

# Python Programming Language

Prepared by: Mohamed Ayman

Algorithm Engineer at Valeo | Machine Learning Researcher  
spring 2019



[sw.eng.MohamedAyman@gmail.com](mailto:sw.eng.MohamedAyman@gmail.com)



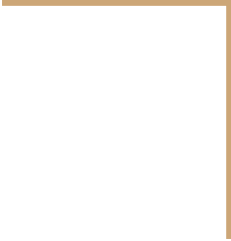
[facebook.com/cs.MohamedAyman](https://facebook.com/cs.MohamedAyman)



[linkedin.com/in/cs-MohamedAyman](https://linkedin.com/in/cs-MohamedAyman)




[codeforces.com/profile/Mohamed\\_Ayman](https://codeforces.com/profile/Mohamed_Ayman)



# Lecture 3

## Basic Operations



# Course Roadmap

---



## Part 1: Python Basics and Functions

Lecture 1: Python Overview

Lecture 2: Variable Types

**Lecture 3: Basic Operations**

Lecture 4: Conditions

Lecture 5: Loops

Lecture 6: Functions

## Part 2: Python Collections and Strings

Lecture 7: Strings

Lecture 8: Lists

Lecture 9: Tuples

Lecture 10: Dictionaries

Lecture 11: Sets

Lecture 12: Numbers

# Lecture Agenda

**We will discuss in this lecture the following topics**

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



Let's  
**STARTUP**

# 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
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

# Lecture Agenda

---



**Section 1: Arithmetic Operators**

Section 2: Comparison Operators

Section 3: Assignment Operators

Section 4: Bitwise Operators

Section 5: Logical Operators

Section 6: Membership Operators

Section 7: Identity Operators

Section 8: Operators Precedence

# Arithmetic Operators



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

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	$a + b = 31$
- Subtraction	Subtracts right hand operand from left hand operand.	$a - b = -11$
* Multiplication	Multiplies values on either side of the operator	$a * b = 210$
/ Division	Divides left hand operand by right hand operand	$b / a = 2.1$
% Modulus	Divides left hand operand by right hand operand and returns remainder	$b \% a = 1$
** 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.	$9 // 2 = 4$ and $9.0 // 2.0 = 4.0$



# Arithmetic Operators

---



Example:

```
x, y = 21, 10
print(x + y)
print(x - y)
print(x * y)
print(x / y)
print(x % y)
x, y = 2, 3
print(x ** y)
x, y = 11, 5
print(x // y)
```

Output:

```
31
11
210
2.1
1
8
2
```

# Lecture Agenda

---



- ✓ Section 1: Arithmetic Operators
- Section 2: Comparison Operators**
- Section 3: Assignment Operators
- Section 4: Bitwise Operators
- Section 5: Logical Operators
- Section 6: Membership Operators
- Section 7: Identity Operators
- Section 8: Operators Precedence

# Comparison Operators



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

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>&gt;</code>	If the value of left operand is greater than the value of right operand, then condition becomes true.	<code>(a &gt; b)</code> is not true.
<code>&lt;</code>	If the value of left operand is less than the value of right operand, then condition becomes true.	<code>(a &lt; b)</code> is true.
<code>&gt;=</code>	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	<code>(a &gt;= b)</code> is not true.
<code>&lt;=</code>	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	<code>(a &lt;= b)</code> is true.

# Comparison Operators

---



Example:

```
x, y = 21, 10
print(x == y)
print(x != y)
print(x > y)
print(x < y)
print(x >= y)
print(x <= y)
```

Output:

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

# Lecture Agenda

---



✓ Section 1: Arithmetic Operators

✓ Section 2: Comparison Operators

**Section 3: Assignment Operators**

Section 4: Bitwise Operators

Section 5: Logical Operators

Section 6: Membership Operators

Section 7: Identity Operators

Section 8: Operators Precedence

# Assignment Operators



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

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



Example:

```
x, y, z = 21, 10, 0
```

```
z += x
```

```
print(z)
```

```
z -= y
```

```
print(z)
```

```
z *= x
```

```
print(z)
```

```
z /= y
```

```
print(z)
```

```
z %= x
```

```
print(z)
```

```
z **= y
```

```
print(z)
```

```
z /= y
```

```
print(z)
```

