Chapter 6

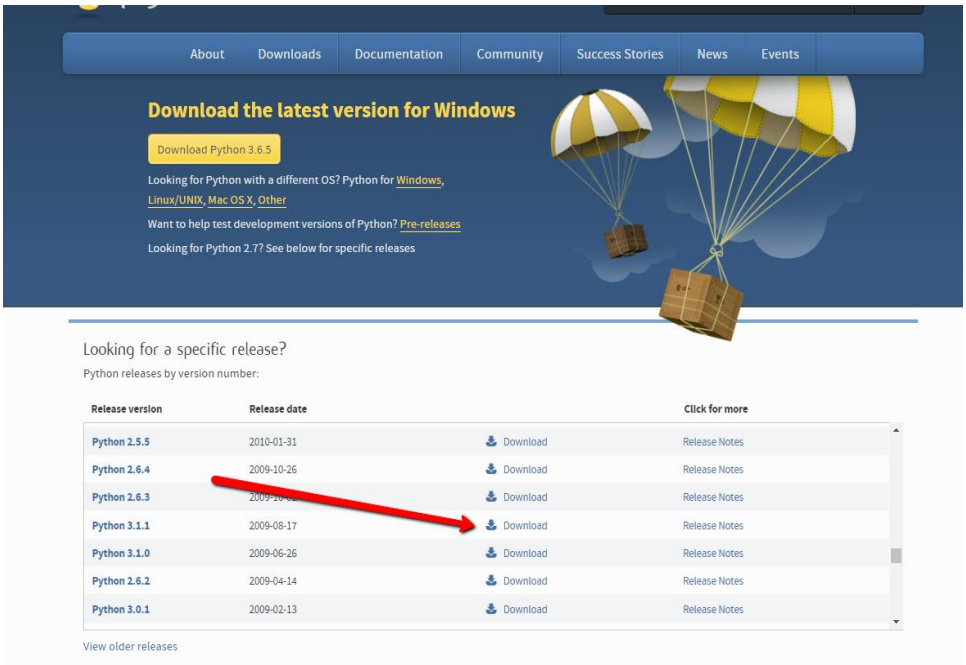
Introduction to Python

In this chapter we will learn the programming language Python. It is an easy to use language, perfect for learning. We will go over basic syntax and semantics of the language.

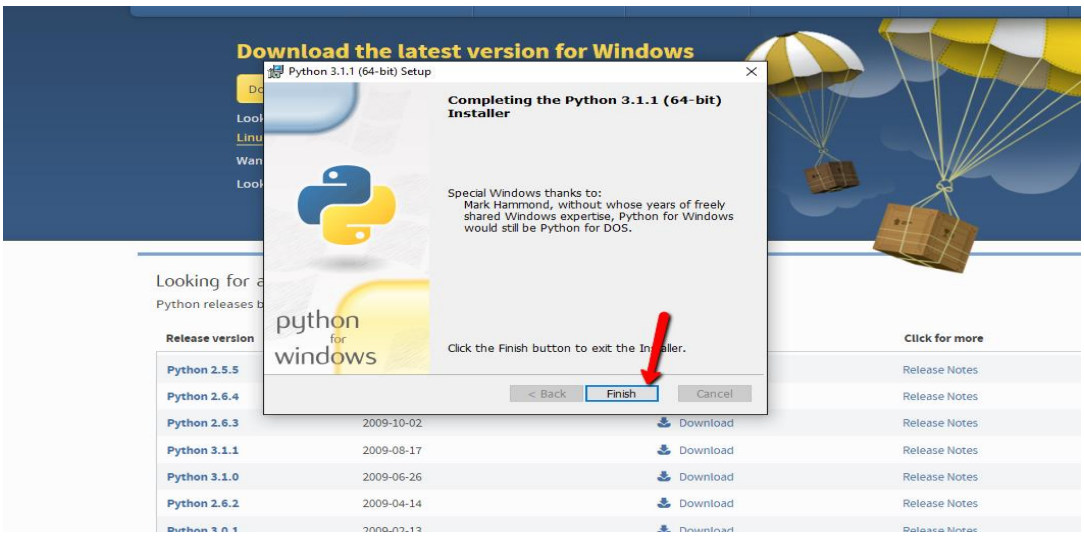
Intro to Python

Python is an object-oriented language, meaning it uses objects to store data in and manipulate through methods, which are blocks of code. This language focuses on having neat and elegant syntax, making it easy to learn programming with. Examples of Python's use can be found in YouTube, Spotify, Instagram, and BitTorrent.

