LEARNING CODING AT HOME

CODING FOR KIDS PYTHON

MARK B. BENNET

A PLAYFUL WAY FOR PROGRAMMING, CODING AND MAKING PROJECTS WITH YOUR KIDS. LEARNING COMPUTER SCIENCE EFFICIENTLY WHILE STIMULATING YOUR KIDS' CREATIVITY.

CODING WITH PYTHON

A Playful Way Of Programming, Coding, And Doing Projects With Your Kids. Learn Computer Science Quickly While Stimulating Your Kids' Creativity.

Mark B. Bennet

Table of Contents

Note to Parents

This book is intended for children who wish to learn to programming, especially in Python language.

As the computer world is very advanced and fast-paced, new solutions for a particular problem appear every day. This means that kids interested in computers have to follow the innovations and news in the field they want to explore further, no matter what it is. For example, if they are interested in making video games (which most kids initially use a computer for, even before they are old enough to start preschool education), they constantly have to follow computer science news or the innovations in the computing field, programming languages, and updates to programs and languages used to create computer programs and applications.

Imagine you have coded a calendar. It is a basic one, in which you have 12 months/365 days with just a small space next to each day, where you can note your appointments and tasks for that day. This is good for today, but as soon as tomorrow comes, a program with more advanced functions will appear on the market. Someone else may have found a way to put, for example, more space for editing besides each day, or may have coded a clock into the calendar. In this new, advanced calendar, besides the date, you may be able to keep track of time during each day, or perhaps holidays

or the ability to set audible reminders in the calendar are now included. This is the kind of innovation we are talking about in this book.

The reasons mentioned here make programming one of the most useful and beneficial skills to possess, for both children and adults alike. As with most other science-based disciplines, it is best to start learning early in childhood - the sooner, the better. Therefore, this book is written to motivate children to start learning to code in order to enhance their imagination and logical reasoning skills, and to prepare them for their future adult lives. Today, computer literacy is mandatory. This book can help you give the children in your life the tools they need to be successful when they grow up.

Introduction

Welcome to Python

Hey there!

If asked what a core subject in school is, English, Mathematics and Science are most likely the subjects that jump to mind. However, in recent years, we have seen coding appearing as a new ‘ must teach ’ subject, although there is an air of controversy over whether it should or shouldn ’ t be taught. Wherever you stand on this debate, there is no denying that learning to code will provide children with valuable skills that can be used in all areas of their life.

Even if you are learning to code as an adult, this book is the perfect starting point for a much-needed foundation. This book is most definitely not just for kids.

This book contains proven steps and strategies on how to start teaching children (or adults) to code. It includes hands-on activities that can be used both in the classroom and at home to consolidate coding concepts, as well as useful resources for working with code on a computer or tablet.

It is written in an easy-to-read way such that even parents or teachers who don ’ t have much coding knowledge or experience will be able to learn and teach the activities.

What is Coding?

Coding sounds daunting if you aren ’ t technologically minded. We imagine it to be complicated jargon, which of course it is until you understand it. In reality, anyone can learn to code, just like anyone can learn a new language, or how to play a musical instrument. It just takes perseverance.

Computer code is simply a set of instructions that tells a computer what to do. We call it code because the words and symbols used in computer languages are known as code.

You see, while we think that they are smart, computers are, in fact, useless unless somebody methodically tells them what to do. A person who writes this code is known as a computer programmer. These instructions are then used to tell a computer how to run a specific program, follow a command, or open an application (app). For example, when you click the button on your mouse, pre-existing code written to tell the computer what to do is activated when the button is clicked.

Why Learn to Code?

According to Code.org, there are more than 500,000 unfilled computing jobs, and programming jobs are the number one source of new wages. Coding truly has evolved into the new literacy of the Augmented Age. Even

if you do not become a software engineer or work a computing job, you will still be impacted by its prominence in the industry. Developing at least a fundamental understanding of programming is a valuable skill-set for future employment opportunities and understanding the implications of technology use. Technology is such an integral part of this generation, that it is rather naive to not have some understanding of software that controls billions of devices and how they work.

Programming is about problem solving and troubleshooting. Computers are innately moronic. It takes the cognitive effort of a human being to get them to perform a task. Before solving a problem or building an application, it is essential you understand the underlying process and can explain it before writing code. Why do you think programmers and trendy tech startups like whiteboards and glass windows sell so much?

Why Python?

Just like humans can understand many different languages, a computer can understand the ideas and concepts that we put in through several different programming languages. In this book, we will focus on the Python programming language, because Python is easy to understand, can be used in many different ways, and can be learnt quickly. Also, it is a popular language that runs on almost every machine and is used at many influential, important organizations like Google, Instagram, NASA, and Spotify.

What Is In This Book?

This book is designed to teach you the basics of coding in the Python programming language so that you become code-literate, and can apply the learned concepts to other applications or languages. As an introduction, this book covers the constructs that are shared across almost all languages, although some are Python-specific. By the end of this book, I cannot

promise that you will be able to build the next Facebook or build an artificially intelligent system like Mark Zuckerberg ’ s project, Jarvis, but you will have the tools you need to learn more about an area such as game development and pursue a project. If you plan on enrolling in AP Computer Science A, at school, or a self-challenging AP test, this book does not cover Java. However, the concepts acquired will still be helpful to develop a fundamental understanding of the material.

CHAPTER 1:

Learning to Program

What You Will Learn

- Python basics (get to know your environment)
- Installing Python (Windows - Mac - Ubuntu)
- Once you have installed Python
- Saving your program
- Using IDLE (A to Z guide)

Python Basic (Get to Know Your Environment)

To start programming, there is need to understand computer language. Computers do not speak like us. They take commands, one at a time, and unfortunately, they do not speak plain English, so we need to learn a computer language. Programming is done in several languages that any computer can understand, such as C, C++, Python, JavaScript, Ruby, and many more. All of these languages enable us to instruct our computers on what to do. In this book, however, we are going to explore Python, which is a simple language, but powerful enough to let you code pretty much everything, including games.

Python is a fully-featured programming language that is taught in many courses around the world. Some start learning it in elementary school, while others study it in software engineering and computer science classes at

university level. This powerful, yet easy-to-understand programming language is even used by some of the most popular technologies in the world. For instance, YouTube and Gmail are some of the most famous examples, among many more. Now, to get you on your path to learning Python, you need to perform three actions. First, you need to download Python from the official website, and then install it on your computer. Once you have done that, you can give it a try by creating a basic program. But first, why learn Python at all? If you like games or if you look at the big software developers, you will see that many of them use C++, C#, or other languages.

Installing Python (Windows - Mac - Ubuntu)

I know you want to dive right into coding, but you cannot do that until you have the right tools. I will walk you through the step-by-step process of installing Python. Let us get started!

ON A PC

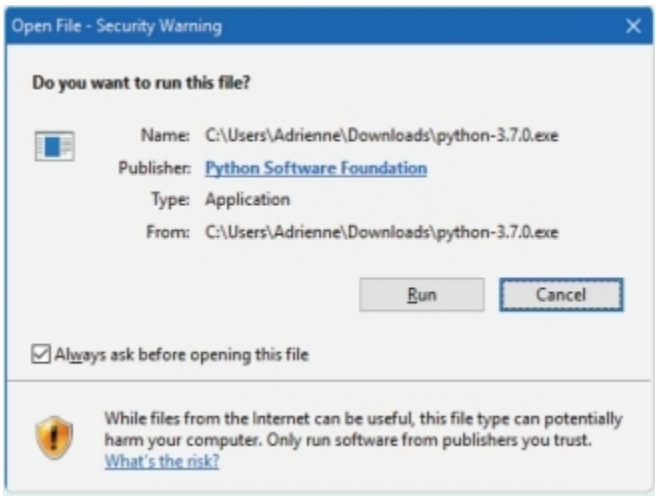
If you are on a Windows computer, you probably do not have Python installed already. This is because Windows operating systems do not usually come with the Python language. That is okay though! We can get it ourselves.

1. On your computer, open an Internet browser like Google Chrome or Mozilla Firefox.
2. In the address bar, type https://www.Python.org/downloads/ to go to the official Python Downloads section.
3. Through the magic of coding, the website will probably know what type of computer you are using, and the **Download** button will show you the correct version of Python to install. In our case, we

wanted Python 3.7.0 which was the latest version as at the time this book was written. Do not worry if it tells you to download a newer version. Go ahead and click the **Download** button.

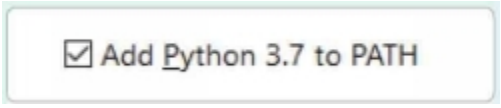
4. A download will start and will probably go to the bottom of your window as shown in the picture.

5. Once your download is complete, click on it to begin the installation. When you do, a window should pop up.

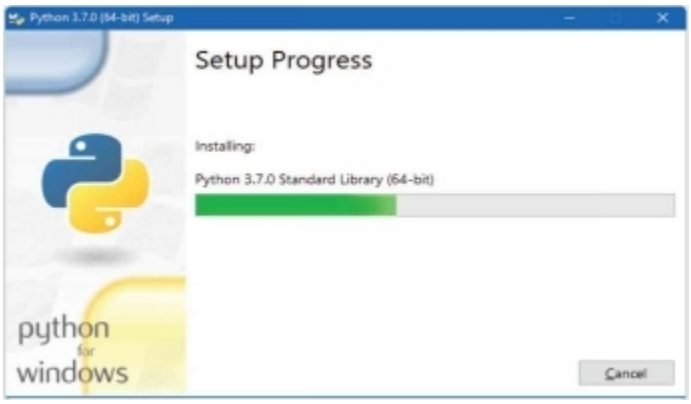


6. Go ahead and click the **Run** button. Then, you will see this window (yours may say 32-bit if that is right for your computer).

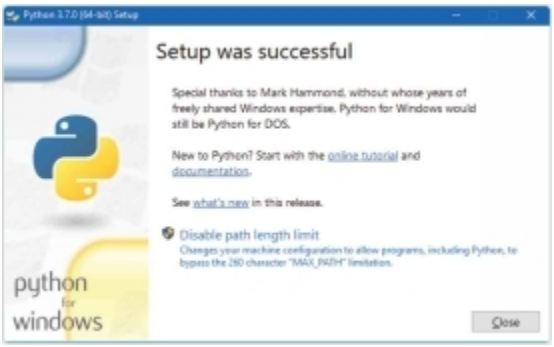
7. Make sure to check the **Add Python 3.7 To PATH** checkbox.



8. Click **Install Now** . Python should start installing. You should see a window like this one:



9. Wait for the installation to finish and the green bar to be complete. Once it is done, the final screen should appear, saying that your installation was successful.



10. You are done! Click the **Close** button and you are ready to go. You have installed Python on Windows!

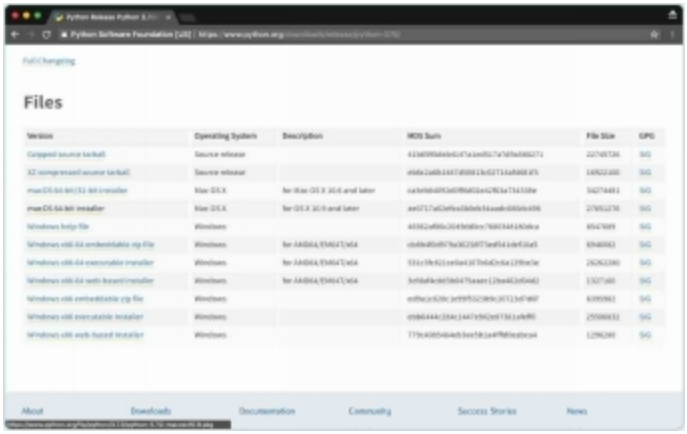
O N A MAC

1. On your computer, open an Internet browser like Google Chrome or Mozilla Firefox.

2. In the address bar, type https://www.Python.org/downloads/ to go to the official Python Downloads section.

The website will probably know what type of computer you are using, and the **Download** button will show you the correct version of Python to install.

You can also find the installer for the specific computer you are using in the Files section.



3. After clicking on the version, a download should start. Wait for it to finish before starting the installer.
4. When you start the installer, you should see a window like the one below.

5. Click the **Continue** button. You will then be presented with some important information that you can choose to read or not.

6. Click the **Continue** button. Next, you will see the license information.

7. Keep going! Click the **Continue** button. You will be asked to agree to the terms of the software license agreement.

8. Click the **Agree** button. You will reach the final window, as shown below.

9. Click the **Install** button. If you need to, enter your personal username and password for your account on your computer. Mac OS sometimes asks for this to make sure you want to install something. If you do not see this pop-up window, you can continue to the next step.

10. The installation should begin.

11. Wait for the installation to finish. Once it is done, you should see the below window.

12. Congratulate yourself! You have just installed Python on your Mac!

Note: You may have noticed we asked you to type https://www.Python.org/downloads/ . But is “ https:// ” really necessary, or could we just start with “ www.”? The answer is this: Python is good at redirecting you to the right site, but adding “ https:// ” before typing web addresses is a good practice to get into, so you can be sure your computer is going to a secure site!

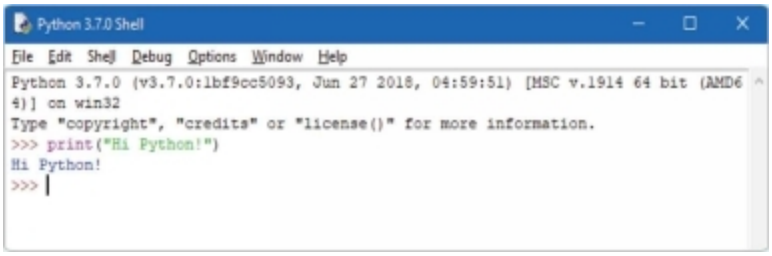
Once You Have Installed Python

Open up IDLE on your computer (if it is not already open). Whenever you open up the IDLE program on your computer, you will always be brought to the Shell first. The Shell is the interactive window that allows you to write Python code within it and then see the results of your code right away. You will know when you are in the Shell because it will say Python 3.7.0 Shell in the title bar of the window.

In the Shell, go ahead and type the following code:

```
print("Hi Python!")
```

Now, hit the **Enter** key. Do you see something like this?



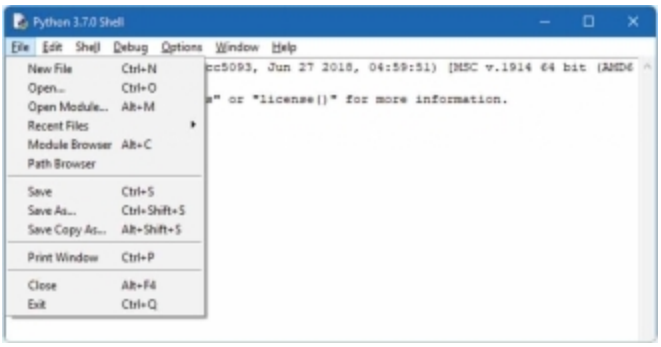
Great job! You have written your first line of Python code! Give yourself a pat on the back or high-five the person closest to you. You are about to learn some awesome things next.

