Python Programming Language

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.

An increasingly popular prototyping language, Python is able to support rapid application and games development. It has a well-defined language specification and innovative syntax features that enable high levels of expressiveness. Python is the preferred language for computer science research and supports multiple modes of development such as:

- functional programming;
- object-oriented or imperative programming.

With its flexibility, speed, and machine learning functionality, it is expected to dominate the machine learning landscape for some time to come.[1]

Why Companies Prefer Python?

Python has top the charts in the recent years over other programming languages like C, C++ and Java and is widely used by the programmers. The language has undergone a drastic change since its release 25 years ago as many add-on features are introduced. The Python 1.0 had the module system of Modula-3 and interacted with Amoeba Operating System with varied functioning tools. Python 2.0 introduced

in the year 2000 had features of garbage collector and Unicode Support. Python 3.0 introduced in the year 2008 had a constructive design that avoids duplicate modules and constructs. With the added features, now the companies are using Python 3.5.

The software development companies prefer Python language because of its versatile features and fewer programming codes. Nearly 14% of the programmers use it on the operating systems like UNIX, Linux, Windows and Mac OS. The programmers of big companies use Python as it has created a mark for itself in the software development with characteristic features like-

- Interactive
- Interpreted
- Modular
- Dynamic
- Object-oriented
- Portable
- High level
- Extensible in C++ & C

Advantages or Benefits of Python

The Python language has diversified application in the software development companies such as in gaming, web frameworks and applications, language development, prototyping, graphic design applications, etc. This provides the language a higher plethora over other programming languages used in the industry. Some of its advantages are-

- **Extensive Support Libraries**

It provides large standard libraries that include the areas like string operations, Internet, web service tools, operating system interfaces and protocols. Most of the highly used programming tasks are already scripted into it that limits the length of the codes to be written in Python.

- **Integration Feature**

Python integrates the Enterprise Application Integration that makes it easy to develop Web services by invoking COM or COBRA components. It has powerful control capabilities as it calls directly through C, C++ or Java via Jython. Python also processes XML and other markup languages as it can run on all modern operating systems through same byte code.

- **Improved Programmer's Productivity**

The language has extensive support libraries and clean object-oriented designs that increase two to ten fold of programmer's productivity while using the languages like Java, VB, Perl, C, C++ and C#.

- **Productivity**

With its strong process integration features, unit testing framework and enhanced control capabilities contribute towards the increased speed for most applications and productivity of applications. It is a great option for building scalable multi-protocol network applications.

Limitations or Disadvantages of Python

Python has varied advantageous features, and programmers prefer this language to other programming languages because it is easy to learn and code too. However, this language has still not made its place in some computing arenas that includes Enterprise Development Shops. Therefore, this language may not solve some of the enterprise solutions, and limitations include-

- **Difficulty in Using Other Languages**

The Python lovers become so accustomed to its features and its extensive libraries, so they face problem in learning or working on other programming languages. Python experts may see the declaring of cast “values” or variable “types”, syntactic requirements of adding curly braces or semi colons as an onerous task.

- **Weak in Mobile Computing**

Python has made its presence on many desktop and server platforms, but it is seen as a weak language for mobile computing. This is the reason very few mobile applications are built in it like Carbonnelle.

- **Gets Slow in Speed**

Python executes with the help of an interpreter instead of the compiler, which causes it to slow down because compilation and execution help it to work normally. On the other hand, it can be seen that it is fast for many web applications too.

- **Run-time Errors**

The Python language is dynamically typed so it has many design restrictions that are reported by some Python developers. It is even seen that it requires more testing time, and the errors show up when the applications are finally run.

- **Underdeveloped Database Access Layers**

As compared to the popular technologies like JDBC and ODBC, the Python's database access layer is found to be bit underdeveloped and primitive. However, it cannot be applied in the enterprises that need smooth interaction of complex legacy data.[2]

Unlike human languages, the Python vocabulary is actually pretty small. We call this "vocabulary" the "reserved words". These are words that have very special meaning to Python. When Python sees these words in a Python program, they have one and only one meaning to Python. Later as you write programs you will make up your own words that have meaning to you called variables. You will have great latitude in choosing your names for your variables, but you cannot use any of Python's reserved words as a name for a variable. When we train a dog, we use special words like "sit", "stay", and "fetch". When you talk to a dog and don't use any of the reserved words, they just look at you with a quizzical look on their face until you say a reserved word. For example, if you say, "I wish more people would walk to improve their overall health", what most dogs likely hear is, "blah blah blah walk blah blah blah blah." That is because "walk" is a reserved word in dog language. Many might suggest that the language between humans and cats has no reserved words1 . The reserved words in the language where humans talk to Python include the following:

```
and       del       global      not       with
as        elif      if          or        yield
assert    else      import      pass
break     except    in          raise
class     finally   is          return
continue  for       lambda      try
def       from      nonlocal    while
```

