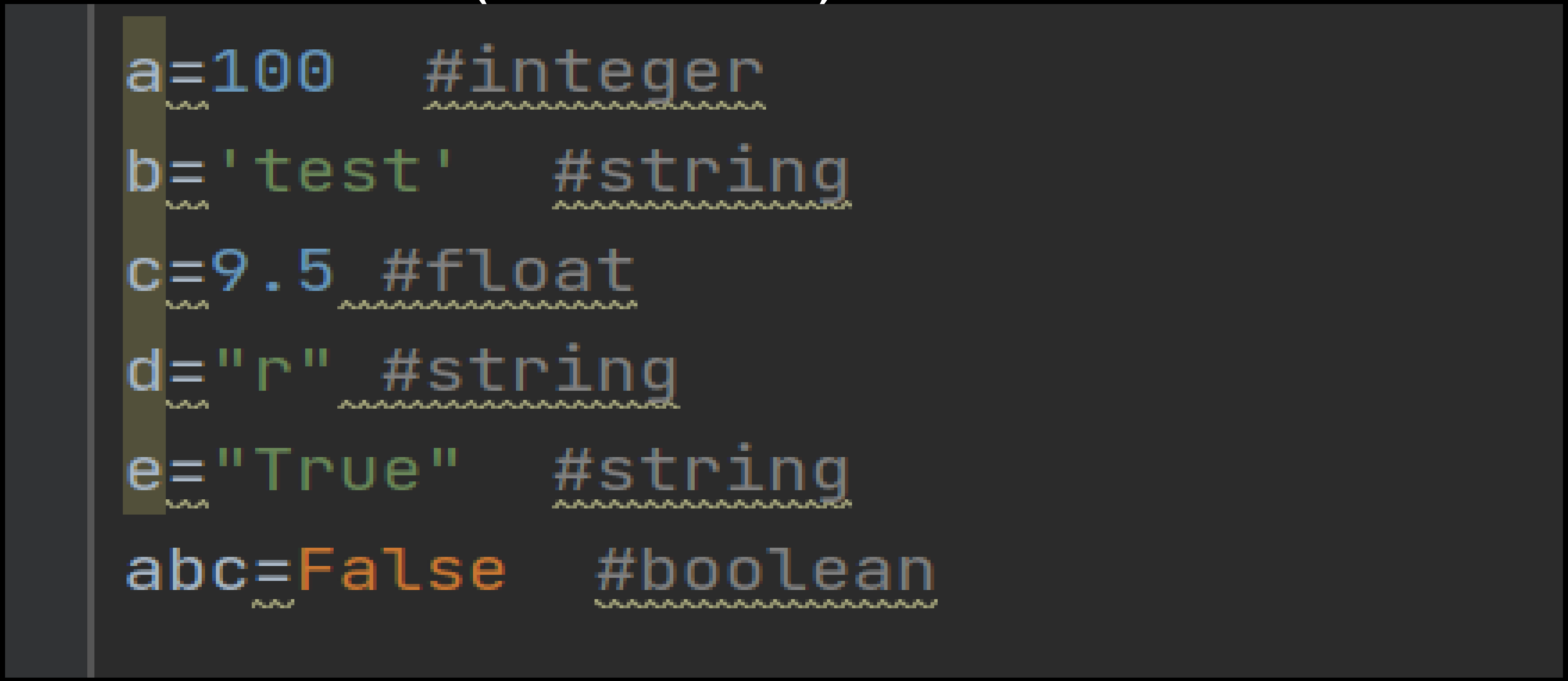
Python Data Types

In programming we categorise data into their specific types.
Python has following data types that is used to identify and categorise data:

- String for any text (str)
- int for all integer/whole numbers
- float for all decimal numbers
- Complex numbers are written with a "j" as the imaginary part:
- bool for boolean values (True / False)



```
a=100  #integer
b='test'  #string
c=9.5 #float
d="r" #string
e="True"  #string
abc=False  #boolean
```

