CODING FOR KIDS IN PYTHON

CREATE YOUR FIRST GAME WITH PYTHON

ELLEN TALE

CODING FOR KIDS IN PYTHON

CREATE YOUR FIRST GAME WITH PYTHON

ELLEN TALE

Coding for Kids in Python

Create Your First Game with Python

Ellen Tale

☼ Introduction

Shay, Matilda, and James are cool. They can code! They go on awesome adventures on their computers. They create games sometimes during some of these adventures. They even sometimes sell these games and get some money. They use this money to get more experience on their computer and sometimes the dreamy things they want in real life.

This book is written to help you to be cool like Shay, Matilda, and James, to have fun like them, create games like them, and maybe sell your games one day.

To do this, this book will teach you how to Code with Python, one of the easiest ways to go on beautiful adventures on your computer. There are so many awesome adventures you can go on with coding, but we would focus mainly on creating games. At the end of this book, you would have the necessary tools to develop yourself further with the python programming language.

Now get yourself a good computer and let's go on this beautiful adventure together. Some parts of it may be annoying, some parts hard and others complex. Still, it is necessary because, in every excellent experience, the hero must face hard and complicated things else. You would have no adventure, and going to the kitchen to stuff your face full of cake would be an adventure.

When things get hard or confusing as you read this book, all you have to do is to try to break down what you are doing into the simplest of terms. For example, when you don't understand a sentence, you should read each word in that sentence understanding the meaning of each word then reread the sentence. If you try this, you will find out that usually, you will appreciate the sentence like magic — the same works for this book.

You will need a good map to find your way around on this adventure, and we have made one for you!

MAP

The first checkpoint (boring adults call them chapters) you will encounter is titled " *the beginning* ." It would tell you why python should be the programming language you will use for this adventure and not other programming languages. Also, it explains how to download the python application so you can use it on your computer.
Next, you will arrive at a checkpoint called " *getting to know python-basic skills and concepts* ." This is where you would be introduced to the computer language. This stage is like the helicopter ride over the city where the adventure will take place.
Next, you arrive at a checkpoint which contains the "levels." On each level, you will encounter many villains (bad guys) which you will fight. They will come in the form of exercises which you would solve (don't worry, they are not like practices form school) and at the end of each level, you would encounter a big evil villain which you would have to fight to finish the degree.
The 1st level is the level of *numbers and variables* . At the end of this level, you would earn the ***medal of courage and bravery*** , which would unlock level 2.
The 2nd level is the level of the *Strings, lists, tuples, dictionaries, and loops*
. The villains here are very nasty and will try to confuse you. At the end of this level, you will earn the ***medal of wisdom*** , which unlocks level 3.
The 3rd level is the level of data flow where you will earn the medal of creation, which unlocks the next level.
The 4th level is the level of *functions and modules* . At the end of this level, you would earn the ***medal of persistence*** .
The 5th level is the level of *the turtles.* At the end of this level, you will earn the ***medal of craft*** , which unlocks level 4.
The 6th and final level is the level where you finally begin to create your games. To enter this level, you need the medals from levels 1to 4. At the end of this level, you earn the ***wizard's badge*** , which shows that you now know the basics of creating games.

Let's hurry and begin our adventure!!

The trademarks that are used are without any consent, and the publication of the trademark is without permission or backing by the trademark owner. All trademarks and brands within this book are for clarifying purposes only and are owned by the owners themselves, not affiliated with this document.

Contents

Checkpoint 1

☼ ☼ The beginning

Welcome to this checkpoint! You may proceed:

Adventure tools

We introduce you to the means of your adventure.
Your computer: this is the most excellent tool of all for this beautiful adventure. Your computer is like a gun in a war game; you can't play without it.
Computers are mighty but very dumb. They have the power to do beautiful things, but they can't do those beautiful things except you command them to do it.
Computers like humans use languages. When you code, you talk to your computer in a language it understands. If you know the language of the machine, you can make it do almost anything you can think of.
There are several languages used by computers. Python is one of these languages.
Python language has several advantages.

1. Several people use it: Several programmers use Python to "talk" to their computers. It is used by big companies, too, like Google. This is very good because it means that your work would quickly be compatible with those of others.
2. It is easy to understand: Python is easily understandable by humans, and this is very helpful because you can tell your computer what you want it to do without confusing it or yourself.
3. It has a library: Python has a library (a place where you borrow books or code in this case) of code that has already been written. With this already written code, you can have references for what you want to create yourself, or you can use this already written code and create something more complex.
4. It's a wonderful tool: with Python, you can create so many great things like games, programs, interfaces, etc.
5. It has an interactive shell: Python has a platform on which you can test your code. This is a playground of sorts to see if the computer understands what you are asking it to do. This is a place where you can practice endlessly with lots of fun.

The End !
I am just kidding!

I hope you are all fired up because I am!
This is where you begin your adventure. You need to follow the following set of instructions to go ahead.

<u>Installation</u>

You should check if your computer is 32 bit or 64 bit.
You check on Windows Os by going to my computer and making a left click on your mouse. You then click on properties.
On Mac Os, you click on the Apple icon then click on about Mac.
You need to download python 3. To do this, download Python 3 for free at https://python.org/ .

<u>Windows OS</u>

Download the latest version of python three which should look like this –
Python 3.8.1 Windows x86MSI Installer
Python 3.8.1 Windows x86-64MSI Installer
Once done with the download, install the program. Once installed, open the python folder and run IDLE.

<u>Mac OS</u>

Download the latest version of python three which should look like this –
Python 3.8.1 Mac OS X 64-bit... (Mac OS X 10.6 and later)
Python 3.8.1 Mac OS X 32-bit... (Mac OS X 10.5 and later)
The file will be downloaded as a “.dmg” file.
Open the file and run the file python.mpkg. This will install the program.
Open the applications folder and then open Python then open IDLE.

<u>Ubuntu Linux</u>

Python comes pre-installed on Linux.
Click the button for the Software Center in the Sidebar (orange bag)
Enter *Python* in the search box in the top-right corner of the Software Center.
Select the latest version of IDLE (using python 3.8) and click Install.
After installing, search for and open IDLE.

☼ ☼ Checkpoint 2

Getting to know python: basic skills and concepts

Welcome, great adventurer! I see you have scaled the hurdles at the first checkpoint. Congratulations. The difficulties here are of a different kind! Do you think you have the strength to pass this checkpoint?
We shall see. This checkpoint takes place in an ancient town called Adak. They used to have dragons, but they don't know. In this chapter, you would get introduced to the people of Adak and their unique way of life.
Good luck, adventurer!

Python Package

Congratulations!! You have installed the python software. Usually, once installed you find python comes with four things:
The IDLE or the interactive environment
The python interpreter
The python manual
The python module documents

We would use mainly the IDLE or interactive environment. You can ignore the others for now.

Open your IDLE and let us begin.
IDLE is a unique text editor dedicated to the computer's needs, and it is essential to a programmer. Other text editors are very fancy, but as a beginner, you only need a straightforward programmer's text editor.

The IDLE or interactive environment or interactive shell shows you the result of whatever you do on it almost immediately. Once you click Enter or Return, it runs your program.
It should be titled **Python 3.8.1 Shell** and once open; it should look something like this:

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license ()" for more information.
```

>>>
For the reason that you will be writing several lines of code, you would need a text editor to write your code. Fortunately, IDLE comes with such a thing.
1. Once you have IDLE opened, go to file; you should see a list.
2. Click new file on that list,
Or simply press Ctrl + N (Command + N on Mac OS). This should open up a blank text editing window named **Untitled** .

You can now type in your code like in the shell, but this time, it wouldn't show a result immediately. Instead, when you click on Enter or Return, you are taken to another line to continue your code.
Other advantages of the text editor include
You can use things like Cut, Copy, and Paste.
You can go through the lines of code using the up and down arrow keys.
You can scroll through your program
Don't worry if a lot of this is confusing. You will get the hang of it as you read on and try what you see. You will make a lot of mistakes, but that is how great programmers are made.

Basic Rules for coding

Print function

The print function lets the python shell or interpreter show you what you have told the computer.
In other words, the print function gives you an output of what you have imputed into the computer or text editor.

1st program

To write your first program, open your blank text editor, and follow the following rules:

Rules

1. Type print: print should be in small letters all through – **p** rint and ~~**not P rint**~~ .

When you type the word **print** , IDLE colors it (usually purple), this is because IDLE is telling you that print is a word it recognizes.

2. Type: Once upon a time in a faraway land should be in 'quotes' e.g.
'Once upon a time in a faraway land.'
Words in quotes are turned green in the IDLE shell or text editor; this its way of telling you that it understands what you are saying it.

3. It should also be in (brackets) e.g.
('Once upon a time in a faraway land')
It should look like this once you are done:
print('Once upon a time in a faraway land')

4. Type Ctrl + S to save
When you do this, you should keep with whatever name you want, but you must end it with **.py** so your python interpreter can read the file.
Examples:

- Dragon story.py
- The city of Adak.py
- Adventure in the forest.py

5. Click on the run then "run module" or just click on F5.
You should see something like this appear on the **Python 3.8.1 Shell**

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license ()" for more information.
>>>
===================== RESTART: C:\Users\uf\Desktop\new.py
=====================
Once upon a time in a far away land
>>>
```

Congrats programmer, you just wrote your first program. You little genius!

<u>Error messages</u>

There is the computer's way of telling you it does not understand what you have said it.
There are two standard error types.
Syntax errors and Exception errors
At this stage, you are likely going t have syntax errors rather than exception errors.

Syntax errors
These are errors that occur when you don't give the computer instructions properly. Common syntax errors are

- Typing **P** rint (with capital letters) instead of **p** rint (with small letters)
- Not using brackets
- hello tom
- Not completing the quotation

print (' hello tom) or
print (hello max ') or
print (hello max)

- Mixing single and double quotes:
- (' hello world ")
- Using minus (-) instead of underscore (_)
- Using different brackets

(},(], [), {)[}, []}(] instead of (), {}, []

Syntax errors show in the text editor. They are not seen in the python shell (except you typed them into the python shell)

Exception errors
These are errors that occur despite you typing the seemly correct code. They usually occur in the python shell and are generally very subtle. The computer often tells you what you have done wrong. You are not likely going to make a lot of this type of error initially.
Examples
- print (20/0)

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license ()" for more information.
>>>
===================== RESTART: C:\Users\uf\Desktop\new.py
=====================
Traceback (most recent call last):
 File "C:\Users\uf\Desktop\new.py", line 20, in <module>
   print (20/0)
ZeroDivisionError: division by zero
>>>
```

- (print 10+ 'book')

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:\Users\uf\Desktop\new.py
=====================
Traceback (most recent call last):
 File "C:\Users\uf\Desktop\new.py", line 22, in <module>
   print (10+ 'book')
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```

When you get errors, you should read the error type it is and try to find out where you make your mistake. You could also highlight the error and run a Google search on it to find the source of the error and a possible solution to it.

PRACTICE

You have made it this far. Good of you. Before you go ahead, you need to go through these practices to find out more about the lives of the peoples of

Adak. Don't skip this.
You should write a few more programs following the rules above.

- There was a peaceful town called Adak.
- Its inhabitants were easy-going people.
- To a newcomer to the town, Adak would seem strange.
- Why? There are these statues all over the place of strange huge animals of various shapes and sizes.
- These statues were made of funny materials that looked and feel like straw.
- Adak is surrounded by a thick forest of trees, which is usually covered with mist.
- The people of Adak are busy; everyone has something to do

Make sure you follow the rules and don't copy and paste, make sure you type them in yourself so you can make your own mistakes and correct them yourself.
If you successfully finished the practice, you have earned yourself two coins form the land of Adak.

Nice one adventurer, you are doing well!

<u>Data types</u>

Everything you type into the text editor or the IDLE shell is data (or input). When you use this data (or information) to form commands for the computer, you are coding. What the computer displays for you after a command is called the output.
Data or Input - Code – Output
There are many types of data, but we will deal with the basic four you need

1. Integers (int)
2. Floating-point numbers or floats
3. Strings or Text (str)
4. Booleans (bool)

Integers
These are whole numbers for example; 1,2,3,4,5,6,7, -1,-2,-3,-4,-756, 10456 etc.
They are used to express certain kinds of data like
Dates: 11-07-2014, 24-05-2017
Phone Numbers: 911, 122, 224
Score in a game: Foxes **5** Rabbits **0** , Player 1 **566** Player 2 **5673,** high score **234562**
Time: 5:56pm, 11:18am etc.

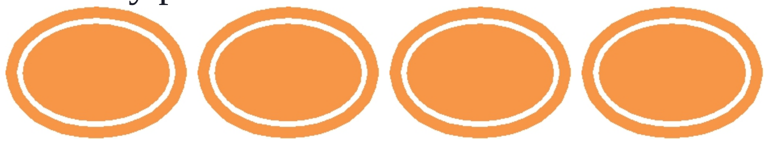
Floating-point numbers
These are numbers with decimal points, for example, 1.75, 23.676, -12.45, etc.
They are used to express data like
The value of pi - 3.14
Floats and integers are similar, but computers can't tell that. They see them as related. Integers can become floats
1 -- 1.0
34 -- 34.0
Note : when using integers or floats, you don't need 'quotation.'

Strings
These are standard texts like the ones you are currently reading. For python to recognize strings, they have to be 'quoted.' The quotation can be single (') or double ("), but the single is easier to use.
Examples
'In Adak, there was no school.'
'Every child was expected to go with his parents to work daily.'
"At the age of 10, every child explores possible jobs by joining each work unit for as long as he or she wants."
"At age 15, every child would have picked his or her field of work."

Boolean

These are statements that can either only be True or False.
Example
You are a boy: you are either a boy or a girl. If you are a boy, the Boolean is exact. If you are a girl, the Boolean is false.
You are asleep: you are either sleeping or awake. Since you are currently reading this book, you are hopefully awake; hence the Boolean is exact.
You have a goat: if you have a goat, the Boolean is exact, but if you don't have a goat, the Boolean is false.
As you read further, you will understand the significance of each of these data types.
The other data types include sound ('wav', mp3), picture (jpeg), videos ('.avi', mp4), variables, lists, dictionaries, tuples etc.
You would come across them as time goes on.

You have reached the end of this checkpoint!
You have earned yourself five coins from Adak.
You may proceed



☼ ☼ Checkpoint 3

Wow! You did it once again.
Welcome to this checkpoint, where you will face the levels.
It would be best if you had some Adak coins by now. As you would have noticed, the Adak coins are shaped like dragon eggs. You will need them in the subsequent chapters.

Level 1
Numbers and Variables
Welcome brave adventurer to the 1st level in this checkpoint. I hope you can win this level to get the coveted **medal of courage and bravery** .

Numbers

<u>**Math Operations**</u>
Math operations are mathematical commands carried out by the computer.
For example:
2 + 3 = 5
2, 3, and 5 are integers while (+) is an operator that signaled the operation 2 plus 3, which gave 5.
If it seems complex, math operations are simple math you are used to.
For you to carry out math operations, you need operators, which are the plus sign (+), minus sign (-), etc.

Operators	Operations
+ plus	**Addition**
- minus	**Subtraction**
/ slash	**Division**
* asterisk	**Multiplication**

% percent
Remainder

** double asterisk
Exponent

// double slash
Integer division

< less-than
Less-than

> greater-than
Greater-than

<= less-than or equal
Less-than or equal

>= greater-than or equal
Greater-than or equal

Note: numbers and floats don't need 'quotes.'
Addition : print (2 + 2) will give you 4 on the python shell
Subtraction : print (3-2) will provide you with 1 on the python shell
Division : print (4 / 2) will give you 2.0 on the python shell
Multiplication : print (5 * 5) will give you 25 on the python shell
The above operations are very straight forward.
Note : that the computer recognizes / (slash) instead of the standard division sign and * (asterisk) instead of the ordinary multiplication sign. X represents X in the computer's language.

Remainder : print (29 % 5) will give you 4
Why? The % sign shows what remains after 29 has been divided by 5.
As such, print (5%5) will give you zero because when you divide 5 by 5, nothing remains.

Exponent : print (2 ** 3) will give you 8
** Means raise to the power of
2 **3 is the same as 2*2*2 = 8
In other words, 2**3 means multiply 2 by itself 3 times.

Integer division: print (29//5) will give you 5
// means divide by but ignore the remainder.
29/5 = 5 remaining 4
29//5 = 5 because the 4 remainings will be ignored by the computer.
The other operations will be discussed later in the book.

Order of Operations

Some math operators carry more weight than others and will be used first if they are used with other fewer important ones.
The order of importance of operators:
Brackets (solve the Math in the brackets 1st)
Exponents
Division
Multiplication
Addition
Subtraction

Examples
10 + 20 * 40
810
Following the order of operators above, the computer will first multiply 20 by 40 = 800 then add 10, which will answer 810.

(10+20)*40
1200
The reason this answer is different is that the computer first adds the numbers in the bracket since brackets are more important than multiplication.
(10+20)=30 then 30*40 =1200.

((10 + 20) * 40) / 5
240
For this example, the computer attends first to the innermost brackets. (10+20)=30. It then attends to the bracket outside ((30)*40) = 1200. This is then divided by 5 (1200)/5= 240.

If we removed the brackets, the result would be different.
10 + 20 * 40 / 5
170
For this one, the division is attended to first
40/5 = 8
Then the multiplication
20*8 = 160
Then the addition
10+160= 170

You can play around with this on your own. Also, when you have Math homework in the future, try to get the computer to solve them for you.

The people of Adak can carry out their trade and daily activities by calculating using the simple operations you have just learned. They can figure how to much money to pay for chicken, cows, grass to feed cows and goats, milk, etc.
It would help if you tried to understand the simple operations because you will be tested on them later to win the **medal of courage and bravery** .

String Concatenation and Replication

The operators are not only used for integers and floating points. They are sometimes used for strings. When the operators are used for strings, they take on new names.

Take +, for example, when used between two strings becomes the string concatenation operator. Don’t worry about the name; you need only know how it works.

Type into your text editor

