Chapter 1

Howler

Write a Python program `howler.py` that will uppercase all the text from the command line or from a file.

```
$ ./howler.py
usage: howler.py [-h] [-o str] STR
howler.py: error: the following arguments are required: STR
$ ./howler.py -h
usage: howler.py [-h] [-o str] STR

Howler (upper-case input)

positional arguments:
  STR                   Input string or file

optional arguments:
  -h, --help            show this help message and exit
  -o str, --outfile str
                        Output filename (default: )
```

Skills

- Reading text from command line or a file
- Transforming text
- Write to a file or STDOUT

howler Solution

```
#!/usr/bin/env python3
"""Howler"""

import argparse
import os
import sys


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Howler (upper-case input)',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('text', metavar='STR', help='Input string or file')

    parser.add_argument('-o',
                        '--outfile',
                        help='Output filename',
                        metavar='str',
                        type=str,
                        default='')

    return parser.parse_args()


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()
    text = args.text
    out_file = args.outfile

    if os.path.isfile(text):
        text = open(text).read()

    out_fh = open(out_file, 'wt') if out_file else sys.stdout
    out_fh.write(text.upper() + '\n')


# --------------------------------------------------
if __name__ == '__main__':
```

```
main()
```

Chapter 2

Jump the Five

Write a program called `jump.py` that will encode any number using "jump-the-five" algorithm that selects as a replacement for a given number the number that is opposite the number on a US telephone pad if you jump over the 5. The numbers 5and 9 will exchange with each other. So, "1" jumps the 5 to become "9," "6" jumps the 5 to become "4," "5" becomes "0," etc.

```
1  2  3
4  5  6
7  8  9
#  0  *
```

If given no arguments, print a usage statement.

Expected Behavior

```
$ ./jump.py
Usage: jump.py NUMBER
$ ./jump.py 555-1212
000-9898
$ ./jump.py 'Call 1-800-329-8044 today!'
Call 9-255-781-2566 today!
```

jump_the_five Solution

```
#!/usr/bin/env python3
"""Jump the Five"""

import os
import sys


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

    args = sys.argv[1:]

    if len(args) != 1:
        print('Usage: {} NUMBER'.format(os.path.basename(sys.argv[0])))
        sys.exit(1)

    num = args[0]
    jumper = {
        '1': '9',
        '2': '8',
        '3': '7',
        '4': '6',
        '5': '0',
        '6': '4',
        '7': '3',
        '8': '2',
        '9': '1',
        '0': '5'
    }

    for char in num:
        print(jumper[char] if char in jumper else char, end='')
    print()


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

Chapter 3

Bottles of Beer Song

Write a Python program called `bottles.py` that takes a single option `-n|--num_bottles` which is an positive integer (default 10) and prints the " bottles of beer on the wall song." If the `-n` argument is less than 1, die with "N () must be a positive integer". The program should also respond to `-h|--help` with a usage statement.

I'd encourage you to think about the program as a formal algorithm. Read the introduction to Jeff Erickson's book *Algorithms* available here:

- http://jeffe.cs.illinois.edu/teaching/algorithms/#book
- http://jeffe.cs.illinois.edu/teaching/algorithms/book/00-intro.pdf

You are going to need to count down, so you'll need to consider how to do that. First, let's examine a list and see how it can be sorted and reversed. We've already used the `sorted` *function*, but we haven't really talked about the `list` class's `sort` *method*. Note that the former does not mutate the list itself:

```
>>> a = ['foo', 'bar', 'baz']
>>> sorted(a)
['bar', 'baz', 'foo']
>>> a
['foo', 'bar', 'baz']
```

But the `sort` method does:

```
>>> a.sort()
>>> a
['bar', 'baz', 'foo']
```

Also, note what is returned by `sort`:

```
>>> type(a.sort())
<type 'NoneType'>
```

So if you did this, you'd destroy your data:

```
>>> a = a.sort()
>>> a
```

As with `sort`/`sorted`, so it goes with `reverse`/`reversed`. The past participle version *returns a new copy of the data without affecting the original* and is therefore the safest bet to use:

```
>>> a = ['foo', 'bar', 'baz']
>>> a
['foo', 'bar', 'baz']
```

```
>>> reversed(a)
<listreverseiterator object at 0x10f0d61d0>
>>> list(reversed(a))
['baz', 'bar', 'foo']
>>> a
['foo', 'bar', 'baz']
```

Compare with:

```
>>> a.reverse()
>>> a
['baz', 'bar', 'foo']
```

Given that and your knowledge of how `range` works, can you figure out how to count down, say, from 10 to 1?

```
$ ./bottles.py -h
usage: bottles.py [-h] [-n INT]

Bottles of beer song

optional arguments:
  -h, --help            show this help message and exit
  -n INT, --num_bottles INT
$ ./bottles.py --help
usage: bottles.py [-h] [-n INT]

Bottles of beer song

optional arguments:
  -h, --help            show this help message and exit
  -n INT, --num_bottles INT
                        How many bottles (default: 10)
$ ./bottles.py -n 1
1 bottle of beer on the wall,
1 bottle of beer,
Take one down, pass it around,
0 bottles of beer on the wall!

$ ./bottles.py | head
10 bottles of beer on the wall,
10 bottles of beer,
Take one down, pass it around,
9 bottles of beer on the wall!

9 bottles of beer on the wall,
9 bottles of beer,
Take one down, pass it around,
```

```
8 bottles of beer on the wall!
```

bottles_of_beer Solution

```
#!/usr/bin/env python3

import argparse
import sys
from dire import die


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Bottles of beer song',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('-n',
                        '--num_bottles',
                        metavar='INT',
                        type=int,
                        default=10,
                        help='How many bottles')

    return parser.parse_args()


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()
    num_bottles = args.num_bottles

    if num_bottles < 1:
        die('N ({}) must be a positive integer'.format(num_bottles))

    line1 = '{} bottle{} of beer on the wall'
    line2 = '{} bottle{} of beer'
    line3 = 'Take one down, pass it around'
    tmpl = ',\n'.join([line1, line2, line3, line1 + '!'])

    for n in reversed(range(1, num_bottles + 1)):
        s1 = '' if n == 1 else 's'
        s2 = '' if n - 1 == 1 else 's'
        print(tmpl.format(n, s1, n, s1, n - 1, s2))
        if n > 1: print()
```

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

Chapter 4

Picnic

Write a Python program called `picnic.py` that accepts one or more positional arguments as the items to bring on a picnic. In response, print "You are bringing …" where "…" should be replaced according to the number of items where:

1. If one item, just state, e.g., if "chips" then "You are bringing chips."
2. If two items, put "and" in between, e.g., if "chips soda" then "You are bringing chips and soda."
3. If three or more items, place commas between all the items INCLUDING BEFORE THE FINAL "and" BECAUSE WE USE THE OXFORD COMMA, e.g., if "chips soda cupcakes" then "You are bringing chips, soda, and cupcakes."

```
$ ./picnic.py
usage: picnic.py [-h] str [str ...]
picnic.py: error: the following arguments are required: str
$ ./picnic.py -h
usage: picnic.py [-h] str [str ...]

Picnic game

positional arguments:
  str         Item(s) to bring

optional arguments:
  -h, --help  show this help message and exit
$ ./picnic.py chips
You are bringing chips.
$ ./picnic.py "potato chips" salad
You are bringing potato chips and salad.
$ ./picnic.py "potato chips" salad soda cupcakes
You are bringing potato chips, salad, soda, and cupcakes.
```

picnic Solution

```
#!/usr/bin/env python3
"""Picnic game"""

import argparse


# --------------------------------------------------
def get_args():
    """Get command-line arguments"""

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

    parser.add_argument('item',
                        metavar='str',
                        nargs='+',
                        help='Item(s) to bring')

    return parser.parse_args()


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

    args = get_args()
    items = args.item
    num = len(items)

    bringing = items[0] if num == 1 else ' and '.join(
        items) if num == 2 else ', '.join(items[:-1] + ['and ' + items[-1]])

    print('You are bringing {}.'.format(bringing))


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

Chapter 5

Apples and Bananas

Perhaps you remember the children's song "Apples and Bananas"?

```
I like to eat, eat, eat apples and bananas
I like to eat, eat, eat apples and bananas

I like to ate, ate, ate ay-ples and ba-nay-nays
I like to ate, ate, ate ay-ples and ba-nay-nays

I like to eat, eat, eat ee-ples and bee-nee-nees
I like to eat, eat, eat ee-ples and bee-nee-nees
```

Write a Python program called `apples.py` that will turn all the vowels in some given text in a single positional argument into just one `-v|--vowel` (default "a") like this song. It should complain if the `--vowel` argument isn't a single, lowercase vowel (hint, see `choices` in the `argparse` documentation). If the given text argument is a file, read the text from the file. Replace all vowels with the given vowel, both lower- and uppercase.

```
$ ./apples.py
usage: apples.py [-h] [-v str] str
apples.py: error: the following arguments are required: str
$ ./apples.py -h
usage: apples.py [-h] [-v str] str

Apples and bananas

positional arguments:
  str                   Input text or file

optional arguments:
  -h, --help            show this help message and exit
  -v str, --vowel str   The only vowel allowed (default: a)
$ ./apples.py -v x foo
usage: apples.py [-h] [-v str] str
apples.py: error: argument -v/--vowel: invalid choice: 'x' (choose from 'a', 'e', 'i', 'o',
$ ./apples.py foo
faa
$ ./apples.py ../inputs/fox.txt
Tha qaack brawn fax jamps avar tha lazy dag.
```

apples_and_bananas Solution

```
#!/usr/bin/env python3

import argparse
import os
import re
import sys


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Apples and bananas',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('text', metavar='str', help='Input text or file')

    parser.add_argument('-v',
                        '--vowel',
                        help='The vowel(s) allowed',
                        metavar='str',
                        type=str,
                        default='a',
                        choices=list('aeiou'))

    return parser.parse_args()


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()
    text = args.text
    vowel = args.vowel

    if os.path.isfile(text):
        text = open(text).read()

    # Method 1: Iterate every character
    # new_text = []
    # for char in text:
    #     if char in 'aeiou':
    #         new_text.append(vowel)
```

```
    #     elif char in 'AEIOU':
    #         new_text.append(vowel.upper())
    #     else:
    #         new_text.append(char)
    # text = ''.join(new_text)

    # Method 2: str.replace
    # for v in 'aeiou':
    #     text = text.replace(v, vowel).replace(v.upper(), vowel.upper())

    # Method 3: Use a list comprehension
    # new_text = [
    #     vowel if c in 'aeiou' else vowel.upper() if c in 'AEIOU' else c
    #     for c in text
    # ]
    # text = ''.join(new_text)

    # Method 4: Define a function, use list comprehension
    def new_char(c):
        return vowel if c in 'aeiou' else vowel.upper() if c in 'AEIOU' else c

    # text = ''.join([new_char(c) for c in text])

    # Method 5: Use a `map` to iterate with a `lambda`
    # text = ''.join(
    #     map(
    #         lambda c: vowel if c in 'aeiou' else vowel.upper()
    #         if c in 'AEIOU' else c, text))

    # Method 6: `map` with the function
    text = ''.join(map(new_char, text))

    # Method 7: Regular expressions
    # text = re.sub('[aeiou]', vowel, text)
    # text = re.sub('[AEIOU]', vowel.upper(), text)

    print(text.rstrip())


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

Chapter 6

Gashlycrumb

Write a Python program called `gashlycrumb.py` that takes a letter of the alphabet as an argument and looks up the line in a `-f|--file` argument (default `gashlycrumb.txt`) and prints the line starting with that letter.

