What is Python ?

Python is an object-oriented, interpreted language that’s easy to use and runs on many operating systems including Windows, Mac OS, Linux etc.

Python supports basic data types such as numbers and strings, as well as more complex types like lists and dictionaries, that can greatly simplify data processing.

Python also supports several programming paradigm, and can be used for procedural programming, functional programming and object oriented programming.

Data in python is strongly typed – i.e., attempting to add a number and a string will give an error. At the same time, it is also dynamically typed i.e., you are freed from worrying about variable declarations.

Interpreter translates just one statement of program at a time into machine code whereas compiler scans the entire program and translates the whole of it into machine code at once. An interpreter takes very less time to analyze the code.

print() function will be used to print data in Python.

Ex: print(‘Hello World’) or print(“Hello World”)

‘#’ is used to add comments in python code

Ex: # print welcome message

print(“Hello, World!”)

There is no ‘;’ or special character to identify end of statement in python. If we just press enter, it will be consider as end of statement.

Python code file will be saved with extension ‘.py’, for example the file name will be ‘hello.py’.

Python files can be run from command line prompt using following command.

>> python E:\PYTHON\hello.py

Exercise:

print('Hello, World!')

print(1 + 2)

print(6 \* 7)

print()

print('The end', 'or is it?', 'keep watching to learn more about Python', 3)

print('My hovercraft is full of eels')

print('6 times 7 :', 6 \* 7)

Strings in python:

Strings are similar to other languages in Python, but while printing strings we can enclose them in either single quotes or double quotes.

If your string contains single quotes then enclose that string in double quotes, similarly if your string contains double quotes enclose the entire string in single quotes.

Strings can be stored in variables and to concatenate strings, use ‘+’ symbol between them.

input(): To read from terminal which is entered by the user

The Escape Character:

\n – to split string into next line

\t – tab space

To split string into multiple lines, use triple codes (‘’’)

Ex: splitString = """This string has been

split over

seveal

lines"""

Again, if you want the above string to be print in single line, add backslash (\) at the end of each line.

Ex: splitString = """This string has been \

split over \

seveal \

lines"""

The strings used for file names will be handled in two ways as follow:

1. print("C:\\Users\\timbuchalka\\notes.txt")
2. print(r"C:\Users\timbuchalka\notes.txt")

**Variable and Types:**

A variable is basically just a way to give a (meaningful) name to an area of memory, into which we can place certain values.

Ex: greeting = “Hello”

In Python, the type of a variable automatically decided by the value assigned to that variable. In the above case, variable ‘greeting’ will be considered as string type after we provide value “Hello” to it.

Rules for variable names:

* Python variable names must began with a letter (either upper or lower case) or an underscore ‘\_’ character
* They can contain letters, numbers or underscore characters, but cannot begun with a number
* Python variables are case sensitive i.e., greeting and Greeting both are different variables
* Variables are created when they first attached with an ‘=’ sign

Use type() to know the data type of a variable

Ex:

greeting = “Hello”

age = 24

print(type(greeting)) - <class ‘str’>

print(type(age)) - <class ‘int’>

In other languages, when a variable is declared with its data type and we cannot change that data type for that particular variable, whereas in Python the variable data type can be decided dynamically based on the assigned value. So, we can say that Python is dynamically and strongly typed language.

Ex:

age = 24

print(age) - 24

print(type(age)) - <class ‘int’>

age = “24 years”

print(age) - 24 years

print(type(age)) - <class ‘str’>

Why Python is strongly typed language ?

In Python, when we try to print concatenation of string and integer variables, it will show error, which is differ from other languages like Java.

Ex:

name = “Suresh”

age = 24

print(“Age of “ + name + “ is: “ + age + “ years of old”)

TypeError: can only concatenate str (not ‘int’) to str

Python has several built-in data types, that can be classified as:

* numeric
* iterator
* sequence (which are also iterators) Ex: str type - string
* mapping
* file
* class
* exception

Python 3 has 3 numeric data types:

* int
* float
* complex

**Note:** Python 2 had another type, ‘**long’,** because its ‘**int’** type couldn’t store very large values. In Python 3, the ‘**int’** type replaces long.

**int -** The Python integer data type is called **‘int’**

integers are just whole numbers – numbers having no fractional parts.

Python 3 – **int** effectively has no maximum size. There is no limit to the size of the values that you can store in int type variables.

**float –** float is another name for real numbers – numbers having a fractional part after the decimal point

Ex: 1.0, 123.456, 3.14159265

The maximum float value on a 64 bit computer is 1.7976931348623157e+308 which means move the decimal point 308 places right.

Python floats have 52 digits of precision, which should be adequate for most purposes. If you need more precise decimal numbers, Python 3 now includes ‘decimal’ data type.

As python strongly typed language, we cannot use float in place of int.

**Numeric Operators:**

‘+’ - Addition

‘– ‘ - Subtraction

‘\*’ - Multiplication

‘/’ - Division

‘//’ - Integer Division

‘%’ - Modulo Operator

The above are the numeric operators available in python.

‘/’ - result as float

‘//’ - result should be integer only

‘%’ - remainder after integer division

Ex:

a = 12

b = 3

print(a + b) - 15

print(a – b) - 9

print(a \* b) - 36

print(a / b) - 4.0

print(a // b) - 4

print(a % b) - 0

Ex: Integer Division

Problem: You have a shop selling buns for $2.40 each. A customer comes in with $15, and would like to buy as many buns as possible. Write the code to calculate how many buns the customer can afford.

