CHAPTER 1
MANIPULATING STRINGS

1. Working with Strings
2. Useful String Methods
3. Project: Password Locker
4. Project: Adding Bullets to Wiki Markup

1.1 Working with strings

String Literals

- String values begin and end with a single quote.
- But we want to use either double or single quotes within a string then we have a multiple ways to do it as shown below.

Double Quotes

- One benefit of using double quotes is that the string can have a single quote character in it.

```
>>> spam = "That is Alice's cat."
```

- Since the string begins with a double quote, Python knows that the single quote is part of the string and not marking the end of the string.

Escape Characters

- If you need to use both single quotes and double quotes in the string, you'll need to use escape characters.
- An escape character consists of a backslash (\) followed by the character you want to add to the string.

```
>>> spam = 'Say hi to Bob\'s mother.'
```

- Python knows that the single quote in Bob\'s has a backslash, it is not a single quote meant to end the string value. The escape characters \' and \" allows to put single quotes and double quotes inside your strings, respectively.

Ex:

```
>>> print("Hello there!\nHow are you?\nI\'m doing fine.")
Hello there!
How are you?
I'm doing fine.
```

- The different special escape characters can be used in a program as listed below in a table.

Escape character	Prints as
\'	Single quote
\"	Double quote
\t	Tab
\n	Newline (line break)
\\	Backslash

Raw Strings

- You can place an r before the beginning quotation mark of a string to make it a raw string. A raw string completely ignores all escape characters and prints any backslash that appears in the string

```
>>> print(r'That is Carol\'s cat.')
That is Carol\'s cat.
```

Multiline Strings with Triple Quotes

- A multiline string in Python begins and ends with either three single quotes or three double quotes.
- Any quotes, tabs, or newlines in between the "triple quotes" are considered part of the string.

Program

```
print('''Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and extortion.

Sincerely,
Bob''')
```

Output

```
Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and extortion.

Sincerely,
Bob
```

- The following print() call would print identical text but doesn't use a multiline string.

```
print('Dear Alice,\n\nEve\'s cat has been arrested for catnapping, cat
burglary, and extortion.\n\nSincerely,\nBob')
```

Multiline Comments

- While the hash character (#) marks the beginning of a comment for the rest of the line.
- A multiline string is often used for comments that span multiple lines.

```
"""This is a test Python program.
Written by Al Sweigart al@inventwithpython.com

This program was designed for Python 3, not Python 2.
"""

def spam():
    """This is a multiline comment to help
    explain what the spam() function does."""
    print('Hello!')
```

Indexing and Slicing Strings

- Strings use indexes and slices the same way lists do. We can think of the string 'Hello world!' as a list and each character in the string as an item with a corresponding index.

H	e	l	l	o		w	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11

- The space and exclamation point are included in the character count, so 'Hello world!' is 12 characters long.
- If we specify an index, you'll get the character at that position in the string.

```
>>> spam = 'Hello world!'
>>> spam[0]
'H'
>>> spam[4]
'o'
>>> spam[-1]
'!'
>>> spam[0:5]
'Hello'
>>> spam[:5]
'Hello'
>>> spam[6:]
'world!'
```

- If we specify a range from one index to another, the starting index is included and the ending index is not.

```
>>> spam = 'Hello world!'
>>> fizz = spam[0:5]
>>> fizz
'Hello'
```

- The substring we get from spam[0:5] will include everything from spam[0] to spam[4], leaving out the space at index 5.

The in and not in Operators with Strings

- The **in** and **not in** operators can be used with strings just like with list values.
- An expression with two strings joined using in or not in will evaluate to a Boolean True or False.

```
>>> 'Hello' in 'Hello World'
True
>>> 'Hello' in 'Hello'
True
>>> 'HELLO' in 'Hello World'
False
>>> '' in 'spam'
True
>>> 'cats' not in 'cats and dogs'
False
```

- These expressions test whether the first string (the exact string, case sensitive) can be found within the second string.

1.2 Useful String Methods

- Several string methods analyze strings or create transformed string values.

The upper(), lower(), isupper(), and islower() String Methods

- The upper() and lower() string methods return a new string where all the letters in the original string have been converted to uppercase or lowercase, respectively.

```
>>> spam = 'Hello world!'
>>> spam = spam.upper()
>>> spam
'HELLO WORLD!'
>>> spam = spam.lower()
>>> spam
'hello world!'
```

- These methods do not change the string itself but return new string values.
- If we want to change the original string, we have to call upper() or lower() on the string and then assign the new string to the variable where the original was stored.
- The upper() and lower() methods are helpful if we need to make a case-insensitive comparison.
- In the following small program, it does not matter whether the user types Great, GREAT, or grEAT, because the string is first converted to lowercase.

```
print('How are you?')
feeling = input()
if feeling.lower() == 'great':
    print('I feel great too.')
else:
    print('I hope the rest of your day is good.')
```

Program

```
How are you?
GREat
I feel great too.
```

Output

- The isupper() and islower() methods will return a Boolean True value if the string has at least one letter and all the letters are uppercase or lowercase, respectively. Otherwise, the method returns False.

```
>>> spam = 'Hello world!'
>>> spam.islower()
False
>>> spam.isupper()
False
>>> 'HELLO'.isupper()
True
>>> 'abc12345'.islower()
True
>>> '12345'.islower()
False
>>> '12345'.isupper()
False
```

- Since the upper() and lower() string methods themselves return strings, you can call string methods on those returned string values as well. Expressions that do this will look like a chain of method calls.

```
>>> 'Hello'.upper()
'HELLO'
>>> 'Hello'.upper().lower()
'hello'
>>> 'Hello'.upper().lower().upper()
'HELLO'
>>> 'HELLO'.lower()
'hello'
>>> 'HELLO'.lower().islower()
True
```