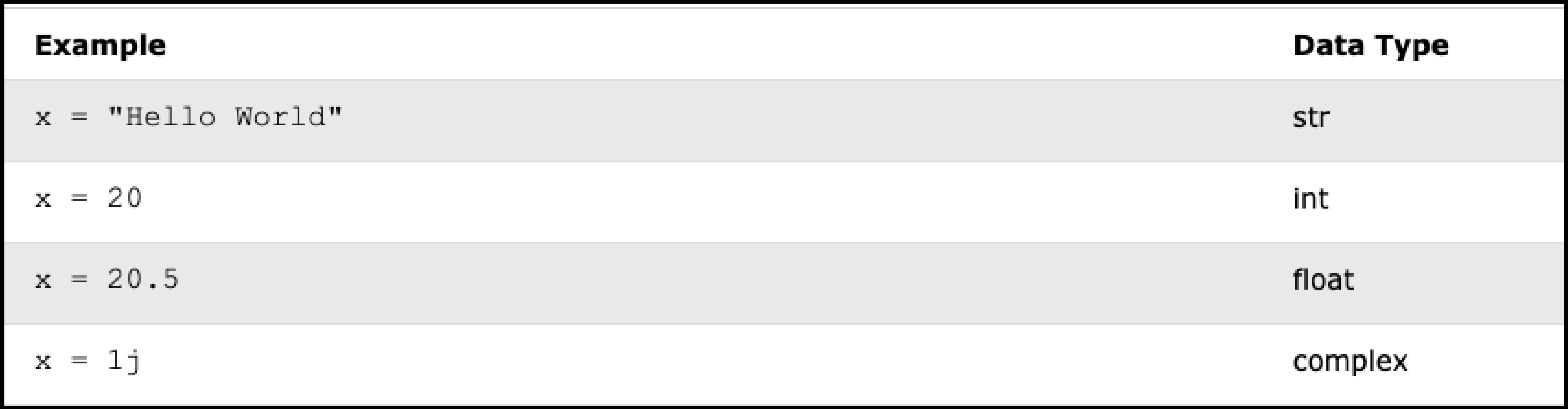
Variables

Variables are containers for storing data values.

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

```
x = 5
```



Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex

Python Type Casting

conversion of one data type into the other data type is known as type casting in python or type conversion in python. Python supports a wide variety of functions

- int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)

```
int 1      # x will be 1
int 2.8   # y will be 2
int "3"   # z will be 3



float 1        # x will be 1.0
float 2.8     # y will be 2.8
float "3"     # z will be 3.0
```

Get the Type

You can get the data type of a variable with the type() function.

```
x = 5
    "John"
print type
print type
```

String variables can be declared either by using single or double quotes

Variable names are case-sensitive.

```
a = 4
    "Sally"
#A will not overwrite a
```

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords.