Note: Your customer won’t be happy if you try to sell them part of a bun.

Sol:

bun\_price = 2.40

money = 15

number\_of\_buns = 15 // 2.40

print(number\_of\_buns) - 6

**Operator Precedence:**

Operator Precedence Acronyms:

PEDMAS – Parenthesis, Exponents, Multiplication/Division, Addition/Subtraction

BEDMAS – Brackets, Exponents, Multiplication/Division, Addition/Subtraction

BODMAS – Brackets, Order, Division/Multiplication, Addition/Subtraction

BIDMAS – Brackets, Index, Division/Multiplication, Addition/Subtraction

What’s wrong with these acronyms?

Well, they all have the same problem of being ambiguous. Students sometimes interpret them as meaning that multiplication has higher precedence than division, and addition has higher precedence than subtraction.

But, multiplication and division have equal precedence. Addition and subtraction also have equal precedence.

So, Multiplication & Division have higher precedence than Addition and Subtraction.

Note: In an expression that mixes operations with equal precedence, they are evaluated from left to right.

**String index:**

**Ex:**

parrot = "Norwegian Blue"

print(parrot) -- prints entire string value

print(parrot[3]) -- prints character at index 3 from string

# To print 'we win' characters from above string

print("# To print 'we win' characters from above string")

print(parrot[3])

print(parrot[4])

print(parrot[9])

print(parrot[3])

print(parrot[6])

print(parrot[8])

# To print 'we win' with negative indexing

print()

print("# To print 'we win' with negative indexing")

print(parrot[-11])

print(parrot[-10])

print(parrot[-5])

print(parrot[-11])

print(parrot[-8])

print(parrot[-6])

# To print 'we win' with negative indexing and string length

print()

print("# To print 'we win' with negative indexing and string length")

print(parrot[3 - 14])

print(parrot[4 - 14])

print(parrot[9 - 14])

print(parrot[3 - 14])

print(parrot[6 - 14])

print(parrot[8 - 14])

**SLICING:**

Python sequence types will create slicing.

With 3 numbers we can produce the slicing – start, stop and step

string[start : stop : step]

The default step values is ‘1’ until unless provides its value

**Ex:**

parrot = "Norwegian Blue"

# To print 5 characters starting from first character

print(parrot[0:6]) -- prints first 5 characters from starting of string, not includes 6th character

i.e., from 0 up to 6 but not including

parrot[3:5] : we -- prints 4 characters from starting index 3

parrot[0:9] : Norwegian

parrot[:9] : Norwegian

parrot[:9] : Norwegian

parrot[10:14] : Blue

parrot[:6] : Norweg

parrot[6:] : ian Blue

parrot[:6] + parrot[6:] : Norwegian Blue

parrot[:] : Norwegian Blue

# Negative index in slicing

parrot[0:6] : Norweg

parrot[-14:-8] : Norweg

parrot[-4:-2] : Bl

parrot[-4:12] : Bl

To use negative indexing, just subtract the positive index from length of the string as shown above;

**SLICING WITH STEP:**

# slicing with step

parrot[0:6:2] : Nre

parrot[0:6:3] : Nw

**SLICING BACKWARDS:**

letters = "abcdefghijklmnopqrstuvwxyz"

To print above string value in reverse order: letters[25::-1]

zyxwvutsrqponmlkjihgfedcba

To print above string value in reverse order: letters[::-1]

zyxwvutsrqponmlkjihgfedcba

To print qpa: letters[16:13:-1]

qpo

To print edcba: letters[4::-1]

edcba

To print last 8 characters in reverse order: letters[25:17:-1]

zyxwvuts

To print last 8 characters in reverse order: letters[:-9:-1]

zyxwvuts

**SLICING IDIOMS**

letters[-4:] : wxyz

letters[-1:] : z

letters[:1] : a

letters[0] : a

**PYTHON SEQUENCE TYPES**

Python - 3 has 5 sequence types built in:

* The string type
* List
* tuple
* range
* bytes and bytearray

What is a sequence ?

A sequence is defined as an ordered set of items.

For example, the string **“Hello World”** contains 11 items, and each item is a character.

A list is also a sequence type.

For example, here’s a Python list of things you might need, when you are buying a new computer:

[“computer”, “monitor”, “keyboard”, ”mouse”, “mouse mat”]

That list contains 5 item, each of which is a string

Ex:

computer\_parts = ["computer", "monitor", "keyboard", "mouse"]

Computer part at index 1 : monitor

Computer part at index 1 and in that at index 0 : m

Note: Not all sequence types can be concatenated or multiplied. **range** is an example of that can’t be concatenated.

**String Operators**

"Hello " \* 5: Hello Hello Hello Hello Hello

"Hello " \* (5 + 4): Hello Hello Hello Hello Hello Hello Hello Hello Hello

"Hello " \* 5 + "4": Hello Hello Hello Hello Hello 4

today = "Sunday"

"day" in today :True

"Sun" in today :True

"fri" in today :False

"parrot" in "fjord":False

**String Replacement Fields**

Str() is used to concatenate other data types with string while printing the result as a string.

Ex:

age = 24

print("My age is " + str(age) + " years")

print("My age is {0} years".format(age))