Output:

21

11

231

23

2

1024

102.4

# Lecture Agenda

---



- ✓ Section 1: Arithmetic Operators
- ✓ Section 2: Comparison Operators
- ✓ Section 3: Assignment Operators

## **Section 4: Bitwise Operators**

Section 5: Logical Operators

Section 6: Membership Operators

Section 7: Identity Operators

Section 8: Operators Precedence



# Recall: Truth Table

---



**and**

0	0	0
0	1	0
1	0	0
1	1	1

**or**

0	0	0
0	1	1
1	0	1
1	1	1

**not**

0	1
1	0

**xor**

0	0	0
0	1	1
1	0	1
1	1	0

# Bitwise Operators



- Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name.

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



Example:

```
x, y = 60, 13
print(x, bin(x))
print(y, bin(y))
print(x & y)
print(x | y)
print(x ^ y)
print(~x)
```

Output:

```
60 0b111100
13 0b1101
12
61
49
-61
```

Binary:

```
111100
001101
001100
111101
110001
-111101

001101
001
001101000
```

```
print(y, bin(y))
print(y >> 3)
print(y << 3)
```

```
13 0b1101
1
104
```

# Lecture Agenda

---



- ✓ Section 1: Arithmetic Operators
- ✓ Section 2: Comparison Operators
- ✓ Section 3: Assignment Operators
- ✓ Section 4: Bitwise Operators

## **Section 5: Logical Operators**

Section 6: Membership Operators

Section 7: Identity Operators

Section 8: Operators Precedence

# Logical Operators



- Logical operators are the and, or, not 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

---



Example:

```
x, y = True, False
print(x and y)
print(x or y)
print(not x)
print(not y)
```

Output:

```
False
True
False
True
```

# Lecture Agenda

---



- ✓ Section 1: Arithmetic Operators
- ✓ Section 2: Comparison Operators
- ✓ Section 3: Assignment Operators
- ✓ Section 4: Bitwise Operators
- ✓ Section 5: Logical Operators

**Section 6: Membership Operators**

Section 7: Identity Operators

Section 8: Operators Precedence

# 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).

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



# Membership Operators



Example:

```
x = 'Hello World'
y = {'a':1, 'b':2, 'c':3}
z = [17, -31, 'Hello World', [20, 61], (15, -9)]
print('H' in x)
print('Hello' in x)
print('b' in y)
print(3 not in y)
print(-31 in z)
print(61 not in z)
print(61 in z[3])
print('W' in z[2])
print('World' in z[2])
print('W' not in z)
```

Output:

```
True
True
True
True
True
True
True
True
True
True
```

# Lecture Agenda

---



- ✓ Section 1: Arithmetic Operators
- ✓ Section 2: Comparison Operators
- ✓ Section 3: Assignment Operators
- ✓ Section 4: Bitwise Operators
- ✓ Section 5: Logical Operators
- ✓ Section 6: Membership Operators

**Section 7: Identity Operators**

Section 8: Operators Precedence

# 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.

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

- Python `id()` function returns the “identity” of the object. The identity of an object is an integer, which is guaranteed to be unique and constant for this object during its lifetime.
- Two objects with non-overlapping lifetimes may have the same `id()` value. In CPython implementation, this is the address of the object in memory.
- Python cache the `id()` value of commonly used data types, such as string, integer, tuples etc. So you might find that multiple variables refer to the same object and have same `id()` value if their values are same.
- Caching can work only with immutable objects, notice that integer, string, tuples are immutable. So Python implementation can use caching to save memory space and improve performance.

