PYTHON OPERATORS:

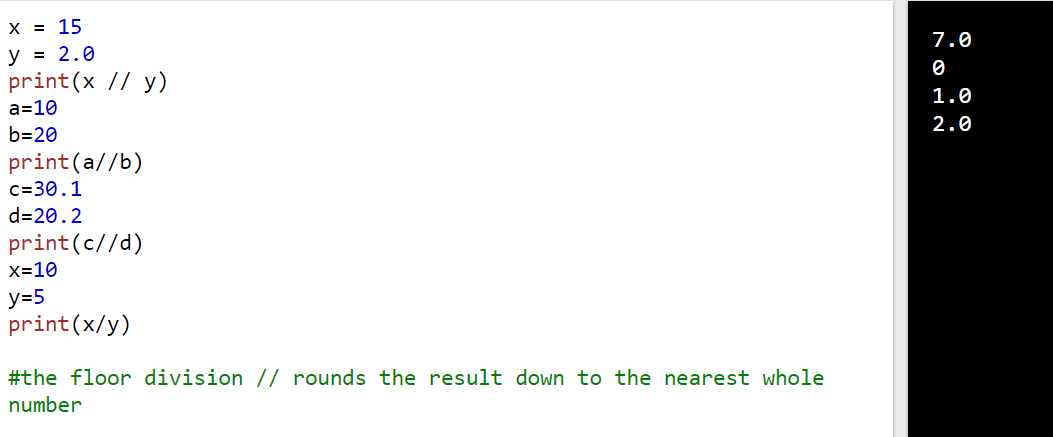
* Arithmetic operators
* Assignment operators
* Relational or Comparison operators
* Logical operators
* Special operators
  + Identity operators
  + Membership operators
* Bitwise operators
* Ternary operator
* Shift operator
* Equality operator
* Operator precedence
* Mathematical functions by using math module

ARITHMETIC OPERATOR:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| + | Addition | x + y |
| - | Subtraction | x - y |
| \* | Multiplication | x \* y |
| / | Division | x / y |
| % | Modulus | x % y |
| \*\* | Exponentiation Or power operator | x \*\* y |
| // | Floor division | x // y |

**/(Division) and //(floor division) operators:**

* It returns always floating point values
* Eg: (10/2)=5.0
* If you want **Int** value as output, go for //(floor division)
* **Floor division will return both integral arithmetic or floating-point arithmetic based on our arguments.**
* If both arguments are Int type the result will be int
* If both arguments are float type the result will be float
* **If one arguments will float the result will be in float type only**

****

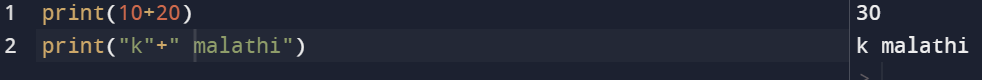
\*\* OPERATOR(POWER OPERATOR OR EXPONENTIAL OPERATOR) :

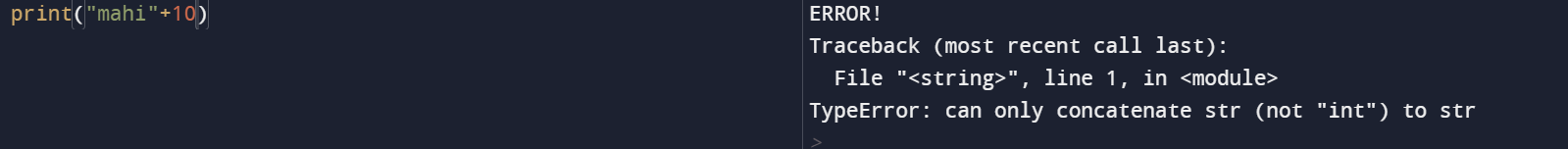
Eg: 10\*\*2 =10^2 = 100

3\*\*3 = 3^3 = 27

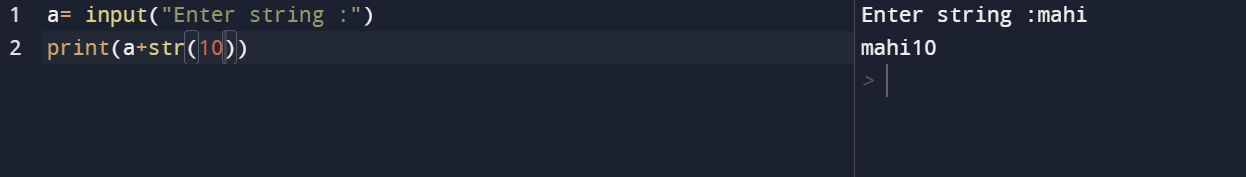
**+ OPERATOR:**

* **We can use + operator for addition as well string concatenation**
* **For string concatenation both arguments should be in string only.**
* **If we use + operator for string it’s called concatenation**
* **Syntax: str + str**

****

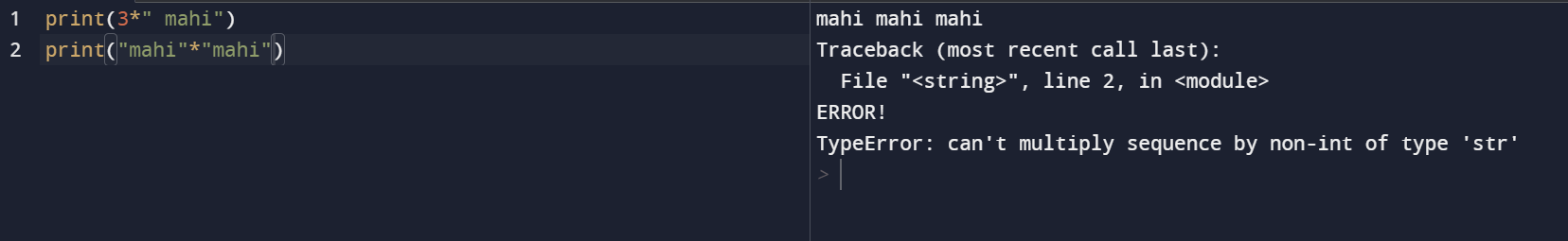






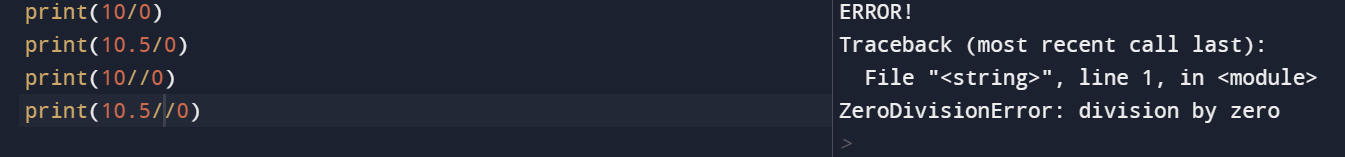
\*OPERATOR:

