python-tutorials Documentation

Release 1.0.2

Jose A. Lerma III

Nov 21, 2018

GETTING STARTED

Persistent Python practice produces prodigious productivity.

CHAPTER

ONE

INTRODUCTION

Python is a great language for getting things done quickly; however, a good deal of resources (mainly RAM (Random Access Memory)) are recommended. There are ways to incorporate C/C++ from within Python, but some may find it easier to port it over.

For more thorough intros, get lost in Python's Beginner's Guide or Wikipedia's Python page for a day or so and come back.

Looks up, then puts down Steam Controller

You're back? Alright, let's continue.

1.1 Installation

There are many ways to install and use Python depending on platform and IDE (Integrated Development Environment) (if any). These docs cover the methods I frequently use.

1.1.1 Windows

For Windows, I use but one editor: *Atom*; however, I give *PyCharm* an honorable mention. The Atom setup is lightweight and portable while the Pycharm setup is extensible and full-featured.

I have Pycharm on a Windows 10 Technical Preview VM (Virtual Machine), and it works well, but is quite bloated for a Windows VM running in Windows (Windows-ception?).

1.1.2 Linux

For Linux, I have two main IDEs: *Vim* and *PyCharm*. The Vim setup is lightweight and available without too much effort while the PyCharm setup is extensible and full-featured.

While I have a couple of Linux boxes (at the moment), I am very security minded when it comes to my Linux machines, so I prefer to run a Development VM of Linux on Windows. Excessive, yes, but taking snapshots, cloning, and reinstalling on VMs is easier than on physical machines.

I've read that some python bots can be run on a Raspberry Pi. I would like to tinker with this concept a bit, but I am concerned that Raspberry Pis do not have enough RAM, so I will be sticking with VMs until I can get more tests done.

1.1.3 Building Documentation

Note: Building the documentation is **not needed or recommended** unless contributing to the documentation. The latest version of the documentation is available at josealermaiii@github.io/python-tutorials or as a PDF in the source code. You have been warned.

Building the docs requires a few more pip packages:

- sphinx
- sphinxcontrib-napoleon
- sphinx-rtd-theme

Now, we can build the docs in HTML format:

```
cd absolute_path_here/python-tutorials/docs
make html
```

This will save the docs website in `../../python-tutorials-docs/`.

Building the PDF is even more involved. First, LaTeX must be installed on the OS. For example, in Ubuntu 18.04:

```
sudo apt-get install texlive-latex-recommended texlive-latex-extra texlive-fonts-
↪recommended texlive-xetex
```

Installing these dependencies is not recommended, if not needed, because they require > 330 MB of disk space.

We also install XeLaTeX, `texlive-xetex`, because some of the book corrections contain code snippets with unicode characters that are not supported by the default LaTeX engine.

Now, we can build the docs in PDF format:

```
cd absolute_path_here/python-tutorials/docs
make latexpdf
```

This will save the doc's PDF in `../manual.pdf`.

1.1.4 Disclaimer

Though covered by the MIT License, I reiterate: executable programs written from code on the Internet can end up doing bad things.

> Read and understand all code you copy and paste before running it.

1.2 Atom

From the Atom.io page:

> Atom is a text editor that's modern, approachable, yet hackable to the core—a tool you can customize to do anything but also use productively without ever touching a config file.

Personally, I **have** had to edit a config file to setup a proxy, so YMMV (Your Mileage May Vary).

Atom is also surprisingly full-featured (e.g. plugins, themes, file system browsing) given that it can be installed in a portable configuration and is multi-platform.

1.2.1 Windows Setup

While Atom is multi-platform, I only use it on Windows.

As aforementioned, I tend to use the zipped Atom files along with the PortableApps.com Platform to create a portable base environment. Next, I extract the zipped Atom files into `X:\PortableApps\Atom\`, as an example.

Then, you'll need to get the atom-runner package so that you can run the Python programs with an `ALT + R` key combo. However, `atom-runner` will not work if you have to input data from terminal, so you will need either the built-in Command Prompt or a PA.com portable enhancement like Console Portable. When you first open Atom, an `.atom` folder will be created in `%USERPROFILE%`, this folder will need to be moved into `X:\PortableApps\` to keep your settings.

As for Python, I get the embeddable zip files and extract them into `X:\PortableApps\CommonFiles\python3\` to continue with the portable theme. If you want different versions of Python, you can make different folders e.g. `python2.7`, `python3.6`, `python3.5`

Finally, the easiest way to get Atom to find your portable Python installation is to use a shebang on the first line of code `#! X:\PortableApps\CommonFiles\python3\python.exe`

1.3 Vim

Vim is a configurable, open source, and cross platform text editor that is an improvement of the vi editor in most Linux distros.

It has nifty things like syntax highlighting, colorization, and a scripting language to make your own plugins, etc.

1.3.1 Setup

As aforementioned, it is cross platform (and open source), so it can run on anything (even Potato). Personally, I prefer to use it on Linux only because it is usually in the default repository and has both syntax highlighting and colorization, which are a great improvement upon vi in CLI.

Linux

If your distro does not have vim in its default repo, then I fear you will have to compile from source code.

Windows

If you should want to use Vim on Windows, and not use gVim at PA.com, then both binaries and executables are available for you.

Other

Believe it or not, Vim is available on even more architectures: Amiga, OS2, Macintosh, Android, iOS, WindowsCE, Cygwin, and others.

1.3.2 Plugins/Scripts

Vim has a library of thousands of powerful scripts that are easy to make if it is missing something you want. Personally, I do not use any since I mainly use Vim as a quick, light editor.

1.4 PyCharm

PyCharm is very much like Python itself: quick to develop on, full-featured, and resource heavy. PyCharm is a true IDE: a console and debugger are all built-in. I especially like the PEP8 checks.

1.4.1 Linux Setup

Though Pycharm is multi-platform, I mainly use it on Linux.

Unless you are willing to pay for a license, you are probably going to want the free community edition. Download the `tar.gz` file wherever you like, then extract the `pycharm-community-20xx.x.x` folder. Therein, run the `pycharm-community-20xx.x.x/bin/pycharm.sh` file from within terminal.

Once setup is complete, on the menu bar, go to "Tools>Create Desktop Entry…" to make it easier to open later. **Do not delete** the `pycharm-community-20xx.x.x` folder because that is where it is running from.

Upgrades follow the same procedure, except that you can delete the previous `pycharm-community-20xx.x.x` version folder.

1.4.2 Python Binaries

Installing Python will depend on your Linux distro. Most will have some version of Python either built-in or available from the package manager. PyCharm can auto-detect and use these installed versions. The Ubuntu Setup of my ClashCallerBot is an example of how easy setting up Python can be.

However, if you are unlucky, you will have to download the source and compile it yourself.

1.4.3 Windows Setup

Windows installation is very straightforward:

- Install Python with the Python executable installer.
- Install PyCharm using the Community Edition executable installer.

Any extra packages or modules would have to be added, but most programs can be run with the base installations.

1.5 AutomateTheBoringStuffWithPython

You'll be seeing a lot of Al Sweigart's books because he provides them for free online at his website. Please consider donating to show your support.

Automate the Boring Stuff with Python is his iconic book for beginners and largely covers automating common computer tasks.

From file manipulation, spreadsheets, and PDFs to web scraping, e-mails, and texts - a little of everything is covered.

I use the `.epub` format of the book, so rather than *pages*, I provide *locations*.

1.5.1 AutomateTheBoringStuffWithPython Corrections

I don't expect to find many more, but I'll update this post if I do.

Note: It's an EPUB copy, published: *2016-01-14T10:12:21-08:00* Also, no page numbers, just reference numbers (refNum/949).

In Chapter 10, on reference number 368.7, paragraph 19.30, the code block:

```
>>> podBayDoorStatus = 'open'
>>> assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
>>> podBayDoorStatus = 'I\'m sorry, Dave. I\'m afraid I can't do that.''
>>> assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
```

should be:

```
>>> podBayDoorStatus = 'open'
>>> assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
>>> podBayDoorStatus = 'I\'m sorry, Dave. I\'m afraid I can\'t do that.'  # Changed
>>> assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
```

July 23, 2018 Update

In Chapter 11, on reference number 447.4, paragraph 20.247, the code block:

```
from selenium import webdriver
browser = webdriver.Firefox()
browser.get('http://inventwithpython.com')
try:
    elem = browser.find_element_by_class_name('bookcover')
    print('Found <%s> element with that class name!' % (elem.tag_name))
except:
    print('Was not able to find an element with that name.')
```

outputs *Was not able to find an element with that name.*

The following does give the intended output:

```
from selenium import webdriver
browser = webdriver.Firefox()
browser.get('http://inventwithpython.com')
```

(continues on next page)

(continued from previous page)

```
try:
    elem = browser.find_element_by_class_name('card-img-top')  # changed
    print('Found <%s> element with that class name!' % (elem.tag_name))
except:
    print('Was not able to find an element with that name.')
```

On reference number 448.7, paragraph 20.249, the code block:

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://inventwithpython.com')
>>> linkElem = browser.find_element_by_link_text('Read It Online')
>>> type(linkElem)
<class 'selenium.webdriver.remote.webelement.WebElement'>
>>> linkElem.click() # follows the "Read It Online" link
```

should be:

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://inventwithpython.com')
>>> linkElem = browser.find_element_by_link_text('Read Online for Free')  # changed
>>> type(linkElem)
<class 'selenium.webdriver.remote.webelement.WebElement'>
>>> linkElem.click() # follows the "Read Online for Free" link  # changed
```

On reference number 449.3, paragraph 20.252, the line:

> As long as Gmail hasn't changed the id of the Username and Password text fields since this book was published...

"Gmail" should be "Yahoo Mail" because of line `>>> browser.get('https://mail.yahoo.com')` in the code block

Aug. 5, 2018 Update

In Chapter 12, on reference number 459.8, paragraph 21.47, the codeblock:

```
>>> wb.get_sheet_names()
['Sheet1', 'Sheet2', 'Sheet3']
>>> sheet = wb.get_sheet_by_name('Sheet3')
>>> sheet
<Worksheet "Sheet3">
>>> type(sheet) <class 'openpyxl.worksheet.worksheet.Worksheet'>
>>> sheet.title
'Sheet3'
>>> anotherSheet = wb.get_active_sheet()
```

should be:

```
>>> wb.sheetnames  # changed
['Sheet1', 'Sheet2', 'Sheet3']
>>> sheet = wb['Sheet3']  # changed
```

(continues on next page)

(continued from previous page)

```
>>> sheet
<Worksheet "Sheet3">
>>> type(sheet) <class 'openpyxl.worksheet.worksheet.Worksheet'>
>>> sheet.title
'Sheet3'
>>> anotherSheet = wb.active  # changed
```

because those methods are now depreciated (using `OpenPyXL 2.5.5`).

Aug. 6, 2018 Update

In Chapter 12, on reference number 463.0, paragraph 21.56, the codeblock:

```
>>> sheet = wb.get_sheet_by_name('Sheet1')
>>> sheet.get_highest_row()
7
>>> sheet.get_highest_column()
3
```

should be:

```
>>> sheet = wb['Sheet1']  # changed
>>> sheet.max_row  # changed
7
>>> sheet.max_column  # changed
3
```

because those methods are also depreciated.

On reference number 463.6, paragraph 21.58, the codeblock:

```
>>> from openpyxl.cell import get_column_letter, column_index_from_string
--snip--  # omitted to save space
>>> sheet = wb.get_sheet_by_name('Sheet1')
>>> get_column_letter(sheet.get_highest_column())
'C'
```

should be:

```
>>> from openpyxl.utils import get_column_letter, column_index_from_string  # changed
--snip-- # omitted to save space
>>> sheet = wb['Sheet1']  # changed
>>> get_column_letter(sheet.max_column)  # changed
'C'
```

because the functions were relocated and methods depreciated. The lines with `openpyxl.cell` in the paragraphs above and below should also be changed. In paragraph 21.59, the line "method like get_highest_column() to get an integer" should be changed to "property like max_column to get an integer."

Aug. 7, 2018 Update

In Chapter 12, on reference number 465.0, paragraph 21.60 is another `>>> sheet = wb.get_sheet_by_name('Sheet1')` that ought to be `>>> sheet = wb['Sheet1']`.

On reference number 466.8, paragraph 21.64, the codeblock:

```
--snip--  # omitted to save space
>>> sheet = wb.get_active_sheet()
>>> sheet.columns[1]
(<Cell Sheet1.B1>, <Cell Sheet1.B2>, <Cell Sheet1.B3>, <Cell Sheet1.B4>,
<Cell Sheet1.B5>, <Cell Sheet1.B6>, <Cell Sheet1.B7>)
>>> for cellObj in sheet.columns[1]:
        print(cellObj.value)
```

outputs `TypeError: 'generator' object is not subscriptable`

The best way to fix it is debatable, but the easiest was to use the `list` function:

```
--snip--  # omitted to save space
>>> sheet = wb.active  # changed
>>> list(sheet.columns)[1]  # changed
(<Cell Sheet1.B1>, <Cell Sheet1.B2>, <Cell Sheet1.B3>, <Cell Sheet1.B4>,
<Cell Sheet1.B5>, <Cell Sheet1.B6>, <Cell Sheet1.B7>)
>>> for cellObj in list(sheet.columns)[1]:  # changed
        print(cellObj.value)
```

On reference number 468.0, paragraph 21.67 the list item `4. Call the get_active_sheet() or get_sheet_by_name() workbook method.` ought to be something like `4. Use the .active property or the ["UseThisSheet"] workbook key.`

On reference number 470.6, paragraph 21.90 the codeblock:

```
--snip--  # omitted to save space
  sheet = wb.get_sheet_by_name('Population by Census Tract')
    countyData = {}

    # TODO: Fill in countyData with each county's population and tracts.
    print('Reading rows...')
  for row in range(2, sheet.get_highest_row() + 1):
--snip-- # omitted to save space
```

ought to be:

```
--snip--  # omitted to save space
  sheet = wb['Population by Census Tract']  # changed
    countyData = {}

    # TODO: Fill in countyData with each county's population and tracts.
    print('Reading rows...')
  for row in range(2, sheet.max_row + 1):  # changed
```

because of depreciated methods. The codeblock on paragraph 21.96 ought to be updated as well.

Aug. 8, 2018 Update

In Chapter 12, on reference number 477.4, paragraph 21.111, the codeblock:

```
>>> wb.get_sheet_names()
['Sheet']
>>> sheet = wb.get_active_sheet()
>>> sheet.title
'Sheet'
>>> sheet.title = 'Spam Bacon Eggs Sheet'
>>> wb.get_sheet_names()
```

ought to be:

```
>>> wb.sheetnames  # changed
['Sheet']
>>> sheet = wb.active  # changed
>>> sheet.title
'Sheet'
>>> sheet.title = 'Spam Bacon Eggs Sheet'
>>> wb.sheetnames  # changed
```

In paragraph 21.113 (codeblock directly below) another `>>> sheet = wb.get_active_sheet()` ought to be `>>> sheet = wb.active`.

On reference number 478.6, paragraph 21.116, the codeblock:

```
>>> wb.get_sheet_names()
['Sheet']
>>> wb.create_sheet()
<Worksheet "Sheet1">
>>> wb.get_sheet_names()
['Sheet', 'Sheet1']
>>> wb.create_sheet(index=0, title='First Sheet')
<Worksheet "First Sheet">
>>> wb.get_sheet_names()
['First Sheet', 'Sheet', 'Sheet1']
>>> wb.create_sheet(index=2, title='Middle Sheet')
<Worksheet "Middle Sheet">
>>> wb.get_sheet_names()
['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']
```

ought to be:

```
>>> wb.sheetnames  # changed
['Sheet']
>>> wb.create_sheet()
<Worksheet "Sheet1">
>>> wb.sheetnames  # changed
['Sheet', 'Sheet1']
>>> wb.create_sheet(index=0, title='First Sheet')
<Worksheet "First Sheet">
>>> wb.sheetnames  # changed
['First Sheet', 'Sheet', 'Sheet1']
>>> wb.create_sheet(index=2, title='Middle Sheet')
<Worksheet "Middle Sheet">
>>> wb.sheetnames  # changed
```

In paragraph 21.118 (codeblock directly below):

```
>>> wb.get_sheet_names()
['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']
>>> wb.remove_sheet(wb.get_sheet_by_name('Middle Sheet'))
>>> wb.remove_sheet(wb.get_sheet_by_name('Sheet1'))
>>> wb.get_sheet_names()
```

ought to be

```
>>> wb.sheetnames  # changed
['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']
>>> wb.remove(wb['Middle Sheet'])  # changed
>>> wb.remove(wb['Sheet1'])  # changed
>>> wb.sheetnames  # changed
```

Aug. 11, 2018 Update

In paragraph 21.121 (codeblock directly below), and on reference number 483.6, paragraph 21.144 (updateProduce.py) are more `>>> sheet = wb.get_sheet_by_name('Sheet')` that should be `>>> sheet = wb['Sheet']`.

On reference number 484.8, paragraph 21.146 (updateProduce.py), the line `for rowNum in range(2, sheet.get_highest_row()): # skip the first row` ought to be `for rowNum in range(2, sheet.max_row): # skip the first row`.

On reference number 486.5, paragraph 21.158, the line:

> To customize font styles in cells, important, import the Font() and Style() functions from the openpyxl.styles module.

Unless, of course, that's an intended pun.

On reference number 486.8, paragraph 21.158, the codeblock:

```
  >>> import openpyxl
  >>> from openpyxl.styles import Font, Style
  >>> wb = openpyxl.Workbook()
  >>> sheet = wb.get_sheet_by_name('Sheet')
>>> italic24Font = Font(size=24, italic=True)
>>> styleObj = Style(font=italic24Font)
>>> sheet['A1'].style = styleObj
  >>> sheet['A1'] = 'Hello world!'
  >>> wb.save('styled.xlsx')
```

should be:

```
  >>> import openpyxl
  >>> from openpyxl.styles import Font, NamedStyle  # changed
  >>> wb = openpyxl.Workbook()
  >>> sheet = wb['Sheet']  # changed
>>> italic24Font = NamedStyle(name="italic24Font")  # changed
>>> italic24Font.font = Font(size=24, italic=True)  # changed
>>> sheet['A1'].style = italic24Font  # changed
  >>> sheet['A1'] = 'Hello world!'
  >>> wb.save('styled.xlsx')
```

because the `Style` class is now depreciated.

Aug. 12, 2018 Update

In Chapter 12, on reference number 488.9, paragraph 21.178, the codeblock:

```
>>> import openpyxl
>>> from openpyxl.styles import Font, Style
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_sheet_by_name('Sheet')
```

```
>>> fontObj1 = Font(name='Times New Roman', bold=True)
>>> styleObj1 = Style(font=fontObj1)
>>> sheet['A1'].style/styleObj
>>> sheet['A1'] = 'Bold Times New Roman'
```

```
>>> fontObj2 = Font(size=24, italic=True)
>>> styleObj2 = Style(font=fontObj2)
>>> sheet['B3'].style/styleObj
>>> sheet['B3'] = '24 pt Italic'
```

```
>>> wb.save('styles.xlsx')
```

should be:

```
>>> import openpyxl
>>> from openpyxl.styles import Font, NamedStyle  # changed
>>> wb = openpyxl.Workbook()
>>> sheet = wb['Sheet']  # changed
```

```
>>> fontObj1 = Font(name='Times New Roman', bold=True)
>>> styleObj1 = NamedStyle(name="styleObj1")  # changed
>>> styleObj1.font = fontObj1  # added
>>> sheet['A1'].style = styleObj1  # changed
>>> sheet['A1'] = 'Bold Times New Roman'
```

```
>>> fontObj2 = Font(size=24, italic=True)
>>> styleObj2 = NamedStyle(name="StyleObj2")  # changed
>>> styleObj2.font = fontObj2  # added
>>> sheet['B3'].style = styleObj2 # changed
>>> sheet['B3'] = '24 pt Italic'
```

```
>>> wb.save('styles.xlsx')
```

Aug. 13, 2018 Update

In Chapter 12, reference number 491.5, paragraphs 21.185 and 21.187 are more `>>> sheet = wb.get_active_sheet()` that should be `>>> sheet = wb.active`. However, the formula evaluation doesn't work for me:

```
>>> import openpyxl
>>> wbFormulas = openpyxl.load_workbook('writeFormula.xlsx')
>>> sheet = wbFormulas.active  # changed
>>> sheet['A3'].value
'=SUM(A1:A2)'
```

```
>>> wbDataOnly = openpyxl.load_workbook('writeFormula.xlsx', data_only=True)
>>> sheet = wbDataOnly.active  # changed
>>> sheet['A3'].value  # not working with LibreOffice 6.0.3.2
500
```

From what I've researched on openpyxl.load_workbook(),

> **data_only** controls whether cells with formulae have either the formula (default) or the value stored the last time Excel read the sheet.

TODO: can someone else confirm with another LibreOffice version?

Reference numbers 493.3, 495.0, 496.2, and 497.6 have more `>>> sheet = wb.get_active_sheet()` that should be `>>> sheet = wb.active`.

Aug. 17, 2018 Update

In Chapter 12, reference number 500.4, paragraph 21.234, the codeblock:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()
>>> for i in range(1, 11):          # create some data in column A
        sheet['A' + str(i)] = i
```

```
>>> refObj = openpyxl.charts.Reference(sheet, (1, 1), (10, 1))
```

```
>>> seriesObj = openpyxl.charts.Series(refObj, title='First series')
```

```
>>> chartObj = openpyxl.charts.BarChart()
>>> chartObj.append(seriesObj)
>>> chartObj.drawing.top = 50        # set the position
>>> chartObj.drawing.left = 100
>>> chartObj.drawing.width = 300     # set the size
>>> chartObj.drawing.height = 200
```

```
>>> sheet.add_chart(chartObj)
>>> wb.save('sampleChart.xlsx')
```

works slightly better as:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.active  # changed
>>> for i in range(1, 11):          # create some data in column A
        sheet['A' + str(i)] = i
```

```
>>> refObj = openpyxl.chart.Reference(sheet, min_row=1, min_col=1, max_row=10, max_
↪col=1)  # changed
```

```
>>> seriesObj = openpyxl.chart.Series(refObj, title='First series')  # changed FIXME:␣
↪Chart layout is wrong (LibreOffice 6.0.3.2)
```

```
>>> chartObj = openpyxl.chart.BarChart()  # changed
>>> chartObj.append(seriesObj)
>>> chartObj.anchor = "B3"  # set the position; changed
>>> chartObj.width = 7.94  # set the size (in centimeters, where 1 cm = 37.8 pixels);␣
↪changed
>>> chartObj.height = 5.29  # changed
```

```
>>> sheet.add_chart(chartObj)
>>> wb.save('sampleChart.xlsx')
```

but the layout of the chart is all wrong. TODO: can someone else confirm it works in Excel?

Aug. 19, 2018 Update

In Chapter 13 (I made it! Woot!), reference number 511.7, paragraph 22.13, the line:

> PyPDF2 uses a zero-based index for getting pages: The first page is page 0, the second is Introduction, and so on.

"Introduction" links to the introduction of the book. Maybe "page 1" was auto-referenced?

On reference number 513.2, paragraph 22.15, the codeblock:

```
>>> pdfReader.decrypt('rosebud')
 1
 >>> pageObj = pdfReader.getPage(0)
```

gave me an `IndexError`, but the following works:

```
 >>> pdfReader = PyPDF2.PdfFileReader(open("encrypted.pdf", "rb"))  # added
>>> pdfReader.decrypt('rosebud')
 1
 >>> pageObj = pdfReader.getPage(0)
```

Aug. 21, 2018 Update

In Chapter 13, reference number 524.8, paragraph 22.60, the codeblock:

```
 #! python3
 # combinePdfs.py - Combines all the PDFs in the current working directory into
 # into a single PDF

 import PyPDF2, os

 # Get all the PDF filenames.
 pdfFiles = []
```

(continues on next page)

(continued from previous page)

```
  for filename in os.listdir('.'):
      if filename.endswith('.pdf'):
        pdfFiles.append(filename)
pdfFiles.sort(key = str.lower)
```

should be:

```
  #! python3
  # combinePdfs.py - Combines all the PDFs in the current working directory into
  # a single PDF  # changed

  import PyPDF2, os

  # Get all the PDF filenames.
  pdfFiles = []
  for filename in os.listdir('.'):
      if filename.endswith('.pdf'):
        pdfFiles.append(filename)
pdfFiles.sort(key=str.lower)  # changed
```

Aug. 22, 2018 Update

In Chapter 13, reference number 531.0, paragraph 22.79, the codeblock:

```
>>> len(doc.paragraphs[1].runs)
 4
>>> doc.paragraphs[1].runs[0].text
 'A plain paragraph with some '
>>> doc.paragraphs[1].runs[1].text
 'bold'
>>> doc.paragraphs[1].runs[2].text
 ' and some '
>>> doc.paragraphs[1].runs[3].text
 'italic'
```

outputs the following in `LibreOffice 6.0.3.2` with `Python-Docx 0.8.7`:

```
>>> len(doc.paragraphs[1].runs)
 5    # changed
>>> doc.paragraphs[1].runs[0].text
 'A plain paragraph with'     # changed
>>> doc.paragraphs[1].runs[1].text
 ' some ' # changed
>>> doc.paragraphs[1].runs[2].text
 'bold'   # changed
>>> doc.paragraphs[1].runs[3].text
 ' and some '     # changed
>>> doc.paragraphs[1].runs[4].text    # added
 'italic'
```

TODO: can someone confirm in Word on Windows?

On reference number 540.1, paragraph 22.163, the codeblock:

```
--snip--  # omitted to save space
>>> doc.paragraphs[1].runs[0].style = 'QuoteChar'
>>> doc.paragraphs[1].runs[1].underline = True
>>> doc.paragraphs[1].runs[3].underline = True
>>> doc.save('restyled.docx')
```

gives a `UserWarning: style lookup by style_id is deprecated. Use style name as key instead. return self._get_style_id_from_style(self[style_name], style_type)` but the following fixes it:

```
--snip--  # omitted to save space
>>> doc.paragraphs[1].runs[0].style = 'Quote Char'  # changed for python-docx 0.8.7
>>> doc.paragraphs[1].runs[1].underline = True
>>> doc.paragraphs[1].runs[3].underline = True
>>> doc.save('restyled.docx')
```

Aug. 23, 2018 Update

In Chapter 13, reference number 540.1, paragraph 22.164, the line:

> We can see that it's simple to divide a paragraph into runs and access each run individiaully.

On reference number 546.9, paragraph 22.183, the codeblock:

```
>>> doc.paragraphs[0].runs[0].add_break(docx.text.WD_BREAK.PAGE)
 >>> doc.add_paragraph('This is on the second page!')
 <docx.text.Paragraph object at 0x00000000037855F8>
 >>> doc.save('twoPage.docx')
```

ought to be:

```
>>> doc.paragraphs[0].runs[0].add_break(docx.enum.text.WD_BREAK.PAGE)  # changed
 >>> doc.add_paragraph('This is on the second page!')
 <docx.text.Paragraph object at 0x00000000037855F8>
 >>> doc.save('twoPage.docx')
```

Aug. 31, 2018 Update

In Chapter 13, reference number 552.0, paragraph 22.228, the line:

> You should try both the uppercase and **lower-case** form of each word.

In Chapter 14, reference number 561.2, paragraph 23.33, the codeblock:

```
 >>> import csv
 >>> csvFile = open('example.tsv', 'w', newline='')
>>> csvWriter = csv.writer(csvFile, delimiter='\t', lineterminator='\n\n')
 >>> csvWriter.writerow(['apples', 'oranges', 'grapes'])
 24
 >>> csvWriter.writerow(['eggs', 'bacon', 'ham'])
 17
 >>> csvWriter.writerow(['spam', 'spam', 'spam', 'spam', 'spam', 'spam'])
 32
```

outputs:

```
  >>> import csv
  >>> csvFile = open('example.tsv', 'w', newline='')
>>> csvWriter = csv.writer(csvFile, delimiter='\t', lineterminator='\n\n')
  >>> csvWriter.writerow(['apples', 'oranges', 'grapes'])
  23  # changed
  >>> csvWriter.writerow(['eggs', 'bacon', 'ham'])
  16  # changed
  >>> csvWriter.writerow(['spam', 'spam', 'spam', 'spam', 'spam', 'spam'])
  31  # changed
```

Sept. 1, 2018 Update

In Chapter 14, reference number 565.5, paragraph 23.54, the codeblock:

```
#! python3
# removeCsvHeader.py - Removes the header from all CSV files in the current
# working directory.

--snip--
# Read the CSV file in (skipping first row).
csvRows = []
csvFileObj = open(csvFilename)
readerObj = csv.reader(csvFileObj)
for row in readerObj:
    if readerObj.line_num == 1:
        continue    # skip first row
    csvRows.append(row)
csvFileObj.close()

# TODO: Write out the CSV file.
```

needs to be indented to match the previous codeblock:

```
#! python3
# removeCsvHeader.py - Removes the header from all CSV files in the current
# working directory.

    --snip--
    print('Removing header from ' + csvFilename + '...')  # added

    # Read the CSV file in (skipping first row).
    csvRows = []
    csvFileObj = open(csvFilename)
    readerObj = csv.reader(csvFileObj)
    for row in readerObj:
        if readerObj.line_num == 1:
            continue    # skip first row
        csvRows.append(row)
    csvFileObj.close()

# TODO: Write out the CSV file.
```

On reference number 568.2, paragraph 23.58:

The CSV Writer object will write the list to a CSV file in headerRemoved using csvFilename (which we also used in the CSV reader). This will overwrite the original file.

I thought the original file won't be overwritten because the new file is in the `headerRemoved` folder? TODO: Can someone please confirm?

On reference number 575.4, paragraph 23.98, the link `http://api.openweathermap.org /data/2.5/forecast/daily?q=%3CLocation%3E&cnt=3` no longer works. The OpenWeatherMap.org API now needs an API key. So, sign up if you *really* want to run quickWeather.py.

Alternatively, the Weather.gov API (United States only, at the moment) does not require an API key (only a User Agent), but it will require one in the future.

Sept. 4, 2018 Update

In Chapter 14, reference number 582.0, paragraph 23.130, the codeblock:

```
for excelFile in os.listdir('.'):
    # Skip non-xlsx files, load the workbook object.
    for sheetName in wb.get_sheet_names():
        # Loop through every sheet in the workbook.
        sheet = wb.get_sheet_by_name(sheetName)

        # Create the CSV filename from the Excel filename and sheet title.
        # Create the csv.writer object for this CSV file.

        # Loop through every row in the sheet.
        for rowNum in range(1, sheet.get_highest_row() + 1):
            rowData = []    # append each cell to this list
            # Loop through each cell in the row.
            for colNum in range(1, sheet.get_highest_column() + 1):
                # Append each cell's data to rowData.

            # Write the rowData list to the CSV file.

        csvFile.close()
```

should be:

```
for excelFile in os.listdir('.'):
   # Skip non-xlsx files, load the workbook object.
   for sheetName in wb.sheetnames:  # changed
       # Loop through every sheet in the workbook.
       sheet = wb[sheetName]  # changed

       # Create the CSV filename from the Excel filename and sheet title.
       # Create the csv.writer object for this CSV file.

       # Loop through every row in the sheet.
       for rowNum in range(1, sheet.max_row + 1):  # changed
           rowData = []    # append each cell to this list
           # Loop through each cell in the row.
           for colNum in range(1, sheet.max_column + 1):  # changed
               # Append each cell's data to rowData.
```

(continues on next page)

(continued from previous page)

```
            # Write the rowData list to the CSV file.

        csvFile.close()
```

Sept. 5, 2018 Update

In Chapter 15, reference number 595.7, paragraph 24.42, the codeblock:

```
>>> datetime.datetime.fromtimestamp(1000000)
datetime.datetime(1970, 1, 12, 5, 46, 40)
>>> datetime.datetime.fromtimestamp(time.time())
datetime.datetime(2015, 2, 27, 11, 13, 0, 604980)
```

might need to be:

```
>>> import time  # added
>>> datetime.datetime.fromtimestamp(1000000)
datetime.datetime(1970, 1, 12, 5, 46, 40)
>>> datetime.datetime.fromtimestamp(time.time())
datetime.datetime(2015, 2, 27, 11, 13, 0, 604980)
```

In case IDLE was closed to write the stopwatch.py program.

Sept. 6, 2018 Update

In Chapter 15, reference number 598.0, paragraph 24.47, the str line in codeblock needs bolding:

```
--snip--  # omitted to save space
>>> str(delta)  # bold me, pls
'11 days, 10:09:08'
```

On reference number 599.5, paragraph 24.49, the line:

> Finally, passing the timedelta object to str() returns a string clearly explaning the duration.

Sept. 7, 2018 Update

In Chapter 15, reference number 612.3, paragraph 24.125, the line:

> To make sure the keyword argument sep=' & ' gets passed to print() in the new thread, we pass kwargs={'sep': '& '} to threading.Thread().

On reference number 616.0, paragraph 24.136 (multidownloadXkcd.py), the codeblock:

```
            --snip-- # omitted
            if comicElem == []:
                print('Could not find comic image.')
            else:
               comicUrl = comicElem[0].get('src')
                # Download the image.
```

(continues on next page)

(continued from previous page)

```
            print('Downloading image %s...' % (comicUrl))
        --snip-- # omitted
```

should be:

```
        --snip-- # omitted
        if comicElem == []:
            print('Could not find comic image.')
        else:
           comicUrl = 'http:' + comicElem[0].get('src')  # changed
            # Download the image.
            print('Downloading image %s...' % (comicUrl))
        --snip-- # omitted
```

On reference number 627.6, paragraph 24.161, the codeblock:

```
>>> subprocess.Popen(['C:\\python34\\python.exe', 'hello.py'])
<subprocess.Popen object at 0x000000000331CF28>
```

might need to be:

```
>>> subprocess.Popen(['C:\\python34\\python.exe', 'hello.py']).communicate()  # changed
<subprocess.Popen object at 0x000000000331CF28>
```

I could not get it to accept input without it in `Ubuntu 18.04`. TODO: Can someone confirm they got it to work in Windows?

Sept. 8, 2018 Update

In Chapter 15, reference number 631.7, paragraph 24.183 (countdown.py), the codeblock:

```
--snip--  # omitted
 timeLeft = 60
  while timeLeft > 0:
     print(timeLeft, end='')
     time.sleep(1)
--snip--  # omitted
```

may need to be:

```
--snip--  # omitted
 timeLeft = 60
  while timeLeft > 0:
     print(timeLeft)  # changed
     time.sleep(1)
--snip--  # omitted
```

It wouldn't print remaining time in `Python 3.6.5 (Ubuntu 18.04)` until the while loop finished. It seemed to wait until the line was done before printing it. TODO: Can someone else please confirm?

Sept. 14, 2018 Update

In Chapter 16, reference number 648.4, paragraph 25.52, the line:

Install imapclient and pyzmail from a Terminal window. Appendix A has steps on how to install third-party modules.

I had to install `pyzmail36` (possibly because I'm using `Python 3.6.5`). Appendix A may have to be updated.

Sept. 15, 2018 Update

In Chapter 16, reference number 658.7, paragraph 25.115, the lines:

```
imapObj.search(['ON 05-Jul-2015']). Returns every message sent on July 5, 2015.
imapObj.search(['SINCE 01-Jan-2015', 'BEFORE 01-Feb-2015', 'UNSEEN']). Returns every␣
↪message sent in January 2015
that is unread. (Note that this means on and after January 1 and up to but not including␣
↪February 1.)
imapObj.search(['SINCE 01-Jan-2015', 'FROM alice@example.com']). Returns every message␣
↪from alice@example.com sent
                                                                    since the start of 2015.
imapObj.search(['SINCE 01-Jan-2015', 'NOT FROM alice@example.com']). Returns every␣
↪message sent from everyone except
                                                                     alice@example.com␣
↪since the start of 2015.
imapObj.search(['OR FROM alice@example.com FROM bob@example.com']). Returns every␣
↪message ever sent from
                                                                    alice@example.com or␣
↪bob@example.com.
imapObj.search(['FROM alice@example.com', 'FROM bob@example.com']). Trick example! This␣
↪search will never return
                                                                    any messages,␣
↪because messages must match all
                                                                    search keywords.␣
↪Since there can be only one
                                                                    "from" address, it␣
↪is impossible for a message
                                                                    to be from both␣
↪alice@example.com and
                                                                    bob@example.com.
```

should be:

```
imapObj.search(['ON', '05-Jul-2015']). Returns every message sent on July 5, 2015.
imapObj.search(['SINCE', '01-Jan-2015', 'BEFORE', '01-Feb-2015', 'UNSEEN']). Returns␣
↪every message sent in January
                                                                              2015 that␣
↪is unread. (Note that this
                                                                              means on␣
↪and after January 1 and up to
                                                                              but not␣
↪including February 1.)
imapObj.search(['SINCE', '01-Jan-2015', 'FROM', 'alice@example.com']). Returns every␣
↪message from alice@example.com
                                                                          sent since the␣
↪start of 2015.
imapObj.search(['SINCE', '01-Jan-2015', 'NOT', 'FROM', 'alice@example.com']). Returns␣
↪every message sent from
```

(continues on next page)

(continued from previous page)

```
                                                                                everyone␣
↪except alice@example.com
                                                                                since the␣
↪start of 2015.
imapObj.search(['OR', 'FROM', 'alice@example.com', 'FROM', 'bob@example.com']). Returns␣
↪every message ever sent
                                                                                  from␣
↪alice@example.com or
                                                                                 ␣
↪bob@example.com.
imapObj.search(['FROM', 'alice@example.com', 'FROM', 'bob@example.com']). Trick example!␣
↪This search will never
                                                                                return any␣
↪messages, because messages
                                                                                must match all␣
↪search keywords. Since
                                                                                there can be␣
↪only one "from" address, it
                                                                                is impossible␣
↪for a message to be from
                                                                                both␣
↪alice@example.com and
                                                                                bob@example.
↪com.
```

because criteria should be a sequence of items. Plus, trying `imapObj.search(['SINCE 01-Jan-2015', 'NOT FROM alice@exmaple.com'])` outputs `imaplib.error: SEARCH command error: BAD [b'Error in IMAP command UID SEARCH: Unexpected string as search key: SINCE 01-Jan-2015 (0.001 + 0.088 + 0.087 secs).']`

Alternatively, `imapObj.search('SINCE "01-Jan-2015" NOT FROM "alice@exmaple.com"')` works, but isn't recommended according to the docs.

On reference number 664.6, paragraph 25.141, the line `>>> message = pyzmail.PyzMessage.factory(rawMessages[40041]['BODY[]'])` gave me a `KeyError` (even after using proper UIDs) that was only fixed by changing it to `>>> message = pyzmail.PyzMessage.factory(rawMessages[40041][b'BODY[]'])`

On reference number 668.9, paragraph 25.148, the line `>>> UIDs = imapObj.search(['ON 09-Jul-2015'])` should be `>>> UIDs = imapObj.search(['ON', '09-Jul-2015'])`

Sept. 16, 2018 Updates

In Chapter 16, reference number 674.0, paragraph 25.168 (sendDuesReminders.py), the codeblock:

```
import openpyxl, smtplib, sys

--snip--  # omitted
  sheet = wb.get_sheet_by_name('Sheet1')

  lastCol = sheet.get_highest_column()
  latestMonth = sheet.cell(row=1, column=lastCol).value
--snip--  # omitted
```

should be:

```
import openpyxl, smtplib, sys, datetime  # changed

--snip--  # omitted
 sheet = wb['Sheet1']  # changed

 lastCol = sheet.max_column  # changed
 latestMonth = sheet.cell(row=1, column=lastCol).value
  latestMonth = datetime.datetime.strftime(latestMonth, '%b %Y')  # added for␣
↪LibreOffice 6.0.3.2
--snip--  # omitted
```

Sept. 17, 2018 Update: In LibreOffice, `latestMonth = 2018-06-01 00:00:00`, so I had to use datetime to format it as `Jun 2018`. TODO: Can someone please confirm it works in Excel?

On reference number 676.3, paragraph 25.170, the line `for r in range(2, sheet.get_highest_row() + 1):` should be `for r in range(2, sheet.max_row + 1):`

On reference number 678.1, paragraph 25.174, the line `body = "Subject: %s dues unpaid.\nDear %s,\nRecords show that you have not paid dues for %s. Please make this payment as soon as possible. Thank you!'" % (latestMonth, name, latestMonth)` should be `body = "Subject: %s dues unpaid.\nDear %s,\nRecords show that you have not paid dues for %s. Please make this payment as soon as possible. Thank you!" % (latestMonth, name, latestMonth)`

Sept. 17, 2018 Update

In Chapter 16, reference number 682.8, paragraph 25.190, the codeblock:

```
>>> from twilio.rest import TwilioRestClient
 --snip--  # omitted
>>> twilioCli = TwilioRestClient(accountSID, authToken)
```

should be:

```
>>> from twilio.rest import Client  # changed
 --snip--  # omitted
>>> twilioCli = Client(accountSID, authToken)  # changed
```

because `TwilioRestClient` has been depreciated (using `twilio 6.16.4`).

On reference number 685.5, paragraph 25.195, the line `>>> updatedMessage = twilioCli.messages.get(message.sid)` should be `>>> updatedMessage = twilioCli.messages(message.sid).fetch()` because the attributes of `messages.get()` were changed.

Sept. 18, 2018 Update

In Chapter 16, reference number 687.8, paragraph 25.201 (textMyself.py), the codeblock:

```
 --snip--  # omitted
 from twilio.rest import TwilioRestClient

 def textmyself(message):
     twilioCli = TwilioRestClient(accountSID, authToken)
 --snip--  # omitted
```

should be:

```
 --snip--  # omitted
 from twilio.rest import Client  # changed

def textmyself(message):
    twilioCli = Client(accountSID, authToken)  # changed
 --snip--  # omitted
```

In paragraph 25.202, the line:

> It then defined textmyself() to take **on** argument , make a TwilioRestClient object , and call create() with the message you passed .

Sept. 27, 2018 Update

In Chapter 17, reference number 724.1, paragraph 26.122, the line `im = im.resize((width, height))` is over indented.

On reference number 734.5, paragraph 26.163, the codeblock:

```
--snip--  # omitted
>>> fontsFolder = 'FONT_FOLDER' # e.g. 'Library/Fonts'
 >>> arialFont = ImageFont.truetype(os.path.join(fontsFolder, 'arial.ttf'), 32)
 >>> draw.text((100, 150), 'Howdy', fill='gray', font=arialFont)
--snip--  # omitted
```

will need to be changed for those on Ubuntu, specifically:

```
--snip--  # omitted
>>> fontsFolder = '/usr/share/fonts/truetype' # e.g. 'Library/Fonts'  # modified
 >>> liberationFont = ImageFont.truetype(os.path.join(fontsFolder, '/liberation/
↪LiberationSerif-Regular.ttf'), 32)  # modified
 >>> draw.text((100, 150), 'Howdy', fill='gray', font=liberationFont)  # modified
--snip--  # omitted
```

However, **everyone** will have to modify it for their system.

Sept. 28, 2018 Update

In Chapter 17, reference number 738.6, paragraph 26.194, the line:

> **Other wise**, it should skip adding the logo.

Sept. 29, 2018 Update

In Chapter 17, reference number 739.4, paragraph 26.198, the codeblock:

```
#! python3 #
Import modules and write comments to describe this program.

--snip--  # omitted
```

may need to be:

```
#! python3
# Import modules and write comments to describe this program.

--snip--  # omitted
```

On reference number 740.0, paragraph 26.200, the line:

> For each of the guests listed in the guests.txt file from the resources at http://nostarch.com/automatestuff/, generate an image file with the guest name and some flowery decoration.

may need to be:

> For each of the guests listed in the guests.txt file from the resources at http://nostarch.com/automatestuff/, generate an image file with the **guest's** name and some flowery decoration.

I couldn't find the public domain flower image mentioned in the book, so I used this one.

Oct. 2, 2018 Update

For Chapter 18, if running `Ubuntu 18.04.1` in a VirtualBox virtual machine, mouse integration needs to be turned off so that the `pyautogui` module can control the mouse. Remember that the Host Key will need to be pressed to manually toggle keyboard/mouse capture.

Oct. 3, 2018 Update

In Chapter 18, reference number 764.0, paragraph 27.77, the codeblock:

```
#! python3
# mouseNow.py - Displays the mouse cursor's current position.
--snip--
        positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)
        pixelColor = pyautogui.screenshot().getpixel((x, y))
        positionStr += ' RGB: (' + str(pixelColor[0]).rjust(3)
        positionStr += ', ' + str(pixelColor[1]).rjust(3)
        positionStr += ', ' + str(pixelColor[2]).rjust(3) + ')'
        print(positionStr, end='')
--snip--
```

may need to be:

```
#! python3
# mouseNow.py - Displays the mouse cursor's current position.
import pyautogui, os  # changed
--snip--
        positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)
        pixelColor = pyautogui.screenshot().getpixel((x, y))
        positionStr += ' RGB: (' + str(pixelColor[0]).rjust(3)
        positionStr += ', ' + str(pixelColor[1]).rjust(3)
        positionStr += ', ' + str(pixelColor[2]).rjust(3) + ')'
        print(positionStr, end='')
        print('\b' * len(positionStr), end='', flush=True)
except KeyboardInterrupt:
    files = os.listdir('./')  # added
```

(continues on next page)

(continued from previous page)

```
    for file in files:  # added
        if file.startswith('.screenshot'):  # added
            os.remove(os.path.join('./', file))  # added
    print('\nDone.')
```

to cleanup all the `.screenshot###` files left behind in `Ubuntu 18.04`. This could be because the exception handler doesn't give `PyAutoGUI` a chance to do it.

Oct. 4, 2018 Update

In Chapter 18, reference number 765.5, paragraph 27.81, the line:

> … replacing **'submit. png'** with the filename of your screenshot:

Oct. 7, 2018 Update

In Chapter 18, reference number 781.5, paragraph 27.192, the line:

> … then mouse over the Name field to figure out its **the** x- and y-coordinates.

Setting up formFiller.py coordinates

To set up the coordinates for formFiller.py, you need to open a terminal window (or command prompt), run the mouseNow.py script, resize it to something small, keep it in the foreground, and hover over the maximized browser in the background as you note the mouseNow.py data.

As you enter data in the form, you may need to keep bringing back the mouseNow.py window into the foreground. For some reason, that wasn't explained clearly enough for me.

Tip: If "This is a required question" appears below the **Name** field, it will affect the coordinates of the **Submit** button.

On reference number 791.8, paragraph 27.213, the lines:

> … whether it has gotten **offtrack**. You can even give PyAutoGUI a **screen-shot** and …

On reference number 793.8, paragraph 27.236, the line:

> Your program will have to take **screen-shots** to guide…

1.6 CrackingCodesWithPython

You'll be seeing a lot of Al Sweigart's books because he provides them for free online at his website. Please consider donating to show your support.

Cracking Codes with Python is his latest release and largely covers how to use and compromise various ciphers with Python.

Granted, most of the ciphers are old enough to be broken with a Raspberry Pi, but the general idea is how they are implemented and what about them are easy to break.

For detailed answers to Practice Questions, check the No Starch Press Website.

1.6.1 CrackingCodesWithPython Corrections

Note: My PDF copy was created *12/1/2017 7:03:06PM* and was last modified *12/4/2017 5:30:14PM*

- Chapter 1 Practice Questions:

The answer for Practice Question 1, part b: Encrypt "GUILLOTINE: A machine which makes a Frenchman shrug his shoulders with good reason." with a key of 17 should be "XlZccfkZeV:NRN4rtyz5vN.yztyN4r2v0NrNW9v5ty4r5N0y9!xNyz0N0y6!3u v90N.z yNx66uN9vr065Q", not "bpdggjodiZ:RVR8vx349zRD34x3R8v6z.RvRa?z9x38v9R.3?B2R34.R.30B7yz?.RD4A3R200yR?zv.09U" (that's key of 21)

Scratch that, with SYMBOLS = "ABCDEFGHIJKLMNOPQRSTUVWXYZ", even the messages should be in all caps with totally different outputs, like the decryption in question 2.

Question 1 answers should be:

a. EQFMHIBXVSYW: EFPI XS TMGO AMXL IUYEP WOMPP E VMKLX-LERH TSGOIX SV E PIJX.

b. XLZCCFKZEV: R DRTYZEV NYZTY DRBVJ R WIVETYDRE JYILX YZJ JYFLCUVIJ NZKY XFFU IVRJFE.

c. DHKDZOT: TJPM DMMZQZMZIXZ OJRVMY HT YZDOT.

and Question 3 should be using all caps as well.

- On page 57, the final paragraph reads:

  "Just as in the reverse cipher in Chapter 5, …"

However, the reverse cipher was in chapter 4 because chapter 5 is the Caesar cipher!

- On page 84, end of the third-to-last paragraph:

  "(All the variables in the reverse cipher and Caesar cipher programs in Chapters 5 and 6, respectively, were global.)"

Chapter 6 was the Caesar cipher hacker program!

- On page 166, the fourth paragraph:

  Line 30 uses string interpolation to print the key currently being tested using string interpolation to provide feedback to the user.

- On page 236, the code block:

```
>>> letterMapping1 = simpleSubHacker.addLettersToMapping(letterMapping1, 'OLQIHXIRCKGNZ',
↪ candidates[0])
>>> letterMapping1
```

Should be

```
>>> simpleSubHacker.addLettersToMapping(letterMapping1, 'OLQIHXIRCKGNZ', candidates[0])
>>> letterMapping1
```

- On page 237, the code blocks:

```
>>> letterMapping1 = simpleSubHacker.addLettersToMapping(letterMapping1, 'OLQIHXIRCKGNZ',
↪ candidates[1])
>>> letterMapping1
```

and

```
>>> letterMapping2 = simpleSubHacker.getBlankCipherletterMapping()
>>> wordPat = makeWordPatterns.getWordPattern('PLQRZKBZB')
>>> candidates = wordPatterns.allPatterns[wordPat]
>>> candidates
['CONVERSES', 'INCREASES', 'PORTENDED', 'UNIVERSES']
>>> for candidate in candidates:
...     letterMapping2 = simpleSubHacker.addLettersToMapping(letterMapping2, 'PLQRZKBZB',
↪ candidate)
...
>>> letterMapping2
```

should be

```
>>> simpleSubHacker.addLettersToMapping(letterMapping1, 'OLQIHXIRCKGNZ', candidates[1])
>>> letterMapping1
```

and

```
>>> letterMapping2 = simpleSubHacker.getBlankCipherletterMapping()
>>> wordPat = makeWordPatterns.getWordPattern('PLQRZKBZB')
>>> candidates = wordPatterns.allPatterns[wordPat]
>>> candidates
['CONVERSES', 'INCREASES', 'PORTENDED', 'UNIVERSES']
>>> for candidate in candidates:
...     simpleSubHacker.addLettersToMapping(letterMapping2, 'PLQRZKBZB', candidate)
...
>>> letterMapping2
```

- On page 238, the code block:

```
>>> letterMapping3 = simpleSubHacker.getBlankCipherletterMapping()
>>> wordPat = makeWordPatterns.getWordPattern('MPBKSSIPLC')
>>> candidates = wordPatterns.allPatterns[wordPat]
>>> for i in range(len(candidates)):
...     letterMapping3 = simpleSubHacker.addLettersToMapping(letterMapping3, 'MPBKSSIPLC
↪', candidates[i])
...
>>> letterMapping3
```

should be

```
>>> letterMapping3 = simpleSubHacker.getBlankCipherletterMapping()
>>> wordPat = makeWordPatterns.getWordPattern('MPBKSSIPLC')
>>> candidates = wordPatterns.allPatterns[wordPat]
>>> for i in range(len(candidates)):
...     simpleSubHacker.addLettersToMapping(letterMapping3, 'MPBKSSIPLC', candidates[i])
...
>>> letterMapping3
```

May 14, 2018 Update

- On page 253, the code block:

```
>>> building = ''
>>> for c in 'Hello world!':
>>>     building += c
>>> print(building)
```

should be

```
>>> building = ''
>>> for c in 'Hello world!':
...     building += c
...
>>> print(building)
```

- On page 254, the code block:

```
>>> building = []
>>> for c in 'Hello world!':
>>>     building.append(c)
>>> building = ''.join(building)
>>> print(building)
```

should be

```
>>> building = []
>>> for c in 'Hello world!':
...     building.append(c)
...
>>> building = ''.join(building)
>>> print(building)
```

May 15, 2018 Update

- On page 260, the last line:

  Similarly, the letters that appear least often in the ciphertext are more likely to have been encrypted from to X, Q, and Z in plaintext.

May 18, 2018 Update

- On page 298, the code:

```
>>> set([1, 2, 3, 3, 4])
set([1, 2, 3, 4])
```

outputs

```
>>> set([1, 2, 3, 3, 4])
{1, 2, 3, 4}
```

for me, but that may be the interactive shell or OS I'm using (Ubuntu 16.04 with Python 3.5.2). TODO: Can anyone else confirm?

- On page 306, the code:

```
>>> def printStuff():
        print('Hello', end='\n')
        print('Howdy', end='')
        print('Greetings', end='XYZ')
        print('Goodbye')
>>> printStuff()
```

should be

```
>>> def printStuff():
...     print('Hello', end='\n')
...     print('Howdy', end='')
...     print('Greetings', end='XYZ')
...     print('Goodbye')
...
>>> printStuff()
```

May 19, 2018 Update

- On page 318, the code block:

```
>>> import secrets
>>> otp = ''
>>> for i in range(55):
        otp += secrets.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
>>> otp
```

should be

```
>>> import secrets
>>> otp = ''
>>> for i in range(55):
...     otp += secrets.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
>>> otp
```

I think. Ubuntu 16.04 LTS doesn't have Python 3.6 or above. TODO: Can someone confirm?

- On page 326, the code block:

```
>>> primeNum.isPrime(13)
True
```

should be

```
>>> primeNum.isPrime(13)
False
```

Here's the thing: `isPrime()` checks a number for divisibility by low prime numbers (which would make it not prime). Therefore, 13 is divisible by the low prime number 13 and is not prime by that definition.

You'd have to add something like:

```
if num in LOW_PRIMES:
    return True  # Low prime numbers are still prime numbers
```

to `isPrime()` to keep it from doing that.

May 20, 2018 Update

- On page 341 and 347, the code:

```
64. print('The private key is a %s and a %s digit number.' % (len(str(publicKey[0])),␣
↪len(str(publicKey[1]))))
```

should be

```
64. print('The private key is a %s and a %s digit number.' % (len(str(privateKey[0])),␣
↪len(str(privateKey[1]))))
```

1.7 wikibook

This is a set of examples from Wikibook's Non-Programmer's Tutorial for Python 3.

I like the concept of an open tutorial for users to learn from and add to so I went along with it and typed up all the relevant examples from all the chapters with code.

They are a great resource to follow along with and contain some notes I added.

1.8 Udacity

It is fantastic that classes can be taken for free, though a machine does all the grading. The only downside is there is little feedback, but that may be what the forums are for.

Included are my answers to the given problems. As I learn more about Python, I will go back and make corrections/improvements. Regardless, given these are tutorials, feedback is welcome.

1.8.1 CS101: Intro to Computer Science

These are sets of problems given in Udacity's CS101 Course. Unfortunately, `Python 2.x` is used in the course, so these problem sets may not be forwards compatible.

The goal of the CS101 class is to create an Internet search engine from scratch in Python. The final is creating a social network in Python. Although the basics of Python are covered, I still recommend reading a primer on Python programming to better understand how programs are written.

To that end, I recommend Cracking Codes with Python by Al Sweigart because it demonstrates in-depth usage of strings, lists, and dictionaries with full explanations in a short-form book.

1.9 CrackingCodesWithPython

1.9.1 CrackingCodesWithPython package

Subpackages

Submodules

CrackingCodesWithPython.Chapter13.PracticeQuestions module

CrackingCodesWithPython.Chapter13.cryptomath module

Module contents

CrackingCodesWithPython.Chapter14 package

Submodules

CrackingCodesWithPython.Chapter14.PracticeQuestions module

CrackingCodesWithPython.Chapter14.affineCipher module

CrackingCodesWithPython.Chapter14.affineKeyTest module

Module contents

CrackingCodesWithPython.Chapter15 package

Submodules

CrackingCodesWithPython.Chapter15.PracticeQuestions module

CrackingCodesWithPython.Chapter15.affineHacker module

Module contents

CrackingCodesWithPython.Chapter16 package

Submodules

CrackingCodesWithPython.Chapter16.PracticeQuestions module

CrackingCodesWithPython.Chapter16.simpleSubCipher module

Module contents

CrackingCodesWithPython.Chapter17 package

Submodules

CrackingCodesWithPython.Chapter17.PracticeQuestions module

CrackingCodesWithPython.Chapter17.makeWordPatterns module

CrackingCodesWithPython.Chapter17.simpleSubHacker module

CrackingCodesWithPython.Chapter17.wordPatterns module

Module contents

CrackingCodesWithPython.Chapter18 package

Submodules

CrackingCodesWithPython.Chapter18.PracticeQuestions module

CrackingCodesWithPython.Chapter18.stringTest module

CrackingCodesWithPython.Chapter18.vigenereCipher module

Module contents

CrackingCodesWithPython.Chapter19 package

Submodules

CrackingCodesWithPython.Chapter19.PracticeQuestions module

CrackingCodesWithPython.Chapter19.freqAnalysis module

Module contents

CrackingCodesWithPython.Chapter20 package

Submodules

CrackingCodesWithPython.Chapter20.PracticeQuestions module

CrackingCodesWithPython.Chapter20.vigenereDictionaryHacker module

CrackingCodesWithPython.Chapter20.vigenereHacker module

Module contents

CrackingCodesWithPython.Chapter22 package

Submodules

1.10 Index