Python

prepared by: Ms. Soofi Shafiya
NET-JRF, JMI

29-01-2024

Lab-1

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Example:

- `print("Hello, World!")`

Python Installation

Check if it already exists:

- To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe): “python --version”
- If it is not installed, then go to the website “python.org” and download the required version.

- In cmd, navigate to the directory where you have saved your file and write:

```
python <file_name>.py
```

- Whenever you are done in the python command line, you can simply type the following to quit the python command line interface:

```
exit()
```

Python Indentation

- Indentation refers to the spaces at the beginning of a code line.
- Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.

```python
if 5 > 2:
  print("Five is greater than two!")
```

- Python will give you an error if you skip the indentation.

Indentation cont…

- The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

- Example

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

- You have to use the same number of spaces in the same block of code, otherwise Python will give you an error

Python Variables

Variables are containers for storing data values.

```
x = 5
y = "Hello, World!“
```

Python has no command for declaring a variable.

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

Variables do not need to be declared with any particular *type* and can even change type after they have been set.

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

•Casting

If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

- Get the type

You can get the datatype of the variable with type() function

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

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

```
x = "John"
# is the same as
x = 'John’
```

- Variable names are case-sensitive.

```
a = 4
A = "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)

- Many Values to Multiple Variables

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

- One Value to Multiple Variables

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Python - Global Variables

- Variables that are created outside of a function (as in all of the examples above) are known as global variables.
- Global variables can be used by everyone, both inside of functions and outside.

```
x="awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

- If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

The global Keyword

- Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.
- To create a global variable inside a function, you can use the global keyword

Example:

If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Python comments (Single line comment)

```
#This is a comment.
print("Hello, World!")
```

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

```
print("Hello, World!") #This is a comment
```

