COMP 205: Survey of Computer Languages
Programming Languages History Paper

Diana Collins

May 24, 2013

Contents

1 Abstract

Python is an open source, multi-paradigm language with numerous and extensive libraries and third party modules available. Among its many features are readability, dynamic typing, and exception handling. There are many modules available that make developing in Python quicker and more productive. There are also a number utilities available for developing in Python.

2 Executive Summary

Python is a scripting language with many features, libraries, and modules which make it a good choice for scientific and web programming. It is used by well known organizations such as Google and NASA. It is also an easy to learn and read language making it a good choice for beginner and seasoned programmers.

Python is an open source language, allowing programmers access to source and the ability to extend and improve the language. This has made Python more stable and has added to its diversity and completeness. There are many libraries and third party modules available to developers.

There are a number of mathematical modules and libraries in existence. SciPy and NumPy are just two examples of such libraries. These libraries make development of math computing software, such as Sage, ideal in Python. Besides being used for performing math computations, the scientific community also uses Python in artificial intelligence programming and natural language processing.

Python is widely used for web development. Using the Web Server Gateway Interface a standard API and a number of web application frameworks have been developed. Twisted is an example of one such framework. It is used to facilitate communications between computers and has been utilized by Dropbox.

3 Tutorial-style Illustration of Language Features

3.1 Language Background Information

Python was developed by Guido Rossum. Rossum became frustrated with his work environment/language interaction and decided to create a new language that would bridge the gaps while at the same time continue to provide the features he liked in the ABC language. The Python language did just that. Not only does it offer the syntax, semantics, and exception handling of ABC's Module 3, but it is also easy to extend and now offers many libraries and third party modules. [1]

3.2 Technical Language Information

Python is an open source language. It is multi-paradigm including object-oriented/imperative and has limited support for functional programming. [2] Python is also dynamically typed. [2] Implementing Python is relatively easy. It can be run directly from the command line or other IDEs. There are a number of desktop IDEs, as well as, browser based IDEs. [2] Version 3.3.2 of the Python documentation page can be found at the following web address: http://docs.python.org/3/. [1]

3.3 Major Features of the Python Language

3.3.1 Easy to Read Code

Python's syntax offers easy to read code. It incorporates white space and english keywords for a user friendly and an easy to learn and read experience. [3] The following are examples in Python and Java. Notice that Python does not require you to specify type nor does it require punctuation at the end of the line or brackets to contain code blocks. It relies on strict syntax and code neatness to accomplish these things.

Python Code

```
1 name = "John"
2 age = 23
3
4 if name == "John" and age == 23:
5     print "Your name is John, and you are also 23 years old."
6
7 if name == "John" or name == "Rick":
8     print "Your name is either John or Rick."
```

*Code example taken from www.learnpython.org [4].

Java Code

```
1 String name = "John";
2 int age = 23;
3
4 if (name == "John" && age == 23) {
5       System.out.println("Your name is John, and you are also 23 years old."
6 }
7
8 if (name == "John" || name == "Rick") {
9     System.out.println("Your name is either John or Rick.");
10 }
```

3.3.2 Strong Introspective Capabilities

Python offers a number of built-in functions which enable the programmer to inspect an object. This collection of functions enable the programmer to examine an object in order to learn what it is and what it is capable of doing [5]. The following example is a function that when called will print information about an object including; name, class, id, type, vale, if it callable, and the first line of the doc. This function accomplishes this by using a number of python's built-in introspective functions.

Function Code

```
1 def interrogate(item):
2     """Print useful information about item."""
3     if hasattr(item, '__name__'):
4         print "NAME:     ", item.__name__
5     if hasattr(item, '__class__'):
6         print "CLASS:    ", item.__class__.__name__
7     print "ID:       ", id(item)
8     print "TYPE:     ", type(item)
9     print "VALUE:    ", repr(item)
10    print "CALLABLE:",
11    if callable(item):
12        print "Yes"
13    else:
14        print "No"
15    if hasattr(item, '__doc__'):
16        doc = getattr(item, '__doc__')
17        doc = doc.strip()   # Remove leading/trailing whitespace.
18        firstline = doc.split('\n')[0]
19        print "DOC:      ", firstline
```

*Code example taken from www.ibm.com [5].

