

## Study Guide

### Intro to Python

Created By: Ioana Avram, Teaching Assistant

#### Module 1: Just the basics

##### Lesson 1.01: Course introduction

*Skills Learned From This Lesson: It's an introduction to the course, no skills here*

- Instructor is Joe Perry
- You need a computer for this course
- VM knowledge also helpful
- Know the Linux command line too, will probably use that 90% of the time, but not mandatory

##### Lesson 1.02: Background information

*Skills Learned From This Lesson: Philosophy of Python, History of Python, Design of Python*

- History of Python
- Index start from 0
- Invented by Guido van Rossum in 1991
- Version 2 of Python is ending in 2020 (end of life)
- Version 3 of Python was released in 2008. Most places and people use version 3, some places still use version 2
- Course will be in version 3 only
- PEP 8 is coding conventions
- PEP20 is the zen of python
- Python uses what is called a REPL

##### Lesson 1.1: Logic part 1(Programming basics)

*Skills Learned From This Lesson: Boolean logic, truth tables and more boolean math*

- Boolean math!
- Mostly truth tables

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- There are 3 boolean operators, AND, OR and NOT and 2 values which are true and false
- Truth tables are tables of values (true and false) and statements (using operators like and, or and not)
- A table has combinations of 2 to the power of the inputs ,so 2 inputs is  $2^{\text{inputs}} = \text{combinations}$
- AND = logical conjunction
  - Both statements/values have to be true for the whole thing to be true
- OR = logical disjunction
  - Only one statement/value needs to be true for the whole thing to be true
- NOT = logical inversion
  - Basically inverts the statement/value, if it's true it will change to false, if its false it will change to true
- 

## Lesson 1.2: Logic part 2 (Laws of Logic)

*Skills Learned From This Lesson: Boolean math, truth tables and more logic laws*

- Law of Idempotence
  - Basically  $A + A(A \text{ or } A) = A$
  - $A \cdot A(A \text{ and } A) = A$
  - Input = Output regardless of statement
  - Exception is the NOT operator which is inversion

That is to say, an equation like:  $(A \text{ OR } A)$  can be simplified to  $A$

A	A . A	A + A
1	1	1
0	0	0

- 
- Law of Association
  - Means order doesn't matter in statements
  - $(A \cdot B) \cdot C$  is the same as  $A \cdot (B \cdot C)$
  - $(A + B) + C$  is the same as  $A + (B + C)$

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

A	B	C	$(A + B) + C$	$A + (B + C)$
1	1	1	1	1
1	1	0	1	1
1	0	1	1	1
1	0	0	1	1
0	1	1	1	1
0	1	0	1	1
0	0	1	1	1
0	0	0	0	0

(truth table for law of association)

- Law of Commutation

- Order also doesn't matter here if you're doing a single operation
- $A \cdot B$  is the same as  $B \cdot A$
- $A + B$  is the same as  $B + A$

A	B	$A \cdot B$	$B \cdot A$
1	1	1	1
1	0	0	0
0	1	0	0
0	0	0	0

- Law of Distribution

- $A \cdot (B + C)$  is basically  $(A \cdot B) + (A \cdot C)$
- $A + (B \cdot C)$  is  $(A + B) \cdot (A + C)$
- Here's the truth table from the video

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

# CYBRARY

A	B	C	$A \cdot (B + C)$	$(A \cdot B) + (A \cdot C)$
1	1	1	1	1
1	1	0	1	1
1	0	1	1	1
1	0	0	0	0
0	1	1	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	0	0

(truth table for law of distribution)

- DeMorgan's Law

- Inverse of  $A \cdot B$  is the same as the inverse of  $A +$  inverse of  $B$

A	B	$\neg(A \cdot B)$	$\neg A + \neg B$	$\neg(A + B)$	$\neg A \cdot \neg B$
1	1	0	0	0	0
1	0	1	1	0	0
0	1	1	1	0	0
0	0	1	1	1	1

- New Operation: XOR
- XOR is exclusive OR or logical exclusion
- Usually OR means 1 or both values have to be true for the whole statement to be true
- But with XOR it has to be only **one** value. If both values are true in an XOR statement then the statement is false. If both are false, the statement is false. If only one value is true, the statement is true.

A	B	$A \text{ xor } B$	$(A + B) \cdot \neg(A \cdot B)$
1	1	0	0
1	0	1	1
0	1	1	1
0	0	0	0

(truth table for XOR)

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

## Lesson 1.3: Variables

*Skills Learned From This Lesson: Variable concepts, numbers, strings, other structures*

- Variables are just things you place other things in.
- You have data and you get data from programs, maybe other users etc
- How do you hold on to the data so you can pass it on? Or analyse it, or process it or anything like that?
- With variables!
- Therefore variables store data.
- It's like math where you have to solve for X.. We don't know what X is, but X holds the data that we need to solve for. Therefore X is a variable.
- Number variables
  - Basically just numbers. Variables can hold all types of numbers
  - There's different types of number variables. Int, float, double (which is ...float) and so on.
- Strings
  - Strings are character 'A' or can be symbols or can be whole sentences or paragraphs. They can be numbers too, but won't be treated as numbers.
  - Mostly printable characters.
  - All strings are enclosed with quotes. Double quotes or single quotes. Just don't use both at the same time, it confuses it.
- Structures
  - Structures are basically anything really. Abstract types
  - They can be arrays, objects, dictionaries, classes etc etc
  - Most of the times they're user defined. Except stuff like arrays and dictionaries.

## Lesson 1.4: If statements

*Skills Learned From This Lesson: flow control, if statements concept, if statement function*

- Flow control
  - Flow control is basically decisions made based on what values they have.
  - They're flowcharts basically. For code. Code flowcharts.
- IF statement examples

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

## IF Examples

- IF it is raining, wear a raincoat
  - IF the subway is late, yell at the sign
  - IF the number X is divisible by 3, print "YES" to the screen
- - Elif
    - Elif or else if is the next part of the if statement
    - Basically if the first statement is false an elif statement will run.
    - You can have as many elifs as you want. Don't have too many or the code won't be as readable.
  - Else
    - Else goes at the end of an if statement
    - It is the default option
    - So basically if this do this, else if this do this
    - After all options are done and all are false, else is the last option
    - Else do this
  - 
  -

### Lesson 1.5: For loop

*Skills Learned From This Lesson: Loop concepts, flow control, for loop function*

- A loop is taking a number of instructions or actions, set a condition that must be met and then repeat the sequence of actions until the condition is met.
- A for loop is just one type of loop
- Loops are usually used to loop through arrays, count to a specific number, apply logic to all members of a set or array etc etc
- For loops:
  - for (int i = 0; i<= 3; i++) would be how you write a for loop in most languages
  - Basically you start with the keyword for
  - Set a variable to 0 for example as a starting point

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- Set an end condition, for example the variable has to be small, larger or equal to a final value.
- Then you increment or decrement that variable until it hits the specified condition or final value.
- And while that's happening you can make it do any number of actions. For example print something a number of times, print all the options in a list, all the items in an array etc

•

## Lesson 1.6: While loop

*Skills Learned From This Lesson: while loops, relationships between while loops and other statements*

- While loops are like for loops
- But they are evaluated against if statements and when the condition is no longer true while loops become false and stop.
- While loops are ongoing loops that are evaluated based on ongoing conditions and loops. Also used for infinite loops.
- Example of how while loops work

### WHILE LOOPS

- While x < 10: print x, then increment x
1. X = 0
  2. X < 10? (TRUE)
  3. Print x
  4. Increment X
  5. REPEAT 1-4 until X == 10
  6. X = 10
  7. X < 10? (FALSE)
  8. End execution

•

- Another sample while loop

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

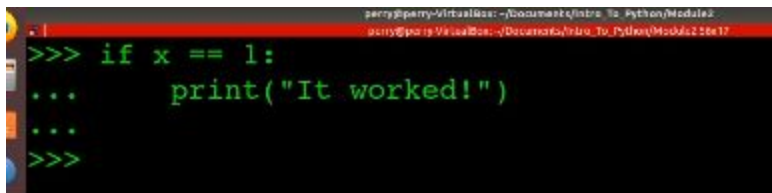
---

- N = 100
- While N > 10: Print N, then Decrement N
- How many numbers will be printed?
  - 90 (11-100, backward)

## Lesson 1.7: Logic to Pseudocode part 1

*Skills Learned From This Lesson: Python syntax and the command line, python scripts and help*

- If you type python in the command line terminal it will give you an interpreter shell
- You can type commands here (but not full scripts)
- But as Joe says if you type python, depending on what python you have installed, it might give you python 2 instead of 3. So, be careful with that. To get specifically to the python 3 shell type python3.
- You could probably write a small script in the python interpreter, with if statements and all that. But it will just be there, in the interpreter, it won't save or anything. And it probably won't run more than once.
- A good thing about the python interpreter is that you can test out any command you want, or write code snippets if you need to see if your code is written correctly for example.
- Python whitespace:
  - Python is delineated by whitespace indentation.
  - As in it uses whitespace to know which code belongs to which code.
  - You can use both tabs and the space bar to add whitespace. But you must be consistent. Using both tabs and spaces will confuse Python and break whatever script or program you're trying to write. And that is because Python sees tabs and spaces as different.
- An example of Python code from Joe's video



```
percy@percy-virtualbox:~/Documents/Intro_To_Python/Module2
percy@percy-virtualbox:~/Documents/Intro_To_Python/Module2$
>>> if x == 1:
...     print("It worked!")
...
>>>
```

- The Python 3 interpreter

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.



---

# CYBRARY

---

```
per@per-virtualbox: ~/Documents/intro_to_python/modules
per@per-virtualbox: ~/Documents/intro_to_python/modules$
>python3
Python 3.6.5 (default, Apr 1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license"
for more information.
>>>
```

## Lesson 1.8: Logic to Pseudocode part 2

*Skills Learned From This Lesson: Python and the command line, writing python scripts*

- In this lesson you take the small hello world program written in the python interpreter and put it in an actual document.
- The document is called a Python script and it has the extension `.py`.
- Joe talks about the python shebang

```
#!/usr/bin/python3
```

- This is usually at the start of a python script. It's not mandatory but its usually there to allow for a python script to be executable.
- Joe also talks about the vim editor.
- Writing a python script is not like writing code in the interpreter. Because after all it's just a doc, a file. The code doesn't run until you execute the `.py` file.
- Here's the script in vim

```
#!/usr/bin/python3
print("Hello, World!")
```

- To run the new python script, depends on whether you included the shebang line.
- With the line, you can make the file executable with `chmod`, `chmod 755` or `777` respectively and then run it like so, `./pythonscript.py`.

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

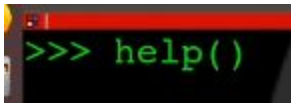
---

- Without the line you will have to write python3 (or just python, if you want 2 instead) and then the name of the script.

## Lesson 1.9: Logic to Pseudocode part 3

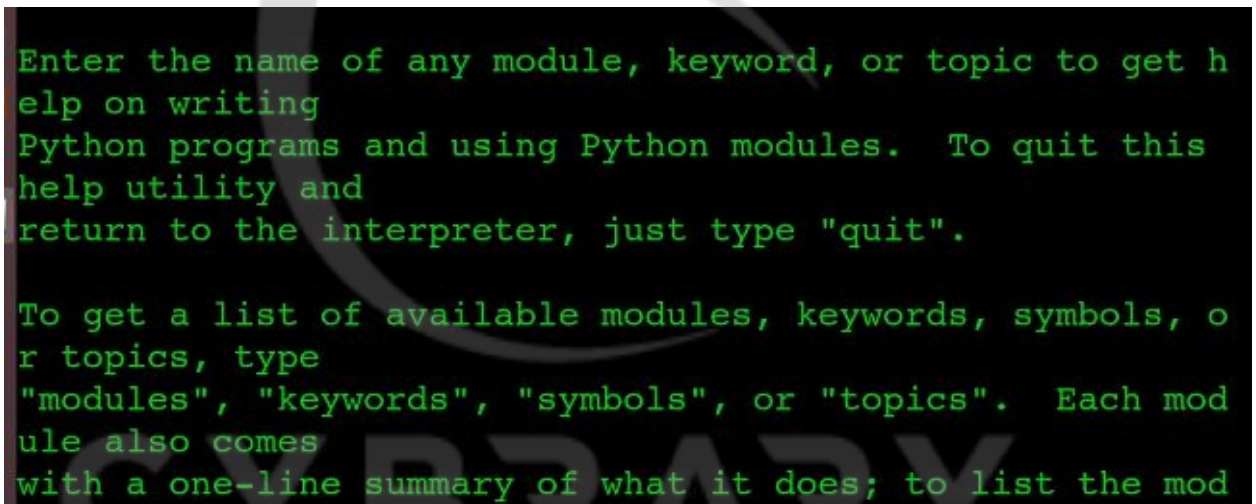
*Skills Learned From This Lesson: Python and the command line, python help and documentation*

- This is the python help function



```
>>> help()
```

- This command opens up the help module for python.



```
Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type "modules", "keywords", "symbols", or "topics". Each module also comes with a one-line summary of what it does; to list the mod
```

- Like so.
- This is the help page for the print function

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

# CYBRARY

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout,
          flush=False)

Prints the values to a stream, or to sys.stdout by default.
```

- The help function is basically like typing man in linux, if you wanna find out about a command or program. It is just the Python manual.
- Now, onto the dir function. Again same as the dir/lis commands. But in Python dir deals with namespaces.
- A namespace is the current environment where you're executing code.
- Dir without any commands

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__',
 '__name__', '__package__', '__spec__']
>>>
```

- If you make a variable and assign it something, it will show up in the dir function.
- If you look up that variable you just made, it will pull up all the functions for that particular type. For example, if it's a number, it will show all the number functions, if it's a string, all the string functions. And so on, and so on.
- And lastly, the python documentation itself



## Lesson 1.10: Python basics part 1

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

*Skills Learned From This Lesson: Python data types, strings and numbers*

- “Variables are tupperware for you brain” --Joe Perry
- This lesson will be on 4 python data types.
- Strings
  - An example of a string variable
  - ```
>>> hw = "Hello, World!"
```
  - You can print the string with just the variable for example, `print (hw)` ;
  - Or you can print the string directly like `print ("Hello world")` ;
  - Joe recommends having the string in a variable and printing the variable. In Python its relatively okay to print the string directly.
  - But generally one should have it in a variable for better security in languages like SQL and the like. It's called parameterized input, but that's for another course. This is an intro to Python course.
- Numbers:
  - There's different types of number variables in programming
    - Int
    - Float/Double
    - Boolean (bool)
    - Long int
    - Short int
    - Byte
    - Chars
    - etc..etc
  - But in Python we don't have these types exactly. We just have plain numbers.
  - Number variables work the same way as string variables. But string variables have quotes around them and numbers don't. If a number has quotes around it, it is classified as a string not a number.
  - An example of number variables
  - ```
>>> y = 10
>>> x = 1524
```
  - With division, Python assigns a float automatically to the result.

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- Like so

```
>>> w = 14/2
>>> w
7.0
>>>
```

- You can do maths in Python
- One example of Python math operations

```
>>> 7**13
96889010407
>>>
```

- You can do all the math operations in Python. And store the results in a variable.

- Like so

```
>>> 5 - 3
2
>>> k = 4*8
>>>
```

- If you print that variable, you will get the result of that operation

## Lesson 1.11: Python basics part 2

*Skills Learned From This Lesson: Python data types, dictionaries and lists*

- Lists:
  - Lists are basically array structures. Array lists, if you will.
  - The way you assign values to a list, is either the square brackets like you would declare arrays in some languages
  - Or by using the list() command.
  - Example of a list

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

```
>>> l1 = [1,23,24,5,2,1,9]
>>> █
```

- 
- To see the contents of the list variable, either type the variable name or print it

```
>>> l1
[1, 23, 24, 5, 2, 1, 9]
>>> print(l1)
[1, 23, 24, 5, 2, 1, 9]
>>>
```

- Like so
  - In a script, to output the contents of a list variable, you need to use the print command, but in the interpreter you don't have to.
  - You can do a bunch of operations on lists. You can use them with for loops, make other lists and combine them into one, make lists inside lists inside lists (*listception!*) and so on.
  - Because you can combine lists with other lists, they can contain multiple types of data. You don't have to have just 1 data type, like in traditional array variables.
- Dictionaries
    - Example of dictionary in python

```
>>> dict_1 = {}
>>> dict_1 = {"A": "APPLE", "B": "BANANA", "C": "CHERRY"}
>>> █
```

- 
- In lists, you access specific items by using their index values
- In dictionaries you access specific values by using their keys.

```
>>> dict_1["A"]
'APPLE'
>>>
```

- Like so
- If you enter a key not in the dictionary (or list) you'll get an error

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

```
>>> dict_1["TEST"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'TEST'
>>>
```

## Lesson 1.12: Summary and review

*Skills Learned From This Lesson: All of the above*

- A lot of things were learned
- Python history, design, function, coding conventions etc
- Logic, a lot of logic, stuff like all the values and operators
- Truth tables, laws of logic and so on.
- Flow control and conditional statements like if statements
- Loops, for and while
- Python and the command line
- Python variables and data types
- Python syntax and how to make a python script

## **Module 2: Datatypes and Logic(and programming)**

### Lesson 2.1: Datatypes and Logic Intro

*Skills Learned From This Lesson: No skill because intro*

- Mostly programming, less theory
- Go over the data types and logic, but in Python code, not so much in the actual truth table logic. (That was covered in module 1)
- All the conditional statements, loops and datatypes will be covered in more detail than they have in the previous module.
- A lot of Python logic explained and implemented.
- Instructor is still Joe Perry

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

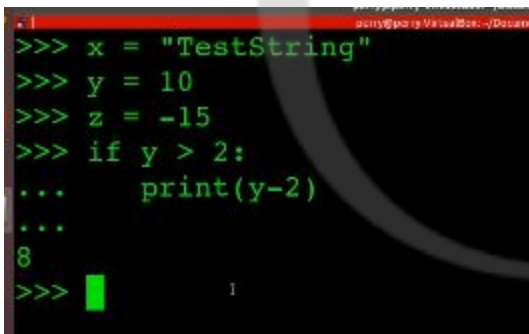
# CYBRARY

---

## Lesson 2.2: If/Elif/else

*Skills Learned From This Lesson: Apply logic in Python code, use and structure of If statement in Python code*

- Starting from your choice terminal we open python3 to get our interpreter
- With if statements as you know, we're making a decision. For example:
- If this action is true, then do this action. If not move on to the next action. If no action in the list is true use the default action. Maybe print a message or something.
- You have an initial value in a variable. How you got the value doesn't really matter. It could be coded in, taken as input, taken from another program or part of code.
- We can use that initial value or values to run it through an if statement. That will run a series of decisions and return a result. What decisions and how many is really up to you.
- Here is an example of an if statement in Python

A screenshot of a terminal window with a black background and green text. The code being executed is: >>> x = "TestString", >>> y = 10, >>> z = -15, >>> if y > 2:, ... print(y-2), ... The output shown is 8. The prompt >>> is visible at the end of the line.

- We have the variables x y and z with values in them.
- Now we can use any one of them (or even multiple at the same time), perform a check and then an action based on the result of that check. If it returns true do an action, if its false do a different action.
- Here we're saying if the variable y is greater than 2, print y-2. That prints 8.
- But what if the variable y is less than 2 (or any other variable defined)?
- We used the elif keyword. Elif is the Python else if. We will also use the variable z for comparison.
- So all we have to do is change the 2 for the variable z.
- Then write elif y is less than z (after you changed the y > 2 statement).
- If y is less than z we will just print the z variable, like so

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.



---

# CYBRARY

---

```
>>> if y > z:
...     print(y)
... elif z > y:
...     print(z)
... █
```

- (If statement with an elif condition)
- And at the end we print an else to end the if statement. The else condition like it was said earlier is just the default or last check. If all the checks for a variable or number of variables return false the if statement will default to the else condition.
- Like so

```
... elif z > y:
...     print(z)
... else:
...     print("EQUAL") █
```

- 
- For extra practice there is an if statement document in the supplementals
- One last example of if statements in a script document

```
#!/usr/bin/python3
#evaluate two strings
s1 = "Test."
s2 = "Test2."

if s1 == s2:
    print(s1)
else:
    print(s1+s2)
```

## Lesson 2.3: For/Else

*Skills Learned From This Lesson: writing for loops in Python code, relationship between for and else in Python*

- This the start of the loops. The following lessons will cover loops such as for loops and while loops in more depth and how you can apply them to Python code.
- Start with writing a small if statement in your terminal of choice and the python interpreter.

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- This is one example of what a for loop looks like in Python

```
>>> for i in range(10):  
...     print(i)
```

- 
- Small note: the range option is non inclusive. It won't print to 10 if you specify 10, it will print to 9.
- With range Python assumes you want 10 numbers, not numbers 0-10. Hence why it only prints to 9 and not 10. If you say to 11, it will print to 10 but not 11. And so on.
- The output of that example

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
>>>
```

- Like we said above, it only prints to 9 because technically 0-9 is 10 numbers and you specified 10. The *i* variable usually starts at 0, but you can set it to whatever value you, for example 1, or any other number. And 10 is the condition that must be met, there for as long as *i* hasnt passed 10 numbers (0-9 for example) it will continue to increase and print. When it hits 10 numbers it stops because the logic is as long as *i* is less than 10, keep printing. If *i* reaches 10 stop printing(however it wont print 10 unless you set the variable condition to be equal to 10).

```
>>> for i<10: print i
```

- 
- Example 2 of for loops

```
>>> l1 = ['a','b','c','d','e','f','g']  
>>> for item in l1:  
...     print(item)
```

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- The variable is an array containing string characters. What the for loop will do is iterate through that array and print out the items. This type of for loop is actually quite common for iterating menu options and that sort of thing. As output you will get the actual items in the array, the strings.

```
...
a
b
c
d
e
f
g
>>>
```

- Like so.
- Now we move onto using the else statement in a for loop. For that we need to know the concept of a break statement.
- A break statement is used in loops pretty often.
- What it does is if a certain condition is met or not met as it were, the break statement will execute and the loop will end, not regarding what stage the loop is at. Break statements as the name says, break the loop.
- Example 3 of for loops

```
>>> ll
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> for item in ll:
...     if item is 'd':
...         print("Found it.")
...
Found it.
>>>
```

- You can also use the == sign and it will give you the same result as if you wrote if item is 'd', for example.

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- Now we introduce break as explained

```
zerry@zerry:~/virtualBox/~/Documents/intro_to_python/modules$  
>>> ll  
['a', 'b', 'c', 'd', 'e', 'f', 'g']  
>>> for item in ll:  
...     if item is 'c':  
...         print(item)  
...         break  
...  
... █
```

- Break is useful because if you have long lists and arrays for thousands or millions of items and you just want a few specific ones, using break will immediately end the loops once you found what you were looking for instead of going through the whole or array, potentially saving a lot of time and money.
- After learning about the break statements we can add in an else to our for loop.
- Here's the example

```
>>> for item in ll:  
...     if item is 'z':  
...         print(item)  
...         break  
...     else:  
...         print("No zed found.") █
```

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

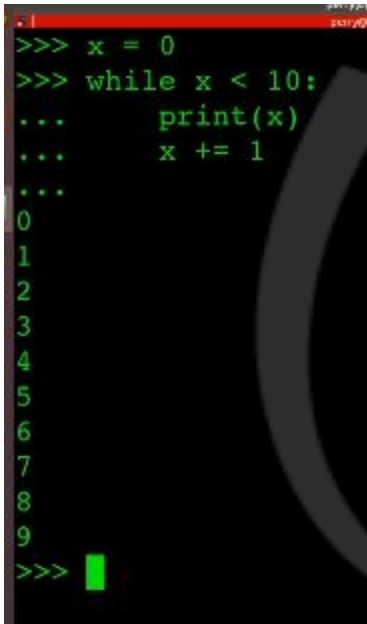
# CYBRARY

---

## Lesson 2.4: While in Python (While Loops)

### Skills Learned From This Lesson: While loops in Python

- While loops are another type of loops you can have
- Here's an example



```
>>> x = 0
>>> while x < 10:
...     print(x)
...     x += 1
...
0
1
2
3
4
5
6
7
8
9
>>>
```

- It is important to note that you must have the increment at the end of the loop, after the print statement.
- If you don't have it, the loop is never going to end. Because a while loop works like, while this variable is in this condition keep doing the action. So for example while the variable is less than 10, keep printing numbers and incrementing.
- When the variable reaches 10, stop printing numbers. But as long as the variable is under 10 the loop will continue to print numbers forever.
- Example 2 of a while loop:

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

```
>>> while x < 1:  
...     ll.pop()  
...     if len(ll) == 0:  
...         break  
...  
5  
4  
3  
2  
1  
>>>
```

- (Example 2 of a while loop)

## Lesson 2.5: Logical escapes in Python loops

*Skills Learned From This Lesson: Break, continue and pass keywords used in loops*

- Break is used in loops like for and while
- Another example of a loop with a break statement

```
>>> ll  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
>>> for i in ll:  
...     if i % 3 == 0:  
...         print(i)  
...         break
```

- If you just want numbers that don't give a remainder, add the break statement.
- That will give you 3. 3 is cleanly divisible by 3 and leaves no remainder.
- If you don't add the break statement you'll just get what is divisible by 3. (3, 6 and 9)
- The continue keyword can also be used to do the same thing as not having the break keyword in.

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- Continue example:

```
>>> ll
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> for i in ll:
...     if i % 3 != 0:
...         continue
...     else:
...         print(i)
```

- Continue is similar to break, it skips the loop if you can't find the variable or value the loop is looking for. It doesn't end the loop like break does. It just skips it over and moves on to the rest of the code, so that the application keeps running and doesn't stop because break said to.

- Pass:

```
>>> for i in ll:
...     if i % 3 == 0:
...         pass
...     print(i)
```

- Pass just skips the if statement. So there is an if statement in place, but if you add pass the code before it does not execute. Useful for stand-in code, if you don't have it functioning or done yet and don't want to execute that code just yet.
- So, we get this.

```
...
1
2
3
4
5
6
7
8
9
10
```

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- Instead of the regular 3,6,9 we would usually get.

## Lesson 2.6: Python Functions

*Skills Learned From This Lesson: writing for loops in Python code, relationship between for and else in Python*

- Functions are reusable bits of code that you can call whenever you want to do any sort of operations on any sort of input. They're isolated blocks of code that can be used anywhere and called from anywhere, even from other files inside your file.
- Some explanations of a sample function in Python

```
#!/usr/bin/python3

def func_1(a,b):
    c = a + b
    d = a - b
    e = c ** d
    return c

#def - Function Definition Keyword
#func_1 - Function Name
#(...) - Indicator of function
#a,b - Argument
#return - Indicator of end of function
#return c - provide the value of c to the caller
```

- This part below is where you call your functions with your arguments

```
def main():
    #return_val = func_1(1,3)
    #return_val1 = func_1(2,4)

main()
```

- This will run the functions and print to the output.

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.



---

# CYBRARY

---

## Lesson 2.7: Strings in Python

*Skills Learned From This Lesson: string methods, string slicing, user input*

- String in Python in more detail.
- We can use dir to look at string functions and variables, like we discussed earlier on.
- These will be specifically string functions, functions and methods that deal with strings.
- Methods are functions that are attached to an object
- Example of the upper function which capitalises a string

```
>>> x
'Hello, World!'
>>> x.upper()
'HELLO, WORLD!'
>>>
```

- The lower function is the opposite. It makes a string all lower case.

```
>>> name_list = ['joe', 'jimmy', 'bob', 'tim']
>>> input_variable = 'Joe'
>>> for i in name_list:
...     if i == input_variable.lower():
...         print('found it')
...
found it
>>>
```

- Here's an example
- You can use it if you have a list of names or items and you're looking for a specific one
- **String slicing:**
- An example:

```
>>> x
'Hello, World!'
>>> print(x[0])
H
>>>
```

- Here we used the index 0.
- The index 0 will give us the first letter. Any other index after that will give us the subsequent letter/character. For example print(x[7]) will print the letter W.

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- W is the 8th character.
- You can also print out not only single characters/letters but a specific set or chunk.
- For example print (x[0:4]) will print the first 4 characters but not the 5th (the one at position 4).
- But you don't have to start at 0. You can start at any index you want.
- For example print (x[3:4]) prints out the letter L, in Hello.
- This wont work however

```
>>> print(x[5:3])
```

- 
- To print backwards you have to add an extra step, which will allow it to print backwards.
- Like so

```
>>> print(x[5:3:-1])
```

- 
- This prints out the whole string backwards

```
>>> print(x[::-1])
!dlrow ,olleH
>>>
```

- 
- You can use different amounts to step and give different letters
- For example this gives every second letter of the string

```
>>> print(x[::2])
Hlo ol!
>>>
```

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- 1 more example

```
>>> print(x[:3])
Hl r!
>>> print(x[::-3])
!r lH
>>> █
```

## Lesson 2.8: Strings in Python, Continued

*Skills Learned From This Lesson: string methods, user input, print statements*

- To take input from the user, you use the input() function.

```
>>> x = input()
█
```

- Like so
- Make sure you add a message in the brackets so you know when you have to provide input, otherwise you'll just be given a blank screen to type in.

```
"Hello, World!"
>>> x
!Hello, World!"
>>> █
```

- Here's an example of string input
- Notice the extra quotes around the message. The input function provides the quotes because usually an input from the user will be in the form of a string.
- Or so the Python interpreter assumes when you provide input.
- It is up to you to change the input to another type once you store it, so that it may further be usable.

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- Example of input function in a script

```
#!/usr/bin/python3

def take_input():
    x = input("Input text> ")
    return x

take_input()
```

- Here it takes input and prints it to the console

```
./io.py
Input text> HERE WE GO
HERE WE GO
```

- For even better looking input we can use the format function, or f strings (which are format strings)
- Example:

```
>>> s1 = "You said {}."
>>> s1
'You said {}.'
>>> s1.format(input())
```

- And here's the output:

```
test
'You said test.'
```

- For extra format strings you just give the input function multiple times, so it will take multiple inputs.

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- Like so

```
>>> s2 = "You said {} and {}."
>>> s2.format(input(),input())
test
test again
'You said test and test again.'
>>> █
```

- The code in a script

```
#print "You entered: <USER TEXT>"
prompted = "You entered: {}"
str_var = take_input()
print prompted.format(str_var) █
```

- Will give the same output as we've seen.

## Lesson 2.9: Numbers in Python

*Skills Learned From This Lesson: math operations, difference between an int and a float*

- Numbers in Python, in more detail.
- Math operations in Python and any other language are pretty much the same as in Math usually.
- You can add, subtract, multiply and divide like in any other place.
- + and - to add and subtract
- / to divide and \* to multiply.
- But in Python (and probably other languages) when you divide a number by another number the result is a float as opposed to an int.
- A float like you may know is a decimal point number.
- An int is a number without a decimal point.
- If you want to divide 2 numbers and don't want the decimal point you do a thing called floor division

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

```
>>> 20 // 5
4
>>> █
```

- 
- Floor division is basically dividing the number and rounding it down.
- Like the `Math.floor()` in Java and other languages.
- You can also use the modulo operation, which gives you the remainder of that operation

```
>>> 20 % 5
0
>>> 21 % 5
1
>>>
```

- Useful for checking odd and even numbers
- Even numbers give no remainder and odd numbers give 1 remainder.
- There are also exponent operations

```
>>> 4 ** 4
256
>>>
```

- Like 4 to the power of 4, which gives 256.
- There's 1 more thing you can do with numbers
- The underscore
- This will give you the last number you used in an operation
- For example if you just add `5 + 3` using the underscore after the operation will give you the result of the last operation

```
>>> 4 ** 4
256
>>> _
256
>>> 5 + 3
8
>>> _
8
>>> █
```

- Like so. But don't use underscore as a variable in itself, because it'll overwrite the default underscore and you won't be able to use it again, until you change or reset it.

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- The underscore can also be used in loops

```
>>> for _ in range(10):  
...     print("hello")  
...  
hello
```

- Here the underscore will print out all the numbers, or all the times a message needs to be printed
- The word hello will be printed 10 times because the loop will loop 10 times.

## Lesson 2.10: Lists in Python

*Skills Learned From This Lesson: list operations and methods, list slicing*

- Lists in Python, in more detail
- Here's our list

```
>>> l1 = [1,2,3,4,5,6,7,8,9,10]  
>>>
```

- Use dir() to look at what methods and functions we can use on lists
- First method for lists: Append

```
Help on built-in function append:  
  
append(...) method of builtins.list instance  
    L.append(object) -> None -- append object to end  
(END)
```

- You can as it says, use the append method to add extra items to the end of a list
- This will append the number 11 to the end of the list

```
>>> l1.append(11)
```

- It has now been added

```
>>> l1  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]  
>>>
```

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- Clear will empty the list array, self explanatory.
- Next method: Pop
- If you're familiar with data algorithms, particularly the stack, the pop method will be familiar. Pop removes the last item in a list array.
- Conversely push adds an item to the array (but unlike append it adds it at the top).
- Unfortunately push is not a method in the list array methods. But you can use append in the meantime.
- Next method: Reverse
- Reverse just reverses the items in the list, it doesn't print anything or create any additional lists.
- Last method: Sort

```
Help on built-in function sort:

sort(...) method of builtins.list instance
  L.sort(key=None, reverse=False) -> None -- stable so
  rt *IN PLACE*
(END)
```

- 
- Example of sort() method

```
>>> l1.sort()
>>> help(l1.sort)

>>> l1
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.



---

# CYBRARY

---

- Slicing lists
- Slicing lists is like slicing strings
- You use the index to give you a specific place or item.
- This will give you the 5th number (starting from 0)

```
>>> 11[5]
6
>>> 11
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Note: The reverse function and using the -1 index aren't actually the same.
- One changes the list and one doesn't change the list, it just prints it backwards.
- Note 2: Don't assign a list to another list (for example l1 to l2) if you're trying to copy a list to another list. Because you'd still be using the same list, just in 2 places now.
- And if you change one, you change the other.
- Instead use list slicing and assign that to the new list

```
>>> 12 = 11[:]
>>> 12
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> 12.reverse()
>>> 11
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> 12
[9, 8, 7, 6, 5, 4, 3, 2, 1]
>>>
```

- Like so

## Lesson 2.11: Dictionaries in Python

*Skills Learned From This Lesson: dictionaries and key operations*

- Dictionaries in Python, in more details

```
>>> dict_one.keys()
dict_keys([1, 3, 5])
>>> for i in dict_one.keys():
...     print(dict_one[i])
```

- First method: Keys
- Keys give you a list of the keys you have in your dictionary
- For example this has 3 keys. The for loop will print out the values of those keys.

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

- Another for loop, this will print out the key along with its value in a neater way

```
>>> for k in dict_one.keys():
...     print("{}:{}".format(k, dict_one[k]))
...
1:2
3:4
5:6
>>> █
```

- We have pop again

```
Help on built-in function pop:

pop(...) method of builtins.dict instance
    D.pop(k[,d]) -> v, remove specified key and return the
    corresponding value.
    If key is not found, d is returned if given, otherwise
    KeyError is raised

(END)
```

- Like you would expect, pop removes a specified key but it doesn't print the key like it would in lists, it prints out the value of the removed key.
- Here's an example

```
>>> dict_one.pop(5)
6
>>> █
```

- We have removed the key called 5 and we get the value back which is 6.
- And that's all for dictionaries.

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.

---

# CYBRARY

---

## Lesson 2.12: Python summary

*Skills Learned From This Lesson: nothing because it's a summary*

- Here's a summary of the course

Python Flow Control	Python Data Types
• If/elif/else	• Strings
• For/else	• Numbers
• While	• Lists
• Logical Escapes and keywords	• Dictionaries
• Functions	

- Congratulations on learning Python.

CYBRARY

---

Brought to you by:

**CYBRARY** | FOR BUSINESS

Develop your team with the **fastest growing catalog** in the cybersecurity industry. Enterprise-grade workforce development management, advanced training features and detailed skill gap and competency analytics.