GLS UNIVERSITY - BCA

SEM – VI

0301601 – INTRODUCTION TO PYTHON

Dr. Disha Shah
Dr. Poonam Dang

Unit – I
Introduction to Python

Contents:

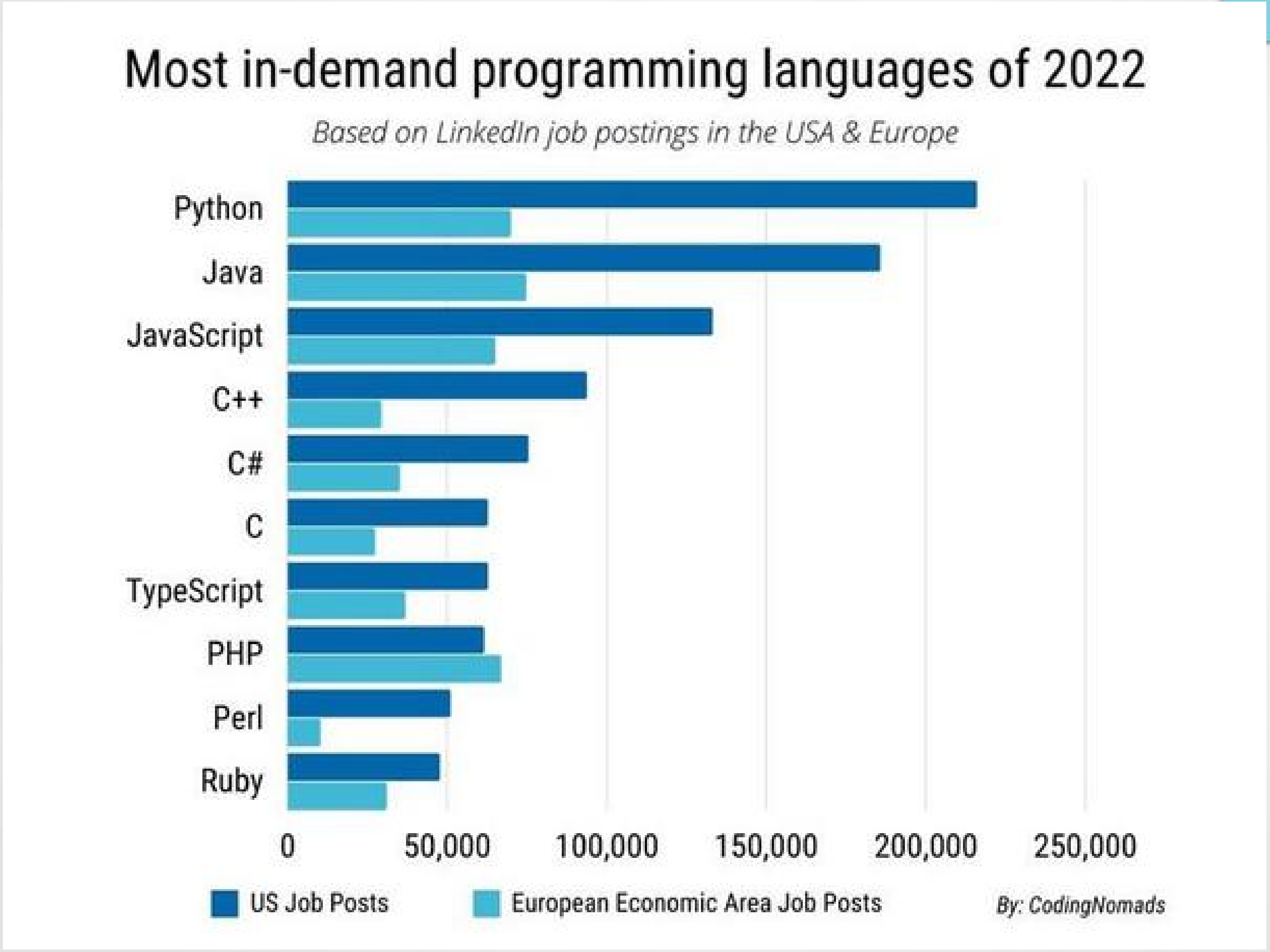
Contents:

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.


Most in-demand programming languages of 2022
Based on LinkedIn job postings in the USA & Europe
Python
Java
JavaScript
C++
C#
C
TypeScript
PHP
Perl
Ruby
0
50,000
100,000
150,000
200,000
250,000
US Job Posts
European Economic Area Job Posts
By: CodingNomads

Introduction to Python

- 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.

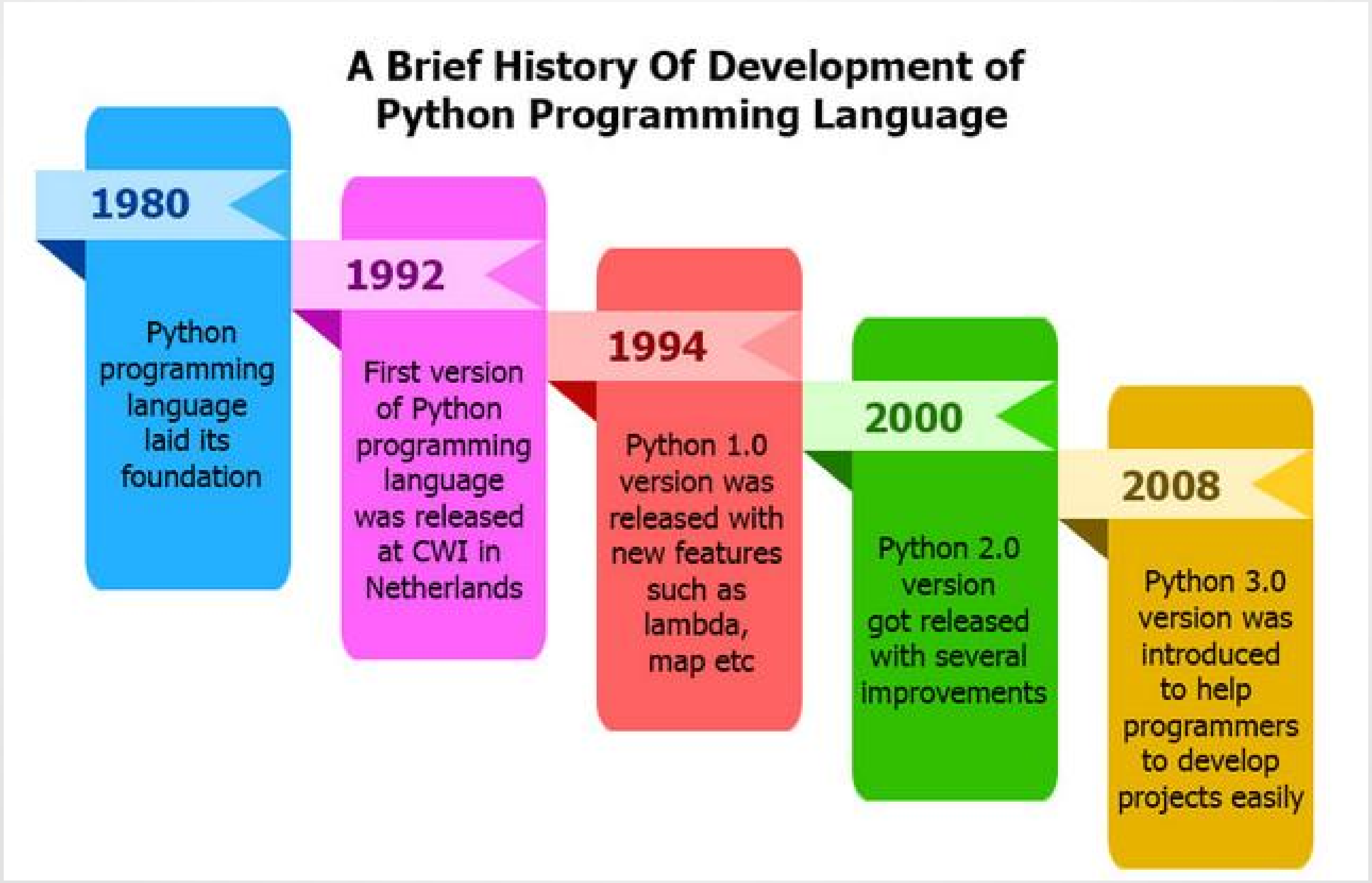
Python was created as a successor of a language called ABC (All Basic Code) and released publicly in 1991 and version 1.0 was released in 1994.

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 – Python 3.11.0 - Oct. 24, 2022

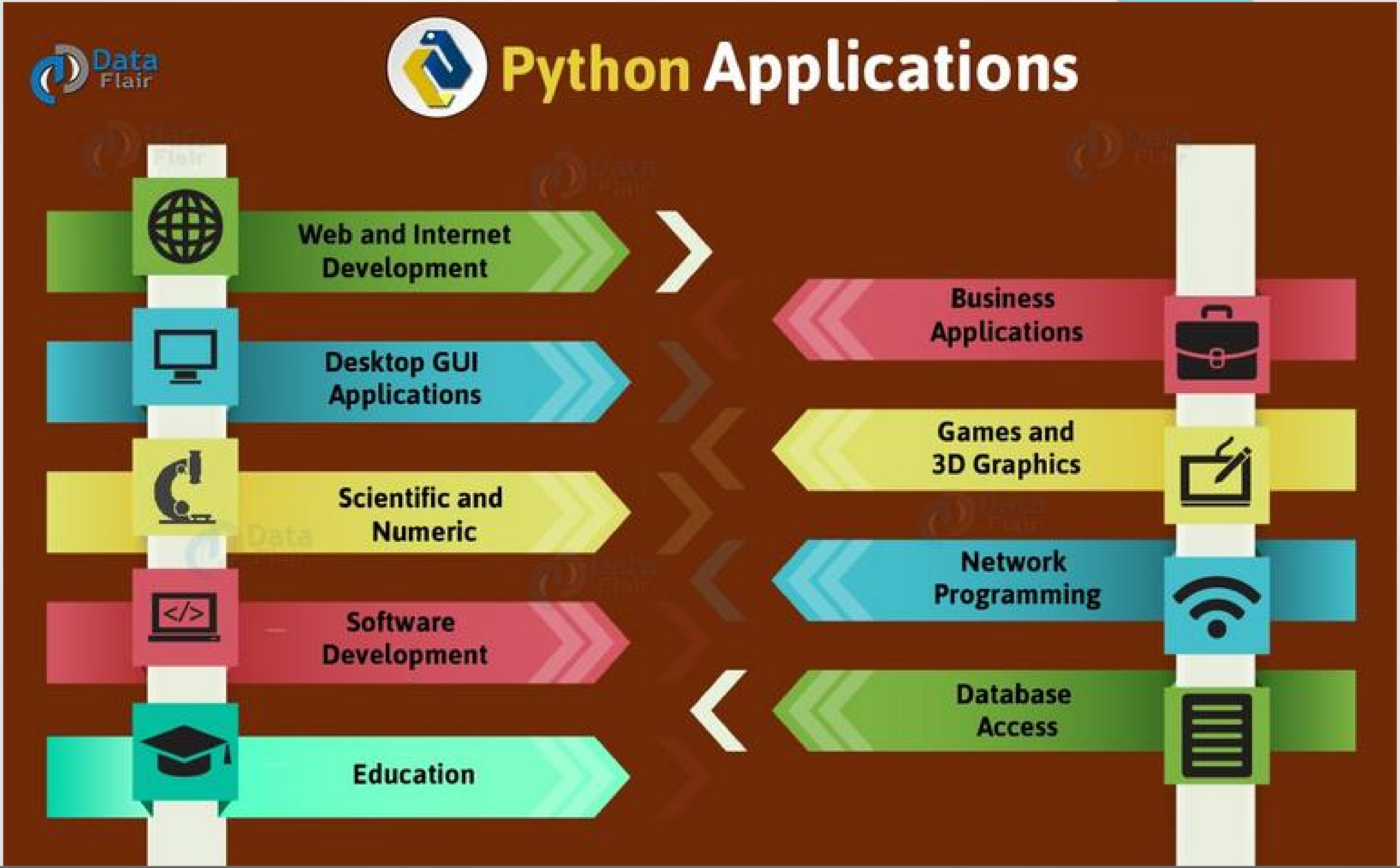
Python is named after the comedy television show Monty Python's Flying Circus. It is not named after the Python snake.

History of Python

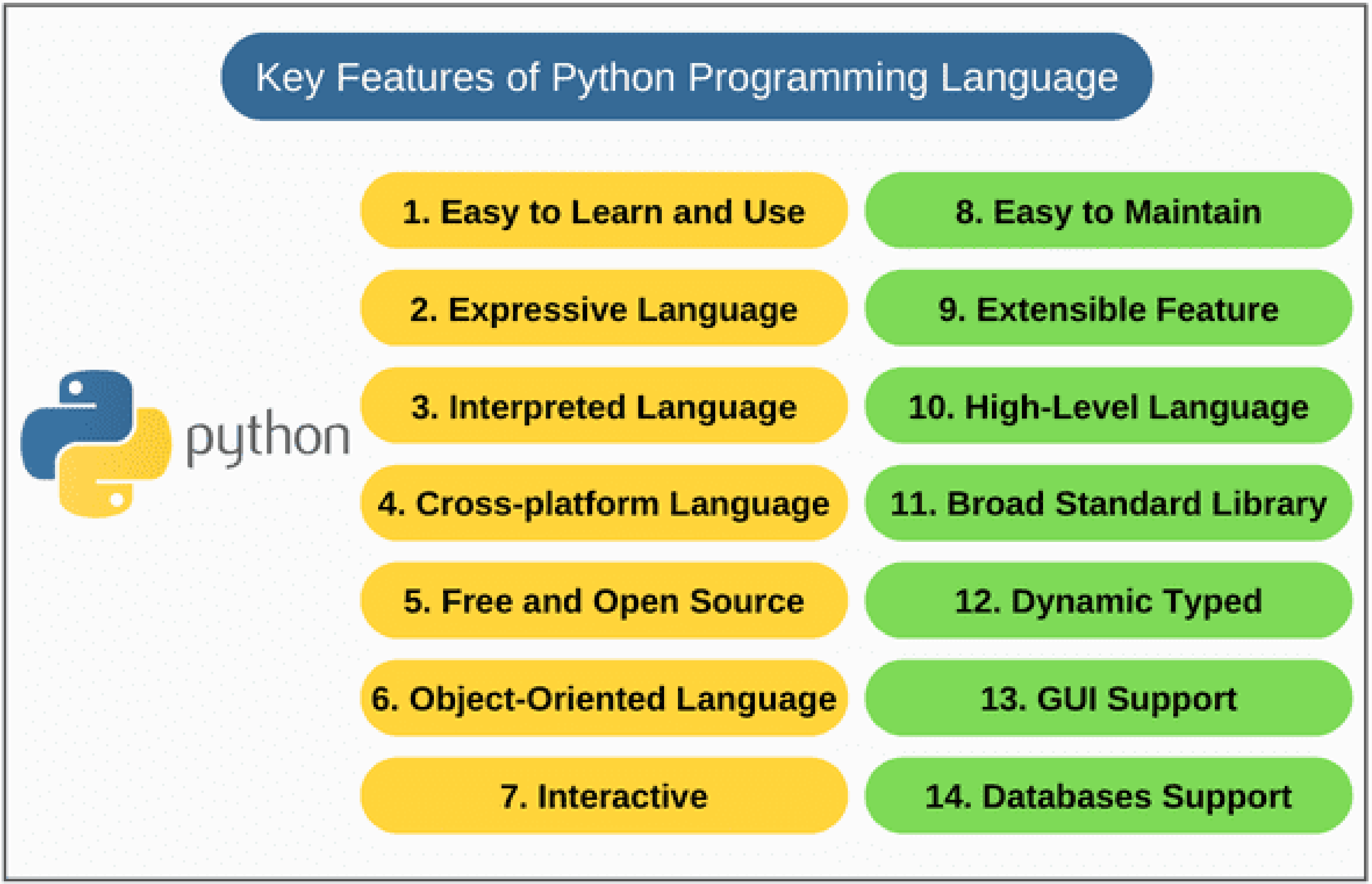

A Brief History Of Development of Python Programming Language
1980
Python programming language laid its foundation
1992
First version of Python programming language was released at CWI in Netherlands
1994
Python 1.0 version was released with new features such as lambda, map etc
2000
Python 2.0 version got released with several improvements
2008
Python 3.0 version was introduced to help programmers to develop projects easily

Where Python is used?

Applications of Python



Features of Python



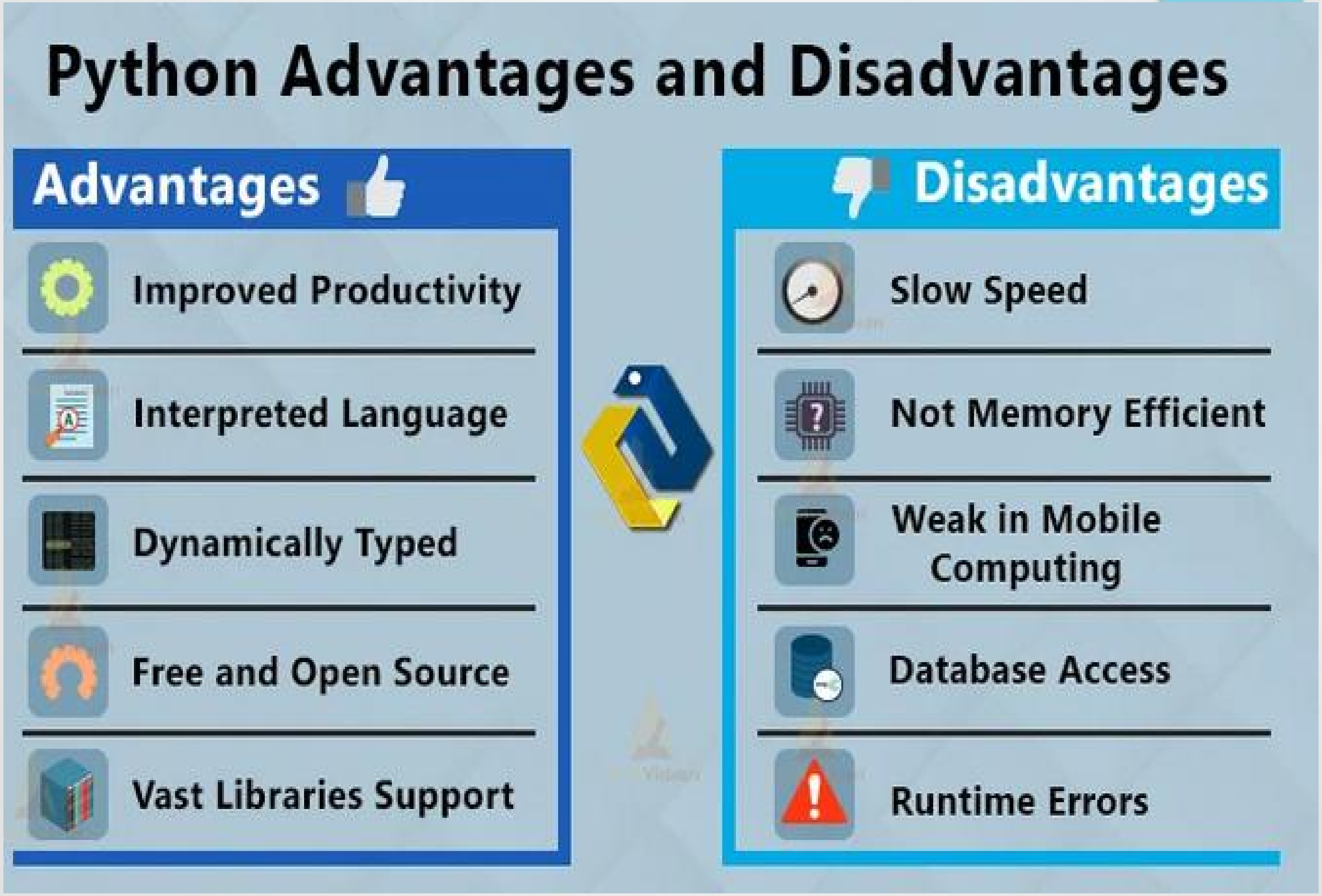
Key Features of Python Programming Language

1. Easy to Learn and Use
2. Expressive Language
3. Interpreted Language
4. Cross-platform Language
5. Free and Open Source
6. Object-Oriented Language
7. Interactive
8. Easy to Maintain
9. Extensible Feature
10. High-Level Language
11. Broad Standard Library
12. Dynamic Typed
13. GUI Support
14. Databases Support

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.**

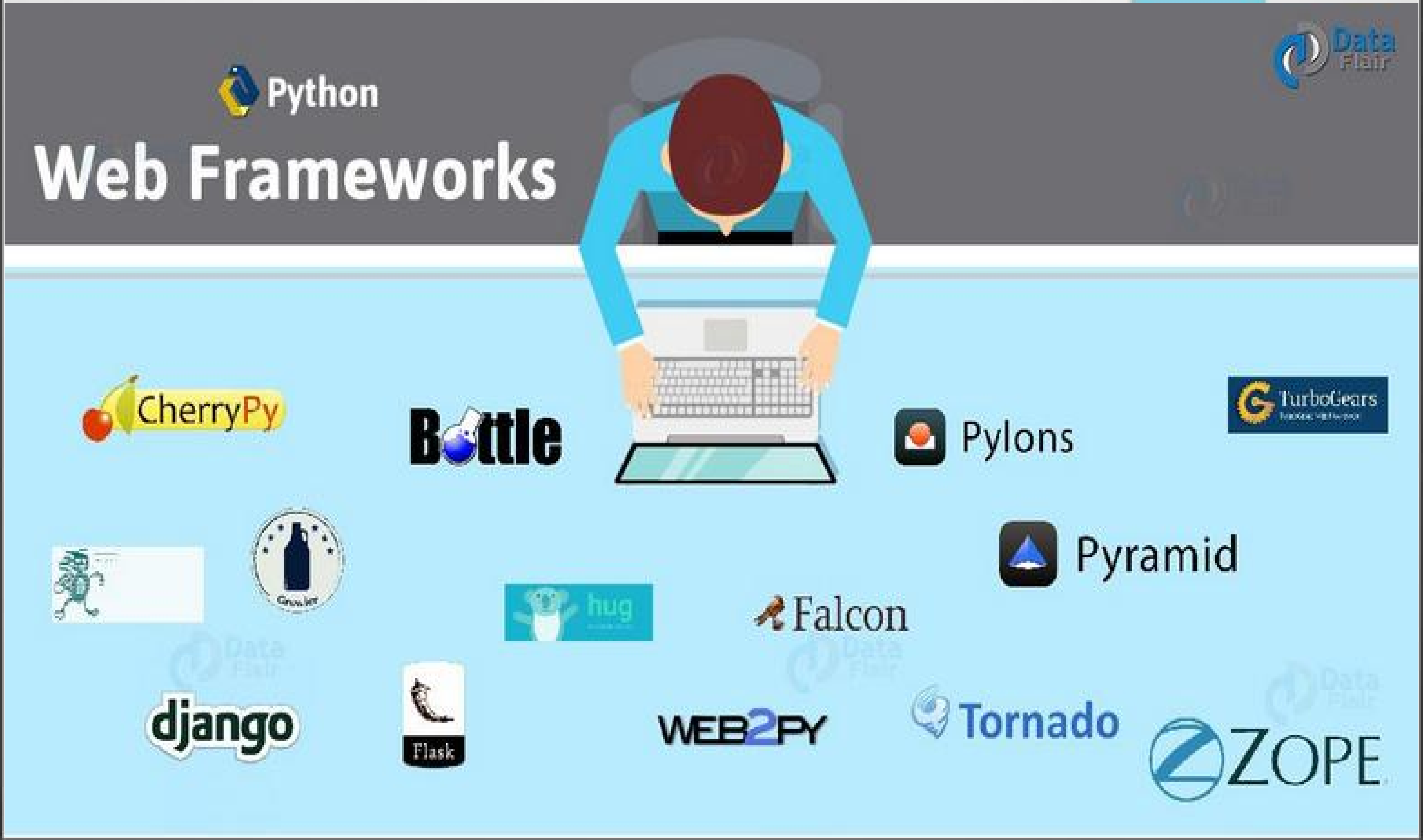
Advantages & Disadvantages of Python



Python Advantages and Disadvantages

Advantages	Disadvantages
Improved Productivity	Slow Speed
Interpreted Language	Not Memory Efficient
Dynamically Typed	Weak in Mobile Computing
Free and Open Source	Database Access
Vast Libraries Support	Runtime Errors

Python Frameworks



Web Frameworks for Python

- A Web framework is a collection of packages or modules which allow developers to write Web applications or services without having to handle such low-level details as protocols, sockets or process/thread management.

- **Django** is the most popular Python framework around, and it’s easy to understand. Thousands of websites are currently using Django, from daily newspapers to social media and sharing sites to major foundations and nonprofits. Django is known for being fast to build and friendly to beginning programmers.

- **Web2py** is easy to learn as Django, but also more flexible and extremely portable. The same code can run with a SQL database or MongoDB. The Web2py framework comes with a code editor, debugger, and deployment tool with which you can develop and debug code, as well as test and maintain applications.

Web Frameworks for Python

- **Pyramid** as the “Goldilocks” framework, feature-rich without enforcing one way of doing things, lightweight without leaving you on your own as your app grows. It’s a favorite framework among many experienced Python developers and transparency, and has been used by small teams as well as tech giants like Dropbox, Yelp, SurveyMonkey, and Mozilla.

- **Bottle** framework is ideal for small applications and is mainly used for building APIs. It is one of the most used python web frameworks as it doesn’t need any other dependency apart from the standard python library to make the application, programmers simply can work with hardware. This framework creates a single source file and it comes under the micro-framework category.

- Other frameworks like Flask, Tornado, etc...

Basic Program Structure

Basically, a Python program consists of individual statements, which in the simplest case take up exactly one line in the source code.

For example, the following statement prints text on the screen:

```
 print("Hello World")
```

Some statements can be divided into a statement header and a statement body, where the body can contain further statements:

Basic Program Structure

```
Statement header:

   Statement

…

   Statement
```

In a real Python program, this may look something like this:

```
if x > 10:
   print("x is greater than 10")
   print("Second line!")
```

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.

Contents:

Python Mode:

Python is a programming language that lets you work quickly and integrate systems more efficiently.

It is a widely-used general-purpose, high-level programming language.

It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

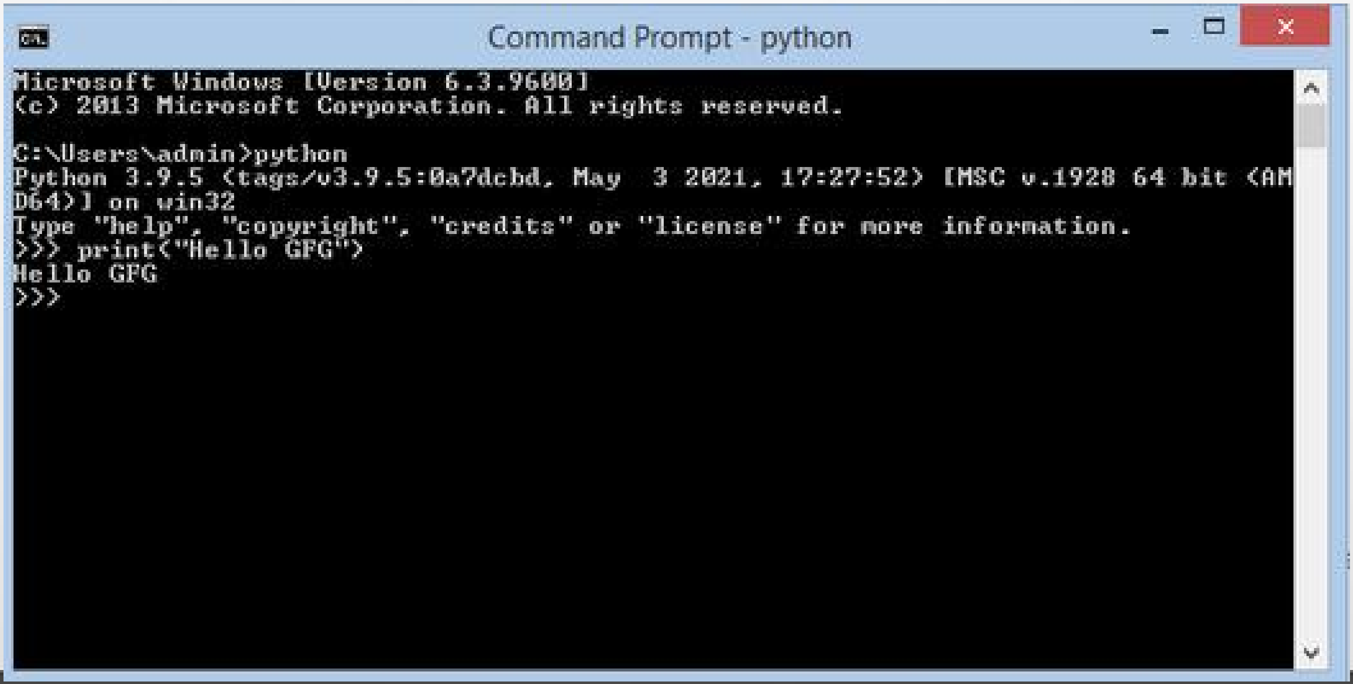
In the Python programming language, there are two ways in which we can run our code:

1. Interactive mode

2. Script mode

Interactive Mode

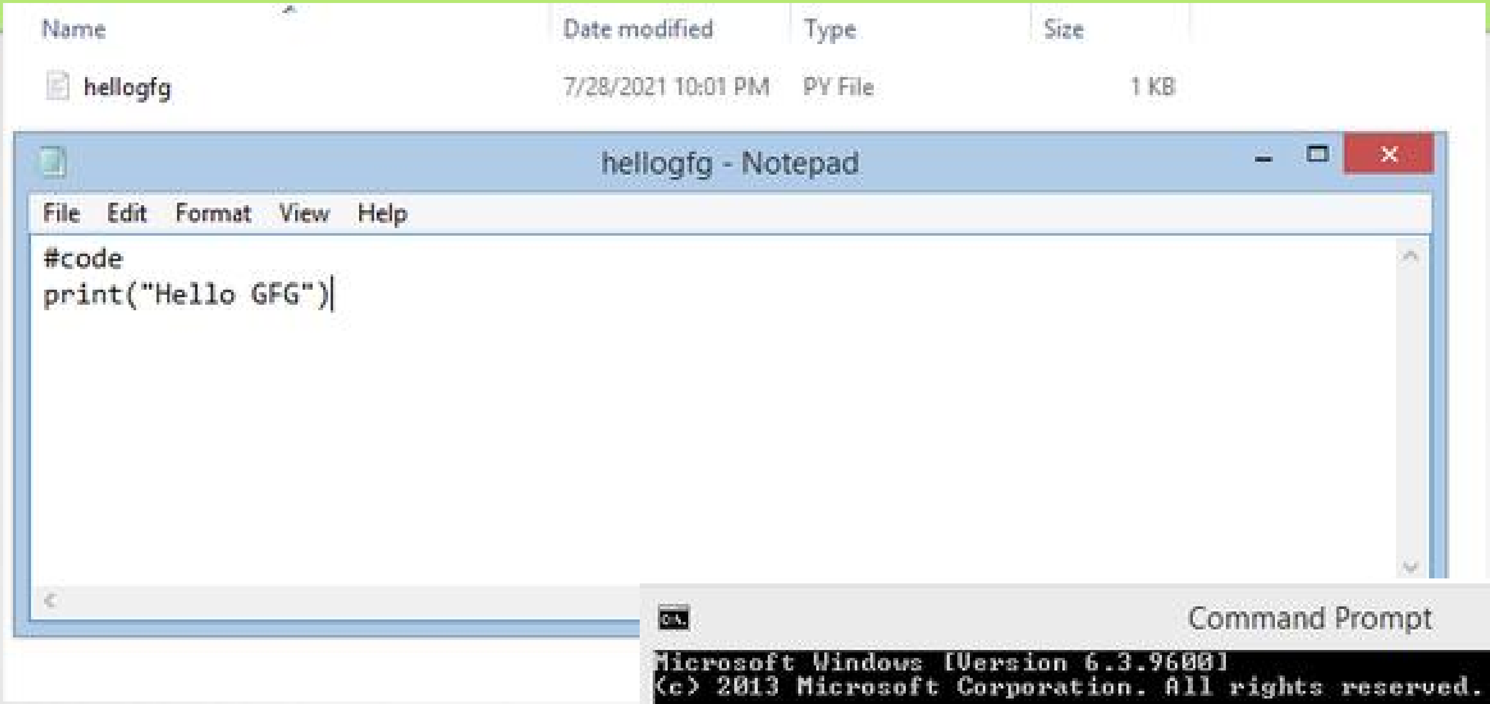
- To run python in command prompt type “python”.
- Then simply type the Python statement on >>> prompt.
- As we type and press enter we can see the output in the very next line.
- print("Hello GFG")



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\admin>python
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello GFG")
Hello GFG
>>>
```

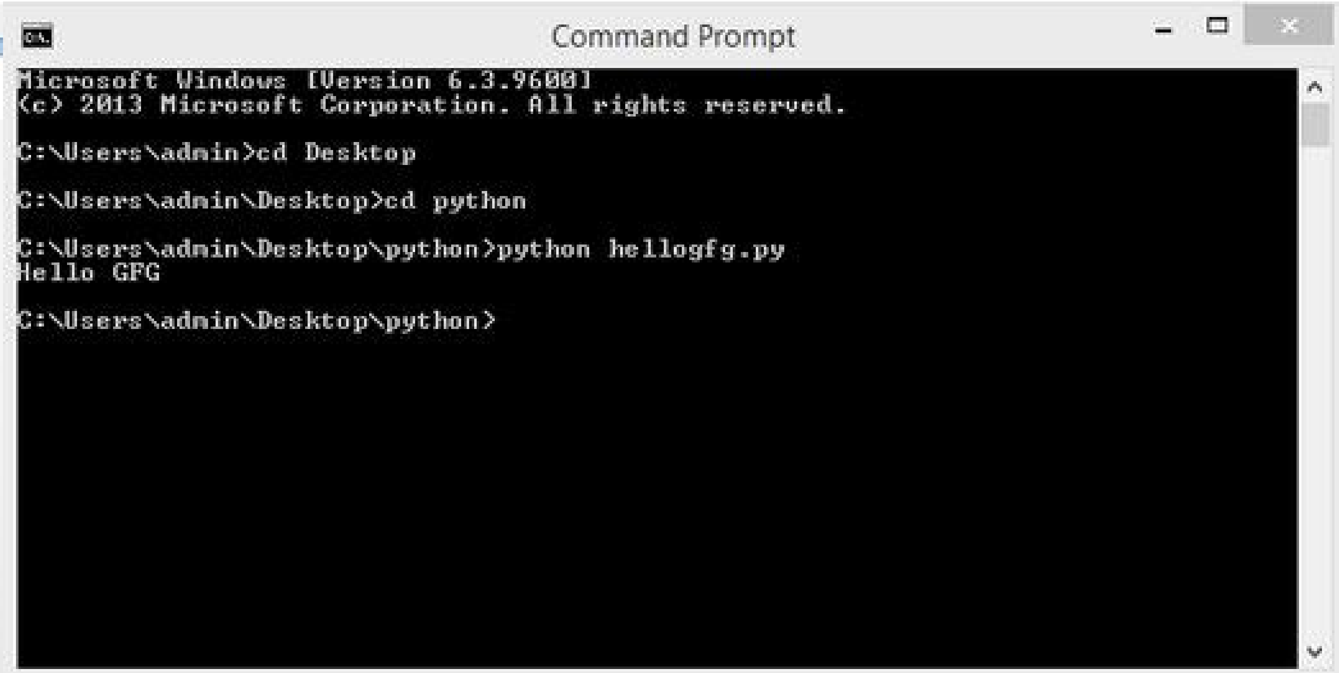
Script Mode



Name	Date modified	Type	Size
hellogfg	7/28/2021 10:01 PM	PY File	1 KB

hellogfg - Notepad

```
#code
print("Hello GFG")
```


Command Prompt

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\admin>cd Desktop

C:\Users\admin\Desktop>cd python

C:\Users\admin\Desktop\python>python hellogfg.py
Hello GFG

C:\Users\admin\Desktop\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.

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:

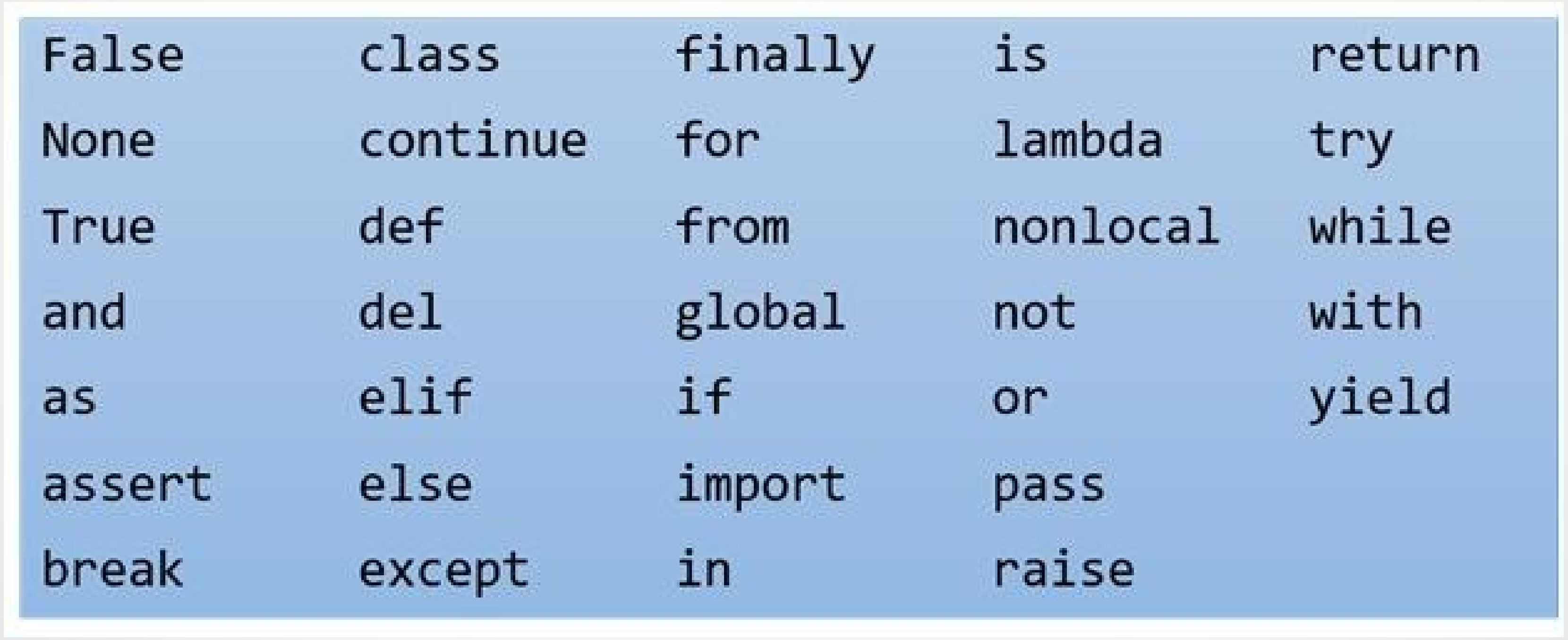
Keywords are the reserved words in Python.
We cannot use a keyword as a variable name, function name or any other identifier.
They are used to define the syntax and structure of the Python language.

In Python, keywords are case sensitive.

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 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

Basics of Python

Variable Naming Rules

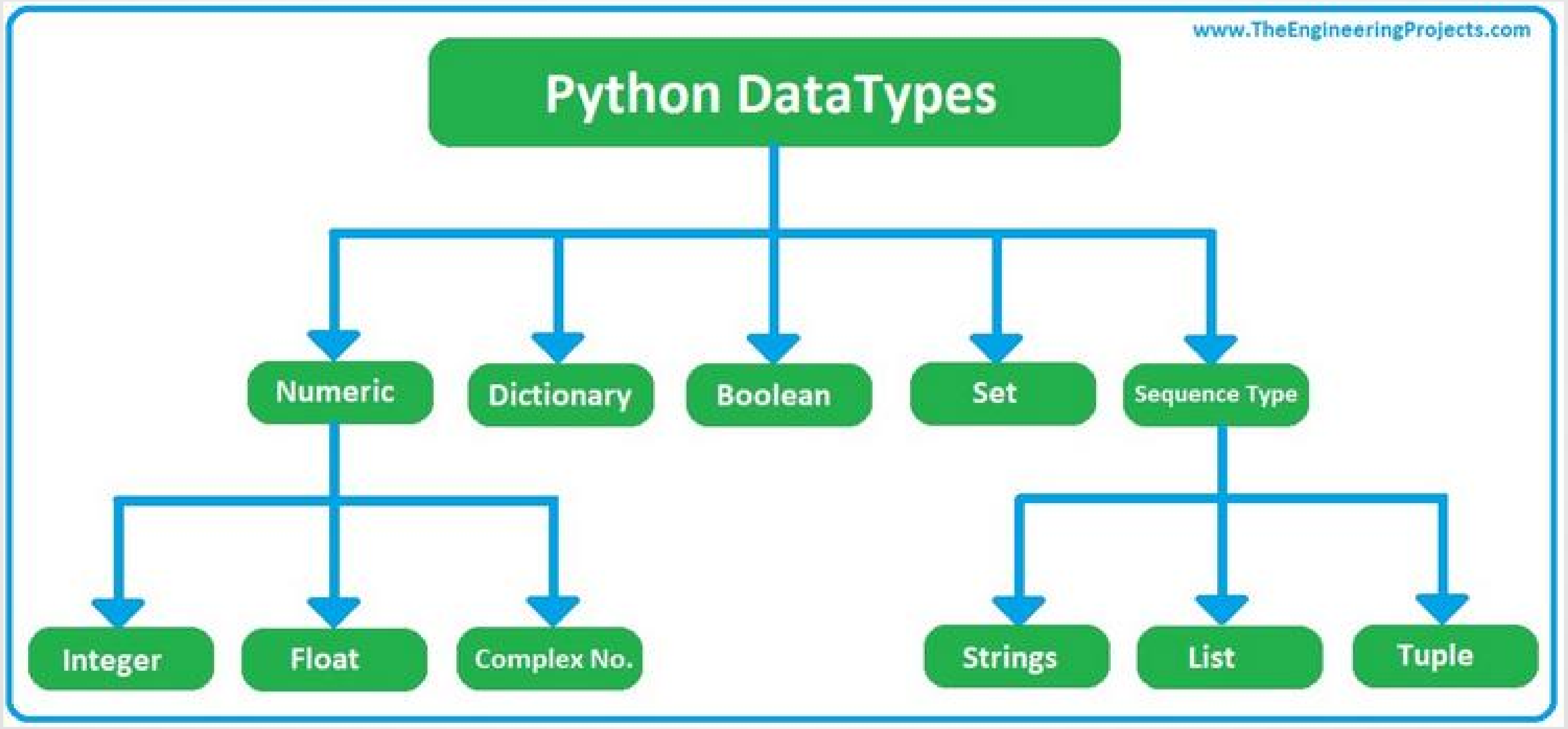
- Must begin with a letter (a - z, A - B) or underscore (_)
- Other characters can be letters, numbers or _
- Case Sensitive
- Reserved words cannot be used as a variable name

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"

Datatypes of Python



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 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

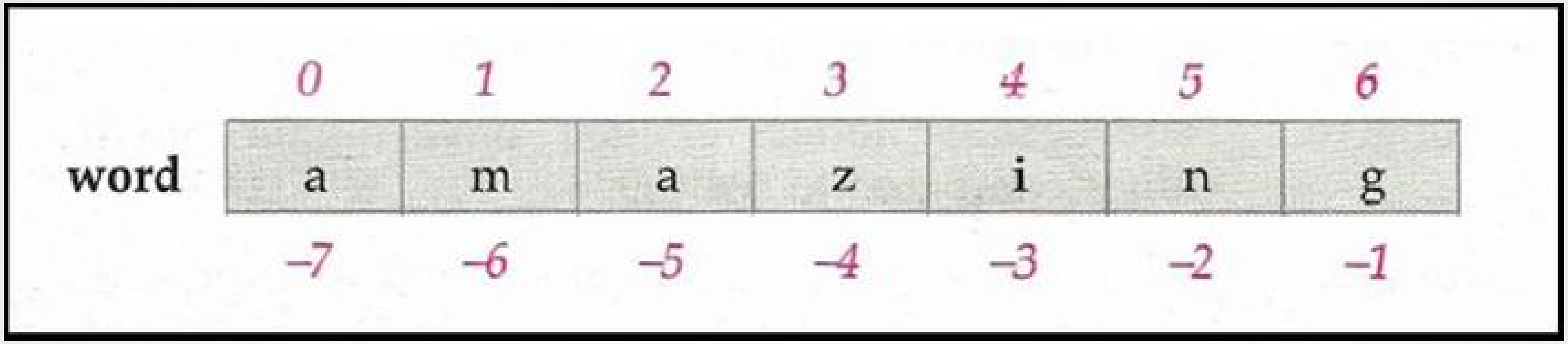
Boolean (bool)

- 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.

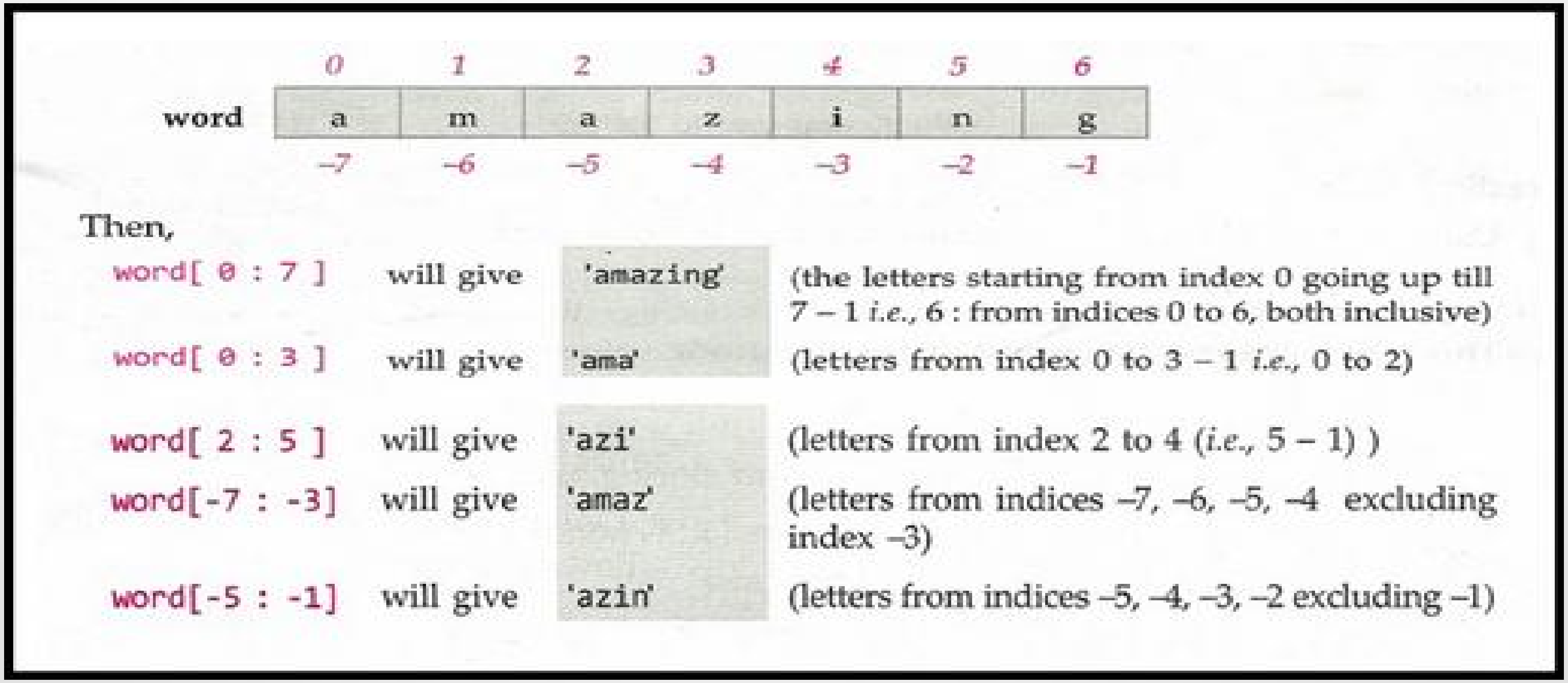
String Basics



Note: The indexes of a string begin from 0 to (length-1) in the forward direction and -1,-2,-3,…, -length in the backward direction.

String Slicing

To do string slicing, we just need to put the name of the string followed by [start:stop:step].
start: Starting index where the slicing of object starts.
stop: Ending index where the slicing of object stops.
step: It is an optional argument that determines the increment between each index for slicing.



	0	1	2	3	4	5	6
word	a	m	a	z	i	n	g
	–7	–6	–5	–4	–3	–2	–1

Then,

`word[ 0 : 7 ]`	will give	'amazing'	(the letters starting from index 0 going up till 7 – 1 *i.e.,* 6 : from indices 0 to 6, both inclusive)
`word[ 0 : 3 ]`	will give	'ama'	(letters from index 0 to 3 – 1 *i.e.,* 0 to 2)
`word[ 2 : 5 ]`	will give	'azi'	(letters from index 2 to 4 (*i.e.,* 5 – 1))
`word[-7 : -3]`	will give	'amaz'	(letters from indices –7, –6, –5, –4 excluding index –3)
`word[-5 : -1]`	will give	'azin'	(letters from indices –5, –4, –3, –2 excluding –1)

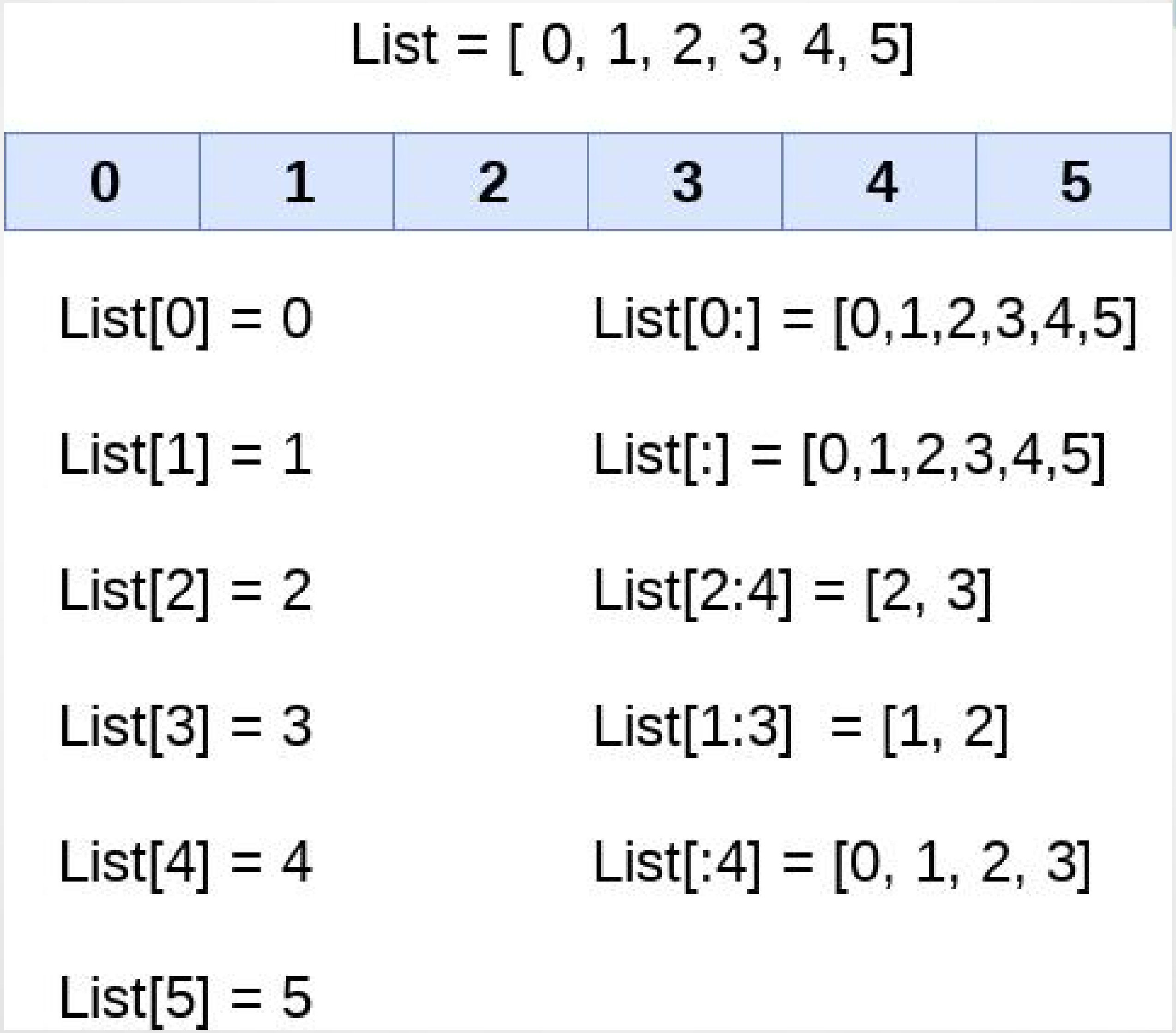
Basics of Python

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.
- Lists are used to store multiple items in a single variable.

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

Basics of Python

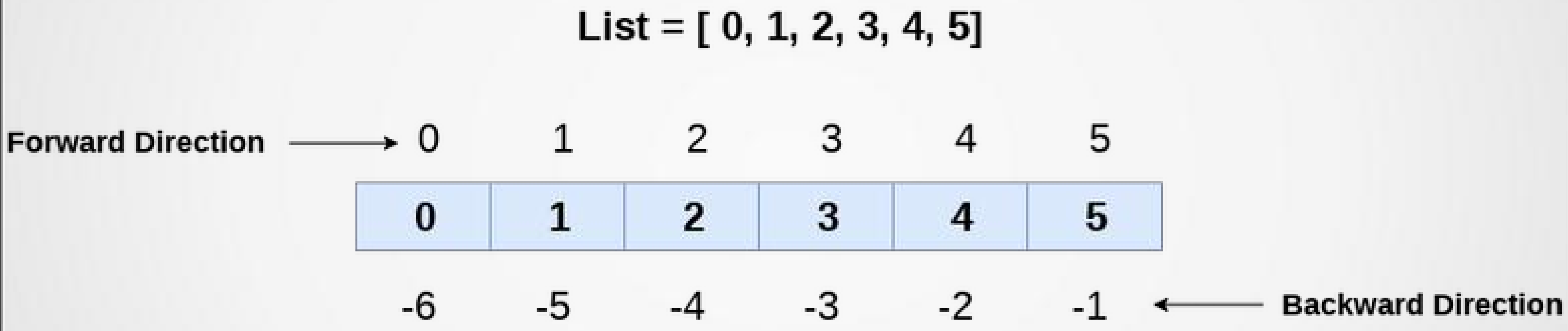


List = [0, 1, 2, 3, 4, 5]

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

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

Basics of Python



Basics of Python

Tuple

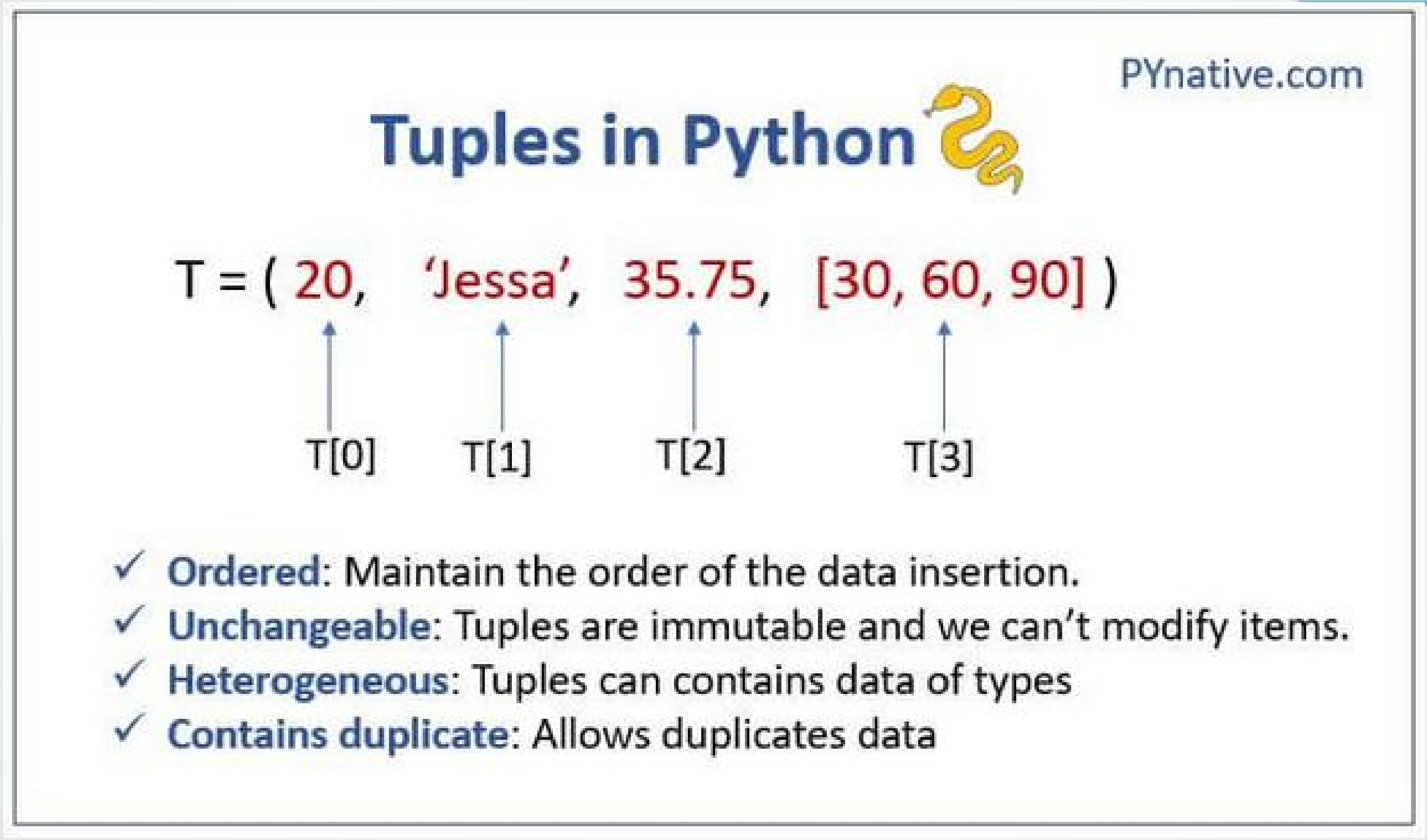
A tuple is another sequence data type that is similar to the list.

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


PYnative.com

Tuples in Python

T = (20, 'Jessa', 35.75, [30, 60, 90])

T[0] T[1] T[2] T[3]

- ✓ **Ordered**: Maintain the order of the data insertion.
- ✓ **Unchangeable**: Tuples are immutable and we can't modify items.
- ✓ **Heterogeneous**: Tuples can contains data of types
- ✓ **Contains duplicate**: Allows duplicates data

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 ([]).

Basics of Python

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

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

Basics of Python

```
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
```

Restrictions on Dictionary

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
(b) 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
```

Restrictions on Dictionary

(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(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

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

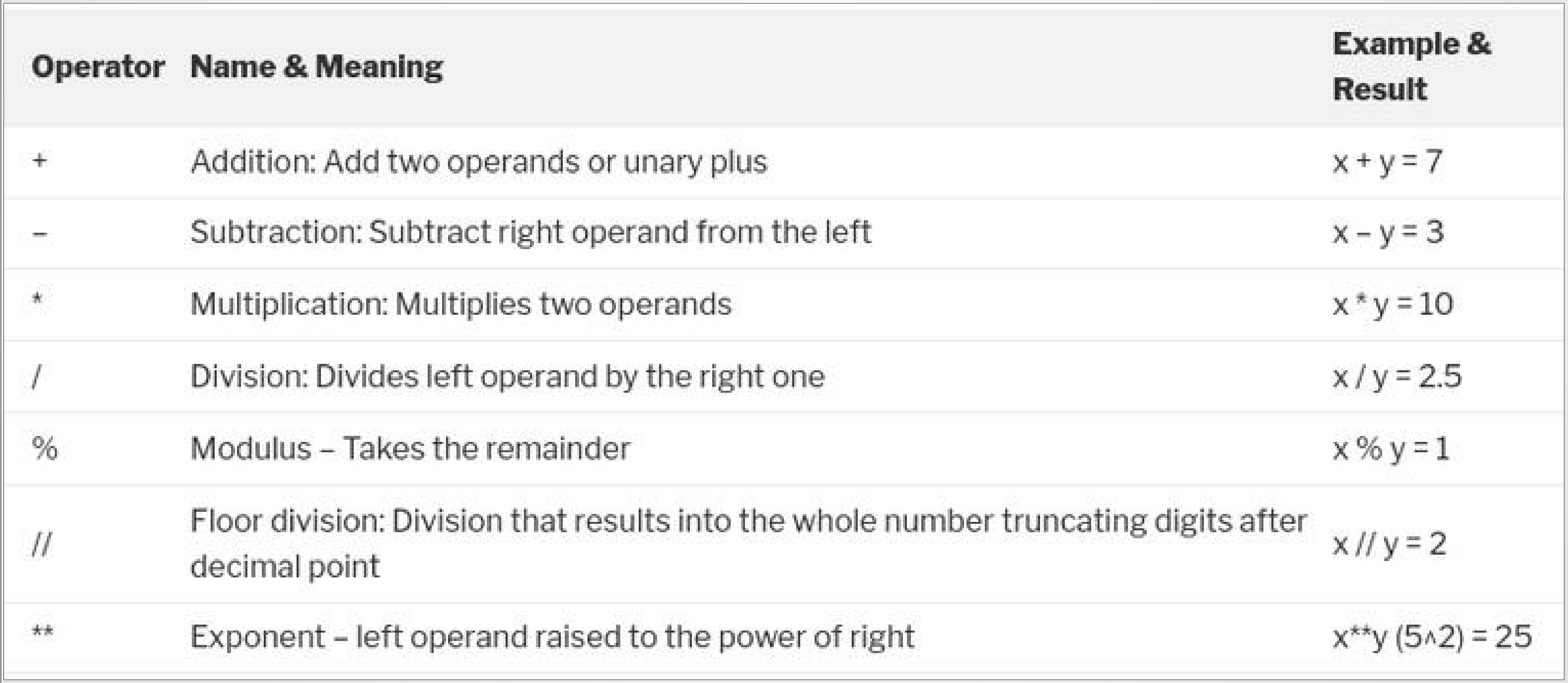
- Operators are special symbols which represent symbols and perform arithmetic and logical computations.
- The values the operator uses are called operands.

- **Python supports following types of Operators:**
  - Arithmetic Operators
  - Comparison/Relational Operators
  - Assignment Operators
  - Logical Operators
  - Membership Operators
  - Identity Operators
  - Bitwise Operators

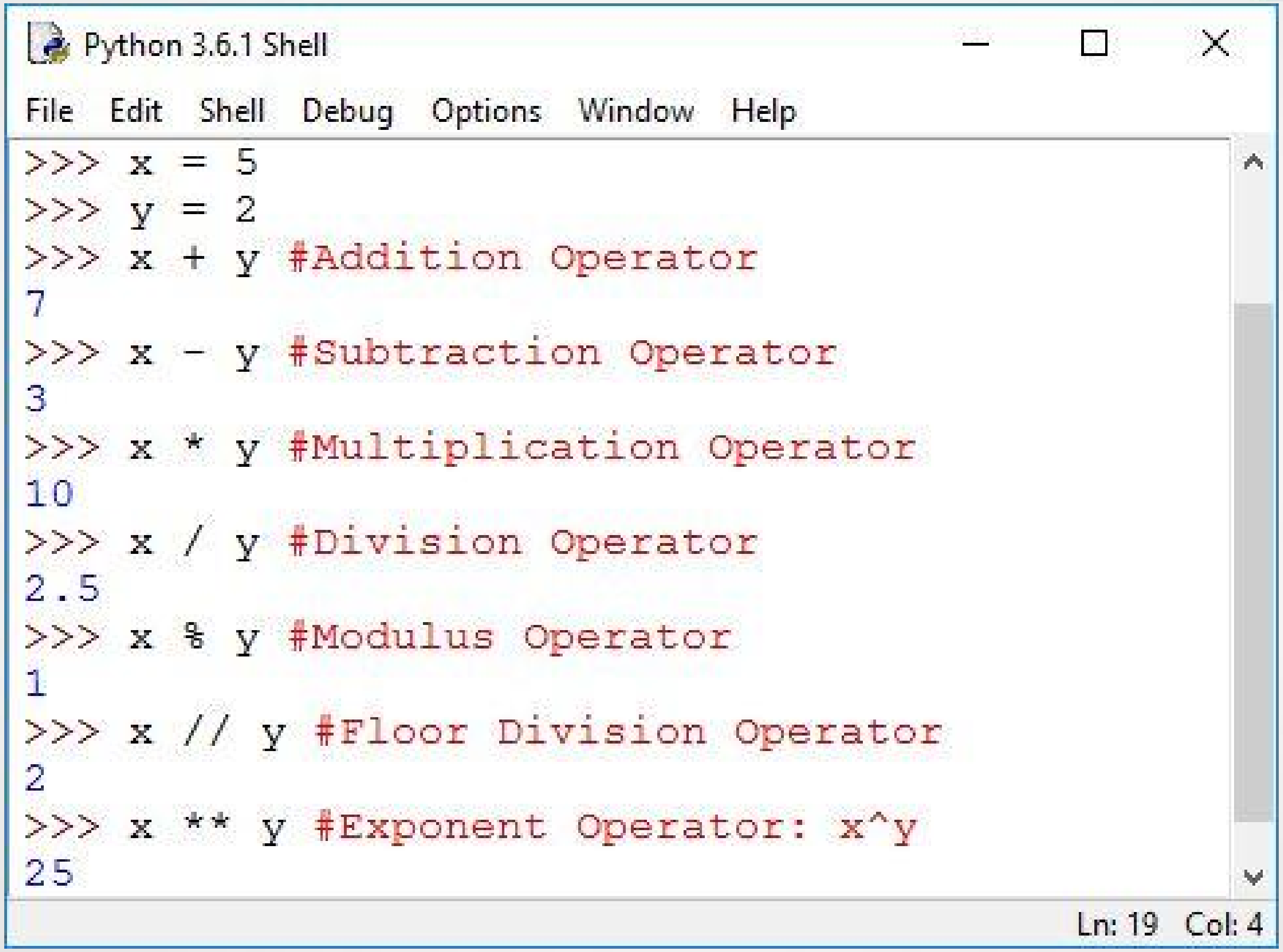
Basics of Python

Arithmetic Operators:

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



Operator	Name & Meaning	Example & Result
+	Addition: Add two operands or unary plus	x + y = 7
–	Subtraction: Subtract right operand from the left	x – y = 3
*	Multiplication: Multiplies two operands	x * y = 10
/	Division: Divides left operand by the right one	x / y = 2.5
%	Modulus – Takes the remainder	x % y = 1
//	Floor division: Division that results into the whole number truncating digits after decimal point	x // y = 2
**	Exponent – left operand raised to the power of right	x**y (5^2) = 25

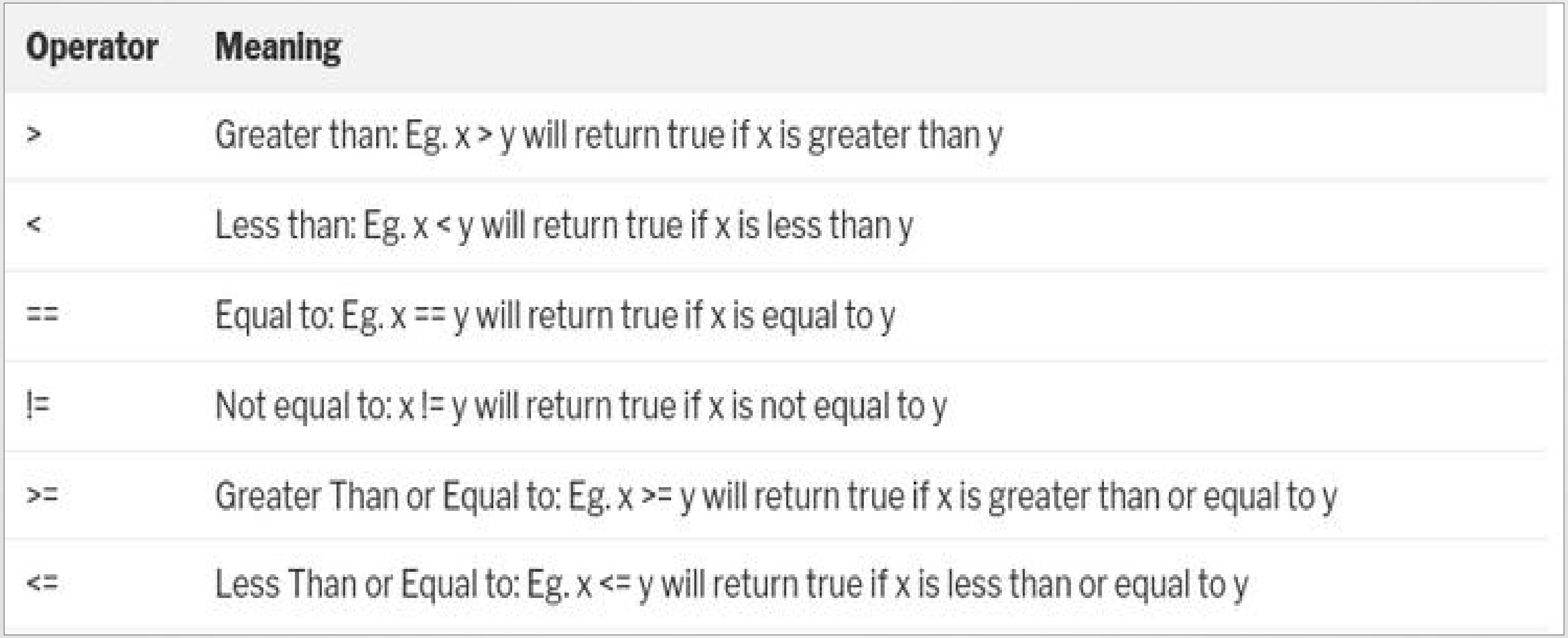


```
>>> x = 5
>>> y = 2
>>> x + y #Addition Operator
7
>>> x - y #Subtraction Operator
3
>>> x * y #Multiplication Operator
10
>>> x / y #Division Operator
2.5
>>> x % y #Modulus Operator
1
>>> x // y #Floor Division Operator
2
>>> x ** y #Exponent Operator: x^y
25
```

Basics of Python

Comparison Operators:

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

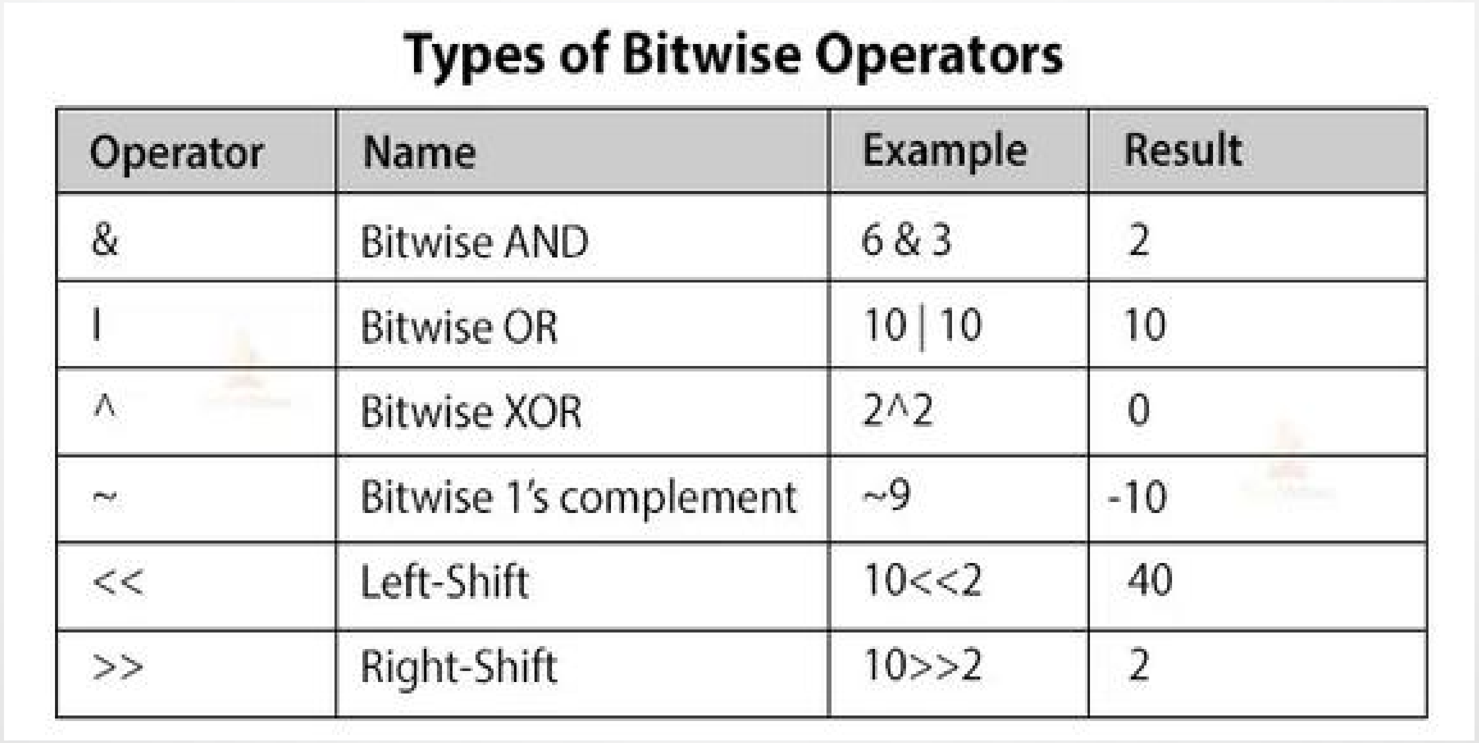


Operator	Meaning
>	Greater than: Eg. x > y will return true if x is greater than y
<	Less than: Eg. x < y will return true if x is less than y
==	Equal to: Eg. x == y will return true if x is equal to y
!=	Not equal to: x != y will return true if x is not equal to y
>=	Greater Than or Equal to: Eg. x >= y will return true if x is greater than or equal to y
<=	Less Than or Equal to: Eg. x <= y will return true if x is less than or equal to y

Basics of Python

Bitwise Operators:

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



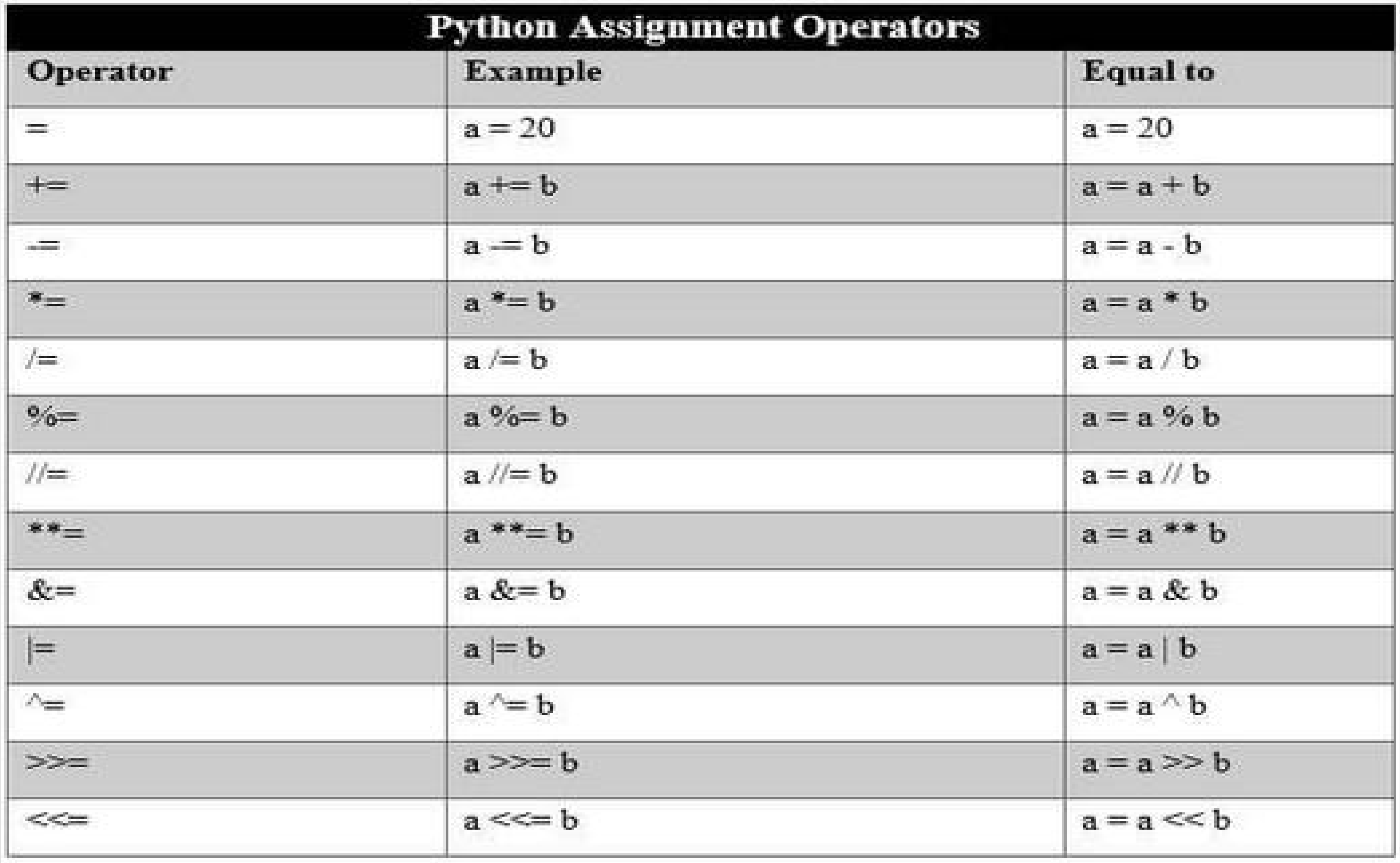
Types of Bitwise Operators

Operator	Name	Example	Result
&	Bitwise AND	6 & 3	2
|	Bitwise OR	10 | 10	10
^	Bitwise XOR	2^2	0
~	Bitwise 1's complement	~9	-10
<<	Left-Shift	10<<2	40
>>	Right-Shift	10>>2	2

Basics of Python

Assignment Operators:

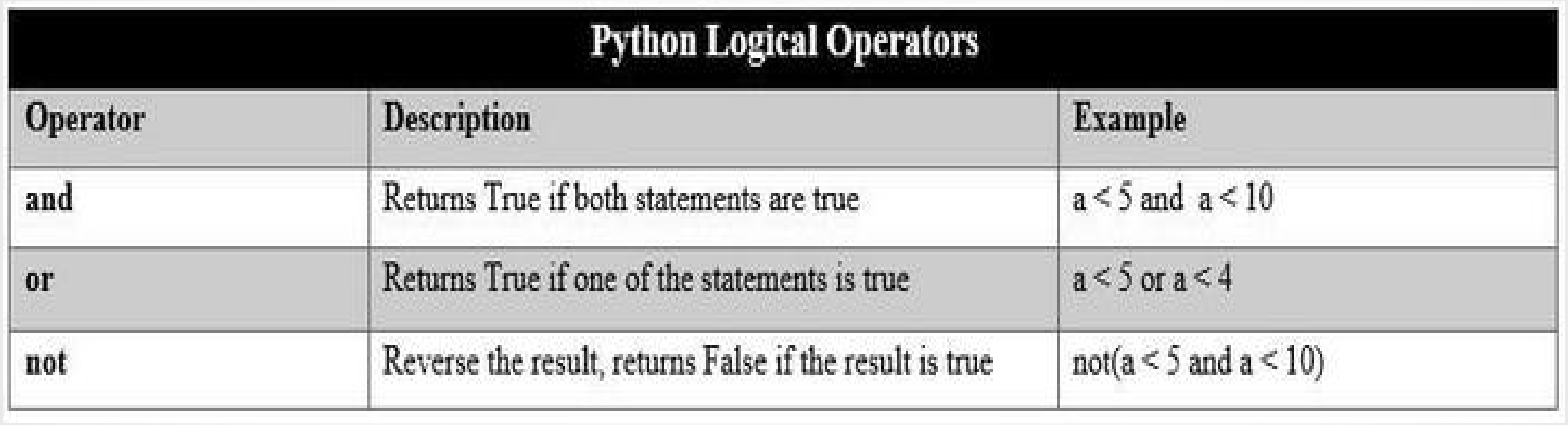
Assignment operators are used in Python to assign values to variables.



Python Assignment Operators		
Operator	**Example**	**Equal to**
=	a = 20	a = 20
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
//=	a //= b	a = a // b
**=	a **= b	a = a ** b
&=	a &= b	a = a & b
|=	a |= b	a = a | b
^=	a ^= b	a = a ^ b
>>=	a >>= b	a = a >> b
<<=	a <<= b	a = a << b

Basics of Python

Logical Operators:

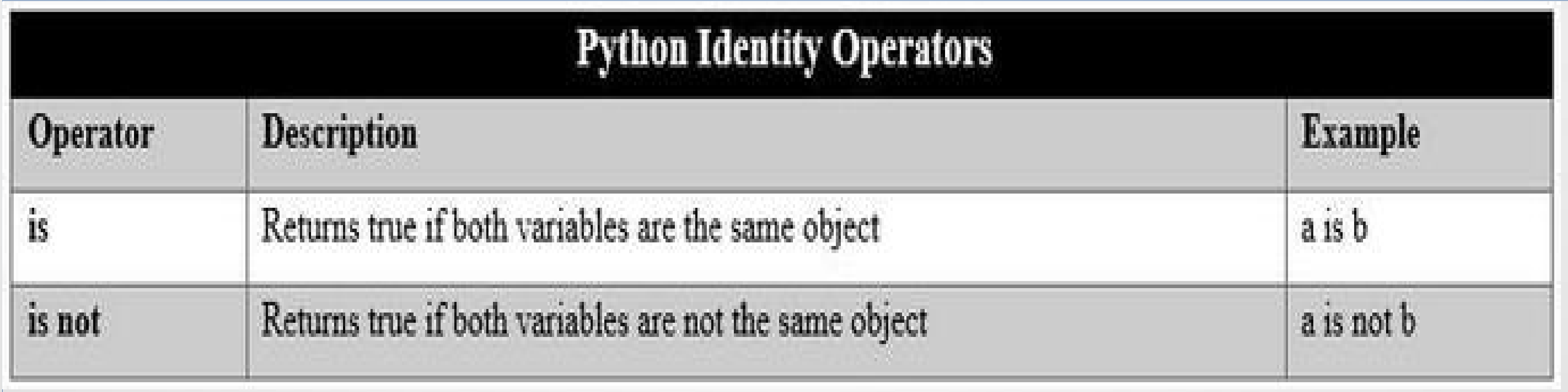


Python Logical Operators		
Operator	**Description**	**Example**
and	Returns True if both statements are true	a < 5 and a < 10
or	Returns True if one of the statements is true	a < 5 or a < 4
not	Reverse the result, returns False if the result is true	not(a < 5 and a < 10)

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.



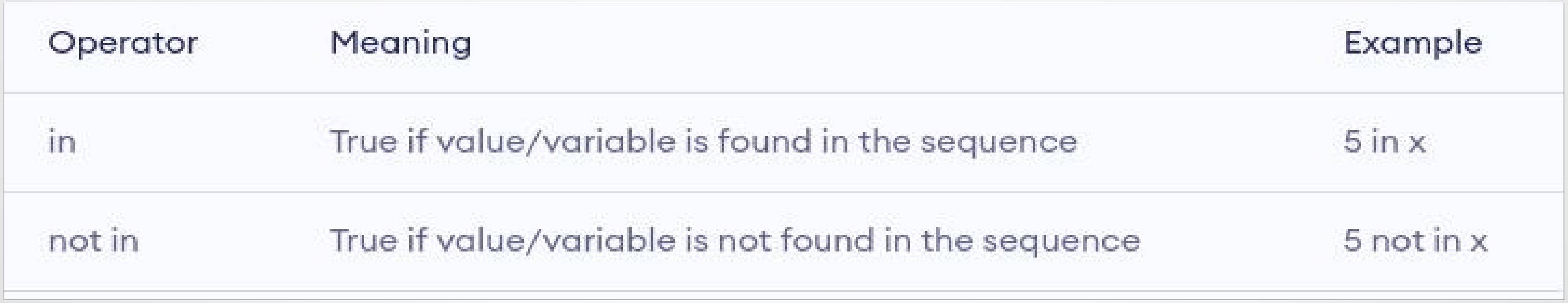
Python Identity Operators

Operator	Description	Example
is	Returns true if both variables are the same object	a is b
is not	Returns true if both variables are not the same object	a is not b

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	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

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 python language”

sub = "i";
print "str.count(sub, 4, 40) : ", //1
sub = "wow";
print "str.count(sub) : " //0
```

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...exam.";
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:

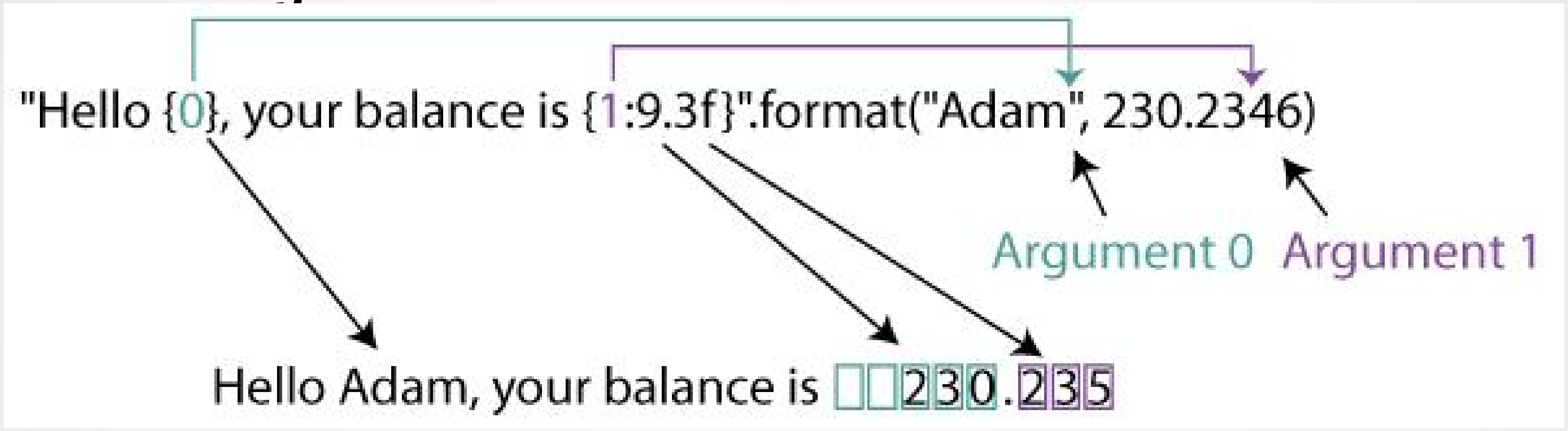
format() method takes any number of parameters. But, is divided into two types of parameters:

- **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}

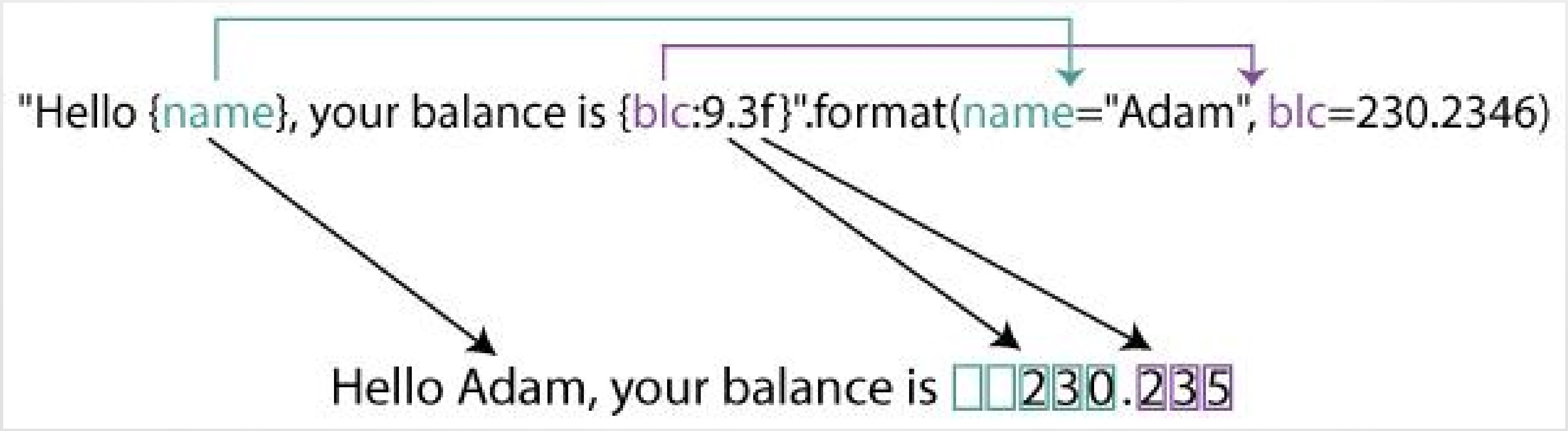
String

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"))

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

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

# string padding with center alignment
# and '*' padding character
print("{:*^5}".format("cat"))
```

Truncating strings with format()

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

```
c a t
```

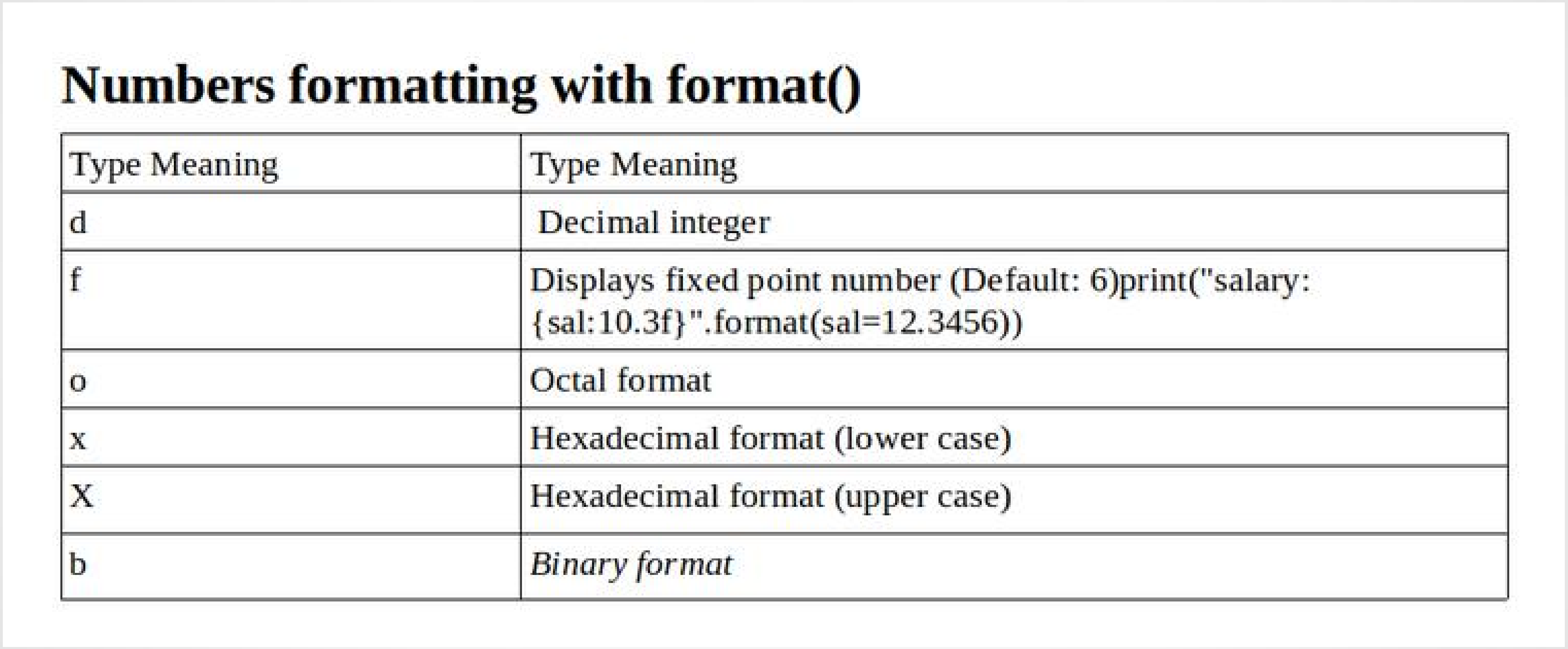
```
# truncating strings to 3 letters
# and padding
print("{:5.3}".format("caterpillar"))
```

```
c a t
```

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

```
c a t
```

Number formating



Numbers formatting with format()

Type Meaning	Type Meaning
d	Decimal integer
f	Displays fixed point number (Default: 6)print("salary: {sal:10.3f}".format(sal=12.3456))
o	Octal format
x	Hexadecimal format (lower case)
X	Hexadecimal format (upper case)
b	*Binary format*

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
```