Python Programming Language

Prepared by: Mohamed Ayman

Algorithm Engineer at Valeo

Deep Learning Researcher and Teaching Assistant

at The American University in Cairo (AUC)

spring 2020

sw.eng.MohamedAyman@gmail.com

linkedin.com/in/cs-MohamedAyman

github.com/cs-MohamedAyman

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

Lecture Agenda

We will discuss in this lecture the following topics

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


Let's
START UP

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

Lecture Agenda

Section 1: Arithmetic Operators

Section 2: Comparison Operators

Section 3: Bitwise Operators

Section 4: Assignment 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
+ Addition	Adds values on either side of the operator.
- Subtraction	Subtracts right hand operand from left hand operand.
* Multiplication	Multiplies values on either side of the operator
/ Division	Divides left hand operand by right hand operand
% Modulus	Divides left hand operand by right hand operand and returns remainder
** Exponent	Performs exponential (power) calculation on operators
// Division	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.

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: Bitwise Operators
- Section 4: Assignment 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
==	If the values of two operands are equal, then the condition becomes true.
!=	If values of two operands are not equal, then condition becomes true.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes 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)

x, y = 'a', 'd'
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

False
True
False
True
False
True
```

Lecture Agenda

- ✓ Section 1: Arithmetic Operators
- ✓ Section 2: Comparison Operators
- **Section 3: Bitwise Operators**
- Section 4: Assignment 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
& Binary AND	Operator copies a bit to the result, if it exists in both operands
| Binary OR	It copies a bit, if it exists in either operand.
^ Binary XOR	It copies the bit, if it is set in one operand but not both.
~ Binary Complement	It is unary and has the effect of 'flipping' bits.
<< Binary Left Shift	The left operand's value is moved left by the number of bits specified by the right operand.
>> Binary Right Shift	The left operand's value is moved right by the number of bits specified by the right operand.

Bitwise Operators

Example:	Output:	Binary:
`x, y = 60, 13`		
`print(x, bin(x))`	60 0b111100	111100
`print(y, bin(y))`	13 0b1101	001101
`print(x & y)`	12	001100
`print(x | y)`	61	111101
`print(x ^ y)`	49	110001
`print(~x)`	-61	-111101
`print(y, bin(y))`	13 0b1101	001101
`print(y >> 3)`	1	001
`print(y << 3)`	104	001101000

$$x \ll y = x * 2^y \qquad \textbf{Binary Left shift}$$

$$x \gg y = x // 2^y \qquad \textbf{Binary Right shift}$$

Lecture Agenda

- ✔ Section 1: Arithmetic Operators
- ✔ Section 2: Comparison Operators
- ✔ Section 3: Bitwise Operators
- **Section 4: Assignment 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
=	Assign value of right side of expression to left side operand
+=	Add AND: Add right side operand with left side operand and then assign to left operand
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand
/=	Divide AND: Divide left operand with right operand and then assign to left operand
%=	Modulus AND: Takes modulus using left and right operands and assign result to left operand
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left

Assignment Operators

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

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

Assignment Operators

Example:	Output:	Equivalent Syntax:
`x, y, z = 21, 10, 0`		
`z += x`		`z = z + x`
`print(z)`	21	
`z -= y`		`z = z - y`
`print(z)`	11	
`z *= x`		`z = z * x`
`print(z)`	231	
`z //= y`		`z = z // y`
`print(z)`	23	
`z %= x`		`z = z % x`
`print(z)`	2	
`z **= y`		`z = z ** y`
`print(z)`	1024	
`z /= y`		`z = z / y`
`print(z)`	102.4	

Lecture Agenda

- ✔ Section 1: Arithmetic Operators
- ✔ Section 2: Comparison Operators
- ✔ Section 3: Bitwise Operators
- ✔ Section 4: Assignment 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
and Logical AND	If both the operands are true then condition becomes true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.
not Logical NOT	Used to reverse the logical state of its operand.

Logical Operators

Example:

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

a, b, c, d = 10, 5, 7, 3
print(a > b and c < d)
print(a > b or c < d)
print(not a > b)
print(not c < d)
```

Output:

```
False
True
False
True

False
True
False
True
```

Lecture Agenda

- ✔ Section 1: Arithmetic Operators
- ✔ Section 2: Comparison Operators
- ✔ Section 3: Bitwise Operators
- ✔ Section 4: Assignment 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: Bitwise Operators
- ✔ Section 4: Assignment 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
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

Python id()

- 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.

