210301601 INTRODUCTION TO PYTHON
UNIT– I

Introduction to Python

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.

Python is a cross-platform programming language, meaning, it runs on multiple platforms like Windows, MacOS, Linux and has even been ported to the Java and .NET virtual machines. **It is free and open source.**

- Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is an object-oriented programming language.
- It is ideally designed for rapid prototyping of complex applications.
- Many large companies use the Python programming language include NASA, Google, YouTube, BitTorrent, etc.

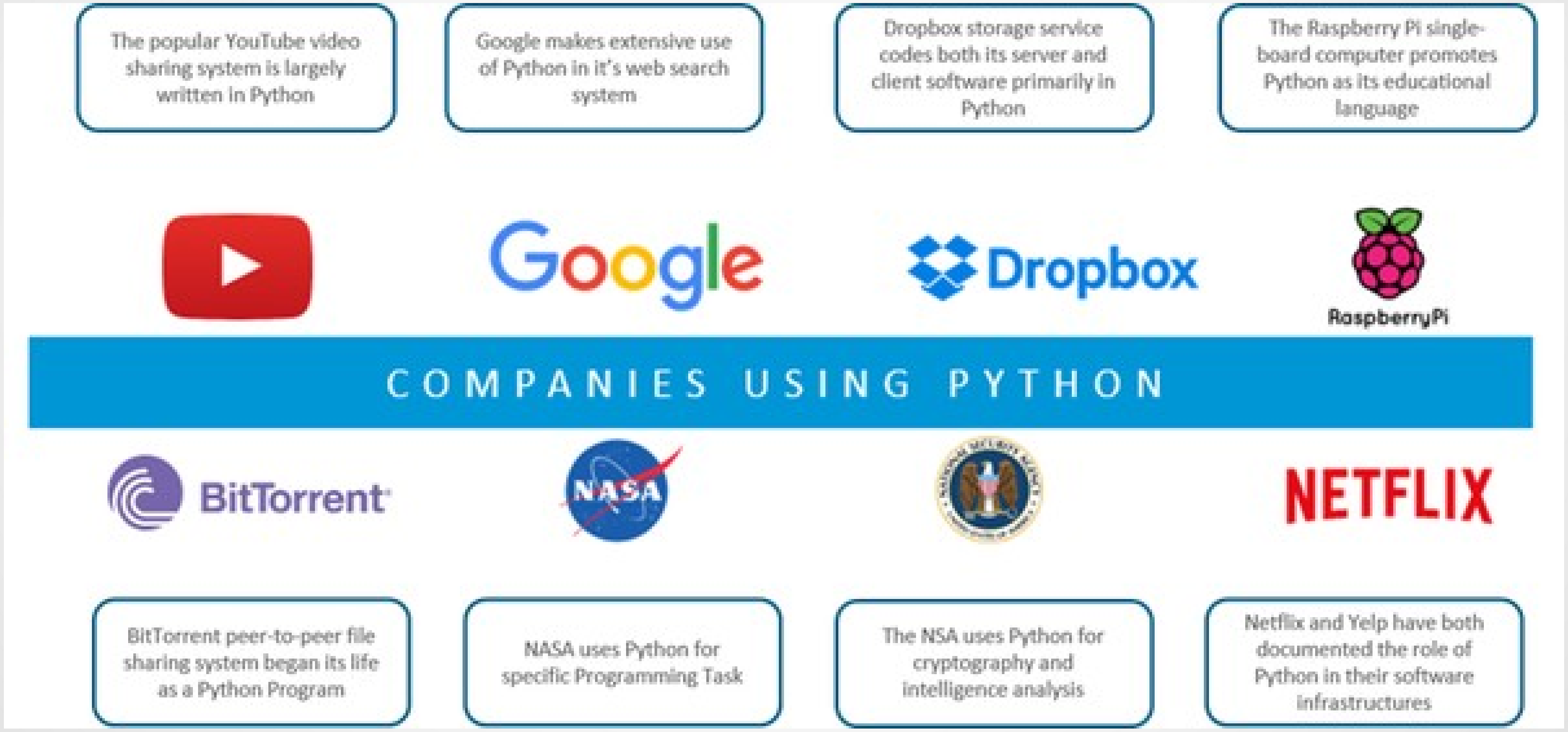
Python is widely used in Artificial Intelligence, Natural Language Generation, Neural Networks and other advanced fields of Computer Science.

History of Python

Python was developed by **Guido van Rossum** in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.
Python was created as a successor of a language called ABC (All Basic Code) and released publicly in 1991 and version 1.0 was released in 1994.
Python is derived from other languages like C, C++, SmallTalk, and Unix shell and other scripting languages. Python source code is now available under the GNU General Public License (GPL).
Latest version of Python – 3.10.0

Interesting fact: Python is named after the comedy television show Monty Python’s Flying Circus. It is not named after the Python snake.

Where Python is used?



The popular YouTube video sharing system is largely written in Python

Google makes extensive use of Python in it's web search system

Dropbox storage service codes both its server and client software primarily in Python

The Raspberry Pi single-board computer promotes Python as its educational language

COMPANIES USING PYTHON

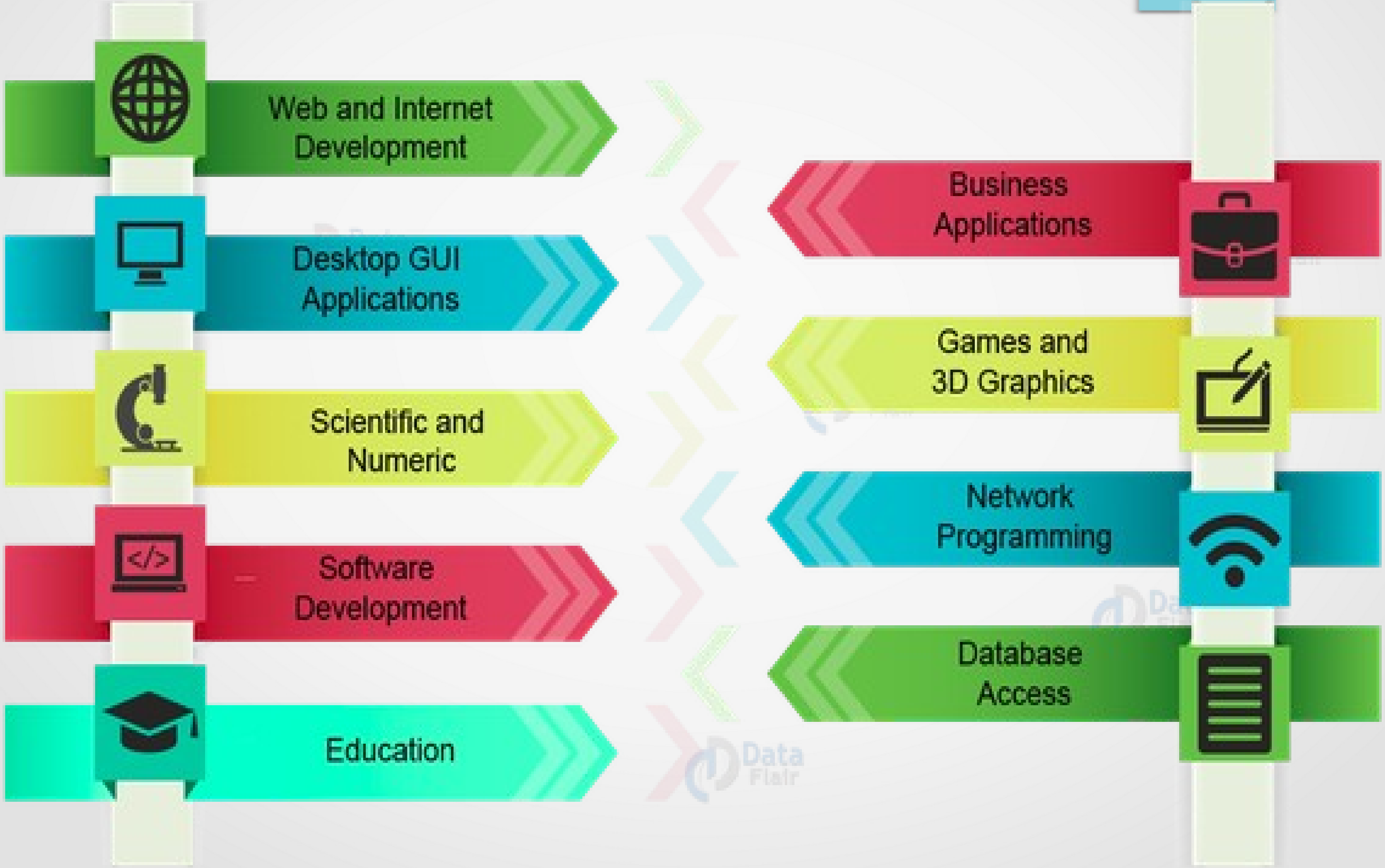
BitTorrent peer-to-peer file sharing system began its life as a Python Program