To start, let's set up Python on your computer. Navigate to Python's official website and go under Downloads, to All Releases. Here, you'll need to scroll down the list of previous Python versions to find Python 3.1.1. Depending on your system type, scroll down on the 3.1.1 release page and download the Windows x86 MSI Installer (3.1.1) OR Windows X86-64 MSI Installer (3.1.1). If your computer is 32-bit system, download the first of the two. If you have a 64-bit computer, be sure to download the other version.

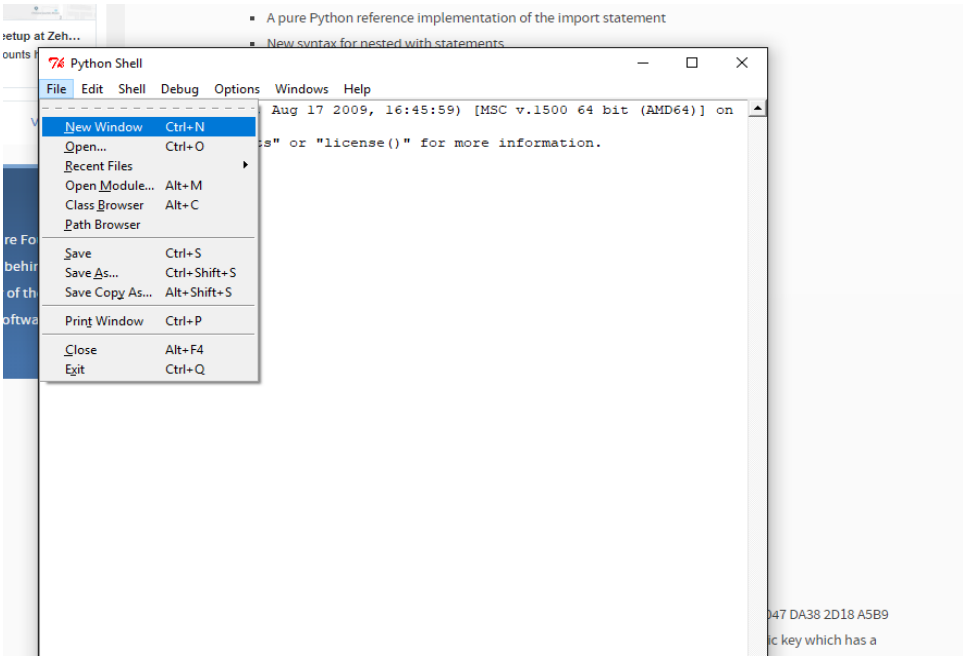


An installer will download to your computer, once it has finished double click the installer in your downloads folder. Click next through the wizard and allow administrative permissions when it asks.

