Contents

Preface

1 Imperative Programming

1.1 Your First Programs .................................................. 5
1.1.1 helloworld.py ................................................. 5
1.1.2 useargument.py ............................................ 5
1.2 Basic Data Types ....................................................... 5
1.2.1 dateformats.py ............................................... 5
1.2.2 sumofsquares.py ........................................... 6
1.2.3 quadratic.py ............................................... 6
1.2.4 leapyear.py ................................................. 6
1.3 Control Flow .......................................................... 7
1.3.1 grade.py ..................................................... 7
1.3.2 flip.py ....................................................... 7
1.3.3 nhellos.py .................................................. 8
1.3.4 powersoftwo.py ........................................... 8
1.3.5 divisorpattern.py ........................................ 8
1.3.6 harmonic.py ............................................... 9
1.3.7 sqrt.py ..................................................... 9
1.3.8 binary.py .................................................. 10
1.3.9 gambler.py ................................................ 10
1.3.10 factors.py ............................................... 11
1.4 Collection Data Types .............................................. 11
1.4.1 sample.py .................................................. 11
1.4.2 couponcollector.py ................................... 12
1.4.3 primesieve.py ........................................... 12
1.4.4 selfavoid.py ............................................... 13
1.5 Input and Output .................................................... 13
1.5.1 randomseq.py ............................................... 13
1.5.2 twentyquestions.py .................................. 14
1.5.3 average.py ................................................ 15
1.5.4 rangefilter.py ............................................ 15
1.5.5 plotfilter.py ............................................. 15
1.5.6 bouncingball.py ......................................... 16
1.5.7 playthattune.py ......................................... 17
1.6 Case Study: What Makes Google Different? (PageRank Algorithm) ............... 17
1.6.1 transition.py ............................................... 17
1.6.2 randomsurfer.py .......................................... 18
1.6.3 markov.py ................................................ 18

2 Procedural Programming .................................................. 19
2.1 Defining Functions .................................................. 19
2.1.1 harmonicredux.py ........................................ 19
2.1.2 couponcollectordedux.py ........................................ 19
2.1.3 playthattunedeluxe.py .................................. 20
2.2 Libraries and Applications ......................................... 21
2.2.1 gaussiantable.py ......................................... 21
2.2.2 gaussian.py ................................................ 21
2.2.3 ifs.py ...................................................... 22
2.2.4 matrix.py .................................................. 23
2.3 Recursion .................................................................. 24
2.3.1 factorial.py ................................................ 24

1 / 68
## 2.4 Case Study: Fermi's Paradox (Percolation Problem)

- 2.4.1 `percolationio.py`
- 2.4.2 `percolation.py`
- 2.4.3 `visualize.py`
- 2.4.4 `estimate.py`
- 2.4.5 `percolationplot.py`

## 3 Object-oriented Programming

### 3.1 Using Data Types
- 3.1.1 `potentialgene.py`
- 3.1.2 `alberssquares.py`
- 3.1.3 `grayscale.py`
- 3.1.4 `fade.py`
- 3.1.5 `cat.py`
- 3.1.6 `split.py`

### 3.2 Defining Data Types
- 3.2.1 `timeops.py`
- 3.2.2 `stopwatch.py`
- 3.2.3 `bernoulli.py`
- 3.2.4 `histogram.py`
- 3.2.5 `drunks.py`
- 3.2.6 `turtle.py`

### 3.3 Design Principles
- 3.3.1 `complex.py`
- 3.3.2 `mandelbrot.py`
- 3.3.3 `vector.py`
- 3.3.4 `sketch.py`
- 3.3.5 `comparerecords.py`
- 3.3.6 `counter.py`
- 3.3.7 `country.py`
- 3.3.8 `fibonacci.py`
- 3.3.9 `words.py`
- 3.3.10 `errorhandling.py`