```
$ ./gashlycrumb.py
usage: gashlycrumb.py [-h] [-f str] str
gashlycrumb.py: error: the following arguments are required: str
$ ./gashlycrumb.py -h
usage: gashlycrumb.py [-h] [-f str] str

Gashlycrumb

positional arguments:
  str                 Letter

optional arguments:
  -h, --help          show this help message and exit
  -f str, --file str  Input file (default: gashlycrumb.txt)
$ ./gashlycrumb.py 3
I do not know "3".
$ ./gashlycrumb.py CH
"CH" is not 1 character.
$ ./gashlycrumb.py a
A is for Amy who fell down the stairs.
$ ./gashlycrumb.py z
Z is for Zillah who drank too much gin.
```

Discussion

If you are not familiar with the work of Edward Gorey, please stop and go read about him immediately, e.g. https://www.brainpickings.org/2011/01/19/edward-gorey-the-gashlycrumb-tinies/!

Write your own version of Gorey's text and pass in your version as the `--file`.

Write an interactive version that takes input directly from the user:

```
$ ./gashlycrumb_i.py
Please provide a letter [! to quit]: a
A is for Amy who fell down the stairs.
Please provide a letter [! to quit]: b
```

```
B is for Basil assaulted by bears.
Please provide a letter [! to quit]: !
Bye
```

gashlycrumb Solution

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

import argparse
import os
from dire import die


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Gashlycrumb',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

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

    parser.add_argument('-f',
                        '--file',
                        help='Input file',
                        metavar='str',
                        type=str,
                        default='gashlycrumb.txt')

    return parser.parse_args()


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()
    letter = args.letter.upper()
    file = args.file

    if not os.path.isfile(file):
        die('--file "{}" is not a file.'.format(file))

    if len(letter) != 1:
        die('"{}" is not 1 character.'.format(letter))

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

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


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

Chapter 7

Movie Reader

Write a Python program called `movie_reader.py` that takes a single positional argument that is a bit of text or the name of an input file. The output will be dynamic, so I cannot write a test for how the program should behave, nor can I include a bit of text that shows you how it should work. Your program should print the input text character-by-character and then pause .5 seconds for ending punctuation like `.`, `!` or `?`, .2 seconds for a pause like `,` `:`, or `;`, and .05 seconds for anything else.

```
$ ./movie_reader.py
usage: movie_reader.py [-h] str
movie_reader.py: error: the following arguments are required: str
$ ./movie_reader.py -h
usage: movie_reader.py [-h] str

Movie Reader

positional arguments:
  str         Input text or file

optional arguments:
  -h, --help  show this help message and exit
$ ./movie_reader.py 'Foo, bar!'
Foo, bar!
$ ./movie_reader.py ../inputs/fox.txt
The quick brown fox jumps over the lazy dog.
```

movie_reader Solution

```
#!/usr/bin/env python3

import argparse
import os
import sys
import time


# --------------------------------------------------
def get_args():
    """Get command-line arguments"""

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

    parser.add_argument('text', metavar='str', help='Input text or file')

    return parser.parse_args()


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

    args = get_args()
    text = args.text

    if os.path.isfile(text):
        text = open(text).read()

    for line in text.splitlines():
        for char in line:
            print(char, end='')
            time.sleep(.5 if char in '.!?\n' else .2 if char in ',:;' else .05)
            sys.stdout.flush()

        print()


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

Chapter 8

Palindromes

Write a Python program called `palindromic.py` that will find words that are palindromes in positional argument which is either a string or a file name.

palindromes Solution

```
#!/usr/bin/env python3

import argparse
import os
import re


# --------------------------------------------------
def get_args():
    """Get command-line arguments"""

    parser = argparse.ArgumentParser(
        description='Find palindromes in text',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('text', metavar='str', help='Input text or file')

    parser.add_argument('-m',
                        '--min',
                        metavar='int',
                        type=int,
                        help='Minimum word length',
                        default=3)

    return parser.parse_args()


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

    args = get_args()
    text = args.text
    min_length = args.min

    if os.path.isfile(text):
        text = open(text).read()

    for line in text.splitlines():
        for word in re.split(r'(\W+)', line.lower()):
            if len(word) >= min_length:
                rev = ''.join(reversed(word))
                if rev == word:
```

```
                    print(word)


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

Chapter 9

Ransom

Create a Python program called `ransom.py` that will randomly capitalize the letters in a given word or phrase. The input text may also name a file in which case the text should come from the file. The program should take a `-s|--seed` argument for the `random.seed` to control randomness for the test suite. It should also respond to `-h|--help` for usage.

Expected Behavior

```
$ ./ransom.py
usage: ransom.py [-h] [-s int] str
ransom.py: error: the following arguments are required: str
$ ./ransom.py -h
usage: ransom.py [-h] [-s int] str

Ransom Note

positional arguments:
  str                 Input text or file

optional arguments:
  -h, --help          show this help message and exit
  -s int, --seed int  Random seed (default: None)
$ cat fox.txt
The quick brown fox jumps over the lazy dog.
$ ./ransom.py fox.txt
the quiCK bROWn fOx JUMps OveR tHe LAzy Dog.
$ ./ransom.py -s 2 'The quick brown fox jumps over the lazy dog.'
the qUIck BROWN fOX JUmps ovEr ThE LAZY DOg.
```

ransom_note Solution

```
#!/usr/bin/env python3

import argparse
import os
import random
import sys


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Ransom Note',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('text', metavar='str', help='Input text or file')

    parser.add_argument('-s',
                        '--seed',
                        help='Random seed',
                        metavar='int',
                        type=int,
                        default=None)

    return parser.parse_args()


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

    random.seed(args.seed)

    text = args.text
    if os.path.isfile(text):
        text = open(text).read()

    #ransom = []
    #for char in text:
    #    ransom.append(char.upper() if random.choice([0, 1]) else char.lower())

    #ransom = [c.upper() if random.choice([0, 1]) else c.lower() for c in text]
```

```
    #ransom = map(lambda c: c.upper() if random.choice([0, 1]) else c.lower(),
    #             text)

    f = lambda c: c.upper() if random.choice([0, 1]) else c.lower()
    ransom = map(f, text)

    print(''.join(ransom))


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

Chapter 10

Simple Rhymer

Write a Python program called `rhymer.py` that will create new words by removing the consonant(s) from the beginning of the word and then creating new words by prefixing the remainder with all the consonants and clusters that were not at the beginning. That is, prefix with all the consonants in the alphabet plus these clusters:

```
bl br ch cl cr dr fl fr gl gr pl pr sc sh sk sl sm sn sp
st sw th tr tw wh wr sch scr shr sph spl spr squ str thr
```

```
$ ./rhymer.py
usage: rhymer.py [-h] str
rhymer.py: error: the following arguments are required: str
$ ./rhymer.py -h
usage: rhymer.py [-h] str

Make rhyming "words"

positional arguments:
  str         A word

optional arguments:
  -h, --help  show this help message and exit
$ ./rhymer.py apple
Word "apple" must start with consonants
$ ./rhymer.py take | head
bake
cake
dake
fake
gake
hake
jake
kake
lake
make
```

rhymer Solution

```
#!/usr/bin/env python3
"""Make rhyming words"""

import argparse
import re
import string
import sys
from dire import die


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Make rhyming "words"',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('word', metavar='str', help='A word')

    return parser.parse_args()


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()
    word = args.word

    vowels = 'aeiou'
    if word[0] in vowels:
        die('Word "{}" must start with consonants'.format(word))

    consonants = [c for c in string.ascii_lowercase if c not in 'aeiou']
    match = re.match('^([' + ''.join(consonants) + ']+)(.+)', word)

    clusters = ('bl br ch cl cr dr fl fr gl gr pl pr sc '
                'sh sk sl sm sn sp st sw th tr tw wh wr '
                'sch scr shr sph spl spr squ str thr').split()

    if match:
        start, rest = match.group(1), match.group(2)
        for c in filter(lambda c: c != start, consonants + clusters):
            print(c + rest)
```

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

Chapter 11

Rock, Paper, Scissors

Write a Python program called `rps.py` that will play the ever-popular "Rock, Paper, Scissors" game.

```
$ ./rps.py
1-2-3-Go! [rps|q] r
You: Rock
Me : Scissors
You win. You are a clammy drate-poke.
1-2-3-Go! [rps|q] t
You dysfunctional dew-beater! Please choose from: p, r, s.
1-2-3-Go! [rps|q] p
You: Paper
Me : Rock
You win. You are a dismal gillie-wet-foot.
1-2-3-Go! [rps|q] q
Bye, you imbecilic fopdoodle!
```

rock_paper_scissors Solution

```
#!/usr/bin/env python3

import sys
import random


# ------------------------------------------------
def insult():
    adjective = """
    truculent fatuous vainglorious fatuous petulant moribund jejune
    feckless antiquated rambunctious mundane misshapen glib dreary
    dopey devoid deleterious degrading clammy brazen indiscreet
    indecorous imbecilic dysfunctional dubious drunken disreputable
    dismal dim deficient deceitful damned daft contrary churlish
    catty banal asinine infantile lurid morbid repugnant unkempt
    vapid decrepit malevolent impertinent decrepit grotesque puerile
    """.split()

    noun = """
    abydocomist bedswerver bespawler bobolyne cumberworld dalcop
    dew-beater dorbel drate-poke driggle-draggle fopdoodle fustylugs
    fustilarian gillie-wet-foot gnashgab gobermouch
    gowpenful-o'-anything klazomaniac leasing-monger loiter-sack
    lubberwort muck-spout mumblecrust quisby raggabrash rakefire
    roiderbanks saddle-goose scobberlotcher skelpie-limmer
    smell-feast smellfungus snoutband sorner stampcrab stymphalist
    tallowcatch triptaker wandought whiffle-whaffle yaldson zoilist
    """.split()

    return ' '.join([random.choice(adjective), random.choice(noun)])


# ------------------------------------------------
def main():
    """Play Rock Paper Scissors"""
    valid = set('rps')

    beats = {'r': 's', 's': 'p', 'p': 'r'}
    display = {'r': 'Rock', 'p': 'Paper', 's': 'Scissors'}

    while True:
        play = input('1-2-3-Go! [rps|q] ').lower()
```

```
        if play.startswith('q'):
            print('Bye, you {}!'.format(insult()))
            sys.exit(0)

        if play not in valid:
            print('You {}! Please choose from: {}.'.format(
                insult(), ', '.join(sorted(valid))))
            continue

        computer = random.choice(list(valid))

        print('You: {}\nMe : {}'.format(display[play], display[computer]))

        if beats[play] == computer:
            print('You win. You are a {}.'.format(insult()))
        elif beats[computer] == play:
            print('You lose, {}!'.format(insult()))
        else:
            print('Draw, you {}.'.format(insult()))


# ------------------------------------------------
main()
```

Chapter 12

Abuse

Write a Python program called `abuse.py` that generates some `-n|--number` of insults (default 3) by randomly combining some number of `-a|--adjectives` (default 2) with a noun (see below). Be sure your program accepts a `-s|--seed` argument to pass to `random.seed`.

