Programming In General

Author:
Brett Langdon

June 16, 2012

Contents

Chapter 1

Introduction

Programming In General is meant to be used as a resource to learn the concepts of computer programming that can be applied to any language or platform.

This resource is and always will be provided for free.

This resource is currently a work in progress so please bear with me if any particular sections or chapters are in complete or if sections contain less than correct information. If you have any comments, questions, suggestions or corrections please feel free to contact me by e-mail at: *brett@blangdon.com*

Thank you and enjoy.

1.1 Who Is This Resource For

This resource is intended for everyone.

This resource is meant to be useful to programmers of all levels, those who have never programmed before, those who are just getting started and even those who have been programming for years. Since this resource will always be growing and changing as the industry changes the information should always be in some way applicable for all developers.

Those who are familiar with programming are most likely going to be able to skip the chapter *Getting Started* and in some cases might not read the resource in sequential order, but rather skip around picking and choosing which sections are applicable to them.

1.2 Code Examples

All code examples in this resource use a sudo language that is not meant to be run or compiled directly. I have chosen to use this approach so that the concepts can be extracted and implemented in any language on any platform. By focusing on the concepts at hand rather than specific implementations I can focus on trying to present the material in a clear and easy to understand manner.

I will cover how to use the sudo language and how to translate it to a useable programming language in the chapter *Getting Started* in the section titled *Sudo Language*.

Chapter 2

Getting Started

This chapter will cover how to get started with programming, how to choose which language or platform to start with and how to go about using this resource.

2.1 Choosing A Programming Language

It is wonderful that you have decided to undertake the hobby of computer programming, but which language should you choose: Python, PHP, Java, C#, C/C++, VB, Ruby, Scala, Groovy, Javascript, or one of the thousands of others languages available to programmers. There are many factors to consider when choosing a programming language especially when getting into programming for the first time, some of which are the languages paradigm, syntax, platform and even the coolness factor of the language.

2.1.1 Paradigm

A languages paradigm refers to the languages overall style of development. For example the three mainly adobted paradigms are Functional, Object Oriented and Multi Paradigm. Functional refers to languages that are based around completing tasks using Mathematical functionals; C is an example of a functional language because rather than using classes or objects to complete it's tasks it used constructed functions. Object Oriented languages on the other hand are constrcuted by designing classes and objects to complete your programming tasks; Java is an example of an Object Oriented programming language because regardless of the type of program you develop you must use classes and objects. Multi Paradigm languages are usually a mix of more than one paradigm. For example Python is a Multi Paradigm language because you can choose whether or not to use classes and objects when programming.

There are many more types of paradigms that languages can follow but most languages you will come across today are either strictly functional, strickly object oriented or they offer the best of both worlds by supporting both.

2.1.2 Syntax

A languages syntax is very important when choosing a language. This is mainly going to be a personal preference. Personally I like C style sytanx languages like C,C++,Java,PHP,Javascript,etc. Other people might prefer other languages because their use of other syntax styles, like the almost sudo code style of Python. Your personal preference will come with time as you move from one language to another and develop your own personal styles and preferences.

2.1.3 Platform

This is a very important factor when choosing which programming language to use. What platforms do you have available to use? Do you only have a Windows computer at your disposal? That might remove some of the options out there as some languages might not support developing on a Windows machine.

When starting out try and choose a language that works on a platform that is readily available to you. Do not try and move to a new or different operating system in order to learn programming. Keep things simple.

2.1.4 Coolness

What seems cool to you? What is everyone else raving about right now? What is new and different?

Some may think that this is a silly factor to introduce when trying to pick a programming language to use, but I can honestly say that it has effected my choices in the past. When I was learning programming in college we were being taught Java, but I picked up and learned PHP myself on the side mainly because my friend was using it and I wanted to impress him. This is not a bad thing. Let others help influence your decisions when programming, that is how you will grow and learn things you might not of experienced without the influence.

