# 1-2 Survey of Languages Assignment

Diana Collins

May 16, 2013

## Contents

**1 Part 1: Fundamental Concepts** **2**

**2 Part 2: Language Classification** **10**
- 2.1 Language 1: C++ . . . 10
- 2.2 Language 2: PHP . . . 10
- 2.3 Language 3: JavaScript . . . 11
- 2.4 Language 4: Visual Basic . . . 12
- 2.5 Language 5: C# . . . 13

**3 Peer Feedback and Revision** **14**

**4 Trunitin.com** **14**

**5 Metacognitive Reflection** **15**

**6 Self-assessment** **16**

**7 References** **16**

# 1 Part 1: Fundamental Concepts

1. What aspects does the syntax of a programming language specify?

   The syntax of a programming language specifies what symbols and what sequence of those symbols are considered valid. It also specifies if the language is case sensitive or in-sensitive.

2. How do you specify syntax of a programming language?

   A formal notation is used to specify a programming language's syntax. The most popular notation is Extended Backus-Naur Form. EBNF begins at the highest level entity breaking them down until the single characters are reached.

3. What aspects does the semantics of a programming language specify?

   Semantics distinguish the meaning of the program in a programming language. It distinguishes the nuances of symbols such as the + symbol. Is it being used as a mathematical operation or to concatenate a string? The semantics of the language can make this distinction.

4. How do you specify semantics of a programming language?

   Specifying the change in the state of the program resulting from the execution of a statement gives the semantics of that statement. The state of a program is composed of a pointer to the next instruction and the memory's data.

5. List and briefly explain the various phases of compilation.

   (a) Source code goes into the front end compiler compiler.

   (b) Source code goes through the syntax analyzer which transforms the "sequence of characters in to abstract syntactical entities".

   (c) Code goes through the semantics analyzer which "assigns meaning to the abstract entities".

   (d) An intermediate form of the program exits the front end compiler.

   (e) The intermediate entity goes into the back end compiler.

   (f) The intermediate form is optimized to make it more efficient, this is still machine independent.

   (g) The optimized intermediate form of the program is then generated into code.

(h) The code is the optimized for a specified machine.

(i) Machine code exits the compiler.

6. How do programs written in *compiled languages* get executed?

   A compiler is used to translate the source code into a machine code for a specific computer.

7. How do programs written in *interpreted languages* get executed?

   The code of an interpreted language is compiled into code that will run on a virtual machine rather than being processed into byte code for a specified machine.

8. Some languages use a hybrid approach — both compilation and interpretation — to execute programs. Explain this approach.

   An interpreter also compiles the source code into machine code, but instead of code for a specific computer it creates code for a virtual machine and then runs the code on that. This eliminates the need for a machine specific compiler. Using a hybrid of compilation and interpretation a programer will have the benefits of optimization that compilers provide and the flexibility on non-specific machine code that interpreters provide.

9. Compare and contrast *compiled* and *interpreted* programming languages. Consider type safety, program execution efficiency, programmer productivity, and debugging ease.

   Interpreted languages offer more flexibility and easier debugging because of more precise error messages. These languages also cope better with variable types and sizes that may depend on runtime input. Compilation, on the other hand, are usually more efficient as major decisions such as variable location have been made prior to runtime.

10. Compare and contrast *general-purpose* and *application-specific* programming languages. That is, in what ways are they similar and what features contribute to make them different.

    Application-specific languages are general-purpose languages with a tweak that enables them to have custom features that can be used only on the specific application they are designed for.

11. What is an *imperative* programming language?

Imperative languages depend on state changes to accomplish results. These languages focus on how the computer does the task rather than what the task is. The two most common type of imperative languages used are object-oriented and procedural.

12. What is a *declarative* programming language?

    Declarative languages depend on specifying relationships and functions rather than assignments to variables to accomplish results. They focus on what the computer is doing rather than how it is to be done. Logic-based and functional languages fall into this category.

13. Why do we categorize procedural and object-oriented languages as *imperative*?

    Procedural and object-oriented languages are categorized together because they both use variable assignment to implement the program and solve the given problem.

14. Why do we categorize functional and logic-based languages as *declarative*?

    Functional and logic-based languages are categorized together because they both rely on a predefined set of functions, in the case of functional, and rules, in the case of logic-based, to assess and determine the best output.

15. Some languages support more than one programming paradigm — multi-programming paradigm. Explain this statement using a language such as Scala.

    Scala uses a combination of object-oriented and functional paradigms. This combination makes it possible for programmers to be more productive and it also reduces the amount of code needed to perform the given task.

16. Some programming languages are characterized as *dynamic*. What factors contribute to making a language dynamic?

    Runtime flexibility is a main factor when classifying a language as dynamic. Runtime flexibility means that the program's code can be changed while it is running usually by changing variable type, classes, modules, and functions. Dynamic languages are also usually interpreted rather than compiled.

17. Type feature of programming languages

- ❍ Example of a language which is statically typed, and typing is weak

  C
- ❍ Example of a language which is statically typed, and typing is strong

  Java
- ❍ Example of a language which is dynamically typed, and typing is weak

  Perl
- ❍ Example of a language which is dynamically typed, and typing is strong

  JavaScript

18. Binding times of attributes in Java language:

| **Binding time** | **Attribute name** |
|---|---|
| Language definition time | types are defined |
| Language implementation time | bit representation of types |
| Program translation/compile time | semantics of operations and types |
| Link edit time | types to variables |
| Load time | assigning memory address |
| Program execution/run-time | values to variables |

19. Binding times can be classified into two types: *static* and *dynamic*. Explain static and dynamic binding with concrete examples using Java syntax.

```
//Vehicle superclass
public class Vehicle
{
public String type = "vehicle";

public String start()
{
String start = "The " + type + " has been started";
return start;
}
```

```
}

//Car subclass extends Vehicle
public class Car extends Vehicle
{
public String type = "vehicle";

public Car (String type)
{
this.type = type;
}

public String start()
{
String start = "The " + type + " has been started";
return start;
}
}
```

Static binding occurs at compile time. Dynamic binding occurs at runtime. Referring to the above code consider the following main method.

```
public static void main(String[] args)
{
Vehicle myCar = new Car("car");
//prints "The vehicle has been started"
System.out.println("The " + myCar.type + " has been started");
//prints "The car has been started"
System.out.println(myCar.start());
}
```

The compiler knows myCar is of type Vehicle, but it doesn't know it is an instance of Car until assignment occurs during runtime. So when System.out.println(myCar.start()); is executed the assignment has teken place and the start() method in Car is run. Because type is statically assigned the value of "vehicle" this is done by the compiler and not at runtime. So when System.out.println("The " + myCar.type + " has been started"); is executed the instance variable type has already been assigned the String "vehicle".

20. *Scope rules* determine in which regions of the program a variable is visible. The scope of the variable are these regions.

    - Explain *static scoping* (aka *lexical scoping*) with a concrete example using Java syntax.

      If a variable is statically scoped it is accessible to the class not just an instance of the class. If the value of the variable is changed in any instance of the class it is changed in all instances. Instance variables, on the other hand, are specific to the instance of the class and if their value is changed it is only changed for that instance not all instances.

      For example, if you have a patient class and would like to assign each patient an id that is unique beginning with 1. You could create a static variable to keep track of the number of patients and find each new and unique id as new patients are created.

      ```
      public class Patient
      {
      //instance variable accessible only from the instance
      private int id = 0;
      //class variable accessible from any instance
      protected static int numOfPatients = 0;
      ...
      }
      ```

    - How is *static scoping* implemented?

      In Java static scoping is implemented using the keyword scope and they are referenced using the the class name.

    - Explain *dynamic scoping* with a concrete example using Java syntax.

      Your answer goes here.

    - How is *dynamic scoping* implemented?

      Your answer goes here.

21. *Heap memory* and *stack memory* are the two kinds of memory associated with program execution. Explain their purpose and also discuss their similarities and differences.

    The stack memory is used to keep track of the program flow. It follows the thread and ensures that each function is executed in order. It also

used for accessing temporary variables for evaluating expressions.
The heap is used to store program variables, objects, and such. It is dynamically allocated and deallocated.
Both are used during runtime, but there is often only one heap for the application while each thread gets its own stack.
The stack is only accessible in LIFO order while the heap can be access at random.
The stack is automatically freed, but heap memory must be manually freed by the programmer unless the language has a garbage collector.

22. How is *procedure-call-return* typically implemented in programming languages? Be specific and complete.

    At runtime a stack is allocated to keep track of the thread or flow of program procedures. As a procedure is called a reference to it is placed in the stack. When the procedure is complete, a return statement is reached the stack is popped so the program can follow the thread back to the previous procedure and complete it. Once that is is completed the stack is popped again and the thread is again followed back to the previous procedure.

23. What does it mean to say that a language is *Turing complete*?

    A programming language is said to be Turning complete if it has integer variables and arithmetic and if it can execute assignment, selection, and loop statements sequentially. It must be able to complete any computation that is performed by a Turning machine.

24. Programming languages can be evaluated using the following factors.

    - How is readability measured?

      Readability refers how easily a human can read and understand the code. In other words, how close to human language the programming language is. There are a number of factors which contribute to the readability of a programming language. Good syntax, the amount of symbols, and white space all contribute to the readability of a language.

    - How is expressivity measured?

      The expressivity of a language is the ability to specify computations. The operations available and the convenience of specifying computations determine the language's expressivity.

- How is simplicity measured?

  Simplicity is related to the availability of primitive constructs and the rules associated with them. The number of constructs and the consistency of the rules determines simplicity.
- How is consistent and common syntax measured?

  The syntax of a programming language is considered consistent if it is consistent with notations commonly used in computer science, math, and science.
- How is support for abstractions measured?

  Abstraction is the ability to write code in a more generic way thus increasing the flexibility and reuse. Abstraction is typically easier with dynamically typed languages.
- How is orthogonality measured?

  Orthogonality means that changing one aspect or property of the program doesn't affect other properties of the program. That is when executing one instruction only that instruction executes. This concepts is very important when considering the ability to debug a program.
- How is reliability measured?

  The reliability of a programming language is reflected in its ability to intercept runtime errors and if it functions in the way it is advertised. The ability for the programmer to predict the results of the software she engineers.
- How is efficiency measured?

  To determine the efficiency of a program several factors must be considered. The compilation time, the ability to be optimized, runtime, and hard ware usage.
- How is portability measured?

  Portability is measured by the ability to run the code on different platforms without modifications.
- How is cost measured?

  To measure cost several other factors must be taken into account. The following are some of those factors.
  availability of programmers who know the language
  portability
  usability
  efficiency
  reliability

# 2 Part 2: Language Classification

## 2.1 Language 1: C++

1. Who is the designer/inventor of the language?

   Bjarne Stroustrup developed the language.

2. Is the language open-source or proprietary?

   C++ is open source.

3. What is the name of the most recent language specification (aka language standards) document?

   The language specification document is the C++11 Standard Library.

4. Principal programming paradigm. If the language is multi-paradigm, specify all the paradigms.

   C++ is a multi-paradigm language. The paradigms include declarative/functional, imperative/procedural, and object-oriented.

5. How is the language implemented? compiled, interpreted, or hybrid.

   This language is compiled.

6. What is the name of a popular programming development environment for this language?

   NetBeans, Visual Studio, and Eclipse are all common IDEs used by C++ programmers.

7. Is the language dynamic or not?

   C++ is a statically typed language.

8. Is the language new or old? If the language is introduced before 1990, treat is as a old language. Otherwise, it is a new language.

   Stroustrup began development of C++ is 1979 so it is an "old" language.

## 2.2 Language 2: PHP

1. Who is the designer/inventor of the language?

   Rasmus Lerdorf created this language in 1995.

2. Is the language open-source or proprietary?

   PHP is open source.

3. What is the name of the most recent language specification (aka language standards) document?

   The document is the Standard PHP Library (SPL).

4. Principal programming paradigm. If the language is multi-paradigm, specify all the paradigms.

   PHP's paradigms include imperative/object-oriented and procedural.

5. How is the language implemented? compiled, interpreted, or hybrid.

   PHP is both interpreted and compiled. Originally developed as an interpreted language, some compilers have been created and when used can increase the execution speed.

6. What is the name of a popular programming development environment for this language?

   PHPEclispe, NetBeans, and Geany are all popular IDEs for PHP.

7. Is the language dynamic or not?

   PHP is a dynamic language.

8. Is the language new or old? If the language is introduced before 1990, treat is as a old language. Otherwise, it is a new language.

   This language is classified as new because it was developed in 1995.

## 2.3 Language 3: JavaScript

1. Who is the designer/inventor of the language?

   JavaScript was developed by Netscape and its beta version was released in September of 1995.

2. Is the language open-source or proprietary?

   JavaScript is open source.

3. What is the name of the most recent language specification (aka language standards) document?

   ECMAScript is the current specification document.

4. Principal programming paradigm. If the language is multi-paradigm, specify all the paradigms.

   JavaScript is a multi-paradigm language. Its paradigms include imperative/object-oriented and declarative/functional.

5. How is the language implemented? compiled, interpreted, or hybrid.

   This language is interpreted.

6. What is the name of a popular programming development environment for this language?

   NetBeans is a popular JavaScript IDE.

7. Is the language dynamic or not?

   JavaScript is dynamic.

8. Is the language new or old? If the language is introduced before 1990, treat is as a old language. Otherwise, it is a new language.

   JavaScript is classified as a new language because it was released in 1995.

## 2.4 Language 4: Visual Basic

1. Who is the designer/inventor of the language?

   Visual Basic was developed by Microsoft.

2. Is the language open-source or proprietary?

   It is a proprietary programming language.

3. What is the name of the most recent language specification (aka language standards) document?

   Visual Basic 10.0 Language Specification is the language specification document.

4. Principal programming paradigm. If the language is multi-paradigm, specify all the paradigms.

   The Visual Basic paradigms are object-oriented and event-driven.

5. How is the language implemented? compiled, interpreted, or hybrid.

   This language is compiled.

6. What is the name of a popular programming development environment for this language?

   Visual Studio is the IDE used when developing in Visual Basic.

7. Is the language dynamic or not?

   This language is statically typed.

8. Is the language new or old? If the language is introduced before 1990, treat is as a old language. Otherwise, it is a new language.

   Visual Basic was release in 1991 so it is classified as a new language.

## 2.5 Language 5: C#

1. Who is the designer/inventor of the language?

   C# was developed by Microsoft. The development team was headed by Anders Hejlsberg.

2. Is the language open-source or proprietary?

   The development of this language is directed by Microsoft so it is considered proprietary.

3. What is the name of the most recent language specification (aka language standards) document?

   C# Language Specification is the language specification document.

4. Principal programming paradigm. If the language is multi-paradigm, specify all the paradigms.

   C# is multi-paradigm. These include imperative/object-oriented, event-driven, and declarative/functional.

5. How is the language implemented? compiled, interpreted, or hybrid.

   This language is compiled.

6. What is the name of a popular programming development environment for this language?

   C# is often developed using Visual Studio.

7. Is the language dynamic or not?

   C# uses a keyword, var, which is thought to make it dynamic, but when compiled the variables are statically typed.

8. Is the language new or old? If the language is introduced before 1990, treat is as a old language. Otherwise, it is a new language.

   Development on the C# language began in 1999, classifying this language as new.

## 3 Peer Feedback and Revision

Name of the person who provided feedback: Timothy Morgan.

Following are the specific suggestions made by the peer:

1. There were typographical errors in questions 5, 6, and one part of 24.
2. There were some gramatical errors in questions 8, 9, and one part of 24.
3. Suggested revision of question 7 because answer wasn't completely accurate.
4. Two parts of question 20 were not addressed.

In the following, we discuss how the peer's suggestions for improvement are incorporated into this document.

1. I corrected the typographical errors in questions 5, 6, and 24. I also reviewed the paper again for typos.
2. I corrected the gramatical errors in questions 8, 9, and 24, as well as, reviewed the paper for other errors.
3. I revised the answer to number 7 to more accurately describe how interpreted languages are executed.
4. Upon further reading and discussion with others I could not fully grasp the concept of dynamic typing with regard to how to accomplish it in Java.

## 4 Trunitin.com

❍ What percent of your document is reported as similar to other documents?

Your answer goes here.

❍ How do you defend if more than 15% of your document is similar to other documents?

Your answer goes here.

# 5 Metacognitive Reflection

1. Did I solve the right problem?

   Yes, I solved the right problem.

2. Did I solve the problem right?

   Yes and no. I did not completely solve the problem because of personal limitations.

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

   I identified what was being asked of me. First, attempted to gain understanding from the materials provided in the course dropbox folder. Then, if further research was necessary, I would conduct google searches to locate the appropriate information that would help me to understand how to answer the questions.

4. What strategies and techniques did I draw upon?

   I refined my ability to use online search engines in an efficient way. I also became more adept at searching and scanning through electronic texts.

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 gain a lot of understanding with this assignment. There were concepts I had never heard of before. The way that programs are run, compiled or interpreted, as well as, stack vs. heap memory was unknown to me before this paper. I also gained an appreciation for the planning that goes into developing a programming language. This was something I had taken for granted prior to this assignment.

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

   While this paper was very time consuming and frustrating at times, I learned a great deal and gained a new appreciation for programming language, what goes into developing them, and their interaction with the machine.

## 6 Self-assessment

| Rubric line item | Max possible points | Earned points |
|---|---|---|
| Answers to section 1 questions are correct | 48 | 46 |
| Five languages are correctly classified (section 2) | 20 | 20 |
| References are formatted using the APA style | 3 | 1 |
| Peer feedback solicited and implemented | 8 | 8 |
| Turnitin.com submission is completed | 4 | 4 |
| Metacognitive reflection is completed | 6 | 6 |
| Self-assessment is performed | 5 | 5 |
| Paper is of professional quality | 6 | 4 |
| Total points | 100 | 94 |

## 7 References


Ben-Ari, M. (2006). Understanding Programming Languages. Chichester, England: John Wiley & Sons.

Hanamasagar, S., Miladinova, M., Naim, R., Nizam, M. F., Noureddine, J. Comparative Studies of 10 Programming Languages within 10 Diverse Criteria

Lee. Specifying Syntax.

Paulson, L. D. (2007). Developers Shift to Dynamic Programming Languages.

Scott, M. Intro to Programming Languages.