```
print (‘goat’ + ‘milk’)
```

When you run it, you should get – goatmilk.

Note : you can’t add a string and an integer or a string and afloat. You will get an error message.

The * sign also works with strings. With strings, it is called a replicator. Like before, you don’t need to remember the name; you need to know what it does.
Type into your text editor

```
print ('goat ' * 7)
```

You should get

```
goat goat goat goat goat goat goat
```

Rather than typing
print (‘goat goat goat goat goat goat goat), you asked the computer to replicate or write out goat 7 times.

This also comes with conditions.
You can only use a replicator between a string and an integer.
You can’t use it between string and float or string and string.

Variables

Introduction

Variables are like storage places for data types (or value). Variables are typically used when you want to keep a data type for later use.

For example, number = 15.

The number is the name of the variable that stores the value 15.

Variables can store almost any data type (or value).

Hence you can have integer variables, float variables, string variables, Boolean variables, etc.
We would take the variables one after the other, but first, let us have a look at the rules which govern the naming of variables.

The rules are:
· Variable names must be a continuous word with no space
"Ratface" can, therefore, not be a variable name. It has to be Rat_face or Ratface.

· Variable names can only use only letters, numbers, and the underscore (_) character.
That is, almost any word can be used as a variable name or any word combined with a number.
Examples: a34, box15, toad75, _56, _toad_fart etc.

· Variable names can't start with a number.
As much as variable names can contain numbers, they can't start with them.
If you want your variable to be a number, start with an underscore (_)
Examples: _54, _56, _78, _789, _78, _34.

- Variable names cannot have special characters.
- characters such as - @, #, $, %, ^, &, *, -, /,”

- Variable names are case sensitive.
- Means that a variable named khan with small letters is not the same as a variable named KHAN with capital letters.
- The same goes for Cat and cat. They are different in the eyes of the computer.

Number variables

This includes variables that store integers and floats.
An example of a number variable is this
a = 5
b=6
a is the name of the variable that stores 5 in it.
b is the name of the variable that stores 6 in it.

On your text editor, type the following
a=5
b=6
total = a + b
print (total) You should get 11.

Total is a variable which stored the sum of 2 variables **a** and **b**
(This shows that variables can store other variables.)

You should also have noticed that when we asked total to print, we did not put it in 'quotes'.

We didn't use quotes because variables are different from strings.

Once a word has something stored in it, it is no longer a string.
As a result, ('total') is now very different from (total) since (total) has a + b stored in it.

NOTE : From now on, >>> will show what you should do in your text editor and R will the results you should get in the interactive shell.
Don't type R or >>> into your text editor and don't expect the R to show also when you run a program.

Examples:

A man has a weight of 67kg and a height of 1.86m. Calculate his body mass index (BMI).
BMI= weight / height2

To solve this, set the variable of height and weight with the value given.
Type in your text editor:
Weight = 67
Height = 1.86
(Weight is now a variable containing 67 and Height a variable containing 1.86.)

```
>>>  print (weight/(height*2))
```

It did not work? The reason is because variables are case sensitive.
Try again with

```
>>> print (Weight / (Height*2))
18.01075268817204
```

Don't forget to add your units
18.01075268817204 kg/m2
The example we just worked with used both integer variables (Weight = 67) and float variables (Height = 1.86)
Booleans will be discussed later. Thus we would move on to string variables.

String variables

String variables are variables with strings stored in them.
Example:
>>>Pet = 'cat' (remember that strings must always be in 'quotation')
The variable pet has the string 'cat' stored inside of it.
String variables can be used to do a lot of beautiful things which we will discuss later.
If you remember your operations, you will remember that we can do this:
>>>print (Pet + 'food') (don't forget capital and small letters)
R: Catfood.

Now that you have seen the types of variables, you should know that variables can have what is stored inside of them changed.
Type in the previous example,

```
            >>>Pet = ‘cat’
>>>Pet = ‘dog’
>>>print (Pet)
                                        R: dog
```

The reason for this is that you have replaced the string 'cat' with the string 'dog.'
Therefore, when you asked the computer to print Pet, it forgot about cat and printed dog.
Think of it like this: a variable is like a man that can only remember one thing at a time. If you tell this man "roses are red" he will remember "roses are red" as long as you don't tell him anything else. Once you say "bananas are green or yellow", the man immediately forgets "roses are red" and remembers only "bananas are green or yellow".
The same thing works also happens with number variables.

```
                            Shoe_laces = 2
                            Shoe_laces = 3
                            print (Shoe_laces) will give you 3
```

The number variable has forgotten 2.

Multiple Assignments
You can store different values in different variables on the same line. This helps shorten your programs and make them easier to read.
Example

```
>>> a, b, c = 4, 3, 2
>>> print (a)
>>> print (b)
>>> print (c)
```

You need to separate the variables with commas and do the same with what you are storing in them.
When you run the above example, you will realize that the computer was able to recognize each of the values you stored in each variable.
Always make sure that the number of variables matches the number of values you want to store in them.

Constants

Remember how a variable is like a man who forgets anything he knew before and remembers only what you just told him? A constant is like that man only that a constant will only remember the first thing you tell him, and he will never hear any other thing.

Well, python does not have such. In order not to mistakenly change the value of your variable as you write a program, you should find ways to write the name of your variable, so you don't make mistakes.
A common way to do this is to write out your variable in capital letters or to use a lot of numbers in your variable.

Example

```
>>> NUMBER = 500
>>> n123umber = 450
```

This way, when you write want to assign (store) a new value to a variable, you will not likely pick a variable name you have picked before.

You have earned five Adak coins and the medal of courage!

☼ Level 2

Strings, lists, tuples, dictionaries and loops

Welcome, o' brave one!

Strings

We have discussed strings and what they are. We have also discussed how operators can be applied to them (string concatenation). We would be discussing strings a little more in this section.

In python, we use 'quotation around stings to distinguish them from other things like variables.

Example

```
>>> 'The people of Adak are kind' (single quotes)
>>> "the people of Adak work hard" (double quotes)
```

<u>Quotations</u>

When you don't use the quotations correctly, you will get an error message when you run your code. We have also discussed this earlier.

When you want to use more than one line of text for your code (called a multiline string), use three single quotes (''')

Example

```
>>> print ('"how old am I?
I am 23 years old!'")
```

R : How old am I?

I am 23 years old!

The (''') 3 single quotes allow you to write multiple lines of code while moving to the next line.

It can also be used with string variables.

Example

```
>>> green = '''I like the color green,
It reminds me of nature'''
                    >>>print (green)
                              R : I like the color green,
It reminds me of nature
```

Sometimes, you have to use quotations in the code you are going to write. Example

```
                    >>> print (‘I can’t go to the movies’)
```

When you run the above example, you will get an error message. This is because the computer (python interpreter) will only read the ‘I can’. The other parts of the code will be left unread why?
The computer not being very smart, does not know that the sentence is complete. The computer reads the apostrophe as a quotation mark. It thinks that it has read what you want to type within the complete quotation around I can. The other words outside are thus read as an error message.

The solution to the above problem is simple

```
>>> print(“I can’t go to the movies”)
```

The double (“) quotation is different from the single (‘) quotation hence the computer reads the data between the 2 double quotation and ignores the single quotation.
This solution also works in reverse if you have a sentence with (“) double quotation in it.

```
>>> print (‘Paul said “I am going to school”’)
```
R: Paul said “I am going to school."
This will run very well.

How about when we have both single and double quotation in a sentence?

```
                    >>>print (“peter said “I can’t buy the shoe and the cup; I can only buy the jug”)
                    >>>print (‘peter said “I can’t buy the shoe and the cup; I can only buy the jug’)
```

In this example, using either the single quote or the double quote will not work.

There are 2 solutions to this.
Its either we use the (’’’) 3 single quotes

```
>>>print (‘‘‘peter said “I can’t buy the shoe and the cup; I can only buy the jug’’’)
```

R : peter said “I can’t buy the shoe and the cup; I can only buy the jug

Or we could escape!

An escape is a back slash (\). It is used before the quotation mark within the string.
Example

```
>>> print (“peter said “I can\’t buy the shoe and the cup, I can only buy the jug”)
```

What you are basically telling the computer (the python interpreter) is that you want it to over look the quotation it is placed behind. It is kind of like a shield for the quotation inside the string.
When you use the backslash this way, it would not show in your printed code.

The 3 single quotes are however, better and easier to use, especially when your code has many apostrophes.

<u>**len Function**</u>

You can find the number of characters in a string using the len function.

Examples

```
>>> country = 'Netherlands'
>>> x = len(country)
>>> print(x)
                                        R:11
                    >>> c= 'I am a boy and I love jumping.'
                    >>>print (len (c))
                                         R: 30
```

<u>Indexing Characters</u>

If we think of a string as a list of characters, then we can think of each character as an element.

The index is the position of an element in a list or the position of a character in a string.

Using the earlier example, the string Netherlands has 11 characters.

N
e
t
h
e
r
l
a
n
d
s

0
1
2
3
4
5
6
7

8
9
10

-11
-10
-9
-8
-7
-6
-5
-4
-3
-2
-1

When the computer starts counting strings, it begins its counting at 0 thus to the computer, N is 0.

Because of the fact that there are 11 characters in the string, the last character is found at index 10.

You can also use negative index to access the characters in a string,

An index is the position (or number) of a character in a string. (It is sometimes referred to as a subscript.)

Example:

N
e
t
h
e
r
l
a
n
d

s

0
1
2
3
4
5
6
7
8
9
10

An index is always represented as an integer therefore; each character has an index (number).

We can pick out any character in a string by using its number or position in the string.
Using Netherlands as an example, it has 11 characters but they are numbered 0 – 10.

To pick out each the number of each character in a string, we simply use this formart of code
Variable name [any number]

Example:

```
>>> country = ‘Netherlands’
>>> print (country [0])
>>> print (country [1])
>>> print (country [2])
>>> print (country [3])
>>> print (country [6])
>>> print (country [9])
>>> print (country [11])
Python 3.8.1 (tags /v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license ()" for more information.
```

```
>>>
==================== RESTART: C: \Users \uf \Desktop \new.py
====================
N
e
t
h
l
d
Traceback (most recent call last):
 File "C: \Users \uf \Desktop \new.py", line 13, in < module >
   print (country [11] )
IndexError: string index out of range
```

Running number [11] gave an error. This is because Netherlands has only 11 numbers and python begins to count from 0 and thus has only up to index 10.

We can also do a negative index.

```
N
e
t
h
e
r
l
a
n
d
s

-11
-10
-9
-8
```

```
   -7
   -6
   -5
   -4
   -3
   -2
   -1
```

Negative indices start from -1. Why? There is nothing like -0.
If you remember your number line, you have
……….5, 4, 3, 2, 1, 0, -1, -2, -3, -4, 5…….

Example

```
>>> country = ‘Netherlands’
>>> print (country [-1])
>>> print (country [-3])
>>> print (country [-5])
>>> print (country [-7])
>>> print (country [-8])
>>> print (country [-11])
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py ====================
s
n
l
e
h
N
>>>
```

Generally, negative indices are not used very often.
It could however be very useful to get the last number of a string which is always -1.

If you try to access a character that is not in the string, there will be an error message:

```
>>> print(country[500]) # only has 11 characters

Traceback (most recent call last):
File "<pyshell#12>", line 1, in <module>
print(country[500])
IndexError: string index out of range
```

You could also use indexing for an empty string
Example

```
>>>String = ‘’
            R:
```

When you have an empty string, the len function or indexing will not apply since the length is zero
Example

```
>>> String =''
>>> print(String)
R: 0
```

Try this yourself

```
                        >>> print (string [])
```

<u>Creating sub-strings or string slices</u>

When you are dealing with strings, you sometimes will want to extract a shorter string from a longer string. This is just like taking a slice of bread out of a big loaf of bread.

To make a slice of a string, we have to specify the start index and the end index of the slice we want to create. When you make a slice in a string, the interpreter makes a copy of the characters in the slice but does not remove those characters from the original string.

To specify a slice, the computer uses this framework
string[Index number : Index number]

The character at the first Index is included as the first character of the substring and the number of the last Index, is the index of the first character that is not included in our slice.

Example:
Name = 'Max Weber'
To get just the first name, we want to take a slice starting at index 0 (the M), through index 2 (the x).

To, therefore, create a substring that includes just the first name, we would ask for this slice:

```
>>> print(Name[0:3])
                                        R: Max
```

To get the last name, we would use this slice:

```
>>> name2 = len(Name)
>>> print (name2)
R: 9
>>> print(Name[4 : name2])
R: Weber
```

You can also leave off the starting index of a string like this:
<String [: Index number]

This means to create a slice starting at the first character of a given string, and get to but don’t include the character at index number included.

```
                   >>> print (Name [: 9])
```

Also, you can also use the reverse of the above:
String[Index :]

This means start at the given index and include all characters through the end of the string.

You can also use this syntax:
String[:]

Examples:

```
>>> example = 'This is an example of a string'
>>> print(example[10:])
R: example of a string
>>> print(example[:16])
R: This is an examp
>>> print(example[:])
R: This is an example of a string
```

<u>Changing string case</u>
Another of the things you can do with strings is change the case of the words in a string.

Example:

```
>>> name = "morris johnson"
>>> print (name.title())
R: Morris Johnson
```

In this example, the variable name refers to the lowercase string " morris johnson ".

The method - .title () - comes after the variable in the print () call.

A method is an action that the computer can perform on a data type.
The dot (.) after name in name.title () tells Python to make the title () method act on the variable name.

Methods are followed by brackets () just like in print. Usually, brackets need extra data to carry out their functions. The title function is unique in that the title () function doesn’t need any additional information, so its brackets are empty.

The title () method changes each word to title case, where each word begins with a capital letter.

There are other methods available for changing case.
Examples:
All uppercase and all lowercase letters:

```
>>> name = " Morris Johnson "
>>> print (name.upper())
>>> print (name.lower())
R: MORRIS JOHNSON
R: morris johnson
```

Using placeholders

%s is called a placeholder. It is used like an (underline) _________.
Whenever you type it, you are telling the Python interpreter to put an underline which you will fill later.

This underline is filled later when you use the % sign. The % sign is placed in front of what you want to place on the line you drew with the placeholder (%s).

Example
```
>>> score = 1000
>>> report = 'I scored %s points'
>>> print (report % score)
R: I scored 1000 points
```

Here, you stored 1000 in the variable score and “I scored %s points” in the variable report.

%s acted like as an underline giving this:

I scored _______ points.

On the next line, we use the % symbol to tell the Python interpreter what to place on the ______ by putting % in front of the variable score. This puts the content of the variable score on the line drawn by %s.

W e can also use different variables, as in this example:

```
>>> farm= '%s: are used on the farm '
>>> ab= 'sticks'
>>> cd = 'shovels'

>>> print(farm % ab)
R: sticks: are used on the farm

>>> print(farm% cd)
R: shovels: are used on the farm
```

Here, you made three variables. The first, farm, includes the string with **%s,** which creates an underline. The other variables are ab and cd.
We can print the variable farm, and once again use the % operator to replace it with the contents of the variables ab and cd to produce different messages.

You can also use more than one placeholder (%s) in a string,
Example:

```
                  >>> r = ‘I ride my bicycle all day’
                  >>>s = ‘when it rains.’
                  >>> t = ‘%s, except %s’
                  >>>print (t %(r, s))
                                      R: I ride my bicycle all day, except
when it rains.
```

When using more than one placeholder, be sure to wrap the replacement values in parentheses, as shown in the example.

The order of the values is the order in which they’ll be used in the string.

***F* strings**

This works like the placeholder, but it is much faster. It is used to insert a variable’s value inside a string.

Example

```
>>> first_name = "Morris"
>>> last_name = "Johnson "
>>> full_name = f"{first_name} {last_name}"
>>> print(full_name)
```

To insert a variable’s value into a string, place the letter f just before the opening quotation mark.

These strings are called f-strings. The f is for format.

You can also use f-strings to write complete sentenses using the information associated with a variable,

Example:

```
>>> first_name = "Morris"
>>> last_name = “Johnson "
>>> full_name = f"{first_name} {last_name}"
>>> print(f"Hello, {full_name.title()}!")
R: Hello, Morris Johnson!
```

You can also store what you assign to F strings in variables.

```
>>> first_name = " Morris"
>>> last_name = "Johnson "
>>> full_name = f"{first_name} {last_name}"
>>> message = f"Hello, {full_name.title()}!"
>>> print(message)
R: Hello, Morris Johnson!
```

Format

This was the older and more stressful method of inserting values or variables into strings or other variables before f strings came along.

Example:

```
full_name = "{} {}".format(first_name, last_name)
                                                  R: Morris Johnson
```

This method is too long and wastes a lot of time. It is better to use the f string method.

Adding Whitespaces

Whitespaces are nonprinting characters, such as spaces, tabs, and end of line symbols.

You can use whitespace to organize what you want to come out on the interactive shell so that it will be easier for users to read.
To add a tab to your text, use the character combination \t
Exaample:

```
>>> print("volume")
R: volume
>>> print("\t volume ")
                                        R: volume
```

To add a new line in a string, use the character combination \n:

```
>>> print ("Animals:\n goat \n cow\n pig")
Animals:
goat
cow
pig
```

You can also combine tabs and newlines in a single string. The string "\n\t" tells Python to move to a new line, and start the next line with a tab.

The following example shows how you can use a one-line string to generate four lines of output:

```
>>> print("Animals:\n \t goat \n\t cow\n\t pig")
Animals:
goat
cow
pig
```

Removing Whitespace
Extra whitespace can be confusing in your programs.

The computer can look for extra whitespace on the right and left sides of a string.
To be sure that there is no whitespace at the right end of a string, use the rstrip() method.

