7 Gashlycrumb: Looking items up in a dictionary

In this chapter, we're going to look up lines of text from an input file that start with the letters provided by the user. The text will come from an input file that will default to Edward Gorey's "The Gashlycrumb Tinies," an abecedarian book that describes various and ghastly ways in which children expire. For instance, figure 7.1 shows that "N is for Neville who died of ennui."

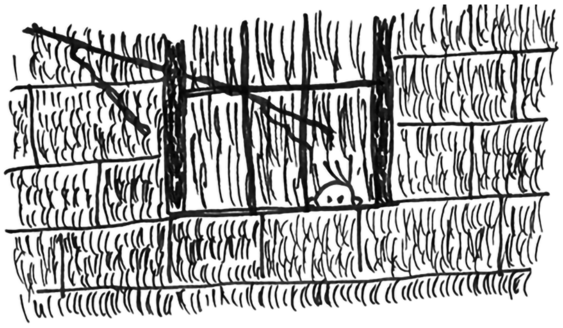


Figure 7.1 N is for Neville who died of ennui.

Our gashlycrumb.py program will take one or more letters as positional arguments and will look up the lines of text that start with that letter from an *optional* input file. We will look up the letters in a *case-insensitive* fashion.

The input file will have the value for each letter on a separate line:

```
$ head -2 gashlycrumb.txt
A is for Amy who fell down the stairs.
B is for Basil assaulted by bears.
```

When our unfortunate user runs this program, here is what they will see:

```
$ ./gashlycrumb.py e f
E is for Ernest who choked on a peach.
F is for Fanny sucked dry by a leech.
```

In this exercise, you will

- Accept one or more positional arguments that we'll call `letter`.
- Accept an optional `--file` argument, which must be a readable text file. The default value will be `'gashlycrumb.txt'` (provided).
- Read the file, find the first letter of each line, and build a data structure that associates the letter to the line of text. (We'll only be using files where each line starts with a single, unique letter. This program would fail with any other format of text.)
- For each `letter` provided by the user, either print the line of text for the `letter` if present, or print a message if it isn't.
- Learn how to "pretty-print" a data structure.

You can draw from several previous programs:

- From chapter 2 you know how to get the first letter of a bit of text.
- From chapter 4 you know how to build a dictionary and look up a value.
- From chapter 6 you know how to accept a file input argument and read it line by line.

Now you'll put all those skills together to recite morbid poetry!

7.1 Writing gashlycrumb.py

Before you begin writing, I encourage you to run the tests with `make test` or `pytest -xv test.py` in the 07_gashlycrumb directory. The first test should fail:

```
test.py::test_exists FAILED
```

This is just a reminder that the first thing you need to do is create the file called gashlycrumb.py. You can do this however you like, such as by running `new.py gashlycrumb.py` in the 07_gashlycrumb directory, by copying the template/template.py file, or by just starting a new file from scratch. Run your tests again, and you should pass the first test and possibly the second if your program produces a usage statement.

Next, let's get the arguments straight. Modify your program's parameters in the `get_args()` function so that it will produce the following usage statement when the program is run with no arguments or with the `-h` or `--help` flags:

```
$ ./gashlycrumb.py -h
usage: gashlycrumb.py [-h] [-f FILE] letter [letter ...]
```

```
Gashlycrumb

positional arguments:
  letter                 Letter(s)

optional arguments:
  -h, --help             show this help message and exit
  -f FILE, --file FILE   Input file (default: gashlycrumb.txt)
```

letter is a required positional argument that accepts one or more values.

The -h and --help arguments are created automatically by argparse.

The -f or --file argument is an option with a default value of gashlycrumb.txt.

Figure 7.2 shows a string diagram of how the program will work.

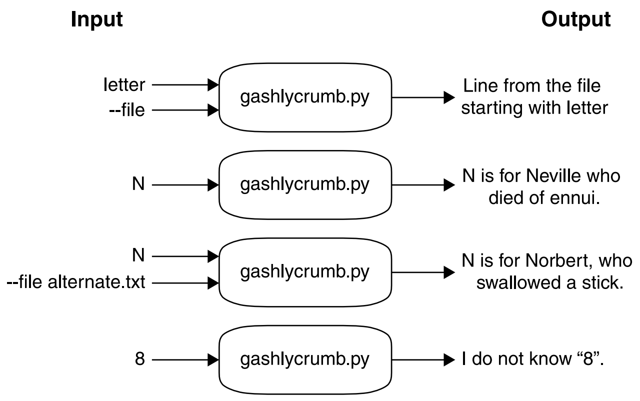


Figure 7.2 Our program will accept some letter(s) and possibly a file. It will then look up the line(s) of the file starting with the given letter(s).

