : value of c is 102.4
```

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 operand's value is moved left by the number of bits specified by the right operand.	a << = 240 (means 1111 0000)
>> Binary Right Shift	The left operand's value is moved right by the number of bits specified by the right operand.	a >> = 15 (means 0000 1111)

Bitwise Operators



```
a, b = 60, 13
```

```
print('a =', a, ':', bin(a))
print('b =', b, ':', bin(b))
print()
```

```
c = a & b
print('Line 1 : result of a & b is ', c, ':', bin(c))
```

```
c = a | b
print('Line 2 : result of a | b is ', c, ':', bin(c))
```

```
c = a ^ b
print('Line 3 : result of a ^ b is ', c, ':', bin(c))
```

```
c = ~a
print('Line 4 : result of ~a is ', c, ':', bin(c))
```

```
c = b >> 3
print('Line 5 : result of b >> 3 is ', c, ':', bin(c))
```

```
c = b << 3
print('Line 6 : result of b << 3 is ', c, ':', bin(c))
```

```
a = 60 : 0b111100
b = 13 : 0b1101
```

```
Line 1 : result of a & b is 12 : 0b1100
Line 2 : result of a | b is 61 : 0b111101
Line 3 : result of a ^ b is 49 : 0b110001
Line 4 : result of ~a is -61 : -0b111101
Line 5 : result of b >> 3 is 1 : 0b1
Line 6 : result of b << 3 is 104 : 0b1101000
```

Logical Operators



Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is False.
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 True.

Logical Operators



```
a, b = True, False

c = a and b
print('Line 1 : result of a and b is ', c)

c = a or b
print('Line 2 : result of a or b is ', c)

c = not a
print('Line 3 : result of not a is ', c)

c = not b
print('Line 4 : result of not b is ', c)
```

```
Line 1 : result of a and b is False
Line 2 : result of a or b is True
Line 3 : result of not a is False
Line 4 : result of not b is True
```

Membership Operators



Operator	Description	Example
<code>in</code>	Evaluates to true, if it finds a variable in the specified sequence and false otherwise.	<code>x in y</code> , here <code>in</code> results in a 1 if <code>x</code> is a member of sequence <code>y</code> .
<code>not in</code>	Evaluates to true, if it does not find a variable in the specified sequence and false otherwise.	<code>x not in y</code> , here <code>not in</code> results in a 1 if <code>x</code> is not a member of sequence <code>y</code> .

Membership Operators



```
x = 'Hello world'
y = {'a':1, 'b':2, 'c':3}
z = [107, -301, "Hello World", [20, 61.7], 15.4]
```

```
c = ('H' in x)
print('Line 1 : H in', x, '\n', c)

c = ('Hello' not in x)
print('Line 2 : Hello not in', x, '\n', c)

c = ('b' in y)
print('Line 3 : b in', y, '\n', c)

c = (3 not in y)
print('Line 4 : 3 not in', y, '\n', c)
```

```
Line 1 : H in Hello world
True
Line 2 : Hello not in Hello world
False
Line 3 : b in {'a': 1, 'b': 2, 'c': 3}
True
Line 4 : 3 not in {'a': 1, 'b': 2, 'c': 3}
True
```


Membership Operators



```
x = 'Hello world'
y = {'a':1, 'b':2, 'c':3}
z = [107, -301, "Hello World", [20, 61.7], 15.4]

c = (-301 in z)
print('Line 5 :  -301 in', z, '\n', c)

c = (61.7 not in z)
print('Line 6 :  61.7 not in', z, '\n', c)

c = (61.7 in z[3])
print('Line 7 :  61.7 in', z[3], '\n', c)

c = ('W' not in z[2])
print('Line 8 :  W not in', z[2], '\n', c)

c = ('World' in z[2])
print('Line 9 :  World in', z[2], '\n', c)
```

```
Line 5 :  -301 in [107, -301, 'Hello World', [20, 61.7], 15.4]
True
Line 6 :  61.7 not in [107, -301, 'Hello World', [20, 61.7], 15.4]
True
Line 7 :  61.7 in [20, 61.7]
True
Line 8 :  W not in Hello World
False
Line 9 :  World in Hello World
True
```

Identity Operators



Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	<code>x is y</code> , here is results in 1 if <code>id(x)</code> equals <code>id(y)</code> .
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	<code>x is not y</code> , here is not results in 1 if <code>id(x)</code> is not equal to <code>id(y)</code> .

Identity Operators



