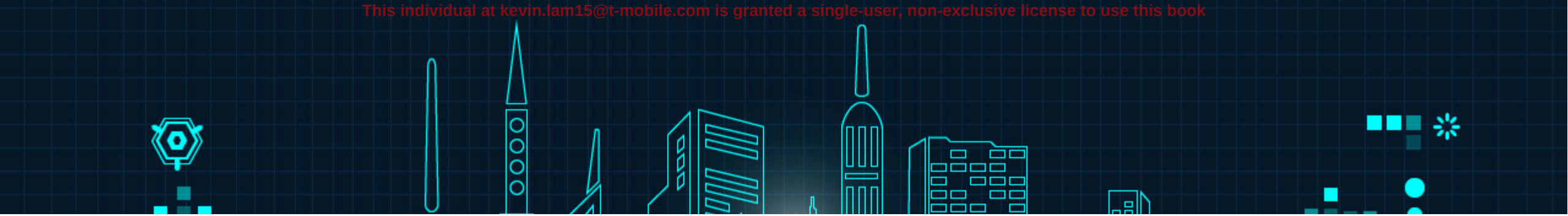
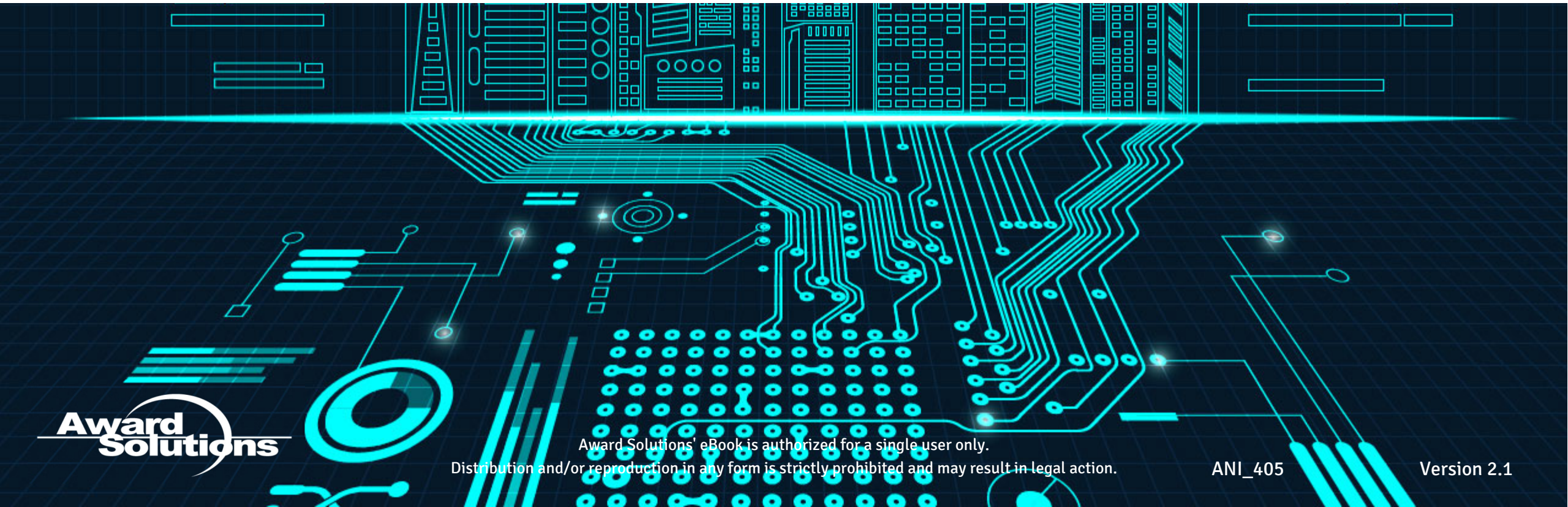


DATA AUTOMATION WORKSHOP USING PYTHON



Plano, Texas USA
Phone: +1.972.664.0727
Website: www.awardsolutions.com

If you have any questions, concerns or comments regarding this course please write to us at: **friends@awardsolutions.com**

The 3GPP, LTE, LTE-Advanced, and 5G logos are the property of Third Generation Partnership Project (*3GPP*). The content of this document is based on 3GPP/LTE specifications which are available at www.3gpp.org.

The Trusted Training Partner to the World's Best Networks

Find more resources on 5G, AI, and Automation at:

www.awardsolutions.com

Thank you for choosing Award Solutions for your training needs.

Enhance Your Skills

5G and IoT

eLearning

Welcome to 5G
5G NR Air Interface Overview Part -- I
5G NR Air Interface Overview Part -- II
5G Core Network Overview

Blended Learning

Introduction to 5G
5G (NSA) RAN Signaling and Operations
5G Core Network Signaling and Operations

Tech Primers

5G Radio Technologies and Deployments
5G Services and Network Architecture
LTE-M and NB-IoT
Multi-Access Edge Computing (MEC)
Network Slicing in 5G
5G RAN Architecture and Transport
Dark Fiber and Ethernet Backhaul

Deep Dives

5G Essentials for Leadership
5G Networks and Services
5G NR Air Interface
5G (NSA) RAN Signaling and Operations
5G RF Planning and Design
5G Core Network Signaling and Operations
Advanced LPWA for IoT
LTE-M and NB-IoT Signaling and Operations

Automation and Insights

eLearning

Welcome to AI

Blended Learning

AI Tools and Technology

Tech Primers

Artificial Intelligence (AI)
Blockchains
Immersive Technologies (AR/VR/MR)

Deep Dives

Data Automation Workshop using Python
Data Manipulation Workshop
Data Visualization Workshop using Power BI
Data Visualization Workshop
Deep Learning Concepts Workshop
Ansible Workshop

Network Virtualization

eLearning

API Overview
Big Data Overview
Cloud RAN Overview
NFV Overview
OpenStack IaaS Overview
SDN Overview
Welcome to SDN and NFV Introduction
Welcome to SDN and NFV Foundations
Welcome to SDN and NFV Technologies
Virtualization and Cloud Overview

Tech Primers

Containers and Microservices in Telecom
Cloud and Virtualization
Network Functions Virtualization (NFV)
ONAP
OpenStack
Orchestration
Software-Defined Networking (SDN)

Deep Dives

Foundation Courses
OpenStack Workshop for SDN and NFV
Scripting Workshop for SDN and NFV
SDN and NFV Architecture and Operations
Advanced Workshops
NETCONF/YANG Configuration Workshop
Introduction to Software Containers Workshop
OpenStack Heat Workshop

LTE/VoLTE

eLearning

Exploring LTE: Architecture and Interfaces
Exploring LTE: Signaling and Operations – Part I
Exploring LTE: Signaling and Operations – Part II
Exploring VoLTE: Architecture and Interfaces
Exploring VoLTE: KPIs and Error Codes
Exploring VoLTE: Signaling and Operations
LTE Air Interface Signaling Overview
LTE Overview
Multiple Antenna Techniques
Overview of OFDM
Welcome to LTE
VoLTE Overview
Overview of IPv6 in LTE Networks

Tech Primers

Licensed-Assisted Access (LAA)
Overview of CBRS

Deep Dives

RF Design Workshop
- Part I - LTE
- Part II - VoLTE and Small Cells

LTE RF Optimization
- Part I - Coverage and Accessibility
- Part II - DL and UL Throughput
- Part III - Mobility and Inter-RAT
- Part IV - Carrier Aggregation and Load Balancing

VoLTE Troubleshooting Workshop

IP/Ethernet

eLearning

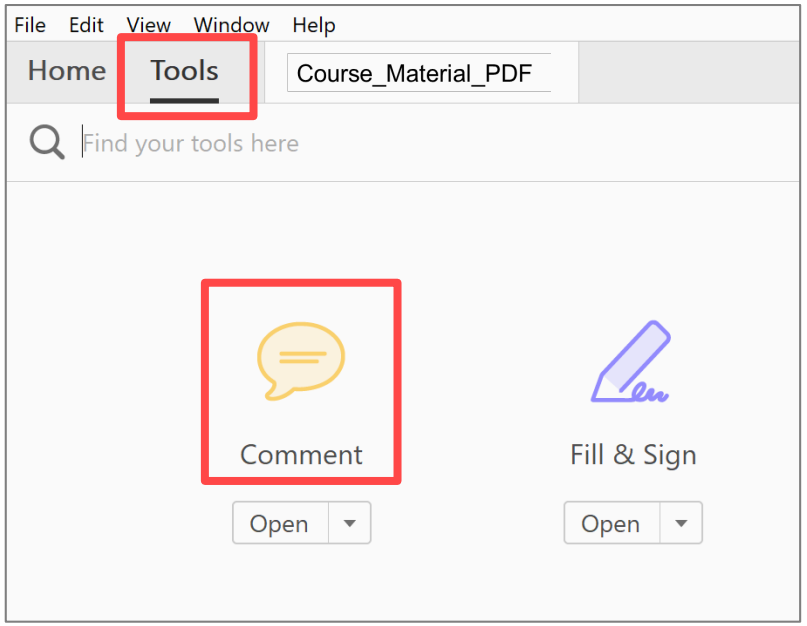
Ethernet Basics
Ethernet VLANs
Ethernet Bridging
IP Basics
Interconnecting in IP Networks
IP Quality of Service (QoS)
IP Routing
TCP and Transport Layer Protocols
Welcome to IPv6
Wireshark Overview

Award Solutions

How to Use the PDF Course Material

These instructions are for Adobe Acrobat Reader DC. Other applications may not perform the same way or support the same functions.

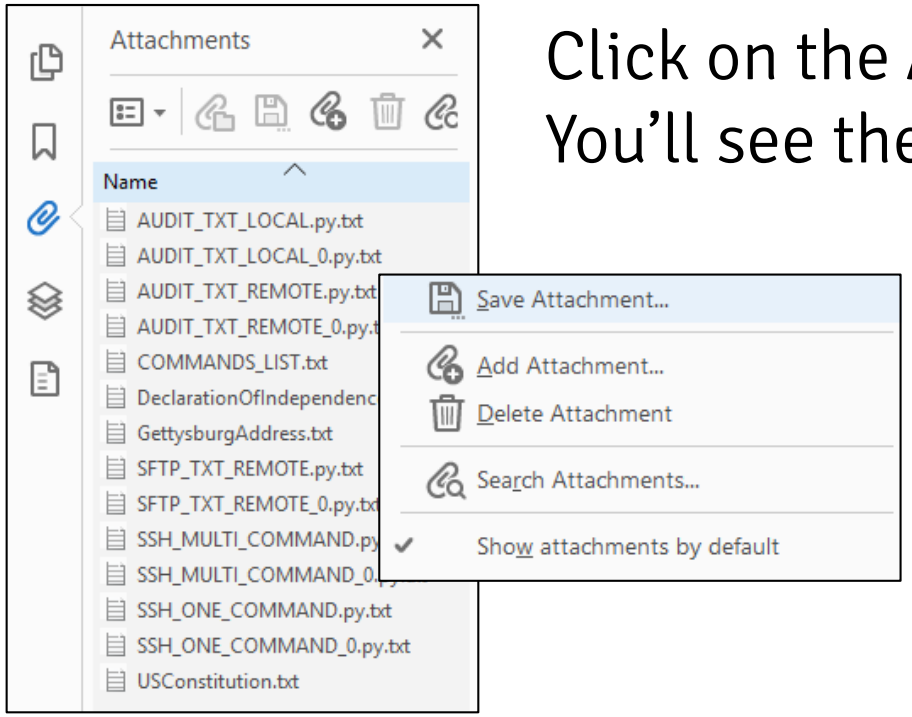
Click on the **Notes** icon to open the notes for one page.



Go to **Tools** and click **Comment** to view the Comments pane with the notes for all pages.

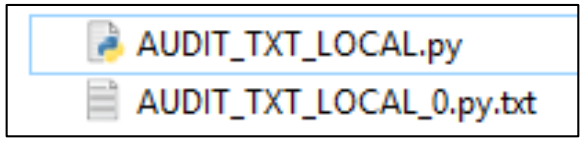
Use the **annotation tools** to add your own notes.

How to Use the PDF Course Material: Attachments



Click on the **Attachments** icon to open the Attachments pane. You'll see the .TXT files for the course.

Select all of them by using **Shift + Click** or **Ctrl + A**. Right-click and choose **Save Attachment** to save them to your computer.



The files with **.py.txt** are the example Python code. **Rename the file to remove the .txt** and you will be able to open and edit it in Notepad++.

Files that end in **_0** are only the pseudocode, and the matching files are the fully commented code written by the instructor.