In the `main()` function, start off by echoing each of the `letter` arguments:

```
def main():
    args = get_args()
    for letter in args.letter:
        print(letter)
```

Try running it to make sure it works:

```
$ ./gashlycrumb.py a b
a
b
```

Next, read the file line by line using a `for` loop:

```
def main():
    args = get_args()
    for letter in args.letter:
        print(letter)

    for line in args.file:
        print(line, end='')
```

Note that I'm using `end=''` with `print()` so that it won't print the newline that's already attached to each `line` of the file:

Try running it to ensure you can read the input file:

```
$ ./gashlycrumb.py a b | head -4
a
b
A is for Amy who fell down the stairs.
B is for Basil assaulted by bears.
```

Use the alternate.txt file too:

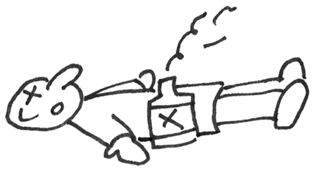
```
$ ./gashlycrumb.py a b --file alternate.txt | head -4
a
b
A is for Alfred, poisoned to death.
B is for Bertrand, consumed by meth.
```

If your program is provided a `--file` argument that does not exist, it should exit with an error and message. Note that if you declare the parameter in `get_args()` using `type=argparse.FileType('rt')` as we did in the previous chapter, this error should be produced automatically by `argparse`:

```
$ ./gashlycrumb.py -f blargh b
usage: gashlycrumb.py [-h] [-f FILE] letter [letter ...]
gashlycrumb.py: error: argument -f/--file: can't open 'blargh': \
[Errno 2] No such file or directory: 'blargh'
```

Now think about how you can use the first letter of each `line` to create an entry in a `dict`. Use `print()` to look at your dictionary. Figure out how to check if the given `letter` is *in* (wink, wink, nudge, nudge) your dictionary.

If your program is given a value that does not exist in the list of first characters on the lines from the input file (when searched without regard to case), you should print a message:



```
$ ./gashlycrumb.py 3
I do not know "3".
$ ./gashlycrumb.py CH
I do not know "CH".
```

If the given `letter` is in the dictionary, print the value for it (see figure 7.3):

```
$ ./gashlycrumb.py a
A is for Amy who fell down the stairs.
```

```
$ ./gashlycrumb.py z
Z is for Zillah who drank too much gin.
```

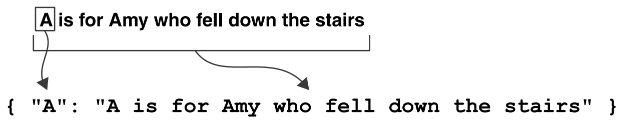

A is for Amy who fell down the stairs
{ "A": "A is for Amy who fell down the stairs" }

Figure 7.3 We need to create a dictionary where the first letter of each line is the key and the line is the value.

Run the test suite to ensure your program meets all the requirements. Read the errors closely and fix your program.

Here are some hints:

- Start with new.py and remove everything but the positional `letter` and optional `--file` parameters.
- Use `type=argparse.FileType('rt')` to validate the `--file` argument.
- Use `nargs='+'` to define the positional argument `letter` so it will require one or more values.
- A dictionary is a natural data structure for associating a value like the letter "A" to a phrase like "A is for Amy who fell down the stairs." Create a new, empty `dict`.
- Once you have an open file handle, you can read the file line by line with a `for` loop.
- Each line of text is a string. How can you get the first character of a string?
- Create an entry in your dictionary using the first character as the key and the line itself as the value.
- Iterate through each `letter` argument. How can you check that a given value is `in` the dictionary?

No skipping ahead to the solution until you have written your own version! If you peek, you will die a horrible death: stampeded by kittens.

7.2 Solution

I really hope you looked at Gorey's artwork for his book. Now let's talk about how to build a dictionary from a file input:

```
#!/usr/bin/env python3
"""Lookup tables"""

import argparse
```

```
# --------------------------------------------------
def get_args():
    """get command-line arguments"""

    parser = argparse.ArgumentParser(
        description='Gashlycrumb',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('letter',
                        help='Letter(s)',
                        metavar='letter',
                        nargs='+',
                        type=str)

    parser.add_argument('-f',
                        '--file',
                        help='Input file',
                        metavar='FILE',
                        type=argparse.FileType('rt'),
                        default='gashlycrumb.txt')

    return parser.parse_args()


# --------------------------------------------------
def main():
    """Make a jazz noise here"""

    args = get_args()

    lookup = {}
    for line in args.file:
        lookup[line[0].upper()] = line.rstrip()

    for letter in args.letter:
        if letter.upper() in lookup:
            print(lookup[letter.upper()])
        else:
            print(f'I do not know "{letter}".')

# --------------------------------------------------
if __name__ == '__main__':
    main()
```

A positional argument called letter uses nargs='+' to indicate that one or more values are required.

