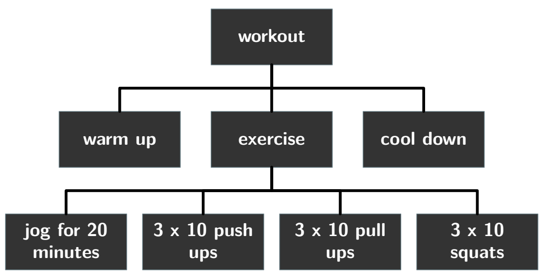
Selected Exercise Solutions

Section 1.1

1.1.2 (a) Quantities of ingredients are missing from the input. The size of the pan and the number of brownies desired are missing from the output.

Section 1.2

1.2.1 (a)



1.2.2 (a) This can be decomposed into 3 subproblems:

1. Find the area of a circle with radius 4.
2. Find the area of a circle with radius 2.
3. Subtract the area of the smaller circle from the area of the larger one.

1.2.4

Trace input: $r = 7$

Step	Line	*volume*	Notes
1	1	$1.\bar{3}$	$volume \leftarrow 4 \div 3 = 1.\bar{3}$
2	2	$4.18\bar{6}$	$volume \leftarrow volume \times \pi = 1.\bar{3} \times 3.14 = 4.18\bar{6}$
3	3	″	*volume* unaffected; execute line 4 three times
4	4	$29.30\bar{6}$	$volume \leftarrow volume \times r = 4.18\bar{6} \times 7 = 29.30\bar{6}$
5	4	$205.14\bar{6}$	$volume \leftarrow volume \times r = 29.30\bar{6} \times 7 = 205.14\bar{6}$
6	4	$1436.02\bar{6}$	$volume \leftarrow volume \times r = 205.14\bar{6} \times 7 = 1436.02\bar{6}$

Output: $volume = 1436.02\bar{6}$

1.2.7

Trace input: *votes* = [John, Laura, Laura, John, Laura]

Step	Line	*laura*	*john*	*winner*	Notes
1	1	0	–	–	*laura* set to 0
2	2	0	0	–	*john* set to 0
3	4	0	0	–	first entry is not Laura, so no changes
4	5	0	1	–	first entry is John, so add 1 to *john*
5	4	1	1	–	second entry is Laura, so add 1 to *laura*
6	5	1	1	–	skip because second entry is Laura
7	4	2	1	–	third entry is Laura, so add 1 to *laura*
8	5	2	1	–	skip because third entry was Laura
9	4	2	1	–	fourth entry is not Laura, so no changes
10	5	2	2	–	fourth entry is John, so add 1 to *john*
11	4	3	2	–	fifth entry is Laura, so add 1 to *laura*
12	5	3	2	–	skip because fifth entry was Laura
13	6	3	2	Laura	*laura* > *john* so *winner* is Laura
14	7	3	2	–	skip because *laura* > *john*

Output: Laura

1.2.9

Algorithm SPHERE VOLUME

Input: the radius r of the sphere

$volume \leftarrow r \times r \times r$
$volume \leftarrow volume \times \pi$
$volume \leftarrow volume \times 4$
$volume \leftarrow volume \div 3$

Output: *volume*

1.2.14

Algorithm WORD COUNT

Input: a *text*

count ← 0
repeat once for each *character* in the *text*:
    if the *character* is a space, add one to *count*

Output: *count*

1.2.16

Algorithm Average Words Per Sentence

Input: a *text*

```
words ← Word Count (text)
sentences ← Sentence Count (text)
average words ← words ÷ sentences
```

Output: *average words*

Section 1.3

1.3.1 `8 / 2 * (2 + 2)`

1.3.2 `838 * 5280 * 12`

1.3.7 (a)

```
temperature = -3
windSpeed = 13
windChill = 13.12 + 0.6215 * temperature \
          + (0.3965 * temperature - 11.37) \
          * windChill ** 0.16
print(windChill)
```

(b) It does not change because you need to recompute the wind chill by repeating the assignment statement:

```
windChill = 13.12 + 0.6215 * temperature \
          + (0.3965 * temperature - 11.37) \
          * windChill ** 0.16
print(windChill)
```

1.3.8 The value of `left` is overwritten by the value of `right` in the first statement. So the second statement does nothing useful. A correct implementation is:

```
temp = left
left = right
right = temp
```

1.3.13 (a) They give different values because the types of the first operands are different. In the first case, both `18` and `10` are integers, so the `*` operator is interpreted as multiplication. In the second case, `'18'` is a string, so the `*` operator is interpreted to be the string repetition operator.

(b) `print('* ' * 20)`

(c) `print('*\n ' * 20)`

(d) `print(('*' * 20 + '\n ') * 20)`

1.3.14

```
temperature = -3
windSpeed = 13
windChill = 13.12 + 0.6215 * temperature \
          + (0.3965 * temperature - 11.37) \
          * windSpeed ** 0.16
windChill = round(windChill)
```

1.3.16
```
number = number - int(number)
```

1.3.19
```
>>> radius = float(input('Radius of your circle? '))
>>> area = 3.14159 * radius * radius
>>> print('The area of your circle is ' + str(area) + '.')
```

1.3.21
```
text = input('Temperature: ')
temperature = float(text)
text = input('Wind speed: ')
windSpeed = float(text)
windChill = 13.12 + 0.6215 * temperature \
          + (0.3965 * temperature - 11.37) \
          * windSpeed ** 0.16
windChill = round(windChill)
print('The wind chill is', windChill, 'degrees Celsius.')
```

1.3.23
```
sum = 0
number = float(input('Number 1: '))
sum = sum + number
print('The current sum is ' + str(sum) + '.')
number = float(input('Number 2: '))
sum = sum + number
print('The current sum is ' + str(sum) + '.')
number = float(input('Number 3: '))
sum = sum + number
print('The final sum is ' + str(sum) + '.')
```

Section 1.4

1.4.3 (a) Some common inputs are $r = 10$, 20, etc. A boundary input is $r = 0$. Disallowed inputs are any negative values of r.

(b) Some common inputs are lists like [Laura, Laura, John, Laura] and [John, John, Laura, Laura, John]. Boundary inputs would be an empty list and lists containing only Laura or only John, including the single-item lists [Laura] and [John]. Any list that contains items other than these two names should be disallowed.

1.4.4 (a) The algorithm has 9 elementary steps. There are five arithmetic operations, counting the two multiplications in the first line, and four assignments. Therefore, it is a constant-time algorithm.

(b) Lines 1 and 2 are each one elementary step. Lines 6 and 7 are together two elementary steps because there is one condition to be tested and exactly one of the two assignments is actually executed. The loop iterates once for each entry in *votes*. Like lines 6–7, each iteration is 2 elementary steps. If there are n entries in *votes*, then this is a total of $2n + 4$ elementary steps. Therefore, it is a linear-time algorithm because the number of elementary steps depends linearly on the length of the input.

Section 2.1

2.1.2

Instance Variable	Description
pairing mode	whether the headphones are in pairing mode
paired device	the device paired with (or none)
volume	current volume, 0–100
muted	whether the headphones are currently muted

Method	Argument	Description
enable pairing	—	place headphones in pairing mode
unpair	—	unpair with current device
volume up	—	increase volume by 1
volume down	—	decrease volume by 1
mute	—	move the turtle to the given position
unmute	—	put the turtle's tail up or down

2.1.6
```
>>> ada.heading()
270.0
```

Turing the turtle 90 degrees to the right is the same as turning it 360 – 90 = 270 degrees to the left.

2.1.8 (a) Both turtles are at position (0,0) and heading 0.0 degrees.

(b)
```
thing2.right(30)
thing2.forward(50)
```

(c) `thing1` is still at position (0,0) and heading 0.0 degrees. `thing2` is now at position (43.30, – 25.00) and heading 330.0 degrees. The values are different because `thing1` and `thing2` are independent objects from the same class.