### 3.4 Case Study: The Music of the Spheres (N-body Problem)
- 3.4.1 `body.py`
- 3.4.2 `universe.py`
- 3.4.3 `nbody.py`

## 4 Algorithms and Data Structures

### 4.1 Analysis of Algorithms
- 4.1.1 `triplesum.py`

### 4.2 Searching and Sorting
- 4.2.1 `linearsort.py`
- 4.2.2 `binarysearch.py`
- 4.2.3 `zipf.py`
- 4.2.4 `insertion.py`
- 4.2.5 `merge.py`

### 4.3 Basic Data Structures
- 4.3.1 `stats.py`
- 4.3.2 `bag.py`
Introduction to Programming in Python

Preface

This document contains listings of the 89 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). Download the file onto your computer and extract it under some folder, say ~/workspace. To open the project, launch PyCharm, and open the folder ~/workspace/ipp/programs. You can run a program on the PyCharm terminal, as follows:

```bash
~/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*. 
1 Imperative Programming

1.1 Your First Programs

1.1.1 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

```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

```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

```python
# sumofsquares.py
# 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)
```

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

1.2.3 quadratic.py

```python
# quadratic.py
# 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))
```

```bash
$ 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

```python
# leapyear.py
# 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)
```

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

6 / 68
1.3 Control Flow

1.3.1 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")
```

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

1.3.2 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)
```

```
$ python3 flip.py
Heads
$ python3 flip.py
Heads
$ python3 flip.py
Tails
$ _
```
1.3.3 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

```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

```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("* ")
```

```
```
1.3.6 harmonic.py

```python
# harmonic.py
# 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)
```

$ python3 harmonic.py 10
2.9289682539682538
$ python3 harmonic.py 1000
7.485470860550343
$ python3 harmonic.py 10000
9.787606036044348
$

1.3.7 sqrt.py

```python
# sqrt.py
# 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)
```

$ python3 sqrt.py 2
1.4142135623730951
$ python3 sqrt.py 10
3.1622776601683795
$ python3 sqrt.py 20
4.472135954999579
1.3.8  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()
```

```
$ python3 sqrt .py 2
1.414213562373095
$ python3 sqrt .py 1000000
1000.0
$ python3 sqrt .py 0.4
0.63245553203386759
$ python3 sqrt .py 1048575
1023.9995117186336
$ python3 sqrt .py 16664444
4082.2106756021303
$ python3 sqrt .py 1e-50
9.999999999999999e-26
$ -
```

1.3.9  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 cash > 0 and cash < goal:
        bets += 1
        cash *= 1 if stdrandom.bernoulli() else -1
    wins += 1 if cash == goal else 0
stdio.writeln(str(100 * wins // trials) + "% wins")
stdio.writeln("Avg # bets: " + str(bets // trials))
```

```
$ python3 binary .py 19
10011
$ python3 binary .py 255
11111111
$ python3 binary .py 512
100000000
$ python3 binary .py 100000000
11011100110101100101001000000000
$ -
```
Introduction to Programming in Python

1.3.10  factors.py

```python
# 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()
```

```
$ 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

```python
# 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()
```

```
$ 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
```

11 / 68
1.4.2 couponcollector.py

```python
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)
```

```bash
$ python3 couponcollector.py 1000
6276
$ python3 couponcollector.py 1000
7038
$ python3 couponcollector.py 1000000
13401736
$ 
```

1.4.3 primesieve.py

```python
import stdarray
import stdio
import sys

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

```bash
$ 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

```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 x > 0 and x < n - 1 and y > 0 and 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")
```

```
$ 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

```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.writeln("%.2f\n", r)
```

```
$ python3 randomseq.py 10 100 200
186.69
102.34
176.05
182.78
161.95
169.34
155.65
```

13 / 68
1.5.2 twentyquestions.py

```python
import stdio
import stdrandom

RANGE = 1000000
secret = stdrandom.uniformInt(1, RANGE + 1)
stdio.writeln("I am thinking of a secret number between 1 and %d
", 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!")
```

