Selecting the optimal programming language

Factors to consider

Jerry Reghunadh
Neha Jain

There are many programming languages to choose from, and it's a personal choice for many--you might just pick your favorite, or you might choose the one with the best performance figures. Sometimes, however, other factors are just as important as performance. In this article, learn how to analyze the relevant factors when selecting a programming language. A few project scenarios are outlined to illustrate different variables in your myriad choices.

Introduction

When planning a software solution, you have many different programming languages to choose from, and it's easy to get lost in the intricacies of each one. Your choice of language can depend on many factors. If it's for a personal project or hobby, you may settle for a language you know. If your choice depends on available resources, you might end up with really cryptic approaches. Or, you could spend a lot of time developing reusable components, which can cause the documentation to become a nightmare.

This article doesn't contain a redundant comparison of procedural, object-oriented, and functional languages. With practical examples and scenarios, it shows how to select a language with maximum efficiency and ease of development for your project. It helps you examine several factors to consider while selecting a programming language, whether it is for personal use or for a large project within an organization.

Factors to consider

There isn't just one factor to think about when choosing a programming language. For example, while developing a dynamic web page, you might consider JavaServer Pages (JSP)/servlets as the best option, and others might prefer using PHP or a similar scripting language. No single language is the "best choice." Though you might give preference to certain factors, such as performance and security in enterprise applications, other factors, such as fewer lines of code, might be lower priorities. There's always some trade-off.
After you're given a project or assignment, there's often preparation work to be done before solving the actual problem. The choice of language is by far the most overlooked component of this preparation.

When selecting a language for a personal project, you may pick a personal favorite. Lines of code are important here; the obvious choice is a language that can get the work done in 10 instead of 20 lines of code. You want to get the solution out first, and then worry about the neatness, or performance.

For projects built for a large organization, it's a different scenario. Teams will build components that are going to interact and interconnect with each other to solve a particular problem. The choice of language might involve factors such as how easily the program can be ported to a different platform or the availability of resources.

Selecting the right programming language can yield solutions that are concise, easy to debug, easy to extend, easy to document, and easy to fix. Factors to consider when selecting a programming language are:

- The targeted platform
- The elasticity of a language
- The time to production
- The performance
- The support and community

**Targeted platform**

The most important factor to consider is the platform where the program will run. Think in terms of the Java™ language and C. If the program is written in C and needs to be run on Windows® and Linux® machines, it would require platform compilers and two different executables. With the Java language, the byte code generated would be enough to run the program on any machine with a Java Virtual Machine (JVM) installed.

A very similar argument applies for websites. They should all look and work the same across all browsers. Using CSS3 tags, HTML5, without checking browser compatibility, will cause the same site to look and behave differently across browsers.

**Elasticity**

The "elasticity" of a language is the ease with which new features can be added to the existing program. Elasticity can involve adding a new set of functions, or using an existing library to add a new feature. Consider the following questions related to elasticity.

- Can I start using a capability of the language without including a new library?
- If not, is the capability available in the language library?
- If it's not a native capability and not available as a library, what is the effort to build the features from scratch?

Before making a decision, you should know how the program has been designed and what features have been set aside as future improvements.
Though a comparison of these languages is not technically correct, consider Perl and Python. Perl has regular expression support built in as a ready-to-use feature. In the case of Python, you have to import the `re` module from the standard library.

**Time to production**

The time to production is the time it takes to make the program go live—when the code is production-ready and will work the way it's intended. The presentation logic should be added to the control logic when calculating time to production.

Time to production is very dependent on the size of the code. Theoretically, the easier it is to learn a language, the smaller the amount of code and, hence, less time to go live.

For example, a content management site can be developed using PHP scripts in days compared to servlets code that can take months, assuming you are learning both languages from scratch.

**Performance**

You can squeeze only so much performance out of a program and a platform, and the language used to develop the program affects performance. There are many studies comparing how fast programming languages are in the same environment. You can see different computer benchmarks to use as a reference, though the figures are not for concrete assessments of the performance of any language.

Consider a web application written in both Java code and Python. The performance data, as shown in the benchmark, would lead you to conclude that, given similar environments, the application written in the Java language should run faster than the one written in Python. But what about the environment itself? If the environment is an x86 Ubuntu Intel Q6600 one core, it's a fair game because the computational power is limited. What if the web application is in the cloud, running on Google App Engine? You now have access to virtually unlimited processing power, and both the programs are going to return results at almost the same time. The choice factor now revolves around lines of code and maintainability.

The performance of a language should matter when the target environment doesn't offer much scope for scaling. Hand-held devices are an example of such an environment.

**Support and community**

Just as good software needs a community following to help it grow, a programming language should also have a strong community behind it. A language with an active forum is likely to be more popular than even a great language that doesn't have help at hand.

Community support generates wikis, forums, tutorials, and, most importantly, additional libraries that help the language to grow. Gone are the days when people operate in silos. People don't want to skim through all the documentation to get one minor problem solved. If a language has a good following, the chances are good that someone else faced your same issue and wrote about it in a wiki or forum.
Perl is a good example of the importance of community. The Comprehensive Perl Archive Network (CPAN) is a community-driven effort. CPAN’s main purpose is to help programmers locate modules and programs not included in the Perl standard distribution. Its structure is decentralized; authors maintain and improve their own modules. Forking, and creating competing modules for the same task or purpose, are common.

**Scenarios**

The project scenarios in this section illustrate different factors that affect the decision-making process when choosing a language.

- REST service for add operation
- A simple feed reader
- Enterprise applications
- Research projects

**REST service for add operation**

This scenario is for a service that will do addition in the format of a REST service. You’ll invoke a URL, http://<url>?num1=number1&amp;num2=number2, and the result should contain the sum of the two numbers passed to it. You could write the program using different languages. The example here uses JSP, as shown in Listing 1, and PHP, as shown in Listing 2. The JSP program was written in the Eclipse IDE.

**Listing 1. REST service using JSP**

```html
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Sum</title>
</head>
<body>
<% if (request.getParameter("num1") == null ||
request.getParameter("num2") == null) { %>
<p>
<b>Wrong URL!!!</b>
</p>
<p>
Enter URL in this format: </p>
<i>
http://&lt;url&gt;?num1=number1&amp;num2=number2</i>
<% } else { %>
<b>Number 1:</b>
<i><%= request.getParameter("num1") %></i>
<br>
<b>Number 2:</b>
<i><%= request.getParameter("num2") %></i>
<br>
<b>Sum:</b>
<i><%= Integer.parseInt(request.getParameter("num1")) +
Integer.parseInt(request.getParameter("num2")) %></i>
<br>
```
Listing 2 shows the same program in PHP.

**Listing 2. REST service using PHP**

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Sum</title>
</head>
<body>
<?php if ($_GET['num1'] == NULL || $_GET['num2'] == NULL) { ?>
  <p><b>Wrong URL!!!</b></p>
  <p>Enter URL in this format: <b>&lt;i&gt;http://&lt;url&gt;?num1=number1&amp;num2=number2&lt;/i&gt;</b></p>
<?php } else { ?>
  <b>Number 1:</b> <i><?= $_GET['num1'] ?></i>
  <br>
  <b>Number 2:</b> <i><?= $_GET['num2'] ?></i>
  <br>
  <b>Sum:</b> <?= $_GET['num1'] + $_GET['num2'] ?><br>
<?php } ?>
</body>
</html>
```

There isn't much difference between the two examples. The program itself doesn't explore all the capabilities of the two languages. It demonstrates that, when it comes to basics, both the languages are at par.

Features of JSP allow it to be used more at an enterprise level. For example, with JSP, the very first time the program is called it's loaded into the memory as a servlet. For every subsequent request the program in the memory is called, giving better response time with subsequent calls. It's also ideal in a Java environment. With PHP, however, each time the program is called it's loaded into the memory, which might increase response time for critical applications.

Another notable feature that makes JSP a better choice in an enterprise is its multi-threading capabilities. PHP has no built-in support for multi-threading.

**A simple feed reader**

The goal in this scenario is to provide the program with a feed link. The program has to get the feed and list all the titles in the feed. To make it a bit more interesting, you'll subscribe to a JSON-formatted feed and not RSS.

The code snipped in Listing 3 is from O'Reilly and is written in Java code.
Listing 3. Feed reader using Java code

```java
import java.io.InputStream;
import java.net.URL;
import java.net.URLConnection;
import org.apache.commons.io.IOUtils;
import net.sf.json.JSONArray;
import net.sf.json.JSONObject;
import net.sf.json.JSONSerializer;

public class JsonParser {
    public static void main(String[] args) throws Exception {
        String urlString = "http://pipes.yahoo.com/pipes/pipe.run?_id=df36e60df711191549c529e1df96884&_render=json&textinput1=and&urlinput1=http%3A%2F%2Ffeeds.wired.com%2Fwired%2Findex";
        URL url = new URL(urlString);
        URLConnection urlCon = url.openConnection();
        InputStream is = urlCon.getInputStream();
        String jsonTxt = IOUtils.toString(is);
        JSONObject json = (JSONObject) JSONSerializer.toJSON(jsonTxt);
        JSONObject value = json.getJSONObject("value");
        JSONArray items = value.getJSONArray("items");
        String title;
        for (Object item : items) {
            title = ((JSONObject)item).getString("title");
            System.out.println("\n" + title);
        }
    }
}
```

Listing 4 shows the program in Python.

Listing 4. Feed reader using Python