Section 2.2

2.2.2 The leftmost figure can be drawn as follows:

```
import turtle

george = turtle.Turtle()
george.speed(0)
george.hideturtle()

george.pencolor('red')
george.fillcolor('red')
george.begin_fill()
for count in range(8):
george.forward(100)
george.left(45)
george.end_fill()
```

2.2.5
```
import turtle

george = turtle.Turtle()
for count in range(4):
    george.forward(200)
    george.left(90)
```

2.2.8
```
import turtle

george = turtle.Turtle()
screen = george.getscreen()
screen.setworldcoordinates(0, 0, 9, 9)

george.speed(0)
george.hideturtle()
george.up()

sites = [(1.45, 7.31), (2.99, 7.55), (7.58, 6.29), (2.17, 4.71),
         (1.07, 5.56), (1.52, 3.89), (1.42, 4.55), (1.83, 7.36),
         (2.38, 4.55), (2.77, 5.25), (1.85, 3.4), (1.13, 4.4),
         (1.51, 5.25), (2.0, 7.09), (4.05, 5.15), (2.99, 3.94),
         (2.29, 7.16), (2.55, 7.91), (1.27, 4.85), (4.83, 2.85),
         (2.64, 2.05), (1.04, 4.91), (2.6, 4.43), (4.06, 4.05),
         (2.49, 4.38), (1.63, 3.17), (2.33, 5.47), (2.23, 5.92),
         (1.3, 6.09), (2.27, 6.09), (1.84, 6.05), (1.54, 7.46),
         (1.22, 7.65), (1.27, 3.77), (1.58, 2.44), (1.7, 6.76),
         (2.39, 6.97), (2.82, 4.7), (1.75, 4.52), (2.08, 3.45),
         (6.71, 5.8), (1.02, 4.39), (1.26, 6.71), (1.87, 5.63),
         (2.06, 3.8), (1.69, 4.91), (1.98, 2.5), (2.04, 3.69),
         (3.85, 7.62), (2.24, 2.85), (1.86, 7.68), (1.72, 5.1),
         (4.23, 2.35), (2.44, 2.95), (0.59, 4.23), (1.34, 6.43),
         (1.38, 6.37), (2.68, 2.89), (2.66, 7.85), (5.94, 4.84),
         (1.11, 7.11), (2.77, 5.07), (2.58, 5.53), (1.61, 7.22),
         (2.16, 3.19), (1.47, 6.81), (1.25, 2.03), (2.65, 5.18),
         (1.97, 2.44), (2.8, 3.4), (1.69, 3.83), (7.63, 4.4),
         (1.67, 3.96), (1.62, 3.78), (1.94, 3.0), (2.53, 2.18),
         (8.77, 6.77), (2.46, 7.02), (1.31, 3.77), (2.23, 6.27),
         (1.84, 6.87), (4.91, 2.79), (2.56, 3.29), (2.82, 5.97),
         (1.45, 5.65), (1.77, 6.13), (1.21, 2.34), (2.23, 5.47),
         (2.11, 5.0), (2.24, 3.71), (1.38, 6.3), (1.32, 3.98),
         (1.8, 3.62), (1.41, 2.53), (1.68, 2.66), (1.28, 7.31),
         (2.55, 7.52), (2.67, 0.65), (1.42, 4.49), (2.39, 6.01),
         (1.49, 7.93), (2.87, 5.08), (2.8, 2.16), (1.33, 2.38),
         (0.91, 8.56), (2.88, 5.65), (1.18, 3.5), (1.67, 2.23),
         (1.69, 2.4), (2.32, 6.68), (2.22, 7.55), (2.86, 3.6),
         (2.06, 4.29), (1.67, 4.01), (1.2, 3.72), (1.85, 6.08),
         (1.57, 0.06), (2.73, 2.23), (5.94, 7.72), (1.46, 5.67),
         (2.34, 4.44), (2.63, 6.27), (1.98, 3.48), (2.3, 2.49),
         (2.58, 7.55), (1.1, 5.47), (1.72, 7.31), (2.25, 6.84),
         (2.14, 4.55), (2.88, 2.13), (1.42, 5.29), (1.27, 7.35),
         (7.08, 4.92), (1.12, 6.21), (1.25, 3.86), (2.35, 6.55),
         (2.38, 6.0), (2.26, 6.72), (2.03, 6.14), (1.76, 2.0),
         (2.13, 2.72), (1.22, 5.01), (5.26, 8.64), (1.74, 4.15),
         (2.84, 2.34), (1.42, 3.29), (1.19, 1.88), (2.6, 8.61),
         (1.62, 2.08), (1.75, 2.15), (4.97, 6.63), (4.99, 0.58)]

for site in sites:
george.goto(site)
george.dot()
```

Section 2.3

2.3.2
```
def bloom(tortoise, color, length):
    tortoise.pencolor('red')
    tortoise.fillcolor(color)
    tortoise.begin_fill()
    for count in range(10):
        tortoise.forward(length)
        tortoise.left(108)
    tortoise.end_fill()
```

2.3.3
```
def bloom(tortoise, color, length, petals):
    tortoise.pencolor('red')
    tortoise.fillcolor(color)
    tortoise.begin_fill()
    for count in range(petals):
        tortoise.forward(length)
        tortoise.left(1080/petals)
    tortoise.end_fill()
```

2.3.5
```
import turtle

def bloom(tortoise, color, length):
    tortoise.pencolor('red')
    tortoise.fillcolor(color)
    tortoise.begin_fill()
    for count in range(8):
        tortoise.forward(length)
        tortoise.left(135)
    tortoise.end_fill()

def stem(tortoise, length):
    tortoise.pencolor('green')
    tortoise.pensize(length / 20)
    tortoise.up()
    tortoise.forward(length / 2)
    tortoise.down()
    tortoise.right(90)
    tortoise.forward(length)

def flower(tortoise, color1, color2, length):
    bloom(tortoise, color1, length)
    tortoise.up()
    tortoise.left(22.5)
    tortoise.forward(length * 1.082387 / 4)
    tortoise.right(22.5)
    tortoise.down()
    bloom(tortoise, color2, length / 2)
    stem(tortoise, length / 2)
    tortoise.forward(length / 2)

george = turtle.Turtle()
george.hideturtle()
george.speed(6)
```

```
flower(george, 'yellow', 'red', 200)
```

2.3.15
```
def horizontalCircles(tortoise):
    tortoise.up()
    tortoise.backward(500)
    tortoise.down()
    for numCircles in range(10):
        tortoise.up()
        tortoise.forward(100)
        tortoise.down()
        tortoise.circle(50)
```

Section 2.4

2.4.1
```
"""
Purpose: Draw a flower
Author: Ima Student
Date: September 15, 2020
CS 111
"""

import turtle

def bloom(tortoise, color, length):
    """Draws a geometric flower bloom.

    Parameters:
        tortoise: a Turtle object with which to draw the bloom.
        color:    a color string to use to fill the bloom.
        length:   the length of each segment of the bloom.

    Return value:
        None
    """

    tortoise.pencolor('red')       # sets tortoise's pen color
    tortoise.fillcolor(color)      #    and fill color
    tortoise.begin_fill()
    for count in range(8):         # draw a filled 8-sided geometric
        tortoise.forward(length)   #   flower bloom
        tortoise.left(135)
    tortoise.end_fill()

def stem(tortoise, length):
    """Draws a flower stem.

    Parameters:
        tortoise: a Turtle object, initially at the bloom starting
                  position.
        length:   the length of the stem and each segment of the bloom.

    Return value:
```

```
        None
    """

    tortoise.pencolor('green')      # sets tortoise's pen color to green
    tortoise.pensize(length / 20)   # sets pen width proportional to length
    tortoise.up()
    tortoise.forward(length / 2)    # stealthily move to top of stem
    tortoise.down()
    tortoise.right(90)
    tortoise.forward(length)       # draw the stem

def flower(tortoise, color, length):
    """Draws a flower.

    Parameters:
        tortoise: a Turtle object with which to draw the flower.
        color:    a color string to use to fill the bloom.
        length:  the length of each segment of the bloom.

    Return value:
        None
    """

    bloom(tortoise, color, length)    # draw the bloom
    stem(tortoise, length)            # and then the stem

def main():
    """Draws a yellow flower with segment length 200, and
       waits for a mouse click to exit."""

    george = turtle.Turtle()
    george.hideturtle()
    george.speed(6)
    flower(george, 'yellow', 200)
    screen = george.getscreen()
    screen.exitonclick()

main()
```

2.4.3

```
import turtle
import math

def draw(beth):
    ""Draws a cool pattern.

    Parameters:
        beth: a Turtle object

    Return value:
        None
    """

    beth.hideturtle()
```

```
    beth.speed(9)
    beth.fillcolor('blue')
    beth.begin_fill()
    beth.pencolor('red')
    for count in range(8):
        beth.circle(75)
        beth.left(45)
        beth.forward(10 * 1.414)   # 10 * sqrt(2)
    beth.end_fill()

def main():
    george = turtle.Turtle()
    draw(george)

main()
```

2.4.5
```
def main():
    age = int(input('Your age? '))
    days = age * 365.25
    print('You have been alive for over ' + str(days) + ' days!')

main()
```

Section 2.5

2.5.1
```
import math

def geometricMean(value1, value2):
    return math.sqrt(value1 * value2)

print(geometricMean(18, 31))
```

2.5.4
```
import math

def quadratic(a, b, c):
    x1 = (-b + math.sqrt(b ** 2 - 4 * a * c)) / (2 * a)
    x2 = (-b - math.sqrt(b ** 2 - 4 * a * c)) / (2 * a)

    return x1, x2
```

2.5.6
```
def distance(x1, y1, x2, y2):
    return math.sqrt((x1 - x2)**2 + (y1 - y2)**2)
```

2.5.9
```
def total(number1, number2):
    return number1 + number2

def main():
    firstNumber = float(input('Number 1: '))
    secondNumber = float(input('Number 2: '))
    print('Sum =', total(firstNumber, secondNumber))
```

2.5.10
```
def power(base, exponent):
    return base ** exponent

def main():
```

```
    base = float(input('Base: '))
    exponent = float(input('Exponent: '))
    print('Answer =', power(base, exponent))
```

2.5.11
```
def football(touchdowns, fieldGoals, safeties):
    score = touchdowns * 7 + fieldGoals * 3 + safeties * 2
    return score

def main():
    td = int(input('Touchdowns: '))
    fg = int(input('Field goals: '))
    safeties = int(input('Safeties: '))
    score = football(td, fg, safeties)
    print('Score =', score)
```

2.5.14
```
def darcy(K, dh, dl):
    return K * dH / dL
```

For the values given, `darcy(130, 50, 1000)` gives `6.5` m^2/day.

2.5.16
```
def songs(capacity, bitrate):
    seconds = 240                                   # 4 minute song in seconds
    capacity_in_bits = capacity * (2 ** 30) * 8     # iPod capacity in bits
    song_space = seconds * bitrate * (2 ** 10)      # bits required for one song

    return capacity_in_bits / song_space            # number of songs
```

2.5.18 No, because the parameters a and b are only assigned the values of the arguments x and y. There is no direct connection between a and x, and b and y. If the value assigned to a or b changes, this just changes the value or a or b, not x or y.

2.5.21
```
def twice(text):
    return text + ' ' + text
```

Section 2.6

2.6.1 No, this does not affect `t` because there is no direct connection between the two variables.

2.6.2 (a) The modified function is:

```
def distance(x1, y1, x2, y2):
    dist = math.sqrt((x1 - x2)**2 + (y1 - y2)**2)
    print('The local namespace is\n\t', locals())
    return dist
```

By calling

```
theDistance = distance(3, 7.25, 9.5, 1)
print(dist)
```

the following should be printed:

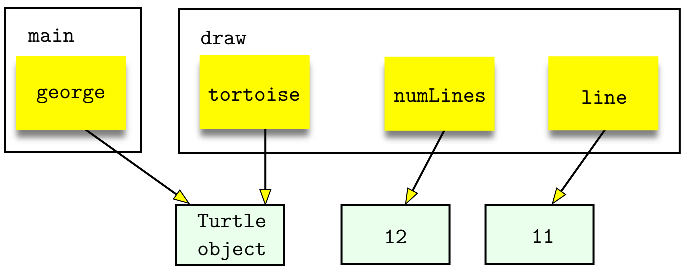
```
The local namespace is
 {'x1': 3, 'y1': 7.25, 'x2': 9.5, 'y2': 1,
  'dist': 9.017344398435716}
```

(b) Insert `print('The global namespace is\n \t ', globals())` at the end of `main()` shows that `main` and `distance` are in the global namespace:

```
The global namespace is
 {..., 'distance': <function distance at 0x111359b80>,
       'main': <function main at 0x111359c10>}
```

(c) A `NameError` is printed because `dist` does not exist in the current namespace.

2.6.5



Section 3.2

3.2.1 The binary number `1101` is equivalent to $1 + 4 + 8 = 13$ in decimal.

3.2.3 The binary number `11.0011` is equivalent to $2 + 1 + 1/8 + 1/16 = 33/16$ in decimal.

3.2.5 The decimal number $22 = 16 + 4 + 2$ is equivalent to `10110` in binary.

3.2.7 The decimal number $3.625 = 2 + 1 + 1/2 + 1/8$ is equivalent to `11.101` in binary.

3.2.10

a	b	a **and** b	**not** (a **and** b)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

3.2.12

a	b	**not** a	**not** b	**not** a **or not** b
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

Notice that **not** a **or not** b = **not** (a **and** b). This is one of *De Morgan's laws*; the other is the solution to the next problem.

Section 3.3

3.3.1
```
2e5 / 4.54e9
```

3.3.4
```
secondsInHour = 60 * 60
secondsInDay = secondsInHour * 24
secondsInYear = secondsInDay * 365.25
years = 4.54e9 / secondsInYear
```

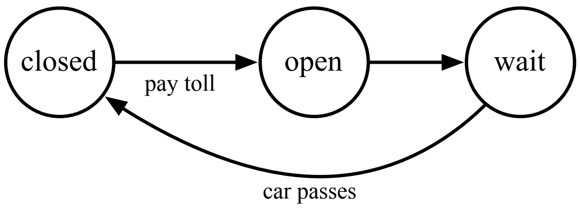
3.3.8
```
dnaCell = 3.4e-10 * 6e9
dnaBody = dnaCell * 50e12
distSun = 1.49598e8 * 1e3
roundTrips = dnaBody / (distSun * 2)
```

3.3.9
```
number = number - (number % 2)
```

3.3.14 The decimal number 0.1 is approximately `0.000110` in binary. This binary number is actually equivalent to $1/16 + 1/32 = 3/32 = 0.09375$ in decimal. The actual binary value is $0.0\overline{0011}$.

Section 3.5

3.5.1



Section 4.1

4.1.1
```
def pond(years, initialPopulation, harvest, rate):
    population = initialPopulation
    print('Year  Population')
    for year in range(years):
        population = (1 + rate) * population - harvest
        print('{0:^4}  {1:>9.2f}'.format(year + 1, population))

    return population
```

4.1.7
```
def triangle():
    for count in range(10):
        print('*' * (count + 1))
```

4.1.10
```
for number in range(50):
    print(2 * number + 1)
```

4.1.11
```
def multiples(n):
    for i in range(0, 101, n):
        print(i)
```

4.1.15
```
import turtle, math

def plot(tortoise, n):
    for x in range(0, n + 1):
        tortoise.goto(x, math.sin(math.radians(x)))

def main():
    george = turtle.Turtle()
    screen = george.getscreen()
    screen.setworldcoordinates(0, -1, 1080, 1)
    plot(george, 1080)
    screen.exitonclick()

main()
```

4.1.17
```
def growth1(totalDays):
    population = 0
```

```
    print('Day Size')
    for day in range(1, totalDays + 1):
        population = population + 3
        print('{0:<3}{1:>5}'.format(day, population))
    return population
```

4.1.21
```
def bacteria(days):
    population = 100
    hours = days * 24
    for hour in range(hours):
        population = population + 0.1 * population
    return population
```