2.1.5 Conclusion

So, we have taken a quick look at how to go about picking a programming language. Some of you might say, "that was not really helpful, you did not tell me which language to use", and your right I didn't, it should not be my choice which language you learn first. I want to try and keep some bias out so that this resource is as lanuage agnostic as possible.

Advice:
If after doing some research you are still unsure which lanauge you want to use, especially for going through this resource try out Python. Python is available for every platform, or at least all of the ones I can think of, it is interpreted (you don't have to compile everytime you want to run your code) and lastly it's syntax is going to be one of the closest to the sudo langauge that this resource uses.

2.2 How To Read This Resource

This resource is going to be laid out a little weird, more so for those who have already had some programming background.

For those who are new to programming I strongly suggest reading through Chapters 3 and 4 thoroughly before continuing with the rest of the resource. Those two chapters contain all of the core concepts needed in order to understand some of the higher level concepts presented with Data Structures and Algorithms. Once you have completed chapters 3 and 4 please feel free to jump around a little between sections presented in chapters 5 and 6 as some data structures or algorithms might interest you more than others.

2.2.1 Keywords

Throughout this resource some words will be highlighted, colored differently or emphasized in order to stand out. These words will generally be referring to code examples presented in the chapters:

Type	Example
Variable	***variableName***
Functions	***functionName***
Class Properties	***propertyName***
Values	**"Sample String Value"**
Program Output	**Console String Output**

Example:
We assign the value of **"Sample String"** to the variable ***sample*** then pass in ***sample*** as a parameter to the function ***printValue*** which will print: **The String Is: Sample String**.

2.3 Sudo Language

For the code examples presented in this resource I am going to be using a sudo language. The concept behind a sudo language is to be able to present programming concepts in a language agnostic form so that the concepts can be translated to your language of choice.

So it is great that you have chosen lanauge X to use throughout this resource, but how is the sudo lanauge going to help you out? Well, lets look at two examples and I will show their implementation in a few different languages. Hopefully this will help you be able to understand how the language should be tanslates (especially if your language of choice is one that I use).

2.3.1 Example 1

Listing 2.1: Example 1 - Sudo Code

```
name = "Brett"
if name == "Brett"
    print "Name Is Brett"
else
    print "Name Is Not Brett"
```

For this example lets break it down line by line to make sure we know exactly what is going on.

1. Store the value **"Brett"** into the variable ***name***
2. Check if the variable ***name*** is equal to the value **"Brett"**
    3. Print **"Name Is Brett"** to the console
4. Otherwise
    5. Print **"Name Is Not Brett"** to the console

As far as programming goes this is a fairly simple process but lets try and translate this example to a few different languages to see how it is done.

Listing 2.2: Example 1 - PHP

```
<?php
$name = 'Brett';
if( $name === 'Brett' ){
    echo 'Name Is Brett';
} else{
    echo 'Name Is Not Brett';
}
```

Listing 2.3: Example 1 - C

```
int main{
    char* name = "Brett";
    if( name == "Brett" ){
        printf("Name Is Brett");
    } else{
        printf("Name Is Not Brett");
    }
    return 0;
}
```

Listing 2.4: Example 1 - Python

```
name = "Brett"
if name is "Brett":
    print "Name Is Brett"
else:
    print "Name Is Not Brett"
```

Listing 2.5: Example 1 - Node.JS

```
var name = "Brett";
if( name == "Brett" ){
    console.log("Name Is Brett");
} else{
    console.log("Name Is Not Brett");
}
```

Listing 2.6: Example 1 - Java

```
class Example1{
    public static void main( String[] args ){
        String name = "Brett";
        if( name.equals("Brett") ){
            System.out.println("Name Is Brett");
        } else{
            System.out.println("Name Is Not Brett");
        }
    }
}
```

Notice that all of the actual examples end up looking the same? That is the point of using the sudo language, so that we can discuss the core concepts of the lesson at hand and then those concepts can be directly applied to any language of choice.

Also, notice the Python implementation, it is almost line for line, word for word identical to the sudo language example.

2.3.2 Example 2

Since we have seen a fairly simple example above, lets take a look at a more complicated example. Do not be afraid if it does not make too much sense right now, but try and notice the similarities between the sudo language and the actual code examples.