* We can use \* operator for strings
* We can use both string and int arguments
* Eg. Print(mahi\*3)= mahimahimahi
* Eg. Print(3\*mahi)= mahimahimahi
* It act as a string repetition
* We cannot multiply both argument as string
* If we use \* operator for string called string multiplication
* Syntax: int \* str or str \* int

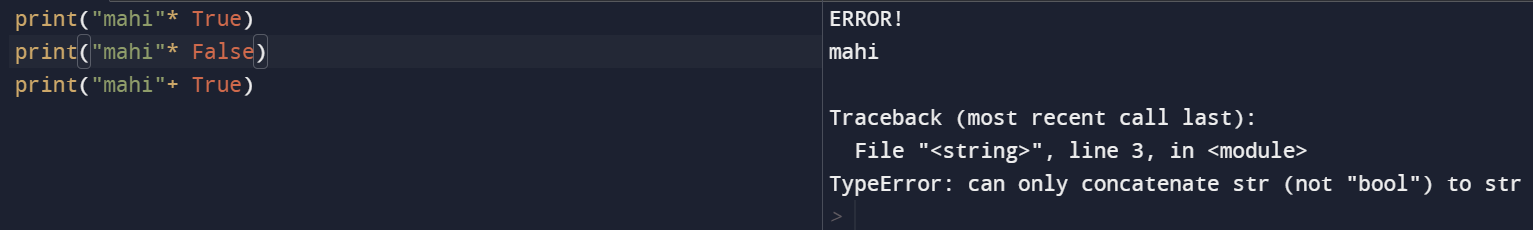


**ZERO DIVISION ERROR:**

* Any number divisible by zero we will get zero division error



Boolean:



RELATIONAL OPERATOR:

Comparison operators compare two values/variables and return a boolean result: True or False. For example,

a = 5

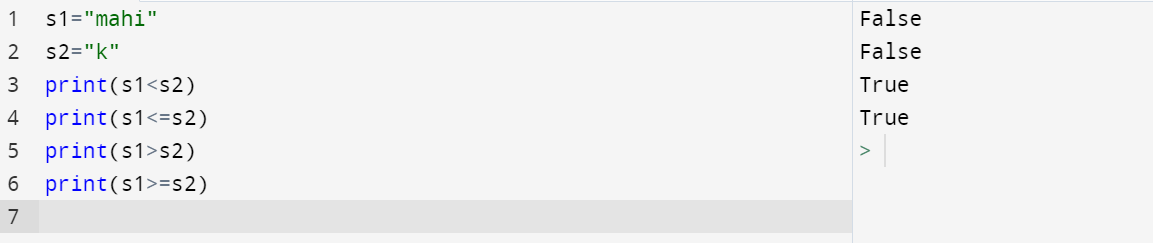
b =2

print (a > b) # True

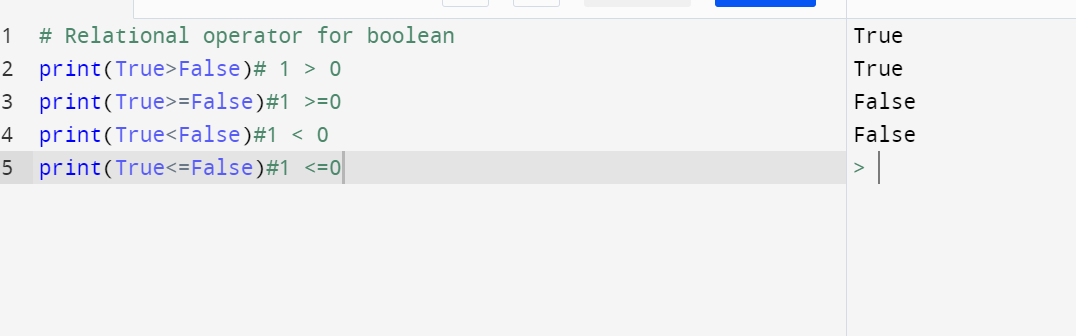
Here, the > comparison operator is used to compare whether a is greater than b or not.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| == | Is Equal To | 3 == 5 gives us **False** |
| != | Not Equal To | 3 != 5 gives us **True** |
| > | Greater Than | 3 > 5 gives us **False** |
| < | Less Than | 3 < 5 gives us **True** |
| >= | Greater Than or Equal To | 3 >= 5 give us **False** |
| <= | Less Than or Equal To | 3 <= 5 gives us **True** |

* Relational operators can be used for **string also**
* For string the comparison based on Unicode value
* If you want Unicode value for corresponding character use ord() function
* Syntax: ord(‘ a’) = 97
* If you want char using Unicode use chr( ) method
* Syntax: chr(‘97’) = ‘a’
* If you compare two strings it checks the first character of the both arguments

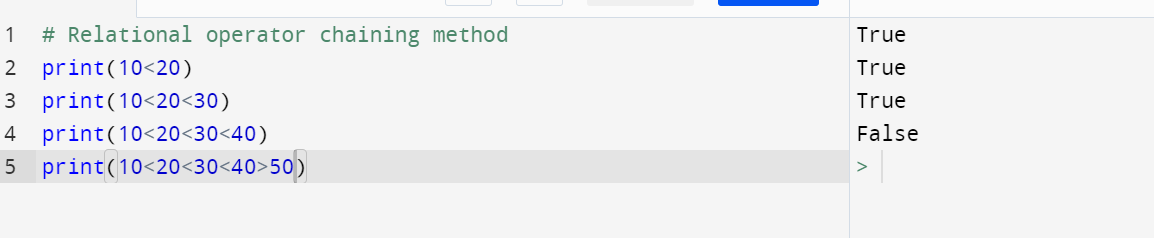


* We can apply relational operators for Boolean also
* We cannot use one argument was int and another was string
* If we use we will get TYPE ERROR.
* **Equality operator cannot applicable for incompatible type**



RELATIONAL OPERATOR CHAINING METHOD:

* For multiple comparison, if all are True it will be True
* If any one comparison was False it will be False