4.1.23
```
def sumNumbers(n):
    total = 0
    for number in range(1, n + 1):
        total = total + number
    return total
```

4.1.26
```
def factorial(n):
    fact = 1
    for number in range(2, n + 1):
        fact = fact * number
    return fact
```

4.1.32 (a)

Trace arguments: totalDays = 4

Step	Line	population	day	Notes
1	2	0	—	population ← 0
2	3	″	—	prints header
3	4	″	1	day ← 1
4	5	3	″	population ← 0 + 3
5	6	″	″	no changes; prints 1 3
6	4	″	2	day ← 2
7	5	6	″	population ← 3 + 3
8	6	″	″	no changes; prints 2 6
9	4	″	3	day ← 3
10	5	9	″	population ← 6 + 3
11	6	″	″	no changes; prints 3 9
12	4	″	4	day ← 4
13	5	12	″	population ← 9 + 3
14	6	″	″	no changes; prints 4 12

Return value: 12

4.1.33
```
def interest(originalAmount, rate, periods):
    amount = originalAmount
    rate = rate / 100 / periods
    for p in range(periods):
        amount = amount + rate * amount
```

```
    return amount - originalAmount

print(interest(1000, 1.0, 365))
print(interest(1000, 1.25, 12))
```

Section 4.2

4.2.1
```
import matplotlib.pyplot as pyplot

def zombieApocalypse(days):
    zombies = 1
    zombieList = []
    zombieList.append(zombies)
    for day in range(days):
        newZombies = zombies * 2
        zombies = zombies + newZombies
        zombieList.append(zombies)

    pyplot.plot(range(days + 1), zombieList)
    pyplot.xlabel('Day')
    pyplot.ylabel('Zombies')
    pyplot.show()

zombieApocalypse(14)
```

4.2.3
```
def profitPlot(maxPrice):
    profitList = []
    priceList = []
    for price in range(1, 2 * maxPrice + 1):
        realPrice = price / 2
        sales = 2500 - 80 * realPrice
        income = sales * realPrice
        profit = income - 8000
        priceList.append(realPrice)
        profitList.append(profit)

    pyplot.plot(priceList, profitList)
    pyplot.xlabel('Ticket price ($)')
    pyplot.ylabel('Profit ($)')
    pyplot.show()
```

4.2.7
```
import matplotlib.pyplot as pyplot

def bacteria(days):
    population1 = 100
    population2 = 100
    popList1 = []
    popList2 = []
    popList1.append(population1)
    popList2.append(population2)
    hours = days * 24
    for hour in range(hours):
        population1 = population1 + 0.1 * population1
```

```
            population2 = population2 + 0.15 * population2
            popList1.append(population1)
            popList2.append(population2)

        pyplot.plot(range(hours + 1), popList1, label = '10% growth')
        pyplot.plot(range(hours + 1), popList2, label = '15% growth')
        pyplot.xlabel('Hour')
        pyplot.ylabel('Colony size')
        pyplot.legend()
        pyplot.show()

    bacteria(3)
```

Section 4.3

4.3.1
```
amount = 1000
year = 0
while amount < 1200:
    amount = 1.03 * amount
    year = year + 1
print(year)
```

4.3.4
```
def profitTable(maxPrice):
    print('Price    Income     Profit')
    print('------  ---------  ---------')
    price = 1.0
    while price <= maxPrice:
        sales = 2500 - 80 * price
        income = sales * price
        profit = income - 8000
        formatString = '${0:>5.2f}  ${1:>8.2f}  ${2:8.2f}'
        print(formatString.format(price, income, profit))
        price = price + 0.5
```

4.3.6
```
def tribbleApocalypse():
    tribbles = 10
    hours = 0
    while tribbles < 1e6:
        tribbles = tribbles + tribbles // 2
        hours = hours + 1
    return hours
```

Section 4.6

4.6.1 (a) The accumulator exhibits quadratic growth.

(c) The accumulator exhibits exponential growth.

Section 5.1

5.1.1 (a) `Candidate two wins!`

(b) `Candidate one wins!`

(c) `Candidate one wins!`

5.1.2
```
if votes1 > votes2:
    print('Candidate one wins!')
elif votes1 < votes2:
    print('Candidate two wins!')
else:
    print('There was a tie.')
```

5.1.7
```
def weather():
    r = random.random()
    if r < 0.7:
        print('RAIN!')
    else:
        print('SUN!')
```

5.1.8
```
def roll():
    r = random.random()
    if r < 1 / 6:
        return 1
    elif r < 1 / 3:
        return 2
    elif r < 1 / 2:
        return 3
    elif r < 2 / 3:
        return 4
    elif r < 5 / 6:
        return 5
    else:
        return 6
```

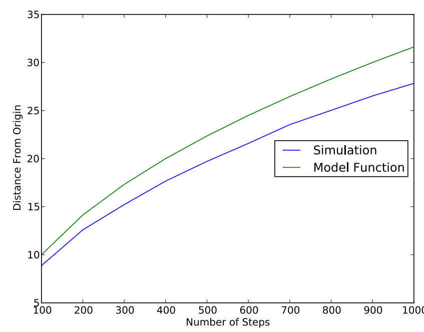
5.1.10
```
def guessMyNumber():
    number = random.randrange(1, 101)
    guess = int(input('Guess my number: '))
    if guess == number:
        print('You guessed it!')
    elif guess < number:
        print('Your guess was too low.  My number was ' + str(number) + '.')
    else:
        print('Your guess was too high.  My number was ' + str(number) + '.')
```

5.1.13
```
def countVotes(votes):
    laura = 0
    john = 0
    for vote in votes:
        if vote == 'Laura':
            laura = laura + 1
        else:
            john = john + 1
    if laura > john:
        winner = 'Laura'
    elif john > laura:
        winner = 'John'
    else:
        winner = 'Tie'
    return winner
```

5.1.16
```
import matplotlib.pyplot as pyplot
import math

def plotDistances(maxSteps, trials):
    distances = [ ]
    y = [ ]
    stepRange = range(100, maxSteps + 1, 100)
    for steps in stepRange:
        distance = rwMonteCarlo(steps, trials)
        distances.append(distance)
        y.append(math.sqrt(steps))
    pyplot.plot(stepRange, distances, label = 'Simulation')
    pyplot.plot(stepRange, y, label = 'Model Function')
    pyplot.legend(loc = 'center right')
    pyplot.xlabel('Number of Steps')
    pyplot.ylabel('Distance From Origin')
    pyplot.show()
```

Your final plot should look like that to the right. By experimenting with different numbers of trials, you should find that the plot of the simulation results gets “smoother” with more trials. But more trials obviously take longer. This is another example of the tradeoffs that we often encounter when solving problems.



Section 5.4

5.4.1
```
def password():
    username = input('Username: ')
    password = input('Password: ')
    return username == 'alan.turing' and password == 'notTouring'
```

5.4.4
```
def leapYear(year):
    return ((year % 4 == 0) and (year % 100 != 0)) or (year % 400 == 0)

def leapYears(beginYear, endYear):
    yearList = []
    for year in range(beginYear, endYear + 1):
        if leapYear(year):
            yearList.append(year)
    return yearList
```

5.4.5
```
def nextLeapYear(afterYear):
    year = afterYear + 1
    while not leapYear(year):
        year = year + 1
    return year
```

5.4.7
```
def between(number, low, high):
    return (number >= low and number <= high)
```

5.4.12
```
def winner(score1, score2):
    if score1 > score2:
        return 1
    else:
        return 2
```

5.4.14
```
def cost(quantity):
    if quantity <= 20:
        return quantity * 1500
    else:
        return quantity * 1500 * 0.95
```

5.4.20
```
def maxOfThree(a, b, c):
    if (a >= b) and (a >= c):
        return a
    elif b >= c:
        return b
    else:
        return c
```

5.4.22
```
def oddFactors(number):
    factorList = []
    for divisor in range(2, number + 1):
        if (number % divisor == 0) and (divisor % 2 == 1):
            factorList.append(divisor)
    return factorList
```

Section 5.5

5.5.1
```
Precondition: radius is a positive number

Postcondition: returns the volume of a sphere with
               the given radius

assert radius > 0, 'radius must be positive'
assert isinstance(radius, int) or isinstance(radius, float), \
       'radius must be a number'
```

5.5.5
```
Preconditions:
    tortoise is a Turtle object
    n is a positive integer representing the positive and negative
    extent of x values

Postcondition: plots x**2 in turtle graphics between
               x = -n and x = n, inclusive

assert isinstance(tortoise, turtle.Turtle), \
       'tortoise must be a Turtle object'
assert isinstance(n, int), 'n must be an integer'
assert n > 0, 'n must be positive'
```

5.5.9
```
# should not allow values of score < 0 and > 100
```

```
    # return -1 in these cases

    def test_assignGP():
        assert assignGP(90) == 4
        assert assignGP(80) == 3
        assert assignGP(75) == 2

        assert assignGP(10000) == -1  # should fix these cases to be invalid
        assert assignGP(-1) == -1

        print('Passed all tests of assignGP!')
```

5.5.12
```
def test_volumeSphere():
    assert volumeSphere(10) > 4188.789 and volumeSphere(10) < 4188.791
    assert volumeSphere(1) > 4.18879 and volumeSphere(1) < 4.18881
    assert volumeSphere(2) > 33.51032 and volumeSphere(2) < 33.51035
    assert volumeSphere(100) > 4188790.20478 and \
           volumeSphere(100) < 4188790.20481
    assert volumeSphere(0) == 0
    assert volumeSphere(360) > 195432195.794 and \
           volumeSphere(360) < 195432195.798

    print('Passed all tests of volumeSphere!')
```

5.5.15
```
def daysAlive():
    good = False
    while not good:
        text = input('How old are you? ')
        try:
            age = int(text)
        except ValueError:
            print('Please enter an integer.')
        else:
            good = True

    print('You have been alive for', round(age * 365.25), 'days!')
```

Section 5.6

5.6.1
```
def ABC():
    answer = input('Enter A, B, or C: ')
    while answer != 'A' and answer != 'B' and answer != 'C':
        answer = input('Enter A, B, or C: ')
    print('Thank you.')
    return answer
```

5.6.5
```
def yearsUntilDoubled(amount, rate):
    finalAmount = amount
    years = 0
    while finalAmount < 2 * amount:
        finalAmount = finalAmount * (1 + rate)
        years = years + 1
    return years
```

Section 6.1

6.1.1
```
def vowels(word):
    word = word.lower()
    return word.count('a') + word.count('e') + word.count('i') + \
           word.count('o') + word.count('u')
```

6.1.3
```
def underscore(sentence):
    return sentence.replace(' ', '_')
```

6.1.6
```
def letters(text):
    for letter in text:
        print(letter)
```

6.1.7
```
def countCharacter(text, letter):
    tally = 0
    for character in text:
        if character == letter:
            tally = tally + 1
    return tally
```

6.1.10
```
def underscore(sentence):
    new = ''
    for character in sentence:
        if character == ' ':
            new = new + '_'
        else:
            new = new + character
    return new
```

6.1.17
```
def wordCount(text):
    return len(wordTokens(text))
```

Section 6.2

6.2.3
```
def lowerCaseFile(fileName):
    textFile = open(fileName, 'r', encoding = 'utf-8')
    for line in textFile:
        line = line.lower()
        print(line)
    textFile.close()
```

6.2.6
```
import textlib
import matplotlib.pyplot as pyplot

def plotWordsPerParagraph(fileName):
    textFile = open(fileName, 'r', encoding = 'utf-8')
    paragraph = ''
    wordCounts = []
    for line in textFile:
        if line != '\n':
            paragraph = paragraph + line
        elif paragraph != '':
            wordCounts.append(textlib.wordCount(paragraph))
            paragraph = ''
    textFile.close()
```

```
    pyplot.plot(range(1, len(wordCounts) + 1), wordCounts)
    pyplot.xlabel('Paragraph number')
    pyplot.ylabel('Word count')
    pyplot.show()
```

6.2.10
```
def writeLineLengths(fileName, outputFileName):
    textFile = open(fileName, 'r', encoding = 'utf-8')
    newTextFile = open(outputFileName, 'w')
    for line in textFile:
        line = line.rstrip('\n')
        newTextFile.write(str(len(line)) + '\n')
    textFile.close()
    newTextFile.close()
```

6.2.13
```
import urllib.request as web
import textlib

def wordCountWeb(url):
    webPage = web.urlopen(url)
    rawBytes = webPage.read()
    webPage.close()
    text = rawBytes.decode('utf-8')

    return textlib.wordCount(text)
```

Section 6.3

6.3.1
(a) `len(word)`
(b) `word[0]`
(c) `word[2]`
(d) `word[-1]`

6.3.2
(a) `quote[5:9]`
(b) `quote[25:]`

6.3.3
(a) `True` because `ord('c')` < `ord('d')`
(b) `True` because the first three characters of the strings are the same but the righthand string is longer
(c) `False` because `ord('c')` ≠ `ord('C')`

6.3.4
```
def digit2String(digit):
    if (digit < 0) or (digit > 9):
        return None
    return chr(ord('0') + digit)
```

6.3.16
```
def marriedName(fullName, spouseLastName, hyphenate):
    if hyphenate:
        return fullName + '-' + spouseLastName
    else:
        newName = ''
        index = 0
        while (fullName[index] != ' '):
```

```
            newName = newName + fullName[index]
            index = index + 1
        return newName + ' ' + spouseLastName
```

Section 6.4

6.4.1 (a)
```
for index in range(len(text)):
    print(text[index])
```

(b)
```
newText = ''
for index in range(len(text)):
    if text[index] != ' ':
        newText = newText + text[index]
```

6.4.2 (a)
```
index = 0
while index < len(text):
    print(text[index])
    index = index + 1
```

(b)
```
newText = ''
index = 0
while index < len(text):
    if text[index] != ' ':
        newText = newText + text[index]
    index = index + 1
```

6.4.3 (a) missing `bigVeggie = ''` before loop

(b) missing initialization of `answer` before loop

(c) `range(veggie)` should be `range(len(veggie))`

6.4.7
```
def encode(word):
    even = ''
    odd = ''
    for index in range(len(word)):
        if index % 2 == 0:
            even = even + word[index]
        else:
            odd = odd + word[index]
    return even + odd
```

6.4.9
```
def palindrome(text):
    text = text.lower()
    text = text.replace(' ', '')
    for index in range(len(text) // 2):
        if text[index] != text[-(index + 1)]:
            return False
    return True
```

Section 6.5

6.5.1
```
def countCharacter(text, letter):
    count = 0
    for index in range(len(text)):
        if text[index] == letter:
```

```
            count = count + 1
    return count
```

6.5.2
```
def count(text, target):
    count = 0
    for index in range(len(text)):
        if text[index:index + len(target)] == target:
            count = count + 1
    return count
```

Section 6.6

6.6.2 There are $4 \cdot 3 = 12$ comparisons total. They are, in order:

```
a x
a y
a z
b x
b y
b z
c x
c y
c z
d x
d y
d z
```

6.6.6
```
def hamming(bits1, bits2):
    distance = 0
    for index in range(len(bits1)):
        if bits1[index] != bits2[index]:
            distance = distance + 1
    return distance
```

6.6.8 In the following solution, for each character considered in the outer loop, we only look at characters that come after it in the inner loop. In this way, we never compare the same characters twice.

```
for index1 in range(len(text) - 1):
    for index2 in range(index1 + 1, len(text)):
        if text[index1] == text[index2]:
            print(text[index1], index1, index2)
```

6.6.11
```
def findRepeats(text, length):
    for index in range(len(text) - 2 * length + 1):
        if text[index] != ' ':
            word = text[index:index + length]
            if text[index + length] == ' ' and \
              text[index + length + 1:index + 2 * length + 1] == word:
                print(index, word)
```

Section 7.1

7.1.1 (a) `print(len(data))`

(b) `print(data[2])`

(c) `print(data[-3:])`

7.1.4
```
def sumOdds(data):
    total = 0
    for item in data:
        if item % 2 == 1:
            total = total + item
    return total
```

7.1.12
```
def shortest(words):
    short = words[0]
    for word in words[1:]:
        if len(word) < len(short):
            short = word
    return short
```

7.1.16
```
def linearSearch(data, target):
    for item in data:
        if item == target:
            return True
    return False
```

7.1.17
```
def linearSearch(data, target):
    for index in range(len(data)):
        if data[index] == target:
            return index
    return -1
```

7.1.18
```
def intersect(data1, data2):
    for item in data1:
        if search(data2, item):
            return True
    return False
```

Section 7.2

7.2.1
```
fruit.append('grapes')
fruit = fruit + ['grapes']
```

7.2.2
```
def squares(n):
    squaresList = []
    for index in range(1, n + 1):
        squaresList.append(index * index)
    return squaresList
```

7.2.4
```
def square(data):
    for index in range(len(data)):
        data[index] = data[index] ** 2
```

7.2.5
```
def swap(data, i, j):
    temp = data[i]
    data[i] = data[j]
    data[j] = temp
```

7.2.8
```
def delete(data, index):
    if (index < 0) or (index >= len(data)):
        return data.copy()
    return data[:index] + data[index + 1:]
```

7.2.14 (a) `groceries.insert(4, 'jelly beans')`

(b) `groceries.insert(1, 'donuts')`

(e) `groceries.pop(7)`

(f) `groceries.pop(7)`

7.2.15
```
import textlib

def getStopWords(fileName):
    stopWordFile = open(fileName, 'r')
    stopWords = []
    for line in stopWordFile:
        word = textlib.removePunctuation(line).rstrip()
        stopWords.append(word)
    stopWordFile.close()
    return stopWords

def removeStopWords(wordList, stopWordList):
    index = 0
    while index < len(wordList):
        if wordList[index] in stopWordList:
            wordList.pop(index)
        else:
            index = index + 1
```

7.2.20
```
def squares(n):
    return [index * index for index in range(1, n + 1)]
```

Section 7.3

7.3.2
```
def main():
    textFile = open('senseandsensibility.txt', 'r')
    text = textFile.read()
    textFile.close()

    words = wordFrequencies(text)
    bigrams = bigramFrequencies(text)
    printMostFrequent(words, 5)
    printMostFrequent(bigrams, 5)

main()
```

The five most frequent words are:

```
Key                     Frequency
---                     ---------
to                       4085
the                      4085
of                       3566
```

```
and                      3371
her                      2510
```

The five most frequent bigrams are:

```
Key                     Frequency
---                     ---------
('to', 'be')              431
('of', 'the')             430
('in', 'the')             356
('it', 'was')             273
('of', 'her')             272
```

7.3.3
```
def wordFrequencies(text):
    wordList = textlib.wordTokens(text)  # get the list of words in text
    stopWords = getStopWords('stopwords.txt')
    removeStopWords(wordList, stopWords)

    wordFreqs = { }
    for word in wordList:
        if word in wordFreqs:    # if word is already a key in wordFreqs,
            wordFreqs[word] = wordFreqs[word] + 1   # count the word
        else:                    # otherwise,
            wordFreqs[word] = 1  #   create a new entry word:1

    return wordFreqs

# Key                     Frequency
# ---                     ---------
# elinor                    616
# mrs                       525
# marianne                  488
# time                      237
# dashwood                  224
```

7.3.5
```
def firstLetterCounts(wordList):
    counts = {}
    for word in wordList:
        first = word[0].lower()
        if first in counts:
            counts[first] = counts[first] + 1
        else:
            counts[first] = 1
    return counts
```

7.3.10
```
def bonus(salaries):
    for name in salaries:
        salaries[name] = salaries[name] * 1.05
```

7.3.13
```
def txtTranslate(txtWord):
    translations = {'lol': 'laugh out loud', 'brb': 'be right back'}
    if txtWord in translations:
        return translations[txtWord]
    else:
        return 'The abbreviation was not found.'
```

7.3.20
```
def wordPredictor(bigramFreqs):
    currentWord = input('First word: ')
    text = currentWord

    done = False
    while not done:
        sortedValues = []     # create list of (freq, next word) pairs
        for bigram in bigramFreqs:
            if bigram[0] == currentWord:
                sortedValues.append((bigramFreqs[bigram], bigram[1]))
        if len(sortedValues) == 0:      # if no next words, quit
            done = True
        else:
            sortedValues.sort(reverse = True)  # sort the list
            choices = []                       # get top 3 choices
            for pair in sortedValues:
                choices.append(pair[1])
            choices = choices[:3]

            choice = input('Choose among ' + str(choices) + ': ')
            if choice not in choices:
                done = True
            else:
                currentWord = choice
                text = text + ' ' + currentWord
                print(text)
```

Section 7.4

7.4.1
```
def plotQuakes():
    url = 'http://earthquake.usgs.gov/earthquakes/feed' + \
                      '/v1.0/summary/all_month.csv'
    quakeFile = web.urlopen(url)
    header = quakeFile.readline()

    longitudes = []
    latitudes = []
    depths = []
    magnitudes = []
    for rawLine in quakeFile:
        line = rawLine.decode('utf-8')
        row = line.split(',')
        latitudes.append(float(row[1]))
        longitudes.append(float(row[2]))
        depths.append(float(row[3]))
        magnitudes.append(float(row[4]))
    quakeFile.close()

    colors = []
    for depth in depths:
        if depth < 10:
            colors.append('yellow')
```

```
        elif depth < 50:
            colors.append('red')
        else:
            colors.append('blue')

    sizes = []
    for mag in magnitudes:
        sizes.append(mag ** 2)

    pyplot.scatter(longitudes, latitudes, sizes, color = colors)
    pyplot.show()
```

7.4.3
```
def login(filename):
    passwordFile = open(filename, 'r', encoding = 'utf-8')
    passwords = {}
    for line in passwordFile:
        values = line.split()
        passwords[values[0]] = values[1].strip()
    passwordFile.close()

    username = input('Username: ')
    password = input('Password: ')
    while username not in passwords or passwords[username] != password:
        print('\nTry again.')
        username = input('Username: ')
        password = input('Password: ')
    print('Success!')
```

7.4.6
```
def plotTemps():
    tempFile = open('madison_temp.csv', 'r')
    header = tempFile.readline()

    years = []
    minTemps = []
    for line in tempFile:
        row = line.split(',')
        date = row[2]
        if date[4:6] == '01':
            years.append(date[:4])
        minTemps.append(int(row[4]) / 10)
    tempFile.close()

    pyplot.plot(range(len(minTemps)), minTemps)
    pyplot.ylabel('Minimum temperature')
    pyplot.xlabel('Year')
    pyplot.xticks(range(0, len(minTemps), 12), years)
    pyplot.xlim(0, len(minTemps))
    pyplot.show()
```

Section 8.1

8.1.1
```
scores = [[10305, 700, 610], [11304, 680, 590], [10254, 710, 730],
          [12007, 650, 690], [10089, 780, 760]]
```

8.1.2 (a) `scores[4][1]`

(b) `scores[1][2]`

8.1.6
```
def readData():
    dataFile = open('madison_temp.csv', 'r')
    header = dataFile.readline()
    table = {}
    for line in dataFile:
        row = line.split(',')
        row[3] = int(row[3])
        row[4] = int(row[4])
        table[row[2]] = row[3:]
    dataFile.close()
    return table

def getMinTemp(table, date):
    if date in table:
        return table[date][1]
    return None
```

Section 8.2

8.2.1 These starting configurations all have established names.

(a) This one is called a “glider.”

(b) This is ”blinker.”

(c) This is a tribute to John Conway, who died of Covid-19 in 2020, by Randall Munroe, the creator of xkcd (`https://xkcd.com/2293/`).

(d) This one is called the “lightweight spaceship.”

(e) This one is called an “acorn.”

8.2.3
```
def printGrid(grid):
    rows = len(grid)
    columns = len(grid[0])

    for r in range(rows):
        for c in range(columns):
            print(grid[r][c], end = ' ')
        print()
```

8.2.5
```
def diagonal(n):
    grid = []
    for r in range(n):
        row = []
        for c in range(n):
            if c == r:
                row.append(1)
            else:
                row.append(0)
        grid.append(row)
    return grid
```

8.2.8
```
def find(grid, target):
    rows = len(grid)
    columns = len(grid[0])
    for row in range(rows):
        for col in range(columns):
            if grid[row][col] == target:
                return (row, col)
    return (-1, -1)
```

Section 8.3

8.3.2
```
def warmPixel(color, factor):
    red = min(255, int((1 + factor) * color[0]))
    green = min(255, int((1 + factor) * color[1]))
    blue = color[2]
    return (red, green, blue)

def warm(photo, factor):
    width = photo.width()
    height = photo.height()
    newPhoto = image.Image(width, height, title = 'Warm image')
    for y in range(height):
        for x in range(width):
            color = photo.get(x, y)
            newPhoto.set(x, y, warmPixel(color, factor))
    return newPhoto
```

8.3.6
```
def flipHorizontal(photo):
    width = photo.width()
    height = photo.height()
    newPhoto = image.Image(width, height, title = 'Flipped image')
    for y in range(height):
        for x in range(width):
            color = photo.get(x, y)
            newPhoto.set(width - x - 1, y, color)
    return newPhoto
```

8.3.8
```
def reducePixels(photo, x, y):
    offsets = [(0, 0), (0, 1), (1, 0), (1, 1)]
    red = 0
    green = 0
    blue = 0
    for offset in offsets:
        color = photo.get(x + offset[0], y + offset[1])
        red = red + color[0]
        green = green + color[1]
        blue = blue + color[2]
    return (red // 4, green // 4, blue // 4)

def reduce(photo):
    width = photo.width()
    height = photo.height()
    newPhoto = image.Image(width // 2, height // 2, title = 'Reduced')
```

```
    for y in range(0, height, 2):
        for x in range(0, width, 2):
            color = reducePixels(photo, x, y)
            newPhoto.set(x // 2, y // 2, color)
    return newPhoto
```

Section 9.1

9.1.1
```
def tree(tortoise, length, depth):
    if depth <= 1:
        tortoise.forward(length)
        tortoise.backward(length)
    else:
        angle1 = random.randrange(10, 61)
        angle2 = random.randrange(10, 61)
        shrink1 = random.random() * 0.25 + 0.5
        shrink2 = random.random() * 0.25 + 0.5
        tortoise.forward(length)
        tortoise.left(angle1)
        tree(tortoise, length * shrink1, depth - 1)
        tortoise.right(angle1 + angle2)
        tree(tortoise, length * shrink2, depth - 1)
        tortoise.left(angle2)
        tortoise.backward(length)
```

9.1.4
```
def kochSnowFlake(tortoise, length, depth, sides):
    for side in range(sides):
        koch(tortoise, length, depth)
        tortoise.right(360 / sides)
```

Section 9.2

9.2.1
```
def sumIt(n):
    if n <= 1:
        return 1
    return sumIt(n - 1) + n
```

9.2.5
```
def length(data):
    if data == []:
        return 0
    return 1 + length(data[1:])
```

9.2.6
```
def minList(data):
    if data == []:
        return None
    if len(data) == 1:
        return data[0]
    return minList(data[0], minList(data[1:]))
```

9.2.7
```
def reverse(text):
    if text == '':
        return ''
    return reverse(text[1:]) + text[0]
```

9.2.12
```
def countUpper(text):
    if text == '':
        return 0
    if text[0].isupper():
        return 1 + countUpper(text[1:])
    return countUpper(text[1:])
```

Section 9.3

9.3.3 (a)
1. Use the classical Tower of Hanoi algorithm to move the n white disks from peg A to peg B, using peg D as the intermediate peg.
2. Use the classical algorithm to move the n black disks from peg C to peg A, using peg D as the intermediate.
3. Use the classical algorithm to move the n white disks from peg B to peg C, using peg D as the intermediate.

(b)
```
def move2(color, source, destination):
    print('Move a ' + str(color) + ' disk from peg ' + str(source)
                                 + ' to peg ' + str(destination) + '.')

def hanoi2(n, color, source, destination, intermediate):
    if n >= 1:
        hanoi2(n - 1, color, source, intermediate, destination)
        move2(color, source, destination)
        hanoi2(n - 1, color, intermediate, destination, source)

def solveTower4(n):
    if n >= 1:
        hanoi2(n, 'white', 'A', 'B', 'D')
        hanoi2(n, 'black', 'C', 'A', 'D')
        hanoi2(n, 'white', 'B', 'C', 'D')
```

(c) Since the Tower of Hanoi algorithm requires $2^n - 1$ moves, this algorithm requires $3 \cdot 2^n - 1$ moves.

Section 9.4

9.4.1
```
def linearSearch(data, target):
    if len(data) == 0:                   # base case 1: not found
        return False
    if target == data[0]:                # base case 2: found
        return True
    return linearSearch(data[1:], target)  # recursive case
```

9.4.2 No, because the slicing operation, which creates a new list, requires linear time by itself. So now the function is performing a linear number of linear-time operations, resulting in a $\mathcal{O}(n^2)$ or quadratic-time algorithm.

Section 9.5

9.5.1
```
def power(a, n):
    if n == 0:
        return 1
```

```
    elif n == 1:
        return a
    elif n % 2 == 0:
        return power(a, n // 2) ** 2
    else:
        return a * power(a, n // 2) ** 2
```

9.5.4
```
def profit(prices, first, last):
    if last <= first:
        return 0

    midIndex = (first + last) // 2
    leftProfit = profit(prices, first, midIndex)
    rightProfit = profit(prices, midIndex + 1, last)

    buy = min(prices[first:midIndex + 1])
    sell = max(prices[midIndex + 1: last + 1])
    midProfit = sell - buy

    return max(leftProfit, rightProfit, midProfit)
```

9.5.8
```
def numPaths(n, row, column):
    if (row > n - 1) or (column > n - 1):
        return 0
    if (row, column) == (n - 1, n - 1):
        return 1
    return numPaths(n, row + 1, column) + numPaths(n, row, column + 1)
```

Section 10.1

10.1.1 Add the following code just before `return -1` in each algorithm.

```
if right == -1:
    print(str(target) + ' would have been to the left of '
                      + str(keys[left]) + '.')
elif left == len(keys):
    print(str(target) + ' would have been to the right of '
                      + str(keys[right]) + '.')
else:
    print(str(target) + ' would have been between '
                      + str(keys[right]) + ' and '
                      + str(keys[left]) + '.')
```

10.1.2 No, they would not. Suppose `target` is smaller than any key in the list. Then `left` will eventually become equal to `right`. When this happens, `mid = (left + right) // 2`, which is the same value as `left` and `right`. So, when `right` is assigned the value of `mid`, it will stay the same rather than becoming one less than `left`. Therefore, `left > right` will never be true. The iterative version of the algorithm will be in an infinite loop and the recursive version will be in an infinite sequence of recursive calls.

Section 10.2

10.2.3 In terms of elementary steps, `selectionSortAlt` is less efficient because it requires that the list be traversed twice in each iteration of the loop. However, because the built-in `min` and `index` functions are highly optimized, it may actually be faster.

10.2.5
```
import urllib.request as web

def readQuakes():
    url = 'http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_month.csv'
    quakeFile = web.urlopen(url)
    header = quakeFile.readline()

    ids = []
    data = []
    for line in quakeFile:
        line = line.decode('utf-8')
        row = line.split(',')
        latitude = float(row[1])
        longitude = float(row[2])
        depth = float(row[3])
        magnitude = float(row[4])
        data.append((latitude, longitude, depth, magnitude))
        ids.append(row[11])
    quakeFile.close()
    return ids, data
```

Section 10.3

10.3.2
```
def timing():
    dictFile = open('/usr/share/dict/words', 'r')
    words = []
    for word in dictFile:
        words.append(word[:-1])
    dictFile.close()

    words1 = words[:]
    start = time.time()
    words1.sort()
    finish = time.time()
    print('The sort method required', finish - start, 'seconds.')

    start = time.time()
    insertionSort(words)
    finish = time.time()
    print('The insertion sort required', finish - start, 'seconds.')
```

The `sort` method will take a fraction of a second while insertion sort will take several minutes.

10.3.4 Selection sort is not a stable sort. To see why not, consider the tiny list `[2, 2, 1]`. Selection sort will swap the first `2` with the `1`, resulting in the two values of `2` being in the opposite order as they were originally.

On the other hand, insertion sort is stable since it never swaps values.

Section 10.4

10.4.5 (a) With linear search, the time complexity is $\mathcal{O}(n)$.

(b) To use binary search, the list must be sorted first. So the time complexity is $n \log_2 n + \log_2 n$, which is $\mathcal{O}(n \log_2 n)$.

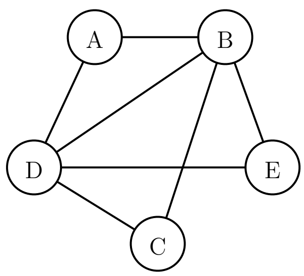
(c) n linear searches have time complexity $\mathcal{O}(n^2)$.

(d) n binary searches have time complexity $n \log_2 n + n \log_2 n$, which is $\mathcal{O}(n \log_2 n)$.

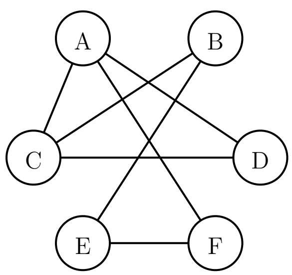
(e) Sorting the list and then using binary search is only worthwhile if you plan to perform sufficiently many searches. If the number of searches is proportional to n, using binary search is definitely better.

Section 11.1

11.1.2 (a)



11.1.3 (a)



11.1.4 (a)

```
graph = [[0, 1, 0, 1, 0, 0, 0, 0],
         [1, 0, 1, 0, 0, 1, 0, 1],
         [0, 1, 0, 0, 1, 0, 0, 0],
         [1, 0, 0, 0, 0, 0, 1, 0],
         [0, 0, 1, 0, 0, 0, 1, 0],
         [0, 1, 0, 0, 0, 0, 1, 0],
         [0, 0, 0, 1, 1, 1, 0, 1],
         [0, 1, 0, 0, 0, 0, 1, 0]]
```

(b)

```
graph = {1: [2, 4], 2: [1, 3, 6, 8], 3: [2, 5],
         4: [1, 7], 5: [3, 7], 6: [2, 7],
         7: [4, 5, 6, 8], 8: [2, 7]}
```

(c) Neighborhoods:

Node 1: 2, 4; Node 2: 1, 3, 6, 8; Node 3: 2, 5; Node 4: 1, 7; Node 5: 3, 7; Node 6: 2, 7; Node 7: 4, 5, 6, 8; Node 8: 2, 7

(d) Degrees:

Node 1: 2; Node 2: 4; Node 3: 2; Node 4: 2; Node 5: 2; Node 6: 2; Node 7: 4; Node 8: 2

Nodes 2 and 7 have the maximum degree.

11.1.6 (a) These networks are equivalent because they can be redrawn to look identical.

11.1.7
```
def readGraph(fileName):
    graphFile = open(fileName, 'r', encoding = 'utf-8')
    graph = { }
    for line in graphFile:
        edge = line.rstrip().split()
        if edge[0] in graph:
            graph[edge[0]].append(edge[1])
        else:
            graph[edge[0]] = [edge[1]]
        if edge[1] in graph:
            graph[edge[1]].append(edge[0])
        else:
            graph[edge[1]] = [edge[0]]
    graphFile.close()
    return graph
```

Section 11.2

11.2.1 Ted, Cathy, Christina, Beth, Dave, Kevin, Ryder, Tyler, Vanessa, Amelia, Nick, Caroline, Lillian

11.2.6 (a)
```
def dfs(network, source, visited):
    if source in visited:
        return

    visited.append(source)

    for neighbor in network[source]:
        dfs(network, neighbor, visited)
```

Section 11.3

11.3.1
```
def averageClusteringCoefficient(network):
    totalCC = 0
    for node in network:
        totalCC = totalCC + clusteringCoefficient(network, node)
    return totalCC / len(network)
```

11.3.3
```
def closenessCentrality(network, node):
    totalDistance = 0
    dist, pred = bfs(network, node)
    for other in network:
        if dist[other] == float('inf'):
            dist[other] = len(network)
        totalDistance = totalDistance + dist[other]
    return totalDistance
```

Section 11.4

11.4.1
```
graph = randomGraph(30, 50/((50 * 49) / 2))
```

11.4.2
```
def avgCCRandom(n, p, trials):
```

```
    totalCC = 0
    for trial in range(trials):
        graph = randomGraph(n, p)
        cc = averageClusteringCoefficient(graph)
        totalCC = totalCC + cc

    return totalCC / trials
```

Section 12.1

12.1.1 In the `Turtle` class, the `xcor`, `ycor`, and `heading` methods are accessors because they return the turtle's current position and heading without modifying the turtle. Most of the methods that we commonly use in turtle graphics are mutator methods. For example, the `forward/backward`, `left/right`, and `goto` methods (to name just a few) are mutators because they each change some aspect of the turtle.

12.1.4
```
def add(self, person):
    self._people.append(person)
    if person.isInfected():
        self._numberInfected = self._numberInfected + 1
```

12.1.9 (a)
```
class BankAccount:
    def __init__(self, balance):
        self._balance = balance

    def getBalance(self):
        return self._balance

    def deposit(self, amount):
        self._balance = self._balance + amount

    def withdraw(self, amount):
        self._balance = self._balance - amount
```

(b)
```
def main():
    amount = float(input('Initial balance? '))
    account = BankAccount(amount)
    response = input('(D)eposit, (W)ithdraw, or (Q)uit? ')
    while response != 'q':
        if response in 'dD':
            amount = float(input('Amount = '))
            account.deposit(amount)
            print('Your balance is now $' +
                  '{0:<4.2f}'.format(account.getBalance()))
        elif response in 'wW':
            amount = float(input('Amount = '))
            account.withdraw(amount)
            print('Your balance is now $' +
                  '{0:<4.2f}'.format(account.getBalance()))
        response = input('(D)eposit, (W)ithdraw, or (Q)uit? ')
```

12.1.14
```
class Dataset:
    def __init__(self):
        self._size = 0
```

```
        self._min = 0
        self._max = 0
        self._sum = 0

    def add(self, x):
        self._size += 1
        if x < self._min:
            self._min = x
        if x > self._max:
            self._max = x
        self._sum = self._sum + x

    def min(self):
        return self._min

    def max(self):
        return self._max

    def average(self):
        return self._sum / self._size

    def size(self):
        return self._size
```

Section 12.2

12.2.1
```
def round(self):
    self._a = round(self._a)
    self._b = round(self._b)
```

12.2.2
```
def main():
    tally = Pair()
    votes = input('Enter votes (q to quit): ')
    while votes != 'q':
        votes = votes.split()
        votes = Pair(int(votes[0]), int(votes[1]))
        tally = tally + votes
        votes = input('Enter votes (q to quit): ')
    print(tally)
```

12.2.3
```
def addSplit(times, split):
    if len(times) > 0:
        elapsed = times[-1].getSecond() + split
    else:
        elapsed = split
    times.append(Pair(split, elapsed))

def main():
    times = []
    addSplit(times, 23)
    addSplit(times, 43)
    addSplit(times, 24)
```

```
    for time in times:
        print(time.get())
```

12.2.9
```
def __mul__(self, pair2):
    return Pair(self[0] * pair2[0], self[1] * pair2[1])

def __truediv__(self, pair2):
    return Pair(self[0] / pair2[0], self[1] / pair2[1])
```