```
>>> play = ' '
>>> print (play)
                                        R:
```

```
>>> favorite_language.rstrip()
R
>>> favorite_language
R
```

When you ask the computer for this value in a terminal session, you can see the space at the end of the value.

When the rstrip() method acts on the variable favorite_language at the right, this extra space is removed temporarily.
If you ask for the value of favorite_language again, you can see that the string looks the same as when it was entered, including the extra whitespace.

To remove the whitespace from the string permanently, you have to associate the stripped value with the variable name:
Example:

```
>>> favorite_language = ' '
>>> favorite_language = favorite_language.rstrip()
>>> favorite_language
                                   R
```

To remove the whitespace from the string, you strip the whitespace from the right side of the string and then associate this new value with the original variable, as shown.

You can also strip whitespace from the left side of a string using the lstrip() method, or from both sides at once using strip():
Example:

```
>>> favorite_language = ' '
>>> favorite_language.rstrip()
                                   R
>>> favorite_language.lstrip()

>>> favorite_language.strip()
```

Congrats welcome to lists!

Lists

Introduction

When we have data like this,

```
a = 10
b = 20
x = 30
y = 40
z = 50
```

or

```
c = ‘Max Weber’
d = ‘James Jones’
e = ‘Janet Simpson’
f = ‘John Meyer’
i= ‘Andrew Anthony’
```

We could store the data the way we have stored it with each string and integer having a variable. We could also have the data stored more efficiently called a list.

A list is in programming an ordered collection of things that are stored in a single variable.

Making a list

A list is represented by a variable name, square brackets [], and the elements separated by commas.

```
List0 = [10, 20, 30, 40, 50]
print (List0)
R: [10, 20, 30, 40, 50]
```

```
List1 = [‘Max Weber’, ‘James Jones’, ‘Janet Simpson’, ‘John Meyer’,
‘Andrew Anthony’]
print (List 1)
                    R: [‘Max Weber’, ‘James Jones’, ‘Janet Simpson’, ‘John
Meyer’, ‘Andrew Anthony’]
```

Elements

Elements are items on a list:
Example
List1= [‘Max Weber’, ‘James Jones’, ‘Janet Simpson’, ‘John Meyer’, ‘Andrew Anthony’].
‘Max Weber’ is an element in list 1
‘James Jones’ is an element in list 1
‘Janet Simpson’ is an element in list 1

A list can have as many elements as possible.

A list can have different types of data types stored in it.
Example:

```
>>> mixed = [False, 7, ‘strings', 789.65]
>>> print(mixed)
R: [False, 7, ‘strings', 789.65]
>>> print(type(mixed))
R: <class 'list'>
```

Mixed has a Boolean, an integer, a string and a float stored inside of it.

Empty List
As much as lists can have as many elements as possible, a list can have no elements at all.
Lists like this are called empty lists.
Example:

```
>>> empty = []
>>> print(empty)
R: []
```

Indexing Lists
The position or number of an element in a list is its index, just like with strings.

List1 = [‘Max Weber’, ‘James Jones’, ‘Janet Simpson’, ‘John Meyer’, ‘Andrew Anthony’]

  ‘Max Weber’
  ‘James Jones’
  ‘Janet Simpson’
  ‘John Meyer’

‘Andrew Anthony’

0
1
2
3
4

-5
-4
-3
-2
-1

……….5, 4, 3, 2, 1, 0, -1, -2, -3, -4, 5…….

In a list, you can refer to and deal individually with each element.

An index is always an integer value. Since every element has an index (number), we can refer to any component of a list by using its index, number or position in the list.

Accessing Elements in a List

To access elements on a list, we will use the following method:
Variable [index or number of element]
Examples

```
>>> List2 = [90, 76, 46, 429]
>>> print (List2[0])
>>> print (List2[1])
>>> print (List2[2])
>>> print (List2[3])
        R: 90
   76
   46
   429
```

```
>>>List1 = [‘Max Weber’, ‘James Jones’, ‘Janet Simpson’, ‘John Meyer’,
‘Andrew Anthony’]
```

```
>>> print (List1 [0])
>>> print (List1 [1])
>>> print (List1 [2])
>>> print (List1 [3])
                    >>> print (List1 [4])
```

```
R: Max Weber
   James Jones
   Janet Simpson
   John Meyer
   Andrew Anthony
```

```
List1 = [‘Max Weber’, ‘James Jones’, ‘Janet Simpson’, ‘John Meyer’,
‘Andrew Anthony’]
>>> print (List1 [-5])
>>> print (List1 [-4])
>>> print (List1 [-3])
>>> print (List1 [-2])
                    >>> print (List1 [-1])
```

```
R: Max Weber
   James Jones
   Janet Simpson
   John Meyer
   Andrew Anthony
```

Changing an elementss in a List

You can change or remove an element in a list

Example:

```
>>> fruits = ['apple', 'banana', 'cherry', 'mango', 'cashew']
>>> fruits[2] = 'cucumber'
>>> print(fruits)
R: ['apple', 'cucumber', 'cherry', 'mango', 'cashew']
```

We just changed banana to cucumber by specifying that the computer should remove what is in 2 and replace it with a new thing cucumber.

Lists, unlike strings, can be edited this way. Thus lists are mutable.

Getting the Number of Elements in a List using the len Function

When we need to get the total number of element in a list, we use the len function.

```
>>> fruits = ['apple', 'banana', 'cherry', 'mango', 'cashew']
>>> n = len(fruits)
>>> print('There are', n, 'items in our shopping list.')
R: There are 5 items in our shopping list.
```

List Manipulation
Lists can be mauipulated with In-Built Operations
Examples of such operations are:

List . append(x) it adds x to the end of a list
List.count (x) tells the number of times x was found in the list
List.extend (Other_List) adds all elements in Other_List to list
List.index (x) shows the first index in the list where x is found
List.insert (x, index) Inserts x into the list at position index
List.pop() removes and returns the last item from a list
List.pop(index) Removes and returns the item from a list at the specified index
List.remove (x) removes the first x in the list.
List.reverse () Reverses the position of all the elements in a list
List.sort () Sort elements in a list from lowest to highest

Slicing lists
Like strings, lists can be indexed and sliced:

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
        [1, 4, 9, 16, 25]
>>> squares [0] # indexing returns the item
        1
>>> squares [-1]
        25
>>> squares [-3:] # slicing returns a new list
        [9, 16, 25]
```

When you slice, python returns a new list containing the requested elements.

```
>>> squares [:]
            [1, 4, 9, 16, 25]
```

Lists can also be concatenated like strings.

```
>>> squares + [36, 49, 64, 81, 100]
            [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Other things you can do to lists

Removing Values

```
>>> letters[2:5] = []
>>> letters
            ['a', 'b', 'f', 'g']
```

Clearing a list
You can clear a list by replacing all the elements with an empty list

```
>>> letters[:] = []
>>> letters
            []
```

We can create create lists containing other lists, for example:

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
            [['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
            ['a', 'b', 'c']
>>> x[0][1]
            'b'
```

We can complie all that we have learnt and do some complicated math with list for example the fibonnacci series

```
>>> # Fibonacci series:
 # the sum of two elements defines the next
>>> a, b = 0, 1
>>> while a < 10:
```

```
>>> print (a)
>>> a, b = b, a+b
```

R: 0 1 1 2 3 5 8

Using the del to remove an element
You can use the del statement to remove an elelment if you know the position of the element
you want to remove from list.
Example
```
cars = ['ford', 'toyota', 'honda']
print(cars)
del cars[0]
print(cars)
```
R: [‘toyota’, ‘honda’]

```
>>> cars = ['ford', 'toyota', 'honda']
>>> print(cars)
>>> del cars[]
>>> print(cars)
```
R: [‘ford’, ‘honda’]
The second motorcycle is deleted from the list:
['honda', 'yamaha', 'suzuki']
['honda', 'suzuki']

In both examples, you can no longer access/use the element that was removed from the list after del was used.

Removing an element Using the pop() Method
When you use the del method, you can use the item you removed. When you use the pop method, you will be able to use the element you removed.

The pop() method removes the last item in a list, but it lets you work with that item after removing it.

Example:
Let’s pop a motorcycle from the list of motorcycles:
```
>>> cars = ['ford', 'toyota', 'honda']
>>> new_cars = cars.pop()
>>> print (cars)
>>> print(new_cars)
            R: ['ford', 'toyota']
honda
```

Since we used the pop method, the hondaa, which we removed, is still a usable element.

You can use pop() to remove any element from any position in a list by adding in front of that element its index (number)

Example

```
>>> cars = ['ford', 'toyota', 'honda']
>>> x_car =cars.pop(0)
>>> print(f"The first car I owned was a {x_car.title()}.")
                                                    R: The first car I owned was a Ford.
```

Removing an element by its name

In cases where you don't know the position or index of the component, you want to remove from a list, you can still remove the part as long as you know its name.

You can do this by the remove() method.

Example

```
>>> cars = ['ford', 'toyota', 'honda']
>>> print(cars)
>>> cars.remove('honda')
>>> print(cars)
R: ['ford', 'toyota', 'honda']
['ford', 'toyota']
```

Lists and Operators

We can join lists by adding them, the way we did with strings; using a plus (+) sign.

Example

```
>>> list1 = [5, 6, 7, 8]
>>> list2 = ['A ', 'trip ', 'to ', 'the ', 'moon ', 'might be ', 'fun']
>>> print (list1 + list2)
R: [5, 6, 7, 8, 'A ', 'trip ', 'to ', 'the ', 'moon ', 'might be ', 'fun']
```

Or

```
>>> list1 = [5, 6, 7, 8]
>>> list2 = ['A ', 'trip ', 'to ', 'the ', 'moon ', 'might be ', 'fun']
>>> list 3= list1 + list2
>>> print (list 3)
R: [5, 6, 7, 8, 'A ', 'trip ', 'to ', 'the ', 'moon ', 'might be ', 'fun']
```

We can multiply a list by a number like with strings too.
Example:

```
>>> list1 = [3, 4]
>>> print (list1 * 5)
R: [3, 4, 3, 4, 3, 4, 3, 4, 3, 4]
```

Division and subtraction will give errors just like with strings

Tuples

A tuple is a list that uses normal brackets () instead of square brackets:
Example:

```
>>> Fibonacci = (0, 1, 1, 2, 3)
>>> print(Fibonacci [3])
R: 2
```

The contrast between tuples and lists is that tuples are not changeable or mutable.
Example

```
                    >>> Fibonacci [0] = 15
R: Traceback (most recent call last):
File "<pyshell>", line 1, in <module>
Fibonacci [0] = 4
TypeError: 'tuple' item does not support item assignment
```

Why would you use a tuple instead of a list? Because sometimes it is useful to use something that you know can never change. If you create a tuple with two elements inside, it will always have those two elements inside.

Dictionaries

Another thing like a list or tuple is a dictionary (dict).

The difference between dictionaries and lists or tuples is that each item in a dictionary has a key and a corresponding value.
Example

Sports played by people: 'Ronaldo' plays 'Football', 'Jordan' plays 'Basketball', 'Ruth' plays 'Baseball', 'Clarke' plays 'Netball', 'John' plays 'Badminton', 'Bridge' plays 'Rugby'.

When represented in a dictionary looks like this:

```
>>> sports = {'Ronaldo': 'Football',
'Jordan': 'Basketball',
'Ruth' : 'Baseball',
'Clarke' : 'Netball',
'John' : 'Badminton',
'Bridge' : 'Rugby'}
```

We use colons to separate keys from their value, and each key and value is surrounded by single quotes.
Also, the elements in a map are enclosed in braces ({}), not standard brackets () or square brackets [].

Now, to get Clarke’s sport, we access our dictionary - sports with the name as the key,

```
>>> print(sports['Clarke'])
R: Netball
```

You can remove or replace values in a dictionary.
To delete a value in a dictinary, we use its key. For example, here’s how to remove john:

```
>>> del sports['john']
>>> print(sports)
R:{'Ronaldo': 'Football', 'Jordan': 'Basketball', 'Ruth': 'Baseball', 'Clarke':
'Netball',        'Bridge': 'Rugby'}
```

We can also use a key to replace values in a dictionary :

```
>>> sports['Ruth'] = 'Ice Hockey'
>>> print(sports)
R: {'Ronaldo': 'Football', 'Jordan': 'Basketball', 'Ruth': 'Ice Hockey', 'Clarke':
'Netball', 'Bridge': 'Rugby'}
```

We replaced the sport of baseball with Ice hockey by using the key Ruth.

We can do many other things with Dictionaries, but we cannot concatenate them. Doing that will give and error message:

Example

```
>>> print (sport + sport)
R: Traceback (most recent call last):
```

```
File "C:\Users\uf\Desktop\new.py", line 56, in <module>
    print (sports + sports)
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

Loops

A loop is a piece of code that is repeated until certain conditions are met.
An example of a loop is this:
Wake up
Eat
Sleep
Repeat

A lot of us people carry out this loop daily.
In other words, a loop is a circle that goes on and on.

Another example of a loop is this.

Wake up
Dress up
Go to school
Go back home
Sleep
Except on Saturday and Sunday

Pseudocode

When you wish to write a program, pseudocode is what you first write down. It is the expression of your code in standard English.
It allows you to think about how you want to solve the problem without first worrying about the details of the code.

Example:
Pseudocode for a game:

```
# Introduction
# get a random number
# create a loop
# ask for programmer input
# tell user result
# ask user to play again
# congratulate user
```

For another game

```
# design interface
# create ball
# create bounce pads
# make ball move
# create scoring system
# create winning criteria.
```

Now under each of these sections, the code needed for the computer to carry out these actions is written.
Pseudocode helps you to easily correct mistakes in your code as you can easily find where an error or a glitch is coming from. It also helps your code to be orderly and neat and also make it easy for other programmers to understand your code.

When you use pseudocode inside your text editor, you need to make it invisible to the computer or python interpreter. For this, you need the hash sign.

Hash Sign and the triple comma
The hash sign and the triple comma are used to make things that you write in your text editor invisible to your python interpreter.

Example

```
>>> #print (‘I love shopping’)
>>> print (“I love volley ball”)
```

R: the first print is not obeyed because it is invisible to the computer. The most critical uses of this is writing pseudo code.

The (,,,) triple comma is used to make large blocks of code invisible to the python interpreter.
Example

```
                              >>> ,,,
>>> #print (‘I love shopping’)
>>> print (“I love volley ball”)
>>> print (‘I will talk’)
>>> print (‘I will run’)
>>> ,,,
>>> print (‘football’)
                R: football
```

This is because the parts of code in between the triple commas above and below are invisible to the computer.

While Loops

The while loop consists of a ‘while’ statement and a piece of code that is to run if the while statement is true.

The while statement is often a Boolean. The piece of code that follows the while statement is commonly referred to as the body of the loop.
The bit of code that comes after the while statement is indented. This is commonly done with 4 spaces rather than a tab.
When you type the while statement, however, IDLE automatically indents for you.
When you are finished entering the statement(s) that make up the body of the loop, you can press Backspace (Windows) or the Delete (Mac) key to move the indenting level back four spaces.

Examples:

As long as the Boolean expression in the while statement evaluates to True, the comments in the body of the loop are repeated.
If the Boolean definition evaluates to False, the body of the loop is skipped, and execution continues with the first statement after the shape of the loop.

Example:
In the following code, we’ll ask the user to type the letter a, and we’ll keep asking until the user types an a:

```
>>> looping = True
>>> while looping == True: answer = input("Please type the letter 'a': ")
>>> if answer == 'a':
>>> looping = False # we're done
>>> else:
>>> print("Come on, type an 'a'!)"
>>> print("Thanks for typing an 'a'")
```

@@@
Before the loop starts, we set a Boolean variable named looping to True.
The Boolean expression in this while statement compares the Boolean to the value True. We also could also have written it as follows:

```
looping = True
while looping:
```

This would work the same way because comparing a Boolean to True is the same as just the value of the Boolean itself.
@@@

At the end of the loop, the Python interpreter automatically goes back to the while statement at the top of the loop.
As long as the variable that is looping has a value of True, the code in the body of the loop will be copied.
When we want to end the loop, something (a piece of code) in the body of the loop must make the loop false.

@@@
In this loop, when the user types the letter a, we set looping to False.
@@@

When this piece of code that falsifies the whole loop is reached, the while statement of the loop is rendered false, and we then exit the loop.

For the above reason, the Boolean expression in the while statement is also called the exit condition — the condition under which you can exit the loop.

If we don’t have any code that changes the exit condition, we would have on our hands an infinite loop (a loop that keeps running without stopping)

Trying Loops in a Real Program

The way the program works is this:
The program will ask the programmer for a number.
The program will then add all the numbers below the picked number to the chosen number.

That is when the programmer enters 9; then the computer will add 1, 2, 3, 4, 5, 6, 7, and 8 to 9 and then give an answer.