Code Output

The following is a call to the above function and the subsequent output.
interrogate('a string')

```
CLASS:    str
ID:       141462040
TYPE:     <type 'str'>
VALUE:    'a string'
CALLABLE: No
DOC:      str(object) -> string
```

3.3.3 Object Orientation

Just like in other OOP languages, Python uses objects, usually instances of classes, to interact with each other thus achieving the goal of the program. Python has a special method, __init__(), which can be defined so that when the object is instantiated it will have a customized initial state [6]. There is also a built-in attribute, __doc__, that will return the class description [6].

A Simple Class

```
1 class Simple:
2     """A simple class example"""
3     i = 12345
4     def f(self):
5         return 'hello world'
```

*Code example taken from docs.python.org [6].

Code Output

The following is a call to the above function and the subsequent output.

```
1 x = Simple()     #create an instance of Simple()
2 x.i              #returns 12345
3 x.f              #returns 'hello world'
4 x.__doc__        #returns "A simple class example"
```

Class with __init__() Defined

```
1 class Complex:
2     """A class with an initial state"""
3     def __init__(self, realpart, imagpart):
4         self.r = realpart
5         self.i = imagpart
```

*Code example taken from docs.python.org [6].

Code Output

The following is a call to the above function and the subsequent output.

```
1 x = Complex(3.0, -4.5)     #create an instance of Complex()
2 x.r, x.i                   #returns (3.0, -4.5)
```

*Code example taken from docs.python.org [6].

3.3.4 Dynamic Data Types

Being dynamically typed slows Python's runtime, but it also makes development in Python faster because the programmer has more freedom and flexibility and doesn't have to waste time making choices about or declaring types.

In statically types languages the sequence of the following lines would cause an error depending on what employeeName is declared to be, an int or string. In dynamically typed languages it would be acceptable. [7]

```
1 employeeName = 9
2 employeeName = "Steve Ferg"
```

*Example taken from pythonconquerstheuniverse.wordpress.com [7].

3.3.5 Exception-Based Error Handling

During development Python was given the ability to throw exceptions in response to errors. This feature allows the programmer to do something with the exception, correcting the error if possible, during runtime so that the program can continue executing entirely [8]. Exceptions are handled in Python using a "try-except" block. The code that might trow an exception in placed in the try portion of the block while the code meant to fix the error is placed in the except block. If the try code throws and exception it is matched to the appropriate exception specified to the right of the keyword exception. That code is then executed and then program execution continues where it left off [9].

The following is an example of a try-except block.

```
1 while True:
2     try:
3         x = int(raw_input("Please␣enter␣a␣number:␣"))
4         break
5     except ValueError:
6         print "Oops!␣␣That␣was␣no␣valid␣number.␣␣Try␣again..."
```

*Code example taken from docs.python.org [9].

3.3.6 Extensive Libraries and Third Party Modules

There is an extensive library available to Python programmers. It is usually included with Python distributions. References for this library can be found at http://docs.python.org/2/library/.

One thing that was important to Guido van Rossum that he want to incorporate into Python during its development is the ability for others to extend the language by writing their own modules. Because this feature was included and because Python is open source there are many third party modules available. A list of these modules can be found at https://pypi.python.org/pypi?:action=browse&c=533&show=all.

3.3.7 Python's Functional Programming Capabilities

While Python does not fully support functional programming, it does have some features that make it possible. Among these features are the iterators, list comprehensions, generator expressions, and lambda [12].

An iterator is an object that represents a stream of data returning one element at a time. Python has a number of data structures that support iteration, the most common being lists and dictionaries [12].

