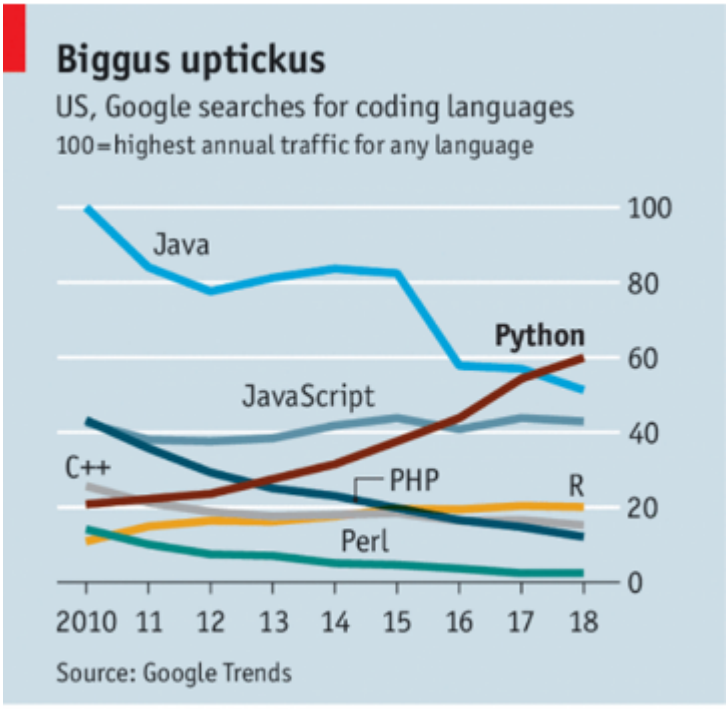
PYTHON VS JAVA: WHY PYTHON IS BECOMING MORE POPULAR THAN JAVA

The Economist magazine, who does normally write about IT, recently reported that Python has or may become more popular than Java. Some people, like me, applaud that. Here I explain why I prefer Python to Java and why many other people would agree.

Recent Google Trends for popular programming languages show Python overtaking Java in the past year or so.



Before Java, there was only COBOL and C++

Like most programmers with a couple decades of experience, I know Java, because it has been around a long time. Sun Microsystems developed Java in 1991. Back then it was nothing short of astounding. People were mightily impressed when the original developers gave a presentation and made this famous original Java logo, named Duke, dance around in the Netscape Browser. At that time Netscape dominated the browser market, and all a browser could do was show text. Making that icon move thrust Java into the mainstream.

Java also made it possible to write object-oriented code without having to use C++. C++ is notoriously difficult, mainly because you have to manage objects in memory yourself. And if you did not use C++ you used COBOL. COBOL programs could not be divided into objects. The code got so long after years of maintenance that it was difficult to understand. And that was made worse because COBOL allowed the GOTO statement, a concept that is in complete opposition to the idea of keeping single blocks of logic near each other.

Since that Netscape demonstration, Java gained in popularity the way most things do. As more programmers got training on it, the open source market added more extensions and tools to make it even more useful, that lead to more users, which led to more extensions and tools, and so forth.

Enter Python

In 2006, I started using a language called Python when other people were using perl and bash shells to do system administration. Python had many advantages over Java which I will describe below. But the main two are: (1) it is easier to understand and (2) it is an interpreted language.

An interpreted language means you can type commands into the command line interpreter and it

responds right away. (Technically, that is called **REPL**, Read, Evaluate, Print, Loop.) That make debugging code easy as you can get small paragraphs working as you work on the larger program. With Java you can only use the debugger, which lets you execute code one line at a time, but not change it.

Nothing could be simpler than REPL. Type **python** and then **3+5** and the computer responds with 5.

```
python

3+2
5
```

Now consider doing this with Java.

First, we cannot write this one line at a time as Java has no command line interpreter. So to print 5, like we did above, we have to write a complete program and then compile it. Here is Print5.java:

```
public class Print5 {

    public static void main(String[] args) {
      System.out.println("3+2=" + (Integer.toString(3+2)));
    }
}
```

Then compile it type **javac Print5.java** and run it with **java Print5**.

```
java Print5
3+2=5
```

We had to make a complete program to print 5. That includes a **class** and a **main** function, which tells Java where to start.

We can also have a main function with Python, which you usually do when you want to pass it arguments. It looks like this:

```
def main():
  print('3+2=', 3+2)

if __name__== "__main__":
  main()
```

The only thing that makes that lengthy is this rather bizarre-looking command:

```
if __name__== "__main__":
```

This tells Python to only execute this function. Otherwise it would try to execute any utilities you imported, like all the code in a JSON interpreter.

Classes

Python code runs top to bottom unless you tell it where to start. But you can also make classes, like with Java, like this:

```
class Number:
  def __init__(self, left, right):
    self.left = left
    self.right = right

number = Number(3, 2)

print("3+2=", number.left + number.right)
```

The class **Number** has two member variables **left** and **right**. The default constructor is __init__. We instantiate the object by calling the constructor **number = Number(3, 2)**. We can then refer to the variables in the class as **number.left** and **number.right**. Referring to variables like that directly is frowned upon in Java. Insead you are supposed to use getter and setter functions, as we show below.

Here is how you would do that same thing In Java. As you can see it is wordy, which is the main complaint people have with Java. Below we explain some of this code.

```
class PrintNumber {
    int left;
    int right;

    PrintNumber(int left, int right) {
      this.left = left;
      this.right = right;
    }

    public int getleft() {
        return left;
    }
    public int getRight() {
        return right;
    }
}

public class Print5 {

    public static void main(String[] args) {
         PrintNumber printNumber = new PrintNumber (3,2);
         String sum = Integer.toString(printNumber.getleft()
              + printNumber.getRight() );
         System.out.println("3+2=" + sum);
    }
}
```

Python is gentle in its treatment of variables. It can, for example, print dictionary objects automatically. With Java you would have to use a function that does specifically that or write your own. And it also casts variables of one type to another to make it easy to, for example, print a string

and an integer.

On the other hand, Java has strict type checking. This helps avoid runtime errors. Below we declare an **array of Strings** called **args**.

```
String[] args
```

You usually put each Java class in its own file. But here we put two classes in one file to make compiling and running the code simpler. We have:

```
class PrintNumber {

    int left;
    int right;
}
```

That class has two member variables **left** and **right**. In Python, we did not need to declare them first. We just did that on-the-fly using the **self** object.

In most cases Java variables should be **private**, meaning you cannot refer to them directly outside of the class. Instead you use getter functions to retrieve their value. Like this.

```
public int getleft() {
        return left;
    }
```

So in the **main** function we instantiate that class and retrieve its values:

```
public int getleft() {
        return left;
    }

public static void main(String[] args) {

         PrintNumber printNumber = new PrintNumber (3,2);
         String sum = Integer.toString(printNumber.getleft()
              + printNumber.getRight() );
```

We just said that Python is gentle in its treatment of variables. That's not true with Java. For example we cannot concatenate and print numbers and letters **"3+2=" + 3 + 2**. So we have to use this function to convert each integer to a string **Integer.toString()** and then print the concatenation of two strings.

The Complexity of Jar Files

Python is built into Linux because some Linux functions depend on it. Java is not, although you can easily install it. And Python functions can be retrieved from public repositories using pip install <module>. With Java you have to download .jar files. That is such complexity in getting the correct versions and correct set of dependencies that even more complex tools like ant and maven have been developed to help gather all that.

A Wealth of Functions

Python opensource contributors have made functions that make difficult tasks easier, like reading csv files and working with JSON objects. So does Java. But Python has even larger functions that do really complicated tasks like working with large n-dimensional arrays, i.e., Pandas.

Python and Big Data

Mentioning Pandas leads us into the next reason you would want to be using Python: big data and analytics. When you work with big data, the data has to be translated into matrices. Pandas makes multiplying and slicing those arrays much simpler.

And data scientists, including those working at Google, are pouring much effort into developing Python language machine learning libraries, like TensorFlow and Keras, which is a simplification of Tensorflow, and scikit-learn, which started in academia. Data scientists have also developed matplotlib in Python which makes creating graphs easier.

To be fair, there are machine learning libraries in Java as well. One example is Apache Spark ML, which is designed to work with the all-memory Apache Spark big data database, but you can you use Spark ML with Java, Python, and Scala. But Spark itself uses Scala and Python in its interactive interpreter.

Move to Scala

Scala is a good bridge for those programmers who want to move away from the wordy syntax, abstract methods, and interfaces that Java requires. It lets you use Java functions yet write code that is more compact and perhaps easier to read.

One problem with Scala, however, is it is a **functional programming language**. That means it is designed such you can write code in a functional way. That means all operations are done as functions. No variable stands by itself. And it uses mathematical concepts like domain checking. That makes for complicated code that ordinary programmers will have a hard time understanding. But you can use Scala and ignore the functional part and just write code as you normally would.

Python Made Easy

So I would say that for programmers who are just starting their careers, or students who are learning coding in college, Python would be the easiest way to go. It's easier for the beginner than Java to understand, yet it has all the advanced features of Java, like the ability to extend objects and create multiple threads.