NASA uses Python for specific Programming Task

The NSA uses Python for cryptography and intelligence analysis

Netflix and Yelp have both documented the role of Python in their software infrastructures

Python Applications



Features of Python

- It provides **rich data types** and **easier to read syntax** than any other programming languages
- It is a **platform independent** scripted language with full access to operating system.
- Compared to other programming languages, it allows more run-time **flexibility**
- Libraries in Pythons are **cross-platform** compatible with Linux, MacIntosh, and Windows
- For building large applications, Python can be compiled to **byte-code.**
- Python supports functional and structured programming as well as OOP
- It supports **interactive mode** that allows interacting Testing and debugging of snippets of code
- In Python, since there is no compilation step, editing, debugging and **testing is fast.**
- It supports **automatic garbage collection.**

Python Frameworks

A framework is a collection of pre-written code designed to simplify development tasks.

It provides a structured way to build and deploy applications.

Benefits of using a framework:

1. Saves time and effort
2. Reduces repetitive tasks
3. Promotes best practices
4. Enhances scalability and maintainability

Python Frameworks

Why Python Frameworks?

1. Python’s versatility: Python can be used for web development, data science, automation, machine learning, etc.

2. Code Reusability: Frameworks offer reusable code, reducing the need to write repetitive code.

3. Faster Development: Built-in tools and libraries accelerate project development.

4. Community Support: Large Python community contributing to libraries and frameworks.

Types of Python Frameworks

Web Development Frameworks:

1. Django
2. Flask
3. FastAPI

Data Science & Machine Learning Frameworks:

1. Pandas
2. Numpy
3. Tensorflow
4. Scikit-learn

GUI Frameworks:

1. Tkinter
2. PuQt
3. Kivy

Django

Django is a high-level Python web framework that promotes rapid development and clean, pragmatic design. It’s one of the most popular web frameworks in the Python ecosystem.

It is known for its "batteries-included" philosophy, meaning it provides a lot of built-in tools and features that help developers build fully functional applications without needing third-party libraries for common tasks.

Use-Cases

1. Large-scale web applications
2. Content management systems
3. E-commerce websites

Flask

Flask is a lightweight, micro web framework designed for simplicity and flexibility. Unlike Django, Flask doesn’t include an ORM, authentication, or other built-in tools. This gives developers more control over the app's components but also means they need to integrate more third-party libraries.
Flask follows the minimalist philosophy, making it a great option for developers who want to create a custom architecture and work with only the components they need.

Use-Cases

1. Small to Medium-Scale Applications
2. Prototyping and MVPs
3. APIs and Microservices

Pandas

Pandas is one of the most widely-used libraries for data manipulation and analysis in Python. It provides fast, flexible, and expressive data structures that make it easy to work with structured data.

Pandas is mainly used for handling tabular data, like data frames, which can be seen as an in-memory 2D database.

Use-Cases

1. Data cleaning and preprocessing
2. Exploratory data analysis (EDA)
3. Data wrangling and reshaping

NumPy

NumPy is a fundamental package for numerical computing in Python. It provides support for arrays, matrices, and a wide range of mathematical functions to manipulate numerical data.

It is the foundation for many other libraries in the data science and machine learning ecosystem.

Use-Cases

1. Scientific computing
2. Handling large datasets and performing complex calculations
3. Statistical analysis and modeling

Tkinter

Tkinter is the standard GUI toolkit that comes bundled with Python. It provides a simple and easy way to create desktop applications with windows, buttons, labels, and other widgets.

Tkinter is based on the Tk GUI toolkit, which is cross-platform and works on Windows, Mac, and Linux.

Use-Cases

1. Small desktop applications
2. Prototyping user interfaces
3. Learning GUI development

Basics of Python

- The Python language has many similarities to Perl, C, and Java.
- Python files have extension **.py.**
- Assuming that you have Python interpreter set in PATH variable.

Execute: python test.py

Python refers to version of the python.

-

Python comments

Comments do not change the outcome of a program, they still play an important role in any programming and not just Python. Comments are the way to improve the readability of a code, by explaining what we have done in code in simple english.

This is especially helpful when someone else has written a code and you are analysing it for bug fixing or making a change in logic, by reading a comment you can understand the purpose of code much faster then by just going through the actual code.

Types of Comments in Python:
There are two types of comments in Python.
1. Single line comment
2. Multiple line comment

Basics of Python

Single line comment

In python we use # special character to start the comment.

Ex:- '#' This is just a comment. Anything written here is ignored by Python

Multi-line comment:

To have a multi-line comment in Python, we use triple single quotes at the beginning and at the end of the comment, as shown below.

- Ex:-

```
'''
This is a
multi-line
comment
'''
```

Python Keywords and Identifiers

Python Keywords

Keywords are the reserved words in Python.

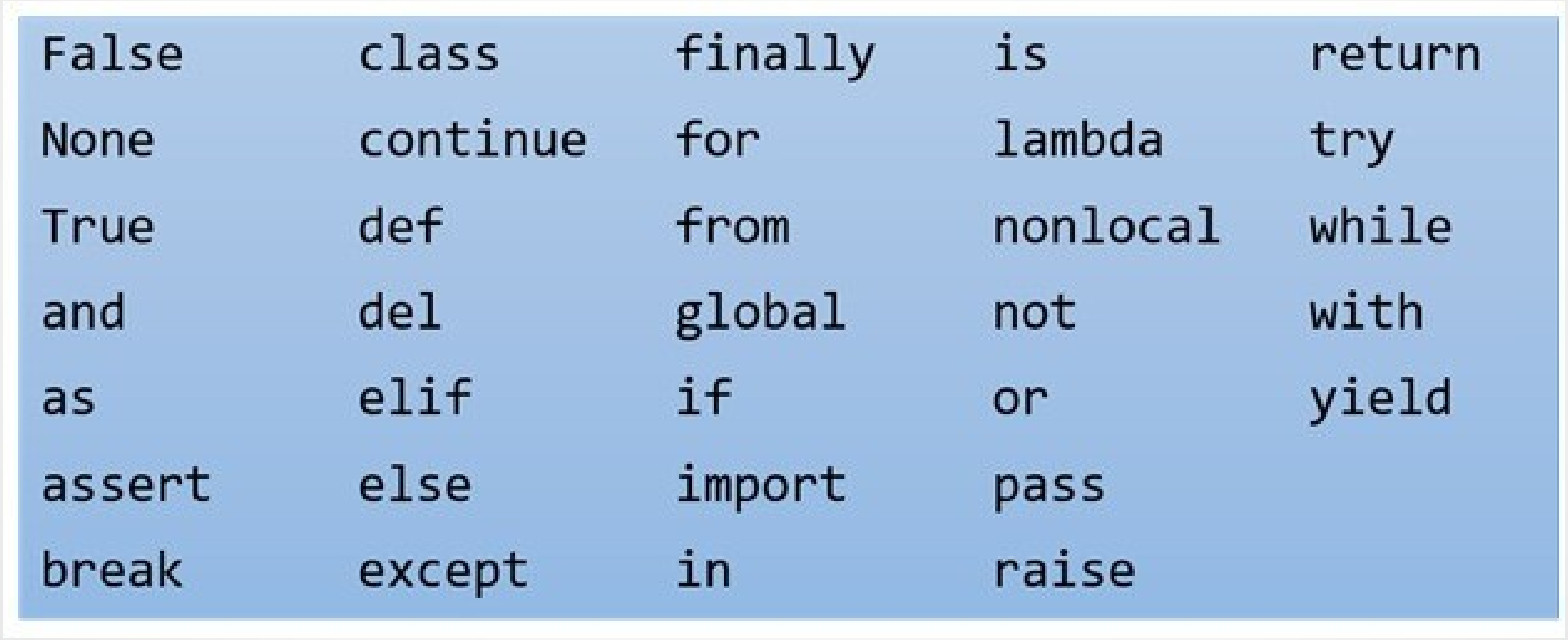
We cannot use a keyword as a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.