Example:

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

Output:

```
10 140457019307584
```

Identity Operators

Example:

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

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

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

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

Output:

```
10 140457019307584
10 140457019307584
True

21 140457019307936
10 140457019307584
True

abc 140514178032752
abc 140514178032752
True

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)

x, y = (1, 2, 3), (2, 3, 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:

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

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

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

[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: Bitwise Operators
- ✓ Section 4: Assignment Operators
- ✓ Section 5: Logical Operators
- ✓ Section 6: Membership Operators
- ✓ Section 7: Identity Operators
- **Section 8: Operators Precedence**

Operators Precedence

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

Operators Precedence

	Operator	Description
8	<= < > >=	Comparison operators
9	== !=	Equality operators
10	= %= /= //= -= **= += *=	Assignment operators
11	is is not	Identity operators
12	in not in	Membership operators
13	not	Logical operators
14	and	Logical operators
15	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
```

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

- [x] `(1 << 4) - (8 >> 1) * 3 - 4 == 0`
- [ ] `(1 << 4) - (8 >> 1) * 3 - 4 != 0`
- [x] `bool(None or 5) == True`
- [ ] `bool(None or 5) == False`
- [x] `bool(3 and -7) == True`
- [ ] `bool(3 and -7) == False`
- [x] `bool(not 0 or None) == True`
- [ ] `bool(not 0 or None) == False`

Lecture Agenda

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

Practice

Problem: Simple equation

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

Test Case 1	Test Case 2	Test Case 3
1	3	6
13	37	193

Problem Solution

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

Test Case 1	Test Case 2	Test Case 3
1	3	6
13	37	193

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

Problem: Square root

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

Test Case 1	Test Case 2	Test Case 3
4	9	16
2.0	3.0	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 float and positive.
- Equation: $\sqrt{x}$
- Test Cases:

Test Case 1	Test Case 2	Test Case 3
4	9	16
2.0	3.0	4.0

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

Problem: Swap numbers

- 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	Test Case 2	Test Case 3
2 3	-6 4	7 -5
3 2	4 -6	-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	Test Case 2	Test Case 3
2 3	-6 4	7 -5
3 2	4 -6	-5 7

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

Problem: Kilometer conversion

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

Hint: $1\ KM = 0.621371\ Mile$

- Test Cases:

Test Case 1	Test Case 2	Test Case 3
5.5	2.7	6.3
3.4175405	1.6777017	3.9146373

Problem Solution

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

Hint: $1\ KM = 0.621371\ Mile$

- Test Cases:

Test Case 1	Test Case 2	Test Case 3
5.5	2.7	6.3
3.4175405	1.6777017	3.9146373

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

Problem: Fahrenheit conversion

- Convert Fahrenheit To Celsius using python basic operations, such that the data type of input number will be float and positive. Hint: $F = 1.8\,C + 32$
- Test Cases:

Test Case 1	Test Case 2	Test Case 3
98.6	100	32
36.999	37.778	0.0

Problem Solution

- Convert Fahrenheit To Celsius using python basic operations, such that the data type of input number will be float and positive. Hint: $F = 1.8\,C + 32$
- Test Cases:

Test Case 1	Test Case 2	Test Case 3
98.6	100	32
36.999	37.778	0.0

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

Problem: Quadratic equation

- Solve Quadratic Equation using python basic operations, such that the data type of input numbers will be float 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:

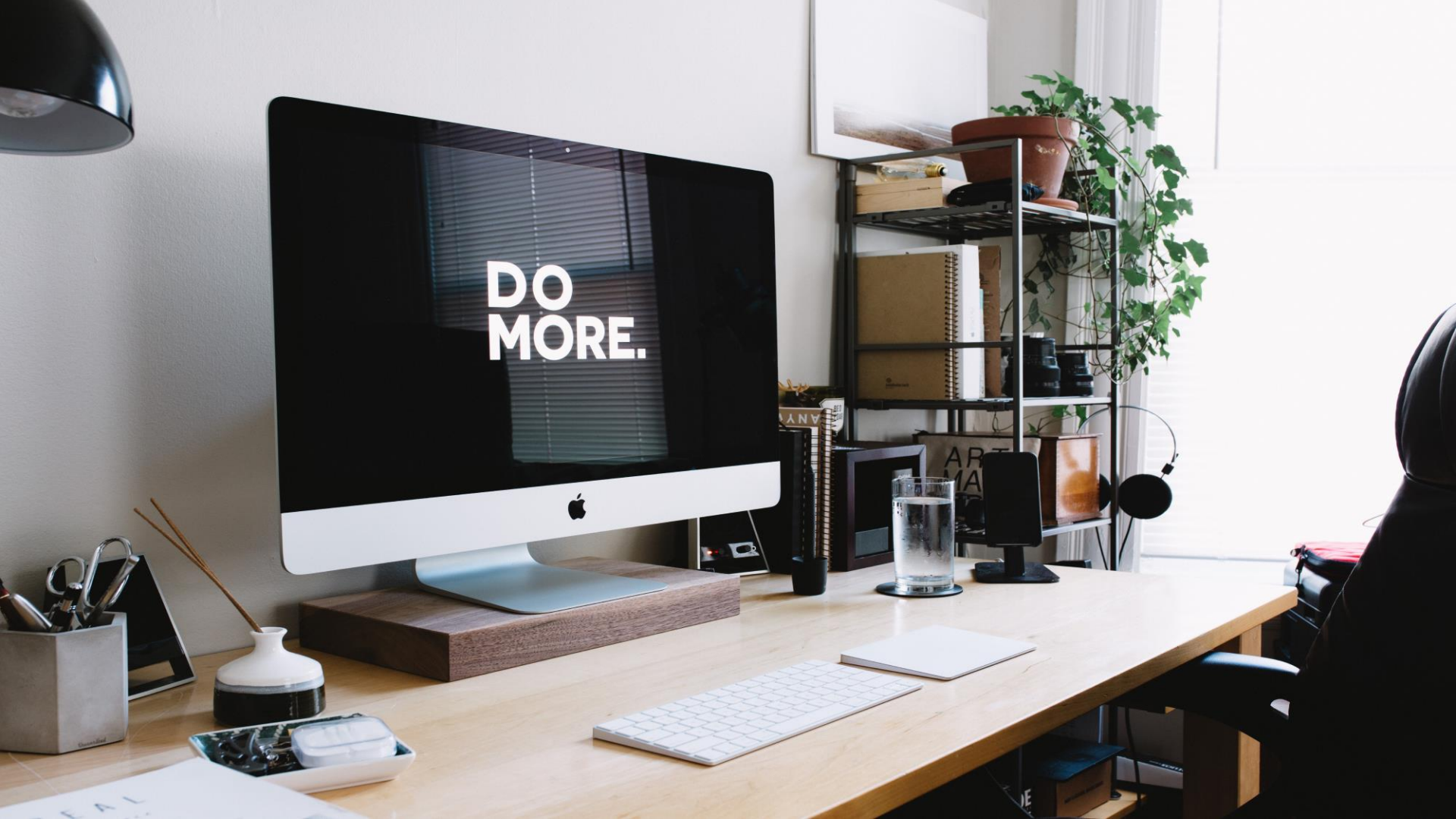
Test Case 1	Test Case 2	Test Case 3
1 5 6	1 -5 6	1 -6 8
-3.0 -2.0	2.0 3.0	2.0 4.0

Problem Solution

- Solve Quadratic Equation using python basic operations, such that the data type of input numbers will be float 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	Test Case 2	Test Case 3
1 5 6	1 -5 6	1 -6 8
-3.0 -2.0	2.0 3.0	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)
```


DO
MORE.