The isX String Methods

- There are several string methods that have names beginning with the word is. These methods return a Boolean value that describes the nature of the string.
- Here are some common isX string methods:
  - **isalpha()** returns True if the string consists only of letters and is not blank.
  - **isalnum()** returns True if the string consists only of letters and numbers and is not blank.
  - **isdecimal()** returns True if the string consists only of numeric characters and is not blank.
  - **isspace()** returns True if the string consists only of spaces, tabs, and newlines and is not blank.
  - **istitle()** returns True if the string consists only of words that begin with an uppercase letter followed by only lowercase letters.

```
>>> 'hello'.isalpha()
True
>>> 'hello123'.isalpha()
False
>>> 'hello123'.isalnum()
True
>>> 'hello'.isalnum()
True
>>> '123'.isdecimal()
True
>>> '    '.isspace()
True
>>> 'This Is Title Case'.istitle()
True
>>> 'This Is Title Case 123'.istitle()
True
>>> 'This Is not Title Case'.istitle()
False
>>> 'This Is NOT Title Case Either'.istitle()
False
```

- The isX string methods are helpful when you need to validate user input.
- For example, the following program repeatedly asks users for their age and a password until they provide valid input.

Program

```
while True:
    print('Enter your age:')
    age = input()
    if age.isdecimal():
        break
    print('Please enter a number for your age.')

while True:
    print('Select a new password (letters and numbers only):')
    password = input()
    if password.isalnum():
        break
    print('Passwords can only have letters and numbers.')
```

output

```
Enter your age:
forty two
Please enter a number for your age.
Enter your age:
42
Select a new password (letters and numbers only):
secr3t!
Passwords can only have letters and numbers.
Select a new password (letters and numbers only):
secr3t
```

The startswith() and endswith() String Methods

- The startswith() and endswith() methods return True if the string value they are called on begins or ends (respectively) with the string passed to the method; otherwise, they return False.

```
>>> 'Hello world!'.startswith('Hello')
True
>>> 'Hello world!'.endswith('world!')
True
>>> 'abc123'.startswith('abcdef')
False
>>> 'abc123'.endswith('12')
False
>>> 'Hello world!'.startswith('Hello world!')
True
>>> 'Hello world!'.endswith('Hello world!')
True
```

- These methods are useful alternatives to the == equals operator if we need to check only whether the first or last part of the string, rather than the whole thing, is equal to another string.

The join() and split() String Methods

Join()

- The join() method is useful when we have a list of strings that need to be joined together into a single string value.
- The join() method is called on a string, gets passed a list of strings, and returns a string. The returned string is the concatenation of each string in the passed-in list.

```
>>> ', '.join(['cats', 'rats', 'bats'])
'cats, rats, bats'
>>> ' '.join(['My', 'name', 'is', 'Simon'])
'My name is Simon'
>>> 'ABC'.join(['My', 'name', 'is', 'Simon'])
'MyABCnameABCisABCSimon'
```

- string join() calls on is inserted between each string of the list argument.
  - **Ex:** when join(['cats', 'rats', 'bats']) is called on the ', ' string, the returned string is 'cats, rats, bats'.
  - join() is called on a string value and is passed a list value.

Split()

- The split() method is called on a string value and returns a list of strings.

```
>>> 'My name is Simon'.split()
['My', 'name', 'is', 'Simon']
```

- We can pass a delimiter string to the split() method to specify a different string to split upon.

```
>>> 'MyABCnameABCisABCSimon'.split('ABC')
['My', 'name', 'is', 'Simon']
>>> 'My name is Simon'.split('m')
['My na', 'e is Si', 'on']
```

- common use of split() is to split a multiline string along the newline characters.

```
>>> spam = '''Dear Alice,
How have you been? I am fine.
There is a container in the fridge
that is labeled "Milk Experiment".

Please do not drink it.
Sincerely,
Bob'''
>>> spam.split('\n')
['Dear Alice,', 'How have you been? I am fine.', 'There is a container in the
fridge', 'that is labeled "Milk Experiment".', '', 'Please do not drink it.',
'Sincerely,', 'Bob']
```

- Passing split() the argument '\n' lets us split the multiline string stored in spam along the newlines and return a list in which each item corresponds to one line of the string.

Justifying Text with rjust(), ljust(), and center()

- The rjust() and ljust() string methods return a padded version of the string they are called on, with spaces inserted to justify the text.
- The **first** argument to both methods is an integer length for the justified string.

```
>>> 'Hello'.rjust(10)
'     Hello'
>>> 'Hello'.rjust(20)
'               Hello'
>>> 'Hello World'.rjust(20)
'         Hello World'
>>> 'Hello'.ljust(10)
'Hello     '
```

- 'Hello'.rjust(10) says that we want to right-justify 'Hello' in a string of total length 10. 'Hello' is five characters, so five spaces will be added to its left, giving us a string of 10 characters with 'Hello' justified right.
- An optional **second** argument to rjust() and ljust() will specify a fill character other than a space character.

```
>>> 'Hello'.rjust(20, '*')
'***************Hello'
>>> 'Hello'.ljust(20, '-')
'Hello---------------'
```

- The center() string method works like ljust() and rjust() but centers the text rather than justifying it to the left or right.

```
>>> 'Hello'.center(20)
'       Hello        '
>>> 'Hello'.center(20, '=')
'=======Hello========'
```

- These methods are especially useful when you need to print tabular data that has the correct spacing.
- In the below program, we define a printPicnic() method that will take in a dictionary of information and use center(), ljust(), and rjust() to display that information in a neatly aligned table-like format.
    - The dictionary that we'll pass to printPicnic() is picnicItems.
    - In picnicItems, we have 4 sandwiches, 12 apples, 4 cups, and 8000 cookies. We want to organize this information into two columns, with the name of the item on the left and the quantity on the right.