```python
#!/usr/bin/python
import urllib.request
url = "http://pipes.yahoo.com/pipes/pipe.run?_id=df36e60df711191549c529e1df96884&_render=json&textinput1=and&urlinput1=http%3A%2F%2Ffeeds.wired.com%2Fwired%2Findex"
HTTPdata = urllib.request.urlopen(url)
json_data = eval(HTTPdata.read())
for item in json_data['value']['items']:
    print (item['title'])
```

The Python program can be further abridged into just three lines. Retain the first two lines of Listing 4, and replace the rest of the code with the line in Listing 5.

Listing 5. Abridged 3rd line

```python
for item in eval(urllib.request.urlopen("http://pipes.yahoo.com/pipes/pipe.run?_id=df36e60df711191549c529e1df96884&_render=json&textinput1=and&urlinput1=http%3A%2F%2Ffeeds.wired.com%2Fwired%2Findex"))
    print (item['title'])
```

The example application showed the elasticity of these languages. None of them had native support for all the required libraries; you have to import the necessary packages. With Python, it was even simpler because you could manipulate JSON by default. With Java code, it was more difficult because you had to get the JSON libraries and their dependencies to make the program work.
Enterprise applications

With enterprise applications, designers and programmers need to walk a tightrope when it comes to performance, security, maintainability, and development time. It's not just about using the programming language that can get you the best performance figures. Other important factors include: time to production, elasticity, and how well the program can integrate with the existing infrastructure.

The environment in which the program will be used also plays an important part. Programs written at an enterprise level are never stand-alone. Each program becomes part of an even larger goal, so interoperability becomes a factor.

Imagine that an enterprise with its web services implemented in Java code wants to add WebSphere® MQ as a reliable platform. It doesn't make any sense to use the C APIs of WebSphere MQ to write the application; the choice would have to be Java code.

Research projects

Suppose your next project is to do research in fields unrelated to information technology and computers. For example, the project might involve image processing, audio processing, watermarking, or possibly stock market research. You need to create code to simulate certain real-time behaviors, but you aren't much of a computer geek.

The project requires a lot of quick and dirty code. The most relevant factor is time to production. In this case, time to production means how soon you can make the component work so you can get back to the bigger task at hand. It's a lot like writing small stubs, without giving any attention to interoperability, at this stage. The project might become a full-fledged product, but right now it's in its initial stages. Your prime requirement is prototyping.

Languages such as MATLAB and LISP might come to the rescue. If you start prototyping in C, you'll delve into the details of the variables and pointers without seeing much of the actual result in terms of research. MATLAB has an integration with C/C++ and Fortran that allows C code to be called from MATLAB, and vice versa.

Conclusion

This article outlined some of the factors to consider when choosing a programming language. The factors discussed here are not the only ones to be considered, however. For example, if a very experienced programmer suggests a language that was not under consideration, perhaps you should assess that language, too.

We hope that the process of selecting a programming language for your next project will now be easier. There are always more languages evolving, and there's always room for improvement.
Related topics

- The Computer Language Benchmarks Game provides provisional facts about the performance of programs written in approximately 24 different programming languages for a dozen simple tasks.
- Read about the Google App Engine, which lets you run your web applications on Google’s infrastructure.
- One of the first benchmark results, the Computer Language Shootout Scorecard from 2003, summarizes the benchmark results and measures each language's performance.
- "The PHP Scalability Myth" maintains that PHP does scale.
- Learn more about Comprehensive Perl Archive Network (CPAN).
- Stack Overflow: Provides discussions about the speed of PHP, ASP, JSP, CGI, and so on.
- Get more information about MATLAB from MathWorks.
- How to Parse JSON in Java, the code snippet for Listing 3, is from O'Reilly answers.
- How to Parse JSON in Java, the code snippet for Listing 3, is from O'Reilly answers.
- Try out IBM software for free. Download a trial version, log into an online trial, work with a product in a sandbox environment, or access it through the cloud. Choose from over 100 IBM product trials.