```
$ 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

```python
#!/usr/bin/env python

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

from stdio import

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

```bash
$ 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

```python
#!/usr/bin/env 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].

from stdio import
from sys import

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

```bash
$ 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

```python
#!/usr/bin/env python

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

from stddraw import
from stdio import

x0 = float(stdin.readFloat())
y0 = float(stdin.readFloat())
x1 = float(stdin.readFloat())
y1 = float(stdin.readFloat())
stddraw.setXscale(x0, x1)
stddraw.setYscale(y0, y1)
while not stdio.isEmpty():
x = float(stdin.readFloat())
y = float(stdin.readFloat())
stddraw.point(x, y)
stddraw.show()
```
Introduction to Programming in Python

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

1.5.6 bouncingball.py

```
bouncingball.py
# 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)
```
1.5.7  playthattune.py

```python
# 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

```python
# 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
```
1.6.2 randomsurfer.py

```python
def randomsurfer(moves):
    import stdarray, stdio, stdrandom, 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.writeln(hit / moves)
```

1.6.3 markov.py

```python
def markov(moves):
    import stdarray, stdio, sys
    moves = int(sys.argv[1])
    transitionMatrix = stdarray.readFloat2D()
    n = len(transitionMatrix)
    ranks = stdarray.create1D(n, 0.0)
    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.writeln(rank)
```

```bash
$ python3 transition.py < ../data/small.txt
0.29000 0.30000 0.09000 0.25000 0.07000
$ python3 randomsurfer.py 100
0.27320 0.26840 0.14590 0.24550 0.06700
$ python3 transition.py < ../data/small.txt | python3 randomsurfer.py 10000
0.27301 0.26569 0.14621 0.24727 0.06782
$ python3 transition.py < ../data/small.txt | python3 randomsurfer.py 10000000
0.27301 0.26569 0.14621 0.24727 0.06782
$ python3 randomsurfer.py 10
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
```
2 Procedural Programming

2.1 Defining Functions

2.1.1 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()
```

```bash
$ python3 harmonicredux.py 10
2.929963253965343
$ python3 harmonicredux.py 1000
7.485470860550343
$ python3 harmonicredux.py 10000
9.787606036044348
$ 
```

2.1.2 couponcollectorredux.py

```python
# Accepts n (int) as 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 strandom
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
    for i in range(n):
        coupon = strandom.uniform(0, n - 1)
        if coupon not in stdarray.list:
            stdarray.append(coupon)
            collectedCount += 1
            if collectedCount == n:
                return count
        count += 1

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

```bash
$ python3 couponcollectorredux.py 10
2.929963253965343
$ python3 couponcollectorredux.py 1000
7.485470860550343
$ python3 couponcollectorredux.py 10000
9.787606036044348
$ 
```
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()
Introduction to Programming in Python

```
47     c[i] = a[i] * aWeight + b[i] * bWeight
48     return c
49
50 if __name__ == "__main__":
51     main()
```

2.2 Libraries and Applications

2.2.1 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
", score, percentile)

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

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

2.2.2 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
```
# 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 1 / 2 + 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(pdf(x, mu, sigma))
    stdio.writeln(cdf(x, mu, sigma))

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

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

### 2.2.4 matrix.py

```python
# 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
```
for i in range(m):
    for j in range(n):
        c[i][j] = a[i][j] - b[i][j]
return c

# 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, b) = "+str(add(a, b)))
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()
def main():
    n = int(sys.argv[1])
    stdio.writeln(_factorial(n))

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

if __name__ == "__main__":
    main()

# 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()

# 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()

2.3.4 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
    xo = x - lineLength / 2
    xi = x + lineLength / 2
    yo = y - lineLength / 2
    y1 = y + lineLength / 2
    stddraw.line(xo, y, xi, y)
    stddraw.line(xo, y, xi, y1)
```python
2.3.5  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 < 2:
        return n
    return _fibonacci(n - 1) + _fibonacci(n - 2)

if __name__ == "__main__":
    main()
```
2.4 Case Study: Fermi’s Paradox (Percolation Problem)