```
 #first loop program
x = input('Enter a target number: ')
x = int(target)
y = 0
your_number = 1

while your_number <= target:

# add in the next value
y = y + your_number #add in the next number
your_number = your_number + 1
print('your answer to', x, 'is:', y)
```

First, we get a number from the programmer and get it converted to an integer.
We then set y to 0; y is to hold the total of all the numbers.

We also set the your_number variable to 1. The your_number variable will be used to add the numbers from 1 to the number that the programmer entered.

What followed was that they wrote our while statement.

We specified that the loop should keep going until the your_number is higher than the number in x. When this happens, the while statement becomes False, and the loop is exited.

Every time through the loop, we add the value of your_number to x.

Eventually, we add one to what is in your_number to get to the next number.
This piece of code is the key to exiting the loop because the code will stop once the while statement (while your_number <= target) is no longer true.

When you run the program, for example, with 8.

You will get
R: your answer to', x, 'is:', y) ???

Increment and Decrement

The program we you just wrote,

your_number = your_number + 1

It is a piece of code that adds your_number to 1 and puts in the result into the variable your_number.
This method of increasing the value of a variable is called an Increment.

For the above program, we had an increment by 1, but an increment can be by any number.

Our above example is a way of incrementing a variable. There is another way to do the same thing.

your_number + =1

In this method, you use the plus and the equals operators, and it gives you the exact same thing as this your_number = your_number + 1.

The opposite of an increment is a decrement.

A decrement is when a variable subtracts from itself.
Example
your_number = your_number - 1
or
your_number -= 1

Built-in Packages

Python language has some keywords (if, elif, else, while, def, and a few more) and built-in functions (int, str, input, and so on).
Apart from these, Python also has some prewritten packages of code that are built-in and are available to programmers.
These packages were installed on your computer when you installed Python.
They are what is called the Python Standard Library.

There are also “external packages” written by older and more experienced programmers. They make their codes available to other programmers.
External packages are downloaded separately from the internet.
Examples of external packages are Loguru, Monkeytype, Pyright, PyGame, mp.py, etc.

These external packages help you to do different things. PyGame, for example, helps you to build games

When you want to use a build in the package, you have to tell python by using ‘import’
Example
import <packageName>

Generating Random Numbers

Random package is an in built package that is used to create random numbers.
It contains several functions that allow it to generate random numbers.
Since it is a built-in package, you already have it on your computer.

To import random package, type into your text editor:
 import random

When you write a program that uses an import statement, you typically place any import statement(s) at the top of your code. If you want to see the documentation of all the functions that are available in this package, you can call the built-in help function and pass in the name of the package, like this:

```
help(random)
```

If you do this, you will get screens and screens worth of documentation. In you are truly interested in the details of all the functions, feel free to read through this documentation. There are a large number of functions that you can call in the random package. For now, we are interested in one specific function named randrange.

The purpose of randrange is to generate a random integer number within a given range. randrange is interesting because the range itself can be specified in a number of different ways, with different numbers of arguments. Using the most straightforward form, we’ll call the randrange function specifying the range as two integers. Here is the way we call it:

```
random.randrange(<lowValue>, <upToButNotIncludingHighValue>)
```

You start by specifying the name of the package—in this case, the word random. After the package name, you type a period (generally read as “dot”). After the dot, you specify the function you want to call; in this case, you type randrange to say that you want to use that specific function. In the preceding line, randrange expects to be called with two arguments: a low-end value and a high-end value. The low-end value is included in the range, but the high-end value is not included in the range. The way that we say this is “up to but not including” the high-end value. (We’ll see this “up to but not including” concept many times in Python.)

The function returns an integer within the specified range. The most typical way to use randrange is in an assignment statement, where you save the returned value in a variable, like this:

```
<resultVariable> = random.randrange(<lowValue>,
<upToButNotIncludingHighValue>)
```

Here are some examples:

```
#random between 1 and 10
aRandomNumber = random.randrange(1, 11)
```

```
#random between 1 and 52, to pick a card number from a deck
anotherRandomNumber = random.randrange(1, 53)
```

The critical thing to remember (which may seem very odd) is that the second argument needs to be one more than the top end of your intended range. That’s because the number you specify here is not included in the range.

As an alternative syntax, you can call randrange with only a single argument: the “up to but not including” high end. If you make this call with only the one argument, randrange assumes that the low end of your range is zero:

```
#random between 0 and 8
myRandomNumber = random.randrange(9) # same as random.randrange(0,
8)
```

Flipping a Coin!

Now it’s time for a good example program that uses random numbers. We will simulate flipping a coin a number of times. The program will run in a loop. Each time through the loop, we randomly generate a 0 or 1. Then we’ll do a mapping. That is, we’ll say that if we randomly get a 0, that means tails. If we get a 1, that means heads. When the loop finishes, we report the results:

```
# Coin flip program
import random
nFlips = 0 # to count the number of flips
nTails = 0 # to count the number of flips that came up as tails
nHeads = 0 # to count the number of flips that came up as heads
maxFlips = input('How many flips do you want to do? ')
maxFlips = int(maxFlips)
while nFlips < maxFlips:

# Randomly choose 0 or 1, because a coin flip can only result in one of
two answers
# (heads or tails)
zeroOrOne = random.randrange(0, 2)
```

```
# If we get a zero, say that was a heads
# If we get a one, we say that was a tails
if zeroOrOne == 0:
nTails = nTails + 1
else:
nHeads = nHeads + 1
nFlips = nFlips + 1
print()
print('Out of', nFlips, 'coin tosses, we had:', nHeads, 'heads, and', nTails,
'tails.')
```

Notice that we didn’t randomly pick heads or tails directly. We randomly picked from a range that encompasses all possible outcomes, and then mapped the numeric answer to the outcomes we were looking for. In this case, there are only two possible outcomes, so we get random values of 0 or 1, map 0 to tails, and 1 to heads.

Other Examples of Using Random Numbers

Another example of this approach is if we were writing a program to play the game of rock-paper-scissors. In this case, there are three possible choices. To make a random choice, we would generate a random number between 0 to 2 (or 1 to 3, or in fact, any range of three consecutive numbers), and use the resulting number to make our choice:

```
import random
choiceNumber = random.randrange(0, 3) # to get a 0, 1, or 2
if choiceNumber == 0:
randomChoice = 'rock'
elif choiceNumber == 1:
randomChoice == 'paper':
else: # not zero and not one, must be 2
randomChoice == 'scissors'
```

Here we use an if/elif/else statement to account for all possible numbers generated by calling random.randrange, and we set another variable to a string representing the actual choice.

```
randomAnswer = random.randrange(0, 8) # pick a random number between
0 and 7
```

```
if randomAnswer == 0:
print('It is certain.')
elif randomAnswer == 1:
print('Absolutely!')
elif randomAnswer == 2:
print('You may rely on it.')
elif randomAnswer == 3:
print('Answer is foggy, ask again later.')
elif randomAnswer == 4:
print('Concentrate and ask again.')
elif randomAnswer == 5:
print('Unsure at this point, try again.')
elif randomAnswer == 6:
print('No way, dude!')
else: # must be 7
print ('No, no, no, no, no.')
```

Now it should be obvious how this works. We generate a random number between 0 and 7 using random.randrange, and then we use an if/elif ... elif/else to pick a message to print, based on the random number that was chosen.

Using break to Exit a Loop

To exit a while loop directly without running any remaining code in the loop, regardless of the results of any restrictive test, use the break statement. The break statement directs the movement of your program; it can be used to command whatever lines of code are executed, and that isn’t, so the program only runs code that you want when you want it to.
For example, contemplate a program that demands the user about places they’ve visited. The while loop can be stoped in this program by calling break as soon
as the programmer types the 'quit' value:

```
country.py prompt = "\nPlease enter the name of a country you have
visited:"
prompt += "\n(Enter 'quit' when you are done.) "
u while True:
city = input(prompt)
if city == 'quit':
```

```
break
else:
print(f"I'd love to go to {country.title()}!")
```

A loop that starts with while True u will run until it reaches a
break statement. The loop in the program continues asking the programmer to enter
the names of the country they’ve been to until they type 'quit'. When they enter
'quit', the break announcement runs, causing Python to exit the loop:

```
Please enter the name of a country you have visited:
(Enter 'quit' when you are finished.) Unite the States of America
I'd love to go to New York!

Please enter the name of a country you have visited:
(Enter 'quit' when you are finished.) Spain
I'd love to go to Spain!
Please enter the name of a country you have visited:
(Enter 'quit' when you are finished.) quit
```

Note You can use the break announcement in any of Python’s loops. For example, you could use
a break to quit a for loop that’s working through a list or a glossary.

Using continue in a Loop

Instead of breaking out of a loop entirely without executing the rest of its code, you can use the continue announcement to return to the beginning of the
loop based on the result of a restrictive test. For example, look at a loop that calculate from 1 to 10 but prints only the odd numbers in that range:

```
counting.py current_number = 0
while current_number < 10:
u current_number += 1
if current_number % 2 == 0:
continue
print(current_number)
```

First, we set current_number to 0. As it’s less than 10, Python runs the while loop. Once in the loop, we increment the count by 1
at u, so current_number is 1. The if proclamation then
run the modulo of

current_number, and 2. Assuming that the modulo is 0 (which means current_number is
divisible by 2), the continue proclamation signals Python to ignore the rest of
the loop and return to the beginning. If the digits are not divisible
by 2, the rest of the loop is completed and Python prints the current number:

```
1
3
5
7
9
```

Avert Infinite Loops

For example, this loop should count from 1 to 5:

```
counting.py x = 1
while x <= 5:
print(x)
x += 1
```

User Input and while Loops

But if you unintentionally omit the line x += 1 (as shown next), the loop will run forever:

```
# This loop runs forever!
x = 1
while x <= 5:
print(x)
```

Once the value of x will start at 1 but never change. As a result, the conditional
test x <= 5 will always evaluate to True, and the while loop will run
forever, printing a series of 1s, like this:

```
1
1
1
1
--snip--
```

Every coder accidentally writes an infinite while loop from time
to time, specially when a program's loops have subtle exit conditions. If
your code gets stuck in an infinite loop, press ctrl-C, or just shut the

terminal window down.
To avoid typing infinite loops, test every while loop and be sure the loop stops when you expect it to. If you want your program to stop when the user enters a specific input value, run the program, and enter that value. If the program doesn’t end, check the way your program handles the value that should cause the loop to exit. Make sure at least one bit of the program can make the loop’s condition False or cause it to reach a break statement.

Running a Program several times

We can decide to run the program we wrote above in such a way that the program will ask the programmer if he or she wants to rerun the program. The programmer will respond with a yes or a no depending on his or her wishes.
If the programmer responds with a yes, then the program runs again, or the loop restarts, and if the programmer responds with no, the loop stops, and the program ends.
We are going to add this feature to our program in this section.

One easy way to do this is to place the significant parts of the program inside a function then build a loop.
In the body of the loop, we call the function, and at the end of the loop, we ask the user if they want to go again.

```
# Calculate x
def calculateSum(x):
y = 0
your_number = 1
while your_number <= target:

# add in the next value
y= y + your_number

#increment
your_number = your_number + 1
return y
ans = 'yes'
```

```
# start off with the value 'yes' to go through the first time

while ans == 'yes':
x1 = input('Enter a number: ')
x1= int(x1)
y1 = calculateSum(x1)

# call our function and get back the answer

print('your answers to', x1, 'is:', y1)
ans = input('Do you want to try again (yes or no): ')
print('thank you')
```

You should realize that all the significant calculations (generating the sum) are done inside of a function, and the function returns the answer.
The main code is mainly for interacting with the user and calling the function.

Also, if you want to run the whole program a certain number of times, you could modify the looping conditions to count the number of times through a loop:

Example:

```
nTimes = 0

while nTimes < 4:
x1 = input('Enter a target number: ')
x1 = int(x1)
y1 = calculateSum(x1)
print('your answers to', x1, 'is:', y1)

nTimes = nTimes + 1
print('thank you')
```

This method allows the user loop to run four times, and this time, the user will not be asked if he or she wants to go again.

Creating an Infinite Loop

Loops are stooped when the Boolean in them is made to be false.

Earlier, we said that the Boolean expression in the while statement is called the exit condition—that is, the test for exiting the loop is done in the while statement.

So far, we’ve shown that the way to handle this is to write some tests, typically in an if statement, where you determine whether you are ready to exit the loop. If you are ready to exit, you set some variable to a known value that will later be checked in the Boolean expression of the while statement. Here is an example:

```
looping = True
while looping:
<statement(s)>
if <found exit condition?>:
looping = False # found the exit condition at this point
else:
<continuing statement(s) inside the loop>
```

In effect, you have found the exit condition, but you can’t exit the loop until execution goes back to the while statement. Unfortunately, this style often makes it more challenging to write the continuing part of the loop that follows. The code that you run if the exit condition has not been reached must get indented.

If you need to detect and handle multiple exit conditions, each if statement would set the same variable (that is later checked in the while statement), and the code that continues the normal execution gets indented further. This excessive indenting makes it challenging to write and even more challenging to read through the standard path through the loop. Fortunately, there is another way to build a while loop.

New Style of Building Loops: while True, and break

I said that as long as nothing changed the value of the Boolean expression in the while statement, the loop would run forever. Therefore, the simplest way to create an infinite loop is like this:

```
while True:
<statement(s)>
```

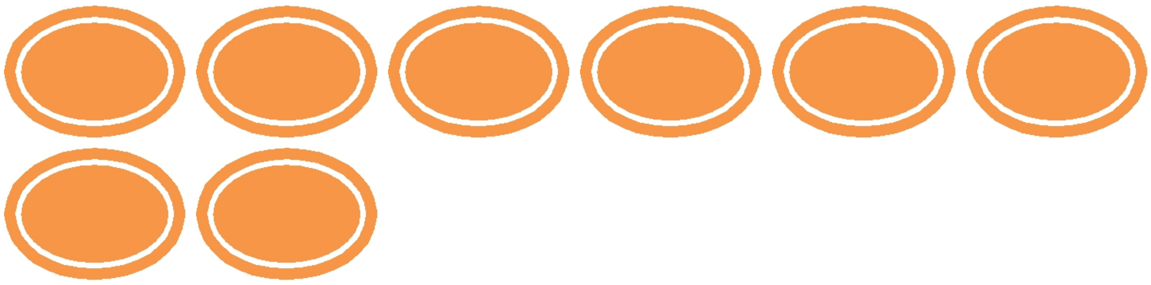
That loop would run forever. Python provides another statement called the break statement, which is made up of just the word break. If your code is running in a while loop, and a break statement is reached, control is immediately transferred to the first statement past the last line of the loop. With the addition of the break statement, we can now think differently about writing loops. Rather than checking for the exit condition in the while statement, we can check for an exit condition anywhere in the body of the loop. If we find an exit condition, we use a break statement to exit out of the loop right at that point.

Here is a simple example:

```
while True: # loop forever
line = input("Type anything, type 'done' to exit: ")
if line == 'done':
break # transfers control out of the loop
print('You entered:', line)
print('Finished')
```

This code allows the user to type anything they want. The program keeps asking the user to type something until the user types the word done. When the user types did, the program detects it and exits the loop immediately, using a break statement. If the user types anything else, the program skips over the indented break statement and prints out a copy of whatever the user entered. Therefore, there is no need to code an else statement for the continuing statement(s) in the loop.

Congratulations on completing this level.
You have earned eight Adak coins!!



● Level 3

Data Flow

Conditional statements: if, elif and else statements

if statement

The ‘if’ statement in python selects a particular action to perform. For example, if Michael scores 10 points, give him candies. The ‘if’ statement can also contain other if statements in a nested fashion, this possibility is known as nested ‘if’ statements.

When the ‘if’ statement runs on a python environment, python executes the code inside the ‘if’ statement if the condition stated evaluates to be true else the code in the ‘if’ statement is not executed. The general format of the ‘if’ statement is stated below.

```
if condition:
statement
```

Basic examples

```
score = 10
if score == 10:
print(‘Give Michael candies’)
```

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright" or "license()" for more advice.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py
=====================
Give Michael candies
>>>
```

What happens when Michael’s score is not 10?

This leads us back to the second part of this lesson.

else statement

The else statement is a statement that performs actions when the conditions stated for the ‘if’ statement evaluates to be false.

Let’s go back to Michael’s score. If Michael happens to score points lesser or greater than 10, what happens when you run the ‘if’ statement in the python environment with score = 9.

```
score = 9
if score == 10:
print(‘Give Michael candies’)
```

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "license()" for more information.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py
====================
>>>
```

As you can see, nothing happened.
This is because the score 9 does not equal to 10, so the statement inside the ‘if’ statement could not be executed.
Now let’s try including an else statement after the if statement and see what happens

```
score = 9
if score == 10:
print(‘Give Michael candies’)
else:
print(‘Tell Michael to try better next time’)
```

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright" "license()" for more information.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py
====================
Tell Michael to try better next time
>>>
```

So moving on, take for instance, Michael scored more than 10 points, which is the requirement for getting candies, the if and else statements in the

previous lesson would tell Michael to try better next time. But Michael scored higher than 10 points and left to me, and I think he deserves more candies for scoring better.
This problem can be rectified by introducing an elif statement to the previous if and else statements.

elif statement

elif statements perform alternative actions specified after the if statement.
Now let’s use the elif statement to make Michael get the reward he deserves if he scores more than 10 points.

```
score = 11
if score == 10:
print(‘Give Michael candies’)
elif score > 10:
print(‘Give Michael extra candies’)
else:
print(‘Tell Michael to try better next time’)
```

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
,32 bit (Intel)] on win32
Type "help", "copyright", "license()" for more information.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py
====================
Give Michael extra candies
>>>
```

Now Michael is getting the extra candies he deserves, and everyone is happy.

Let’s play a little game

Guessing game

This game is called the guessing game. We’ll have to write the code to implement this game together using simple if, elif, and else statements and a function called random.

Now let’s copy the following code into our python environment. Please make sure you indent properly when it is necessary.

```
import random
print('TYPE A NUMBER BETWEEN 1 AND 9 INCLUDING 1 AND 9')
myNumber =int(input('Guess a number:'))
randomNumber = random.randint(1,9)
print("Computer's input is", randomNumber, "and you guessed",
myNumber)
if myNumber > randomNumber:
print ('You guessed too high')
elif myNumber < randomNumber:
print ('You guessed too low')
else:
print('Your guess is correct')
```

Now let's test and see if it works.

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "license()" for more information.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py
=====================
Guess a number: 3
Computer's input is 9 and you guessed 3
You guessed too low
>>>
```

It worked perfectly. Now let me explain what happens in each line of code so that you'll be able to replicate it with ease next time.
The first line of the code *import random* imports the library which imports random numbers or digits or any other valid variable.
The second line of the code *print ('ENTER A NUMBER BETWEEN 1 AND 9 INCLUDING 1 AND 9')* prints an instruction to the player of the game that he/she should only guess between 1 and 9

The third line of the code *myNumber = int(input('Guess a number: '))* collects the player's input. input() is used to collect any string input from the user, while the int just behind the input is used to convert the string input to an integer input.