Saving Your Program

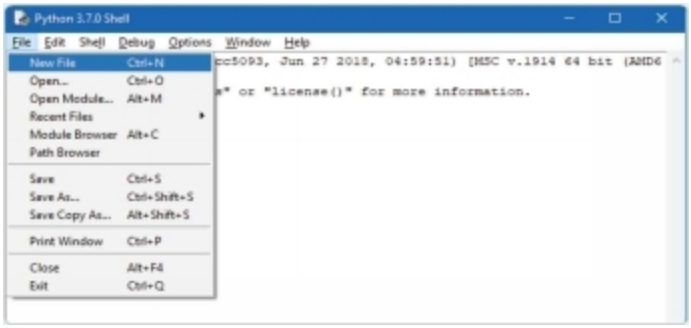
Even though it is a short program, Let us save our Python greeting to a separate file, so you can see how easy it is to save your work.

First, let us create a new file:

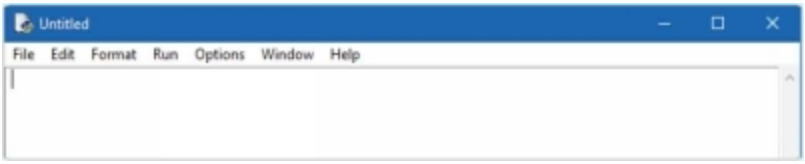
1. On the Menu bar in your Shell, click the File tab to open its context menu, which is a list of actions that you can perform.



2. Click **New File** .



3. A new window should pop up, like the picture below.



4. Type in your greeting, using Python code: print("Hi Python!").

We have to put our greeting into this piece of Python code so that the computer knows to write this message for us on the screen (you will learn more about this later).

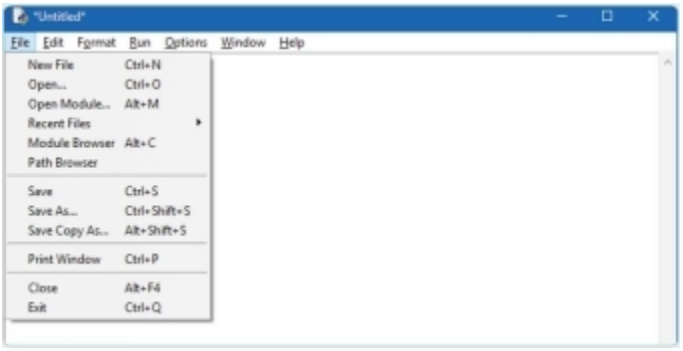
Great! Now you have your code in a file that we can save. This is important because the first code we wrote was in the Shell, which means it will not be saved once you close the window. Writing code directly in the Shell is just a

quicker way to run Python code and see the results right away. Always create a new file and save it, to keep track of your work and save your progress!

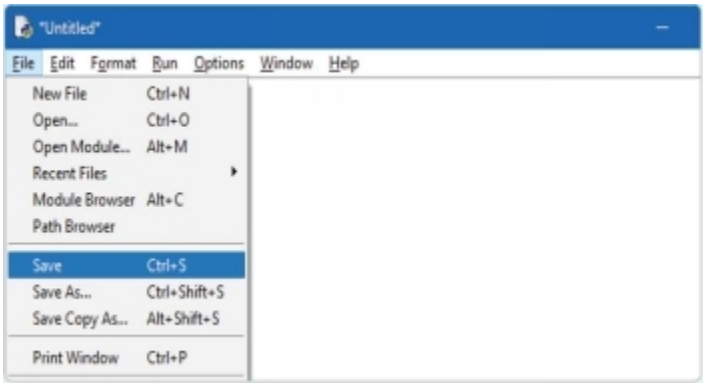
Now that we have created a file with our greeting code, let us save it.

You can save your program in IDLE by following these next steps.

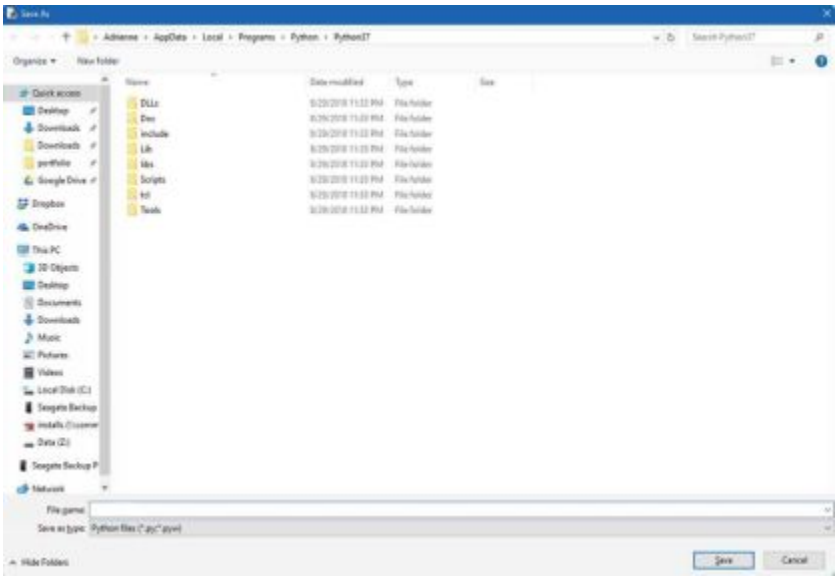
5. On the Menu bar of your Shell, click the File tab to open its context menu.



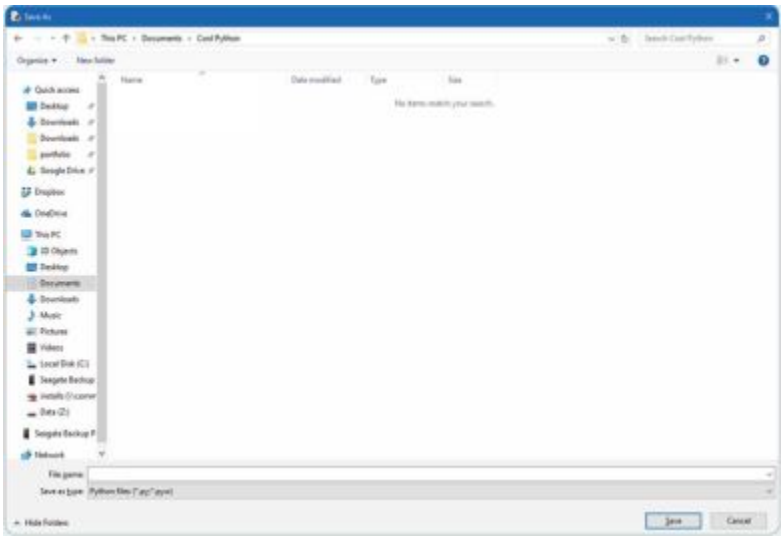
6. Click **Save** .



7. The next window will ask you to name your file. Go ahead and give it a name. I will call mine ‘ greeting. ’



8. Make sure to save your Python program in a place that you will not forget! If you do not choose another place, new files are usually saved in the same folder where Python was downloaded, so go ahead and change the Save In folder to a better one. I created a folder called Cool Python in my Documents directory, so that is where I will save my programs.



9. Click **Save** . That is it!

Using IDLE (Guide A to Z)

When you download and install Python, it will also install an application called IDLE. IDLE is short for Integrated Development and Learning Environment (that is a mouthful!), and it is an integrated development environment (IDE), that helps us in writing Python programs. Think of it as an electronic notepad with some additional tools to help us write, debug, and run our Python code. To work in Python, you will need to open IDLE; opening Python files directly will not work!

Let us take a look!

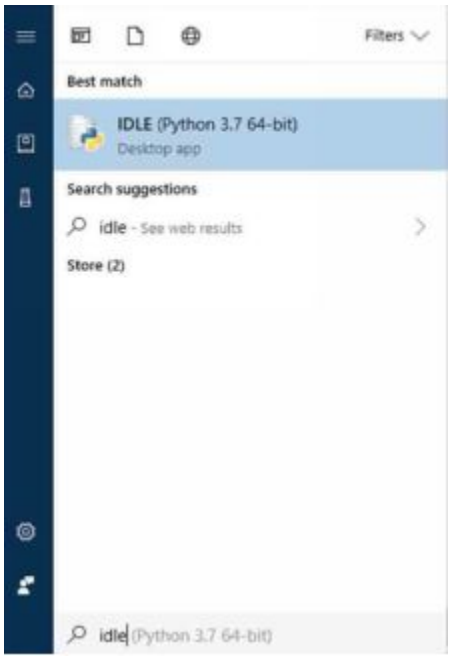
ON A PC

1. Click the Windows Start menu.

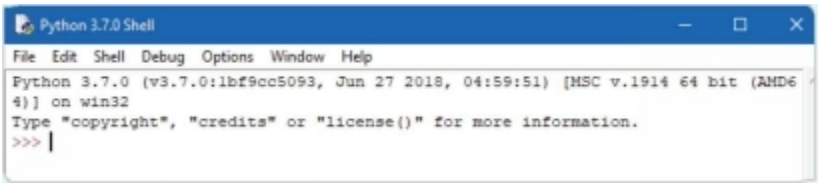
- The above images are being used in another book named [1]

2. Start typing ‘ idle, ’ then select the search result ‘ IDLE (Python 3.7 64-bit). ’

Note: Yours might say IDLE (Python 3.7 32-bit) if that is the kind of computer you have.



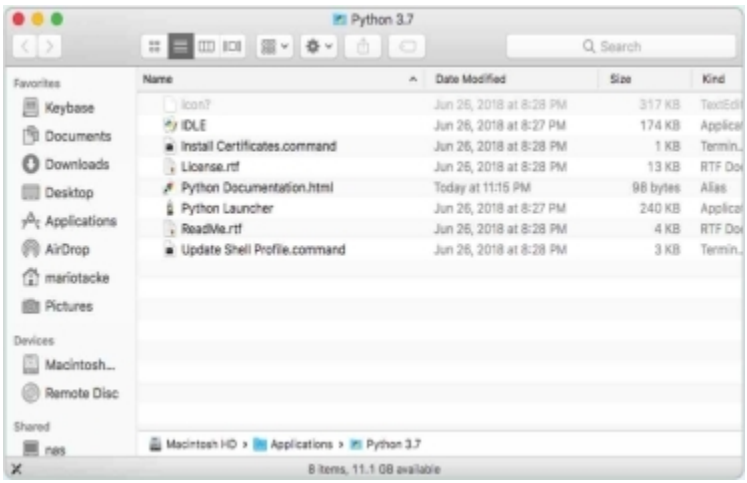
3. A window should pop up that looks like this:



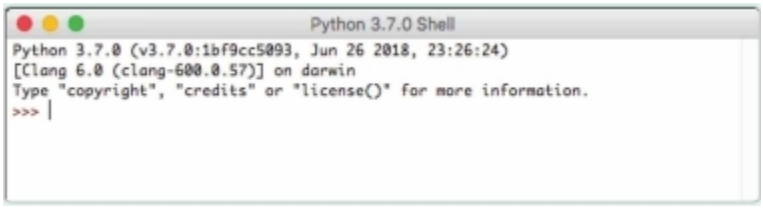
4. Ta-da! Awesome! You opened IDLE in Windows, and are now ready to start writing some code in Python! :)

O N A MAC

- Navigate to **Go** > **Applications** .
- Find the Python 3.7 folder and open it.



- Double-click on the IDLE icon.
- A window should pop up that looks like this:



- Whoo-hoo! Congratulations! You opened IDLE on a Mac and are now ready to start writing code in Python!

Activity 1: Rock Paper Scissors

The first game will be Rock, Paper, Scissors, which is normally played by two people, but in this case, it ’ s going to be you against the computer. The first thing we need to do when creating a game is brainstorming. Using a pen and paper, map out how the game should be designed. Start by first

considering the rules of the game, and only then worry about the programming side.

This classic game involves choosing one of three objects, as the name suggests. Once both selections are made, the items are revealed to see who wins. The player who wins is determined by three simple rules: Rock will crush Scissors, Scissors will cut Paper, and Paper covers Rock.

To handle these rules we are going to create a list of choices, similar to the list of colors we created before in some of our drawing programs. Then, we will add a random selection function that will represent the choice the computer makes. Next, the human player will have to make his or her choice. Finally, the winner is decided with the help of several if statements.

Before we continue with the code, you should start performing these steps on your own. You already have the plan and you know which steps you need to take. So, simply break down the game into easy sections, and work on each section one at a time. If you don ’ t remember how to write an if statement correctly, go back to the chapter about if statements and refresh your memory. The point of this chapter is to help you apply what you already know. So give it a try before you read the following code.

Have you tried to create your version of the game yet? If so, good job! Even if you didn ’ t finish it or you wrote the game and are still getting some errors, you should still reward yourself for trying. Now, let ’ s go through the code and see how this game should turn out:

```
import random

selectionChoices = [ “ rock”, “ paper”, “ scissors ”]

print ( “ Rock beats scissors. Scissors cut paper. Paper covers rock.”)
```

```
while player != “ quit ”:
   player = player.lower ()
computer  =  random.choice(selectionChoices)
                print("You  selected " +player+ ",
                and  the  computer  selected"  +computer+ ".") if player ==
        computer:
print("Draw!")
elif  player == "rock":
                if computer == "scissors":
                print ("Victory!")
                  else:
                  print("You lose!")
elif  player == "paper":
                if computer == "rock":
                  print("Victory!") else:
                  print("You lose!")
elif player == "scissors":
                if computer == "paper":
print ("Victory!") else:
                print("You lose!")
                else:
print("Something went wrong…")
                print()
```

First, we import the random package which allows us to use a number of functions that we are going to take advantage of, giving the computer the ability to make random choices. Then, we create a list for the three game objects and print the game rules so that the human player knows them. The computer will already know what to do because it is programmed, after all. Next, we ask the player to type his or her choice, and then a loop is executed to check the choice of the player. The player also has the option of quitting the Prompt window, and when that happens the game is over. Our loop makes sure that if the player doesn ’ t select the quit option, the game will continue to run.

The next step is to ask the computer to select one of the three game objects. This choice is done randomly, and the selected item is stored inside a variable called “ computer.” After the choice is memorized, the testing phase begins to see which player will win. First, a check is performed to see whether the two players have chosen the same item. If they did, then the result is a draw, and nobody wins. Next, the program verifies whether the player chose rock, and then it looks at the computer to see if it chose scissors. If this is the case, then the rule says Rock beats Scissors, so the player wins. If the computer neither selected Rock nor Scissors, then it certainly chose Paper. In this case, the computer will win. Next, we have two elif statements, where we perform two more tests that check whether the player selected Paper or Scissors. Here, we also have a statement that checks to see if the player chose something that isn ’ t one of the three possible items. If that is the case, an error message is sent that tells the player he either chose something that is not allowed, or he mistyped the command.

Lastly, the user is prompted to type the next selection. This is where the main loop goes back to the beginning. In other words, the game starts

another round of Rock, Paper, Scissors.

