# Part II

# Collections and Files

## 11 Strings

11.1 Boolean.

11.2 (a)
- i. `"i"`
- ii. `"age"`
- iii. `"im"`
- iv. `"iae"`
- v. `"rote"`
- vi. `"protein synth"`
- vii. `"protein image"`

(b)
- i. `word[3]`
- ii. `word[1:4]`
- iii. `phrase[:3]`
- iv. `phrase[-6:]` or `phrase[11:]`
- v. `phrase[8:13]`
- vi. `word + phrase[8:]`

11.3 (a) Get every other character from `s`.

(b) Reverse `s`.

11.4 (a) `s[:-1]`

(b) `s[:]`

11.5
```
if s[0] in "0123456789":
    return False
```

11.6 (a) `piglatin("spy")`

→ `firstvowelindex("spy")`

$i = 0, i = 1, i = 2$

returns $\boxed{2}$

returns `"y"+ "sp"+ "ay"`, which is $\boxed{\texttt{"yspay"}}$

(b) `piglatin("nix")`

→ `firstvowelindex("nix")`

$i = 0, i = 1$

returns $\boxed{1}$

returns `"ix"+ "n"+ "ay"`, which is $\boxed{\texttt{"ixnay"}}$

(c) The variable `i` is an accumulator. It allows looking at each successive character in the word until a vowel is reached.

(d) If `i` is 0, then the function returns, so the only way the next line of code can be reached is if `i` is not 0.

11.7 Option.

11.8
```
def piglatin(word):
    i = firstvowelindex(word)
    if i == 0 or i == -1:
        return word + "-yay"
    return word[i:] + "-" + word[:i] + "ay"
```

11.9

```
def firstvowelindex(word):
    i = 0
    if word[0] == "y":  # initial y is consonant
        i += 1
    while i < len(word) and word[i] not in "aeiouy":
        i += 1
    return i

def piglatin(word):
    i = firstvowelindex(word)
    if i == 0 or i == -1:
        return word + "yay"
    return word[i:] + word[:i] + "ay"
```

11.10

```
def firstvowelindex(word):
    i = 0
    while i < len(word) and word[i] not in "aeiouy":
        i += 1
    if i == len(word):  # no vowel found
        return -1
    return i

def piglatin(word):
    i = firstvowelindex(word)
    if i == 0 or i == -1:
        return word + "yay"
    return word[i:] + word[:i] + "ay"
```

11.11

```
# username.py

def username(first, last):
    return last[:12] + first[0] + '1'

def main():
    first = input('Enter first name: ')
    last = input('Enter last name: ')
    print(username(first, last))

main()
```

## 12 Building Strings

12.1 (a) `complement()`: `result` accumulates on the right with `+=`. `random_dna()`: `fragment` accumulates on the right with `+=`.

(b) The `[::-1]` in line 19.

(c)

```
def reverse(s):
    return s[::-1]

def reversecomp(dna):
    return reverse(complement(dna))
```

(d) Chooses a random character from the string `"ACGT"`.

12.2 (a) If `s = "a"` and `t = "b"` then `s = s + t` gives `s = "ab"`, whereas `s = t + s` gives `s = "ba"`.

(b) If `s` is the same as `t`, then concatenation on the right will be the same as on the left.

12.3
```
def reverse(s):
    result = ""
    for c in s:
        result = c + result
    return result
```

12.4
```
def is_dna(s):
    for c in s:
        if c not in "ACGT":
            return False
    return True
```

12.5
```
def random_rna(length=30):
    fragment = ""
    for j in range(length):
        fragment += choice("ACGU")
    return fragment
```

12.6
```
def is_rna(s):
    for c in s:
        if c not in "ACGU":
            return False
    return True
```

12.7
```
def transcription(dna):
    result = ""
    for c in dna:
        if c == "T":
            result += "U"
        else:
            result += c
    return result
```

12.8
```
def ispalindrome(dna):
    return dna == reversecomp(dna)
```

12.9
```
def countbases(dna):
    countA = 0
    countC = 0
    countG = 0
    countT = 0
    for c in dna:
        if c == "A":
            countA += 1
        elif c == "C":
            countC += 1
        elif c == "G":
            countG += 1
        elif c == "T":
            countT += 1
    return countA, countC, countG, countT
```

12.10

```
def dec_to_bin(n):
    # Convert nonnegative integer n to a binary string.
    result = ""
    while n > 0:
        result = str(n % 2) + result
        n //= 2
    return result
```

12.11

```
def bin_to_dec(s):
    # Convert binary string to decimal integer.
    result = 0
    for c in s:
        result = 2 * result + int(c)
    return result
```

## Project: ISBN Check Digits

```
# isbn.py

def check10(s):
    total = 0
    i = 1
    for c in s:
        total += i * int(c)
        i += 1
    check = total % 11
    return str(check) if check < 10 else "X"

def check13(s):
    total = 0
    i = 0
    for c in s:
        multiplier = 1 if i % 2 == 0 else 3
        total += multiplier * int(c)
        i += 1
    return (10 - (total % 10)) % 10

def main():
    s = input("Enter a 9-digit ISBN without hyphens: ")
    print("The check digit is", check10(s))
    s = input("Enter a 12-digit ISBN without hyphens: ")
    print("The check digit is", check13(s))

main()
```

## 13 Computer Memory: Text

13.1 0–127.

