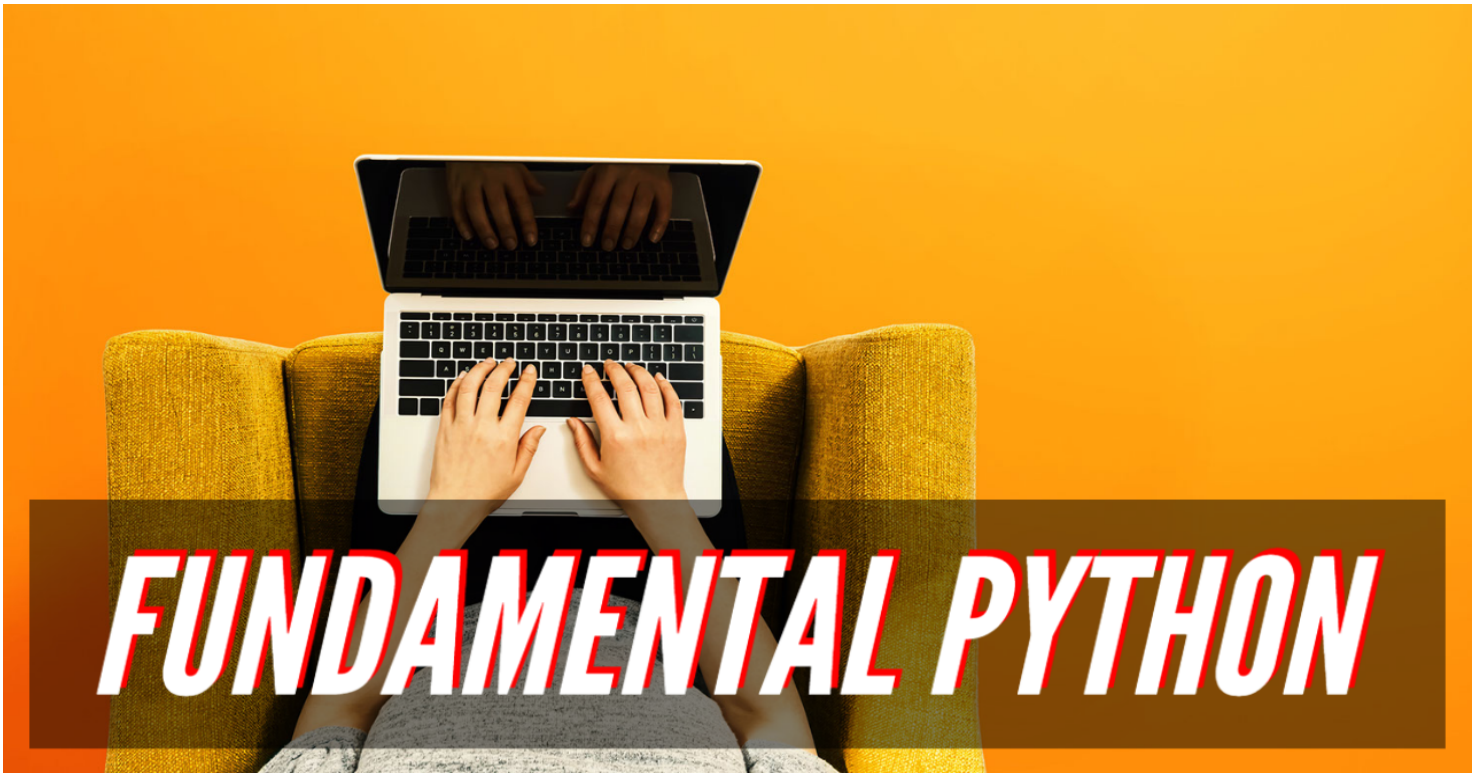


FIRST EDITION – CHAPTER 4 REV 1

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.

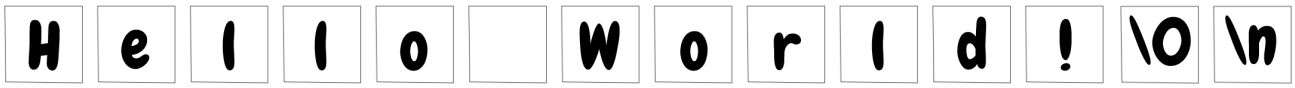
Let's dive into what a string is. A string is a string of characters or letters. Imagine if we had a bunch of little boxes on a table.

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.



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: Datatypes & Numbers

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!

Chapter 3: Conditional Logic

Today we are going to discuss Python conditional logic and application flow chart design.

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

* Written a 0006_career_councelor.py app which will ask some basic questions and suggest a potential Software Engineering career path.

* Written a 0007_heads_or_tails_game app which have us type either the A or B key to choose heads or tails and randomize a coin toss and display the result in the console.

One of the most important parts of good Software Engineering is to take a moment and think about what it is you are designing rather than just diving in and beginning to code something.

We are very early into our most amazing journey and it is very important at this stage to develop good design patterns and procedures so that we can scale our amazing creations as we develop our skills!

To design any app we should first make a flow chart. In this course we will use the FREE draw.io online app to design our projects however feel free to use pen and paper. Either will be just fine.