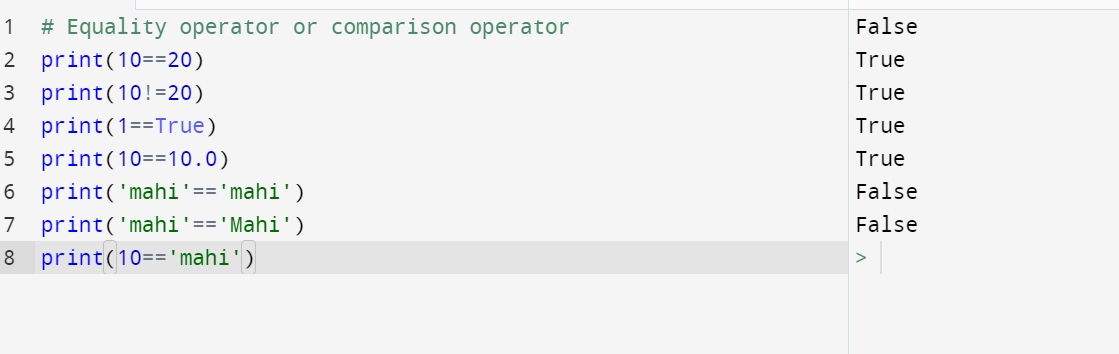


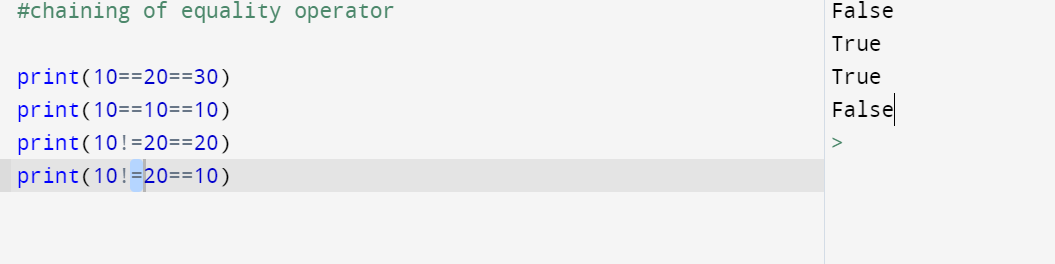
**EQUALITY OPERATOR:**

* **To check whether the two values are equal or not**
* **Equality operator can applicable for incompatible type**
* **Eg: print(10==’mahi’)**
* **If we are trying to use one argument was int another one was str we always get False only**
* **Chaining of equality operator is always applicable**
* **If all comparison returns true it will be True**
* **If any one comparison returns False it we be always False**

Comparison operators are used to compare two values:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

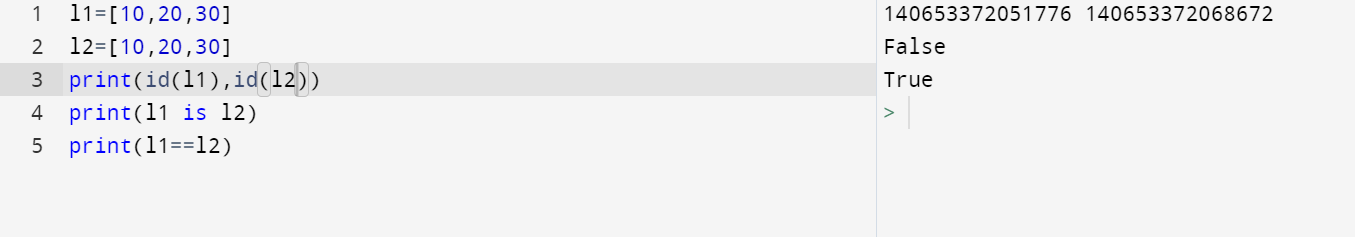
****

****

**==** OPERATOR AND **IS** OPERATOR:

Is operator means reference operator

== operator means for compare the content

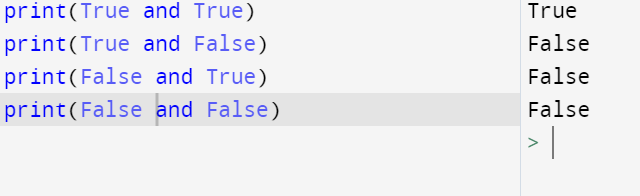


LOGICAL OPERATORS FOR BOOLEAN TYPES:

* AND
* OR
* NOT
* We can apply it for Boolean type and non Boolean type

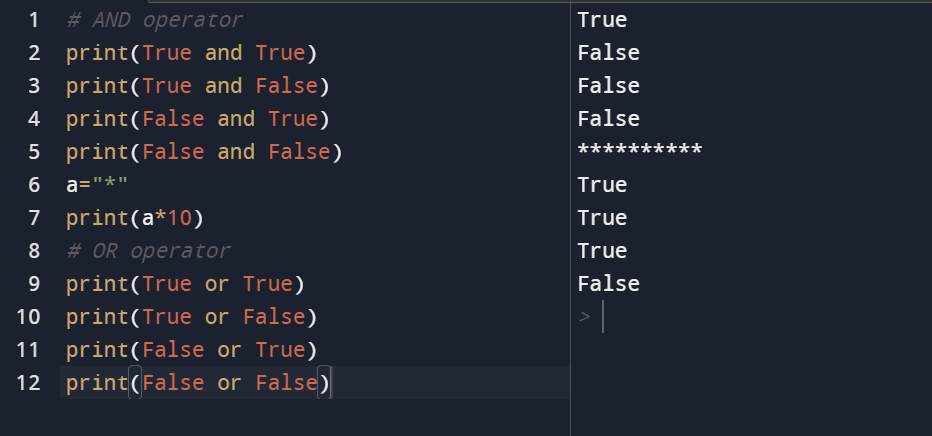
FOR BOOLEAN TYPE:

* AND operator returns true only if both arguments are true
* Otherwise it return always False



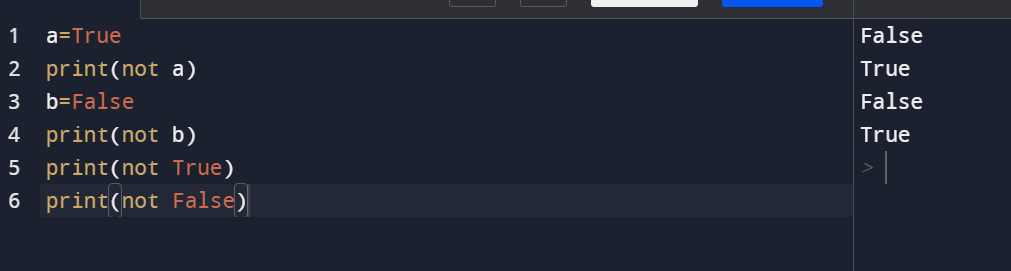
OR OPERATOR:

* OR operator returns True if any one argument was True
* If both argument was False it always return False only



NOT OPERATOR:

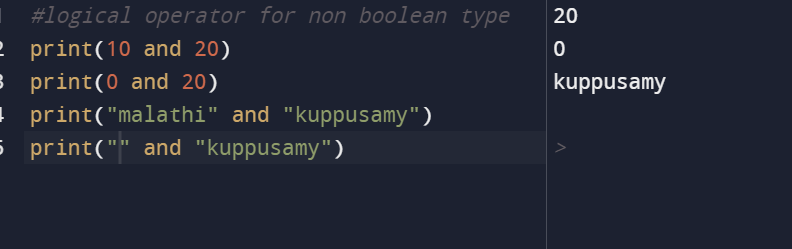
* Complement
* Not True means False
* Not False means True



LOGICAL OPERATOR FOR NON BOOLEAN TYPE:

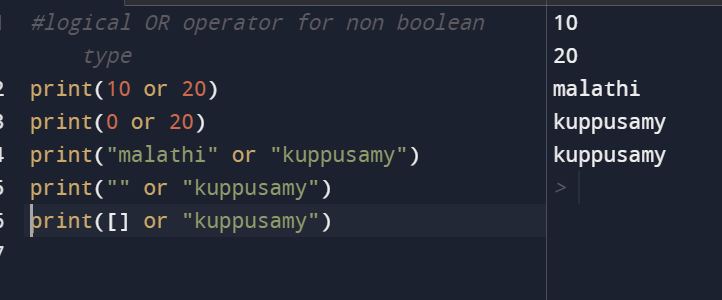
AND,OR,NOT

* In logical operators non Boolean type means
* 0 means False
* Non-zero means True
* Empty string, list ,set, tuple, dict means False.
* Whenever we apply logical AND operator for non Boolean type we will get non Boolean type result only
* Eg: X and Y ======🡺If x evaluates to False, then the result is X
* If x evaluates to True ,then the result is Y



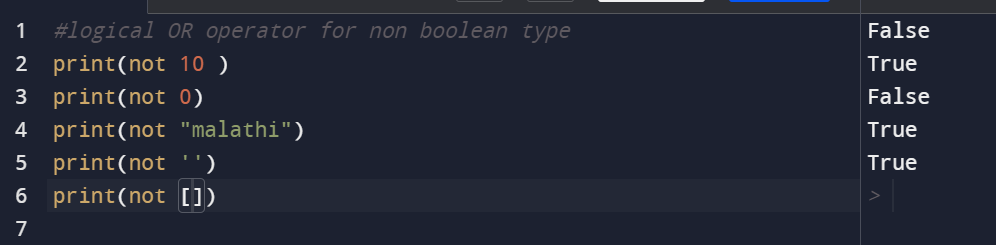
LOGICAL OR OPERATOR FOR NON BOOLEAN TYPE:

* Whenever we apply logical OR operator for non Boolean type we will get non Boolean type result only
* Eg: X and Y ======🡺If X evaluates to TRUE, then the result X
* If X evaluates to FALSE ,then the result is Y



LOGICAL NOT OPERATOR FOR NON BOOLEAN TYPE:

* Whenever we apply NOT operator for Boolean and non Boolean type **the result will be always Boolean type**

****

BITWISE OPERATOR:

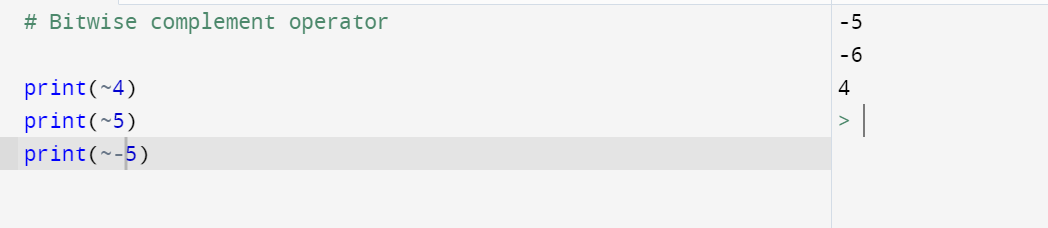
* & -------🡪 BITWISE AND
* | -------🡪 BITWISE OR
* ^ ------🡪 BITWISE X-OR
* ~ -------🡪BITWISE COMPLEMENT OPERATOR
* << -----🡪 BITWISE LEFT SHIFT OPERATOR
* >> ------🡪 BITWISE RIGHT SHIFT OPERATOR
* We can apply **bitwise operator only for int and bool data types.**
* Why Boolean allowed? Bool data internally represent int type(0 and 1)

BITWISE AND:

* **& ->if both bits are ‘1’ then result is ‘1’ otherwise ‘ 0 ’**
* **| -> if one bit is ‘1’ then result will be ‘1’ otherwise result is ‘0’**
* **^ -> if both bits are different, then result is ‘1’**
* **If both bits are same, then result is ‘0’**

**BITWISE COMPLEMENT OPERATOR:**

* **The most significant bit acts as sign bit**
  + **0 means positive number**
  + **1 means -ve number**
* **Positive number will be represented directly in the memory**
* **Negative number will be represented in 2’s complement form**
* **Find 1’s complement?**
* **Ans: Interchange 0’s and 1’s**
  + **0 🡪 1**
  + **1 🡪 0**
* **Find 2’s complement?**
* **Ans: 2’s complement= 1’s complement+1**

****

**SHIFT OPERATOR:**

**There are 2 types of operator available in python**

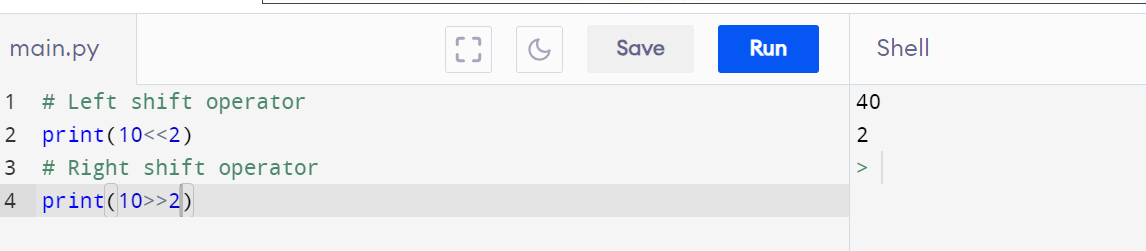
* **<< Left shift operator**
* **>> Right shift operator**