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"
print
print
print
```

Note: Make sure the number of variables matches the number of values, or else you will get an error.

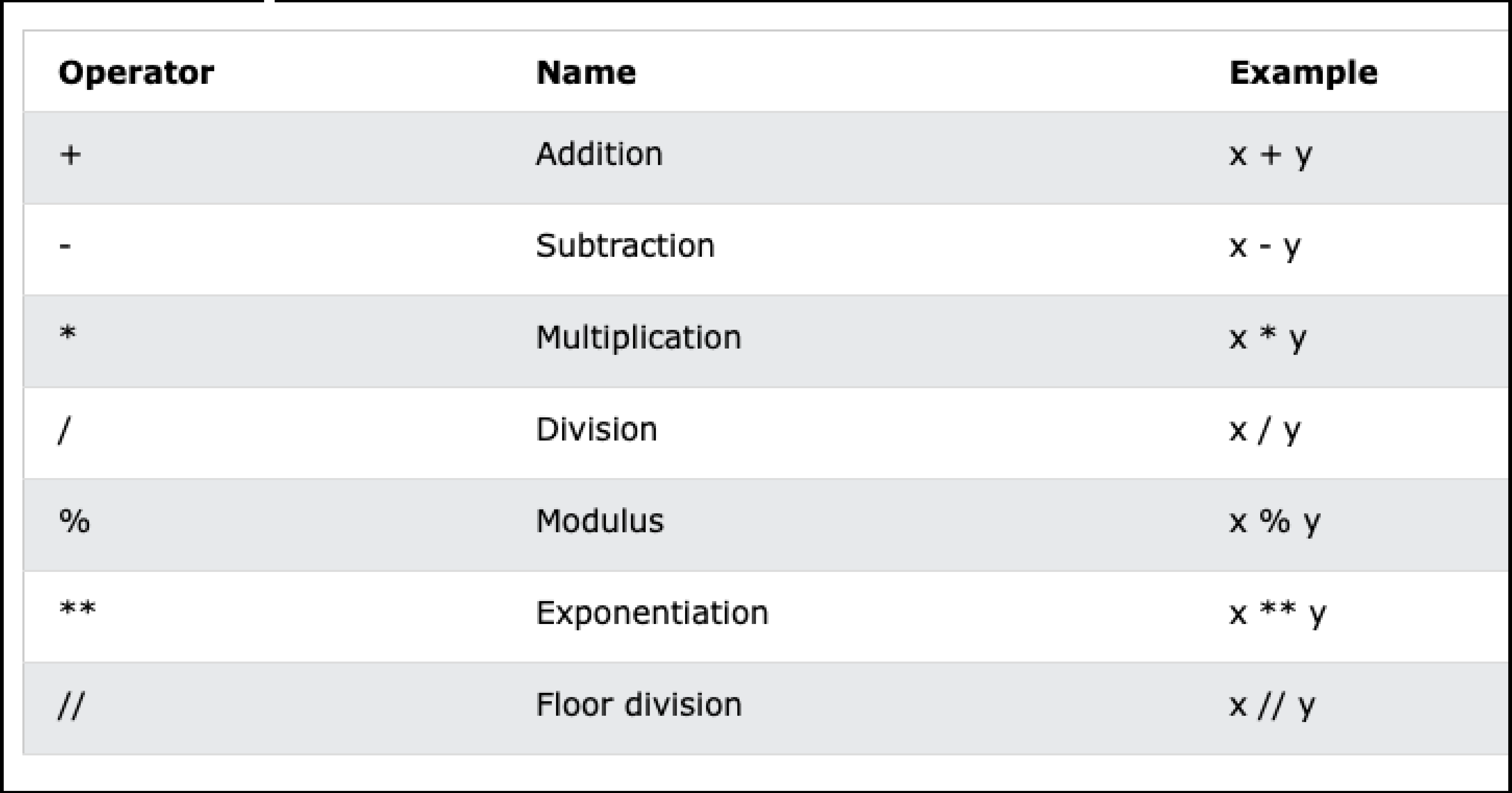
One Value to Multiple Variables

And you can assign the same value to multiple variables in one line:

```
x = y = z = "Orange"
```

Python Arithmetic Operators

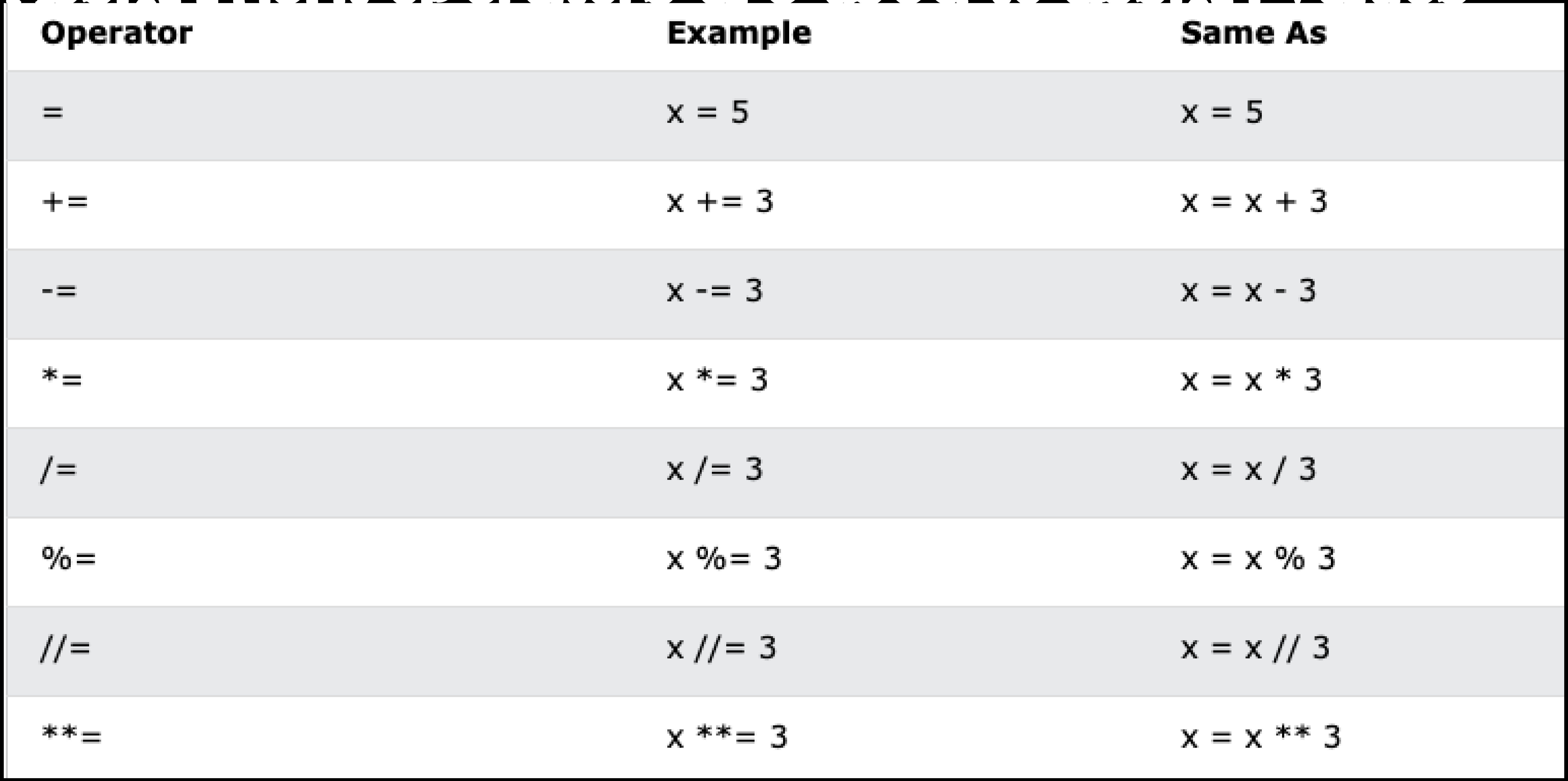
Arithmetic operators are used with numeric values to perform common mathematical



Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Assignment Operators

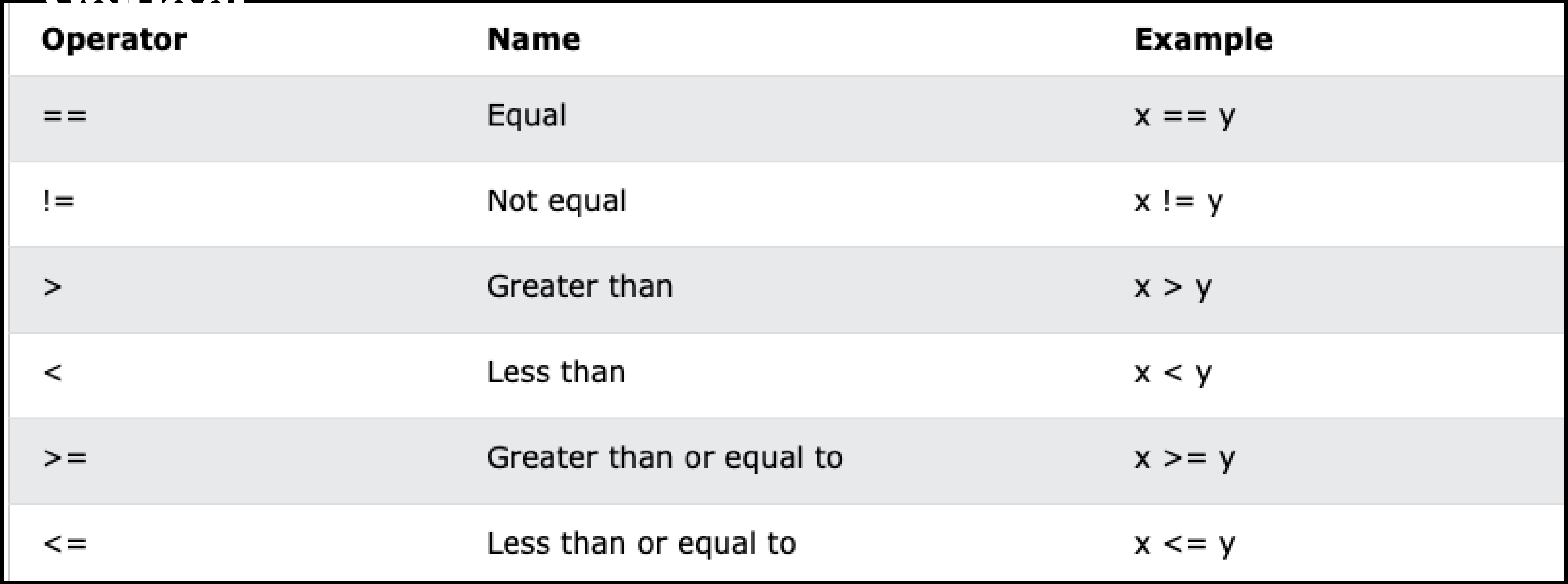