Adjectives:

bankrupt base caterwauling corrupt cullionly detestable dishonest false filthsome filthy foolish foul gross heedless indistinguishable infected insatiate irksome lascivious lecherous loathsome lubbery old peevish rascaly rotten ruinous scurilous scurvy slanderous sodden-witted thin-faced toad-spotted unmannered vile wall-eyed

Nouns:

nouns = """ Judas Satan ape ass barbermonger beggar block boy braggart butt carbuncle coward coxcomb cur dandy degenerate fiend fishmonger fool gull harpy jack jolthead knave liar lunatic maw milksop minion ratcatcher recreant rogue scold slave swine traitor varlet villain worm

```
$ ./abuse.py -h
usage: abuse.py [-h] [-a int] [-n int] [-s int]

Argparse Python script

optional arguments:
  -h, --help            show this help message and exit
  -a int, --adjectives int
                        Number of adjectives (default: 2)
  -n int, --number int  Number of insults (default: 3)
  -s int, --seed int    Random seed (default: None)
$ ./abuse.py
You slanderous, rotten block!
You lubbery, scurilous ratcatcher!
You rotten, foul liar!
$ ./abuse.py -s 1 -n 2 -a 1
You rotten rogue!
You lascivious ape!
$ ./abuse.py -s 2 -n 4 -a 4
You scurilous, foolish, vile, foul milksop!
You cullionly, lubbery, heedless, filthy lunatic!
You foul, lecherous, infected, slanderous degenerate!
You base, ruinous, slanderous, false liar!
```

Skills

- Setting random seed from argument
- Random selecting/sampling from a list
- Iterating through a loop a defined number of times
- Formatting string output

abuse Solution

```
#!/usr/bin/env python3

import argparse
import random
import sys

adjectives = """
bankrupt base caterwauling corrupt cullionly detestable dishonest
false filthsome filthy foolish foul gross heedless indistinguishable
infected insatiate irksome lascivious lecherous loathsome lubbery old
peevish rascaly rotten ruinous scurilous scurvy slanderous
sodden-witted thin-faced toad-spotted unmannered vile wall-eyed
""".strip().split()

nouns = """
Judas Satan ape ass barbermonger beggar block boy braggart butt
carbuncle coward coxcomb cur dandy degenerate fiend fishmonger fool
gull harpy jack jolthead knave liar lunatic maw milksop minion
ratcatcher recreant rogue scold slave swine traitor varlet villain worm
""".strip().split()


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Argparse Python script',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('-a',
                        '--adjectives',
                        help='Number of adjectives',
                        metavar='int',
                        type=int,
                        default=2)

    parser.add_argument('-n',
                        '--number',
                        help='Number of insults',
                        metavar='int',
                        type=int,
                        default=3)
```

```
    parser.add_argument('-s',
                        '--seed',
                        help='Random seed',
                        metavar='int',
                        type=int,
                        default=None)

    return parser.parse_args()


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()
    num_adj = args.adjectives
    num_insults = args.number

    random.seed(args.seed)

    for _ in range(num_insults):
        adjs = random.sample(adjectives, k=num_adj)
        noun = random.choice(nouns)
        print('You {} {}!'.format(', '.join(adjs), noun))


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

Chapter 13

BACRONYM

Write a Python program called `bacronym.py` that takes an string like "FBI" and retrofits some `-n|--number` (default 5) of acronyms by reading a `-w|--wordlist` argument (defualt "/usr/share/dict/words"), skipping over words to `-e|--exclude` (default "a, an, the") and randomly selecting words that start with each of the letters. Be sure to include a `-s|--seed` argument (default `None`) to pass to `random.seed` for the test suite.

```
$ ./bacronym.py
usage: bacronym.py [-h] [-n NUM] [-w STR] [-x STR] [-s INT] STR
bacronym.py: error: the following arguments are required: STR
$ ./bacronym.py -h
usage: bacronym.py [-h] [-n NUM] [-w STR] [-x STR] [-s INT] STR

Explain acronyms

positional arguments:
  STR                   Acronym

optional arguments:
  -h, --help            show this help message and exit
  -n NUM, --num NUM     Maximum number of definitions (default: 5)
  -w STR, --wordlist STR
                        Dictionary/word file (default: /usr/share/dict/words)
  -x STR, --exclude STR
                        List of words to exclude (default: a,an,the)
  -s INT, --seed INT    Random seed (default: None)
$ ./bacronym.py FBI -s 1
FBI =
 - Fecundity Brokage Imitant
 - Figureless Basketmaking Ismailite
 - Frumpery Bonedog Irregardless
 - Foxily Blastomyces Inedited
 - Fastland Bouncingly Idiospasm
```

Skills

- Using `argparse`
- Reading words from a file into a list
- Randomly selecting words from a list

- Formatting string output

bacronym Solution

```
#!/usr/bin/env python3
"""Make guesses about acronyms"""

import argparse
import sys
import os
import random
import re
from collections import defaultdict


# --------------------------------------------------
def get_args():
    """get arguments"""
    parser = argparse.ArgumentParser(
        description='Explain acronyms',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('acronym', help='Acronym', type=str, metavar='STR')

    parser.add_argument('-n',
                        '--num',
                        help='Maximum number of definitions',
                        type=int,
                        metavar='NUM',
                        default=5)

    parser.add_argument('-w',
                        '--wordlist',
                        help='Dictionary/word file',
                        type=str,
                        metavar='STR',
                        default='/usr/share/dict/words')

    parser.add_argument('-x',
                        '--exclude',
                        help='List of words to exclude',
                        type=str,
                        metavar='STR',
                        default='a,an,the')

    parser.add_argument('-s',
                        '--seed',
```

```
                        help='Random seed',
                        type=int,
                        metavar='INT',
                        default=None)

    return parser.parse_args()


# --------------------------------------------------
def main():
    """main"""

    args = get_args()
    acronym = args.acronym
    wordlist = args.wordlist
    limit = args.num
    goodword = r'^[a-z]{2,}$'
    badwords = set(re.split(r'\s*,\s*', args.exclude.lower()))

    random.seed(args.seed)

    if not re.match(goodword, acronym.lower()):
        print('"{}" must be >1 in length, only use letters'.format(acronym))
        sys.exit(1)

    if not os.path.isfile(wordlist):
        print('"{}" is not a file.'.format(wordlist))
        sys.exit(1)

    seen = set()
    words_by_letter = defaultdict(list)
    for word in open(wordlist).read().lower().split():
        clean = re.sub('[^a-z]', '', word)
        if not clean:  # nothing left?
            continue

        if re.match(goodword,
                    clean) and clean not in seen and clean not in badwords:
            seen.add(clean)
            words_by_letter[clean[0]].append(clean)

    len_acronym = len(acronym)
    definitions = []
    for i in range(0, limit):
        definition = []
        for letter in acronym.lower():
```

```
            possible = words_by_letter.get(letter, [])
            if len(possible) > 0:
                definition.append(
                    random.choice(possible).title() if possible else '?')

        if len(definition) == len_acronym:
            definitions.append(' '.join(definition))

    if len(definitions) > 0:
        print(acronym.upper() + ' =')
        for definition in definitions:
            print(' - ' + definition)
    else:
        print('Sorry I could not find any good definitions')


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

Chapter 14

Python Blackjack Game

Write a Python program called `blackjack.py` that plays an abbreviated game of Blackjack. You will need to `import random` to get random cards from a deck you will construct, and so your program will need to accept a `-s|--seed` that will set `random.seed()` with the value that is passed in so that the test suite will work. The other arguments you will accept are two flags (Boolean values) of `-p|--player_hits` and `-d|--dealer_hits`. As usual, you will also have a `-h|--help` option for usage statement.

To play the game, the user will run the program and will see a display of what cards the dealer has (noted "D") and what cards the player has (noted "P") along with a sum of the values of the cards. In Blackjack, number cards are worth their value, face cards are worth 10, and the Ace will be worth 1 for our game (though in the real game it can alternate between 1 and 11).

To create your deck of cards, you will need to use the Unicode symbols for the suites ( ) [which won't display in the PDF, so consult the Markdown file].

Combine these with the numbers 2-10 and the letters "A", "J", "Q," and "K" (hint: look at `itertools.product`). Because your game will use randomness, you will need to sort your deck and then use the `random.shuffle` method so that your cards will be in the correct order to pass the tests.

When you make the initial deal, keep in mind how cards are actually dealt – first one card to each of the players, then one to the dealer, then the players, then the dealer, etc. You might be tempted to use `random.choice` or something like that to select your cards, but you need to keep in mind that you are modeling an actual deck and so selected cards should no longer be present in the deck. If the `-p|--player_htis` flag is present, deal an additional card to the player; likewise with the `-d|--dealer_hits` flag.

After displaying the hands, the code should:

1. Check if the player has more than 21; if so, print 'Player busts! You lose, loser!' and exit(0)
2. Check if the dealer has more than 21; if so, print 'Dealer busts.' and exit(0)
3. Check if the player has exactly 21; if so, print 'Player wins. You probably cheated.' and exit(0)
4. Check if the dealer has exactly 21; if so, print 'Dealer wins!' and exit(0)
5. If the either the dealer or the player has less than 18, you should indicate "X should hit."

NB: Look at the Markdown format to see the actual output as the suites won't display in the PDF version!

```
$ ./blackjack.py
D [11]:  J  A
P [18]:  8  10
Dealer should hit.
$ ./blackjack.py
D [13]:  3  J
P [16]:  6  10
Dealer should hit.
Player should hit.
$ ./blackjack.py -s 5
D [ 5]:  4  A
P [19]:  10  9
Dealer should hit.
$ ./blackjack.py -s 3 -p
D [19]:  K  9
P [22]:  3  9  J
Player busts! You lose, loser!
$ ./blackjack.py -s 15 -p
D [19]:  10  9
P [21]:  10  8  3
Player wins. You probably cheated.
```

blackjack Solution

```
#!/usr/bin/env python3

import argparse
import random
import re
import sys
from itertools import product
from dire import die


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Argparse Python script',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument(
        '-s',
        '--seed',
        help='Random seed',
        metavar='int',
        type=int,
        default=None)

    parser.add_argument(
        '-d',
        '--dealer_hits',
        help='Dealer hits',
        action='store_true')

    parser.add_argument(
        '-p',
        '--player_hits',
        help='Player hits',
        action='store_true')

    return parser.parse_args()


# --------------------------------------------------
def bail(msg):
    """print() and exit(0)"""
```

```
    print(msg)
    sys.exit(0)

# --------------------------------------------------
def card_value(card):
    """card to numeric value"""
    val = card[1:]
    faces = {'A': 1, 'J': 10, 'Q': 10, 'K': 10}
    if val.isdigit():
        return int(val)
    elif val in faces:
        return faces[val]
    else:
        die('Unknown card value for "{}"'.format(card))


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

    random.seed(args.seed)

    # seed = args.seed
    # if seed is not None:
    #     random.seed(seed)

    suites = list('  ')
    values = list(range(2, 11)) + list('AJQK')
    cards = sorted(map(lambda t: '{}{}'.format(*t), product(suites, values)))
    random.shuffle(cards)

    p1, d1, p2, d2 = cards.pop(), cards.pop(), cards.pop(), cards.pop()
    player = [p1, p2]
    dealer = [d1, d2]

    if args.player_hits:
        player.append(cards.pop())

    if args.dealer_hits:
        dealer.append(cards.pop())

    player_hand = sum(map(card_value, player))
    dealer_hand = sum(map(card_value, dealer))

    print('D [{:2}]: {}'.format(dealer_hand, ' '.join(dealer)))
```

```
    print('P [{:2}]: {}'.format(player_hand, ' '.join(player)))

    if player_hand > 21:
        bail('Player busts! You lose, loser!')
    elif dealer_hand > 21:
        bail('Dealer busts.')
    elif player_hand == 21:
        bail('Player wins. You probably cheated.')
    elif dealer_hand == 21:
        bail('Dealer wins!')

    if dealer_hand < 18:
        print('Dealer should hit.')

    if player_hand < 18:
        print('Player should hit.')


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

Chapter 15

Family Tree

Write a program called `tree.py` that will take an input file as a single positional argument and produce a graph of the family tree described therein. The file can have only three kinds of statements:

1. `INITIALS = Full Name`
2. `person1 married person2`
3. `person1 and person2 begat child1[, child2...]`

Use the `graphviz` module to generate a graph like the `kyc.gv.pdf` included here that was generated from the following input:

```
$ cat tudor.txt
H7 = Henry VII
EOY = Elizabeth of York
H8 = Henry VIII
COA = Catherine of Aragon
AB = Anne Boleyn
JS = Jane Seymour
AOC = Anne of Cleves
CH = Catherine Howard
CP = Catherine Parr
HDC = Henry, Duke of Cornwall
M1 = Mary I
E1 = Elizabeth I
E6 = Edward VI

H7 married EOY
H7 and EOY begat H8
H8 married COA
H8 married AB
H8 married JS
H8 married AOC
H8 married CH
H8 married CP
H8 and COA begat HDC, M1
H8 and AB begat E1
H8 and JS begat E6
$ ./tree.py tudor.txt
Done, see output in "tudor.txt.gv".
```

Chapter 17

Guessing Game

Write a Python program called `guess.py` that plays a guessing game for a number between a `-m|--min` and `-x|--max` value (default 1 and 50, respectively) with a limited number of `-g|--guesses` (default 5). Complain if either `--min` or `--guesses` is less than 1. Accept a `-s|--seed` for `random.seed`. If the user guesses something that is not a number, complain about it.

The game is intended to actually be interactive, which makes it difficult to test. Here is how it should look in interactive mode:

```
$ ./guess.py -s 1
Guess a number between 1 and 50 (q to quit): 25
"25" is too high.
Guess a number between 1 and 50 (q to quit): foo
"foo" is not a number.
Guess a number between 1 and 50 (q to quit): 12
"12" is too high.
Guess a number between 1 and 50 (q to quit): 6
"6" is too low.
Guess a number between 1 and 50 (q to quit): 9
"9" is correct. You win!
```

Because I want to be able to write a test for this, I also want the program to accept an `-i|--inputs` option so that the game can also be played exactly the same but without the prompts for input:

```
$ ./guess.py -s 1 -i 25 foo 12 6 9
"25" is too high.
"foo" is not a number.
"12" is too high.
"6" is too low.
"9" is correct. You win!
```

You should be able to handle this in your inifinite game loop.

guess Solution

```
#!/usr/bin/env python3

import argparse
import random
import re
import sys
from dire import die


# --------------------------------------------------
def get_args():
    """get args"""
    parser = argparse.ArgumentParser(
        description='Guessing game',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('-m',
                        '--min',
                        help='Minimum value',
                        metavar='int',
                        type=int,
                        default=1)

    parser.add_argument('-x',
                        '--max',
                        help='Maximum value',
                        metavar='int',
                        type=int,
                        default=50)

    parser.add_argument('-g',
                        '--guesses',
                        help='Number of guesses',
                        metavar='int',
                        type=int,
                        default=5)

    parser.add_argument('-s',
                        '--seed',
                        help='Random seed',
                        metavar='int',
                        type=int,
                        default=None)
```

```
    parser.add_argument('-i',
                        '--inputs',
                        help='Inputs',
                        metavar='str',
                        type=str,
                        nargs='+',
                        default=[])

    return parser.parse_args()


# --------------------------------------------------
def main():
    """main"""
    args = get_args()
    low = args.min
    high = args.max
    guesses_allowed = args.guesses
    inputs = args.inputs
    random.seed(args.seed)

    if low < 1:
        die('--min "{}" cannot be lower than 1'.format(low))

    if guesses_allowed < 1:
        die('--guesses "{}" cannot be lower than 1'.format(guesses_allowed))

    if low > high:
        die('--min "{}" is higher than --max "{}"'.format(low, high))

    secret = random.randint(low, high)
    prompt = 'Guess a number between {} and {} (q to quit): '.format(low, high)
    num_guesses = 0

    while True:
        guess = inputs.pop(0) if inputs else input(prompt)
        num_guesses += 1

        if re.match('q(uit)?', guess.lower()):
            print('Now you will never know the answer.')
            sys.exit()

        # Method 1: test if the guess is a digit
        if not guess.isdigit():
            print('"{}" is not a number.'.format(guess))
```

```
            continue
        num = int(guess)

        # Method 2: try/except
        num = 0
        try:
            num = int(guess)
        except:
            warn('"{}" is not an integer'.format(guess))
            continue

        if not low <= num <= high:
            print('Number "{}" is not in the allowed range'.format(num))
        elif num == secret:
            print('"{}" is correct. You win!'.format(num))
            break
        else:
            print('"{}" is too {}.'.format(num,
                                           'low' if num < secret else 'high'))

        if num_guesses >= guesses_allowed:
            print(
                'Too many guesses, loser! The number was "{}."'.format(secret))
            sys.exit(1)


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

Chapter 18

Kentucky Fryer

Write a Python program called `fryer.py` that reads some input text from a single positional argument on the command line (which could be a file to read) and transforms the text by dropping the "g" from words two-syllable words ending in "-ing" and also changes "you" to "y'all". Be mindful to keep the case the same on the first letter, e.g, "You" should become "Y'all," "Hunting" should become "Huntin"'.

```
$ ./fryer.py
usage: fryer.py [-h] str
fryer.py: error: the following arguments are required: str
$ ./fryer.py -h
usage: fryer.py [-h] str

Southern fry text

positional arguments:
  str         Input text or file

optional arguments:
  -h, --help  show this help message and exit
$ ./fryer.py you
y'all
$ ./fryer.py Fishing
Fishin'
$ ./fryer.py string
string
$ cat tests/input1.txt
So I was fixing to ask him, "Do you want to go fishing?" I was dying
to go for a swing and maybe do some swimming, too.
$ ./fryer.py tests/input1.txt
So I was fixin' to ask him, "Do y'all want to go fishing?" I was dyin'
to go for a swing and maybe do some swimmin', too.
```

kentucky_fryer Solution

```
#!/usr/bin/env python3

import argparse
import os
import re
import sys


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Southern fry text',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('text', metavar='str', help='Input text or file')

    return parser.parse_args()


# --------------------------------------------------
def fry(word):
    """
    Drop the 'g' from '-ing' words, change "you" to "y'all"
    """

    ing_word = re.search('(.+)ing([:;,.?])?$', word)
    you = re.match('([Yy])ou$', word)

    if ing_word:
        prefix = ing_word.group(1)
        if re.search('[aeiouy]', prefix):
            return prefix + "in'" + (ing_word.group(2) or '')
    elif you:
        return you.group(1) + "'all"

    return word


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

```
    args = get_args()
    text = args.text

    if os.path.isfile(text):
        text = open(text).read()

    for line in text.splitlines():
        print(' '.join(map(fry, line.rstrip().split())))


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

Chapter 20

Markov Chains for Words

Write a Python program called `markov.py` that uses the Markov chain algorithm to generate new words from a set of training files. The program should take one or more positional arguments which are files that you read, word-by-word, and note the options of letters after a given `-k|--kmer_size` (default `2`) grouping of letters. E.g., in the word "alabama" with `k=1`, the frequency table will look like:

```
a = l, b, m
l = a
b = a
m = a
```

That is, given this training set, if you started with `l` you could only choose an `a`, but if you have `a` then you could choose `l`, `b`, or `m`.

The program should generate `-n|--num_words` words (default `10`), each a random size between `k` + 2 and a `-m|--max_word` size (default `12`). Be sure to accept `-s|--seed` to pass to `random.seed`. My solution also takes a `-d|--debug` flag that will emit debug messages to `.log` for you to inspect.

Chose the best words and create definitions for them:

- yulcogicism: the study of Christmas gnostics
- umjamp: skateboarding trick
- callots: insignia of officers in Greek army
- urchenev: fungal growth found under cobblestones

```
$ ./markov.py
usage: markov.py [-h] [-n int] [-k int] [-m int] [-s int] [-d] FILE [FILE ...]
markov.py: error: the following arguments are required: FILE
$ ./markov.py -h
usage: markov.py [-h] [-n int] [-k int] [-m int] [-s int] [-d] FILE [FILE ...]

Markov chain for characters/words

positional arguments:
  FILE                  Training file(s)

optional arguments:
  -h, --help            show this help message and exit
  -n int, --num_words int
                        Number of words to generate (default: 10)
  -k int, --kmer_size int
```

```
                        Kmer size (default: 2)
  -m int, --max_word int
                        Max word length (default: 12)
  -s int, --seed int    Random seed (default: None)
  -d, --debug           Debug to ".log" (default: False)
$ ./markov.py /usr/share/dict/words -s 1
  1: oveli
  2: uming
  3: uylatiteda
  4: owsh
  5: uuse
  6: ismandl
  7: efortai
  8: eyhopy
  9: auretrab
 10: ozogralach
$ ./markov.py ../inputs/const.txt -s 2 -k 3
  1: romot
  2: leasonsusp
  3: gdoned
  4: bunablished
  5: neithere
  6: achmen
  7: reason
  8: nmentyone
  9: effereof
 10: eipts
```

markov_words Solution

```
#!/usr/bin/env python3

import argparse
import logging
import os
import random
import re
import sys
from collections import defaultdict


# --------------------------------------------------
def get_args():
    """Get command-line arguments"""

    parser = argparse.ArgumentParser(
        description='Markov chain for characters/words',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('file',
                        metavar='FILE',
                        nargs='+',
                        help='Training file(s)')

    parser.add_argument('-n',
                        '--num_words',
                        help='Number of words to generate',
                        metavar='int',
                        type=int,
                        default=10)

    parser.add_argument('-k',
                        '--kmer_size',
                        help='Kmer size',
                        metavar='int',
                        type=int,
                        default=2)

    parser.add_argument('-m',
                        '--max_word',
                        help='Max word length',
                        metavar='int',
                        type=int,
```

```
                        default=12)

    parser.add_argument('-s',
                        '--seed',
                        help='Random seed',
                        metavar='int',
                        type=int,
                        default=None)

    parser.add_argument('-d',
                        '--debug',
                        help='Debug to ".log"',
                        action='store_true')

    return parser.parse_args()


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

    args = get_args()
    k = args.kmer_size
    random.seed(args.seed)

    logging.basicConfig(
        filename='.log',
        filemode='w',
        level=logging.DEBUG if args.debug else logging.CRITICAL)

    # debate use of set/list in terms of letter frequencies
    chains = defaultdict(list)
    for file in args.file:
        for line in open(file):
            for word in line.lower().split():
                word = re.sub('[^a-z]', '', word)
                for i in range(0, len(word) - k):
                    kmer = word[i:i + k + 1]
                    chains[kmer[:-1]].append(kmer[-1])

    logging.debug(chains)

    kmers = list(chains.keys())
    starts = set()

    for i in range(1, args.num_words + 1):
```

```
        word = ''
        while not word:
            kmer = random.choice(kmers)
            if not kmer in starts and chains[kmer] and re.search(
                    '[aeiou]', kmer):
                starts.add(kmer)
                word = kmer

        length = random.choice(range(k + 2, args.max_word))
        logging.debug('Make a word {} long starting with "{}"'.format(
            length, word))
        while len(word) < length:
            if not chains[kmer]: break
            char = random.choice(list(chains[kmer]))
            logging.debug('char = "{}"'.format(char))
            word += char
            kmer = kmer[1:] + char

        logging.debug('word = "{}"'.format(word))
        print('{:3}: {}'.format(i, word))


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

Chapter 21

Pig Latin

Write a Python program named `piggie.py` that takes one or more file names as positional arguments and converts all the words in them into "Pig Latin" (see rules below). Write the output to a directory given with the flags `-o|--outdir` (default `out-yay`) using the same basename as the input file, e.g., `input/foo.txt` would be written to `out-yay/foo.txt`.

if a file argument names a non-existent file, print a warning to STDERR and skip that file. If the output directory does not exist, create it.

Pig Latin Rules

1. If the word begins with consonants, e.g., "k" or "ch", move them to the end of the word and append "ay" so that "mouse" becomes "ouse-may" and "chair" becomes "air-chay."
2. If the word begins with a vowel, simple append "-yay" to the end, so "apple" is "apple-yay."

Expected Output

```
$ ./piggie.py
usage: piggie.py [-h] [-o str] FILE [FILE ...]
piggie.py: error: the following arguments are required: FILE
$ ./piggie.py -h
usage: piggie.py [-h] [-o str] FILE [FILE ...]

