PYTHON

LEARN PYTHON FAST

The Ultimate Crash Course to Learning the Basics of the Python Programming Language In No Time

Stephen Hoffman

SATISFACTION 100% GUARANTEE

JAVA

THE ULTIMATE GUIDE TO LEARN JAVA PROGRAMMING FAST

Java

Peter Hoffman

Java

The Ultimate Guide to Learn Java and Python Programming (java for beginners, java for dummies, java apps, how to program, python, computer programming)

JAVA

THE ULTIMATE GUIDE TO LEARN JAVA PROGRAMMING FAST

Java™

Peter Hoffman

Java

The Ultimate Guide to Learn Java Programming Fast (Java for Beginners, Java for dummies, how to program, java apps, java programming)

PETER HOFFMAN

CONTENTS
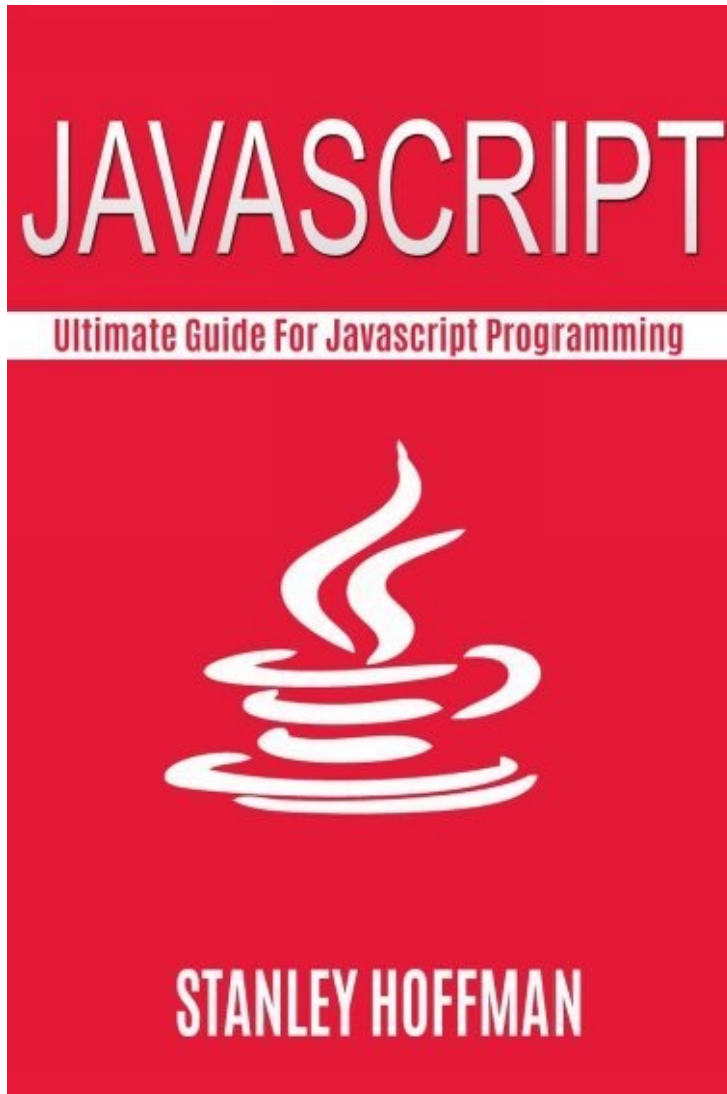
I think next books will also be interesting for you:
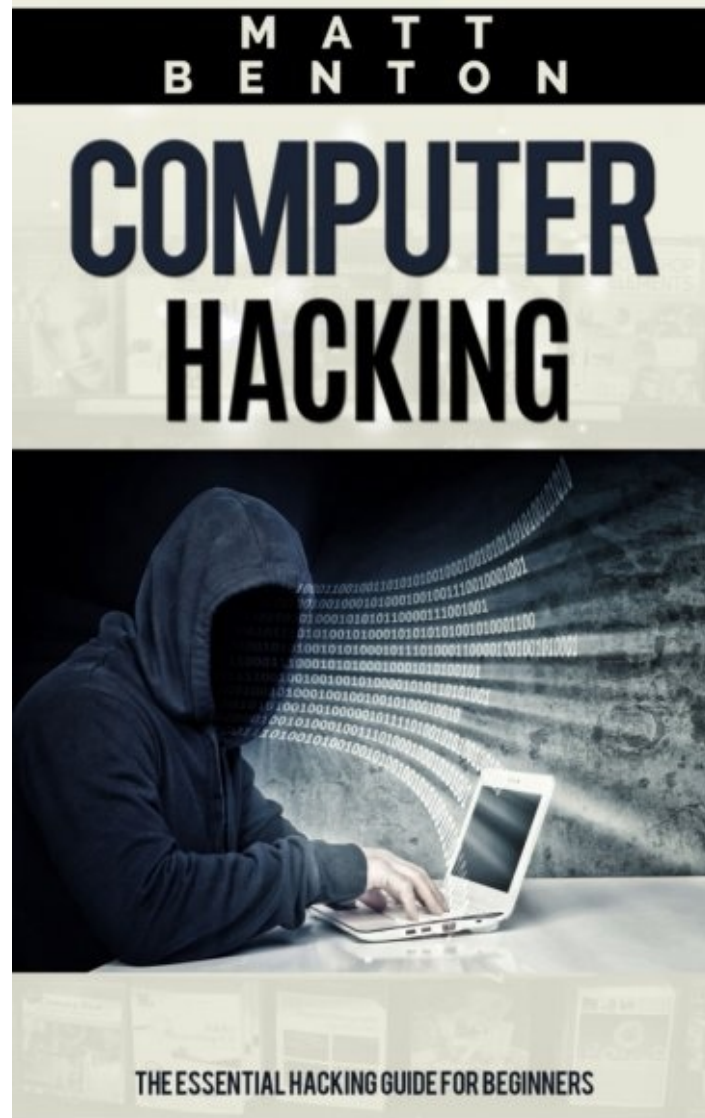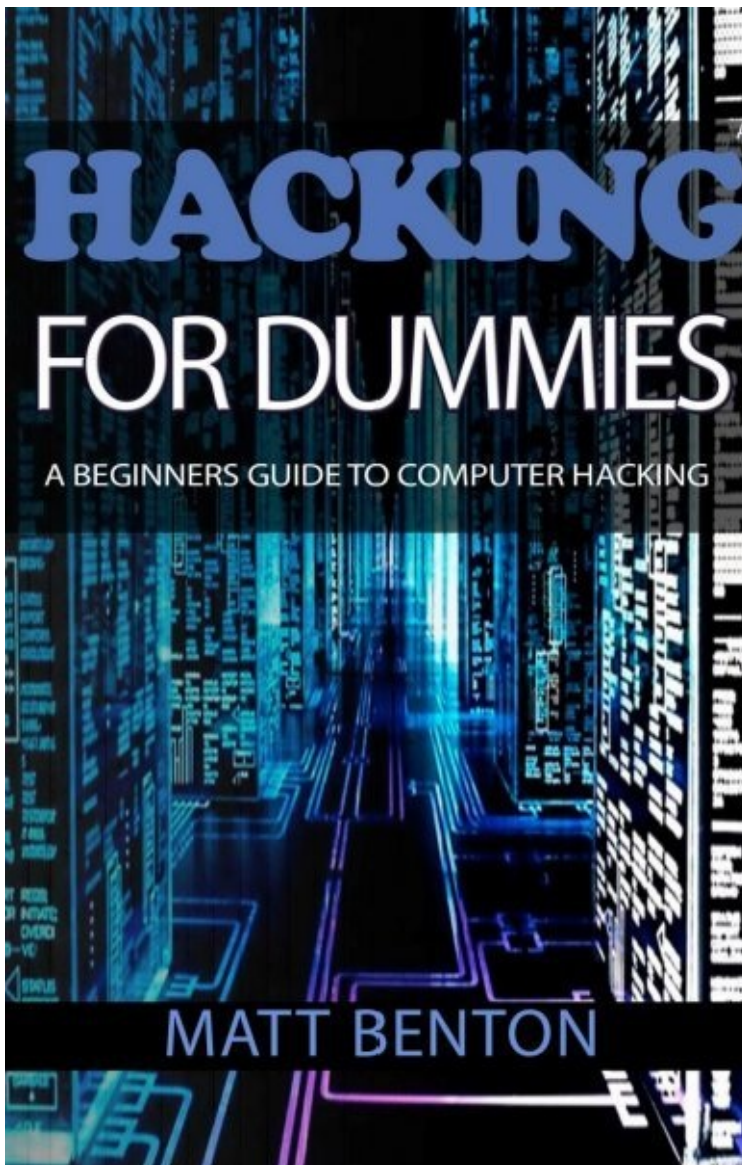
[C++](#)
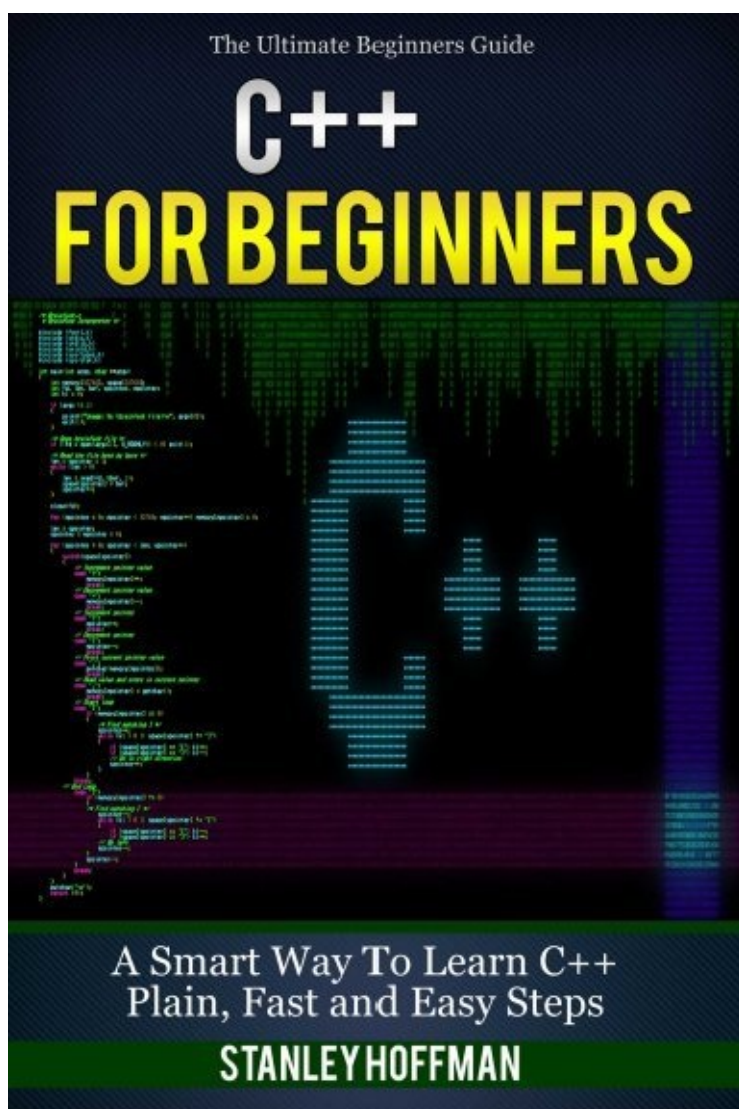
[Javascript](#)