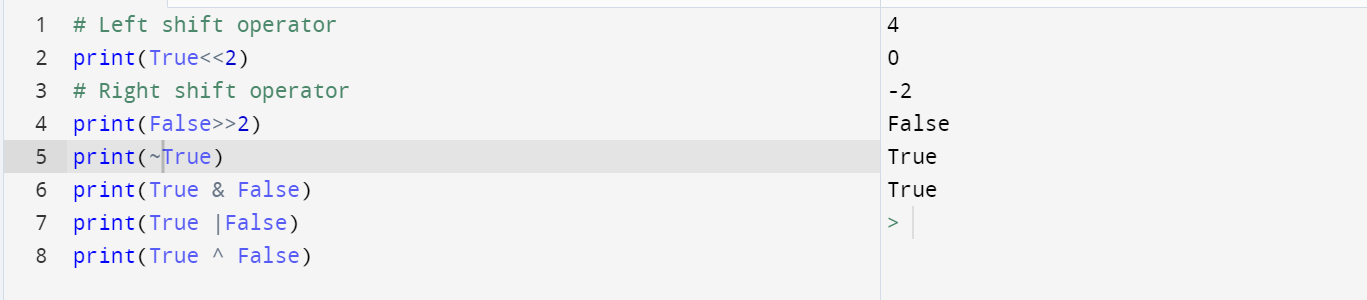
**LEFT SHIFT OPERATOR:**

* **Left shift operator means shift 2 values to left side and create 2 vaccant cells to the right side**
* **Those 2 vaccant cells filled with 0**
* **Here also bitwise complement logic applied**

**RIGHT SHIFT OPERATOR:**

* Bitwise right shift:**Shifts the bits of the number to the right and fills SIGN BIT on voids left( fills 1 in the case of a negative number) as a result.**
* **Similar effect as of dividing the number with some power of two.**

****

****

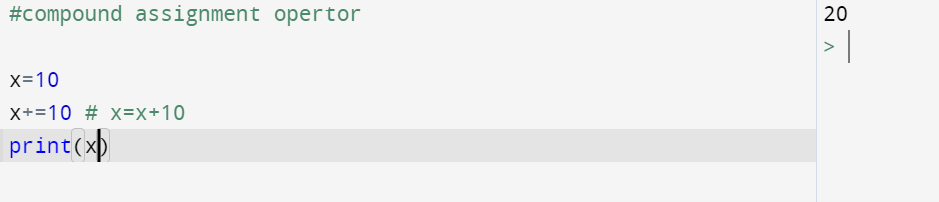
**ASSIGNMENT OPERATOR:**

[Python](https://www.javatpoint.com/python-tutorial) has an **assignment operator** that helps to assign values or expressions to the left-hand-side variable.

The assignment operator is represented as the "=" symbol used in assignment statements and assignment expressions.

In the assignment operator, the right-hand side value or operand is assigned to the left-hand operand.

**Assignment operator combined with other operator called compound assignment operator**

****

**Various compound assignment operator possible in python:**

* **Every arithmetic operator can combined with arithmetic operator to form compound assignment.**

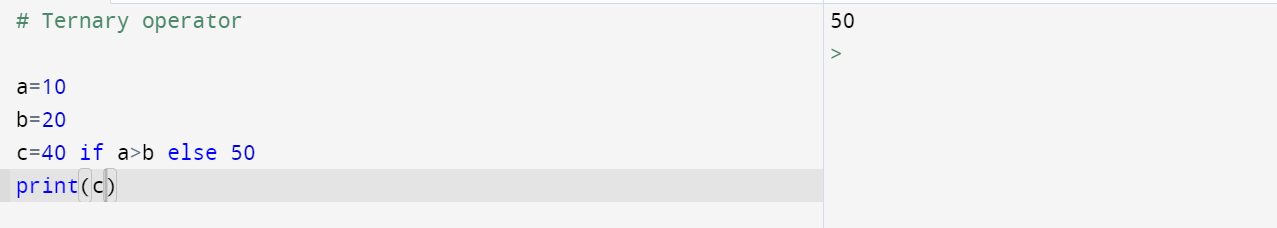
|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x - 3 |
| \*= | x \*= 3 | x = x \* 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| //= | x //= 3 | x = x // 3 |
| \*\*= | x \*\*= 3 | x = x \*\* 3 |
| &= | x &= 3 | x = x & 3 |
| |= | x |= 3 | x = x | 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

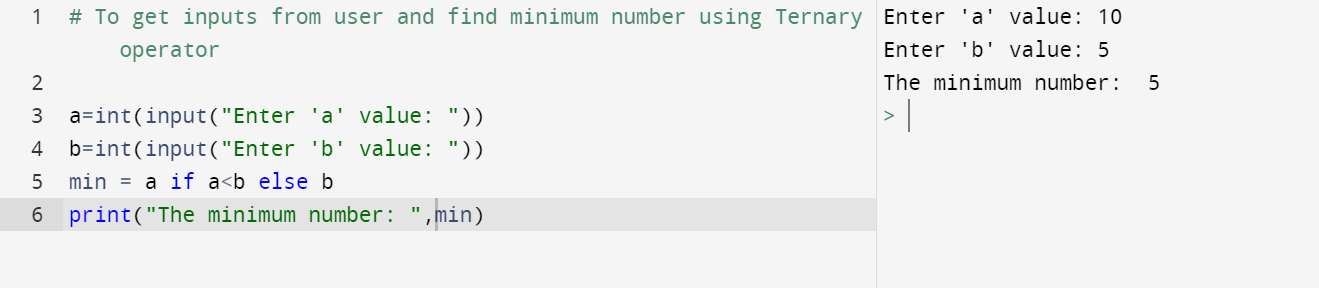
* **Increment and decrement operator not eligible in python**
* **Eg: x++,x—**