13.2 0–255.

13.3 Lowercase letter ASCII codes are all 32 more than the corresponding uppercase letter. This allows changing case by changing only one bit.

13.4 Curly brackets have ASCII codes 123 and 125, at the end of the printable seven-bit characters. They would come after alphabetic entries if all symbols were not listed at the front of the index.

13.5
```
# ascii.py

def main():
    for n in range(32, 127):
        print(n, chr(n))

main()
```

13.6
```
def encode(msg):
    result = ""
    for c in msg:
        result += str(ord(c)) + " "
    return result[:-1]
```

# 14 Lists

14.1 The dot between an object and a method name followed by parentheses.

14.2 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ ~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~ . . .

14.3
```
k = 2
     for j in range(4, 11, 2)::
          j = 4 removed
          j = 6 removed
          j = 8 removed
          j = 10 removed
k = 3
     for j in range(6, 11, 3)::
          j = 6 (not in primes)
          j = 9 removed
```

14.4
- (a) Method call: `primes.remove(j)`. The object is `primes`, the name of the method is `.remove()`, and the one argument is `j`.
- (b) The ranges need to include `n`, so the second argument to `range()` must be `n + 1`.
- (c) No, it does not work, because the list `primes` is changing inside the loop.
- (d) The represents crossing out all of the multiples of `k`, by taking steps of size `k`.
- (e) Otherwise, the `.remove()` method will raise an exception when `j` is not in the list.

14.5 Because if $n$ is not prime, at least one of its factors must be less than or equal to $\sqrt{n}$.

```
def sieve(n):
    primes = list(range(2, n + 1))
    for k in range(2, int(ceil(sqrt(n)))):
        if k in primes:
            for j in range(2 * k, n + 1, k):
                if j in primes:
                    primes.remove(j)
    return primes
```

14.6 Because multiples of $j$ less than $j^2$ have already been crossed off (they are $2j$, $3j$, $\ldots (j-1)j$). Combined with the previous exercise, this gives:

```
def sieve(n):
    primes = list(range(2, n + 1))
    for k in range(2, int(ceil(sqrt(n)))):
        if k in primes:
            for j in range(k * k, n + 1, k):
                if j in primes:
                    primes.remove(j)
    return primes
```

14.7

```
def randints(n, a, b):
    result = []
    for i in range(n):
        result.append(randint(a, b))
    return result
```

14.8

```
def randintsdistinct(n, a, b):
    result = []
    while len(result) < n:
        candidate = randint(a, b)
        if candidate not in result:
            result.append(candidate)
    return result
```

14.9

```
def randfloats(n, a, b):
    result = []
    for i in range(n):
        result.append(uniform(a, b))
    return result
```

14.10

```
def mysum(items):
    total = 0
    for x in items:
        total += x
    return total
```

14.11 Python `min()` throws an exception on empty lists, so it is reasonable to assume the list is nonempty.

```
def mymin(items):
    smallest = items[0]
    for x in items[1:]:
        if x < smallest:
            smallest = x
    return smallest
```

14.12 Python `max()` throws an exception on empty lists, so it is reasonable to assume the list is nonempty.

```
def mymax(items):
    largest = items[0]
    for x in items[1:]:
        if x > largest:
            largest = x
    return largest
```

14.13
```
def mean(items):
    return sum(items)/len(items)
```

14.14
```
def median(items):
    copy = items[:]
    copy.sort()
    n = len(copy)
    if n % 2 == 1:
        return copy[n // 2]
    else:
        return (copy[n // 2 - 1] + copy[n // 2]) / 2
```

14.15
```
def evens(items):
    result = []
    for x in items:
        if x % 2 == 0:
            result.append(x)
    return result
```

14.16
```
def odds(items):
    result = []
    for x in items:
        if x % 2 == 1:
            result.append(x)
    return result
```

14.17
```
def block(names, blocked):
    result = []
    for name in names:
        if name not in blocked:
            result.append(name)
    return result
```

14.18
```
def mychoice(items):
    return items[randrange(len(items))]
```

14.19
```
def delrand(items):
    return items.pop(randrange(len(items)))
```

14.20

```
# countflips.py

from random import choice

def flip():
    return choice(["heads", "tails"])

def main():
    n = int(input("Enter the number of trials: "))
    heads = 0
    tails = 0
    for i in range(n):
        if flip() == "heads":
            heads += 1
        else:
            tails += 1
    print("Heads:", heads)
    print("Tails:", tails)

main()
```