Program

```
def printPicnic(itemsDict, leftWidth, rightWidth):
    print('PICNIC ITEMS'.center(leftWidth + rightWidth, '-'))
    for k, v in itemsDict.items():
        print(k.ljust(leftWidth, '.') + str(v).rjust(rightWidth))
picnicItems = {'sandwiches': 4, 'apples': 12, 'cups': 4, 'cookies': 8000}
printPicnic(picnicItems, 12, 5)
printPicnic(picnicItems, 20, 6)
```

output

```
---PICNIC ITEMS--
sandwiches..     4
apples......    12
cups........     4
cookies..... 8000
-------PICNIC ITEMS-------
sandwiches.....      4
apples.............    12
cups...............      4
cookies.............  8000
```

Removing Whitespace with strip(), rstrip(), and lstrip()

- The strip() string method will return a new string without any whitespace characters at the beginning or end.
- The lstrip() and rstrip() methods will remove whitespace characters from the left and right ends, respectively.

```
>>> spam = '    Hello World     '
>>> spam.strip()
'Hello World'
>>> spam.lstrip()
'Hello World     '
>>> spam.rstrip()
'    Hello World'
```

- Optionally, a string argument will specify which characters on the ends should be stripped.

```
>>> spam = 'SpamSpamBaconSpamEggsSpamSpam'
>>> spam.strip('ampS')
'BaconSpamEggs'
```

- Passing strip() the argument 'ampS' will tell it to strip occurences of a, m, p, and capital S from the ends of the string stored in spam.
- The order of the characters in the string passed to strip() does not matter: strip('ampS') will do the same thing as strip('mapS') or strip('Spam').

Copying and Pasting Strings with the pyperclip Module

- The pyperclip module has copy() and paste() functions that can send text to and receive text from your computer's clipboard.

```
>>> import pyperclip
>>> pyperclip.copy('Hello world!')
>>> pyperclip.paste()
'Hello world!'
```

- Of course, if something outside of your program changes the clipboard contents, the paste() function will return it.

```
>>> pyperclip.paste()
'For example, if I copied this sentence to the clipboard and then called
paste(), it would look like this:'
```

1.3 Project: Password Locker

- We probably have accounts on many different websites.
- It's a bad habit to use the same password for each of them because if any of those sites has a security breach, the hackers will learn the password to all of your other accounts.
- It's best to use password manager software on your computer that uses one master password to unlock the password manager.
- Then you can copy any account password to the clipboard and paste it into the website's Password field

- The password manager program you'll create in this example isn't secure, but it offers a basic demonstration of how such programs work.

Step 1: Program Design and Data Structures

- We have to run this program with a command line argument that is the account's name--for instance, email or blog. That account's password will be copied to the clipboard so that the user can paste it into a Password field. The user can have long, complicated passwords without having to memorize them.
- We need to start the program with a #! (shebang) line and should also write a comment that briefly describes the program. Since we want to associate each account's name with its password, we can store these as strings in a dictionary.

```
#! python3
# pw.py - An insecure password locker program.

PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
             'blog': 'VmALvQyKAxiVH5G8vO1if1MLZF3sdt',
             'luggage': '12345'}
```

Step 2: Handle Command Line Arguments

- The command line arguments will be stored in the variable sys.argv.
- The **first** item in the sys.argv list should always be a string containing the program's filename ('pw.py'), and the **second** item should be the first command line argument.

```
#! python3
# pw.py - An insecure password locker program.

PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
             'blog': 'VmALvQyKAxiVH5G8vO1if1MLZF3sdt',
             'luggage': '12345'}

import sys
if len(sys.argv) < 2:
    print('Usage: python pw.py [account] - copy account password')
    sys.exit()

account = sys.argv[1]    # first command line arg is the account name
```

Step 3: Copy the Right Password

- The account name is stored as a string in the variable account, you need to see whether it exists in the PASSWORDS dictionary as a key. If so, you want to copy the key's value to the clipboard using pyperclip.copy().

```
#! python3
# pw.py - An insecure password locker program.
PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
             'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
             'luggage': '12345'}

import sys, pyperclip
if len(sys.argv) < 2:
    print('Usage: py pw.py [account] - copy account password')
    sys.exit()

account = sys.argv[1]    # first command line arg is the account name

if account in PASSWORDS:
    pyperclip.copy(PASSWORDS[account])
    print('Password for ' + account + ' copied to clipboard.')
else:
    print('There is no account named ' + account)
```

- This new code looks in the PASSWORDS dictionary for the account name. If the account name is a key in the dictionary, we get the value corresponding to that key, copy it to the clipboard, and print a message saying that we copied the value. Otherwise, we print a message saying there's no account with that name.
- On Windows, you can create a batch file to run this program with the win-R Run window. Type the following into the file editor and save the file as pw.bat in the C:\Windows folder:

```
@py.exe C:\Python34\pw.py %*
@pause
```

- With this batch file created, running the password-safe program on Windows is just a matter of pressing win-R and typing pw <account name>.

1.4 Project: Adding Bullets to Wiki Markup

- When editing a Wikipedia article, we can create a bulleted list by putting each list item on its own line and placing a star in front.
- But say we have a really large list that we want to add bullet points to. We could just type those stars at the beginning of each line, one by one. Or we could automate this task with a short Python script.
- The bulletPointAdder.py script will get the text from the clipboard, add a star and space to the beginning of each line, and then paste this new text to the clipboard.