Multiline comment

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

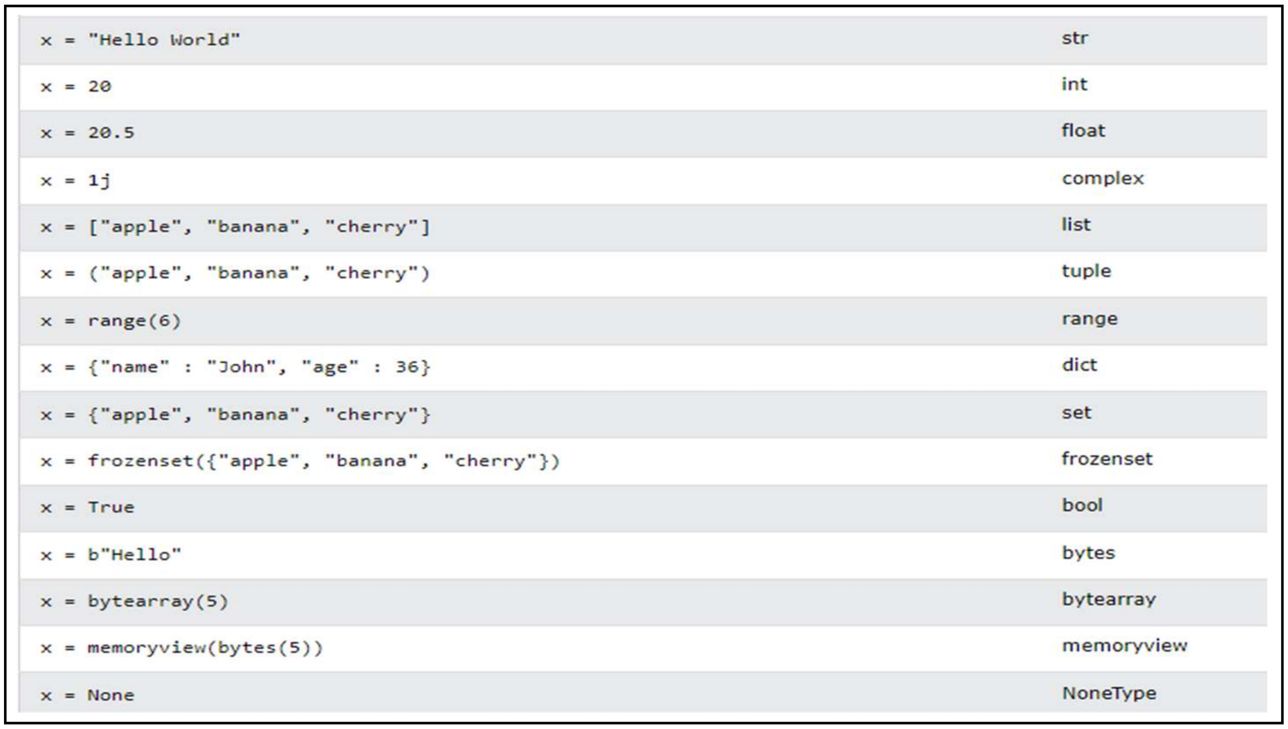
OR

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

Python Datatypes

- Python has the following data types built-in by default, in these categories:

Text type: str
Numeric types: int, float, complex
Sequence types: list, tuple, range
Mapping type: dict
Set types: set, frozenset
Boolean type: bool
Binary types: bytes, bytearray, memoryview
None type: NoneType



`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview
`x = None`	NoneType

Random number

- Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

```
import random

print(random.randrange(1, 10))
```

Strings are Arrays

- Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.
- However, Python does not have a character data type, a single character is simply a string with a length of 1.
- Square brackets can be used to access elements of the string.
- Example
- Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"
print(a[1])
```

Looping Through a String

- Since strings are arrays, we can loop through the characters in a string, with a `for` loop.
- Example

Loop through the letters in the word "banana":

```
for x in "banana":
  print(x)
```

String Length

- Example

```
a = "Hello, World!"
print(len(a))
```

Check String

- To check if a certain phrase or character is present in a string, we can use the keyword “in”:

```
txt = "The best things in life are free!"
print("free" in txt)
```

- Use it in an “if” statement

```
txt = "The best things in life are free!"
if "free" in txt:
  print("Yes, 'free' is present.")
```

Check if NOT

- To check if a certain phrase or character is NOT present in a string, we can use the keyword “not in”

```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

```
txt = "The best things in life are free!"
if "expensive" not in txt:
  print("No, 'expensive' is NOT present.")
```

Python - Slicing Strings

Slicing

- 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.

```
b = “Hello World!”
print(b[2:5])
```

Note: The first character has index 0.

Slice from start

-
  ```
  b = "Hello, World!"
  print(b[:5])
  ```

Slice from end

-
  ```
  b = "Hello, World!"
  print(b[2:])
  ```

Negative Indexing

-
  ```
  b = "Hello, World!"
  print(b[-5:-2])
  ```

```
a = "Hello, World!"
print(a.upper()) # returns  the string in upper case

a = "Hello, World!"
print(a.lower()) # returns the string in lower case

a = " Hello, World! "
print(a.strip()) # remove white spaces

a = "Hello, World!"
print(a.replace("H", "J")) #replaces a string with
                            another string
```