# Identity Operators



## Example:

```
x, y = 10, 10
print(x, id(x))
print(y, id(y))
print(x is y)
```

## Output:

```
10 140457019307584
10 140457019307584
True
```

```
x, y = 21, 10
print(x, id(x))
print(y, id(y))
print(x is not y)
```

```
21 140457019307936
10 140457019307584
True
```

```
x, y = 'abc', 'abc'
print(x, id(x))
print(y, id(y))
print(x is y)
```

```
abc 140514178032752
abc 140514178032752
True
```

```
x, y = 'abc', 'bca'
print(x, id(x))
print(y, id(y))
print(x is not y)
```

```
abc 140514178032752
bca 140514101786544
True
```

# Identity Operators



## Example:

```
x, y = (1, 2, 3), (1, 2, 3)
print(x, id(x))
print(y, id(y))
print(x is y)
```

## Output:

```
(1, 2, 3) 139987257611040
(1, 2, 3) 139987257128272
False
```

```
x, y = (1, 2, 3), (2, 3, 1)
print(x, id(x))
print(y, id(y))
print(x is y)
```

```
(1, 2, 3) 139987257140960
(2, 3, 1) 139987257142560
False
```

```
x, y = [1, 2, 3], [1, 2, 3]
print(x, id(x))
print(y, id(y))
print(x is y)
```

```
[1, 2, 3] 140708366561680
[1, 2, 3] 140708425321520
False
```

```
x, y = [1, 2, 3], [2, 3, 1]
print(x, id(x))
print(y, id(y))
print(x is y)
```

```
[1, 2, 3] 140708363738720
[2, 3, 1] 140708364030352
False
```

# Identity Operators



## Example:

```
x = {'a':1, 'b':2, 'c':3}
y = {'a':1, 'b':2, 'c':3}
print(x, id(x))
print(y, id(y))
print(x is y)
```

```
x = {'a':1, 'b':2, 'c':3}
y = {'a':2, 'b':3, 'c':1}
print(x, id(x))
print(y, id(y))
print(x is y)
```

```
x, y = {1, 2, 3}, {1, 2, 3}
print(x, id(x))
print(y, id(y))
print(x is y)
```

```
x, y = {1, 2, 3}, {2, 3, 1}
print(x, id(x))
print(y, id(y))
print(x is y)
```

## Output:

```
{'a': 1, 'b': 2, 'c': 3} 139737533283936
{'a': 1, 'b': 2, 'c': 3} 139737465145856
False
```

```
{'a': 1, 'b': 2, 'c': 3} 139737464980656
{'a': 2, 'b': 3, 'c': 1} 139737464980416
False
```

```
{1, 2, 3} 139710539334432
{1, 2, 3} 139710539334192
False
```

```
{1, 2, 3} 139710539335392
{1, 2, 3} 139710539334912
False
```

# Lecture Agenda

---



- ✓ Section 1: Arithmetic Operators
- ✓ Section 2: Comparison Operators
- ✓ Section 3: Assignment Operators
- ✓ Section 4: Bitwise Operators
- ✓ Section 5: Logical Operators
- ✓ Section 6: Membership Operators
- ✓ Section 7: Identity Operators
- Section 8: Operators Precedence**



# Operators Precedence



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

# Operators Precedence



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

# Operators Precedence



Example:

```
a, b, c, d = 20, 10, 15, 5
print(a + b * c / d)
print((a + b) * c / d)
print(a + b * (c / d))
print(a + b ** d * c)
print(a + b * c >> d)
print(a + b * c << d)
a, b, c, d = True, False, False, True
print(a and b or c and d)
print(a or b and c or d)
print(a and not b or c and d)
print(a or b and not c or d)
```

Output:

```
50.0
90.0
50.0
1500020
5
5440

False
True
True
True
```

# Lecture Agenda

---



- ✓ Section 1: Arithmetic Operators
- ✓ Section 2: Comparison Operators
- ✓ Section 3: Assignment Operators
- ✓ Section 4: Bitwise Operators
- ✓ Section 5: Logical Operators
- ✓ Section 6: Membership Operators
- ✓ Section 7: Identity Operators
- ✓ Section 8: Operators Precedence

# Quiz



- Which of the following statement is true according to python basic operations?

- ☐ `-2 ** 4 == 16`
- ☐ `-2 ** 4 == -16`
- ☐ `None is None`
- ☐ `None is not None`
- ☐ `1 << 1 == 2 << 0`
- ☐ `1 << 1 != 2 << 0`
- ☐ `None is []`
- ☐ `None is not []`

# Quiz Solution



- Which of the following statement is true according to python basic operations?