- **Ex:**

Program

```
Lists of animals
Lists of aquarium life
Lists of biologists by author abbreviation
Lists of cultivars
```

output

```
* Lists of animals
* Lists of aquarium life
* Lists of biologists by author abbreviation
* Lists of cultivars
```

Step 1: Copy and Paste from the Clipboard

- You want the bulletPointAdder.py program to do the following:
    1. Paste text from the clipboard
    2. Do something to it
    3. Copy the new text to the clipboard
- Steps 1 and 3 are pretty straightforward and involve the pyperclip.copy() and pyperclip.paste() functions. saving the following program as bulletPointAdder.py:

```
#! python3
# bulletPointAdder.py - Adds Wikipedia bullet points to the start
# of each line of text on the clipboard.

import pyperclip
text = pyperclip.paste()
# TODO: Separate lines and add stars.

pyperclip.copy(text)
```

Step 2: Separate the Lines of Text and Add the Star

- The call to pyperclip.paste() returns all the text on the clipboard as one big string. If we used the "List of Lists of Lists" example, the string stored in text.
- The \n newline characters in this string cause it to be displayed with multiple lines when it is printed or pasted from the clipboard.
- We could write code that searches for each \n newline character in the string and then adds the star just after that. But it would be easier to use the split() method to return a list of strings, one for each line in the original string, and then add the star to the front of each string in the list.

```
#! python3
# bulletPointAdder.py - Adds Wikipedia bullet points to the start
# of each line of text on the clipboard.

import pyperclip
text = pyperclip.paste()

# Separate lines and add stars.
lines = text.split('\n')
for i in range(len(lines)):    # loop through all indexes in the "lines" list
    lines[i] = '* ' + lines[i] # add star to each string in "lines" list

pyperclip.copy(text)
```

- We split the text along its newlines to get a list in which each item is one line of the text. For each line, we add a star and a space to the start of the line. Now each string in lines begins with a star

Step 3: Join the Modified Lines

- The lines list now contains modified lines that start with stars.
- pyperclip.copy() is expecting a single string value, not a list of string values. To make this single string value, pass lines into the join() method to get a single string joined from the list's strings.

```
#! python3
# bulletPointAdder.py - Adds Wikipedia bullet points to the start
# of each line of text on the clipboard.

import pyperclip
text = pyperclip.paste()

# Separate lines and add stars.
lines = text.split('\n')
for i in range(len(lines)):    # loop through all indexes for "lines" list
    lines[i] = '* ' + lines[i] # add star to each string in "lines" list
text = '\n'.join(lines)
pyperclip.copy(text)
```

- When this program is run, it replaces the text on the clipboard with text that has stars at the start of each line.

CHAPTER -2 READ ING AND WRITING FI LES

1. Files and File Paths

- A file has two key properties: a filename (usually written as one word) and a path.
- The part of the filename after the last period is called the file's extension and tells you a file's type. project.docx is a Word document, and Users, asweigart, and Documents all refer to folders
- Folders can contain files and other folders. For example, project.docx s in the Documents folder, which is inside the asweigart folder, which is inside the Users folder.

1.1 Backslash on Windows and Forward Slash on OS X and Linux

- On Windows, paths are written using backslashes (\) as the separator between folder names. OS X and Linux, however, use the forward slash (/) as their path separator.
- Fortunately, this is simple to do with the os.path.join() function. If you os.path.join() will return a string with a file path using the correct path separators.

```
>>> import os
>>> os.path.join('usr', 'bin', 'spam')
'usr\\bin\\spam'
```

- The os.path.join() function is helpful if you need to create strings for filenames.

Program:

```
>>> myFiles = ['accounts.txt', 'details.csv', 'invite.docx']
>>> for filename in myFiles:
print(os.path.join('C:\\Users\\asweigart', filename))
C:\Users\asweigart\accounts.txt
C:\Users\asweigart\details.csv
C:\Users\asweigart\invite.docx
```

1.2 The Current Working Directory

- Every program that runs on your computer has a *current working directory*, or *cwd*
- Any filenames or paths that do not begin with the root folder are assumed to be under the current working directory.
- can get the current working directory as a string value with the os.getcwd() function and change it with os.chdir().

```
>>> import os
>>> os.getcwd()
'C:\\Python34'
>>> os.chdir('C:\\Windows\\System32')
>>> os.getcwd()
'C:\\Windows\\System32'
```

- The current working directory is set to C:\Python34, so the filename project.docx refers to C:\Python34\project.docx.
- When we change the current working directory to C:\Windows, project.docx is interpreted as C:\Windows\project.docx.
- Python will display an error if you try to change to a directory that does not exist.

```
>>> os.chdir('C:\\ThisFolderDoesNotExist')
Traceback (most recent call last):
File "<pyshell#18>", line 1, in <module>
os.chdir('C:\\ThisFolderDoesNotExist')
FileNotFoundError: [WinError 2] The system cannot find the file specified:
'C:\\ThisFolderDoesNotExist'
```

1.3 Absolute vs. Relative Paths

There are two ways to specify a file path.

- An absolute path, which always begins with the root folder
- A relative path, which is relative to the program's current working directory

- There are also the *dot* (.) and *dot-dot* (..) folders. These are not real folders but special names that can be used in a path
- A single period ("dot") for a folder name is shorthand for "this directory." Two periods ("dot-dot") means "the parent folder."
- When the current working directory is set to *C:\bacon*, the relative paths for the other folders and files are set as they are in the figure.

