CHAPTER 1: MANIPULATING STRINGS

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

3.1 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.

Note: slicing a string does not modify the original string.

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.

3.2 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.

```
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.')
```

Program

```
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
```

output

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']
```

- A 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

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

character.

- 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.

```
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)
```

Program

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

output

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:'
```

3.3 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': 'VmALvQyKAxiVH5G8vO1if1MLZF3sdt',
             '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>.

3.4 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:**

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

Program

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

output

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.