Assignment operators are used to assign values



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

Comparison operators

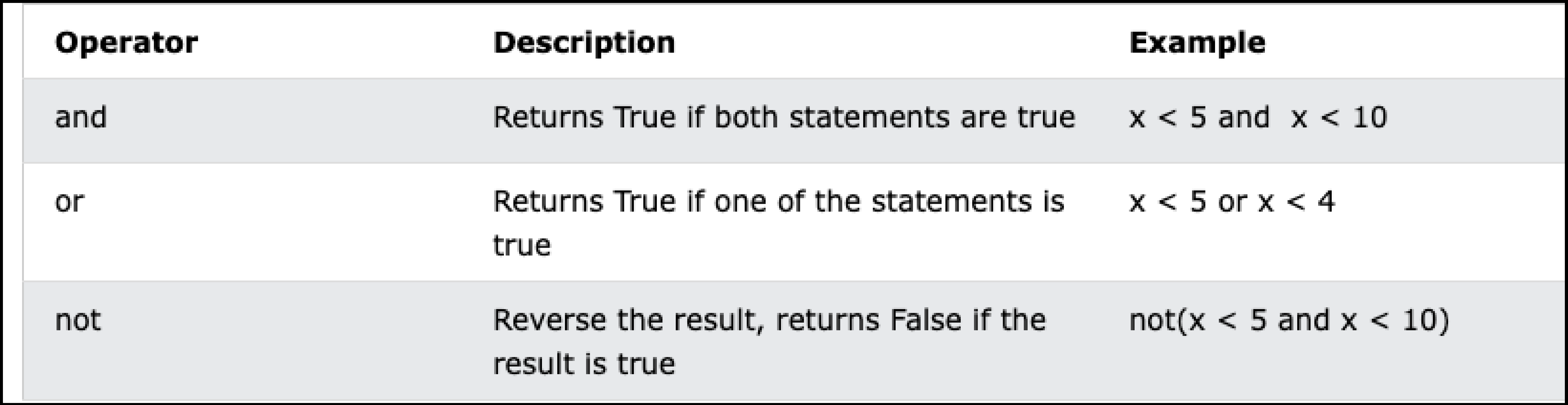
Comparison operators are used to compare two



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

Logical Operators