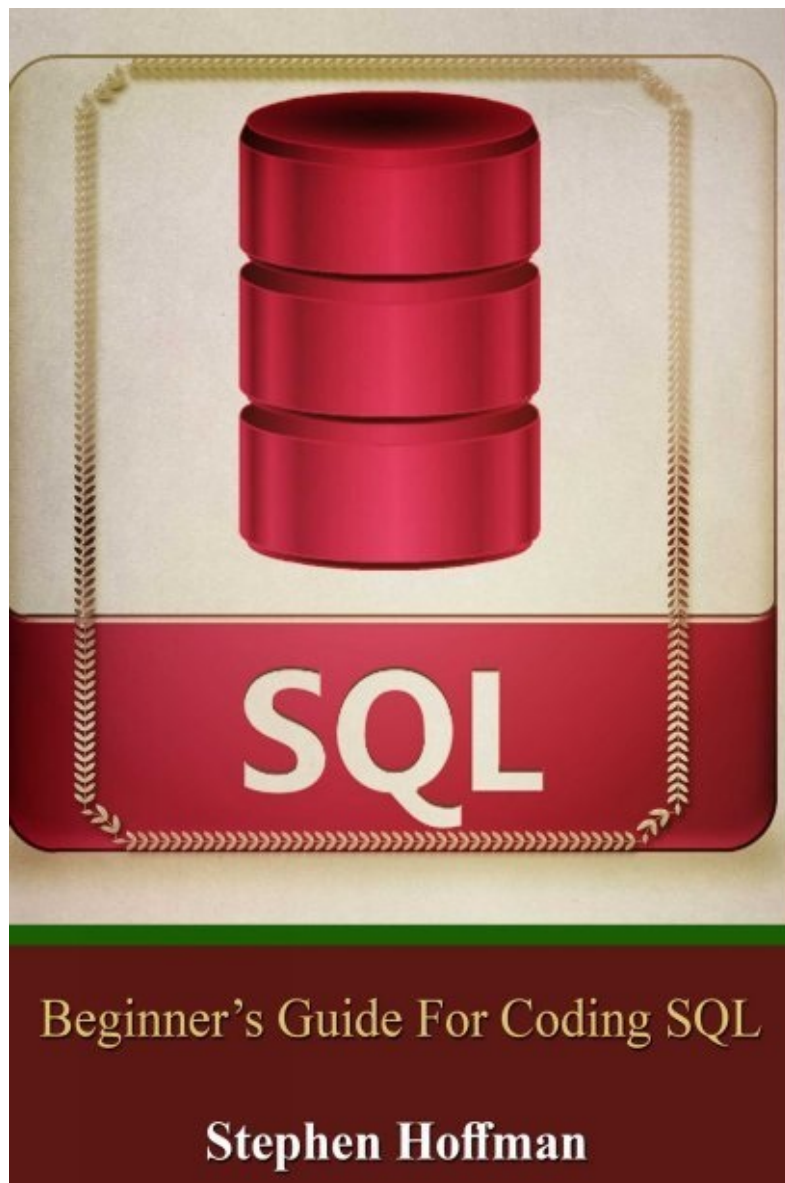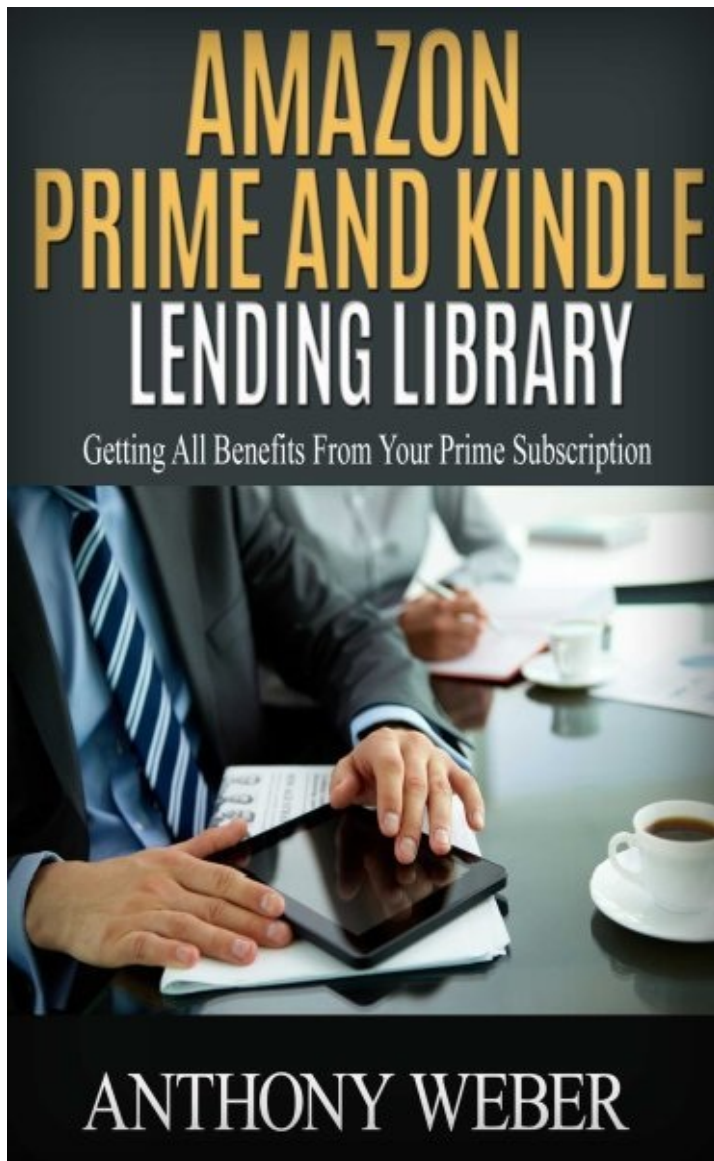[Computer Hacking](#)

Hacking for Dummies

[Amazon Prime and Kindle Lending Library](#)

[Amazon Prime Kindle Lending Library](#)

Introduction

The company behind the Java programming language is the Sun Microsystems. The language was initially introduced in the year 1991. It was developed by a team of people consisting Mike Sheridan, Ed Frank, Chris Warth, Patrick Naughton, and James Gosling. Due to its object-oriented nature, the Java language can be reused with regards to data focus, classes creation, written instructions, and the creation of both instances and objects.