Listing 2.7: Example 2 - Sudo Code

```
class Person
    private name

    function getName()
        return this.name

    function setName( newName )
        this.name = newName


p = new Person()
p.setName("Brett")
```

```
13
14
15 if p.getName() == "Brett"
16     print "Name Is Brett"
17 else
18     print "Name Is Not Brett"
```

Just like the last one, lets break down this example line by line to fine out whats going on.

1. Create a new class called ***Person***
   2. Create a private property ***name***
   4. Create a method called ***getName*** that requires no parameters
      5. When the function is called return the class property ***name***
   7. Create a method called ***setName*** that takes a single parameter ***newName***
      8. When called set the class property ***name*** equal to the parameter ***newName***
11. Create a new ***Person*** object and store it in the variable ***p***
12. Call ***p***'s ***setName*** method passing in the value **"Brett"**
14. Call ***p***'s ***getName*** method and check if the returned value is equal to **"Brett"**
    15. Print **"Name Is Brett"** to the console
16. Otherwsie
    17. Print **"Name Is Not Brett"** to the console

Do not worry if this example does not make sense to you, you will be able to understand it well before the end of this resource.

Just like with Example 1, here are some translations of the example.

Listing 2.8: Example 2 - PHP

```
1  <?php
2  class Person{
3      private $name;
4
5      public function getName(){
6          return $this->name;
7      }
8
9      public function setname( $newName ){
10         $this->name = $newName;
11     }
12 }
13
14 $p = new Person();
15 $p->setName('Brett');
16
17 if( $p->getName() === 'Brett' ){
18     echo 'Name Is Brett';
19 } else{
20     echo 'Name Is Not Brett';
21 }
```

Listing 2.9: Eample 2 - Java

```java
class Person{
    private String name;

    public String getName(){
        return this.name;
    }

    public void setName( String newName ){
        this.name = newName;
    }

    public static void main(String[] args){
        Person p = new Person();
        p.setName("Brett");

        if( p.getName() == "Brett" ){
            System.out.println("Name Is Brett");
        } else{
            System.out.println("Name Is Not Brett");
        }
    }

}
```

Listing 2.10: Example 2 - Node.JS

```javascript
var Person = function(){}
Person.prototype.getName = function(){
    return this.name;
}
Person.prototype.setName = function( newName ){
    this.name = newName;
}

var p = new Person();
p.setName("Brett");

if( p.getName() == "Brett" ){
    console.log("Name Is Brett");
} else{
    console.log("Name Is Not Brett");
}
```

Listing 2.11: Example 2 - Python

```python
class Person:
    def getname( self ):
        return self.name
    def setName( self, newName ):
        self.name = newName

p = Person()
p.setName("Brett");

if p.getName() is "Brett":
    print "Name Is Brett"
```

```
12 else:
13     print "Name Is Not Brett"
```

This example does a better job of showing how each language can tackle the concepts in a different manner but the core concepts laid out by the sudo language can still be extrapolated and translated to each individual programming language. As long as the language supports the concepts. As you may notice that I left out the implementation of C in this example. It is because C does not support the use of classes and objects, yes there are ways of completing this example in C using structs but that is something that you should learn on your own.

So now you have seen a few examples, hopefully enough to give you an idea of how the examples in this resource will be presented.

Chapter 3

Functional Programming

In this chapter we are going to cover the basic concepts of functional programming. This could mean a few things to different people, but in regard to this resource we are going to refer to functional programming as programming without the use of classes and objects. Yes, some people are cringing a little in their seats as that is not the best definition of functional programming but to try and keep things simple and organized that is what we are going to refer to it as.

I am going to use this chapter to introduce topics other than just functions. Topics including control statements, loops and some input output (io).

Functional Programming:
Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids state and mutable data.
Wikipedia (2012)

3.1 Variables

Variables act as aliases to the values that we want them to represent and they allow us to access and manipulate the values that we assign to them. For example we could use the variable ***name*** to represent the value **"Brett Langdon"**. We do this with programming so that we can then access the value **"Brett Langdon"** with a shorter representation, ***name***.

