FIRST EDITION – CHAPTER 1 REV 1

Kevin Thomas & Aaron Lehmann

Forward

Over the next decade we will personally witness the full Cyber and Automation revolution in a way even twenty years ago would never have been thought possible.

Python is one of the most popular programming languages and used extensively in Automation, Machine Learning, Web Development and the list goes on.

This is a no-nonsense tutorial that will get you up and running with Python from zero to hero!

This tutorial assumes you have any Python 3.5+ version installed on either Windows, MAC or Linux or you can use the FREE online Python editor **Replit** https://replit.com which is the easiest option if you are not already set up.

I want to thank Aaron Lehmann, Senior Software Engineer, who I have the pleasure of working with for his contributions which expand upon some of the basic pillars that we constructed throughout the book.

Table Of Contents

Chapter 1: Basic I/O

We are going to dive right into basic input and output.

By the end of the lesson we will have completed the following.

> * Written a 0001_hello_world.py app which will output "Hello World!"
> to the console.
>
> * Written a 0002_basic_io.py app which will demonstrate the ability for us to obtain
> keyboard input and dynamically populate logic in the console based on the user
> submission.

STEP 1: Open Your Python Editor Or replit.

STEP 2: Create File 0001_hello_world.py

STEP 3: Type Code

```
print('Hello World!')
```

STEP 4: Run File In Terminal

```
python3 0001_hello_world.py
```

STEP 5: Review Output

```
Hello World!
```

We begin our understanding of Python with the *print* function. The *print* function in Python is a *built-in* function which literally prints strings or words into the console. In order for the *print* function to be executed we need to add a pair of parenthesis () after the function name.

The words or string that goes between the parenthesis are what is called a *function argument*. In our case, we are going to pass a string surrounded by a set of single or double quotes. In this course we will be using the single quote convention primarily as it is simply a design choice.

The contents of the *print* function is nothing more than *print('Hello World!')* which in this situation will print out simply the words 'Hello World!' to the console. Whatever word or words we put inside the parenthesis will determine what gets printed to the console.

Let's dive into what a string is. A string is a string of characters or letters. Imagine if we had a bunch of little boxes on a table.

So we have our string, *Hello World!* which takes up 12 boxes.

Strangely if we count the boxes we see 14. Let's examine why.

Each box contains a letter or character which we refer to as an element in Python. There is also what we call a null terminating character '\0' and a new line character '\n' that are two additional characters inside the print function. The good news is when the Python team built Python from C, they built-in what we refer to as *default arguments* inside the *print* function so you, the developer, does not have to type them every time you want to print something to the console.

Now let's look at the boxes with all of the letters, spaces, null terminating character and new line character.

| H | e | l | l | o | | W | o | r | l | d | ! | \0 | \n |

That is exactly what is going on inside your computer's memory under the hood.

When programming we all make typos or mistakes. If you leave out a quote you will get what is referred to a *SyntaxError* as you will notice the color scheme on that line will be slightly off. This is an indicator of a syntax error which is nothing more than a syntactical error when the Python interpreter parses your line of code.

Let's look at an example:

```
print('Hello World!)
```

Notice we are missing the other quote mark after the exclamation point.

Re-run your program in the console.

```
Traceback (most recent call last):
  File "main.py"
SyntaxError: invalid syntax
```

In addition, Python will give you an error if you start a line of code that is not at the very beginning of the line as it will be an indentation error as it will say it sees an unexpected indent error.

Let's look at an example:

```
   print('Hello World!')
```

If you notice, the word print starts three spaces after it should.

Re-run your program in the console.

```
Traceback (most recent call last):
  File "main.py"
IndentationError: unexpected indent
```

We see an *IndentationError*.

Now that you have a handle on all of the steps to create and save your code we will an additional example as well to help solidify these concepts.

Let's clear out our code and rename our new file to **0002_basic_io.py** and type the following code.

```
# We introduce the concept of a variable to which
# we reserve a little box in our computer's memory
# to hold the string which we are going to type
# when prompted to provide our favorite food and
# favorite drink
favorite_food = input('What is your favorite food? ')
favorite_drink = input('What is your favorite drink? ')

# Here we use Python's built-in format method
# which is part of the string module's Formatter
# class as the following line of code will provide
# a response back based to the console based on
# our two variables which we inputted above
print('I love {0} and {1} as well!'.format(favorite_food, favorite_drink))
```

I want to introduce the concept of adding comments. We see a *#* and then everything after the *#* on a line is what we call a *comment*. These are helpful to remind us what is going on in our code.

When we start out we can use as many comments as we want. As we get more comfortable we will tend to use fewer comments as we will get a better handle of what is going on by looking at the Python code.

A variable holds a value in those little boxes like we saw earlier and we can use this to store any information we want during our app's run. The difference here is that variables can change during our app and not stay constant.