14.21

```
# countrolls.py

from random import randint

def roll():
    # Imitate rolling a six-sided die.
    return randint(1, 6)

def main():
    n = input("Enter the number of times to roll: ")
    count = [0] * 7
    for i in range(n):
        result = roll()
        count[result] += 1
    print count[1:]

main()
```

14.22

```
def madlib():
    verb = ["runs", "swims", "kicks", "jumps", "catapults", "fries"]
    noun = ["lamp", "bear", "TJ", "flower", "sandbag", "Abraham Lincoln"]
    adjective = ["hairy", "slimy", "fuzzy", "cold", "orange", "smelly"]
    adverb = ["quickly", "swiftly", "jokingly", "merrily", "softly"]
    sentence = "The " + choice(adjective) + " " + choice(noun) + " " + \
               choice(verb) + " " + choice(adverb) + "."
    return sentence
```

14.23

```
def swap(items, i, j):
    items[i], items[j] = items[j], items[i]
```

```
14.24 def myreverse(items):
          n = len(items)
          for i in range(n // 2):
              swap(items, i, n - i - 1)

14.25 def myshuffle(items):
          n = len(items)
          for i in range(n - 1):
              j = randint(i, n - 1)
              swap(items, i, j)

14.26 def issorted(items):
          for i in range(len(items) - 1):
              if items[i + 1] < items[i]:
                  return False
          return True

14.27 def minindex(items, start):
          smallest = items[start]
          smallestindex = start
          for k in range(start + 1, len(items)):
              if items[k] < smallest:
                  smallest = items[k]
                  smallestindex = k
          return smallestindex

      def mysort(items):
          n = len(items)
          for i in range(n):
              j = minindex(items, i)
              swap(items, i, j)

14.28 def primefactors(n):
          result = []
          for i in range(2, n + 1):
              if isprime(i) and divides(i, n):
                  result.append(i)
          return result

14.29 def primefactorization(n):
          result = []
          while n > 1:
              k = smallestdivisor(n)
              result.append(k)
              n /= k
          return result

14.30 def divisors(n):
          result = []
          for k in range(1, n):
              if n % k == 0:
                  result.append(k)
          return result
```

14.31
```
def isperfect(n):
    return n > 0 and sum(divisors(n)) == n
```

14.32
```
def fib(n):
    result = [1, 1]
    for i in range(2, n):
        nextelt = result[i - 1] + result[i - 2]
        result.append(nextelt)
    return result
```

## Project: Program Performance

1. Other software, including the operating system, will likely run in between the consecutive calls to `time.clock()`.

2.
```
def primelist(n):
    primes = list(range(2, n + 1))
    for k in range(2, n + 1):
        if not isprime(k):
            primes.remove(k)
    return primes
```

3.
```
def sieve2(n):
    primes = [True] * (n + 1)
    primes[0] = False
    primes[1] = False
    for k in range(2, int(ceil(sqrt(n)))):
        if primes[k]:
            for j in range(k * k, n + 1, k):
                primes[j] = False
    return primes
```

4.
```
def test(plist, blist, n):
    for i in range(2, n + 1):
        if blist[i] and i not in plist:
            return False
        if not blist[i] and i in plist:
            return False
    return True
```

5. 
```
def main():
    n = 20000
    print("Timing for n =", n)
    start = time.clock()
    p = sieve(n)
    stop = time.clock()
    q = sieve2(n)
    stop2 = time.clock()
    r = primelist(n)
    stop3 = time.clock()
    print("Sieve:", stop - start)
    print("Sieve2:", stop2 - stop)
    print("Primelist:", stop3 - stop2)
    print(test(p, q, n))
    print(p == r)
```

## Project: Heat Diffusion

```
# diffusion.py

def init(left, right, inittemp, n):
    temps = [inittemp] * (n + 1)
    temps[0] = left
    temps[n] = right
    return temps

def average(t1, t2):
    return (t1 + t2) / 2.0

def main():
    length = 5
    n = 10
    duration = 1
    temps = init(30, 50, 0, n)
    h = length / n
    dt = 0.5 * h ** 2
    newtemps = temps[:]
    print(temps)
    while duration > 0:
        for i in range(1, n):
            newtemps[i] = average(temps[i-1], temps[i+1])
        temps = newtemps[:]
        print(temps)
        duration -= dt

main()
```

# 15 Files

15.1 (a) In the `matches()` function, there is a list accumulator. The easiest way to spot its type is the starting value.

(b) It is a loop over a list. The `wordlist()` function returns a list from the call to `.split()`.

(c)

```
with open(fname) as f:
    text = f.read()
return text.split()
```

15.2 (a) `["e", "f", "i", "l"]`, `["g", "i", "n", "r", "s", "t"]`, `["h", "n", "o", "p", "t", "y"]`

(b) No. The string `word` does not have a `.sort()` method.