2.4.1 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()
```

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

```
# A library of percolation functions.

```python
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 or 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()
```

$ python3 percolation.py < ../data/test5.txt
5
0 1 0 1
0 0 1 1
0 0 0 1
0 1 1 1
True
$ python3 percolation.py < ../data/test8.txt
8
0 0 0 0 1 1 0 0
0 0 1 1 1 1 1 1
0 0 0 0 1 1 0 0
0 0 0 0 1 1 1 1
0 0 0 0 0 0 1 1
0 0 0 0 0 1 1 0
0 0 0 0 0 1 0 0
True
$ -
### 2.4.3 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()
```

### 2.4.4 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])
    for i in range(trials):
        isOpen = percolationio.random(n, p)
        stdio.writeln(isOpen)
        stdio.writeln(percolationio.flow(isOpen))
        stdio.writeln(percolationio.draw(isOpen, True))
    stdio.writeln('Fraction that percolates: ' + str(percolationio.percolationRate(isOpen)))
    stdio.writeln('Fraction that percolates: ' + str(percolationio.percolationRate(isOpen)))

if __name__ == '__main__':
    main()
```
Introduction to Programming in Python

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

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

3.1 Using Data Types

3.1.1 potentialgene.py

```python
# 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):
        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()
```

Introduction to Programming in Python
3.1.2 alberssquares.py

```python
# 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()
```

3.1.3 grayscale.py

```python
# grayscale.py

# 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 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 = pixel.toGray()
            picture.set(col, row, gray)
    stddraw.picture(picture)
    stddraw.show()

if __name__ == "__main__":
    main()

$ ~/workspace/ipp/programs
$ python3 grayscale.py ../data/mandrill.jpg
$ 

3.1.4  fade.py

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)
```python
picture.set(col, row, c)
stdraw.picture(picture)
stdraw.show(1000)
stdraw.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()
```

3.1.5 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()
```

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

```bash
$ cat ../data/in1.txt
This is a tiny test.
$ python3 cat.py ../data/in1.txt ../data/in2.txt out.txt
$ cat out.txt
This is a tiny test.
$ 
```
3.1.6 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].writeLine(fields[i])

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

3.2 Defining Data Types

3.2.1 timeops.py

```python
# Accepts n (int) as command-line argument; computes the sum 1\^0.5 + 2\^0.5 + \ldots + 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.writeInt("math.sqrt() is %.2f times faster than math.pow()\n", time2 / time1)

if __name__ == "__main__":
    main()
3.2.3 bernoulli.py

```python
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()

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

3.2.4 histogram.py

```python
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)

# Unit tests the data type.
def _main():
    trials = int(sys.argv[1])
```

$ python3 bernoulli.py 100 0.5 10000
$_
```python
histogram = Histogram(6)
for t in range(trials):
    roll = stdrandom.uniformInt(0, 6)
    histogram.addDataPoint(roll)
stddraw.setCanvasSize(500, 200)
histogram.draw()
stddraw.show()
```

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

3.2.5 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()
```

```bash
$ python3 histogram.py 1000
```

```bash
$ python3 drunks.py 20 1000 0.01
```
3.2.6 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 f'({self._x}, {self._y}, {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()
```

$ python3 turtle.py 10
$ _
3.3 Design Principles

3.3.1 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, x=0.0, y=0.0):
        self._x = x  # the real part
        self._y = y  # the imaginary part

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

    # Returns the sum of this and the other complex number.
    def __add__(self, other):
        x = self._x + other._x
        y = self._y + other._y
        return Complex(x, y)

    # Returns the difference of this and the other complex number.
    def __sub__(self, other):
        x = self._x - other._x
        y = self._y - other._y
        return Complex(x, y)