Data Automation Workshop Using Python

COURSE OBJECTIVES

Install both Python and Notepad++

Discover overarching concepts of programming that you can apply to your own needs

Use Python to create short programs

Learn by doing – implement concepts and follow along

Troubleshoot along with other participants

Prologue: Getting Started with Python

OBJECTIVES

Outline the background of Python

Use the terminal window and Notepad++ to start writing and executing Python programs

Start writing programs using recommended best practices

- **Installing Python and Notepad++**
- Programming Environment
- About Python
- Getting Started
- Programming Basics

Installing Python

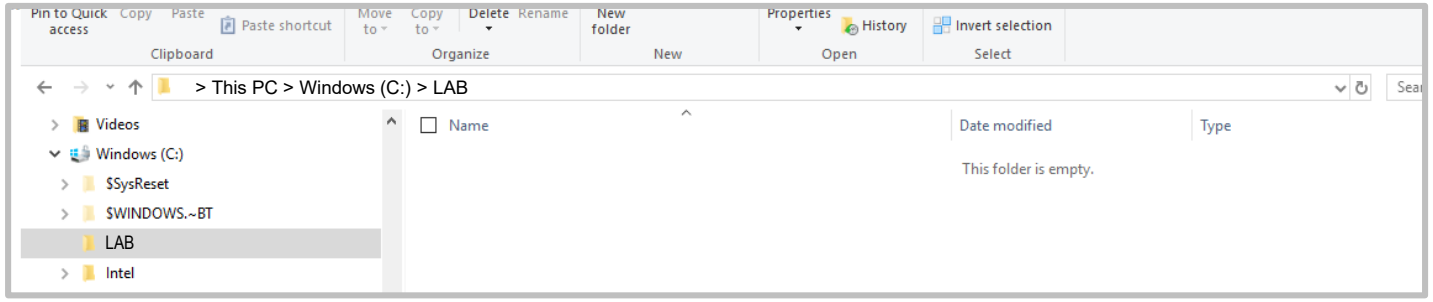
Watch the video

1. Create a folder: **C:\PYTHON**
   - This will be the folder where you will install the Python software. Once the installation is complete, you will not need to use this folder for anything.
2. Create a folder: **C:\LAB**
   - This will be your working folder. Store all your class material, .py files and data files here.
3. Download the **latest version** of Python.

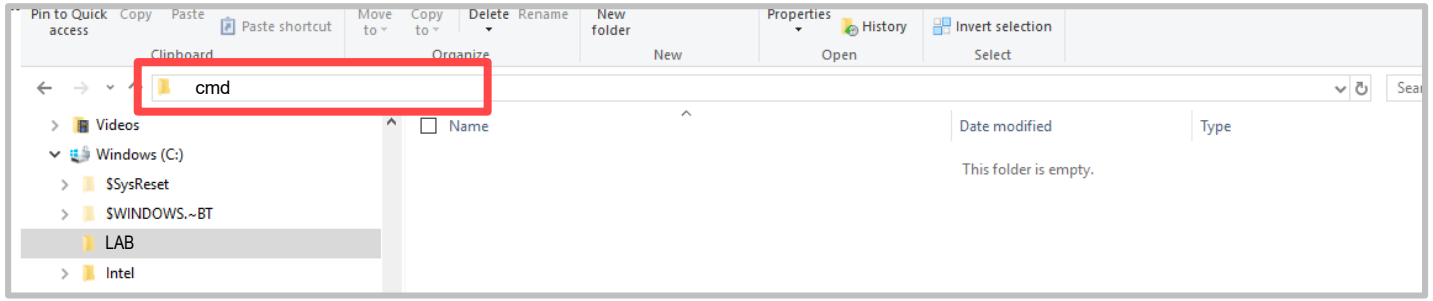
   https://www.python.org/
4. Install Python by double-clicking on the downloaded EXE file.
   - On the first screen, check the box to **Add Python to PATH.**
   - Choose **Customize Installation** and install Python in the **C:\PYTHON** folder.

Validate the Python Installation

Open Windows Explorer and navigate to your working folder: **C:\LAB**



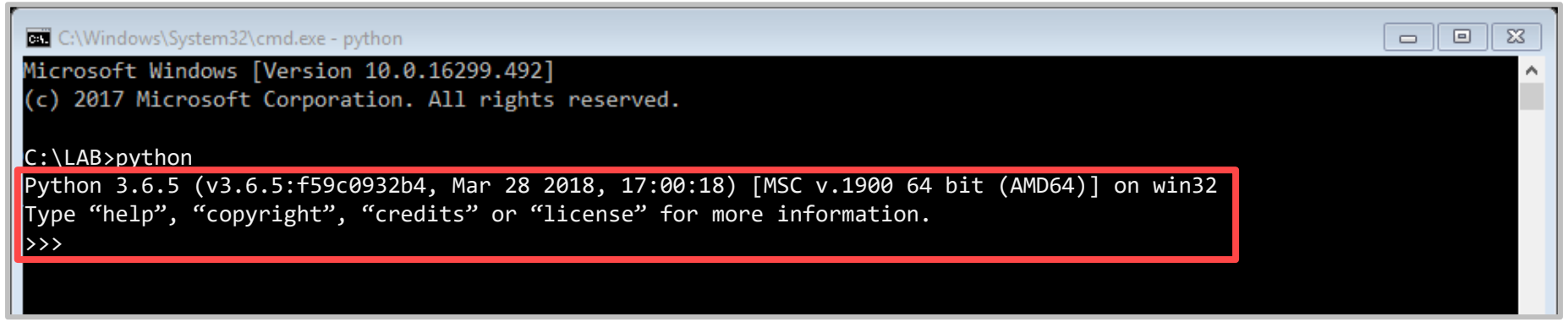
Type **cmd** in the address bar and hit **Enter** to open the Command Prompt.



The Command Prompt current working folder is **C:\LAB**

Type **python** in the Command Prompt and **hit Enter.**

It should display your **Python version details** and the **>>> prompt** to let you know you're in the Python programming environment.



The Python versions displayed are examples only. You may be installing a different version.

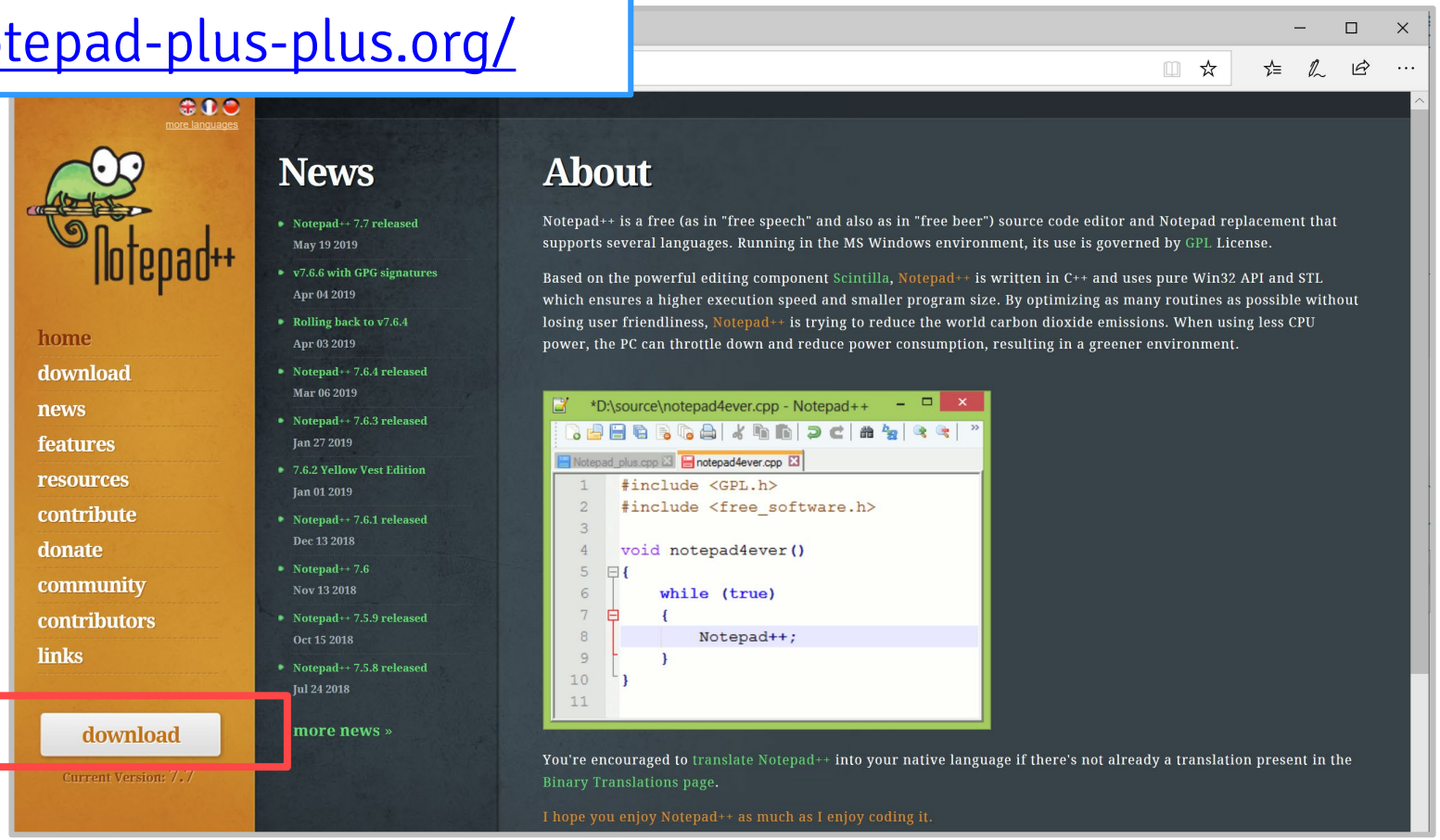
Installing Notepad++

Watch the video

Go to the **Notepad++ website** and **click on Download** to download the latest release.

Install Notepad++ by **double-clicking the EXE file** after it finishes downloading.

https://notepad-plus-plus.org/



Installing Python and Notepad++

Programming Environment

About Python

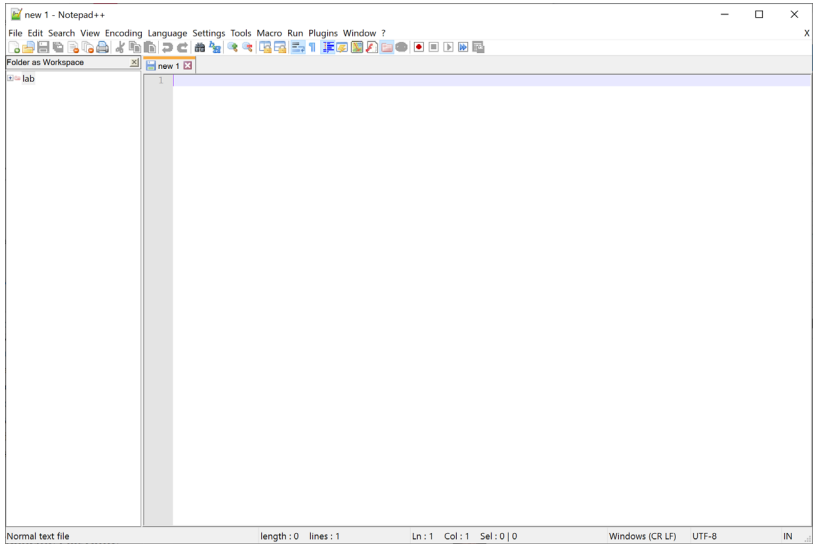
Getting Started

Programming Basics

Programming Environment

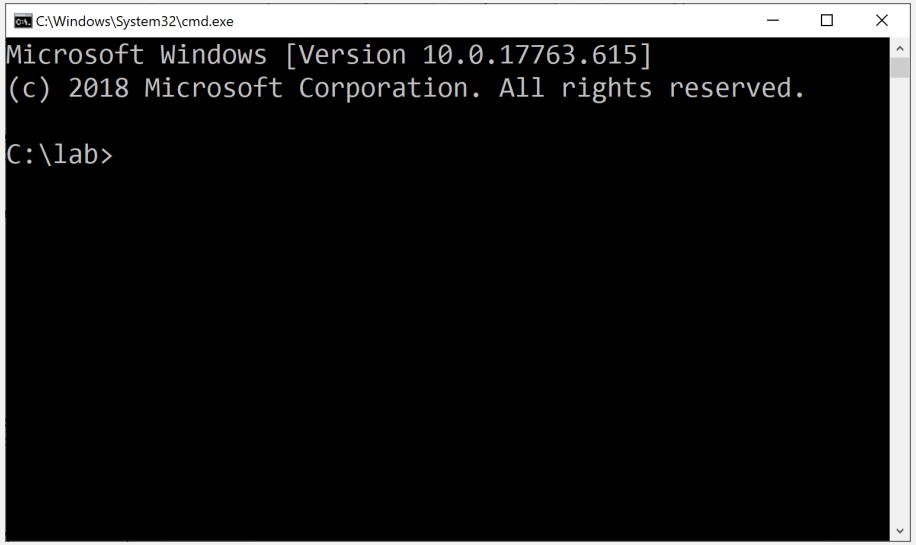
Notepad++

Write or edit the program itself



Command Prompt (Windows Terminal)

Run the program



Installing Python and Notepad++

Programming Environment

About Python

Getting Started

Programming Basics

What is Python?