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

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!’]
```

String Concatenation:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

String Format

- As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

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

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

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

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

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

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

Escape Character

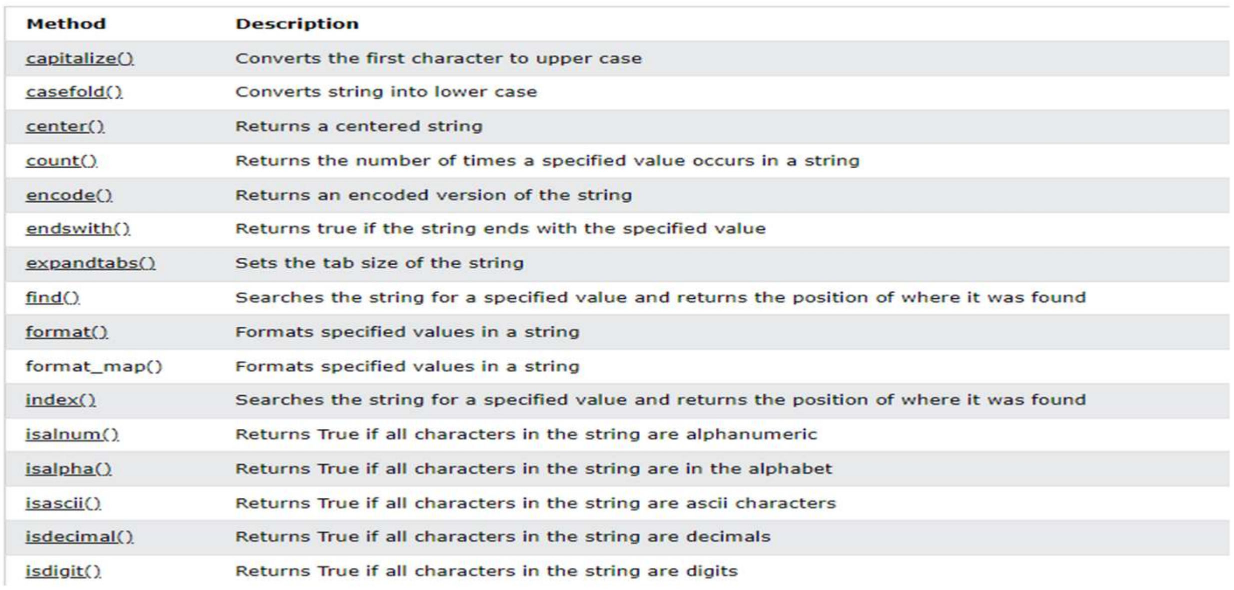
To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash “\” followed by the character you want to insert.

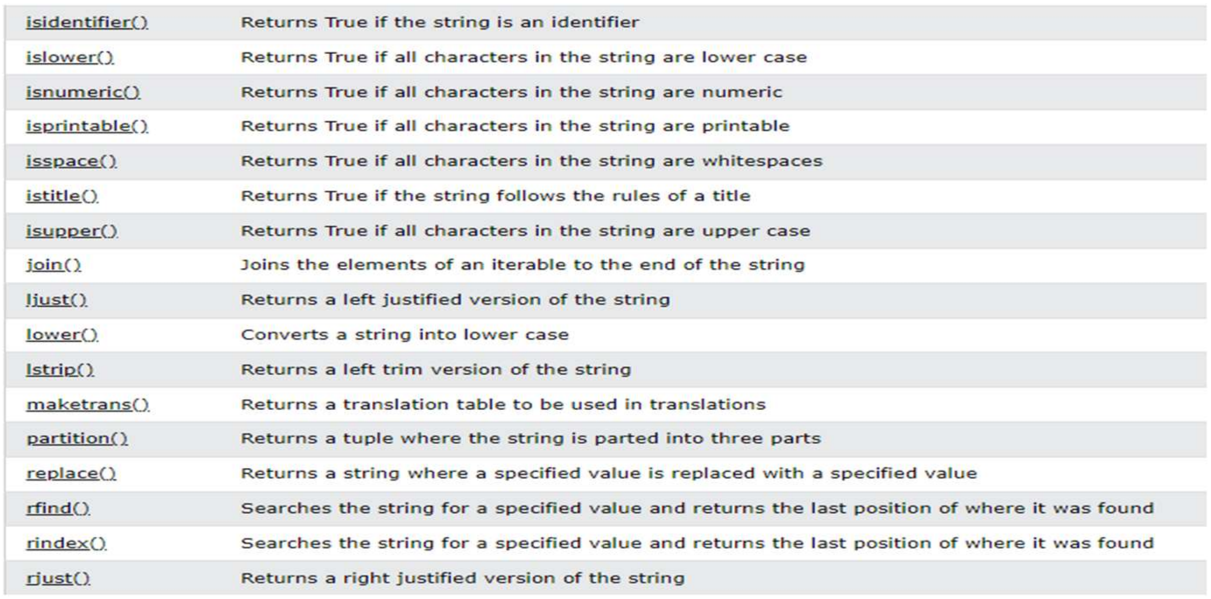
An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

