Contents

Preface

This document contains listings of the programs I use to teach an introductory course on programming in Python. Most of the programs are from the excellent textbook *Introduction to Programming in Python: An Interdisciplinary Approach* by Robert Sedgewick, Kevin Wayne, and Robert Dondero. The programs have been modified for clarity and consistency. The listing for each program includes colorized Python code for the program with line numbers, and a terminal session showing the command to run the program along with the program output.

The programs are available as a PyCharm project (`https://www.cs.umb.edu/~siyer/teaching/ipp.zip`). Consult the section *Programming Environment Setup* for instructions on how to setup your personal computer with the environment needed to run the programs. To open the project, launch PyCharm and open `~/workspace/ipp/programs`. You can run a program on the PyCharm terminal, as follows:

```
>_ ~/workspace/ipp/programs
$ python3 helloworld.py
Hello, World
$ _
```

You can also run/debug a program within PyCharm by selecting an appropriate *Run/Debug configuration* and clicking *Run* or *Debug*.

Input data files are available under the `data` folder, also included in `ipp.zip`. The table in section *Data Files* lists the names of the data-driven programs, references to them within this document, and the relevant data files.

Programming Environment Setup

- In the following sections, lines which start with the `$` symbol denote commands that **must be** run in the terminal, and lines which start with the `#` symbol denote comments, which **must not be** run.
- When you run the commands, do **not** copy-paste the commands into the terminal; type them out instead, without including the `$` symbol.

On Ubuntu Linux

1. Launch a terminal and run the following commands.

```
>_ ~/
# Change to your home folder.
$ cd $HOME
# Install python3-pip and python3-tk.
$ sudo apt-get install python3-pip python3-tk
# Install numpy and pygame.
$ sudo pip3 install numpy pygame
# Create lib and workspace folders.
$ mkdir lib workspace
# Download stdlib-python.zip.
$ wget https://www.cs.umb.edu/~siyer/teaching/stdlib-python.zip
# Unzip stdlib-python.zip under ~/lib.
$ unzip stdlib-python.zip -d lib
# Remove stdlib-python.zip.
$ rm stdlib-python.zip
# Set the PYTHONPATH environment variable.
$ echo "export PYTHONPATH=.:$HOME/lib/stdlib-python" >> $HOME/.bashrc
# Download ipp.zip.
$ wget https://www.cs.umb.edu/~siyer/teaching/ipp.zip
# Unzip ipp.zip under ~/workspace.
$ unzip ipp.zip -d workspace
# Remove ipp.zip.
$ rm ipp.zip
```

2. Download and install PyCharm (Community Edition).

On Mac OS X

1. Launch a terminal and run the following commands.

```
>_ ~/
# Change to your home folder.
$ cd $HOME
# Install brew.
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
# Install python.
$ brew install python3
# Install numpy, pygame, and tk.
$ sudo pip3 install numpy pygame tk
# Create lib and workspace folders.
$ mkdir lib workspace
# Download stdlib-python.zip.
$ wget https://www.cs.umb.edu/~siyer/teaching/stdlib-python.zip
# Unzip stdlib-python.zip under ~/lib.
$ unzip stdlib-python.zip -d lib
# Remove stdlib-python.zip.
$ rm stdlib-python.zip
# Set the PYTHONPATH environment variable.
$ echo "export PYTHONPATH=.:$HOME/lib/stdlib-python" >> $HOME/.bash_profile
# Download ipp.zip.
$ wget https://www.cs.umb.edu/~siyer/teaching/ipp.zip
# Unzip ipp.zip under ~/workspace.
$ unzip ipp.zip -d lib
# Remove ipp.zip.
$ rm ipp.zip
```

```
# Change your shell to bash.
$ chsh -s /bin/bash
```

2. Download and install PyCharm (Community Edition). Launch PyCharm, go to *Configure* → *Settings* → *Tools* → *Terminal* and set *Shell path* to `/bin/bash`.

On Windows

1. Install Python 3. Make surre you have checked the box that says "Add Python 3.x to PATH".

2. Launch a powershell terminal and run the following commands.

```
>_ ~/
# Change to your home folder.
$ cd $HOME
# Install numpy and pygame.
$ pip3 install numpy pygame
# Create lib and workspace folders.
$ mkdir lib,workspace
# Download stdlib-python.zip.
$ wget -O stdlib-python.zip https://www.cs.umb.edu/~siyer/teaching/stdlib-python.zip
# Unzip stdlib-python.zip under ~/lib.
$ Expand-Archive -LiteralPath stdlib-python.zip -DestinationPath lib
# Remove stdlib-python.zip.
$ rm stdlib-python.zip
# Set the PYTHONPATH environment variable.
$ setx PYTHONPATH ".;$HOME\lib\stdlib-python"
# Download ipp.zip.
$ wget -O ipp.zip https://www.cs.umb.edu/~siyer/teaching/ipp.zip
# Unzip ipp.zip under ~/workspace.
$ Expand-Archive -LiteralPath ipp.zip -DestinationPath workspace
# Remove ipp.zip.
$ rm ipp.zip
```

3. Download and install PyCharm (Community Edition). Launch PyCharm, go to *Configure* → *Settings* → *Tools* → *Terminal* and set *Shell path* to `powershell`.

Data Files

Program	Reference	Data Files
plotfilter.py	1.5.5	usa.txt
playthattune.py	1.5.7	ascale.txt, elise.txt, entertainer.txt, firstcut.txt, freebird.txt, looney.txt, stairwaytoheaven.txt
transition.py	1.6.1	small.txt, medium.txt
playthattunedeluxe.py	2.1.3	ascale.txt, elise.txt, entertainer.txt, firstcut.txt, freebird.txt, looney.txt, stairwaytoheaven.txt
ifs.py	2.2.3	barnsley.txt, coral.txt, culcita.txt, cyclosorus.txt, dragon.txt, fishbone.txt, floor.txt, koch.txt, sierpinski.txt, spiral.txt, swirl.txt, tree.txt, zigzag.txt
percolation.py	2.4.2	test5.txt, test8.txt
grayscale.py	3.1.4	darwin.jpg, darwin.png, mandrill.png, mandrill.png
fade.py	3.1.5	darwin.jpg, darwin.png, mandrill.png, mandrill.png
cat.py	3.1.6	in1.txt, in2.txt
split.py	3.1.7	amino.csv, djia.csv, elements.csv, ip-by-country.csv, ip.csv, morse.csv, phone-na.csv
sketch.py	3.3.4	actg.txt, constitution.txt, djia.csv, huckfinn.txt, prejudice.txt, tale.txt, tomsawyer.txt
comparedocuments.py	3.3.5	documents.txt, actg.txt, constitution.txt, djia.csv, huckfinn.txt, prejudice.txt, tale.txt, tomsawyer.txt
nbody.py	3.4.3	2bodytiny.txt, 2body.txt, 3body.txt, 4body.txt
threesum.py	4.1.1	1Kints.txt, 2Kints.txt, 4Kints.txt, 8Kints.txt, 16Kints.txt, 32Kints.txt
linearsearch.py	4.2.1	tinyW.txt, tinyT.txt, largeW.txt, largeT.txt
binarysearch.py	4.2.2	tinyW.txt, tinyT.txt, largeW.txt, largeT.txt
zipf.py	4.2.3	constitution.txt, huckfinn.txt, mobydick.txt, prejudice.txt, tale.txt, tomsawyer.txt
insertion.py	4.2.4	constitution.txt, huckfinn.txt, mobydick.txt, prejudice.txt, tale.txt, tomsawyer.txt
merge.py	4.2.5	constitution.txt, huckfinn.txt, mobydick.txt, prejudice.txt, tale.txt, tomsawyer.txt
arraystack.py	4.3.2	tobe.txt
arrayqueue.py	4.3.4	tobe.txt
frequencycounter.py	4.3.5	constitution.txt, huckfinn.txt, mobydick.txt, prejudice.txt, tale.txt, tomsawyer.txt
graph.py	4.4.1	tinygraph.txt, routes.txt
pathfinder.py	4.4.2	tinygraph.txt, routes.txt
separation.py	4.4.3	tinymovies.txt, movies.txt

1 Imperative Programming

1.1 Your First Programs

1.1.1 helloworld.py

helloworld.py

```python
# Writes the message "Hello, World" to standard output.

import stdio

stdio.writeln("Hello, World")
```

~/workspace/ipp/programs

```
$ python3 helloworld.py
Hello, World
$ _
```

1.1.2 useargument.py

useargument.py

```python
# Accepts a name as command-line argument; and writes a message containing that name to standard
# output.

import stdio
import sys

stdio.write("Hi, ")
stdio.write(sys.argv[1])
stdio.writeln(". How are you?")
```

~/workspace/ipp/programs

```
$ python3 useargument.py Alice
Hi, Alice. How are you?
$ python3 useargument.py Bob
Hi, Bob. How are you?
$ python3 useargument.py Carol
Hi, Carol. How are you?
$ _
```

1.2 Basic Data Types

1.2.1 dateformats.py

dateformats.py

```python
# Accepts d (str), m (str), and y (str) representing a date as command-line arguments; and writes
# the date in different formats to standard output.

import stdio
import sys

d = sys.argv[1]
m = sys.argv[2]
y = sys.argv[3]
dmy = d + "/" + m + "/" + y
mdy = m + "/" + d + "/" + y
ymd = y + "/" + m + "/" + d
stdio.writeln(dmy)
stdio.writeln(mdy)
stdio.writeln(ymd)
```

~/workspace/ipp/programs

```
$ python3 dateformats.py 14 03 1879
14/03/1879
03/14/1879
1879/03/14
$ _
```

1.2.2 sumofsquares.py

sumofsquares.py

```python
# Accepts x (int) and y (int) as command-line arguments; and writes the sum of their squares to
# standard output.

import stdio
import sys

x = int(sys.argv[1])
y = int(sys.argv[2])
result = x * x + y * y
stdio.writeln(result)
```

~/workspace/ipp/programs

```
$ python3 sumofsquares.py 3 4
25
$ python3 sumofsquares.py 6 8
100
$ _
```

1.2.3 quadratic.py

quadratic.py

```python
# Accepts a (float), b (float), and c (float) as command-line arguments; and writes the two roots
# of the quadratic equation ax^2 + bx + c = 0 to standard output.

import math
import stdio
import sys

a = float(sys.argv[1])
b = float(sys.argv[2])
c = float(sys.argv[3])
discriminant = b * b - 4 * a * c
root1 = (-b + math.sqrt(discriminant)) / (2 * a)
root2 = (-b - math.sqrt(discriminant)) / (2 * a)
stdio.writeln("Root # 1 = " + str(root1))
stdio.writeln("Root # 2 = " + str(root2))
```

~/workspace/ipp/programs

```
$ python3 quadratic.py 1 -5 6
Root # 1 = 3.0
Root # 2 = 2.0
$ python3 quadratic.py 1 -1 -1
Root # 1 = 1.618033988749895
Root # 2 = -0.6180339887498949
$ _
```

1.2.4 leapyear.py

leapyear.py

```python
# Accepts y (int) as command-line argument representing a year; and writes to standard output
# whether the year is a leap year or not.

import stdio
import sys

y = int(sys.argv[1])
result = y % 4 == 0 and y % 100 != 0 or y % 400 == 0
stdio.writeln(result)
```

~/workspace/ipp/programs

```
$ python3 leapyear.py 2020
True
$ python3 leapyear.py 1900
False
$ python3 leapyear.py 2000
True
```

```
$ _
```

1.3 Control Flow

1.3.1 grade.py

grade.py

```python
# Accepts a percentage score (float) as command-line argument; and writes the corresponding
# letter grade to standard output.

import stdio
import sys

score = float(sys.argv[1])
if score >= 93:
    stdio.writeln("A")
elif score >= 90:
    stdio.writeln("A-")
elif score >= 87:
    stdio.writeln("B+")
elif score >= 83:
    stdio.writeln("B")
elif score >= 80:
    stdio.writeln("B-")
elif score >= 77:
    stdio.writeln("C+")
elif score >= 73:
    stdio.writeln("C")
elif score >= 70:
    stdio.writeln("C-")
elif score >= 67:
    stdio.writeln("D+")
elif score >= 63:
    stdio.writeln("D")
elif score >= 60:
    stdio.writeln("D-")
else:
    stdio.writeln("F")
```

>_ ~/workspace/ipp/programs

```
$ python3 grade.py 97
A
$ python3 grade.py 56
F
$ _
```

1.3.2 flip.py

flip.py

```python
# Simulates a coin flip by writing 'Heads' or 'Tails' to standard output.

import stdio
import stdrandom

result = "Heads" if stdrandom.bernoulli() else "Tails"
stdio.writeln(result)
```

>_ ~/workspace/ipp/programs

```
$ python3 flip.py
Heads
$ python3 flip.py
Heads
$ python3 flip.py
Tails
$ _
```

1.3.3 nhellos.py

nhellos.py

```python
# Accepts n (int) as command-line argument; and writes n Hellos to standard output.

import stdio
import sys

n = int(sys.argv[1])
i = 1
while i <= n:
    stdio.writeln("Hello # " + str(i))
    i += 1
```

~/workspace/ipp/programs

```
$ python3 nhellos.py 10
Hello # 1
Hello # 2
Hello # 3
Hello # 4
Hello # 5
Hello # 6
Hello # 7
Hello # 8
Hello # 9
Hello # 10
$ _
```

1.3.4 powersoftwo.py

powersoftwo.py

```python
# Accepts n (int) as command-line argument; and writes to standard output a table of powers of 2
# that are less than or equal to 2^n.

import stdio
import sys

n = int(sys.argv[1])
power = 1
for i in range(n + 1):
    stdio.writeln(str(i) + " " + str(power))
    power *= 2
```

~/workspace/ipp/programs

```
$ python3 powersoftwo.py 8
0 1
1 2
2 4
3 8
4 16
5 32
6 64
7 128
8 256
$ _
```

1.3.5 divisorpattern.py

divisorpattern.py

```python
# Accepts n (int) as command-line argument; and writes to standard output a table where entry
# (i, j) is a "*" if j divides i or i divides j and a " " otherwise.

import stdio
import sys

n = int(sys.argv[1])
for i in range(1, n + 1):
    for j in range(1, n + 1):
        if i % j == 0 or j % i == 0:
            stdio.write("* ")
```

```
        else:
            stdio.write("  ")
    stdio.writeln(i)
```

```
>_ ~/workspace/ipp/programs
$ python3 divisorpattern.py 20
* * * * * * * * * * * * * * * * * * * 1
* *   *   *   *   *   *   *   *   *   * 2
*   *     *     *     *     *     *     3
* *   *       *       *       *       * 4
*       *         *         *         * 5
* * *     *           *           *     6
*           *             *             7
* *   *       *               *         8
*   *           *                 *     9
* *     *         *                   * 10
*                   *                   11
* * * *   *           *                 12
*                       *               13
* *         *             *             14
*   *   *                 *           15
* *   *       *               *         16
*                               *       17
* * *     *     *                 *     18
*                                   *   19
* *   * *         *                   * 20
$ _
```

1.3.6 harmonic.py

harmonic.py

```python
# Accepts n (int) as command-line argument; and writes the nth harmonic number (1 + 1/2 + ... +
# 1/n) to standard output.

import stdio
import sys

n = int(sys.argv[1])
total = 0.0
for i in range(1, n + 1):
    total += 1 / i
stdio.writeln(total)
```

```
>_ ~/workspace/ipp/programs
$ python3 harmonic.py 10
2.9289682539682538
$ python3 harmonic.py 1000
7.485470860550343
$ python3 harmonic.py 10000
9.787606036044348
$ _
```