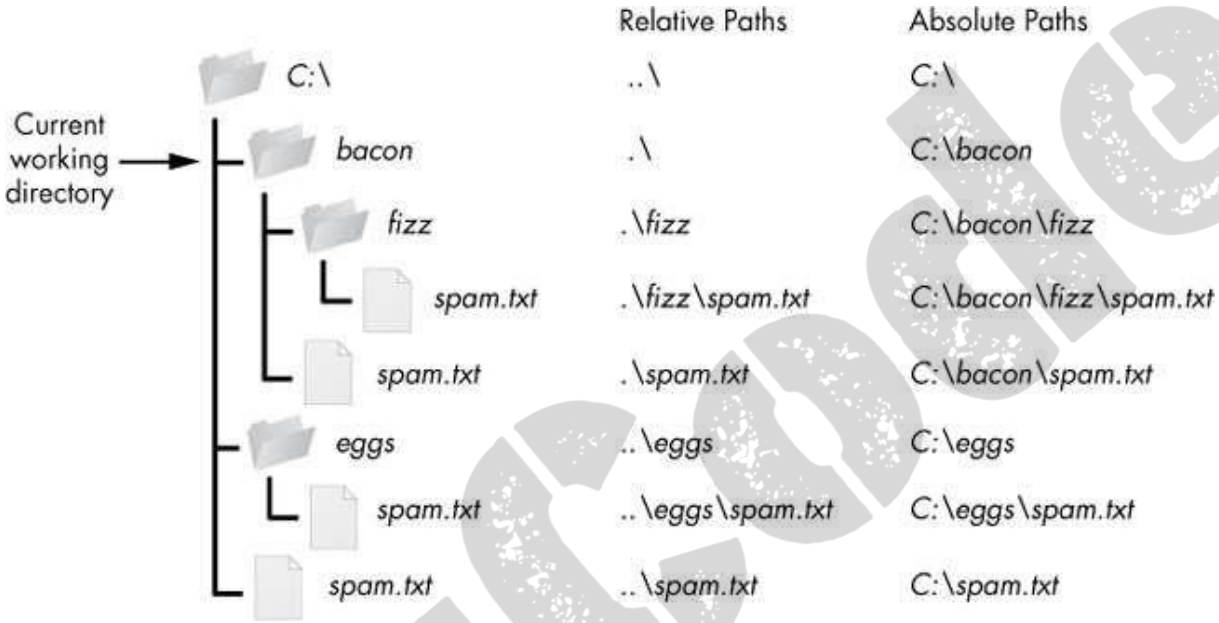

Figure 8-2: The relative paths for folders and files in the working directory C:\bacon

- The .\ at the start of a relative path is optional. For example, *.\spam.txt* and *spam.txt* refer to the same file.

1.4 Creating New Folders with os.makedirs()

- Your programs can create new folders (directories) with the os.makedirs() function.

```
>>> import os
>>> os.makedirs('C:\\delicious\\walnut\\waffles')
```

- This will create not just the C:\delicious folder but also a walnut folder inside C:\delicious and a waffles folder inside C:\delicious\walnut.

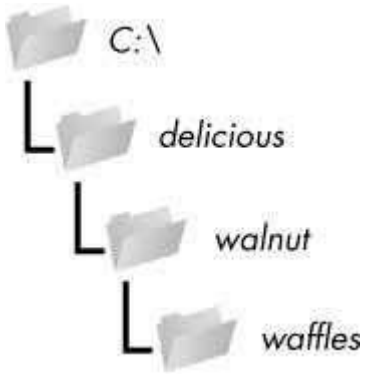

Figure : The result of os.makedirs('C:\\delicious \\walnut\\waffles')

2. The os.path Module

- The os.path module contains many helpful functions related to filenames and file paths
- Since os.path is a module inside the os module, you can import it by simply running import os.

2.1 Handling Absolute and Relative Paths

- The os.path module provides functions for returning the absolute path of a relative path and for checking whether a given path is an absolute path.
- Calling os.path.abspath(path) will return a string of the absolute path of the argument. This is an easy way to convert a relative path into an absolute one.
- Calling os.path.isabs(path) will return True if the argument is an absolute path and False if it is a relative path
- Calling os.path.relpath(path, start) will return a string of a relative path from the start path to path. If start is not provided, the current working directory is used as the start path.

```
>>> os.path.abspath('.')
'C:\\Python34'
>>> os.path.abspath('.\\Scripts')
'C:\\Python34\\Scripts'
>>> os.path.isabs('.')
False
>>> os.path.isabs(os.path.abspath('.'))
True
```

- Enter the following calls to os.path.relpath() into the interactive shell

Program:

```
>>> os.path.relpath('C:\\Windows', 'C:\\')
'Windows'
>>> os.path.relpath('C:\\Windows', 'C:\\spam\\eggs')
'..\\..\\Windows'
>>> os.getcwd()
'C:\\Python34'
```

- Calling os.path.dirname(*path*) will return a string of everything that comes before the last slash in the path argument
- Calling os.path.basename(*path*) will return a string of everything that comes after the last slash in the path argument.

Figure: The base name follows the last slash in a path and is the same as the filename. The dirname is everything before the last slash.

For example

```
>>> path = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.basename(path)
'calc.exe'
>>> os.path.dirname(path)
'C:\\Windows\\System32'
```

- If you need a path's dir name and base name together, you can just call os.path.split() to get a tuple value with these two strings

Program:

```
>>> calcFilePath = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.split(calcFilePath)
('C:\\Windows\\System32', 'calc.exe')
```

- you could create the same tuple by calling os.path.dirname() and os.path.basename() and placing their return values in a tuple.

```
>>> (os.path.dirname(calcFilePath), os.path.basename(calcFilePath))
('C:\\Windows\\System32', 'calc.exe')
```

- But os.path.split() is a nice shortcut if you need both values.
- os.path.split() does *not* take a file path and return a list of strings of each folder. For that, use the split() string method and split on the string in os.sep.

For example,

```
>>> calcFilePath.split(os.path.sep)
['C:', 'Windows', 'System32', 'calc.exe']
```

- On OS X and Linux systems, there will be a blank string at the start of the returned list:

```
>>> '/usr/bin'.split(os.path.sep)
['', 'usr', 'bin']
```

2.2 Finding File Sizes and Folder Contents

- folders. The os.path module provides functions for finding the size of a file in bytes and the files and folders inside a given folder
- Calling os.path.getsize(*path*) will return the size in bytes of the file in the *path* argument.
- Calling os.listdir(*path*) will return a list of filename strings for each file in the *path* argument. (Note that this function is in the os module, not os.path.)

Program:

```
>>> os.path.getsize('C:\\Windows\\System32\\calc.exe')
776192
>>> os.listdir('C:\\Windows\\System32')
['0409', '12520437.cpx', '12520850.cpx', '5U877.ax', 'aaclient.dll',
--snip--
'xwtpdui.dll', 'xwtpw32.dll', 'zh-CN', 'zh-HK', 'zh-TW', 'zipfldr.dll']
```

- If you want to find the total size of all the files in this directory, I can use os.path.getsize() and os.listdir() together.