In Python, keywords are case sensitive. There are 33 keywords in Python 3.7.

All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords is given below.

Basics of Python

Reserve Keywords



False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Python Identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

Rules for writing identifiers:-

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _. Names like myClass, var_1 and print_this_to_screen, all are valid example.
- An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
- Keywords cannot be used as identifiers.
- We cannot use special symbols like !, @, #, $, % etc. in our identifier.
- Identifier can be of any length.

Basics of Python

Python Identifiers

- Here are naming conventions for Python identifiers −
  - Starting an identifier with a single leading underscore indicates that the identifier is private.
  - Starting an identifier with two leading underscores indicates a strongly private identifier.
  - If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Basics of Python

Python Variables :- A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming. For example: number =10

Variable name is known as identifier. There are few rules that you have to follow while naming the variables in Python.

1. The name of the variable must always start with either a letter or an underscore (_). For example: _str, str, num, _num are all valid name for the variables.

2. The name of the variable cannot start with a number.
For example: 9num is not a valid variable name.

3. The name of the variable cannot have special characters such as %, $, # etc, they can only have alphanumeric characters and underscore (A to Z, a to z, 0-9 or _).

4. Variable name is case sensitive in Python which means num and NUM are two different variables in python.

Basics of Python

Variable Naming Rules

- Must begin with a letter (a - z, A - B) or underscore (_)
- Other characters can be letters, numbers or _
- Case Sensitive
- Reserved words cannot be used as a variable name
-

Equalto sign

- Python variables do not need explicit declaration to reserve memory space.
- The equal to sign creates new variables and gives them values.
- <variable> = <expr>.
- It read right to left only.
- **Multiple assignment:**
  - a = b = c = 1
  - a,b,c = 1,2,"john"

Basics of Python

Standard Data Types: Data type defines the type of the variable, whether it is an integer variable, string variable, tuple, dictionary, list etc.

Python data types

Python data types are divided in two categories, mutable data types and immutable data types.

Immutable Data types in Python

1. Numeric
2. String
3. Tuple

Mutable Data types in Python

1. List
2. Dictionary
3. Set

•

Basics of Python

Numbers

Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python.

- int (signed integers)
  - long (long integers, they can also be represented in octal and hexadecimal)
  - float (floating point real values)
  - complex (complex numbers)

- Python displays long integers with an uppercase L.
- A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit.

- We can use the type() function to know which class a variable or a value belongs to

Basics of Python

Boolean (bool)

- It represents the truth values False and True.

String

- String is sequence of Unicode characters.
- Follows (left-to- right order) of characters
- We can use single quotes or double quotes to represent strings.
- Multi-line strings can be denoted using triple quotes, '''.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

h	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

Basics of Python

List

- List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible.
- It is similar to array in C only difference is that elements it contains have different data type.
- We can use the slicing operator [ ] to extract an item from a list. Index starts form 0 in Python.

List Examples

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

List = [0, 1, 2, 3, 4, 5]

0	**1**	**2**	**3**	**4**	**5**

List[0] = 0

List[1] = 1

List[2] = 2

List[3] = 3

List[4] = 4

List[5] = 5

List[0:] = [0,1,2,3,4,5]

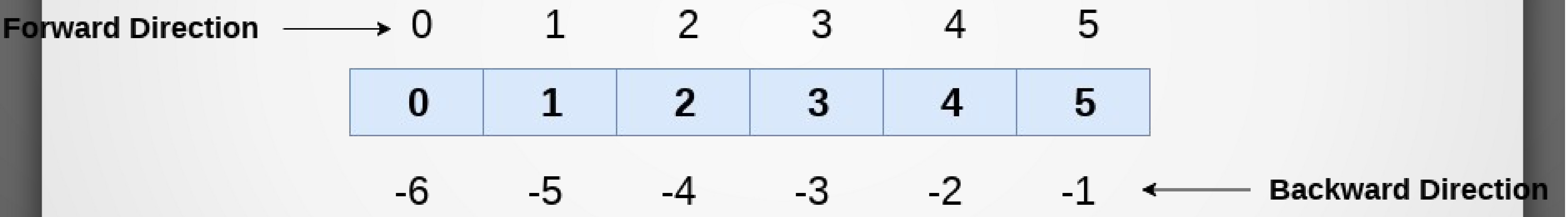
List[:] = [0,1,2,3,4,5]

List[2:4] = [2, 3]

List[1:3] = [1, 2]

List[:4] = [0, 1, 2, 3]

List = [0, 1, 2, 3, 4, 5]


Forward Direction
0 1 2 3 4 5
0 1 2 3 4 5
-6 -5 -4 -3 -2 -1
Backward Direction

Tuple

A tuple is another sequence data type that is similar to the list.

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.

Tuples are read-only lists.

Tuples are used to write-protect data so it is faster than list as it cannot change dynamically.

Basics of Python

Dictionary

- Dictionary is an unordered collection of key-value pairs.
- They work like associative arrays or hashes found in Perl and consist of key-value pairs.
- A dictionary key can be almost any Python type, but are usually numbers or strings.
- Key and value can be of any type.
- It is generally used when we have a huge amount of data.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

```
dict['Name']:  Zara
dict['Age']:  7
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

```
dict['Age']:  8
dict['School']:  DPS School
```

There are two important points to remember about dictionary keys −
(a) More than one entry per key not allowed. Which means no duplicate key is allowed
. When duplicate keys encountered during assignment, the last assignment wins.

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}
print "dict['Name']: ", dict['Name']
When the above code is executed, it produces the following
result −
dict['Name']:  Manni
```

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

```
dict = {['Name']: 'Zara', 'Age': 7}
print "dict['Name']: ", dict['Name']
```

Basics of Python

Input/Output

- Functions like **input() and print()** are widely used for standard input and output operations respectively.

- **print() function**
  - It is used to output data to the standard output device (screen).
  - You can write print(argument) and this will print the argument in the next line when you press the ENTER key
  - To format our output str.format() method is used.

  - Syntax of print():
  - print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

Basics of Python

print() Parameters

objects - object to the printed. * indicates that there may be more than one object
sep - objects are separated by sep. Default value: ' '
end - end is printed at last
file - must be an object with write(string) method. If omitted it, sys.stdout will be used which prints objects on the screen.
flush - If True, the stream is forcibly flushed. Default value: False

Note: sep, end, file and flush are keyword arguments.

Basics of Python

- print(1,2,3,4)
- # Output: 1 2 3 4

- print(1,2,3,4, sep ='*')
- # Output: 1*2*3*4

- print(1,2,3,4,sep='#',end='&')
- # Output: 1#2#3#4&

Basics of Python

The input() is used to take the input from the user.

- **Syntax: input([prompt])**

where prompt is the string we wish to display on the screen. It is optional.

In Python 2, you have a built-in function raw_input(), whereas in Python 3, you have input().

```
num = input('Enter a number: ')

Enter a number: 10
'10'
```

Basics of Python

Import

- When program size increases it is divided into different modules.
- A module is a file containing Python definitions and statements.
- Definitions inside a module can be imported to another module, **import** keyword to do this.
- Syntax

```
import module_name
```

- **We can also import some specific attributes and functions only, using the *from* keyword.**

Basics of Python

Operators

- Operators are special symbols which represent symbols and perform arithmetic and logical computations.
- The values the operator uses are called operands.

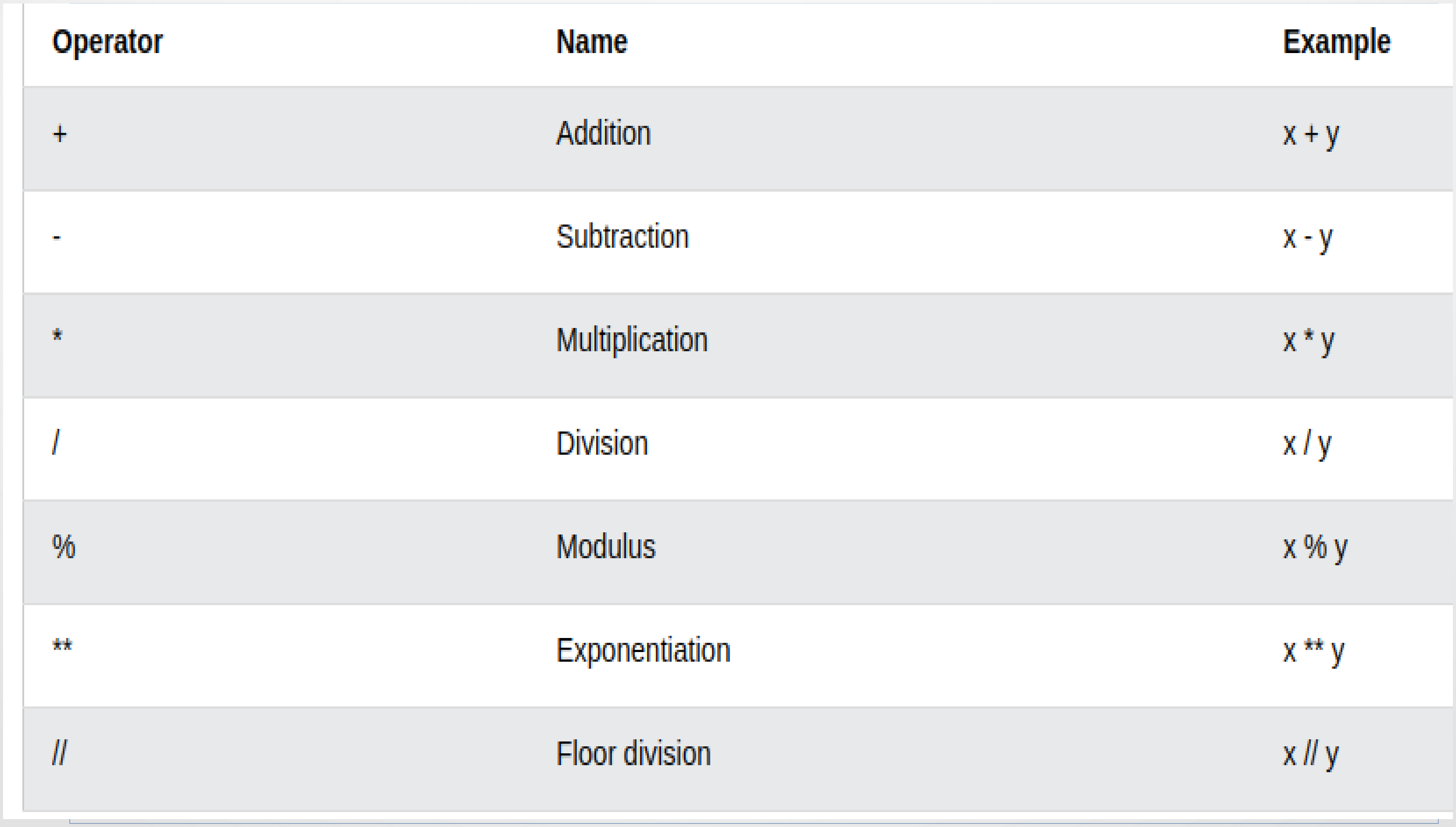
- **Python supports following types of Operators:**

  - Arithmetic Operators
  - Comparison/Relational Operators
  - Assignment Operators
  - Logical Operators
  - Membership Operators
  - Identity Operators
  - Bitwise Operators

Basics of Python

Arithmetic Operators:

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

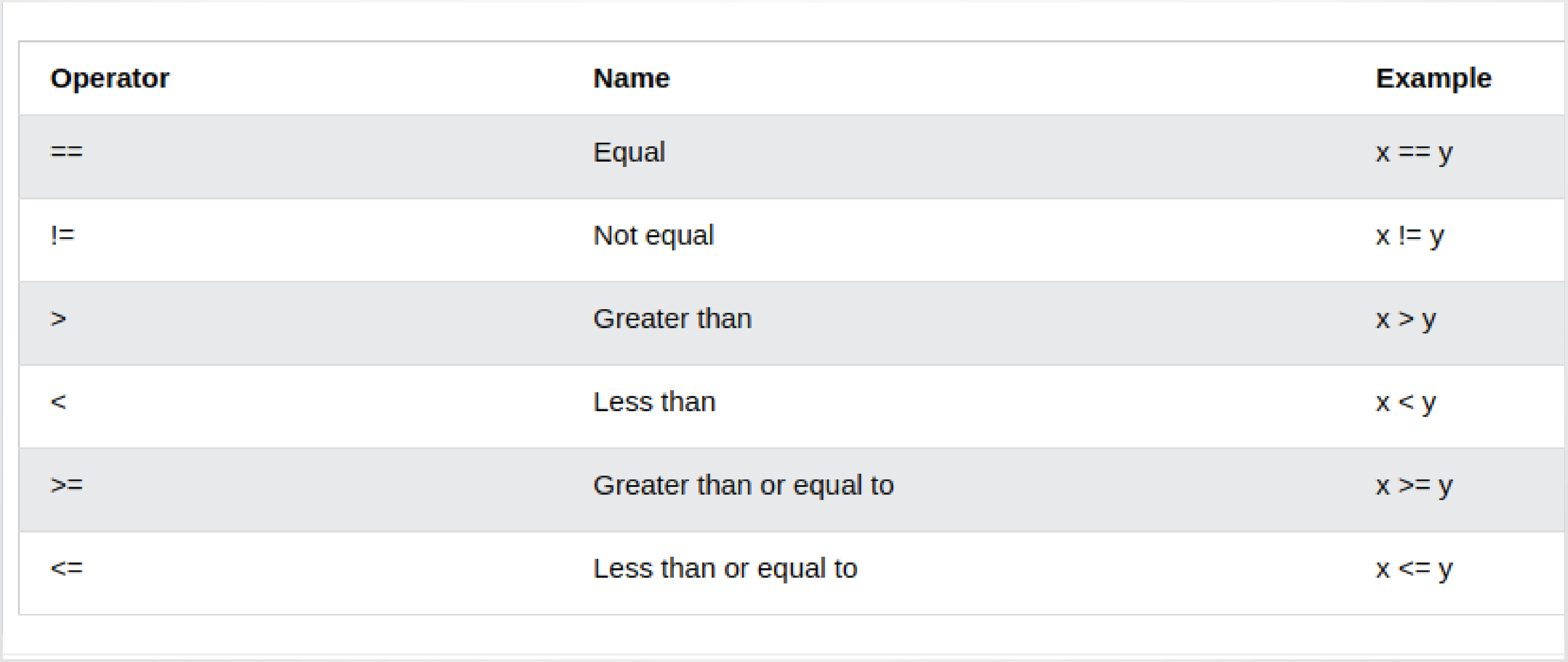


Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Basics of Python

Comparison Operators:

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

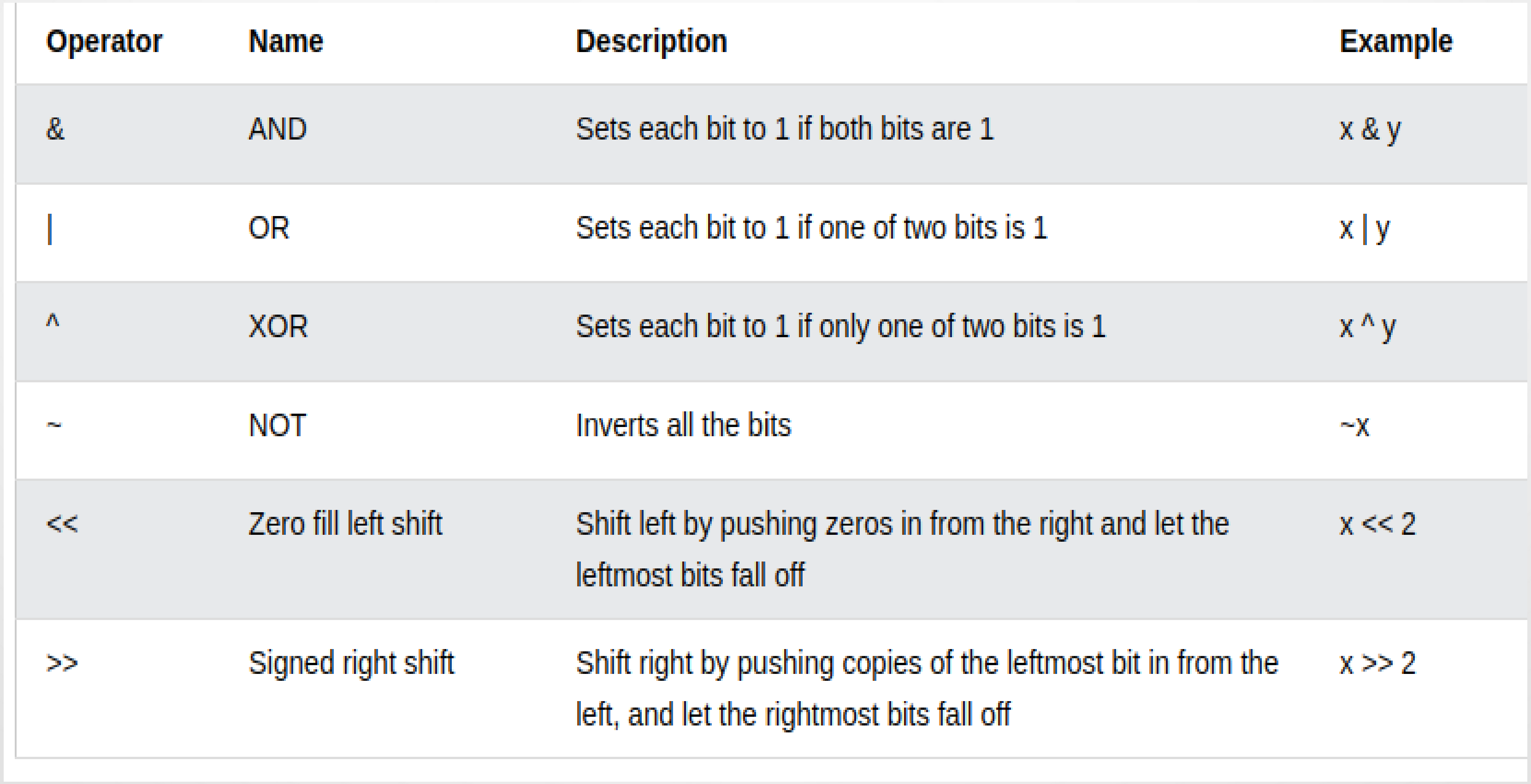


Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Basics of Python

Bitwise Operators:

Bitwise operator works on bits and performs bit by bit operation.

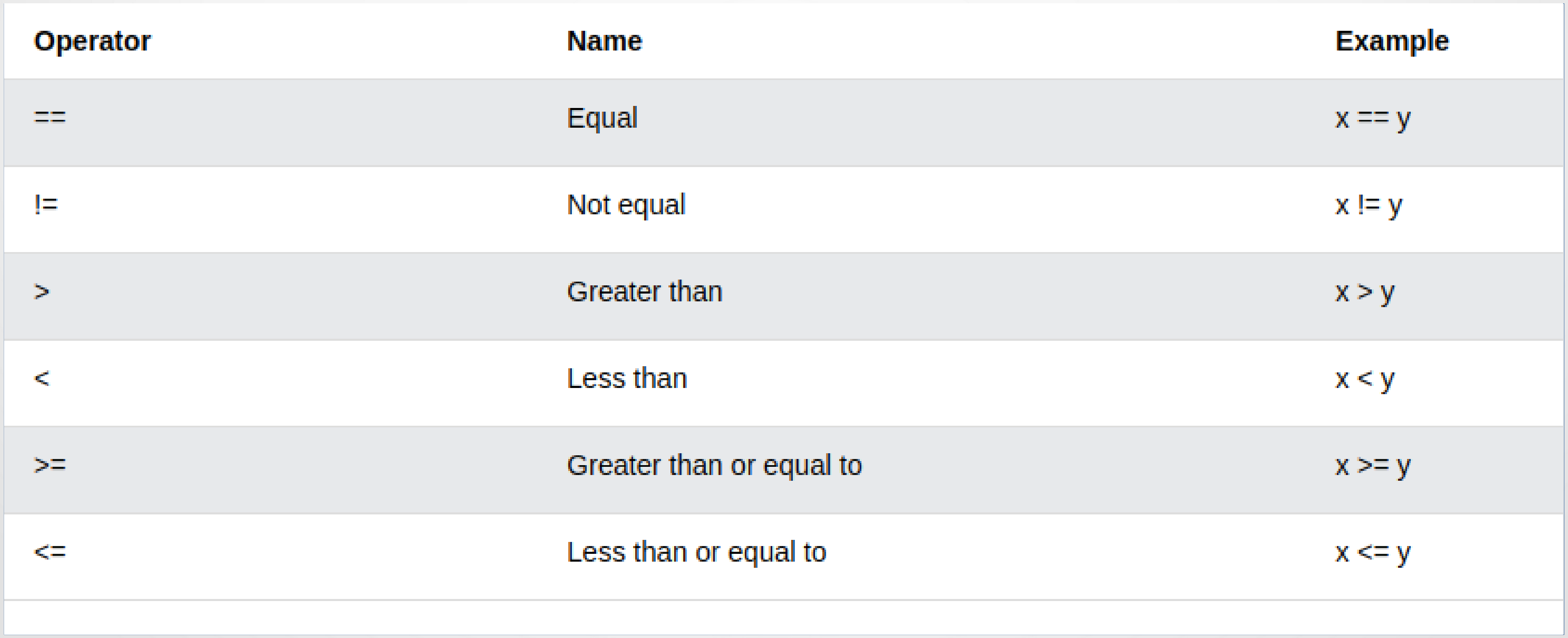


Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and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

Basics of Python

Comparison Operators:

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

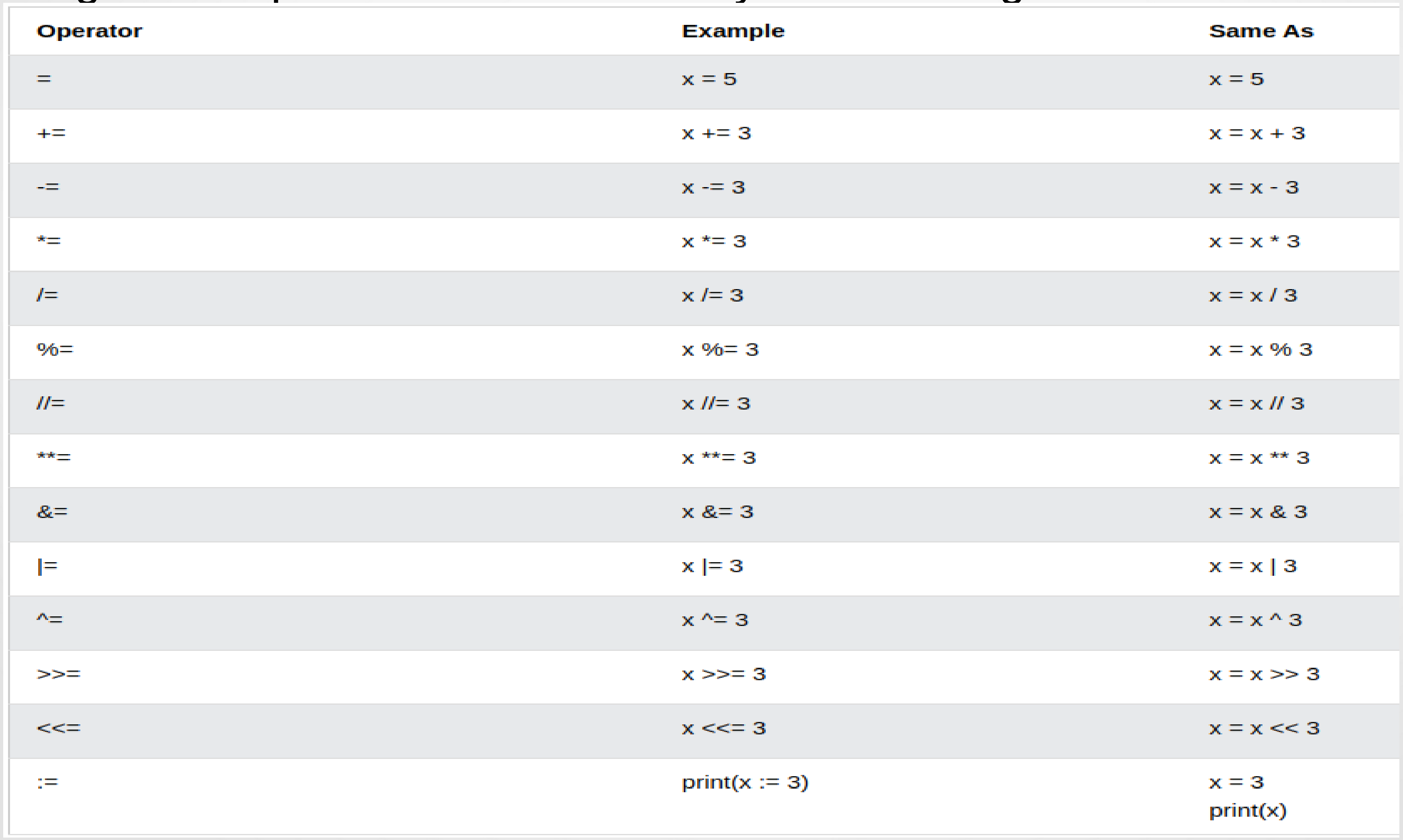


Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Basics of Python

Assignment Operators:

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

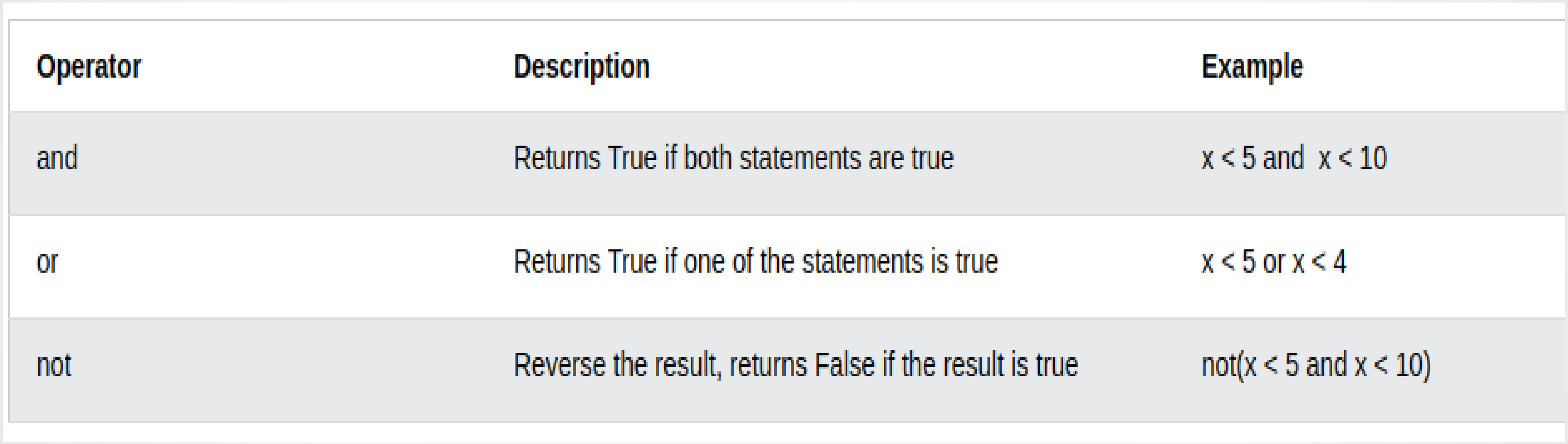


Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

:=	print(x := 3)	x = 3
		print(x)

Basics of Python

Logical Operators:

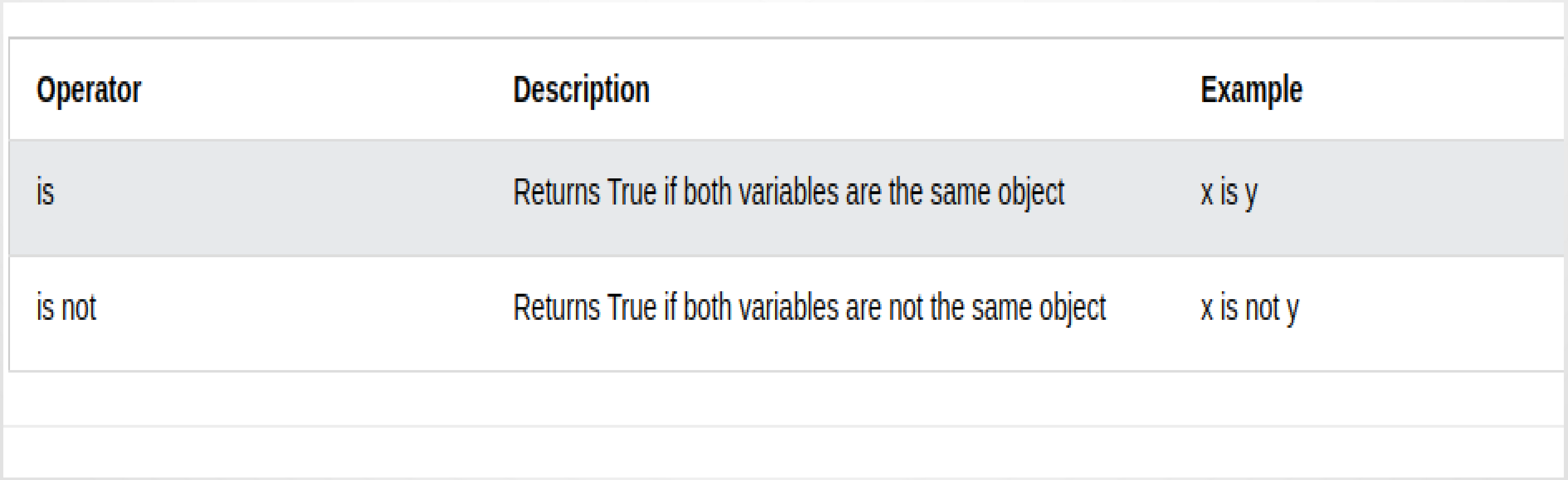


Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Basics of Python

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.

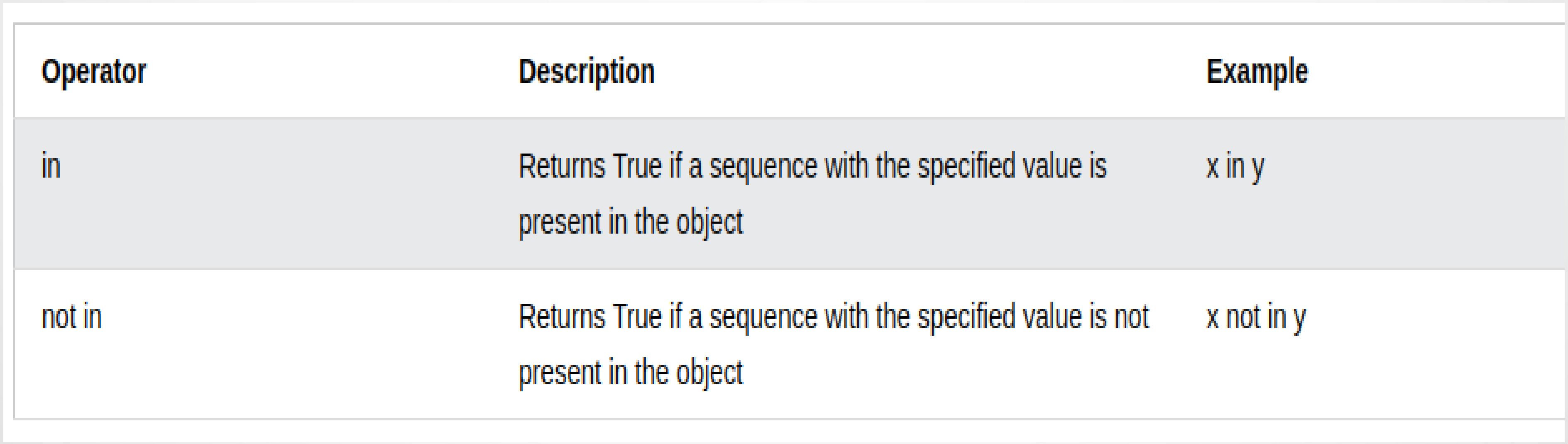


Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Basics of Python

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).



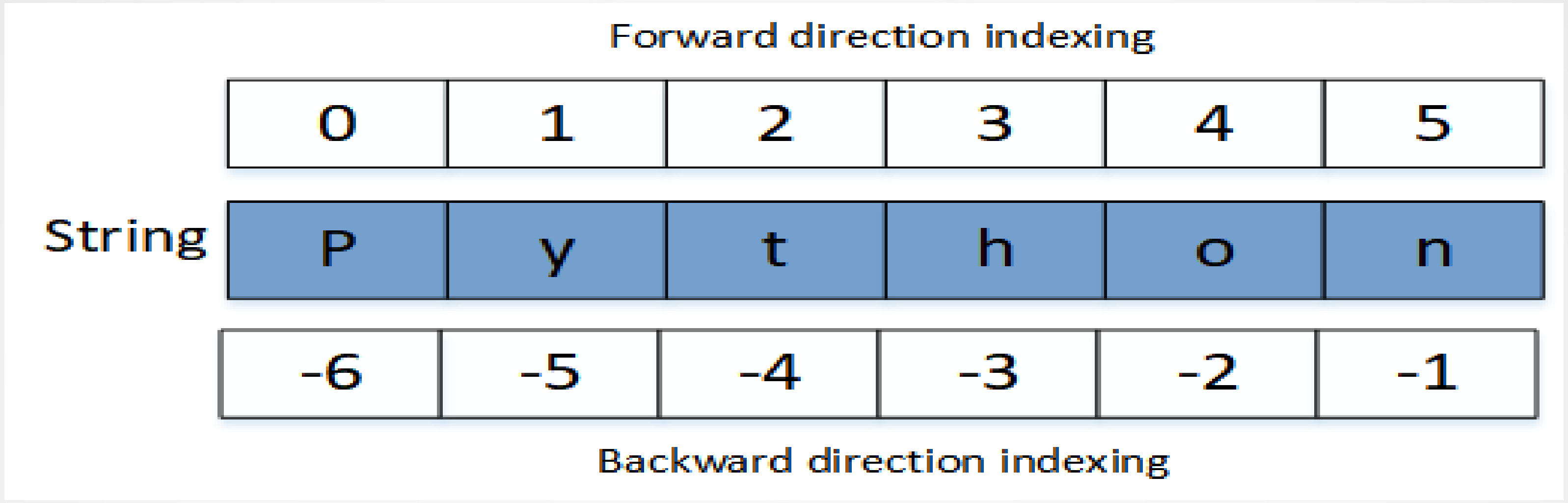
Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

String

Creating String

- A string is a sequence of characters.
- Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.
  - var1 = 'Hello World!'
  - var2 = "Python Programming"
- In Python, Strings are stored as individual characters in a contiguous memory location.
- The benefit of using String is that it can be accessed from both the directions in forward and backward.
- Both forward as well as backward indexing are provided using Strings in Python.
  - Forward indexing starts with 0,1,2,3,....
  - Backward indexing starts with -1,-2,-3,-4,....

String



-

- str[0] = 'P' = str[-6]
- str[1] = 'Y' = str[-5]
- str[2] = 'T' = str[-4]
- str[3] = 'H' = str[-3]
- str[4] = 'O' = str[-2]
- str[5] = 'N' = str[-1].

String

String Functions

- Capitalize
- Count
- Endswith
- Find
- Index
- Isalnum
- Isalpha
- Isdigit
- Islower
- Isupper
- Isspace
- len
- Lower
- Upper

String Functions

- Startwith
- Swapcase
- Lstrip
- Rstrip

String Functions

Capitalize():
It capitalizes the first character of the String.

    Syntax:

```
str.capitalize()
```

- Example:

```
str = "this is string example";
print "str.capitalize() : "
```

Output:

```
This is string example
```

String

count(string,begin,end):
Counts number of times substring occurs in a String between begin and end index.

- **Syntax:**
- str.count(sub, start= 0,end=len(string))

Parameters

sub − This is the substring to be searched.
start − Search starts from this index. First character starts from 0 index. By default search starts from 0 index.
end − Search ends from this index. First character starts from 0 index. By default search ends at the last index.

```
str = "this is string example....";
sub = "i";
print "str.count(sub, 4, 40) : ", //2
sub = "wow";
print "str.count(sub) : " //1
```

String

endswith(suffix ,begin=0,end=n):

Returns a Boolean value if the string terminates with given suffix between begin and end.

Syntax

    str.endswith(suffix[, start[, end]])

Parameters

suffix − This could be a string or could also be a tuple of suffixes to look for.

start − The slice begins from here.

end − The slice ends here

```
str = "this is string example....wow!!!";
suffix = "wow!!!";
print str.endswith(suffix) //True
print str.endswith(suffix,20) //True
suffix = "is";
print str.endswith(suffix, 2, 4) //True
```

String

find(substring ,beginIndex, endIndex):

It returns the index value of the string where substring is found between begin index and end index. It returns index if it is found else -1.

Syntax

str.find(str, beg=0, end=len(string))

Parameters

str − This specifies the string to be searched.

beg − This is the starting index, by default its 0.

end − This is the ending index, by default its equal to the length of the string.

```
str1 = "this is string example....";
str2 = "exam";

print str1.find(str2) #15
print str1.find(str2, 10) #15
print str1.find(str2, 40) #-1
```

String

index(subsring, beginIndex, endIndex):
Same as find() except it raises an exception if string is not found.
Syntax
str.index(str, beg = 0 end = len(string))
Parameters
    str − This specifies the string to be searched.
    beg − This is the starting index, by default its 0.
    end − This is the ending index, by default its equal to the length of the string.

```
str1 = "this is string example....";
str2 = "exam";

print str1.index(str2) #15
print str1.index(str2, 10) #15
print str1.index(str2, 40) #Error
```

String

isalnum():

It returns True if characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise it returns False.

Syntax:

```
str.isalnum()

str = "ty2018";  # No space in this string True
print str.isalnum()

str = "this is string example...; #False
print str.isalnum()
```

String

Isalpha():

It returns True when all the characters are alphabets and there is at least one character, otherwise False.

Syntax

```
str.isalpha()

str = "this";  # No space & digit in this string True
print str.isalpha()

str = "this is string example....wow!!!"; False
print str.isalpha().
```

String

Isdigit():

It returns True if all the characters are digit and there is at least one character, otherwise False.

Syntax

```
str.isdigit()

str = "123456";  # Only digit in this string
print str.isdigit() #True

str = "this is string example";
print str.isdigit() #False
```

String

islower():

It returns True if the characters of a string are in lower case, otherwise False.

Syntax

```
str.islower()

str = "THIS is string example..";
print str.islower() #False

str = "this is string example..";
print str.islower() #True
```

String

isupper():

It returns False if characters of a string are in Upper case, otherwise False.

Syntax

```
str.isupper()

str = "THIS IS STRING EXAMPLE..";
print str.isupper() #True

str = "THIS is string example..";
print str.isupper() #False
```

String

isspace():
It returns True if the characters of a string are whitespace, otherwise false.
Syntax
str.isspace()

```
str = "       ";
print str.isspace() #True

str = "This is string example";
print str.isspace() #False
```

String

len(string):

len() returns the length of a string.

Syntax

len(str)

```
str = "this is string example..";
print "Length of the string: ", len(str)
```

String

Lower():

Converts all the characters of a string to Lower case.

Syntax

```
str.lower()

str = "THIS IS STRING EXAMPLE..";
print str.lower() #This is string example
```

String

Upper():
Converts all the characters of a string to Upper Case.
Syntax
str.upper()

```
str = "this is string example..";
print "str capital : ", str.upper() #THIS IS STRING EXAMPLE..
```

String

startswith(str ,begin=0,end=n):

Returns a Boolean value if the string starts with given str between begin and end.

Syntax

str.startswith(str, beg=0,end=len(string));

Parameters

str − This is the string to be checked.

beg − This is the optional parameter to set start index of the matching boundary.

end − This is the optional parameter to end start index of the matching boundary.

```
str = "this is string example..";
print str.startswith( 'this' ) #True
print str.startswith( 'is', 2, 4 ) #True
print str.startswith( 'this', 2, 4 ) #False
```

String

Swapcase():

Inverts case of all characters in a string.

Syntax

```
str.swapcase();

str = "this is string example..";
print str.swapcase() # THIS IS STRING EXAMPLE..

str = "THIS IS STRING EXAMPLE..";
print str.swapcase() #this is string example..
```

String

lstrip():
Remove all leading whitespace of a string. It can also be used to remove particular character from leading.
Syntax
str.lstrip([chars])

Parameters
chars − You can supply what chars have to be trimmed.

```
str = "     this is string example...     ";
print str.lstrip()
str = "88888888this is string example..8888888";
print str.lstrip('8')
```

Output
```
this is string example..
this is string example..8888888
```

String

rstrip():

Remove all trailing whitespace of a string. It can also be used to remove particular character from trailing.