That is it, and unlike a dog, Python is already completely trained. When you say "try", Python will try every time you say it without fail. We will learn these reserved words and how they are used in good time, but for now we will focus on the Python equivalent of "speak" (in human-to-dog language). The nice thing about telling Python to speak is that we can even tell it what to say by giving it a message in quotes:

```
print('Hello world!')
```

And we have even written our first syntactically correct Python sentence. Our sentence starts with the function print followed by a string of text of our choosing enclosed in single quotes.

Values and types

A value is one of the basic things a program works with, like a letter or a number. The values we have seen so far are 1, 2, and 'Hello, World!'. These values belong to different types: 2is an integer, and 'Hello, World!'is a string, so-called because it contains a "string" of letters. You (and the interpreter) can identify strings because they are enclosed in quotation marks. If you are not sure what type a value has, the interpreter can tell you.

```
>>> type('Hello, World!')
<type 'str'>
>>> type(17)
<type 'int'>
```

Not surprisingly, strings belong to the type strand integers belong to the type int. Less obviously, numbers with a decimal point belong to a type called float, because these numbers are represented in a format called floating-point.

```
>>> type(3.2)
<type 'float'>
```

What about values like '17'and '3.2'? They look like numbers, but they are in quotation marks like strings.

```
>>> type('17')
<type 'str'>
>>> type('3.2')
<type 'str'>
```

Variables

One of the most powerful features of a programming language is the ability to manipulate variables. A variable is a name that refers to a value. An assignment statement creates new variables and gives them values:

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.1415926535897932
```

This example makes three assignments. The first assigns a string to a new variable named message; the second gives the integer 17to n; the third assigns the (approximate) value of π to pi. A common way to represent variables on paper is to write the name with an arrow pointing to the variable's value. This kind of figure is called a state diagram because it shows what state each of the variables is in (think of it as the variable's state of mind). Figure 2.1 shows the result of the previous example. The type of a variable is the type of the value it refers to[3].

```
>>> type(message)
<type 'str'>
>>> type(n)
<type 'int'>
>>> type(pi)
<type 'float'>
```

Variable names and keywords

Programmers generally choose names for their variables that are meaningful—they document what the variable is used for. Variable names can be arbitrarily long. They can contain both letters and numbers, but they have to begin with a letter. It is legal to use uppercase letters, but it is a good idea to begin variable names with a lowercase letter (you'll see why later).

The underscore character, _, can appear in a name. It is often used in names with multiple words, such as my_nameor airspeed_of_unladen_swallow. If you give a variable an illegal name, you get a syntax error:

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

76trombonesis illegal because it does not begin with a letter. more@is illegal because it contains an illegal character, @. But what's wrong with class? It turns out that classis one of Python's keywords. The interpreter uses keywords to recognize the structure of the program, and they cannot be used as variable names. Python 2 has 31 keywords:

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

Operators and operands

Operators are special symbols that represent computations like addition and multiplication. The values the operator is applied to are called operands [1]. The

operators **+, -, *,** /and ****** perform addition, subtraction, multiplication, division and exponentiation, as in the following examples:

```
20+32   hour-1   hour*60+minute   minute/60   5**2   (5+9)*(15-7)
```

Expressions and statements

An expression is a combination of values, variables, and operators. A value all by itself is considered an expression, and so is a variable, so the following are all legal expressions (assuming that the variable xhas been assigned a value):

```
17
x
x + 17
```

A statement is a unit of code that the Python interpreter can execute. We have seen two kinds of statement: print and assignment. Technically an expression is also a statement, but it is probably simpler to think of them as different things. The important difference is that an expression has a value; a statement does not [1].

Functions Not all operators use the binary operator syntax. An alternative syntax is termed the function call notation. In this notation the name of the operation is given first, followed by a list of the arguments surrounded by parenthesis. For example, the abs operation returns the absolute value of the argument:

```
>>> abs(-3)
3
```

Just as the parenthesis in an arithmetic expression indicated that the enclosed expression needed to be evaluated first, the arguments to a function are first calculated, and then the function is applied:

```
>>> abs(2 - 3 * 7)
```

first calculate 2-3*7, which is -19 19

The function len returns the number of characters (that is, the length) of a string

```
>>> len('abc')

 3

>>> len('ha' * 4)