****

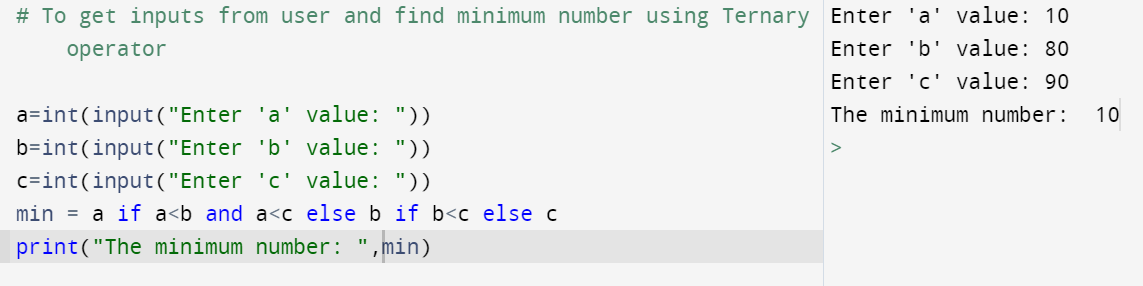
**TERNARY OPERATOR:**

* **If operator applicable for only one variable called unary operator**
* **If operator applicable for two arguments called binary operator**
* **If operator applicable for three operands called ternary operator**
* **Syntax:**
  + **Variable name= first\_value if condition second\_value**
* **If condition is True first value considered**
* **If condition is False second value considered**

****

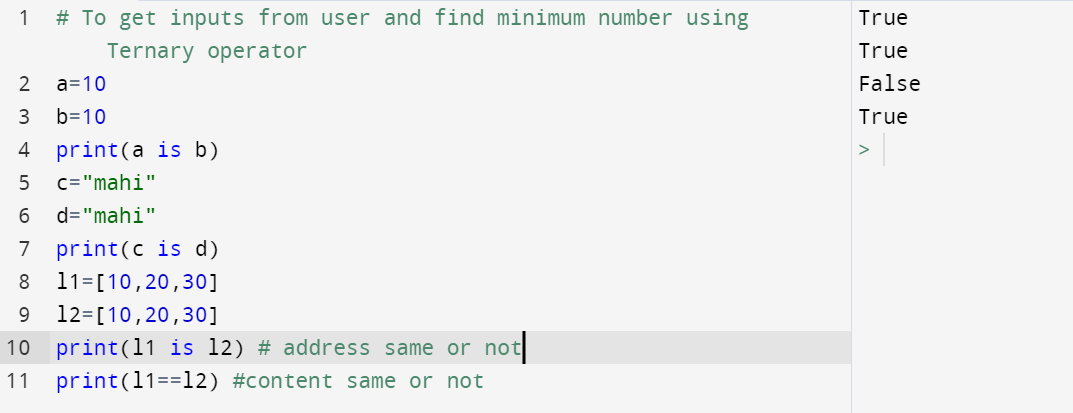
****

* **We can use nested ternary operator**

****

**SPECIAL OPERATOR:**

* **IDENTITY AND MEMBERSHIP OPERATOR:**
  + **IS**
  + **IS NOT**
* **IDENTITY OPERATOR:**
* **EG: r1 is r2====**🡺 **True if both references pointing to the same object**
* **R1 is not r2===**🡺 **True if both r1 and r2 are not pointing to the same object**

****

**MEMBERSHIP OPERATOR:**

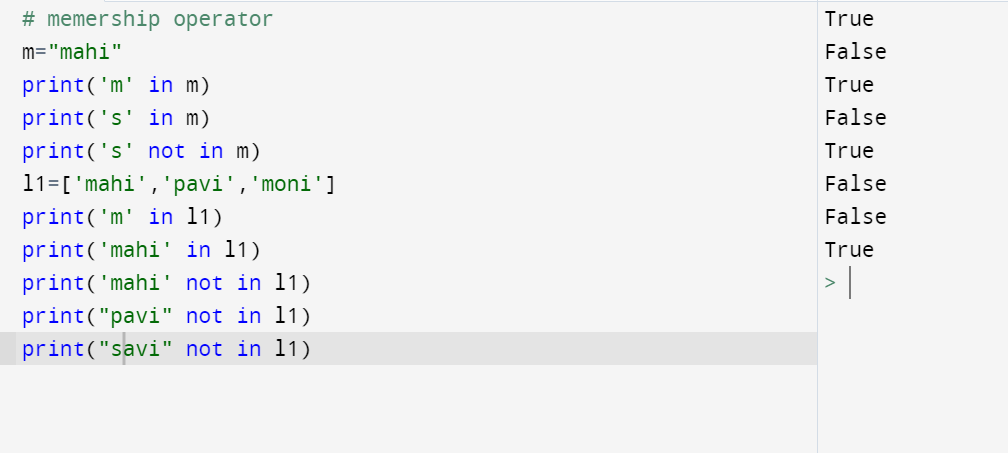
## Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| in | Returns True if a sequence with the specified value is present in the object | x in y |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y |

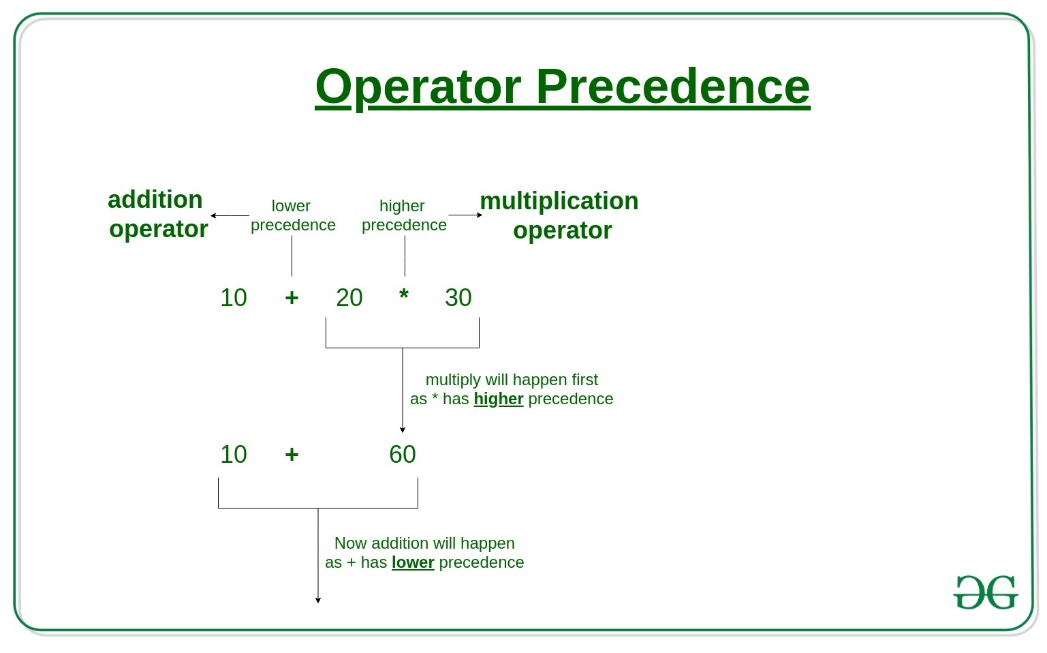
**Eg: a in sequence =🡺True**

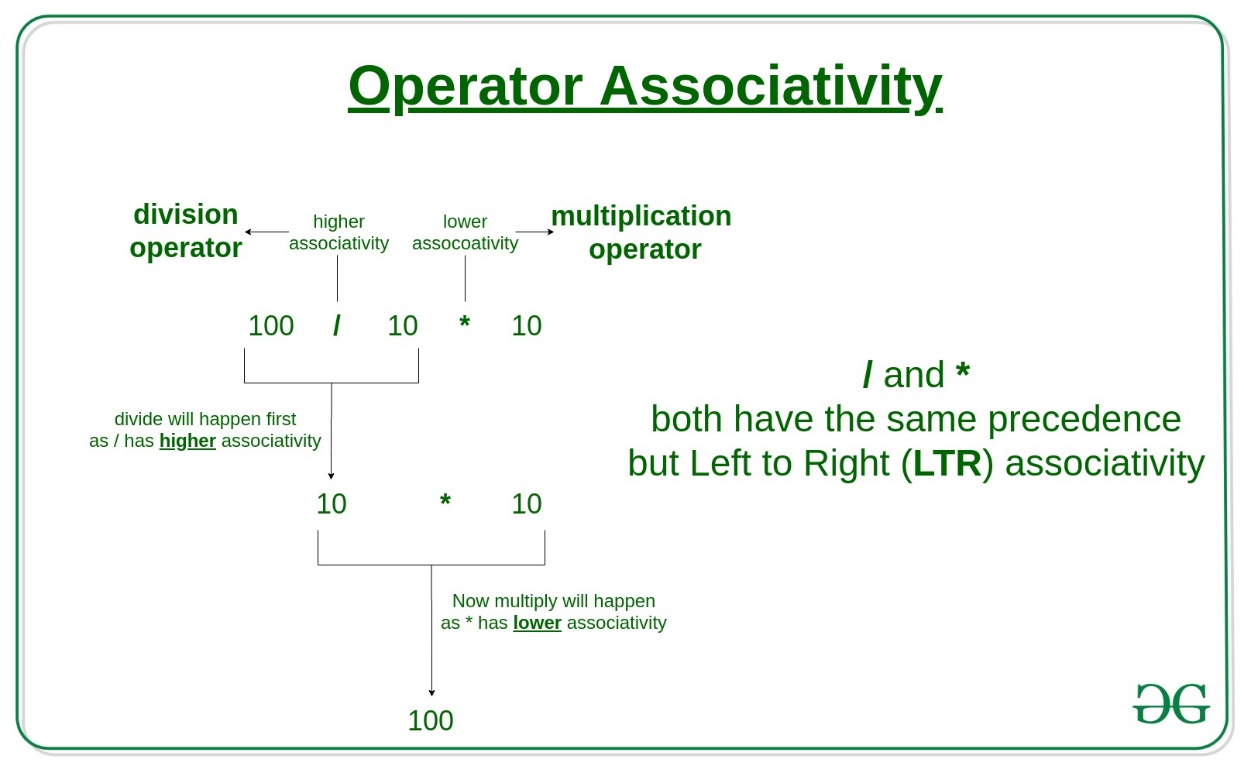
**A not in sequence =🡺True**

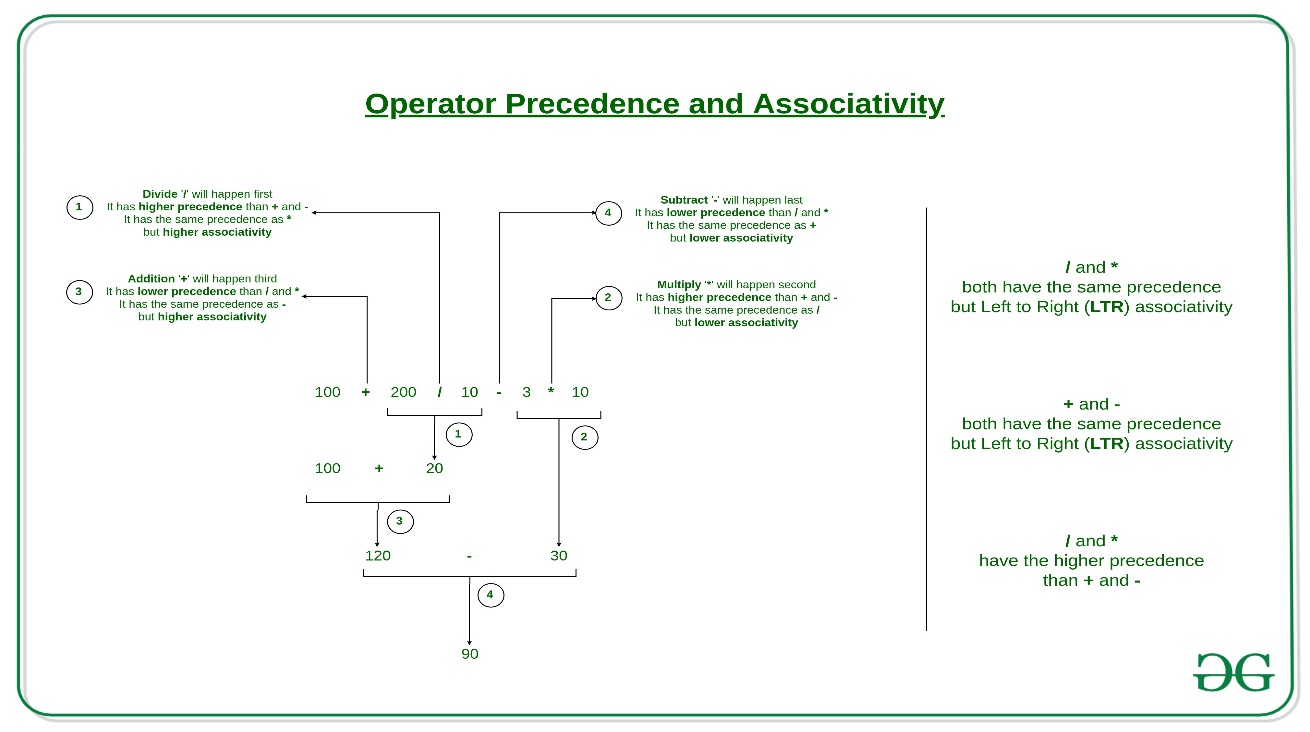
****

**OPERATOR PRECEDENCE:**

* **If multiple operators are there, which order operator should executed called operator precedence**
* **Every programming language having it’s own precedence**
* **This is used in an expression with more than one operator with different precedence to determine which operation to perform first.**
* **If both are having same precedence left to right**
* **Example:**
* 10 + 20 \* 30



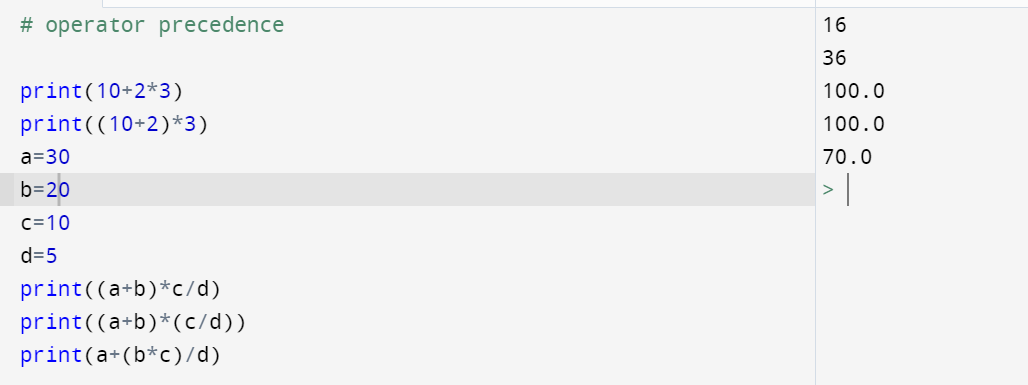




## Operator Precedence and Associativity in Python

**Please see the following precedence and associativity table for reference. This table lists all operators from the highest precedence to the lowest precedence.**

| **Precedence** | **Operators** | **Description** | **Associativity** |
| --- | --- | --- | --- |
| 1 | **()** | Parentheses | Left to right |
| 2 | **x[index]**, **x[index:index]** | Subscription, slicing | Left to right |
| 3 | [await x](https://www.geeksforgeeks.org/asyncio-in-python/) | Await expression | N/A |
| 4 | [\*\*](https://www.geeksforgeeks.org/python-operators/) | Exponentiation | Right to left |
| 5 | **+x**, **-x**, **~x** | Positive, negative, bitwise NOT | Right to left |
| 6 | **\***, **@**, **/**, **//**, **%** | Multiplication, matrix, division, floor division, remainder | Left to right |
| 7 | [**+**, **–**](https://www.geeksforgeeks.org/python-operators/) | Addition and subtraction | Left to right |
| 8 | [**<<**, **>>**](https://www.geeksforgeeks.org/python-operators/) | Shifts | Left to right |
| 9 | [**&**](https://www.geeksforgeeks.org/python-bitwise-operators/) | Bitwise AND | Left to right |
| 10 | [**^**](https://www.geeksforgeeks.org/python-bitwise-operators/) | Bitwise XOR | Left to right |
| 11 | [**|**](https://www.geeksforgeeks.org/python-bitwise-operators/) | Bitwise OR | Left to right |
| 12 | **in**, **not in**, **is**, **is not**, **<**, **<=**, **>**, **>=**, **!=**, **==** | Comparisons, membership tests, identity tests | Left to Right |
| 13 | **not x** | Boolean NOT | Right to left |
| 14 | [**and**](https://www.geeksforgeeks.org/python-operators/) | Boolean AND | Left to right |
| 15 | [**or**](https://www.geeksforgeeks.org/python-or-operator/) | Boolean OR | Left to right |
| 16 | [**if-else**](https://www.geeksforgeeks.org/python-if-else/) | Conditional expression | Right to left |
| 17 | [**lambda**](https://www.geeksforgeeks.org/python-lambda-anonymous-functions-filter-map-reduce/) | Lambda expression | N/A |
| 18 | **:=** | Assignment expression (walrus operator) | Right to left |



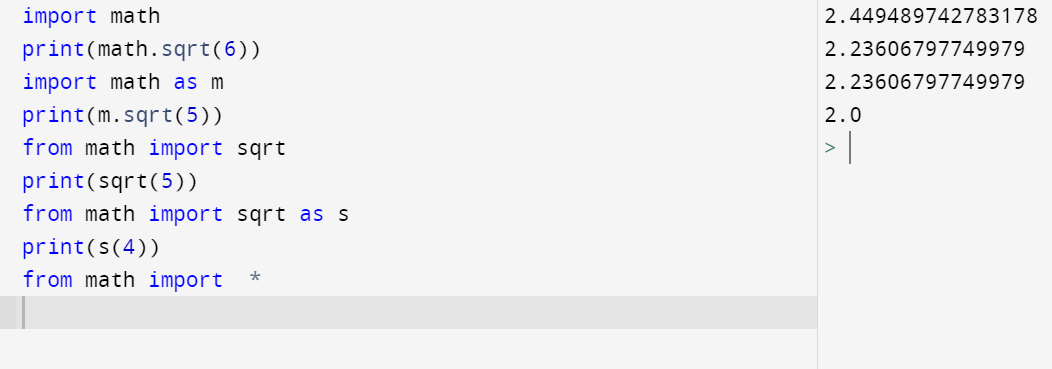
MATHEMATICAL FUNCTIONS FROM MATH MODULE:

What is module?

* Module contain a group of a collection of variables, function, classes saved to file called module.
* Every python file acts as module.
* Any collection of variables ,function and classes saved to a python file is called a module.
* Uses of module:
  + Time saving
  + code reusability
  + once write a file use many times.

important functions present in math module in python?

* The math [module](https://www.programiz.com/python-programming/modules) is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using import math.



We can create alias name for module as well as variable.