Creating Data Types
Outline

1 Basic Elements of a Data Type

2 Examples of Data Types
Basic Elements of a Data Type

We implement a data type as a class — the keyword `class`, followed by the class name, followed by a colon, and then a list of method definitions.

A class typically defines a constructor, instance variables (aka attributes of the class), and methods.

A constructor creates an object of the specified type and returns a reference to that object.

When a client calls a constructor, Python calls the `__init__()` method of the data type to define and initialize the instance variables, and returns a reference to the new object.

A method definition consists of its signature — the `def` keyword followed by its name, a list of parameter variables, and a colon — and its body.

By convention, the first parameter of a method is named `self`.

When a client calls a method, the `self` parameter variable references the object to be manipulated, i.e., the object that was used to invoke the method; in the case of `__init__()`, it is a reference to the newly created object.
Basic Elements of a Data Type

We implement a data type as a class — the keyword `class`, followed by the class name, followed by a colon, and then a list of method definitions.

A class typically defines a constructor, instance variables (aka attributes of the class), and methods.

A constructor creates an object of the specified type and returns a reference to that object.

When a client calls a constructor, Python calls the `__init__()` method of the data type to define and initialize the instance variables, and returns a reference to the new object.

A method definition consists of its signature — the `def` keyword followed by its name, a list of parameter variables, and a colon — and its body.

By convention, the first parameter of a method is named `self`.

When a client calls a method, the `self` parameter variable references the object to be manipulated, i.e., the object that was used to invoke the method; in the case of `__init__()`, it is a reference to the newly created object.
Basic Elements of a Data Type

We implement a data type as a class — the keyword `class`, followed by the class name, followed by a colon, and then a list of method definitions.

A class typically defines a constructor, instance variables (aka attributes of the class), and methods.

A constructor creates an object of the specified type and returns a reference to that object.

When a client calls a constructor, Python calls the `__init__()` method of the data type to define and initialize the instance variables, and returns a reference to the new object.

A method definition consists of its signature — the `def` keyword followed by its name, a list of parameter variables, and a colon — and its body.

By convention, the first parameter of a method is named `self`.

When a client calls a method, the `self` parameter variable references the object to be manipulated, i.e., the object that was used to invoke the method; in the case of `__init__()`, it is a reference to the newly created object.
Basic Elements of a Data Type

We implement a data type as a class — the keyword `class`, followed by the class name, followed by a colon, and then a list of method definitions.

A class typically defines a constructor, instance variables (aka attributes of the class), and methods.

A constructor creates an object of the specified type and returns a reference to that object.

When a client calls a constructor, Python calls the `__init__()` method of the data type to define and initialize the instance variables, and returns a reference to the new object.

A method definition consists of its signature — the `def` keyword followed by its name, a list of parameter variables, and a colon — and its body.

By convention, the first parameter of a method is named `self`.

When a client calls a method, the `self` parameter variable references the object to be manipulated, i.e., the object that was used to invoke the method; in the case of `__init__()`, it is a reference to the newly created object.
Basic Elements of a Data Type

We implement a data type as a class — the keyword `class`, followed by the class name, followed by a colon, and then a list of method definitions.

A class typically defines a constructor, instance variables (aka attributes of the class), and methods.

A constructor creates an object of the specified type and returns a reference to that object.

When a client calls a constructor, Python calls the `__init__()` method of the data type to define and initialize the instance variables, and returns a reference to the new object.

A method definition consists of its signature — the `def` keyword followed by its name, a list of parameter variables, and a colon — and its body.

By convention, the first parameter of a method is named `self`.

When a client calls a method, the `self` parameter variable references the object to be manipulated, i.e., the object that was used to invoke the method; in the case of `__init__()`, it is a reference to the newly created object.
Basic Elements of a Data Type

We implement a data type as a class — the keyword `class`, followed by the class name, followed by a colon, and then a list of method definitions.

A class typically defines a constructor, instance variables (aka attributes of the class), and methods.

A constructor creates an object of the specified type and returns a reference to that object.

When a client calls a constructor, Python calls the `__init__()` method of the data type to define and initialize the instance variables, and returns a reference to the new object.

A method definition consists of its signature — the `def` keyword followed by its name, a list of parameter variables, and a colon — and its body.

By convention, the first parameter of a method is named `self`.

When a client calls a method, the `self` parameter variable references the object to be manipulated, i.e., the object that was used to invoke the method; in the case of `__init__()`, it is a reference to the newly created object.
Basic Elements of a Data Type

We implement a data type as a class — the keyword `class`, followed by the class name, followed by a colon, and then a list of method definitions.

A class typically defines a constructor, instance variables (aka attributes of the class), and methods.

A constructor creates an object of the specified type and returns a reference to that object.

When a client calls a constructor, Python calls the `__init__()` method of the data type to define and initialize the instance variables, and returns a reference to the new object.

A method definition consists of its signature — the `def` keyword followed by its name, a list of parameter variables, and a colon — and its body.

By convention, the first parameter of a method is named `self`.

When a client calls a method, the `self` parameter variable references the object to be manipulated, i.e., the object that was used to invoke the method; in the case of `__init__()`, it is a reference to the newly created object.
Basic Elements of a Data Type

We implement a data type as a class — the keyword `class`, followed by the class name, followed by a colon, and then a list of method definitions.

A class typically defines a constructor, instance variables (aka attributes of the class), and methods.

A constructor creates an object of the specified type and returns a reference to that object.

When a client calls a constructor, Python calls the `__init__()` method of the data type to define and initialize the instance variables, and returns a reference to the new object.

A method definition consists of its signature — the `def` keyword followed by its name, a list of parameter variables, and a colon — and its body.

By convention, the first parameter of a method is named `self`.

When a client calls a method, the `self` parameter variable references the object to be manipulated, i.e., the object that was used to invoke the method; in the case of `__init__()`, it is a reference to the newly created object.
Instance variables implement the values of a data type

An instance variable belongs to a particular instance of a class, i.e., to a particular object.

By convention, instance variable names begin with an underscore.

A method typically uses three kinds of variables:

- The `self` object’s instance variables
- The method’s parameter variables
- Local variables

The key difference between functions and methods is that a method is associated with a specified object, with direct access to its instance variables.

To support the operation `str(o)`, where `o` is an object of data type `T`, we must implement the method `__str__() in T`.

A client should access a data type only through the methods in its API.
Basic Elements of a Data Type

Instance variables implement the values of a data type

An instance variable belongs to a particular instance of a class, ie, to a particular object

By convention, instance variable names begin with an underscore

A method typically uses three kinds of variables

- The self object’s instance variables
- The method’s parameter variables
- Local variables

The key difference between functions and methods is that a method is associated with a specified object, with direct access to its instance variables

To support the operation \texttt{str(o)}, where \(o\) is an object of data type \(T\), we must implement the method \texttt{__str()} in \(T\)

A client should access a data type only through the methods in its API
Basic Elements of a Data Type

Instance variables implement the values of a data type

An instance variable belongs to a particular instance of a class, ie, to a particular object

By convention, instance variable names begin with an underscore

A method typically uses three kinds of variables

- The `self` object’s instance variables
- The method’s parameter variables
- Local variables

The key difference between functions and methods is that a method is associated with a specified object, with direct access to its instance variables

To support the operation `str(o)`, where `o` is an object of data type `T`, we must implement the method `__str__() in T`

A client should access a data type only through the methods in its API
Basic Elements of a Data Type

Instance variables implement the values of a data type

An instance variable belongs to a particular instance of a class, ie, to a particular object

By convention, instance variable names begin with an underscore

A method typically uses three kinds of variables

• The self object’s instance variables
• The method’s parameter variables
• Local variables

The key difference between functions and methods is that a method is associated with a specified object, with direct access to its instance variables

To support the operation \( \texttt{str(o)} \), where \( o \) is an object of data type \( T \), we must implement the method \( \texttt{__str__} \) in \( T \)

A client should access a data type only through the methods in its API
Instance variables implement the values of a data type

An instance variable belongs to a particular instance of a class, i.e., to a particular object

By convention, instance variable names begin with an underscore

A method typically uses three kinds of variables

- The `self` object’s instance variables
- The method’s parameter variables
- Local variables

The key difference between functions and methods is that a method is associated with a specified object, with direct access to its instance variables

To support the operation `str(o)`, where `o` is an object of data type `T`, we must implement the method `__str__()` in `T`

A client should access a data type only through the methods in its API
Basic Elements of a Data Type

Instance variables implement the values of a data type

An instance variable belongs to a particular instance of a class, i.e., to a particular object

By convention, instance variable names begin with an underscore

A method typically uses three kinds of variables
  • The `self` object’s instance variables
  • The method’s parameter variables
  • Local variables

The key difference between functions and methods is that a method is associated with a specified object, with direct access to its instance variables

To support the operation `str(o)`, where `o` is an object of data type `T`, we must implement the method `__str__` in `T`

A client should access a data type only through the methods in its API
Basic Elements of a Data Type

Instance variables implement the values of a data type

An instance variable belongs to a particular instance of a class, i.e., to a particular object

By convention, instance variable names begin with an underscore

A method typically uses three kinds of variables
  • The `self` object’s instance variables
  • The method’s parameter variables
  • Local variables

The key difference between functions and methods is that a method is associated with a specified object, with direct access to its instance variables

To support the operation `str(o)`, where `o` is an object of data type `T`, we must implement the method `__str__()` in `T`

A client should access a data type only through the methods in its API
Basic Elements of a Data Type

Instance variables implement the values of a data type

An instance variable belongs to a particular instance of a class, i.e., to a particular object

By convention, instance variable names begin with an underscore

A method typically uses three kinds of variables
  • The `self` object’s instance variables
  • The method’s parameter variables
  • Local variables

The key difference between functions and methods is that a method is associated with a specified object, with direct access to its instance variables

To support the operation `str(o)`, where `o` is an object of data type `T`, we must implement the method `__str__()` in `T`

A client should access a data type only through the methods in its API
**Examples of Data Types**

A data type `Charge` for charged particles

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Charge(x0, y0, q0)</code></td>
<td>a new charge $c$ centered at $(x_0, y_0)$ with charge value $q_0$</td>
</tr>
<tr>
<td><code>c.potentialAt(x, y)</code></td>
<td>electric potential of $c$ at point $(x, y)$</td>
</tr>
<tr>
<td><code>str(c)</code></td>
<td>string representation of $c$</td>
</tr>
</tbody>
</table>
Examples of Data Types

A data type `charge` for charged particles

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Charge(x0, y0, q0)</code></td>
<td>a new charge $c$ centered at $(x_0, y_0)$ with charge value $q_0$</td>
</tr>
<tr>
<td><code>c.potentialAt(x, y)</code></td>
<td>electric potential of $c$ at point $(x, y)$</td>
</tr>
<tr>
<td><code>str(c)</code></td>
<td>string representation of $c$</td>
</tr>
</tbody>
</table>
Examples of Data Types

charge.py: Definition of Charge data type.

```python
import math
import stdio
import sys

class Charge:
    def __init__(self, x0, y0, q0):
        self._rx = x0
        self._ry = y0
        self._q = q0

    def potentialAt(self, x, y):
        COULOMB = 8.99e09
        dx = x - self._rx
        dy = y - self._ry
        r = math.sqrt(dx * dx + dy * dy)
        if r == 0.0:
            return float('inf')
        return COULOMB * self._q / r

    def __str__(self):
        result = str(self._q) + ' at ('
        result += str(self._rx) + ', ' + str(self._ry) + ')
        return result
```

Examples of Data Types

charge.py: Definition of Charge data type.

```python
import math
import stdio
import sys

class Charge:
    def __init__(self, x0, y0, q0):
        self._rx = x0
        self._ry = y0
        self._q = q0

    def potentialAt(self, x, y):
        COULOMB = 8.99e09
        dx = x - self._rx
        dy = y - self._ry
        r = math.sqrt(dx * dx + dy * dy)
        if r == 0.0:
            return float('inf')
        return COULOMB * self._q / r

    def __str__(self):
        result = str(self._q) + ' at ('
        result += str(self._rx) + ', ' + str(self._ry) + ')
        return result
```

Examples of Data Types

def _main():
    x = float(sys.argv[1])
    y = float(sys.argv[2])
    c = Charge(.51, .63, 21.3)
    stdio.writeln(c)
    stdio.writeln(c.potentialAt(x, y))

if __name__ == '__main__':
    _main()

$ python charge.py .5 .5
21.3 at (0.51, 0.63)
1.46863824819e+12
def _main():
    x = float(sys.argv[1])
    y = float(sys.argv[2])
    c = Charge(.51, .63, 21.3)
    stdio.writeln(c)
    stdio.writeln(c.potentialAt(x, y))

if __name__ == '__main__':
    _main()

$ python charge.py .5 .5
21.3 at (0.51, 0.63)
1.46863824819e+12
def _main():
    x = float(sys.argv[1])
    y = float(sys.argv[2])
    c = Charge(.51, .63, 21.3)
    stdio.writeln(c)
    stdio.writeln(c.potentialAt(x, y))

if __name__ == '__main__':
    _main()

$ python charge.py .5 .5
21.3 at (0.51, 0.63)
1.46863824819e+12
A data type `Stopwatch` representing a stopwatch

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Stopwatch()</code></td>
<td>a new stopwatch (w)</td>
</tr>
<tr>
<td>(w).elapsedTime()</td>
<td>elapsed time since (w) was created, in seconds</td>
</tr>
</tbody>
</table>
A data type `Stopwatch` representing a stopwatch

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Stopwatch()</code></td>
<td>a new stopwatch $w$</td>
</tr>
<tr>
<td><code>w.elapsedTime()</code></td>
<td>elapsed time since $w$ was created, in seconds</td>
</tr>
</tbody>
</table>
Examples of Data Types

stopwatch.py: Definition of Stopwatch data type.

```python
import stdio
import sys
import time

class Stopwatch:
    def __init__(self):
        self._creationTime = time.clock()

    def elapsedTime(self):
        return time.clock() - self._creationTime

def _main():
    n = int(sys.argv[1])
    total1 = 0.0
    watch1 = Stopwatch()
    for i in range(1, n + 1):
        total1 += i ** 2
    time1 = watch1.elapsedTime()
    total2 = 0.0
    watch2 = Stopwatch()
    for i in range(1, n + 1):
        total2 += i * i
    time2 = watch2.elapsedTime()
    stdio.writef('%f
%f
', total1 / total2, time1 / time2)

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

$ python stopwatch.py 1000000
1.000000
1.280855
Examples of Data Types

stopwatch.py: Definition of Stopwatch data type.

```python
import stdio
import sys
import time

class Stopwatch:
    def __init__(self):
        self._creationTime = time.clock()

    def elapsedTime(self):
        return time.clock() - self._creationTime

def _main():
    n = int(sys.argv[1])
    total1 = 0.0
    watch1 = Stopwatch()
    for i in range(1, n + 1):
        total1 += i ** 2
    time1 = watch1.elapsedTime()
    total2 = 0.0
    watch2 = Stopwatch()
    for i in range(1, n + 1):
        total2 += i * i
    time2 = watch2.elapsedTime()
    stdio.writef('%f\n%f\n', total1 / total2, time1 / time2)

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

$ python stopwatch.py 100000
1.000000
1.280855
import stdio
import sys
import time

class Stopwatch:
    def __init__(self):
        self._creationTime = time.clock()

    def elapsedTime(self):
        return time.clock() - self._creationTime

def _main():
    n = int(sys.argv[1])
    total1 = 0.0
    watch1 = Stopwatch()
    for i in range(1, n + 1):
        total1 += i ** 2
    time1 = watch1.elapsedTime()
    total2 = 0.0
    watch2 = Stopwatch()
    for i in range(1, n + 1):
        total2 += i * i
    time2 = watch2.elapsedTime()
    stdio.writef('%f\n%f\n', total1 / total2, time1 / time2)

if __name__ == '__main__':
    _main()
A data type `Histogram` for graphically representing distributions of data

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Histogram(n)</code></td>
<td>a new histogram ( h ) from the integer values in ( 0, 1, \ldots, n - 1 )</td>
</tr>
<tr>
<td><code>h.addDataPoint(i)</code></td>
<td>add an occurrence of integer ( i ) to ( h )</td>
</tr>
<tr>
<td><code>h.draw()</code></td>
<td>draw ( h ) to standard draw</td>
</tr>
</tbody>
</table>
A data type **Histogram** for graphically representing distributions of data

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Histogram(n)</td>
<td>a new histogram $h$ from the integer values in 0, 1, . . . , $n - 1$</td>
</tr>
<tr>
<td>h.addDataPoint(i)</td>
<td>add an occurrence of integer $i$ to $h$</td>
</tr>
<tr>
<td>h.draw()</td>
<td>draw $h$ to standard draw</td>
</tr>
</tbody>
</table>
Examples of Data Types

```python
import stdarray
import stddraw
import stdrandom
import stdstats
import sys

class Histogram:
    def __init__(self, n):
        self._freqCounts = stdarray.create1D(n, 0)

    def addDataPoint(self, i):
        self._freqCounts[i] += 1

    def draw(self):
        stddraw.setYscale(0, max(self._freqCounts))
        stdstats.plotBars(self._freqCounts)

def _main():
    n = int(sys.argv[1])
    p = float(sys.argv[2])
    trialCount = int(sys.argv[3])
    histogram = Histogram(n + 1)
    for trial in range(trialCount):
        heads = stdrandom.binomial(n, p)
        histogram.addDataPoint(heads)
    stddraw.setCanvasSize(500, 200)
    histogram.draw()
    stddraw.show()

if __name__ == '__main__':
    _main()
```
import stdarray
import stddraw
import stdrandom
import stdstats
import sys

class Histogram :
  def __init__(self, n):
    self._freqCounts = stdarray.create1D(n, 0)

  def addDataPoint(self, i):
    self._freqCounts[i] += 1

  def draw(self):
    stddraw.setYscale(0, max(self._freqCounts))
    stdstats.plotBars(self._freqCounts)

def _main():
  n = int(sys.argv[1])
  p = float(sys.argv[2])
  trialCount = int(sys.argv[3])
  histogram = Histogram(n + 1)
  for trial in range(trialCount):
    heads = stdrandom.binomial(n, p)
    histogram.addDataPoint(heads)
  stddraw.setCanvasSize(500, 200)
  histogram.draw()
  stddraw.show()

if __name__ == '__main__':
  _main()
Examples of Data Types

$ python histogram.py 50 .5 1000000

$ python histogram.py 50 .2 1000000

$ python histogram.py 50 .8 1000000
Examples of Data Types

$ python histogram.py 50 .5 1000000

$ python histogram.py 50 .2 1000000

$ python histogram.py 50 .8 1000000
Examples of Data Types

$ python histogram.py 50 .5 1000000

$ python histogram.py 50 .2 1000000

$ python histogram.py 50 .8 1000000
Examples of Data Types

$ python histogram.py 50 .5 1000000

$ python histogram.py 50 .2 1000000

$ python histogram.py 50 .8 1000000
A data type `Turtle` for producing turtle graphics\(^1\)

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Turtle(x0, y0, a0)</code></td>
<td>a new turtle ( t ) at ((x_0, y_0)) facing ( a_0 ) degrees from the ( x )-axis</td>
</tr>
<tr>
<td><code>t.turnLeft(delta)</code></td>
<td>instruct ( t ) to turn left (counter-clockwise) by ( delta ) degrees</td>
</tr>
<tr>
<td><code>t.goForward(step)</code></td>
<td>instruct ( t ) to move forward distance ( step ), drawing a line</td>
</tr>
</tbody>
</table>

---

\(^1\)Turtle graphics was part of the original Logo programming language developed by Wally Feurzig and Seymour Papert in 1966 for introducing programming to kids
A data type `Turtle` for producing turtle graphics\(^1\)

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Turtle(x_0, y_0, a_0)</code></td>
<td>a new turtle (t) at ((x_0, y_0)) facing (a_0) degrees from the (x)-axis</td>
</tr>
<tr>
<td><code>t.turnLeft(delta)</code></td>
<td>instruct (t) to turn left (counter-clockwise) by (delta) degrees</td>
</tr>
<tr>
<td><code>t.goForward(step)</code></td>
<td>instruct (t) to move forward distance (step), drawing a line</td>
</tr>
</tbody>
</table>

---

\(^1\) Turtle graphics was part of the original Logo programming language developed by Wally Feurzig and Seymour Papert in 1966 for introducing programming to kids
import math
import stddraw
import sys

class Turtle:
    def __init__(self, x, y, angle):
        self._x = x
        self._y = y
        self._angle = angle

    def turnLeft(self, delta):
        self._angle += delta

    def goForward(self, step):
        oldx = self._x
        oldy = self._y
        self._x += step * math.cos(math.radians(self._angle))
        self._y += step * math.sin(math.radians(self._angle))
        stddraw.line(oldx, oldy, self._x, self._y)

def _main():
    n = int(sys.argv[1])
    turtle = Turtle(.5, .0, 180.0 / n)
    stepSize = math.sin(math.radians(180.0 / n))
    for i in range(n):
        turtle.goForward(stepSize)
        turtle.turnLeft(360.0 / n)
    stddraw.show()

if __name__ == '__main__':
    _main()
Examples of Data Types

turtle.py: Definition of Turtle data type.

```python
import math
import stddraw
import sys

class Turtle:
    def __init__(self, x, y, angle):
        self._x = x
        self._y = y
        self._angle = angle

    def turnLeft(self, delta):
        self._angle += delta

    def goForward(self, step):
        oldx = self._x
        oldy = self._y
        self._x += step * math.cos(math.radians(self._angle))
        self._y += step * math.sin(math.radians(self._angle))
        stddraw.line(oldx, oldy, self._x, self._y)

def _main():
    n = int(sys.argv[1])
    turtle = Turtle(.5, .0, 180.0 / n)
    stepSize = math.sin(math.radians(180.0 / n))
    for i in range(n):
        turtle.goForward(stepSize)
        turtle.turnLeft(360.0 / n)
    stddraw.show()

if __name__ == '__main__':
    _main()
```
Examples of Data Types

$ python turtle.py 3

$ python turtle.py 7

$ python turtle.py 30
Examples of Data Types

$ python turtle.py 3

$ python turtle.py 7

$ python turtle.py 30
Examples of Data Types

$ python turtle.py 3

$ python turtle.py 7

$ python turtle.py 30
Examples of Data Types

$ python turtle.py 3

$ python turtle.py 7

$ python turtle.py 30
koch.py: Accept integer \( n \) as a command-line argument and plot a Koch curve of order \( n \).

```python
import stddraw
import sys
from turtle import Turtle

def koch(n, stepSize, myTurtle):
    if n == 0:
        myTurtle.goForward(stepSize)
        return
    koch(n - 1, stepSize, myTurtle)
    myTurtle.turnLeft(60.0)
    koch(n - 1, stepSize, myTurtle)
    myTurtle.turnLeft(-120.0)
    koch(n - 1, stepSize, myTurtle)
    myTurtle.turnLeft(60.0)
    koch(n - 1, stepSize, myTurtle)

def main():
    n = int(sys.argv[1])
    stddraw.setCanvasSize(512, 256)
    stddraw.setYscale(-.1, 0.4)
    stddraw.setPenRadius(0.0)
    stepSize = 1.0 / (3.0 ** n)
    myTurtle = Turtle(0.0, 0.0, 0.0)
    koch(n, stepSize, myTurtle)
    stddraw.show()

if __name__ == '__main__':
    main()
```
Examples of Data Types

koch.py: Accept integer $n$ as a command-line argument and plot a Koch curve of order $n$.

```python
import stddraw
import sys
from turtle import Turtle

def koch(n, stepSize, myTurtle):
    if n == 0:
        myTurtle.goForward(stepSize)
        return
    koch(n - 1, stepSize, myTurtle)
    myTurtle.turnLeft(60.0)
    koch(n - 1, stepSize, myTurtle)
    myTurtle.turnLeft(-120.0)
    koch(n - 1, stepSize, myTurtle)
    myTurtle.turnLeft(60.0)
    koch(n - 1, stepSize, myTurtle)

def main():
    n = int(sys.argv[1])
    stddraw.setCanvasSize(512, 256)
    stddraw.setYscale(-.1, 0.4)
    stddraw.setPenRadius(0.0)
    stepSize = 1.0 / (3.0 ** n)
    myTurtle = Turtle(0.0, 0.0, 0.0)
    koch(n, stepSize, myTurtle)
    stddraw.show()

if __name__ == '__main__':
    main()
```
Examples of Data Types

$ python koch.py 0

$ python koch.py 1

$ python koch.py 2

$ python koch.py 3

$ python koch.py 4
Examples of Data Types

```
$ python koch.py 0
```

```
$ python koch.py 1
```

```
$ python koch.py 2
```

```
$ python koch.py 3
```

```
$ python koch.py 4
```
Examples of Data Types

$ python koch.py 0

$ python koch.py 1

$ python koch.py 2

$ python koch.py 3

$ python koch.py 4
Examples of Data Types

$ python koch.py 0

$ python koch.py 1

$ python koch.py 2

$ python koch.py 3

$ python koch.py 4
Examples of Data Types

```
$ python koch.py 0
```

```
$ python koch.py 1
```

```
$ python koch.py 2
```

```
$ python koch.py 3
```

```
$ python koch.py 4
```
Examples of Data Types

$ python koch.py 0

$ python koch.py 1

$ python koch.py 2

$ python koch.py 3

$ python koch.py 4
Examples of Data Types

spiral.py: Accept command-line arguments $n$ (an integer specifying a number of sides), $wraps$ (an integer specifying a wrap count), and $decay$ (a float specifying a decay factor), and draw a spiral with those characteristics.

```python
import math
import stddraw
import sys
from turtle import Turtle

def main():
    n = int(sys.argv[1])
    wraps = int(sys.argv[2])
    decay = float(sys.argv[3])
    angle = 360.0 / n
    step = math.sin(math.radians(angle/2.0))
    turtle = Turtle(0.5, 0, angle/2.0)
    stddraw.setPenRadius(0.0)
    for i in range(wraps * n):
        step /= decay
        turtle.forward(step)
        turtle.turnLeft(angle)
    stddraw.show()

if __name__ == '__main__':
    main()
```
Examples of Data Types

spiral.py: Accept command-line arguments \( n \) (an integer specifying a number of sides), \( \text{wraps} \) (an integer specifying a wrap count), and \( \text{decay} \) (a float specifying a decay factor), and draw a spiral with those characteristics.

```python
import math
import stddraw
import sys
from turtle import Turtle

def main():
    n = int(sys.argv[1])
    wraps = int(sys.argv[2])
    decay = float(sys.argv[3])
    angle = 360.0 / n
    step = math.sin(math.radians(angle/2.0))
    turtle = Turtle(0.5, 0, angle/2.0)
    stddraw.setPenRadius(0.0)
    for i in range(wraps * n):
        step /= decay
        turtle.goForward(step)
        turtle.turnLeft(angle)
    stddraw.show()

if __name__ == '__main__':
    main()
```
Examples of Data Types

$ python spiral.py 3 10 1.2

$ python spiral.py 1440 10 1.0004
Examples of Data Types

```bash
$ python spiral.py 3 10 1.2

$ python spiral.py 1440 10 1.0004
```
Examples of Data Types

$ python spiral.py 3 10 1.2

$ python spiral.py 1440 10 1.0004
Examples of Data Types

drunk.py: Accept as command-line arguments an integer `stepCount` specifying the number of iterations, and a float `stepSize` specifying a step size, create a Turtle object, and have it make `stepCount` random steps of size `stepSize`.

```python
import stddraw
import stdrandom
import sys
from turtle import Turtle

def main():
    stepCount = int(sys.argv[1])
    stepSize = float(sys.argv[2])
    stddraw.setPenRadius(0.0)
    myTurtle = Turtle(0.5, 0.5, 0.0)
    for i in range(stepCount):
        myTurtle.turnLeft(360.0 * stdrandom.uniformFloat(0.0, 360.0))
        myTurtle.goForward(stepSize)
        stddraw.show(0.0)
        stddraw.show()

if __name__ == '__main__':
    main()
```
Examples of Data Types

drunk.py: Accept as command-line arguments an integer $stepCount$ specifying the number of iterations, and a float $stepSize$ specifying a step size, create a Turtle object, and have it make $stepCount$ random steps of size $stepSize$.

```python
import stddraw
import stdrandom
import sys
from turtle import Turtle

def main():
    stepCount = int(sys.argv[1])
    stepSize = float(sys.argv[2])
    stddraw.setPenRadius(0.0)
    myTurtle = Turtle(0.5, 0.5, 0.0)
    for i in range(stepCount):
        myTurtle.turnLeft(360.0 * stdrandom.uniformFloat(0.0, 360.0))
        myTurtle.goForward(stepSize)
        stddraw.show(0.0)
    stddraw.show()

if __name__ == '__main__':
    main()
```
Examples of Data Types

```
$ python drunk.py 10000 .01
```
Examples of Data Types

$ python drunk.py 10000 .01
Examples of Data Types

drunks.py: Accept as command-line arguments an integer \texttt{turtleCount}, an integer \texttt{stepCount}, and a float \texttt{stepSize}, create \texttt{turtleCount} Turtle objects, and have them make \texttt{stepCount} random steps of size \texttt{stepSize}.

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

def main():
    turtleCount = int(sys.argv[1])
    stepCount = int(sys.argv[2])
    stepSize = float(sys.argv[3])
    stddraw.setPenRadius(0.0)
    turtles = stdarray.create1D(turtleCount)
    for i in range(turtleCount):
        x = stdrandom.uniformFloat(0.0, 1.0)
        y = stdrandom.uniformFloat(0.0, 1.0)
        turtles[i] = Turtle(x, y, 0.0)
    for j in range(stepCount):
        for i in range(turtleCount):
            turtles[i].turnLeft(stdrandom.uniformFloat(0.0, 360.0))
            turtles[i].goForward(stepSize)
        stddraw.show(0.0)
    stddraw.show()

if __name__ == '__main__':
    main()
```
import stdarray
import stddraw
import stdrandom
import sys
from turtle import Turtle

def main():
    turtleCount = int(sys.argv[1])
    stepCount = int(sys.argv[2])
    stepSize = float(sys.argv[3])
    stddraw.setPenRadius(0.0)
    turtles = stdarray.create1D(turtleCount)
    for i in range(turtleCount):
        x = stdrandom.uniformFloat(0.0, 1.0)
        y = stdrandom.uniformFloat(0.0, 1.0)
        turtles[i] = Turtle(x, y, 0.0)
    for j in range(stepCount):
        for i in range(turtleCount):
            turtles[i].turnLeft(stdrandom.uniformFloat(0.0, 360.0))
            turtles[i].goForward(stepSize)
            stddraw.show(0.0)
    stddraw.show()

if __name__ == '__main__':
    main()
Examples of Data Types

$ python drunk.py 20 5000 .005
Examples of Data Types

$ python drunk.py 20 5000 .005
Examples of Data Types

A complex number \( z \) in the cartesian form is expressed as 
\[ z = x + yi, \] 
where \( x \) (the real part) and \( y \) (the imaginary part) are real numbers and \( i = \sqrt{-1} \)

Complex arithmetic

- Conjugate: \((x + yi)^* = x - yi\)
- Addition: \((x + yi) + (v + wi) = (x + v) + (y + w)i\)
- Multiplication: \((x + yi) \times (v + wi) = (xv - yw) + (yv + xw)i\)
- Magnitude: \(|x + yi| = \sqrt{x^2 + y^2}\)

A data type Complex for representing complex numbers

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex(x, y)</td>
<td>a new complex object ( c ) with value ( x + yi )</td>
</tr>
<tr>
<td>c.re()</td>
<td>real part of ( c )</td>
</tr>
<tr>
<td>c.im()</td>
<td>imaginary part of ( c )</td>
</tr>
<tr>
<td>c.conjugate()</td>
<td>conjugate of ( c )</td>
</tr>
<tr>
<td>c + d</td>
<td>sum of ( c ) and ( d )</td>
</tr>
<tr>
<td>c * d</td>
<td>product of ( c ) and ( d )</td>
</tr>
<tr>
<td>abs(c)</td>
<td>magnitude of ( c )</td>
</tr>
<tr>
<td>str(c)</td>
<td>string representation of ( c )</td>
</tr>
</tbody>
</table>
Examples of Data Types

A complex number $z$ in the cartesian form is expressed as $z = x + yi$, where $x$ (the real part) and $y$ (the imaginary part) are real numbers and $i = \sqrt{-1}$.

Complex arithmetic
- Conjugate: $(x + yi)^* = x - yi$
- Addition: $(x + yi) + (v + wi) = (x + v) + (y + w)i$
- Multiplication: $(x + yi) \times (v + wi) = (xv - yw) + (yi + xv)i$
- Magnitude: $|x + yi| = \sqrt{x^2 + y^2}$

A data type `Complex` for representing complex numbers

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex(x, y)</td>
<td>a new complex object $c$ with value $x + yi$</td>
</tr>
<tr>
<td>c.re()</td>
<td>real part of $c$</td>
</tr>
<tr>
<td>c.im()</td>
<td>imaginary part of $c$</td>
</tr>
<tr>
<td>c.conjugate()</td>
<td>conjugate of $c$</td>
</tr>
<tr>
<td>c + d</td>
<td>sum of $c$ and $d$</td>
</tr>
<tr>
<td>c * d</td>
<td>product of $c$ and $d</td>
</tr>
<tr>
<td>abs(c)</td>
<td>magnitude of $c$</td>
</tr>
<tr>
<td>str(c)</td>
<td>string representation of $c$</td>
</tr>
</tbody>
</table>
Examples of Data Types

A complex number \( z \) in the cartesian form is expressed as \( z = x + yi \), where \( x \) (the real part) and \( y \) (the imaginary part) are real numbers and \( i = \sqrt{-1} \)

Complex arithmetic
- **Conjugate:** \((x + yi)^* = x - yi\)
- **Addition:** \((x + yi) + (v + wi) = (x + v) + (y + w)i\)
- **Multiplication:** \((x + yi) \times (v + wi) = (xv - yw) + (yv + xw)i\)
- **Magnitude:** \(|x + yi| = \sqrt{x^2 + y^2}\)

A data type `Complex` for representing complex numbers

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex(x, y)</td>
<td>a new complex object ( c ) with value ( x + yi )</td>
</tr>
<tr>
<td>c.re()</td>
<td>real part of ( c )</td>
</tr>
<tr>
<td>c.im()</td>
<td>imaginary part of ( c )</td>
</tr>
<tr>
<td>c.conjugate()</td>
<td>conjugate of ( c )</td>
</tr>
<tr>
<td>c + d</td>
<td>sum of ( c ) and ( d )</td>
</tr>
<tr>
<td>c * d</td>
<td>product of ( c ) and ( d )</td>
</tr>
<tr>
<td>abs(c)</td>
<td>magnitude of ( c )</td>
</tr>
<tr>
<td>str(c)</td>
<td>string representation of ( c )</td>
</tr>
</tbody>
</table>
Examples of Data Types

A complex number $z$ in the cartesian form is expressed as $z = x + yi$, where $x$ (the real part) and $y$ (the imaginary part) are real numbers and $i = \sqrt{-1}$.

Complex arithmetic
- Conjugate: $(x + yi)^* = x - yi$
- Addition: $(x + yi) + (v + wi) = (x + v) + (y + w)i$
- Multiplication: $(x + yi) \times (v + wi) = (xv - yw) + (yv + xw)i$
- Magnitude: $|x + yi| = \sqrt{x^2 + y^2}$

A data type `Complex` for representing complex numbers

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex(x, y)</td>
<td>a new complex object $c$ with value $x + yi$</td>
</tr>
<tr>
<td>c.re()</td>
<td>real part of $c$</td>
</tr>
<tr>
<td>c.im()</td>
<td>imaginary part of $c$</td>
</tr>
<tr>
<td>c.conjugate()</td>
<td>conjugate of $c$</td>
</tr>
<tr>
<td>c + d</td>
<td>sum of $c$ and $d$</td>
</tr>
<tr>
<td>c * d</td>
<td>product of $c$ and $d</td>
</tr>
<tr>
<td>abs(c)</td>
<td>magnitude of $c$</td>
</tr>
<tr>
<td>str(c)</td>
<td>string representation of $c$</td>
</tr>
</tbody>
</table>
Examples of Data Types

complex.py: Definition of Complex data type.

```python
import math
import stdio

class Complex:
    def __init__(self, re = 0.0, im = 0.0):
        self._re = re
        self._im = im

    def re(self):
        return self._re

    def im(self):
        return self._im

    def conjugate(self):
        return Complex(self._re, -self._im)

    def __add__(self, other):
        re = self._re + other._re
        im = self._im + other._im
        return Complex(re, im)

    def __mul__(self, other):
        re = self._re * other._re - self._im * other._im
        im = self._re * other._im + self._im * other._re
        return Complex(re, im)
```

import math
import stdio

class Complex:
    def __init__(self, re = 0.0, im = 0.0):
        self._re = re
        self._im = im

    def re(self):
        return self._re

    def im(self):
        return self._im

    def conjugate(self):
        return Complex(self._re, -self._im)

    def __add__(self, other):
        re = self._re + other._re
        im = self._im + other._im
        return Complex(re, im)

    def __mul__(self, other):
        re = self._re * other._re - self._im * other._im
        im = self._re * other._im + self._im * other._re
        return Complex(re, im)
Examples of Data Types

def __abs__(self):
    return math.sqrt(self._re * self._re + self._im * self._im)

def __str__(self):
    return str(self._re) + ' + ' + str(self._im) + 'i'

def _main():
    z0 = Complex(1.0, 1.0)
    z = z0
    z = z * z + z0
    z = z * z + z0
    stdio.writeln(z)

if __name__ == '__main__':
    _main()

$ python complex.py
-7.0 + 7.0i
Examples of Data Types

def __abs__(self):
    return math.sqrt(self._re * self._re + self._im * self._im)

def __str__(self):
    return str(self._re) + ' + ' + str(self._im) + 'i'

def _main():
    z0 = Complex(1.0, 1.0)
    z = z0
    z = z * z + z0
    z = z * z + z0
    stdio.writeln(z)

if __name__ == '__main__':
    _main()

$ python complex.py
-7.0 + 7.0i
Examples of Data Types

def __abs__(self):
    return math.sqrt(self._re * self._re + self._im * self._im)

def __str__(self):
    return str(self._re) + ' + ' + str(self._im) + 'i'

def _main():
    z0 = Complex(1.0, 1.0)
    z = z0
    z = z * z + z0
    z = z * z + z0
    stdio.writeln(z)

if __name__ == '__main__':
    _main()

$ python complex.py
-7.0 + 7.0i
Examples of Data Types

mandelbrot.py: Accept float command-line arguments \( x_c, y_c \), and \( size \) that specify the center and size of a square region of interest, and make a digital image showing the result of sampling the Mandelbrot set in that region. The algorithm is as follows:

- Consider \( z_0, z_1, \ldots, z_t \), where \( z_{t+1} = z_t^2 + z_0 \)
- If the sequence \( |z_t| \) diverges to infinity, then \( z_0 \) is not in the Mandelbrot set; if the sequence is bounded, then \( z_0 \) is in the set

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

def mandel(z0, limit):
    z = z0
    for t in range(limit):
        if abs(z) > 2.0:
            return t
        z = z * z + z0
    return limit
```
Examples of Data Types

mandelbrot.py: Accept float command-line arguments \( xc, yc, \) and \( size \) that specify the center and size of a square region of interest, and make a digital image showing the result of sampling the Mandelbrot set in that region. The algorithm is as follows:

- Consider \( z_0, z_1, \ldots, z_t \), where \( z_{t+1} = z_t^2 + z_0 \)
- If the sequence \( |z_t| \) diverges to infinity, then \( z_0 \) is not in the Mandelbrot set; if the sequence is bounded, then \( z_0 \) is in the set

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

def mandel(z0, limit):
    z = z0
    for t in range(limit):
        if abs(z) > 2.0:
            return t
        z = z * z + z0
    return limit
```
def main():
    MAX = 255
    n = int(sys.argv[1])
    xc = float(sys.argv[2])
    yc = float(sys.argv[3])
    size = float(sys.argv[4])
    pic = 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 = MAX - mandel(z0, MAX)
            color = Color(gray, gray, gray)
            pic.set(col, n - 1 - row, color)
    stddraw.setCanvasSize(n, n)
    stddraw.picture(pic)
    stddraw.show()

if __name__ == '__main__':
    main()
def main():
    MAX = 255
    n = int(sys.argv[1])
    xc = float(sys.argv[2])
    yc = float(sys.argv[3])
    size = float(sys.argv[4])
    pic = 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 = MAX - mandel(z0, MAX)
            color = Color(gray, gray, gray)
            pic.set(col, n - 1 - row, color)
    stddraw.setCanvasSize(n, n)
    stddraw.picture(pic)
    stddraw.show()

if __name__ == '__main__':
    main()
Examples of Data Types

$ python mandelbrot.py 512 -.5 0 2

$ python mandelbrot.py 512 .1015 -.633 .01
Examples of Data Types

$ python mandelbrot.py 512 -0.5 0 2

$ python mandelbrot.py 512 0.1015 -0.633 0.01
Examples of Data Types

$ python mandelbrot.py 512 -.5 0 2

$ python mandelbrot.py 512 .1015 -.633 .01
A data type `StockAccount` for maintaining customer accounts containing shares of various stocks

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StockAccount(filename)</td>
<td>a new account <em>a</em>, created from data from file <em>filename</em></td>
</tr>
<tr>
<td>a.valueOf()</td>
<td>total value of <em>a</em></td>
</tr>
<tr>
<td>a.write(filename)</td>
<td>write <em>a</em> to <em>filename</em></td>
</tr>
<tr>
<td>a.writeReport()</td>
<td>write to standard output a detailed report for <em>a</em></td>
</tr>
</tbody>
</table>
A data type `StockAccount` for maintaining customer accounts containing shares of various stocks

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>StockAccount(filename)</code></td>
<td>a new account ( a ), created from data from file ( filename )</td>
</tr>
<tr>
<td>( a ).valueOf()</td>
<td>total value of ( a )</td>
</tr>
<tr>
<td>( a ).write(filename)</td>
<td>write ( a ) to ( filename )</td>
</tr>
<tr>
<td>( a ).writeReport()</td>
<td>write to standard output a detailed report for ( a )</td>
</tr>
</tbody>
</table>
Examples of Data Types

stockaccount.py: Accept as a command-line argument the name of a file, read from the file to create a StockAccount object, and write a report to standard output.

```python
import stdarray
import stdio
import stockquote
import sys
from instream import InStream
from outstream import OutStream

class StockAccount:
    def __init__(self, filename):
        inStream = InStream(filename)
        self._name = inStream.readLine()
        self._cash = inStream.readFloat()
        self._stockCount = inStream.readInt()
        self._stocks = stdarray.create1D(self._stockCount, 0)
        self._shares = stdarray.create1D(self._stockCount, 0)
        for i in range(self._stockCount):
            self._shares[i] = inStream.readInt()
            self._stocks[i] = inStream.readString()

    def valueOf(self):
        total = self._cash
        for i in range(self._stockCount):
            price = stockquote.priceOf(self._stocks[i])
            amount = self._shares[i]
            total += amount * price
        return total
```
Examples of Data Types

stockaccount.py: Accept as a command-line argument the name of a file, read from the file to create a StockAccount object, and write a report to standard output.

```python
import stdarray
import stdio
import stockquote
import sys
from instream import InStream
from outstream import OutStream

class StockAccount:
    def __init__(self, filename):
        inStream = InStream(filename)
        self._name = inStream.readLine()
        self._cash = inStream.readFloat()
        self._stockCount = inStream.readInt()
        self._stocks = stdarray.create1D(self._stockCount, 0)
        self._shares = stdarray.create1D(self._stockCount, 0)
        for i in range(self._stockCount):
            self._shares[i] = inStream.readInt()
            self._stocks[i] = inStream.readString()

    def valueOf(self):
        total = self._cash
        for i in range(self._stockCount):
            price = stockquote.priceOf(self._stocks[i])
            amount = self._shares[i]
            total += amount * price
        return total
```
Examples of Data Types

```python
def write(self, filename):
    outStream = OutStream(filename)
    outStream.writeln(self._name)
    outStream.writeln(self._cash)
    outStream.writeln(self._stockCount)
    for i in range(self._stockCount):
        outStream.writef('%3d', self._shares[i])
        outStream.write(' ')
        outStream.writeln(self._stocks[i])

def writeReport(self):
    stdio.writeln(self._name)
    total = self._cash
    for i in range(self._stockCount):
        amount = self._shares[i]
        price = stockquote.priceOf(self._stocks[i])
        total += amount * price
        stdio.writef('%4d %4s ', amount, self._stocks[i])
        stdio.writef(' %7.2 f %9.2 f
', price, amount * price)
    stdio.writef('%21 s %10.2 f
', 'Cash:', self._cash)
    stdio.writef('%21 s %10.2 f
', 'Total:', total)

def _main():
    acct = StockAccount(sys.argv[1])
    acct.writeReport()

if __name__ == '__main__':
    _main()
```
Examples of Data Types

def write(self, filename):
    outStream = OutStream(filename)
    outStream.writeln(self._name)
    outStream.writeln(self._cash)
    outStream.writeln(self._stockCount)
    for i in range(self._stockCount):
        outStream.writef('%3d', self._shares[i])
        outStream.write(' ')
        outStream.writeln(self._stocks[i])

def writeReport(self):
    stdio.writeln(self._name)
    total = self._cash
    for i in range(self._stockCount):
        amount = self._shares[i]
        price = stockquote.priceOf(self._stocks[i])
        total += amount * price
        stdio.writef('%4d %4s ', amount, self._stocks[i])
        stdio.writef(' %7.2f %9.2f
', price, amount * price)
    stdio.writeln(' %21s %10.2f
', 'Cash:', self._cash)
    stdio.writeln(' %21s %10.2f
', 'Total:', total)

def _main():
    acct = StockAccount(sys.argv[1])
    acct.writeReport()

if __name__ == '__main__':
    _main()
Examples of Data Types

$ more turing.txt
Turing, Alan
10.24
4
100 ADBE
25 GOOG
97 IBM
250 MSFT

$ python stockaccount.py turing.txt
Turing, Alan
  100  ADBE     91.32     9132.00
  25   GOOG    740.34   18508.50
  97    IBM    135.21   13115.37
 250   MSFT    54.10   13525.00
    Cash:    10.24
   Total:  54291.11
Examples of Data Types

$ more turing.txt
Turing, Alan
10.24
4
100 ADBE
25 GOOG
97 IBM
250 MSFT

$ python stockaccount.py turing.txt
Turing, Alan
100 ADBE  91.32  9132.00
25 GOOG  740.34  18508.50
97 IBM  135.21  13115.37
250 MSFT  54.10  13525.00

Cash:  10.24
Total:  54291.11
Examples of Data Types

$ more turing.txt
Turing, Alan
10.24
4
100 ADBE
25 GOOG
97 IBM
250 MSFT

$ python stockaccount.py turing.txt
Turing, Alan
100 ADBE 91.32 9132.00
25 GOOG 740.34 18508.50
97 IBM 135.21 13115.37
250 MSFT 54.10 13525.00
Cash: 10.24
Total: 54291.11