8
```

Overview on numerical add-on modules

NumPy and SciPy are open-source add-on modules to Python that provide common mathematical and numerical routines in pre-compiled, fast functions. These are growing into highly mature packages that provide functionality that meets, or perhaps exceeds, that associated with common commercial software like MATLAB. The NumPy (Numeric Python) package provides basic routines for manipulating large arrays and matrices of numeric data. The SciPy (Scientific Python) package extends the functionality of NumPy with a substantial collection of useful algorithms, like minimization, Fourier transformation, regression, and other applied mathematical techniques.

If you installed Python(x,y) on a Windows platform, then you should be ready to go. If not, then you will have to install these add-ons manually after installing Python, in the order of NumPy and then SciPy. Installation files are available for both at:

http://www.scipy.org/Download

Follow links on this page to download the official releases, which will be in the form of .exe install files for Windows and .dmg install files for MacOS. [5]

Importing the NumPy module

There are several ways to import NumPy. The standard approach is to use a simple import statement:

```
>>> import numpy
```

However, for large amounts of calls to NumPy functions, it can become tedious to write numpy.X over and over again. Instead, it is common to import under the briefer name np:

```
>>> import numpy as np
```

The central feature of NumPy is the array object class. Arrays are similar to lists in Python, except that every element of an array must be of the same type, typically a numeric type like float or int. Arrays make operations with large amounts of numeric data very fast and are generally much more efficient than lists. An array can be created from a list:

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1., 4., 5., 8.])
>>> type(a)
 <type 'numpy.ndarray'>
```

Here, the function array takes two arguments: the list to be converted into the array and the type of each member of the list. Array elements are accessed, sliced, and manipulated just like lists:

```
>>> a[:2]
array([ 1., 4.])
>>> a[3] 8.0
>>> a[0] = 5.
>>> a
 array([ 5., 4., 5., 8.])
```

Arrays can be multidimensional. Unlike lists, different axes are accessed using commas inside bracket notation. Here is an example with a two-dimensional array (e.g., a matrix):

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
 >>> a
 array([[ 1., 2., 3.], [ 4., 5., 6.]])
>>> a[0,0]
1.0
>>> a[0,1]
2.0
```

Conditionals

The if construct

```
if condition:
    block
```

executes a block of statements (which must be indented) if the condition returns true. If the condition returns false, the block skipped. The if conditional can be followed by any number of elif (short for "else if") constructs

```
elif condition:
    block
```

which work in the same manner. The else clause

```
else:
    block
```

can be used to define the block of statements which are to be executed if none of the if-elif clauses are true. The function sign of a below illustrates the use of the conditionals.

```
def sign_of_a(a):
    if a < 0.0:
        sign = 'negative'
    elif a > 0.0:
        sign = 'positive'
    else:
        sign = 'zero'
    return sign

a = 1.5
print 'a is ' + sign_of_a(a)
```

Running the program results in the output

```
a is positive
```

Loops

The while construct

```
while condition:
    block
```

executes a block of (indented) statements if the condition is true. After execution of the block, the condition is evaluated again. If it is still true, the block is executed again. This process is continued until the condition becomes false. The else clause

```
else:
    block
```

can be used to define the block of statements which are to be executed if condition is false. Here is an example that creates the list [1, 1/2, 1/3,...]:

```
nMax = 5
n = 1
a = []               # Create empty list
while n < nMax:
    a.append(1.0/n)  # Append element to list
    n = n + 1
print a
```

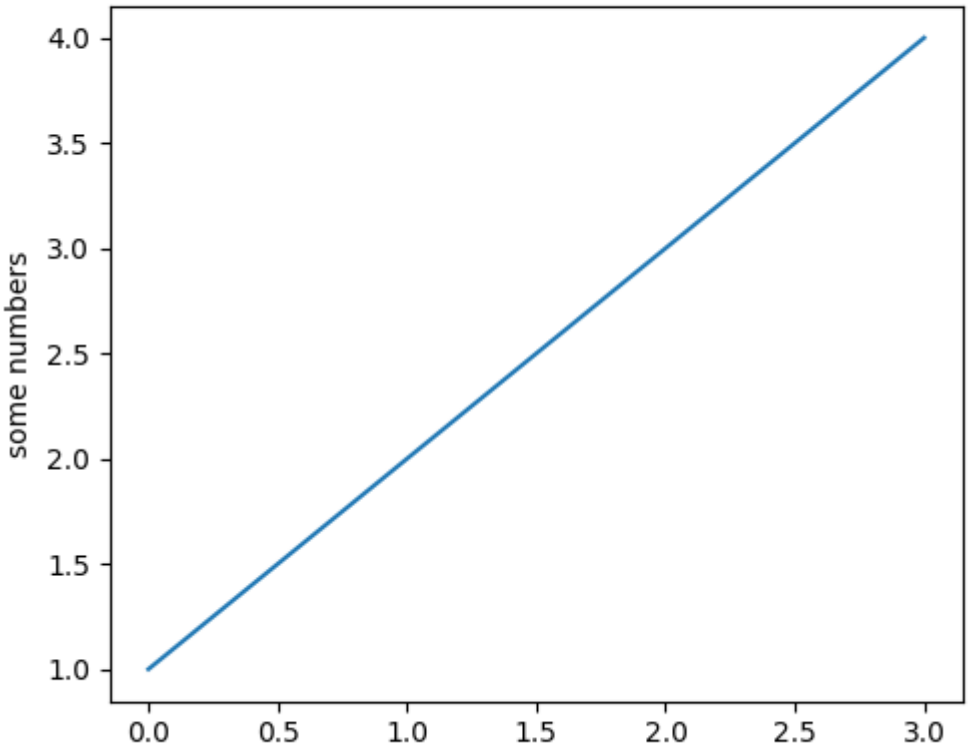
The output of the program is