```
txt = "We are the so-called \"Vikings\" from the north."
```

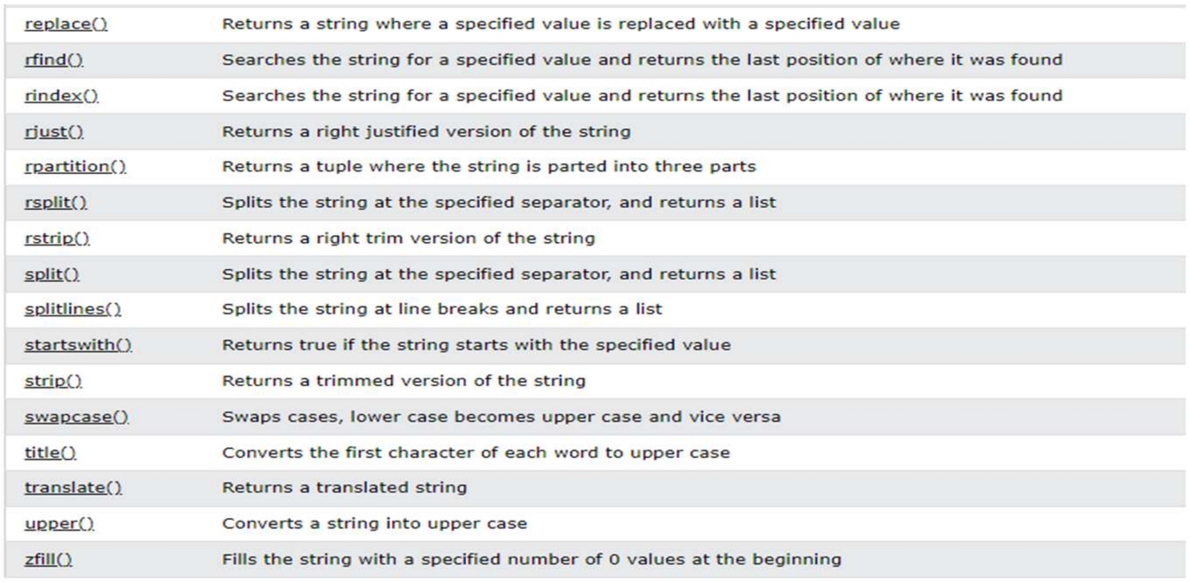
String Methods



Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits



isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string



replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Python Booleans

- Booleans represent one of two values: True/False

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

O/p:

```
True
False
False
```

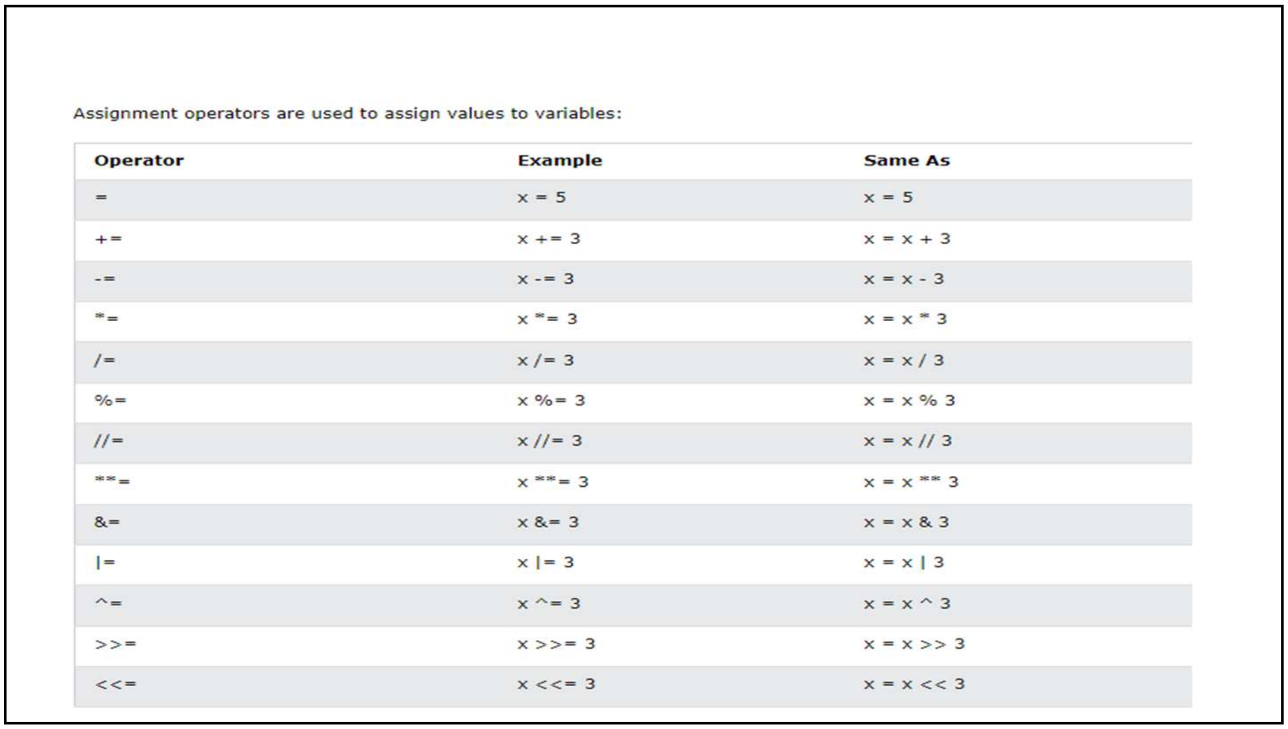
```
a = 200
b = 33