1.3.7 sqrt.py

sqrt.py

```python
# Accepts c (float) as command-line argument; and writes the square root of c to standard output,
# computed using Newton's method.

import stdio
import sys

c = float(sys.argv[1])
EPSILON = 1e-15
t = c
while abs(1 - c / (t * t)) > EPSILON:
    t = (c / t + t) / 2
stdio.writeln(t)
```

```
>_ ~/workspace/ipp/programs
$ python3 sqrt.py 2
1.414213562373095
$ python3 sqrt.py 1000000
1000.0
$ python3 sqrt.py 0.4
0.6324555320336759
$ python3 sqrt.py 1048575
1023.9995117186336
$ python3 sqrt.py 16664444
4082.2106756021303
$ python3 sqrt.py 1e-50
9.999999999999999e-26
$ _
```

1.3.8 binary.py

binary.py

```python
# Accepts n (int) as command-line argument; and writes the binary representation of n to standard
# output.

import stdio
import sys

n = int(sys.argv[1])
v = 1
while v <= n // 2:
    v *= 2
while v > 0:
    if n < v:
        stdio.write("0")
    else:
        stdio.write("1")
        n -= v
    v //= 2
stdio.writeln()
```

```
>_ ~/workspace/ipp/programs
$ python3 binary.py 19
10011
$ python3 binary.py 255
11111111
$ python3 binary.py 512
1000000000
$ python3 binary.py 1000000000
111011100110101100101000000000
$ _
```

1.3.9 gambler.py

gambler.py

```python
# Accepts stake (int), goal (int), and trials (int) as command-line arguments; runs trials
# experiments (dollar bets) that start with stake dollars and terminate on 0 dollars or goal; and
# writes the percentage of wins and the average number of bets per experiment to standard output.

import stdio
import sys
import stdrandom

stake = int(sys.argv[1])
goal = int(sys.argv[2])
trials = int(sys.argv[3])
bets = 0
wins = 0
for t in range(trials):
    cash = stake
    while 0 < cash < goal:
        bets += 1
        if stdrandom.bernoulli():
            cash += 1
        else:
            cash -= 1
```

```
    if cash == goal:
        wins += 1
stdio.writeln(str(100 * wins // trials) + "% wins")
stdio.writeln("Avg # bets: " + str(bets // trials))
```

```
>_ ~/workspace/ipp/programs
$ python3 gambler.py 10 20 1000
46% wins
Avg # bets: 97
$ python3 gambler.py 50 250 100
19% wins
Avg # bets: 12069
$ python3 gambler.py 500 2500 100
19% wins
Avg # bets: 1155781
$ _
```

1.3.10 factors.py

factors.py

```
# Accepts n (int) as command-line argument; and writes the prime factors of n to standard output.

import stdio
import sys

n = int(sys.argv[1])
factor = 2
while factor * factor <= n:
    while n % factor == 0:
        stdio.write(str(factor) + " ")
        n //= factor
    factor += 1
if n > 1:
    stdio.write(n)
stdio.writeln()
```

```
>_ ~/workspace/ipp/programs
$ python3 factors.py 3757208
2 2 2 7 13 13 397
$ python3 factors.py 287994837222311
17 1739347 9739789
$ _
```

1.4 Collection Data Types

1.4.1 sample.py

sample.py

```
# Accepts m (int) and n (int) as command-line arguments; and writes to standard output a random
# sample of m integers in the range [0, n), with no duplicates.

import stdarray
import stdio
import stdrandom
import sys

m = int(sys.argv[1])
n = int(sys.argv[2])
perm = stdarray.create1D(n, 0)
for i in range(n):
    perm[i] = i
for i in range(m):
    r = stdrandom.uniformInt(i, n)
    temp = perm[r]
    perm[r] = perm[i]
    perm[i] = temp
for i in range(m):
    stdio.write(str(perm[i]) + " ")
stdio.writeln()
```

```
>_ ~/workspace/ipp/programs
$ python3 sample.py 6 16
10 7 11 1 8 5
$ python3 sample.py 10 1000
258 802 440 28 244 256 564 11 515 24
$ python3 sample.py 20 20
15 11 13 1 5 8 16 7 0 4 10 18 19 14 3 12 2 6 9 17
$ _
```

1.4.2 couponcollector.py

```
couponcollector.py
# Accepts n (int) as a command-line argument; and writes to standard output the number of coupons
# you collect before obtaining one of each of n types.

import stdarray
import stdio
import stdrandom
import sys

n = int(sys.argv[1])
count = 0
collectedCount = 0
isCollected = stdarray.create1D(n, False)
while collectedCount < n:
    value = stdrandom.uniformInt(0, n)
    count += 1
    if not isCollected[value]:
        collectedCount += 1
        isCollected[value] = True
stdio.writeln(count)
```

```
>_ ~/workspace/ipp/programs
$ python3 couponcollector.py 1000
6276
$ python3 couponcollector.py 1000
7038
$ python3 couponcollector.py 1000000
13401736
$ _
```

1.4.3 primesieve.py

```
primesieve.py
# Accepts n (int) as command-line argument; and writes the number of primes <= n to standard output.

import stdarray
import stdio
import sys

n = int(sys.argv[1])
isPrime = stdarray.create1D(n + 1, True)
for i in range(2, n):
    if isPrime[i]:
        for j in range(2, n // i + 1):
            isPrime[i * j] = False
count = 0
for i in range(2, n + 1):
    if isPrime[i]:
        count += 1
stdio.writeln(count)
```

```
>_ ~/workspace/ipp/programs
$ python3 primesieve.py 100
25
$ python3 primesieve.py 1000
168
$ python3 primesieve.py 1000000
78498
$ python3 primesieve.py 100000000
5761455
```

```
$ _
```

1.4.4 selfavoid.py

selfavoid.py

```python
# Accepts n (int) and trials (int) as command-line arguments; and writes to standard output the
# percentage of dead ends encountered out of trials self-avoiding random walks on an n-by-n lattice.

import stdarray
import stdio
import stdrandom
import sys

n = int(sys.argv[1])
trials = int(sys.argv[2])
deadEnds = 0
for t in range(trials):
    a = stdarray.create2D(n, n, False)
    x = n // 2
    y = n // 2
    while 0 < x < n - 1 and 0 < y < n - 1:
        a[x][y] = True
        if a[x - 1][y] and a[x + 1][y] and a[x][y - 1] and a[x][y + 1]:
            deadEnds += 1
            break
        r = stdrandom.uniformInt(1, 5)
        if r == 1 and not a[x + 1][y]:
            x += 1
        elif r == 2 and not a[x - 1][y]:
            x -= 1
        elif r == 3 and not a[x][y + 1]:
            y += 1
        elif r == 4 and not a[x][y - 1]:
            y -= 1
stdio.writeln(str(100 * deadEnds // trials) + "% dead ends")
```

>_ ~/workspace/ipp/programs

```
$ python3 selfavoid.py 5 1000
0% dead ends
$ python3 selfavoid.py 20 1000
33% dead ends
$ python3 selfavoid.py 40 1000
78% dead ends
$ python3 selfavoid.py 80 1000
98% dead ends
$ _
```

1.5 Input and Output

1.5.1 randomseq.py

randomseq.py

```python
# Accepts n (int), lo (float), and hi (float) as command-line arguments; and writes to standard
# output n random floats in the range [lo, hi), each up to 2 decimal places.

import stdio
import stdrandom
import sys

n = int(sys.argv[1])
lo = float(sys.argv[2])
hi = float(sys.argv[3])
for i in range(n):
    r = stdrandom.uniformFloat(lo, hi)
    stdio.writef("%.2f\n", r)
```

>_ ~/workspace/ipp/programs

```
$ python3 randomseq.py 10 100 200
186.69
102.34
176.05
```

```
182.78
161.95
169.34
155.65
154.96
194.41
103.91
$ _
```

1.5.2 twentyquestions.py

twentyquestions.py

```python
# Generates a random integer; repeatedly accepts user guesses from standard input; writes "Too
# low" or "Too high" to standard output, as appropriate, in response to each guess; and writes
# "You win!" to standard output and exits when the user"s guess is correct.

import stdio
import stdrandom

RANGE = 1000000
secret = stdrandom.uniformInt(1, RANGE + 1)
stdio.writef("I am thinking of a secret number between 1 and %d\n", RANGE)
guess = 0
while guess != secret:
    stdio.write("What is your guess? ")
    guess = stdio.readInt()
    if guess < secret:
        stdio.writeln("Too low")
    elif guess > secret:
        stdio.writeln("Too high")
    else:
        stdio.writeln("You win!")
```

>_ ~/workspace/ipp/programs

```
$ python3 twenty questions
I am thinking of a secret number between 1 and 1000000
What is your guess? 500000
Too low
What is your guess? 750000
Too high
What is your guess? 625000
Too high
What is your guess? 562500
Too high
What is your guess? 531250
Too high
What is your guess? 515625
Too high
What is your guess? 507812
Too high
What is your guess? 503906
Too high
What is your guess? 501953
Too high
What is your guess? 500977
Too low
What is your guess? 501465
Too low
What is your guess? 501709
Too high
What is your guess? 501587
Too low
What is your guess? 501648
Too low
What is your guess? 501679
Too low
What is your guess? 501694
Too high
What is your guess? 501686
You win!
$ _
```

1.5.3 average.py

average.py

```python
# Accepts floats from standard input; and writes their average to standard output.

import stdio

total = 0.0
count = 0
while not stdio.isEmpty():
    x = stdio.readFloat()
    total += x
    count += 1
average = total / count
stdio.writeln("Average is " + str(average))
```

>_ ~/workspace/ipp/programs

```
$ python3 average.py
10.0 5.0 6.0 <enter>
3.0 7.0 32.0 <enter>
<ctrl-d>
Average is 10.5
$ _
```

1.5.4 rangefilter.py

rangefilter.py

```python
# Accepts lo (int) and hi (int) as command-line arguments and integers from standard input; and
# writes to standard output those integers that are in the range [lo, hi].

import stdio
import sys

lo = int(sys.argv[1])
hi = int(sys.argv[2])
while not stdio.isEmpty():
    x = stdio.readInt()
    if lo <= x <= hi:
        stdio.write(str(x) + " ")
stdio.writeln()
```

>_ ~/workspace/ipp/programs

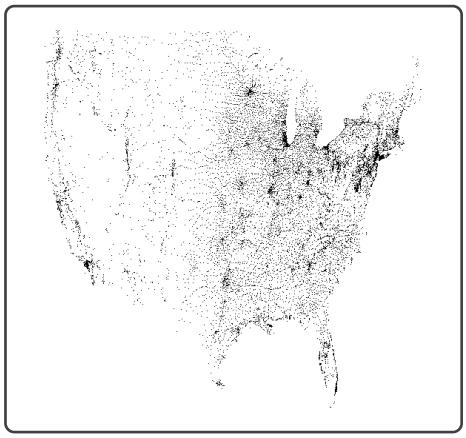
```
$ python3 rangefilter.py 100 400
358 1330 55 165 689 1014 3066 387 575 843 203 48 292 877 65 998 <enter>
358 165 387 203 292 <ctrl-d>
$ _
```

1.5.5 plotfilter.py

plotfilter.py

```python
# Accepts x and y scales and (x, y) points from standard input; configures standard draw
# appropriately; and plots the points using standard draw.

import stddraw
import stdio

x0 = stdio.readFloat()
y0 = stdio.readFloat()
x1 = stdio.readFloat()
y1 = stdio.readFloat()
stddraw.setXscale(x0, x1)
stddraw.setYscale(y0, y1)
stddraw.setPenRadius(0.0)
while not stdio.isEmpty():
    x = stdio.readFloat()
    y = stdio.readFloat()
    stddraw.point(x, y)
stddraw.show()
```

```
>_ ~/workspace/ipp/programs
$ python3 plotfilter.py < ../data/usa.txt
$ _
```

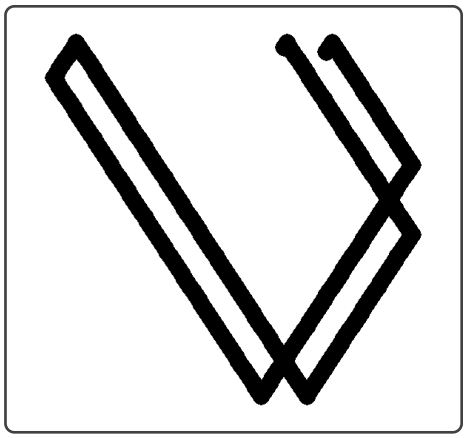


1.5.6 bouncingball.py

bouncingball.py

```python
# Draws a bouncing ball using standard draw.

import stddraw

RADIUS = 0.05
DT = 1.0
PAUSE = 20
stddraw.setXscale(-1.0, 1.0)
stddraw.setYscale(-1.0, 1.0)
rx = 0.480
ry = 0.860
vx = 0.015
vy = 0.023
while True:
    if abs(rx + vx * DT) + RADIUS > 1.0:
        vx = -vx
    if abs(ry + vy * DT) + RADIUS > 1.0:
        vy = -vy
    rx += vx * DT
    ry += vy * DT
    stddraw.clear(stddraw.WHITE)
    stddraw.filledCircle(rx, ry, RADIUS)
    stddraw.show(PAUSE)
```

```
>_ ~/workspace/ipp/programs
$ python3 bouncingball.py
$ _
```



1.5.7 playthattune.py

playthattune.py

```
# Accepts sound samples, each characterized by a pitch and a duration, from standard input; and
# plays the sound using standard audio.

import math
import stdarray
import stdaudio
import stdio

SPS = 44100
NOTES_ON_SCALE = 12
CONCERT_A = 440.0
while not stdio.isEmpty():
    pitch = stdio.readInt()
    duration = stdio.readFloat()
    hz = CONCERT_A * math.pow(2, pitch / NOTES_ON_SCALE)
    n = int(SPS * duration)
    note = stdarray.create1D(n + 1, 0.0)
    for i in range(n + 1):
        note[i] = math.sin(2 * math.pi * i * hz / SPS)
    stdaudio.playSamples(note)
stdaudio.wait()
```

>_ ~/workspace/ipp/programs

```
$ python3 playthattune.py < ../data/elise.txt
$ _
```

1.6 Case Study: What Makes Google Different? (PageRank Algorithm)

1.6.1 transition.py

transition.py

```
# Accepts links from standard input; and writes the corresponding transition matrix to standard
# output, computed using the 90-10 rule.

import stdarray
import stdio

n = stdio.readInt()
outDegrees = stdarray.create1D(n, 0)
linkCounts = stdarray.create2D(n, n, 0)
while not stdio.isEmpty():
    i = stdio.readInt()
    j = stdio.readInt()
    outDegrees[i] += 1
    linkCounts[i][j] += 1
stdio.writeln(str(n) + " " + str(n))
for i in range(n):
    for j in range(n):
        p = 0.9 * linkCounts[i][j] / outDegrees[i] + 0.1 / n
```

```
        stdio.writef("%8.5f", p)
    stdio.writeln()
```

```
>_ ~/workspace/ipp/programs
$ python3 transition.py < ../data/small.txt
5 5
 0.02000 0.92000 0.02000 0.02000 0.02000
 0.02000 0.02000 0.38000 0.38000 0.20000
 0.02000 0.02000 0.02000 0.92000 0.02000
 0.92000 0.02000 0.02000 0.02000 0.02000
 0.47000 0.02000 0.47000 0.02000 0.02000
$ _
```