We are also introducing the concept of basic input in Python which we refer to as *input*. This is a built-in function like the *print* function that allows us to display a message in the console and then whatever we type will be then stored into the variable.

We see *favorite_food = input('What is your favorite food? ')* and all this does is display the words, What is your favorite food? and then allow us to type a string response and then it will be stored in *favorite_food*. For example if we typed *pizza* then the string *pizza* would be stored in the *favorite_food* variable.

We repeat the process for *favorite_drink* in the exact same way.

Finally we use the *print* function again and we use what we refer to as a *format method*. Notice we see *{0}* and *{1}* which are placeholders for our variables so what will happen is that if we used the word *pizza* for *favorite_food* it would replace the *{0}* with *pizza* and if we used *Pepsi* for *favorite_drink* the *{1}* would be replaced with *Pepsi*.

Run your program in the console.

```
What is your favorite food? pizza
What is your favorite drink? Pepsi
I love pizza and Pepsi as well!
```

It is now time for our first project!

Project 1 - Create a Candy Name Generator app - You are hired as a contract Python Software Developer to help Mr. Willy Wonka rattle off whatever candy title he comes up with in addition to a flavor of that candy. When the program is complete, Mr. Willy Wonka will be able to type into the console a candy title he dreams up in addition to the flavor of that candy. For example, Scrumpdiddlyumptious Strawberry.

STEP 1: Prepare Our Coding Environment

Let's clear out our code and rename our new file to **p_0001_candy_name_generator.py** and follow all of the steps we learned so far.

Give it your best shot and really spend some time on this so these concepts become stronger with you which will help you become a better Python Developer in the future.

The real learning takes place here in the projects. This is where you can look back at what you learned and try to build something brand-new all on your own.

This is the hardest and more rewarding part of programming so do not be intimidated and give it your best!

If you have spent a few hours and are stuck you can find the solution here to help you review a solution. Look for the **Part_1_Basic_IO** folder and click on **p_0001_candy_name_generator.py** in GitHub.

https://github.com/mytechnotalent/Fundamental-Python

I really admire you as you have stuck it out and made it to the end of your first step in the Python Journey! Great job!

In our next lesson we will learn about Python data types and numbers!

Chapter 2: Basic I/O

Today we are going to discuss datatypes and numbers as it relates to Python.

By the end of the lesson we will have accomplished the following.

```
* Written a 0003_calculator.py app which will output add, subtract, multiply and divide
two numbers in console.

* Written a 0004_square_footage_repl app which will take a width and height in
feet from the repl and print and display the square footage in our micro:bit display LED
matrix.

* Written a 0005_final_score app which will calculate a final score of a player and
indicate the result of a hardcoded boolean.
```

We will focus on 4 primary primitive datatypes that are built-in to Python.

```
* string
* integer
* float
* boolean
```

string

We are familiar with the concept of the string from last lesson. A string is nothing more than a string of characters.

Let's open up our Python Web Editor (full instructions were in Part 1 if you need to double-check) and type the following.

```
name = 'Kevin'
print(name)


Kevin
```

We see that our name variable properly prints the word *'Kevin'*.

Let's demonstrate that a *string* really is a string of characters. Each letter or character in a string is referred to as an *element*. Elements in Python are what we refer to as *zero-indexed* meaning that the first *element* starts at *0* not *1*.

Let's try an example:

```
name = 'Kevin'
print(name[0])
print(name[1])
print(name[2])
print(name[3])
print(name[4])

print(name[-1])

print(name[1:4])

print(name[1:])


K
e
v
i
n

n

evi

evin
```

We see that we do have 5 characters starting at *0* and ending with *4* so the first *element* is *0* and the fifth *element* is *4*. We can also see that the *-1* allows us to get the last *element*. In addition we can see *1:4* prints the 2nd, 3rd and 4th *element* (1, 2, 3) but not the 5th *element*. We see that if we do *1:* that will print the 2nd *element* and the remaining elements.

If we try print an element that is out of bounds we will get an *IndexError*.

```
name = 'Kevin'
print(name[5])


IndexError: str index out of range
```

A *string* is what we refer to as *immutable* as you can't change individual letters or characters in a string however you can change the entire string to something else if it is a variable.

Let's look at an example to illustrate.

```
# CAN'T DO
name = 'Kevin'
name[0] = 'L'

TypeError: 'str' object doesn't support item assignment

# CAN DO
name = 'Kevin'
print(name)
name = 'Levin'
print(name)


Kevin
Levin
```

We can see that we in fact can't change an individual element in a *string* but we can change the *string* to be something else completely by reassigning the variable *name*.

When you add strings together you concatenate them rather than add them. Let's see what happens when we add two strings that are numbers.

```
print('1' + '2')


12
```

We can see we get the *string '12'* which are not numbers as they are strings concatenated together.

integer

An *integer* or *int* is a whole number without decimal places.

```
print(1 + 2)


3
```

Here we see something that we would naturally expect. When we add two integers together we get another integer as shown above.

float

A *float* is a number with fractions or decimals. One thing to remember about a *float* is that if you add, multiply, subtract or divide an *integer* with a *float* the result will ALWAYS be a *float*.

```
print(10.2 + 2)


12.2
```

boolean

A *boolean* has only two values which are either *True* or *False*. Make sure you keep note that you must use a capital letter at the beginning of each word.

A *True* value means anything that is not *0* or *None* and a *False* value is *None* or *0*.

The *None* keyword is used to define a null value, or no value at all. *None* is not the same as *0*, *False*, or an *empty string*.

None is a datatype of its own (*NoneType*) and only *None* can be *None*.

```
is_happy = True
print(is_happy)

is_angry = False
print(is_angry)

score = None
print(score)


True
False
None
```

A *boolean* can be used in so many powerful way such as setting an initial condition in an app or changing conditions based on other conditions, etc.

Type Checking & Type Conversion

We can check the datatype very easily by doing the following.

```
my_string = 'Kevin'
print(type(my_string))

my_int = 42
print(type(my_int))

my_float = 77.7
print(type(my_float))
```

```
my_boolean = True
print(type(my_boolean))


<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

You can also convert datatypes by what we refer to as casting or changing one datatype to another.

```
my_int = 42
print(type(my_int))


<class 'int'>
<class 'str'>
```

Math Operations In Python

In Python we have an order of operations that are as follows.

```
Parentheses ()
Exponents **
Multiplication *
Division /
Addition +
Subtraction -
```

If you look at them together you can see we have *PEMDAS*. This is a way we can remember.

In addition *multiplication* and *division* are of equal weight and the calculation which is left-most will be prioritized. This is the same for *addition* and *subtraction*.

```
print(5 * (9 + 5) / 3 - 3)

# First: (9 + 5) = 14
# Second: 5 * 14 = 70
# Third: 70 / 3 = 23.33334
# Fourth: 23.33334 - 3 = 20.33334


20.33334
```

APP 1

Let's create our first app for the day and call it
0003_calculator.py.

```
first_number = int(input('Enter First Number: '))
second_number = int(input('Enter Second Number: '))

my_addition = first_number + second_number
my_subtraction = first_number - second_number
my_multiplication = first_number * second_number
my_division = first_number / second_number

print('Addition = {0}'.format(my_addition))
print('Subtraction = {0}'.format(my_subtraction))
print('Multiplication = {0}'.format(my_multiplication))
print('Division = {0}'.format(my_division))
print(type(my_division))


Enter First Number: 3
Enter Second Number: 3
Addition = 6
Subtraction = 0
Multiplication = 9
Division = 1.0
<class 'float'>
```

Notice when we use division in Python that the result is a float.
This will always be the case.

APP 2

Let's create our second app for the day and call it
0004_square_footage.py:

```
length = float(input('Enter length: '))
width = float(input('Enter width: '))

square_footage = length * width

print('Your room size is {0} square feet.'.format(square_footage))


Enter length: 4
Enter width: 5
Your room size is 20.0 square feet.
```

APP 3

Let's create our third app for the day and call it
0007_final_score.py:

```
player_score = int(input('Enter Player Score: '))
player_score_bonus = int(input('Enter Player Score Bonus: '))
player_has_golden_ticket = True

player_final_score = player_score + player_score_bonus

print('Player final score is {0} and has golden ticket is
{1}.'.format(player_final_score, player_has_golden_ticket))


Enter Player Score: 4
Enter Player Score Bonus: 5
Player final score is 9 and has golden ticket is True.
```

Project 2 - Create a Mad Libs app

Today we are going to create a talking mad libs app and call it
p_0002_madlibs.py:

Start out by thinking about the prior examples from today and spend
an hour or two making a logical strategy based on what you have
learned.

It will get a noun from the user and then get a verb from the user
and then finally create a madlib then print this out to the console.

Give it your best shot and really spend some time on this so these
concepts become stronger with you which will help you become a better
Python Developer in the future.

The real learning takes place here in the projects. This is where you
can look back at what you learned and try to build something brand-
new all on your own.

This is the hardest and more rewarding part of programming so do not
be intimidated and give it your best!

If you have spent a few hours and are stuck you can find the solution
here to help you review a solution. Look for the
Part_2_DataTypes_+_Numbers folder and click on **p_0002_madlibs.py** in
GitHub.

https://github.com/mytechnotalent/Fundamental-Python

I really admire you as you have stuck it out and made it to the end of your second step in the Python Journey! Great job!

In our next lesson we will learn about Python conditional logic!