Logical operators are used to combine conditional statements:



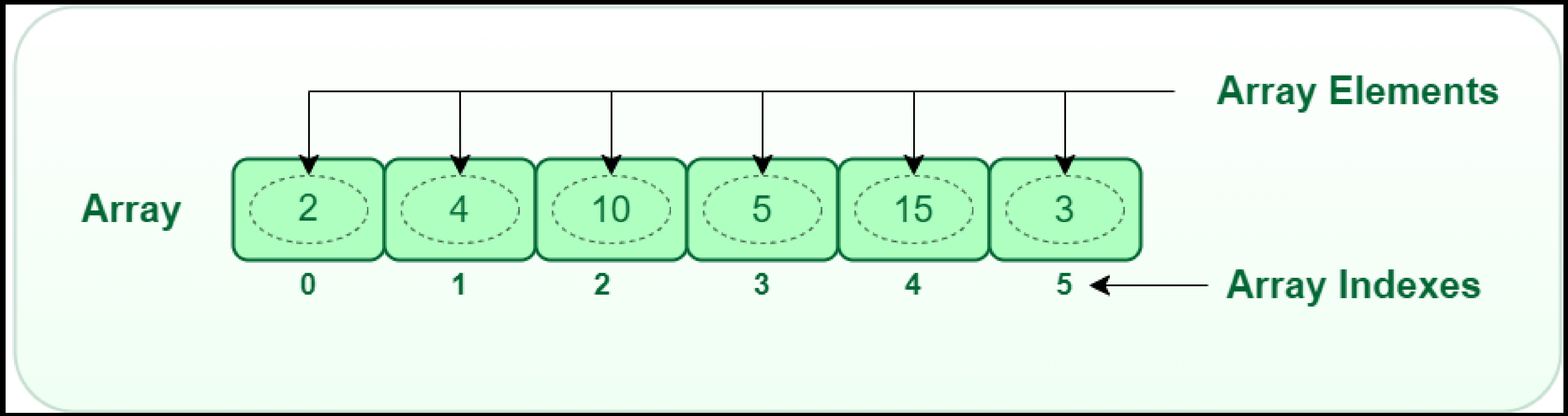
Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

You can assign a multiline string to a variable by using three quotes. You can use three double quotes Or three single quotes:

```
print "Hello"
print '    Hello'
```



Python does not have a character data type, a single character is simply a string with a length of 1.

"Hello, World!"

print 1

Check String

To check if a certain phrase or character is present in a string, we can use the keyword in.

Check if "free" is present in the following text:

"The best things in life are free!"

```
print "free" in
```

Slicing Strings

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string

Get the characters from position 2 to position 5 (not included):

Modify Strings

Upper Case: The upper() method returns the string in upper case:

```
    "Hello, World!"
print(a.upper())
```

Lower Case: The lower() method returns the string in lower case:

Split String: The split() method returns a list where the text between the specified separator becomes the list items.

```
      "Hello, World!"
print             ","    # returns ['Hello', ' World!']
```

String Concatenation: To concatenate, or combine, two strings you can use the + operator.

Format - Strings

we cannot combine strings and numbers like this:

```
age = 36
       "My name is John, I am "
print(txt)
```

But we can combine strings and numbers by using the format() method!

```
age = 36
       "My name is John, and I am {}"
print(txt.format(age))
```

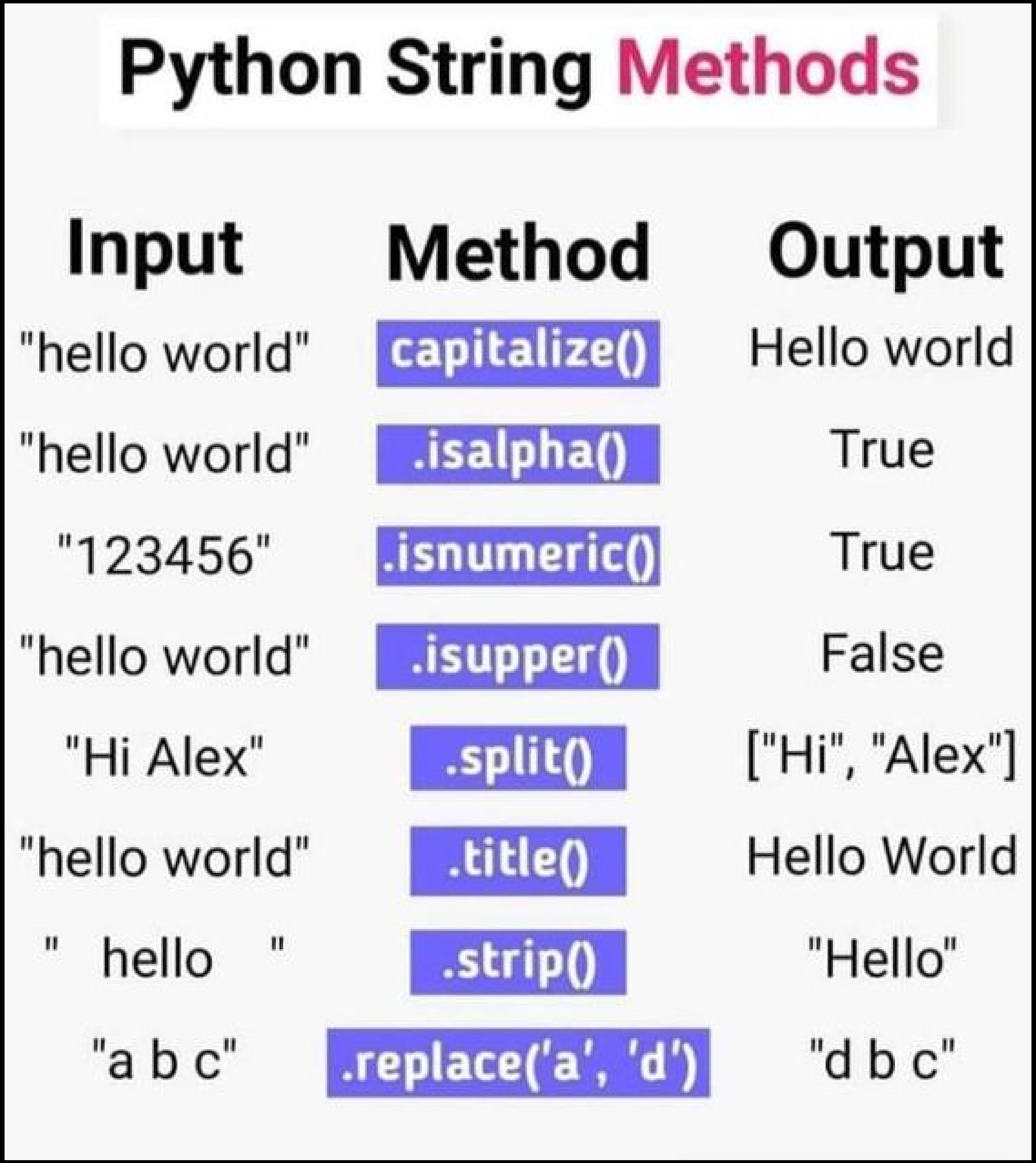
```
quantity = 3
itemno = 567
price = 49.95
              "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

```
quantity = 3
itemno = 567
price = 49.95
            "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

f-strings

F-strings provide a concise and convenient way to embed python expressions inside string literals for formatting



Python String Methods

Input	Method	Output
"hello world"	capitalize()	Hello world
"hello world"	.isalpha()	True
"123456"	.isnumeric()	True
"hello world"	.isupper()	False
"Hi Alex"	.split()	["Hi", "Alex"]
"hello world"	.title()	Hello World
" hello "	.strip()	"Hello"
"a b c"	.replace('a', 'd')	"d b c"

Boolean Values

Booleans represent one of two values: True or False.

```
print 10   9
print 10     9
print 10   9
```

```
bool "abc"
bool 123
bool  "apple"  "cherry"  “   banana"
```

In fact, there are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

User Input

Syntax:

```
input ( prompt )
```

input (): This function first takes the input from the user and converts it into a string. The type of the returned object always will be < c l a s s ‘s t r’> . It does not evaluate the expression it just returns the complete statement as String.

When the input function is called it stops the program and waits for the user’ s input.

When the user presses enter, the program resumes and returns what the user typed.

```
name = input('What is your name?\n')
```

```
num = input ("Enter number :")
print(num)
name1 = input("Enter name : ")
print(name1)

# Printing type of input value
print ("type of number", type(num))
print ("type of name", type(name1))
```

Type casting input string:

```
num = int(input("Enter a number: "))
print(num, " ", type(num))


floatNum = float(input("Enter a decimal number: "))
print(floatNum, " ", type(floatNum))
```