1.6.2 randomsurfer.py

```
randomsurfer.py
# Accepts moves (int) as command-line argument and a transition matrix from standard input;
# performs moves transitions as prescribed by the transition matrix; and writes the relative
# frequency of hitting each page to standard output.

import stdarray
import stdio
import stdrandom
import sys

moves = int(sys.argv[1])
transitionMatrix = stdarray.readFloat2D()
n = len(transitionMatrix)
hits = stdarray.create1D(n, 0)
page = 0
for m in range(moves):
    page = stdrandom.discrete(transitionMatrix[page])
    hits[page] += 1
for hit in hits:
    stdio.writef("%8.5f", hit / moves)
stdio.writeln()
```

```
>_ ~/workspace/ipp/programs
$ python3 transition.py < ../data/small.txt | python3 randomsurfer.py 100
 0.29000 0.30000 0.09000 0.25000 0.07000
$ python3 transition.py < ../data/small.txt | python3 randomsurfer.py 10000
 0.27320 0.26840 0.14590 0.24550 0.06700
$ python3 transition.py < ../data/small.txt | python3 randomsurfer.py 10000000
 0.27301 0.26569 0.14621 0.24727 0.06782
$ _
```

1.6.3 markov.py

```
markov.py
# Accepts moves (int) as command-line argument and a transition matrix from standard input;
# computes the probabilities that a random surfer lands on each page (page ranks) after moves
# matrix-vector multiplications; and writes the page ranks to standard output.

import stdarray
import stdio
import sys

moves = int(sys.argv[1])
transitionMatrix = stdarray.readFloat2D()
n = len(transitionMatrix)
ranks = stdarray.create1D(n, 0.0)
ranks[0] = 1.0
for m in range(moves):
    newRanks = stdarray.create1D(n, 0.0)
    for j in range(n):
        for i in range(n):
            newRanks[j] += ranks[i] * transitionMatrix[i][j]
    ranks = newRanks
for rank in ranks:
    stdio.writef("%8.5f", rank)
stdio.writeln()
```

```
>_ ~/workspace/ipp/programs
$ python3 transition.py < ../data/small.txt | python3 markov.py 20
 0.27245 0.26515 0.14669 0.24764 0.06806
$ python3 transition.py < ../data/small.txt | python3 markov.py 40
 0.27303 0.26573 0.14618 0.24723 0.06783
$ _
```

2 Procedural Programming

2.1 Defining Functions

2.1.1 harmonicredux.py

harmonicredux.py

```python
# Accepts n (int) as command-line argument; and writes the nth harmonic number (1 + 1/2 + ... +
# 1/n) to standard output.

import stdio
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
    stdio.writeln(_harmonic(n))


# Returns the nth harmonic number.
def _harmonic(n):
    total = 0.0
    for i in range(1, n + 1):
        total += 1 / i
    return total


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 harmonicredux.py 10
2.9289682539682538
$ python3 harmonicredux.py 1000
7.485470860550343
$ python3 harmonicredux.py 10000
9.787606036044348
$ _
```

2.1.2 couponcollectorredux.py

couponcollectorredux.py

```python
# Accepts n (int) as a command-line argument; and writes to standard output the number of coupons
# you collect before obtaining one of each of n types.

import stdarray
import stdio
import stdrandom
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
    stdio.writeln(_collect(n))


# Collects coupons until getting one of each value in the range 0 to n - 1, and returns the
# number of coupons collected.
def _collect(n):
    count = 0
    collectedCount = 0
```

```
    isCollected = stdarray.create1D(n, False)
    while collectedCount < n:
        value = _getCoupon(n)
        count += 1
        if not isCollected[value]:
            collectedCount += 1
            isCollected[value] = True
    return count


# Returns a random coupon between 0 and n - 1.
def _getCoupon(n):
    return stdrandom.uniformInt(0, n)


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 couponcollectorredux.py 1000
6276
$ python3 couponcollectorredux.py 1000
7038
$ python3 couponcollectorredux.py 1000000
13401736
$ _
```

2.1.3 playthattunedeluxe.py

playthattunedeluxe.py

```
# Reads sound samples, each characterized by a pitch and a duration, from standard input; and plays
# the sound using standard audio.

import math
import stdarray
import stdaudio
import stdio


# Entry point.
def main():
    while not stdio.isEmpty():
        pitch = stdio.readInt()
        duration = stdio.readFloat()
        stdaudio.playSamples(_createRichNote(pitch, duration))
    stdaudio.wait()


# Returns an array of samples for a note superposed from three notes (at pitch, 2 x pitch,
# and 0.5 x pitch) and having the specified duration.
def _createRichNote(pitch, duration):
    NOTES_ON_SCALE = 12
    CONCERT_A = 440.0
    hz = CONCERT_A * math.pow(2, pitch / NOTES_ON_SCALE)
    mid = _createNote(hz, duration)
    hi = _createNote(2 * hz, duration)
    lo = _createNote(hz / 2, duration)
    hiAndLo = _superpose(hi, lo, 0.5, 0.5)
    return _superpose(mid, hiAndLo, 0.5, 0.5)


# Returns an array of samples for a note of specified frequency and duration.
def _createNote(hz, duration):
    SPS = 44100
    n = int(SPS * duration)
    note = stdarray.create1D(n + 1, 0.0)
    for i in range(n + 1):
        note[i] = math.sin(2 * math.pi * i * hz / SPS)
    return note


# Superposes arrays a and b, using respective weights aWeight and bWeight, and returns the
# superposed array.
def _superpose(a, b, aWeight, bWeight):
    c = stdarray.create1D(len(a), 0.0)
    for i in range(len(a)):
```

```
        c[i] = a[i] * aWeight + b[i] * bWeight
    return c


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 playthattunedeluxe.py < ../data/elise.txt
$ _
```

2.2 Modules and Applications

2.2.1 gaussiantable.py

gaussiantable.py

```
# Accepts mu (float) and sigma (float) as command-line arguments; and writes to standard output a
# table of the percentage of students scoring below certain scores on the SAT, assuming the test
# scores obey a Gaussian distribution with the given mean and standard deviation.

import gaussian
import stdio
import sys


# Entry point.
def main():
    mu = float(sys.argv[1])
    sigma = float(sys.argv[2])
    for score in range(400, 1600 + 1, 100):
        percentile = gaussian.cdf(score, mu, sigma)
        stdio.writef("%4d  %.4f\n", score, percentile)


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 gaussiantable.py 1019 209
 400  0.0015
 500  0.0065
 600  0.0225
 700  0.0635
 800  0.1474
 900  0.2845
1000  0.4638
1100  0.6508
1200  0.8068
1300  0.9106
1400  0.9658
1500  0.9893
1600  0.9973
$ _
```

2.2.2 gaussian.py

gaussian.py

```
# A library of Gaussian functions.

import math
import stdio
import sys


# Returns the value of the Gaussian probability density function with mean mu and standard
# deviation sigma at the given x value.
def pdf(x, mu=0.0, sigma=1.0):
    z = (x - mu) / sigma
    return _pdf(z) / sigma


```

```python
# Returns the value of the Gaussian cumulative distribution function with mean mu and standard
# deviation sigma at the given x value.
def cdf(x, mu=0.0, sigma=1.0):
    z = float(x - mu) / sigma
    return _cdf(z)


# Returns the value of the Gaussian probability density function with mean 0 and standard
# deviation 1 at the given z value.
def _pdf(z):
    return math.exp(-z * z / 2) / math.sqrt(2 * math.pi)


# Returns the value of the Gaussian cumulative distribution function with mean 0 and standard
# deviation 1 at the given z value.
def _cdf(z):
    if z < -8.0:
        return 0.0
    if z > +8.0:
        return 1.0
    total = 0.0
    term = z
    i = 3
    while total != total + term:
        total += term
        term *= z * z / i
        i += 2
    return 0.5 + total * _pdf(z)


# Unit tests the library.
def _main():
    x = float(sys.argv[1])
    mu = float(sys.argv[2])
    sigma = float(sys.argv[3])
    stdio.writeln(cdf(x, mu, sigma))


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ python3 gaussian.py 820 1019 209
0.17050966869132106
$ python3 gaussian.py 1500 1019 209
0.9893164837383885
$ python3 gaussian.py 1500 1025 231
0.9801220907365491
$ _
```

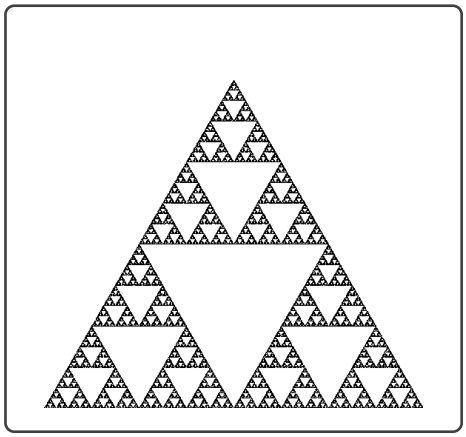
2.2.3 ifs.py

ifs.py

```python
# Accepts n (int) as command-line argument, a 1-by-m vector (probabilities) and two m-by-3
# matrices (coefficients for updating x and y, respectively) from standard input; and plots the
# results as a set of n points using standard output.

import matrix
import stdarray
import stddraw
import stdrandom
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
    dist = stdarray.readFloat1D()
    cx = stdarray.readFloat2D()
    cy = stdarray.readFloat2D()
    x, y = 0.0, 0.0
    stddraw.setPenRadius(0.0)
    for i in range(n):
        r = stdrandom.discrete(dist)
        col = [x, y, 1]
        x0 = matrix.dot(matrix.row(cx, r), col)
```

```
        y0 = matrix.dot(matrix.row(cy, r), col)
        x = x0
        y = y0
        stddraw.point(x, y)
    stddraw.show()


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 ifs.py 100000 < ../data/sierpinski.txt
$ _
```



2.2.4 matrix.py

matrix.py

```
# A library of matrix functions.

import stdarray
import stdio


# Returns the ith row of matrix a.
def row(a, i):
    return a[i]


# Returns the jth column of matrix a.
def col(a, j):
    c = []
    for r in a:
        c += [r[j]]
    return c


# Returns the sum of the matrices a and b.
def add(a, b):
    m, n = len(a), len(a[0])
    c = stdarray.create2D(m, n, 0.0)
    for i in range(m):
        for j in range(n):
            c[i][j] = a[i][j] + b[i][j]
    return c


# Returns the difference of matrices a and b.
def subtract(a, b):
    m, n = len(a), len(a[0])
    c = stdarray.create2D(m, n, 0.0)
    for i in range(m):
        for j in range(n):
            c[i][j] = a[i][j] - b[i][j]
    return c
```

```python


# Returns the product of matrices a and b.
def multiply(a, b):
    m, n = len(a), len(b[0])
    c = stdarray.create2D(m, n, 0.0)
    for i in range(m):
        for j in range(n):
            c[i][j] = dot(row(a, i), col(b, j))
    return c


# Returns the transpose of matrix a.
def transpose(a):
    m, n = len(a), len(a[0])
    c = stdarray.create2D(n, m, 0.0)
    for i in range(m):
        for j in range(n):
            c[j][i] = a[i][j]
    return c


# Returns the dot product of the 1-by-n matrices a and b.
def dot(a, b):
    total = 0.0
    for x, y in zip(a, b):
        total += x * y
    return total


# Unit tests the library.
def _main():
    a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    b = [[1], [2], [3]]
    stdio.writeln("a              = " + str(a))
    stdio.writeln("b              = " + str(b))
    stdio.writeln("row(a, 1)      = " + str(row(a, 1)))
    stdio.writeln("col(a, 1)      = " + str(col(a, 1)))
    stdio.writeln("add(a, a)      = " + str(add(a, a)))
    stdio.writeln("subtract(a, a) = " + str(subtract(a, a)))
    stdio.writeln("multiply(a, b) = " + str(multiply(a, b)))
    stdio.writeln("transpose(b)   = " + str(transpose(b)))


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ python3 matrix.py
a              = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
b              = [[1], [2], [3]]
row(a, 1)      = [4, 5, 6]
col(a, 1)      = [2, 5, 8]
add(a, a)      = [[2, 4, 6], [8, 10, 12], [14, 16, 18]]
subtract(a, a) = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
multiply(a, b) = [[14.0], [32.0], [50.0]]
transpose(b)   = [[1, 2, 3]]
$ _
```

2.3 Recursion

2.3.1 factorial.py

factorial.py

```python
# Accepts n (int) as command-line argument; and writes n! to standard output.

import stdio
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
    stdio.writeln(_factorial(n))

```

```python

# Returns n! computed recursively.
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 factorial.py 0
1
$ python3 factorial.py 5
120
$ _
```

2.3.2 euclid.py

euclid.py

```python
# Accepts p (int) and q (int) as command-line arguments; and writes gcd(p, q) to standard output.

import stdio
import sys


# Entry point.
def main():
    p = int(sys.argv[1])
    q = int(sys.argv[2])
    stdio.writeln(_gcd(p, q))


# Returns the gcd of p and q computed recursively using Euclid's algorithm.
def _gcd(p, q):
    if q == 0:
        return p
    return _gcd(q, p % q)


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 euclid.py 1440 408
24
$ python3 euclid.py 314159 271828
1
$ _
```

2.3.3 towersofhanoi.py

towersofhanoi.py

```python
# Accepts n (int) as command-line argument; and writes to standard output the instructions to move n
# Towers of Hanoi disks to the left.

import stdio
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
    _moves(n, True)


# Writes to standard output the instructions to move n Towers of Hanoi disks to the left (if
# parameter left is True) or to the right (if parameter left is False).
def _moves(n, left):
    if n == 0:
```

```
        return
    _moves(n - 1, not left)
    if left:
        stdio.writeln(str(n) + " left")
    else:
        stdio.writeln(str(n) + " right")
    _moves(n - 1, not left)


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 towersofhanoi.py 1
1 left
$ python3 towersofhanoi.py 2
1 right
2 left
1 right
$ python3 towersofhanoi.py 3
1 left
2 right
1 left
3 left
1 left
2 right
1 left
$ python3 towersofhanoi.py 4
1 right
2 left
1 right
3 right
1 right
2 left
1 right
4 left
1 right
2 left
1 right
3 right
1 right
2 left
1 right
$ _
```

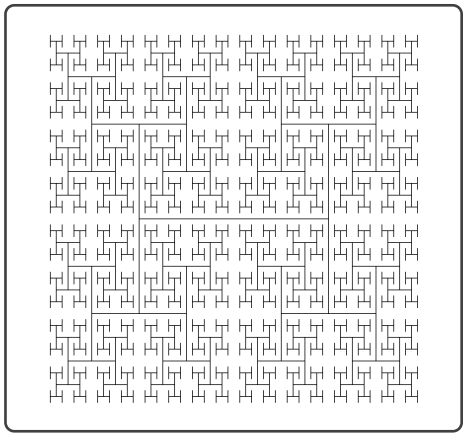
2.3.4 htree.py

htree.py

```
# Accepts n (int) as a command-line argument; and draws using standard draw a level n H-tree
# centered at (0.5, 0.5) with lines of length 0.5.

import stddraw
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
    stddraw.setPenRadius(0.0)
    _draw(n, 0.5, 0.5, 0.5)
    stddraw.show()


# Draws to standard draw a level n H-tree centered at (x, y) with lines of length lineLength.
def _draw(n, lineLength, x, y):
    if n == 0:
        return
    x0 = x - lineLength / 2
    x1 = x + lineLength / 2
    y0 = y - lineLength / 2
    y1 = y + lineLength / 2
    stddraw.line(x0, y, x1, y)
    stddraw.line(x0, y0, x0, y1)
    stddraw.line(x1, y0, x1, y1)
    _draw(n - 1, lineLength / 2, x0, y0)
    _draw(n - 1, lineLength / 2, x0, y1)
```

```
    _draw(n - 1, lineLength / 2, x1, y0)
    _draw(n - 1, lineLength / 2, x1, y1)


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 htree.py 5
$ _
```