3.1.1 Declaration

To start with variables we need to declare their existence. By declaring a variable we are saying to the program, here is our alias and here is the value that we want it to represent.

Listing 3.1: Variable Declaration

```
1 a = 10
2 print a
```

In the above example we are saying that we want to store the integer value **10** into the variable ***a***. We can then use the variable ***a*** to access the value **10**. This program will output **10** rather than **a**.

When we declare variables we are telling the programming language to allocate some space in your computers memory in order to store the value that you need it to. The amount of space needed to store each variable depends based on your specific language being used and which data type is being used to store the value.

3.1.2 Data Types

Programming languages support different types of data types or different types of values that they can represent in variables. Some programming languages use multiple different types of values but most of them support the basic types: string, integer (multiple kinds) and boolean (true or false).

Listing 3.2: Data Types

```
1 string = "Brett"
2 integer = 10
3 boolean = false
```

Please keep in mind that each programming language supports different data types and you should research those types to better understand variables in that language. As well some programming languages that are strickly typed which requires us to define the data type of the variable on declaration (unlike our sudo language).

3.1.3 Operations

3.1.4 Conclusion

3.2 Control Statements

3.2.1 If Statements

3.2.2 If-Else Statements

3.2.3 For Loops

3.2.4 While Loops

3.2.5 Do-While Loops

3.2.6 Switch Stataments

3.2.7 Break Statments

3.2.8 Continue Statements

3.2.9 Conclusion

3.3 Functions

3.3.1 Declaration

3.3.2 Returns

3.3.3 Parameters

3.3.4 Recursion

Chapter 4

Object Oriented Programming

4.1 Classes and Objects

Classes *AND* Objects? What is the difference?

Well I am glad you asked. A class is the definition or blueprint of an object. A class tells a program what to expect when coming across an object of the given class. What methods and properties to expect and even how to create and destroy objects.

An object refers to a single instance of a class
Objects are refered to as being instances of a class. When you deine a class you are not creating a usable object that you can then call methods on or access properties of. You must then create an instance of that class (object) to be able to use it throughout your program.

4.1.1 Classes

Ok, so as I mentioned before we need to first define a class before we can start creating objects and using them in our program. How do we do this?

Listing 4.1: Class Definition

```
1 class Person
```

Ok...? That seems too easy?
Yes creating classes is usually fairly easy, just make sure to check how to create a class in your language of choice.

4.1.2 Objects

Ok, so we have our class definition from above, but how do we create an instance of this class so we can use it in our program?

Listing 4.2: Object Declaration

```
1 class Person
2
3 p = new Person()
```

That is it. We can create an instance of our ***Person*** class by using the ***new*** keyword and calling ***Person()***. We can assign this instance to a variable, ***p***, and then use ***p*** as an alias for our object throughout our program.

Can we only have one object? No, you can have as many instances as your would like.

Listing 4.3: Multiple Object Instances

```
1 class Person
2
3 p1 = new Person()
4 p2 = new Person()
5 p3 = new Person()
```

This then allows us to act on each of these instances as though they are separate. What does that mean? It means that if we were to modify a property of ***p1*** then it would not have any effect on the same properties in ***p2*** and ***p3***.

4.1.3 Properties

We are able to store variables inside of a class, these are called properties. To define a property we must define its name, access modifier and default value (if any).

An access modifier can either be *public*, *private* or *protected* (some languages do not support access modifiers). The *public* modifier means that anyone who has access to the object can read and modify that property. The *private* modifier means that no one outside of the object can read and modify the property, meaning that only the object itself has acess to the given property. The *protected* modifier means that the given object and its children (we will get to this later in the chapter) will have access to read and modify the property. Lets look at an example.

Listing 4.4: Class Properties

```
1 class Person
2       public name
3       private age = 22
4
5 p = new Person()
6 p.name = ''Brett Langdon''
7
8 p.age = 23 //this will cause an error
```

In this example we are creating a class with two properties, one is public (***name***) and the other is private (***age***). We then create a new instance of our class assigning it to the variable ***p***. Then we set the public property ***name*** to **"Brett Langdon"**. In line 8 there is the comment "this will cause an error" this is because the property ***age*** is private and cannot be accessed from outside of the class.

4.1.4 Methods

So what is a Method? A method, simply put, is a function that belongs to a class. We use methods for the same reasons that we use functions for, to provide code reuse within our applications. Ok, so we know how to use functions, but how do we use them from within a class?

Listing 4.5: Class Methods

```
1  class Person
2        public name
3        private age
4
5        function printName()
6                print this.name
7
8  p = new Person()
9  p.name = ''brett''
10 p.printName()
```

The output of this code would be **brett**.

4.1.5 Special Methods

4.2 Classes and Objects

" Classes *AND* Objects? What is the difference?

Well I am glad you asked. A class is the definition or blueprint of an object. A class tells a program what to expect when coming across an object of the given class. What methods and properties to expect and even how to create and destroy objects.

An object refers to a single instance of a class
Objects are refered to as being instances of a class. When you deine a class you are not creating a usable object that you can then call methods on or access properties of. You must then create an instance of that class (object) to be able to use it throughout your program.

4.2.1 Classes

Ok, so as I mentioned before we need to first define a class before we can start creating objects and using them in our program. How do we do this?

Listing 4.6: Class Definition

```
1 class Person
```

Ok...? That seems too easy?
Yes creating classes is usually fairly easy, just make sure to check how to create a class in your language of choice.

4.2.2 Objects

Ok, so we have our class definition from above, but how do we create an instance of this class so we can use it in our program?

Listing 4.7: Object Declaration

```
1 class Person
2
3 p = new Person()
```

That is it. We can create an instance of our ***Person*** class by using the ***new*** keyword and calling ***Person()***. We can assign this instance to a variable, ***p***, and then use ***p*** as an alias for our object throughout our program.

Can we only have one object? No, you can have as many instances as your would like.

Listing 4.8: Multiple Object Instances

```
1 class Person
2
3 p1 = new Person()
4 p2 = new Person()
5 p3 = new Person()
```

This then allows us to act on each of these instances as though they are separate. What does that mean? It means that if we were to modify a property of ***p1*** then it would not have any effect on the same properties in ***p2*** and ***p3***.

4.2.3 Properties

We are able to store variables inside of a class, these are called properties. To define a property we must define its name, access modifier and default value (if any).

An access modifier can either be *public*, *private* or *protected* (some languages do not support access modifiers). The *public* modifier means that anyone who has access to the object can read and modify that property. The *private* modifier means that no one outside of the object can read and modify the property, meaning that only the object itself has acess to the given property. The *protected* modifier means that the given object and its children (we will get to this later in the chapter) will have access to read and modify the property. Lets look at an example.

Listing 4.9: Class Properties

```
1 class Person
2       public name
3       private age = 22
4
5 p = new Person()
6 p.name = ''Brett Langdon''
7
8 p.age = 23 //this will cause an error
```

In this example we are creating a class with two properties, one is public (***name***) and the other is private (***age***). We then create a new instance of our class assigning it to the variable ***p***. Then we set the public property ***name*** to **"Brett Langdon"**. In line 8 there is the comment "this will cause an error" this is because the property ***age*** is private and cannot be accessed from outside of the class.

4.2.4 Methods

So what is a Method? A method, simply put, is a function that belongs to a class. We use methods for the same reasons that we use functions for, to provide code reuse within our applications. Ok, so we know how to use functions, but how do we use them from within a class?

Listing 4.10: Class Methods

```
1  class Person
2        public name
3        private age
4
5        function printName()
6                 print this.name
7
8  p = new Person()
9  p.name = ''brett''
10 p.printName()
```

The output of this code would be **brett**.

4.2.5 Special Methods

"

4.3 Inheritence

4.4 Polymorphism

4.5 Design Patterns

Chapter 5

Data Structures

Chapter 6

Algorithms