The optional --file argument must be a readable file because of type=argparse.FileType('rt'). The default value is gashlycrumb.txt, which I know exists.

Create an empty dictionary to hold the lookup table.

Iterate through each line of the args.file, which will be an open file handle.

Uppercase the first character of the line to use as the key into the lookup table and set the value to be the line stripped of whitespace on the right side.

Use a for loop to iterate over each letter in args.letter.

Check if the letter is in the lookup dictionary, using letter.upper() to disregard case.

If so, print the line of text from the lookup for the letter.

Otherwise, print a message saying the letter is unknown.

7.3 Discussion

Did the frightful paws of the kittens hurt much? Let's talk about how I solved this problem. Remember, mine is just one of many possible solutions.

7.3.1 Handling the arguments

I prefer to have all the logic for parsing and validating the command-line arguments in the `get_args()` function. In particular, `argparse` can do a fine job of verifying tedious things, such as an argument being an existing, readable text file, which is why

I use `type=argparse.FileType('rt')` for that argument. If the user doesn't supply a valid argument, `argparse` will throw an error, printing a helpful message along with the short usage statement, and will exit with an error code.

By the time I get to the line `args = get_args()`, I know that I have one or more "letter" arguments and a valid, open file handle in the `args.file` slot. In the REPL, I can use `open` to get a file handle, which I usually like to call `fh`. For copyright purposes, I'll use my alternate text:

```
>>> fh = open('alternate.txt')
```

7.3.2 Reading the input file

We want to use a dictionary where the keys are the first letters of each line and the values are the lines themselves. That means we need to start by creating a new, empty dictionary, either by using the `dict()` function or by setting a variable equal to an empty set of curly brackets (`{}`). Let's call the variable `lookup`:

```
>>> lookup = {}
```

We can use a `for` loop to read each `line` of text. From the Crow's Nest program in chapter 2, you know we can use `line[0].upper()` to get the first letter of `line` and uppercase it. We can use that as the key into `lookup`.

Each `line` of text ends with a newline that I'd like to remove. The `str.rstrip()` method will strip whitespace from the right side of the `line` ("rstrip" = *right strip*). The result of that will be the value for my `lookup`:

```
for line in fh:
    lookup[line[0].upper()] = line.rstrip()
```

Let's look at the resulting `lookup` dictionary. We can `print()` it from the program or type `lookup` in the REPL, but it's going to be hard to read. I encourage you to try it.

Luckily there is a lovely module called `pprint` to "pretty-print" data structures. Here is how you can import the `pprint()` function from the `pprint` module with the alias `pp`:

```
>>> from pprint import pprint as pp
```

Figure 7.4 illustrates how this works.

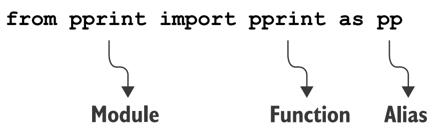


Figure 7.4 We can specify exactly which functions to import from a module and even give the function an alias.

Now let's take a peek at the lookup table:

```
>>> pp(lookup)
{'A': 'A is for Alfred, poisoned to death.',
 'B': 'B is for Bertrand, consumed by meth.',
 'C': 'C is for Cornell, who ate some glass.',
 'D': 'D is for Donald, who died from gas.',
 'E': 'E is for Edward, hanged by the neck.',
 'F': 'F is for Freddy, crushed in a wreck.',
 'G': 'G is for Geoffrey, who slit his wrist.',
 'H': "H is for Henry, who's neck got a twist.",
 'I': 'I is for Ingrid, who tripped down a stair.',
 'J': 'J is for Jered, who fell off a chair,',
 'K': 'K is for Kevin, bit by a snake.',
 'L': 'L is for Lauryl, impaled on a stake.',
 'M': 'M is for Moira, hit by a brick.',
 'N': 'N is for Norbert, who swallowed a stick.',
 'O': 'O is for Orville, who fell in a canyon,',
 'P': 'P is for Paul, strangled by his banyan,',
 'Q': 'Q is for Quintanna, flayed in the night,',
 'R': 'R is for Robert, who died of spite,',
 'S': 'S is for Susan, stung by a jelly,',
 'T': 'T is for Terrange, kicked in the belly,',
 'U': "U is for Uma, who's life was vanquished,",
 'V': 'V is for Victor, consumed by anguish,',
 'W': "W is for Walter, who's socks were too long,",
 'X': 'X is for Xavier, stuck through with a prong,',
 'Y': 'Y is for Yoeman, too fat by a piece,',
 'Z': 'Z is for Zora, smothered by a fleece.'}
```