2.3.5 fibonacci.py

```
fibonacci.py
# Accepts n (int) as command-line argument; and writes the nth Fibonacci number to standard output.

import stdio
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
    stdio.writeln(_fibonacci(n))


# Returns the nth Fibonacci number computed recursively.
def _fibonacci(n):
    if n == 1 or n == 2:
        return 1
    return _fibonacci(n - 1) + _fibonacci(n - 2)


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 fibonacci.py 0
0
$ python3 fibonacci.py 1
1
$ python3 fibonacci.py 2
1
$ python3 fibonacci.py 3
2
$ python3 fibonacci.py 10
55
$ _
```

2.4 Case Study: Fermi's Paradox (Percolation Problem)

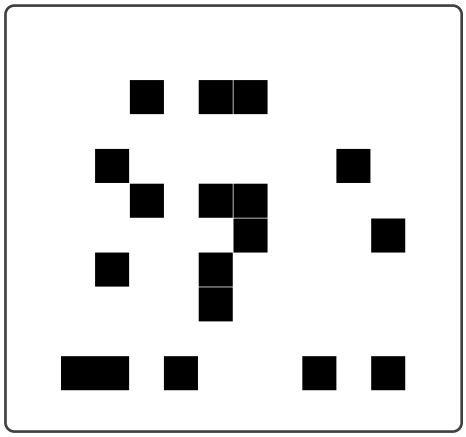
2.4.1 percolationio.py

percolationio.py

```python
# A library of percolation support functions.

import stdarray
import stddraw
import stdrandom
import sys


# Returns an n-by-n percolation system with vacancy probability p.
def random(n, p):
    a = stdarray.create2D(n, n, False)
    for i in range(n):
        for j in range(n):
            a[i][j] = stdrandom.bernoulli(p)
    return a


# Draws the percolation system a to standard draw. Parameter which indicates whether to display
# the entries corresponding to True or to False.
def draw(a, which):
    n = len(a)
    stddraw.setXscale(-1, n)
    stddraw.setYscale(-1, n)
    for i in range(n):
        for j in range(n):
            if a[i][j] == which:
                stddraw.filledSquare(j, n - i - 1, 0.5)


# Unit tests the library.
def _main():
    n = int(sys.argv[1])
    p = float(sys.argv[2])
    isOpen = random(n, p)
    draw(isOpen, False)
    stddraw.show()


if __name__ == "__main__":
    _main()
```

>_ ~/workspace/ipp/programs

```
$ python3 percolationio.py 10 0.8
$ _
```



2.4.2 percolation.py

percolation.py

```python
# A library of percolation functions.

import stdio
import stdarray


# Computes and returns the full sites of the given percolation system.
def flow(isOpen):
    n = len(isOpen)
    isFull = stdarray.create2D(n, n, False)
    for j in range(n):
        _flow(isOpen, isFull, 0, j)
    return isFull


# Given the full sites of a percolation system, returns True if the system percolates, and False
# otherwise.
def percolates(isFull):
    n = len(isFull)
    for j in range(n):
        if isFull[n - 1][j]:
            return True
    return False


# Given the open and full sites of a percolation system, updates the full sites by marking every
# site of that system that is open and reachable from site (i, j).
def _flow(isOpen, isFull, i, j):
    n = len(isFull)
    if i < 0 or i >= n or j < 0 or j >= n:
        return
    if not isOpen[i][j] or isFull[i][j]:
        return
    isFull[i][j] = True
    _flow(isOpen, isFull, i + 1, j)
    _flow(isOpen, isFull, i, j + 1)
    _flow(isOpen, isFull, i, j - 1)
    _flow(isOpen, isFull, i - 1, j)


# Unit tests the library.
def _main():
    isOpen = stdarray.readBool2D()
    isFull = flow(isOpen)
    stdarray.write2D(isFull)
    stdio.writeln(percolates(isFull))


if __name__ == "__main__":
    _main()
```

~/workspace/ipp/programs

```
$ python3 percolation.py < ../data/test5.txt
5 5
0 1 1 0 1
0 0 1 1 1
0 0 0 1 1
0 0 0 0 1
0 1 1 1 1
True
$ python3 percolation.py < ../data/test8.txt
8 8
0 0 1 1 1 0 0 0
0 0 0 1 1 1 1 1
0 0 0 0 0 1 1 0
0 0 0 0 0 1 1 1
0 0 0 0 0 1 1 0
0 0 0 0 0 0 1 1
0 0 0 0 1 1 1 1
0 0 0 0 0 1 0 0
True
$ _
```

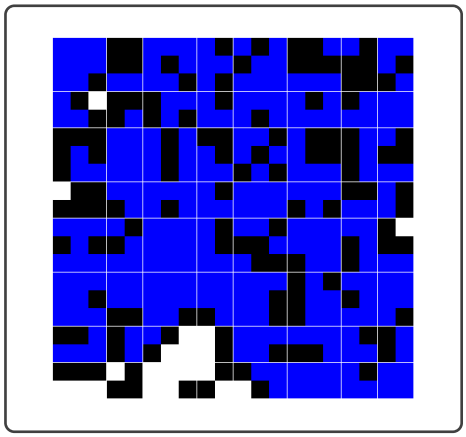
2.4.3 visualize.py

visualize.py

```python
# Accepts n (int), p (float), and trials (int) as command-line arguments; generates an n-by-n
# random percolation system with vacancy probability p; computes the directed percolation flow;
# and draws the result trials times using standard draw.

import percolation
import percolationio
import stddraw
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
    p = float(sys.argv[2])
    trials = int(sys.argv[3])
    for i in range(trials):
        isOpen = percolationio.random(n, p)
        stddraw.clear()
        stddraw.setPenColor(stddraw.BLACK)
        percolationio.draw(isOpen, False)
        stddraw.setPenColor(stddraw.BLUE)
        isFull = percolation.flow(isOpen)
        percolationio.draw(isFull, True)
        stddraw.show(1000)
    stddraw.show()


if __name__ == "__main__":
    main()
```

>_ ~/workspace/ipp/programs

```
$ python3 visualize.py 20 0.65 1
$ _
```



2.4.4 estimate.py

estimate.py

```python
# Accepts n (int), p (float), and trials (int) as command-line arguments; creates trials random
# n-by-n percolation systems with vacancy probability p; determines the fraction of them that
# percolates; and writes that fraction to standard output.

import percolation
import percolationio
import stdio
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
```

```
    p = float(sys.argv[2])
    trials = int(sys.argv[3])
    stdio.writeln(evaluate(n, p, trials))


# Generates a random n-by-n percolation system with vacancy probability p and determines if the
# system percolates. Repeats trials times. Returns an estimate of the empirical percolation
# probability of such systems.
def evaluate(n, p, trials):
    count = 0
    for i in range(trials):
        isOpen = percolationio.random(n, p)
        isFull = percolation.flow(isOpen)
        if percolation.percolates(isFull):
            count += 1
    return count / trials


if __name__ == "__main__":
    main()
```

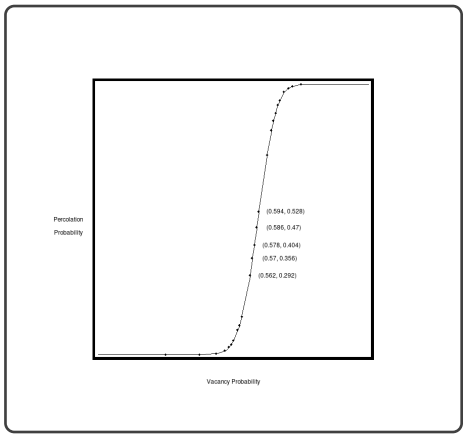
```
>_ ~/workspace/ipp/programs
$ python3 estimate.py 20 0.55 100
0.22
$ python3 estimate.py 20 0.65 100
0.84
$ python3 estimate.py 40 0.55 100
0.12
$ python3 estimate.py 40 0.65 100
0.97
$ _
```

2.4.5 percolationplot.py

```
percolationplot.py
# Accepts n (int) as command-line argument; and draws using standard draw a plot of the percolation
# probability (experimental observation) against the vacancy probability (control variable).

import estimate
import stddraw
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
    stddraw.setCanvasSize(750, 750)
    stddraw.setXscale(-0.2, 1.2)
    stddraw.setYscale(-0.2, 1.2)
    stddraw.setPenRadius(0.0)
    stddraw.square(0.5, 0.5, 0.52)
    stddraw.setPenColor(stddraw.WHITE)
    stddraw.filledSquare(0.5, 0.5, 0.51)
    stddraw.setPenColor(stddraw.BLACK)
    stddraw.text(0.5, -0.1, "Vacancy Probability")
    stddraw.text(-0.11, 0.5, "Percolation")
    stddraw.text(-0.11, 0.45, "Probability")
    _curve(n, 0.0, 0.0, 1.0, 1.0)
    stddraw.show()


# Plots the percolation curve (percolation probability vs vacancy probability) recursively.
def _curve(n, x0, y0, x1, y1, trials=10000, gap=0.01, error=0.0025):
    xm = (x0 + x1) / 2
    ym = (y0 + y1) / 2
    fxm = estimate.evaluate(n, xm, trials)
    if x1 - x0 < gap or abs(ym - fxm) < error:
        stddraw.line(x0, y0, x1, y1)
        stddraw.show(0)
        return
    _curve(n, x0, y0, xm, fxm)
    stddraw.filledCircle(xm, fxm, 0.005)
    if 0.55 < xm < 0.6:
        xyLabel = "(" + str(round(xm, 3)) + ", " + str(round(fxm, 3)) + ")"
        stddraw.text(xm + 0.1, fxm, xyLabel)
    stddraw.show(0)
```

```
    _curve(n, xm, fxm, x1, y1)


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 percolationplot.py 20
$ _
```



3 Object-oriented Programming

3.1 Using Data Types

3.1.1 potentialgene.py

potentialgene.py

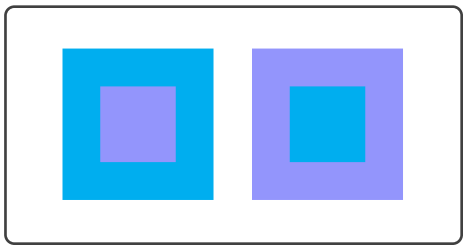
```
# Accepts dna (str) as command-line argument; and writes to standard output whether dna
# corresponds to a potential gene or not.

import stdio
import sys


# Entry point.
def main():
    dna = sys.argv[1]
    stdio.writeln(_isPotentialGene(dna))


# Returns True if dna corresponds to a potential gene, and False otherwise.
def _isPotentialGene(dna):
    ATG, TAA, TAG, TGA = "ATG", "TAA", "TAG", "TGA"
    if len(dna) % 3 != 0:
        return False
    if not dna.startswith(ATG):
        return False
    for i in range(len(dna) - 3):
        if i % 3 == 0:
            codon = dna[i:i + 3]
            if codon == TAA or codon == TAG or codon == TGA:
                return False
    return dna.endswith(TAA) or dna.endswith(TAG) or dna.endswith(TGA)


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 potentialgene.py ATGCGCCTGCGTCTGTACTAG
True
$ python3 potentialgene.py ATGCGCTGCGTCTGTACTAG
False
$ _
```

3.1.2 alberssquares.py

```
alberssquares.py
# Accepts r1 (int), g1 (int), b1 (int), r2 (int), g2 (int), and b2 (int) as command-line
# arguments; and draws using standard draw Albers' squares with colors (r1, g1, b1) and (r2, g2,
# b2).

from color import Color
import stddraw
import sys


# Entry point.
def main():
    r1 = int(sys.argv[1])
    g1 = int(sys.argv[2])
    b1 = int(sys.argv[3])
    r2 = int(sys.argv[4])
    g2 = int(sys.argv[5])
    b2 = int(sys.argv[6])
    c1 = Color(r1, g1, b1)
    c2 = Color(r2, g2, b2)
    stddraw.setCanvasSize(512, 256)
    stddraw.setYscale(0.25, 0.75)
    stddraw.setPenColor(c1)
    stddraw.filledSquare(0.25, 0.5, 0.2)
    stddraw.setPenColor(c2)
    stddraw.filledSquare(0.25, 0.5, 0.1)
    stddraw.setPenColor(c2)
    stddraw.filledSquare(0.75, 0.5, 0.2)
    stddraw.setPenColor(c1)
    stddraw.filledSquare(0.75, 0.5, 0.1)
    stddraw.show()


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 alberssquares.py 0 174 239  147 149 252
$ _
```



3.1.3 luminance.py

```
luminance.py
# A library of color-related functions.

from color import Color
import stdio
import sys


```

```python
# Returns the luminance of the Color c.
def luminance(c):
    r = c.getRed()
    g = c.getGreen()
    b = c.getBlue()
    if r == g and r == b:
        return r
    return 0.299 * r + 0.587 * g + 0.114 * b


# Returns the grayscale equivalent of Color c.
def toGray(c):
    y = int(round(luminance(c)))
    gray = Color(y, y, y)
    return gray


# Returns True if Color c1 is compatible with Color c2, and False otherwise.
def areCompatible(c1, c2):
    return abs(luminance(c1) - luminance(c2)) >= 128.0


# Unit tests the library.
def _main():
    r1 = int(sys.argv[1])
    g1 = int(sys.argv[2])
    b1 = int(sys.argv[3])
    r2 = int(sys.argv[4])
    g2 = int(sys.argv[5])
    b2 = int(sys.argv[6])
    c1 = Color(r1, g1, b1)
    c2 = Color(r2, g2, b2)
    stdio.writeln(str(c1) + " compatible with " + str(c2) + "? " + str(areCompatible(c1, c2)))


if __name__ == "__main__":
    _main()
```

>_ ~/workspace/ipp/programs

```
$ python3 luminance.py 0 0 0 0 0 255
(0, 0, 0) compatible with (0, 0, 255)? False
$ _
```

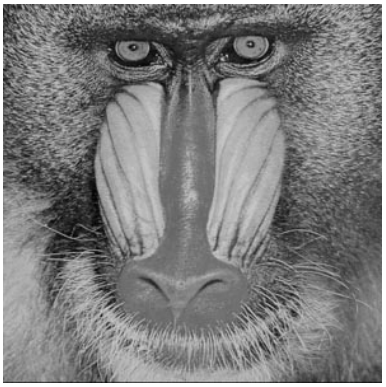
3.1.4 grayscale.py

grayscale.py

```python
# Accepts filename (str) as command-line argument; reads an image from a file with that name; and
# renders a gray scale version of that image.

from picture import Picture
import luminance
import stddraw
import sys


# Entry point.
def main():
    filename = sys.argv[1]
    picture = Picture(filename)
    for col in range(picture.width()):
        for row in range(picture.height()):
            pixel = picture.get(col, row)
            gray = luminance.toGray(pixel)
            picture.set(col, row, gray)
    stddraw.setCanvasSize(picture.width(), picture.height())
    stddraw.picture(picture)
    stddraw.show()


if __name__ == "__main__":
    main()
```

>_ ~/workspace/ipp/programs

```
$ python3 grayscale.py ../data/mandrill.jpg
$ _
```



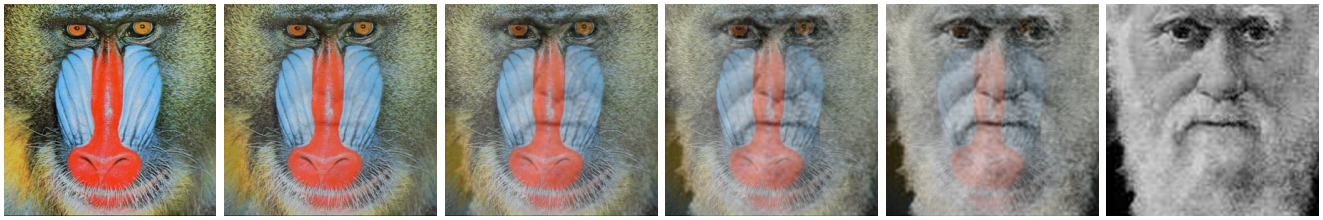
3.1.5 fade.py

fade.py

```python
# Accepts sourceFile (str), targetFile (str), and n (int) as command-line arguments; reads images
# from sourceFile and targetFile; over the course of n frames, gradually replaces the image from
# sourceFile with the image from targetFile; and displays each intermediate image using standard
# draw.

from color import Color
from picture import Picture
import stddraw
import sys


# Entry point.
def main():
    sourceFile = sys.argv[1]
    targetFile = sys.argv[2]
    n = int(sys.argv[3])
    source = Picture(sourceFile)
    target = Picture(targetFile)
    width = source.width()
    height = source.height()
    stddraw.setCanvasSize(width, height)
    picture = Picture(width, height)
    for i in range(n + 1):
        for col in range(width):
            for row in range(height):
                c0 = source.get(col, row)
                cn = target.get(col, row)
                alpha = i / n
                c = _blend(c0, cn, alpha)
                picture.set(col, row, c)
        stddraw.picture(picture)
        stddraw.show(1)
    stddraw.show()


# Returns a new Color object which blends Color objects c1 and c2 using alpha as the blending
# factor.
def _blend(c1, c2, alpha):
    r = (1 - alpha) * c1.getRed() + alpha * c2.getRed()
    g = (1 - alpha) * c1.getGreen() + alpha * c2.getGreen()
    b = (1 - alpha) * c1.getBlue() + alpha * c2.getBlue()
    return Color(int(r), int(g), int(b))


if __name__ == "__main__":
    main()
```

>_ ~/workspace/ipp/programs

```
$ python3 fade.py ../data/mandrill.jpg ../data/darwin.jpg 5
$ _
```



3.1.6 cat.py

cat.py

```python
# Accepts sys.argv[1:n-2] files or web pages as command-line arguments; and copies them to the
# file whose name is accepted as command-line argument sys.argv[n-1].

from instream import InStream
from outstream import OutStream
import sys


# Entry point.
def main():
    n = len(sys.argv)
    outStream = OutStream(sys.argv[n - 1])
    for i in range(1, n - 1):
        inStream = InStream(sys.argv[i])
        s = inStream.readAll()
        outStream.write(s)


if __name__ == "__main__":
    main()
```

>_ ~/workspace/ipp/programs

```
$ cat ../data/in1.txt
This is
$ cat ../data/in2.txt
a tiny
test.
$ python3 cat.py ../data/in1.txt ../data/in2.txt out.txt
$ cat out.txt
This is
a tiny
test.
$ _
```

3.1.7 split.py

split.py

```python
# Accepts filename (str) and n (int) as command-line arguments; and splits the file whose name is
# filename.csv, by field, into n files named filename1.txt, filename2.txt, etc.

from instream import InStream
from outstream import OutStream
import stdarray
import sys


# Entry point.
def main():
    filename = sys.argv[1]
    n = int(sys.argv[2])
    outStreams = stdarray.create1D(n, None)
    for i in range(n):
        outStreams[i] = OutStream(filename + str(i + 1) + ".txt")
    inStream = InStream(filename + ".csv")
    while inStream.hasNextLine():
        line = inStream.readLine()
        fields = line.split(",")
```

```
        for i in range(n):
            outStreams[i].writeln(fields[i])


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ head -5 ../data/ip.csv
www.princeton.edu,128.112.128.15
www.cs.princeton.edu,128.112.136.35
www.math.princeton.edu,128.112.18.11
www.cs.harvard.edu,140.247.50.127
www.harvard.edu,128.103.60.24
$ python3 split.py ../data/ip 2
$ head -5 ../data/ip1.txt
www.princeton.edu
www.cs.princeton.edu
www.math.princeton.edu
www.cs.harvard.edu
www.harvard.edu
$ head -5 ../data/ip2.txt
128.112.128.15
128.112.136.35
128.112.18.11
140.247.50.127
128.103.60.24
$ _
```

3.2 Creating Data Types

3.2.1 timeops.py

```
timeops.py
# Accepts n (int) as command-line argument; computes the sum 1^0.5 + 2^0.5 + ... + n^0.5 using
# math.sqrt(x) and math.pow(x) to calculate the square root of x; and writes to standard output a
# comparison of the performance characteristics of the two functions.

from stopwatch import Stopwatch
import math
import stdio
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
    watch1 = Stopwatch()
    total = 0.0
    for i in range(1, n + 1):
        total += math.sqrt(i)
    time1 = watch1.elapsedTime()
    watch2 = Stopwatch()
    total = 0.0
    for i in range(1, n + 1):
        total += math.pow(i, 0.5)
    time2 = watch2.elapsedTime()
    stdio.writef("math.sqrt() is %.2f times faster than math.pow()\n", time2 / time1)


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 timeops.py 10000000
math.sqrt() is 2.05 times faster than math.pow()
$ _
```

3.2.2 stopwatch.py

stopwatch.py

```python
# An immutable data type to measure the running (wall clock) time of a program.

import stdio
import sys
import time


class Stopwatch:
    # Constructs a new stopwatch.
    def __init__(self):
        self.creationTime = time.time()  # creation time of stopwatch

    # Returns the elapsed time (in seconds) since creation.
    def elapsedTime(self):
        return time.time() - self.creationTime

    # Returns a string representation of this stopwatch.
    def __str__(self):
        return str(self.elapsedTime())


# Unit tests the data type.
def _main():
    n = int(sys.argv[1])
    watch = Stopwatch()
    primes = 0
    for i in range(2, n + 1):
        j = 2
        while j <= i / j:
            if i % j == 0:
                break
            j += 1
        if j > i / j:
            primes += 1
    t = watch.elapsedTime()
    stdio.writef("pi(%d) = %d computed in %.5f seconds\n", n, primes, t)


if __name__ == "__main__":
    _main()
```

>_ ~/workspace/ipp/programs

```
$ python3 stopwatch.py 1000000
pi(1000000) = 78498 computed in 6.43457 seconds
$ _
```

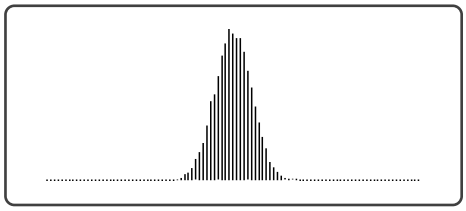
3.2.3 bernoulli.py

bernoulli.py

```python
# Accepts n (int), p (float), and trials (int) as command-line arguments; performs trials
# experiments, each of which counts the number of heads found when a coin with bias p is
# flipped n times; and draws the results using standard draw.

from histogram import Histogram
import stddraw
import stdrandom
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
    p = float(sys.argv[2])
    trials = int(sys.argv[3])
    histogram = Histogram(n + 1)
    for t in range(trials):
        heads = stdrandom.binomial(n, p)
        histogram.addDataPoint(heads)
    stddraw.setCanvasSize(500, 200)
    histogram.draw()
    stddraw.show()
```

```python


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 bernoulli.py 100 0.5 10000
$ _
```

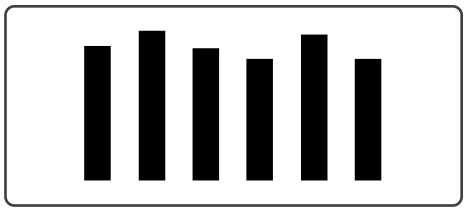


3.2.4 histogram.py

histogram.py

```python
# A data type that represents histograms of the frequency of occurrence of values in [0, n).

import stdarray
import stddraw
import stdrandom
import stdstats
import sys


class Histogram:
    # Constructs a histogram to store frequency of occurrence of values [0, n).
    def __init__(self, n):
        self.freq = stdarray.create1D(n, 0)  # array of frequencies

    # Adds one occurrence of the value i.
    def addDataPoint(self, i):
        self.freq[i] += 1

    # Draws this histogram to standard draw.
    def draw(self):
        stddraw.setYscale(-1, max(self.freq) + 1)
        stdstats.plotBars(self.freq)

    # Returns a string representation of this histogram.
    def __str__(self):
        return str(self.freq)


# Unit tests the data type.
def _main():
    trials = int(sys.argv[1])
    histogram = Histogram(6)
    for t in range(trials):
        roll = stdrandom.uniformInt(0, 6)
        histogram.addDataPoint(roll)
    stddraw.setCanvasSize(500, 200)
    histogram.draw()
    stddraw.show()


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ python3 histogram.py 1000
$ _
```



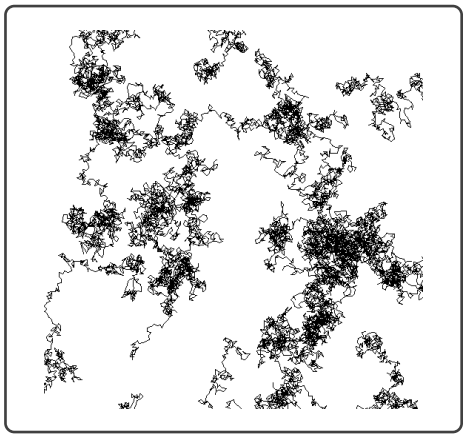
3.2.5 drunks.py

drunks.py

```python
# Accepts n (int), steps (int), and stepSize (float) as command-line arguments; creates n Turtle
# objects; and has the turtles take steps random steps, each of size stepSize.

from turtle import Turtle
import stdarray
import stddraw
import stdrandom
import sys


# Entry point.
def main():
    n = int(sys.argv[1])
    steps = int(sys.argv[2])
    stepSize = float(sys.argv[3])
    turtles = stdarray.create1D(n, None)
    for i in range(n):
        x = stdrandom.uniformFloat(0.0, 1.0)
        y = stdrandom.uniformFloat(0.0, 1.0)
        theta = stdrandom.uniformFloat(0.0, 360.0)
        turtles[i] = Turtle(x, y, theta)
    stddraw.setPenRadius(0.0)
    for i in range(steps):
        for turtle in turtles:
            theta = stdrandom.uniformFloat(0.0, 360.0)
            turtle.turnLeft(theta)
            turtle.goForward(stepSize)
            stddraw.show(0.0)
    stddraw.show()


if __name__ == "__main__":
    main()
```

>_ ~/workspace/ipp/programs

```
$ python3 drunks.py 20 1000 0.01
$ _
```



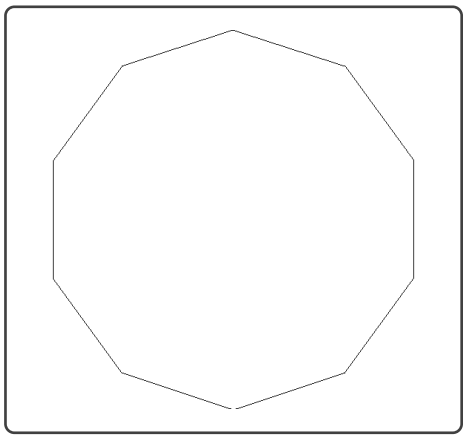
3.2.6 turtle.py

turtle.py

```python
# A data type for turtle graphics using standard draw.

import math
import stddraw
import sys


class Turtle:
    # Constructs a turtle given its coordinates and angle.
    def __init__(self, x, y, theta):
        self.x = x  # x-coordinate of turtle
        self.y = y  # y-coordinate of turtle
        self.theta = theta  # ccw angle (in degrees) of turtle

    # Rotates this turtle by theta in ccw direction.
    def turnLeft(self, theta):
        self.theta += theta

    # Moves this turtle forward by given amount, with the pen down.
    def goForward(self, stepSize):
        xOld = self.x
        yOld = self.y
        self.x += stepSize * math.cos(math.radians(self.theta))
        self.y += stepSize * math.sin(math.radians(self.theta))
        stddraw.line(xOld, yOld, self.x, self.y)

    # Returns a string representation of this turtle.
    def __str__(self):
        return "(" + str(self.x) + ", " + str(self.y) + ", " + str(self.theta) + ")"


# Unit tests the data type.
def _main():
    n = int(sys.argv[1])
    turtle = Turtle(0.5, 0.0, 180.0 / n)
    stepSize = math.sin(math.radians(180.0 / n))
    stddraw.setPenRadius(0.0)
    for i in range(n):
        turtle.goForward(stepSize)
        turtle.turnLeft(360.0 / n)
    stddraw.show()


if __name__ == "__main__":
    _main()
```

>_ ~/workspace/ipp/programs

```
$ python3 turtle.py 10
$ _
```



3.3 Designing Data Types

3.3.1 complex.py

complex.py

```python
# An immutable data type to represent a complex number using cartesian coordinates.

import math
import stdio


class Complex:
    # Constructs a complex number given its cartesian coordinates.
    def __init__(self, re=0.0, im=0.0):
        self.re = re  # the real part
        self.im = im  # the imaginary part

    # Returns the real part of this complex number.
    def re(self):
        return self.re

    # Returns the imaginary part of this complex number.
    def im(self):
        return self.im

    # Returns the conjugate of this complex number.
    def conjugate(self):
        return Complex(self.re, -self.im)

    # Returns the sum of this and the other complex number.
    def __add__(self, other):
        re = self.re + other.re
        im = self.im + other.im
        return Complex(re, im)

    # Returns the product of this and the other complex number.
    def __mul__(self, other):
        re = self.re * other.re - self.im * other.im
        im = self.re * other.im + self.im * other.re
        return Complex(re, im)

    # Returns the magnitude of this complex number.
    def __abs__(self):
        return math.sqrt(self.re * self.re + self.im * self.im)

    # Returns True if this and other denote the same complex number, and False otherwise.
    def __eq__(self, other):
        return self.re == other.re and self.im == other.im

    # Returns a string representation of this complex number.
    def __str__(self):
        SUFFIX = "i"
        if self.im == 0:
            return str(self.re)
        elif self.re == 0:
            return str(self.im) + SUFFIX
        elif self.im < 0:
            return str(self.re) + " - " + str(-self.im) + SUFFIX
        else:
            return str(self.re) + " + " + str(self.im) + SUFFIX


# Unit tests the data type.
def _main():
    a = Complex(5.0, -6.0)
    b = Complex(3.0, 4.0)
    stdio.writeln("a       = " + str(a))
    stdio.writeln("b       = " + str(b))
    stdio.writeln("conj(a) = " + str((a.conjugate())))
    stdio.writeln("a + b   = " + str(a + b))
    stdio.writeln("a * b   = " + str(a * b))
    stdio.writeln("|b|     = " + str(abs(b)))


if __name__ == "__main__":
    _main()
```

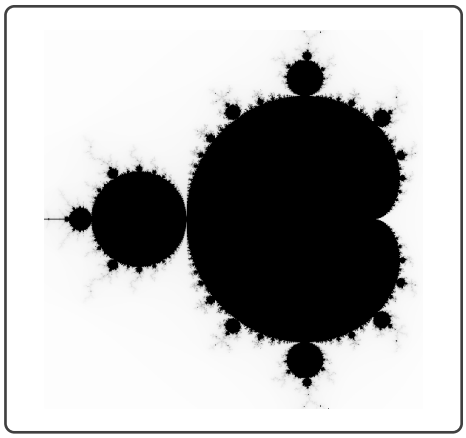
```
>_ ~/workspace/ipp/programs
$ python3 complex.py
a       = 5.0 - 6.0i
b       = 3.0 + 4.0i
conj(a) = 5.0 + 6.0i
a + b   = 8.0 - 2.0i
a * b   = 39.0 + 2.0i
|b|     = 5.0
$ _
```

3.3.2 mandelbrot.py

mandelbrot.py

```python
# Accepts xc (float), yc (float), and size (float) as command-line arguments; and draws using
# standard draw the size-by-size region of the Mandelbrot set, centered at (xc, yc).

from color import Color
from complex import Complex
from picture import Picture
import stddraw
import sys


# Entry point.
def main():
    xc = float(sys.argv[1])
    yc = float(sys.argv[2])
    size = float(sys.argv[3])
    N = 512
    ITERATIONS = 255
    picture = Picture(N, N)
    for col in range(N):
        for row in range(N):
            x0 = xc - size / 2 + size * col / N
            y0 = yc - size / 2 + size * row / N
            z0 = Complex(x0, y0)
            gray = ITERATIONS - _mandel(z0, ITERATIONS)
            color = Color(gray, gray, gray)
            picture.set(col, N - 1 - row, color)
    stddraw.setCanvasSize(N, N)
    stddraw.picture(picture)
    stddraw.show()


# Returns the number of iterations to check if z0 is in the Mandelbrot set.
def _mandel(z0, iterations):
    z = z0
    for i in range(iterations):
        if abs(z) > 2.0:
            return i
        z = z * z + z0
    return iterations


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 mandelbrot.py -0.5 0 2
$ _
```



3.3.3 vector.py

vector.py

```python
# An immutable data type to represent an n-dimensional vector.

import math
import stdarray
import stdio


class Vector:
    # Constructs a vector given its components.
    def __init__(self, a):
        self.n = len(a)  # dimension of vector
        self.coords = a[:]  # defensive copy of array of components

    # Returns the ith component of this vector.
    def __getitem__(self, i):
        return self.coords[i]

    # Returns the sum of this and the other vector.
    def __add__(self, other):
        result = stdarray.create1D(self.n, 0)
        for i in range(self.n):
            result[i] = self.coords[i] + other.coords[i]
        return Vector(result)

    # Returns the difference of this and the other vector.
    def __sub__(self, other):
        result = stdarray.create1D(self.n, 0)
        for i in range(self.n):
            result[i] = self.coords[i] - other.coords[i]
        return Vector(result)

    # Returns the dot product of this and the other vector.
    def dot(self, other):
        result = 0
        for i in range(self.n):
            result += self.coords[i] * other.coords[i]
        return result

    # Returns a scaled (by factor alpha) copy of this vector.
    def scale(self, alpha):
        result = stdarray.create1D(self.n, 0)
        for i in range(self.n):
            result[i] = alpha * self.coords[i]
        return Vector(result)

    # Returns a unit vector in the direction of this vector.
    def direction(self):
        return self.scale(1.0 / abs(self))

    # Returns the magnitude of this vector.
    def __abs__(self):
        return math.sqrt(self.dot(self))

    # Returns the dimension of this vector.
```

```python
    def dimension(self):
        return self.n

    # Returns a string representation of this vector.
    def __str__(self):
        return str(self.coords)


# Unit tests the data type.
def _main():
    xCoords = [1.0, 2.0, 3.0, 4.0]
    yCoords = [5.0, 2.0, 4.0, 1.0]
    x = Vector(xCoords)
    y = Vector(yCoords)
    stdio.writeln("x       = " + str(x))
    stdio.writeln("y       = " + str(y))
    stdio.writeln("x + y   = " + str(x + y))
    stdio.writeln("x - y   = " + str(x - y))
    stdio.writeln("x dot y = " + str(x.dot(y)))
    stdio.writeln("10x     = " + str(x.scale(10.0)))
    stdio.writeln("xhat    = " + str(x.direction()))
    stdio.writeln("|x|     = " + str(abs(x)))
    stdio.writeln("ydim    = " + str(y.dimension()))


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ python3 vector.py
x       = [1.0, 2.0, 3.0, 4.0]
y       = [5.0, 2.0, 4.0, 1.0]
x + y   = [6.0, 4.0, 7.0, 5.0]
x - y   = [-4.0, 0.0, -1.0, 3.0]
x dot y = 25.0
10x     = [10.0, 20.0, 30.0, 40.0]
xhat    = [0.1825741858350553 6, 0.3651483716701107, 0.5477225575051661, 0.7302967433402214]
|x|     = 5.477225575051661
ydim    = 4
$ _
```

3.3.4 sketch.py

```python
sketch.py
# An immutable data type that represents a profile of a string, as a d-dimensional unit vector.

from vector import Vector
import stdarray
import stdio
import sys


class Sketch:
    # Constructs a new sketch which is a profile of text, as a d-dimensional unit vector.
    # Component i of the vector indicates how many k-grams in the text hash to i.
    def __init__(self, text, k, d):
        freq = stdarray.create1D(d, 0)
        for i in range(len(text) - k + 1):
            kgram = text[i:i + k]
            h = hash(kgram)
            freq[abs(h % d)] += 1
        vector = Vector(freq)
        self.sketch = vector.direction()  # string profile as a d-dimensional unit vector

    # Returns the similarity measure between this and the other sketch, as a number between 0 and 1.
    # The value 0 indicates that the sketches are dissimilar, and 1 indicates that they are similar.
    def similarTo(self, other):
        return self.sketch.dot(other.sketch)

    # Returns a string representation of this sketch.
    def __str__(self):
        return str(self.sketch)


# Unit tests the data type.
def _main():
```

```
    k = int(sys.argv[1])
    d = int(sys.argv[2])
    text = stdio.readAll()
    sketch = Sketch(text, k, d)
    stdio.writeln(sketch)


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ python3 sketch.py 2 16 < ../data/genome20.txt
[0.35603449745815596, 0.0, 0.0, 0.4747126632775413, 0.11867816581938533, 0.23735633163877065, 0.0, 0.0, 0.0,
0.4747126632775413, 0.0, 0.0, 0.5933908290969266, 0.0, 0.0, 0.0]
$ _
```

3.3.5 comparedocuments.py

```
comparedocuments.py
# Accepts k (int), d (int), and path (str) as command-line arguments; reads a document list
# from standard input; computes d-dimensional profiles based on k-gram frequencies for all those
# documents under the path directory; and writes to standard output a matrix of similarity measures
# between all pairs of documents.

from instream import InStream
from sketch import Sketch
import stdarray
import stdio
import sys


# Entry point.
def main():
    k = int(sys.argv[1])
    d = int(sys.argv[2])
    path = sys.argv[3]
    filenames = stdio.readAllStrings()
    n = len(filenames)
    sketches = stdarray.create1D(n, None)
    for i in range(n):
        inStream = InStream(path + "/" + filenames[i])
        text = inStream.readAll()
        sketches[i] = Sketch(text, k, d)
    stdio.write("    ")
    for filename in filenames:
        stdio.writef("%8.4s", filename)
    stdio.writeln()
    for i in range(n):
        stdio.writef("%.4s", filenames[i])
        for j in range(n):
            stdio.writef("%8.2f", sketches[i].similarTo(sketches[j]))
        stdio.writeln()


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 comparedocuments.py 5 10000 ../data < ../data/documents.txt
        cons    toms    huck    tale    prej    actg    djia
cons    1.00    0.66    0.60    0.67    0.64    0.11    0.18
toms    0.66    1.00    0.93    0.92    0.88    0.15    0.23
huck    0.60    0.93    1.00    0.84    0.81    0.13    0.21
tale    0.67    0.92    0.84    1.00    0.87    0.14    0.21
prej    0.64    0.88    0.81    0.87    1.00    0.15    0.24
actg    0.11    0.15    0.13    0.14    0.15    1.00    0.12
djia    0.18    0.23    0.21    0.21    0.24    0.12    1.00
$ _
```

3.3.6 counter.py

counter.py

```python
# A comparable data type to represent a counter.

import stdarray
import stdio
import stdrandom
import sys


class Counter:
    # Initializes a new counter with the given id.
    def __init__(self, id):
        self.id = id  # counter name
        self.count = 0  # current value

    # Increments this counter by 1.
    def increment(self):
        self.count += 1

    # Returns the current value of this counter.
    def tally(self):
        return self.count

    # Resets this counter to zero.
    def reset(self):
        self.count = 0

    # Returns True if this counter is less than the other counter by count, and False otherwise.
    def __lt__(self, other):
        return self.count < other.count

    # Returns True if this and the other counter have the same count, and False otherwise.
    def __eq__(self, other):
        return self.count == other.count

    # Returns a string representation of this counter.
    def __str__(self):
        return str(self.count) + " " + self.id


# Unit tests the data type.
def _main():
    n = int(sys.argv[1])
    trials = int(sys.argv[2])
    counters = stdarray.create1D(n, None)
    for i in range(n):
        counters[i] = Counter("counter " + str(i))
    for i in range(trials):
        counters[stdrandom.uniformInt(0, n)].increment()
    for counter in sorted(counters):
        stdio.writeln(counter)


if __name__ == "__main__":
    _main()
```

>_ ~/workspace/ipp/programs

```
$ python3 counter.py 6 10000
1620 counter 0
1629 counter 3
1653 counter 2
1686 counter 1
1686 counter 4
1726 counter 5
$ _
```

3.3.7 country.py

country.py

```python
# A comparable data type that represents a country by its name, capital, and population.

import stdarray
import stdio


class Country:
    # Constructs a country given its name, capital, and population.
    def __init__(self, name, capital, population):
        self.name = name  # name
        self.capital = capital  # capital city
        self.population = population  # population

    # Returns True if this country is less than the other country by name, and False otherwise.
    def __lt__(self, other):
        return self.name < other.name

    # Returns True if this and the other country have the same name, and False otherwise.
    def __eq__(self, other):
        return self.name == other.name

    # Returns a string representation of this country.
    def __str__(self):
        return self.name + " (" + self.capital + "): " + str(self.population)


# Unit tests the data type.
def _main():
    countries = stdarray.create1D(5, None)
    countries[0] = Country("United States", "Washington, D.C.", 329334246)
    countries[1] = Country("Pakistan", "Islamabad", 218719520)
    countries[2] = Country("India", "New Delhi", 1358989650)
    countries[3] = Country("China", "Beijing", 1401463880)
    countries[4] = Country("Indonesia", "Jakarta", 266911900)
    stdio.writeln("Unsorted:")
    for country in countries:
        stdio.writeln(country)
    stdio.writeln()
    stdio.writeln("Sorted by name:")
    for country in sorted(countries):
        stdio.writeln(country)
    stdio.writeln()
    stdio.writeln("Sorted by capital:")
    for country in sorted(countries, key=lambda x: x.capital):
        stdio.writeln(country)
    stdio.writeln()
    stdio.writeln("Sorted by population:")
    for country in sorted(countries, key=lambda x: x.population):
        stdio.writeln(country)
    stdio.writeln()
    stdio.writeln("Reverse sorted by population:")
    for country in sorted(countries, key=lambda x: x.population, reverse=True):
        stdio.writeln(country)


if __name__ == "__main__":
    _main()
```

>_ ~/workspace/ipp/programs

```
$ python3 country.py
Unsorted:
United States (Washington, D.C.): 329334246
Pakistan (Islamabad): 218719520
India (New Delhi): 1358989650
China (Beijing): 1401463880
Indonesia (Jakarta): 266911900

Sorted by name:
China (Beijing): 1401463880
India (New Delhi): 1358989650
Indonesia (Jakarta): 266911900
Pakistan (Islamabad): 218719520
United States (Washington, D.C.): 329334246

Sorted by capital:
```

```
China (Beijing): 1401463880
Pakistan (Islamabad): 218719520
Indonesia (Jakarta): 266911900
India (New Delhi): 1358989650
United States (Washington, D.C.): 329334246

Sorted by population:
Pakistan (Islamabad): 218719520
Indonesia (Jakarta): 266911900
United States (Washington, D.C.): 329334246
India (New Delhi): 1358989650
China (Beijing): 1401463880

Reverse sorted by population:
China (Beijing): 1401463880
India (New Delhi): 1358989650
United States (Washington, D.C.): 329334246
Indonesia (Jakarta): 266911900
Pakistan (Islamabad): 218719520
$ _
```

3.3.8 fibonaccisequence.py

fibonaccisequence.py

```python
# An iterable data type to iterate over the first n numbers from the Fibonacci sequence.

import stdio
import sys


class FibonacciSequence:
    # Constructs a FibonacciSequence object given the length of the sequence.
    def __init__(self, n):
        self.n = n  # length of the sequence

    # Returns an iterator that iterates over the numbers in the sequence.
    def __iter__(self):
        self._a = -1  # previous number in the sequence
        self._b = 1  # current number in the sequence
        self._count = 0  # count of numbers iterated so far
        return self

    # Returns the next number in the sequence if there is one, and raises StopIteration
    # otherwise.
    def __next__(self):
        self._count += 1
        if self._count > self.n:
            raise StopIteration()
        temp = self._a
        self._a = self._b
        self._b += temp
        return self._b

    # Returns a string representation of this object.
    def __str__(self):
        s = ""
        for v in self:
            s += str(v) + " "
        return s.strip()


# Unit tests the data type.
def _main():
    n = int(sys.argv[1])
    for v in FibonacciSequence(n):
        stdio.writeln(v)


if __name__ == "__main__":
    _main()
```

>_ ~/workspace/ipp/programs

```
$ python3 fibonaccisequence.py 10
0
1
1
```

```
2
3
5
8
13
21
34
$ _
```

3.3.9 words.py

words.py

```python
# An iterable data type to iterate over the words in a sentence.

import stdio
import sys


class Words:
    # Constructs a Words object from the given sentence.
    def __init__(self, sentence):
        self.words = sentence.split()  # words in the sentence

    # Returns an iterator to iterate over the words in a sentence.
    def __iter__(self):
        self._current = 0  # index of the current word
        return self

    # Returns the next word in the sentence if there is one, and raises StopIteration otherwise.
    def __next__(self):
        if self._current == len(self.words):
            raise StopIteration
        word = self.words[self._current]
        self._current += 1
        return word

    # Returns a string representation of this object.
    def __str__(self):
        s = ""
        for v in self:
            s += str(v) + " "
        return s.strip()


# Unit tests the data type.
def _main():
    sentence = sys.argv[1]
    words = Words(sentence)
    for word in words:
        stdio.writeln(word)


if __name__ == "__main__":
    _main()
```

>_ ~/workspace/ipp/programs

```
$ python3 words.py "it was the best of times it was the worst of times"
it
was
the
best
of
times
it
was
the
worst
of
times
$ _
```

3.3.10 errorhandling.py

errorhandling.py

```
# Accepts x (float) as command-line argument; and writes to standard output the square root of
# x, reporting an error if x is not specified, is not a float, or is negative.

import math
import stdio
import sys


# Entry point.
def main():
    try:
        x = float(sys.argv[1])
        result = _sqrt(x)
        stdio.writeln(result)
    except IndexError:
        stdio.writeln("x not specified")
    except ValueError:
        stdio.writeln("x must be a float")
    except Exception as e:
        stdio.writeln(e)
    finally:
        stdio.writeln("Done!")


# Returns the square root of x. Raises an Exception if x is negative.
def _sqrt(x):
    if x < 0:
        raise Exception("x must be positive")
    return math.sqrt(x)


if __name__ == "__main__":
    main()
```

>_ ~/workspace/ipp/programs

```
$ python3 errorhandling.py
x not specified
Done!
$ python3 errorhandling.py two
x must be a float
Done!
$ python3 errorhandling.py -2
x must be positive
Done!
$ python3 errorhandling.py 2
1.4142135623730951
Done!
$ _
```

3.4 Case Study: The Music of the Spheres (N-body Problem)

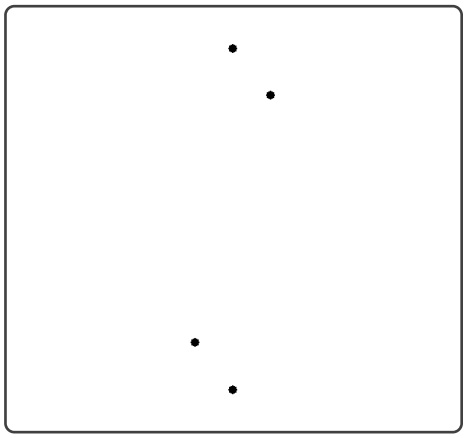
3.4.1 body.py

body.py

```
# A data type to represent an individual body in the n-body system.

from vector import Vector
import stddraw


class Body:
    # Constructs a body given its initial position and velocity, and mass.
    def __init__(self, r, v, mass):
        self.r = r  # current position of the body
        self.v = v  # current velocity of the body
        self.mass = mass  # mass of the body

    # Updates the velocity and position of this body based on a force acting on it for a time
    # period.
    def move(self, f, dt):
        a = f.scale(1 / self.mass)
```

```python
        self.v += a.scale(dt)
        self.r += self.v.scale(dt)

    # Returns the force on this body due to the other body.
    def forceFrom(self, other):
        G = 6.67e-11
        delta = other.r - self.r
        dist = abs(delta)
        magnitude = (G * self.mass * other.mass) / (dist * dist)
        return delta.direction().scale(magnitude)

    # Draws this body on standard draw.
    def draw(self):
        stddraw.setPenRadius(0.0125)
        stddraw.point(self.r[0], self.r[1])

    # Returns a string representation of this object.
    def __str__(self):
        return "(" + str(self.r) + ", " + str(self.v) + ", " + str(self.mass) + ")"


# Unit tests the data type.
def _main():
    stddraw.setXscale(-5.0e10, 5.0e10)
    stddraw.setYscale(-5.0e10, 5.0e10)
    aRCoords = [0.0e00, 4.5e10]
    aVCoords = [1.0e04, 0.0e00]
    bRCoords = [0.0e00, -4.5e10]
    bVCoords = [-1.0e04, 0.0e00]
    a = Body(Vector(aRCoords), Vector(aVCoords), 1.5e30)
    b = Body(Vector(bRCoords), Vector(bVCoords), 1.5e30)
    a.draw()
    b.draw()
    stddraw.show(1000)
    fab = a.forceFrom(b)
    fba = b.forceFrom(a)
    a.move(fab, 1000000)
    b.move(fba, 1000000)
    a.draw()
    b.draw()
    stddraw.show()


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ python3 body.py
$ _
```



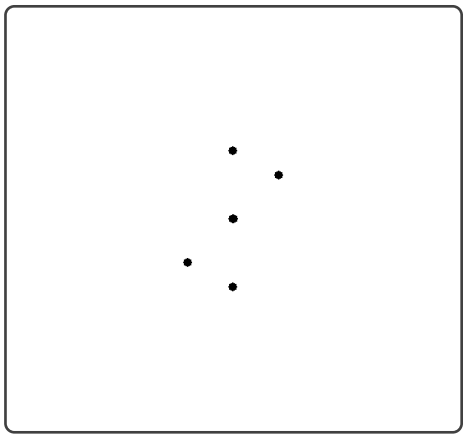
3.4.2 universe.py

universe.py

```python
# A data type to represent a universe.

from body import Body
from instream import InStream
from vector import Vector
import stdarray
import stddraw
import sys


class Universe:
    # Constructs an n-body universe from the given file containing the number (n) of bodies,
    # their initial positions and velocities, and their masses.
    def __init__(self, filename):
        inStream = InStream(filename)
        n = inStream.readInt()
        radius = inStream.readFloat()
        stddraw.setXscale(-radius, +radius)
        stddraw.setYscale(-radius, +radius)
        self.bodies = stdarray.create1D(n, None)  # list of n bodies
        for i in range(n):
            rx = inStream.readFloat()
            ry = inStream.readFloat()
            vx = inStream.readFloat()
            vy = inStream.readFloat()
            mass = inStream.readFloat()
            r = Vector([rx, ry])
            v = Vector([vx, vy])
            self.bodies[i] = Body(r, v, mass)

    # Updates the state of this universe to what it would be after the given time period.
    def increaseTime(self, dt):
        n = len(self.bodies)
        f = stdarray.create1D(n, Vector([0, 0]))
        for i in range(n):
            for j in range(n):
                if i != j:
                    f[i] += self.bodies[i].forceFrom(self.bodies[j])
        for i in range(n):
            self.bodies[i].move(f[i], dt)

    # Draws this universe to standard draw.
    def draw(self):
        for body in self.bodies:
            body.draw()

    # Returns a string representation of this object.
    def __str__(self):
        s = ""
        for body in self.bodies:
            s += str(body) + ", "
        s = "[" + s[:-2] + "]" if len(self.bodies) > 0 else "[]"
        return s


# Unit tests the data type.
def _main():
    filename = sys.argv[1]
    universe = Universe(filename)
    universe.draw()
    stddraw.show(1000)
    universe.increaseTime(1000000)
    universe.draw()
    stddraw.show()


if __name__ == "__main__":
    _main()
```

>_ ~/workspace/ipp/programs

```
$ python3 universe.py ../data/3body.txt
$ _
```



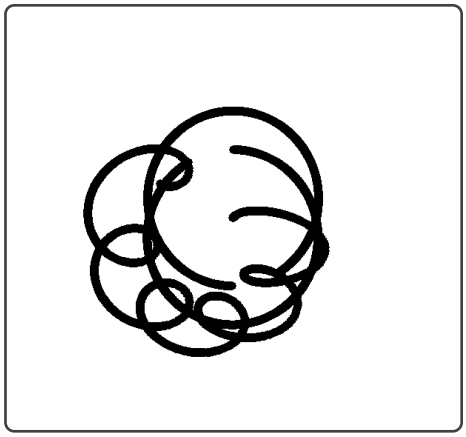
3.4.3 nbody.py

nbody.py

```python
# Accepts filename (String) and dt (float) as command-line arguments; uses the file with that name,
# specifying the number (n) of bodies, their initial positions and velocities, and their masses,
# to create an n-body universe; and simulates the universe using dt as the time step.

from universe import Universe
import stddraw
import sys


# Entry point.
def main():
    filename = sys.argv[1]
    dt = float(sys.argv[2])
    universe = Universe(filename)
    while True:
        universe.increaseTime(dt)
        stddraw.clear()
        universe.draw()
        stddraw.show(10)


if __name__ == "__main__":
    main()
```

>_ ~/workspace/ipp/programs

```
$ python3 nbody.py ../data/3body.txt 10000
$ _
```



4 Data Structures and Algorithms

4.1 Performance

4.1.1 threesum.py

threesum.py

```python
# Accepts a filename as command-line argument; reads the integers in the file; and writes to
# standard output the number of unordered triples (x, y, z) such that x + y + z = 0.

from instream import InStream
import stdio
import sys


# Entry point.
def main():
    inStream = InStream(sys.argv[1])
    a = inStream.readAllInts()
    stdio.writeln(_count(a))


# Returns the number of triples(i, j, k) with i < j < k such that a[i] + a[j] + a[k] == 0.
def _count(a):
    n = len(a)
    count = 0
    for i in range(0, n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if a[i] + a[j] + a[k] == 0:
                    count += 1
    return count


if __name__ == "__main__":
    main()
```

>_ ~/workspace/ipp/programs

```
$ python3 threesum.py ../data/1Kints.txt
70
$ _
```

4.2 Searching and Sorting

4.2.1 linearsearch.py

linearsearch.py

```python
# A library that implements linear search.

from instream import InStream
import stdio
import sys


# Returns the index of key in the array a, or -1.
def indexOf(a, key):
    for i in range(len(a)):
        if key == a[i]:
            return i
    return -1


# Unit tests the library.
def _main():
    inStream = InStream(sys.argv[1])
    whiteList = inStream.readAllInts()
    while not stdio.isEmpty():
        key = stdio.readInt()
        if indexOf(whiteList, key) == -1:
            stdio.writeln(key)


if __name__ == "__main__":
```

```
    _main()
```

```
>_ ~/workspace/ipp/programs
$ /usr/bin/time --format='%e seconds' python3 linearsearch.py ../data/tinyW.txt < ../data/tinyT.txt
50
99
13
0.05 seconds
$ /usr/bin/time --format='%e seconds' python3 linearsearch.py ../data/largeW.txt < ../data/largeT.txt | tail -5
Takes way too long
$ _
```

4.2.2 binarysearch.py

binarysearch.py

```python
# A library that implements binary search.

from instream import InStream
import stdio
import sys


# Returns the index of key in the sorted array a, or -1.
def indexOf(a, key):
    lo = 0
    hi = len(a) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if key < a[mid]:
            hi = mid - 1
        elif key > a[mid]:
            lo = mid + 1
        else:
            return mid
    return -1


# Unit tests the library.
def _main():
    inStream = InStream(sys.argv[1])
    whiteList = inStream.readAllInts()
    whiteList.sort()
    while not stdio.isEmpty():
        key = stdio.readInt()
        if indexOf(whiteList, key) == -1:
            stdio.writeln(key)


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ /usr/bin/time --format='%e seconds' python3 binarysearch.py ../data/tinyW.txt < ../data/tinyT.txt
50
99
13
0.05 seconds
$ /usr/bin/time --format='%e seconds' python3 binarysearch.py ../data/largeW.txt < ../data/largeT.txt | tail -5
29798919
9505145
32449528
38862597
69830567
75.47 seconds
$ _
```

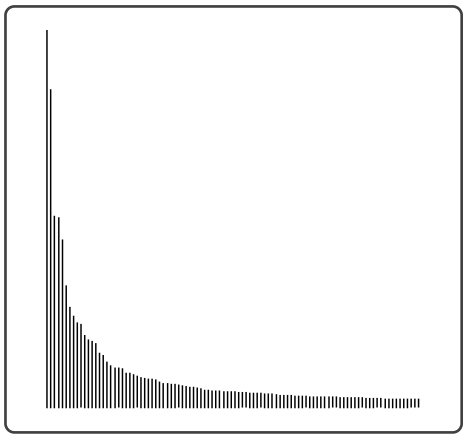
4.2.3 zipf.py

zipf.py

```python
# Accepts k (int) as command-line argument and words from standard input; computes the number of
# times each word appears; writes to standard output the k most frequent words in reverse
# order of their frequencies; and draws using standard draw the corresponding histogram
# demonstrating Zipf's law (ie, the power-law relationship between word frequencies and their
# ranks).

from counter import Counter
from histogram import Histogram
import merge
import stddraw
import stdio
import sys


# Entry point.
def main():
    k = int(sys.argv[1])
    words = stdio.readAllStrings()
    merge.sort(words)
    counters = []
    for i in range(len(words)):
        if (i == 0) or (words[i] != words[i - 1]):
            entry = Counter(words[i])
            counters += [entry]
        counters[len(counters) - 1].increment()
    merge.sort(counters)
    counters.reverse()
    histogram = Histogram(k)
    for i, counter in enumerate(counters[:k]):
        stdio.writeln(counter)
        for j in range(counter.tally()):
            histogram.addDataPoint(i)
    histogram.draw()
    stddraw.show()


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

>_ ~/workspace/ipp/programs

```
$ python3 zipf.py 10 < ../data/tomsawyer.txt
3452 the
2908 and
1758 a
1741 to
1539 of
...
89 other
88 under
87 know
87 ever
87 come
$ _
```



4.2.4 insertion.py

insertion.py

```python
# This library implements insertion sort.

import stdio
import sys


# Sorts the specified array according to the natural ordering of its objects, or according to
# the order induced by key, if one is specified.
def sort(a, key=None):
    n = len(a)
    for i in range(1, n):
        for j in range(i, 0, -1):
            v, w = a[j], a[j - 1]
            if key:
                v, w = key(v), key(w)
            if v >= w:
                break
            _exchange(a, j, j - 1)


# Exchanges two objects in the specified array.
def _exchange(a, i, j):
    temp = a[i]
    a[i] = a[j]
    a[j] = temp


# Unit tests the library.
def _main():
    a = stdio.readAllStrings()
    if sys.argv[1] == "-":
        sort(a, key=lambda x: x.lower())
    elif sys.argv[1] == "+":
        sort(a)
    else:
        raise Exception("Illegal command-line argument")
    for s in a:
        stdio.write(s + " ")
    stdio.writeln()


if __name__ == "__main__":
    _main()
```

>_ ~/workspace/ipp/programs

```
$ python3 insertion.py -
I n s e r t i o n S o r t <ctrl-d>
e I i n n o o r r s S t t
$ python3 insertion.py +
I n s e r t i o n S o r t <ctrl-d>
I S e i n n o o r r s t t
$ _
```

4.2.5 merge.py

merge.py

```python
# This library implements merge sort.

import stdarray
import stdio
import sys


# Sorts the specified array according to the natural ordering of its objects, or according to
# the order induced by key, if one is specified.
def sort(a, key=None):
    aux = stdarray.create1D(len(a), None)
    _sort(a, aux, 0, len(a) - 1, key)


# Sorts the specified array from index lo to index hi according to the natural ordering of its
```

```python
# elements, or according to the order induced by key, if one is specified.
def _sort(a, aux, lo, hi, key=None):
    if hi <= lo:
        return
    mid = lo + (hi - lo) // 2
    _sort(a, aux, lo, mid, key)
    _sort(a, aux, mid + 1, hi, key)
    _merge(a, aux, lo, mid, hi, key)


# Merges the two halves(index lo to index mid and index mid + 1 to index hi) in the specified
# array according to the natural ordering of its elements, or according to the order induced by
# key, if one is specified.
def _merge(a, aux, lo, mid, hi, key=None):
    for k in range(lo, hi + 1):
        aux[k] = a[k]
    i, j = lo, mid + 1
    for k in range(lo, hi + 1):
        if i > mid:
            a[k] = aux[j]
            j += 1
        elif j > hi:
            a[k] = aux[i]
            i += 1
        else:
            v, w = aux[i], aux[j]
            if key:
                v, w = key(v), key(w)
            if w < v:
                a[k] = aux[j]
                j += 1
            else:
                a[k] = aux[i]
                i += 1


