



# The Search for a Title

A Profound Subtitle

Author



Copyright © 2019 John Smith

PUBLISHED BY PUBLISHER

WWW.ZIMCODE.ORG

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the “License”). You may not use this file except in compliance with the License. You may obtain a copy of the License at <http://creativecommons.org/licenses/by-nc/3.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

*First printing, March 2019*

# Contents

I	Part One	
<b>1</b>	<b>Introduction</b> .....	<b>7</b>
1.1	What is Zim Code?	7
1.2	Purpose of the textbook	7
<b>2</b>	<b>Introduction to Computers and Python</b> .....	<b>9</b>
<b>2.1</b>	<b>Introduction to Computers</b>	<b>9</b>
2.1.1	Operating Systems .....	10
2.1.2	File Systems .....	10
2.1.3	The Command Line and the Graphic User Interface (GUI) .....	11
<b>2.2</b>	<b>Introduction to Programming</b>	<b>13</b>
2.2.1	Why Python? .....	14
2.2.2	The IDE .....	14
<b>3</b>	<b>Variables</b> .....	<b>19</b>
<b>3.1</b>	<b>Abstraction</b>	<b>19</b>
<b>3.2</b>	<b>Variables</b>	<b>20</b>
<b>3.3</b>	<b>Rules for naming variables</b>	<b>21</b>
<b>3.4</b>	<b>Python keywords and Comments</b>	<b>22</b>
<b>3.5</b>	<b>Syntax and Semantics</b>	<b>22</b>

<b>4</b>	<b>Presenting Information</b> .....	<b>27</b>
	<b>Bibliography</b> .....	<b>29</b>
	Articles	29
	Books	29
	<b>Index</b> .....	<b>31</b>



# Part One

<b>1</b>	<b>Introduction .....</b>	<b>7</b>
1.1	What is Zim Code?	
1.2	Purpose of the textbook	
<b>2</b>	<b>Introduction to Computers and Python .</b>	<b>9</b>
2.1	Introduction to Computers	
2.2	Introduction to Programming	
<b>3</b>	<b>Variables .....</b>	<b>19</b>
3.1	Abstraction	
3.2	Variables	
3.3	Rules for naming variables	
3.4	Python keywords and Comments	
3.5	Syntax and Semantics	





# 1. Introduction

## 1.1 What is Zim Code?

Zim Code is a nonprofit coding school that was founded by Alvin Chitena in 2016. Our mission at Zim Code is to bring coding to the forefront of youth education and empowerment in Zimbabwe. Code is the language of the present and the future, and so we believe that it is necessary for everyone to have the opportunity to learn it. Zim Code is made of young and dedicated Zimbabweans from all walks of life who are passionate about coding, education, entrepreneurship and philanthropy.

Given the vast socio-economic issues that plague Zimbabwe, we see it necessary to remove the many barriers that people may face when trying to learn how to code, namely quality information and resources. This course, the Introduction to Programming: Python (12-weeks) is one of the many initiatives we are spearheading to further our mission.

To find out more about Zim Code, visit our website: [www.zimcode.org](http://www.zimcode.org)

## 1.2 Purpose of the textbook

This text is designed to cater specifically to high school students in Zimbabwe and other African countries whose education system is based on the GCE A level curriculum. The idea is to provide a low barrier text in writing that is easy to understand for African high school students with very little assumptions made about their background.

A wide range of relatable examples are used throughout the text to provide analogies and help students grasp the abstract concepts in Computer Science and Programming. This is done with the hope that students may begin to appreciate how Computer Scientists and Software Engineers solve problems and how programming skills can be applied in the real world.

In addition to learning the basics of coding, students will also inevitably learn and improve in other subject areas. The link between Computer Science and Mathematics is undeniable and in fact, fundamental. While it is easy to fall into the temptation of providing mostly computational and mathematical examples, we tried our best to include a wide range of examples appealing to all three major fields of study: Science, Commerce and Humanities.

This course will give a 12-week introduction to programming in Python which is one of the

most accessible languages to beginners due to its similarity to written English and white space delimiting. White space delimiting is the use of indentations and line breaks rather than brackets to group code blocks e.g. in a function/loop. Python is also a very powerful tool in computation (The NumPy package is used in various fields for numerical computation) and web design (Instagram is written in Django/Flask, Python frameworks) with a very good support and development network. It is also easy to install and is often preinstalled on most operating systems (unfortunately, not on Windows).

An important part of this text is that it should be fun and interactive to the students. This text is "minimal" in that to benefit the most out of the content, the reader has to go through all the readings, examples and exercises. It involves live demos and even tries to introduce real world applications early on. Examples are chosen from real world problems so that it becomes immediately apparent to the students how programming applies to real life. Students will also learn a lot of broader concepts in programming such as abstraction, functional programming, and algorithmic thinking.

This course is designed to be open, fun and interactive. We encourage deviating from the standard lecture model because the best way to learn programming is by actually doing it. The style will be a combination of lecture and lab type classes where a concept is introduced for a few minutes, the students get to use it and then reflection on the use of the concept will hopefully bring to light its value and clear any misconceptions.

While we provide a detailed lesson plan to teachers in addition to this textbook, experimentation and flexibility is encouraged so that the most can be made from every class. If something does not work particularly well, change it and if the teacher has better ways of teaching the class in their context, they should.

We would like feedback on what worked and what didn't, if you have any, email [info@zimcode.org](mailto:info@zimcode.org)





## 2. Introduction to Computers and Python

### 2.1 Introduction to Computers

A computer is a programmable electronic device that manipulates information, or data. It has the ability to store, retrieve, and process data. Your laptop, desktop, smartphone or tablet are all examples of computers.

The computer system is made up of three things:


**Input** – This is the information that a computer gets from outside, it will never act on its own e.g. when you type or touch a screen or talk into a microphone, you are giving an input of character. Things like keyboards, mice and touchscreens are called input devices.

**Output** – This is what the computer gives out e.g. the picture on your screen, a sound or turning off the lights.

**Processing** – These are the instructions and calculations that the computer must follow to do something with the input. For all the devices you use, you are used to being involved with the input and output part. In programming, you get to directly design the processing part of a computer.

**Problem 2.1** Can you name different examples of input, processing, output devices?

Here is how to think about the way a computer works that you should use about throughout this entire book.

 A computer takes inputs and processes them using a set of rules designed by a programmer(a human) to produce an output.

The fact that the rules that govern a computer are designed by a human is very important. It means that computers can make mistakes if the human programmers made mistakes! The computer will follow instructions exactly as you *program* it to, not exactly as you *want* it to. As you start your journey as a programmer, you will make lots of mistakes and that is normal but it is important to remember to make sure things work properly, we will try and teach you how in this book.

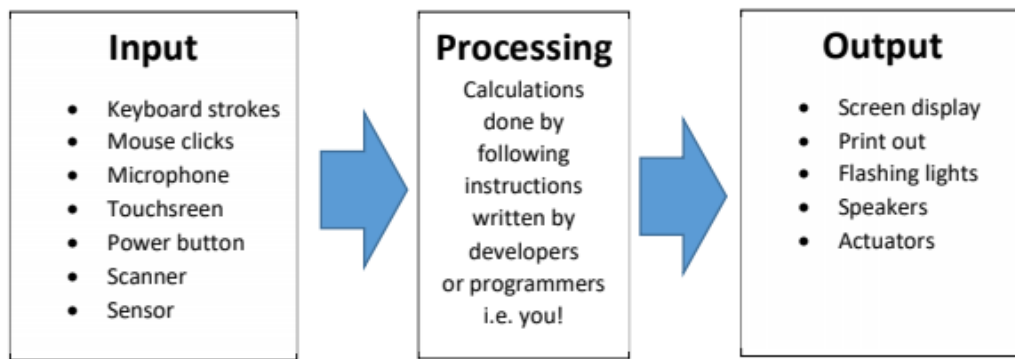


Figure 1

Figure 2.1: (Figure 1) Picture of the Computer Model: a

### 2.1.1 Operating Systems

An operating system is the primary software that manages all the hardware (screen, keyboard, touchscreen etc.) and software (Microsoft Word, Google Chrome, Whatsapp etc.) on a computer. The operating system, also known as an “OS,” *interfaces* (connects) with the computer’s hardware and provides services that applications can use. For example, if you click print in Microsoft Word, Word sends the document you want to print to the OS, which in turn communicates with the printer using special software called “drivers” in order to tell the cartridge or laser inside the printer what shapes to make on the paper using ink.

Examples of Operating Systems are: Windows 10, MAC-OS, LINUX etc. Operating Systems are constantly being updated to work more efficiently and cater to new demands. This course uses Windows since that is the most popular OS available in Zimbabwe.



Figure 2.2: (Figure 2) Icons from various operating systems: null - needs to be made

### 2.1.2 File Systems

One of the crucial things an OS does is to organize our files and folders: documents, movies, pictures, etc. A file system is a way of organizing and storing files on a hard drive, flash drive, or any other storage device.

**Definition 2.1.1 — Hard drive.** A storage device for data. Data is any kind of information like pictures, documents, videos, music etc. All the data your computer needs to function will be stored there.

In Windows, the Hard Drive on your computer is usually denoted as “C:” although it can be denoted by other letters. C: means all the stuff stored on this hard drive. In that C:, you can have files and folders inside it.

**Definition 2.1.2 — File.** Anything that stores data that can be opened by a special program e.g. an mp3 file can be opened by a music player, a .docx file can be opened by Microsoft Word, An executable file can run by the OS, a .py file can be run by the Python interpreter.

There are many different types of files, and their type is determined by their extensions. For example, you can tell that *ZimCode.txt* is a text file because of its extension *.txt* at the end of the file name. This extension is important because it helps the OS know how to open and edit that file. If you click a *.txt* file, the OS will use Notepad or MS Word to open it. If you click a *.mp3* file, the OS will use a Media Player to open it. It decides how to open the files using their extension.

**Definition 2.1.3 — Folder/Directory.** The software object that stores files or other folders in a file system. It is also called a *directory*. It is not a file, but just a storage location for files and folders.

Here is an example of a folder structure:



Figure 2.3: Image of a folder hierarchy from windows explorer, preferably showing C:/, Users, Documents, Pictures etc. and matching the description below

Figure 2.3 shows a folder "hierarchy" that shows you what is contained in each folder. Notice that the harddrive C: is at the root meaning that everything is stored on C:. The folder "Documents" is inside the folder ZimCode which is inside the folder Users and contains 3 files and 1 folder. The folder inside it contains 2 files. The folder doesn't have any extensions on it because it doesn't store any data on its own. It is just a storage reference location for files and other folders. Think of it as a paper bag. You can put an apple, banana and water bottle inside it. You can also put another paper bag inside the paper bag with pencils. And so on, and so forth.

**Definition 2.1.4 — Path.** The location of a file or folder on the hard drive for a given filesystem. This is similar to the address of the place where you live. In computer terms, we commonly refer to it as a path i.e. the path you need to take to reach the folder/file. The path of "Star Wars: A New Hope (1977).mp4" could be something like "C:\Users\ZimCode\Documents\Star Wars: A New Hope (1977).mp4". Notice that the "Documents" folder is stored in another folder. Just like addresses, it is impossible for 2 files to have the same address on one computer.

**Problem 2.2** Can you think of why it would be bad for two files to have the same path?

### 2.1.3 The Command Line and the Graphic User Interface (GUI)

A GUI (graphical user interface) is a system of interactive visual components for computers. It helps us visualize the files stored on a computer in a human-readable format. For examples, if you want to open the Movies folder to watch your favorite action film stored inside, all you do is move the mouse cursor over the Movies folder and click on it. The whatsapp symbol is also an example of GUI component that you touch to open Whatsapp on a phone.

■ **Example 2.1** Example where they create a new folder and open a file using the GUI ■

Before we had the GUI, we only had the Command Line Interface (CLI). Back then, if you

wanted to watch your favorite film, you had to do so by typing a command in the command line. This is like what they do in the movies with hackers.

To open the command line (a program called command prompt) in Windows, go to start and type **cmd** and press enter. You can open command prompt any other way you like. You should see a screen like the one shown in Figure 2.6.

Generally, your command has to specify some or all of the following, using memorized keywords:

1. The name of the program you are using to perform the action.
2. The path of the file you want to perform an action on.
3. The specific action you want to perform.
4. Optional parameters of the action e.g. how many times you want to perform the action



Figure 2.4: Command line figure showing the same Documents folder as above

If all I had was the command line, instead of right-clicking my movie file, clicking Open with, and choosing VLC Media Player I would have to type a command like this in the command line to watch it:

---

```
> start vlc "C:\Users\ZimCode\Documents\Star Wars: A New Hope (1977).mp4".
```

---

Imagine having to do this every time you want to do anything on a computer? Well, before the invention of the GUI, that is how people generally did things. An example of an early operating system that used this system is MS-DOS, which stands for Microsoft Disk Operating System

We still have the Command Line in modern computers because it helps us complete certain tasks quickly and more concisely. We can do most of the things we can do using a GUI with the CLI, in the following example, we repeat what we did before but this time using the CLI.

■ **Example 2.2** Example where they create a new folder and open the same file using the command line ■

Here are some more basic command line commands:

- **help** - line lists all the basic built-in commands at your disposal.
- **dir** - list all the contents inside the current directory
- **cd** - change directory
- **mkdir** - make a new directory
- **cls** - clear the contents of the command line screen

### Exercise 2.1 Master Hacker!

You are going to use the following commands on the command line (Command Prompt) to create new text file called “myfile.txt” and put it in a new folder you make in “My Documents.”

1. **cd** into drive C:
2. Use **dir** to check if there is a folder called something like “users”

3. **cd** into the users folder
4. Use **dir** to see the name of the user folder (something like ZimCode) and **cd** into it
5. Make a new folder called “MyText” using **mkdir**
6. **cd** into the folder
7. Use the command  

```
notepad myfile.txt
```

  
to create a new text file and save it.
8. Use the GUI browser (Windows Explorer) to see if the file is where you put it.

### Exercise 2.2 Genius Hacker!

Try to figure out how to open the Zim Code website ([zimcode.org](http://zimcode.org)) in the Google Chrome browser, using only the command line and what you have learnt so far. Remember, you can ask for help from the Command Line by literally typing "help".

This is just a small example of the many things you can do in the command line. The whole idea of the command line is a standardized way to tell a computer what to do. In the next topic, you will learn about one very special way of doing this.

## 2.2 Introduction to Programming

Programming is writing down a set of instructions for the computer to follow. How do you give instructions to someone? You have to communicate using a language. There are many programming languages that exist, we are going to teach you Python. Python is called a high level interpreted language. We will explain what this means a bit later.

Computers “think” only in ones and zeroes yet they are capable of doing so many things. How is this possible? A computer is basically a calculator, all it actually does is calculations and from these calculations, you can do more complicated things. So if a computer only understands ones and zeroes, how do we communicate with it? Just like any other language, you have to translate it so that other people can understand, the ones and zeroes are translated to other languages until they are translated into a high level language. A high level language is a language that humans can read and understand easily unlike ones and zeros.

When we say Python is an interpreted language, we mean that it follows instructions one by one in the order that you write them then if it gets to an error it crashes. If you give two instructions like:

1. Go outside
2. Jump to the moon

python will go outside and then it will realize that it can't jump to the moon and give you an error.

The opposite of an interpreted language is a *compiled* language. A compiled language first checks to make sure there are no errors before it runs. So if you gave it the two instructions above it will not even try and go outside, it will immediately tell you that it cannot jump to the moon.

Unfortunately, computers are not very clever, in fact a computer is a very stupid machine that blindly follows any instructions you give it. It will always try to do whatever you tell it to do. It also follows its language very strictly, if you put a small (lower case) letter where there is supposed to be a capital (upper case) letter, it will be completely confused and give you an error. You will see this as we go along because you will definitely have lots of errors.

We choose to teach Python because it is a relatively simple yet extremely powerful programming language to understand. Some examples of apps/websites written in Python are Instagram, Pinterest, Parts of Google and most of YouTube and many more. The most important reason is that because it is simple and very useful, it will allow you to learn the core concepts in programming which allow you to learn other languages much faster.

I hope at that we have convinced you to learn Python because no matter who you are, it will likely be useful at some point in your life if you need to program anything.

So how do we write code? We use something called an Integrated Development Environment (IDE)

After following the instructions on how to install Python 3, open the program IDLE that was installed with Python. (Try doing this using the Command Line!) IDLE is the IDE that we will be using to write and run most of our code in this course. Whenever you install Python, IDLE will also be installed. When you open it, your screen should look like :



Figure 2.5: What you see when you open IDLE on Windows

This is called the Python Shell, it is what allows you to enter in commands one by one. Can you think of another shell we have seen before?

Now it's time to write our first line of code! In the shell in front of >>>, type in the following command

You have written your first line of code! It tells Python to output/print the words “Hello World” to standard output. Standard output means it just prints the words in the shell after the command is run. Notice how you typed in an input (command), the computer processed that input and it gave you as an output (printed words).

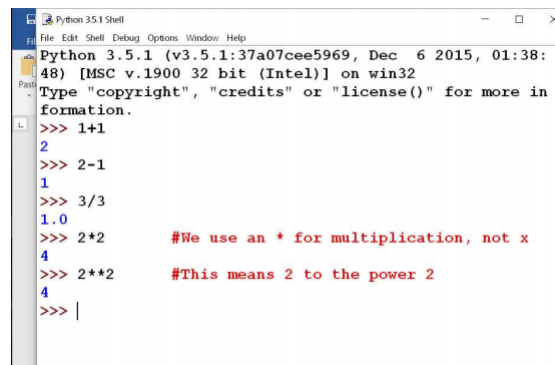


**Exercise 2.3 Say my name** Change the words inside the quotation marks("") so that it say "Hello" and followed by your name. Do not remove or add anything else! You have learned your first piece of code using the print() function! ■

### Arithmetic

Earlier we said a computer is a calculator, let's use the computer as a calculator. Execute the following code, pressing enter after every line to see the output and see if it gives you the right answer. Do not include the things that come after the "#" symbol.

■ **Example 2.3** Python for calculations ■



```
Python 3.5.1 Shell
File Edit Shell Debug Options Window Help
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:
48) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more in
formation.
>>> 1+1
2
>>> 2-1
1
>>> 3/3
1.0
>>> 2*2           #We use an * for multiplication, not x
4
>>> 2**2          #This means 2 to the power 2
4
>>> |
```

Figure 3

Figure 2.6: Using the Python shell as a calculator

**Exercise 2.4 Back to first grade** Use the symbols you used above to try and do more calculations. Now execute the following code:

```
>>> 5//2
>>> 3//2
>>> 29//5
```

What do you think the "//" symbol does? This is called floor division, it gives you the whole number from when you divide two numbers. E.g. 5/3 is 1 remainder 2 but the floor division operator only gives you the whole number part which is 1. Now execute this code and try and figure out what it does.

```
>>> 3%2
>>> 3%3
>>> 3%4
>>> 3%7
```

The "%" symbol is called the "mod operator", it gives you the remainder after division. Let's make our calculations a bit more complicated. The rules of BODMAS (Brackets, pOwers, Division, Multiplication, Addition, Subtraction) that you learned in primary school work here just like for your calculator. Decimals in Python use a full stop which is usually called a period (.), not a comma (,). Notice that even when you are multiplying things in brackets, you have to put the \* operator!

**Exercise 2.5** BODMAS Do the following calculation by head and then confirm it on the shell

```
>>> 2*(2)
>>> 2(2) #Why is this an error?
>>> 17//7 + 23%3 + 3**2
>>> (13+(34/2)*23%4-12)
>>> (23//5)*7/(12%5)+5 \#19.0
>>> ((67%5)*(-4*-0.5))*0.5 \#2.0
```

### Max/Min/Round

We can also find the maximum and minimum of a group of numbers separated by commas. What This is called importing the math module, we will be importing many more modules so remember how it's done.

```
>>> max(1,2,3,4,5)
>>> min(1,2,3,4,5)
>>> max(24,3,7,-3)
>>> min(23,-4,0,89)
```

To round off numbers we use the round() function.

```
>>> round(9.4)
>>> round(9.7)
```

### The math module

If we want to use Python as a scientific calculator, to do this we have to borrow the tools from somewhere else using the following code:

```
>>> import math
```

To use the more advanced calculations, we always have to execute the code above. Let's see what we can do.

```
>>> math.sqrt(4) #Find squareroot of 4
>>> math.sin(1) #Find sin(1)
>>> math.pi #The number or pi
>>> math.cos(math.pi) #Find cos(\(|pi|\)) or cosine of pi
>>> math.log10(100) #Find log(100) to base 10
```

Remember that this will not work if you do not import the math module first!

**Exercise 2.6** Scientific calculator a) Close IDLE and open it again. b) Calculate the area of a circle of radius 2 using math.pi c) Copy the answer and round it off to the nearest unit using round()



So far you have been typing in commands one by one and executing them, but what if you are writing some really complicated code like a music player. It would be disaster if it executed your code right after you've typed it, it's like writing a composition with an examiner who marks after every sentence. You want to first write out parts of your code, test them and then write more and if you need to change anything, you can go back. In Python, you can write code using a script which allows you to write a lot of code at once. In IDLE, go to File > New or press Ctrl + N. You should see a blank screen. Type in the "Hello World" code from before and then go to Run > Run Module or press F5. It should give the same output as before.

Now let's write some more complex code.

```
1 name = input("What is your name?: ")
2 print("Hi", name)
```

This code asks you to type in your name and then when you press enter, it greets you. When you run a script, it never prints anything unless you use the print() function unlike the shell where if you type in a calculation, it immediately gives you an output.

### Exercise 2.7 So many questions

Change the words inside the quotation marks only so that it asks you different things and gives you a different answer e.g. asking for your age then make it print "You are 17". Only edit the text in quotes. ■

### Summary

You should now know:

- That a computer take inputs, processes them and gives outputs, and give examples
- The basics of what an Operating System and File system are
- The differences between a GUI and CLI
- How to make a folder, change directory and open files using both the GUI and CLI
- That Python is a high level interpreted programming language and what that means
- What is IDLE and how do you open it?
- The differences between a script and a shell
- Writing the Hello World application in both a script and in the python shell
- How to do basic arithmetic operations as well as using floor, mod operators, min, max and round functions with BODMAS rules
- How to import import the math module and do basic calculations using cos, sin, pi etc.
- How to write a simple application that takes input from the user and prints it to standard output





## 3. Variables

### 3.1 Abstraction

So far, we have only been considering things that an ordinary calculator can do with just a few simple operations. The first and probably the most powerful tool in programming is abstraction. Abstraction is the use of an idea rather than an event. This is best shown through examples.

■ **Example 3.1 Using a formula** If I were to tell you that to calculate the distance travelled by a car I did the following calculation:

$$distance = 2 * 3 = 6$$

Is it possible for you to know how I did it? What is 2? What is 3? What units did I use? Is it 6 metres per second or 6 kilometres per minute? Clearly this is not useful if I want to learn how to calculate distance by myself. How do we know how to calculate distance?

The answer is that we know the formula:

$$distance = speed * time$$

Now, we can calculate any distance given any speed and any time. This is what we mean by abstraction. Rather than using an event (i.e. exact numbers) we use an idea (i.e. a formula). ■

#### **Exercise 3.1 Science – Making a complicated calculation easy**

Imagine that you are the assistant to a crazy physicist named Larmour who needs to know the power radiated from a point (The point is that this is a complicated calculation, you do not even need to know what it means). Larmour says he wrote his formula on the board and needs the answer to finish making a machine that gets rid of all homework so you happily help him.

The formula is:

$$P = \frac{e^2 a^2}{6\pi\epsilon_0 c^3}$$

He also wrote what number each of the symbols is:

$$e = 1.6 \times 10^{-19}$$

$$a = 1$$

$$\epsilon_0 = 8.85418782 \times 10^{-12}$$

$$c = 3 \times 10^8$$

- i What is P? The point is that you can calculate a very complicated thing just because you know the formula or because Larmour “abstracted” the idea.
- ii Suppose Larmour says he made a mistake and a is actually 2, how do you use your answer from i) to fix this easily?



## 3.2 Variables

Let's go back to the example with distance where we have:

$$distance = speed * time$$

We have multiplied before so we know what the “\*” does. There are two new things i.e. the three words and the “=” sign. The three words are called variables, which has the same meaning as in mathematics. Variables are containers that hold something e.g. “time” holds the number for a time, “speed” the number for a speed. When we want to calculate the final answer we replace each of those “variables” with the actual number. When calculating the distance and we want an actual number for the distance, we need to know the speed and time first. If we know the speed and time first, we first multiply the numbers on the right and then the “=” sign puts them in the “distance” variable. How do we know it is a variable? Most of the things we are going to work with in Python are going to be variables, except the few cases we will point out.

**Exercise 3.2 Exercise 2.1 – Declaring variables** In the shell, execute (type it then press enter) the following code:

```
>>> x = 3
>>> print(x)
>>> y = 4
>>> print(y)
>>> print(x+y)
>>> x = y
>>> print(y)
>>> name = "Python"           #You can put your own name here
>>> print(name)
```

Notice how the letter **x** now stands for 3 and **y** stands for 4 and you can do the things you can do to numbers to **x** and **y** because **x** and **y** actually represent the numbers in the calculation.

■ **Example 3.2 Look, no hands!** Here is another way to understand variables in Python. Let us say each of your hands is a variable. Your hands can only hold one thing at a time and you have 5 pens of different colors in front of you. Pretend you are a computer, meaning you follow instructions exactly without thinking about it and explain what happens in the following cases (The answers are as simple as you think!)

- a I put (assign) a black pen in your right hand and ask you to write your name.
- b I put a red pen in your left hand and ask you to write the date
- c If your hands are empty what goes wrong if I ask you to write the time?
- d Now let's say you have 3 hands (left, middle and right). What if you have a black pen in your right hand, a red pen in your left hand and I ask you to put the black pen in the left hand and red pen in your right hand? You can use your middle hand but you can only have one pen in a hand at a time.

### Exercise 3.3 Python Siri

- a Now that we can code, we want to make our own simple version of iPhone's Siri that can greet someone by their name. Write a script that asks for the user's name using **input()** and then asks for the age and then prints out a greeting that says "Hi, [name], you are [age] years old" where [name] is replaced by the person's name and [age] is replaced by the person's age. Think about what variables you can use and how you would name them.
- b Write a script that exchanges the values of two variables **x** and **y**. Think about the demonstration you saw/did and how you would do it in the demonstration and do it in code using only three lines. (Hint: You can use as many variables as you want)

## 3.3 Rules for naming variables

You may have noticed how much freedom we have when naming variables, this allows us to be able to make smart decisions about how we name them. If you have a long script with lots of variables, you want to be able to know what that variable is easily instead of all your variables being **x** or **y**. Here are some rules for naming variables, the first 5 rules are not optional, you will get errors if you do them wrong.

1. Never use python keywords (e.g. **print**, **def**, **import** etc.) Basically, if the word changes colour when you type it, do not use it as a variable.
2. Do not use the name of a module you have imported (e.g. **math**). If you import something, its name has been taken, you should not use it for a variable.
3. Do not use any punctuation except underscores (**\_**) in your variable names.
4. Do not use spaces in your variables, if you want to have a space, use an underscore (**\_**) e.g. **two\_words**.
5. Your variable name should never start with a number (e.g. **1st\_variable**).
6. Give your variables names that anyone can understand (e.g. **name**, **age**, **number\_of\_cats**).
7. If your variable is going to be changing, use only lower case letters e.g. **cost\_of\_food**.
8. If your variable is just a value that you input once at the beginning of the document use all

upper case letters e.g. `NUM_WHEELS_ON_CAR`.

### 3.4 Python keywords and Comments

Python has special words that mean special things, you can always see a keyword by the way it changes colour when you type it in e.g. `if`, `import`, `elif`, `print` etc. These should never be used as variable names.

We say Python is human readable but sometimes our code gets complicated and someone who did not write the code might want to understand what it does. The person who writes the code can write comments to explain what is going on. Comments always start with the pound sign (aka hashtag) symbol (`#`). Anything written on the same line in a different colour after the `#` is ignored by Python, so you can write anything you want. Usually they are used to explain steps. You are encouraged to always write your own comments to explain what you are doing, even to yourself. Here is an example where comments are used:

```
1 x = 1          #Declare a variable x as 1
2 x += 1        #Add 1 to x
3 print(x)      #Print the new x
```

### 3.5 Syntax and Semantics

So you have been typing code and you have probably run into errors already, for now all the errors you have come across are “syntax” errors. Syntax has to do with rules of a language, things like grammar, spelling and punctuation are the syntax of the English language. Similarly in Python, every bit of code that you type must follow the syntax of Python otherwise you will get a syntax error like the one below when you forget to close the round bracket.



Figure 3.1: Missing figure

Another type of error is what is called a semantic error, this is an error where your code runs but it does not do what you want it to, for example if you want to add two numbers but Python subtracts them instead. Another example is when you say or write something in English but it does not mean what you think it means even though it is spelled and punctuated correctly. These errors are very dangerous because Python will not tell you that your code is wrong, so you have to test your code and if it has problems, you “debug” it. Debugging means fixing code that does not work properly.

#### Toolbox 3.5.1 The basics

1. Variables
2. Comments

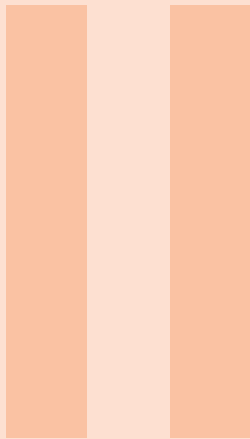
---

3. Coding vocabulary (syntax, semantics, debugging)

---







# Part Two

<b>4</b>	<b>Presenting Information .....</b>	<b>27</b>
	<b>Bibliography .....</b>	<b>29</b>
	Articles	
	Books	
	<b>Index .....</b>	<b>31</b>





## 4. Presenting Information

Referencing Table ?? in-text automatically.





## Bibliography

### Articles

- [1] James Smith. “Article title”. In: 14.6 (Mar. 2013), pages 1–8.

### Books

- [2] John Smith. *Book title*. 1st edition. Volume 3. 2. City: Publisher, Jan. 2012, pages 123–200.



## Index

### C

Citation .....	8
Corollaries .....	10

### D

Definitions .....	9
-------------------	---

### E

Examples .....	10
Equation and Text .....	10
Paragraph of Text .....	11
Exercises .....	11

### F

Figure .....	15
--------------	----

### L

Lists .....	8
Bullet Points .....	8
Descriptions and Definitions .....	8
Numbered List .....	8

### N

Notations .....	10
-----------------	----

### P

Paragraphs of Text .....	7
Problems .....	11
Propositions .....	10
Several Equations .....	10
Single Line .....	10

### R

Remarks .....	10
---------------	----

### T

Table .....	15
Theorems .....	9
Several Equations .....	9
Single Line .....	9

### V

Vocabulary .....	11
------------------	----