(c) No. The list `.sort()` method returns `None`.

(d) No. The list `.sort()` method returns `None`.

(e) No. The list `.sort()` method returns `None`.

15.3

```
# printfile.py

def version1(fname):
    with open(fname) as f:
        print(f.read())

def version2(fname):
    with open(fname) as f:
        lines = f.readlines()
    for line in lines:
        print(line)

def version3(fname):
    with open(fname) as f:
        line = f.readline()
        while line != "":
            print(line)
            line = f.readline()

def version4(fname):
    with open(fname) as f:
        for line in f:
            print(line)

def main():
    fname = input('Enter a file name: ')
    version1(fname)
    version2(fname)
    version3(fname)
    version4(fname)

main()
```

All but the first appear double-spaced because the line-based methods include a newline character at the end of each line.

15.4 (a) Replace each `print(line)` with `print(line, end="")`.

(b) Replace each `print(line)` with `print(line[:-1])`.

15.5

```
# reverse.py

def lines(fname):
    with open(fname) as f:
        return f.readlines()

def main():
    fname = input('Enter a file name: ')
    for line in lines(fname)[::-1]:
        print(line, end="")

main()
```

15.6

```
# wotd.py

from random import choice

def wordlist(fname):
    with open(fname) as f:
        return f.read().split()

def chooseword():
    return choice(wordlist("dictionary.txt"))

def main():
    print(chooseword())

main()
```

15.7 This is easiest with the string `.count()` method (as shown below) and should move to Chapter 16.

```
# wc.py

def wc(fname):
    with open(fname) as f:
        text = f.read()
    linecount = text.count("\n")
    wordcount = len(text.split())
    charcount = len(text)
    return linecount, wordcount, charcount

def main():
    fname = input("Enter the file name: ")
    lines, words, chars = wc(fname)
    print("Lines:", lines)
    print("Words:", words)
    print("Characters:", chars)

main()
```

15.8
```
# cp.py

def cp(source, dest):
    with open(source) as f:
        text = f.read()
    with open(dest, "w") as f:
        f.write(text)

def main():
    source = input("Enter the file to copy: ")
    dest = input("Destination file: ")
    cp(source, dest)
    print("File copied.")

main()
```

15.9
```
# avgword.py

def wordlist(fname):
    with open(fname) as f:
        return f.read().split()

def avgword(fname):
    words = wordlist(fname)
    total = 0
    for word in words:
        total += len(word)
    return total / len(words)

def main():
    fname = input("Enter the file name: ")
    print("The average word length is", avgword(fname))

main()
```

15.10
```
# linenum.py
# Add line numbers to a file.

def getlines(fname):
    with open(fname) as f:
        return f.readlines()

def printwithnumbers(lines):
    num = 1
    for line in lines:
        print(num, line, end="")
        num += 1

def main():
    fname = input("Enter the name of a file: ")
    printwithnumbers(getlines(fname))

main()
```

15.11
```
def decode(msg):
    result = ""
    for code in msg.split():
        result += chr(int(code))
    return result
```

15.12
```
def main():
    maxword = ""
    maxcount = 0
    maxanagrams = []
    for word in wordlist("dictionary.txt"):
        anagrams = matches(word)
        if len(anagrams) > maxcount:
            maxword = word
            maxcount = len(anagrams)
            maxanagrams = anagrams
            print(maxword, maxcount)
    print("The word with the most anagrams is", maxword)
    print("Its anagrams are:", maxanagrams)
```

15.13
```
# wordvalue.py

def score(word):
    result = 0
    for c in word:
        result += ord(c) - ord("a") + 1
    return result

def wordlist(fname):
    with open(fname) as f:
        return f.read().split()

def main():
    w = wordlist("dictionary.txt")
    maxword = ""
    maxscore = 0
    for word in wordlist("dictionary.txt"):
        if score(word) > maxscore:
            maxword = word
            maxscore = score(word)
            print("New candidate:", maxword, maxscore)
    print("Final:", maxword, maxscore)

main()
```

# 16 String Methods

16.1 (a) There is a string accumulator in the `process()` function. It begins with a string and uses two concatenations inside the **`while`** loop.

(b) The **`return`** ensures that the following code is only executed if the condition was false.

(c)

```
from urlib.request import urlopen

with urlopen(url) as f:
```

16.2 (a) See part (b). We are attempting to keep the page at this link consistent and updated with current content.

(b) The web page may have changed by the time you run this program. Earlier code examples did not depend on an external resource like a web page.

(c) Meal information sometimes has extra `<p>` tags that the `strip()` is designed to remove.

16.3 (a) In addition to the content, it is helpful to know the index so that future searches can continue from that point.

(b) It should return whatever is between the end of the `opentag` and the beginning of the `closetag`.

16.4 One solution is to modify `getpage()`:

```
def getpage(url):
    with urllib.request.urlopen(url) as f:
        return str(f.read()).lower()
```

16.5 For example:

```
if "steak" in meal.lower():
    result += "  *** " + meal.strip("<>p") + " ***\n"
else:
    result += "  " + meal.strip("<>p") + "\n"
```

16.6 Option.

16.7 Option.

16.8

```
# weather.py

import urllib.request
import random
import string

URL = "http://forecast.weather.gov/zipcity.php?inputstring="

def getpage(url):
    with urllib.request.urlopen(url) as f:
        return str(f.read())

def gettag(page, tag, cls=None, start=0):
    opentag = '<' + tag
    if cls is not None:
        opentag += ' class="' + cls + '"'
    closetag = "</" + tag + ">"
    startopen = page.find(opentag, start)
    if startopen == -1:
        return None, -1
    endopen = page.find(">", startopen)
    end = page.find(closetag, endopen)
    return page[endopen + 1:end], end

def process(page):
    conditions, i = gettag(page, "td", "big")
    if conditions is None:
        return "No Data"
    enddesc = conditions.find("<br>")
    description = conditions[:enddesc]
    temp = conditions[enddesc + 4:]
    temp = temp.replace("<br>", " ").replace(" &deg;", " deg ")
    return description + ":" + temp

def main():
    zipcode = input("Enter zip code for current conditions: ")
    page = getpage(URL + zipcode)
    print(process(page))

def randzip():
    result = ""
    for i in range(5):
        result += random.choice(string.digits)
    return result

def testrandom():
    for zipcode in [randzip() for i in range(10)]:
        print(zipcode, process(getpage(URL + zipcode)))

main()
```

16.9

```
def removepunc(s):
    result = ""
    for c in s:
        if c not in string.punctuation:
            result += c
    return result
```

16.10

```
def removepunc(s):
    return s.translate(s.maketrans("", "", string.punctuation))
```

16.11

```
def alphaonly(s):
    result = ""
    for c in s:
        if c.isalpha():
            result += c
    return result
```

16.12

```
def mycapitalize(s):
    return s[0].upper() + s[1:].lower()
```

16.13 A general solution to this problem is difficult, but for simple strings that have a single space between each word, this is what had been intended:

```
def mytitle(s):
    result = ""
    for word in s.split():
        result += word.capitalize() + " "
    return result
```

16.14

```
def is_dna(s):
    for c in s.upper():
        if c not in "ACGT":
            return False
    return True
```

16.15

```
def is_rna(s):
    for c in s.upper():
        if c not in "ACGU":
            return False
    return True
```

16.16

```
def transcription(dna):
    return dna.replace("T", "U")
```

16.17

```
def countbases(dna):
    return dna.count("A"), dna.count("C"), dna.count("G"), dna.count("T")
```

16.18

```
def complement(dna):
    NUCLEOTIDES = "ACGT"
    return dna.translate(dna.maketrans(NUCLEOTIDES, NUCLEOTIDES[::-1]))
```

16.19

```
def acronym(phrase):
    acronym = ""
    for word in phrase.split():
        acronym += word[0].upper()
    return acronym
```

16.20

```
def isidentifier(s):
    if keyword.iskeyword(s):
        return False
    for c in s:
        if not c.isalpha() and not c.isdigit() and c != "_":
            return False
    if s[0].isdigit():
        return False
    return True
```

16.21

```
def reverse(s):
    return s[::-1]

def alphaonly(s):
    result = ""
    for c in s:
        if c.isalpha():
            result += c
    return result

def ispalindrome(s):
    s = alphaonly(s).lower()
    return s == reverse(s)
```

16.22

```
def longestpalindrome():
    bestsofar = ""
    for word in wordlist("dictionary.txt"):
        if ispalindrome(word) and len(word) > len(bestsofar):
            bestsofar = word
    return bestsofar
```

16.23 It is sufficient to modify the signature:

```
def signature(word):
    chars = list(word.lower())
    chars.sort()
    return chars
```

16.24

```
# printfile.py

def version2(fname):
    with open(fname) as f:
        for line in f:
            print(line.rstrip())

def version3(fname):
    with open(fname) as f:
        lines = f.readlines()
    for line in lines:
        print(line.rstrip())

def version4(fname):
    with open(fname) as f:
        line = f.readline()
        while line != "":
            print(line.rstrip())
            line = f.readline()

def main():
    fname = input('Enter a file name: ')
    version2(fname)
    version3(fname)
    version4(fname)

main()
```

16.25

```
# avgword.py

import string

def removepunc(s):
    return s.translate(s.maketrans("", "", string.punctuation))

def wordlist(fname):
    with open(fname) as f:
        return f.read().split()

def avgword(fname):
    words = wordlist(fname)
    total = 0
    for word in words:
        total += len(removepunc(word))
    return total / len(words)

def main():
    fname = input("Enter the file name: ")
    print("The average word length is", avgword(fname))

main()
```

16.26

```
# caesar.py

def encodeletter(c, n):
    # maintain case of c
    base = ord("A") if c.isupper() else ord("a")
    orig = ord(c) - base
    coded = (orig + n) % 26
    return chr(coded + base)

def encode(s, n):
    result = ""
    for c in s:
        if c.isalpha():
            result += encodeletter(c, n)
        else:
            result += c
    return result

def decode(s, n):
    return encode(s, 26-n)

def main():
    s = input("Enter a message: ")
    n = int(input("Enter n: "))
    coded = encode(s, n)
    decoded = decode(coded, n)
    print(coded, decoded, s == decoded)

main()
```

16.27

```
# piglatinfile.py

def firstvowelindex(word):
    i = 0
    if word[0] == "y":  # initial y is consonant
        i += 1
    while i < len(word) and word[i] not in "aeiouy":
        i += 1
    if i == len(word):  # no vowel found
        return -1
    return i

def piglatin(word):
    i = firstvowelindex(word)
    if i == 0 or i == -1:
        return word + "-yay"
    return word[i:] + "-" + word[:i] + "ay"

def gettext(fname):
    with open(fname) as f:
        return f.read()

def getpunc(word):
    i = 0
    while i < len(word) and not word[i].isalpha():
        i += 1
    j = len(word) - 1
    while j >= 0 and not word[j].isalpha():
        j -= 1
    return word[:i], word[i:j+1], word[j+1:]

def translate(text):
    result = ""
    words = text.split()
    for word in words:
        pf, w, pb = getpunc(word)
        if len(w) > 0:
            result += pf + piglatin(w) + pb + " "
        else:
            result += pf + pb
    return result

def main():
    fname = input("Enter a file name: ")
    print(translate(gettext(fname)))

main()
```

16.28

```
# spellcheck.py

import string

def removepunc(s):
    return s.translate(s.maketrans("", "", string.punctuation))

def wordlist(fname):
    with open(fname) as f:
        return f.read().lower().split()

def check(fname):
    result = []
    validwords = wordlist("dictionary.txt")
    for word in wordlist(fname):
        w = removepunc(word)
        if w not in validwords:
            result.append(w)
    return result

def main():
    fname = input("Enter a file to spellcheck: ")
    print("These words are not in the dictionary:")
    for word in check(fname):
        print(word)

main()
```

## Project: File Compression

```
# compress.py

MINREPS = 4
ESC = "~"

def double_esc(c):
    if c == ESC:
        return ESC + ESC
    return c

def encode(current, consec):
    if consec >= MINREPS:
        return ESC + str(consec) + current
    return double_esc(current) * consec

def compress(text):
    current = text[0]
    consec = 1
    result = ""
    for c in text[1:]:
        if c == current and c != ESC:
            consec += 1
        else:
            result += encode(current, consec)
            consec = 1
            current = c
    return result + encode(current, consec)

def int_at_start(s):
    # return positive integer at beginning of string
    #  with index of next character
    i = 0
    while s[i].isdigit():
        i += 1
    return int(s[:i]), i

def decompress(text):
    result = ""
    i = 0
    while i < len(text):
        if text[i] == ESC:
            if text[i + 1] == ESC:
                result += ESC
                i += 2
            else:
                n, skip = int_at_start(text[i + 1:])
                c = text[i + 1 + skip]
                result += c * n
                i += skip + 2
        else:
            result += text[i]
            i += 1
    return result

def main():
    with open("compress.py") as f:
        orig = f.read()
    coded = compress(orig)
    decoded = decompress(coded)
    print(coded)
    print(decoded)
    print("Original length:", len(orig))
    print("Coded length:", len(coded))
    print("Decoded length:", len(decoded))
    for i in range(len(orig)):
        if orig[i] != decoded[i]:
            print(i, orig[i], decoded[i])
    print(orig == decoded)

main()
```

## 17 Mutable and Immutable Objects

17.1 (a) `STRING METHODS WITH STRINGS`

(b) `Monty Python`

(c) Error: cannot assign to a slice (or index) of a string.

17.2 `s = s[:2] + "-" + s[3:]`

17.3

```
def signature(word):
    chars = list(word)
    chars.sort()
    return "".join(chars)
```

The program will work; the rest of the code does not depend on the precise signature that is used.

17.4

```
def ownsignature():
    result = []
    for word in wordlist("dictionary.txt"):
        if signature(word) == word:
            result.append(word)
    return result
```

These are words whose letters appear in alphabetical order.

17.5

```
def anagram(word1, word2):
    list1 = list(word1)
    list2 = list(word2)
    list1.sort()
    list2.sort()
    return list1 == list2
```

17.6

```
def jumble(word):
    chars = list(word)
    shuffle(chars)
    return "".join(chars)
```

17.7
```
# jumbleguess.py

from random import choice, shuffle

def getword():
    with open("dictionary.txt") as f:
        return choice(f.read().split())

def jumble(word):
    chars = list(word)
    shuffle(chars)
    return "".join(chars)

def main():
    secret = getword()
    jumbled = jumble(secret)
    print("The jumbled word is....", jumbled)
    word = input("Guess the word: ")
    count = 1
    while word != secret and word != "bye":
        print("Sorry.  That is not it.")
        word = input("Guess the word (bye to quit): ")
        count += 1
    if count == 1 and word != "bye":
        print("Great work!  Only 1 guess.")
    elif word != "bye":
        print("Nice work.  It took you", count, "guesses.")
    else:
        print("Ok...  The word was", secret)

main()
```

17.8
```
def mutate(dna):
    loc = randrange(len(dna))
    current = dna[loc]
    newbase = choice("ACGT".replace(current, ""))
    return dna[:loc] + newbase + dna[loc + 1:]
```

17.9
```
# mutate.py

from random import randrange, choice

def countbases(dna):
    return dna.count("A"), dna.count("C"), dna.count("G"), dna.count("T")

def mutate(dna):
    loc = randrange(len(dna))
    current = dna[loc]
    newbase = choice("ACGT".replace(current, ""))
    return dna[:loc] + newbase + dna[loc + 1:]

def main():
    dna = "A" * 50
    print(dna)
    for i in range(100):
        dna = mutate(dna)
        print(dna)
    print(countbases(dna))

main()
```

## Project: Hangman

```
# hangman.py

from random import choice

NUMGUESSES = 6

def randomword():
    with open("dictionary.txt") as f:
        return choice(f.read().split())

def template(secret, guesses):
    result = ""
    for c in secret:
        if c in guesses:
            result += c
        else:
            result += "-"
    return result

def nextguess(guesses):
    letter = input("Guess a letter: ")
    while letter in guesses:
        print("You already guessed that.")
        letter = input("Guess a letter: ")
    return letter

def addguess(guesses, letter):
    guesses += letter
    chars = list(guesses)
    chars.sort()
    return "".join(chars)

def main():
    secret = randomword()
    guesses = ""
    count = NUMGUESSES
    while count > 0 and secret != template(secret, guesses):
        print(template(secret, guesses))
        print("Letters guessed: ", guesses)
        print("You have", count, "guesses remaining.")
        letter = nextguess(guesses)
        guesses = addguess(guesses, letter)
        if letter in secret:
            print("Good guess!")
        else:
            print("Sorry, there are no " + letter + "'s.")
            count -= 1
    print("The word was", secret)

main()
```

# 18 Dictionaries

18.1 Integer, float, string, boolean. The key type must be immutable.

18.2 Any type may be used as a value.

18.3 `{"Title1":rating1, "Title2":rating2, "Title3":rating3, "Title4":rating4}`

18.4
- (a) The purpose is to remove punctuation before counting frequency.
- (b) An exception is thrown if you attempt to access an entry for a key that is not in the dictionary.
- (c) The entries will be displayed in what appears to be a random order.
- (d) List of strings that is returned by the `split()` on line 9.

18.5
```python
def removepunc(s):
    return s.translate(s.maketrans("", "", string.punctuation))

def getwords(fname):
    with open(fname) as f:
        s = f.read().lower()
    return removepunc(s).split()
```

18.6
```python
def frequency(words):
    d = {}
    for word in words:
        d[word] = d.get(word, 0) + 1
    return d
```

18.7
```python
import collections

def frequency(words):
    d = collections.defaultdict(int)
    for word in words:
        d[word] += 1
    return d
```

18.8
```python
def removepunc(s):
    # keep hyphens and apostrophes
    punc = string.punctuation.replace("-", "")
    punc = punc.replace("'", "")
    s = s.translate(s.maketrans("", "", punc))
    # but replace dashes with spaces to separate words
    return s.replace("--", " ")
```

18.9
```python
def gettext(fname):
    with open(fname) as f:
        return f.read().lower()

def letterfreq(text):
    d = {}
    for c in text:
        if c.isalpha():
            d[c] = d.get(c, 0) + 1
    return d
```

18.10
```
# dict.py

def getinput():
    print("\t[a]dd or change a definition")
    print("\t[l]ookup a word")
    print("\t[q]uit")
    request = input("Your choice: ").lower()
    while request not in "alq":
        request = input("Your choice: ")
    return request

def process(request, d):
    if request == "a":
        word = input("Word: ").lower()
        defn = input("Definition: ")
        d[word] = defn
    elif request == "l":
        word = input("Word to look up: ").lower()
        if word in d:
            print("Definition:", d[word])
        else:
            print("Not in dictionary")
    return

def main():
    d = {}
    print("Welcome to PyDict")
    request = getinput()
    while request != "q":
        process(request, d)
        request = getinput()
    print("Thanks!")

main()
```

18.11

```
# findanagrams.py

def signature(word):
    chars = list(word)
    chars.sort()
    return "".join(chars)

def wordlist(fname):
    with open(fname) as f:
        return f.read().split()

def matches(word, wordlist):
    sign = signature(word)
    result = []
    for other in wordlist:
        if signature(other) == sign:
            result.append(other)
    return result

def filldict(words):
    d = {}
    for word in words:
        sign = signature(word)
        d[sign] = d.get(sign, 0) + 1
    return d

def findkey(d, value):
    for key in d:
        if d[key] == value:
            return key
    return None

def mostanagrams(words):
    d = filldict(words)
    m = max(d.values())
    return findkey(d, m)

def main():
    words = wordlist("dictionary.txt")
    sign = mostanagrams(words)
    print("Signature with most anagrams:", sign)
    print("The anagrams:", matches(sign, words))

main()
```

## Project: ELIZA

```
# eliza.py

from random import choice
import string

def default(name):
    generic = ["I’m not sure I understand you.\nCould you say more?",
        "Really?",
        "No way.  I never thought of that.",
        "You are very perceptive, " + name + "."]
    return choice(generic)

def clean(text):
    text = text.translate(text.maketrans("", "", string.punctuation))
    return text.lower()

def respond(text, name):
    keywords = {
        "family":"Families can be very strange.\nTell me more about yours.",
        "work":"Do you enjoy your work, " + name + "?",
        "you":"We’re not here to talk about me.\nLet’s talk about you.",
        "money":"It’s hard when finances are tight.\nWhat do you do for a living?"}

    for word in text.split():
        word = clean(word)
        if word in keywords:
            return keywords[word]
    return default(name)

def finished(text):
    text = clean(text)
    return text.startswith("bye") or text.startswith("quit") or \
            text.startswith("exit")

def main():
    print("Hello.  My name is Eliza.\nWhat is your name?")
    name = input()
    print("How can I help you, " + name + "?")
    text = input()
    while not finished(text):
        print(respond(text, name))
        text = input()
    print("I’m sorry you have to leave.  I hope we speak again soon.")

main()
```

## Project: Shannon Entropy

```
# entropy.py

import collections
from math import log

def gettext(fname):
    with open(fname) as f:
        return f.read()

def measure(text):
    d = collections.defaultdict(int)
    for c in text:
        d[c] += 1
    return d

def entropy(text):
    counts = measure(text)
    H = 0
    for key in counts:
        p = counts[key]/len(text)
        H -= p * log(p, 2)
    return H

def main():
    t = gettext("moby.txt")
    print("The entropy of Moby Dick is", entropy(t))

main()
```

# Project: Reading DNA Frames

```
# amino.py

CODONS = {"UUU":"F","UUC":"F","UUA":"L","UUG":"L",
          "UCU":"S","UCC":"S","UCA":"S","UCG":"S",
          "UAU":"Y","UAC":"Y","UAA":"_","UAG":"_",
          "UGU":"C","UGC":"C","UGA":"_","UGG":"W",
          "CUU":"L","CUC":"L","CUA":"L","CUG":"L",
          "CCU":"P","CCC":"P","CCA":"P","CCG":"P",
          "CAU":"H","CAC":"H","CAA":"Q","CAG":"Q",
          "CGU":"R","CGC":"R","CGA":"R","CGG":"R",
          "AUU":"I","AUC":"I","AUA":"I","AUG":"M",
          "ACU":"T","ACC":"T","ACA":"T","ACG":"T",
          "AAU":"N","AAC":"N","AAA":"K","AAG":"K",
          "AGU":"S","AGC":"S","AGA":"R","AGG":"R",
          "GUU":"V","GUC":"V","GUA":"V","GUG":"V",
          "GCU":"A","GCC":"A","GCA":"A","GCG":"A",
          "GAU":"D","GAC":"D","GAA":"E","GAG":"E",
          "GGU":"G","GGC":"G","GGA":"G","GGG":"G"}

NUCLEOTIDES = "ACGT"

def reverse(s):
    return s[::-1]

def complement(dna):
    return dna.translate(dna.maketrans(NUCLEOTIDES,reverse(NUCLEOTIDES)))

def revcomp(dna):
    return reverse(complement(dna))

def translate(rna, frame=0):
    result = ""
    for i in range(frame, len(rna), 3):
        result += CODONS.get(rna[i:i+3], "")
    return result

def transcription(dna):
    return dna.replace("T", "U")

def readfasta(fname):
    with open(fname) as f:
        lines = f.readlines()
    result = ""
    for line in lines[1:]:
        result += line.strip().upper()
    return result

def stops(seq):
    return seq.replace("_","\n")

def main():
    dna = readfasta("sample.dna")
    print("Original")
    for frame in range(3):
        print("-" * 10 + "Frame " + str(frame))
        print(stops(translate(transcription(dna), frame)))
    print("\n\nReverse Complement")
    for frame in range(3):
        print("-" * 10 + "Frame " + str(frame))
        print(stops(translate(transcription(revcomp(dna)), frame)))

main()
```