# Unit tests the library.
def _main():
    a = stdio.readAllStrings()
    if sys.argv[1] == "-":
        sort(a, key=lambda x: x.lower())
    elif sys.argv[1] == "+":
        sort(a)
    else:
        raise Exception("Illegal command-line argument")
    for s in a:
        stdio.write(s + " ")
    stdio.writeln()


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ python3 merge.py -
M e r g e S o r t
M e e g r S o r t
$ python3 merge.py +
M e r g e S o r t
M S e e g o r r t
$ _
```

4.3 Stacks, Queues, and Symbol Tables

4.3.1 reverse.py

```python
reverse.py
# Accepts a sequence of strings from standard input; and writes the strings in reverse order to
# standard output.

from arraystack import ArrayStack
import stdio


# Entry point.
```

```python
def main():
    stack = ArrayStack()
    while not stdio.isEmpty():
        s = stdio.readString()
        stack.push(s)
    for s in stack:
        stdio.write(s + " ")
    stdio.writeln()


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 reverse.py
b o l t o n <ctrl-d>
n o t l o b
$ _
```

4.3.2 arraystack.py

arraystack.py

```python
# An iterable data type to represent the Last-In-First-Out (LIFO) stack data structure.

import stdio


class ArrayStack:
    # Initializes an empty stack.
    def __init__(self):
        self.a = []  # items in the stack

    # Returns True if this stack is empty, and False otherwise.
    def isEmpty(self):
        return len(self) == 0

    # Returns the number of items in this stack.
    def __len__(self):
        return len(self.a)

    # Adds item to the top of this stack.
    def push(self, item):
        self.a.append(item)

    # Returns the item at the top of this stack.
    def peek(self):
        if self.isEmpty():
            raise Exception("Stack is empty")
        return self.a[-1]

    # Removes and returns the item at the top of this stack.
    def pop(self):
        if self.isEmpty():
            raise Exception("Stack is empty")
        return self.a.pop(-1)

    # Returns an iterator that iterates over the items in this stack.
    def __iter__(self):
        return reversed(self.a)

    # Returns a string representation of this stack.
    def __str__(self):
        s = ""
        for v in self:
            s += str(v) + ", "
        s = "[" + s[:-2] + "]" if len(self) > 0 else "[]"
        return s


# Unit tests the data type.
def _main():
    stack = ArrayStack()
    while not stdio.isEmpty():
        item = stdio.readString()
        if item != "-":
            stack.push(item)
```

```
        elif not stack.isEmpty():
            stdio.write(str(stack.pop()) + " ")
    stdio.writeln()
    stdio.writeln(str(len(stack)) + " items left in the stack")
    stdio.writeln(stack)


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ python3 arraystack.py < ../data/tobe.txt
to be not that or be
2 items left in the stack
[is, to]
$ _
```

4.3.3 kthfromlast.py

kthfromlast.py

```
# Accepts k (int) as command-line argument and a sequence of strings from standard input; and writes
# the kth string from the end to standard output.

from arrayqueue import ArrayQueue
import stdio
import sys


# Entry point.
def main():
    k = int(sys.argv[1])
    queue = ArrayQueue()
    while not stdio.isEmpty():
        s = stdio.readString()
        queue.enqueue(s)
    n = len(queue)
    for i in range(n - k):
        queue.dequeue()
    stdio.writeln(queue.peek())


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 kthfromlast.py 5
she sells sea shells on the sea shore <ctrl-d>
shells
$ _
```

4.3.4 arrayqueue.py

arrayqueue.py

```
# An iterable data type to represent the First-In-First-Out (FIFO) queue data structure.

import stdio


class ArrayQueue:
    # Initializes an empty queue.
    def __init__(self):
        self.a = []  # items in the queue

    # Returns True if this queue is empty, and False otherwise.
    def isEmpty(self):
        return len(self) == 0

    # Returns the number of items in this queue.
    def __len__(self):
        return len(self.a)

```

```
    # Adds item to the end of this queue.
    def enqueue(self, item):
        self.a.append(item)

    # Returns the item at the front of this queue.
    def peek(self):
        if self.isEmpty():
            raise Exception("Queue is empty")
        return self.a[0]

    # Removes and returns the item at the front of this queue.
    def dequeue(self):
        if self.isEmpty():
            raise Exception("Queue is empty")
        return self.a.pop(0)

    # Returns an iterator that iterates over the items in this queue.
    def __iter__(self):
        return iter(self.a)

    # Returns a string representation of this stack.
    def __str__(self):
        s = ""
        for v in self:
            s += str(v) + ", "
        s = "[" + s[:-2] + "]" if len(self) > 0 else "[]"
        return s


# Unit tests the data type.
def _main():
    queue = ArrayQueue()
    while not stdio.isEmpty():
        item = stdio.readString()
        if item != "-":
            queue.enqueue(item)
        elif not queue.isEmpty():
            stdio.write(str(queue.dequeue()) + " ")
    stdio.writeln()
    stdio.writeln(str(len(queue)) + " items left in the queue")
    stdio.writeln(queue)


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ python3 arrayqueue.py < ../data/tobe.txt
to be or not to be
2 items left in the queue
[that, is]
$ _
```

4.3.5 frequencycounter.py

```
frequencycounter.py
# Accepts minLen (int) as command-line argument, and words from standard input; and for the words
# that have at least minLen characters, writes to standard output the total word count, the number
# of distinct words, and the most frequent word.

from symboltable import SymbolTable
import stdio
import sys


# Entry point.
def main():
    minLen = int(sys.argv[1])
    distinct, words = 0, 0
    st = SymbolTable()
    while not stdio.isEmpty():
        word = stdio.readString()
        if len(word) < minLen:
            continue
        words += 1
        if word in st:
```

```
            st[word] += 1
        else:
            st[word] = 1
            distinct += 1
    maxFreq = 0
    maxFreqWord = ""
    for word in st.keys():
        if st[word] > maxFreq:
            maxFreq = st[word]
            maxFreqWord = word
    stdio.writeln("Word count: " + str(words))
    stdio.writeln("Distinct word count: " + str(distinct))
    stdio.writef("Most frequent word: %s (%d repetitions)\n", maxFreqWord, maxFreq)


if __name__ == "__main__":
    main()
```

```
>_ ~/workspace/ipp/programs
$ python3 frequencycounter.py 8 < ../data/tale.txt
Word count: 13525
Distinct word count: 4371
Most frequent word: business (134 repetitions)
$ _
```

4.3.6 symboltable.py

```
symboltable.py
# A data type to represent a symbol table of key-value pairs.

import stdio


class SymbolTable:
    # Constructs an empty symbol table.
    def __init__(self):
        self.st = {}  # dictionary of key-value pairs

    # Returns True if this symbol table is empty, and False otherwise.
    def isEmpty(self):
        return len(self.st) == 0

    # Returns the number of key-value pairs in this symbol table.
    def __len__(self):
        return len(self.st)

    # Returns True if this symbol table contains key, and False otherwise.
    def __contains__(self, key):
        return key in self.st

    # Returns the value associated with key in this symbol table.
    def __getitem__(self, key):
        return self.st[key]

    # Inserts a key-value pair into this symbol table.
    def __setitem__(self, key, val):
        self.st[key] = val

    # Returns the keys in this symbol table, as an iterable object.
    def keys(self):
        return self.st.keys()

    # Returns the values in this symbol table, as an iterable object.
    def values(self):
        return self.st.values()

    # Returns a string representation of this symbol table.
    def __str__(self):
        s = ""
        for key in self.st:
            s += str(key) + ": " + str(self.st[key]) + ", "
        s = "{" + s[:-2] + "}" if len(self) > 0 else "{}"
        return s


# Unit tests the data type.
```

```python
def _main():
    st = SymbolTable()
    st["Gautama"] = "Siddhartha"
    st["Darwin"] = "Charles"
    st["Einstein"] = "Albert"
    stdio.writeln(st["Gautama"])
    stdio.writeln(st["Darwin"])
    stdio.writeln(st["Einstein"])
    if "Einstein" in st:
        stdio.writeln("Einstein found")
    else:
        stdio.writeln("Einstein not found")
    if "Newton" in st:
        stdio.writeln("Newton found")
    else:
        stdio.writeln("Newton not found")
    for key in st.keys():
        stdio.writeln(key + ": " + st[key])
    for value in st.values():
        stdio.writeln(value)


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ python3 symboltable.py
Siddhartha
Charles
Albert
Einstein found
Newton not found
Gautama: Siddhartha
Darwin: Charles
Einstein: Albert
Siddhartha
Charles
Albert
$ _
```

4.4 Case Study: Six Degrees of Separation (Small-world Problem)

4.4.1 graph.py

graph.py

```python
# A data type to represent an undirected symbol graph.

from instream import InStream
from symboltable import SymbolTable
import stdio
import sys


class Graph:
    # Constructs an empty graph, or one from the given file using the specified delimiter.
    def __init__(self, filename=None, delimiter=None):
        self.adj = SymbolTable()  # maps each vertex to its neighbors
        self.e = 0  # number of edges in graph
        if filename:
            inStream = InStream(filename)
            while inStream.hasNextLine():
                line = inStream.readLine()
                names = line.split(delimiter)
                for i in range(1, len(names)):
                    self.addEdge(names[0], names[i])

    # Adds an undirected edge between vertices v and w in this graph.
    def addEdge(self, v, w):
        if not self.hasVertex(v):
            self.adj[v] = set()
        if not self.hasVertex(w):
            self.adj[w] = set()
        if not self.hasEdge(v, w):
            self.adj[v].add(w)
            self.adj[w].add(v)
```

```
            self.e += 1

    # Returns True if v is a vertex in this graph, and False otherwise.
    def hasVertex(self, v):
        return v in self.adj

    # Returns True if v-w is an edge in this graph, and False otherwise.
    def hasEdge(self, v, w):
        return v in self.adj and w in self.adj[v]

    # Returns the number of vertices in this graph.
    def countV(self):
        return len(self.adj)

    # Returns the number of edges in this graph.
    def countE(self):
        return self.e

    # Returns the degree of vertex v in this graph.
    def degree(self, v):
        return len(self.adj[v])

    # Returns the vertices adjacent to vertex v in this graph, as an iterable object.
    def adjacentTo(self, v):
        return iter(self.adj[v])

    # Returns all the vertices in this graph, as an iterable object.
    def vertices(self):
        return self.adj.keys()

    # Returns a string representation of this graph.
    def __str__(self):
        s = ""
        for v in self.vertices():
            s += v + ": "
            for w in self.adjacentTo(v):
                s += w + " "
            s += "\n"
        return s


# Unit tests the data type.
def _main():
    filename = sys.argv[1]
    delimiter = sys.argv[2]
    graph = Graph(filename, delimiter)
    stdio.writeln(graph)


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ python3 graph.py ../data/routes.txt " "
JFK: ORD ATL MCO
MCO: ATL JFK
ORD: ATL HOU DFW PHX JFK DEN
DEN: PHX ORD LAS
HOU: DFW ATL ORD
ATL: ORD JFK MCO HOU
DFW: PHX ORD HOU
PHX: DFW ORD LAX DEN
LAX: PHX LAS
LAS: LAX DEN
$ _
```

4.4.2 pathfinder.py

pathfinder.py

```
# A data type to represent paths within an undirected symbol graph from a fixed source vertex.

from arrayqueue import ArrayQueue
from arraystack import ArrayStack
from graph import Graph
from symboltable import SymbolTable
import stdio
```

```python
import sys


class PathFinder:
    # Constructs a path finder given the graph and source vertex.
    def __init__(self, graph, s):
        self.graph = graph  # the graph
        self.s = s  # the source vertex
        self.distTo = SymbolTable()  # maps a vertex to its distance from source
        self.edgeTo = SymbolTable()  # maps a vertex to previous vertex on path
        queue = ArrayQueue()
        queue.enqueue(s)
        self.distTo[s] = 0
        self.edgeTo[s] = None
        while not queue.isEmpty():
            v = queue.dequeue()
            for w in graph.adjacentTo(v):
                if w not in self.distTo:
                    queue.enqueue(w)
                    self.distTo[w] = 1 + self.distTo[v]
                    self.edgeTo[w] = v

    # Returns the distance of vertex v from the source vertex.
    def distanceTo(self, v):
        return self.distTo[v]

    # Returns True if there is a path to vertex v from the source vertex, and False otherwise.
    def hasPathTo(self, v):
        return v in self.distTo

    # Returns the path to vertex v from the source vertex.
    def pathTo(self, v):
        path = ArrayStack()
        while v is not None:
            path.push(v)
            v = self.edgeTo[v]
        return path

    # Returns a string representation of this object.
    def __str__(self):
        s = ""
        for t in self.graph.vertices():
            if self.hasPathTo(t):
                s += self.s + " -> " + t + ": "
                for v in self.pathTo(t):
                    s += v + " "
                s += "(" + str(self.distanceTo(t)) + ")\n"
        return s.strip()


# Unit tests the data type.
def _main():
    filename = sys.argv[1]
    delimiter = sys.argv[2]
    s = sys.argv[3]
    graph = Graph(filename, delimiter)
    pf = PathFinder(graph, s)
    stdio.writeln(pf)


if __name__ == "__main__":
    _main()
```

```
>_ ~/workspace/ipp/programs
$ python3 pathfinder.py ../data/routes.txt " " JFK
JFK -> JFK: JFK (0)
JFK -> MCO: JFK MCO (1)
JFK -> ORD: JFK ORD (1)
JFK -> DEN: JFK ORD DEN (2)
JFK -> HOU: JFK ORD HOU (2)
JFK -> ATL: JFK ATL (1)
JFK -> DFW: JFK ORD DFW (2)
JFK -> PHX: JFK ORD PHX (2)
JFK -> LAX: JFK ORD PHX LAX (3)
JFK -> LAS: JFK ORD DEN LAS (3)
$ _
```

4.4.3 separation.py

separation.py

```python
#  Accepts filename (str), delimiter (str), and s (str) as command-line arguments; and query
#  vertices from standard input; constructs a graph using filename, delimiter, and s; and
#  for each query vertex, writes to standard output the shortest path between the source vertex s
#  and the query vertex, and the distance between the two.

from graph import Graph
from pathfinder import PathFinder
import stdio
import sys


def main():
    filename = sys.argv[1]
    delimiter = sys.argv[2]
    s = sys.argv[3]
    graph = Graph(filename, delimiter)
    pf = PathFinder(graph, s)
    while stdio.hasNextLine():
        t = stdio.readLine()
        if pf.hasPathTo(t):
            distance = pf.distanceTo(t)
            for v in pf.pathTo(t):
                stdio.writeln("   " + v)
            stdio.writeln("distance: " + str(distance))


if __name__ == "__main__":
    main()
```

>_ ~/workspace/ipp/programs

```
$ python3 separation.py ../data/movies.txt "/" "Bacon, Kevin"
Kidman, Nicole
   Bacon, Kevin
   Wild Things (1998)
   Dillon, Matt (I)
   To Die For (1995)
   Kidman, Nicole
distance: 4
Hanks, Tom
   Bacon, Kevin
   Apollo 13 (1995)
   Hanks, Tom
distance: 2
<ctrl-d>
$ _
```