```
a, b = 21, 21

print('a =', a, ':', id(a))
print('b =', b, ':', id(b))

c = a is b
print('Line 1 :  a is b ?', c)

c = a is not b
print('Line 2 :  a is not b ?', c)
```

```
a = 21 : 139861573581184
b = 21 : 139861573581184
Line 1 :  a is b ? True
Line 2 :  a is not b ? False
```

Identity Operators



```
a, b = 21, 20

print('a =', a, ':', id(a))
print('b =', b, ':', id(b))

c = a is b
print('Line 1 : a is b ?', c)

c = a is not b
print('Line 2 : a is not b ?', c)
```

```
a = 21 : 139861573581184
b = 20 : 139861573581152
Line 1 : a is b ? False
Line 2 : a is not b ? True
```

Operators Precedence



Operator	Description
<code>**</code>	Exponentiation (raise to the power)
<code>~ + -</code>	Complement, unary plus and minus (method names for the last two are <code>+@</code> and <code>-@</code>)
<code>* / % //</code>	Multiply, divide, modulo and floor division
<code>+ -</code>	Addition and subtraction
<code>>> <<</code>	Right and left bitwise shift
<code>&</code>	Bitwise 'AND'
<code>^ </code>	Bitwise exclusive 'OR' and regular 'OR'

Operators Precedence



Operator	Description
<code><= < > >=</code>	Comparison operators
<code><> == !=</code>	Equality operators
<code>= %= /= //= -= += *= **=</code>	Assignment operators
<code>is is not</code>	Identity operators
<code>in not in</code>	Membership operators
<code>not or and</code>	Logical operators

Operators Precedence



```
a, b, c, d = 20, 10, 15, 5

print('a, b, c, d = %d, %d, %d, %d\n' % (a, b, c, d))

e = a + b * c / d
print('Line 1 : a + b * c / d =', e)

e = (a + b) * c / d
print('Line 2 : (a + b) * c / d =', e)

e = a + b * (c / d)
print('Line 3 : a + b * (c / d) =', e)

e = a + b ** d * c
print('Line 4 : a + b ** d * c =', e)

e = a + b * c >> d
print('Line 5 : a + b * c >> d =', e)

e = a - b + c << d
print('Line 6 : a - b + c << d =', e)
```

```
a, b, c, d = 20, 10, 15, 5

Line 1 : a + b * c / d = 50.0
Line 2 : (a + b) * c / d = 90.0
Line 3 : a + b * (c / d) = 50.0
Line 4 : a + b ** d * c = 1500020
Line 5 : a + b * c >> d = 5
Line 6 : a - b + c << d = 800
```

Operators Precedence



```
a, b, c, d = True, False, False, True

print('a, b, c, d = %d, %d, %d, %d\n' % (a, b, c, d))

e = a and b or c and d
print('Line 1 : a and b or c and d =', e)

e = a or b and c or d
print('Line 2 : a or b and c or d =', e)

e = a and not b or c and d
print('Line 3 : a and not b or c and d =', e)

e = a or b and not c or d
print('Line 4 : a or b and not c or d =', e)
```

```
a, b, c, d = 1, 0, 0, 1
```

```
Line 1 : a and b or c and d = False
```

```
Line 2 : a or b and c or d = True
```

```
Line 3 : a and not b or c and d = True
```

```
Line 4 : a or b and not c or d = True
```


Outline

- ☒ 1- Types of Operator
- ☒ 2- Arithmetic Operators
- ☒ 3- Comparison Operators
- ☒ 4- Assignment Operators
- ☒ 5- Bitwise Operators
- ☒ 6- Logical Operators
- ☒ 7- Membership Operators
- ☒ 8- Identity Operators
- ☒ 9- Operators Precedence



Practice

Problem 1

Solve Equation



- Take as an input x and get the result from this equation
- $F(x) = (x-1)^3 + (x+1)^2 + 2x + 7$; such that $x \geq 0$
- Test Cases:

0 7	4 67	2 21	10 877
1 13	5 117	3 37	20 7347

Problem 1 Solution

Solve Equation



```
x = int(input())  
y = (x-1)**3 + (x+1)**2 + 2*x + 7  
print(y)
```

Problem 2

Solve Equation Cycle



- Take as an input x and get the result from this equation between $[0,10]$
- $F(x) = (x-1)^3 + (x+1)^2 + 2x + 7$; such that $x \geq 0$
- Test Cases:

0 7	4 1	2 10	10 8
1 2	5 7	3 4	20 10

Problem 2 Solution

Solve Equation Cycle



```
x = int(input())  
y = (x-1)**3 + (x+1)**2 + 2*x + 7  
print(y%11)
```

Problem 3

Domino piling



- You are given a rectangular board of $M \times N$ squares.
- You are given an unlimited number of standard domino pieces of 2×1 squares.
- You are asked to place as many dominoes as possible on the board so as to meet the following conditions:
 - Each domino completely covers two squares.
 - No two dominoes overlap.
 - Each domino lies entirely inside the board
- Find the maximum number of dominoes, which can be placed under these restrictions such that $N, M > 0$
- Test Cases:

4 4 8	5 7 17
5 5 12	4 7 14
4 6 12	5 6 15

Problem 3 Solution

Domino piling



```
n, m = map(int, input().split(' '))  
print(n*m//2)
```


Problem 4

Theatre Square



- Theatre Square has a rectangular shape with the size $n \times m$ meters.
- A decision was taken to pave the Square with square granite flagstones. Each flagstone is of the size $a \times a$.
- What is the least number of flagstones needed to pave the Square?
- It's allowed to cover the surface larger than the Theatre Square, but the Square has to be covered.
- It's not allowed to break the flagstones.
- Input will be: $n \ m \ a$ such that $n, m, a > 0$
- Test Cases:

4 4 2 4	5 5 3 4
5 5 2 9	5 6 3 4
4 4 3 4	5 6 2 9

Problem 4 Solution

Theatre Square



```
n, m, k = map(int, input().split(' '))  
w = (n+k-1) // k  
h = (m+k-1) // k  
print(w*h)
```

Problem 5

Calculating Function



- For a positive integer n let's define a function f :
- $f(n) = -1 + 2 - 3 + \dots + (-1)^n n$
- Your task is to calculate $f(n)$ for a given n , such that $n > 0$ and integer
- Test Cases:

1 -1	6 3	4 2	9 -5	3 -2
2 1	7 -4	5 -3	10 5	8 4

Problem 5 Solution

Calculating Function



```
n = int(input())  
print( n//2 - n*(n%2) )
```

```
n = int(input())  
print( (n+1)//2 * (-1) ** (n%2) )
```



Questions ?

References



- | | | |
|------|---------------------------------------|---|
| [01] | Online Courses YouTube playlists | http://bit.ly/2EcbLvZ |
| [02] | Programiz Tutorial | https://bit.ly/2paX2xA |
| [03] | Python 3 Tutorial Point PDF | http://bit.ly/2xcqp66 |
| [04] | Python Guru Tutorial | https://bit.ly/28TtLRr |
| [05] | Python for Everybody PDF | http://bit.ly/2gSWkSS |
| [06] | A Byte of Python PDF | http://bit.ly/1l9Yqp9 |
| [07] | Python Course Tutorial | https://bit.ly/2l076hU |
| [08] | Python Basics Tutorial PDF | http://bit.ly/1jMlluB |
| [09] | Learning to Program Using Python | http://bit.ly/2zO4f91 |
| [10] | Think Python PDF | http://bit.ly/2cTgLIk |
| [11] | Python 3 Patterns, Recipes and Idioms | http://bit.ly/2by2bsN |