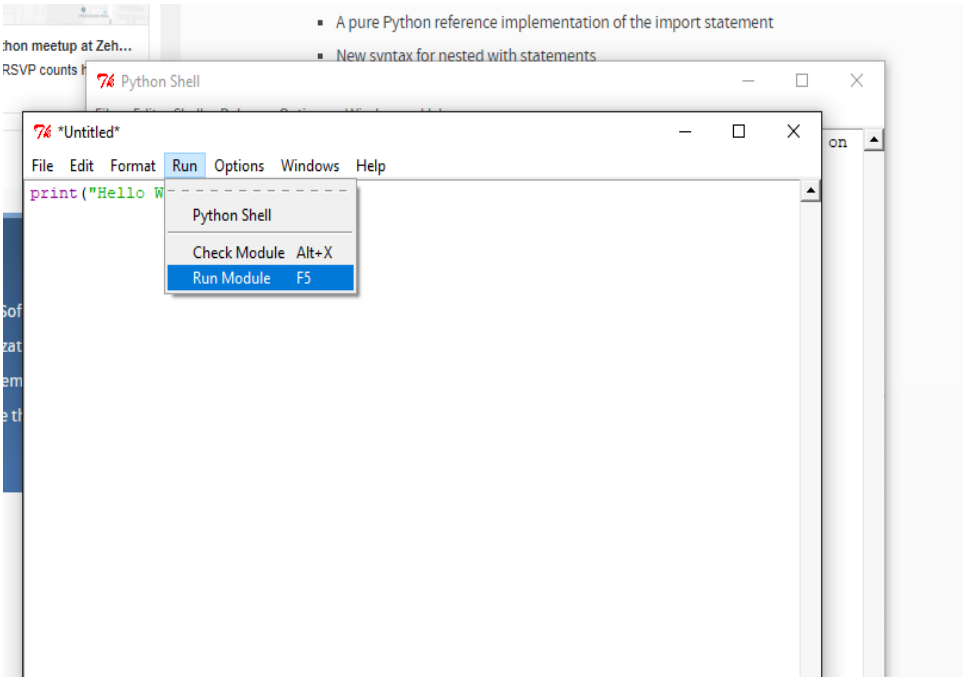


Now, both Python and IDLE (Integrated Development and Learning Environment) should be properly set up on your machine. To begin coding in Python, search and open up IDLE on your computer.

To start coding, go to File, then to New Window. Here is where you can write a python program, save, and run it. To save, you can go to File and press Save, OR you can press Ctrl-S. Save the file in an appropriate place like a coding practice folder.



After that, you'll want to go to the *Options* tab at the top and click Run Module, or just press F5. This will run the program you just wrote and return an output.



You don't always need to do this, most of this lesson can be done in IDLE without needing to make a New Window.

So, let's start coding!

Hello World!

One of the most common and well-known beginner programs, is the Hello World one. All this program will do is print the words "Hello World!". As simple as this program is, it will be used as our foundation to build off of throughout this section. So, let's begin.

It's best practice for programmers to follow three main steps: Assessing and rewriting the problem, planning it out with pseudocode, and then actually writing the code.

First: What do we need to do?

- We need to have python output back to the user the phrase Hello World

Second: How might the code look for that? Or, what's the pseudocode?

```
# This is a comment in code, also to help organize

# We need our program to print out the words, or string, hello world

print( Hello World )
```

Finally: What does the actual code look like? Lucky for us, Python was built for beauty, and is not too far off at all from the pseudocode version.

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

Hello World!
```

The **print() statement i**n Python will have the console or IDLE print out what we want it

The **" " tell Python that what we want printed is a string**. A string is just letters and words, they aren't numbers, but something meant to be read or printed out. Inside the quotes is what we want printed out, thus **the statement above prints out the string, *Hello World!***

Press Enter after typing this into IDLE and you'll be returned the statement "Hello World!" IDLE often color codes program outputs as blue to help users see what they printed.

It's a simple program, but as the complexity of our programs increase, this three-step process becomes more useful in organizing code.

Basic Arithmetic and Strings

Now, let's crunch some numbers. Arithmetic and basic math is very easy in Python. If we wanted two numbers to be added or multiplied or used in any way, we can do that in the print statement.

The following code multiplies 5 and 8, then after it prints the answer, divides 10 by 5.

```
>>> #This multiplies 5 times 8

>>> print(5 * 8)

45

>>> #This divides 10 by 2

>>> print(9 / 5)

1.8

>>> print(9 - 5)

4
```

There are a few important things to note. First, we **don't use quotes here because we aren't dealing with strings, we are dealing with numbers**. To explain the difference, the below code shows the difference between using quotes as a string, and not.

```
>>> print(5 + 8)

13

>>> print(“5 + 8”)

5 + 8
```

String Concatenation is important in Python; it is when we append a string onto another. It’s important to note this is *not* addition of strings, as in we can’t add an integer to a string only append it on.

```
>>> print(“Hello ” + “World!”)

Hello World!



>>> #This program incorrectly adds a str and int, and we are given an error

>>> print(“Hello ” + 7)

Traceback (most recent call last):

  File "<pyshell#5>", line 1, in <module>

    print("Hello" + 7)

TypeError: Can't convert 'int' object to str implicitly
```

The above error says we **cannot just add a string object and an integer together**, you can't do that. But we can add the two strings to get the above output! This is the basics of string concatenation.

If we want to **append a number at the end of the string**, we have to call the **string function, str( ),** to do so. The below example shows just that.

```
>>> print("Hello " + str(7))

Hello 7
```

This tells Python to convert the value of 7, which is an integer, to a string, allowing us to concatenate it with our string "Hello ".

There are many Arithmetic Operators in Python. Look at the table below and those are the operators we can use.

Operator	Meaning	Example
+	Addition	11 + 2 → 13
-	Subtraction	5 - 4 → 1
*	Multiplication	9 * 5 → 45
/	Division	15 / 3 → 5
%	Modulus (Remainder)	10 % 4 → 2 10 % 5 → 0
//	Quotient	5 // 3 → 1 18 // 5 → 3
**	Exponent	3 ** 5 → 243 3 ** 3 → 27

Some of these may look confusing, so let's focus on the ones you've likely never seen before, the Modulus and the Quotient.

A **Modulus returns the remainder of two numbers**. As we see above, the **remainder of 10 / 4, is two**. Thus, if we ever needed to get the remainder of one number going into another number, Modulus does just that. Another example is looking at the remainder when 5 goes into 10. Because **five can go into ten without any remainders, it returns a zero.**

The next item to note is the Quotient. **The Quotient the value of one number divided by a number and always rounds down if there is a remainder, it is "floored"**. For instance, **if we divide 18 by 5, regularly we get 3.6 . But, if we take a quotient of 18 by 5, the .6 is floored and the number is rounded down, giving us 3.** This is the basics of a quotient and can be manipulated in many ways further down the line with programming.

Exponents simple multiple by itself a given number of times. You probably already know this one, just checking. These are the main arithmetics you'll be using in Python and can be extremely useful in many cases.

Variables

Programmers can't always just put what they want directly in a print statement, many times we have to change a plethora of numbers which could be painstakingly have each put in its own print statement. Or, we could store values we want in variables. Simply put, **variables store a value to be changed or used later on**.

As you progress in programming, variables become more and more useful.

When we declare a variable, we give it a value and Python is clever enough to tell whether that value is an integer or string, or anything we want it to be for the most part.

```
>>> #We make a variable, a, equal to 5

>>> myNum = 5

>>> constantNum = 10

>>> print(constantNum + myNum)

15
```

We can also do this with strings:

```
>>> myName = “Leon”

>>> print(“My name is ” + myName)

My name is Leon
```

Remember, we can’t say myName = Leon because Python will think myName equals another variable named Leon, which is not what we want.

Variables can also do arithmetic too, though. Let’s say we have integer A and integer B, and we want to get integer C equal to both squared, then multiplied. How would we do that?

Here's what we need to do:

1. Square both A and B, which is multiplying each by itself.
2. Add then together, and that is what C will equal
3. Be sure to follow to **Order of Operations** or else we won't get the answer we want

For our Pseudocode:

A = 4

B = 3

C = Square Root Of((A^2) + (B^2))

Print(C)

The code:

```
>>> A = 4

>>> B = 3

>>> C = (A * A) + (B * B)

>>> #Above, by following the order of operations, it will multiple A time A,

>>> #then B times B, and then add the two.

>>> print("The value of C is " + str(C))

The value of C is 25
```

Booleans

A Boolean is a logic operator, meaning it has two values: true and false. We can use this in a variety of ways, especially with conditions in the next unit, but for now let's see what they look like.

Look at the example below (You'll want to open a New Window in IDLE to code this):

```
a = True        ## True must be capitalized

print(str(a))

True
```

This is all we can do for now with Booleans, but they will be explored later on in more depth.

The If-Else Statement and Conditions

The If-Else statement is used in Python to test a condition and provide an output depending on the condition. Essentially, if something is true, then do this. If not, then do this instead. It is a fairly straightforward but very useful.

Look at the example below (You'll want to open a New Window in IDLE to code this):

```
a = 6

if a < 5:

   print(a + 5)

else:

   print(a)
```

```
>>> ## This is the output in your IDLE window that should pop up

6
```

So, what did that code just do?

First, and pretty straightforward, we made a new variable a and gave it the value 6, as an integer. Next, we began our If-Else statement.

The syntax to start our If-Else statement if and then following that with a condition. This tells Python to take in a **condition** and if that condition is met, like if a is less than 5, then print something. If that condition is not met, print something else.

A **condition statement** comes in a variety of ways and can test if something is true or false, equal to, less than or greater than, and more. The condition where was **if a is less than 5, print a plus 5; else just print a.** That is the use of a *less than* condition in the if statement.

What other conditions can we do? Look at the example below (You'll want to open a New Window in IDLE to code this):

```
a = 6

## This tests if a equals 5, if a is any number other than 5 than it won't print a + 5

if a == 5:

    print(a + 5)
```

```
## if a is greater than 5

if a > 5:

    print(a)
```

```
## if a is Less Than OR Equal to ten, then print something

if a <= 10:

    print("something")
```

```
## if a is Greater Than OR Equal to ten, then print anything

if a >= 10:

    print("anything")
```

We can also test strings as well! When we test strings, Python can test either string length or the strings themselves. Example:

```
a = “hi”

B = “hi”

## This tests if a has the same string as b

if a == b:

    print(5)

else:

    print(10)
```

5

```
a = “hi”

B = “ho”

## If they aren’t EXACTLY the same, they are not equal

if a == b:

    print(5)

else:

    print(10)
```

10

We can compare string lengths as well.

```
a = “hi”

b = “his”

## This tests if a has the same string as b

if a < b:

   print(5)

else:

   print(10)
```

5

```
a = “hi”

b = “his”

## This tests which string is longer

if a < b:

   print(“b is longer”)

else:

   print(“a is longer”)
```

B is longer

Boolean Conditions:

```
## if a is true, then print something

## with Boolean variables, you CAN'T be greater than or less than true or false

if a == true:

   print("something")
```

```
## if a is false, then print despair

if a == false:

   print("despair")
```

Another condition we can use is the **not** operator, which looks like **!**. This can be used to say if something *is not equal to* something else, then do this.

Example:

Look at the example below (You'll want to open a New Window in IDLE to code this):

```
a = 6

if a != 5:

   print(a + 5)

## a is not equal to five, thus is prints out the rest of the statement

11
```

This works with Booleans too:

```
a = True

if a != True:

    print("Blasphemy!")
```

The While Loop

The While Loop is a program that runs a certain piece of code while a certain condition is met. Once that condition is met, it moves on and doesn't run the code inside the loop. For instance, we could say while a car has gas, the car moves. But when it no longer has gas, it doesn't.

Let's look at the example below:

```
# Initiate our variable

a = 1



# Initiate the while loop with our condition statement

while a < 5:

   print(a)

   a += 1     # Ensure there is a way to escape the loop
```

This is what the syntax for while loops looks like. We **initiate the while loop with while, and then give it a condition**. The code under and indented the while loop is ran while that condition is met.

It is important to note that if we do not program **a way for the while loop to stop**, you will be given an error for having an infinite (dangerous) loop. That is the reason we have a += 1. **What this does is it says a equals a plus one, essentially just adding one to a each loop through.**

Functions

Functions in Python are blocks of code that run when they are called. For instance, a function could print a string. Functions are **made with the keyword def and have parentheses after its name.** These **parentheses hold the parameters of a function**, or what the **function takes in or is given when it is called**. This can be a number or a string, there are many types of parameters functions can be given, including empty ones. A function with nothing in its parentheses simply doesn't take in any parameters.

Let's examine the syntax of the function below:

```
## First, we start our function
def my_function():
    print("This is my function.")
my_function()      #we call our function
```

```
This is my function.
```

As we see, we must **start the function with def, the state the name of the function, my_function**. We don't pass it any parameters in this function because we don't need to. Then indented and under, we have the function print something out when it is called.

After we made our function, all we need to do is **call it, this is simply done by stating my_function()** below. Again, we don't put anything in the parameters when we call it. This will have the console output to us the print statement in the function.

Let's look another where we have actual parameters to our function:

```
## This function will take in a number and multiply it by five

def multiplyBySeven(someNumber):

    return 7 * someNumber



print(multiplyBySeven(5))

35

print(7 + multiplyBySeven(7))

56
```

Our function takes in a number which is assigned to our variable we created in the parameter called someNumber. someNumber is then multiplied by 7 in the function and **returns the result.**

It's important to note we are **not printing anything in the function but returning a number**. This means when we call the function and pass it a number, the returns a number back that we could add to another number later or, as seen, printed to the console.

So, what if a program needs something passed to its parameters but we don't pass it anything?

Let's examine the syntax of the function below:

```
## We pass our parameters with pName and have it default to equal "Leon"

def my_name(pName = "Leon"):

    print("My name is " + pName)



my_name("Miles")

My name is Miles



my_name()    ## No name is passed to the function here

My name is Leon

my_name("Gary")

My name is Gary
```

This is an example of how to **make parameters have a default value if nothing is passed in it when it is called**.