Why am I doing this conversion?
I am sure you’re wondering this. The reason is because I can’t compare a string with an integer, I can only compare two strings or two integers, hence the need to do the conversion.

The fourth line of the code *randomNumber = random.randint(1,9)* uses the random library to get random numbers between 1 and 9 by using the *randint* together with random in the form specified above.

The fifth line of the code *print(“Computer’s input is”, randomNumber, “and you guessed”, myNumber)* prints out the number that was randomly gotten when the random library was used and the number that the player guessed initially (number obtained from the user).

The sixth line of the code *if myNumber > randomNumber:* is an ‘if’ statement that checks whether the player’s guess is greater than the computer’s input

The seventh line of code *print ('You guessed too high')* is the statement that is executed when the ‘if’ statement is true.

The eighth line of code *elif myNumber < randomNumber:* is an elif statement that checks whether the player’s guess is lesser than the computer’s input.

The ninth line of code *print ('You guessed too low')* is the statement that is executed when the ‘elif’ statement is true.

The tenth line of code *else:* is an else statement that checks every other possibility that can occur and the only option left in this case is checking whether the player’s guess is equal to the computer’s input.

The last line of code *print ('Your guess is correct')* is the statement that is executed when the ‘else’ statement is true.

From the explanation, I hope you’ve been able to understand each line of code written in order to develop the guessing game.
You can now play the game as much as you like and giving it a twist of your own by increasing the range by which you can guess from 9 to 20.

Here is an assignment for you

Write code using the if, elif, and else statement that can collect the score of a user and use that score to grade the user.
Hints:
Users that score below 40 should get an ‘F’
Users that score between 40 and 50 should get a ‘D’
Users that score between 50 and 60 should get a ‘C’
Users that score between 60 and 70 should get a ‘B’
Users that score between 70 and 100 should get an ‘A’
Users should not be able to score above 100

This is the end of this lesson. I hope you’ve been able to understand how conditional statements work and how to write simple conditional statements.

The ‘for’ loop

The ‘for’ loop is a python iterator, just like the while loop. It works on strings, lists, tuples, and dictionaries as well as other built-in objects that can be iterated through

In this lesson, we’ll be learning how to use the ‘for’ loop to solve fundamental mathematics problems and doing other exciting stuff.
Don’t be scared because I mentioned mathematics; you do not need to know a lot of mathematics before you can use the ‘for’ loop.

The generic format of a ‘for’ loop can be seen below.

```
for target in objects:
    statements
```

Before we proceed to learn more about ‘for’ loops, there is an actual need to know about the ‘break’, ‘continue’ and pass statements

Anywhere you see the ‘break’ statement in a loop or a conditional statement, it is used to terminate or exit that loop or conditional statement
Let’s take an example

```
for x in range(1,8):
    if x == 5:
        break
    else:
        print(x)
```

The result can be seen below that when x actually became 5, the ‘for’ loop was terminated.

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "license()" for more information.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py
====================
1
2
3
4
>>>
```

Also, wherever you see the ‘continue’ statement in a loop or conditional statement, it is used to jump to the top or beginning of that loop or conditional statement.

Let’s also take an example here

```
for x in range(1,8):
    if x == 5:
        continue
    else:
        print(x)
```

It can be seen from the result below that the code jumps the number 5 and proceed to print other numbers.
Note that range(1, 8) is a built-in function which contains number 1 to 7 which could be iterated though using a loop.

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "license()" for more information.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py
====================
1
2
3
4
6
7
>>>
```

The ‘pass’ statement is often used when there is nothing useful to say or do in a conditional or loop statement. It is commonly used to code an empty body for a compound statement. For example

```
number = 3
if number == 3:
    pass
else:
    print ('The number is not 3')
```

It can be seen from the result below that the code did precisely nothing.

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "license()" for more information.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py
====================
>>>
```

Let’s take a few practical examples
Let’s try to calculate the mean of different numbers in a list using the ‘for’ loop. I am sure you all remember what a list is and how to use a list and perform different operations using a list from previous lessons.

From our mathematics class in school, do we still remember how to calculate mean? If the answer is No, there is no need to worry, I explain it to you.
In order to calculate the mean of some scores, you have to find out how many scores there are to find the way. Say you have to find the way of these scores, “2, 6, 8, 12, 16, 20, 10, 5, 8, 7”
You count the number of ratings given, and right here, we have 10 scores.
The next step is to add these scores together
The last step is to divide the sum of the scores by the number of scores given
And there you have your mean calculated
Mean = (2 + 6 + 8 + 12 + 16 + 20 + 10 + 5 + 8 +7)/10
This should give the final answer 9.4
Now let’s use python to solve it
Given the list, scores = [2, 6, 8, 12, 16, 20, 10, 5, 8, 7]
The code for calculating the mean can be seen below.

```
scores = [2, 6, 8, 12, 16, 20, 10, 5, 8, 7]
length = len(scores)
summation = 0
for score in scores:
    summation += score
mean = summation/length
print(mean)
```

Now let’s test and see if it works.

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "license()" for more information.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py
=====================
9.4
>>>
```

Wow, look at that. It gave exactly the same answer as the one calculated by hand. Now let me explain what happens in each line of code so that you’ll

be able to replicate it with ease next time.
The first line of code *scores = [2, 6, 8, 12, 16, 20, 10, 5, 8, 7]* assigns the list of scores to the variable name scores

The second line of code *length = len(scores)* uses the ‘len’ function in a list to count the number of scores in the list and assigns it to the variable name length.

The third line of code *summation = 0* initializes summation as 0. Why? We’ll discuss this as we move down the lines of code.

The fourth line of code *for a score in scores* : is the beginning of the ‘for’ loop, It means for each score in the list of scores, execute the next statement below.

The fifth line of code *summation += score* is used to add the score in the list of scores to the summation and assigns the result back to summation. In order words, it does exactly this *summation = summation + score*. For example if summation = 8 and score = 2, when the statement summation += score is performed, this is what occurs, summation = 8 + 2 resulting in summation becoming 10.
Now back to why we initialized summation as 0. If summation wasn’t initialized, you wouldn’t be able to perform the fifth line of code. It’ll result in an error. Try doing this and see the error you get. You’ll get exactly this error.

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "license()" for more information.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py
====================
Traceback (most recent call last):
  File "C:\Users\uf\Desktop\new.py ", line 5, in <module>
    summation += score
NameError: name 'summation' is not defined
>>>
```

The sixth line of code *mean = summation/length divides* the summation by the length and assigns the value to the variable name mean

The last line of code *print(mean)* displays or prints out the value of the mean on the python shell or whatever python supported platform you're using.

From the explanation, I hope you've been able to understand each line of code written in order to calculate the mean of scores.
You can also give it a twist of your own by increasing the number of scores in the list.

I'm sure this lesson has been as interesting for you as it has been for me.
Let's take one more practical example before we proceed to the next lesson.
Write a code that prints out the multiples of 6 between 1 and 100
What do you understand by multiples?
Multiples are whole numbers that can be divided by a certain amount without remainder
Therefore it is safe to say multiples of 6 are numbers that can be divided by 6 without the remainder. Examples are 6, 12, 18, …
Now back to python.
The code for printing multiples of 6 between 1 and 100 is writing below.

```
for x in range(1, 101):
    if x%6 == 0:
        print(x)
```

You can see it very simple and short.
The result can be verified below.

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "license()" for more information.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py
=====================
6
12
18
```

```
24
30
36
42
48
54
60
66
72
78
84
90
96
>>>
```

It worked perfectly. Now let me explain what happens in each line of code so that you’ll be able to replicate it with ease next time.
The first line of code *for x in range(1, 101):* is a for loop that would be used to iterate integers between 1 and 100

The second line of code *if x%6 == 0:* is an ‘if’ statement, which checks if the current number modulo 6 equals zero. Do you remember what the modulo function does? If No, I’ll explain.
The modulo function gives the remainder when a number is divided by another number.
Why do you think we’re using the modulo function here?
We’re using the modulo function to check whether the current number divided by 6 does not give a remainder and remember the definition for multiple, a number that divides another number without a rest.

The last line of code *print(x)* prints numbers that evaluates the ‘if’ statement as true.

From the explanation, I hope you’ve been able to understand each line of code written in order to get multiples of 6.
You can also give it a twist of your own by getting multiples of 7, 8, 9, and also working on getting factors of a number. You can proceed to also get even and odd numbers within a specific range.

Here is an assignment for you
Write code using the ‘for’ loop statement to create a list of Fibonacci series based on the number of Fibonacci sequences a user requires. For example, if a user wants 4 numbers of Fibonacci series, the result should be [1, 1, 2, 3].
Hints:
A Fibonacci series in mathematics is a series in which the next number is the sum of the two preceding numbers. From the sequence above,
1 + 1 resulted in 2 which was the next number
Also, 1 + 2 led into 3 which was the last number
To get another Fibonacci number, who can guess what the next number might be?
Let me help out here. It's 2 + 3, which will result in 5.

Also, you need to know how to use the append() operation in a list.

This is the end of this lesson. I hope you’ve been able to understand how ‘for’ loop statements work and how to write simple ‘for’ loop statements. I wish I have also been able to increase your mathematical knowledge, which is an essential requirement for writing ‘for’ loops.

Goodbye for now. See you in the next lesson.

Congratulations!! You have finished this level. You have earned 10 Adak coins and the medal of creativity!!!

Level 4

Functions and Modules

Functions

Functions are the most basic program structure, which provides means of maximizing code re-use, which also leads you to the broader notion of program design, which is not within the scope of this group.

Statement or expression associated with functions

The call expression

The call expression is used to call an already existing function. For example, if you have a function bakeCake(), the call expression can be used to call the function.
Format of the call expression

```
bakeCake('egg', 'flour', 'milk', 'water', 'sugar')
```

The def statement

The def statement is used to create a function object and assigns it to a name
The general format of the def statement

```
def name (argument1, argument2, …, argumentN):
    statement
```

The return statement

The return statement sends a return object to the caller (user). When a function computes a value, it sends back the result when the return statement is called. A return without a value will return to the caller a None value, which is the default result.
The general format of the return statement

```
def name (value1, value2):
    return value3
```

Arguments

Arguments are passed into functions by assignment. Changing an argument name does not change the corresponding name of the caller, but changing the passed-in objects in place of the former objects can result in a change in the result of the function.
Arguments are also passed-in by position unless you state otherwise.
Arguments are not declared ahead of time in a function. This rule also applies to return values and variables. They are not declared ahead of time in a function.

General format of arguments

```
def name (argument1, argument2, …, argumentN):
    statement
```

A function will take as many arguments as you want.

Now it’s time to go to the practical aspect of this lesson.
Let’s print a function that accepts two arguments and return strings that are common to those arguments in a list.
This should be fun
Follow me carefully in every step of this journey.

```
def intersect(arg1, arg2):
    common = []
    for x in arg1:
        if x in arg2:
            common.append(x)
    return common

value = intersect('sing', 'sang')
print (value)
```

Do not panic, I’ll explain this later. But in the meantime, let us see the result of this code in our python shell or any other python environment currently been used.
Make sure you follow the indentation rule here strictly.

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "license()" for more information.
>>>
```

```
==================== RESTART: C:\Users\uf\Desktop\new.py
=====================
['s', 'n', 'g']
>>>
```

It worked perfectly. Now let me explain what happens in each line of code so that you'll be able to replicate it with ease next time.
The first line of code *def intersect(arg1, arg2):* creates the function and assigns it to the name 'intersect'

The second line of code *common = []* assigns an empty list to the variable name common

The third line of code *for x in arg1:* is a 'for' loop, which iterates through the first argument, arg1.

The fourth line of the code *if x in arg2:* is an 'if' statement that checks whether the strings in the first argument, arg1 are contained in the second argument, arg2.

The fifth line of the code *common.append(x)* consist of a list operation, append(), which adds the strings common to both arg1 and srg2 to the list common.

The sixth line of the code *return common* is a return statement that returns the value of the function 'intersect', which are all the strings common to arg1 and arg2 in a list.

The seventh line of code is a blank space that is actually not required but used to keep the code clean.

The eighth line of code *value = intersect('sing', 'sang')* is a call expression which calls the function 'intersect' and passes two arguments ('sing' and 'sang') inside it to get the strings common to both cases. This call expression is then assigned to the variable name value.

The ninth line of code print *(value)* prints or displays the value of call expression.

As you can see, it isn't anything complicated. It is simple and easy to understand. You can try playing with the code by inserting your arguments and see if it works perfectly.

Should we take another example before we leave this lesson?

Well, I heard you, I’ll take a final example.

Let’s write a function that takes an argument and finds the divisors of the argument and returns a list of those divisors.

```
def divisors(number):
    result = [ ]
    for x in range(1, int(number/2)+1):
        if number % x == 0:
            result.append(x)
    result.append(number)
    return result

value = divisors(24)
print (value)
```

Just calm down. I’ll explain it. Let’s see the result of this in the python environment.

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "license()" for more information.
>>>
==================== RESTART: C:\Users\uf\Desktop\new.py
=====================
[1, 2, 3, 4, 5, 8, 12, 24]
>>>
```

The explanation for this code is quite logical, but I’ll do my best to try to make you understand. Just trust me.

What are the divisors of a number?
Divisors of a number are numbers that can divide that particular number without a remainder. For example, the divisors of 6 are 1, 2, 3, and 6. Take note that the penultimate divisors are always equal to or lesser than half the number. As you can see, 3 is just half of 6.

Now let’s leave the mathematics and explain the code.

The first line of code *def divisors(number):* creates the function and assigns it to the name ‘divisors’

The second line of code *result = [ ]* assigns an empty list to the variable name common

The third line of code *for x in range(1, int(number/2)+1):* is a ‘for’ loop, which iterates from 1 to half the number of the argument entered by the caller.

The fourth line of code *if number % x == 0:* is an ‘if’ statement that checks if the number is divisible by the numbers (x) being iterated through.

The fifth line of the code *result.append(x)* consists of a list operation, append(), which adds the numbers that are factors of the argument into the list, result.

The sixth line of the code *return result* is a return statement that returns the value of the function ‘divisors’ which are all factors of the argument in a list.

The seventh line of code is a blank space that is not required but used to keep the code clean.

The eighth line of code *value = divisors(24)* is a call expression that calls the function ‘divisors’ and passes an argument (24) inside it to get the factors of the argument. This call expression is then assigned to the variable name value.

The ninth line of code *print (value)* prints or displays the value of call expression.

As you can see, it isn’t anything complicated. It is simple and easy to understand. You can try playing with the code by inserting your argument and see if it works perfectly.

We have come to the end of this lesson, so it’s time to say goodbye for now. Don’t think for a second that you won’t have the assignment to do in this lesson.

Your assignment now is to write a function that accepts an argument and finds out whether that argument is a prime number or not.

Hints:
Prime numbers have only two divisors. 1 and the number itself. You've learned how to work with divisors here, I am sure you're up to this challenge.

Do not be scared to use nested loops.
Nested loops are loops that are inside each other. For example

```
for x in listA:
    for y in listB:
        print (x + y)
```

That is an excellent example of a nested 'for' loops.

Have a great time. See you in the next lesson.

Modules

Modules are pieces of code that could contain functions, variables, loops, etc. combined into a useable program.

Modules are done can be used for what they have been designed to do.

Some modules are built into Python; that is where they downloaded with the python program. Others can be downloaded separately. You could also do modules of your own if you so wish.

We would focus on modules that came with the python program.

Importing modules
When modules are to be used, you 'import' them.

For example
import time
I will import the time and date on your computer.

import turtle
Will import the turtle module

Common modules

Time Module
To calculate the current date and time using a built-in module called time:

```
>>> import time
```

You have used the import command to tell Python that we want to use the module time.

You can use the functions of the time module, using the dot.
(we can also call the features of other modules using dot)

Asctime function
asctime function of the time module shows the current date and time.

Example

```
>>> print(time.asctime())
```

```
R: 'Mon Nov 5 12:40:27 2012'
```

Localtime function
Unlike asctime, the function localtime brings out the current date and time as an object.
If you print the object, you’ll see the name of the class, and each of the values labeled as tm_year, tm_mon (for month), tm_mday (for day of the month), tm_hour, and so on.

```
>>> import time
>>> print(time.localtime())
R: time.struct_time(tm_year=2020, tm_mon=1, tm_mday=17, tm_hour=15,
tm_min=38, tm_sec=53, tm_wday=4, tm_yday=17, tm_isdst=0)
```

Sleep function
The sleep function is used when you want to slow down a program.

Example

```
>>> import time
>>> for a in range(1, 360):
print(a)
time.sleep(1)
```

R: 1 to 360 will be printed out, but there will be a lag or pause between the printing of each number.

Random Item from a List
This is for randomly picking an item from a list.
Example

```
>>> import random
>>>numbers = [23,34,56,78,89,788,686,5757,54,90]
>>> print (random.choice(numbers))
>>> print (random.choice(numbers))
>>> print(random.choice(numbers))
>>> print(random.choice(numbers))
>>> print(random.choice(numbers))
```

```
R: 3
5757
90
34
5757
```

You would likely get results different from mine when you try.

Shuffle module

This shuffles the items in list.

```
>>> import random
>>> numbers = [23,34,56,78,89,788,686,5757,54,90]
>>> random.shuffle(numbers)
>>> print(numbers)
R: [23, 56, 788, 34, 78, 89, 54, 686, 90, 5757]
```

Keyword Module

A keyword in python is any word that is used in the language itself, such as if, else, and for.

Some functions of the keyword module are
iskeyword
and
kwlist.

Iskeyword is a function which returns true if any string is a Python keyword.

Kwlist is a variable which returns a list of all Python keywords.

```
>>> import keyword
>>> print(keyword.iskeyword('else'))
R: True
>>> print(keyword.iskeyword('wanda'))
R: False
>>> print(keyword.kwlist)
R: ['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if',
```

'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

You can try using the iskeyword and kwlist for other words.

Copy Module

The copy module consists of functions for creating copies of objects. Usually, when typing a program, you’ll make new objects and sometimes it’s good to type a text of an
object, and then use that copy to
make a new object, particularly
when the process of creating an
the purpose takes several steps.

Example

```
>>> class school:
def __init__(name, term, number_of_students, uniform_color):
name.term= term
name.number_of_students = number_of_students
name.uniform_color=uniform_color
```

We could make a new object in the class school using the following code.

```
>>> h = school(3, 645, 'pink')
```

We can repeat the previous code over and over again, when we need it more than once but that will be too stressful. We could instead use the copy module:

Example

```
>>> import copy
>>> h = school(3, 645, 'pink')
>>> x = copy.copy(h)
>>> print(x.term)
R: 3
>>> print(h.term)
```

R: 3

sys Module
The sys module is used to control the python shell.
It contains functions such as
Exit, stdin and stdout objects, and version variable.

exit function
This function will shut down the python shell.

When you enter the code,

```
>>> import sys
>>> sys.exit()
```

You'll be prompted with a message asking if you want to exit. Click Yes, and the shell will close.

Version variable
You can find out the version of python you are using especially when you are on an unfamiliar computer:

```
>>> import sys
>>> print(sys.version)
R: 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)]
```

Turtle Module

The turtle module is used to create a sort of canvas for drawing.

Example

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

Doing this will call the turtle module. We would discuss the turtle module extensively at the next level.

Congratulations, you have finished this level. You have earned 12 Adak coins and the medal of craft!!

Level 5

Turtle

Turtle graphics is a popular module for programming for kids (and adults). You already know that to use turtle, you need to ‘import’ turtle. When you import turtle, and you 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 anyhow you want. 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 () function (note that the second T is capital).
You need to assign the turtle.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 be 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 call your turtle screen any name.

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 would use the following functions:

```
turtle.forward (distance)
turtle.backward (distance)
turtle.left (angle)
turtle.right (angle)
```

turtle.forward
this is 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 ()

turtle.left ()

this moves the turtle to the left across a 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 he turtle.left 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)
```

Coordinates

You should notice that the turtle arrow usually starts in the middle with equal spaces around it.
If you divide the turtle screen into 4 equal halves with two imaginary lines going through the middle, the point the two lines meet is where you find the turtle arrow.

The first line goes from up to down while the second line goes from right to left.
The first line from up down is called the y axis, and the second line is called the x axis.

The point where the lines meet is and where the turtle is the center. That point also can be represented with numbers.
The point can be represented by the numbers 0,0. This means that where the turtle is point 0 on the y axis and also point 0 on the x axis.

The part of the y axis above the turtle is positive, and the sections below the turtle on the x axis are negative.

The part of the x axis to the right of the turtle is positive and to the left is negative.

It is just like in trigonometry when you draw graphs.

When you move the turtle, you can always get its coordinates.
To do this use the function .position

Example

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

>>> turtle.fd (150)

>>> print (turtle.position())
R: (150.00, 0.00)
```

The result will be seen in the python shell.

It means that the turtle arrow has moved 150 points on the x axis and 0 points on the y axis.

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)

>>>print (turtle.position ())

R: (225.00,129.90)
```

This means that the turtle has moved 225 points on the X axis and 129.9 points on the Y axis.

Note that because the angle is 60 degrees, the turtle moved extra on the X axis as the second line was drawn. Also, the line drawn upwards is bent and not straight; hence the turtle did not move the full 150 points upwards.

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)

>>>print (turtle.position ())
```

R:

```
=============== RESTART: C:\ Users \ uf\ Desktop \ practice.py
===============
(-150.00,-150.00)
```

This means that the turtle moved -150 points on both the x and y axes.

The coordinates will come in very handy when we begin to develop our game.

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 come back.

The code for pen up = turtle.penup(), turtle.pu(), turtle.up() you can use any.
The code for pen down = turtle.pendown(), turtle.pd(), 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 earlier created. 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.

Drawing a full Square

We already know how to command the turtle to draw lines. Next, we will learn how to make the turtle draw squares.

```
>>> 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.bk(150)
>>> turtle.left(90)
>>> turtle.bk(150)
```

The above code will draw you a simple square. You could try using different line lengths to see what will happen.

The code above, however, is a little too long. We can draw the same square using a for loop.

Example

```
>>> for x in range(1, 5):
turtle.bk(150)
turtle.left(90)
```

We start a for loop that will count from 1 to 4 with the code range(1, 5). Then, the next lines tell the turtle that in each run of the loop, it should move backward 150 pixels and turn left 90 degrees.

Pen speed

You would have noticed that the pen moves a little slow whenever it's drawing the square. You can make it run 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 of 1 to 10 are increasingly fast, with 10 being the fastest.

The 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.

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 pen, the turtle pen is, the bigger the lines.
The code for changing the pen size = turtle.width, 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 to 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 piece code tells the turtle to move backward 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 the distance. This gives us or rectangle.
You can play with the code, 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.

<u>**Drawing a Triangle**</u>

Drawing a triangle is a little different from drawing a square.
There are different types of triangles. We will attempt to draw 3 types.

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.fd (200)
>>> t.left (120)
>>> t.fd (200)
>>> t.left (120)
>>> t.fd(200)
```

This code tells the turtle to move forward by 200 points, then to turn 120 degrees then move forward by 200 points, then to turn another 120 degrees then move another 200 points giving us an equilateral triangle (a triangle which has all its sides equal)

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.fd (200)
>>> t.left (90)
>>> t.fd (200)
>>> t.left (135)
>>> t.fd(280)
```

This code draws a right-angled triangle.

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.fd (140)
>>> t.left (110)
>>> t.fd (200)
>>> t.left (140)
>>> t.fd(200)
```

This code gives you an isosceles triangle. Two sides of the isosceles triangle are equal.
You can vary thee parameters and get different results.

<u>Circle</u>
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 the square simpler? 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 thje square, we looped 4 times with the range (1,5), we loop this we loop eight times with range(1, 9).

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

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.

Spiraling star:

```
>>> t= turtle
>>> t.speed(8)
>>> t.width(3)

>>> for x in range(1, 25):
t.forward(120)
t.left(95)
```

<u>Coloring!</u>

The color function in python has three main colors, which are specified by numbers.

These colors are ‘red’, green, and ‘blue’. The default numbers are denoted by = (0 to 1).

These colors are the primary colors on your computer. They all mix to form all the colors you see on your computers.

We already learned how to change the background color of the screen. Next, we will learn how to change the color of the pen and how to fill shapes.

<u>Changing the pen colour</u>

To change the pen color, you need to use the function; .pencolor

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.pencolor (1,0,0)

>>> t.fd (300)
```

R: this will return a red line.

Using t.pencolor (0,1,0) will give green and t.pencolor (0,0,1) will give blue. This is because the first number represents red, the second green, and the third blue. Together, they form your computer's basic colors.

You can use different quantities of these basic colors to get any color you want.

Example

```
>>> t= turtle
>>> t.speed(8)
>>> t.width(3)
>>> t.pencolor (1,1,0)

>>> t.fd (300)
```

The above t.pencolor (1,1,0) mixes 100 % red and 100% green with)% blue to form a yellow line.

We can also use colors to fill shapes.

Filling shapes with colors

To fill shapes, we, first of all, specify the filling color with the function .fillcolor. We then use the functions .begin_fill and the .end_fill.

Example

```
>>> import turtle
>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width = 800, height = 500)

>>> t= turtle
>>> t.speed(0)
>>> t.width(3)
>>> t.pencolor(0,0,1)
>>> t.fillcolor (1,1,0)

>>> t.begin_fill()
>>> t.circle (100)
>>> t.end_fill()
```

This code tells the python interpreter to draw a blue pen color. It also tells it to fill any shape to be filled with yellow (fillcolour ()). The code t.begin fill and t.endfill allow the circle we have drawn to be filled.

A Function to Draw a Filled Circle

As we learned earlier, we use functions to make our code easier. We will now create a function to draw a filled circle.

Example

```
>>> t= turtle
>>> t.speed(0)
>>> t.width(3)

>>> def filled_circle (radius, red, green, blue):
t.color(red, green, blue)
t.begin_fill()
t.circle(radius)
t.end_fill()
```

We can draw a green circle with a radius of 150, with this code:

```
>>> filled_circle(150, 0, 1, 0)
```

(You should remember that your indentation must be constant when creating your functions)

We can make the green color darker by using only half the green paint (0.5):

```
>>> filled_circle(150, 0, 0.5, 0)
```

You can play around with the function you just created and try using different parameters.

Example

```
>>> filled_circle(150, 1, 0, 0)
>>> filled_circle(150, 0, 0.5, 0.5)

>>> filled_circle(100, 0.8, 0.5, 0.5)
>>> filled_circle(200, 0.9, 0.75, 0.0)
>>> filled_circle(400, 0.95, 0.7, 0.74)
>>> filled_circle(40, 0.9, 0.5, 0.0)
>>> filled_circle(50, 0.9, 0.5, 0.2)
>>> filled_circle(170, 0, 0.5, 0.4)
```

Creating Pure Black and White

To create black, you will turn off all the basic colors

Example

```
>>> filled_circle(100, 0,0,0)
```

This will give you a pure black circle.

To create white, you should use all colors fully.

Example

```
>>> import turtle
>>> sr= turtle.Screen ()
>>> sr.bgcolor (‘red’)
>>> sr.title (‘learning turtle’)
>>> sr.setup (width = 800, height = 500)

>>> t= turtle
>>> t.speed(0)
>>> t.width(3)

>>> def filled_circle (radius, red, green, blue):
t.color(red, green, blue)
t.begin_fill()
t.circle(radius)
t.end_fill()

>>> filled_circle(100, 1, 1,1)
```

For this, we changed the background color of the screen to read so the white circle can be seen clearly.

<u>A Square-Drawing Function</u>
We can do what we did with the circle with almost any shape. We can do it with Squares too. Let’s create a square drawing function.
Example

```
>>> t= turtle
>>> t.speed(0)
>>> t.width(3)


>>> def filled_circle (radius, red, green, blue):
                        t.color(red, green, blue)
                        t.begin_fill()
                        t.circle(radius)
                        t.end_fill()

>>> filled_circle (100, 0.95, 0.7, 0.05)


>>> t.penup()
>>> t.fd(200)
>>> t.pd()
>>> t.pencolor(1,0,0)

>>> def mysquare(size, angle):
                        for x in range(1, 5):
                                                t.forward(size)
                                                t.left(angle)

>>> mysquare (70, 90)
```

R: we get a golden circle and a square drawn with red.
We used the pen up and penned down functions to move the turtle 200 points from the circle. We then created our square drawing function.

You can try drawing squares of different sizes 25, 50, 125, 75, and 100.

```
>>> mysquare(25)
>>> mysquare(50)
>>> mysquare(125)
>>> mysquare(75)
>>> mysquare(100)
```

You will notice that the squares don't touch the circle.

Drawing Filled Squares

To draw a filled square, we simply upgrade our square drawing function:
Example

```
>>> def mysquare (size, angle, red, green, blue):
t.color (red, green, blue)
t.pencolor(1,0,0)
t.begin_fill ()
for x in range (1, 5):
                                        t.forward(size)
                                        t.left(angle)
                    t.end_fill()
```

```
mysquare (70, 90, 1, 1, 0)
```

R: you should get a yellow square with a red outline. We expanded our function by adding in more parameters. We added in the red, green, and blue and the function .color. After this, we specified a pen color. We need to add the pen color into our function else the function will draw a yellow square without the red line. We also added in the beginning and end fill functions.

You can play around with this square drawing function.

Let's draw with our functions a golden circle with a pink outline and a pink square with a golden outline.

Golden Circle with pink Outline and Pink Square with a Golden Outline

```
import turtle


>>> sr= turtle.Screen ()
>>>sr.bgcolor ('white')
```

```
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)

>>> t= turtle
>>> t.speed(0)
>>> t.width(5)
>>> t.pencolor(1,0,0)


>>>def filled_circle (radius, red, green, blue):
                        t.color(red, green, blue)
                        t.pencolor(1, 0.7, 0.75)
                        t.begin_fill()
                        t.circle(radius)
                        t.end_fill()

>>> filled_circle (100, 0.9, 0.7, 0.05)

>>> t.penup()
>>> t.fd(200)
>>> t.pd()

>>> def mysquare (size, angle, red, green, blue):
                        t.color (red, green, blue)
                        t.pencolor(0.9, 0.7, 0.05)
                        t.begin_fill ()
                        for x in range (1, 5):
                                                t.forward(size)
t.left(angle)
                        t.end_fill()


>>> mysquare (70, 90, 1, 0.7, 0.75)
```

We have increased the thickness of the pen and also added a pen colour to the circle drawing function.
Other than that, we just attributed the color parameters for pink and orange.

Drawing Filled Stars

Let’s do some beautiful things with stars.

Example:

```
>>> t= turtle
>>> t.speed (0)

>>> def star (size, angle):
    t.pencolor (0,0,0)
    t.width (2)

    for x in range (1, 100):
          t.forward(size)
          t.left(angle)
          size= size - 1

>>> star (150, 145)
```

This function makes a beautiful star. We will edit it as we go along to make more beautiful stars.

First, we change the color

Example

```
>>> t= turtle
>>> t.speed (0)

>>> def star (size, angle):
                              t.pencolor (1,0,0)
                              t.width (2)

                              for x in range (1, 100):
                                                    t.forward(size)
                                                    t.left(angle)
                                                    size= size - 1

>>> star (150, 200)
```

This gives a red star!

Let’s have more fun.
We can make the colors of the lines in the star random. To do this, we need to use a function of turtle called color mode.

Colour mode

So far, we have been using numbers between 0 and 1 for our colors. We can also use between 0 and 255. 255 is used so we can have better control over the colors. It’s not very easy to maneuver between 0 and 1 because of the “limited” amount of numbers between o and 1.
Example

```
>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)


>>> t= turtle
>>> t.colourmode (255)

>>> t.pencolour (255,0,0)

>>> t.fd (300)
```

R: this will give you a red line. (0,255,0) will give a green line and (0,0,255) will give a blue line.

Let’s get back to our star. To accomplish our goal of lines of different colors, we will import random and use the random. randint function and the colour mode function we just learned.

Example

```
>>> import turtle
>>> import random
>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)

>>> t= turtle
>>> t.speed(0)
>>> t.width(5)

>>> def star (size, angle):
                    t.pencolor (random.randint (0,255), random.randint
(0,255), random.randint (0,255))
                    t.width (2)

                    for x in range (1, 100):
                        t.forward(size)
                        t.left(angle)
                        size= size - 1

>>> star (150, 200)
```

R: When you run this code, the color of the star changes each time. We imported the random module and then used it random.randint function to pick a random number between 0 and 255 for each of the primary colors.

We are getting close to our goal.

```
>>> import turtle
>>> import random
>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)

>>> t= turtle
>>> t.speed(0)
>>> t.width(5)

>>> def star (size, angle):
                    t.pencolor (random.randint (0,255), random.randint
(0,255), random.randint (0,255))
                    t.width (2)

                    for x in range (1, 100):
                        t.forward(size)
                        t.left(angle)
                        size= size – 1
                        t.pencolor (random.randint (0,255), random.randint
(0,255), random.randint (0,255))
>>> star (150, 200)
```

With this, the colors of each line are different. Can you figure out how?

Other things about the turtle module

Turtle Movement

<u>Set Position</u>

Remember when we discussed the coordinates? Well, we can also make the turtle go to wherever we want it to go by setting the specific coordinates we wish.

There are several functions we can use for this and we will examine them one after the other starting with these

turtle.goto (x, y)
turtle.setpos (x, y)
turtle.setposition (x, y)

- **x** – a number or zero
- **y** – a number or zero

Example

```
>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)

>>> t= turtle

>>> t.fd (300)
>>> t.setpos(60,30)
>>> t.setpos (0,0)
```

This code will give you a triangle. We made the turtle draw a line of 300 points or pixels then asked it to go to coordinates x= 60 and y= 30. We then asked it to return to coordinates x= 0 and y=0 where the turtle began. The triangle created is scalene. No sides are equal.

We can also use the functions set position and goto to accomplish the same task.
Example

```
>>> t= turtle

>>> t.fd (300)
>>> t.goto (60,30)
>>> t.goto (0,0)
 Or
>>> t= turtle

>>> t.fd (300)
>>> t.setposition (60,30)
>>> t.setposition (0,0)
```

We can also set the x and y coordinates individually with the sets and sety functions

turtle.setx(x)

x – a number (integer or float)
Example

```
>>> t= turtle
>>> t.setx (60)
```

This will give you a straight line which m0oved from coordinate x = 0, y = 0 to coordinate x = 60 and y = 0.

turtle.sety(y)
y – a number (integer or float)
Example

```
>>> t= turtle

>>> t.sety (50)
>>> t.setx(60)
>>> t.sety (-50)
>>> t.setx (0)
>>> t.sety (0)
```

This code creates a rectangle.

turtle.home()

this function returns the turtle to point 0,0 (x = 0, y = 0) no matter where the turtle is on the screen.

Example

```
>>> t= turtle

>>> t.sety (50)
>>> t.setx(60)
>>> t.sety (-50)
>>> t.home ()
```

This code creates a trapezium. The turtle is made to go back to position 0, 0 (x = 0 , y= 0) from position 60,-50 (x = 60 , y = -50).

turtle.dot

This is a simpler way to create a filled-in circle. The turtle.dot function allows you to specify both the size and color of the circle you want to draw (picking a color is optional).

turtle.dot(*size, *color*)

· **size** – an integer >= 1 (if given)
· **color** – a colorstring or a numeric color tuple

Example

```
>>> t= turtle

>>> t.dot (100)
```

This will give a black circle with a diameter of 100 points.

You can confirm this with the following piece of code.

```
>>> t= turtle
>>> turtle.dot(100)
>>> t.pencolor('red')
>>> t.width (5)
>>> t.bk(50)
```

You will get the black circle with a red line of 50 forming its radius.
The radius is the length from the center of a circle to its edge. It is also ½ of the diameter of a circle.

You can get the circle to be any color you wish
Example

```
>>> t= turtle
>>> turtle.dot(100, ‘blue’)
>>> t.pencolor('red')
>>> t.width (5)
>>> t.bk(50)
```

R: this gives you a blue circle with a red radius.

Example

```
>>> import turtle
>>> import random

>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)

>>> t= turtle

>>> t.colormode (255)

>>> t.dot(100, (200,100,50))

>>> t.pencolor('red')
>>> t.width (5)

>>> t.bk(50)
```

You will have a lot of fun with the dot code and create multiple dots with different colors.

Let’s have some fun with it.

Polka Dots Program

To begin the game, we need to import turtle and set or configure the screen we want to use!

```
>>> import turtle
```

```
>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)
```

Next, we give our turtle some speed and we set our colour mode to 255

```
>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)
```

We have not yet given the turtle any instructions at this point. If we run it, we will see the turtle screen we specified and the turtle arrow in the middle. Running it without getting an error is good enough for us for now.

The next thing we want to do is to write the code for our filled circle.
We use the t.dot

```
>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)
```

```
>>> t.dot(15)
```

this will give us a filled-in black circle with a diameter of 15

We are looking to use multicolored circles. Hence we need to change the colors rapidly. We, therefore, need to assign the diameter of the ring and the colors to variables.
Let’s use a and b

```
>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)
```

```
>>> a = 15
>>> b = ‘red’

>>> t.dot (a, b)
```

When you run this, you get a red dot with a diameter of 15.

The next thing to do is to import the random module.

```
>>> import turtle
>>> import random

>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)
>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)

>>> a = 15
>>> b = ‘red’

>>> t.dot (a, b)
```

After importing the random module, we set the diameter to a random value.

```
>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)

>>> a = random.randint (5, 30)
>>> b = ‘red’

>>> t.dot (a, b)
```

You should run this like 5 to 6 times in a row. You will notice that the size of the red dot keeps on changing.

Next, we use the colormode function of the turtle module and apply the random.randint () function to the color of our circle.

```
>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)
>>> t.colormode (255)

>>> a = random.randint (5, 30)
>>> b = (random.randint (0,255),random.randint (0,255),random.randint
(0,255))

>>> t.dot (a, b)
```

Running this several times will show that the color and diameter of the circle changes any time you run the code.
The next thing we do is create an infinite loop with the while loop.

```
>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)
>>> t.colormode (255)

>>> a = random.randint (5, 30)
>>> b = (random.randint (0,255),random.randint (0,255),random.randint
(0,255))

>>> while a < 4 and a > 31:
t.dot (a, b)
```

When you run it at this point, you would notice that a circle of the same color keeps forming on the same location.
The next thing to do is to give the turtle x and y coordinates to go to. Since the screen is 800 points in width or on the x-axis and 500 points in height or the y-axis, we assign values on the x and y-axis to the setx and sety

```
>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)
>>> t.colormode (255)

>>> a = random.randint (5, 30)
>>> b = (random.randint (0,255),random.randint (0,255),random.randint
(0,255))

>>> while a < 4 and a > 31:
```

```
t.dot (a, b)
t.setx(random.randint(-390, 390))
t.sety(random.randint(-240, 240))
```

We set x and y to move randomly between -390 and 390 for x (390 + 390 = 780) and – 240 and 240 for y (240 + 240 = 480) so that the turtle stays within the screen we created.
At this point, the turtle keeps forming circles of the same diameter all over the screen. The rings are all connected with lines.
The next action is to remove the lines.
We will do that with the pen up and pen down functions

```
>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)
>>> t.colormode (255)

>>> a = random.randint (5, 30)
>>> b = (random.randint (0,255),random.randint (0,255),random.randint
(0,255))

>>> while a < 4 and a > 31:
t.dot (a, b)
t.pu()
t.setx(random.randint(-390, 390))
t.sety(random.randint(-240, 240))
t.pd()
```

With the pen up and down functions, the lines are gone. The circles are formed all over the screen, but they are still circles of the same size and color.
We need to make the circles change sizes and colors. To do this, we will copy and paste into the loop the values of variables a and b. This will make the circles change both color and width as they form around the screen.

```
>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)
>>> t.colormode (255)
>>> a = random.randint (5, 30)
>>> b = (random.randint (0,255),random.randint (0,255),random.randint
(0,255))

>>> while a < 4 and a > 31:
t.dot (a, b)
t.pu()
t.setx(random.randint(-390, 390))
t.sety(random.randint(-240, 240))
t.pd()
a= random.randint (5, 30)
b= (random.randint (0,255),random.randint (0,255),random.randint (0,255))
```

Once this is done, we have circles of different colors and with forming randomly across the screen. Your polka dots game is complete. You can vary the parameters of the program and get something else.

The game runs on an infinite loop. You could use a for loop to make the game run in a finite loop.

```
>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)
>>> t.colormode (255)

>>> a = random.randint (5, 30)
>>> b = (random.randint (0,255),random.randint (0,255),random.randint
(0,255))

>>> for i in range (1, 400):
t.dot (a, b)
t.pu()
t.setx(random.randint(-390, 390))
t.sety(random.randint(-240, 240))
t.pd()
a= random.randint (5, 30)
b= (random.randint (0,255),random.randint (0,255),random.randint (0,255))
```

The code of the program can be changed to anything you wish. You could increase the range to 1000 or 10000 if you want to.

There are still other functions of the turtle module.
You can control your turtle itself and do several beautiful things to it.

turtle.shape()
With this function, you can change the shape of the turtle. The turtle is currently an arrow. You can change

turtle.shape (*name*)
name – a string that is a valid shape name.
Shape names = “arrow”, “turtle”, “circle”, “square”, “triangle”, “classic”.
Example

```
>>> t= turtle
>>> t.shape ('turtle')
>>> t.fd (50)'
```

The most interesting of the shapes is the actual turtle shape. The others are circle, arrow, square, triangle, and classic.

We can have more fun with the turtle if we apply the turtle to the polka dots program.

```
>>> t= turtle
>>> t.colormode (255)
>>> t.speed(3)
>>> t.colormode (255)
>>> t.shape (‘turtle’)

>>> a = random.randint (5, 30)
>>> b = (random.randint (0,255),random.randint (0,255),random.randint
(0,255))

>>> for i in range (1, 400):
                        t.fillcolor(random.randint (0,255),random.randint
(0,255),random.randint (0,255))
t.dot (a, b)
t.pu()
t.setx(random.randint(-390, 390))
t.sety(random.randint(-240, 240))
```

t.pd()
a= random.randint (5, 30)
b= (random.randint (0,255),random.randint (0,255),random.randint (0,255))
In this modification, we have made the speed of the turtle 3 (so you can see the changes). We also made the fillcolor to be random. This makes the color of the turtle to change randomly as the turtle moves around to spread the multicolored circles.

turtle.stamp()
The stamp function makes the turtle leave an imprint on the screen. If the fill color is set, the turtle and its color are stamped on the screen.
Let's use this to upgrade our game.

```
 >>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)
>>> t.colormode (255)
>>> t.shape ('turtle')

>>> a = random.randint (5, 30)
>>> b = (random.randint (0,255),random.randint (0,255),random.randint
(0,255))

>>> for i in range (1, 400):
                    t.stamp ()
                    t.fillcolor(random.randint (0,255),random.randint
(0,255),random.randint (0,255))
t.dot (a, b)
t.pu()
t.setx(random.randint(-390, 390))
t.sety(random.randint(-240, 240))
t.pd()
a= random.randint (5, 30)
b= (random.randint (0,255),random.randint (0,255),random.randint (0,255))
```

with this modification, the turtle outline and color is stamped on the screen together with the turtle.

turtle.reset()
This function is used to clear all the code written for the turtle screen. Only the code written after the rest will be honored.

```
>>> import turtle
>>> import random

>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)

>>> t= turtle
>>> t.colormode (255)
>>> t.speed(3)

>>> t.shape ('turtle')


>>> a= random.randint (5, 30)
>>> b= (random.randint (0,255),random.randint (0,255),random.randint
(0,255))

>>> for i in range(1, 10):
                t.dot(a, b)
                t.pu()
                t.setx(random.randint(-390, 390))
                t.sety(random.randint(-240, 240))
                t.pd()
                a= random.randint (5, 30)
                b= (random.randint (0,255),random.randint
(0,255),random.randint (0,255))

>>> t.reset()

>>> for i in range(1, 30):
                t.stamp()
                t.fillcolor(random.randint (0,255),random.randint
(0,255),random.randint (0,255))
                t.pu()
                t.setx(random.randint(-390, 390))
                t.sety(random.randint(-240, 240))
                t.pd()
```

When you run this code, the first polka dot typed on the screen will disappear, leaving the screen blank for the stamps of the turtle. This is very useful when you begin to design your games.

turtle.hideturtle() and turtle.show turtle
This function hides your turtle. It is helpful when you want to create a game. It also makes your game execute faster.

You use the codes;

```
turtle.hideturtle()
turtle.showturtle()
turtle.st()
turtle.ht()
```

```
>>> import turtle
>>> import random

>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)


>>> t= turtle
>>> t.colormode (255)
>>> t.speed()

>>> t.shape ('turtle')


>>> a= random.randint (5, 30)
>>> b= (random.randint (0,255),random.randint (0,255),random.randint
(0,255))


>>> for i in range(1, 100):
                t.ht()
                t.dot(a, b)
                t.pu()
```

```
                    t.setx(random.randint(-390, 390))
                    t.sety(random.randint(-240, 240))
                    t.pd()
                    a= random.randint (5, 30)
                    b= (random.randint (0,255),random.randint
(0,255),random.randint (0,255))
```

With this code, the polka dots appear to pop out of the screen.

With certain games, you need animation. You need the turtle to move very fast sometimes and sometimes very slow. To make the turtle move slower for a particular part of the code, you need to use the animation tools. The animation tools are functions that help slow down turtle for several milliseconds. They include

turtle.delay(***delay*** **)**
delay – a positive integer
Set or return the drawing *delay* in milliseconds. (This is the time interval between two consecutive canvas updates.) The longer the drawing delay, the slower the animation.
turtle.tracer(*n* , *delay*)

- **n** – nonnegative integer
- **delay** – nonnegative integer

in this function, you can set the delay and also the speed of the program. Usually, when it is used, the n part (speed part) alone is used.

turtle.update()
turtle. Update () is also like t.speed. It is an alternative that can be used in place of t.speed.
Example
In this example, we will use t.tracer instead of t.speed for our polka dots program.

```
>>> import turtle
>>> import random

>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)


>>> t= turtle
>>> t.colormode (255)
>>> t.tracer (10)

>>> a= random.randint (5, 30)
>>> b= (random.randint (0,255),random.randint (0,255),random.randint
(0,255))


>>> for i in range(1, 100):
                    t.ht()
                    t.dot(a, b)
                    t.pu()
                    t.setx(random.randint(-390, 390))
                    t.sety(random.randint(-240, 240))
                    t.pd()
                    a= random.randint (5, 30)
                    b= (random.randint (0,255),random.randint
(0,255),random.randint (0,255))
```

the programs runs as fast as if you used t.speed (10)

Example

For this example, we will use the polka dots and turtle stamp program. We will use a delay on the turtle stamp program. We use a delay of 50 seconds.

```
>>> import turtle
>>> import random

>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)

>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)
>>> t.shape ('turtle')


>>> a= random.randint (5, 30)
>>> b= (random.randint (0,255),random.randint (0,255),random.randint
(0,255))

>>> for i in range(1, 30):
                    t.dot(a, b)
                    t.pu()
                    t.setx(random.randint(-390, 390))
                    t.sety(random.randint(-240, 240))
                    t.pd()
                    a= random.randint (5, 30)
                    b= (random.randint (0,255),random.randint
(0,255),random.randint (0,255))

>>> t.reset()
>>> t.delay (50)

>>> for i in range(1, 30):
                    t.stamp()
                    t.fillcolor(random.randint (0,255),random.randint
(0,255),random.randint (0,255))
                    t.pu()
                    t.setx(random.randint(-390, 390))
                    t.sety(random.randint(-240, 240))
                    t.pd()
```

You will see that the turtle stamp part of the program does not move fast. There is a long delay between each stamp interval. You could remove the t.delay and see the program run a whole lot faster. Don’t forget that t.reset clears the entire screen.

Let’s go now to functions which control the turtle screen directly.
We have already discussed most of the turtle screen controls except the background picture.
turtle.bgpic(picname)

picname – a string, gif-file or "nopic"
This function, you can set a background image or return the name of the current background image.
If the picname is a filename, it will set the corresponding image as the background.
To use the background picture option, you need to have the picture file in your python files already.

Key binding

This part is a very important part of game development. It shows how to make your program to be controlled by your computer keyboard or mouse.
turtle.listen ()

This is the first function to type when you want the turtle to listen to your keyboard or mouse.
When you use it, it will enable the other functions which allow a keyboard or mouse binding to work.

turtle.onkeypress(fun, key)
turtle.onkeyrelease(fun, key)

- **fun** – a function with no arguments
- **key** – a string: key (e.g. “a”) or key-symbol (e.g., “space”)

Bind function to the key when the key is pressed or released.
Key bindings don’t work when there is no function.
Example

```
>>> import turtle
>>> import random

>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)

>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)

>>> t.shape ('turtle')
>>> def joe ():
                    for i in range(1, 50):
                                        t.stamp()
t.fillcolor(random.randint (0,255),random.randint (0,255),random.randint
(0,255))
                                        t.pu()
                                        t.setx(random.randint(-390, 390))
                                        t.sety(random.randint(-240, 240))
                                        t.pd()
>>> t.listen()
>>> t.onkeypress (joe, 'e')
```

In this example, we converted the turtle stamp program into a function. Once done, we used the listen to function and the onkeypress function to assign the function, which we named joe to the key ‘e’ on the keyboard. When you run the program, the turtle screen comes up, but nothing happens. When you press ‘e’ on your keyboard, however, the program runs. You can press e a 1000 times, and the program runs a thousand times.

Click Binding

turtle.onclick(fun, btn)
turtle.onscreenclick(fun, btn)

· **fun** – a function with two arguments with the coordinates of the clicked point on the canvas

· **btn** – number of the mouse-button, defaults to 1 (left mouse button)

the on-screen click works similar to the on keypress. It would help if you tried to attach it to the polka dots program and see how it runs.

turtle.bye()

This shuts down the turtle module.
Shut the turtle graphics window.
Example

```
>>> import turtle
>>> import random

>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)

>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)

>>> t.shape ('turtle')
>>> def joe ():
                    for i in range(1, 50):
                                        t.stamp()
t.fillcolor(random.randint (0,255),random.randint (0,255),random.randint
(0,255))
                                        t.pu()
                                        t.setx(random.randint(-390, 390))
                                        t.sety(random.randint(-240, 240))
                                        t.pd()
>>> t.listen()
>>> t.onkeypress (joe, 'e')
>>> t.bye
```

The turtle screen opens and runs briefly then closes because of the t.bye.

turtle.exitonclick()

This shuts down the turtle screen when the mouse clicks on the screen.
Bind bye() approach to mouse clicks on the Screen

Example

```
>>> import turtle
>>> import random

>>> sr= turtle.Screen ()
>>> sr.bgcolor ('white')
>>> sr.title ('learning turtle')
>>> sr.setup (width= 800, height=500)

>>> t= turtle
>>> t.colormode (255)
>>> t.speed(8)
>>> t.shape ('turtle')
>>> def joe ():
                    for i in range(1, 50):
                                        t.stamp()
t.fillcolor(random.randint (0,255),random.randint (0,255),random.randint
(0,255))
                                        t.pu()
                                        t.setx(random.randint(-390, 390))
                                        t.sety(random.randint(-240, 240))
                                        t.pd()
>>> t.listen()
>>> t.onkeypress (joe, 'e')
>>> t.exitonclick()
```

This will allow the program to run when you press ‘e’ on your keyboard. Once you click on the turtle screen the program stops running. It then gives a very long error message especially when the program has started to run.

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "license()" for more information.
>>>
====================== RESTART: C:\Users\uf\Desktop\new.py
======================
>>>
====================== RESTART: C:\Users\uf\Desktop\new.py
======================
Exception in Tkinter callback
Traceback (most recent call last):
 File "C:\Users\uf\AppData\Local\Programs\Python\Python38-
32\lib\tkinter\__init__.py", line 1883, in __call__
   return self.func(*args)
 File "C:\Users\uf\AppData\Local\Programs\Python\Python38-
32\lib\turtle.py", line 701, in eventfun
   fun()
 File "C:\Users\uf\Desktop\new.py", line 27, in joe
   t.stamp()
  File "<string>", line 5, in stamp
turtle.Terminator
```

```
>>>
```

••

With all we have discussed here, you should be a master of the turtle module.

Congrats!!

Level 6

Welcome brave programmer to the final level of this checkpoint. At this level, you will learn the secrets of making a game. We will build a game together and thus gain the basics of game making.

Here, you win your wizard’s badge.
Let’s go!!!!!!!!!!!!!!!!!!!

Ping Pong

In this project, we will code a version of the ping pong game. While Ping pong is not particularly exciting compared to today's video games, ping pong is simple to build and provides an excellent opportunity to work on most of the skills that you will need to create a game.

```
import turtle

sr= turtle.Screen ()
sr.title('ping pong game')
sr.colormode (255)
sr.bgcolor(50,100,50)
sr.setup (width= 800, height= 500)
sr.tracer (8)

#GAME OBJECTS
#ball
ball = turtle.Turtle()
ball.speed(0)
ball.shape('circle')
ball.color (105,0,0)
ball.penup()
ball.setpos (0,0)
ball.dx = 1
ball.dy = -1

#paddle_a
```

```
pa = turtle.Turtle()
pa.speed(0)
pa.shape('square')
pa.color ('black')
pa.shapesize(5,1)
pa.penup()
pa.setpos (-370,0)

#paddle_b
pb = turtle.Turtle()
pb.speed(0)
pb.shape('square')
pb.color ('black')
pb.shapesize(5,1)
pb.penup()
pb.setpos (370,0)

#move paddle a up and down
#move paddle a up
def pa_up ():
   y= pa.ycor()
   y+=15
   pa.sety(y)

#move paddle a down
def pa_down ():
   y= pa.ycor()
   y-=15
   pa.sety(y)


#move paddle b up and down
#move paddle b up
def pb_up ():
   y= pb.ycor()
   y+=15
   pb.sety(y)
```

```
#move paddle b down
def pb_down ():
   y= pb.ycor()
   y-=15
   pb.sety(y)

# keyboard binding
sr.listen()
sr.onkeypress(pa_up, 'w')
sr.onkeypress(pa_down, 's')
sr.onkeypress(pb_up, 'i')
sr.onkeypress(pb_down, 'k')

#score
score= turtle.Turtle()
score.speed(0)
score.color(255,123,20)
score.pu()
score.ht()
score.goto(0,210)
score.write('Paddle a: 0 Paddle b: 0', align= 'center', font = ('calibri', 25,
'bold'), )
score_a = 0
score_b = 0

#GAME LOOP

while True:
   sr.update ()
   # moving the ball
   ball.setx (ball.xcor()+ ball.dx)
   ball.sety (ball.ycor()+ ball.dy)

   # border checking upper border
   if ball.ycor()> 235:
      ball.sety(235)
      ball.dy*=-1
```

```
  # border checking lower border
  if ball.ycor()< -235:
    ball.sety(-235)
    ball.dy*=-1

  # border checking left border
  if ball.xcor()< -390:
    ball.setx(-390)
    ball.dx*=-1
    ball.setpos(0,0)
    score_b +=1
    score.clear()
    score.write('Paddle a: {} Paddle b: {}'.format(score_a, score_b), align=
'center', font = ('calibri', 25, 'bold'))


  # border checking right border
  if ball.xcor()> 390:
    ball.setx(390)
    ball.dx*=-1
    ball.setpos(0,0)
    score_a +=1
    score.clear()
    score.write('Paddle a: {} Paddle b: {}'.format(score_a, score_b), align=
'center', font = ('calibri', 25, 'bold'))


  # ball interacting with paddle b

  if (ball.xcor()> 350 and ball.xcor() < 370) and (ball.ycor() < pb.ycor() +
50 and ball.ycor()> pb.ycor() - 50):
    ball.setx(350)
    ball.dx*=-1

  # ball interacting with paddle a

  if (ball.xcor() < -330 and ball.xcor() < -370) and (ball.ycor() < pa.ycor() +
50 and ball.ycor()> pa.ycor() - 50):
    ball.setx(-330)
    ball.dx*=-1
```

This is the code for the complete ping pong game.
We will go through it section by section.

We might ignore certain parts initially, but we will eventually address it.

Creating a screen

First, we create the screen we want to use.

```
import turtle

sr= turtle.Screen ()
sr.title('ping pong game')
sr.colormode (255)
sr.bgcolor(50,100,50)
sr.setup (width= 800, height= 500)
sr.tracer (8)
```

We import the turtle module on a then set the screen using turtle.screen ().
We assign the turtle.screen function to a string called ‘sr’.

Next, we name the screen we wish to use using the function .title (). I named mine ping pong game. You can call yours whatever you prefer.

Next is the color mode, which we set to 255 instead of 1 so we can have a broader range of control over the colors on the screen.

We set the background color with the .bgcolor function then set the screen with and height to what we want. You could use different parameters for your own game.
We also set the speed of the window to 8 using the .tracer function.

Game objects

The game objects are the thing we want to put on the screen, the objects we want to see on the screen.
The objects we want to see on the screen include the ball, the paddles, and the score of the game. We will address them one after the other.

#GAME OBJECTS

The Ball

The ball is one of the objects needed on the screen.

```
#ball
ball = turtle.Turtle()
ball.speed(0)
ball.shape('circle')
ball.color (105,0,0)
ball.penup()
ball.setpos (0,0)
ball.dx = 1
ball.dy = -1
```

We make the ball a turtle object by assigning it to the variable turtle.Turtle. We set the animation speed of the ball to 0, so it can move very fast when we need to move it and set the shape to circle using the function ball.shape (). We set the color of the ball to a slightly deep red (you can use any color).
We set the pen to not show on lines on the screen using the penup function. Also, we set the ball to the center of the screen with setpos. The code ball.dx and ball.dy will be explained later.

Paddles

```
#paddle_a
pa = turtle.Turtle()
pa.speed(0)
pa.shape('square')
pa.color ('black')
pa.shapesize(5,1)
pa.penup()
pa.setpos (-370,0)


#paddle_b
pb = turtle.Turtle()
pb.speed(0)
pb.shape('square')
pb.color ('black')
pb.shapesize(5,1)
pb.penup()
pb.setpos (370,0)
```

The paddles are created just like the ball but with slightly different parameters. We assign the first paddle to a variable called ‘pa’ and make it a turtle object using turtle.Turtle. Unlike the ball, we made the turtle a square. We also made the square to get longer using the function turtle.shapesize (width= 1, height = 5). This made the square to become 5 times taller. All the other things we did to the paddles, we did with the ball except that we left the paddles black.

What we did for paddle a is the same thing we did for paddle b except that for paddle b, we used the negative x coordinates to get it on the other side of the screen.

Moving the paddles

Moving the paddles up and down requires two parts. We need to make the paddles respond to the keyboard and define what the paddles will do when we type the assigned letter on the keyboard.

```
#move paddle a up and down
#move paddle a up
def pa_up ():
   y= pa.ycor()
   y+=15
   pa.sety(y)
```

To have proper keybinding, we need to have a function. We assigned the upward movement of paddle to the function pa_up. We explained to the computer in the function that the computer should assign the y coordinates of paddle a to a variable y and that the task whenever it is called upon should add 15 to the y coordinate of the paddle. This means that every time we use the function paddle up, paddle a should move upwards by 15 points

```
#move paddle a down
def pa_down ():
   y= pa.ycor()
```

```
  y-=15
  pa.sety(y)
```

What we did with the downwards movement is similar to what we did with the upward movement. The only difference is that we made the downward movement – 15 instead of 15. That is any rtime the function pa_down is called upon, paddle a moves down by 15 points or pixels.

```
#move paddle b up and down
#move paddle b up
def pb_up ():
  y= pb.ycor()
  y+=15
  pb.sety(y)

#move paddle b down
def pb_down ():
  y= pb.ycor()
  y-=15
  pb.sety(y)
```

Coding the movement of paddle b is similar to that of paddle a.

```
# keyboard binding
sr.listen()
sr.onkeypress(pa_up, 'w')
sr.onkeypress(pa_down, 's')
sr.onkeypress(pb_up, 'i')
sr.onkeypress(pb_down, 'k')
```

The keyboard binding assigns a function to a particular key on the keyboard. For the keyboard binding to work, we need to use the listen () function.

Afterward, we gained the functions pa_up to the key w, pa_down to the key s, pb_up to the key I, and pa_down to the key k.

Score

```
#score
```

```
score= turtle.Turtle()
score.speed(0)
score.color(255,123,20)
score.pu()
score.ht()
score.goto(0,210)
score.write('Paddle a: 0 Paddle b: 0', align= 'center', font = ('calibri', 25,
'bold'), )
score_a = 0
score_b = 0
```

For the score of the game, we assigned the score of the game to a paddle object by using the function turtle.turtle.
We made the speed of the score up to date 0 so that when there is a score, the score showing on the screen updates almost immediately.

We used the pen up function so that the lines of turtle movement of the turtle do not show. We also hide the turtle so that the turtle does not show when the turtle is moving. We also set the place where the score will show by assigning the score to x = 0 and y = 210.

We made the turtle show paddle a and b and the scores of 0, 0 on the screen. We also specified that the writing should be center-aligned and that the font show is ‘calibri, 25 amd bold.’

We also created two other variables for the score. We will explain these variables later.

Game loop

The game loop is where the action in the game is created.
A while loop was used so that the game can continue indefinitely.

GAME LOOP

```
while True:
```

```
sr.update ()
# moving the ball
ball.setx (ball.xcor()+ ball.dx)
ball.sety (ball.ycor()+ ball.dy)
```

First, we make the ball move. We specify in our loop that the ball should move by 1 pixel on the x-axis and -1 pixel on the y-axis do that the ball can move in a diagonal way.
Remember we earlier set ball.dx to 1 and baal.dy to -1.

This piece of code will cause the ball to begin to move.
We have created a problem, however. The ball begins to move off the turtle screen.

That brings us to the border checking.

Border Checking
Border checking is activated so that the ball moves within the screen. We use an if statement for the border checking.

```
# border checking upper border
if ball.ycor()> 235:
  ball.sety(235)
  ball.dy*=-1
```

We start with the upper border. We tell the turtle that once the ball gets past 235 on the y coordinate, the ball should be returned to position 235 on the same y coordinate and begin to move in the opposite direction.

The opposite direction command is specified with the code
Ball.dy*= -1, which means that the ball by which moves the ball on the y-axis by -1 pixel should begin to move the ball on the y-axis by +1 since -1 * -1 = +1.

```
# border checking lower border
if ball.ycor()< -235:
  ball.sety(-235)
  ball.dy*=-1
```

What we did to the lower border is similar to what we did to the upper edge. We only used a negative value this time around.

```
  # border checking right border
  if ball.xcor()< -390:
    ball.setx(-390)
    ball.dx*=-1
    ball.setpos(0,0)
    score_b +=1
    score.clear()
    score.write('Paddle a: {} Paddle b: {}'.format(score_a, score_b), align=
'center', font = ('calibri', 25, 'bold'))
```

For the right border, what we did is a little different. We need the score to change whenever the right edge is hit.
As such, we used an if statement. Whenever the ball gets to -390 on the x coordinate, the ball should move in the opposite direction after being a setback to position 390. Also, when the ball goes past – 390 on the x-axis, it should go back to the center of the screen, and from there, it can begin to go to the left.

For the score, we asked the turtle to add 1 to the score of the other paddle when the ball hits the right edge of the screen. We asked the turtle to clear the previous score with the function clear (). This is so that the new score does not write on top of the former score.
We then assigned the format function to the score. An f statement would have been a better way to do this. You should try using an if statement for your own code.

```
  # border checking left border
  if ball.xcor()> 390:
    ball.setx(390)
    ball.dx*=-1
    ball.setpos(0,0)
    score_a +=1
    score.clear()
    score.write('Paddle a: {} Paddle b: {}'.format(score_a, score_b), align=
'center', font = ('calibri', 25, 'bold'))
```

The process of checking the left border is similar to the operation of checking the right border.

Ball interaction with paddles
This is one of the critical parts of the game. It allows the paddles to hit the ball. It is the last part of our game.

```
  # ball interacting with paddle b

  if (ball.xcor()> 350 and ball.xcor() < 370) and (ball.ycor() < pb.ycor() +
50 and ball.ycor()> pb.ycor() - 50):
    ball.setx(350)
    ball.dx*=-1
```

We also used an if statement for this. We instructed the turtle that once the ball gets to between coordinates 350 and 370 on the x axis and the ball is within the height of the paddle (this was specified by the code: (ball.ycor() < pb.ycor() + 50 and ball.ycor()> pb.ycor() - 50)).
The ball should return to the x coordinate 350 and begin to move in the opposite direction.

```
  # ball interacting with paddle a

  if (ball.xcor() < -330 and ball.xcor() < -370) and (ball.ycor() < pa.ycor() +
50 and ball.ycor()> pa.ycor() - 50):
    ball.setx(-330)
    ball.dx*=-1
```

This is similar to what we specified for the paddle b.

In the end, we have the code for our ping pong game.

```
import turtle

sr= turtle.Screen ()
sr.title('ping pong game')
sr.colormode (255)
sr.bgcolor(50,100,50)
sr.setup (width= 800, height= 500)
sr.tracer (8)

#GAME OBJECTS
#ball
ball = turtle.Turtle()
ball.speed(0)
ball.shape('circle')
ball.color (105,0,0)
ball.penup()
ball.setpos (0,0)
ball.dx = 1
ball.dy = -1

#paddle_a
pa = turtle.Turtle()
pa.speed(0)
pa.shape('square')
pa.color ('black')
pa.shapesize(5,1)
pa.penup()
pa.setpos (-370,0)

#paddle_b
pb = turtle.Turtle()
pb.speed(0)
pb.shape('square')
pb.color ('black')
pb.shapesize(5,1)
pb.penup()
pb.setpos (370,0)

#move paddle a up and down
#move paddle a up
def pa_up ():
   y= pa.ycor()
   y+=15
   pa.sety(y)

#move paddle a down
def pa_down ():
   y= pa.ycor()
   y-=15
   pa.sety(y)


#move paddle b up and down
#move paddle b up
def pb_up ():
   y= pb.ycor()
   y+=15
   pb.sety(y)

#move paddle b down
def pb_down ():
   y= pb.ycor()
   y-=15
   pb.sety(y)

# keyboard binding
sr.listen()
sr.onkeypress(pa_up, 'w')
sr.onkeypress(pa_down, 's')
sr.onkeypress(pb_up, 'i')
sr.onkeypress(pb_down, 'k')

#score
score= turtle.Turtle()
score.speed(0)
score.color(255,123,20)
score.pu()
score.ht()
score.goto(0,210)
score.write('Paddle a: 0 Paddle b: 0', align= 'center', font = ('calibri', 25,
'bold'), )
score_a = 0
score_b = 0

#GAME LOOP

while True:
  sr.update ()
  # moving the ball
  ball.setx (ball.xcor()+ ball.dx)
  ball.sety (ball.ycor()+ ball.dy)

  # border checking upper border
  if ball.ycor()> 235:
    ball.sety(235)
    ball.dy*=-1

  # border checking lower border
  if ball.ycor()< -235:
    ball.sety(-235)
    ball.dy*=-1

  # border checking left border
  if ball.xcor()< -390:
    ball.setx(-390)
    ball.dx*=-1
    ball.setpos(0,0)
    score_b +=1
    score.clear()
    score.write('Paddle a: {} Paddle b: {}'.format(score_a, score_b), align=
'center', font = ('calibri', 25, 'bold'))


  # border checking right border
  if ball.xcor()> 390:
    ball.setx(390)
    ball.dx*=-1
    ball.setpos(0,0)
```

```
        score_a +=1
        score.clear()
        score.write('Paddle a: {} Paddle b: {}'.format(score_a, score_b), align=
'center', font = ('calibri', 25, 'bold'))

    # ball interacting with paddle b

    if (ball.xcor()> 350 and ball.xcor() < 370) and (ball.ycor() < pb.ycor() +
50 and ball.ycor()> pb.ycor() - 50):
        ball.setx(350)
        ball.dx*=-1

    # ball interacting with paddle a

    if (ball.xcor() < -330 and ball.xcor() < -370) and (ball.ycor() < pa.ycor() +
50 and ball.ycor()> pa.ycor() - 50):
        ball.setx(-330)
        ball.dx*=-1
```

You should realize that the turtle module is drawing the whole thing, but the drawing is happening really fast.

This game has several bugs. You should try to find them and try to reduce the amount of code we used.

Congrats, you have created a game on turtle!!!!

You have earned it. You are now a full wizard of Adak. ____ the great, wizard of the order of Growrher, Merlin and Python.

Signed   High Mage Eragdan order of Growrher, Merlin and Python.

Final Notes

There are several things that could be done with python. What you have been introduced to in this book is but a very small part of a much larger whole. Fear not, however. You have learned the very basics of coding.

You should work on meaningful projects that interest you. Coding is more tempting when you're solving relevant and compelling problems, and you now have the skills to engage in an array of projects.

You should try to create more games. You could try to make more arcade type games like Tetris or make new games which you made yourself totally from scratch.

If possible, invite people to try using your programs. If you write a game, let people play it. Listen to your users and try to incorporate their feedback into your projects; you'll become a better programmer.

Whenever you run into hard walls when writing your program, go online, and find help. There are numerous communities of coders online, willing to help you with any problems you might encounter.

You could also download other books on python and learn more about it. [1]

Maybe it will be a job for you, or maybe it will be a hobby. You'll need some aid to make sure you continue on the right path and get the most joy out of your newly chosen activity.

Coding as a profession, it can be a good job.
People who can program in biology, medicine, government, sociology, physics, history, and mathematics are admired and can do awesome things to advance those disciplines.

You should try programming to improve your life in any way you can. Go out and explore this wonderful and new rational pursuit.

Learning to code changes you and makes you different. Not better or worse, just different. You may find that people treat you firmly because you can program, maybe using words like “nerd.” Perhaps you’ll find that because you can dissect their logic, they hate arguing with you.

Don’t ever listen what they say. The world needs more curious people who know how things work and who love to solve problems.

Being different is not bad, and people who tell you it is are just envious of you.

A lot of free documentation is available online. You can use Java on most operating systems.

Congratulations wizard, you have reached the end of this book!!!!!

---

[1]