List comprehensions and generator expressions were borrowed from the functional programming language, Haskell [12]. Often there is a to perform some operation on each element or choose a sub set of elements in an iterator's output. These common tasks can be done using the listcomps and genexps that were inspired by Haskell. List comprehensions, or listcomps, return a list containing the result of the operation, while the generator expressions return an iterator that computes values as needed. Listcomps are therefore better suited for finite operations and genexps are better able to handle infinite or much larger tasks [12].

Lambda is a statement that can be used to write quick functions. It takes parameters, an expression to combine them, and returns a value. It can be used to combine a number of small functions, thus reducing code. Because it is possible to "put too much" in a Lambda statement it can become difficult to read and is therefore recommended that it be used sparingly [12].

The following is an example of a function written using the Lambda statement and then without, the latter is easier to read and understand.

Function written with Lambda

```
1 lowercase = lambda x: x.lower()
```

Function written without Lambda

```
1 def lowercase(x):
2     return x.lower()
```

*Example taken from docs.python.org [12]

3.3.8 Python vs. Java

There are a number of differences between the Python and Java programming languages. Java is a more "verbose" language than Python [10]. You can accomplish the same task with just one line in Python and in Java it takes 7 lines.

Java Code

```
1 public class HelloWorld
2 {
3     public static void main (String[] args)
4     {
5         System.out.println("Hello, world!");
6     }
7 }
```

Python Code

```
1 print "Hello, world!"
```

*Example taken from pythonconquerstheuniverse.wordpress.com [10].

Another way these languages differ is their typing. Java is statically typed while Python uses dynamic typing. Speed is another difference. Java is compiled making its runtime fast than interpreted Python. Both language are object-oriented, reflective, and imperative, but Python also incorporates some functional and procedural principles too.

Both languages have a faithful following and there are pros and cons to each. There are also a couple ways to implement Python and Java together. You can use Jython or its predecessor JPython [11].

4 A Real-World Example

The following script will read in a file containing numbers. Each line will be parsed, separated, and stripped of whitespace and new line characters. Each line item in the list is the separated, converted into an int, summed, and the sum is appended to an array. The original contents free of new lines and whitespace and the sum array are then printed out.

Python Sum of Digits Script

```
1
2 s = 0
3 sum = []
4
5 f = open('Desktop/numbers.txt', 'r')
6
7 contents = list(f)
8
9 contents = [c.strip('\n') for c in contents]
10 contents = [c.replace('␣', '') for c in contents]
11
12 for c in contents:
13     nums = list(c)
14     for n in nums:
15         num = int(n)
16         s = s + num
17     sum.append(s)
18     s = 0
19
20 print contents, sum
```

Comments for above code.

```
lines 1 and 2          #declare variables s and sum for later use
line 4                 #open file numbers.txt
line 6                 #list contents of file
line 8                 #strip newline characters from contents strings
line 9                 #removes all whitespace
lines 11 and 12        #list each character of each item in contents
```

```
lines 13, 14, and 15    #convert string items in nums to int and sum them
lins 16 and 17          #append sum of nums to sum array
line 19                 #print contents list and sum array
```

Input Text File, numbers.txt

```
12
123
12 34
12345
```

Script Output

```
['12', '123', '1234', '12345'] [3, 6, 10, 15]
```

5 Instructor Feedback and Revision

The following suggestions were made to me by Professor Gudivada.

Abstract is too short. It does not convey the essence of Python language.

Executive Summary also lacks technical detail. In addition to its syntactic simplicity and readability, Python is heavily used in Natural Language Processing Research and Scientific Computing (NumPy and SciPy). It is also popular for symbolic and mathematical computing (SAGE is an excellent example here). Dictionary is a major data structure that Python program heavily use. These are just a few examples. This type of detail is not present. History is not so important in the Executive Summary section.

In section 3:

In addition to the features you have mentioned, Python is also well known for functional programming features.