Syntax

str.rstrip([chars])

Parameters

chars − You can supply what chars have to be trimmed.

```
str = "     this is string example..     ";
print str.rstrip()
str = "88888888this is string example..8888888";
print str.rstrip('8')
```

output

```
this is string example..
88888888this is string example..
```

String

Math and Comparison

- **Adding Strings Together:**
  - fname = “GLS”
  - lname = “BCA”
  - **print(fname+lname)**
- **Multiplication(aestrick)**:
  - s = “hello”
  - print(s * 5)
  - print(s * -5)   #returns empty string when negative number is passed.
  - print(s * 5.5)   #float gives an error
- **Comparing Strings:**
  - a1 = "hello "
  - a2 = "Hello"
  - a1 == a2   # returns false as Python is case sensitive

String

Formatting Strings

- Python uses C-style string formatting to create new, formatted strings. The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string, which contains normal text together with "argument specifiers", special symbols like "%s" and "%d".
- **The string format() method formats the given string in Python.**
- **Syntax:**
- template.format(p0, p1, ..., k0=v0, k1=v1, ...)

  - Here, p0, p1,... are positional arguments and, k0, k1,... are keyword arguments

-

String

Formatting Strings:

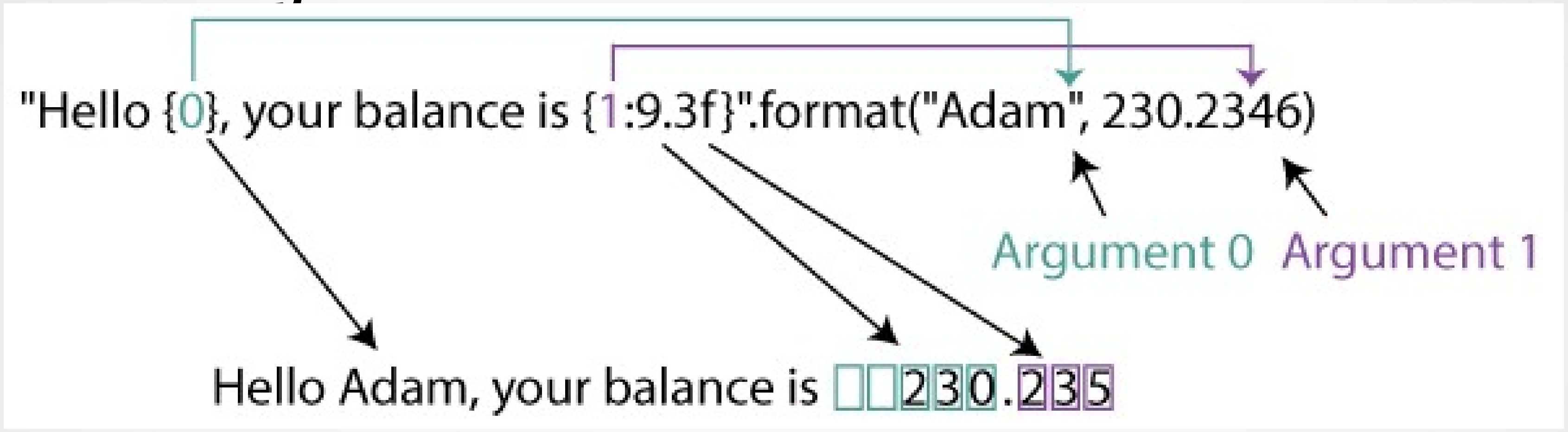
format() method takes any number of parameters. But, is divided into two types of parameters:

- **Positional parameters -** list of parameters that can be accessed with index of parameter inside curly braces {index}
- **Keyword parameters -** list of parameters of type key=value, that can be accessed with key of parameter inside curly braces {key}

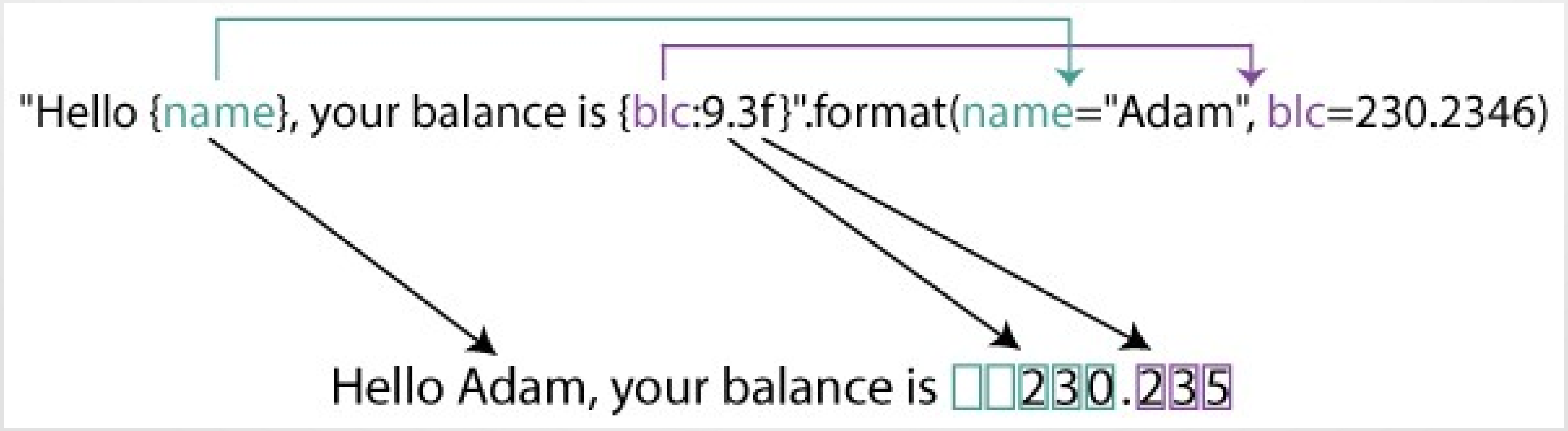
String

Formatting Strings:

Positional Argument


"Hello {0}, your balance is {1:9.3f}".format("Adam", 230.2346)
Argument 0
Argument 1
Hello Adam, your balance is □□230.235

- **Keyword Argument**


"Hello {name}, your balance is {blc:9.3f}".format(name="Adam", blc=230.2346)
Hello Adam, your balance is □□230.235

String

Formatting Strings:

```
# default arguments
print("Hello {}, your balance is {}.".format("Adam", 230.2346))

# positional arguments
print("Hello {0}, your balance is {1}.".format("Adam", 230.2346))

# keyword arguments
print("Hello {name}, your balance is {blc}.".format(name="Adam",
blc=230.2346))

# mixed arguments
print("Hello {0}, your balance is {blc}.".format("Adam", blc=230.2346))
```

Output:

```
Hello Adam, your balance is 230.2346.
```

Formatting with alignment

Type	**Meaning**
<	Left aligned to the remaining space
^	Center aligned to the remaining space
>	Right aligned to the remaining space
=	Forces the signed (+) (-) to the leftmost position

String formatting with format()

```
# string padding with left alignment
print("{:5}".format("cat"))            c    a    t

# string padding with right alignment
print("{:>5}".format("cat"))
                                                  c    a    t

# string padding with center alignment
print("{:^5}".format("cat"))
                                             c    a    t

# string padding with center alignment
# and '*' padding character
print("{:*^5}".format("cat"))         *    c    a    t    *
```

Truncating strings with format()

```
# truncating strings to 3 letters
print("{:.3}".format("caterpillar"))
                    c     a     t

# truncating strings to 3 letters
# and padding
print("{:5.3}".format("caterpillar"))
                    c     a     t

# truncating strings to 3 letters,
# padding and center alignment
print("{:^5.3}".format("caterpillar"))
                          c     a     t
```

String

Conversion Functions

- **int():** string, floating point → integer
- **float():** string, integer → floating point number
- **str():** integer, float, list, tuple, dictionary → string
- **list():** string, tuple, dictionary → list
- **tuple():** string, list → tuple

Range Function

- The built-in range function is very useful to generate sequences of numbers in the form of a list.
- The range() constructor returns an immutable sequence object of integers between the given start integer to the stop integer.

  - **range() constructor has two forms of definition:**
    - range(stop)
    - range(start, stop[, step])

- **range() takes mainly three arguments having the same use in both definitions**
  - **start -** integer starting from which the sequence of integers is to be returned
  - **stop -** integer before which the sequence of integers is to be returned. The range of integers end at stop - 1.
  - **step (Optional) -** integer value which determines the increment between each integer in the sequence.

Range Function

```
x = range(3, 6)
for n in x:
  print(n)  # 3 4 5

x = range(3, 20, 2)
for n in x:
  print(n) # 3 5 7 9 11 13 15 17 19
```