Program:

```
>>> totalSize = 0
>>> for filename in os.listdir('C:\\Windows\\System32'):
totalSize = totalSize + os.path.getsize(os.path.join('C:\\Windows\\System32',
filename))
>>> print(totalSize)
1117846456
```

2.3 Checking Path Validity

- The os.path module provides functions to check whether a given path exists and whether it is a file or folder.
- Calling os.path.exists(*path*) will return True if the file or folder referred to in the argument exists and will return False if it does not exist.
- Calling os.path.isfile(*path*) will return True if the path argument exists and is a file and will return False otherwise.
- Calling os.path.isdir(*path*) will return True if the path argument exists and is a folder and will return False otherwise.

```
>>> os.path.exists('C:\\Windows')
True
>>> os.path.exists('C:\\some_made_up_folder')
False
>>> os.path.isdir('C:\\Windows\\System32')
True
>>> os.path.isfile('C:\\Windows\\System32')
False
>>> os.path.isdir('C:\\Windows\\System32\\calc.exe')
False
>>> os.path.isfile('C:\\Windows\\System32\\calc.exe')
True
```

3. The File Reading/Writing Process

- *Plaintext files* contain only basic text characters and do not include font, size, or color information.
- Text files with the *.txt* extension or Python script files with the *.py* extension are examples of plaintext files.
- These can be opened with Windows's Notepad or OS X's TextEdit application.
- *Binary files* are all other file types, such as word processing documents, PDFs, images, spreadsheets, and executable programs.
- If you open a binary file in Notepad or TextEdit, it will look like scrambled

Figure: The Windows calc.exe program opened in Notepad

- Since every different type of binary file must be handled in its own way, this book will not go into reading and writing raw binary files directly.
- There are three steps to reading or writing files in Python.
    1. Call the open() function to return a File object.
    2. Call the read() or write() method on the File object.
    3. Close the file by calling the close() method on the File object.

3.1 Opening Files with the open() Function

- To open a file with the open() function, you pass it a string path indicating the file you want to open; it can be either an absolute or relative path.
- The open() function returns a File object.
- Try it by creating a text file named *hello.txt* using Notepad or TextEdit. Type Hello world! as the content of this text file and save it in your user home folder.

```
>>> helloFile = open('C:\\Users\\your_home_folder\\hello.txt')
```

- If you're using OS X, enter the following into the interactive shell instead:

```
>>> helloFile = open('/Users/your_home_folder/hello.txt')
```

- When a file is opened in read mode, Python lets you only read data from the file; you can't write or modify it in any way.

- Read mode is the default mode for files you open in Python.
- if you don't want to rely on Python's defaults, you can explicitly specify the mode by passing the string value 'r' as a second argument to open().
- open('/Users/asweigart/ hello.txt', 'r') and open('/Users/asweigart/hello.txt')

3.2 Reading the Contents of Files

- If you want to read the entire contents of a file as a string value, use the File object's read() method

```
>>> helloContent = helloFile.read()
>>> helloContent
'Hello world!'
```

- Alternatively, you can use the readlines() method to get a *list* of string values from the file, one string for each line of text.
- For example, create a file named *sonnet29.txt* in the same directory as *hello.txt* and write the following text in it:
- Make sure to separate the four lines with line breaks

When, in disgrace with fortune and men's eyes,

I all alone beweep my outcast state,

And trouble deaf heaven with my bootless cries,

And look upon myself and curse my fate,

```
>>> sonnetFile = open('sonnet29.txt')
>>> sonnetFile.readlines()
[When, in disgrace with fortune and men's eyes,\n', ' I all alone beweep my outcast state,\n', And trouble deaf heaven with my bootless cries,\n', And look upon myself and curse my fate,']
```

3.3 Writing to Files

- Python allows you to write content to a file in a way similar to how the print() function "writes" strings to the screen.
- You can't write to a file you've opened in read mode, though. Instead, you need to open it in "write plaintext" mode or "append plaintext" mode, or write mode and append mode for short.
- Write mode will overwrite the existing file and start from scratch, just like when you overwrite a variable's value with a new value
- Pass 'w' as the second argument to open() to open the file in write mode Append mode, on the other hand, will append text to the end of the existing file.
- Pass 'a' as the second argument to open() to open the file in append mode.
- If the filename passed to open() does not exist, both write and append mode will create a new, blank file.
- Example:

```
>>> baconFile = open('bacon.txt', 'w')
>>> baconFile.write('Hello world!\n')
13
>>> baconFile.close()
>>> baconFile = open('bacon.txt', 'a')
>>> baconFile.write('Bacon is not a vegetable.')
25
>>> baconFile.close()
>>> baconFile = open('bacon.txt')
>>> content = baconFile.read()
>>> baconFile.close()
>>> print(content)
Hello world!
Bacon is not a vegetable.
```

4. Saving Variables with the shelve Module

- You can save variables in your Python programs to binary shelf files using the shelve module.
- This way, your program can restore data to variables from the hard drive.
- The shelve module will let you add Save and Open features to your program.
- For example, if you ran a program and entered some configuration settings, you could save those settings to a shelf file and then have the program load them the next time it is run.

```
>>> import shelve
>>> shelfFile = shelve.open('mydata')
>>> cats = ['Zophie', 'Pooka', 'Simon']
>>> shelfFile['cats'] = cats
>>> shelfFile.close()
```

- To read and write data using the shelve module, you first import shelve. Call shelve.open() and pass it a filename, and then store the returned shelf value in a variable.
- create a list cats and write shelfFile['cats'] = cats to store the list in shelfFile as a value associated with the key 'cats' (like in a dictionary). Then we call close() on shelfFile.
- programs can use the shelve module to later reopen and retrieve the data from these shelf files
- Shelf values don't have to be opened in read or write mode—they can do both once opened
- **Program:**

```
>>> shelfFile = shelve.open('mydata')
>>> type(shelfFile)
<class 'shelve.DbfilenameShelf'>
>>> shelfFile['cats']
['Zophie', 'Pooka', 'Simon']
>>> shelfFile.close()
```