Here is a link to **draw.io** that we will be using.

https://app.diagrams.net

When developing an app one of the most fundamental tools that we will need is an ability to allow the user to make choices. Once a user has made a choice we then want our app to do something specific based on their selection.

Conditional Logic

Literally everything in Computer Engineering is based on conditional logic. I would like to share these two videos by Computerphile that goes over the very core of logic gates in the machine code of all microcontrollers and computers.

https://youtu.be/UvI-AMAtrvE
https://youtu.be/VPw9vPN-3ac

```
AND
---
INPUT  OUTPUT
A      B      A AND B
0      0      0
0      1      0
1      0      0
1      1      1

OR
--
INPUT  OUTPUT
A      B      A OR B
0      0      0
0      1      1
1      0      1
1      1      1

NOT
---
INPUT   OUTPUT
A      NOT A
0      1
1      0

XOR
---
INPUT  OUTPUT
A      B      A XOR B
0      0      0
0      1      1
1      0      1
1      1      0
```

In Python we have what we refer to as if/then logic or conditional logic that we can use and add to our tool box.

Here is some basic logic that will help demonstrate the point.

```
if something_a_is_true:
    do_something_specific_based_on_a
elif something_b_is_true:
    do_something_specific_based_on_b
else:
    do_something_that_is_default
```

The above is not runnable code it is referred to as pseudo code which is very important when we are trying to think about concepts.

The above has some rather long variable names which we would not necessarily have as long when we write our actual code however when

we pseudo code there are no rules and we can do what is natural for us to help us to better visualize and idea generate our process.

I want to discuss the concept of *Truthy* and *Falsey*. In conditional logic we have either *True* or *False*.

In addition to these two absolutes we have four values that when you apply conditional logic to they will be considered False.

These four conditions are *None*, *''*, *0* and *False*.

Let's review the following code to better understand.

```
my_none = None
my_empty_quotes = ''
my_zero = 0
my_false = False

if my_none:
    print('I will never print this line.')
elif my_empty_quotes:
    print('I will never print this line.')
elif my_zero:
    print('I will never print this line.')
elif my_false:
    print('I will never print this line.')
else:
    print('All of the above are falsey.')


All of the above are falsey.
```

The above is short-hand we can use in Python it is doing the exact same thing as below however it is more readable above.

```
my_none = None
my_empty_quotes = ''
my_zero = 0
my_false = False

if my_none == True:
    print('I will never print this line.')
elif my_empty_quotes == True:
    print('I will never print this line.')
elif my_zero == True:
    print('I will never print this line.')
elif my_false == True:
    print('I will never print this line.')
else:
    print('All of the above are falsey.')
```

```
All of the above are falsey.
```

We can also utilize the NOT operator as well to make the opposite of the above.

```
my_none = None
my_empty_quotes = ''
my_zero = 0
my_false = False

if not my_none:
    print('I will print this line.')
if not my_empty_quotes:
    print('I will print this line.')
if not my_zero:
    print('I will print this line.')
if not my_false:
    print('I will print this line.')
else:
    print('I will never print this line.')


I will print this line.
I will print this line.
I will print this line.
I will print this line.
```

The above is short-hand we can use in Python it is doing the exact same thing as below however it is more readable above.

```
my_none = None
my_empty_quotes = ''
my_zero = 0
my_false = False

if not my_none == True:
    print('I will print this line.')
if not my_empty_quotes == True:
    print('I will print this line.')
if not my_zero == True:
    print('I will print this line.')
if not my_false == True:
    print('I will print this line.')
else:
    print('I will never print this line.')


I will print this line.
I will print this line.
I will print this line.
I will print this line.
```

Outside of those conditions if a variable has something in it is considered *Truthy*.

```
my_empty_space = ' '
my_name = 'Kevin'
my_number = 42
my_true = True

if my_empty_space:
    print('I will print this line.')
if my_name:
    print('I will print this line.')
if my_number:
    print('I will print this line.')
if my_true:
    print('I will print this line.')
else:
    print('I will never print this line.')


I will print this line.
I will print this line.
I will print this line.
I will print this line.
```

Conversely we have the following with the NOT operator.

```
my_empty_space = ' '
my_name = 'Kevin'
my_number = 42
my_true = True

if not my_empty_space:
    print('I will never print this line.')
if not my_name:
    print('I will never print this line.')
if not my_number:
    print('I will never print this line.')
if not my_true:
    print('I will never print this line.')
else:
    print('All of the above are truthy.')


All of the above are truthy.
```

APP 1

Let's create our first app and call it **0006_career_councelor.py**:

In our last two lessons we would normally dive into direct coding however now we are going to take our next steps toward good software design and create our first flow chart!