Convert to Pig Latin

positional arguments:
  FILE                  Input file(s)

optional arguments:
  -h, --help            show this help message and exit
  -o str, --outdir str  Output directory (default: out-yay)
[cholla@~/work/python/playful_python/piggie]$ ./piggie.py
usage: piggie.py [-h] [-o str] FILE [FILE ...]
piggie.py: error: the following arguments are required: FILE
[cholla@~/work/python/playful_python/piggie]$ ./piggie.py -h
usage: piggie.py [-h] [-o str] FILE [FILE ...]
```

```
Convert to Pig Latin

positional arguments:
  FILE                  Input file(s)

optional arguments:
  -h, --help            show this help message and exit
  -o str, --outdir str  Output directory (default: out-yay)
$ ./piggie.py ../inputs/sonnet-29.txt
  1: sonnet-29.txt
Done, wrote 1 file to "out-yay".
$ head out-yay/sonnet-29.txt
onnet-Say 29-yay
illiam-Way akespeare-Shay

en-Whay, in-yay isgrace-day ith-way ortune-fay and-yay en-may's-yay eyes-yay,
I-yay all-yay alone-yay eweep-bay y-may outcast-yay ate-stay,
And-yay ouble-tray eaf-day eaven-hay ith-way y-may ootless-bay ies-cray,
And-yay ook-lay upon-yay elf-mysay and-yay urse-cay y-may ate-fay,
ishing-Way e-may ike-lay o-tay one-yay ore-may ich-ray in-yay ope-hay,
eatured-Fay ike-lay im-hay, ike-lay im-hay ith-way iends-fray ossessed-pay,
esiring-Day is-thay an-may's-yay art-yay and-yay at-thay an-may's-yay ope-scay,
```

piggie Solution

```
#!/usr/bin/env python3
"""Convert text to Pig Latin"""

import argparse
import os
import re
import string
from dire import warn


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

    parser = argparse.ArgumentParser(
        description='Convert to Pig Latin',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('file',
                        metavar='FILE',
                        nargs='+',
                        help='Input file(s)')

    parser.add_argument('-o',
                        '--outdir',
                        help='Output directory',
                        metavar='str',
                        type=str,
                        default='out-yay')

    return parser.parse_args()


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

    args = get_args()
    out_dir = args.outdir

    if not os.path.isdir(out_dir):
        os.makedirs(out_dir)
```

```
    num_files = 0
    for i, file in enumerate(args.file, start=1):
        basename = os.path.basename(file)
        out_file = os.path.join(out_dir, basename)
        out_fh = open(out_file, 'wt')
        print('{:3}: {}'.format(i, basename))

        if not os.path.isfile(file):
            warn('"{}" is not a file.'.format(file))
            continue

        num_files += 1
        for line in open(file):
            for bit in re.split(r"([\w']+)", line):
                out_fh.write(pig(bit))

        out_fh.close()

    print('Done, wrote {} file{} to "{}".'.format(
        num_files, '' if num_files == 1 else 's', out_dir))


# --------------------------------------------------
def pig(word):
    """Create Pig Latin version of a word"""

    if re.match(r"^[\w']+$", word):
        consonants = re.sub('[aeiouAEIOU]', '', string.ascii_letters)
        match = re.match('^([' + consonants + ']+)(.+)', word)
        if match:
            word = '-'.join([match.group(2), match.group(1) + 'ay'])
        else:
            word = word + '-yay'

    return word


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

Chapter 23

Substring Guessing Game

Write a Python program called `sub.py` that plays a guessing game where you read a `-f|--file` input (default `/usr/share/dict/words`) and use a given `-k|--ksize` to find all the words grouped by their shared kmers. Remove any kmers where the number of words is fewer than `-m|--min_words`. Also accept a `-s|--seed` for `random.seed` for testing purposes. Prompt the user to guess a word for a randomly chosen kmer. If their guess is not present in the shared list, taunt them mercilessly. If their guess is present, affirm their worth and prompt to guess again. Allow them to use `!` to quit and `?` to be provided a hint (a word from the list). For both successful guesses and hints, remove the word from the shared list. When they have quit or exhausted the list, quit play. At the end of the game, report the number of found words.

substring Solution

```
#!/usr/bin/env python3

import argparse
import os
import random
import re
import sys
from collections import defaultdict
from dire import die


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Find words sharing a substring',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('-f',
                        '--file',
                        metavar='str',
                        help='Input file',
                        type=str,
                        default='/usr/share/dict/words')

    parser.add_argument('-s',
                        '--seed',
                        help='Random seed',
                        metavar='int',
                        type=int,
                        default=None)

    parser.add_argument('-m',
                        '--min_words',
                        help='Minimum number of words for a given kmer',
                        metavar='int',
                        type=int,
                        default=3)

    parser.add_argument('-k',
                        '--ksize',
                        help='Size of k',
                        metavar='int',
```

```
                        type=int,
                        default=4)

    return parser.parse_args()


# --------------------------------------------------
def get_words(file):
    """Get words from input file"""

    if not os.path.isfile(file):
        die('"{}" is not a file')

    words = set()
    for line in open(file):
        for word in line.split():
            words.add(re.sub('[^a-zA-Z0-9]', '', word.lower()))

    if not words:
        die('No usable words in "{}"'.format(file))

    return words


# --------------------------------------------------
def get_kmers(words, k, min_words):
    """ Find all words sharing kmers"""

    if k <= 1:
        die('-k "{}" must be greater than 1'.format(k))

    shared = defaultdict(list)
    for word in words:
        for kmer in [word[i:i + k] for i in range(len(word) - k + 1)]:
            shared[kmer].append(word)

    # Select kmers having enough words (can't use `pop`!)

    # Method 1: for loop
    ok = dict()
    for kmer in shared:
        if len(shared[kmer]) >= min_words:
            ok[kmer] = shared[kmer]

    # Method 2: list comprehension
    # ok = dict([(kmer, shared[kmer]) for kmer in shared
```

```
    #            if len(shared[kmer]) >= min_words])

    # Method 3: map/filter
    # ok = dict(
    #     map(lambda kmer: (kmer, shared[kmer]),
    #         filter(lambda kmer: len(shared[kmer]) >= min_words,
    #                shared.keys())))

    return ok


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

    args = get_args()

    random.seed(args.seed)

    shared = get_kmers(get_words(args.file), args.ksize, args.min_words)

    # Choose a kmer, setup game state
    kmer = random.choice(list(shared.keys()))
    guessed = set()
    found = []
    prompt = 'Name a word that contains "{}" [!=quit, ?=hint] '.format(kmer)
    compliments = ['Nice', 'Rock on', 'Totes', 'Fantastic', 'Excellent']
    taunts = [
        'Surely you jest!', 'Are you kidding me?',
        'You must have rocks for brains.', 'What is wrong with you?'
    ]

    #print(kmer, shared[kmer])

    while True:
        num_left = len(shared[kmer])
        if num_left == 0:
            print('No more words!')
            break

        guess = input(prompt + '({} left) '.format(num_left)).lower()

        if guess == '?':
            # Provide a hint
            pos = random.choice(range(len(shared[kmer])))
            word = shared[kmer].pop(pos)
```

```
            print('For instance, "{}"...'.format(word))

        elif guess == '!':
            # Bail
            print('Quitter!')
            break

        elif guess in guessed:
            # Chastise
            print('You have already guessed "{}"'.format(guess))

        elif guess in shared[kmer]:
            # Remove the word, feedback with compliment
            pos = shared[kmer].index(guess)
            word = shared[kmer].pop(pos)
            print('{}! "{}" is found!'.format(random.choice(compliments),
                                              word))
            found.append(word)
            guessed.add(guess)

        else:
            # Taunt
            print(random.choice(taunts))

    # Game over, man!
    if found:
        n = len(found)
        print('Hey, you found {} word{}! Not bad.'.format(
            n, '' if n == 1 else 's'))
    else:
        print('Wow, you found no words. You suck!')


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

Chapter 24

Tic-Tac-Toe Outcome

Create a Python program called `outcome.py` that takes a given Tic-Tac-Toe state as it's only (positional) argument and reports if X or O has won or if there is no winner. The state should only contain the characters ".", "O", and "X", and must be exactly 9 characters long. If there is not exactly one argument, print a "usage" statement.

Expected Behavior

```
$ ./outcome.py
Usage: outcome.py STATE
$ ./outcome.py ..X.OA..X
State "..X.OA..X" must be 9 characters of only ., X, O
$ ./outcome.py ..X.OX...
No winner
$ ./outcome.py ..X.OX..X
X has won
```

tictactoe Solution

```
#!/usr/bin/env python3

import os
import re
import sys


# --------------------------------------------------
def main():
    args = sys.argv[1:]

    if len(args) != 1:
        print('Usage: {} STATE'.format(os.path.basename(sys.argv[0])))
        sys.exit(1)

    state = args[0]

    if not re.search('^[.XO]{9}$', state):
        print('State "{}" must be 9 characters of only ., X, O'.format(state),
              file=sys.stderr)
        sys.exit(1)

    winning = [[0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 3, 6], [1, 4, 7],
               [2, 5, 8], [0, 4, 8], [2, 4, 6]]

    winner = 'No winner'

    # for player in ['X', 'O']:
    #     for combo in winning:
    #         i, j, k = combo
    #         if state[i] == player and state[j] == player and state[k] == player:
    #             winner = player
    #             break

    # for player in ['X', 'O']:
    #     for combo in winning:
    #         chars = []
    #         for i in combo:
    #             chars.append(state[i])

    #         if ''.join(chars) == player * 3:
    #             winner = player
    #             break
```

```
    # for player in ['X', 'O']:
    #     for i, j, k in winning:
    #         chars = ''.join([state[i], state[j], state[k]])
    #         if ''.join(chars) == '{}{}{}'.format(player, player, player):
    #             winner = player
    #             break

    for player in ['X', 'O']:
        for i, j, k in winning:
            combo = [state[i], state[j], state[k]]
            if combo == [player, player, player]:
                winner = '{} has won'.format(player)
                break

    # for combo in winning:
    #     group = list(map(lambda i: state[i], combo))
    #     for player in ['X', 'O']:
    #         if all(x == player for x in group):
    #             winner = player
    #             break

    print(winner)


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

Chapter 25

Twelve Days of Christmas

Write a Python program called `twelve_days.py` that will generate the "Twelve Days of Christmas" song up to the `-n|--number_days` argument (default `12`), writing the resulting text to the `-o|--outfile` argument (default STDOUT).

```
$ ./twelve_days.py -h
usage: twelve_days.py [-h] [-o str] [-n int]

Twelve Days of Christmas

optional arguments:
  -h, --help            show this help message and exit
  -o str, --outfile str
                        Outfile (STDOUT) (default: )
  -n int, --number_days int
                        Number of days to sing (default: 12)
$ ./twelve_days.py -n 1
On the first day of Christmas,
My true love gave to me,
A partridge in a pear tree.

$ ./twelve_days.py -n 3
On the first day of Christmas,
My true love gave to me,
A partridge in a pear tree.

On the second day of Christmas,
My true love gave to me,
Two turtle doves,
And a partridge in a pear tree.

On the third day of Christmas,
My true love gave to me,
Three French hens,
Two turtle doves,
And a partridge in a pear tree.

$ ./twelve_days.py -o out
$ wc -l out
     113 out
```

twelve_days_of_christmas Solution

```
#!/usr/bin/env python3

import argparse
import sys
from dire import die


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Twelve Days of Christmas',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('-o',
                        '--outfile',
                        help='Outfile (STDOUT)',
                        metavar='str',
                        type=str,
                        default='')

    parser.add_argument('-n',
                        '--number_days',
                        help='Number of days to sing',
                        metavar='int',
                        type=int,
                        default=12)

    return parser.parse_args()


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()
    out_file = args.outfile
    num_days = args.number_days

    out_fh = open(out_file, 'wt') if out_file else sys.stdout

    days = {
        12: 'Twelve drummers drumming',
        11: 'Eleven pipers piping',
```

```
    10: 'Ten lords a leaping',
    9: 'Nine ladies dancing',
    8: 'Eight maids a milking',
    7: 'Seven swans a swimming',
    6: 'Six geese a laying',
    5: 'Five gold rings',
    4: 'Four calling birds',
    3: 'Three French hens',
    2: 'Two turtle doves',
    1: 'a partridge in a pear tree',
}

cardinal = {
    12: 'twelfth',
    11: 'eleven',
    10: 'tenth',
    9: 'ninth',
    8: 'eighth',
    7: 'seventh',
    6: 'sixth',
    5: 'fifth',
    4: 'fourth',
    3: 'third',
    2: 'second',
    1: 'first',
}

if not num_days in days:
    die('Cannot sing "{}" days'.format(num_days))

def ucfirst(s):
    return s[0].upper() + s[1:]

for i in range(1, num_days + 1):
    first = 'On the {} day of Christmas,\nMy true love gave to me,'
    out_fh.write(first.format(cardinal[i]) + '\n')
    for j in reversed(range(1, i + 1)):
        if j == 1:
            if i == 1:
                out_fh.write('{}.\n'.format(ucfirst(days[j])))
            else:
                out_fh.write('And {}.\n'.format(days[j]))
        else:
            out_fh.write('{},\n'.format(days[j]))

    if i < max(days.keys()):
```

```
            out_fh.write('\n')


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

Chapter 26

War Card Game in Python

> The generation of random numbers is too important to be left to chance. – Robert R. Coveyou

Create a Python program called "war.py" that plays the card game "War." The program will use the `random` module to shuffle a deck of cards, so your program will need to accept a `-s|--seed` argument (default: `None`) which you will use to call `random.seed`, if present.

First you program will need to create a deck of cards. You will need to use the Unicode symbols for the suites ( ) [which won't display in the PDF, so consult the Markdown file] and combine those with the numbers 2-10 and the letters "J", "Q," "K," and "A." (hint: look at `itertools.product`).

```
>>> from itertools import product
>>> a = list('ABC')
>>> b = range(3)
>>> list(product(a, b))
[('A', 0), ('A', 1), ('A', 2), ('B', 0), ('B', 1), ('B', 2), ('C', 0), ('C', 1), ('C', 2)]
```

NB: You must sort your deck and then use the `random.shuffle` method so that your cards will be in the correct order to pass the tests!

In the real game of War, the cards are shuffled and then dealt one card each first to the non-dealer, then to the dealer, until all cards are dealt and each player has 26 cards. We will not be modeling this behavior. When writing your version of the game, simply `pop` two cards off the deck as the cards for player 1 and player 2, respectively. Compare the two cards by ignoring the suite and evaluating the value where 2 is the lowest and Aces are the highest. When two cards have the same values (e.g., two 5s or two Jacks), print "WAR!" In the real game, this initiates a sub-game of War which is a "recursive" algorithm which we will not bother modeling. Keep track of which player wins each round where no points are awarded in a tie. At the end, report the points for each player and state the winner. In the event of a tie, print "DRAW."

```
$ ./war.py -h
usage: war.py [-h] [-s int]

"War" cardgame

optional arguments:
  -h, --help          show this help message and exit
  -s int, --seed int  Random seed (default: None)
$ ./war.py -s 1
```

```
 9  J P2
 A  5 P1
 4  8 P2
 6  3 P1
 5  3 P1
 K 10 P1
 7  7 WAR!
 2  4 P2
 2 10 P2
 6  5 P1
 2  6 P2
 4  8 P2
 J  9 P1
10  Q P2
 8  7 P1
 K  Q P1
10  2 P1
 9  9 WAR!
 8  J P2
 3  5 P2
 Q  4 P1
 6  A P2
 K  7 P1
 Q  3 P1
 A  K P1
 A  J P1
P1 14 P2 10: Player 1 wins
$ ./war.py -s 2
 4  6 P2
 K  J P1
 J  4 P1
 7  4 P1
 Q 10 P1
 5  3 P1
 K  9 P1
 2  Q P2
 7  A P2
 3  A P2
 5  8 P2
 2 10 P2
10  K P2
 2  3 P2
 Q  8 P1
 6  J P2
 6  8 P2
 8  7 P1
```

```
 5  2 P1
 6  J P2
 9  9 WAR!
 K  A P2
10  Q P2
 7  5 P1
 9  A P2
 4  3 P1
P1 11 P2 14: Player 2 wins
$ ./war.py -s 10
 J  3 P1
 2  5 P2
 Q 10 P1
10  4 P1
 6  5 P1
 3  J P2
 K  8 P1
 5  8 P2
 5  3 P1
 J 10 P1
10  J P2
 A  7 P1
 K  Q P1
 7  A P2
 9  9 WAR!
 2  6 P2
 K  A P2
 6  Q P2
 8  9 P2
 3  7 P2
 8  Q P2
 6  4 P1
 7  2 P1
 4  4 WAR!
 9  2 P1
 K  A P2
P1 12 P2 12: DRAW
```

war Solution

```
#!/usr/bin/env python3

import argparse
import random
import sys
from itertools import product


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='"War" cardgame',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('-s',
                        '--seed',
                        help='Random seed',
                        metavar='int',
                        type=int,
                        default=None)

    return parser.parse_args()


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()
    seed = args.seed

    if seed is not None:
        random.seed(seed)

    suits = list('♠♥♦♣')
    values = list(map(str, range(2, 11))) + list('JQKA')
    cards = sorted(map(lambda t: '{}{}'.format(*t), product(suits, values)))
    random.shuffle(cards)

    p1_wins = 0
    p2_wins = 0

    card_value = dict(
```

```
        list(map(lambda t: list(reversed(t)), enumerate(list(values)))))

    while cards:
        p1, p2 = cards.pop(), cards.pop()
        v1, v2 = card_value[p1[1:]], card_value[p2[1:]]
        res = ''

        if v1 > v2:
            p1_wins += 1
            res = 'P1'
        elif v2 > v1:
            p2_wins += 1
            res = 'P2'
        else:
            res = 'WAR!'

        print('{:>3} {:>3} {}'.format(p1, p2, res))

    print('P1 {} P2 {}: {}'.format(
        p1_wins, p2_wins, 'Player 1 wins' if p1_wins > p2_wins else
        'Player 2 wins' if p2_wins > p1_wins else 'DRAW'))


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

Chapter 27

Anagram

Write a program called `presto.py` that will find anagrams of a given positional argument. The program should take an optional `-w|--wordlist` (default `/usr/share/dict/words`) and produce output that includes combinations of `-n|num_combos` words (default `1`) that are anagrams of the given input.

```
$ ./presto.py
usage: presto.py [-h] [-w str] [-n int] [-d] str
presto.py: error: the following arguments are required: str
$ ./presto.py -h
usage: presto.py [-h] [-w str] [-n int] [-d] str

Find anagrams

positional arguments:
  str                   Input text

optional arguments:
  -h, --help            show this help message and exit
  -w str, --wordlist str
                        Wordlist (default: /usr/share/dict/words)
  -n int, --num_combos int
                        Number of words combination to test (default: 1)
  -d, --debug           Debug (default: False)
$ ./presto.py presto
presto =
   1. poster
   2. repost
   3. respot
   4. stoper
$ ./presto.py listen
listen =
   1. enlist
   2. silent
   3. tinsel
$ ./presto.py listen -n 2 | tail
  82. sten li
  83. te nils
  84. ten lis
  85. ten sil
  86. ti lens
  87. til ens
```

88. til sen
89. tin els
90. tin les
91. tinsel

anagram Solution

```
#!/usr/bin/env python3

import argparse
import logging
import os
import re
import sys
from collections import defaultdict, Counter
from itertools import combinations, permutations, product, chain
from dire import warn, die


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Find anagrams',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('text', metavar='str', help='Input text')

    parser.add_argument('-w',
                        '--wordlist',
                        help='Wordlist',
                        metavar='str',
                        type=str,
                        default='/usr/share/dict/words')

    parser.add_argument('-n',
                        '--num_combos',
                        help='Number of words combination to test',
                        metavar='int',
                        type=int,
                        default=1)

    parser.add_argument('-d', '--debug', help='Debug', action='store_true')

    return parser.parse_args()


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

```
args = get_args()
text = args.text
word_list = args.wordlist

if not os.path.isfile(word_list):
    die('--wordlist "{}" is not a file'.format(word_list))

logging.basicConfig(
    filename='.log',
    filemode='w',
    level=logging.DEBUG if args.debug else logging.CRITICAL)

words = defaultdict(set)
for line in open(word_list):
    for word in line.split():
        clean = re.sub('[^a-z0-9]', '', word.lower())
        if len(clean) == 1 and clean not in 'ai':
            continue
        words[len(clean)].add(clean)

text_len = len(text)
counts = Counter(text)
anagrams = set()
lengths = list(words.keys())
for i in range(1, args.num_combos + 1):
    key_combos = list(
        filter(
            lambda t: sum(t) == text_len,
            set(
                map(lambda t: tuple(sorted(t)),
                    combinations(chain(lengths, lengths), i)))))

    for keys in key_combos:
        logging.debug('Searching keys {}'.format(keys))
        word_combos = list(product(*list(map(lambda k: words[k], keys))))

        for t in word_combos:
            if Counter(''.join(t)) == counts:
                for p in filter(
                        lambda x: x != text,
                        map(lambda x: ' '.join(x), permutations(t))):
                    anagrams.add(p)

        logging.debug('# anagrams = {}'.format(len(anagrams)))

logging.debug('Finished searching')
```

```
    if anagrams:
        print('{} ='.format(text))
        for i, t in enumerate(sorted(anagrams), 1):
            print('{:4}. {}'.format(i, t))
    else:
        print('No anagrams for "{}".'.format(text))


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

Chapter 28

Hangman

Write a Python program called `hangman.py` that will play a game of Hangman which is a bit like "Wheel of Fortune" where you present the user with a number of elements indicating the length of a word. For our game, use the underscore `_` to indicate a letter that has not been guessed. The program should take `-n|--minlen` minimum length (default `5`) and `-l|--maxlen` maximum length options (default `10`) to indicate the minimum and maximum lengths of the randomly chosen word taken from the `-w|--wordlist` option (default `/usr/share/dict/words`). It also needs to take `-s|--seed` to for the random seed and the `-m|--misses` number of misses to allow the player.