- ☐ `-2 ** 4 == 16`
- ☒ `-2 ** 4 == -16`
- ☒ `None is None`
- ☐ `None is not None`
- ☒ `1 << 1 == 2 << 0`
- ☐ `1 << 1 != 2 << 0`
- ☐ `None is []`
- ☒ `None is not []`

# Quiz



- Which of the following statement is true according to python basic operations?

☐ `(1 << 4) - (8 >> 1) * 3 - 4 == 0`

☐ `(1 << 4) - (8 >> 1) * 3 - 4 != 0`

☐ `bool(None or 5) == True`

☐ `bool(None or 5) == False`

☐ `bool(3 and -7) == True`

☐ `bool(3 and -7) == False`

☐ `bool(not 0 or None) == True`

☐ `bool(not 0 or None) == False`

# Quiz Solution



- Which of the following statement is true according to python basic operations?

☒ `(1 << 4) - (8 >> 1) * 3 - 4 == 0`

☐ `(1 << 4) - (8 >> 1) * 3 - 4 != 0`

☒ `bool(None or 5) == True`

☐ `bool(None or 5) == False`

☒ `bool(3 and -7) == True`

☐ `bool(3 and -7) == False`

☒ `bool(not 0 or None) == True`

☐ `bool(not 0 or None) == False`







# Practice





# Problem 1

---



- Find the output of the following equation using python basic operations, such that the input number data type will be float and positive.
- Equation:  $(x - 1)^3 + (x + 1)^2 + 2x + 7$
- Test Cases:

Test Case 1

1

13

Test Case 2

3

37

Test Case 3

6

193

# Problem Solution



- Find the output of the following equation using python basic operations, such that the input number data type will be float and positive.
- Equation:  $(x - 1)^3 + (x + 1)^2 + 2x + 7$
- Test Cases:

Test Case 1

1

13

Test Case 2

3

37

Test Case 3

6

193

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

# Problem 2

---



- Find the square root of a number using python basic operations, such that the data type of input number will be floating and positive.
- Equation:  $\sqrt{x}$
- Test Cases:

Test Case 1

4

2.0

Test Case 2

9

3.0

Test Case 3

16

4.0

# Problem Solution

---



- Find the square root of a number using python basic operations, such that the data type of input number will be floating and positive.
- Equation:  $\sqrt{x}$
- Test Cases:

Test Case 1

4

2.0

Test Case 2

9

3.0

Test Case 3

16

4.0

```
x = float(input())  
print(x ** 0.5)
```

# Problem 3



- Solve Quadratic Equation using python basic operations, such that the data type of input numbers will be floating and positive, you are given a, b, c by the following equation an input:

- Quadratic Equation:  $ax^2 + bx + c = 0$  Hint:  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

- Test Cases:

Test Case 1

1 5 6

-3.0 -2.0

Test Case 2

1 -5 6

2.0 3.0

Test Case 3

1 -6 8

2.0 4.0



# Problem Solution



- Solve Quadratic Equation using python basic operations, such that the data type of input numbers will be floating and positive, you are given a, b, c by the following equation an input:

- Quadratic Equation:  $ax^2 + bx + c = 0$  Hint:  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

- Test Cases:

Test Case 1

1 5 6

-3.0 -2.0

Test Case 2

1 -5 6

2.0 3.0

Test Case 3

1 -6 8

2.0 4.0

```
a, b, c = map(float, input().split())
d = (b ** 2) - (4 * a * c)
sol1 = (-b - (d ** 0.5)) / (2 * a)
sol2 = (-b + (d ** 0.5)) / (2 * a)
print(sol1, sol2)
```

# Problem 4

---



- Swap two numbers using python basic operations, then print them, such that the data type of input numbers will be integer.
- Test Cases:

Test Case 1

2 3

3 2

Test Case 2

-6 4

4 -6

Test Case 3

7 -5

-5 7

# Problem Solution

---



- Swap two numbers using python basic operations, then print them, such that the data type of input numbers will be integer.
- Test Cases:

Test Case 1

2 3

3 2

Test Case 2

-6 4

4 -6

Test Case 3

7 -5

-5 7

```
x, y = map(int, input().split())  
x, y = y, x  
print(x, y)
```

# Problem 5

---



- Convert Kilometers to Miles using python basic operations, such that the data type of input number will be floating.