if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Python Operators

Operators are used to perform operations on variables and values.

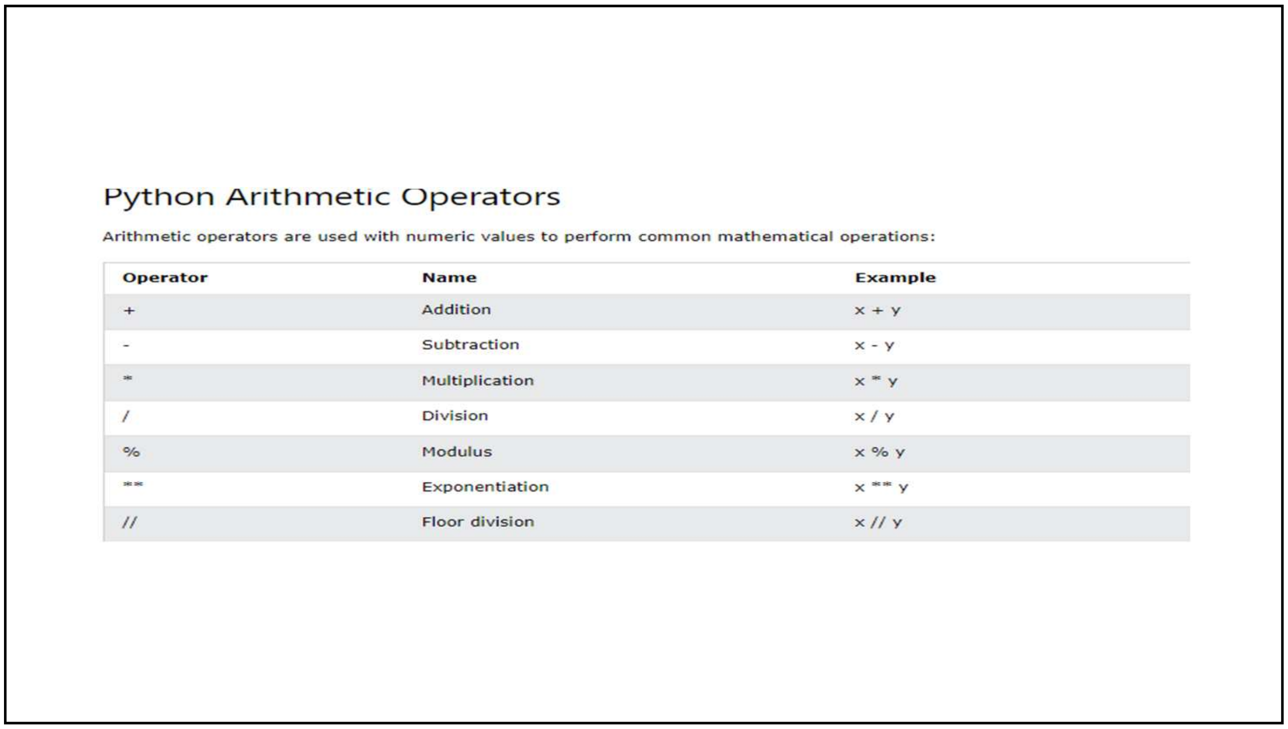
Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators



Assignment operators are used to assign values to variables:

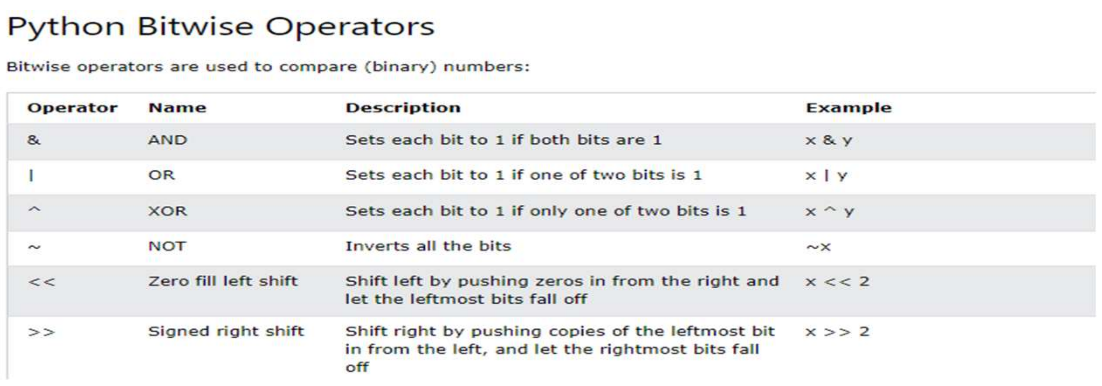
Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3



Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

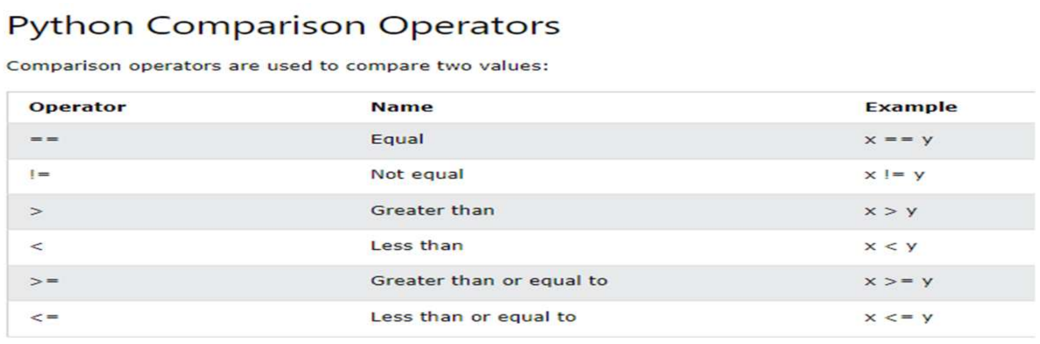
Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y



Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2



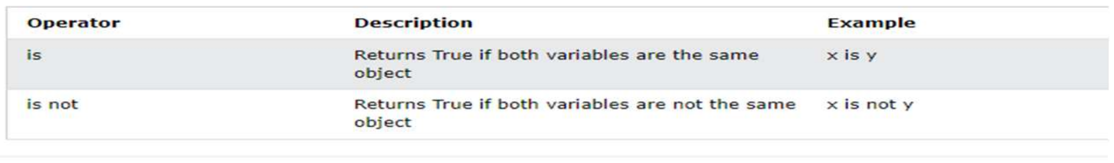
Python Comparison Operators

Comparison operators are used to compare two values:

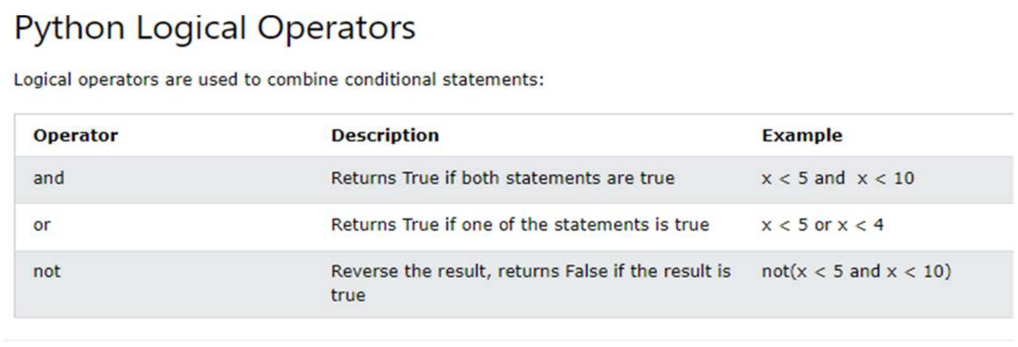
Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:



Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y



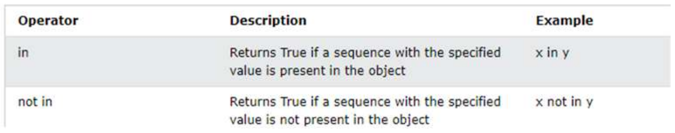
Python Logical Operators

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)

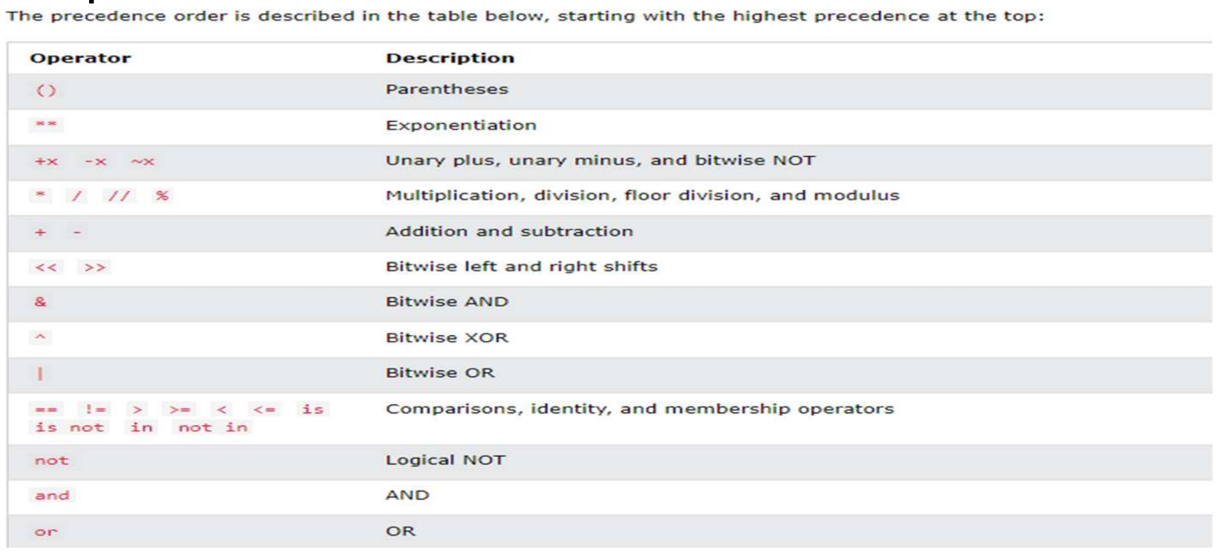
Membership Operator

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

Operator Precedence



The precedence order is described in the table below, starting with the highest precedence at the top:

Operator	Description
`()`	Parentheses
`**`	Exponentiation
`+x` `-x` `~x`	Unary plus, unary minus, and bitwise NOT
`*` `/` `//` `%`	Multiplication, division, floor division, and modulus
`+` `-`	Addition and subtraction
`<<` `>>`	Bitwise left and right shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==` `!=` `>` `>=` `<` `<=` `is` `is not` `in` `not in`	Comparisons, identity, and membership operators
`not`	Logical NOT
`and`	AND
`or`	OR

Python Lists

```
mylist = ["apple", "banana", "cherry"]
```

- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
- Lists are created using square brackets: