**Python Notes**

**Introduction To Python:-**

**Python 3.x**

**Version Release Date**

3.7 2017-05-08

3.6 2016-12-23

3.5 2015-09-13

3.4 2014-03-17

3.3 2012-09-29

3.2 2011-02-20

3.1 2009-06-26

3.0 2008-12-03

**Python 2.x**

**Version Release Date**

2.7 2010-07-03

2.6 2008-10-02

2.5 2006-09-19

2.4 2004-11-30

2.3 2003-07-29

2.2 2001-12-21

2.1 2001-04-15

2.0 2000-10-16

**Introduction:-**

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library.

Two major versions of Python are currently in active use:

Python 3.x is the current version and is under active development.

Python 2.x is the legacy version and will receive only security updates until 2020. No new features will be implemented. Note that many projects still use Python 2, although migrating to Python 3 is getting easier.

**Verify if Python is installed:-**

To confirm that Python was installed correctly, you can verify that by running the following command in your terminal (If you are using Windows OS, you need to add path of python to the environment variable before using it in command prompt):

$ python –version

**How to create Hello, World program in IDLE:**

Open IDLE on your system of choice.

In older versions of Windows, it can be found at All Programs under the Windows menu.

In Windows 8+, search for IDLE or find it in the apps that are present in your system.

On Unix-based (including Mac) systems you can open it from the shell by typing $ idle

python\_file.py.

It will open a shell with options along the top.

In the shell, there is a prompt of three right angle brackets:

>>>

Now write the following code in the prompt:

>>> print("Hello, World")

**Hello World Python file**

Create a new file hello.py that contains the following line:

print('Hello, World')

**There are multiple ways to close the Python shell**:

>>> exit()

or

>>> quit()

Alternatively, CTRL + D will close the shell and put you back on your terminal's command line.

If you want to cancel a command you're in the middle of typing and get back to a clean command prompt, while staying inside the Interpreter shell, use CTRL + C .

**Creating variables and assigning values:-**

To create a variable in Python, all you need to do is specify the variable name, and then assign a value to it.

<variable name> = <value>

Python uses = to assign values to variables. There's no need to declare a variable in advance (or to assign a data type to it), assigning a value to a variable itself declares and initializes the variable with that value. There's no way to declare a variable without assigning it an initial value.

# Integer

a = 2

print(a)

# Output: 2

# Integer

b = 9223372036854775807

print(b)

# Output: 9223372036854775807

# Floating point

pi = 3.14

print(pi)

# Output: 3.14

# String

c = 'A'

print(c)

# Output: A

# String

name = 'John Doe'

print(name)

# Output: John Doe

# Boolean

q = True

print(q)

# Output: True

# Empty value or null data type

x = None

print(x)

# Output: None

Variable assignment works from left to right. So the following will give you an syntax error.

0 = x

=> Output: SyntaxError: can't assign to literal

You can not use python's keywords as a valid variable name. You can see the list of keyword by:

import keyword

print(keyword.kwlist)

**Rules for variable naming:**

1. Variables names must start with a letter or an underscore.
2. The remainder of your variable name may consist of letters, numbers and underscores.
3. Names are case sensitive. Even though there's no need to specify a data type when declaring a variable in Python, while allocating the necessary area in memory for the variable, the Python interpreter automatically picks the most suitable built-in type for it:

a = 2

print(type(a))

# Output: <type 'int'>

**Note:-** You can assign multiple values to multiple variables in one line. Note that there must be the same number of arguments on the right and left sides of the = operator:

a, b, c = 1, 2, 3

print(a, b, c)

# Output: 1 2 3

**Note:-** You can also assign a single value to several variables simultaneously.

a = b = c = 1

print(a, b, c)

# Output: 1 1 1

**Note:-** When using such cascading assignment, it is important to note that all three variables a, b and c refer to the same object in memory, an int object with the value of 1. In other words, a, b and c are three different names given to the same int object. Assigning a different object to one of them afterwards doesn't change the others, just as expected:

a = b = c =1 # all three names a, b and c refer to same int object

with value 1

print(a, b, c)

# Output: 1 1 1

b = 2 # b now refers to another int object, one with value of 2

print(a, b, c)

# Output: 1 2 1 # so output is as expected.

**Note:-** The above is also true for mutable types (like list, dict, etc.) just as it is true for immutable types (like int, string,

tuple, etc.):

x = y = [7, 8, 9] # x and y refer to the same list object just created, [7, 8, 9]

x = [13, 8, 9] # x now refers to a different list object just created, [13, 8, 9]

print(y) # y still refers to the list it was first assigned

# Output: [7, 8, 9]

**Note:-** Nested lists are also valid in python. This means that a list can contain another list as an element.

x = [1, 2, [3, 4, 5], 6, 7] # this is nested list

print x[2]

# Output: [3, 4, 5]

print x[2][1]

# Output: 4

Lastly, variables in Python do not have to stay the same type as which they were first defined -- you can simply use = to assign a new value to a variable, even if that value is of a different type.

a = 2

print(a)

# Output: 2

a = "New value"

print(a)

# Output: New value

**Block indentation:-**

Python uses indentation to define control and loop constructs. This contributes to Python's readability, however, it requires the programmer to pay close attention to the use of whitespace.

Python uses the colon symbol (:) and indentation for showing where blocks of code begin and end.

Python, such as functions, loops, if clauses and other constructs, have no ending identifiers. All blocks start with a colon and then contain the indented lines below it.

For example:

def my\_function(): # This is a function definition. Note the colon (:)

a = 2 # This line belongs to the function because it's indented

return a # This line also belongs to the same function

print(my\_function()) # This line is OUTSIDE the function block

or

if a > b: # If block starts here

print(a) # This is part of the if block

else: # else must be at the same level as if

print(b) # This line is part of the else block

Blocks that contain exactly one single-line statement may be put on the same line, though this form is generally not considered good style:

if a > b: print(a)

else: print(b)

**DataTypes:-**

**Built-in types:-**

1. **Booleans:-**

A boolean value of either True or False. Logical operations like and,

or, not can be performed on booleans.

x or y # if x is False then y otherwise x

x and y # if x is False then x otherwise y

not x # if x is True then False, otherwise True

1. **Numbers:-**

* int: Integer number

Ex:- a = 2

b = 100

c = 123456789

d = 38563846326424324

* float: Floating point number

Ex:- a = 2.0

b = 100.e0

c = 123456789.e1

* complex: Complex numbers

Ex:- a = 2 + 1j

b = 100 + 10j

1. **Strings:-**

Collection of one or more characters.

Ex:- a=’Hello’

1. **Sequences and collections**

Python differentiates between ordered sequences and unordered collections (such as set and dict).

* tuple: An ordered collection of n values of any type (n >= 0).

a = (1, 2, 3)

b = ('a', 1, 'python', (1, 2))

b[2] = 'something else' # returns a TypeError

Supports indexing; immutable; hashable if all its members are hashable

* list: An ordered collection of n values (n >= 0)

a = [1, 2, 3]

b = ['a', 1, 'python', (1, 2), [1, 2]]

b[2] = 'something else' # allowed

Not hashable; mutable.

* set: An unordered collection of unique values. Items must be hashable.

a = {1, 2, 'a'}

* dict: An unordered collection of unique key-value pairs; keys must be hashable.

a = {1: 'one', 2: 'two'}

b = {'a': [1, 2, 3], 'b': 'a string'}