Python is a **programming language** that lets you work quickly and integrate systems more effectively.

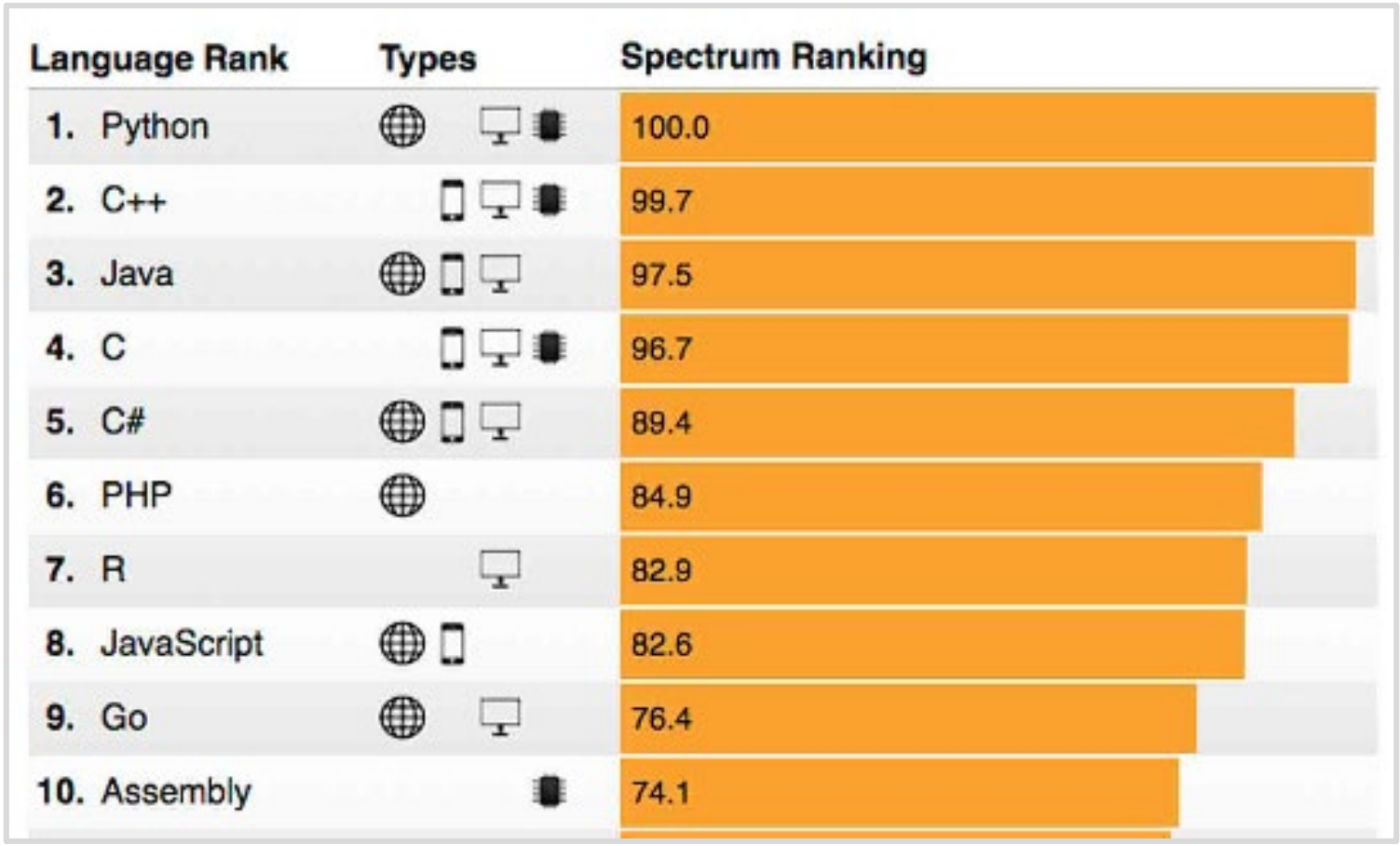
https://www.python.org/

The Python Software Foundation is an organization devoted to advancing open source technology related to the Python programming language.

Version 2.x is the legacy version of Python. Its last release was in 2010. Only use if you have a large legacy code base to support.

Version 3.x is the currently maintained and updated version of Python. It has been improved for usability and consistency.

Top Programming Languages of 2018



Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C++		99.7
3. Java		97.5
4. C		96.7
5. C#		89.4
6. PHP		84.9
7. R		82.9
8. JavaScript		82.6
9. Go		76.4
10. Assembly		74.1

July 31, 2018: https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages

Installing Python and Notepad++

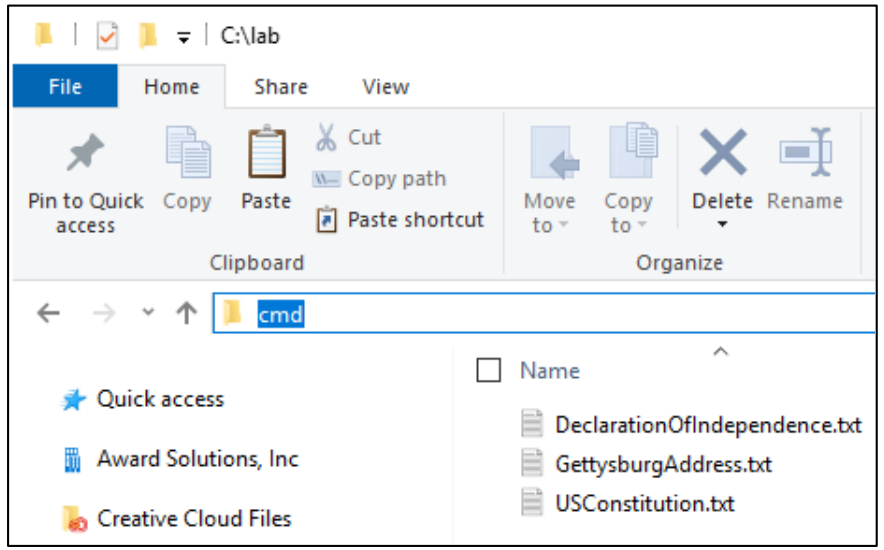
Programming Environment

About Python

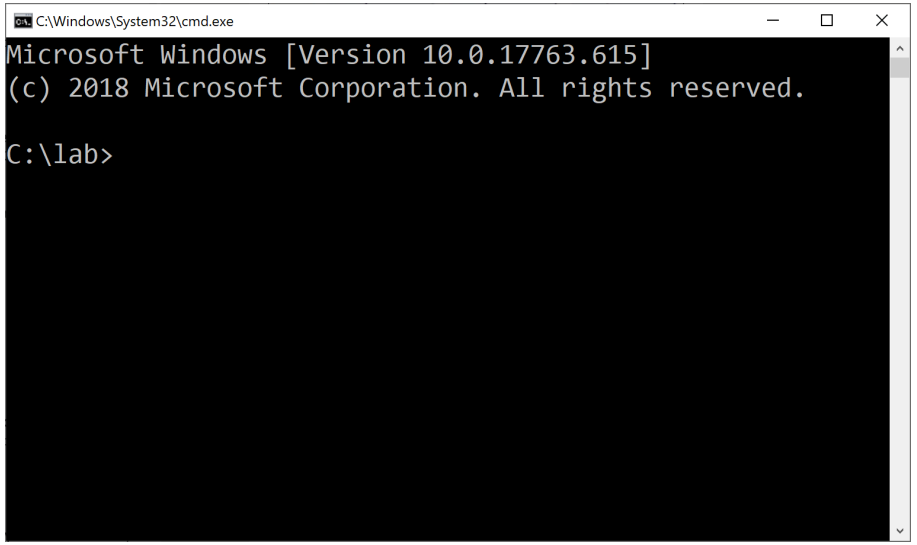
Getting Started

Programming Basics

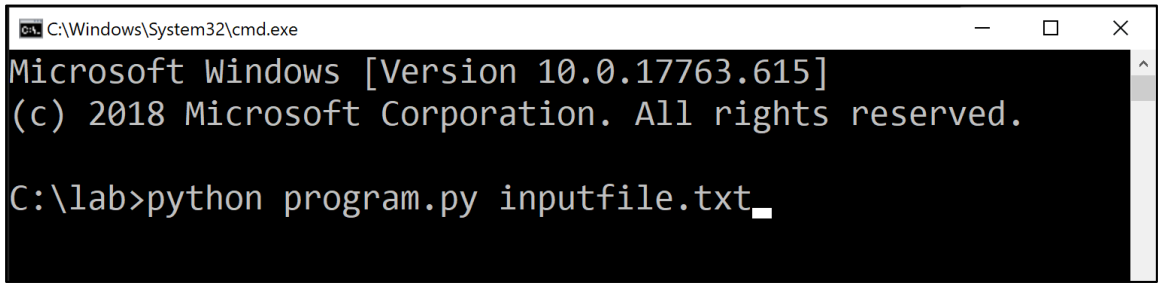
Getting Started in the Terminal Window



From your working folder (C:\lab), type **cmd** in the address bar.



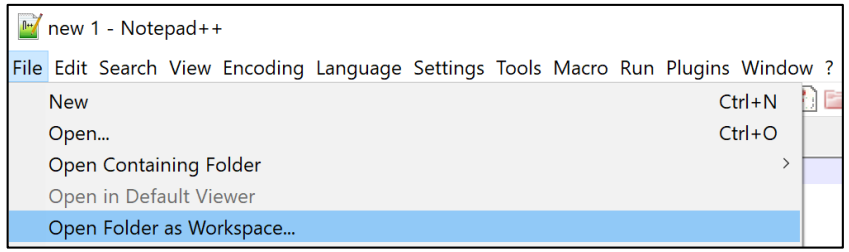
This opens a terminal window based out of the working folder.



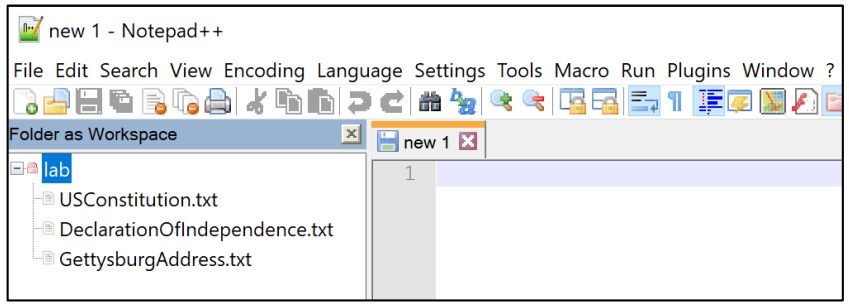
Run Python programs by typing **python [filename].py**. Our use cases will use input files entered after the program name.

Use the **up arrow key** to find the last command entered.

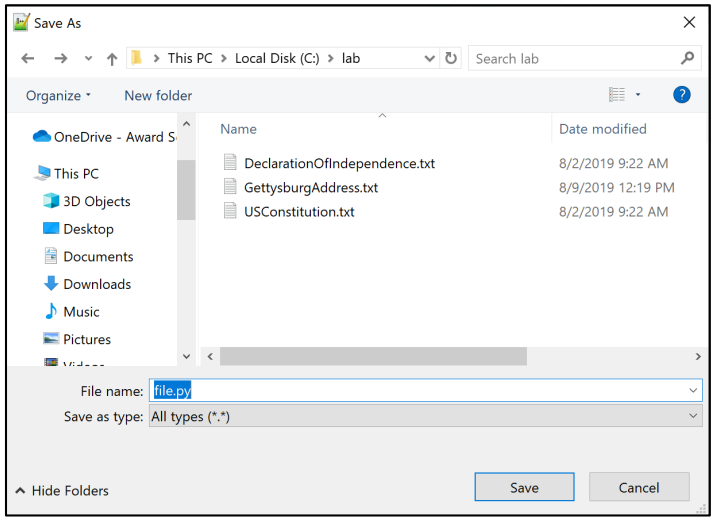
Getting Started in Notepad++



In Notepad++, go to **File > Open Folder as Workspace** and navigate to your working folder (C:\lab).



Using this workspace, you can easily see and open all the files in the working folder.



Save the default “new” Notepad++ file as **file.py**.

Installing Python and Notepad++

Programming Environment

About Python

Getting Started

Programming Basics

What is a variable?

A variable is a **user-defined container of information**. A variable can have any name except for “reserved” names.