- Just like dictionaries, shelf values have keys() and values() methods that will return list-like values of the keys and values in the shelf
- Since these methods return list-like values instead of true lists, you should pass them to the list() function to get them in list form

```
>>> shelfFile = shelve.open('mydata')
>>> list(shelfFile.keys())
['cats']
>>> list(shelfFile.values())
[['Zophie', 'Pooka', 'Simon']]
>>> shelfFile.close()
```

5. Saving Variables with the pprint.pformat() Function

- pprint.pprint() function will "pretty print" the contents of a list or dictionary to the screen,
- while the pprint.pformat() function will return this same text as a string instead of printing it.
- file will be your very own module that you can import whenever you want to use the variable stored in it.
- **For example,**

```
>>> import pprint
>>> cats = [{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc': 'fluffy'}]
>>> pprint.pformat(cats)
"[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy', 'name': 'Pooka'}]"
>>> fileObj = open('myCats.py', 'w')
>>> fileObj.write('cats = ' + pprint.pformat(cats) + '\n')
83
>>> fileObj.close()
```

- Python programs can even generate other Python programs. You can then import these files into scripts.

```
>>> import myCats
>>> myCats.cats
[{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc': 'fluffy'}]
>>> myCats.cats[0]
{'name': 'Zophie', 'desc': 'chubby'}
>>> myCats.cats[0]['name']
'Zophie'
```

6. Project: Generating Random Quiz Files

- The program does:
  - Creates 35 different quizzes.
  - Creates 50 multiple-choice questions for each quiz, in random order.
  - Provides the correct answer and three random wrong answers for each question, in random order.
  - Writes the quizzes to 35 text files.
  - Writes the answer keys to 35 text files.

- This means the code will need to do the following:
  - Store the states and their capitals in a dictionary.
  - Call open(), write(), and close() for the quiz and answer key text files.
  - Use random.shuffle() to randomize the order of the questions and multiple-choice options.

Step 1: Store the Quiz Data in a Dictionary

- The first step is to create a skeleton script and fill it with your quiz data. Create a file named randomQuizGenerator.py,

```
#! python3
# randomQuizGenerator.py - Creates quizzes with questions and answers in
# random order, along with the answer key.

import random
# The quiz data. Keys are states and values are their capitals.
capitals = {'Alabama': 'Montgomery', 'Alaska': 'Juneau', 'Arizona': 'Phoenix',
'Arkansas': 'Little Rock', 'California': 'Sacramento', 'Colorado': 'Denver',
'Connecticut': 'Hartford', 'Delaware': 'Dover', 'Florida': 'Tallahassee',
'Georgia': 'Atlanta', 'Hawaii': 'Honolulu', 'Idaho': 'Boise', 'Illinois':
'Springfield', 'Indiana': 'Indianapolis', 'Iowa': 'Des Moines', 'Kansas':
'Topeka', 'Kentucky': 'Frankfort', 'Louisiana': 'Baton Rouge', 'Maine':
'Augusta', 'Maryland': 'Annapolis', 'Massachusetts': 'Boston', 'Michigan':
'Lansing', 'Minnesota': 'Saint Paul', 'Mississippi': 'Jackson', 'Missouri':
'Jefferson City', 'Montana': 'Helena', 'Nebraska': 'Lincoln', 'Nevada':
'Carson City', 'New Hampshire': 'Concord', 'New Jersey': 'Trenton', 'New
Mexico': 'Santa Fe', 'New York': 'Albany', 'North Carolina': 'Raleigh',
'North Dakota': 'Bismarck', 'Ohio': 'Columbus', 'Oklahoma': 'Oklahoma City',
'Oregon': 'Salem', 'Pennsylvania': 'Harrisburg', 'Rhode Island': 'Providence',
'South Carolina': 'Columbia', 'South Dakota': 'Pierre', 'Tennessee':
'Nashville', 'Texas': 'Austin', 'Utah': 'Salt Lake City', 'Vermont':
'Montpelier', 'Virginia': 'Richmond', 'Washington': 'Olympia', 'West
Virginia': 'Charleston', 'Wisconsin': 'Madison', 'Wyoming': 'Cheyenne'}
# Generate 35 quiz files.
for quizNum in range(35):
# TODO: Create the quiz and answer key files.
# TODO: Write out the header for the quiz.
# TODO: Shuffle the order of the states.
# TODO: Loop through all 50 states, making a question for each.
```

- Since this program will be randomly ordering the questions and answers, you'll need to import the random module u to make use of its functions.
- The capitals variable v contains a dictionary with US states as keys and their capitals as values. And since you want to create 35 quizzes, the code that actually generates the quiz and answer key files (marked with TODO comments for now) will go inside a for loop that loops 35 times

Step 2: Create the Quiz File and Shuffle the Question Order

- The code in the loop will be repeated 35 times—once for each quiz— so you have to worry about only one quiz at a time within the loop
- First you'll create the actual quiz file.
- It needs to have a unique filename and should also have some kind of standard header in it, with places for the student to fill in a name, date, and class period.
- Add the following lines of code to *randomQuizGenerator.py*:

```
#! python3
# randomQuizGenerator.py - Creates quizzes with questions and answers in
# random order, along with the answer key.

--snip—

# Generate 35 quiz files.
for quizNum in range(35):
# Create the quiz and answer key files.
quizFile = open('capitalsquiz%s.txt' % (quizNum + 1), 'w')
 answerKeyFile = open('capitalsquiz_answers%s.txt' % (quizNum + 1), 'w')
# Write out the header for the quiz.
w quizFile.write('Name:\n\nDate:\n\nPeriod:\n\n')
quizFile.write((' ' * 20) + 'State Capitals Quiz (Form %s)' % (quizNum + 1))
quizFile.write('\n\n')
# Shuffle the order of the states.
states = list(capitals.keys())
random.shuffle(states)

# TODO: Loop through all 50 states, making a question for each.
```