This game is simple, but it is fun because anyone can win. The computer has a chance of beating you and there ’ s also a real chance of the game ending up in a draw. Now that you understand how to create a random chance type of game, let ’ s look at other examples to add to our game library while also learning Python programming.

Activity 2: Guess!

This project will be another fun chance-based game that will make use of the random module. The purpose of the game will be choosing a number between a minimum and a maximum limit, and then the opponent tries to guess that number. If the player guesses a higher number, he will have to try a smaller one, and vice versa. Only a perfect match will turn into a win.

Comparing numbers is something we already did in the previous chapters by using the if statement. We have also used the input function to interact with the program, and we are going to make use of it here once again. In addition, we will need a while loop as well.

In this project, the random module is needed because of certain functions. For instance, we know that we need to generate a random number, therefore we will use a function called “ randint” which stands for random integer. The function will have two parameters, which represent the minimum number we can have, as well as the maximum number. You can try out this function on its own. Just import the module and then type the following:

```
import random

random.randint (1, 20)
```

Python will now automatically generate a random figure between 1 and 20. Keep in mind that the minimum and maximum values are included in the number generation; therefore, Python can also generate numbers 1 or 20. You can test this command as many times as you want to make sure that you are truly getting random values. If you execute it often enough, you will see that some values will repeat themselves, and if the range is large enough you might not even encounter certain numbers, no matter how many times you run the code. What ' s interesting about this function though, is that it isn ' t truly random. This is just a side note that won ' t affect your program, but it is intriguing nonetheless. The randint function actually follows a specific pattern; the chosen numbers only appear to be random, but they aren ' t. Python follows a complex algorithm for this pattern instead, and therefore, we experience the illusion of randomness. With that being said, let ' s get back to the fun and games. Let ' s create our game with the following code:

```
import random

randomNumbers = random.randint (1, 100)

myGuess = int (input ( " Try to guess the number. It can be anywhere from 1 to 100:"))

while guess != randomNumbers:

  if myGuess > randomNumbers:

  print (myGuess, "was larger than the number. Guess again!"

  if myGuess < randomNumbers:

  print (myGuess, "was smaller than the number. Guess again!"

myGuess = int (input ( " Try and guess again! " ))
```

print (myGuess, “ you got it right! You won!”)

That ’ s it! Hopefully, you tried to create this game on your own because you already have the tools for the job. Remember that programming is only easy as long as you practice it enough on your own. Just take it one step at a time. With that being said, let ’ s discuss the code in case you need some help figuring the game out. Just like before, we first need to import the random module so that we can use the random number generating function. Next, we use the randint function with two parameters. As mentioned before, these parameters are the lowest number we can guess, which is 1, and the highest number we can guess, which is 100. The random number generator will generate a number within this range. Once the number is generated, it is stored inside the “ randomNumbers” variable which we declared. This number will not be known by the player because he or she needs to guess it. That ’ s the point of the game.

Next up, the player needs to guess the hidden number. This guess will then be stored inside a new variable called “ myGuess.” In order to check whether the guess is equal to the number, we are using a while loop with the “ not equal to” operator. We do this because if the player gets lucky and guesses the number correctly on the first attempt, the loop simply doesn ’ t finish executing because there ’ s no need.

Next, if the player guesses the wrong number, we have two if statements that check whether the guess is a higher value than the hidden number or a lower one. An appropriate message is then printed for the player in each case. In either scenario, the player receives another chance to make the right guess. Finally, in the end, if the user guessed the number correctly, the program declares victory by printing a message and then the program stops running.

To make the game more interesting, you can challenge yourself to modify the random number generator to include different values. You can also add a statement that enables the game to print the score to see how many times the player tried to guess the number. Also, since the game ends when the player guesses correctly, you could write the main loop so that the player can choose to restart the game instead of automatically quitting. Have fun, and don ’ t be afraid to try anything.

Activity 3: Choose a Card

Card games are always fun, and they also rely on random elements to some degree. No matter what the card game is, chances of having multiple identical games are quite small. This means you won ’ t get bored any time soon. With what we discussed so far about Python programming, we can create a card game. It might not look good unless you have an artistic friend to draw everything for you, but you could still create the graphics with the help of the Turtle module, as we did for other projects. This will require some patience though. In any case, we can create a card game even without graphics by simply generating the name of each card. Instead of seeing a virtual card, we will see the name “ four of spades,” or “ queen of hearts.”

Before we continue, you should take note that this project is your challenge. You have learned everything you need to write such a game, and we already created two other fairly similar projects. So, this time, you are almost entirely on your own. As usual, start with a pen and paper and figure

everything out in logical order. Worry about the code afterward. However, to help you out a little, we are going to brainstorm together just to give you some ideas.

One of the simplest card games we can create involves a game with two players that battle each other to see who draws the card with the highest value. Each player will randomly pull a card from the deck, and the one who has the higher card will win. It is a simple yet fun game due to the element of randomness.

Since we won ’ t be using any graphics, we will have to create our deck of cards some other way. We are going to set them all up as a list of strings as we will be using their names instead. Next, we need to give the players the ability to randomly pull a card from the deck. This means that we are going to use the random module once again, and we will add a choice function that randomly distributes cards to the players. Finally, we need a way to compare the two cards that are drawn by the two players. As you probably guessed, this is a case for comparison operators.

That is pretty much all it takes to create a card game. You can add more features, or remove some, if you aren ’ t interested in them. Whatever you do, design the game on paper so that you know your goals beforehand. Then, work on those goals by writing one line of code at a time. This way you will write your game in no time and you will be able to fix whatever problems you encounter fairly quickly.

Activity 4: Random Number Generators: Mimic a Coin Flip

Now, before we can write a coin flip/toss program, we need a way to produce random numbers. That may sound like something difficult if not impossible. After all, how in the world do you make something random?

Well, the good news is that there is a way for you to generate random numbers with the help of Python programming. As you might have guessed, it requires some serious math. But don ’ t worry. Someone else has done the math for you.

You don ’ t have to come up with the mathematical formula or the algorithm that will create random numbers. It has all been done for you. In fact, there are several ways to produce random numbers in Python.

We will introduce you to one of these Python language constructs so you can create the coin toss game. To do that, we will need to use a function called choice( ). This is only one of several functions in Python that are used to generate random numbers.

Not only do these functions produce random numbers, but they can also manipulate the randomness of the numbers being created. In a way, they give you some degree of control, so you can decide which set of numbers can be produced.

Note that these functions are used in many games, apps, lottery programs, and many other applications where an element of randomness is needed. They are pretty useful actually.

The choice( ) Function

As mentioned earlier, for this coin flip or coin toss game we will use the choice( ) function. So, what is it?

Remember, it is spelled with a small letter “ c” at the beginning. The choice( ) function will output only one random number. That makes things easier for now since all we want is something that will produce either one of two results.

We can use the choice( ) function to randomly generate either the number 1 or 0—well, we can also choose 1 or 2, too. It ’ s all up to you which two numbers you will choose. The next question is how does this function work?

Here is an example of how the choice( ) function will look like in a Python program:

```
print (random.choice([5, 4, 3, 2, 1]))
```

You are already familiar with the print( ) function. Next, as you can see from the sample above, you use the choice( ) function by using the following line of code:

```
random.choice( )
```

From the said example above, you will also notice a set of numbers enclosed within a pair of square brackets, which are the following:

```
[5, 4, 3, 2, 1]
```

This function will choose any of the numbers inside the set contained within the square brackets. Note that only the numbers in this container will be used. That is the control that will be given to you when using this random number generator.

What we have below is called the syntax of a statement. In programming terms, the syntax is the proper arrangement of terms in a programming language so that it can be interpreted correctly (or translated correctly into a language that can be understood by a computer).

The following is the official syntax of the choice( ) function:

random.choice([sequence])

Here are the parts of this function:

- random.choice – This is the function call or the right way to make use of this function. You need to add the word “ random.” (followed by a dot) before the word “ choice.” So you might be thinking what is this “ random ” part of the statement? Well, that is called the module (we ’ ll talk about modules in a minute). What this part of the code is telling us is that “ choice” is part of “ random ” or contained inside “ random.”

- [sequence] – This part will contain a sequence of numbers or in the case of our coin toss program it will contain either of two words (i.e. heads or tails). This part of the choice( ) function is the list of items from which the output will be selected.

choice( ) is a useful function if you want to specify exactly which numbers will be included in the selection. There are downsides, of course. What if you want to choose any number ranging from 1 to 500,000?

Writing all those numbers in your Python code will become way too long if you do it that way. Don ’ t worry. Other functions can handle such a task. For now, let ’ s just concentrate on using the choice( ) function since we want a limited set of numbers to choose from.

Open your Python console and enter the following lines code:

```
>>> import random

>>> print(random.choice(["heads","tails"]))
```

Import Statement

We used the following statement:

```
>>> import keyword
```

And now you have:

```
>>> import random
```

The reserved word “ import ” is a statement that is used to import/make use of/bring in predefined codes. Don ’ t let that technical jargon scare you. This statement makes use of the import system in Python programming.

You remember that it was explained earlier that other people have written the algorithms and the Python code for a lot of tasks that you will need in programming. In this case, when you need a program that will generate random numbers, other programmers have already done the job for you.

All you need to do is to use their code. That means someone else already wrote the code for the choice( ) function that we were discussing earlier. Now, for you to use that function, you need to import it from the code that they wrote into your own code.

That eliminates the need to write what they already wrote. All you need to do is to import it. In this case, you will import something called “ random.”

In Python programming, “ random ” is something called a module. Think of a module as a collection of programming code that has already been made for you to use. You have now learned two modules in this programming language—random and keyword.

You can ’ t use the choice( ) function without importing the random module first. That is why you start with an import statement first and then use the choice( ) function.

Moving forward, notice that when you press enter after this line of code:

```
>>> print(random.choice(["heads","tails"]))
```

The system will display either heads or tails. Press the Up arrow key to display that command again. Pressing the Up arrow key on the command console of Python will display the last command that you entered. That way you don ’ t have to retype everything over and over again. This only works on the command-line console.

Notice that the pattern produced is random. There is no specified number of times the words “ heads” or “ tails” will be selected.

Activity 5: Coin Flip Game Algorithm

Now we are ready to create the algorithm for the coin flip game. Here it is:

1. Greet the player and mention the name of the game.

2. Explain the rules of this game: a virtual coin will be tossed. There will be no graphics involved. Just an imaginary or virtual coin toss for now.

3. The player will guess whether the coin will show heads or tails.

4. Flip or toss the coin.

5. The player that guesses the side of the coin correctly gets 1 point, the player who doesn ’ t guess correctly has 1 point deducted. The player who gets 3 points first, wins.

CHAPTER 2:

Making Choices and Decisions

What You Will Learn

- If statement
- Indentation
- If-else statements
- For-in loops
- While loops
- Input texts

If Statement

```
if  test-expression:
statements
```

- The above graphics are too basic. We'd like to have something more elaborate. [2]

The if statement executes the statement only if a specified condition is true; it does not execute any statement if the condition is false.

Example

```
a = 200
b = 100
if a > b:
    print ("a is greater than b.")
```

Output:

a is greater than b.

Explanation

a > b is a test expression; it tests whether 200 > 100. If it returns true, it will execute the code print( ), if it returns false, it will not execute the code print( ).

If-else Statement

```
if  test-expression:
  statements      # run when text-expression is true
else:
  statements      # run when text-expression is false
```

Example

```
a = 100

b = 200

if a > b:

    print ("a is greater than b.")

else:

    print ("a is less than b")
```

Output:

a is less than b.

Explanation

a > b is a test expression; it tests whether 100 > 200. If it returns true, it will execute the code print(‘ a is greater than b ’). If it returns false, it will execute the code print(‘ a is less than b ’).

Indentation

In Python, indentation is used to mark a block of code. To indicate a block of code, you should indent each line of the block of code by four spaces, which is the typical amount of indentation in Python.

Example

- testo da rivedere, carattere deve essere animato [3]

```
a = 100
```

```
b = 200
if a > b:
    print ("a is greater than b.")      # indent four spaces
else:
    print ("a is less than b")       # indent four spaces
```

The print() statements are indented four spaces. Correct!

Note:

```
if a > b:
print ("a is greater than b.")      # error !
else:
print ("a is less than b")     # error !
```

The print() expression is not indented, so errors occur!

If-elif Statement

```
if   test-expression:
    statements      # run when this text-expression is true
elif   test-expression:
    statements      # run when this text-expression is true
else:
    statements     # run when all text-expressions are false
```

Example

```
Num = 200
if num < 100:
    print ("num is less than 100")
elif 100 < num < 150:
    print ("num is between 100 and 150")
else:
    print ("num is greater than 150")
```

Output:

```
num is greater than 150
```

Explanation

elif is short for else if.

For-In Loops

The for-in loop repeats a given block of code a specified number of times.

```
for <variable> in <sequence> :
<statements>
```

A variable stores the value of each item.

A sequence may be a string, a collection, or a range( ) which implies the number of times of loop.

Example

```
for str in 'Good':
   print ('Current Character :', str)
```

Output:

Current Character: G

Current Character: o

Current Character: o

Current Character: d

Explanation

for str in 'Good' loops four times because ‘ Good ’ has 4 characters. str stores the value of each character.

For Variable In Range( )

A for variable in range( ) can generate a sequence number.

```
for var in range( n)
for var in range(n1, n2)
```

range(n) generates a sequence from 0 to n-1.

range(n1, n2) generates a sequence from n1 to n2-1.

(1)

Example

```
for var in range(6):
 print (var)
```

Output: 0,1,2,3,4,5.

(2)

```
for num in range(3,10) :
print (num)
```

Output: 3,4,5,6,7,8,9.

Explanation

A for variable in range( ) can generate a sequence number.

While Loops

While loops are used repeatedly to execute blocks of code.

```
while <test-expression> :
<statement>
```

<test-expression> looks like a < 100, b! = 200. c==d [4] , etc.

Example

```
n = 0
while n <  9:
    print (n)
    n = n + 1
```

Output:

012345678

Explanation

n < 9 tests the n value. If n is less than 9, while-loops will execute the print (n) and will continue to run the next loop. Until n is greater than or equal to 9, while-loops will terminate the loop.

n = n + 1 adds one to n in each loop.

Continue

```
continue
```

The continue keyword can skip the next command and continue the next iteration of the loop.

Example

```
num=0
while num<10:
num = num + 1
if num==5:
    continue
print (num)
```

Output:

1234678910

Explanation

Note that the output has the number 5.

The statement, if num==5: continue, skips the next command, print (num), when num is 5, and then continues the next while loop.

Break Statement

```
break
```

The keyword break is used to stop the running of a loop according to the condition.

Example

```
num=0
while num<10:
if num==5:
```

```
    break
num=num+1
print (num)
```

Output:

5

Explanation

The statement, if num==5: break, will run the break command if num is 5; the break statement will exit from the while loop, and will run print (num).

Input Texts (1)

Sometimes users need to input some text via the keyboard.

```
variable = input(“prompt”)
```

Note : Please use double quotation marks to enclose your input (in some Python versions).

Example

```
name = input("Please input your name: ")
print ("Your name is: " + name )
age = input("Please input your age: ")
print ("Your age is: " + age )
```

Output:

Please input your name: “Jack”

Jack

Please input your age: “16”

16

Explanation

The input( ) function can accept the data from the user ’ s keyboard input.

Input Texts (2)

Sometimes users need to input some text by keyboard.

```
variable = raw_input(“prompt”)
```

Example

```
name = raw_input("Please input your name: ")
print ("Your name is: " + name )
age = raw_input("Please input your age: ")
print ("Your age is: " + age )
```

Output:

Please input your name: Jack

Jack

Please input your age: 16

16

Explanation

The statement raw_input( ) can accept the data from the user ’ s keyboard input.

Note : The statement raw_input() is no longer used nowadays, but you need to know a little bit about Python history.

Pass Statement

The pass statement is a null operation; it means ‘ does nothing. ’

It is also very useful as a temporary placeholder for future code that needs to be inserted later.

Example

```
condition = True
if condition:
    print ('The condition is very good!')
elif True:
    pass       # insert code later
else:

pass             # insert code later
```

Output:

The condition is very good!

Explanation

The pass statement is just a temporary placeholder.

Exercise: Traffic Light

If-elif-else statement

Please go to **Start** > **Programs** > **Python3.5** > **IDLE** (Python GUI).

Write the following code into IDLE editor:

```
trafficLight = input("Please input traffic light -- red, green or yellow: ")
if trafficLight == "red":
    print ("The traffic light is " + trafficLight)
elif trafficLight == "green":
    print ("The traffic light is " + trafficLight)
else:
    print ("The traffic light is " + trafficLight)
```

Save the file, and run the program by pressing **F5** .

(**Run** > **Run Module**).

Output:

The traffic light is green.

Explanation

```
if   test-expression:
   statements    # runs when this text-expression is true
elif   test-expression:
   statements    # runs when this text-expression is true
else:
   statements    # runs when all text-expressions are false
```

Note : Please use double quotation marks to enclose your input (in some Python versions).The statement raw_input() is no longer used nowadays, but you need to know a little bit about Python history.

Summary

The if statement executes only if a specified condition is true; it does not execute any statement if the condition is false.

In Python, indentation is used to mark a block of code. To indicate a block of code, you should indent each line of the block of code by four spaces, which is the typical amount of indentation in Python.

The for-in loop repeats a given block of code a specified number of times.

The for variable in range() can generate a sequence number.

While loops are used repeatedly to execute blocks of code.

The keyword, continue, can skip the next command and continue to the next iteration of the loop.

The keyword, break, is used to stop the running of a loop according to the condition.

Sometimes users need to input some text via keyboard.

variable = input(“ prompt ”)

The pass statement is a null operation; it means ‘ does nothing. ’

The pass statement is very useful to work as a temporary placeholder for future code that needs to be inserted later.

If/Then Games

These games are based on conditional statements.

Quiz

1. What is an iteration?

   A program test

   A decision

   The repetition of certain steps

2. Which statements will use an iteration?

   if and while

   for and while

   if and else

3. Is the following statement true? A while loop will iterate until told otherwise.

True

False

4. Which symbol is translated as ‘ equal to ’ in Python?

   =

   !=

   ==

5. Is the statement 5 >= 5 true?

   Yes

   No

6. Which loop is used to repeat a statement a specific number of times?

   Indentation

   For-loop

   While-loop

7. Which data type can only be true or false?

   Integer

   Boolean

   Float

8. Which symbol needs to be placed at the end of a conditional statement?

   ;

:

…

9. Is the following statement correct? Repeat loops will repeat if the condition is true.

   True

   False

10. Which symbol means greater than or equal to?

    >>

    =>

    >=

Activity 6: There ’ s A Loop For That!

Let’s say we want to output a greeting to our friends and tell them what our favorite dessert is: print("Hi! My name is Adrienne. My favorite dessert is ice cream.")

This works if your name happens to be Adrienne. Oh, and if your favorite dessert also happens to be ice cream. What if it was chocolate? Or cookies? Or cake? What if you had a different name? How would you change the print() function to output your name and favorite dessert?

You could write a print() function for each combination. It would look like this: print("Hi! My name is Adrienne. My favorite dessert is ice cream.")

print("Hi! My name is Mario. My favorite dessert is creme brulee.")

print("Hi! My name is Neo. My favorite dessert is cake.")

That’s a lot of work, though. If you look at the three print() functions, do you notice any kind of pattern? All of them are exactly the same, except for the name and the dessert! This would be a great case to use an f-string and some loops!

What to Do

Write a loop that outputs the name of a person and their favorite dessert using the two lists below. The order of favorite desserts matches the order of the people who like them, so don’t worry about that. Use an f-string to print out the message.

people = ['Mario', 'Peach', 'Luigi', 'Daisy', 'Toad', 'Yoshi']

desserts = ['Star Pudding', 'Peach Pie', 'Popsicles', 'Honey Cake', 'Cookies', 'Jelly Beans']

Expected Result

Hi! My name is Mario. My favorite dessert is Star Pudding.

Hi! My name is Peach. My favorite dessert is Peach Pie.

. . . (and continued for the rest of the list)

Activity 7: Loop De Loop, Which Hula Hoop Loop?

Nacho, the cat, is walking through the neighborhood when he sees some hula hoops by a playground. He notices that there are a few placed together

by the swings and another few propped up by the basketball court. Nacho gets the idea to invite his friends to come and play.

What to Do

Using your knowledge of loops, write either a for-loop or while loop to cycle through Nacho's cat friends and send them to a specific set of hula hoops.

Nacho has requested that his more athletic or younger friends be sent to the hula hoops by the swings, since those hula hoops are more difficult to jump through while the swings are in motion. If the cat friends are older or less athletic, they should go to the hula hoops propped up by the basketball court, as they are easier to jump through.

Here's some code to help you get started:

nachos_friends = ['athletic', 'not athletic', 'older', 'athletic', 'younger', 'athletic', 'not athletic', 'older', 'athletic', 'older', 'athletic']

hula_hoops_by_swings = 0

hula_hoops_by_basketball_court = 0

As you cycle through Nacho's friends, determine which group they belong to. Then, add another count to that group to keep track of how many cats are in each. Finally, print how many cats are at the hula hoops by the swings and how many cats are at the hula hoops by the basketball court.

Sample Expected Output

Cats at Hula Hoops by Swings: 6

Cats at Hula Hoops by Basketball Court: 5

Activity 8: Iffy Legs

Imagine that we worked at a zoo and needed to organize the animals based on the number of legs they have. After organizing them, we also count the total number of animals we have in each group. How would we do this? In real life, we would probably take each animal one by one, look at the number of legs it has, and then put it in an area we've marked as animals having a specific number of legs. After sorting, we could then count the total number of animals in each area.

Let's try writing a small program to help us sort our animals instead. Sound good?

What to Do

To start, let's create some variables for the different groups of animal legs and assign a starting count of 0 (since we haven't sorted any yet!):

has_zero_legs = 0

has_two_legs = 0

has_four_legs = 0

Cool! For now, we know that these are the three types of groups that an animal from our zoo can be placed into: a group for animals with no legs, another group for animals with two legs, and a third group for animals that have four legs. Here's some information about the various animals and their number of legs:

moose = 4

snake = 0

penguin = 2

lion = 4

monkey = 2

dolphin = 0

bear = 2

elephant = 4

giraffe = 4

koala = 2

shark = 0

kangaroo = 2

komodo_dragon = 4

Create a list with the animal leg information, use a loop to iterate through them all, and keep count of which group we add each animal to. Print out the total number of animals in each group.

Sample Expected Output

Animals with no legs: x

Animals with two legs: y

Animals with four legs: z

Activity 9: Password-protected Secret Message

There are times when we need to share secrets with our friends. Wouldn’t it be cool to write a small program that only allows users to see the contents if they provide the right password? Well, we can do that using while loops!

What to Do

Create a new Python file called ‘secret-message’ and save it. In your program, create three variables: one for a password, one for a user’s guess, and another for your secret message. I started some below for some inspiration:

password = 'cupcakes'

guess = ''

secret_message = 'Tomorrow, I will bring cookies for me and you to

share at lunch!'

Now, create a while loop. Our while loop will be checking the password a person tries through the guess variable. Our program should continue to ask for a password if the person’s guess is incorrect!

To make sure that only those with the right password can view your message, have your while loop check to see that your password variable is not equal to the guess variable. If it isn’t, that means the person using your

program has not entered either the right password or any input at all. In that case, continue the while loop and use a print() function to ask the user for a password. Also, within the while loop, keep re-assigning your guess variable to whatever the user types into your program like this: guess = input()

You should only stop your while loop once the user enters the correct password. Once that happens, use another print() function to show your secret message!

Save your program, then run it. You should see it continue to ask you for the right password and only show you the secret message once you provide it!

Sample Expected Output [5]

Activity 10: Guess-the-Number Game

Using Python's built-in random module (see here) and while loops, build a simple number guessing game! The computer will pick a random number and assign it to a variable, while you take turns trying to guess that number. Let's code!

What to Do

Create a new Python file called 'guess-the-number-game' and save it. In your program, import the random module (by typing 'import random' as

shown) and create two variables: one to store the number the computer randomly picks, and one for the number of guesses you will allow in your game:

```
import random

# selects a random number between 1 and 100

number = random.randint(1,100)

number_of_guesses = 0
```

Remember, you can change the range of the random number picked. This is your game!

Now, create a while loop that checks your number_of_guesses variable to see if it's less than the maximum number of guesses you will allow for your game.

<Write some code here>

If it is, that means you still have guesses remaining. In that case, continue the while loop and write a print() function to ask for a number between the range you have selected.

<Write some code here>

Also, within the while loop, assign a new guess variable to whatever you type into your program, like this: guess = input()

By default, anything you enter into your shell is a string type. To make sure you can check your number of guesses correctly, transform your guess variable into an int type by using Python's built-in int() function: guess = int(guess)

Now that you've taken another guess, you should increase your number_of_guesses variable as well.

<Write some code here>

Finally, you need to check that the guess you've input is equal to the number the computer chose at the beginning of your game. Use an if statement for this, and break out of (stop) the loop if it is. To do this, simply type the reserved code keyword, break.

<Write some code here>

You should only stop your while loop once you either guess the correct number or when you've run out of chances to guess. In either case, feel free to write a print() function that tells you it's game over or that you've correctly guessed the right number!

Save your program, then run it. You should be able to play your secret number guessing game!

Sample Expected Output [6]

CHAPTER 3:

Turtle Graphics

What You Will Learn

- Bringing up the screen
- Turtle setup
- Drawing a rectangle, circle, and star

Turtle graphics is a popular module for programming for kids (and adults).

You already know that to use turtle, you need to ‘ import ’ it. When you import turtle and run it, nothing happens. This is okay.

The next thing you should know is how to bring up the turtle screen, and edit it to look the way you want it to. To do this, we would discuss a few functions of the turtle module.

Bringing up the Screen

To bring up the screen, you need to use the turtle. turTle () [7] function (note that the second T is capital [8]).

You need to assign turTle () to a variable to make editing the screen easy. Here we are assigning it to the variable sr.

Example

>>> import turtle

>>> sr = turtle. Turtle ()

R: You would see a screen come up with an arrow in the center.

Congrats! You have just opened up the turtle screen.

You can also use the turtle Screen () function. (note that the S in ‘ Screen ’ is in capital letters)

>>> import turtle

>>> sr= turtle. Screen ()

Next, we change the background of the screen.

Changing the Background Of The Screen

To change the background of the screen, we use the bgcolor () function.

Example

>>> import turtle

>>> sr= turtle. Screen ()

>>> sr.bgcolor (‘ red ’)

This will cause the turtle screen to turn red.

You can change the color by typing in any color you want.

We would use white as the background color as we go on.

Naming our Screen

We could name our turtle screen by using the function .title().

Example

>>> import turtle

```
>>> sr= turtle.Screen ()

>>> sr.bgcolor ( ‘ white ’ )

>>> sr.title ( ‘ learning turtle ’ )
```

This will cause the turtle screen to be named ‘ turtle screen. ’ You could name your turtle screen anything.

Turtle Setup

For the turtle setup, we will specify the size of the turtle. To do this, we use the .setup function.

Example

```
>>> import turtle

>>> sr= turtle.Screen ()

>>> sr.bgcolor ( ‘ white ’ )

>>> sr.title ( ‘ learning turtle ’ )

>>> sr.setup (width = 800, height = 500)
```

This makes the turtle screen 800 points wide and 500 points high.

This has a lot of significance, which will become very obvious when we begin to draw on the turtle screen.

Next, we move the turtle around.

Moving The Turtle Around

To move the turtle around, we can use the following functions:

turtle.forward (distance)

turtle.backward (distance)

turtle.left (angle)

turtle.right (angle)

turtle.forward

This is used to move the turtle forward by a specific number of points or pixels.

Example

>>> import turtle

>>> sr= turtle.Screen ()

>>> sr.bgcolor (‘ white ’)

>>> sr.title (‘ learning turtle ’)

>>> sr.setup (width = 800, height = 500)

>>> turtle.forward (150)

R: This will move the turtle forward by 150 points to the direction the arrow is facing.

We can accomplish the same by using:

>>> turtle.fd (150)

turtle.backward (distance)

This moves the turtle in the opposite direction.

Example

>>> import turtle

>>> sr= turtle.Screen ()

>>> sr.bgcolor (‘ white ’)

>>> sr.title (‘ learning turtle ’)

>>> sr.setup (width = 800, height = 500)

>>> turtle.backward (150)

R: This moves the turtle in the opposite direction.

We end up with the same result by using the turtle.bk () or turtle.back () function.

turtle.left ()

This moves the turtle to the left, across the angle specified.

Example

>>> import turtle

>>> sr= turtle.Screen ()

>>> sr.bgcolor (‘ white ’)

>>> sr.title (‘ learning turtle ’)

>>> sr.setup (width = 800, height = 500)

>>> turtle.fd (150)

>>>turtle.left (60)

>>> turtle.fd (150)

R: You should have straight lines at right angles to each to other.

turtle.right ()

The turtle.right function moves the turtle in the direction opposite to that which the turtle.left function moved it.

Example

```
>>> import turtle
>>> sr= turtle.Screen ()
>>> sr.bgcolor ( ‘ white ’ )
>>> sr.title ( ‘ learning turtle ’ )
>>> sr.setup (width = 800, height = 500)
>>> turtle.fd (150)
>>>turtle.right (60)
>>> turtle.fd (150)
```

Pen Up and Pen Down

These turtle functions help to command the turtle pen to show or not show some of the lines that it has drawn.

Pen up means that no line will show as the turtle moves, while pen down means the lines will show.

The codes for pen up are turtle.penup(), turtle.pu(), and turtle.up(). You may use any one of them.

The codes for pen down are turtle.pendown(), turtle.pd(), and turtle.down().