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

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

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

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

```
else:
    return str(self._x) + " + " + str(self._y) + 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("a * b = " + str(a * b))
stdio.writeln("|b| = " + str(abs(b)))
stdio.writeln("a == b = " + str(a == b))

if __name__ == "__main__":
    _main()

$ 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 = 2.0 - 10.0i
a * b = 39.0 + 2.0i
|b| = 5.0
a == b = False
$

3.3.2 mandelbrot.py

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

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

### 3.3.3 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[i] + other[i]
        return Vector(result)

    # Returns the difference of this and the other vector.
    def __sub__(self, other):
        return self + other.scale(-1)

    # Returns the dot product of this and the other vector.
    def dot(self, other):
        result = 0
        for i in range(self._n):
            result += self[i] * other[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[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.
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()
# Returns a string representation of this sketch.
def __str__(self):
    return str(self._sketch)

# Returns a hash of the given string. Unlike Python’s hash(), this function is deterministic.
def _hash(kgram):
    hash = 0
    for c in kgram:
        hash = 31 * hash + ord(c)
    return hash

# Unit tests the data type.
def _main():
    a = sys.argv[1]
    sketchA = Sketch(a, 2, 3)
    b = sys.argv[2]
    sketchB = Sketch(b, 2, 3)
    stdio.writeln("a = " + str(a))
    stdio.writeln("sketchA = " + str(sketchA))
    stdio.writeln("b = " + str(b))
    stdio.writeln("sketchB = " + str(sketchB))
    stdio.writeln("sketchA.similarTo(sketchB) = " + str(sketchA.similarTo(sketchB)))

if __name__ == "__main__":
    _main()

$ python3 sketch.py ATAGATGCATAGCGCATAGC "to be or not to be that is the question"
a = ATAGATGCATAGCGCATAGC
sketchA = [0.9079593845004517, 0.0, 0.4190581774617469]
b = "to be or not to be that is the question"
sketchB = [0.43685202833051895, 0.43685202833051895, 0.7863336509949341]
sketchA.similarTo(sketchB) = 0.7261634454235303

3.3.5 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.writeln("")
        for filename in filenames:
            stdio.writelnf("%.4s", filename)
        stdio.writeln()
    for i in range(n):
        for j in range(n):
            stdio.writelnf("%8.2f", sketches[i].similarTo(sketches[j]))
    stdio.writeln()
3.3.6 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.tally() < other.tally()

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

    # Returns a string representation of this counter.
    def __str__(self):
        return str(self.tally()) + ' ' + 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):
        for counter in sorted(counters):
            stdio.writeln(counter)

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

3.3.6 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.tally() < other.tally()

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

    # Returns a string representation of this counter.
    def __str__(self):
        return str(self.tally()) + ' ' + 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):
        for counter in sorted(counters):
            stdio.writeln(counter)

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

```
3.3.7 country.py

```python
import stdarray
import stdio

class Country:
    # Constructs a country given its name, capital, and population.
    def __init__(self, name, capital, population):
        self._name = name
        self._capital = capital
        self._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("Brazil", "Brasilia", 218689757)
    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)
    stdio.writeln()

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

Unsorted:
Brazil (Brasilia): 218689757
3.3.8 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 a string representation of this object.
    def __str__(self):
        s = ''
        for v in self:
            s += str(v) + ' '  # previous number in the sequence
        return s.strip()

    # Returns an iterator that iterates over the numbers in the Fibonacci sequence.
    def __iter__(self):
        return FibonacciSequence.FibonacciIterator(self._n)

class FibonacciIterator:
    def __init__(self, n):
        self._n = n  # length of the sequence
        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

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

Russia (Moscow): 141698923
India (New Delhi): 1425775850
China (Beijing): 1409670000
South Africa (Cape Town): 58048332

Sorted by name:
Brazil (Brasilia): 218689757
China (Beijing): 1409670000
India (New Delhi): 1425775850
Russia (Moscow): 141698923
South Africa (Cape Town): 58048332

Sorted by capital:
China (Beijing): 1409670000
Brazil (Brasilia): 218689757
South Africa (Cape Town): 58048332
Russia (Moscow): 141698923
India (New Delhi): 1425775850

Sorted by population:
South Africa (Cape Town): 58048332
Russia (Moscow): 141698923
Brazil (Brasilia): 218689757
China (Beijing): 1409670000
India (New Delhi): 1425775850

Reverse sorted by population:
India (New Delhi): 1425775850
China (Beijing): 1409670000
Brazil (Brasilia): 218689757
Russia (Moscow): 141698923
South Africa (Cape Town): 58048332

$ _
# Unit tests the data type.
def _main():
    n = int(sys.argv[1])
    fib = FibonacciSequence(n)
    stdio.writeln(fib)
if __name__ == "__main__":
    _main()

3.3.9 words.py

# 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._sentence = sentence # the sentence

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

    # Returns an iterator to iterate over the words in a sentence.
    def __iter__(self):
        return WordsIterator(self._sentence)

# An iterator that iterates over the words in a sentence.
class WordsIterator:
    # Constructs a WordsIterator object given the sentence.
    def __init__(self, sentence):
        self._words = sentence.split() # words in the sentence
        self._current = 0 # index of the current word

    # 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

if __name__ == "__main__":
    _main()
3.3.10 errorhandling.py

```python
# 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 non-negative")
    return math.sqrt(x)

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

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

3.4.1 body.py

```python
# 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)
```
# 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()
3.4.2 universe.py

```python
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()
```

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

52 / 68
3.4.3 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]
dx = float(sys.argv[2])
universe = Universe(filename)
while True:
    universe.increaseTime(dx)
    stddraw.clear()
    universe.draw()
    stddraw.show(10)

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

```bash
~/workspace/ipp/programs
$ python3 nbody.py ../data/3body.txt 10000
$ \
```

4 Algorithms and Data Structures

4.1 Analysis of Algorithms

4.1.1 triplesum.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.readInts()
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()
```

4.2 Searching and Sorting

4.2.1 linearsearch.py

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

from instream import InStream
import stdio
import sys

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

# Unit tests the library.
def _main():
inStream = InStream(sys.argv[1])
whiteList = inStream.readInts()
```

54 / 68
Introduction to Programming in Python

```python
while not stdio.isEmpty():
    key = stdio.readInt()
    if indexOf(whiteList, key) == -1:
        stdio.writeln(key)

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

4.2.2 binarysearch.py

```python
# A library that implements binary search.
from instream import InStream
import stdio
import sys

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

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

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

```bash
$ /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
$ _
```

```bash
$ /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 9508145 32448528 38862597 69830567
75.47 seconds
$ _
```
4.2.3 zipf.py

# 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 (i.e., 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)
    histogram = Histogram(k)
    for i in range(k):
        counter = counters[len(counters) - i - 1]
        stdio.writeln(counter)
        for j in range(counter.tally()):
            histogram.addDataPoint(i)
            histogram.draw()
    stddraw.show()

if __name__ == '__main__':
    main()
4.2.4 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=lambda x: x):
    n = len(a)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if key(a[j]) < key(a[j - 1]):
                _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] == "String":
        sort(a)
    elif sys.argv[1] == "string":
        sort(a, key=lambda x: x.lower())
    else:
        raise Exception("Illegal command-line argument")
    for s in a:
        stdio.write(s + " ")
        stdio.writeln()

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

4.2.5 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=lambda x: x):
    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
# elements, or according to the order induced by key, if one is specified.
def _sort(a, aux, lo, hi, key):
```
if hi <= lo:
    return
mid = lo + (hi - lo) // 2
_sorted(a, aux, lo, mid, key)
_sorted(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):
    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
        elif key(aux[j]) < key(aux[i]):
            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] == "String":
    sort(a)
elif sys.argv[1] == "string":
    sort(a, key=lambda x: x.lower())
else:
    raise Exception("Illegal command-line argument")
for s in a:
    stdio.write(s + " ")
stdio.writeln()
if __name__ == "__main__":
    _main()

4.3 Basic Data Structures

4.3.1 stats.py

```python
if stats.py
    # Accepts a sequence of floats from standard input; and writes their mean and standard deviation
    # to standard output.
    from bag import Bag
    import math
    import stdio

    # Entry point.
    def main():
        bag = Bag()
        while not stdio.isEmpty():
```
```python
import stdio
import math

class Bag:
    def __init__(self):
        self._first = None
        self._n = 0

    def isEmpty(self):
        return self._n == 0

    def __len__(self):
        return self._n

    def add(self, item):
        oldFirst = self._first
        self._first = Bag.Node()
        self._first._item = item
        self._first._next = oldFirst
        self._n += 1

    def __iter__(self):
        return BagIterator(self._first)

class Node:
    def __init__(self):
        self._item = None
        self._next = None

class BagIterator:
    def __init__(self, first):
        self._current = first

    def __next__(self):
        if self._current == None:
            raise StopIteration

    def __iter__(self, self):
        return BagIterator(self._first)

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

```
```python
49  item = self._current._item
50  self._current = self._current._next
51  return item
52
53 # Unit tests the data type.
54 def _main():
55  bag = Bag()
56  while not stdio.isEmpty():
57    bag.add(stdio.readString())
58    stdio.writeln(str(len(bag)) + " items in the bag")
59    for s in bag:
60      stdio.write(s + " ")
61    stdio.writeln()
62
63 if __name__ == "__main__":
   _main()
```

```
4.3.3 reverse.py

```bash
$ python3 bag.py < ../data/tobe.txt
14 items in the bag
[is, -, -, -, that, -, -, be, -, to, not, or, be, to]
$ _
```

```
4.3.3 reverse.py

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

from stack import Stack
import stdio

# Entry point.
def main():
  stack = Stack()
  while not stdio.isEmpty():
    stack.push(stdio.readString())
  for s in stack:
    stdio.write(s + " ")
  stdio.writeln()

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

```
4.3.4 stack.py

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

import stdio

class Stack:
  # Initializes an empty stack.
  def __init__(self):
    self._first = None # top of the stack
    self._n = 0 # number of 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 self._n

# Adds item to the top of this stack.
def push(self, item):
    oldFirst = self._first
    self._first = Stack.Node()
    self._first._item = item
    self._first._next = oldFirst
    self._n += 1

# Returns the item at the top of this stack.
def peek(self):
    if self.isEmpty():
        raise Exception("Stack is empty")
    return self._first._item

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

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

# A data type representing a linked-list of nodes. Each node contains an item and a reference
# to the next node in the list.
class Node:
    def __init__(self):
        self._item = None
        self._next = None

# An iterator that iterates over the items in a stack.
class StackIterator:
    # Constructs an iterator.
    def __init__(self, first):
        self._current = first

    # Returns the next item in the stack if there is one, and raises StopIteration otherwise.
    def __next__(self):
        if self._current == None:
            raise StopIteration
        item = self._current._item
        self._current = self._current._next
        return item

# Unit tests the data type.
def _main():
    stack = Stack()
    while not stdio.isEmpty():
        item = stdio.readString()
        if item != "-":
            stack.push(item)
        elif not stack.isEmpty():
            stdio.writeln(str(stack.peek()) + " ")
    stack.pop()

    stdio.writeln(str(len(stack)) + " items left in the stack")
    for s in stack:
        stdio.write(s + " ")

if __name__ == "__main__":
    _main()

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

```python
# 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 queue import Queue
import stdio
import sys

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

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

4.3.6 queue.py

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

import stdio

class Queue:
    # Initializes an empty queue.
def __init__(self):
        self._first = None  # front of the queue
        self._last = None   # back of the queue
        self._n = 0         # number of 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 self._n

    # Adds item to the end of this queue.
def enqueue(self, item):
        oldLast = self._last
        self._last = Queue.Node()
        self._last._item = item
        if self.isEmpty():
            self._first = self._last
        else:
            oldLast._next = self._last
            self._n += 1

    # Returns the item at the front of this queue.
def peek(self):
        if self.isEmpty():
            raise Exception("Queue is empty")
```
```python
    return self._first._item

    # Removes and returns the item at the front of this queue.
    def dequeue(self):
        if self.isEmpty():
            raise Exception("Queue is empty")
        item = self._first._item
        self._first = self._first._next
        self._n -= 1
        if self.isEmpty():
            self._last = self._first
        return item

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

    # A data type representing a linked-list of nodes. Each node contains an item and a reference
    # to the next node in the list.
    class Node:
        def __init__(self):
            self._item = None
            self._next = None

    # An iterator that iterates over the items in a queue.
    class QueueIterator:
        # Constructs an iterator.
        def __init__(self, first):
            self._current = first

        # Returns the next item in the queue if there is one, and raises StopIteration otherwise.
        def __next__(self):
            if self._current == None:
                raise StopIteration
            item = self._current._item
            self._current = self._current._next
            return item

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

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

4.3.7 frequencycounter.py

```python
# 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.writeln("Most frequent word: %s (%d repetitions)
        
if __name__ == "__main__":
    main()
if key not in self:
    self._n += 1
    self._st[key] = value

# Deletes key and the associated value from this symbol table.
def _delitem_(self, key):
    if key == None:
        raise Exception("key is None")
    self._st.pop(key)

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

# Unit tests the data type.
def _main_():
    st = SymbolTable()
    sti.write(lr"Filling st with characters from the English alphabet ...")
    for i, c in enumerate("abcdefghijklmnopqrstuvwxyz"):
        st[c] = i + 1
    sti.writeln(st)
    sti.writeln(lr"Deleting vowels from st ...")
    for c in "aeiou":
        del(st[c])
    for c in st.keys():
        sti.writeln(c + ": " + str(st[c]))

if __name__ == "__main__":
    _main_()
```python
# 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 self._adj[v]

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

# Unit tests the data type.
def _main():
    filename = sys.argv[1]
    delimiter = sys.argv[2]
    v = sys.argv[3]
    graph = Graph(filename, delimiter)
    stdio.writeln("V: " + str(graph.countV()))
    stdio.writeln("E: " + str(graph.countE()))
    stdio.writeln(" adj (" + v + ":")
    for w in graph.adjacentTo(v):
        stdio.writeln(w + " ")
    stdio.writeln()

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

### 4.4.2 pathfinder.py

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

from graph import Graph
from queue import Queue
from stack import Stack
from symboltable import SymbolTable
import stdio
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 = Queue()
        queue.enqueue(s)
        self._distTo[s] = 0
        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] = self._distTo[v] + 1
                    self._edgeTo[w] = v
```

```bash
$ python3 graph.py ../data/routes.txt "ORD"
V: 10
E: 17
adj(ORD): ATL JFK PHX DFW HOU DEN
$ _
```
# 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 = Stack()
    while v != 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 + ': ' + 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()
```python
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))
    else:
        stdio.writeln("" + s + "" is not connected to " + t + ":")

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

```
$ python3 separation.py ../data/movies.txt /* "Bacon, Kevin"
Kidman, Nicole
Bacon, Kevin
Grier, David Alan
Bewitched (2005)
Kidman, Nicole
distance: 4
Hanks, Tom
Bacon, Kevin
Apollo 13 (1995)
Hanks, Tom
distance: 2
Einstein, Albert
"Bacon, Kevin" is not connected to "Einstein, Albert"
<ctrl-d>
$ _
```