The Four Java Platforms

There are four primary platforms in the Java programming language. They are as follows:

- Java Standard Edition (Java SE)
- Java Enterprise Edition (Java EE)
- Java Micro Edition (Java ME)
- JavaFX

Java-based applications that were being used for computer software were developed on the standard edition of Java. Java-based applications for web servers were developed on the enterprise edition while Java-based applications for multimedia platforms were developed on JavaFX. Most JavaFX-based applications are what we usually call flash players. Lastly, Java-based applications for mobile devices were developed on the micro edition Java platform.

Tools for Beginners

In order to begin your journey to the world of Java programming, you need to choose which IDE to use. IDE, also known as Integrated Development Environment, serves as your workspace in writing Java applications. Imagine it as your notepad in writing HTML codes. However, IDEs were specifically designed for the Java language which means it features all the tools you might need in Java development.

Here are the following IDEs that you can use:

- BlueJ – this is an ideal IDE for beginners. You can easily navigate due to its simplistic interface. It was also originally designed to help beginners in familiarizing the primary concepts of the Java language.

- DrJava – this is a lightweight IDE. Similar to BlueJ, DrJava was also designed for beginners. Although it features a simplistic user interface, DrJava has enough tools that every advanced Java developer can use as well.

- Eclipse – one of the popular IDEs today, the Eclipse is an open source platform with responsive navigation and design. Apparently, it has the cleanest user interface compared to other IDEs in this list.

- NetBeans – this is another popular IDE which is commonly used in Java development nowadays. This is highly recommended for beginners due to its fast and powerful tools that are compatible in all platforms of the Java language.

Each IDE has a unique user interface and ease of navigation. Feel free to use any IDE where you feel most comfortable with. Keep in mind that all of them use the same language although they may appear to be different from one another.

Aside from Java IDEs, another tool you need to begin learning the Java programming language is the JDK. JDK, which stands for the Java Development Kit, serves as your computer's interpreter and compiler for the Java language. Without this, your computer will not be able to translate Java codes into the universal machine language. All versions of JDK are available from the official website of Oracle.

Take note that you need to choose the JDK version that is most appropriate to your computer's operating system to avoid future issues when compiling written Java

programs.

Misinterpretations

Before we proceed, I would like to clarify that the Java language is not related to the JavaScript language. These are two different programming languages whereas Java is used for the development of stand-alone applications. On the other hand, the JavaScript language is an integrated language for web development. It is used to assert functions that are not achievable through HTML language alone.

This book is about the Java programming language and not about the JavaScript language.

Chapter 1 – Writing a Java Program

Once you are done from choosing your preferred IDE and from downloading the latest version of JDK, you now have the tools to begin programming using the Java language. However, having the tools does not mean that you are already prepared to jump into writing your very first Java application.

It is also important to understand the structure of a Java program – its primary components and their components in the Java application.

Software Types

You can create two types of software using the Java programming language – applets and applications.

Applets are smaller pieces of programming codes that were designed to run on web browsers. We can consider the applets as smaller and lighter applications. They are mainly used to provide either navigation enhancements or additional interactivity to the browser. In contrast with stand-alone Java applications, applets do not need any interpreter in order to execute.

Another type of software that you can develop through Java is a console application. Console applications are stand-alone programs that run within a console environment. All

IDEs feature an integrated console environment in order for you to test these kinds of programs. As a beginner, it is important for you to learn console operations to have a good grasp on the basics of Java language.

Classes, Instances, and Objects

The classes, instances, and objects are the three vital elements of a program created using an object-oriented language such as the Java. In order to understand these three, you need to first look at their relationship from one another.

Classes are the highest and they cover everything in Java programming language. Specific elements in a class are called Objects. Once you add the additional specification to an Object, then it becomes an Instance.

In an analogy, for instance, let us say our class here is the animal. Under animal class, we have different kinds of animals like dogs, cats, birds, and so on. These are the objects under animal class. Under the category of dogs, we have different breeds like poodles, golden retrievers, Labradors, and so on. These are the instances of the object dogs.

Project Creation

The first step in writing your Java program is to start a new empty project in your chosen IDE. You can do this by navigating through the top most toolbar as follows:

File -> New Project

After selecting New Project, a new window will appear where you will be able to choose from a list of categories. Under the Categories list, select Java and then select Java Application from the adjacent window panel. Afterward, click on the Next button and you will be asked for the project name and the location where you want to save your project.

Always remember the project name since you will be asked to go back to the same project along the way.

Main Class Creation

After the creation of a new project, the IDE will automatically create both the package and the main class. Packages, also known as source packages, serves as folders where you can efficiently organize all assets of the Java project such as classes and instances. Take note, however, that packages are simply identifiers for the organization of assets. Consider them as folders in your computer where you organize your files accordingly.

There should be a main class under the package. If the IDE did not automatically create the main class, then you can create it by navigating on the main menu as follows:

File -> New File -> Java Class

The main class has its vital role in a Java program. The main method, which is what initially executed when opening a Java program, is contained in the main class. In other programming languages, the main method is known as the entry point.

Writing the Package Statement

Now that we already have the main class in our Java project, we are now ready to write our first Java programming language line – the package statement. This statement is an optional part of the program and is only introduced to you for learning purposes. This is written using the following syntax:

package [package name];

The IDE will automatically look for the default package if this line is not written. In other words, the package statement might come in handy if you are working on a Java project with multiple packages. Keep in mind that package name must match with the name of the package where the class you need belongs to. Let us say, for instance, that the name of your package is "javatestproject". The syntax should look like this in your IDE:

```
1
2    project javatestproject;
3
```

Writing the Class Declaration

Next to the package statement is the class declaration. The syntax we need to follow here is this:

[access modifier] class [class name]

There are two kinds of access modifiers that you can use: public and private. The private modifier prevents the other classes from accessing the declared class while the public modifier allows other classes to access the declared class. For the meantime, let us focus and use the public as our access modifier for our first Java application.

Class name, as what the name suggests, is the name you designated to the main class. Take note that the Java programming language is case sensitive so make sure that you name of the main class with correct upper case and lower case characters. Let us say that the name of our main class is "JavaTestProject", then this is what should we have right now:

```
1
2   package javatestproject;
3
4   public class JavaTestProject {
5
6   }
7
```

The open ({) and close (}) curly braces refer to the opening and closing of a Java code block. This helps the program in determining where a certain block of code begins and where it ends.

Writing the Main Method

Now that we are done declaring the class, it is now time to declare the main method for our first program. In our main method, we will be using two keywords: void and static.

The void keyword signals the Java Virtual Machine, also known as JVM, that the program successfully ran and finished. The keyword static signals the program that the field refers to the class and neither the objects nor the instances in it. In other words, the program will be able to freely go through the class without creating instances in that class. Due to the nature of the main method to act as the program's entry point, it is essential to declare it as static. Hence, our syntax for the main method declaration should follow this format:

[access modifier] static void

However, this is not a complete declaration for the main method yet. We should also add the parameter "main(String[] args)" to allow the program in executing command-line arguments. And so, our final code for the main method should look like this:

public static void main(String[] args)

If added to our line of codes above, our IDE should look like this:

```
1
2   package javatestproject;
3
4   public class JavaTestProject {
5
6      public static void main(String[] args) {
7
8      }
9
10  }
11
```

Writing the Program Output

As of now, you already covered the basic parts of our Java program. Unfortunately, we are not yet telling the program what exactly it needs to do.

What we want for the program to execute is to deliver a message. This is what we call as the Program Output or Output Stream. In order for the program to deliver a message, we should use the following syntax:

System.out.print("[message]");

Instead of the parameter "print", you may replace it with "println". This will print the output to a new line once executed. Hence, the alternative format should look like this:

System.out.println("[message]");

Let us now tell the program to print the statement "Java is Awesome!" using the command line above. Our syntax should be like this:

System.out.print("Java is Awesome!");

And so, our IDE should look like this once we insert the output code:

```
1
2    package javatestproject;
3
4    public class JavaTestProject {
5
6        public static void main(String[] args) {
7
8            System.out.print("Java is Awesome!");
9
```

```
10    }
11
12  }
13
```

Allowing User Input

The Java programming language allows you to let users communicate with the program by sending inputs. We can do this by using the built-in class known as the Scanner.

The Scanner class collects the user's input from the computer's keyboard. It stores the collected data into variables that can also be executed in the program's output stream. In order to do this, we need to add the following lines right after the package statement:

<div align="center">import java.util.Scanner;</div>

This is an example of an import statement which signals the program to import the Scanner class from the built-in Java package known as java.util. This is where we add them in our IDE:

```
1
2   package javatestproject;
3    import java.util.Scanner;
4
```

This, however, does not do anything yet. We need to assign a variable name to the scanner class first in order to call the scanner's collected data into our output stream. We can assign a variable name to the scanner class by using the following statement:

<div align="center">Scanner [variable name] = new Scanner(System.in);</div>

You can assign any name to the variable. It is a common practice to start the name with an uppercase character followed by lowercase characters. It is also a good practice to separate words by using uppercase characters. Let us say that we want to name our scanner variable as "userinput". Instead of writing the entire variable name in lowercases, replace the first letters of the words "user" and "input" to uppercases. Hence, the statement will look like this:

<div align="center">Scanner UserInput = new Scanner(System.in);</div>

We should insert the statement within the blocks of code after the declaration of the main method. Hence, our IDE should be as follows:

```
1
2   package javatestproject;
3    import java.util.Scanner;
4
5    class JavaTestProject {
6
7       public static void main(String[] args) {
8
9           Scanner UserInput = new Scanner(System.in);
10
11      }
12
13  }
14
```

With this set of Java codes, we are commanding the program to assign the user input to the variable named as "UserInput". Let us now tell the program to call the input. As an example, let us try to ask the user's name through our program and let the program itself return the user's name by delivering another output. Our program should be following this schema:

First Output -> Input -> Second Output

In order to create the first output, we will be using what we have learned from writing an output. Let us ask the user's name by asserting the following statement above the scanner assigns variable name statement:

System.out.print("What should we call you?");

Next, let us return the user's answer by calling the collected data into the second output.

We can do that by writing the second output below the scanner variable name statement. Our second output should be:

System.out.println("Hello "+UserInput.nextLine());

With this, we can come up with our final Java program written as follows:

```
1
2    package javatestproject;
3    import java.util.Scanner;
4
5    class JavaTestProject {
6
7       public static void main(String[] args) {
8
9          System.out.print("What should we call you?");
10         Scanner UserInput = new Scanner(System.in);
11          System.out.println("Hello "+UserInput.nextLine());
12
13      }
14
15   }
16
```

Chapter 2 – Variables

Due to the introduction to the scanner class above, we are now familiar with variables. However, the full potential of variables was still unknown to us.

Variables serve as containers of data. They allow us to call specific data in a more convenient way. We can also create a new set of data using inconsistent values through the use of variables. Most programming languages use variables and developers were able to solve both arithmetic and logical operations through them.

In order to utilize a variable, we need to declare it first and then assign a data type upon its declaration. By doing so, the program will be able to recognize what kind of value was stored inside a variable.

Edit Java Embedding

General | Sensors | Annotations

Name: InvokeSessionBean

Language: java

Version: 1.4

Code Snippet:

```java
try {
    Object homeObj = lookup("ejb/session/CreditRating");
    Class cls = Class.forName(
        "com.otn.samples.sessionbean.CreditRatingServiceHome");
    CreditRatingServiceHome ratingHome = (CreditRatingServiceHome)
            PortableRemoteObject.narrow(homeObj,cls);
    if (ratingHome == null) {
        addAuditTrailEntry("Failed to lookup 'ejb.session.CreditRating'"
                        + ". Please make sure that the bean has been"
                        + " successfully deployed");
        return;
    }
    CreditRatingService ratingService = ratingHome.create();

    // Retrieve ssn from scope
    Element ssn =
        (Element)getVariableData("input","payload","/ssn");

    int rating = ratingService.getRating( ssn.getNodeValue() );
    addAuditTrailEntry("Rating is: " + rating);

    setVariableData("output", "payload",
        "/tns:rating", new Integer(rating));
} catch (NamingException ne) {
    addAuditTrailEntry(ne);
```

Help   Apply   OK   Cancel

Overview of Data Types

There are different data types and we can use each type in various operations and functions. Before we can learn how to assign a data type to a variable, it is important to first familiarize ourselves with the data types. Here is a table of the data types we can use for variables in the Java programming language and their brief descriptions:

Data Type	Description
Boolean	This data type has only two possible values: true or false.
Byte	This data type has the tiniest range compared to all number-based data types. Its value by default is 0 and it may store an 8-bit 2s integer.
Character	This data type only stores a single character. By default, the value will always be an empty space.
Double	This data type can store numerical value of 64-bit IEEE 754 and has a value of 0.0d by default. These are numerical values that have decimal points.
	This data type stores 32-bit IEEE 754 which is less

Float	specific than the Double data type. By default, data is set to 0.0f.
Integer	This data type is the simplest way to handle numerical values. When not specified, it will automatically translate the value into 0. It does not support decimal points which make it inefficient for precise values.
Long	This data type allows you to store numerical value between -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Its default value is 0.
Short	This data type allows you to store a numerical value between -32,768 to 32,767. By default, it is set to 0.
String	This data stores alphanumeric values. Empty string variables will give null values by default.

Creation of Variables

We need to write a declaration statement in order to create a variable for a certain data type. Any name can be assigned to a variable which we will also use when calling the stored data in the future block of codes.

Let us create a variable using the integer data type and let us name this variable as "MyAge". Our declaration syntax should be:

int MyAge;

By default, the program will assign the zero value to the MyAge variable since we did not specify an integer value. In order to assign a specific value inside the MyAge variable, we will use the equal (=) operator sign. Let us say that we want to assign the value 20 to the variable. Our syntax will now look like this:

int MyAge = 20;

The same goes with other data types. Here is the list of examples for each data type when declaring them in our IDE: (Assigned variable name is "TestVar")

Syntax Formula	Example
Boolean [variable name] = [true or false];	Boolean TestVar = true;
byte [variable name] = [numerical value];	byte TestVar = 0;

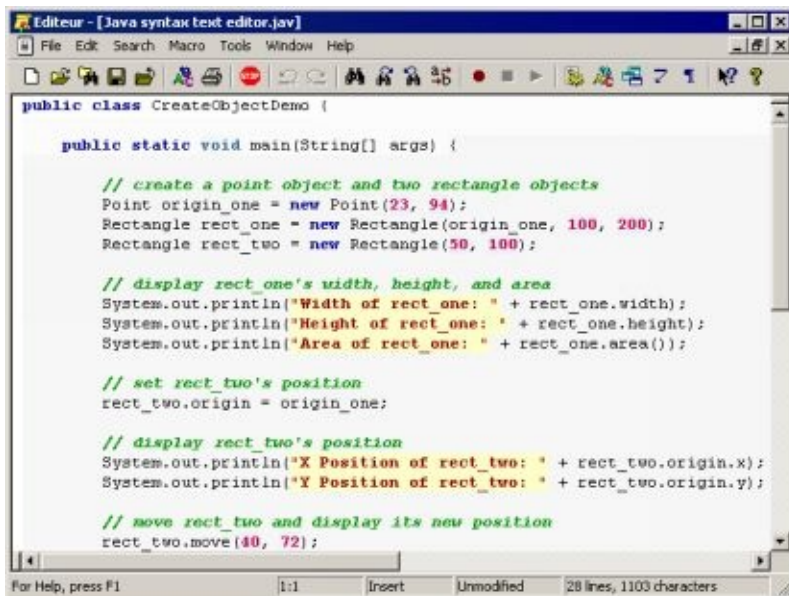|	
char [variable name] = [single character];	char TestVar = L;
double [variable name] = [numerical value]. [numerical value]d;	double TestVar = 31.13d;
float [variable name] = [numerical value]. [numerical value]f;	float TestVar = 1.3f
int [variable name] = [numerical value]	int TestVar = 5
long [variable name] = [numerical value]	long TestVar = 0L
short [variable name] = [numerical value]	short TestVar = 32000
String [variable name] = [alphanumeric value]	String TestVar = null

Data Type and User Input

We already know that we can collect input data with the use of Scanner. Now, we will use the Scanner class to collect other data types. In this example, we will ask for a numerical value from the user and then ask the program to save the value as an integer value.

Our first step is to tell the Scanner class that the value it needs to store is a value of the integer data type. We can do it by changing the nextLine() parameter into nextInt() parameter in our output.