Types of Variables

Variable Type	In Python	Definition	Example
String	str	A collection of ASCII characters, signified by single or double quotes	`var_str = “This is a string”` `var_str = ‘This is a string’`
Integer	int	A whole number	`var_int = 7`
Floating Point	float	A decimal number	`var_float = 7.7`
Boolean	bool	True or False (always capitalized)	`var_bool = False`

The String Data Type

Single Quotes

```
>>> 'spam eggs'
'spam eggs'
```

Single Quotes

```
>>> 'doesn\'t'
"doesn't"
```

use \' to escape the single quote

Double Quotes

```
>>> "doesn't"
"doesn't"
```

or use double quotes instead

```
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
```

```
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
>>> print('"Isn\'t," she said.')
"Isn't," she said.
```

Using \n

```
>>> s = 'First line.\nSecond line.'  # \n means newline
>>> s  # without print(), \n is included in the output
'First line.\nSecond line.'
>>> print(s)  # with print(), \n produces a new line
First line.
Second line.
>>> print('C:\some\name')  # here \n means newline!
C:\some
ame
```

Raw String

```
>>> print(r'C:\some\name')  # note the r before the quote
C:\some\name
>>> print("""\
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
""")
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

string literals can span multiple lines

The Boolean Data Type

There are two possible states for Boolean values: **True** or **False**

```
>>>
>>> myBoolean = True
>>> myBoolean
True
>>> type(myBoolean)
<class 'bool'>
>>>
```

Note: In Python, Booleans start with a **capital T and F**.

Comparison Operators:
Compare two values and evaluate down to a single Boolean value: **True** or **False**

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

```
>>>
>>> 32 == 32.0
True
>>>
>>> 'Hello' != 'Goodbye'
True
>>>
>>> 42 > 55
False
>>>
```

What is a module?

A module is a **set of functions** designed to interact with specific programs or perform specific tasks.

List all **installed** modules by typing **pip list**

To install a module

In the terminal window, type **pip install [module]**

```
c:\lab>pip install openpyxl
```

To use a module in your program

At the beginning of your code, type **import [module]**

```
1  # python 3
2  import os
3  import sys
4  import openpyxl
```

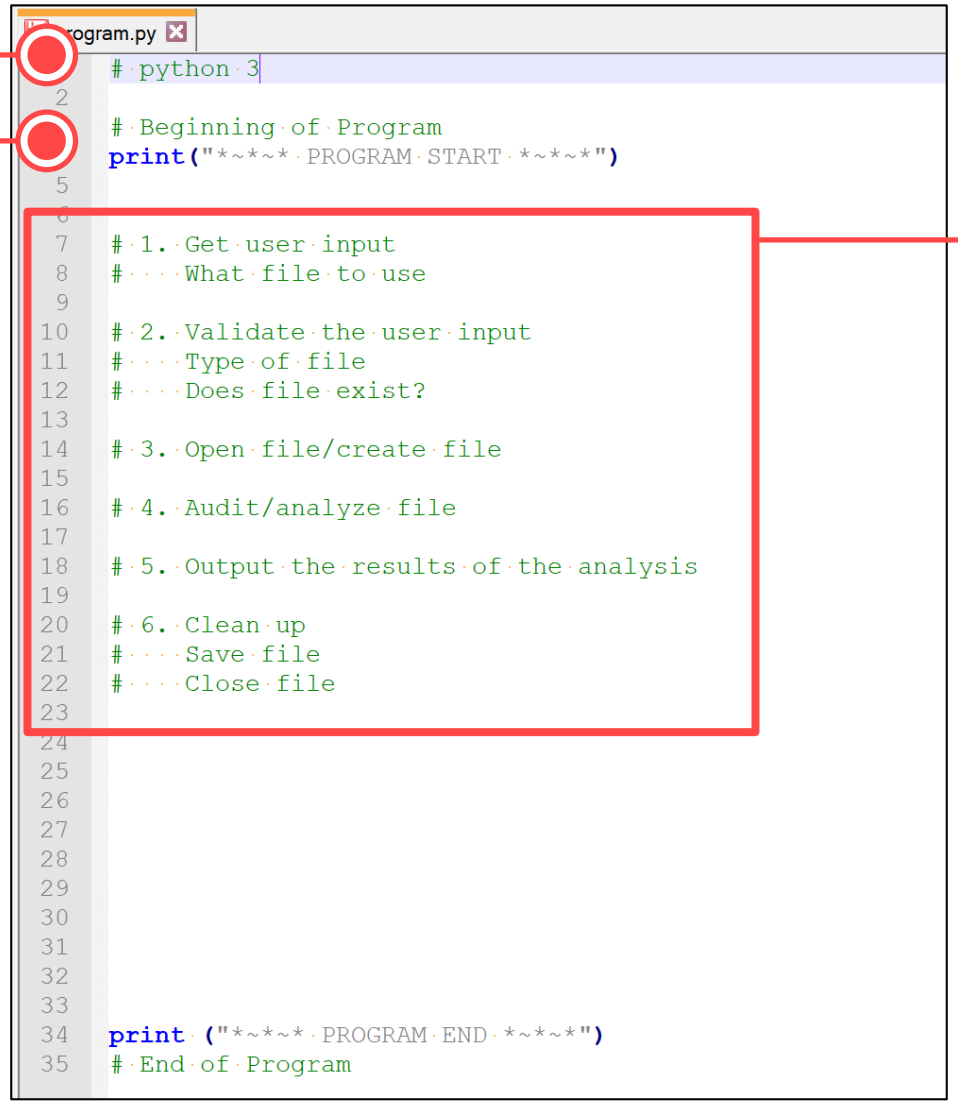
Example Modules

os	Allows Python to use operating system functionality
sys	Allows Python to access system-specific functions and parameters
openpyxl	Allows Python to read from and write to Microsoft Excel files

Coding Best Practices

Declare the **language used**

Add **comments and print statements** as you go as milestones so you and others can navigate the output of your program



Start by writing **pseudocode.**

Break the larger tasks down into logical steps. Work on one step at a time.

Lines beginning with a pound sign (**#**) are **comments** and are not executed by the program.

print is a built-in utility function in Python. It prints information on the terminal.

Set a variable using an equal sign (**=**).

Within a string, use **\n** to add a line break.

file1.py

```
1  # First command
2  print("This is a string of text.")
3
4  # Set a variable
5  my_name = "Spike"
6
7  # Print the variable as an argument
8  print("My name is", my_name)
9  print("\n")
10
11 my_age = 300
12 my_weight = 201.75
13 print("My name is", my_name)
14 print("I am", my_age, "years old.")
15 print("I weigh", my_weight, "pounds.")
```

```
C:\lab>python file1.py
This is a string of text.
```

Arguments (also called parameters) are used to give information to functions. Here, the arguments tell the print function what to print. Multiple arguments are separated by a comma (,).

```
C:\lab>python file1.py
This is a string of text.
My name is Spike


My name is Spike
I am 300 years old.
I weigh 201.75 pounds.
```

Python: Built-In Functions

The Python interpreter has a number of functions and types built into it that are always available.

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

print(...)
Print text and other data types to the terminal display

```
>>> print ('Welcome to Python')
Welcome to Python
```

https://docs.python.org/3/library/functions.html

Pseudocode

Use pseudocode as the **building blocks** for any program.

Before you start writing code, break down your larger task into **smaller logical steps**.

You can choose to work on any of the smaller steps to make a large task **more approachable**.

The tasks you need to accomplish maybe be different, but the building blocks you use will be similar.

Python Syntax: Numerical Operators

Addition

```
>>> 2 + 99
101
```

Multiplication

```
>>> 75 * 87
6525
```

Division

```
>>> 17 / 3
5.666666667
```

classic division returns a float

Floor Division

```
>>> 17 // 3
5
```

floor division discards the fraction

Remainder

```
>>> 17 % 3
2
```

return the remainder

Multiple Operations

```
>>> 5 * 3 + 2
17
```

result * divisor + remainder

Square

```
>>> 9 ** 2
81
```

9 squared

Exponent

```
>>> 2 ** 7
128
```

2 to the power of 7

Defined Variables

```
>>> width = 40
>>> height = 9 * 9
```

Calculation with Variables

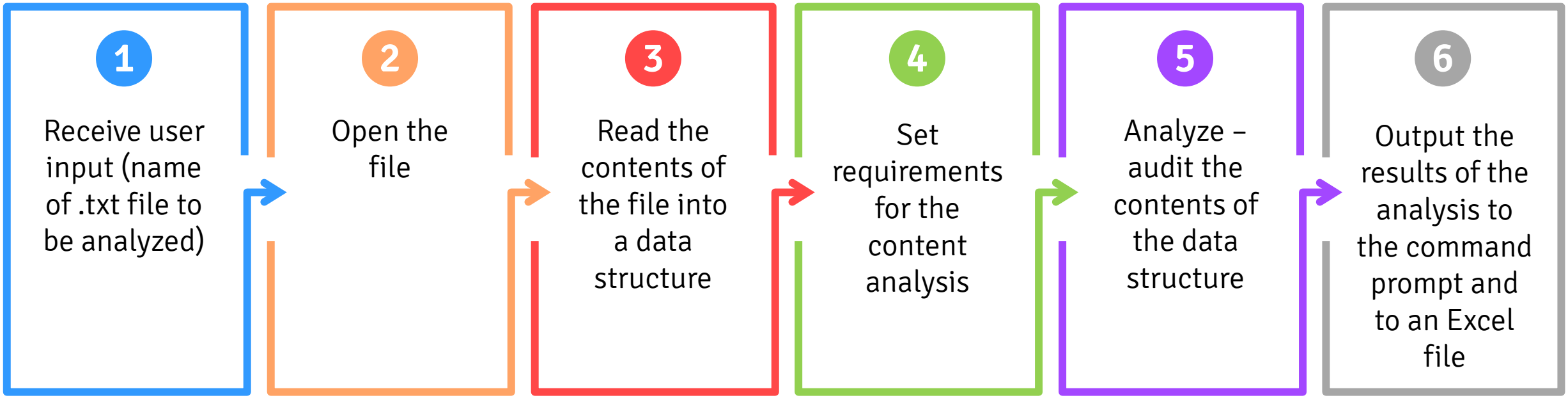
```
>>> width * height
3240
```

Undefined Variable

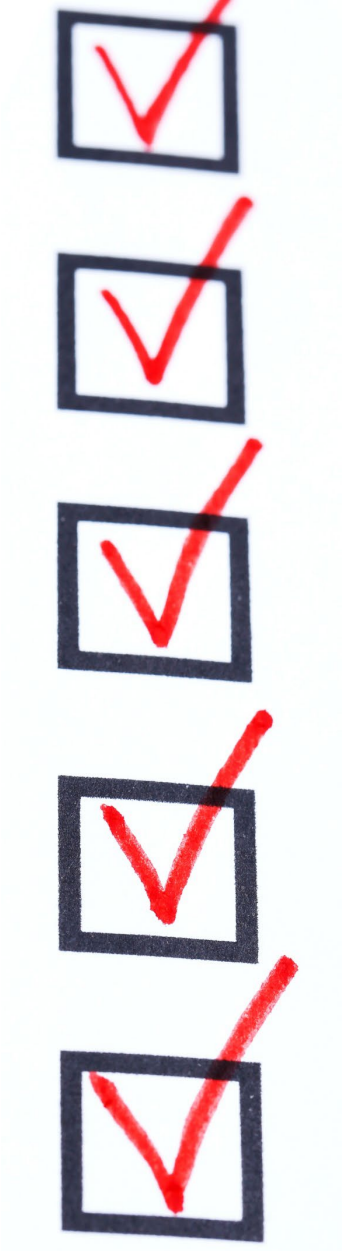
```
>>> x # try to access an undefined variable
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    x # try to access an undefined variable
NameError: name 'x' is not defined
```

Chapter 1: Local Text File Analysis

Flow of the Program



Requirements



Analyze the .txt file to find:

- Total number of lines
- Total number of words
- Total number of characters
- Total number of characters, excluding white spaces and new line characters
- Number of words, characters, and “real” characters for each line
- The number of words of length 0 to 16+, in the form of a frequency table

Create a pie chart and a bar chart in Excel with the results of the frequency table

Expected Outcome in Excel

(for GettysburgAddress.txt)

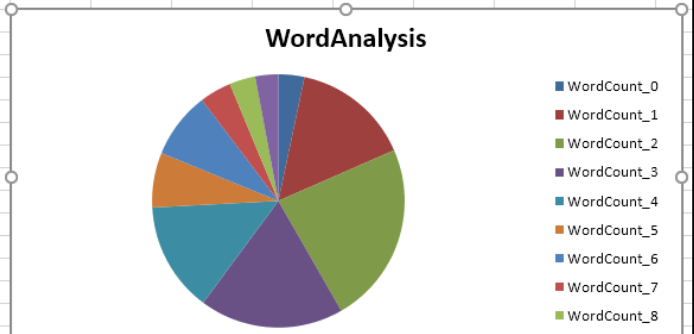
The command terminal output should include line count, word count, character count, and “no white space” character count, formatted in the way you choose, and the content of the text file that you analyze.

```
1 | 3 | 20 | 24 | 24 | THE GETTYSBURG ADDRESS

2 | 0 | 0 | 1 | 25 |

3 | 1 | 2 | 4 | 29 | OF

4 | 0 | 0 | 1 | 30 |

5 | 2 | 14 | 17 | 47 | Abraham Lincoln

6 | 0 | 0 | 1 | 48 |

7 | 3 | 15 | 19 | 67 | NOVEMBER 19, 1863

8 | 0 | 0 | 1 | 68 |

9 | 8 | 51 | 60 | 128 | Copied from Facsimile of the Original Manuscript published

10 | 7 | 37 | 45 | 173 | in 'The Century Magazine', February , 1894.

11 | 0 | 0 | 1 | 174 |

12 | 12 | 56 | 69 | 243 | Four score and seven years ago our fathers brought forth, upon this

13 | 12 | 62 | 75 | 318 | continent, a new nation, conceived in liberty, and dedicated to the prop-

14 | 7 | 34 | 42 | 360 | osition that "all men are created equal"
```

The Excel sheet output should include columns for line count, word count, character count, and “no white space” character count, and the content of the file. The second sheet should have the word count frequency table and pie chart.

	A	B	C	D	E
1	LineCount	WordCount	CharCount	RealCharCount	LineContent
2	1	3	24	20	THE GETTYSBURG ADDRESS
3	2	0	1	0	
4	3	1	4	2	OF
5	4	0	1	0	
6	5	2	17	14	Abraham Lincoln
7	6	0	1	0	
8	7	3	19	15	NOVEMBER 19, 1863
9	8	0	1	0	
10	9	8	60	51	Copied from Facsimile of the Original Manuscript published
11	10	7	45	37	in 'The Century Magazine', February , 1894.
12	11	0	1	0	
13	12	12	69	56	Four score and seven years ago our fathers brought forth, upon this
14	13	12	75	62	continent, a new nation, conceived in liberty, and dedicated to the prop-
15	14	7	42	34	osition that "all men are created equal"
16	15	0	1	0	

	A	B
1	WordLength	WordCount
2	WordCount_0	0
3	WordCount_1	9
4	WordCount_2	41
5	WordCount_3	63
6	WordCount_4	50
7	WordCount_5	38
8	WordCount_6	19
9	WordCount_7	23
10	WordCount_8	11
11	WordCount_9	9
12	WordCount_10	8
13	WordCount_11	0
14	WordCount_12	0
15	WordCount_13	0
16	WordCount_14	0
17	WordCount_15	0
18	WordCount_16	0

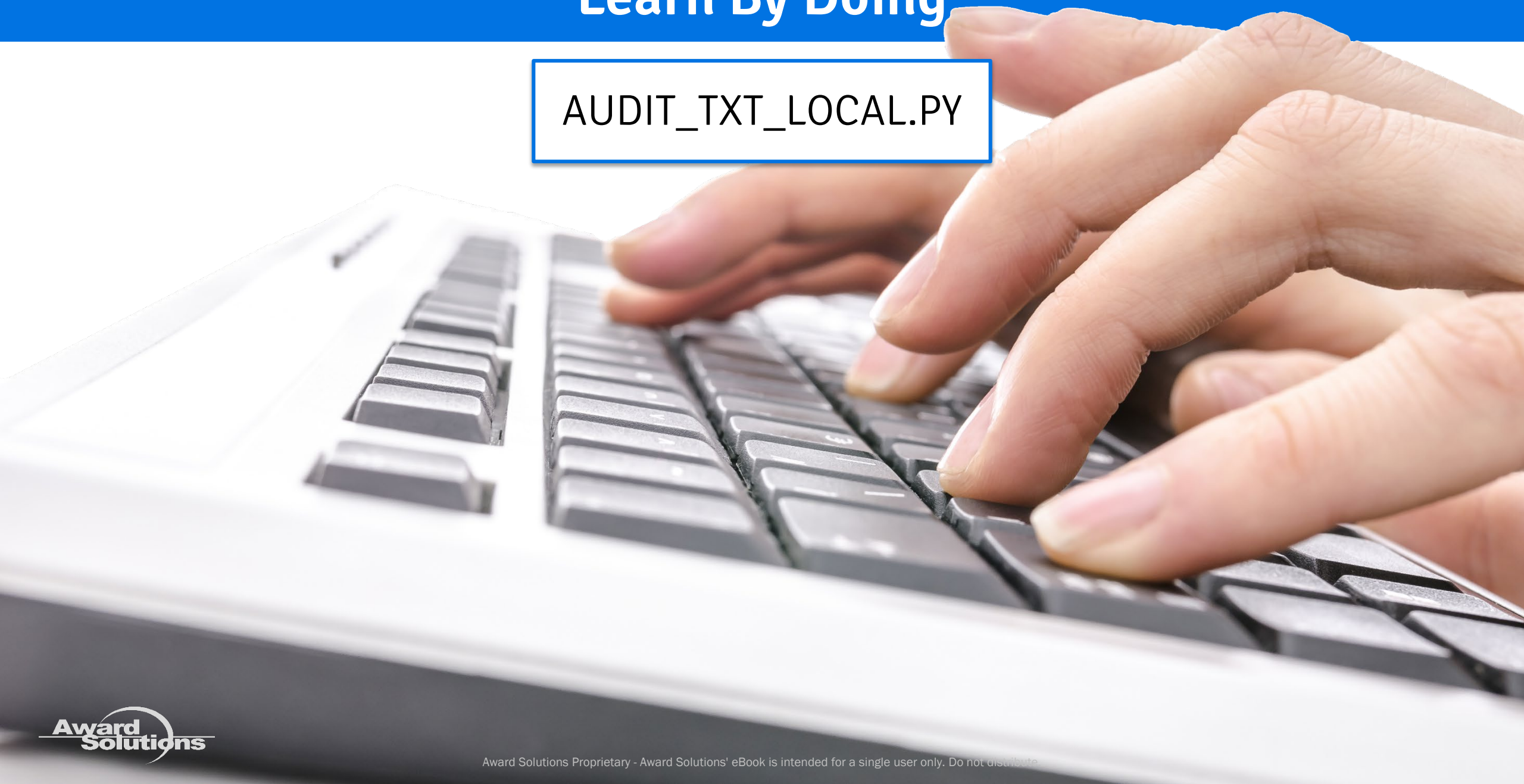


Pseudocode

```
print("\n\n++-->> Execution START ... AUDIT_TXT_LOCAL.py")
print("++")

#####
# 1. GET input
#    get the input from the user command line
#    and make sure that the user has provided 2 input parameters at the command
#    prompt and the second parameter is the name of a text file that needs to
#    be audited

#####
# 2. VALIDATE user input
#    validate whether the file name contained in a the input_filename variable
#    actually exists in the working folder - the folder where the program is
#    executing

#####
# 3. OPEN/CREATE files
#
# 3.1 now that we have done all the checks we are ready to open the input file.

# 3.2 next, we need to create the xlsx file to output the contents of the audit.

#####
# 4. AUDIT the file
#
# now that we have done the prep-work for the audit/analysis of the text file
# contents let's do the actual audit and output to the terminal and the xlsx
# file.

#####
# 5. OUTPUT the contents and the result of the audit

#####
# 6. CLEANUP and end the program

print("++")
print("++-->> Execution END ... AUDIT_TXT_LOCAL.py\n\n")
```

Learn By Doing

AUDIT_TXT_LOCAL.PY



New Modules

os	Allows Python to use operating system functionality
sys	Allows Python to access system-specific functions and parameters
openpyxl	Allows Python to read from and write to Microsoft Excel files

New Functions

`exit()`	Exit the program
`os.getcwd()`	Get current working directory
`append()`	Add an item to the end of a list
`len()`	Find the length of a string
`open()`	Open a file

New Data Structures

`sys.argv[]`	Arguments provided by user in the command line stored in a list

os: Allows Python to use operating system functionality

sys: Allows Python to access system-specific functions and parameters

```
25  # the import system of python
26  # https://docs.python.org/3/reference/import.html
27  #
    import os
    import sys

30
31  # pip install openpyxl
32  import openpyxl
33
34  #####
35  # link to python.org documentation page
36  # https://docs.python.org/3/
```

openpyxl allows Python to read from and write to Microsoft Excel files

Links for Reference

Import

Python 3 Documentation

print a statement that lets you know the program is starting

sys.argv contains the arguments entered on the command line

```
# https://docs.python.org/3/library/sys.html

# print the name of the program file on the windows command prompt.
#
# the print() function
#
# https://docs.python.org/3/library/functions.html?highlight=open#print
#
print("\n\n++-->> Execution START ... AUDIT_TXT_LOCAL.py")
print("++")

# use the sys module to capture all the strings from the command prompt that the
# user entered. the sys module captures each string (separated by whitespace)
# and puts them in the argv list to make them available to the program.

# print the argv list
# the list data structure is a very useful and important data structure
#
# https://docs.python.org/3/library/stdtypes.html#list
#
print("sys.argv is", sys.argv)

# print the name of the program file, it is always at index 0 of sys.argv
#
print("name of program file at sys.argv[0] is ", sys.argv[0])
```

```
c:\lab>python audit_txt_local.py GettysburgAddress.txt
```

sys.argv[0] is the first item in the list

sys.argv[1] is the second item in the list

Links for Reference

- Sys Module
- Print Function
- List Data Structure

Create an **if statement** to check if there are exactly two arguments – if not, exit the program.

Create a **local variable** to store the string for the file name

This **if statement** uses **negative logic.** If the file name does not end with .txt, then exit the program.

```
if len(sys.argv) != 2:
    # exit this program, do not process anything beyond this line
    # https://docs.python.org/3/library/constants.html?highlight=exit#exit
    #
    exit("who are you trying to fool? exiting ...")

# create a local variable to store the string for the name of the file the user
# has provided. expect the filename to be at index location 1 of sys.argv list.
# print the name of the file to the command prompt
#
input_filename = sys.argv[1]
print("input_filename:",input_filename)

# the input_filename variable is of type string (str) or a string object
# https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str
#
# there are many convenient string manipulation capabilities (functions) that
# python provides endswith() is one of them. it allows you to check is the str
# endswith a specific sequence of characters
#
# here we are using negative logic for the if control statement
#
# we want to make sure that if the filename does not end with ".txt" then the
# user has not provided a text file for the audit and we will exit the program
#
if not input_filename.endswith(".txt"):
    exit("who are you trying to fool ... again? exiting ...")
```

Links for Reference

Exit Function

String Object

if else

An **if else statement** tests a condition and performs actions based on whether the condition is true or false.

```
1  shopping_list_nextweek = []
2  item_found = find item in store
3  if item_found == False:
4        shopping_list_nextweek.append(item)
5  else:
6        put item in cart
7        shopping_list.crossout(item)
```

os.getcwd() gets the current working directory path. We store it in a local variable.

os.listdir() lists the contents of the folder. Storing it in a working variable makes it easy to perform actions on it.

```
# https://docs.python.org/3/library/os.html
#
# use the built-in os module and use getcwd() (get current working directory)
# function in the module to get the path of the working folder. print it.
#
# https://docs.python.org/3/library/os.html?highlight=getcwd#os.getcwd
#
working_folder = os.getcwd()
print("working_folder:", working_folder)

# use the built-in os module and use listsir() (list directory)function in the
# module to get the contents of the working folder.
# the function returns a list object with all the names of files and folders in
# the working directory. print the list so we can visually see the list
#
# https://docs.python.org/3/library/os.html?highlight=listdir#os.listdir
#
files_list = os.listdir()
print("files_list:", files_list)

# create a variable of type boolean and set it to False. we will use it to keep
# track of whether we have found the input file in the working directory
#
file_found = False
```

Set a **Boolean variable** to keep track of whether the input file is found or not.

Links for Reference

- OS Module
- os.getcwd() Function
- os.listdir() Function

A **for loop** works through the number of elements in a list and performs actions on each one.

This **if statement** is within the for loop. If the item in the list is equal to the variable set earlier, then set the Boolean variable to True.

The **break** statement breaks out of a loop and makes the program move on to the next line of code.

The **else statement** is part of the **if statement**. It performs alternative actions with the if statement is false.

```
146  # the for statement
147  # https://docs.python.org/3/reference/compound_stmts.html#the-for-statement
148  #
149  # execute the code block of the for loop as many times as there are number of
150  # elements in the files_list. for each execution of the for loop set the local
151  # my_file variable contents to the contents of list a specific index, starting
152  # from index 0 all the way to the last index
153  #
154  for my_file in files_list:
155      # print the value of my_file for this execution of the for loop code block
156      print(my_file)
157
158      # compare the contents of my_file with the contents of input_filename.
159      #
160      # only if the contents are the same then print and set the value of the
161      # file tracking boolean to True because we found the file we are looking for
162      if my_file == input_filename:
163          print("Yay ... i found a match")
164          file_found = True
165
166          # break out of the execution of the for loop - do not go through more
167          # iterations of the for loop
168          #
169          #
                 https://docs.python.org/3/reference/simple_stmts.html#grammar-token-break
                 -stmt
170          #
171          break
172      # if the contents of my_file and the contents of input_filename are anything
173      # other than the same, print something and go to the next iteration of the
174      # for loop
175      else:
176          print("Match not found ... yet!")
177      # end of if
178  # end of for
```

Links for Reference

- For Statement
- Break Statement

for Loop

A **for** loop goes through each item in a list and performs one or more actions for each item.

```
1  for item in shopping_list:
2       find item in store
3       put item in cart
4  checkout_counter.unload(cart)
5  checkout_counter.pay()
```

Working in Excel with Python using openpyxl

https://openpyxl.readthedocs.io/en/stable/

Basic File Management

Create a new workbook	Save the openpyxl.Workbook() function as a variable so you can perform actions on it	`WB = openpyxl.Workbook()`
Set the active worksheet	Save the workbook's "active" parameter as a variable so you can perform actions on it	`WS1 = WB.active`
Create a new worksheet	Use the create_sheet function and save it as another variable	`WS2 = WB.create_sheet("Title")`
Save the file name	Save the XLSX file with the original file name	`xlsx_filename = input_filename.replace(".txt", ".xlsx")`
Save the file	Don't forget to save the file – add this command after all code that makes changes to the XSLX file, so you can see your results	`WB.save(xlsx_filename)`
Close the file	Close the file	`WB.close()`

Assigning workbooks and worksheets to variables makes it easy to refer to them later.

Change the title of worksheet 1

Create a new worksheet

Make sure to save the workbook so you can see your changes.

```
211  # openpyxl is a python library to read/write excel 2010 xlsx/xlsm files. to
212  # create a new workbook(xlsxS file) in program memory instantiate an openpyxl
213  # Workbook() object and then save the workbook object into a variable so that
214  # we can take actions on the workbook. a workbook is a container for all the
215  # parts of the xlsx file.
216  #
     WB = openpyxl.Workbook()

219  # when openpyxl creates a new workbook in program memory it creates a one empty
220  # worksheet called "Sheet" in the workbook. every xlsx file/workbook maintains a
221  # a "parameter" or property called - active. it stores the current worksheet
222  # that is active. use the property to get the worksheet object and store it in
     abs
223  # variable.
224  #
225  WS1 = WB.active
226
227  # change the name of the worksheet by updating the title property. we will use
228  # the string manipulation function called replace(). we will write the output
229  # from the audit into this worksheet.
     #
     WS1.title = input_filename.replace(".txt", "")
232
233  # now let's create a second sheet in the workbook. in this worksheet we will
234  # write the word length frequency table and create charts to represent the
235  # table with a visualization. we will name this sheet "WORD_ANALYSIS"
     WS2 = WB.create_sheet("WORD_ANALYSIS")

238  # create a variable to store the name of the xlsx file. similar to what we did
239  # earlier for the worksheet name
240  xlsx_filename = input_filename.replace(".txt", ".xlsx")
241
242  # let's save the xlsx file. the save function of the workbook object tak
243  # current state of the workbook in program memory and writes it to disk.
244  # we will do this again at the end of the program.
     WB.save(xlsx_filename)
```

Links for Reference

openpyxl

Simple Usage in openpyxl

Create a variable that is an **empty list**

Using a **for loop** to build out the list

```
freq_table = []

# let's initialize the table - lots of ways to do it, we will leverage a for
# loop to add counters with an intial value of 0.
# we will use variable to set how many times to loop,
#
# use the range() function specially designed for these kinds of things
# https://docs.python.org/3/tutorial/controlflow.html?highlight=range
#
# use the append() function of the list object to add a new item to a list
# https://docs.python.org/3/library/stdtypes.html?highlight=list%20append
#
# note the nuance of the for loop while working with range. when the count
# value equals the stop value, the loop breaks out and does not execute the
# code block
#
print("freq_table:", freq_table)
COUNTERS = 17
for index in range(0, COUNTERS, 1):
    freq_table.append(0)
    print(index, "~", freq_table)
print("freq_table:", freq_table, "\n")
```

Links for Reference

- range() Function
- append() Method

The List Data Structure

A **list** is an indexed collection of items. Lists can be made up of strings, integers, floating points, or any other data structure.

Be aware! In Python, lists start at 0. In this list, item 0 is “milk” and item 1 is “eggs.”

Define an **empty list** using square brackets

Or a list with items **separated by commas** (,)

```
1
2  shopping_list = []
3  shopping_list = ["milk", "eggs", "apples", "bread"]
4
5  print(shopping_list)
6
```

Sample List Methods

append() Add an element to the end of a list

clear() Clear the list

count() Count the number of elements in a list

insert() Insert an item somewhere in the list

reverse() Reverse the order of the list

sort() Sort the list

Working with Lists

x = [5, 6, 7] y = [8, 9, 10]

Concatenation	x + y	[5, 6, 7, 8, 9, 10]
Multiplication	x * 2	[5, 6, 7, 5, 6, 7]

Writing to Worksheets with openpyxl

Hard-coding row and column values

```
WS1.cell(row=1, column=1).value = "Cell Contents"
```

It is an option to **hard-code** data into a table. However, this makes the program **less flexible** if you need to use a source file with different contents.

Using counters to dynamically enter data

```
line_count = 0
for my_line in input_file object:
    ...
   WS1.cell(row=line_count+1,column=1).value = my_line
    ...
   line_count += 1
```

Using a **for loop** and **defined variables**, you can **dynamically enter data** in a way that can adapt to different data sources.

Hard-coding in cell values

```
# let's create the column headers in row 1 for all the data we want to write
# into the xlsx workbook
#
# column headers for worksheet 1
#
WS1.cell(row=1,column=1).value = "LINE_COUNT"
WS1.cell(row=1,column=2).value = "WORD_COUNT"
WS1.cell(row=1,column=3).value = "CHAR_COUNT"
WS1.cell(row=1,column=4).value = "NOWS_CHAR_COUNT"
WS1.cell(row=1,column=5).value = "LINE_CONTENT"

# column headers for worksheet 2
#
WS2.cell(row=1,column=1).value = "WORD_LENGTH"
WS2.cell(row=1,column=2).value = "WORD_COUNT"
```

Using **variables** to dynamically enter data in the worksheet

```
# output the totals counter values to the terminal and the xlsx worksheet 1
print("\n")
print("total number of lines in the file:", line_count)
WS1.cell(row=line_count+3,column=5).value = "total number of lines in the file:"
WS1.cell(row=line_count+3,column=1).value = line_count

print("total number of words in the file:", total_word_count)
WS1.cell(row=line_count+4,column=5).value = "total number of words in the file:"
WS1.cell(row=line_count+4,column=2).value = total_word_count

print("total number of chars in the file:", total_char_count)
WS1.cell(row=line_count+5,column=5).value = "total number of chars in the file:"
WS1.cell(row=line_count+5,column=3).value = total_char_count

print("total number of no_ws chars in the file:", total_nows_char_count)
WS1.cell(row=line_count+6,column=5).value = "total number of no_ws chars in the
file:"
WS1.cell(row=line_count+6,column=4).value = total_nows_char_count
```

Using the openpyxl chart functions to **add charts** inside a worksheet

```
412  # now the charts
413  #
414  # create openpyxl Reference objects to identify the location of the data and the
415  # labels for the data. note the use of the COUNTERS variable.
416  #
417  labels = openpyxl.chart.Reference(WS2, min_col=1, min_row=2, max_row=COUNTERS+1)
418  data = openpyxl.chart.Reference(WS2, min_col=2, min_row=1, max_row=COUNTERS+1)
419
420  # create the pie chart in the program memory
421  # https://openpyxl.readthedocs.io/en/stable/charts/pie.html
422  #
423  pie = openpyxl.chart.PieChart()
424
425  # now, provide the data and labels to the pie chart object using the created
426  # Reference objects. give a name to the pie chart
427  #
428  pie.add_data(data, titles_from_data=True)
429  pie.set_categories(labels)
430  pie.title = "WORD_ANALYSIS"
431
432  # insert the pie chart into the "WORD_ANALYSIS" worksheet anchored at cell F1
433  #
434  WS2.add_chart(pie, "F1")
```

Links for Reference

Pie Chart in openpyxl

Use the same data to add a bar chart

```
436  # for kicks and grins let's add a bar chart for the same data
437  # https://openpyxl.readthedocs.io/en/stable/charts/bar.html
438  #
439  bar = openpyxl.chart.BarChart()
440  bar.type = "col"
441  bar.style = 10
442  bar.title = "WORD_ANALYSIS"
443  bar.y_axis.title = 'WORD_COUNT'
444  bar.x_axis.title = 'WORD_LENGTH'
445
446  # now, provide the data and labels to the pie chart object using the created
447  # Reference objects. give a name to the pie chart
448  #
449  bar.add_data(data, titles_from_data=True)
450  bar.set_categories(labels)
451  bar.shape = 4
452
453  # insert the bar chart into the "WORD_ANALYSIS" worksheet anchored at cell F30
454  #
455  WS2.add_chart(bar, "F30")
456
457  #####
458  # 6. CLEANUP and end the program
459  #
460  # let's save the xlsx file. the save function of the workbook object takes the
461  # current state of the workbook in program memory and writes it to disk.
462  # then close the file
463  WB.save(xlsx_filename)
464  WB.close()
```

Save and close the workbook so your work is visible

Links for Reference

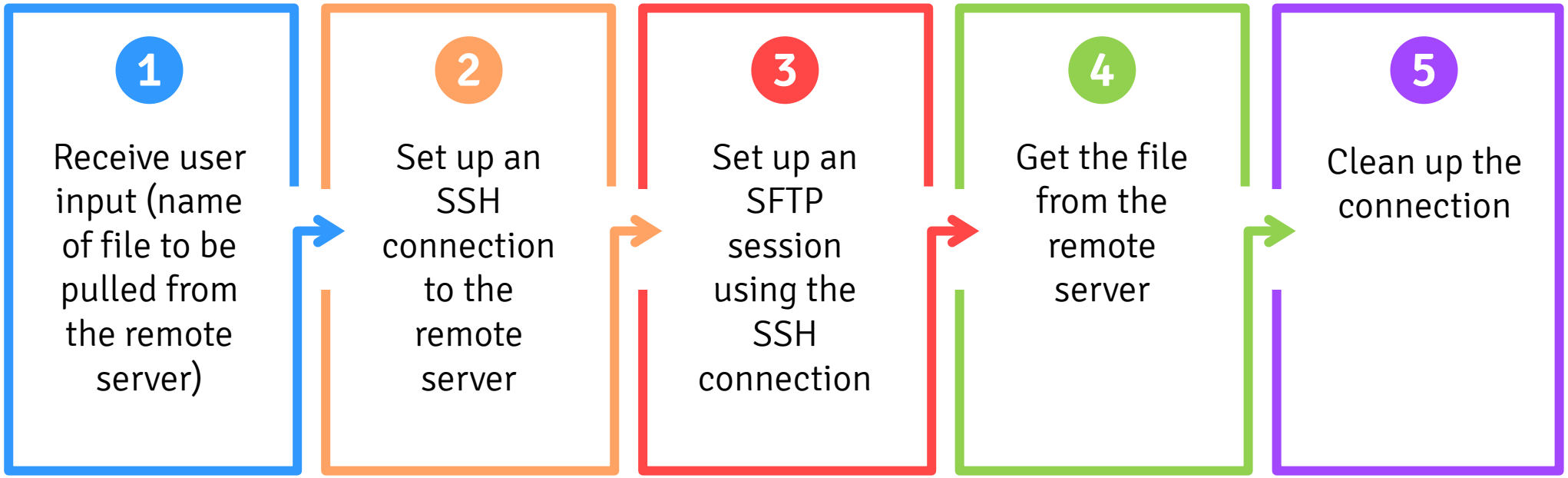
Bar Chart in openpyxl

Chapter 2: Remote Text File Analysis

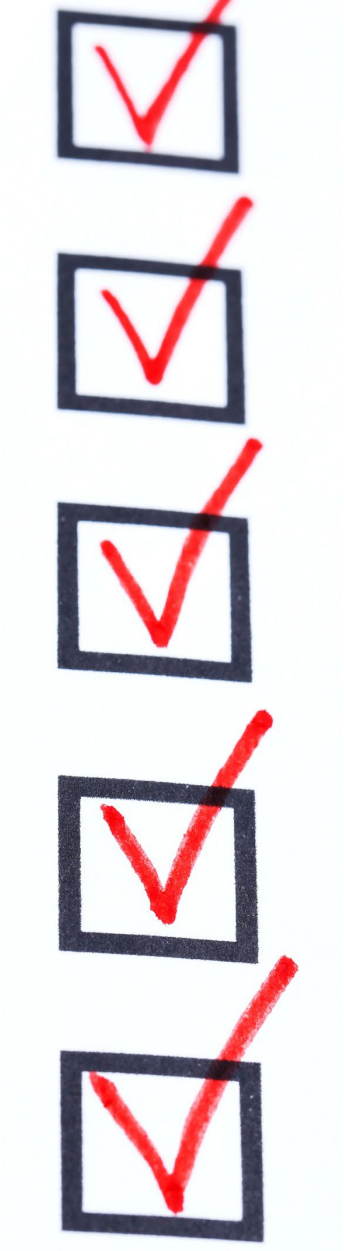
Part 1: Build a Program using Secure FTP

Part 2: Modify the Text Audit Program to Use SFTP

Flow of the Program



Requirements



Set up an SSH connection to the remote server

Set up an SFTP session over the SSH connection

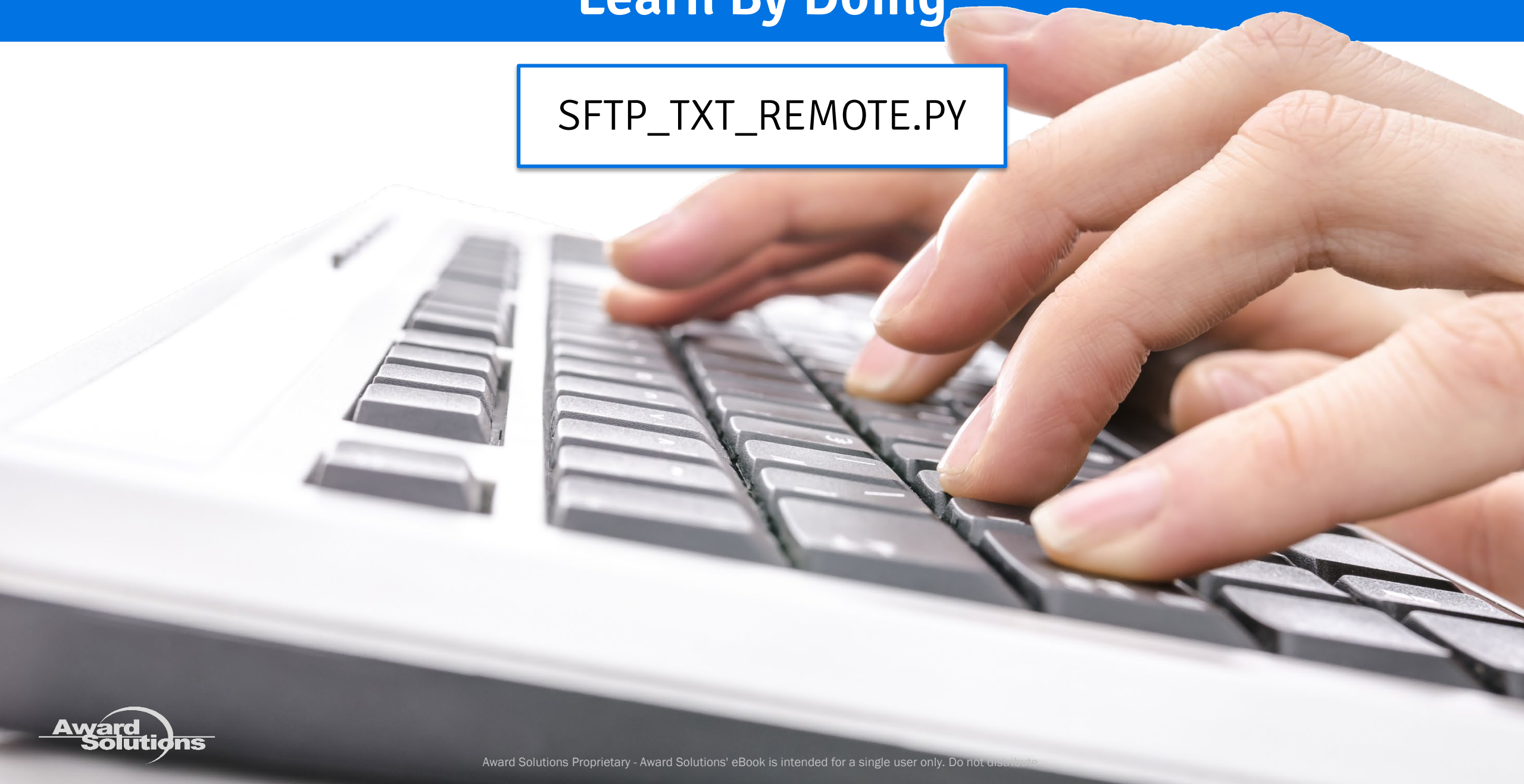
Build the parts of the program as functions, to make them easily reuseable

Pseudocode

```
26  print("\n\n++-->> Execution START ... SFTP_TXT_REMOTE.py")
27  print("++")
28
29  # 1. get input from the user, the filename to be SFTP from the remote server
30  #    BUILD this as a FUNCTION
31  #
32
33  # 2. setup a SSH connection to the remote server
34  #    BUILD this as a FUNCTION
35  #
36
37  # was the SSH connection setup successful, if not then let's clean up
38  #
39
40  # 3. setup an SFTP session using the SSH connection
41  #    BUILD this as a FUNCTION
42  #
43
44  # was the SFTP session open successful, if not then let's clean up
45  #
46
47  # 4. get the file from the remote server
48  #    BUILD this as a FUNCTION
49
50  # 5. cleanup the connection
51  #    BUILD this as a FUNCTION
52  #
53
54  print("++")
55  print("++-->> Execution END ... SFTP_TXT_REMOTE.py\n\n")
56
57  # END OF FILE
```

Learn By Doing

SFTP_TXT_REMOTE.PY



```
25  import os
26  import sys
27
28  # pip install paramiko
29  import paramiko
```

Paramiko is a Python implementation of the SSH protocol.

To install, in the command line window, use the command **pip install paramiko**.

Paramiko

In this program, we write several **functions** that can be called multiple times within the main program.

Functions are defined using the syntax `def FunctionName(arguments):` and the function code must be in an **indented block**.

We use **Paramiko** functions to initialize the SSH client and access it.

```
def SetupSSHConnection(host, user, key):
    print("I am inside ... SetupSSHConnection() function")

    print("host ~", host)
    print("user ~", user)
    print("key ~", key)

    ret_value = True

    # http://www.paramiko.org/
    #
    # http://http://docs.paramiko.org
    #
    # 1. initialize SSHClient
    #    the SSHClient is responsible for establishing a session with the SSH
    #    server. It takes care of setting up the transport, channels, SFTP
    #    sessions and authentication - based on user request and use.
    #
    # http://docs.paramiko.org/en/2.6/api/client.html
    #
    sshclient = paramiko.SSHClient()

    # 2. enable policy to support hosts not on the remote server
    #    AutoAddPolicy - policy for automatically adding the hostname and new
    #    host key to the local HostKeys object, and saving it.
    #
    #    set_missing_host_key_policy - set policy to use when connecting to
    #    servers without a known host key.
    #
    sshclient.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    # 3. load the local system host keys
    #    load host keys from a system (read-only) file.
    #    this method can be called multiple times. Each new set of host k
    #    will be merged with the existing set (new replacing old if there
    #    conflicts). if filename is left as None, an attempt will be made to
    #    read keys from the user's local "known hosts" file, as used by C
    #    and no exception will be raised if the file can't be read.
    sshclient.load_system_host_keys()
```

Links for Reference

- Paramiko Documentation
- Paramiko SSH Client

Functions

A function is a **reusable user-defined block of code** that can be called on to perform a certain action and output certain values. Instead of rewriting the same lines of code in a program, you can use a function to execute the same steps several times. A program can use many functions.

```
3   def driveToWork():
4           open garage
5           turn on car
6           reverse car
7           go forward
8           turn left on Main Street
9           turn right on Clark Avenue
10          ...
15          turn left on Parking Street
16          park in garage
17          turn off car
```

```
30  wakeUp(self)
31  shower
32  clothes.putOn(self)
33  if late:
34          hadBreakfast = False
35  else:
36          hadBreakfast = True
37          breakfast = [cereal,coffee]
38  leave(house)
39  driveToWork()
```

A **try statement** tests the code to see if there are errors.

The **except statement** outlines what to do if errors are found.

At the end of a function, you can set **variables that are returned**. These variables can be used outside the function in the main body of code.

```
76      # 4. connect to the remote server
77      #    connect to an SSH server and authenticate to it. The server's host key
78      #    is checked against the system host keys and any local host keys. if the
79      #    server's hostname is not found in either set of host keys, the missing
80      #    host key policy is used. the default policy is to reject the key and
81      #    raise an SSHException.
82      #
83      # use the try except statements for exception handling
84      #
85      # https://docs.python.org/3/reference/compound_stmts.html#the-try-statement
86      #
87      try:
88          sshclient.connect(host, username=user, key_filename=key)
89          print("SSH Connection setup ... successful")
90      except:
91          print("SSH Connection setup ... NOT successful")
92          ret_value = False
93      # end of try except
94
95      # return the success boolean and sshclient object
96      #
97      return(ret_value, sshclient)
98  # END OF FUNCTION
```

Links for Reference

Try Statement

try except finally

The **try** statement tests code to see if there are any errors. The **except** statement outlines what to do if errors are found. The **finally** clause is used for any cleanup operations. It is executed before leaving the try statement whether or not an exception has occurred.

```
1   try:
2         checkout_counter.unload(cart)
3         checkout_counter.pay()
4   except:
5         go to next register
6         checkout_counter.unload(cart)
7         checkout_counter.pay()
8   finally:
9         leave store
10        driveHome()
```

This function sets up an SFTP session using the SSH session.

```
def OpenSFTP(sshclient):
    print("I am inside .... OpenSFTP() function")

    ret_value = True

    # open the sftp session using the SSH connection on the SSH server. it will
    # return a new SFTPClient session object, so save that to a local variable.
    #
    try:
        ftpclient = sshclient.open_sftp()
        print("SFTP Open .... successful")
    except:
        ret_value = False
        print("SFTP Open .... NOT successful")
    # try except

    return(ret_value, ftpclient)
# END OF FUNCTION
```

It returns the variables **ret_value** and **ftpclient**.

This function uses the SFTP connection to pull the file requested by the user.

The except statement can define what to do if a **specific error** happens.

```
def GetFile(ftpclient, filename):
    print("I am inside .... GetFile() function")

    # call functions on the sftp client to first check if the file exits on the
    # remote server and if it does then get it to the local machine.
    #
    # http://docs.paramiko.org/en/2.6/api/sftp.html
    #
    # exception handling in python
    #
    # https://docs.python.org/3.7/library/exceptions.html
    #
    try:
        ftpclient.stat(filename)
    except IOError as e:
        print("response from server ~", e)
        print("File does not exist on the remote server")
        return(False)
    # end try except

    ftpclient.get(filename, filename)
    print("SFTP success in getting file ...", filename)

    return(True)
# END OF FUNCTION
```

Links for Reference

Paramiko SFTP

Exception Handling in Python

This function is the main function of the program.

Make sure the user has entered the correct number of arguments on the command line

Enter the credentials to be able to access the remote server

```
def my_main():
    print("\n\n++-->> Execution START ... SFTP_TXT_REMOTE.py")
    print("++")

    # 1. get input from the user, the filename to be SFTP from the remote server
    #
    print("sys.argv ~", sys.argv)
    if len(sys.argv) != 2:
        print("who are you trying to fool? ... exiting!")
        exit()
    else:
        input_filename = sys.argv[1]
        print("input_filename ~", input_filename)
    # end of if else

    # we could get this as input from the user but it's ok to hardcode for now
    #
    HOST = "35.172.116.24"
    USER = "student"

    # this file is required to be in the working folder so, let's make sure
    #
    KEY = "dnp423_student.pem"
    if not os.path.isfile(KEY):
        print("RSA private key file missing ... exiting!")
        exit()
    else:
        print("RSA private key file ... found!")