Hint: **1 KM = 0.621371 Mile**

- Test Cases:

Test Case 1

5.5

3.4175405

Test Case 2

2.7

1.6777017

Test Case 3

6.3

3.9146373

# Problem Solution

---



- Convert Kilometers to Miles using python basic operations, such that the data type of input number will be floating.

Hint: **1 KM = 0.621371 Mile**

- Test Cases:

Test Case 1

5.5

3.4175405

Test Case 2

2.7

1.6777017

Test Case 3

6.3

3.9146373

```
k = float(input())  
print(k * 0.621371)
```

# Problem 6

---



- Convert Fahrenheit To Celsius using python basic operations, such that the data type of input number will be floating.

Hint:  $F = 1.8 C + 32$

- Test Cases:

Test Case 1

98.6

36.999

Test Case 2

100

37.778

Test Case 3

32

0.0

# Problem Solution

---



- Convert Fahrenheit To Celsius using python basic operations, such that the data type of input number will be floating.

Hint:  $F = 1.8 C + 32$

- Test Cases:

Test Case 1

98.6

36.999

Test Case 2

100

37.778

Test Case 3

32

0.0

```
f = float(input())  
print((f - 32) / 1.8)
```

# Problem 7

---



- Calculate the summation of a series by the following pattern  $-1, 2, -3, 4, -5, 6, \dots, (-1)^n \cdot n$  using python basic operations, such that the data type of input number  $n$  will be integer and positive.
- Test Cases:

Test Case 1	Test Case 2	Test Case 3	Test Case 4
1	3	6	10
-1	-2	3	5



# Problem Solution



- Calculate the summation of a series by the following pattern  $-1, 2, -3, 4, -5, 6, \dots, (-1)^n n$  using python basic operations, such that the data type of input number  $n$  will be integer and positive.
- Test Cases:

Test Case 1

1

-1

Test Case 2

3

-2

Test Case 3

6

3

Test Case 4

10

5

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



# Assignment



# Basic Operator Problems

---



- [01] CF-Round 340: <http://codeforces.com/problemset/problem/617/A>
- [02] CF-Round 304: <http://codeforces.com/problemset/problem/546/A>
- [03] CF-Round 395: <http://codeforces.com/problemset/problem/764/A>
- [04] CF-Round 107: <http://codeforces.com/problemset/problem/151/A>
- [05] CF-Round 256: <http://codeforces.com/problemset/problem/448/A>
- [06] CF-Round 327: <http://codeforces.com/problemset/problem/591/A>
- [07] CF-Round 332: <http://codeforces.com/problemset/problem/599/A>
- [08] CF-Round 125: <http://codeforces.com/problemset/problem/199/A>
- [09] CF-Round 350: <http://codeforces.com/problemset/problem/670/A>
- [10] CF-Round 123: <http://codeforces.com/problemset/problem/195/A>

# Basic Operator Problems Solution

---



- [01] CF-Round 340: <http://codeforces.com/contest/617/submission/26588156>
- [02] CF-Round 304: <http://codeforces.com/contest/546/submission/26636084>
- [03] CF-Round 395: <http://codeforces.com/contest/764/submission/26777792>
- [04] CF-Round 107: <http://codeforces.com/contest/151/submission/30904768>
- [05] CF-Round 256: <http://codeforces.com/contest/448/submission/30730442>
- [06] CF-Round 327: <http://codeforces.com/contest/591/submission/30936968>
- [07] CF-Round 332: <http://codeforces.com/contest/599/submission/30728573>
- [08] CF-Round 125: <http://codeforces.com/contest/199/submission/26970850>
- [09] CF-Round 350: <http://codeforces.com/contest/670/submission/30936450>
- [10] CF-Round 123: <http://codeforces.com/contest/195/submission/26694589>



DO  
MORE.