Let us create a program where we will ask the year of birth of the user. Afterward, let us return that value to the user as an integer data type. Our IDE should be like this:

```
1
2    package javatestproject;
3    import java.util.Scanner;
4
5    class JavaTestProject {
6
7        public static void main(String[] args) {
8
```

```
9          System.out.print("What year were you born?");
10         Scanner BirthYear = new Scanner(System.in);
11          System.out.println("You were born in "+BirthYear.nextInt());
12
13     }
14
15  }
16
```

Storing User Input into Another Variable

When calling the Scanner class to collect data from user input, we create a variable as well. In our Java program with regards to the birth year, for instance, we have created a new variable named as "BirthYear" and we store the collected data in it. The problem here is that we also write the parameter right after the variable name when executing the Scanner class data in our program's output. This is too much hassle when you need to call the collected data in multiple statements. Hence, we can store the entire syntax into another variable to simplify our codes. Do this by declaring a variable while assigning the Scanner syntax as the variable's value.

Let us create another program where we calculate the user's age five years from now. This time, we will also store the Scanner call statement into another variable. This is how it should look like:

```java
1
2   package javatestproject;
3   import java.util.Scanner;
4
5   class JavaTestProject {
6
7      public static void main(String[] args) {
8
9          System.out.print("How old are you?");
10         Scanner UserAge = new Scanner(System.in);
11          Age = UserAge.nextInt();
12         System.out.println("You will be "+(Age+5)+" five years from now");
13
14      }
15
16  }
17
```

From our example program, we stored the Scanner call statement into a new variable which we named as "Age". On the next line, we called the user collected integer and wrote a simple arithmetic operation in order to provide how old our users will be after five years from now. Notice that the mathematical expression is enclosed in parentheses. Without the parentheses, the program will connect the stored value in the Age variable with the numerical value 5 instead of adding the two values together.

In addition to integers, you may also use the following parameters to look for other types of data using the Scanner class call statement:

- nextByte() for Byte data type values
- nextShort() for Short data type values
- nextLong() for Long data type values
- nextFloat() for Float data type values
- nextDouble() for Double data type values

Chapter 3 – Operators

Since the first chapter, we have been using different operators and the most common operator we have used is the assignment operator. It is defined by the equal (=) sign and it signals the program to designate a certain value to a variable.

Aside from the assignment operator, there are five other operators that you use to manipulate data in the Java programming language. They are as follows and their role as mathematical operators:

- Operator for Addition which is represented by the plus sign (+). It commands the program to give you the sum of multiple values.

- Operator for Subtraction which is represented by the hyphen sign (-). It commands the program to give you the difference of multiple values.

- Operator for Multiplication which is represented by the asterisk sign (*). It commands the program to give you the product of multiples values.

- Operator for Division which is represented by the slash sign (/). It commands the program to give you the quotient of multiple values.

- Operator for Remainder which is represented by the percentage sign (%). It commands the program to divide two numerical values and gives you the remainder of the numerical values.

Summary in a Single Program

The best way to familiarize with the operators is by creating a Java program where we can use at least one of the mathematical operators. We will also integrate what we have learned so far. In other words, we will also use the Scanner class to collect data from the user and then execute an arithmetic expression to provide efficient output back to the user.

In this program, we will ask the user to input two numerical values and we will let the program to provide the sum of the numerical values. Let us begin writing our Java program:

```
1
2    package javatestproject;
3    import java.util.Scanner;
4
5    class JavaTestProject {
6
7        public static void main(String[] args) {
8
9            int Value1, Value2, Total;
10           System.out.print("Please enter the first value");
11            Scanner FirstValue = next Scanner(System.in);
12           Value1 = FirstValue.nextInt();
13           System.out.println("Please enter the second value");
14           Scanner SecondValue = next Scanner(System.in);
15           Value2 = SecondValue.nextInt();
16           Total = Value1 + Value2;
17           System.out.println("The total of the two values is "+Total);
18
19       }
20
21   }
```

In this program, you will notice that we declared three variables as containers for integer values. This was written on line 3 where we used the following syntax:

int Value1, Value2, Total;

Because of this, we no longer need to enclose the two variables inside the parentheses as we call them for a mathematical expression on line 16.

We also allow users to enter two numerical values by using two Scanner call statements. Then, we stored these values into two separate variables that added to one another inside another variable. This way, we were able to call the sum of the two values for our final output stream.

Conclusion

The Java programming language gives you unlimited tools to achieve different operations. However, it is extremely important to get to know the IDE you chose to use. Beginners should find which IDE they feel most comfortable working on. Although all IDEs use a single programming language, each has its own unique interface that might change your programming pace.

We also learned how to start a new project and the essential elements a project needs such as the main class and the main method. Packages were also introduced and although the package declaration is optional, we have learned that declaring packages in our Java program will help us efficiently organize multiple classes for bigger projects.

The main class and the main method may also contain access modifiers that will allow the program in defining whether it should allow other classes in accessing the statement or not. It is a general practice to use "Public" as the main class's access modifier.

Allowing the program to return a message as its output has also been discussed in the first chapter of this book. You may use the keyword "print" as the simplest form of the parameter or you may rather sign the Java program to show the string in another line. This can be done with the use of the keyword "println".

We also discussed how dull it is to have a one-way-only program where users were not capable of sending an input. The Java programming language has an answer to this by allowing you to use its built-in Scanner class which can be found in the built-in local package known as java.util. The Scanner class allows us to collect data entered by the user

from their keyboard. This opened new operations such as returning the user input as the program output.

The next chapter introduces variables and how do they bring out the full potential of the Java programming language. We discussed the different data types that we can store inside variables and how to call these data for our program's output. It is also important to remember that we can also store variables into another variable for efficient programming later on.

Lastly, our last chapter discusses the different operators aside from the assignment operator represented by the equal sign (=). In the end, we combined everything we have learned to create a Java program which can solve a mathematical equation based on user's input.

At this point, I am confident that you already grasped the idea behind the Java programming language as an object-oriented language. I personally congratulate you for finishing this book and I am quite sure that you are now ready to move forward to a more advanced method in Java programming.

Do not forget that you have only took a glimpse on what the Java language is capable of. This book has a continuation where you will learn more advanced techniques such as flow control, a deeper understanding of access modifiers, a closer look to objects and classes, constructors, serials, and inheritance.

Consider this book as your big leap into the world of Java programming language. Always remember that this programming language is fun to learn, yet filled with tons of challenges ahead. Nevertheless, mastering the language will open new opportunities to you such as being able to develop your own mobile application. You may even end up developing a new stand-alone game that may stand the test of time like Minecraft.

PYTHON

LEARN PYTHON FAST

The Ultimate Crash Course to Learning the Basics of the Python Programming Language In No Time

SATISFACTION 100% GUARANTEE

Stephen Hoffman

Python

Learn Python FAST - The Ultimate Crash Course to Learning the Basics of the Python Programming Language In No Time

STEPHEN HOFFMAN

CONTENTS

Introduction

Python can be used for a wide variety of projects, and there's a reason many programmers will reach for Python before they reach for any other programming language. Python can be used for web applications, automating tasks on systems, finding colors in images, and so many other different programming tasks!

Python is a general-purpose programming language that was created in the late 80's, and it was named after the infamous Monty Python. It's used by thousands of programmers to do anything from testing microchips at Intel to building video games with a PyGame library. It's a small language that resembles the English language and has hundreds of third-party libraries that already exist.

There are two main reasons as to why programmers will choose Python over many different programming languages.

The first reason is readability. Because it's so close to the English language, using words such as 'not' and 'in' make it so that you can read a script out loud and not feel like you're reading some long-forgotten language. This is also helped by the strict punctuation rules in Python that mean you don't have curly braces all over the code.

In addition, Python has some set rules, known as PEP 8, that tell Python developers how to format their code. This means you will always know where to put new lines and every code that you pick up from someone else will look similar and be just as easy as yours to read. That makes for some very easy collaboration between developers!

The second reason is that there are preexisting libraries for almost anything you want to do in Python. The language has been around for over twenty years, so there are many programmers out there who have mastered the language and know how to write what you want. So if you find yourself in doubt, try to find someone else who has already done it!

So the main reasons why you should learn Python are that it's simple and it's easy for you to collaborate with others in order to hone your skills. So let's get started!

Chapter One – Setting up Python

When it comes to programming in python, the first thing you should know is that python is already installed on many of the operating systems. Most operating systems, other than Windows, already come with Python installed by default. If you want to check to be sure it's installed on your computer, open the command line by running the terminal program, and type in 'python –V'.

If you see version information, then Python is already installed on your computer. However, if you get the 'bash: python: command not found' prompt, then there is a possibility that it's not installed on yours.

Follow the instructions for downloading and installing ActivePython on your computer.

Installation

The first step is to go to the Python website, www.activestate.com/Products/ActivePython, this will take you to a website where you can choose the operating system you have. If you have a Windows version that's older than Windows XP, you'll need to install the Windows Installer 2.0 before you continue.
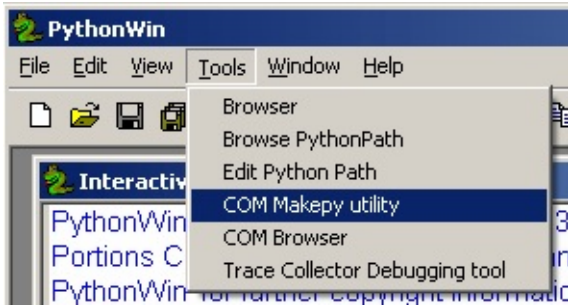
Once you find the installer you need, double click on it and go through the installation steps. After you've installed it, go to your Start-Programs-ActiveState ActivePython (versions #)-PythonWin IDE and you'll see a little script telling you what version you have and what operating system it's installed on.

Interpreter

Once you start the interactive mode with Python, it'll prompt you for the next command. This is shown with three greater than symbols, >>>. If you want to run python interpreter interactively, then you need to type in 'python' in the terminal.

It'll look something like this:

```
??$ python
Python 2.7.2 (default, Jun 20 2012, 16:23:33)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>

>>> 1 + 1
2
>>> you can type expressions here ... use ctrl-d to exit
```

How to Run Programs

So once Python is all set up and you know how to use the interpreter, how are you going to run programs in python? The simplest way to run a program is to type 'python helloworld.py'. If your executable is set on a .py, then the file can be run by name without having to type in the python first.

Now, set the execute bit with a 'chmod' command, such as this:

- $ chmod +x hello.py

Now you're able to run your program as ./hello.py.

Text Editor

Before you're able to write a python program in a source file, you need an editor to write that source file. When it comes to programming python, there are many editors you can choose from. Just pick one that will work with your platform. The editor you choose is going to depend on your experience with computers, what you have to do, and what platform you need in order to do it.

If you're a beginner, choose something like Notepad ++ or TextEdit.

Beginning Writing

Writing and running a python program is just a text file that is being edited by you directly. In the command line or your terminal, you can run any number of programs you want, just as you did with the aforementioned program of 'python helloworld.py'.

When the command line prompt comes up, just hit the up-arrow key in order to recall the commands that were typed previously, that way it's easy to run a previous command without retyping it.

To try out the editor you have, edit the helloworld.py program. Instead of using the word 'Hello', use the word 'Greetings'. Save the edits and run the program again to see its output. Then try adding a 'print yay!' just below the print that exists with the same indentation.

Run the program and see what the edits look like.

Congratulations! You just edited your first Python program! Now that you know how to edit a program let's take a look at variables you can use in your programs.

Chapter Two – Variables

Variables the characters you're able to use in Python. You can use any letter, every number, and every special character, but you cannot start with a number. White spaces and the signs with the special meanings like + and – are not allowed.

Either capital or lowercase letters are allowed, but most prefer to use lowercase letters in order to keep it all uniform. Variables names are case sensitive, and python is typed dynamically, so that means you don't need to declare what type every variable is. In python, the variables are a storage placeholder for texts and numbers, so it has to have a name in order to find it again.

The variable is assigned with an equal sign, followed by the value of your variable. There are a few reserved words in python that you cannot use for your variable. It's also important to know that variables can be changed later on if you need to.

For example, you can use these variables for numbers and create this formula:

- foo = 10
- bar = foo + 10

Here are a few different variable types to get you started.

# integer	X = 123
# long integer	X = 123L
# double float	X = 3.14
# string	X = "hello"
# list	X = [0,1,2]

# tuple	X = (0,1,2)
# file	X = open('hello.py', 'r')

You're also able to assign a single value to several variables at the same time with multiple assignments. Variable a, b, and c are allocated to the identical memory location with the value of 1; a = b = c = 1.

For example:

```
length = 1.10
width  = 2.20
area   = length * width
print "The area is: " , area


This will print out: The area is:  2.42
```

Now that you know how to use variables let's take a look at the interpreter in more detail.

Chapter Three – Interpreter

The interpret is usually installed at the /usr/local/bin/python location on machines where it's available. Putting /usr/local/bin in the Unix shell's search path is going to make it able to start it by typing the command: python to the shell.

When you begin the python interactive mode, it will prompt for the following command using three greater than symbols, >>>. It will then print a welcome message that states its version number and the copyright notice before it prints the first prompt.

Interactively

When used interactively, it's a good idea to have some standard comments executed each time the interpreter is begun. You can do this when you set an environment variable called PYTHONSTARTUP in the title of a file that contains start-up commands. This is like the .profile feature of the Unix shells. To do this, add this line to the .bashrc file:

>> export PYTHONSTARTUP=$HOME/.pythonstartup

Create or change the .pythonstartup file and place the python code in it, like this:

```
import os
os.system('ls -l')
```

To exit your interactive prompt, hit Ctrl+D if you're using a Linux machine.

Chapter Four – Importance of Comments

The comments in python are strictly for the person reading the code, whether it's you or someone else. It's a good way to communicate with groups of people what the code is supposed to do so that they're able to collaborate better. However, there are two ways you can add comments to python that are universally accepted in the programming industry.

A single line comment is going to start with the # symbol and is terminated by the end of the line. Python will ignore all text that comes after the # symbol to the end of that line because it views that as not part of the command.

Comments that will span more than a single line is achieved by inserting a mutli-line string with """" as the delimiter on either end. They are meant as documentation for anyone who is reading the code.

Let's take a look at two examples so that you can see what I'm talking about.

So now you know how to collaborate with others and just keep your code organized with comments for yourself, let's take a look at python docstrings.

Chapter Five - Python Docstrings

Docstrings, or python documentation strings, provide a nice way to associate documentation with python functions, modules, methods, and classes. An object's docstring is made up of a string constant as the first statement in the definition of the object.

It's specified in the source code that's used to record a specific piece of code. Unlike regular source code comments, the docstring should be a description of what the function does rather than how. All functions need to have a docstring.

This lets the program inspect the comments during run time, for example, as an interactive help system or as metadata. You can use the _doc_ attribute on objects to access the docstring.

What It Should Look Like

The docstringline is going to begin with a capital letter and end with a period. The first line is going to be a short description. Don't write the name of your object. If there's more than one line in the documentation string, the second one should be blank so that it visually separates the summary from the remainder of the description. The rest of thelinesshould be one or more paragraphs that describe the object's calling conventions and anything else that's important.

An example of a docstring is as follows:

```
def my_function():
    """Do nothing, but document it.

    No, really, it doesn't do anything.
    """
    pass
```

And this is what it would like when it was printed.

```
>>> print my_function.__doc__
Do nothing, but document it.

    No, really, it doesn't do anything.
```

Declaration

This python file shows you a declaration of a docstring in a python source file.

```python
"""
Assuming this is file mymodule.py, then this string, being the
first statement in the file, will become the "mymodule" module's
docstring when the file is imported.
"""

class MyClass(object):
    """The class's docstring"""

    def my_method(self):
        """The method's docstring"""

def my_function():
    """The function's docstring"""
```

Accessing the Docstring

The following session will show you how the docstring can be accessed:

```
>>> import mymodule
>>> help(mymodule)
```

Assuming this is the file mymodule.py, then the string that is the primary declaration in the file will be the mymoduleelement docstring when the file is introduced.

```
>>> help(mymodule.MyClass)
The class's docstring

>>> help(mymodule.MyClass.my_method)
The method's docstring

>>> help(mymodule.my_function)
The function's docstring
```

So now that you know what a docstring is let's talk more about some keywords in python that you need to know.

Chapter Six - Keywords in Python

Keywords in python are reserved words that are not able to be used ordinary identifiers. They have to be spelled exactly as you see them written. If you want to print the keyword list in python, then use the following commands.

```
$ python
>>>
>>> import keyword

>>> print keyword.kwlist

['and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else',
'except', 'exec', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is',
'lambda', 'not', 'or', 'pass', 'print', 'raise', 'return', 'try', 'while', 'with',
'yield']
```

Definitions of Keywords

print

- print to console

while

- controls the movement of the platform

for

- repeat over parts of a collection in the order they appear

break

- break the loop cycle

continue

- made to interrupt the present cycle without getting out of the whole cycle. A new cycle will start.

If

- Used to figure out which statement is going to be executed

Elif

- Is and abbreviation of else if; if the first test evaluates to false, then it will continue to the next one.

Else

- This means the command is optional. The statement following the else word is executed unless the condition is true.

Is

- Looks for object identity

Not

- Disproves a Boolean value

and

- all circumstances in a boolean expression must happen

or

- at the minimum one condition must happen

import

- bring in other elements into a Python script

as

- gives a module a different alias

from

- for introducing an exact variable, class or a function from an element

def

- makes a new user-defined function

return

- gets out of the function and yields a value

lambda

- makes a new unspecified function

global

- accesses variables defined exterior to functions

try

- states exemption handlers

except

- holds the exception and completes codes

finally

- is constantly completed in the end and is needed to clean up resources

raise

- make a user-defined exclusion

del

- deletes objects

pass

- does nothing

assert

- used for restoring purposes

class

- used to create new user defined objects

exec

- executes Python code dynamically

yield

- is used with generators

Chapter Seven - Booleans, True or False in Python

These values are the two continuous objects of true and false. They're used to symbolize truth values, which are other values that can be considered true or false. In a numeric context, like if they're used as an argument to a mathematic operator, they will behave like the integers zero and one respectively.

The preexisting function bool() is able to be used to create any value to a Boolean if the value is able to be interpreted as a truthvalue. They're written as false and true.

Boolean Strings

A string can be tested for truth value. The return type is going to be in Boolean value or True or False. So let's take a look at an example. First, create a new variable and assign it a value.

numberone_string = "Hello World"

numberone_string.isalnum()   #check if all char are numbers

numberone_string.isalpha()   #check if all char in the string are alphabetic

numberone_string.isdigit()   #test if string has digits

numberone_string.istitle()   #test to see if the string has title words

numberone_string.isupper()   # test to see if the string has upper case letters

numberone_string.islower()   #test to see if the string has lower case letters

numberone_string.isspace()   #test to see if the string has spaces

numberone_string.endswith('d')   #test if the string ends with a 'd'

numberone_string.startswith('H')   #test if the string begins with 'H'

To see what the return value would be, print it out.

numberone_string=”Hello World”

print numberone_string.isalnum()   #false

print numberone_string.isalpha()   #false

print numberone_string.isdigit()   #false

print numberone_string.istitle()   #true

print numberone_string.isupper()   #false

print numberone_string.islower()   #false

print numberone_string.isspace()   #false

print numberone_string.endswith(‘d’)   #true

print numberone_string.startswith(‘H’)   #true

Logical Operator Boolean

These values tend to be respond to logical operators and/or

>>>True and False

    False

>>>True and True

    True

>>>False and True

    False

>>>False or True

    True

>>>False or False

    False

Remember the built-in type of Boolean can have only one or two possible answers: True or False.

Now that you know a little more about Booleans and how to use them let’s talk about python operators.

Chapter Eight - Python Operators

There are a few different types of operators. There's arithmetic, comparison, and logical operators. Let's take a look at each one in a little more detail.

Arithmetic Operators

Python has the modulus -, +,*, /, % and ** operators. Assume that the variable 'a' has a value of ten and variable 'b' has a value of twenty. Therefore:

- +
  - a+b=30
- -
  - a-b=-10
- *
  - a*b=200
- /
  - b/a=2
- %
  - b%a=0
- **
  - a**b=10^{20}
- //
  - 9//2=4

Comparison Operators

The plain comparison operators like ==, <, >=, and the others are utilized on all different manner of values. Strings, numbers, mappings, and sequences are all able to be compared. The following bullets will show you a list of comparison operators.

- < is less than
- == equal
- <= less than or equal to
- >= greater than or equal to
- > greater than
- <> not equal
- != not equal

Logical Operators

The logical operators and or also give back a Boolean value when they're used in a decision structure. There are three logical operators, or, and, and not.

For example, x>0 and x<10 is only true if x is greater than 0 and less than 10.

Operator Description:

- and: logical AND
- or: logical OR
- not: logical NOT

Now that you know about operators let's talk more about using math in python.

Chapter Nine - Using Math in Python

The Python language has the Python interpreter, which is a simple development environment known as IDLE, tools, libraries, and documentation. It's preinstalled on almost all Linux machines and Mac systems, but it can be an older version.

In order to begin using the calculator in the Python interpreter, type in the following script:

```
>>> 2 + 2
4

>>> 4 * 2
8

>>> 10 / 2
5

>>> 10 - 2
8
```

Counting With Variables

To put in values in the variables to count the area of a rectangle, try this code out:

```
>>> length = 2.20
>>> width = 1.10
>>> area = length * width
>>> area
2.4200000000000004
```

Counter

Counters are needed tools in programming in order to increase or decrease a value every time it is run. The following code is a counter:

```
>>> i = 0
>>> i = i + 1
>>> i
1
>>> i = 1 + 2
>>> i
3
```

Counting with a While Loop

And here's an example of when it's useful to use a counter.

```
>>> i = 0
>>> while i < 5:
...    print i
...    i = i + 1
...
0
1
2
3
4
```

The above program will count from zero to four. Between the world while and the colon, there's an expression that is going to be true at first and then become false. As long as that expression is true, the following code is going to run. The code that has to be run has to be indented. The last declaration is a counter that adds one to the value for every time the loop runs.

Multiplication Table

To make a multiplication table in Python, use the following code:

```
table = 8
start = 1
max = 10
print "-" * 20
print "The table of 8"
print "-" * 20
i = start
while i <= max:
    result = i * table
    print i, " * ", table, " =", result
    i = i + 1
print "-" * 20
print "Done counting..."
print "-" * 20
```

The output for this code is going to be as follows:

\>\>Output:

The table of 8

1*8=8

2*8=16

3*8=24

4*8=32

5*8=40

6*8=48

7*8=56

8*8=64

8*9=72

8*10=80

Done counting.

And that's how you use math in Python. Now let's look at exception handling in Python in the following chapter.

Chapter Ten - Exception Handling in Python

Before you get into what you can do with exceptions, you first have to know what they are. An exception is an error that occurs while a program is running. When that error happens, Python will generate an exception that can be used in order to avoid crashing the program.

Exceptions are convenient for handling special conditions and errors in programs. When you believe you have a code that is able to produce an error, then you can put in an exception handle. You can raise this exception on your own by with the raise exception declaration. Rising an exception will break the current code execution and return the exception to the previous code until it can be handled.

Here are some common errors you might see in your program

- IOError – the file is unable to be opened.
- ImportError – Python is not able to find the module.
- ValueError – this is brought up when a built-in operation or function obtains an argument that might have the right type but not the right value.
- KeyboardInterupt – This is brought up when the user will hit the interrupt key on the keyboard, this is usually the Cntrl – C or Del keys.
- EOFError – this comes up when one of the built-in functions, input() or raw_input() gets to an end-of-file conditions or EOF without scanning the data.

Set Up Exception Handling Blocks

In order to use exception handling in Python, first you have to have a catch-all except clause. The words 'try' and 'except' are Python keywords that programmers use in order to catch exceptions.

For example:

try – except [exception-name] (insert exception) blocks

The code in the try clause is going to be executed statement by statement. If an exception happens, the rest of the block is going to be skipped and the except clause is going to be run.

Try:

some statements here

Except:

exception handling

So let's see an example of this.

try:

print 1/0

except zerodividionerror:

print "You can't divide by zero."

How does it work?

Error handling is handled through the use of exceptions that are made in try blocks and handled in except blocks. If your code encounters an error, a try block code execution is going to stop and transfer down to the except block.

In addition to using the except block after the try block, you're also able to use the finally block. The code in this block is going to be executed whether or not the exception happens.

Let's take a look at some code to see what will happen when you do not use error handling in a program.

Code Example

This program is going to ask the user to input a number between one and ten and then print that number.

number = int(raw_input("Enter number between 1-10"))

print "you entered number", number

This program will work just fine as long as the user enters the number, but what happens if they put in something else, like a string?

Enter a number between 1-10

hello

You can see that the program will throw an error when the string is entered.

Traceback (most recent call last):

File "enter_number.py", line 1, in

    number = int(raw_input("Enter number between 1-10\n"))

valueerror: invalid literal for int() with base 10: 'hello'

Thevalueerror is a type of exception. Now let's see how you can utilize exemption handling to repair the aforementioned program.

import sys

print "Let's fix the preceding code with exception handling"

try:

    number = int(raw_input("Enter a number between 1-10 \n"))

except valueerror:

```
    print "Only numbers are accepted."
sys.exit()


print "you entered number \n", number
```

If you now start the program and input a string in lieu of amounts, you can see that you'll get a different response. So now, let's run the program to be sure it worked. The response should be:

Enter number between 1-10

hello

Only numbers are accepted.

Try … Except … Else Clause

The else clause in a try, except statement has to follow all except clauses and is used for code that has to be executed if the try clause will not raise an exception. For example:

```
try:


    data = something_that_might_go_wrong
except IOError:
    handling_the_exception_error
else:
    trying_a_different_exception_handling
```

Exceptions in an else clause will not be handled by the preceding exception clauses. Be sure that the else clause is run before the final block.

Try … Finally Clause

Thefinallyclause is an optional clause. It's used to define clean-up actions that have to be executed under all circumstances. For example:

```
try:
    raise KeyboardInterrupt
finally:
    print 'Goodbye!'
…
    Goodbye!
    KeyboardInterrupt
```

A final clause will be executed before leaving the try statement, regardless of whether an exception has happened.

Remember that if you don't specify the exception type on the exception line, it's going to catch all exceptions. This is not a good thing because it means the program will ignore any unexpected errors, as well as errors the exception block is able to handle.

Now that you know about exceptions in Python let's take a look at Strings Built-In Methods in the following chapter.

Chapter Eleven - Strings Built-In Methods

This chapter is just going to contain a list of built-in methods you can use for each string. Let's first make a string with the value of 'Hello'.

It'll look a lot like this:

    string = 'Hello'

In order to manipulate this string, take a look at some of the following built-in methods and their explanations.

- string.upper()
    - This makes all the letters in the string uppercase, so if the string were 'hello jane', it would come out as 'Hello Jane'.
- string.lower()
    - This will make all the letters in your string lowercase, so if the string reads 'Hello Jane', it will come out as 'hello jane' in the program.
- string.capitalize()
    - This will capitalize only the first letter. Therefore, if the string was 'hello jane', it would read as 'Hello jane' in the program.
- string.title()
    - This would capitalize the first letter of the words. So if the string were 'hello jane' in the program, it would read 'Hello Jane'.
- string.swapcase()
    - This string will convert all uppercase letters to lowercase and vice versa, so the following string, 'hello jane' would read 'HELLO JANE' in the program.
- string.strip()
    - This string removes all white space from the string. So if the string was 'Hello Jane', it would read 'HelloJane' in the program.

- string.lstrip()
    - This string removes all whitespace from the left. Therefore, if the string were ' Hello Jane ' with space before hello and one after Jane, then the code would spit out 'HelloJane'.
- string.rstrip()
    - This string will remove all whitespace from the right. Therefore, if the same line of code were to read 'Hello Jane ' with a space after Jane, the space after Jane would be removed first to read 'HelloJane'.
- string.split()
    - This command will split words. So if the code read 'HelloJane', then it'll be split into 'Hello Jane'.
- string.split(',')
    - This means the code will split words by a comma, so the following string 'Hello Jane' would come out 'Hello, Jane'.
- string.count('l')
    - This string will count the amount of times l, or any other letter you choose to put into the string, occur in the string. So the input would be 'Hello Jane' and the program would tell you 2.
- string.find(Ja)
    - This string will find the word 'Ja' in the string.
- string.index("Ja")
    - This will find the letters Ja in the string.
- ":".join(string)
    - This will add a : between every character. So 'Hello Jane' would come out 'H:e:l:l:o:J:a:n:e'
- "".join(string)
    - This will add white space between every character, so 'Hello Jane' would read 'H e l l o J a n e'
- len(string)
    - This will find the length of the string in characters, so if the string is 'Hello Jane' then the program would say ten.

So now that you know some of the basic strings that are built-in to the Python code, let's take a look at list examples in the following chapter.

Chapter Twelve – Lists

Lists are created with square brackets, [], and the elements that are between those brackets are what the program will use. The elements in a list do not have to be the same type, ergo, you can have letters, numbers, and strings between just one set of brackets.

Let's take a look at an example.

```
myList = [1,2,3,5,8,2,5,2]
i = 0
while i < len(myList):
    print myList[i]
    i = i + 1
```

So what does this list do? The script is going to create a list, labeled (list1), with the values of 1, 2, 3, 5, 8, 2, 5, 2. The while loop is going to print each part of the list. Each part of the list list1 is going to be obtained by the index or the letter in the square brackets.

The len function will be used to go through the entire length of the list. And the final line tells the program to increase each variable by one for every time the loop runs.

So the output for this program is going to be: 1 2 3 5 8 2 5.2

Let's take a look at another example.

This example is going to count the average value of the elements that are in the list.

```
list1 = [1,2,3,5,8,2,5.2]
total = 0
i = 0
while i < len(list1):
    total = total + list1[i]
    i = i + 1

average = total / len(list1)
print average
```

The output for this will be:

#Output >> 3.74285714286

As you can see, lists are pretty easy to create and manipulate. Let's take a look at using dictionaries in Python.

Chapter Thirteen - How to Use Dictionaries in Python

You can use the curly brackets, {}, to make a dictionary in Python. You can use the square brackets, [], to index the dictionary. Separate the key and the value with some colons and with commas between the pairs. Keys have to be quoted just like with lists, and you can print out the dictionary by printing the reference to the dictionary.

A dictionary will map a set of objects (keys) to another set of objects (values). A python dictionary will map unique keys to values. Dictionaries are changeable or mutable in python. The values the keys point to are able to be any Python value. They are unordered, so the order the keys are added does not reflect with the order they might be reported back as.

Let's take a look at how to create a new dictionary in python.

In order to make a dictionary, you first have to start with an empty one. Use:

```
>>>mydict={}#
```

Example 1 – Creating a New Dictionary

This creates a dictionary that has six key-value pairs to begin with, where iPhone* is the key and the years the values.

```
released = {
                "iphone" : 2007,
                "iphone 3G" : 2008,
                "iphone 3GS" : 2009,
                "iphone 4" : 2010,
                "iphone 4S" : 2011,
                "iphone 5" : 2012

        }
print released
```

The output for this would be:

```
>>Output

{'iphone 3G': 2008, 'iphone 4S': 2011, 'iphone 3GS': 2009, '

        iphone': 2007, 'iphone 5': 2012, 'iphone 4': 2010}
```

Now let's add a value to your dictionary.

```
#the syntax is: mydict[key] = "value"
released["iphone 5S"] = 2013
print released
```

```
>>Output

{'iphone 5S': 2013, 'iphone 3G': 2008, 'iphone 4S': 2011, 'iphone 3GS': 2009,

'iphone': 2007, 'iphone 5': 2012, 'iphone 4': 2010}
```

Now that you know how to do that let's remove a key and the value from the code using the del operator.

```
del released["iphone"]

print released
```

The output would be:

```
>>output

{'iphone 3G': 2008, 'iphone 4S': 2011, 'iphone 3GS': 2009, 'iphone 5': 2012,

'iphone 4': 2010}
```

Now let's check the length of the function by using the len() function.

```
print len(released)
```

Now let's test out the dictionary! Check to see if a key exists in a given dictionary by using an operator such as this:

```
>>> my_dict = {'a' : 'one', 'b' : 'two'}
>>> 'a' in my_dict
True
>>> 'b' in my_dict
True
>>> 'c' in my_dict
False
```

or this if you have a loop:

```
for item in released:
    if "iphone 5" in released:
        print "Key found"
        break
    else:
        print "No keys found"
```

or this if you need a value of a specified key:

```
print released.get("iphone 3G", "none")
```

or this if you need to print all key with a for loop:

```
print "-" * 10
print "iphone releases so far: "
print "-" * 10
for release in released:
    print release
```

or this if you want to print all the key and values:

```
for key,val in released.items():
    print key, "=>",val
```

or this if you need to get only the keys from your dictionary:

```
phones = released.keys()
print phones
```

Now that you know how to do those, why not look at how to print the values?

```
print "Values:\n",
for year in released:
    releases= released[year]
    print releases
```

Here's how to sort your dictionary:

```
for key, value in sorted(released.items()):
    print key, value
```

And that's how you use a dictionary in Python!

Conclusion

There's a lot that you can do with Python code. Many developers will use it to do something as simple as sorting some inventory to something complex like making a video game.

The important part about Python is that there are user groups all over the globes, known as PUGs, who do major conferences on continents across the globe so that Python users can interact with one another. Many of these conferences allow children to attend, who will learn how to use Python on their new Raspberry Pi's they're given at the conference.

Overall, the community surrounding the language of Python is very positive and upbeat. They are a group of people who want to make the world a better place with their knowledge. So never be afraid to reach out to forums and discussion groups to figure out how to write the code you need.

Thank you for reading. I hope you enjoy it. I ask you to leave your honest feedback.