Example

```
>>> import turtle
```

```
>>> sr= turtle.Screen ()

>>> sr.bgcolor ( ‘ white ’ )

>>> sr.title ( ‘ learning turtle ’ )

>>> sr.setup (width = 800, height = 500)

>>> turtle.bk (150)

>>>turtle.left (90)

>>> turtle.bk (150)

>>> turtle.left (90)

>>> turtle.pu()

>>> turtle.bk(150)

>>> turtle.pd()

>>> turtle.left(90)

>>> turtle.bk(150)
```

The code above draws a square on the Python screen we created earlier. The square has its bottom missing.

This is because the pen up function was applied when the turtle was to draw the bottom of the square. Hence the line did not show.

You can try to apply the pen up function on your own, on any line, and also with lines drawn at various angles.

Pen Speed

You may have noticed that the pen moves a little slower whenever it is drawing the square. You can make it move faster.

The function for that is the pen speed function, which is applied by using the code turtle.speed().

There are specific speeds at which the turtle can move:

“ fastest ”: 0

“ fast”: 10

“ normal”: 6

“ slow”: 3

“ slowest” : 1

If you type in a number greater than 10 or smaller than 0.5, the speed is returned to 0.

Speeds ranging from 1 to 10 makes the pen become increasingly fast, with 10 being the fastest.

A speed of 0 means the turtle will instantly do what you ask; thus, you will not see the turtle drawing a square; you will just see the square on your turtle screen. [9]

You can play around with different speeds of the turtle.

Pen Width

The pen width controls the thickness of the lines drawn by the turtle. The bigger the turtle pen is, the bigger the lines are.

The code for changing the pen size is turtle.width or turtle.pensize.

You can increase the pen size from 0 (the thinnest) to as high as you want.

Note: From here on, we will assign our turtle the variable, t.

Example

```
>>> t= turtle
```

Drawing a Rectangle

Drawing a rectangle is comparable to drawing a square.

Example

```
>>> import turtle
>>> sr = turtle.Screen ()
>>> sr.bgcolor ( ‘ white ’ )
>>> sr.title ( ‘ learning turtle ’ )
>>> sr.setup (width = 800, height = 500)
>>> t= turtle
>>> t.speed(8)
>>> t.width(3)
>>> t.bk (200)
>>> t.left (90)
>>> t.bk (100)
>>> t.left (90)
>>> t.bk (200)
>>> t.left (90)
>>> t.bk (100)
```

This block of code tells the turtle to move backwards by 200 points, then turn 90 degrees, then to draw a line going down by 100 points, then turn another 90 degrees, then to move another 200 points, then to turn another 90 degrees, this time upwards, then draw another line 100 points in distance. This gives us a rectangle.

You can play with the code, thereby changing the length of your lines.

Note, that I am using a turtle speed of 8 for my drawing. You can use whatever pace you fancy.

I am also using a pen width of 3 for my rectangle.

Circle

To draw a circle, you use the function turtle.circle (). In the parenthesis, you input the radius of the circle you want to draw.

Example

```
>>> t= turtle

>>> t.speed(8)

>>> t.width(3)

>>> t.circle(50)
```

Stars

Remember the loop we used for drawing a square simply? We can modify the loop to give a star!

Example

```
>>> t= turtle

>>> t.speed(8)
```

```
>>> t.width(3)
>>> for i in range(1, 9):
t.forward(100)
t.left(225)
```

This code produces an eight-point star.

Unlike with the square where we looped 4 times with range (1,5), we loop this eight times with range(1, 9).

Also, we tell the turtle to move forward 100 points after each loop. We also make the turtle turn 225 degrees to the left, instead of 90 degrees to the left.

We can have a lot of fun with this star-drawing loop.

Examples

```
>>> t= turtle
>>> t.speed(8)
>>> t.width(3)
>>> for i in range(1, 9):
t.forward(100)
t.left(125)
>>> t= turtle
>>> t.speed(8)
>>> t.width(3)
```

>>> for i in range(1, 9):

t.forward(100)

t.left(145)

You can just keep changing the size of the angles and the distance the turtle will travel each time.

Activity 11: Let ’ s Draw A Star!

Now that we ’ ve learned about the turtle module and what it can do, let ’ s draw a star! We ’ ll create a small program that can do this for us.

What to Do

1. Create a Python file and save it with the name ‘ star. ’

2. Import the turtle module:

import turtle

3. Set the color mode to 255: turtle.colormode(255)

4. Create a pen variable and assign a turtle object to it. This makes it easier to understand that we ’ re drawing something, instead of dealing with a turtle!

pen = turtle.Turtle()

5. Choose some RGB values for a shade of yellow you like, or choose a different color. For this activity, I ’ m using a bright yellow: pen.color(255, 215, 0)

6. Let ’ s also change the pen size so our star is nice and visible! You can choose whatever size you want: pen.pensize(5)

7. Now let ’ s hide the shape so we can see our star a bit better: pen.ht()

8. Let ’ s draw! We ’ ll move our pen forward by 100 points, then turn our pen 144 degrees to the right. We ’ ll do this five times to create a five-pointed star. The code will look like this:

pen.forward(100)

pen.right(144)

pen.forward(100)

pen.right(144)

pen.forward(100)

pen.right(144)

pen.forward(100)

pen.right(144)

pen.right(144)

9. Finished! Save your code by pressing the **CTRL** and **S** keys together. Then press the **F5** key to see your star drawn in front of you! Bonus: Can you optimize the code above to use a for-loop instead?

Sample Expected Output [10]

Activity 12: Fortune-teller

What to Do

Create a new file called ‘ fortune-teller ’ and save it. Within it, import the turtle module and the random module: import turtle

import random

Create a new copy of the turtle object and call it ‘ pointer ’ ; you can leave it as the default arrow shape because that ’ s what we ’ ll need! Also, set its size: pointer = turtle.Turtle()

pointer.turtlesize(3, 3, 2)

Create another copy of the turtle object and call it a ‘ pen. ’ We ’ ll use this to create our fortune-teller board: pen = turtle.Turtle()

Finally, create a variable to hold your spinner amount, and use the random module to pick a random number: spin_amount = random.randint(1,360)

Now, lift your pen so that it doesn ’ t start drawing. We only want it to draw on the spots we tell it to: pen.penup()

Use the goto() function to move your pen to the four sides of your screen. On each side, write some answers that your fortune-teller pointer can land on. These can be simple “ Yes” or “ No” answers or silly ones like “ Never in a million years!” To help you out, I ’ ve provided the coordinates for the four sides of the screen:

```
# right side
pen.goto(200, 0)
pen.pendown()
pen.write("Yes!", font=('Open Sans', 30))
pen.penup()
# left side
pen.goto(-400, 0)
pen.pendown()
pen.write("Absolutely not!", font=('Open Sans', 30))
pen.penup()
# top side
pen.goto(-100, 300)
pen.pendown()
pen.write("Uhh, maybe?", font=('Open Sans', 30))
pen.penup()
# bottom side
pen.goto(0, -200)
pen.pendown()
pen.write("Yes, but after 50 years!", font=('Open Sans', 30))
pen.ht()
```

Finally, pass your spin_amount variable to your pointer ’ s left() or right() function to make it spin a certain direction.

Save your file. Now, every time you run your fortune-teller program, you will get a random answer to your questions!

Activity 13: Rainbow Turtles!

What to Do

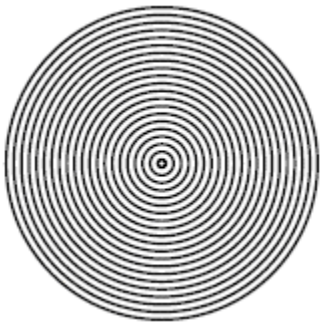
Using your knowledge of the stamp() function, create a program that stamps a turtle in each color of the rainbow. Make sure the turtles are in the same order as the colors of the rainbow!

Helpful Hints

Use a for-loop to iterate through the steps you need to repeat. This includes changing the color of the turtle, and stamping and moving the turtle a certain amount.

Sample Expected Output

Activity 14: Circleception



Create a circle within a circle within a circle . . .

What to Do

Using the circle() function and your knowledge of filling in shapes with colors, draw one big circle, and fill it with a color. Then, draw a medium-sized circle and fill it with a different color. At this point, make sure that you can still see the medium-sized circle and that it ’ s contained within the big circle. Finally, draw one smaller circle, fill it with a different color, and make sure it is contained within the two larger circles.

Helpful Hints

Again, for-loops will be your best friend in creating this drawing, as a lot of steps are repeated! First, keep track of each step you take to draw a circle and fill it with color. Once you have found the repeated steps, try moving it into a for-loop. Then, figure out which parts you need to change to create different-sized circles and change colors.

Activity 15: Tooga ’ s House

Now that we know how to use the turtle module ’ s built-in functions, let ’ s create a proper home for Tooga!

What to Do

Create a new turtle named Tooga and a new pen to build Tooga ’ s home:

tooga = turtle.Turtle()

pen = turtle.Turtle()

Using the penup() and pendown() functions, and the changing colors and pen sizes, draw some shapes to create a house for Tooga. Make sure Tooga is actually inside the house you created for him!

This can be a simple square around Tooga with a triangle above the square to represent the roof. Get creative with the colors and pen sizes. Tooga would surely appreciate a non-boring, colorful home!

Helpful Hints

Use the penup() and pendown() functions to lift and drop the pen when you need to draw and not draw, respectively. This will make sure you don ’ t draw on Tooga!

Sample Expected Output

CHAPTER 4:

Variables in Python

What You Will Learn

- Data types and variables
- Variable input exercise
- Declaring variables
- String operations
- Casting

Data Types and Variables

A variable is a programming construct that represents data, such as a number or a word, that can be modified or used later. In math, you have encountered variables in courses such as Algebra I, where x represents a value and can be plugged into an algebraic equation. There are different forms of data we encounter in the world, such as sentences composed of multiple words; decimal numbers that represent temperature; whole numbers that represent shirt sizes; and binary conditions, such as whether it is day or night. All these scenarios contain different data types or classifications of data.

Variables are essential to any decent-sized program, to keep track of user input, data that is being processed, program features, and a plethora of other factors. For example, in many games, like Angry Birds, there is a variable

that represents the player ’ s score, and this changes its value according to whether the player gains points or not. In applications that require an account login, the username and password entered by the user are often stored in variables. Variables are vital constructs to any programming language and software development project. Trying to code without them would be like trying to ride a bike without wheels: an utter waste of time.

Variable Input Exercise

Let us try writing a small program that uses a variable to greet the user with their username in the Python console. Create a new Python file (Click **File** > **New File** in IDLE), then save it as ‘ input_exercise.py. ’ Add the following code:

```
# take input from the user and print welcome #message
name = input ( “What is your name? ” )
print ( “Welcome ” + name)
```

Run the program (**Run** > **Run Module**), and notice how it prompts you to enter input. Enter your name and press **Return** . Notice how the welcome message is printed in the console.

```
What is your name? Cole
Welcome, Cole
```

This code uses the built-in Python input function to prompt the user to enter a value. After they enter their name, this value is assumed in place of the variable in the print statement on the following line. Without the use of

variables, we would not be able to take user input, because we would either have to hard code a name in the print statement, or we would need to call the input function inside the print statement, which would be messy.

Data Types

To represent the various types of data, it is important you fully understand data types, which is the way data is classified in programming. Not all objects are created equal. The classification of data types is like the different states of matter: each has its own qualities and properties; e.g. a liquid can change shape to fit the container it is in, but a solid retains the same shape. Here are the main data types you should be aware of:

String: It represents words or a series of characters encapsulated within a set of double or single quotations. Examples: a sentence is a string, jar, word, 4534%$@%.

```
# various variable declarations of type String
message = “A message is considered a string”
name = “John Doe”
area = “32” # string since it is in quotes
secret_code = “%%$5&*(&((”
```

Integer : It represents a whole number, and can be either negative or positive. Examples: 5, -43563, 25466, 1, 0, 12.

```
# various variables that are Integers
# Notice there are no quotations
```

```
# Quotes around a number make it a string
student_count = 31
building_number = 244
num = 10
```

Float : It represents a decimal number, and can also be either negative or positive. Examples: 3.14, 22.0, 2.3333333.

```
# various variables that are Floats
percentage_complete = 89.53
temperature = 72.0   # still considered a float since it is written with a decimal
example = 657.3432445423
```

Boolean : It represents a condition that only assumes two values: True or False. (In assigning a Boolean value, the first letters of the words True and False must be capitalized.)

```
# various variables that are Boolean values
working = True
passed test = False
lights_on = True
```

Variables in Python are dynamic, meaning they can change data types. While this can be advantageous, some drawbacks could cause your programs to perform an unintended action. For example, you may make a variable an integer, but then try to print it to the screen as a string, thus causing your program to crash. Dynamic typing in Python is very different from other languages, such as Java, where you must set the type when declaring the variable, in what is known as static typing, after which it cannot be changed. For example, if you create an integer in Java called ‘ score, ’ and decide you want to make it into a float point value, you cannot simply assign your old variable a decimal value. Instead, you must create an entirely new variable to store this value.

Declaring Variables

Initializing a variable in Python is very simple; you choose the name of your variable and assign it a value with an equal sign (=). Unlike other languages, you do not need to declare the variable data type before the name. See the structure below for the syntax.

```
# the variable is named var
# the data can be any type or be any object
var = data goes here
```

You can give your variable any name, except you cannot use Python reserved words such as in, as, and assert. It is important to name your variable something relevant to the data it will contain so that your code is more readable in the future. For multiword variable names, camel casing is

the standard naming convention in languages such as Java. However, in Python, the convention is to separate words with an underscore. To be aware of it for other programming languages, camel casing is a naming convention for compound names, in which the first letter of the first word is not capitalized, but the first letter of subsequent words are. For example, userData or eventStartTime would both follow proper camel casing conventions. For the variable naming convention in Python, see the structure below:

```
my_variable_name = data goes here
```

String Operations

Strings can be manipulated using several Python functions available to us. An important operation in programming is concatenation – the action of combining strings. The plus sign (+) is used for concatenation, and you can combine two or more strings. For example, the code below combines a string and a string variable type, to print out a full statement.

```
to_destination = “Orange County”

from_destination = “New York”

print(“The plane landed in “ + to_destination + “ after flying from ” + from_destination)
```

The code above would output the following after concatenating the strings:

```
>>> The plane landed in Orange County after flying from New York
```

A common practice while programming is to check two strings to see if they are equal. We check for string equality using two equals signs, which will return a Boolean value depending on if the compared strings are exactly the same. The comparison is case-sensitive, and spaces count.

For example:

```
“Mikey” == “Mikey”    # returns true since they #are exactly the same

“mikey” == “Mikey”    # returns false since the #first m is lowercase

“ Mikey” == “Mikey”    # returns false since #there is a space in the first string
```

Sometimes, to make string comparisons easy, we want to make a string either all uppercase or all lowercase. For example, if you ask a user to enter a color and you want to use if statements to check if the color they entered is red or green, you should convert the input to all uppercase or lowercase letters, and compare it to the corresponding capitalization. This is important because if the program checks for the input ‘ red, ’ but the user enters ‘ reD, ’ the expression will return as false. As a result, the contents within the if statement will not be executed since the string values are not equal. The lower and upper functions are available for us to convert to a capitalization standard. See the example below for syntax.

```
company1 = “Acorns”

print (company1.upper()) # notice the upper #function is called with its name
```

```
print (company1.lower()) # two parenthesis are #used to call the function


company2 = “SouthWEST”
print (company2.upper()) # output - SOUTHWESTs
print (company2.lower())  # output - southwest
```

Casting

Sometimes in programming, you need to use casting to convert a variable from one data type to another. For example, you may need to convert a string to an integer, so that you can perform mathematical operations on it. You will often find that you need to print a statement and some quantitative value such as an integer, but you cannot pass an integer into the print function without casting it into a string. A common scenario for casting occurs when you use the input function to collect a number the user wants to enter. However, the input function returns a string, so you must convert it to an integer or float. Reference the next example.

```
degrees_farenheight = input ( “What is the temperature in fahrenheit? ” ) # string value
result = ( float (degrees_farenheight) * 32 ) - ( 5 / 9 )  # cast to float so we can perform math
print ( str (result) + “ degrees Celsius” ) # cast # result to string so it can be printed
```

The following would happen when the program is ran:

```
What is the temperature in fahrenheit? 72.5
22.5 degrees celsius
>>>
```

The subsequent table contains function calls to cast from one data type to another. Each function takes a variable or data as input, what we call an argument in Python (sometimes called parameters in other languages). Note that the argument you would like to cast must contain the appropriate value or you will receive a value error. For example, the string (73) can be cast to an integer (73) or a float (73.0), but the string ($) cannot be cast to either of these types, due to the value of the string. The ($) symbol cannot be quantified as an integer or float.

Function	Description
str(argument)	Converts data to a string
int(argument)	Converts data to an integer
float(argument)	Converts data to a decimal value
bool(argument)	Converts data to a boolean

The code below is an example of how values would change when casted:

```
num = 72 # originally an integer
```

```
num = str (num) # now a string so it is #equivalent to num = “72”

num = float (num) # now a decimal number, 72.0

num = bool (num) # not 0 so it returns true
```

Activity 16: Introduce Yourself

We ’ re going to be working with a computer a lot, asking it to do a bunch of cool things for us. We might as well introduce ourselves and become friends!

What to Do

Use the print() function to introduce yourself to the computer. Your introduction should be seen in the console window (see here for the lesson that will help you do this).

Sample Expected Output

"Hi! My name is Adrienne."

Activity 17: To Quote A Quote

A quote in non-coding terms is a sentence or short phrase that you want to repeat from a person word for word. You usually see it written like this:

“ These are the words you are repeating a.k.a. quoting.”—Person Who Said This

What to Do

Find a quote online or use one of your own. It can be about something that inspires you, a funny line from a movie, or even something a family member said. Use the print() function to write a proper quote (as shown) in

the console window. Remember to print out characters like double quotes; you need to properly encapsulate [11] them with the backslash (\) character (see here for the lesson that will help you do this).

Sample Expected Output

"Coding is a superpower! You can do many cool things with your imagination and code." —Adrienne Tacke

Activity 18: These Are A Few Of My Favorite Things

Now that you know how to create lists, try creating one with five of your favorite things! Remember, lists can have a mix of objects in them.

What to Do

Create a list named ‘ my_favorite_things ’ and add five things to it. Print out a message that says “ These are {your name} ’ s favorite things: [‘ your ’ , ‘ favorite ’ , ‘ things ’]. Use an f-string to print out this message with your name and your list of favorite things!

Sample Expected Output

'These are Adrienne's favorite things: ['Blue', 3, 'Desserts',

'Running', 33.3].'

Activity 19: Shapeshifters

One day, you and your friend decide to go to the park and watch the clouds. You want to keep track of the different clouds you see and what shapes they look like to both of you, so you each create empty lists (in brackets []) before you begin: your_cloud_shapes = []

friend_cloud_shapes = []

While watching, you continue to add the shapes of clouds you see to your lists. Once you go home, you take a look at each other ’ s lists: your_cloud_shapes = ['circle', 'turtle', 'dolphin', 'truck', 'apple', 'spoon']

friend_cloud_shapes = ['apple', 'turtle', 'spoon', 'truck', 'circle', 'dolphin']

Interesting! Both of you mostly have the same shapes, but probably saw them at different times!

What to Do

Using if statements, the == operator, and indices, write some code to check if your cloud shape matches your friend ’ s cloud shape at the same position in each of your lists. You can do this by comparing your object with your friend ’ s object at each index. If your shapes match at the same position, print out “ We saw the same shape!” If they don ’ t match, print out “ We saw different shapes this time.” Go one by one, and compare each item on your lists.

Helpful Hints

Remember, you can access specific items in lists by using the indices! Example: your_list[2]

Activity 20: Random Factory

What to Do

Using your knowledge of string concatenation and accessing list items by index, use the following list of random_items to create a proper answer to each scenario that follows. Use f-strings to print the result of your code.

random_items = ['basket', 'tennis', 'bread', 'table', 'ball', 'game', 'box']

Example

Marie is playing ping-pong with her friends. Another friend, Pierre, says that ping-pong is called something different in his country. Can you form the other name for ping-pong using the random_items list?

print(f"{random_items[3]} {random_items[1]}")

Example Output

table tennis

Scenario 1

Andre is about to play tennis with some friends. He has his tennis racket, but he needs one more thing. Write a code to print out what he needs!

Scenario 2

Jean just baked some fresh bread. He wants to bring a few loaves home to share. What can you make from the random_items list that can help him carry his bread home?

Scenario 3

Christina is singing the lyrics to a popular song that is usually sung at a baseball game. Can you finish the lyrics? “ Take me out to the_____ _____!”

Scenario 4

Leslie is writing a story about her favorite sport. It involves a hoop, five players on each team, and a recognizable orange ball with black stripes. Which sport is it?

Scenario 5

Julia just received one of the fresh loaves of bread from Jean. Thanking him, she quickly puts the loaf she received in this item to keep it warm.

Scenario 6

Mario has a lot of board games and video games. Luckily, he can store most of them in this item to keep his room nice and clean!

CHAPTER 5:

Learning Games in Python

What You Will Learn

- Rock, Paper, Scissors
- Guessing games

Activity 21: Rock Paper Scissors

The first game will be ‘ Rock, Paper, Scissors, ’ which is normally played by two people. But in this case, it is going to be you against the computer. The first thing we need to do when creating a game is to brainstorm. Using a pen and paper, think about how the game should be designed. Start by first considering the rules of the game, and only then worry about the programming side.

This classic game involves choosing one of three objects, as the name suggests. Once both selections are made, the items are revealed to see who wins. The player who wins is determined by three simple rules: the rock

will crush the scissors, while the scissors cut the paper, and the paper covers the rock.

To handle these rules, we are going to create a list of choices, similar to the list of colors we created before in some of our drawing programs. Then, we will add a random selection function that will represent the choice the computer makes. Next, the human player will have to make his or her choice. Finally, the winner is decided with the help of several if statements.

Have you tried to create your version of the game yet? If so, good job! Even if you did not finish it, or you wrote the game and are getting some errors, you should still reward yourself for trying. Now, let us go through the code and see how this game should turn out:

```
import random

selectionChoices = [ “ rock”, “ paper”, “ scissors”]

print ( “ Rock beats scissors. Scissors cut paper. Paper covers rock.” )

while player != “ quit ”:

    player = player.lower ()

computer = random.choice(selectionChoices)

print("You selected " +player+ ",

and the computer selected" +computer+ ".")


if player == computer:

print("Draw!")
```

```
elif  player == "rock":
if computer == "scissors":
print ("Victory!")
else:
print("You lose!")
elif  player == "paper":
if computer == "rock":
print("Victory!")
                                else:
print("You lose!")
elif  player  == "scissors":
if computer == "paper":
print
("Victory!")
                                          else:
print("You lose!")
else:
print("Something went wrong…")
print()
```

First, we import the random package, which allows us to use some functions that we are going to take advantage of when giving the computer the ability to make random choices. Then we create a list for the three game objects and print the game rules so that the human player knows them. The computer will already know what to do because it is programmed after all. Next, we ask the player to type his or her choice, and then a loop is executed to check the choice of the player. The player also has the option of quitting the prompt window. When that happens the game is over. Our loop makes sure that if the player does not select the quit option, the game will run.

The next step is to ask the computer to select one of the three game objects. This choice is done randomly, and the selected item is stored inside a variable called the computer. After the choice is memorized, the testing phase begins to see which player will win. First, a check is performed to see whether the two players have chosen the same item. If they did, then the result is a draw, and nobody wins. Next, the program verifies whether the player chose ‘ rock, ’ and then it looks at the computer to see if it chose ‘ scissors. ’ If this is the case, then the rule is ‘ rock beats scissors, ’ so the player wins. If the computer neither selected a ‘ rock ’ nor ‘ scissors, ’ then it certainly chose ‘ paper. ’ In this case, the computer will win. Next, we have two elif statements, by which we perform two more tests that check whether the player selected ‘ paper ’ or ‘ scissors. ’ Here we also have a statement that checks to see if the player chose something that is not one of the three possible items. If that is the case, an error message is sent, that tells the player he either chose something that he is not allowed to, or he mistyped the command.

Lastly, the user is prompted to type the next selection. This is where the main loop goes back to the beginning. In other words, the game starts

another round of Rock, Paper, Scissors.

This game is simple, but it is fun because anyone can win. The computer has a chance of beating you, and there is also a real chance of ending up in a draw. Now that you understand how to create a random chance game, let us look at other examples to add to our game library while also learning Python programming. [12]

Activity 22: Guessing Game

The purpose of the game will be to choose a number between a minimum and a maximum. Then the opponent tries to guess that number. If the player guesses a higher number, he will have to try a smaller one, and the other way around as well. Only a perfect match will turn into a win.

Comparing numbers is something we already did, by using the if statement. We have also used the input function to interact with the program, and we are going to make use of it here once again. Besides this, we will need a while-loop as well.

In this project, the random module is needed because of certain specific functions. For instance, we know that we need to generate a random number, therefore we will use a function called randint, which stands for ‘ random integer. ’ The function will have two parameters, which represent the minimum number we can have, as well as the maximum. You can try out this function on its own. Just import the module and type the following:

```
import random

random.randint (1, 20)
```

Python will now automatically generate a random figure between 1 and 20. Keep in mind that the minimum and maximum values are included in the number generation. Therefore, Python can also generate numbers 1 or 20. You can test this command as many times as you want, to make sure that you are truly getting random values. If you execute it often enough, you will see that some values will repeat themselves. If the range is large enough you might not even encounter certain numbers, no matter how many times you run the code.

What is interesting about this function though, is that it is not truly random. This is just a side note that will not affect your program, but it is intriguing, nonetheless. The randint function follows a specific pattern and the chosen numbers only appear to be random, but they are not. Python follows a complex algorithm for this pattern instead, and therefore we experience the illusion of randomness. With that being said, let us get back to fun and games. Let us create our game with the following code:

```
import random

randomNumbers = random.randint (1, 100)

myGuess = int (input ( “ Try to guess the number. It can be anywhere from 1 to 100:” ))

while guess != randomNumbers:

    if myGuess>randomNumbers:

            print (myGuess, “ was larger than the number. Guess again!”
```

```
        if myGuess<randomNumbers:
                print (myGuess, " was smaller than the number. Guess
    again!"
    myGuess = int (input ( " Try and guess again! " ))
    print (myGuess, " you got it right! You won!" )
```

That is it! Hopefully, you tried to create this game on your own because you already have the tools for the job. Remember that programming is only easy as long as you practice it enough on your own. Just take it one step at a time. With that being said, let us discuss the code, in case you need some help figuring the game out.

Just like before, we first need to import the random module so that we can use the random number generating function. Next, we use the randint function with two parameters. As mentioned before, these parameters are the lowest number we can guess, which is 1, and the highest number we can guess, which is 100. The random number generator will generate a number within this range. Once the number is generated, it is stored inside the randomNumbers variable we declared. This number will not be known by the player, because he or she needs to guess it. That is the point of the game.

Next up, the player needs to guess the hidden number. This guess will then be stored inside a new variable called myGuess. To check whether the guess is equal to the number, we are using a while-loop with the not equal to operator. We do this because if the player gets lucky and guesses the number correctly on the first attempt, the loop simply does not finish executing because there is no need.

Next, if the player guesses the wrong number, we have two if statements that check whether the guess is a higher value than the hidden number, or a

lower one. An appropriate message is then printed for the player in each case. In either scenario, the player receives another chance to make the right guess. Finally, in the end, if the user guessed the number correctly, the program declares victory for the user by printing a message, and then the program stops running. [13]

Activity 23: Drawing Game Boards

A lot of games require a game board made up of a different number of squares. Let ’ s try creating a module that creates any size game board we need by simply giving it a number!

What to do

Create a file called ‘ game-board ’ and save it. Then, define two functions: one to print some horizontal lines and one to print some vertical lines: def print_horizontal_line():

Def print_vertical_line():

Next, use the print() function to print out the lines: def print_horizontal_line():

Print(" --- ")

Def print_vertical_line():

Print("| _ ")

Next, we need to ask the player what size game board they need. We should capture their input in a variable: board_size = int(input("what size game board do you need?"))

Finally, create a for-loop that iterates as many times as the board size requested by the player, and print the lines using your defined print line functions!

<write some code here>

Now, to correctly print the game board, we need to change our print line functions a bit. For the print_horizontal_line() function, how would you change it to print as many lines as the requested game board size? (hint: remember that weird operator that we can use to “ multiply” strings? Hmm...) Def print_horizontal_line():

Print(" --- " <write some code here>)

For the print_vertical_line() function, you ’ ll need to print out as many lines as the requested board size, plus one.

Def print_vertical_line():

Print("| _ " <write some code here>)

Finally, print one last horizontal line to finish your board after your for loop: print(" --- " * board_size)

That ’ s it! When you save and run your file, it will ask you what size board you need. Give it a number, and it will print out a board for you, making the

board the number of squares in height and width. As you can see here, the number 3 gave us a board three squares wide and three squares high!

Activity 24: If This, Then That

As we grow older, who we are, what we look like, and what we are interested in will probably change. Let ’ s capture that in an if statement and print out what we think we will be like in the next 5, 10, 15, and 20 years.

What to Do

Write an if statement that checks for the year, and then output the different predictions you have about yourself for that year! As you can see, I ’ ve helped you get started. Write the remaining elif statements, and make sure to update your variables properly for each year.

Let ’ s capture three things to output with some variables. Create age, favorite_outfit, and favorite_hobby variables, and assign each of them to what they are today.

year == 2019

age = 10

favorite_outfit = "red dress"

favorite_hobby = "coding"

Next, start your if statement and check for the current year: if year == 2019:

Then, print out your current description:

if year == 2019:

print(f"It is 2019. I am currently {age} years old, love wearing a {favorite_outfit}, and currently, {favorite_hobby} takes up all my time!")

Now, create four more elif statements for 5, 10, 15, and 20 years from now. Adjust your variables, too!

Activity 25: Slicing And Dicing

This activity is about a boy named Tony. He has crates of fruits and vegetables coming in and needs someone to sort them. If the crate has vegetables, they need to be taken out and moved to the “ dicing” area, so his helpers can begin dicing them for the restaurant. If you find fruits though, they need to be brought to the “ slicing” area, so his bakers can prepare the fruits for their desserts.

What to Do

Using slice ranges and the different methods we ’ ve learned to add items to a list, write some code for each crate to properly separate the fruits and vegetables, and add them to the right area.

I ’ ve created two variables for you to start:

slicing_area = []

dicing_area = []

Once you ’ ve gone through all of the crates, print out all of the separated fruits and vegetables.

Here are the crates:

crate_1 = ['onions', 'peppers', 'mushrooms', 'apples', 'peaches']

crate_2 = ['lemons', 'limes', 'broccoli', 'cauliflower', 'tangerines']

crate_3 = ['squash', 'potatoes', 'cherries', 'cucumbers', 'carrots']

Activity 26: To Change Or Not To Change

What to Do

For each collection of items, create either a tuple or list, and store those items within it. Then, print out the contents of the list and which type it is stored in.

Collection 1:

first_name, last_name, eye_color, hair_color, number_of_fingers, number_of_toes

Collection 1 Data: "Adrienne", "Tacke", "brown", "black", 10, 10

Collection 2: favorite animals Collection 2 Data: "cats", "dogs", "turtles", "bunnies"

Collection 3: colors of the rainbow Collection 3 Data:

"red", "orange", "yellow", "green", "blue", "indigo", "violet"

Sample Expected Output

('red', 'green', 'blue') are stored in a tuple!

Activity 27: Choose Your Adventure

To help you get started, follow these instructions:

1. First, create a Python file called ‘ choose-your-adventure ’ and save it.

2. Use the following code to start defining your game:

```
# Change to your name so you can have your own game!

name = "Your name here"

# Adventure begins.

print(f"Welcome to {name}'s Choose Your Own Adventure game! As you follow the story, you will be presented with choices that decide your fate. Take care and choose wisely! Let's begin.")

print("You find yourself in a dark room with 2 doors. The first door is red, the second is white!")

# This input function allows you to type in your choice. By assigning it to a variable, you can use the choice that has been made to decide on the next

# part of the story!

door_choice = input("Which door do you want to choose? red=red door or white=white door")

if door_choice == "red":

print("Great, you walk through the red door and are now in the future! You meet a scientist who gives you the mission of helping him save the world!")

choice_one = input("What do you want to do? 1=Accept or 2=Decline")

if choice_one=="1":

print("""_____SUCCESS_____

You helped the scientist save the world! In gratitude, the scientist builds a time machine and sends you home!""")

else:
```

print("""_____GAME OVER_____

Too bad! You declined the scientist's offer and now you are stuck in the future!""")

else:

print("Great, you walked through the white door and now you are in the past! You meet a princess who asks you to go on a quest.")

quest_choice = input("Do you want to accept her offer and go on the quest, or do you want to stay where you are? 1=Accept and go on quest or 2=Stay")

if quest_choice=="1":

print("The princess thanks you for accepting her offer. You begin the quest.")

else:

print("""_____GAME OVER_____

Well, I guess your story ends here!""")

Use what you ’ ve learnt about if statements, along with your knowledge of variables, the print() function, and several data types to continue this story. Change the outcomes, have more than one decision to make or set your story in a different setting. It ’ s up to you! Once you are finished, save your game and then run it. You or a friend can now choose your adventure, and it will be the game you created!

CHAPTER 6:

Working with Python Functions

What You Will Learn

- How to define and call a function
- Understanding functions better
- Return statement
- Multiple parameters
- Lambada function

Creating and calling a function is easy. The primary purpose of a function is to allow you to organize, simplify, and modularize your code. Whenever you have a set of code that you will need to execute in sequence from time to time, defining a function for that set of code will save you time and space in your program. Instead of repeatedly typing code or even copy-pasting, you simply define a function.

We began with almost no prior knowledge about Python, except for the fact that it was a programming language that is in great demand these days. Now, look at you: creating simple programs, executing codes, and fixing small-scale problems on your own. Not bad at all! However, learning always comes to a point where things can get rather tricky.

In quite a similar fashion, functions are docile-looking things; you call them when you need to get something done. But did you know that these

functions have so much going on at the back? Imagine every function as a mini-program. It is also written by programmers like us to carry out specific things, without having us write lines and lines of codes. You only do it once, save it as a function, and then just call the function where it is applicable or needed.

The time has come for us to dive into a complex world of functions, where we do not just learn how to use them effectively, but we also look into what goes on behind these functions, and how we can come up with our very own personalized function. This will be slightly challenging, but I promise you, there are more references that you will enjoy, to keep the momentum going.

How To Define And Call Function?

To start, we need to take a look at how we can define our own functions in this language. The function in Python is going to be defined when we use the def statement, and then follow it up with a function name and parentheses in place as well. This lets the compiler know that you are defining a function, and the specific function you would like to define at this time as well. There are going to be a few rules in place when it comes to defining one of these functions though, and it is important to do these properly, to ensure your code acts in the way that you would like. Some of the Python rules that we need to follow for defining these functions will include the below rules:

1. Any of the arguments or input parameters that you would like to use, have to be placed within the parentheses so that the compiler knows what is going on.

2. The function ’ s first statement is something that can be an optional statement, something like a documentation string that goes with your function, if needed.
3. The code found within all functions we are working with needs to start with a colon, and needs to be indented as well.
4. The return statement that we get, or the expression, will need to exit a function at this time. We can then have the option of passing back a value to the caller. A return statement, that does not have an argument with it, is going to give us the same return as none.

Before we get too familiar with some of the work that can be done with these Python functions, we need to take some time to understand the rules of indentation, when we are declaring these functions in Python. The same kinds of rules will be applicable to some of the other elements of Python as well, such as declaring conditions, variables, and loops, so learning how this works can be important here.

You will find that Python is going to follow a particular style when it comes to indentation, to help define the code, because the functions in this language are not going to have any explicit beginning or end, like the curly braces in other languages that help indicate the start and the stop for that function. This is why we are going to rely on indentation instead. When we work with proper indentation here, we can really see some good results and ensure that the compiler is going to know when the function is being used.

Understanding Functions Better

Functions are like containers that store lines and lines of codes within themselves, just like a variable that contains one specific value. There are two types of functions we get to deal with in Python. The first ones are

built-in or predefined, while the others are custom-made or user-created functions.

Either way, each function has a specific task that it can perform. The code that is written before creating any function is what gives that function an identity and a task. Now, the function knows what it needs to do whenever it is called in.

When we began our journey, we wrote ‘ I made it! ’ on the console as our first program, right? We used our first function there as well: the print() function. Functions are generally identified by parentheses that follow the name of the function. Within these parentheses, we pass arguments called parameters. Some functions accept a certain kind of parenthesis while others accept different ones.

Let us look a little deeper and see how functions greatly help us reduce our work, and help better organize our codes. Imagine we have a program that runs during the live streaming of an event. The purpose of the program is to provide our users with a customized greeting. Imagine just how many times you would need to write the same code if there were quite a few users who decided to join your stream. Using functions, you can cut down on your work easily.

For us to create a function, we first need to ‘ define ’ it. That is where a keyword, called def comes in. When you start typing def, Python immediately knows you are about to define a function. You will see the color of the three letters change to orange (if you are using PyCharm as your IDE). That is another sign of confirmation that Python knows what you are about to do.

```
def say_hi():
```

Here, say_hi is the name I have decided to go with; you can choose any that you prefer. Remember, keep your name descriptive, so that it is understandable and easy to read for anyone. After you have named your function, follow it up with parentheses. Lastly, add a colon to let Python know we are about to add a block of code. Press **Enter** to start a new indented line.

Now, we shall print out two statements for every user who will join the stream.

```
print("Hello there!")

print('Welcome to My Live Stream!')
```

After this, give two lines of space to take away those wiggly lines that appear the minute you start typing something else. Now, to have this printed out easily, just call the function by typing its name, and then run the program. In our case, it would be:

- say_hi()
- Hello there!
- Welcome to My Live Stream!

See how easy this can make our work in the future? We do not have to repeat our codes over and over again. Let us make this function a little more interesting by giving it a parameter. Right at the top line, where it says def say_hi(), let us add a parameter. Type in the word name as a parameter within the parenthesis. Now, the word should be greyed out to confirm that Python has understood the same as a parameter.

Now, you can use this to your advantage and further personalize the greetings to something like this:

If you are doing it on Shell, you need to delete the previously defined function with

Del sayhi

Now, let us write the new function:

```
def say_hi(user):

print(f"Hello there, {user}!")

print('Welcome to My Live Stream!')

user = input("Please enter your name to begin: ")

say_hi(user)
```

The output would now ask the user their name. This will then be stored in a variable called ‘ user. ’ Since this is a string value, say_hi() should be able to accept this easily. Bypassing the user as an argument, we get it as an output:

```
Please enter your name to begin: Johnny

Hello there, Johnny!

Welcome to My Live Stream!
```

Now that is more like it! Personalized to perfection. We can add as many lines as we want, the function will continue to update itself, and provide greetings to various users with different names.

There may be times where you may need more than just the user ’ s first name. You might want to inquire about the last name of the user as well. To add that, add it to the first line and follow the same process accordingly:

```
def say_hi(first_name, last_name):
```

```
    print(f"Hello there, {first_name} {last_name}!")
    print('Welcome to My Live Stream!')
first_name = input("Enter your first name: ")
last_name = input("Enter your last name: ")
say_hi(first_name, last_name)
```

Now, the program will begin by asking the user for their first name, followed by the last name. Once that is sorted, the program will provide a personalized greeting with both the first and last names.

However, these are positional arguments, meaning that each value you input is in order. If you were to change the positions of the first and surnames in ‘ John Doe, ‘ ’ Doe ’ will become the first name and ‘ John ’ will become the last name. You might want to be a little careful about that.

Hopefully, now you have a good idea of what functions are, and how you can access and create them. Now, we will jump towards a more complex front of return statements.

Wait! There is more! Well, I could have explained it; however, you may not have understood it completely. Since we have covered all the bases, it is appropriate enough for us to see exactly what these are, and how these gel along with functions.

Quiz

1. Which function will print an output to the screen?

    echo

    print

eval

2. What is the output of this code?

   def x:

   pass

   print type (f())

3. What is the purpose of the following code?

   def a (b, c, d)

   pass

4. What is the output of the following block of code?

   a = [1,2,3,None,(),[],]

   print len(a)

   4

   5

   6

5. State the value of the item with index 1 from the following list:

   colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']

   red

   orange

   violet

6. Is the statement “ There are only two value types in Python, strings, and numbers” correct?

True

False

7. What is the result of this code? num = '5'*'5'

   333

   5

   Type Error

8. What is the result of this code? print abc

   “ abc”

   abc

   Error

9. Is the statement “ There are only two types of numbers in Python, assigned values and integers” correct?

   True

   False

10. What is the result of this code? print 3 / 5

    0.6

    0

    None

Activity 28: Super Function!

What to Do

Create a function called superpower(). Have your superpower() function accept two parameters: one called ‘ name ’ and another called ‘ power. ’ Using these parameters, have your function print out an f-string that says who you are and what your superpower is!

Sample Expected Output

'Hi, I'm Super Adrienne and my superpower is coding!'

Activity 29: Funny Functions

What to Do

Create a function called funny_greeting(). Have your funny_greeting() function accept two parameters: one called ‘ color ’ and another called ‘

dessert. ’ Using these parameters, have your function print out an f-string that mixes up the parameters on purpose to produce a silly message!

Sample Expected Output

'My favorite dessert is blueberry pie because it tastes so good, and my favorite color red is because it is very pretty!!'

Activity 30: What Time Is It Over There?

When you have friends around the world, keeping track of the right times to call them can get a little tricky. Depending on where they are, they can be hours ahead of or behind you! To help coordinate, let ’ s write a function that helps us figure out what time it is in our friends ’ cities.

What to Do

Using the datetime() and timedelta() functions from the Datetime module (see here) and some math, write a function that prints out the current time in your home city and the following three cities:

Berlin, Germany

Baguio City, Philippines

Tokyo, Japan

My Home: Las Vegas, United States

First, be sure to import the following functions so you can use them: from datetime, import datetime; from datetime, import timedelta [14] .

Next, create a function called world_times(). I ’ ve already started the function for you, so just fill in the blanks to calculate the other cities ’ times and then print out the final string!

def world_times():

my_city = datetime.now()

berlin = <Write some code here>

baguio = <Write some code here>

tokyo = <Write some code here>

all_times = f'"It is {my_city:%I:%M} in my city.

That means it's {berlin:%I:%M} in Berlin, {baguio:%I:%M} in Baguio City and {tokyo:%I:%M} in Tokyo!'"

<Write some code here> # print your all_times variable!

To calculate the other cities ’ times, you ’ ll probably need to add some hours to the my_city variable. You can add hours to a variable by using the timedelta() function. The timedelta() function gives us an easy way to properly add units of time (like days, months, hours, minutes, etc.) to a date or time.

In this activity, you ’ ll only need to add hours. You do this by adding a datetime object to a specific amount of hours. So as an example, if you wanted to add 9 hours to the current time and then assign this result to a variable called nine_hours_from_now, you ’ d do the following: nine_hours_from_now = datetime.now() + timedelta(hours=9)

>>> Helpful Hint: You can use the Internet to find the time differences between your home city and the three cities mentioned. Once you figure out those numbers, use them in your function ’ s calculations!

>>> Helpful Hint: Don ’ t change the f-string I ’ ve provided! The resulting times you calculate should print out to a nice, readable format.

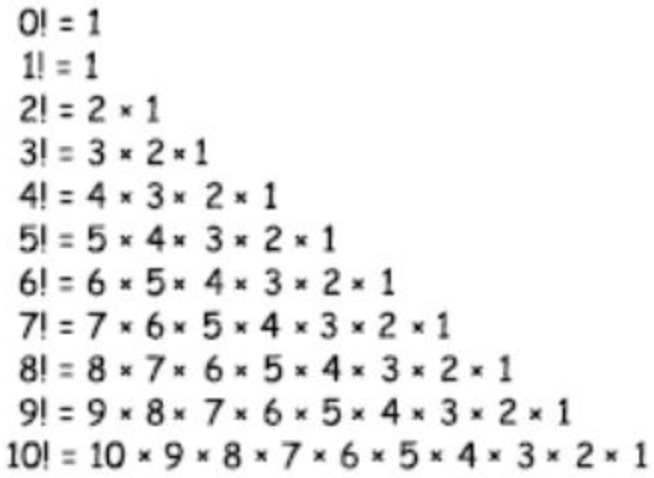
Sample Expected Output

```
>>> world_times()
```

It is 07:37 in Las Vegas.

That means it's 04:37 in Berlin, 10:37 in Baguio City, and 11:37 in Tokyo!

Activity 31: Factorial Function



One of the most common functions every coder has to write is called a factorial function. It ’ s a function that calculates the factorial of the number you pass into it. And yes, it sounds like something to do with multiplication, because it is! In math, a factorial is the product of a number and all the numbers that come before it. So, if I asked you to calculate the factorial of the number 4, you would have to multiply 4 by, 3, 2, and 1. The factorial of 4 is 24.

What to Do

Write a function called factorial() that takes one parameter. This parameter will be a number. Then, write the code to calculate the factorial of the number that is passed in as a parameter. Have your factorial() function return the answer!

Sample Expected Output

>>> factorial(4)

24

Activity 32: Math Codes

You can get in a bit of mathematics practice while playing these coding games, as well. This can be either as part of a treasure hunt or can just be written down on paper for the fun of solving them. You can add in an extra challenge by saying, “ How many can you solve in three/five/ten minutes?”, and setting a timer.

For example, using the numbers instead of letters code, where A = 1 and B = 2, etc., you can leave spaces or draw a line for each letter, and underneath it, write a math ’ s sum. The child works out the sum, which gives them a number, which they then have to translate into a letter.

An example of I LOVE YOU would be

—	———	———	———	———	———	———	———
3x3	15-3	5x3	11x2	10-5	30-5	30-15	20 – 1

For example, 3x3 = 9, which corresponds to the letter I; 15 – 3 = 12, which corresponds to the letter L; and so on. Of course, these are just examples. Make the sums as easy or difficult as you want, depending on the age and ability of the children. For five- and six-year-olds, you may want to use subtraction and addition sums; for older children who are familiar with multiplication and division, you can start using those.

Activity 33: Cupcakecookie

Dolores and Maeve are having a party together and are setting up the dessert tables. Dolores likes cupcakes and Maeve loves cookies! Unfortunately, when they go to the kitchen, all of their boxes have been mixed up! Each dessert is in a special box, but all the boxes look the same! Dolores and Maeve don ’ t fret, though. They know that they can tell which dessert is which, because the cookies are in a box with a 3 on them, and the cupcakes are in a box with a 5. Let ’ s write a function to help them organize their desserts!

What to do

Write a function called dessert_sorter() that takes one parameter. Call the parameter total_desserts. Then, write some code that will help Dolores and Maeve separate the cupcakes from the cookies. This should be a for-loop that goes through the total_desserts and checks for these things:

•If it ’ s a number that ’ s divisible by 3, print out the word “ cupcake.”

•If it ’ s a number that ’ s divisible by 5, print out the word “ cookie.”

•If it ’ s a number that ’ s divisible by both 3 and 5, print out “ it ’ s a cupcakecookie!”

When you ’ re done creating your dessert_sorter() function, pass 200 as the total_desserts parameter, because that ’ s how many boxes Dolores and Maeve have to sort!

>>> dessert_sorter(15)

Cupcake

Cookie

Cupcake

Cupcake

Cookie

Cupcake

It's a cupcakecookie!

ACTIVITY 7: ROCK PAPER SCISSORS

Rock, paper, scissors, go! This game is a very popular game to play with friends. For as many turns as you like. You and a friend can choose between

Rock, Paper, or Scissors and see who wins between the two of you. Let ’ s create this game in Python, where you can battle friends on the computer!

What to Do

Create a file called ‘ rock-paper-scissors-game, ’ and save it. Next, begin creating your game!

Let ’ s start by greeting the players:

print("Welcome to the Rock Paper Scissors Game!")

Now, create two variables that will store the names of each player

player_1 = <Write some code here>

player_2 = <Write some code here>

Next, define a function called compare() and have it accept two parameters. This function will compare the players ’ choices (which are the two parameters it accepts) and will tell us who won, based on the rules of Rock, Paper, Scissors: def compare(item_1, item_2):

Now, within our compare() function, we have to write a few if statements! Check for each combination possible in Rock, Paper, Scissors, and then print out the winner in each combination. Keep in mind that each item is stronger than one other item but weaker than another. To help you write the Boolean expressions for your if statements, I ’ ve provided a list of Rock, Paper, Scissors combinations, and who would win in each combination based on the rules:

Choice 1 Choice 2 Winner Between the Two

Rock Paper Paper (paper covers rock)

Rock Scissors Rock (rock breaks scissors)

Rock Rock It ’ s a tie!

Paper Rock Paper (paper covers rock)

Paper Scissors Scissors (scissors cut paper)

Paper Paper It ’ s a tie!

Scissors Rock Rock (rock breaks scissors)

Scissors Paper Scissors (scissors cut paper)

Scissors Scissors It ’ s a tie!

Be sure to add one last elif statement to deal with any choice that is not Rock, Paper, or Scissors. It would be a good idea to also tell the players that they have entered a choice that is not possible if they do this.

Now that we have a compare() function that can check the combinations for us, the last part is to capture the choices our players make! Create two variables to store the player ’ s choices: player_1_choice = <Write some code here>

player_2_choice = <Write some code here>

Lastly, use a print() function to print the results of the compare() function when you pass the players ’ choices into it!

print(compare(player_1_choice, player_2_choice))

That ’ s it! Save your file, then press **F5** to run it so you can play Rock, Paper, Scissors with a friend! Take turns entering your choices, and see who ’ s won!

CHAPTER 7 ☆ CHALLENGES

CHALLENGE 1: HANGMAN GAME

Using everything you have learnt, try finishing this Hangman game. I have provided the structure for a Hangman game below for you to use. However, it ’ s up to you to fill in the blanks! Once you have filled in all of the missing code, noted by the <Write some code here> placeholders, save your file. At this point, you should be able to play hangman when you press **F5** and run your game!

What to Do

Create a new file called ‘ hangman, ’ and save it. Using the template below, start writing the code into your hangman.py file. When you come to a placeholder that says **<Write some code here>** , remove the placeholder and replace it with the proper code. Use the comments to help you figure out what kind of code to write.

```
# importing the time module

import time

# Welcome the user and capture their name in a name variable

name = input("What is your name?")

# Use a print function to greet the user by their name

<Write some code here>

# Wait for 1 second

time.sleep(1)

print("Start guessing...")

time.sleep(0.5)
```

Create a variable called secret_word to store the word to be guessed

<Write some code here>

Create a variable called guesses and assign it to an empty string ''

We'll store the letters the player guesses here

<Write some code here>

Create a variable to store the maximum number of turns the game will allow

<Write some code here>

Start a while loop

and check if we have more than 0 turns available

<Write some code here>

If we have turns available:

Create a counter variable that starts at 0 to hold the number of incorrect

guesses we make

<Write some code here>

Start a for loop

and iterate through every character in your secret_word variable

<Write some code here>

As you iterate through each character:

use an if statement to check if the letter is

in the player's guess, aka the guesses variable

```
<Write some code here>
# If it is, print then out the character
<Write some code here>
else:
# If it isn't, print an underscore ...
print("_")
# ...and increase the failed counter by 1
<Write some code here>
# Check if your incorrect guesses are equal to 0
<Write some code here>
# If it is, tell the user they've won!
<Write some code here>
# ...then exit the game
break
# Otherwise, ask the player to guess another character
guess = input("Guess a character:")
# Add the player's guess to the guesses variable
guesses += guess
# Create an if statement
# and check if the guess is not found in the word
<Write some code here>
```

Decrease your turns by 1

<Write some code here>

...and tell the player their guess was wrong

<Write some code here>

Also tell the player how many turns they have left

<Write some code here>

Create an if statement to check if your turns are equal to 0

<Write some code here>

If they are, tell the player they've lost

<Write some code here>

CHALLENGE 2: TURTLE RACE!

Let ’ s race some Toogas! We ’ ll create a race track and some colorful turtles, and then send them off! Play with your friends by choosing a turtle at the beginning of the race and seeing if it finishes first!

What to Do

Create a new file called ‘ turtle-race-game, ’ and save it. Then, begin coding your turtle race game!

First, import the turtle and random modules like this: from turtle import *

From the random module, import randint.

Next, let ’ s set up the race track:

speed()

```
penup()
goto(-140, 140)
# Create a for loop that iterates from 0 - 15
<Write some code here>
# Use the write() function to write the number of your for loop iterator.
# Set the align parameter to 'center'. These will be your steps or distances
# in the race!
<Write some code here>
right(90)
# Create another for loop that iterates from 0 - 8
<Write some code here>
# Use the penup(), forward(), and pendown() functions
# to draw dashes for your race track
# First, lift your pen
<Write some code here>
# Second, move forward 10 pixels
<Write some code here>
# Third, put your pen down
<Write some code here>
# Last, move forward another 10 pixels
<Write some code here>
```

```
# Go backward so you can draw the dashes
# for the other steps/distances
# First, lift your pen
<Write some code here>
# Then, move backward 160 pixels
<Write some code here>
# Turn left 90 degrees
<Write some code here>
# Last, move forward 20 pixels
<Write some code here>
# Now, begin creating turtles! I'll create four, but feel free to create more
# Create a turtle
<Write some code here>
# Set its shape to a turtle
<Write some code here>
# Set its color
<Write some code here>
# Lift your pen
<Write some code here>
# Now, move this first turtle to the top left
# Use the goto() function to move it to x = -160, y = 100
```

<Write some code here>

Put the pen back down

<Write some code here>

Finally, make your first turtle do a little spin

when they get to the starting line!

Create a for loop that iterates from zero to a number you choose

<Write some code here>

Turn your first turtle to the right by a number of degrees you choose

<Write some code here>

Create three (or more!) turtles with different names and colors

Make sure each turtle repeats all of the steps and code we wrote

for the first turtle :)

#

When you get to the goto() function for each turtle

use these coordinates:

2nd turtle: x = -160, y = 70

3rd turtle: x = -160, y = 40

4th turtle: x = -160, y = 10

any other turtle afterward: x = -160, y = the last turtle's y coordinate minus 30

<Write lots of code here>

<Code for the three other turtles>

Finally, after your code for three other turtles,

make the turtles race!

Create a for loop that iterates 100 times

<Write some code here>

For each turtle, move them forward by a random number

chosen by the random function. Give the random function

a range of 1 - 5 to pick from

<Write some code here>

That ’ s it! Save your game, pick a turtle, and press **F5** to run your game. You ’ ll get to watch your race track be drawn, and all of your turtles race!

Conclusion

This brings us to the end of this guide. I hope that you enjoyed learning more about the world of programming, and how powerful and versatile it can be. Programming has become one of the most valuable skills you can acquire today, not only in your professional life but in your personal life as well. I cannot count how many times I have written a small script to help me with my daily tasks. It is something you can use for the rest of your life.

I hope that you were able to grasp the core gaming concepts, as well as the fundamental programming techniques. Note that the lessons you learnt do

not only apply while programming games. They can also be used for creating apps and other types of programs as well.

Thus far, you have only gone through most of the basics—yes most of them. There are a few more ideas that we could have covered like file processing, dictionaries, tuples, but they are for another time.

There is still a ton of stuff to know about programming in Python. But you have at least completed all the fun stuff. From here you can learn more advanced topics of Python programming, such as graphics and sound, which should be fun as well.

Other advanced topics will include classes, objects, networking, database access, sending emails, processing user interfaces, and a lot of other things. If you have enjoyed the programs that you have created here, then you will find the rest of the other programs to be easier, since you now have a good grip of the basics.

Programming also enhances logical reasoning skills, because, for even the simplest programming task, the use of logic is essential. Furthermore, this logical way of thinking is good for developing children ’ s math and science skills. Programming and the use of computers is also a science in itself.

You might ask what is next. By now you probably have an idea of which programming language you prefer. You really cannot go wrong with any of these languages. However, if you are still undecided, I suggest just trying one out. If you do not like it, you can simply move on to the next one.

The most important part of learning any programming language is practice.

Here are countless reasons you might want to learn Python programming. The age we are in is evolving at a breath-taking pace. In the near future,

there will be a high demand for programmers. Now is the best time to brace yourself.

Although there are other popular computing tools used for data analysis, Python is the only reliable programming language intended for general use. Learning the actions of various Python libraries mentioned in this e-book will help aspirants to specialize in data manipulation.

I hope that the lessons that you learned here have inspired you to learn more about Python programming.

Glossary

When learning a programming language, or just programming in general, you cannot help but come across many terms that you do not understand. A lot of these terms are not used in our daily vocabulary, or they have a completely different meaning in the world of programming. Therefore, in this glossary, you will find the most important terms that are used throughout this book.

Algorithm

A collection of instructions that are used to achieve a goal. Think of it as ingredients for a recipe.

Append

Introducing something to the end of something. For instance, adding items to the end of a list.

Argument

Passing a value to a function. Sometimes called a parameter.

Boolean

An expression that can only be true or false.

Concatenate

Combining two strings to form just one.

Conditional expression

A statement that gives the program its ability to check a certain value, and then perform a set of actions based on it.

Expression

A collection of variables, values, functions, and operators that lead to a result.

For-loop

A type of statement in programming languages that repeats a section of code based on a range of values.

Function

A collection of statements that can be reused to perform various actions.

Import

Bringing a block of reusable code or a set of functions inside any program, to have access to its functionality.

Index

The position of an element inside a list.

Initialize

Assigning the first value to a variable, or any other item, in other words, giving it its initial value.

Input

It is the act of entering data into a program or a system. An input can come from a keyboard, mouse, or any other device that can record information. These devices are also called input devices.

List

A collection of items or values.

Loop

A collection of commands that the program will repeat a certain number of times.

Module

A type of file that contains various functions, classes, and variables that can be used in any program once imported.

Parameter

A variable that is attached to a function within its definition.

Range

A collection of values found between a minimum and a maximum value.

Shell

A command-line user interface that reads and executes your commands directly. IDLE is an example of a Shell.

String

A character sequence that can form words or sentences. They include letters, symbols, numbers, as well as spaces.

Syntax

The programming structure or rules of a certain coding element. In a way, it is the grammar of programming.

Variable

A value with a name, which can always change inside the program.

While loop

A statement that repeats a collection of instructions while a certain condition is true.

Answer Key

Chapter 3: Making Choices and Decisions

1. Repetition of certain steps.
2. For and while
3. True
4. ==
5. Yes
6. For-loop
7. Boolean
8. :
9. False
10. >=

Chapter 7: Working with Python Functions

1. print
2. <type 'NoneType'>
3. Defines a function that does not do anything.
4. 6
5. Orange
6. True
7. Type Error
8. Error/Name Error
9. False
10. 0

---

[1]

I removed those that can be removed.

-- The latest Python version is 3.8.5. I only found 3.7 as the latest version from the Amazon books.

I made a few adjustments on the images so it won't look too 'copied' from another book

[2]

I couldn't find better images like the client wants ☹

[3]

Review.

[4]

Review if correct.

[5]

Incomplete. Review.

[6]

Incomplete, review.

[7]

See if this is meant to be double with a . & space in between.

[8]

Review and see if this needs to mimiced whereever ‘Turtle’ is mentioned.

[9]

These seem like contradictory statements.

[10]

Review.

[11]

Review if choice of word is correct.

[12]

This is the repeated Activity 1. not removed in the clean version.

[13]

Save for the final paragraph in Activity 2, the rest is all copied. Not omitted in clean version.

[14]

Review and capitalize as appropriate.