To play, you will initiate an inifinite loop and keep track of the game state, e.g., the word to guess, the letters already guessed, the letters found, the number of misses. As this is an interactive game, I cannot write an test suite, so you can play my version and then try to write one like it. If the user guesses a letter that is in the word, replace the `_` characters with the letter. If the user guesses the same letter twice, admonish them. If the user guesses a letter that is not in the word, increment the misses and let them know they missed. If the user guesses too many times, exit the game and insult them. If they correctly guess the word, let them know and exit the game.

```
$ ./hangman.py -h
usage: hangman.py [-h] [-l MAXLEN] [-n MINLEN] [-m MISSES] [-s SEED]
                  [-w WORDLIST]

Hangman

optional arguments:
  -h, --help            show this help message and exit
  -l MAXLEN, --maxlen MAXLEN
                        Max word length (default: 10)
  -n MINLEN, --minlen MINLEN
                        Min word length (default: 5)
  -m MISSES, --misses MISSES
                        Max number of misses (default: 10)
  -s SEED, --seed SEED  Random seed (default: None)
  -w WORDLIST, --wordlist WORDLIST
                        Word list (default: /usr/share/dict/words)
$ ./hangman.py
_ _ _ _ _ _ _ _ (Misses: 0)
Your guess? ("?" for hint, "!" to quit) a
_ _ _ _ _ _ _ _ (Misses: 1)
```

```
Your guess? ("?" for hint, "!" to quit) i
_ _ _ _ _ _ i _ (Misses: 1)
Your guess? ("?" for hint, "!" to quit) e
_ _ _ _ _ _ i _ (Misses: 2)
Your guess? ("?" for hint, "!" to quit) o
_ o _ _ _ _ i _ (Misses: 2)
Your guess? ("?" for hint, "!" to quit) u
_ o _ _ _ _ i _ (Misses: 3)
Your guess? ("?" for hint, "!" to quit) y
_ o _ _ _ _ i _ (Misses: 4)
Your guess? ("?" for hint, "!" to quit) c
_ o _ _ _ _ i _ (Misses: 5)
Your guess? ("?" for hint, "!" to quit) d
_ o _ _ _ _ i _ (Misses: 6)
Your guess? ("?" for hint, "!" to quit) p
_ o _ _ _ _ i p (Misses: 6)
Your guess? ("?" for hint, "!" to quit) m
_ o _ _ _ _ i p (Misses: 7)
Your guess? ("?" for hint, "!" to quit) n
_ o _ _ _ _ i p (Misses: 8)
Your guess? ("?" for hint, "!" to quit) s
_ o s _ s _ i p (Misses: 8)
Your guess? ("?" for hint, "!" to quit) t
_ o s t s _ i p (Misses: 8)
Your guess? ("?" for hint, "!" to quit) h
You win. You guessed "hostship" with "8" misses!
$ ./hangman.py -m 2
_ _ _ _ _ _ _ _ _ _ (Misses: 0)
Your guess? ("?" for hint, "!" to quit) a
_ _ _ _ _ _ a _ _ a (Misses: 0)
Your guess? ("?" for hint, "!" to quit) b
_ _ _ _ _ _ a _ _ a (Misses: 1)
Your guess? ("?" for hint, "!" to quit) c
You lose, loser!  The word was "metromania."
```

hangman Solution

```
#!/usr/bin/env python3

import argparse
import os
import random
import re
import sys
from dire import die


# --------------------------------------------------
def get_args():
    """parse arguments"""
    parser = argparse.ArgumentParser(
        description='Hangman',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('-l',
                        '--maxlen',
                        help='Max word length',
                        type=int,
                        default=10)

    parser.add_argument('-n',
                        '--minlen',
                        help='Min word length',
                        type=int,
                        default=5)

    parser.add_argument('-m',
                        '--misses',
                        help='Max number of misses',
                        type=int,
                        default=10)

    parser.add_argument('-s',
                        '--seed',
                        help='Random seed',
                        type=str,
                        default=None)

    parser.add_argument('-w',
                        '--wordlist',
```

```
                        help='Word list',
                        type=str,
                        default='/usr/share/dict/words')

    return parser.parse_args()


# --------------------------------------------------
def bail(msg):
    """Print a message to STDOUT and quit with no error"""
    print(msg)
    sys.exit(0)


# --------------------------------------------------
def main():
    """main"""
    args = get_args()
    max_len = args.maxlen
    min_len = args.minlen
    max_misses = args.misses
    wordlist = args.wordlist

    random.seed(args.seed)

    if not os.path.isfile(wordlist):
        die('--wordlist "{}" is not a file.'.format(wordlist))

    if min_len < 1:
        die('--minlen must be positive')

    if not 3 <= max_len <= 20:
        die('--maxlen should be between 3 and 20')

    if min_len > max_len:
        die('--minlen ({}) is greater than --maxlen ({})'.format(
            min_len, max_len))

    good_word = re.compile('^[a-z]{' + str(min_len) + ',' + str(max_len) +
                           '}$')
    words = [w for w in open(wordlist).read().split() if good_word.match(w)]

    word = random.choice(words)
    play({'word': word, 'max_misses': max_misses})
```

```
# ------------------------------------------------
def play(state):
    """Loop to play the game"""
    word = state.get('word') or ''

    if not word: die('No word!')

    guessed = state.get('guessed') or list('_' * len(word))
    prev_guesses = state.get('prev_guesses') or set()
    num_misses = state.get('num_misses') or 0
    max_misses = state.get('max_misses') or 0

    if ''.join(guessed) == word:
        msg = 'You win. You guessed "{}" with "{}" miss{}!'
        bail(msg.format(word, num_misses, '' if num_misses == 1 else 'es'))

    if num_misses >= max_misses:
        bail('You lose, loser!  The word was "{}."'.format(word))

    print('{} (Misses: {})'.format(' '.join(guessed), num_misses))
    new_guess = input('Your guess? ("?" for hint, "!" to quit) ').lower()

    if new_guess == '!':
        bail('Better luck next time, loser.')
    elif new_guess == '?':
        new_guess = random.choice([x for x in word if x not in guessed])
        num_misses += 1

    if not re.match('^[a-z]$', new_guess):
        print('"{}" is not a letter'.format(new_guess))
        num_misses += 1
    elif new_guess in prev_guesses:
        print('You already guessed that')
    elif new_guess in word:
        prev_guesses.add(new_guess)
        last_pos = 0
        while True:
            pos = word.find(new_guess, last_pos)
            if pos < 0:
                break
            elif pos >= 0:
                guessed[pos] = new_guess
                last_pos = pos + 1
    else:
        num_misses += 1
```

```
    play({
        'word': word,
        'guessed': guessed,
        'num_misses': num_misses,
        'prev_guesses': prev_guesses,
        'max_misses': max_misses
    })


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

Chapter 29

Markov Chain

Write a Python program called `markov.py` that takes one or more text files as positional arguments for training. Use the `-n|--num_words` argument (default `2`) to find clusters of words and the words that follow them, e.g., in "The Bustle" by Emily Dickinson:

```
The bustle in a house
The morning after death
Is solemnest of industries
Enacted upon earth,-

The sweeping up the heart,
And putting love away
We shall not want to use again
Until eternity.
```

If `n=1`, then we find that "The" can be followed by "bustle," "morning," and "sweeping. There is a"the" followed by "heart," but we're not going to alter the text in any way, including removing punctuation, so just use `str.split` on the text to break up the words.

To begin your text, choose a random word (or words) that begin with an uppercase letter. Then randomly select the next word in the chain, keep track of the floating window of the `-n` words, and keep selecting the next words until you have matched or exceeded the `-l|--length` argument of the number of characters (default 500) to emit at which point you should stop when you find a word that terminates with `.`, `!`, or `?`.

If you use `str.split` to get the words from the training text, you'll be removing any newlines from the text, so use a `-w|--text_width` argument (default 70) to introduce newlines in the output before the text exceeds that number of characters on the line.

Because of the use of randomness, you should include a `-s|--seed` argument (default `None`) to pass to `random.seed`.

Occassionally you may chose a path that terminates. That is, in selecting the next word, you may find there is no next-next word. In that case, just exit the program.

My implementation includes a `-d|--debug` option that will write a `.log` file so you can inspect my data structures and logic as you write your own version.

You should find many diverse texts and use them all as training files with varying numbers for `-n` to see how the texts will be mixed. The results are endlessly

entertaining.

```
$ ./markov.py
usage: markov.py [-h] [-l int] [-n int] [-s int] [-w int] [-d] FILE [FILE ...]
markov.py: error: the following arguments are required: FILE
$ ./markov.py -h
usage: markov.py [-h] [-l int] [-n int] [-s int] [-w int] [-d] FILE [FILE ...]

Markov Chain

positional arguments:
  FILE                  Training file(s)

optional arguments:
  -h, --help            show this help message and exit
  -l int, --length int  Output length (characters) (default: 500)
  -n int, --num_words int
                        Number of words (default: 2)
  -s int, --seed int    Random seed (default: None)
  -w int, --text_width int
                        Max number of characters per line (default: 70)
  -d, --debug           Debug to ".log" (default: False)
$ ./markov.py ../inputs/const.txt
Discoveries; To constitute Tribunals inferior to the seat of the
Senate and House of Representatives shall have been committed, which
district shall have the Qualifications requisite for Electors of the
sixth Year, so that one third may be imposed on such Importation, not
exceeding three on the Journal. Neither House, during the Time of
Adjournment, he may require it. No Bill of Attainder or ex post facto
Law shall be established by Law: but the Party convicted shall
nevertheless be liable and subject to their Consideration such
Measures as he shall nominate, and by and with the Advice and Consent
of the government of the United States under this Constitution, or,
on the List the said Office, the same State claiming Lands under
Grants of different States; between Citizens of each shall constitute
a Quorum to do Business; but a smaller number may adjourn from day to
day, and may be included within this Union, according to their
Consideration such Measures as he shall nominate, and by and with the
Advice and Consent of the United States.
```

markov_chain Solution

```
#!/usr/bin/env python3
"""Markov Chain"""

import argparse
import logging
import os
import random
import string
import sys
from pprint import pprint as pp
from collections import defaultdict


# --------------------------------------------------
def get_args():
    """Get command-line arguments"""

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

    parser.add_argument('training',
                        metavar='FILE',
                        nargs='+',
                        type=argparse.FileType('r'),
                        help='Training file(s)')

    parser.add_argument('-l',
                        '--length',
                        help='Output length (characters)',
                        metavar='int',
                        type=int,
                        default=500)

    parser.add_argument('-n',
                        '--num_words',
                        help='Number of words',
                        metavar='int',
                        type=int,
                        default=2)

    parser.add_argument('-s',
                        '--seed',
```

```
                        help='Random seed',
                        metavar='int',
                        type=int,
                        default=None)

    parser.add_argument('-w',
                        '--text_width',
                        help='Max number of characters per line',
                        metavar='int',
                        type=int,
                        default=70)

    parser.add_argument('-d',
                        '--debug',
                        help='Debug to ".log"',
                        action='store_true')

    return parser.parse_args()


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

    args = get_args()
    num_words = args.num_words
    char_max = args.length
    text_width = args.text_width

    random.seed(args.seed)

    logging.basicConfig(
        filename='.log',
        filemode='w',
        level=logging.DEBUG if args.debug else logging.CRITICAL)

    all_words = defaultdict(list)
    for fh in args.training:
        words = fh.read().split()

        for i in range(0, len(words) - num_words):
            l = words[i:i + num_words + 1]
            all_words[tuple(l[:-1])].append(l[-1])

    logging.debug('all words = {}'.format(all_words))
```

```
    prev = ''
    while not prev:
        start = random.choice(
            list(
                filter(lambda w: w[0][0] in string.ascii_uppercase,
                       all_words.keys())))
        if all_words[start]:
            prev = start

    logging.debug('Starting with "{}"'.format(prev))

    p = ' '.join(prev)
    char_count = len(p)
    print(p, end=' ')
    line_width = char_count

    while True:
        if not prev in all_words: break

        new_word = random.choice(all_words[prev])
        new_len = len(new_word) + 1
        logging.debug('chose = "{}" from {}'.format(new_word, all_words[prev]))

        if line_width + new_len > text_width:
            print()
            line_width = new_len
        else:
            line_width += new_len

        char_count += new_len
        print(new_word, end=' ')
        if char_count >= char_max and new_word[-1] in '.!?': break
        prev = prev[1:] + (new_word, )

    logging.debug('Finished')
    print()