```
[1.0, 0.5, 0.33333333333333331, 0.25]
```

Plotting in Python

matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that “axes” here and in most places in the documentation refers to the *axes* part of a figure and not the strict mathematical term for more than one axis).

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```

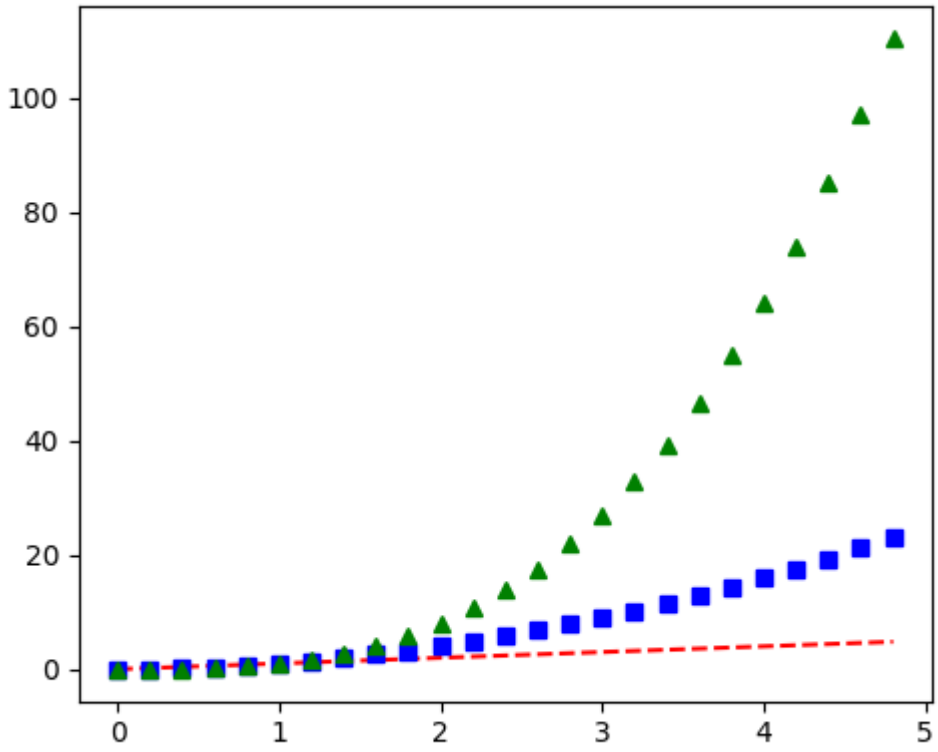


If matplotlib were limited to working with lists, it would be fairly useless for numeric processing. Generally, you will use numpy arrays. In fact, all sequences are converted to numpy arrays internally. The example below illustrates a plotting several lines with different format styles in one command using arrays [7].

```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```


100
80
60
40
20
0
0
1
2
3
4
5

References:

[1]. The Python Language Reference Website https://docs.python.org/3/reference

[2]. Mindfire Solutions software service provider," Advantages and Disadvantages of Python Programming Language", https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121

[3]. Allen Downey, "Think Python", Green Tea Press, Needham, MA, USA

[4]. Timothy A. Budd, "Exploring Python", PythonAnywhere.com.

[5]. Introduction to Numeric Python, Principles of modern molecular simulation methods Course, College of Engineering,UC Santa Barbara University, USA

[6]. Jaan Kiusalaas, "NUMERICAL METHODS IN ENGINEERING WITH Python", Cambridge University press, Cambridge UK.

[7]. Pyplot tutorial, Matplotlib's Python 2D plotting library, https://matplotlib.org.