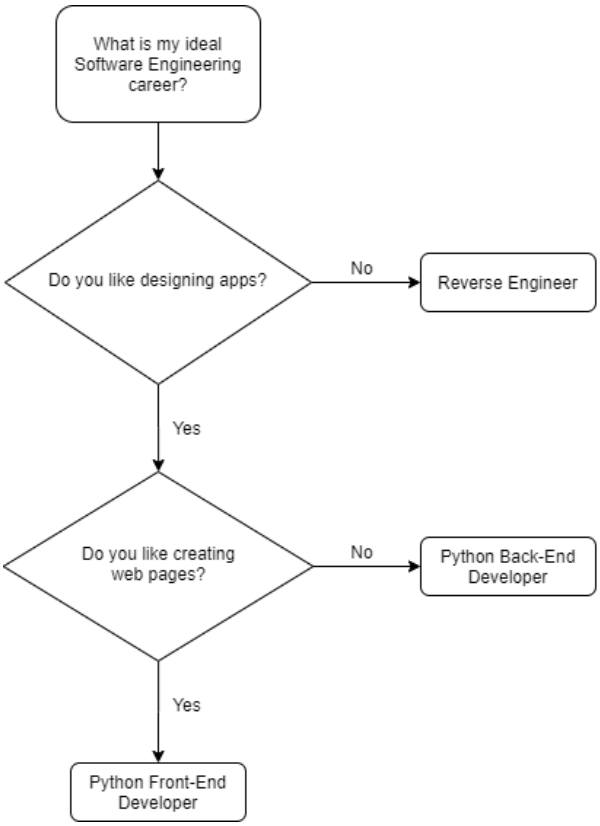
In this app we will ask two potential basic questions of the user and based on their responses we will make a suggestion on a potential Software Engineering career path to explore.

Let's open **draw.io** and start designing!

When we visit the site we see an option to create a flow chart. Let's select that option and click Create then it will prompt us to save our design. Let's call it **0006_career_councelor** and it will save the file with the **.xml** extension and load up a default pre-populated example which we will modify.

We start by defining a problem or by asking a question that we are looking to solve. In our case our design begins with a question which is 'What is my ideal Software Engineering career?'.

We are going to make this app very simple so it will not have all of the options that would be more practical however we are going to ask three basic questions and based on those answers we will suggest a Software Engineering career path.



As we can see above this is a very simple design as we only have three options but it is perfect for us to start thinking about how we might make a powerful design!

We start off with a rounded-rectangle where we define the purpose of our app. In our case we want to figure out the best Software Engineering career for us.

The diamond represents a decision that we need to make or have our app ask the user and based on the response will suggest a career option or continue to ask an additional question which based on that response will suggest one or another career path.

Now that we have a basic level design we can start coding!

```
print('What is my ideal Software Engineering career?\n')

like_designing_apps = input('Do you like designing apps? (\'y\' or \'n\'): ').lower()

if like_designing_apps == 'n':
    print('Research a Reverse Engineering career path.\n')
else:
    like_creating_web_pages = input('Do you like creating web pages? (\'y\' or \'n\'):
').lower()
    if like_creating_web_pages == 'y':
        print('Research a Python Front-End Developer career path.\n')
    else:
        print('Research a Python Back-End Developer career path.\n')
```

We first greet the user with a larger conceptual question. We then use the newline character to make a new blank line for our code to be more readable.

We then create a variable where we ask the user a question and based on the response will either suggest a career path or ask another question.

If the answer is not '*n*' then we will ask another question and if that answer is '*y*' then we make a suggestion otherwise we make another suggestion.

In this very simple example we do not check for bad responses meaning anything other than a '*y*' or '*n*' as I did not want to over complicate our early development. As we progress we will build more robust solutions that will account for accidental or improper input.

In this simple example we could have very easily designed this without the flow chart but what if our logic was more robust?

Taking the time to design a flow chart in **draw.io** or on paper is a good Software Engineering design methodology to use as you progress in your journey no matter if you want to use Software Engineering to teach, create or do it as your career.

APP 2

Let's create our second app and call it **0007_heads_or_tails_game.py**:

In this app we are going to introduce the random module. The random module allows Python to pick a random number between a series of values. In our case we are going to choose a random number between *1* and *2*.

If our user chooses ‘A’ that will be heads and if it chooses ‘B’ it will be tails.

After our app has made a choice it will then prompt the user to type either *A* or *B* to guess *heads* or *tails*.

If the user guessed the selection which the app chose they win otherwise they lost.

Now that we are familiar with the decision chart we will not have to continue to draw it out as you should be comfortable to make your own moving forward. If you are not comfortable take some time and review the lessons above.

Let's code!

```
from random import randint

random_number = randint(1, 2)

guess = input('Enter A for heads or B for tails: ')
guess = guess.lower()

if random_number == 1 and guess == 'a':
    print('You WON!')
elif random_number == 2 and guess == 'b':
    print('You WON!')
else:
    print('You Lost')
```

Here we import the *random* module. We are going to use the *randint* function where we put in a range of numbers to pick from. In our case we are asking the app to choose a random number between *1* and *2* and print some basic instructions.

We await the player to type either *A* or *B* (we use the *lower()* method to convert if necessary). Upon the selection we print either a win or lose and if they choose something other than ‘A’ or ‘a’ or ‘B’ or ‘b’ they will always lose.

Comparison Operators

In addition we can use comparison operators to check for the following.

```
>  greater than
<  less than
>= greater than or equal to
<= less than or equal to
== equal to
!= not equal to
```

Let's work with some examples.

```
number = 4

if number <= 5:
    print('Number is less than 5.\n')
elif number < 8:
    print('Number is less than 8 but not less than or equal to 5.\n')
else:
    print('Number is greater than or equal to 8.\n')
```

Let's talk through our logic. We first define a *number* as *4*. Our first conditional checks to see if the number is less than or equal to *5*. This is the case so that line will print.

```
Number is less than 5.
```

Now let's make the number 8.

```
number = 8

if number <= 5:
    print('Number is less than 5.\n')
elif number < 8:
    print('Number is less than 8 but not less than or equal to 5.\n')
else:
    print('Number is greater than or equal to 8.\n')
```

We can see that the number will fall into the else block as because it is does not fall under the first or second condition.

```
Number is greater than or equal to 8.
```

I have deliberately not discussed the *if* vs *elif* conditionals until now so that I can really illustrate a point.

When we use if and there are other if statements in the conditional it will continue to check regardless if the first condition was met. Let's look at the same example however modified to two *if* statements rather than an *if* and *elif* statement.

```
number = 4

if number <= 5:
    print('Number is less than 5.\n')
if number < 8:
    print('Number is less than 8 but not less than or equal to 5.\n')
else:
    print('Number is greater than or equal to 8.\n')
```

Here we see the first two conditions met.

```
Number is less than 5.

Number is less than 8 but not less than or equal to 5.
```

Is this what you expected? Let's think about it logically. When we say elif we mean else if rather than simply if. This means if the condition in the elif has been met, do not go any further otherwise keep checking.

Project 3 - Create a Number Guessing Game app

Now it is time for our project! This is going to be a FUN one as we are going to use the talking features we have used in our last two lessons and include all of the following features.

Let's call name it **p_0003_number_guessing_game.py**

- Create a random number generator and have the app choose a random number between *1* and *9* by creating a *random_number* variable while adding functionality within the *random* module to use the *randint* method.
- Obtain input from the user and store it in a *guess variable* and prompt the user to ask, *'Pick a number between 1 and 9.'*.
- Create conditional logic such that if you input 1 up to 9 from the *guess* variable, compare it to the *random_number* variable and if random_number is equal to guess print, *'Correct!'* otherwise print, *'The number I chose is {0}'* to display the winning *random_numbe*r and end the game.

- For the first time I want you to start thinking about input validation. I want you to come up with a solution to handle input that is first not between *1* and *9* and secondly input that is not an integer.

A Developer utilizes Google for asking questions, reaching out to a mentor/peer to discuss ideas in addition to reading the official documentation of the language they are using in addition to reviewing other code examples on the web.

I would like you to visit the Python docs and get familiar with the site.

https://docs.python.org

Here I would like you to review the Input/Output pins. You will use

If after several hours or days and you get stuck you can find my solution below. Look for the **Part_3_Conditional_Logic** folder and click on **p_0003_number_guessing_game.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 third step in the Python Journey! Today was particularly challenging! Great job!

In our next lesson we will learn about lists, dictionaries and loops!

Chapter 4: Lists, Tuples, Dictionaries & Loops

Today we are going to learn about the powerful data structures called lists and dictionaries in addition to looping logic.

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

```
* Written a 0008_rock_paper_scissors app which will take a simple list and create a FUN game where you play against the computer working with the random module.

* Written a 0009_journal app which will take a dictionary of journal entries and have the ability to add new entries.

* Written a 0010_high_score app which will you enter in a list of scores and use a for loop to find the highest score.
```

Lists

Unlike variables which store only one piece of data a list can store many items or pieces of data that have a connection with each other.

Lists are often called arrays in other languages.

Imagine you wanted to store all of the letters of the alphabet. If you had individual variables you would have to make 26 different variables. With a list you can use one variable and store an array or collection of each of the letters.

Let's look at an example of a list in the console.

```
chocolates = ['caramel', 'dark', 'milk']

print(chocolates)
print(chocolates[0])
print(chocolates[1])
print(chocolates[-1])
print(chocolates[-2])
print(chocolates[:])
print(chocolates[:-1])
print(chocolates[1:])

chocolates.append('sweet')

print(chocolates)

chocolates.remove('milk')

print(chocolates)
```

```
['caramel', 'dark', 'milk']
caramel
dark
milk
dark
['caramel', 'dark', 'milk']
['caramel', 'dark']
['dark', 'milk']
['caramel', 'dark', 'milk', 'sweet']
['caramel', 'dark', 'sweet']
```

Wow! WOOHOO! Look at all that chocolate! Let's break down exactly what is happening.

We first create list of chocolates and have 3 elements in it. Unlike strings, lists are mutable or changeable so we can keep on adding chocolate! How cool is that?

We then print the entire list of chocolates where it will print each chocolate on a separate line.

We then add a new chocolate, *'sweet'*, to the list.

We then print the updated list.

We then remove a chocolate, *'milk'*, from the list.

We then print the updated list.

In the last chapter we discussed the importance of flow charts to design our work. Going forward we will work with what we call an Application Requirements document which is a written step of TODO items.

APP 1

Let's create our first app and call it **0008_rock_paper_scissors**:

Let's create our Rock Paper Scissors Application Requirements document and call it **0010_rock_paper_scissors_ar**:

```
Rock Paper Scissors Application Requirements
-------------------------------------------
1. Define the purpose of the application.
   a. Create a game where a computer randomly chooses a number between
      0 and two and we choose a selection of either rock, paper or
      scissors.  Based on the rules either the computer or the player
      will win.
2. Define the rules of the application.
   a. Rock wins against scissors.
   b. Scissors wins against paper.
   c. Paper wins against rock.
3. Define the logical steps of the application.
   a. Create a game_choices list and populate it with the 3 str choices.
   b. Create a random number between 0 and 2 and assign that into a
      computer_choice variable.
   c. Create player_choice variable and get an input from the player
      and cast to an int with a brief message.
   d. Create conditional logic to cast the int values into strings
      for both the computer and player.
   e. Display computer choice and player choice.
   f. Create conditional logic to select a winner based on rules
      and print winner.
```

Let's create our app based on the above criteria.

```
import random

game_choices = ['Rock', 'Paper', 'Scissors']

computer_choice = random.randint(0, 2)

player_choice = int(input('What do you choose?  Type 0 for Rock, 1 for Paper, 2 for
Scissors. '))

if computer_choice == 0:
    computer_choice = 'Rock'
elif computer_choice == 1:
    computer_choice = 'Paper'
else:
    computer_choice = 'Scissors'
if player_choice == 0:
    player_choice = 'Rock'
elif player_choice == 1:
    player_choice = 'Paper'
else:
    player_choice = 'Scissors'

print('Computer chose {0} and player chose {1}.'.format(computer_choice, player_choice))

if computer_choice == 'Rock' and player_choice == 'Scissors':
  print('Computer - {0}'.format(game_choices[0]))
  print('Player - {0}'.format(game_choices[2]))
  print('Computer Wins!')
elif computer_choice == 'Scissors' and player_choice == 'Rock':
```

```
  print('Computer - {0}'.format(game_choices[2]))
  print('Player - {0}'.format(game_choices[0]))
  print('Player Wins!')
elif computer_choice == 'Scissors' and player_choice == 'Paper':
  print('Computer - {0}'.format(game_choices[2]))
  print('Player - {0}'.format(game_choices[1]))
  print('Computer Wins!')
elif computer_choice == 'Paper' and player_choice == 'Scissors':
  print('Computer - {0}'.format(game_choices[1]))
  print('Player - {0}'.format(game_choices[2]))
  print('Player Wins!')
elif computer_choice == 'Paper' and player_choice == 'Rock':
  print('Computer - {0}'.format(game_choices[1]))
  print('Player - {0}'.format(game_choices[0]))
  print('Computer Wins!')
elif computer_choice == 'Rock' and player_choice == 'Paper':
  print('Computer - {0}'.format(game_choices[0]))
  print('Player - {0}'.format(game_choices[1]))
  print('Player Wins!')
else:
  if computer_choice == 'Rock':
    print('Computer - {0}'.format(game_choices[0]))
    print('Player - {0}'.format(game_choices[0]))
  elif computer_choice == 'Paper':
    print('Computer - {0}'.format(game_choices[1]))
    print('Player - {0}'.format(game_choices[1]))
  else:
    print('Computer - {0}'.format(game_choices[2]))
    print('Player - {0}'.format(game_choices[2]))
  print('Draw!')
```

Woah! That is a lot of code! No worry! We are Pythonista's now and we shall be victorious!

We start by creating a *list* of three *str* items.

We then pick a random number between *0* and *2* and assign to the computer.

We prompt the player for a number between *0* and *2* representing the choices.

We then create conditional logic (ONE OF OUR PYTHON SUPERPOWERS) to convert all computer numbers into *str* logic.

We then print the results.

We then create more conditional logic to figure out and decide a winner.

Tuples

A Python t*uple* is just like a list but it is *immutable*. You would use this if you wanted to create a list of constants that you do not want to change.

```
RED = (255, 0, 0)
```

The elements such as *RED[0]* will be *255* but you can't reassign it.

Dictionaries

A Python dictionary is what we refer to as a key/value pair.

During one of the most amazing events I ever took part of was called the micro:bit LIVE 2020 Virtual to which I presented an educational app called the Study Buddy. The Study Buddy gave birth to the concept of an Electronic Educational Engagement Tool designed to help students learn a new classroom subject with the assistance of a micro:bit TED (Talking Educational Database) and a micro:bit TEQ (Talking Educational Quiz).

Below is a video showing the POWER of such an amazing tool!

https://youtu.be/0OG5Vfdh5bM

This concept would have never been possible without a Python dictionary!

We started out with a simple key/value pair called a *generic_ted* or *Talking Educational Database* like the following.

```
generic_ted = {
                'your name': 'My name is Mr. George.',
                'food': 'I like pizza.',
              }
```

We have a *str* key called *'your name'* and a *str* value of *'My name is Mr. George.'*, which we can retrieve a value by doing the following by adding to the code above.

```
print(generic_ted['your name'])


My name is Mr. George.
```

Dictionaries are also *mutable* which means you can add to them! Let's add to the code above.

```
generic_ted['drink'] = 'Milkshake'

print(generic_ted)


{'your name': 'My name is Mr. George.', 'drink': 'Milkshake', 'food': 'I like pizza.'}
```

We see the structure come back as a list of key/value pairs. This is an unordered datatype however we see above how we can get the value of a particular key and we see how to create a new value as well.

We can also nest lists inside of a dictionary as well. Let's add to the code above.

```
generic_ted['interests'] = ['Python', 'Hiking']

print(generic_ted)


{'your name': 'My name is Mr. George.', 'drink': 'Milkshake', 'interests': ['Python',
'Hiking'], 'food': 'I like pizza.'}
```

You can see that we do have a list as a value in the *interests* key.

We can also remove an item from a dictionary. Let's add to the code above.

```
generic_ted.pop('drink')

print(generic_ted)


{'your name': 'My name is Mr. George.', 'interests': ['Python', 'Hiking'], 'food': 'I
like pizza.'}
```

We see that the *'drink'* key and *'Milkshake'* value is now removed from our dictionary.

One thing to remember about dictionaries is that the key MUST be unique in a given dictionary.

Dicts are often used to simplify if/elif/else blocks to take fewer lines and avoid a bunch of repeated conditionals. Here’s an example that takes the user’s favorite food and prints out the right condiment for it:

```
food = input(“What is your favorite food? (‘steak’, ‘french fries’, ‘sushi’, ‘chicken’,
```

```
‘shrimp’, ‘hamburger’, ‘bratwurst’)”)

if food == “steak”:
    print(“Use Worchestershire sauce!”)
elif food == “french fries”:
    print(“Use ketchup”)
elif food == “sushi”:
    print(“Use soy sauce!”)
```

```
elif food == “chicken”:
    print(“Use barbecue sauce!”)
elif food == “shrimp”:
    print(“Use cocktail sauce!”)
elif foo == “hamburger”:
    print(“Use ketchup and mustard!”)
elif foo == “bratwurst”:
    print(“Use German mustard!”)
else:
    print(“Use some food, dude!”)
```

There’s an awful lot of lines, there, and we have to scan a bit to get from the food to the condiment! We also had to type if and print over and over. We also typed the choices twice, once for the instructions and once again amongst the tests. We can use dicts to do better. We’ll also make a couple other minor fixes, keep an eye out and see if you can see them:

```
choices = {
  'steak': 'Worchestershire sauce',
  'french fries': 'ketchup',
  'sushi': 'soy sauce',
  'chicken': 'barbecue sauce',
  'shrimp': 'cocktail sauce',
  'hamburger': 'ketchup and mustard',
  'bratwurst': 'German mustard'
}

food = input('What is your favorite food? ({}) '.format(', '.join(choices)))

if food in choices:
  print('Use {}!'.format(choices[food]))
else:
  print('Use some food, dude!')


What is your favorite food? Chicken
Use barbecue sauce!
```

Now it’s really clear what condiments go with what food. We can also get the food choices directly out of the dict, which means we’ll not have to update the prompt every time we add a favorite food. We’re able to check if the food the user picked was in the choices with the

"*in*" operator, as well. It's checking if the food is one of the keys in choices. Similarly, *join()* is joining the keys in the input prompt.

The else to handle the situation where the user types something we don't expect is a little ugly. Now is the time to see how we can have a default value, to avoid that.

```
choices = {
  'steak': 'Worchestershire sauce',
  'french fries': 'ketchup',
  'sushi': 'soy sauce',
  'chicken': 'barbecue sauce',
  'shrimp': 'cocktail sauce',
  'hamburger': 'ketchup and mustard',
  'bratwurst': 'German mustard'
}

food = input(
  'What is your favorite food? ({}) '.format(', '.join(choices.keys())))

print('Use {}!'.format(choices.get(food, 'some food, dude')))


What is your favorite food? (steak, french fries, sushi, chicken, shrimp, hamburger,
bratwurst) chicken
Use barbecue sauce!
```

Now we have used the magic of dicts to get rid of that giant string of if/elif/else statements and it's much clearer which foods take which condiments! We can also add and remove pairs much easier.

APP 1 (improved)

Remember how long the if/elif/else chain was in Rock, Paper, Scissors? Remember how we had to have integers and convert them to words? Since we can nest dicts inside dicts and since keys can be strings, we can solve those problems with dict-fu! We can replace the if/elif/else stuff with a nested dict.

```
import random

rules = {
  'Rock': {
    'Rock': 'Draw!',
    'Paper': 'Player Wins!',
    'Scissors': 'Computer Wins!'},
  'Paper': {
    'Rock': 'Computer Wins!',
    'Paper': 'Draw!',
    'Scissors': 'Player Wins!'},
```

```
  'Scissors': {
    'Rock': 'Player Wins!',
    'Paper': 'Computer Wins!',
    'Scissors': 'Draw!'}
}

game_is_running = True
while game_is_running:
  player_choice = input('What do you choose ({})? '.format(', '.join(rules.keys())))
  player_choice = player_choice.capitalize()

  if player_choice == 'Rock' or player_choice == 'Paper' or player_choice == 'Scissors':
    computer_choice = random.choice(list(rules.keys()))
  else:
    print('Please enter a valid response!')
    break

  print('Computer chose {0} and player chose {1}.'.format(computer_choice,
player_choice))
  print('Computer - {0}'.format(computer_choice))
  print('Player - {0}'.format(player_choice))
  print(rules[computer_choice][player_choice])

  game_is_running = False
```

Once again, shorter and more readable. It will also be easier to maintain, should you decide to extend it to include Lizard and Spock. Using *random.choice* (which chooses from an iterable object) helps make things more readable by removing the need for a conversion to int and back.

APP 2

Let's create our second app and call it **0009_journal**:

Let's create our Journal Application Requirements document and call it **0009_journal_ar**:

```
Journal Application Requirements
--------------------------------
1. Define the purpose of the application.
   a. Create a Journal application where we have a starting dictionary of
      entries and create the ability to add new ones.
2. Define the logical steps of the application.
   a. Create a journal dictionary and populate it with some starting values.
   b. Create a new empty dictionary called new_journal_entry.
   c. Create a new entry and date in the new_journal_entry.
   d. Append the new_journal_entry dictionary into the journal dictionary.
   e. Print the new results of the appended journal dictionary.
```

Let's create our app based on the above criteria.

```
journal = [
    {
        'entry': 'Today I started an amazing new journey with Python!',
        'date': '12/15/20',
    },
    {
        'entry': 'Today I created my first app!',
        'date': '12/16/20',
    }
]

new_journal_entry = {}

new_journal_entry['entry'] = 'Today I created my first dictionary!'
new_journal_entry['date'] = '01/15/21'
journal.append(new_journal_entry)

print(journal)


[{'date': '12/15/20', 'entry': 'Today I started an amazing new journey with Python!'},
{'date': '12/16/20', 'entry': 'Today I created my first app!'}, {'date': '01/15/21',
'entry': 'Today I created my first dictionary!'}]
```

Very cool! Let's break this down.

We first create a *journal* dictionary and populate it.

We then create a *new_journal_entry* dictionary which is an empty dictionary.

We then add values into the *new_journal_entry*.

We then append the *new_journal_entry* dictionary into the *journal* dictionary.

We then print the new appended *journal* dictionary.

Loops

This is so much fun! Now it is time to REALLY jazz up our applications! In Python we have a *for* loop which allows us to iterate over a list of items and then do something with each item and we also have the ability to take a variable and iterate over it with a range of numbers and finally we have the *while* loop which allows us to do something while a condition is true.

Woah that is a lot to process! No worries! We will take it step-by-step!

Let us start with a *for* loop that allows us to iterate over an item and then do something with each iteration. Let's go back to our YUMMY chocolates!

```
chocolates = ['caramel', 'dark', 'milk']

for chocolate in chocolates:
    print(chocolate)


caramel
dark
milk
```

WOW! WOOHOO! We do not have to use three print lines anymore only one! WOOHOO! Does a dance! LOL!

We can begin to see the POWER of such a concept as we could add other things to do for each chocolate as well!

```
chocolates = ['caramel', 'dark', 'milk']

for chocolate in chocolates:
    print(chocolate)
    print('I am eating {0} chocolate!  YUMMY!'.format(chocolate))


caramel
I am eating caramel chocolate!  YUMMY!
dark
I am eating dark chocolate!  YUMMY!
milk
I am eating milk chocolate!  YUMMY!
```

We can see that we printed the individual chocolate iteration and then we ate it! YAY!

We also have the ability to take a variable and iterate over it with a range of numbers.

```
for number in range(1, 5):
  print(number)

print()

for number in range(3, 9, 3):
  print(number)


1
2
```

```
3
4

3
6
```

Here we printed an int number variable and started from 1 and went up to but NOT including the last number *5*. We printed *1*, *2*, *3*, *4* as shown.

We can also start at another number *3* and go up to but NOT include *9* so *3* to *8* and print every 3rd number which would be *3* and *6*.

We also have the *while* loop which allows us to do a series of things while something is true.

```
has_chocolate = True

while has_chocolate:
    print('I have chocolate!')
    print('Yay I just ate it!')
    print('So yummy!')
    has_chocolate = False

print('Oh no I ate all my chocolate! :(')


I have chocolate!
Yay I just ate it!
So yummy!
Oh no I ate all my chocolate! :(
```

We can see that we start with a boolean flag called *has_chocolate* which is *True*.

We then enter the *while* loop because and ONLY because *has_chocolate* is *True*.

We then print a line, print another line and print another line and then set the flag *False* causing the while loop to break.

This won't make a good deal of sense yet until we start to get into functions where we can really see the power of such a loop!

APP 3

Let's create our third app and call it **0010_high_score**:

Let's create our High Score Application Requirements document and call it **0010_high_score_ar**:

```
High Score Application Requirements
-----------------------------------
1. Define the purpose of the application.
   a. Create a High Score application where you enter in a list of scores and use
      a for loop to find the highest score
2. Define the logical steps of the application.
   a. Create a str scores variable and input getting a list of numbers
      separated by a space and when the user presses enter it will
      split each number based on the space into a scores list.
   b. Cast the entire list from a str list into an int list using a for
      loop.
   c. Print the new int list.
   d. Create a highest_score variable and init to 0.
   e. Create a for loop where we iterate over each score and create a
      conditional to check if each score is greater than the hightest
      score and if it is then take that score value and assign it
      to the highest_score variable and keep iterating.  If we find
      another score that is bigger than the highest_score than
      assign that new score to highest_score.
   f. Print highest_score var.
```

Let's create our app based on the above criteria.

```
# We need to make sure we are entering in a str to avoid a
# TypeError: can't convert list to int
scores = input('Input each score separated by a space: ').split()

# Convert str list into an int list
for n in range(0, len(scores)):
  scores[n] = int(scores[n])

print(scores)

highest_score = 0

for score in scores:
  if score > highest_score:
    highest_score = score

print('The highest score is {0}!'.format(highest_score))


Input each score separated by a space: 1 5 2 8 3 1
[1, 5, 2, 8, 3, 1]
The highest score is 8!
```

Loops are pretty powerful, but there’s some cleanup we can do to cut down on the code and increase readability using list comprehensions.

List comprehensions are a simple loop in the definition of a list. The statement at the beginning is the definition of the element that is create each iteration. They can optionally have an “if” statement at the end to filter what goes in the list.

```
# lc = [<statement> for var in <some iterable> if <other statement>] # general form
lc = [x for x in range(5)]  # get entire range
print(lc)
lc = [x for x in range(5) if x % 2]  # get elements from the range that are not even
print(lc)
lc = [2 * x for x in range(5)]  # get entire range, multiplying each element by 2
print(lc)
lc = [2 * x for x in range(5) if x % 2]  # get elements from the range that are not even,
then multiply by 2
print(lc)


[0, 1, 2, 3, 4]
[1, 3]
[0, 2, 4, 6, 8]
[2, 6]
```

A list comprehension will let us define the list and convert all the strings to integers in one step. We can clean up the code even more by using the *max* function to find the highest value, rather than using another loop. Python likes to make your code as short as possible, without sacrificing readability!

```
# We need to make sure we are entering in a str to avoid a
# TypeError: can't convert list to int
prompt = 'Input each score separated by a space: '
scores = [int(score) for score in input(prompt).split()]

print(scores)

print('The highest score is {0}!'.format(max(scores)))


Input each score separated by a space: 1 5 2 8 3 1
[1, 5, 2, 8, 3, 1]
The highest score is 8!
```

Project 4 - Create a Caramel Chocolate Adventure Game app

Now it is time for our project! This is going to be a FUN one as we are going to create an adventure game where we use dictionaries, lists and loops to find the hidden caramel chocolate!

Let's call name it **p_0004_caramel_chocolate_adventure_game.py**

* Create a Caramel Chocolate Adventure Game Application Requirements document.
* Create a *chocolates* list and populate it with *milk*, *white*, *dark*, *caramel* and *mint* strings.
* Create an empty *rooms* dictionary.
* Create a *guesses* variable and initialize it to *2*.
* Create a *for* loop and iterate a room variable in *range(1, len(chocolates) + 1* as we have to remember that our range does not include the final number. Within the scope of the for loop create a *random_chocolate* variable and assign into it a *choice(chocolates)* to which choice is part of the random module so you have to import it and the argument is chocolates which is our list. This will randomly pick one of the 5 list items and assign into *random_chocolate*. Then we want to assign the *random_chocolate* variable into *rooms[room]* which will take the random value that the Python interpeter chose and put it into the given room iteration. If we are going through the first iteration then room value will be *1*. If we are going through the second iteration the room value will be *2*, etc. We want to make sure we do not duplicate our chocolates so before we leave an iteration we have to *chocolates.remove(random_chocolate)* so that the *random_chocolate* value is unique in future iterations of the *for* loop.
* We want to create our sequence that we have used in the past to have our Python interpreter print the following lines, *'Welcome to the Talking Caramel Chocolate Adventure Game!'. 'Select a room 1 through 5 to go through the'. 'different rooms as your goal find the room with the caramel '. 'chocolate.' 'You get two guesses.'. 'If you type and room number and if the caramel chocolate '. 'is in that room you win!'. 'Let the games begin!'.*
* Create a *game_in_on* bool and assign it *True*. Then create a while loop where *while game_is_on*, get user input and store into *room_number* variable. As extra credit you can add input validation, if you get stuck the solution provided below will demonstrate it.
* You want to check if guesses <= 1 and if so print, *'Sorry about that. Please try again.'*, and assign the *game_is_on* to *False*.
* Then check if rooms[room_number] == 'carmel' and if so print, '*You found the caramel chocolate! Great job!*' and assign *game_is_on* to *False*.
* Finally print out another conditional if the others are not met to say, *'Sorry this room has {0} chocolate.'.format(rooms[room_number])*.

This is a very tough challenge and you are putting together all of the things you have learned today. Think about running through all of the code you learned today over a few hours and then to make this final project will likely take 4-5 hours.

If after several hours or days and you get stuck you can find my solution below. Look for the **Part_4_Lists_Dictionaries_Loops** folder and click on **p_0004_talking_caramel_chocolate_adventure_game.py** and **p_0004_caramel_chocolate_adventure_game_ar** 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 fourth step in the Python Journey! Today was insanely challenging! Great job!

In our next lesson we will learn about functions!