Hey, that looks like a handy data structure. Hooray for us! Please don't discount the value of using lots of `print()` calls when you are trying to write and understand a program, and of using the `pprint()` function whenever you need to see a complex data structure.

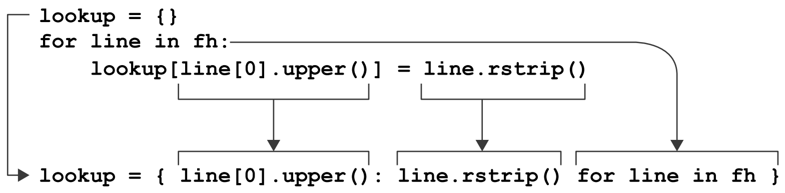
7.3.3 Using a dictionary comprehension

In chapter 4 you saw that you can use a list comprehension to build a list by putting a `for` loop inside `[]`. If we change the brackets to curlies (`{}`), we create a dictionary comprehension:

```
>>> fh = open('gashlycrumb.txt')
>>> lookup = { line[0].upper(): line.rstrip() for line in fh }
```

See in figure 7.5 how we can rearrange three lines of our `for` loop into a single line of code.

If you print the `lookup` table again, you should see the same output as before. It may seem like showing off to write one line of code instead of three, but it really does make a good deal of sense to write compact, idiomatic code. More code always means more chances for bugs, so I usually try to write code that is as simple as possible (but no simpler).

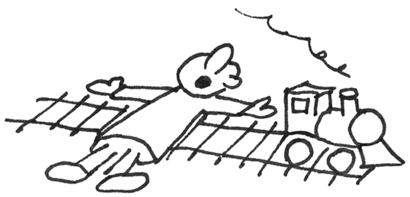


```
lookup = {}
for line in fh:
    lookup[line[0].upper()] = line.rstrip()

lookup = { line[0].upper(): line.rstrip() for line in fh }
```

Figure 7.5 The `for` loop we used to build a dictionary can be written using a dictionary comprehension.

7.3.4 Dictionary lookups



Now that I have a `lookup` table, I can ask whether some value is `in` the keys. I know the letters are in uppercase, and since the user could give me a lowercase letter, I use `letter.upper()` to only compare that case:

```
>>> letter = 'a'
>>> letter.upper() in lookup
True
>>> lookup[letter.upper()]
'A is for Amy who fell down the stairs.'
```

If the letter is found, I can print the line of text for that letter; otherwise, I can print a message saying that I don't know that letter:

```
>>> letter = '4'
>>> if letter.upper() in lookup:
...     print(lookup[letter.upper()])
... else:
...     print('I do not know "{}".'.format(letter))
...
I do not know "4".
```

An even shorter way to write that would be to use the `dict.get()` method:

```
def main():
    args = get_args()
    lookup = {line[0].upper(): line.rstrip() for line in args.file}

    for letter in args.letter:
        print(lookup.get(letter.upper(), f'I do not know "{letter}".'))
```

lookup.get() will return the value for letter.upper() or the warning about a value not being found in our lookup.

7.4 Going further

- Write a phonebook that reads a file and creates a dictionary from the names of your friends and their email or phone numbers.
- Create a program that uses a dictionary to count the number of times you see each word in a document.

- Write an interactive version of the program that takes input directly from the user. Use `while True` to set up an infinite loop and keep using the `input()` function to get the user's next `letter`:

```
$ ./gashlycrumb_interactive.py
    Please provide a letter [! to quit]: t
    T is for Titus who flew into bits.
    Please provide a letter [! to quit]: 7
    I do not know "7".
    Please provide a letter [! to quit]: !
    Bye
```

- Interactive programs are fun to write, but how would you go about testing them? In chapter 17 I'll show you one way to do this.

Summary

- A dictionary comprehension is a way to build a dictionary in a one-line `for` loop.
- Defining file input arguments using `argparse.FileType` saves you time and code.
- Python's `pprint` module is used to pretty-print complex data structures.