Step 3: Create the Answer Options

- Now you need to generate the answer options for each question, which will be multiple choice from A to D.
- You'll need to create another for loop—this one to generate the content for each of the 50 questions on the quiz

```
#! python3
# randomQuizGenerator.py - Creates quizzes with questions and answers in
# random order, along with the answer key.

--snip—

    # Loop through all 50 states, making a question for each.
    for questionNum in range(50):

    # Get right and wrong answers.
    correctAnswer = capitals[states[questionNum]]
    wrongAnswers = list(capitals.values())
    del wrongAnswers[wrongAnswers.index(correctAnswer)]
    wrongAnswers = random.sample(wrongAnswers, 3)
    answerOptions = wrongAnswers + [correctAnswer]
    random.shuffle(answerOptions)

    # TODO: Write the question and answer options to the quiz file.
    # TODO: Write the answer key to a file.
```

- The correct answer is easy to get—it's stored as a value in the capitals dictionary
- This loop will loop through the states in the shuffled states list, from states[0] to states[49], find each state in capitals, and store that state's corresponding capital in correctAnswer.

- The list of possible wrong answers is trickier. You can get it by duplicating *all* the values in the capitals dictionary
- deleting the correct answer w, and selecting three random values from this list
- The random.sample() function makes it easy to do this selection. Its first argument is the list you want to select from; the second argument is the number of values you want to select. The full list of answer options is the combination of these three wrong answers with the correct answers
- Finally, the answers need to be randomized z so that the correct response isn't always choice D.

Step 4: Write Content to the Quiz and Answer Key Files

```
#! python3
# randomQuizGenerator.py - Creates quizzes with questions and answers in
# random order, along with the answer key.

--snip—
# Loop through all 50 states, making a question for each.
for questionNum in range(50):
--snip—

# Write the question and the answer options to the quiz file.
quizFile.write('%s. What is the capital of %s?\n' % (questionNum + 1,
        states[questionNum]))
for i in range(4):
        quizFile.write(' %s. %s\n' % ('ABCD'[i], answerOptions[i]))
quizFile.write('\n')

# Write the answer key to a file.
answerKeyFile.write('%s. %s\n' % (questionNum + 1, 'ABCD'[
        answerOptions.index(correctAnswer)]))
quizFile.close()
answerKeyFile.close()
```

- A for loop that goes through integers 0 to 3 will write the answer options in the answerOptions list . The expression 'ABCD'[i] at treats the string 'ABCD' as an array and will evaluate to 'A','B', 'C', and then 'D' on each respective iteration through the loop.

Name:
Date:
Period:

State Capitals Quiz (Form 1)

1. What is the capital of West Virginia?
A. Hartford
B. Santa Fe
C. Harrisburg
D. Charleston

2. What is the capital of Colorado?
A. Raleigh
B. Harrisburg
C. Denver
D. Lincoln
--snip—

- The corresponding *capitalsquiz_answers1.txt* text file

1. D
2. C
3. A
4. C

--snip--

7. Project: Multiclipboard

- Say you have the boring task of filling out many forms in a web page or software with several text fields.
- The clipboard saves you from typing the same text over and over again. But only one thing can be on the clipboard at a time.
- The program will save each piece of clipboard text under a keyword.
- For example, when you run py mcb.pyw save spam, the current contents of the clipboard will be saved with the keyword *spam*.

  Here's what the program does:

  • The command line argument for the keyword is checked.

  • If the argument is save, then the clipboard contents are saved to the keyword.

  • If the argument is list, then all the keywords are copied to the clipboard.

  • Otherwise, the text for the keyword is copied to the keyboard. This means the code will need to do the following:

  • Read the command line arguments from sys.argv.

  • Read and write to the clipboard.

  • Save and load to a shelf file.

Step 1: Comments and Shelf Setup

```
#! python3
# mcb.pyw - Saves and loads pieces of text to the clipboard.
# Usage: py.exe mcb.pyw save <keyword> - Saves clipboard to keyword.
        # py.exe mcb.pyw <keyword> - Loads keyword to clipboard.
        # py.exe mcb.pyw list - Loads all keywords to clipboard.
import shelve, pyperclip, sys
mcbShelf = shelve.open('mcb')

# TODO: Save clipboard content.
# TODO: List keywords and load content.
mcbShelf.close()
```

Step 2: Save Clipboard Content with a Keyword

- The program does different things depending on whether the user wants to save text to a keyword, load text into the clipboard, or list all the existing keywords.

```
#! python3
# mcb.pyw - Saves and loads pieces of text to the clipboard.
--snip—

# Save clipboard content.
 if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
 mcbShelf[sys.argv[2]] = pyperclip.paste()
elif len(sys.argv) == 2:
 # TODO: List keywords and load content.

mcbShelf.close()
```

- If the first command line argument (which will always be at index 1 of the sys.argv list) is 'save'
- The second command line argument is the keyword for the current content of the clipboard.
- The keyword will be used as the key for mcbShelf, and the value will be the text currently on the clipboard

Step 3: List Keywords and Load a Keyword's Content

- The user wants to load clipboard text in from a keyword, or they want a list of all available keywords

```
#! python3
# mcb.pyw - Saves and loads pieces of text to the clipboard.
--snip—

# Save clipboard content.
if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
        mcbShelf[sys.argv[2]] = pyperclip.paste()
elif len(sys.argv) == 2:
        # List keywords and load content.
        if sys.argv[1].lower() == 'list':
                pyperclip.copy(str(list(mcbShelf.keys())))
        elif sys.argv[1] in mcbShelf:
                 pyperclip.copy(mcbShelf[sys.argv[1]])
mcbShelf.close()
```

- If there is only one command line argument, first let's check whether it's 'list'
- If so, a string representation of the list of shelf keys will be copied to the clipboard
- The user can paste this list into an open text editor to read it.
- Otherwise, you can assume the command line argument is a keyword. If this keyword exists in the mcbShelf shelf as a key, you can load the value onto the clipboard