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

Chapter 30

Morse Encoder/Decoder

Write a Python program called `morse.py` that will encrypt/decrypt text to/from Morse code. The program should expect a single positional argument which is either the name of a file to read for the input or the character `-` to indicate reading from STDIN. The program should also take a `-c|--coding` option to indicate use of the `itu` or standard `morse` tables, `-o|--outfile` for writing the output (default STDOUT), and a `-d|--decode` flag to indicate that the action is to decode the input (the default is to encode it).

```
$ ./morse.py
usage: morse.py [-h] [-c str] [-o str] [-d] [-D] FILE
morse.py: error: the following arguments are required: FILE
$ ./morse.py -h
usage: morse.py [-h] [-c str] [-o str] [-d] [-D] FILE

Encode and decode text/Morse

positional arguments:
  FILE                  Input file or "-" for stdin

optional arguments:
  -h, --help            show this help message and exit
  -c str, --coding str  Coding version (default: itu)
  -o str, --outfile str
                        Output file (default: None)
  -d, --decode          Decode message from Morse to text (default: False)
  -D, --debug           Debug (default: False)
$ ./morse.py ../inputs/fox.txt
- .... .  --.- ..- .. -.-. -.-  -... .-. --- .-- -.  ..-. --- -..-  .--- ..- -- .--. ...  --
[cholla@~/work/python/playful_python/morse]$ ./morse.py ../inputs/fox.txt | ./morse.py -d -
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.
```

morse Solution

```
#!/usr/bin/env python3
"""Morse en/decoder"""

import argparse
import logging
import random
import re
import string
import sys


# --------------------------------------------------
def get_args():
    """Get command-line arguments"""

    parser = argparse.ArgumentParser(
        description='Encode and decode text/Morse',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('input',
                        metavar='FILE',
                        help='Input file or "-" for stdin')

    parser.add_argument('-c',
                        '--coding',
                        help='Coding version',
                        metavar='str',
                        type=str,
                        choices=['itu', 'morse'],
                        default='itu')

    parser.add_argument('-o',
                        '--outfile',
                        help='Output file',
                        metavar='str',
                        type=str,
                        default=None)

    parser.add_argument('-d',
                        '--decode',
                        help='Decode message from Morse to text',
                        action='store_true')
```

```
    parser.add_argument('-D', '--debug', help='Debug', action='store_true')

    return parser.parse_args()


# -----------------------------------------------
def encode_word(word, table):
    """Encode word using given table"""

    coded = []
    for char in word.upper():
        logging.debug(char)
        if char != ' ' and char in table:
            coded.append(table[char])

    encoded = ' '.join(coded)
    logging.debug('endoding "{}" to "{}"'.format(word, encoded))

    return encoded


# -----------------------------------------------
def decode_word(encoded, table):
    """Decode word using given table"""

    decoded = []
    for code in encoded.split(' '):
        if code in table:
            decoded.append(table[code])

    word = ''.join(decoded)
    logging.debug('dedoding "{}" to "{}"'.format(encoded, word))

    return word


# -----------------------------------------------
def test_encode_word():
    """Test Encoding"""

    assert encode_word('sos', ENCODE_ITU) == '... --- ...'
    assert encode_word('sos', ENCODE_MORSE) == '... .,. ...'


# -----------------------------------------------
def test_decode_word():
```

```
    """Test Decoding"""

    assert decode_word('... --- ...', DECODE_ITU) == 'SOS'
    assert decode_word('... .,. ...', DECODE_MORSE) == 'SOS'


# --------------------------------------------------
def test_roundtrip():
    """Test En/decoding"""

    random_str = lambda: ''.join(random.sample(string.ascii_lowercase, k=10))
    for _ in range(10):
        word = random_str()
        for encode_tbl, decode_tbl in [(ENCODE_ITU, DECODE_ITU),
                                       (ENCODE_MORSE, DECODE_MORSE)]:

            assert word.upper() == decode_word(encode_word(word, encode_tbl),
                                               decode_tbl)


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()
    action = 'decode' if args.decode else 'encode'
    output = open(args.outfile, 'wt') if args.outfile else sys.stdout
    source = sys.stdin if args.input == '-' else open(args.input)

    coding_table = ''
    if args.coding == 'itu':
        coding_table = ENCODE_ITU if action == 'encode' else DECODE_ITU
    else:
        coding_table = ENCODE_MORSE if action == 'encode' else DECODE_MORSE

    logging.basicConfig(
        filename='.log',
        filemode='w',
        level=logging.DEBUG if args.debug else logging.CRITICAL)

    word_split = r'\s+' if action == 'encode' else r'\s{2}'

    for line in source:
        for word in re.split(word_split, line):
            if action == 'encode':
                print(encode_word(word, coding_table), end='  ')
            else:
```

```
                print(decode_word(word, coding_table), end=' ')
        print()


# ------------------------------------------------
def invert_dict(d):
    """Invert a dictionary's key/value"""

    #return dict(map(lambda t: list(reversed(t)), d.items()))
    return dict([(v, k) for k, v in d.items()])


# ------------------------------------------------
# GLOBALS

ENCODE_ITU = {
    'A': '.-', 'B': '-...', 'C': '-.-.', 'D': '-..', 'E': '.', 'F': '..-.',
    'G': '--.', 'H': '....', 'I': '..', 'J': '.---', 'K': '-.-', 'L': '.-..',
    'M': '--', 'N': '-.', 'O': '---', 'P': '.--.', 'Q': '--.-', 'R': '.-.',
    'S': '...', 'T': '-', 'U': '..-', 'V': '...-', 'W': '.--', 'X': '-..-',
    'Y': '-.--', 'Z': '--..', '0': '-----', '1': '.----', '2': '..---', '3':
    '...--', '4': '....-', '5': '.....', '6': '-....', '7': '--...', '8':
    '---..', '9': '----.', '.': '.-.-.-', ',': '--..--', '?': '..--..', '!':
    '-.-.--', '&': '.-...', ';': '-.-.-.', ':': '---...', "'": '.----.', '/':
    '-..-.', '-': '-....-', '(': '-.--.', ')': '-.--.-',
}

ENCODE_MORSE = {
    'A': '.-', 'B': '-...', 'C': '..,.', 'D': '-..', 'E': '.', 'F': '.-.', 'G':
    '--.', 'H': '....', 'I': '..', 'J': '-.-.', 'K': '-.-', 'L': '+', 'M':
    '--', 'N': '-.', 'O': '.,.', 'P': '.....', 'Q': '..-.', 'R': '.,..', 'S':
    '...', 'T': '-', 'U': '..-', 'V': '...-', 'W': '.--', 'X': '.-..', 'Y':
    '..,..', 'Z': '...,.', '0': '+++++', '1': '.--.', '2': '..-..', '3':
    '...-.', '4': '....-', '5': '---', '6': '......', '7': '--..', '8':
    '-....', '9': '-..-', '.': '..--..', ',': '.-.-', '?': '-..-.', '!':
    '---.', '&': '.,...', ';': '...,..', ':': '-.-,.,.', "'": '..-.,.-..', '/':
    '..-,-', '-': '....,.-..', '(': '.....,.-..', ')': '.....,..,..',
}

DECODE_ITU = invert_dict(ENCODE_ITU)
DECODE_MORSE = invert_dict(ENCODE_MORSE)

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

Chapter 31

ROT13 (Rotate 13)

Write a Python program called `rot13.py` that will encrypt/decrypt input text by shifting the text by a given `-s|--shift` argument or will move each character halfway through the alphabet, e.g., "a" becomes "n," "b" becomes "o," etc. The text to rotate should be provided as a single positional argument to your program and can either be a text file, text on the command line, or `-` to indicate STDIN so that you can round-trip data through your program to ensure you are encrypting and decrypting properly.

Discussion

The way I approached the solution is to think of adding time. If it's 8 in the morning and I want to know the time in 6 hours on a 12-hour (not military/24-hour) clock, I need to think in terms of 12 when the clock rolls over from AM to PM. To do that, I need to know the remainder of dividing by 12, which is given by the modulus `%` operator:

```
>>> now = 8
>>> (now + 6) % 12
2
```

And 6 hours from 8AM is, indeed, 2PM.

Similarly if I want to know how many hours (in decimal) are a particular number of minutes, I need to mod by 60:

```
>>> minutes = 90
>>> int(minutes / 60) + (minutes % 60) / 60
1.5
>>> minutes = 204
>>> int(minutes / 60) + (minutes % 60) / 60
3.4
```

If you `import string`, you can see all the lower/uppercase letters

```
>>> import string
>>> string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
>>> string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

So I think about "rot13" like adding 13 (or some other shift interval) to the position of the letter in the list and modding by the length of the list to wrap

it around. If the shift is 13 and we are at "a" and want to know what the letter 13 way is, we can use `pos` to find "a" and add 13 to that:

```
>>> lcase = list(string.ascii_lowercase)
>>> lcase.index('a')
0
>>> lcase[lcase.index('a') + 13]
'n'
```

But if we want to know the value for something after the 13th letter in our list, we are in trouble!

```
>>> lcase[lcase.index('x') + 13]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

`%` to the rescue!

```
>>> lcase[(lcase.index('x') + 13) % len(lcase)]
'k'
```

It's not necessary in this algorithm to shift by any particular number. 13 is special because it's halfway through the alphabet, but we could shift by just 2 or 5 characters. If we want to round-trip our text, it's necessary to shift in the opposite direction on the second half of the trip, so be sure to use the negative value there!

Expected Behavior

```
$ ./rot13.py
usage: rot13.py [-h] [-s int] str
rot13.py: error: the following arguments are required: str
$ ./rot13.py -h
usage: rot13.py [-h] [-s int] str

Argparse Python script

positional arguments:
  str                   Input text, file, or "-" for STDIN

optional arguments:
  -h, --help            show this help message and exit
  -s int, --shift int   Shift arg (default: 0)
$ ./rot13.py AbCd
NoPq
$ ./rot13.py AbCd -s 2
```

```
CdEf
$ ./rot13.py fox.txt
Gur dhvpx oebja sbk whzcf bire gur ynml qbt.
$ ./rot13.py fox.txt | ./rot13.py -
The quick brown fox jumps over the lazy dog.
$ ./rot13.py -s 3 fox.txt | ./rot13.py -s -3 -
The quick brown fox jumps over the lazy dog.
```

rot13 Solution

```
#!/usr/bin/env python3

import argparse
import os
import re
import string
import sys


# --------------------------------------------------
def get_args():
    """get command-line arguments"""
    parser = argparse.ArgumentParser(
        description='ROT13 encryption',
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)

    parser.add_argument('text',
                        metavar='str',
                        help='Input text, file, or "-" for STDIN')

    parser.add_argument('-s',
                        '--shift',
                        help='Shift arg',
                        metavar='int',
                        type=int,
                        default=0)

    return parser.parse_args()


# --------------------------------------------------
def main():
    """Make a jazz noise here"""
    args = get_args()
    text = args.text

    if text == '-':
        text = sys.stdin.read()
    elif os.path.isfile(text):
        text = open(text).read()

    lcase = list(string.ascii_lowercase)
    ucase = list(string.ascii_uppercase)
```

```
    num_lcase = len(lcase)
    num_ucase = len(ucase)
    lcase_shift = args.shift or int(num_lcase / 2)
    ucase_shift = args.shift or int(num_ucase / 2)

    def rot13(char):
        if char in lcase:
            pos = lcase.index(char)
            rot = (pos + lcase_shift) % num_lcase
            return lcase[rot]
        elif char in ucase:
            pos = ucase.index(char)
            rot = (pos + ucase_shift) % num_ucase
            return ucase[rot]
        else:
            return char

    print(''.join(map(rot13, text)).rstrip())


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