The following revisions were made based on these suggestions.

```
The abstract was rewritten.

The Executive summary was revised and rewritten to better reflect
Python's technical features.

A functional programming sub-section was added to section 3.
```

6 Peer Feedback and Revision

```
The following suggestion was made to me by Bradley Woods.

Grammatical error on page 2, consider revising sentence,
"This is counter balance though with a quick development time
and the availability IDEs."

The following revision was made based on Bradley's suggestion.

Sentence on page 2 was revised to be more grammatically correct.
```

7 Metacognitive Reflection

1. Did I solve the right problem?

   While there is still much for me to learn about the Python programming language, I believe I have thoroughly researched and described it her. I have also included a number of examples to demonstrate how it is implemented and have created my own script which could easily be modified for a real-world application.

2. Did I solve the problem right?

   After careful review and much research I wrote this paper, included examples, and wrote a script of my own, following all guidelines given to me.

3. How did I approach the solution to the problem?

   I first began with an online search, seeking as much information as I could find on the Python language. I quickly found the official Python website, python.org, which was a big contributor to the knowledge I

acquired while writing this paper. I took notes, organized them according to the template provided, and then looked for code samples which demonstrated the various features of this language. After completing tutorial section of my paper I began looking for inspiration for a real-world example that I would write myself. Being that I am not good and coming up with my own programming assignments I looked on a website called CodeEval.com. This site gives a number of programming challenges. I found one that interested me and that I thought could be modified and applied in a real life scenario and completed it. I believe the methods I used to write this paper served my purpose well.

4. What strategies and techniques did I draw upon?

   I used my ability to comprehend readings, search the internet, and not loose confidence and ambition when my code failed.

5. Did I learn a new strategy in completing this assignment? If so, how is it different from and similar to the repertoire of techniques that I have already acquired?

   I don't think I learned a new strategy, but I did learn a lot. I learned a lot about the Python programming language and I learned how to run scripts from the terminal, something I have been trying to get better at using.

6. Any other information you may wish to add $\cdots$

   I enjoyed this assignment very much. If I can find time I might use this template to guide my learning and experimenting with other languages. This is a good guide that helps to keep you organized as you go through the process of researching and learning a new language, something that is very necessary for this task.

8 Self-assessment

Rubric line item	Max possible points	Earned points
Abstract concise and precise	5	5
Executive Summary section concise, comprehensive, and uses precise technical terminology	10	10
Language features are illustrated with simple and relevant examples elegantly	35	35
Real-world example application is substantial and aptly demonstrates the salient features	15	15
Instructor feedback is solicited and incorporated	5	5
Peer feedback is solicited and incorporated	5	5
Meta-cognitive reflection is completed	5	5
Turnitin.com task is completed	5	5
Self-assessment is performed	5	5
References are formatted according to the IEEE style	5	5
Paper is of professional quality	5	5
Total points	100	100

9 References

[1] Python Programming Language – Official Website (Online). Available: http://python.org.

[2] Python (programming language) (Online). Available: http://en.wikipedia.org/wiki/Python_(programming_language).

[3] Python syntax and semantics (Online). Available: http://en.wikipedia.org/wiki/Python_syntax_and_semantics

[4] Conditions Tutorial (Online). Available: http://www.learnpython.org/Conditions.

[5] O'Brien, P. Guide to Python introspection (Online). Available: http://www.ibm.com/developerworks/library/l-pyint/index.html.

[6] Classes (Online). Available: http://docs.python.org/2/tutorial/classes.html.

[7] Ferg, S. (2012). Static vs. dynamic typing of programming languages (Online). Available: http://pythonconquersthe universe.wordpress.com/2009/10/03/static-vs-dynamic-typing-of-programming-languages/.

[8] Exception handling (Online). Available: http://en.wikipedia.org/wiki/Exception_handling.

[9] Errors and Exceptions (Online). Available: http://docs.python.org/2/tutorial/errors.html.

[10] Ferg, S. (2009). Python & Java: A Side-by-Side Comparison (Online). Available: http://pythonconquerstheuniverse.wordpress.com/2009/10/03/python-java-a-side-by-side-comparison/.

[11] What is Jython? (Online). Available: http://www.jython.org/archive/21/docs/whatis.html

[12] Functional Programming HOWTO (Online). Available: http://docs.python.org/2/howto/functional.html.