python-tutorials Documentation

Release 1.2.0

Jose A. Lerma III

Dec 29, 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?).

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.

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`

1.1.2 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.

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.

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.

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.

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.

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.

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.

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.

1.1.3 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 AutomateTheBoringStuff

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*.

1.2.1 AutomateTheBoringStuff 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')
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
>>> 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 = []
  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
```

(continues on next page)

(continued from previous page)

```
>>> 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.

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

(continues on next page)

(continued from previous page)

```
    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.
            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
                                                                               everyone␣
↪except alice@example.com
                                                                               since the␣
↪start of 2015.
```

(continues on next page)

(continued from previous page)

```
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…

1.3 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.

1.3.1 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]))))
```

1.4 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.

1.5 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.

1.5.1 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.

1.6 books

1.6.1 AutomateTheBoringStuff package

Subpackages

AutomateTheBoringStuff.Ch01 package

Submodules

AutomateTheBoringStuff.Ch01.P1_basics module

Basics

This program performs basic Python instructions.

`AutomateTheBoringStuff.Ch01.P1_basics.main()`

AutomateTheBoringStuff.Ch01.P2_hello module

Hello

This program says hello and asks for my name.

`AutomateTheBoringStuff.Ch01.P2_hello.main()`

Module contents

AutomateTheBoringStuff.Ch02 package

Submodules

AutomateTheBoringStuff.Ch02.P01_vampire module

Vampire

This program checks name and age.

`AutomateTheBoringStuff.Ch02.P01_vampire.main()`

AutomateTheBoringStuff.Ch02.P02_vampire2 module

Vampire 2.0

This program checks name and age, but won't work as planned.

`AutomateTheBoringStuff.Ch02.P02_vampire2.main()`

AutomateTheBoringStuff.Ch02.P03_littleKid module

Little kid

This program checks name and age, but not as many ages.

`AutomateTheBoringStuff.Ch02.P03_littleKid.main()`

AutomateTheBoringStuff.Ch02.P04_yourName module

Your name

This program forces you to type your name

`AutomateTheBoringStuff.Ch02.P04_yourName.main()`

AutomateTheBoringStuff.Ch02.P05_infiniteloop module

Infinite loop

This program runs in an infinite loop (don't run unless you know how to stop it!).

Note:

- CTRL-C usually works

`AutomateTheBoringStuff.Ch02.P05_infiniteloop.main()`

AutomateTheBoringStuff.Ch02.P06_swordfish module

Swordfish

This program asks for a name and password.

`AutomateTheBoringStuff.Ch02.P06_swordfish.main()`

AutomateTheBoringStuff.Ch02.P07_fiveTimes module

Five times

This program runs five times.

`AutomateTheBoringStuff.Ch02.P07_fiveTimes.main()`

AutomateTheBoringStuff.Ch02.P08_busywork module

Busy work

This program adds numbers 0 to 100.

`AutomateTheBoringStuff.Ch02.P08_busywork.main()`

AutomateTheBoringStuff.Ch02.P09_fiveTimes2 module

Five times 2.0

This program also runs five times, but using a while loop.

`AutomateTheBoringStuff.Ch02.P09_fiveTimes2.main()`

AutomateTheBoringStuff.Ch02.P10_printRandom module

Print random

This program prints five random numbers using the `random` module.

`AutomateTheBoringStuff.Ch02.P10_printRandom.main()`

AutomateTheBoringStuff.Ch02.P11_exitExample module

Exit example

This program terminates itself when told to using the `sys` module.

`AutomateTheBoringStuff.Ch02.P11_exitExample.main()`

AutomateTheBoringStuff.Ch02.P12_yourName2 module

Your name 2.0

This program also forces you to type your name, but using a different way to exit the loop.

`AutomateTheBoringStuff.Ch02.P12_yourName2.main()`

Module contents

AutomateTheBoringStuff.Ch03 package

Subpackages

AutomateTheBoringStuff.Ch03.Projects package

Submodules

AutomateTheBoringStuff.Ch03.Projects.P1_makeCollatzSeq module

Make Collatz Sequence

This program makes a Collatz Sequence for a given number.

Write a function named *`collatz()`* that has one parameter named number. If number is even, then *`collatz()`* should print *number // 2* and return this value. If number is odd, then *`collatz()`* should print and return *3 * number + 1*.

Then write a program that lets the user type in an integer and that keeps calling *`collatz()`* on that number until the function returns the value *1*.

Example

```
Enter number:
3
10
5
16
8
4
2
1
```

AutomateTheBoringStuff.Ch03.Projects.P1_makeCollatzSeq.**collatz**(*number: int*) → int
Collatz

If number is even, then return *number // 2*. If number is odd, then return *3 * number + 1*.

Parameters **number** – Integer to generate a Collatz conjecture term for.

Returns Integer that is either a quotient or a product and sum.

AutomateTheBoringStuff.Ch03.Projects.P1_makeCollatzSeq.**main**()

AutomateTheBoringStuff.Ch03.Projects.P2_inputValidation module

Input validation

This program adds input validation to `P01_make_collatz_seq`

Add try and except statements to the previous project to detect whether the user types in a noninteger string. Normally, the `int` function will raise a `ValueError` error if it is passed a noninteger string, as in *int('puppy')*. In the except clause, print a message to the user saying they must enter an integer.

AutomateTheBoringStuff.Ch03.Projects.P2_inputValidation.**main**()

Module contents

Submodules

AutomateTheBoringStuff.Ch03.P01_helloFunc module

Hello function

This program uses a function to say hello.

AutomateTheBoringStuff.Ch03.P01_helloFunc.**hello**() → None
Hello

Prints hello three different ways.

Returns None. Only prints three statements.

AutomateTheBoringStuff.Ch03.P01_helloFunc.**main**()

AutomateTheBoringStuff.Ch03.P02_helloFunc2 module

Hello function 2.0

This program uses a function with an input argument to say hello to Alice and Bob.

AutomateTheBoringStuff.Ch03.P02_helloFunc2.**hello**(*name: str*) → None
Hello

Says hello to a given name.

Parameters **name** – String containing name of person to say hello to.

Returns None. Prints out a statement.

AutomateTheBoringStuff.Ch03.P02_helloFunc2.**main**()

AutomateTheBoringStuff.Ch03.P03_magic8Ball module

Magic 8 ball

This program answers your questions with a function that knows all.

AutomateTheBoringStuff.Ch03.P03_magic8Ball.**getAnswer**(*answerNumber: int*) → str
Get answer

Uses *if* ... *elif* sequence to return a response based on an inputted number.

Parameters **answerNumber** – Any integer between 1 and 9.

Returns String containing a response based on the given number.

AutomateTheBoringStuff.Ch03.P03_magic8Ball.**main**()

AutomateTheBoringStuff.Ch03.P04_sameName module

Same name

This program uses the same variable name throughout.

AutomateTheBoringStuff.Ch03.P04_sameName.**eggs**
String denoting global variable.

Type str

Note: Not recommended, but possible.

AutomateTheBoringStuff.Ch03.P04_sameName.**bacon**() → None
Bacon

Prints its local variable called eggs and also calls *spam()*.

Returns None. Prints local variables.

AutomateTheBoringStuff.Ch03.P04_sameName.**main**()

AutomateTheBoringStuff.Ch03.P04_sameName.**spam**() → None
Spam

Prints its local variable called eggs.

Returns None. Prints the local variable.

AutomateTheBoringStuff.Ch03.P05_sameName2 module

Same name 2.0

This program has only one variable.

AutomateTheBoringStuff.Ch03.P05_sameName2.**eggs**
String denoting global variable.

Type str

AutomateTheBoringStuff.Ch03.P05_sameName2.**main**()

AutomateTheBoringStuff.Ch03.P05_sameName2.**spam**() → None
Spam

Reassigns global variable called eggs.

Returns None.

AutomateTheBoringStuff.Ch03.P06_sameName3 module

Same name 3.0

This program demonstrates global and local variable rules.

AutomateTheBoringStuff.Ch03.P06_sameName3.**eggs**
Integer defining the answer to all life, universe, and everything.

Type int

AutomateTheBoringStuff.Ch03.P06_sameName3.**bacon**() → None
Bacon

Assigns a local variable called eggs.

Returns None.

AutomateTheBoringStuff.Ch03.P06_sameName3.**ham**() → None
Ham

Prints global variable called eggs.

Returns None. Prints global variable, eggs.

AutomateTheBoringStuff.Ch03.P06_sameName3.**main**()

AutomateTheBoringStuff.Ch03.P06_sameName3.**spam**() → None
Spam

Reassigns the global variable called eggs.

Returns None.

AutomateTheBoringStuff.Ch03.P07_sameName4 module

Same name 4.0

This program produces an error trying to print a local global variable while assigning a local variable with the same name.

AutomateTheBoringStuff.Ch03.P07_sameName4.**eggs**
String denoting global variable.

Type str

AutomateTheBoringStuff.Ch03.P07_sameName4.**main**()

AutomateTheBoringStuff.Ch03.P07_sameName4.**spam**() → None
Spam

Attempts to print global variable, eggs, while assigning local variable, eggs.

Returns None.

AutomateTheBoringStuff.Ch03.P08_zeroDivide module

Zero Divide

This program also produces an error by dividing by zero.

AutomateTheBoringStuff.Ch03.P08_zeroDivide.**main**()

AutomateTheBoringStuff.Ch03.P08_zeroDivide.**spam**(*divideBy: int*) → float
Spam

Divides integer 42 by given integer.

Parameters **divideBy** – Integer to divide 42 by.

Returns Float result of 42 divided by given integer.

AutomateTheBoringStuff.Ch03.P09_zeroDivide2 module

Zero Divide 2.0

This program handles a ZeroDivisionError.

AutomateTheBoringStuff.Ch03.P09_zeroDivide2.**main**()

AutomateTheBoringStuff.Ch03.P09_zeroDivide2.**spam**(*divideBy: int*) → float
Spam

Divides integer 42 by given integer, but also handles a ZeroDivisionError.

Parameters **divideBy** – Integer to divide 42 by.

Returns Float result of 42 divided by given integer.

AutomateTheBoringStuff.Ch03.P10_zeroDivide3 module

Zero divide 3.0

This program also handles a ZeroDivisionError, but in the *main()* function.

AutomateTheBoringStuff.Ch03.P10_zeroDivide3.**main**()

AutomateTheBoringStuff.Ch03.P10_zeroDivide3.**spam**(*divideBy: int*) → float
Spam

Divides integer 42 by given integer.

Parameters **divideBy** – Integer to divide 42 by.

Returns Float result of 42 divided by given integer.

AutomateTheBoringStuff.Ch03.P11_guessTheNumber module

Guess the number

This is a guess the number game.

Note: Numbers between 1 and 20.

`AutomateTheBoringStuff.Ch03.P11_guessTheNumber.main()`

Module contents

AutomateTheBoringStuff.Ch04 package

Subpackages

AutomateTheBoringStuff.Ch04.Projects package

Submodules

AutomateTheBoringStuff.Ch04.Projects.P1_commaCode module

Comma code

This program converts a list to a comma separated string.

Write a function, *to_string()* that takes a list value as an argument and returns a string with all the items separated by a comma and a space, with *and* inserted before the last item.

Example

```
>>> from pythontutorials.books.AutomateTheBoringStuff.Ch04.Projects.P01_comma_code␣
↪import to_string
>>> spam = ['apples', 'bananas', 'tofu', 'cats']
>>> to_string(spam)
'apples, bananas, tofu, and cats'
```

But your function should be able to work with any list value passed to it.

`AutomateTheBoringStuff.Ch04.Projects.P1_commaCode.main()`

`AutomateTheBoringStuff.Ch04.Projects.P1_commaCode.to_string`(*input_list: list*) → str

To string

Converts elements in `list` to comma-separated `str`.

Parameters `input_list` – List to convert into a string.

Returns String with each element in the list separated by a comma and a space with *and* inserted before the last element.

AutomateTheBoringStuff.Ch04.Projects.P2_charPicGrid module

Character Picture Grid

This program converts a matrix to an image.

Say you have a list of lists where each value in the inner lists is a one-character string, like this:

```
grid = [['.', '.', '.', '.', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['.', 'O', 'O', 'O', 'O', 'O'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.']]
```

You can think of *grid[x][y]* as being the character at the x- and y-coordinates of a "picture" drawn with text characters. The *(0, 0)* origin will be in the upper-left corner, the x-coordinates increase going right, and the y-coordinates increase going down.

Copy the previous grid value, and write a function, *matrix_to_pic()*, that uses it to print the image:

```
..OO.OO..
.OOOOOOO.
.OOOOOOO.
..OOOOO..
...OOO...
....O....
```

AutomateTheBoringStuff.Ch04.Projects.P2_charPicGrid.**main**()

AutomateTheBoringStuff.Ch04.Projects.P2_charPicGrid.**matrix_to_pic**(*matrix: list*) → None

Matrix to picture

Converts a matrix of lists with one-character values into ASCII Art.

Parameters **matrix** – List with lists containing one-character elements.

Returns None. Prints the contents of the lists as ASCII Art

Module contents

Submodules

AutomateTheBoringStuff.Ch04.P1_allMyCats1 module

All my cats 1.0

This program inefficiently showcases your cats by prompting for user input for each cat.

AutomateTheBoringStuff.Ch04.P1_allMyCats1.**main**()

AutomateTheBoringStuff.Ch04.P2_allMyCats2 module

All my cats 2.0

This program efficiently showcases your cats by using a list to store the cat's names and a while loop to prompt for them. Then, using a for loop on the list to print them all.

AutomateTheBoringStuff.Ch04.P2_allMyCats2.**main**()

AutomateTheBoringStuff.Ch04.P3_myPets module

My pets

This program checks if a given pet's name is in a list of pet names.

AutomateTheBoringStuff.Ch04.P3_myPets.**main**()

AutomateTheBoringStuff.Ch04.P4_magic8Ball2 module

Magic 8 Ball 2.0

This program more efficiently answers your questions by indexing a list with the answer strings.

AutomateTheBoringStuff.Ch04.P4_magic8Ball2.**main**()

AutomateTheBoringStuff.Ch04.P5_passingReference module

Passing reference

This program demonstrates how mutable data types are passed to functions by using the *eggs()* function to append a string to a list of integers.

AutomateTheBoringStuff.Ch04.P5_passingReference.**eggs**(*someParameter: list*) → None
Appends *'hello'* to a given list.

Parameters **someParameter** – List to append to.

Returns None.

AutomateTheBoringStuff.Ch04.P5_passingReference.**main**()

Module contents

AutomateTheBoringStuff.Ch05 package

Subpackages

AutomateTheBoringStuff.Ch05.Projects package

Submodules

AutomateTheBoringStuff.Ch05.Projects.P1_gameInventory module

Fantasy game inventory

This program models a player's inventory from a fantasy game.

You are creating a fantasy video game. The data structure to model the player's inventory will be a `dictionary` where the keys are `string` values describing the item in the inventory and the value is an `integer` value detailing how many of that item the player has.

For example, the dictionary value:

```
{'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}
```

means the player has 1 rope, 6 torches, 42 gold coins, and so on.

Write a function named *`displayInventory()`* that would take any possible "inventory" and display it like the following:

```
Inventory:
12 arrow
42 gold coin
1 rope
6 torch
1 dagger

Total number of items: 62
```

AutomateTheBoringStuff.Ch05.Projects.P1_gameInventory.**stuff**
Dictionary with item names as keys and their counts as values.

Type dict

AutomateTheBoringStuff.Ch05.Projects.P1_gameInventory.**displayInventory**(*inventory: dict*) → None
Display inventory

Displays each key in a given inventory dictionary.

Parameters **inventory** – Inventory dictionary to display.

Returns None. Prints out inventory.

AutomateTheBoringStuff.Ch05.Projects.P1_gameInventory.**main**()

AutomateTheBoringStuff.Ch05.Projects.P2_gameInventory module

Fantasy game inventory 2.0

This program models a player's inventory from a fantasy game with the ability to add to the inventory.

Imagine that a vanquished dragon's loot is represented as a `list` of `strings` like this:

```
dragonLoot = ['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby']
```

Write a function named *`addToInventory(inventory, addedItems)`*, where the inventory parameter is a `dictionary` representing the player's inventory (like in the `previous project`) and the *addedItems* parameter is a `list` like *dragonLoot*. The *`addToInventory()`* function should return a `dictionary` that represents the updated inventory.

Note: The *addedItems* list can contain multiples of the same item.

AutomateTheBoringStuff.Ch05.Projects.P2_gameInventory.addToInventory(*inventory: dict*, *addedItems: list*) → dict

Add to inventory

Adds given list of items to given dictionary inventory.

Parameters

- **inventory** – Dictionary inventory to add items to.
- **addedItems** – List of strings to add to inventory.

Note: If the item already exists in the dictionary, its count is incremented.

AutomateTheBoringStuff.Ch05.Projects.P2_gameInventory.main()

Module contents

Submodules

AutomateTheBoringStuff.Ch05.P1_birthdays module

Birthdays

This program makes a birthday database using a `dict`.

AutomateTheBoringStuff.Ch05.P1_birthdays.main()

AutomateTheBoringStuff.Ch05.P2_characterCount module

Character count

This program counts how often each character appears in a string.

AutomateTheBoringStuff.Ch05.P2_characterCount.main()

AutomateTheBoringStuff.Ch05.P3_prettyCharacterCount module

Pretty character count

This program also counts the number of times a character appears in a string but with a prettier output via the `pprint` module.

AutomateTheBoringStuff.Ch05.P3_prettyCharacterCount.main()

AutomateTheBoringStuff.Ch05.P4_ticTacToe module

Tic Tac Toe

This program plays a game of tic-tac-toe using a `dict` to store the board.

AutomateTheBoringStuff.Ch05.P4_ticTacToe.theBoard
Dictionary containing the current status of the tic-tac-toe board.

Type dict

Note:

- Space names are defined as follows:

```
top-L | top-M | top-R
------+-------+------
mid-L | mid-M | mid-R
------+-------+------
low-L | low-M | low-R
```

- Intended for two players.

`AutomateTheBoringStuff.Ch05.P4_ticTacToe.main()`

`AutomateTheBoringStuff.Ch05.P4_ticTacToe.printBoard(`*board: dict*`)` → None
Print board

Prints each row of a given tic-tac-toe board.

Parameters `board` – Dictionary containing space names as keys and contents as values.

Returns None. Prints rows of tic-tac-toe board.

AutomateTheBoringStuff.Ch05.P5_totalBrought module

Total brought

This program totals everything being brought to a picnic by storing the guest's names and items brought as a `dict` of `dicts`.

`AutomateTheBoringStuff.Ch05.P5_totalBrought.allGuests`
Dictionary of dictionaries with guest names as keys and values of dictionaries with items as keys and number of items as values.

Type dict

`AutomateTheBoringStuff.Ch05.P5_totalBrought.main()`

`AutomateTheBoringStuff.Ch05.P5_totalBrought.totalBrought(`*guests: dict*, *item: str*`)` → int
Total brought

Totals given item from given guest dictionary and returns result.

Parameters

- `guests` – Dictionary with guest's names and what they are bringing.
- `item` – Specific item in guest dictionary that is to be totaled.

Returns Integer total of given item that will be brought.

Module contents

AutomateTheBoringStuff.Ch06 package

Subpackages

AutomateTheBoringStuff.Ch06.Projects package

Submodules

AutomateTheBoringStuff.Ch06.Projects.P1_tablePrinter module

Table printer

This program displays a list of strings in a table.

Write a function named *print_table()* that takes a list of lists of strings and displays it in a well-organized table with each column right-justified. Assume that all the inner lists will contain the same number of strings.

For example, the value could look like this:

```
tableData = [['apples', 'oranges', 'cherries', 'banana'],
             ['Alice', 'Bob', 'Carol', 'David'],
             ['dogs', 'cats', 'moose', 'goose']]
```

Your *print_table()* function would print the following:

```
  apples Alice  dogs
 oranges   Bob  cats
cherries Carol moose
  banana David goose
```

AutomateTheBoringStuff.Ch06.Projects.P1_tablePrinter.**main**()

AutomateTheBoringStuff.Ch06.Projects.P1_tablePrinter.**print_table**(*matrix: list*) → None

Print table

Prints given matrix with right justification based on the longest word in each row.

Parameters **matrix** – List of lists containing strings.

Returns None. Prints out matrix as table.

Module contents

Submodules

AutomateTheBoringStuff.Ch06.P1_catnapping module

Cat napping

This program prints a message for Alice using a multi-line string with triple quotes.

AutomateTheBoringStuff.Ch06.P1_catnapping.**main**()

AutomateTheBoringStuff.Ch06.P2_great module

Great

This program asks how you're doing and responds by filtering input through an if-else statement.

`AutomateTheBoringStuff.Ch06.P2_great.main()`

AutomateTheBoringStuff.Ch06.P3_validateInput module

Validate input

This program asks for numerical age and alphanumeric password until they are valid inputs.

`AutomateTheBoringStuff.Ch06.P3_validateInput.main()`

AutomateTheBoringStuff.Ch06.P4_picnicTable module

Picnic table

This program prints a table of everything taken to a picnic using a dictionary with two different sets of left and right justification.

`AutomateTheBoringStuff.Ch06.P4_picnicTable.main()`

`AutomateTheBoringStuff.Ch06.P4_picnicTable.printPicnic`(*itemsDict: dict*, *leftWidth: int*, *rightWidth: int*) → None

Print picnic

Prints given picnic dictionary with given justification.

Parameters

- **itemsDict** – Dictionary with picnic items as keys and amounts as values.
- **leftWidth** – Left justification of keys.
- **rightWidth** – Right justification of values.

Returns None. Prints out dictionary with justification.

AutomateTheBoringStuff.Ch06.P5_pw module

Password locker

An insecure password locker program. This program stores different passwords as key value pairs in a dictionary. Keys are the name of the account and the values are the passwords for each account.

`AutomateTheBoringStuff.Ch06.P5_pw.main()` → None

P5_pw.py

If given account name is in the dictionary, the matching password is copied to the clipboard via `pyperclip`. Otherwise, an error is printed.

Returns None. Status or error messages are printed.

Note: If called without arguments, program exits with error message.

AutomateTheBoringStuff.Ch06.P6_bulletPointAdder module

Bullet point adder

Adds Wikipedia style bullet points to the start of each line of text on the clipboard.

`AutomateTheBoringStuff.Ch06.P6_bulletPointAdder.main()`

Module contents

AutomateTheBoringStuff.Ch07 package

Subpackages

AutomateTheBoringStuff.Ch07.Projects package

Submodules

AutomateTheBoringStuff.Ch07.Projects.P1_strongPwDetect module

Strong password detection

This program ensures passwords entered are "strong."

Write a function, *is_strong_pw()*, that uses regular expressions to make sure the password string it is passed is strong.

A strong password is defined as one that is at least eight characters long, contains both uppercase and lowercase characters, and has at least one digit.

Note: You may need to test the string against multiple regex patterns to validate its strength.

`AutomateTheBoringStuff.Ch07.Projects.P1_strongPwDetect.is_strong_pw(`*text: str*`)` → bool

Is strong password

Uses three `re` object patterns to check if a given text is at least 8 numbers and characters long, has at least one uppercase and lowercase character, and has at least one digit.

Parameters `text` – String containing password to test strength of.

Returns True if the given text matches the regex patterns, False otherwise.

`AutomateTheBoringStuff.Ch07.Projects.P1_strongPwDetect.main()`

AutomateTheBoringStuff.Ch07.Projects.P2_regexStrip module

Regex version of strip()

This program acts like `str.strip()` but using `re`.

Write a function, *regex_strip()*, that takes a string and does the same thing as the strip() string method.

If no other arguments are passed other than the string to strip, then whitespace characters will be removed from the beginning and end of the string. Otherwise, the characters specified in the second argument to the function will be removed from the string.

AutomateTheBoringStuff.Ch07.Projects.P2_regexStrip.**main**()

AutomateTheBoringStuff.Ch07.Projects.P2_regexStrip.**regex_strip**(*text: str*, *replace: str = ''*) → str

Regex strip

Implements the `str.strip()` method using `re` by removing the given replace string in the given text.

Parameters

- **text** – String with text to remove given replace string from.
- **replace** – String to remove from given text. Defaults to empty string.

Returns String with replace string removed from text string.

Module contents

Submodules

AutomateTheBoringStuff.Ch07.P1_isPhoneNumber module

Is phone number

This program demonstrates *isPhoneNumber()* which returns True if a string is a phone number and False if not.

However, *isPhoneNumber()* is not very efficient because it uses if statements and for loops to check 12-segment chunks for a phone number pattern.

AutomateTheBoringStuff.Ch07.P1_isPhoneNumber.**isPhoneNumber**(*text: str*) → bool

Is phone number

Function tests if given text is a phone number by checking a given text for two consecutive '###-' patterns followed by a '####' pattern using if statements and for loops.

Parameters **text** – String to check for a phone number pattern.

Returns True if the given string is a phone number, False otherwise.

AutomateTheBoringStuff.Ch07.P1_isPhoneNumber.**main**()

AutomateTheBoringStuff.Ch07.P2_phoneAndEmail module

Phone and email

Finds phone numbers and email addresses in the clipboard using `re` and `pyperclip`.

AutomateTheBoringStuff.Ch07.P2_phoneAndEmail.**phoneRegex**

Regular expression object representing a phone number pattern.

Type re.compile

AutomateTheBoringStuff.Ch07.P2_phoneAndEmail.**emailRegex**

Regular expression object representing an email pattern.

Type re.compile

AutomateTheBoringStuff.Ch07.P2_phoneAndEmail.**main**() → None

P2_phoneAndEmail.py

Checks clipboard text for phone numbers and emails using `re`. If found, matches are copied to the clipboard and printed to terminal.

Returns None. Prints and copies matches to clipboard or prints status message.

Module contents

AutomateTheBoringStuff.Ch08 package

Subpackages

AutomateTheBoringStuff.Ch08.Projects package

Submodules

AutomateTheBoringStuff.Ch08.Projects.P2_madlibs module

Mad libs

Create a Mad Libs program that reads in text files and lets the user add their own text anywhere the word ADJECTIVE, NOUN, ADVERB, or VERB appears in the text file.

For example, a text file may look like this:

```
The ADJECTIVE panda walked to the NOUN and then VERB. A nearby NOUN was
unaffected by these events.
```

The program would find these occurrences and prompt the user to replace them.

```
Enter an adjective:
silly
Enter a noun:
chandelier
Enter a verb:
screamed
Enter a noun:
pickup truck
```

The following text file would then be created:

```
The silly panda walked to the chandelier and then screamed. A nearby pickup
truck was unaffected by these events.
```

The results should be printed to the screen and saved to a new text file.

`AutomateTheBoringStuff.Ch08.Projects.P2_madlibs.main()`

AutomateTheBoringStuff.Ch08.Projects.P3_regexSearch module

Regex search

Write a program that opens all .txt files in a folder and searches for any line that matches a user-supplied regular expression.

The results should be printed to the screen.

AutomateTheBoringStuff.Ch08.Projects.P3_regexSearch.main()

Module contents

Submodules

AutomateTheBoringStuff.Ch08.P1_randomQuizGenerator module

Random quiz generator

Creates quizzes with questions and answers in random order, along with the answer key.

AutomateTheBoringStuff.Ch08.P1_randomQuizGenerator.main() → None
P1_randomQuizGenerator.py

Uses given dictionary with US states and their capitals to create 35 randomly generated quizzes with answer keys.

Returns None. Files are generated in same folder as program.

Module contents

AutomateTheBoringStuff.Ch09 package

Subpackages

AutomateTheBoringStuff.Ch09.Projects package

Submodules

AutomateTheBoringStuff.Ch09.Projects.P1_selectiveCopy module

Selective copy

Write a function, *selective_copy()*, that walks through a folder tree and searches for files with a certain file extension (such as .pdf or .jpg). Copy these files from whatever location they are in to a new folder.

Note: Defaults are to check parent directory for *.zip* files and put them in a folder called *new_folder*.

AutomateTheBoringStuff.Ch09.Projects.P1_selectiveCopy.main()

AutomateTheBoringStuff.Ch09.Projects.P1_selectiveCopy.selective_copy(*src_folder: str = None*, *ext: str = None*, *dest_folder: str = None*) → None

Selective copy

Searches for given extension in given source folder (and sub folders) then copies files to given destination folder.

Parameters

- **src_folder** – String with path to source folder. Relative paths are okay.
- **ext** – Extension to look for in source folder.
- **dest_folder** – String with name of destination folder.

Returns None. Prints status messages and makes copies within destination folder.

Raises AttributeError – If *src_folder*, *ext*, or *dest_folder* are not given.

Note: Destination folder is made inside source folder. Absolute path of source folder is automatically found.

AutomateTheBoringStuff.Ch09.Projects.P2_deleteBigFiles module

P2_deleteBigFiles.py

It's not uncommon for a few unneeded but humongous files or folders to take up the bulk of the space on your hard drive. If you're trying to free up room on your computer, you'll get the most bang for your buck by deleting the most massive of the unwanted files. But first you have to find them.

Write a program that walks through a folder tree and searches for exceptionally large files or folders—say, ones that have a file size of more than 100MB. (Remember, to get a file's size, you can use os.path.getsize() from the os module.) Print these files with their absolute path to the screen.

Note:

- `testfile.txt` was created by typing `truncate -s 101M testfile.txt` in terminal.
- Defaults to current working directory and > 100 MiB files

AutomateTheBoringStuff.Ch09.Projects.P2_deleteBigFiles.**delete_big_files**(*folder: str = None*, *filesize: str = None*) → None

Delete big files

Checks files in given folder (and subfolders) for given filesize. If greater, file is deleted.

Parameters

- **folder** – String with folder to check files of. Relative paths are okay.
- **filesize** – Maximum allowed size for files in given folder.

Returns None. Deletes files.

Raises AttributeError – If *folder* or *filesize* are not given.

Note: In debug mode - files to delete are printed to terminal. Uncomment after testing.

AutomateTheBoringStuff.Ch09.Projects.P2_deleteBigFiles.**main**()

AutomateTheBoringStuff.Ch09.Projects.P3_fillGaps module

P3_fillGaps.py

Write a program that finds all files with a given prefix, such as spam001.txt, spam002.txt, and so on, in a single folder and locates any gaps in the numbering (such as if there is a spam001.txt and spam003.txt but no spam002.txt).

Have the program rename all the later files to close this gap.

`AutomateTheBoringStuff.Ch09.Projects.P3_fillGaps.seqRegex`
Regular expression object used to group numbers in filename.

Type re.compile

`AutomateTheBoringStuff.Ch09.Projects.P3_fillGaps.fill_gaps`(*folder: str*) → None
Fill gaps

Fills gaps in file name numbering of a given folder.

Parameters `folder` – String containing path of folder to fill filename gaps. Relative paths are okay.

Returns None. Prints applicable error message and renames files.

Note: Running in debug mode. Files to be renamed are printed. Uncomment after testing to rename files.

`AutomateTheBoringStuff.Ch09.Projects.P3_fillGaps.get_filenames`(*folder: str*) → list
Get file names

Called by *fill_gaps()* to makes a list of all numbered file names in a given folder.

Breaks each file into the prefix (before numbering), numbering, and suffix (after numbering).

Parameters `folder` – String containing folder path to get file names of. Relative paths are okay.

Returns List of lists of strings with segmented names of files in alphanumerical order.

`AutomateTheBoringStuff.Ch09.Projects.P3_fillGaps.get_gap`(*numberlist: list*)
Get gap

Can take an integer list returned by *get_numbers()* and determine the missing sequence number.

Parameters `numberlist` – List of numbers to find a gap in.

Returns Missing number in the sequence or None if there is no gap.

`AutomateTheBoringStuff.Ch09.Projects.P3_fillGaps.get_numbers`(*files: list*) → list
Get numbers

Can take a list returned by *get_filenames()* and make an integer list of the numerical parts of the file names.

Parameters `files` – List of segmented file names.

Returns List of integers from file names in numerical order.

`AutomateTheBoringStuff.Ch09.Projects.P3_fillGaps.is_sequence`(*numberlist: list*) → bool
Is sequence

Can take a list returned by *get_numbers()* and determine if it is a sequence based on the property `list_length == (last_element - first_element + 1)`.

Parameters `numberlist` – List containing integers to check for a sequence.

Returns True if list contains a sequence of numbers, False otherwise.

`AutomateTheBoringStuff.Ch09.Projects.P3_fillGaps.main()`

AutomateTheBoringStuff.Ch09.Projects.P4_makeGaps module

P4_makeGaps.py

Write a program that finds all files with a given prefix, such as spam001.txt, spam002.txt, and so on, in a single folder and locates any gaps in the numbering (such as if there is a spam001.txt and spam003.txt but no spam002.txt).

Have the program rename all the later files to close this gap.

As an added challenge, write another program that can insert gaps into numbered files so that a new file can be added.

Note: By default, uses not provided `./testdir` and a gap of 2.

`AutomateTheBoringStuff.Ch09.Projects.P4_makeGaps.main()`

`AutomateTheBoringStuff.Ch09.Projects.P4_makeGaps.make_gaps`(*folder: str*, *gap: int*) → None

Make gaps

Makes given integer a gap in a sequence of numerical file names from a given folder.

Parameters

- `folder` – String containing path to folder to make gaps in file names. Relative paths are okay.
- `gap` – Integer number to make a gap.

Returns None. Prints applicable error message and renames files.

Example

If folder `test` has files `text1.txt`, `text2.txt`, and `text3.txt`. Then,

```
>>> make_gaps('test', 2)
```

would change the file names to `text1.txt`, `text3.txt`, and `text4.txt`.

Note: Running in debug mode. Files to be renamed are printed. Uncomment after testing to rename files.

Module contents

Submodules

AutomateTheBoringStuff.Ch09.P1_delete module

Delete

This program permanently deletes files ending with a .txt extension in the current directory using `os`.

Note: Demonstrates testing/debugging using comments.

```
AutomateTheBoringStuff.Ch09.P1_delete.main()
```

AutomateTheBoringStuff.Ch09.P2_tree module

Tree

This program walks a directory tree and prints the contents.

Note: Walks the `./delicious` directory.

```
AutomateTheBoringStuff.Ch09.P2_tree.main()
```

AutomateTheBoringStuff.Ch09.P3_zipfile module

ZIP file

This program manipulates a compressed file using `zipfile` and `os`.

Note: Works with provided ZIP file `example.zip`.

```
AutomateTheBoringStuff.Ch09.P3_zipfile.main()
```

AutomateTheBoringStuff.Ch09.P4_renameDates module

Rename dates

Renames filenames with American MM-DD-YYYY date format to European DD-MM-YYYY date format.

Uses `os` to get the list of files, `re` to find the files with the American date format, and `shutil` to rename them.

Note:

- Assumes only files with American date format are in the folder. May also match files with European date format.
- Using debug mode: Prints out files to be renamed. Uncomment to rename files.

```
AutomateTheBoringStuff.Ch09.P4_renameDates.main()
```

AutomateTheBoringStuff.Ch09.P5_backupToZip module

P5_backupToZip.py

Implements a function that copies an entire folder and its contents into a ZIP file whose filename increments.

Note: Uses provided `./delicious` folder as a demonstration.

`AutomateTheBoringStuff.Ch09.P5_backupToZip.backupToZip`(*folder: str*) → None
Backup to ZIP

Copies given folder and its contents into a ZIP file with the name `folder_#.zip`, where `folder` is the given folder and `#` is an incremented integer starting from `1`.

Parameters `folder` – String with path to folder that is to be archived. Function automatically converts to absolute path, so relative paths are okay.

Returns None. Prints status updates and creates ZIP file in same folder as given folder.

`AutomateTheBoringStuff.Ch09.P5_backupToZip.main`()

Module contents

AutomateTheBoringStuff.Ch10 package

Subpackages

AutomateTheBoringStuff.Ch10.Projects package

Submodules

AutomateTheBoringStuff.Ch10.Projects.debugCoinToss module

Debug coin toss

This program is a simple coin toss guessing game. The player gets two guesses.

`AutomateTheBoringStuff.Ch10.Projects.debugCoinToss.main`()

Module contents

Submodules

AutomateTheBoringStuff.Ch10.P1_boxPrint module

P1_boxPrint.py

This program prints out a box based on input sizes.

`AutomateTheBoringStuff.Ch10.P1_boxPrint.boxPrint`(*symbol: str*, *width: int*, *height: int*) → None
Box print

Prints a box of given width and height using the given symbol.

Parameters

- **symbol** – String to use to make sides of box.
- **width** – Integer width of box.
- **height** – Integer height of box.

Returns None. Prints box to specified dimensions.

Raises Exception – If *symbol* != 1 or if either *width* or *height* <= 2.

AutomateTheBoringStuff.Ch10.P1_boxPrint.main()

AutomateTheBoringStuff.Ch10.P2_errorExample module

Error Example

This program raises an exception and automatically displays the traceback.

AutomateTheBoringStuff.Ch10.P2_errorExample.bacon() → None
Bacon

Raises base exception.

Returns None.

Raises Exception – Always

AutomateTheBoringStuff.Ch10.P2_errorExample.main()

AutomateTheBoringStuff.Ch10.P2_errorExample.spam() → None
Spam

Calls *bacon()*.

Returns None.

AutomateTheBoringStuff.Ch10.P3_writeLogfile module

Write logfile

This program raises an exception but outputs traceback to a logfile.

Note: Default logfile is errorInfo.txt

AutomateTheBoringStuff.Ch10.P3_writeLogfile.main()

AutomateTheBoringStuff.Ch10.P4_podBayDoor module

Pod Bay Door

This program raises an AssertionError.

Note: Correction submitted for line 13

AutomateTheBoringStuff.Ch10.P4_podBayDoor.main()

AutomateTheBoringStuff.Ch10.P5_trafficLight module

Traffic light

This program emulates traffic lights at intersections with assertions.

AutomateTheBoringStuff.Ch10.P5_trafficLight.**market_2nd**
Stoplight at the corner of Market and 2nd streets with North-South face and East-West face as keys.

Type dict

AutomateTheBoringStuff.Ch10.P5_trafficLight.**mission_16th**
Stoplight at the corner of Mission and 16th streets with North-South face and East-West face as keys.

Type dict

AutomateTheBoringStuff.Ch10.P5_trafficLight.**main**()

AutomateTheBoringStuff.Ch10.P5_trafficLight.**switchLights**(*stoplight: dict*) → None
Switch lights

Takes stoplight dictionary and changes values: from 'green' to 'yellow', 'yellow' to 'red', and 'red' to 'green'.

Parameters **stoplight** – Dictonary representing stoplight with face directions as keys and status as values.

Returns None. Changes dictionary values.

Raises AssertionError – If none of the dictionary values are 'red'.

AutomateTheBoringStuff.Ch10.P6_factorialLog module

Factorial log

This program calculates factorial and logs debug messages.

AutomateTheBoringStuff.Ch10.P6_factorialLog.**factorial**(*n: int*) → int
Factorial

Calculates factorial of given number.

Parameters **n** – Integer number to calculate factorial.

Returns Factorial of given number.

AutomateTheBoringStuff.Ch10.P6_factorialLog.**main**()

AutomateTheBoringStuff.Ch10.P7_buggyAddingProgram module

Buggy adding program

This program adds three user inputted numbers and displays the sum.

Note: May not work as expected.

AutomateTheBoringStuff.Ch10.P7_buggyAddingProgram.**main**()

AutomateTheBoringStuff.Ch10.P8_coinFlip module

Coin flip

This program simulates flipping a coin 1000 times and prints the number of times it landed on heads.

```
AutomateTheBoringStuff.Ch10.P8_coinFlip.main()
```

Module contents

AutomateTheBoringStuff.Ch11 package

Subpackages

AutomateTheBoringStuff.Ch11.Projects package

Submodules

AutomateTheBoringStuff.Ch11.Projects.P1_commandLineEmailer module

Command line emailer

Write a program that takes an email address and string of text on the command line and then, using Selenium, logs into your email account and sends an email of the string to the provided address. (You might want to set up a separate email account for this program.)

Note:

- "string of text" must be within quotes, like shown
- "new" gmail layout is used with keyboard shortcuts enabled
- not fully tested because account could end up banned for going against TOS

```
AutomateTheBoringStuff.Ch11.Projects.P1_commandLineEmailer.main()
```

AutomateTheBoringStuff.Ch11.Projects.P2_imageDownloader module

Image downloader

Write a program that goes to a photo-sharing site like Flickr or Imgur, searches for a category of photos, and then downloads all the resulting images. You could write a program that works with any photo site that has a search feature.

Note: Many photo-sharing sites do not want direct download links easily accessible. Therefore, it is likely this script will not work in the future.

```
AutomateTheBoringStuff.Ch11.Projects.P2_imageDownloader.main()
```

AutomateTheBoringStuff.Ch11.Projects.P3_2048 module

2048

2048 is a simple game where you combine tiles by sliding them up, down, left, or right with the arrow keys. You can actually get a fairly high score by repeatedly sliding in an up, right, down, and left pattern over and over again.

Write a program that will open the game at https://gabrielecirulli.github.io/2048/ and keep sending up, right, down, and left keystrokes to automatically play the game.

class AutomateTheBoringStuff.Ch11.Projects.P3_2048.**ElementDoesNotHaveText**(*locator*, *text*)

Bases: object

Element does not have text

An expectation for checking that an element does not have specified text. Returns the WebElement if it doesn't have the specified text

locator

Used to find the element

AutomateTheBoringStuff.Ch11.Projects.P3_2048.**main**()

AutomateTheBoringStuff.Ch11.Projects.P4_linkVerification module

Link verification

Write a program that, given the URL of a web page, will attempt to download every linked page on the page. The program should flag any pages that have a 404 "Not Found" status code and print them out as broken links.

AutomateTheBoringStuff.Ch11.Projects.P4_linkVerification.**main**()

Module contents

Submodules

AutomateTheBoringStuff.Ch11.P1_mapIt module

Map it

Launches a map in the browser using an address from the command line or clipboard.

AutomateTheBoringStuff.Ch11.P1_mapIt.**main**()

AutomateTheBoringStuff.Ch11.P2_parseHTML module

Parse HTML

This program uses requests to fetch an HTML page and bs4 to parse it.

AutomateTheBoringStuff.Ch11.P2_parseHTML.**main**()

AutomateTheBoringStuff.Ch11.P3_lucky module

Lucky

Opens top Google search results for given query.

`AutomateTheBoringStuff.Ch11.P3_lucky.main()`

AutomateTheBoringStuff.Ch11.P4_downloadXkcd module

Download XKCD

Downloads every single XKCD comic.

`AutomateTheBoringStuff.Ch11.P4_downloadXkcd.main()`

AutomateTheBoringStuff.Ch11.P5_seleniumBrowser module

Selenium browser

This program uses selenium to parse and interact with websites

---

Note:

- geckodriver is needed for Linux
  - download from https://github.com/mozilla/geckodriver/releases
  - place in /usr/local/bin
  - more info https://github.com/SeleniumHQ/selenium/blob/master/py/docs/source/index.rst

---

`AutomateTheBoringStuff.Ch11.P5_seleniumBrowser.main()`

Module contents

AutomateTheBoringStuff.Ch12 package

Subpackages

AutomateTheBoringStuff.Ch12.Projects package

Submodules

AutomateTheBoringStuff.Ch12.Projects.P1_multiplicationTable module

Multiplication table

Create a program multiplicationTable.py that takes a number N from the command line and creates an N×N multiplication table in an Excel spreadsheet. Row 1 and column A should be used for labels and should be in bold.

`AutomateTheBoringStuff.Ch12.Projects.P1_multiplicationTable.main()`

AutomateTheBoringStuff.Ch12.Projects.P2_blankRowInserter module

Blank row inserter

Create a program blankRowInserter.py that takes two integers and a filename string as command line arguments. Let's call the first integer N and the second integer M. Starting at row N, the program should insert M blank rows into the spreadsheet.

`AutomateTheBoringStuff.Ch12.Projects.P2_blankRowInserter.main()`

AutomateTheBoringStuff.Ch12.Projects.P3_cellInverter module

Cell inverter

Write a program to invert the row and column of the cells in the spreadsheet. For example, the value at row 5, column 3 will be at row 3, column 5 (and vice versa). This should be done for all cells in the spreadsheet.

Note: Gets full file path from commandline arguments.

`AutomateTheBoringStuff.Ch12.Projects.P3_cellInverter.main()`

AutomateTheBoringStuff.Ch12.Projects.P4_textToExcel module

Text to Excel

Write a program to read in the contents of several text files (you can make the text files yourself) and insert those contents into a spreadsheet, with one line of text per row. The lines of the first text file will be in the cells of column A, the lines of the second text file will be in the cells of column B, and so on.

Note:

- Default folder is `./p4files/`
- Default output file is `textToExcel.xlsx`

`AutomateTheBoringStuff.Ch12.Projects.P4_textToExcel.main()`

AutomateTheBoringStuff.Ch12.Projects.P5_excelToText module

Excel to text

Write a program that performs the tasks of the previous program in reverse order: The program should open a spreadsheet and write the cells of column A into one text file, the cells of column B into another text file, and so on.

Note: Default output folder is `./p5files/`.

`AutomateTheBoringStuff.Ch12.Projects.P5_excelToText.main()`

Module contents

Submodules

AutomateTheBoringStuff.Ch12.P1_readingExcel module

Reading Excel

This program uses `openpyxl` to read Excel documents.

`AutomateTheBoringStuff.Ch12.P1_readingExcel.main()`

AutomateTheBoringStuff.Ch12.P2_readCensusExcel module

Read census Excel

Tabulates population and number of census tracts for each county.

Note:

- The censuspopdata.xlsx workbook can be downloaded from https://nostarch.com/automatestuff/

`AutomateTheBoringStuff.Ch12.P2_readCensusExcel.main()`

AutomateTheBoringStuff.Ch12.P3_writingExcel module

Writing Excel

This program uses `openpyxl` to write Excel documents.

`AutomateTheBoringStuff.Ch12.P3_writingExcel.main()`

AutomateTheBoringStuff.Ch12.P4_updateProduce module

Update produce

Corrects costs in produce sales spreadsheet.

Note:

- The produceSales.xlsx workbook can be downloaded from https://nostarch.com/automatestuff/

`AutomateTheBoringStuff.Ch12.P4_updateProduce.main()`

AutomateTheBoringStuff.Ch12.P5_stylingExcel module

Styling Excel

This program uses `openpyxl` to format Excel documents

`AutomateTheBoringStuff.Ch12.P5_stylingExcel.main()`

Module contents

AutomateTheBoringStuff.Ch13 package

Subpackages

AutomateTheBoringStuff.Ch13.Projects package

Submodules

AutomateTheBoringStuff.Ch13.Projects.P1_encryptPDFparanoia module

Encrypt PDF paranoia

Using `os.walk()`, write a script that will go through every PDF in a folder (and its subfolders) and encrypt the PDFs using a password provided on the command line. Save each encrypted PDF with an _encrypted.pdf suffix added to the original filename.

Before deleting the original file, have the program attempt to read and decrypt the file to ensure that it was encrypted correctly.

Notes

- Default folder is parent directory.
- Default suffix is '_encrypted.pdf'.
- Running in debug mode, uncomment to delete original file.

`AutomateTheBoringStuff.Ch13.Projects.P1_encryptPDFparanoia.main()`

AutomateTheBoringStuff.Ch13.Projects.P2_decryptPDFparanoia module

Decrypt PDF paranoia

Write a program that finds all encrypted PDFs in a folder (and its subfolders) and creates a decrypted copy of the PDF using a provided password. If the password is incorrect, the program should print a message to the user and continue to the next PDF.

Note:

- Default input folder is parent directory.
- Default output suffix is '_decrypted.pdf'.

`AutomateTheBoringStuff.Ch13.Projects.P2_decryptPDFparanoia.main()`

AutomateTheBoringStuff.Ch13.Projects.P3_invitations module

Invitations

Say you have a text file of guest names. This guests.txt file has one name per line, as follows:

```
Prof. Plum
Miss Scarlet
Col. Mustard
Al Sweigart
RoboCop
```

Write a program that would generate a Word document with custom invitations.

Notes

- Uses provided `guests.txt` file.
- Default output file is `invitations.docx`.

`AutomateTheBoringStuff.Ch13.Projects.P3_invitations.main()`

AutomateTheBoringStuff.Ch13.Projects.P4_PDFbreaker module

PDF breaker

Say you have an encrypted PDF that you have forgotten the password to, but you remember it was a single English word. Trying to guess your forgotten password is quite a boring task. Instead you can write a program that will decrypt the PDF by trying every possible English word until it finds one that works.

Create a list of word strings by reading dictionary.txt. Then loop over each word in this list, passing it to `PyPDF2.PdfFileReader.decrypt()`. If this method returns the integer 0, the password was wrong and your program should continue to the next password. If `PyPDF2.PdfFileReader.decrypt()` returns 1, then your program should break out of the loop and print the hacked password. You should try both the uppercase and lowercase form of each word.

Note:

- Dictionary text file can be downloaded from http://nostarch.com/automatestuff/
- Default input file is 'allminutes_encrypted.pdf' generated by *P3_combinePDFs* and *P1_encryptPDFparanoia*.

`AutomateTheBoringStuff.Ch13.Projects.P4_PDFbreaker.main()`

Module contents

Submodules

AutomateTheBoringStuff.Ch13.P1_readPDF module

Read PDF

This program uses PyPDF4 to read PDF files.

Note:

- Example PDFs can be downloaded from http://nostarch.com/automatestuff/

- Book uses PyPDF2; I'm an overachiever that uses PyPDF4

AutomateTheBoringStuff.Ch13.P1_readPDF.main()

AutomateTheBoringStuff.Ch13.P2_writePDF module

Write PDF

This program uses PyPDF4 to write PDF documents.

Note:

- Example PDFs can be downloaded from http://nostarch.com/automatestuff/
- Book uses PyPDF2; I'm an overachiever that uses PyPDF4

AutomateTheBoringStuff.Ch13.P2_writePDF.main()

AutomateTheBoringStuff.Ch13.P3_combinePDFs module

Combine PDFs

Combines all the PDFs in the current working directory into a single PDF.

Note:

- Example PDFs can be downloaded from http://nostarch.com/automatestuff/
- Book uses PyPDF2; I'm an overachiever that uses PyPDF4

AutomateTheBoringStuff.Ch13.P3_combinePDFs.main()

AutomateTheBoringStuff.Ch13.P4_readWord module

Read Word

This program uses `docx` to read Word documents.

AutomateTheBoringStuff.Ch13.P4_readWord.main()

AutomateTheBoringStuff.Ch13.P5_readDocx module

Read docx

Accepts a filename of a .docx file and returns a single string value of its text.

Note:

- Example .docx files can be downloaded from http://nostarch.com/automatestuff/

AutomateTheBoringStuff.Ch13.P5_readDocx.**getText**(*filename: str*) → str
Get text

Gets text from a given .docx file.

Parameters `filename` – Path to .docx file to get text from.

Returns String with all document text.

AutomateTheBoringStuff.Ch13.P6_writeWord module

Write Word

This program uses `docx` to write Word documents.

Note:

- Example files can be downloaded from http://nostarch.com/automatestuff/

AutomateTheBoringStuff.Ch13.P6_writeWord.**main**()

Module contents

AutomateTheBoringStuff.Ch14 package

Subpackages

AutomateTheBoringStuff.Ch14.Project package

Submodules

AutomateTheBoringStuff.Ch14.Project.excelToCSV module

Excel to CSV

Using `openpyxl`, write a program that reads all the Excel files in the current working directory and outputs them as CSV files.

A single Excel file might contain multiple sheets; you'll have to create one CSV file per sheet. The filenames of the CSV files should be `<excel filename>_<sheet title>.csv`, where `<excel filename>` is the filename of the Excel file without the file extension (for example, 'spam_data', not 'spam_data.xlsx') and `<sheet title>` is the string from the Worksheet object's title variable.

Notes

- Example Excel files can be downloaded from http://nostarch.com/automatestuff/
- Default input folder is `./excelSpreadsheets`.
- Default output folder is `./csvFiles`.

AutomateTheBoringStuff.Ch14.Project.excelToCSV.**main**()

Module contents

Submodules

AutomateTheBoringStuff.Ch14.P1_readCSV module

Read CSV

This program uses `csv` to read .csv files.

Note: Uses provided `example.csv` file.

`AutomateTheBoringStuff.Ch14.P1_readCSV.main()`

AutomateTheBoringStuff.Ch14.P2_writeCSV module

Write CSV

This program uses `csv` to write .csv files.

Note: Creates 'output.csv' and 'example.tsv' files.

`AutomateTheBoringStuff.Ch14.P2_writeCSV.main()`

AutomateTheBoringStuff.Ch14.P3_removeCsvHeader module

Remove CSV header

Removes the header from all CSV files in the current working directory.

Note: Outputs to `./headerRemoved` directory.

`AutomateTheBoringStuff.Ch14.P3_removeCsvHeader.main()`

AutomateTheBoringStuff.Ch14.P4_readWriteJSON module

Read/write JSON

This program uses `json` to manipulate JSON data.

`AutomateTheBoringStuff.Ch14.P4_readWriteJSON.main()`

AutomateTheBoringStuff.Ch14.P5_quickWeather module

Quick weather

Prints 3-day weather information for a location specified in the command line.

AutomateTheBoringStuff.Ch14.P5_quickWeather.**getWeather**(*loc: str*, *apikey: str*) → dict
Get weather

Uses OpenWeatherMap.org API to get JSON data of given location with given API key. Data is stored as a `dict` and the current date and time (in UTC) is also stored using `datetime.datetime.now()`.

Parameters

- **loc** – Location to get weather data of in `City,Country Code` format.
- **apikey** – API key used to interface with OpenWeatherMap.org's API.

Returns Dictionary with weather JSON data and current date time (in UTC) added.

AutomateTheBoringStuff.Ch14.P5_quickWeather.**main**() → None
P5_quickWeather.py

Displays given location's 3-day weather information.

Returns None. Weather data is printed to terminal and JSON data is stored in a `shelve` shelf, `weather`.

Note: To prevent excessive API requests, JSON data is stored in a `shelve` shelf and only re-downloaded every 10 minutes. Time is kept track using `datetime.datetime.now()` and `datetime.timedelta`.

Module contents

AutomateTheBoringStuff.Ch15 package

Subpackages

AutomateTheBoringStuff.Ch15.Projects package

Submodules

AutomateTheBoringStuff.Ch15.Projects.P1_prettifiedStopwatch module

Prettified stopwatch

Expand *P3_stopwatch* from this chapter so that it uses `str.rjust()` and `str.ljust()` string methods to "prettify" the output.

Instead of output such as this:

```
Lap #1: 3.56 (3.56)
Lap #2: 8.63 (5.07)
Lap #3: 17.68 (9.05)
Lap #4: 19.11 (1.43)
```

... the output will look like this:

```
Lap # 1:  3.56 (  3.56)
Lap # 2:  8.63 (  5.07)
Lap # 3: 17.68 (  9.05)
Lap # 4: 19.11 (  1.43)
```

Next, use `pyperclip` to copy the text output to the clipboard so the user can quickly paste the output to a text file or email.

`AutomateTheBoringStuff.Ch15.Projects.P1_prettifiedStopwatch.main()`

AutomateTheBoringStuff.Ch15.Projects.P2_scheduledComicDownloader module

Scheduled comic downloader

Write a program that checks the websites of several web comics and automatically downloads the images if the comic was updated since the program's last visit.

Your operating system's scheduler (Scheduled Tasks on Windows, launchd on OS X, and cron on Linux) can run your Python program once a day.

The Python program itself can download the comic and then copy it to your desktop so that it is easy to find. This will free you from having to check the website yourself to see whether it has updated.

Note: This only downloads from http://www.lefthandedtoons.com/ and http://buttersafe.com/ because all websites are different.

`AutomateTheBoringStuff.Ch15.Projects.P2_scheduledComicDownloader.check_key(`*shelf_arg: shelve.open*, *url_arg: str*`)` → bool

Check key

Checks if given url is a key in the given shelf.

Parameters

- **shelf_arg** – `shelve` object with urls as keys and `datetime.datetime.date()` as values.
- **url_arg** – String with website url.

Returns True if the url is in the shelf, False otherwise.

`AutomateTheBoringStuff.Ch15.Projects.P2_scheduledComicDownloader.compare_timestamps(`*timestamp_arg: str*, *shelf_arg: shelve.open*, *url_arg: str*`)` → bool

Compare timestamps

Compares timestamp of current comic to last downloaded comic timestamp of given url.

Parameters

- **timestamp_arg** – String with date in `Month DD, YYYY` format.
- **shelf_arg** – `shelve` object with urls as keys and `datetime.datetime.date()` as values.
- **url_arg** – String with website url.

Returns True if comic's timestamp is after saved timestamp, False otherwise.

AutomateTheBoringStuff.Ch15.Projects.P2_scheduledComicDownloader.**get_soup**(*url_arg: str*) → <sphinx.ext.autodoc.importer._Mo object at 0x7fe68cdfb3c8>

Get soup

Downloads given url with `requests` and converts it to `bs4.BeautifulSoup`.

Parameters **url_arg** – String with url to soupify.

Returns BeautifulSoup object of given url.

Raises **requests.exceptions.HTTPError** – If download of website url failed.

AutomateTheBoringStuff.Ch15.Projects.P2_scheduledComicDownloader.**main**()

AutomateTheBoringStuff.Ch15.Projects.P2_scheduledComicDownloader.**save_comic**(*comic_url_arg: str*, *shelf_arg: shelve.open*, *url_arg: str*) → None

Save comic

Downloads given comic url and saves to desktop, then updates download time of given website url in given shelf.

Parameters

- **comic_url_arg** – String with url of comic image.
- **shelf_arg** – `shelve` object with urls as keys and `datetime.datetime.date()` as values.
- **url_arg** – String with website url.

Returns None. Comic image is saved to desktop.

Raises **requests.exceptions.HTTPError** – If download of comic url failed.

Module contents

Submodules

AutomateTheBoringStuff.Ch15.P1_timeModule module

Time module

This program uses `time` to reference the Unix epoch as a timestamp.

AutomateTheBoringStuff.Ch15.P1_timeModule.**main**()

AutomateTheBoringStuff.Ch15.P2_calcProd module

Calculate product

Uses `time` to profile a function that calculates the product of the first 100,000 numbers.

Note:

- Added `cProfile` for an execution profile. Does add overhead, so not suitable for benchmarking.
- Added `timeit` for accurate execution timing.

`AutomateTheBoringStuff.Ch15.P2_calcProd.calcProd()` → int

Calculate product

Calculates the product of the first 100000 integers using a for loop.

Returns Integer product.

`AutomateTheBoringStuff.Ch15.P2_calcProd.main()`

AutomateTheBoringStuff.Ch15.P3_stopwatch module

Stopwatch

A simple stopwatch program with lap times.

`AutomateTheBoringStuff.Ch15.P3_stopwatch.main()`

AutomateTheBoringStuff.Ch15.P4_datetimeModule module

datetime module

This program uses `datetime` to manipulate dates and times.

`AutomateTheBoringStuff.Ch15.P4_datetimeModule.main()`

AutomateTheBoringStuff.Ch15.P5_threadDemo module

Thread demo

This program uses `threading` to demonstrate multithreading.

`AutomateTheBoringStuff.Ch15.P5_threadDemo.main()`

`AutomateTheBoringStuff.Ch15.P5_threadDemo.takeANap()` → None

Take a nap

Simple 5 second timer using `time.sleep()` followed by a 'Wake up!' print statement.

Returns None. Waits 5 seconds, then prints an exclamation.

AutomateTheBoringStuff.Ch15.P6_multithreading module

Multithreading

This program demonstrates `threading.Thread` on `print()`.

`AutomateTheBoringStuff.Ch15.P6_multithreading.main()`

AutomateTheBoringStuff.Ch15.P7_multidownloadXkcd module

Multidownload XKCD

Downloads 1400 XKCD comics much faster by using `threading`.

Note: Default output directory is `./xkcd`.

`AutomateTheBoringStuff.Ch15.P7_multidownloadXkcd.`**`downloadXkcd`**(*startComic: int*, *endComic: int*) → None

Download XKCD

Uses `requests` and `bs4` to download all comics in a given range.

Parameters

- **`startComic`** – Comic ID number to start from.
- **`endComic`** – Comic ID number to end at.

Returns None. Prints status updates and downloads comics to download directory.

`AutomateTheBoringStuff.Ch15.P7_multidownloadXkcd.main()`

AutomateTheBoringStuff.Ch15.P8_popenFunction module

Popen function

This program uses `subprocess.Popen` to launch programs.

`AutomateTheBoringStuff.Ch15.P8_popenFunction.main()`

AutomateTheBoringStuff.Ch15.P9_countdown module

Countdown

A simple countdown script that plays an alarm after a given number of seconds.

Note:

- Sound file can be downloaded from http://nostarch.com/automatestuff/

`AutomateTheBoringStuff.Ch15.P9_countdown.main()`

Module contents

AutomateTheBoringStuff.Ch16 package

Subpackages

AutomateTheBoringStuff.Ch16.Projects package

Submodules

AutomateTheBoringStuff.Ch16.Projects.P1_assignChores module

Assign chores

Write a program that takes a list of people's email addresses and a list of chores that need to be done and randomly assigns chores to people. Email each person their assigned chores.

If you're feeling ambitious, keep a record of each person's previously assigned chores so that you can make sure the program avoids assigning anyone the same chore they did last time.

For another possible feature, schedule the program to run once a week automatically.

Notes

- `smtp_info` file has each item on a separate line.
- Use `input()` for password to prevent storing in unencrypted file.
- It may be easier to:
  - Setup a crontab to run weekly.
  - Store *saved_time* and *prev_chores* in a `shelve` database.

`AutomateTheBoringStuff.Ch16.Projects.P1_assignChores.main()`

AutomateTheBoringStuff.Ch16.Projects.P2_rememberUmbrella module

Remember umbrella

Use `requests` to scrape data from http://weather.gov/.

Write a program that runs just before you wake up in the morning and checks whether it's raining that day. If so, have the program text you a reminder to pack an umbrella before leaving the house.

`AutomateTheBoringStuff.Ch16.Projects.P2_rememberUmbrella.check_time(`*time_arg: datetime.time*`)` → bool

Check time

Checks if given time is after current time as given by `datetime.datetime.now()`.

Parameters `time_arg` – `datetime.time` object to compare with current time.

Returns True if given time is after current time.

AutomateTheBoringStuff.Ch16.Projects.P2_rememberUmbrella.**get_weather**(*url_arg: str*) → str

Get weather

Uses `requests` to download given weather page url, then uses `bs4` to get the current weather data text.

Parameters **url_arg** – String containing url to specified city's http://weather.gov/ weather page.

Returns String with current weather data text.

AutomateTheBoringStuff.Ch16.Projects.P2_rememberUmbrella.**main**()

AutomateTheBoringStuff.Ch16.Projects.P2_rememberUmbrella.**remember_umbrella**(*weather_arg: str*) → bool

Remember umbrella

Checks current weather data text from *get_weather()* for keywords indicating rain.

Parameters **weather_arg** – String containing current weather text of specified city.

Returns True if any of the rain keywords are found, False otherwise.

AutomateTheBoringStuff.Ch16.Projects.P3_autoUnsubscribe module

Auto unsubscribe

Write a program that scans through your email account, finds all the unsubscribe links in all your emails, and automatically opens them in a browser. This program will have to log in to your email provider's IMAP server and download all of your emails. You can use BeautifulSoup to check for any instance where the word unsubscribe occurs within an HTML link tag.

Once you have a list of these URLs, you can use `webbrowser.open()` to automatically open all of these links in a browser.

Notes

- `imap_info` file has each item on a separate line.
- Email address used is specially created for this chapter.
- Use `input()` for password to prevent storing in unencrypted file.

AutomateTheBoringStuff.Ch16.Projects.P3_autoUnsubscribe.**main**()

AutomateTheBoringStuff.Ch16.Projects.P4_autoDownloadTorrent module

Auto download torrent

Write a program that checks an email account every 15 minutes for any instructions you email it and executes those instructions automatically.

For example, BitTorrent is a peer-to-peer downloading system. Using free BitTorrent software such as qBittorrent, you can download large media files on your home computer. If you email the program a (completely legal, not at all piratical) BitTorrent link, the program will eventually check its email, find this message, extract the link, and then launch qBittorrent to start downloading the file. This way, you can have your home computer begin downloads while you're away, and the (completely legal, not at all piratical) download can be finished by the time you return home.

Notes

- Shutting down after downloading is considered "Hit 'n' run" and goes against torrenting.
  - Consider setting up a seed ratio limit and let it stop sharing afterward.
- Transmission torrent client is used since it is available in Ubuntu by default.
  - A bash script is ultimately needed to shutdown Transmission.
  - Remote access is needed to run the bash script (hint).

`AutomateTheBoringStuff.Ch16.Projects.P4_autoDownloadTorrent.`**`autodownload_torrent`**`(`*url_arg: str*`)` → None

Auto download torrent

Starts `subprocess` with Transmission client and waits until given torrent url is downloaded.

Parameters **`url_arg`** – String with url of torrent to download.

Returns None. Torrent client specifies where torrent is downloaded to.

Note: Configured specifically for Transmission torrent client in Ubuntu.

`AutomateTheBoringStuff.Ch16.Projects.P4_autoDownloadTorrent.`**`fetch_emails`**`(`*imap_obj_arg: <sphinx.ext.autodoc.importer._Mock object at 0x7fe68cc64668>*`)` → tuple

Fetch emails

Gets emails from IMAP server and returns the email's uids and their raw messages.

Parameters **`imap_obj_arg`** – Configured `imapclient.IMAPClient` object from *`login_imap()`*.

Returns Tuple with a list of message uids and a dictionary of raw messages with message uids as keys.

`AutomateTheBoringStuff.Ch16.Projects.P4_autoDownloadTorrent.`**`fetch_torrents`**`(`*uids_arg: list*, *raw_messages_arg: dict*`)` → dict

Fetch torrents

Takes given list of message uids and dictionary of raw messages from *`fetch_emails()`* and parses out the torrent urls.

Parameters

- **`uids_arg`** – List of message uids.
- **`raw_messages_arg`** – Dictionary of raw messages with message uids as keys and message data as values.

Returns Dictionary with message uids as keys and the torrent url string as values.

AutomateTheBoringStuff.Ch16.Projects.P4_autoDownloadTorrent.**login_imap**(*file_arg: str*) → <sphinx.ext.autodoc.importer._MockO object at 0x7fe68cc64898>

Login IMAP

Logs into IMAP server with credentials from given file, then returns the configured imapclient.IMAPClient object.

Parameters **file_arg** – String with path to IMAP server information file.

Returns Configured imapclient.IMAPClient object.

AutomateTheBoringStuff.Ch16.Projects.P4_autoDownloadTorrent.**login_smtp**(*file_arg: str*) → tuple

Login SMTP

Logs into SMTP server with credentials from given file, then returns the configured smtplib.SMTP_SSL object and account email.

Parameters **file_arg** – String with path to SMTP server information file.

Returns Tuple with smtplib.SMTP_SSL object and account email.

AutomateTheBoringStuff.Ch16.Projects.P4_autoDownloadTorrent.**main**()

Module contents

Submodules

AutomateTheBoringStuff.Ch16.P1_sendingEmail module

Sending email

This program uses smtplib to send emails.

Notes

- smtp_info file has each item on a separate line.
- Email address used is specially created for this chapter.
- Use input() for password to prevent storing in unencrypted file.

AutomateTheBoringStuff.Ch16.P1_sendingEmail.**main**()

AutomateTheBoringStuff.Ch16.P2_receivingEmail module

Receiving email

This program uses imapclient.IMAPClient and pyzmail36 to retrieve emails.

Notes

- imap_info file has each item on a separate line.
- Email address used is specially created for this chapter.

- Use `input()` for password to prevent storing in unencrypted file.

`AutomateTheBoringStuff.Ch16.P2_receivingEmail.main()`

AutomateTheBoringStuff.Ch16.P3_sendDuesReminders module

Send dues reminders

Sends emails based on payment status in spreadsheet.

Notes

- `smtp_info` file has each item on a separate line.
- Email address used is specially created for this chapter.
- Use `input()` or `sys.argv[1]` for password to prevent storing in unencrypted file.

`AutomateTheBoringStuff.Ch16.P3_sendDuesReminders.main()`

AutomateTheBoringStuff.Ch16.P4_sendSMS module

Send SMS

This program uses twilio to send SMS messages to a phone number.

Note:

- `twilio_info` file has each item on a separate line.
- Use `input()` to prevent storing sensitive info in unencrypted file.

`AutomateTheBoringStuff.Ch16.P4_sendSMS.main()`

AutomateTheBoringStuff.Ch16.P5_textMyself module

P5_textMyself.py

Defines *`textmyself()`* that texts a string message passed to it.

Note:

- `twilio_info` file has each item on a separate line.
- Use `input()` to prevent storing sensitive info in unencrypted file.

`AutomateTheBoringStuff.Ch16.P5_textMyself.textmyself`(*message: str*) → None

Text myself

Sends given message as SMS to twilio account specified in `./twilio_info`.

Parameters `message` – String containing message to be sent.

Returns None. Message is sent to twilio phone number.

Module contents

AutomateTheBoringStuff.Ch17 package

Subpackages

AutomateTheBoringStuff.Ch17.Projects package

Submodules

AutomateTheBoringStuff.Ch17.Projects.P1_remixResizeAndAddLogo module

Remix resize and add logo

The *P3_resizeAndAddLogo* program in this chapter works with PNG and JPEG files, but `PIL.Image` supports many more formats than just these two. Extend *P3_resizeAndAddLogo* to process GIF and BMP images as well.

Another small issue is that the program modifies PNG and JPEG files only if their file extensions are set in lowercase. For example, it will process `zophie.png` but not `zophie.PNG`. Change the code so that the file extension check is case insensitive.

Finally, the logo added to the bottom-right corner is meant to be just a small mark, but if the image is about the same size as the logo itself, the result will look ugly. Modify *P3_resizeAndAddLogo* so that the image must be at least twice the width and height of the logo image before the logo is pasted. Otherwise, it should skip adding the logo.

```
AutomateTheBoringStuff.Ch17.Projects.P1_remixResizeAndAddLogo.main()
```

AutomateTheBoringStuff.Ch17.Projects.P2_findPhotoFolders module

Find photo folders

This program identifies all folders in `/home/jose` where over 50% of the files are images of a specified size.

```
AutomateTheBoringStuff.Ch17.Projects.P2_findPhotoFolders.main()
```

AutomateTheBoringStuff.Ch17.Projects.P3_seatingCards module

Seating cards

Chapter 13 included a practice project to create custom invitations from a list of guests in a plaintext file - *P3_invitations*. As an additional project, use `PIL` to create images for custom seating cards for your guests. For each of the guests listed in `guests.txt`, generate an image file with the guest's name and some flowery decoration.

To ensure that each seating card is the same size, add a black rectangle on the edges of the invitation image so that when the image is printed out, there will be a guideline for cutting. The PNG files that `PIL.Image` produces are set to 72 pixels per inch, so a 4×5-inch card would require a 288×360-pixel image.

Note:

- Flowery decoration from: http://www.reusableart.com/flower-02.html

---

AutomateTheBoringStuff.Ch17.Projects.P3_seatingCards.main()

Module contents

Submodules

AutomateTheBoringStuff.Ch17.P1_imageFundamentals module

Image fundamentals

This program uses PIL.ImageColor.getcolor() to convert color names to RGBA values.

AutomateTheBoringStuff.Ch17.P1_imageFundamentals.main()

AutomateTheBoringStuff.Ch17.P2_manipulatingImages module

Manipulating images

This program uses PIL.Image to manipulate digital images.

AutomateTheBoringStuff.Ch17.P2_manipulatingImages.main()

AutomateTheBoringStuff.Ch17.P3_resizeAndAddLogo module

Resize and add logo

Resizes all images in current working directory to fit in a 300x300 square, then adds ./catlogo.png to the lower-right corner.

AutomateTheBoringStuff.Ch17.P3_resizeAndAddLogo.main()

AutomateTheBoringStuff.Ch17.P4_drawingOnImages module

Drawing on images

This program uses PIL.Image and PIL.ImageDraw to draw on digital images.

AutomateTheBoringStuff.Ch17.P4_drawingOnImages.main()

Module contents

AutomateTheBoringStuff.Ch18 package

Subpackages

AutomateTheBoringStuff.Ch18.Projects package

Submodules

AutomateTheBoringStuff.Ch18.Projects.P1_lookingBusy module

Looking busy

Write a script to nudge your mouse cursor slightly every ten seconds. The nudge should be small enough so that it won't get in the way if you do happen to need to use your computer while the script is running.

`AutomateTheBoringStuff.Ch18.Projects.P1_lookingBusy.main()`

Module contents

Submodules

AutomateTheBoringStuff.Ch18.P1_mouseMovement module

Mouse movement

This program uses `PyAutoGUI` to control mouse movement.

`AutomateTheBoringStuff.Ch18.P1_mouseMovement.main()`

AutomateTheBoringStuff.Ch18.P2_mouseNow module

Mouse now

Displays the mouse cursor's current position in terminal.

`AutomateTheBoringStuff.Ch18.P2_mouseNow.main()`

AutomateTheBoringStuff.Ch18.P3_mouseInteraction module

Mouse interaction

This program uses `PyAutoGUI` to control mouse interaction.

`AutomateTheBoringStuff.Ch18.P3_mouseInteraction.main()`

AutomateTheBoringStuff.Ch18.P4_spiralDraw module

Spiral draw

Program uses `PyAutoGUI` to draw a spiral pattern in an opened drawing program.

`AutomateTheBoringStuff.Ch18.P4_spiralDraw.main()`

AutomateTheBoringStuff.Ch18.P5_screenshots module

Screenshots

This program uses `PyAutoGUI` to take and analyze screenshots.

Note:

- For Linux distros, the `scrot` program needs to be installed.

`AutomateTheBoringStuff.Ch18.P5_screenshots.main()`

AutomateTheBoringStuff.Ch18.P6_mouseNow module

Mouse now 2.0

Extends *P2_mouseNow* to display the mouse cursor's current position and RGB color in terminal.

`AutomateTheBoringStuff.Ch18.P6_mouseNow.main()`

AutomateTheBoringStuff.Ch18.P7_controlKeyboard module

Control Keyboard

This program uses `PyAutoGUI` to manipulate keyboard input.

`AutomateTheBoringStuff.Ch18.P7_controlKeyboard.commentAfterDelay()` → None
Comment after delay

Automatically types then comments out a line in IDLE after waiting two seconds.

Returns None. Executes keyboard commands.

`AutomateTheBoringStuff.Ch18.P7_controlKeyboard.main()`

AutomateTheBoringStuff.Ch18.P8_formFiller module

Form filler

Automatically fills in the form at http://autbor.com/form

`AutomateTheBoringStuff.Ch18.P8_formFiller.main()`

Module contents

Module contents

1.6.2 CrackingCodesWithPython package

Subpackages

CrackingCodesWithPython.Chapter01 package

Submodules

CrackingCodesWithPython.Chapter01.PracticeQuestions module

Chapter 1 Practice Questions.

Answers Chapter 1 Practice Questions via Python code.

Notes

- Contains spoilers from Chapter 5 (caesar cipher), Chapter 6 (caesar hacker), and Chapter 7 (functions)
- Corrections submitted for Questions 1, 3, 4, and 5

`CrackingCodesWithPython.Chapter01.PracticeQuestions.main()`

CrackingCodesWithPython.Chapter01.caesarCipher module

Caesar Cipher improved.

Rewritten as function with wrapper functions for importing.

Note: Contains spoilers from Chapter 5 (caesarCipher) and Chapter 7 (functions)

`CrackingCodesWithPython.Chapter01.caesarCipher.caesarCipher(`*key: int*, *message: str*, *mode: str*`)` → str

Implement caesar cipher.

Encrypts or decrypts given message with given key depending on given mode.

Parameters

- **key** – Key to use for [de|en]cryption.
- **message** – Message to encrypt/decrypt.
- **mode** – Specifies encryption or decryption.

Returns Encrypted/decrypted message string.

Example

```
>>> from pythontutorials.books.CrackingCodesWithPython.Chapter01.caesarCipher␣
↪import caesarCipher
>>> caesarCipher(4, 'IMPIETY: YOUR IRREVERENCE TOWARD MY DEITY.', 'encrypt')
'MQTMIXc:AcSYVAMVVIZIVIRGIAXSaEVHAQcAHIMXcD'
```

`CrackingCodesWithPython.Chapter01.caesarCipher.decryptMessage(`*key: int*, *message: str*`)` → str

Decrypts encrypted caesar cipher.

Wrapper function that calls caesarCipher() to decrypt given message with given key.

Parameters

- **key** – Key to use to decrypt message.
- **message** – Message to decrypt.

Returns Decrypted message string.

`CrackingCodesWithPython.Chapter01.caesarCipher.encryptMessage(`*key: int*, *message: str*`)` → str

Encrypts message with caesar cipher.

Wrapper function that calls caesarCipher() to encrypt given message with given key.

Parameters

- **key** – Key to use to encrypt message.
- **message** – Message to encrypt.

Returns Encrypted message string.

CrackingCodesWithPython.Chapter01.caesarHacker module

Caesar Hacker improved.

Rewritten as function for importing.

Note: Contains spoilers from Chapter 6 (caesarHacker) and Chapter 7 (functions)

CrackingCodesWithPython.Chapter01.caesarHacker.**hackCaesar**(*message: str*) → None
Hacks caesar cipher.

Loops through and displays every possible key.

Parameters **message** – Message to be decrypted.

Returns Prints each decryption with every possible key.

CrackingCodesWithPython.Chapter01.constants module

Configuration file with global variables.

Mainly contains definition of every possible encryptable symbol.

CrackingCodesWithPython.Chapter01.constants.**SYMBOLS**
Every possible symbol that can be encrypted.

Type str

Module contents

CrackingCodesWithPython.Chapter02 package

Submodules

CrackingCodesWithPython.Chapter02.PracticeQuestions module

Chapter 2 Practice Questions

Answers Chapter 2 Practice Questions via Python code.

Note: To check these questions, they should be entered in IDLE; otherwise print statements would be needed.

CrackingCodesWithPython.Chapter02.PracticeQuestions.**main**()

Module contents

CrackingCodesWithPython.Chapter03 package

Submodules

CrackingCodesWithPython.Chapter03.PracticeQuestions module

Chapter 3 Practice Questions

Answers Chapter 3 Practice Questions via Python code.

`CrackingCodesWithPython.Chapter03.PracticeQuestions.main()`

CrackingCodesWithPython.Chapter03.hello module

Asks for name and says hello.

This program says hello and asks for my name.

Notes

- Using double quotes for strings because I'm a nitpicker - author admits that he uses single quotes because it is easier to type and it technically doesn't matter.
- Nov. 22, 2018 Update: Switching back to single quotes because a system was compromised because of double quotes.

`CrackingCodesWithPython.Chapter03.hello.main()`

Module contents

CrackingCodesWithPython.Chapter04 package

Submodules

CrackingCodesWithPython.Chapter04.PracticeQuestions module

Chapter 4 Practice Questions

Answers Chapter 4 Practice Questions via Python code.

`CrackingCodesWithPython.Chapter04.PracticeQuestions.main()`

CrackingCodesWithPython.Chapter04.reverseCipher module

Reverse Cipher

https://www.nostarch.com/crackingcodes/ (BSD Licensed)

Note: Pretty much the same, except I use double quotes, expand variable names for readability, simplify the while loop, and use fancier operators

`CrackingCodesWithPython.Chapter04.reverseCipher.main()`

Module contents

CrackingCodesWithPython.Chapter05 package

Subpackages

CrackingCodesWithPython.Chapter05.PracticeQuestions package

Submodules

CrackingCodesWithPython.Chapter05.PracticeQuestions.Question1 module

Chapter 5 Practice Question 1

Using caesarCipher.py, encrypt the following sentences with the given keys.

Note: Contains spoilers for Chapter 7 (functions)

`CrackingCodesWithPython.Chapter05.PracticeQuestions.Question1.main()`

CrackingCodesWithPython.Chapter05.PracticeQuestions.Question2 module

Chapter 5 Practice Question 2

Using caesarCipher.py, decrypt the following ciphertexts with the given keys

Note: Contains spoilers for Chapter 7 (functions)

`CrackingCodesWithPython.Chapter05.PracticeQuestions.Question2.main()`

CrackingCodesWithPython.Chapter05.PracticeQuestions.Question3 module

Chapter 5 Practice Question 3

Which Python instruction would import a module named watermelon.py?

Note: Contains spoilers for Chapter 7 (functions)

`CrackingCodesWithPython.Chapter05.PracticeQuestions.Question3.main()`

CrackingCodesWithPython.Chapter05.PracticeQuestions.Question4 module

Chapter 5 Practice Question 4

What do the following pieces of code display on the screen?

Note: Contains spoilers for Chapter 7 (functions)

CrackingCodesWithPython.Chapter05.PracticeQuestions.Question4.**main**()

CrackingCodesWithPython.Chapter05.PracticeQuestions.watermelon module

Watermelon.py

Demonstration for *CrackingCodesWithPython.Chapter05.PracticeQuestions.Question3*

Note: Contains spoilers for Chapter 7 (functions)

CrackingCodesWithPython.Chapter05.PracticeQuestions.watermelon.**main**()

CrackingCodesWithPython.Chapter05.PracticeQuestions.watermelon.**nutrition**() → None
Watermelon nutrition info.

Contains nutrition facts of a serving of watermelon.

Returns Prints a series of strings containing the nutrition facts of a serving of watermelon.

Module contents

Submodules

CrackingCodesWithPython.Chapter05.caesarCipher module

Caesar Cipher

Demonstrates the use of a caesar cipher. Prints output and copies to clipboard.

Note: https://www.nostarch.com/crackingcodes/ (BSD Licensed)

CrackingCodesWithPython.Chapter05.caesarCipher.**main**()

CrackingCodesWithPython.Chapter05.checkPw module

Password checker.

Checks given input to saved password.

CrackingCodesWithPython.Chapter05.checkPw.**main**()

Module contents

CrackingCodesWithPython.Chapter06 package

Submodules

CrackingCodesWithPython.Chapter06.PracticeQuestion module

Chapter 6 Practice Questions

Answers Chapter 6 Practice Questions via Python code.

Break the following ciphertext one line at a time because each line has a different key. Remember to escape any quote characters

Note: Contains spoilers for chapter 7 (functions)

`CrackingCodesWithPython.Chapter06.PracticeQuestion.main()`

CrackingCodesWithPython.Chapter06.caesarHacker module

Caesar Cipher Hacker

Demonstrates how to implement a program that hacks a caesar cipher.

Note: https://www.nostarch.com/crackingcodes/ (BSD Licensed)

`CrackingCodesWithPython.Chapter06.caesarHacker.main()`

Module contents

CrackingCodesWithPython.Chapter07 package

Subpackages

CrackingCodesWithPython.Chapter07.PracticeQuestions package

Submodules

CrackingCodesWithPython.Chapter07.PracticeQuestions.Question1 module

Chapter 7 Practice Question 1

Encrypt the following with the transposition cipher (with paper and pencil, *cough*).

Note: Contains spoilers for Chapter 9 (importing transpositionEncrypt)

`CrackingCodesWithPython.Chapter07.PracticeQuestions.Question1.main()`

CrackingCodesWithPython.Chapter07.PracticeQuestions.Question2 module

Chapter 7 Practice Question 2

Is each spam a global or local variable?

`CrackingCodesWithPython.Chapter07.PracticeQuestions.Question2.foo()` → None
Prints spam.

Prints the contents of the spam variable.

Returns Prints spam variable.

`CrackingCodesWithPython.Chapter07.PracticeQuestions.Question2.main()`

CrackingCodesWithPython.Chapter07.PracticeQuestions.Question3 module

Chapter 7 Practice Question 3

What value does each of the following expressions evaluate to?

Note: aka "The power of lists"

`CrackingCodesWithPython.Chapter07.PracticeQuestions.Question3.main()`

CrackingCodesWithPython.Chapter07.PracticeQuestions.Question4 module

Chapter 7 Practice Question 4

What value does each of the following expressions evaluate to?

Note: aka "Lists are OP"

`CrackingCodesWithPython.Chapter07.PracticeQuestions.Question4.main()`

CrackingCodesWithPython.Chapter07.PracticeQuestions.Question5 module

Chapter 7 Practice Question 5

What are the four augmented assignment operators?

Note: Hint: Table 7-1 on pg 92

`CrackingCodesWithPython.Chapter07.PracticeQuestions.Question5.main()`

Module contents

Submodules

CrackingCodesWithPython.Chapter07.addNumbers module

Addition function

Contains a function that adds two numbers.

CrackingCodesWithPython.Chapter07.addNumbers.**addNumbers**(*a: int*, *b: int*) → int
Adds two numbers.

Performs addition operation to two numbers.

Parameters

- **a** – Input to add to
- **b** – Input to be added

Returns Result of addition of two inputs.

CrackingCodesWithPython.Chapter07.addNumbers.**main**()

CrackingCodesWithPython.Chapter07.helloFunction module

Hello function.

Contains function that prints hello to given name.

CrackingCodesWithPython.Chapter07.helloFunction.**hello**(*name: str*) → None
Prints hello.

Prints hello to given name.

Parameters **name** – Name to say hello to.

Returns Prints hello to given name.

CrackingCodesWithPython.Chapter07.helloFunction.**main**()

CrackingCodesWithPython.Chapter07.transpositionEncrypt module

Transposition Cipher Encryption

Demonstrates how to implement a transposition cipher.

Note: https://www.nostarch.com/crackingcodes/ (BSD Licensed)

CrackingCodesWithPython.Chapter07.transpositionEncrypt.**encryptMessage**(*key: int*, *message: str*) → str
Transposition Cipher Encrypt

Encrypts given message using a transposition cipher with given key.

Parameters

- **key** – Numeric key to encrypt with.
- **message** – Message to encrypt.

Returns Message encrypted in a string.

Example

```
>>> encryptMessage(9, 'Underneath a huge oak tree there was of swine a huge company,
↪')
'Uhot  on ahoamdakef pe  r harhtesunnur wgyegewie,aeean t  sec'
```

CrackingCodesWithPython.Chapter07.transpositionEncrypt.**main**()

Module contents

CrackingCodesWithPython.Chapter08 package

Subpackages

CrackingCodesWithPython.Chapter08.PracticeQuestions package

Submodules

CrackingCodesWithPython.Chapter08.PracticeQuestions.Question1 module

Chapter 8 Practice Question 1

Using paper and pencil (*cough*), decrypt the following messages with the key 9.

Note: Contains spoilers for Chapter 9 (importing transpositionDecrypt)

CrackingCodesWithPython.Chapter08.PracticeQuestions.Question1.**main**()

CrackingCodesWithPython.Chapter08.PracticeQuestions.Question2 module

Chapter 8 Practice Question 2

When you enter the following code into the interactive shell (*cough*), what does each line print?

CrackingCodesWithPython.Chapter08.PracticeQuestions.Question2.**main**()

CrackingCodesWithPython.Chapter08.PracticeQuestions.Question3 module

Chapter 8 Practice Question 3

Draw the complete truth tables for the and, or, and not operators.

CrackingCodesWithPython.Chapter08.PracticeQuestions.Question3.**andTruthTable**() → None
And truth table.

Prints a truth table for the and operator.

Returns None. Only prints out a table.

CrackingCodesWithPython.Chapter08.PracticeQuestions.Question3.**main**()

CrackingCodesWithPython.Chapter08.PracticeQuestions.Question3.**notTruthTable**() → None
Not truth table.

Prints a truth table for the not operator.

Returns None. Only prints out a table.

CrackingCodesWithPython.Chapter08.PracticeQuestions.Question3.**orTruthTable**() → None
Or truth table.

Prints a truth table for the or operator.

Returns None. Only prints out a table.

CrackingCodesWithPython.Chapter08.PracticeQuestions.Question4 module

Chapter 8 Practice Question 4

Which of the following is correct?

if ___name___ == '___main___': if ___main___ == '___name___': if _name_ == '_main_': if _main_ == '_name_':

Note: answer variable needs to be decrypted with the specified key

---

CrackingCodesWithPython.Chapter08.PracticeQuestions.Question4.**main**()

Module contents

Submodules

CrackingCodesWithPython.Chapter08.transpositionDecrypt module

Transposition Cipher Decryption

Decrypts transposition cipher messages.

Note: https://www.nostarch.com/crackingcodes/ (BSD Licensed)

CrackingCodesWithPython.Chapter08.transpositionDecrypt.**decryptMessage**(*key: int*, *message: str*) → str
Decrypt transposition cipher.

Decrypts transposition cipher messages with given key.

Parameters

- **key** – Numeric key to use for decryption.
- **message** – Message string to decrypt.

Returns Decrypted message in a string.

CrackingCodesWithPython.Chapter08.transpositionDecrypt.**main**()

Module contents

CrackingCodesWithPython.Chapter09 package

Submodules

CrackingCodesWithPython.Chapter09.PracticeQuestions module

Chapter 9 Practice Questions

Answers Chapter 9 Practice Questions via Python code.

`CrackingCodesWithPython.Chapter09.PracticeQuestions.main()`

CrackingCodesWithPython.Chapter09.passingReference module

Passing references in a function

Demonstrates how to pass a reference to a function.

`CrackingCodesWithPython.Chapter09.passingReference.eggs`(*someParameter: list*) → None
Append to a parameter.

Appends 'Hello' to a given parameter.

Parameters `someParameter` – List of elements.

Returns None. Only appends a string to a provided parameter.

`CrackingCodesWithPython.Chapter09.passingReference.main()`

CrackingCodesWithPython.Chapter09.transpositionTest module

Transposition Cipher Test

Demonstrates a unit test for the transposition encrypt and decrypt functions.

Note: https://www.nostarch.com/crackingcodes/ (BSD Licensed)

`CrackingCodesWithPython.Chapter09.transpositionTest.main()`

Module contents

CrackingCodesWithPython.Chapter10 package

Submodules

CrackingCodesWithPython.Chapter10.PracticeQuestions module

Chapter 10 Practice Questions

Answers Chapter 10 Practice Questions via Python code.

CrackingCodesWithPython.Chapter10.PracticeQuestions.main()

CrackingCodesWithPython.Chapter10.transpositionFileCipher module

Transposition Cipher Encrypt/Decrypt File

Implements a transposition cipher that can encrypt/decrypt a file.

Note: https://www.nostarch.com/crackingcodes/ (BSD Licensed)

CrackingCodesWithPython.Chapter10.transpositionFileCipher.main()

Module contents

CrackingCodesWithPython.Chapter11 package

Submodules

CrackingCodesWithPython.Chapter11.PracticeQuestions module

Chapter 11 Practice Questions

Answers Chapter 11 Practice Questions via Python code.

CrackingCodesWithPython.Chapter11.PracticeQuestions.main()

CrackingCodesWithPython.Chapter11.detectEnglish module

Detect English Module

Provides functions to determine whether a given string is in the English language.

CrackingCodesWithPython.Chapter11.detectEnglish.UPPERLETTERS
String containing all latin-based letters in uppercase.

Type str

CrackingCodesWithPython.Chapter11.detectEnglish.LETTERS_AND_SPACE
String containing upper and lowercase letters as well as space, newline, and tab.

Type str

CrackingCodesWithPython.Chapter11.detectEnglish.DICTIONARY_FILE
String containing absolute path of dictionary.txt file.

Type str

CrackingCodesWithPython.Chapter11.detectEnglish.ENGLISH_WORDS
Dictionary containing all words from dictionary.txt as keys.

Type dict

Example

```
>>> import pythontutorials.books.CrackingCodesWithPython.Chapter11.detectEnglish as␣
↪detectEnglish
>>> someString = 'Enthusiasm is contagious. Not having enthusiasm is also contagious.'
>>> detectEnglish.isEnglish(someString)  # Returns True or False
True
```

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)
- There must be a "dictionary.txt" file in this directory with all English words in it, one word per line. You can download this from https://www.nostarch.com/crackingcodes/.

CrackingCodesWithPython.Chapter11.detectEnglish.**getEnglishCount**(*message: str*) → float
Get count of English words

For given message, counts number of words in English dictionary and returns ratio of English words out of total words.

Parameters **message** – String with message to check for English words.

Returns Ratio of number of English words / total number of words.

CrackingCodesWithPython.Chapter11.detectEnglish.**isEnglish**(*message: str*, *wordPercentage: int = 20*, *letterPercentage: int = 85*) → bool
Determines whether message is English

Using given word percentage and letter percentage, determines if a given message is in the English language.

Parameters

- **message** – String containing message to determine if it is English.
- **wordPercentage** – Integer representing percentage of words in message that must be English.
- **letterPercentage** – Integer representing percentage of characters in message that must be letters or spaces.

Returns True if message is in English language, False otherwise.

Note:

- By default, 20% of the words must exist in the dictionary file, and 85% of all the characters in the message must be letters or spaces (not punctuation or numbers).

CrackingCodesWithPython.Chapter11.detectEnglish.**loadDictionary**() → dict
Load dictionary file

Loads dictionary.txt file and creates a dictionary with all words as keys.

Returns Dictionary with all words in dictionary.txt as keys.

CrackingCodesWithPython.Chapter11.detectEnglish.**removeNonLetters**(*message: str*) → str
Removes non-letters

Removes non-letter characters from given message.

Parameters `message` – String with message to remove non-letter characters from.

Returns New string with non-letter characters removed.

Module contents

CrackingCodesWithPython.Chapter12 package

Submodules

CrackingCodesWithPython.Chapter12.PracticeQuestions module

Chapter 12 Practice Questions

Answers Chapter 12 Practice Questions via Python code.

`CrackingCodesWithPython.Chapter12.PracticeQuestions.main()`

CrackingCodesWithPython.Chapter12.transpositionHacker module

Transposition Cipher Hacker

Implements a function that can hack a transposition cipher encrypted message.

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)

`CrackingCodesWithPython.Chapter12.transpositionHacker.hackTransposition(`*message: str*`)`

Hacks transposition cipher encrypted messages

Brute-forces a given encrypted message by looping through all the keys, checking if the result is English, and prompting the user for confirmation of decryption.

Parameters `message` – String with message to brute-force.

Returns Prints out possible results and prompts user for confirmation. If confirmed, prints out and returns full decrypted message, otherwise returns None.

`CrackingCodesWithPython.Chapter12.transpositionHacker.main()`

Module contents

CrackingCodesWithPython.Chapter13 package

Submodules

CrackingCodesWithPython.Chapter13.PracticeQuestions module

Chapter 13 Practice Questions

Answers Chapter 13 Practice Questions via Python code.

CrackingCodesWithPython.Chapter13.PracticeQuestions.main()

CrackingCodesWithPython.Chapter13.cryptomath module

Cryptomath Module

Provides mathematical functions for use in cryptography. (Discrete mathematics FTW!)

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)

CrackingCodesWithPython.Chapter13.cryptomath.**findModInverse**(*a: int*, *m: int*)
Modular inverse

Returns modular inverse of given inputs using Euclid's extended algorithm.

Parameters

- **a** – First integer input.
- **m** – Second integer input.

Returns Modular inverse as an integer if it exists, None otherwise.

CrackingCodesWithPython.Chapter13.cryptomath.**gcd**(*a: int*, *b: int*) → int
Greatest common divisor

Returns greatest common divisor of given inputs using Euclid's algorithm.

Parameters

- **a** – First integer input.
- **b** – Second integer input.

Returns Integer representing GCD.

Module contents

CrackingCodesWithPython.Chapter14 package

Submodules

CrackingCodesWithPython.Chapter14.PracticeQuestions module

Chapter 14 Practice Questions

Answers Chapter 14 Practice Questions via Python code.

CrackingCodesWithPython.Chapter14.PracticeQuestions.main()

CrackingCodesWithPython.Chapter14.affineCipher module

Affine Cipher

Provides functions that implement affine cipher encryption and decryption.

CrackingCodesWithPython.Chapter14.affineCipher.SYMBOLS
String containing all symbols that can be encrypted/decrypted.

Type str

Example

```
>>> import pythontutorials.books.CrackingCodesWithPython.Chapter14.affineCipher as␣
↪affineCipher
>>> someString = 'Enthusiasm is contagious. Not having enthusiasm is also contagious.'
>>> key = affineCipher.getRandomKey()  # key = 921, in this example
>>> affineCipher.encryptMessage(key, someString)
'xq3eBprFpdLrpLf4q3FRr4BpyLi43LeF0rqRL6q3eBprFpdLrpLFQp4Lf4q3FRr4Bpy'
```

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)
- There must be a "dictionary.txt" file in this directory with all English words in it, one word per line. You can download this from https://www.nostarch.com/crackingcodes/.

CrackingCodesWithPython.Chapter14.affineCipher.**checkKeys**(*keyA: int*, *keyB: int*, *mode: str*) → None
Checks keys for validity.

Prevents keyA from being 1 and keyB from being 0 (if encrypting). Makes sure keyA is relatively prime with the length of SYMBOLS. Ensures keyA is greater than 0 and that keyB is between 0 and length of SYMBOLS.

Parameters

- **keyA** – Integer integral of the original key after floor division by length of SYMBOLS.
- **keyB** – Integer remainder of the original key after modulus by length of SYMBOLS.
- **mode** – String specifying whether to 'encrypt' or 'decrypt'.

Returns None if successful, exits program with error message otherwise.

CrackingCodesWithPython.Chapter14.affineCipher.**decryptMessage**(*key: int*, *message: str*) → str
Affine cipher decryption

Decrypts given affine cipher encrypted message with given key.

Parameters

- **key** – Integer decryption key to decrypt affine cipher.
- **message** – Message string to decrypt.

Returns Decrypted message string.

CrackingCodesWithPython.Chapter14.affineCipher.**encryptMessage**(*key: int*, *message: str*) → str
Affine cipher encryption

Encrypts given message with given key using the affine cipher.

Parameters

- **key** – Integer encryption key to encrypt with affine cipher.

- **message** – Message string to encrypt.

Returns Encrypted message string.

CrackingCodesWithPython.Chapter14.affineCipher.**getKeyParts**(*key: int) -> (<class 'int'>, <class 'int'>)*

Split key into parts

Splits key into keyA and keyB via floor division and modulus by length of SYMBOLS.

Parameters **key** – Integer key used to encrypt message.

Returns Tuple containing the integral and remainder.

CrackingCodesWithPython.Chapter14.affineCipher.**getRandomKey**() → int

Affine cipher key generator

Generates a random key that can be used with the affine cipher.

Returns Random, valid integer key

CrackingCodesWithPython.Chapter14.affineCipher.**main**()

CrackingCodesWithPython.Chapter14.affineKeyTest module

Test affine cipher keyspace

This program proves that the keyspace of the affine cipher is limited to less than len(SYMBOLS) ^ 2.

Note: Tests every key from 2 through 80 and prints it with the encrypted message if the key and length of SYMBOLS have a gcd.

CrackingCodesWithPython.Chapter14.affineKeyTest.**main**()

Module contents

CrackingCodesWithPython.Chapter15 package

Submodules

CrackingCodesWithPython.Chapter15.PracticeQuestions module

Chapter 15 Practice Questions

Answers Chapter 15 Practice Questions via Python code.

CrackingCodesWithPython.Chapter15.PracticeQuestions.**main**()

CrackingCodesWithPython.Chapter15.affineHacker module

Affine Cipher Hacker

Implements a function that can hack an affine cipher encrypted message.

CrackingCodesWithPython.Chapter15.affineHacker.**SILENT_MODE**

Specifies whether to print all key attempts.

Type bool

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)

`CrackingCodesWithPython.Chapter15.affineHacker.hackAffine(`*message: str*`)`
Hacks affine cipher encrypted messages

Brute-forces a given encrypted message by looping through all the keys, checking if the result is English, and prompting the user for confirmation of decryption.

Parameters `message` – String with message to brute-force.

Returns Prints out possible results and prompts user for confirmation. If confirmed, prints out and returns full decrypted message, otherwise returns None.

`CrackingCodesWithPython.Chapter15.affineHacker.main()`

Module contents

CrackingCodesWithPython.Chapter16 package

Submodules

CrackingCodesWithPython.Chapter16.PracticeQuestions module

Chapter 16 Practice Questions

Answers Chapter 16 Practice Questions via Python code.

`CrackingCodesWithPython.Chapter16.PracticeQuestions.main()`

CrackingCodesWithPython.Chapter16.simpleSubCipher module

Simple Substitution Cipher

Provides functions that implement a substitution cipher.

`CrackingCodesWithPython.Chapter16.simpleSubCipher.LETTERS`
String containing uppercase latin letters.

Type str

Example

```
>>> import pythontutorials.books.CrackingCodesWithPython.Chapter16.simpleSubCipher as␣
↪simpleSubCipher
>>> key = simpleSubCipher.getRandomKey()  # key = 'VIAXLGJBKSZDUTRPYCEWNFHOMQ', in this␣
↪example
>>> message = 'You\'d be surprised what you can live through.'
>>> simpleSubCipher.encryptMessage(key, message)
"Mrn'x il encpckelx hbvw mrn avt dkfl wbcrnjb"
```

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)

CrackingCodesWithPython.Chapter16.simpleSubCipher.**decryptMessage**(*key: str*, *message: str*) → str

Substitution Cipher Decrypt

Wrapper function that decrypts given substitution cipher encrypted message with the given key.

Parameters

- **key** – String containing key used to decrypt substitution cipher.
- **message** – String containing message to decrypt.

Returns Decrypted message.

CrackingCodesWithPython.Chapter16.simpleSubCipher.**encryptMessage**(*key: str*, *message: str*) → str

Substitution Cipher Encrypt

Wrapper function that encrypts given message with the given key using the substitution cipher.

Parameters

- **key** – String containing key used to encrypt with substitution cipher.
- **message** – String containing message to encrypt.

Returns Encrypted message.

CrackingCodesWithPython.Chapter16.simpleSubCipher.**getRandomKey**() → str

Substitution cipher key generator

Generates a random key that can be used with the substitution cipher.

Returns String with a random, valid key.

CrackingCodesWithPython.Chapter16.simpleSubCipher.**keyIsValid**(*key: str*) → bool

Checks key for validity.

Ensures key contains all letters in LETTERS.

Parameters **key** – String containing key used to encrypt with substitution cipher.

Returns True if key and LETTERS match, False otherwise.

CrackingCodesWithPython.Chapter16.simpleSubCipher.**main**()

CrackingCodesWithPython.Chapter16.simpleSubCipher.**translateMessage**(*key: str*, *message: str*, *mode: str*) → str

Substitution Cipher

Implements a substitution cipher that can encrypt or decrypt messages depending on the given mode.

Parameters

- **key** – String containing key used to decrypt/encrypt messages.
- **message** – String containing message to decrypt/encrypt.
- **mode** – String specifying whether to 'encrypt' or 'decrypt'.

Returns Encrypted or decrypted message.

Module contents

CrackingCodesWithPython.Chapter17 package

Submodules

CrackingCodesWithPython.Chapter17.PracticeQuestions module

Chapter 17 Practice Questions

Answers Chapter 17 Practice Questions via Python code.

`CrackingCodesWithPython.Chapter17.PracticeQuestions.main()`

CrackingCodesWithPython.Chapter17.makeWordPatterns module

Make wordPatterns.py file

Creates *CrackingCodesWithPython.Chapter17.wordPatterns* based on the words in our dictionary text file, dictionary.txt. A word pattern assigns a number to each letter in a word, then generates a pattern representation of that word based on the number assigned to each letter.

`CrackingCodesWithPython.Chapter17.makeWordPatterns.DICTIONARY_FILE`
String containing absolute path to dictionary.txt file.

Type str

Note:

- Download the dictionary file from https://invpy.com/dictionary.txt
- https://www.nostarch.com/crackingcodes (BSD Licensed)

`CrackingCodesWithPython.Chapter17.makeWordPatterns.getWordPattern(`*word: str*`)` → str
Get word pattern

Returns a string of the pattern form of the given word.

Parameters **word** – String containing word to convert into word pattern.

Example: >>> import pythontutorials.books.CrackingCodesWithPython.Chapter17.makeWordPatterns as makeWordPatterns >>> makeWordPatterns.getWordPattern('DUSTBUSTER') '0.1.2.3.4.1.2.3.5.6'

Returns String containing word pattern.

`CrackingCodesWithPython.Chapter17.makeWordPatterns.main()`

CrackingCodesWithPython.Chapter17.simpleSubHacker module

Simple Substitution Cipher Hacker

Implements a function that can hack a substitution cipher encrypted message.

`CrackingCodesWithPython.Chapter17.simpleSubHacker.LETTERS`
String containing uppercase latin letters.

Type str

CrackingCodesWithPython.Chapter17.simpleSubHacker.**nonLettersOrSpacePattern**
Regular expression object representing all non-letter characters and space.

Type re._sre.SRE_Pattern

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)

CrackingCodesWithPython.Chapter17.simpleSubHacker.**addLettersToMapping**(*letterMapping: dict*, *cipherword: str*, *candidate: str*) → None
Add letters to cipherletter mapping

The letterMapping parameter takes a dictionary value that stores a cipherletter mapping, which is copied by the function. The cipherword parameter is a string value of the ciphertext word. The candidate parameter is a possible English word that the cipherword could decrypt to.

This function adds the letters in the candidate as potential decryption letters for the cipherletters in the cipherletter mapping.

Parameters

- **letterMapping** – Dictionary containing a cipherletter mapping.
- **cipherword** – String containing an encrypted ciphertext word.
- **candidate** – String containing an English word the cipherword could potentially decrypt to.

Returns None. Modifies contents of letterMapping by adding letters to the cipherletter mapping.

CrackingCodesWithPython.Chapter17.simpleSubHacker.**decryptWithCipherletterMapping**(*ciphertext: str*, *letterMapping: dict*) → str
Decrypt substitution cipher message with cipherletter map

Decrypts given substitution cipher encrypted message with given dictionary containing a cipherletter map.

Parameters

- **ciphertext** – Substitution cipher encrypted message to decrypt.
- **letterMapping** – Dictionary with cipherletter map that may decrypt the ciphertext.

Returns String containing decrypted ciphertext message.

Note:

- Ambiguous decrypted letters are replaced with an underscore, '_'

CrackingCodesWithPython.Chapter17.simpleSubHacker.**getBlankCipherletterMapping**() → dict

Get blank cipherletter mapping

Returns a dictionary value that is a blank cipherletter mapping

Returns Returns dictionary with uppercase latin letters as keys and empty lists as values.

CrackingCodesWithPython.Chapter17.simpleSubHacker.**hackSimpleSub**(*message: str*) → dict

Hack simple substitution cipher

Hacks simple substitution cipher and returns dictionary with cipherletter map that may be able to decrypt given message.

Parameters **message** – String containing substitution cipher encrypted message.

Returns Dictionary with cipherletter map that may decrypt given message.

CrackingCodesWithPython.Chapter17.simpleSubHacker.**intersectMappings**(*mapA: dict*, *mapB: dict*) → dict

Intersects two cipherletter mappings

Checks each letter in LETTERS and adds to intersected map if it exists in both given maps. If either map is empty, the non-empty map is copied to the intersected map.

Parameters

- **mapA** – Dictionary containing potential decryption letters.
- **mapB** – Dictionary containing potential decryption letters.

Returns Dictionary containing intersected map of potential decryption letters.

CrackingCodesWithPython.Chapter17.simpleSubHacker.**main**()

CrackingCodesWithPython.Chapter17.simpleSubHacker.**removeSolvedLettersFromMapping**(*letterMapping: dict*) → dict

Removes solved letters from cipherletter mapping

Cipherletters in the mapping that map to only one letter are "solved" and can be removed from the other letters.

For example, if 'A' maps to potential letters ['M', 'N'] and 'B' maps to ['N'], then we know that 'B' must map to 'N', so we can remove 'N' from the list of what 'A' could map to. So 'A' then maps to ['M'].

Note that now that 'A' maps to only one letter, we can remove 'M' from the list of letters for every other letter. (This is why there is a loop that keeps reducing the map.)

Parameters **letterMapping** – Cipherletter map dictionary to remove solved letters from.

Returns Dictionary containing cipherletter map with solved letters removed.

CrackingCodesWithPython.Chapter17.wordPatterns module

Word patterns file

Dictionary with word patterns as keys and a list of words matching the word pattern as values.

CrackingCodesWithPython.Chapter17.wordPatterns.**allPatterns**

Dictionary containing all word patterns in dictionary.txt

Type dict

Example

```
>>> {'0.0.1': ['EEL']}
```

Note:

- Docstring gets erased when wordPatterns.py is generated by *CrackingCodesWithPython.Chapter17.makeWordPatterns*

Module contents

CrackingCodesWithPython.Chapter18 package

Submodules

CrackingCodesWithPython.Chapter18.PracticeQuestions module

Chapter 18 Practice Questions

Answers Chapter 18 Practice Questions via Python code.

CrackingCodesWithPython.Chapter18.PracticeQuestions.**main**()

CrackingCodesWithPython.Chapter18.stringTest module

Create string test

Timing string concatenation vs list appending to make a string.

Note:

- Prints time to make a 10000 character string 10000 times as seconds since the Unix epoch.

CrackingCodesWithPython.Chapter18.stringTest.**main**()

CrackingCodesWithPython.Chapter18.vigenereCipher module

Vigenère Cipher (Polyalphabetic Substitution Cipher)

Provides functions that implement a Vigenère cipher.

CrackingCodesWithPython.Chapter18.vigenereCipher.**LETTERS**
String containing uppercase latin letters.

Type str

Example

```
>>> import pythontutorials.books.CrackingCodesWithPython.Chapter18.vigenereCipher as␣
↪vigenereCipher
>>> key = 'supercalifragilisticexpialidocious'
>>> message = 'A soul shines brightest when it stands alongside the darkness. -Anon,␣
↪probably'
>>> vigenereCipher.encryptMessage(key, message)
'S mdyc uhtvjj bxqrplxav aetv ie awoplg udghvwzfe epj uaxsymkl. -Ipsk, ezomieza'
```

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)

CrackingCodesWithPython.Chapter18.vigenereCipher.**decryptMessage**(*key: str*, *message: str*) → str

Vigenère cipher decryption

Wrapper function that decrypts given message with given key using the Vigenère cipher.

Parameters

- **key** – String decryption key to encrypt with Vigenère cipher.
- **message** – Message string to decrypt.

Returns Decrypted message string.

CrackingCodesWithPython.Chapter18.vigenereCipher.**encryptMessage**(*key: str*, *message: str*) → str

Vigenère cipher encryption

Wrapper function that encrypts given message with given key using the Vigenère cipher.

Parameters

- **key** – String encryption key to encrypt with Vigenère cipher.
- **message** – Message string to encrypt.

Returns Encrypted message string.

CrackingCodesWithPython.Chapter18.vigenereCipher.**main**()

CrackingCodesWithPython.Chapter18.vigenereCipher.**translateMessage**(*key: str*, *message: str*, *mode: str*) → str

Vigenère cipher

Implements a Vigenère cipher that can encrypt or decrypt messages depending on the given mode.

Parameters

- **key** – String containing key used to decrypt/encrypt messages.
- **message** – String containing message to decrypt/encrypt.
- **mode** – String specifying whether to 'encrypt' or 'decrypt'.

Returns Encrypted or decrypted message.

Module contents

CrackingCodesWithPython.Chapter19 package

Submodules

CrackingCodesWithPython.Chapter19.PracticeQuestions module

Chapter 19 Practice Questions

Answers Chapter 19 Practice Questions via Python code.

`CrackingCodesWithPython.Chapter19.PracticeQuestions.main()`

CrackingCodesWithPython.Chapter19.freqAnalysis module

Frequency Finder

Analyzes frequency of letters in given message compared to the most common occurring letters to determine if message is in the English language.

`CrackingCodesWithPython.Chapter19.freqAnalysis.ETAOIN`
String containing uppercase latin letters in order from most to least common.

Type str

`CrackingCodesWithPython.Chapter19.freqAnalysis.LETTERS`
String containing uppercase latin letters in alphabetical order.

Type str

Note:

- Compares six most and six least common letters in the English language.
- https://www.nostarch.com/crackingcodes/ (BSD Licensed)

`CrackingCodesWithPython.Chapter19.freqAnalysis.englishFreqMatchScore(`*message: str*`)` → int
English Frequency Match Score

Calculates number of matches that the string in the message parameter has when its letter frequency is compared to English letter frequency.

Parameters **message** – String containing message to calculate English match score.

Returns Number representing message's matches to English letter frequency.

Note:

- A "match" is how many of its six most frequent and six least frequent letters are among the six most frequent and six least frequent letters for English.
- A "perfect score" is 12

`CrackingCodesWithPython.Chapter19.freqAnalysis.getFrequencyOrder(`*message: str*`)` → str
Get frequency order

Analyzes frequency of each letter in given message and returns string with each letter from most to least frequent.

Parameters **message** – String containing message to analyze frequency.

Returns String of the alphabet letters arranged in order of most frequently occurring in the message parameter.

`CrackingCodesWithPython.Chapter19.freqAnalysis.getItemAtIndexZero(items: tuple)`
Get element at index zero

Helper function that returns the first element of a given tuple.

Parameters `items` – Tuple containing a latin letter and its frequency count.

Returns the latin letter.

Return type The first element of the given tuple

`CrackingCodesWithPython.Chapter19.freqAnalysis.getLetterCount(message: str)` → dict
Get letter count

Counts the frequency of all latin letters in a given message.

Parameters `message` – String containing message to analyze letter frequency.

Returns Dictionary with keys of single letters and values of the count of how many times they appear in the message parameter.

Module contents

CrackingCodesWithPython.Chapter20 package

Submodules

CrackingCodesWithPython.Chapter20.PracticeQuestions module

Chapter 20 Practice Questions

Answers Chapter 20 Practice Questions via Python code.

`CrackingCodesWithPython.Chapter20.PracticeQuestions.main()`

CrackingCodesWithPython.Chapter20.vigenereDictionaryHacker module

Vigenère Cipher Dictionary Hacker

Implements a function that can hack a Vigenère cipher encrypted message using a dictionary.

`CrackingCodesWithPython.Chapter20.vigenereDictionaryHacker.DICTIONARY_FILE`
String with absolute location of dictionary.txt file.

Type str

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)

`CrackingCodesWithPython.Chapter20.vigenereDictionaryHacker.hackVigenereDictionary(ciphertext: str)`
Hack Vigenère Dictionary

Brute-forces ciphertext by using every word in the dictionary file as a key. Checks if decrypted message is English with the `isEnglish()` module, and prompts user for confirmation by displaying first 100 characters.

Parameters `ciphertext` – String containing Vigenère cipher encrypted message.

Returns Decrypted message, if confirmed, None otherwise.

`CrackingCodesWithPython.Chapter20.vigenereDictionaryHacker.main()`

CrackingCodesWithPython.Chapter20.vigenereHacker module

Vigenère Cipher Hacker

Implements a series of functions that can hack a Vigenère cipher encrypted message by brute-forcing key lengths.

`CrackingCodesWithPython.Chapter20.vigenereHacker.LETTERS`
String with uppercase latin letters.

Type str

`CrackingCodesWithPython.Chapter20.vigenereHacker.MAX_KEY_LENGTH`
Will not attempt keys longer than this.

Type int

`CrackingCodesWithPython.Chapter20.vigenereHacker.NUM_MOST_FREQ_LETTERS`
Attempt this many letters per subkey.

Type int

`CrackingCodesWithPython.Chapter20.vigenereHacker.SILENT_MODE`
If set to True, program doesn't print anything.

Type bool

`CrackingCodesWithPython.Chapter20.vigenereHacker.NONLETTERS_PATTERN`
Regular expression object representing all non-letter characters.

Type re._sre.SRE_Pattern

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)

`CrackingCodesWithPython.Chapter20.vigenereHacker.attemptHackWithKeyLength(`*ciphertext: str*, *mostLikelyKeyLength: int*`)`

Attempt hack with key length

Brute-forces ciphertext using every key of a given length, checks if decrypted message is English with the `isEnglish()` module, and prompts user for confirmation by displaying first 200 characters.

Parameters

- `ciphertext` – String with encrypted message.
- `mostLikelyKeyLength` – Integer representing the length of the key used to encrypt message.

Returns Decrypted message, if confirmed, None otherwise.

Note:

- Key length is not limited to likely key lengths from *kasiskiExamination()*.

CrackingCodesWithPython.Chapter20.vigenereHacker.**findRepeatSequencesSpacings**(*message: str*) → dict

Find spacing between repeat sequences

Goes through the message and finds any 3- to 5-letter sequences that are repeated. Then counts the number of letters between the repeated sequences.

Parameters **message** – String with message to find repeat sequence spacing.

Returns Dictionary with the keys of the sequence and values of a list of spacings (num of letters between the repeats).

CrackingCodesWithPython.Chapter20.vigenereHacker.**getItemAtIndexOne**(*x: tuple*) → int

Get item at index one

Helper function that returns the second element of given tuple.

Parameters **x** – Tuple with integers as values.

Returns Second element of x.

CrackingCodesWithPython.Chapter20.vigenereHacker.**getMostCommonFactors**(*seqFactors: dict*) → list

Get most common factors

Counts how often each factor in the seqFactors dictionary occurs and returns a list of tuples with each factor and its count.

Parameters **seqFactors** – Dictionary with 3- to 5- letter sequences as keys and the factors of the spacings between them as values.

Returns A list of tuples of each factor and its count.

CrackingCodesWithPython.Chapter20.vigenereHacker.**getNthSubkeysLetters**(*nth: int*, *keyLength: int*, *message: str*) → str

Get nth subkeys letters

Gets every nth letter for each set of letters of a given length in a given text.

Parameters

- **nth** – Integer representing desired letter in message (similar to an index number).
- **keyLength** – Integer representing length of key to use (spacing between nth letters).
- **message** – String containing text to extract subkey letters from.

Returns String with every nth letter for each specified key length.

Examples

```
>>> getNthSubkeysLetters(1, 3, 'ABCABCABC')
'AAA'
>>> getNthSubkeysLetters(2, 3, 'ABCABCABC')
'BBB'
>>> getNthSubkeysLetters(3, 3, 'ABCABCABC')
'CCC'
>>> getNthSubkeysLetters(1, 5, 'ABCDEFGHI')
'AF'
```

CrackingCodesWithPython.Chapter20.vigenereHacker.**getUsefulFactors**(*num: int*) → list
Get useful factors

Returns a list of useful factors of num. By "useful" we mean factors less than MAX_KEY_LENGTH + 1 and not 1.

Parameters **num** – Integer to get useful factors of.

Returns List of useful factors, if found, empty list otherwise.

Example

```
>>> getUsefulFactors(144)
[2, 3, 4, 6, 8, 9, 12, 16]
```

CrackingCodesWithPython.Chapter20.vigenereHacker.**hackVigenere**(*ciphertext: str*)
Hack vigenere

Hacks Vigenère cipher encrypted message using likely key lengths, otherwise all possible key lengths.

Parameters **ciphertext** – String containing Vigenère cipher encrypted message.

Returns Decrypted message, if confirmed, None otherwise.

CrackingCodesWithPython.Chapter20.vigenereHacker.**kasiskiExamination**(*ciphertext: str*) → list
Kasiski Examination

Uses Kasiski Examination to determine the likely length of the key used to encrypt the given ciphertext.

Parameters **ciphertext** – String containing encrypted message.

Returns List of likely key lengths used to encrypt message.

CrackingCodesWithPython.Chapter20.vigenereHacker.**main**()

Module contents

CrackingCodesWithPython.Chapter21 package

Submodules

CrackingCodesWithPython.Chapter21.PracticeQuestions module

Chapter 21 Practice Questions

Answers Chapter 21 Practice Questions via Python code.

CrackingCodesWithPython.Chapter21.PracticeQuestions.main()

Module contents

CrackingCodesWithPython.Chapter22 package

Submodules

CrackingCodesWithPython.Chapter22.PracticeQuestions module

Chapter 22 Practice Questions

Answers Chapter 22 Practice Questions via Python code.

CrackingCodesWithPython.Chapter22.PracticeQuestions.main()

CrackingCodesWithPython.Chapter22.primeNum module

Prime Number Sieve

Implements a series of functions that determine if a given number is prime.

CrackingCodesWithPython.Chapter22.primeNum.LOW_PRIMES
List containing prime numbers <= 100 (aka 'low primes').

Type list

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)

CrackingCodesWithPython.Chapter22.primeNum.generateLargePrime(*keysize: int = 1024*) → int
Generate large prime number

Generates random numbers of given bit size until one is prime.

Parameters **keysize** – Number of bits prime number should be.

Returns Random prime number that is keysize bits in size.

Note:

- keysize defaults to 1024 bits.

CrackingCodesWithPython.Chapter22.primeNum.isPrime(*num: int*) → bool
Is prime

This function checks divisibility by LOW_PRIMES before calling rabinMiller().

Parameters **num** – Integer to check if prime.

Returns True if num is prime, False otherwise.

Note:

- If a number is divisible by a low prime number, it is not prime.

CrackingCodesWithPython.Chapter22.primeNum.**isPrimeTrialDiv**(*num: int*) → bool
Is prime trial division

Uses the trial division algorithm for testing if a given number is prime.

Parameters **num** – Integer to determine if prime.

Returns True if num is a prime number, otherwise False.

CrackingCodesWithPython.Chapter22.primeNum.**primeSieve**(*sieveSize: int*) → list
Prime sieve

Calculates prime numbers using the Sieve of Eratosthenes algorithm.

Parameters **sieveSize** – Largest number to check if prime starting from zero.

Returns List containing prime numbers from 0 to given number.

CrackingCodesWithPython.Chapter22.primeNum.**rabinMiller**(*num: int*) → bool
Rabin-Miller primality test

Uses the Rabin-Miller primality test to check if a given number is prime.

Parameters **num** – Number to check if prime.

Returns True if num is prime, False otherwise.

Note:

- The Rabin-Miller primality test relies on unproven assumptions, therefore it can return false positives when given a pseudoprime.

Module contents

CrackingCodesWithPython.Chapter23 package

Submodules

CrackingCodesWithPython.Chapter23.PracticeQuestions module

Chapter 23 Practice Questions

Answers Chapter 23 Practice Questions via Python code.

CrackingCodesWithPython.Chapter23.PracticeQuestions.**main**()

CrackingCodesWithPython.Chapter23.makePublicPrivateKeys module

Public Key Generator

Implements series of functions capable of creating a textbook RSA public/private keypair and saves them to text files.

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)
- 'Textbook/Plain' RSA keys are not secure and should not be used to encrypt sensitive data.

CrackingCodesWithPython.Chapter23.makePublicPrivateKeys.**generateKey**(*keySize: int*) → tuple

Generate public/private keypair

Creates public/private keys keySize bits in size.

Parameters **keySize** – Bit size to make public/private keys.

Returns Tuples containing the public and private keypair split into their two halves.

CrackingCodesWithPython.Chapter23.makePublicPrivateKeys.**main**()

CrackingCodesWithPython.Chapter23.makePublicPrivateKeys.**makeKeyFiles**(*name: str*, *keySize: int*) → None

Make key files

Creates two files 'x_pubkey.txt' and 'x_privkey.txt' (where x is the value in name) with the n,e and d,e integers written in them, delimited by a comma.

Parameters

- **name** – Name to append to public/private key files.
- **keySize** – Bit size to make public/private keys.

Returns None. Key files are created in current working directory.

Note:

- Checks if key files with given name already exist and exits with warning if so.

Module contents

CrackingCodesWithPython.Chapter24 package

Submodules

CrackingCodesWithPython.Chapter24.publicKeyCipher module

Public Key Cipher

Implements a series of functions capable of encrypting and decrypting with textbook RSA public/private keypairs.

CrackingCodesWithPython.Chapter24.publicKeyCipher.**SYMBOLS**
String with all characters to be encrypted/decrypted.

Type str

CrackingCodesWithPython.Chapter24.publicKeyCipher.**PUBLIC_KEY_PATH**
String with absolute location of public key file.

Type str

CrackingCodesWithPython.Chapter24.publicKeyCipher.PRIVATE_KEY_PATH
String with absolute location of private key file.

Type str

Note:

- https://www.nostarch.com/crackingcodes/ (BSD Licensed)
- The public and private keys are created by the *CrackingCodesWithPython.Chapter23.makePublicPrivateKeys* module.
- 'Textbook/Plain' RSA keys are not secure and should not be used to encrypt sensitive data.

CrackingCodesWithPython.Chapter24.publicKeyCipher.**decryptMessage**(*encryptedBlocks: list*, *messageLength: int*, *key: tuple*, *blockSize: int*) → str

Decrypt Message

Decrypts a list of encrypted block integers back to the original message string.

Parameters

- **encryptedBlocks** – List containing block integers encrypted with PUBLIC key.
- **messageLength** – Length of the original message.
- **key** – Tuple with PRIVATE key used to decryption.
- **blockSize** – Bit size of block integers (usually specified in PRIVATE key file).

Returns Original message before block integer conversion and PUBLIC key encryption.

Notes

- The original message length is required to properly decrypt the last block.
- Ensure to pass the PRIVATE key to decrypt.

CrackingCodesWithPython.Chapter24.publicKeyCipher.**encryptAndWriteToFile**(*messageFilename: str*, *keyFilename: str*, *message: str*, *blockSize: int = None*) → str

Encrypt and write to file

Using a key from a keyfile, encrypt the message and save it to a file.

Parameters

- **messageFilename** – String containing name of file to save encrypted message to.
- **keyFilename** – String containing absolute file path of PUBLIC key file.
- **message** – String containing message to encrypt and save.
- **blockSize** – Bit size of blocks of integers used to convert and encrypt message (usually specified in PUBLIC key file).

Returns Encrypted message string.

CrackingCodesWithPython.Chapter24.publicKeyCipher.**encryptMessage**(*message: str*, *key: tuple*, *blockSize: int*) → list

Encrypt message

Converts the message string into a list of block integers, and then encrypts each block integer.

Parameters

- **message** – String containing message to encrypt with PUBLIC key.
- **key** – Tuple with PUBLIC key used for encryption.
- **blockSize** – Bit size of block integers (usually specified in the PUBLIC key file).

Returns List of block integers encrypted with PUBLIC key.

Note:

- Ensure to pass the PUBLIC key to encrypt.

CrackingCodesWithPython.Chapter24.publicKeyCipher.**getBlocksFromText**(*message: str*, *blockSize: int*) → list

Get blocks from text

Converts a string message to a list of block integers.

Parameters

- **message** – String containing message to convert into blocks of integers.
- **blockSize** – Size of each block of integers.

Returns List with blocks of integers of the given size.

Note:

- If a character in the message is not in SYMBOLS, program exits with an error.

CrackingCodesWithPython.Chapter24.publicKeyCipher.**getTextFromBlocks**(*blockInts: list*, *messageLength: int*, *blockSize: int*) → str

Get text from blocks

Converts a list of block integers to the original message string.

Parameters

- **blockInts** – List of block integers of specified size.
- **messageLength** – Length of the original message.
- **blockSize** – Bit size of each block of integers.

Returns Original message string before block integer conversion.

Note:

- The original message length is needed to properly convert the last block integer.

CrackingCodesWithPython.Chapter24.publicKeyCipher.**main**()

CrackingCodesWithPython.Chapter24.publicKeyCipher.**readFromFileAndDecrypt**(*messageFilename: str*, *keyFilename: str*) → str

Read from file and decrypt

Using a key from a key file, read an encrypted message from a file and then decrypt it.

Parameters

- **messageFilename** – String containing name of file with encrypted message saved to it.
- **keyFilename** – String containing absolute file path of PRIVATE key file.

Returns Decrypted message string.

Note:

- Checks block size in key file and exits with error if too large.

CrackingCodesWithPython.Chapter24.publicKeyCipher.**readKeyFile**(*keyFilename: str*) → tuple

Read key from key file

Reads the given public/private key file and returns the key.

Parameters **keyFilename** – String containing absolute path to public/private key file.

Returns The key as a (n,e) or (n,d) tuple value.

Module contents

Module contents

1.7 Index

PYTHON MODULE INDEX

a

C