    # 2. setup a SSH connection to the remote server
    #
    ssh_success, ssh_client = SetupSSHConnection(HOST, USER, KEY)
    print("SSH successful? ...", ssh_success)
```

```
215     # was the SSH connection setup successful, if not then let's clean up
216     #
217     if ssh_success:
218
219         # 3. setup an SFTP session using the SSH connection
220         #
221         sftp_success, ftp_client = OpenSFTP(ssh_client)
222
223         # was the SFTP session open successful, if not then let's clean up
224         #
225         if sftp_success:
226
227             # 4. get the file from the remote server
228             get_success = GetFile(ftp_client, input_filename)
229         #end of if
230
231     # end of if
232
233     # 5. cleanup the connection
234     #
235     CloseSSHConnection(ssh_client)
236
237     print("++")
238     print("++-->> Execution END ... SFTP_TXT_REMOTE.py\n\n")
239 # END OF FUNCTION
```

If the SSH session was set up successfully, call the **SFTP** function.

If the SFTP session was set up successfully, call the **GetFile** function.

Call the **CloseSSHConnection** function to close the connection.

This is the main program body

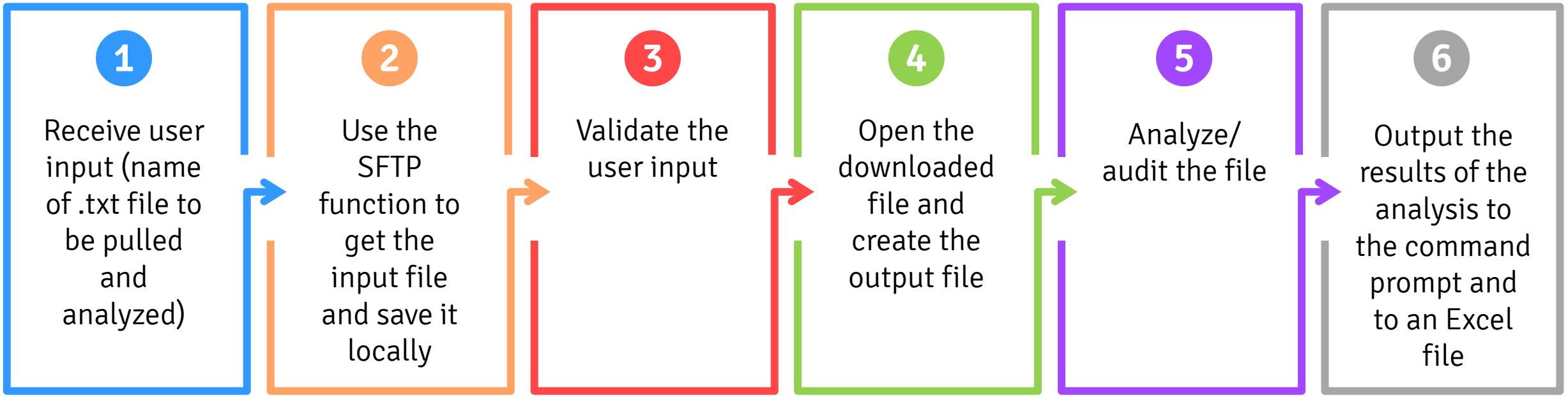
__name__ is a special variable. When a program is being run as a standalone program, the Python interpreter sets __name__ to "__main__". If a program is imported as a module, __name__ is set to the name of the program. **This if/else statement tells the user if the program is being run standalone or as an imported module.**

```
242 ##############################################################################
243 #
244 # This is REALLY the main body of my program
245 #
246 ##############################################################################
247 if __name__ == "__main__":
248     print("++-->> Name mangled variable ... __name__::", __name__)
249     print("++-->> SFTP_TXT_REMOTE.py program is being run STANDALONE!!\n\n")
250     my_main()
251 else:
252     print("++-->> Name mangled variable ... __name__::", __name__)
253     print("++-->> SFTP_TXT_REMOTE.py program is being called by SOMEONE!!\n\n")
254 # end of if else
255
256 # END OF FILE
```

Part 1: Build a Program using Secure FTP

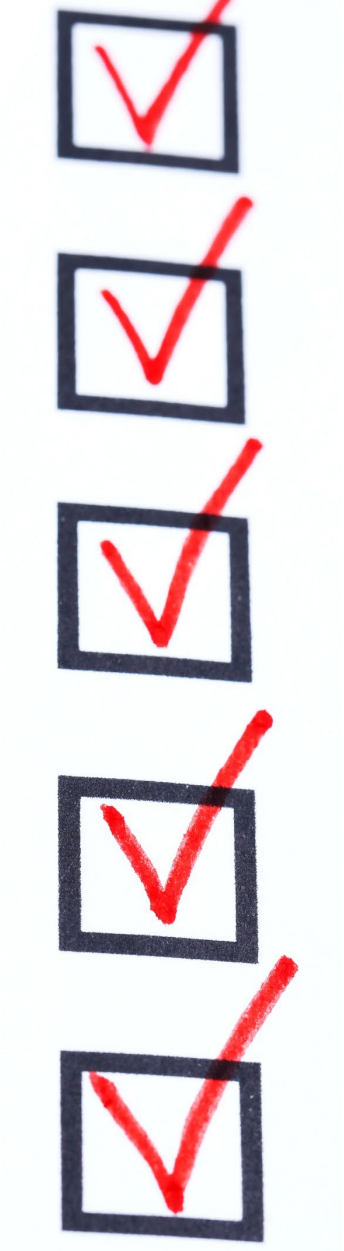
Part 2: Modify the Text Audit Program to Use SFTP

Flow of the Program



★ This is the only new part of this program – inserting the SFTP function.

Requirements



Import the SFTP program as a module within the text audit program

Reuse the analysis code from the previous example to perform the analysis on the file

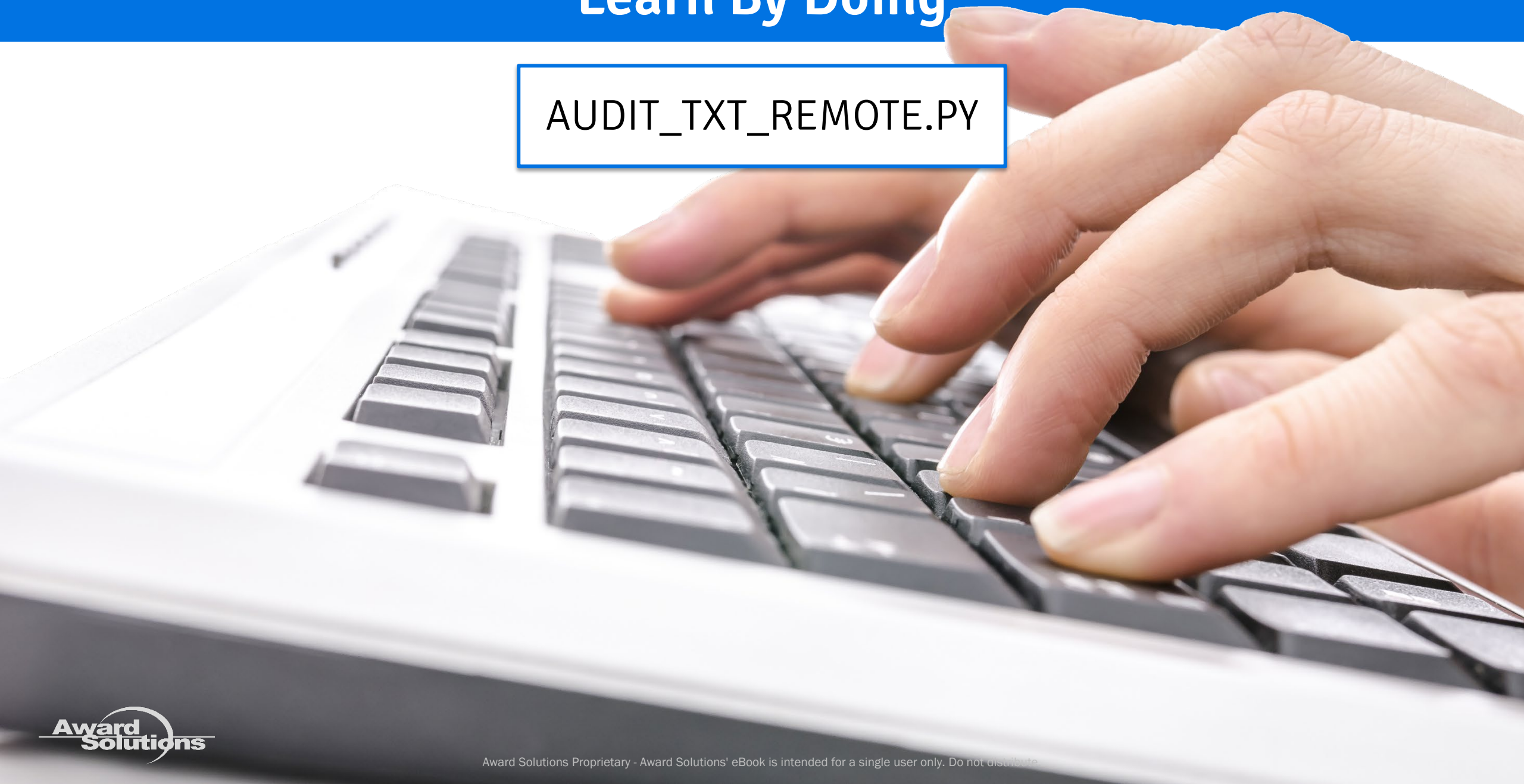
Pseudocode

This is the only new part of this program – inserting the SFTP function.

```
26  print("\n\n++-->> Execution START ... AUDIT_TXT_REMOTE.py")
27  print("++")
28
29  #####
30  # 1. GET input
31  #    get the input from the user command line
32  #    and make sure that the user has provided 2 input parameters at the command
33  #    prompt and the second parameter is the name of a text file that needs to
34  #    be audited
35
36  #####
37  # 2. Use SFTP function implemented in another module to get the input file from
38  #    the remote server and save it locally with the same name
39
40  #####
41  # 3. VALIDATE user input
42  #    validate whether the file name contained in a the input_filename variable
43  #    actually exists in the working folder - the folder where the program is
44  #    executing
45
46  #####
47  # 4. OPEN/CREATE files
48  #
49  # 4.1 now that we have done all the checks we are ready to open the input file.
50
51  # 4.2 next, we need to create the xlsx file to output the contents of the audit.
52
53  #####
54  # 5. AUDIT the file
55  #
56  # now that we have done the prep-work for the audit/analysis of the text file
57  # contents let's do the actual audit and output to the terminal and the xlsx
58  # file.
59
60  #####
61  # 6. OUTPUT the contents and the result of the audit
62
63  #####
64  # 7. CLEANUP and end the program
65
66  print("++")
67  print("++-->> Execution END ... AUDIT_TXT_REMOTE.py\n\n")
68
```

Learn By Doing

AUDIT_TXT_REMOTE.PY



```
25  # the import system of python
26  # https://docs.python.org/3/reference/import.html
27  #
28  import os
29  import sys
30  import openpyxl
31
32  # import a created module
33  import SFTP_TXT_REMOTE
34
```

Just like you can import modules written by others, you can import modules that you've written. **A module is any file containing Python definitions and statements.**

This module is the program that you wrote in Part 1.

You can call functions from a module using the dot operator (.). From this module, you could call:

SFTP_TXT_REMOTE.SetupSSHConnection()

SFTP_TXT_REMOTE.CloseSSHConnection()

SFTP_TXT_REMOTE.OpenSFTP()

SFTP_TXT_REMOTE.GetFile()

SFTP_TXT_REMOTE.my_main()

Recognize this code from the SFTP program? The credentials need to be in the main program so the SFTP function can access the remote server.

```
#####
# 2. Use SFTP function implemented in another module to get the input file from
#    the remote server and save it locally with the same name
    # we could get this as input from the user but it's ok to hardcode for now
    #
HOST = "35.172.116.24"
USER = "student"

# 2.1 this file is required to be in the working folder so, let's make sure
#
KEY = "dnp423_student.pem"
if not os.path.isfile(KEY):
    print("RSA private key file missing ... exiting!")
    exit()
else:
    print("RSA private key file ... found!")
```

```
130  # 2.2 setup a SSH connection to the remote server
131  #
132  ssh_success, ssh_client = SFTP_TXT_REMOTE.SetupSSHConnection(HOST, USER, KEY)
133  print("SSH successful? ...", ssh_success)
134
135  # was the SSH connection setup successful, if not then let's clean up
136  #
137  if ssh_success:
138      
139      # 2.3 setup an SFTP session using the SSH connection
140      #
141      sftp_success, ftp_client = SFTP_TXT_REMOTE.OpenSFTP(ssh_client)
142      
143      # was the SFTP session open successful, if not then let's clean up
144      #
145      if sftp_success:
146          
147          # 2.4 get the file from the remote server
148          get_success = SFTP_TXT_REMOTE.GetFile(ftp_client, input_filename)
149      #end of if
150      
151  # end of if
152
153  # 2.5 cleanup the connection
154  #
155  SFTP_TXT_REMOTE.CloseSSHConnection(ssh_client)
156
```

These lines call functions from the **SFTP_TXT_REMOTE** module and assign the return values to variables.

The rest of the program runs the same text audit actions on the file pulled from the remote server.

Websites and Documentation

1. Python: https://www.python.org/
2. Notepad++: https://notepad-plus-plus.org/
3. OpenPyXL: https://openpyxl.readthedocs.io/
4. Paramiko: https://www.paramiko.org/

Articles

1. Stephen Cass, “The 2018 Top Programming Languages,” IEEE Spectrum, July 31, 2018. https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages