Using Data Types
Outline

1. Methods

2. Basic Data Types

3. Collection Data Types

4. User-Defined Data Types
Methods

A method is a function associated with a specific object (and, by extension, with the type of that object).

A method corresponds to a data-type operation.

We call (or invoke) a method using a variable name, followed by the dot operator (.), followed by the method name, followed by its arguments separated by commas and enclosed in parentheses.

```python
>>> import stdio
>>> x, y, z = 200, 300, 600
>>> xbits, ybits, zbits = x.bit_length(), y.bit_length(), z.bit_length()
>>> stdio.writeln(xbits)
8
>>> stdio.writeln(ybits)
9
>>> stdio.writeln(zbits)
10
```
A method is a function associated with a specific object (and, by extension, with the type of that object)
A method is a function associated with a specific object (and, by extension, with the type of that object)

A method corresponds to a data-type operation
Methods

A method is a function associated with a specific object (and, by extension, with the type of that object).

A method corresponds to a data-type operation.

We call (or invoke) a method using a variable name, followed by the dot operator (.), followed by the method name, followed by its arguments separated by commas and enclosed in parentheses.
Methods

A method is a function associated with a specific object (and, by extension, with the type of that object)

A method corresponds to a data-type operation

We call (or invoke) a method using a variable name, followed by the dot operator (.), followed by the method name, followed by its arguments separated by commas and enclosed in parentheses

```python
>>> import stdio
>>> x, y, z = 200, 300, 600
>>> xbits, ybits, zbits = x.bit_length(), y.bit_length(), z.bit_length()
>>> stdio.writeln(xbits)
8
>>> stdio.writeln(ybits)
9
>>> stdio.writeln(zbits)
10
```
Basic Data Types

Methods in the built-in `int` data type

```
>>> dir ( int )
['bit_length', 'conjugate']
```

Methods in the built-in `float` data type

```
>>> dir ( float )
['as_integer_ratio', 'conjugate', 'fromhex', 'hex', 'is_integer']
```

Methods in the built-in `bool` data type

```
>>> dir ( bool )
['bit_length', 'conjugate']
```

Methods in the built-in `str` data type

```
>>> dir ( str )
```
Basic Data Types

Methods in the built-in `int` data type

```
>>> dir(int)
['bit_length', 'conjugate']
```
Basic Data Types

Methods in the built-in `int` data type

```python
>>> dir(int)
['bit_length', 'conjugate']
```

Methods in the built-in `float` data type

```python
>>> dir(float)
['as_integer_ratio', 'conjugate', 'fromhex', 'hex', 'is_integer']
```
Basic Data Types

Methods in the built-in `int` data type

```python
>>> dir(int)
['bit_length', 'conjugate']
```

Methods in the built-in `float` data type

```python
>>> dir(float)
['as_integer_ratio', 'conjugate', 'fromhex', 'hex', 'is_integer']
```

Methods in the built-in `bool` data type

```python
>>> dir(bool)
['bit_length', 'conjugate']
```
Basic Data Types

Methods in the built-in \texttt{int} data type

```python
>>> dir(int)
['bit_length', 'conjugate']
```

Methods in the built-in \texttt{float} data type

```python
>>> dir(float)
['as_integer_ratio', 'conjugate', 'fromhex', 'hex', 'is_integer']
```

Methods in the built-in \texttt{bool} data type

```python
>>> dir(bool)
['bit_length', 'conjugate']
```

Methods in the built-in \texttt{str} data type

```python
>>> dir(str)
['capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format',
  'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join',
  'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition',
  'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate',
  'upper', 'zfill']
```
Basic Data Types

Program: potentialgene.py

• Command-line input: dna (str)
• Standard output: whether dna corresponds to a potential gene or not

```
$ python3 potentialgene.py ATGCGCCTGCGTCTGTACTAG
True

$ python3 potentialgene.py ATGCGCTGCGTCTGTACTAG
False
```
Basic Data Types

Program: potentialgene.py
Program:  potentialgene.py

- Command-line input:  dna (str)
Program: potentialgene.py

- Command-line input: dna (str)
- Standard output: whether dna corresponds to a potential gene or not
Program: potentialgene.py

- Command-line input: dna (str)
- Standard output: whether dna corresponds to a potential gene or not

```bash
$ python3 potentialgene.py ATGCGCCTGCGTCTGTACTAG
True
$ python3 potentialgene.py ATGCGCTGCGTCTGTACTAG
False
$ 
```
```
import stdio
import sys

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

def _isPotentialGene(dna):
    ATG = 'ATG  
    TAA , TAG , TGA = 'TAA ', 'TAG ', 'TGA ' 
    if len(dna) % 3 != 0:
        return False
    if not dna.startswith(ATG):
        return False
    for i in range(len(dna) - 3):
        if i % 3 == 0:
            codon = dna[i:i+3]
            if codon == TAA or codon == TAG or codon == TGA:
                return False
    return dna.endswith(TAA) or dna.endswith(TAG) or dna.endswith(TGA)

if __name__ == '__main__':
    main()
```
import stdio
import sys

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

def _isPotentialGene(dna):
    ATG = 'ATG'
    TAA, TAG, TGA = 'TAA', 'TAG', 'TGA'
    if len(dna) % 3 != 0:
        return False
    if not dna.startswith(ATG):
        return False
    for i in range(len(dna) - 3):
        if i % 3 == 0:
            codon = dna[i:i + 3]
            if codon == TAA or codon == TAG or codon == TGA:
                return False
    return dna.endswith(TAA) or dna.endswith(TAG) or dna.endswith(TGA)

if __name__ == '__main__':
    main()
**Collection Data Types**

Methods in the built-in `list` data type:

```
>>> dir(list)
['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Methods in the built-in `tuple` data type:

```
>>> dir(tuple)
['count', 'index']
```

Methods in the built-in `dict` data type:

```
>>> dir(dict)
['clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

Methods in the built-in `set` data type:

```
>>> dir(set)
['add', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersection', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric_difference', 'symmetric_difference_update', 'union', 'update']
```
Collection Data Types

Methods in the built-in list data type

```python
>>> dir(list)
['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```
Methods in the built-in `list` data type

```~/.workspace/ipp/programs
>>> dir(list)
['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```
Collection Data Types

Methods in the built-in list data type

```python
>>> dir(list)
['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Methods in the built-in tuple data type

```python
>>> dir(tuple)
['count', 'index']
```

Methods in the built-in dict data type

```python
>>> dir(dict)
['clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```
## Collection Data Types

### Methods in the built-in list data type

```python
>>> import list
>>> dir(list)
['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

### Methods in the built-in tuple data type

```python
>>> import tuple
>>> dir(tuple)
['count', 'index']
```

### Methods in the built-in dict data type

```python
>>> import dict
>>> dir(dict)
['clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

### Methods in the built-in set data type

```python
>>> import set
>>> dir(set)
['add', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersection', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric_difference', 'symmetric_difference_update', 'union', 'update']
```
User-Defined Data Types

/\list
\\Color
\Color(r, g, b)
\constructs a new color \c with red, green, and blue components \r, \g, and \b, all integers between 0 and 255.
\c.getRed()
\returns the red component of \c.
\c.getGreen()
\returns the green component of \c.
\c.getBlue()
\returns the blue component of \c.
\str(c)
\returns a string representation of \c.

To create an object of a user-defined data type, we call its constructor, using the name of the data type, followed by the constructor's arguments.

We use a variable name to identify the object to be associated with the method we intend to call.

In any data-type implementation, it is worthwhile to include an operation that converts an object's value to a string.

We use the following form of the \import statement to import a data type:

\from xyz import XYZ
**User-Defined Data Types**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color(r, g, b)</td>
<td>constructs a new color $c$ with red, green, and blue components $r$, $g$, and $b$, all integers between 0 and 255</td>
</tr>
<tr>
<td>c.getRed()</td>
<td>returns the red component of $c$</td>
</tr>
<tr>
<td>c.getGreen()</td>
<td>returns the green component of $c$</td>
</tr>
<tr>
<td>c.getBlue()</td>
<td>returns the blue component of $c$</td>
</tr>
<tr>
<td>str(c)</td>
<td>returns a string representation of $c$</td>
</tr>
</tbody>
</table>
User-Defined Data Types

<table>
<thead>
<tr>
<th>Color</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color(r, g, b)</td>
<td>constructs a new color c with red, green, and blue components r, g, and b, all integers between 0 and 255</td>
</tr>
<tr>
<td>c.getRed()</td>
<td>returns the red component of c</td>
</tr>
<tr>
<td>c.getGreen()</td>
<td>returns the green component of c</td>
</tr>
<tr>
<td>c.getBlue()</td>
<td>returns the blue component of c</td>
</tr>
<tr>
<td>str(c)</td>
<td>returns a string representation of c</td>
</tr>
</tbody>
</table>

To create an object of a user-defined data type, we call its constructor, using the name of the data type, followed by the constructor’s arguments.
**User-Defined Data Types**

<table>
<thead>
<tr>
<th>Color</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Color(r, g, b)</td>
<td>constructs a new color (c) with red, green, and blue components (r), (g), and (b), all integers between 0 and 255</td>
</tr>
<tr>
<td>c.getRed()</td>
<td>returns the red component of (c)</td>
</tr>
<tr>
<td>c.getGreen()</td>
<td>returns the green component of (c)</td>
</tr>
<tr>
<td>c.getBlue()</td>
<td>returns the blue component of (c)</td>
</tr>
<tr>
<td>str(c)</td>
<td>returns a string representation of (c)</td>
</tr>
</tbody>
</table>

To create an object of a user-defined data type, we call its constructor, using the name of the data type, followed by the constructor’s arguments.

We use a variable name to identify the object to be associated with the method we intend to call.
User-Defined Data Types

<table>
<thead>
<tr>
<th>Color</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Color(r, g, b)</td>
<td>constructs a new color <code>c</code> with red, green, and blue components <code>r</code>, <code>g</code>, and <code>b</code>, all integers between 0 and 255</td>
</tr>
<tr>
<td>c.getRed()</td>
<td>returns the red component of <code>c</code></td>
</tr>
<tr>
<td>c.getGreen()</td>
<td>returns the green component of <code>c</code></td>
</tr>
<tr>
<td>c.getBlue()</td>
<td>returns the blue component of <code>c</code></td>
</tr>
<tr>
<td>str(c)</td>
<td>returns a string representation of <code>c</code></td>
</tr>
</tbody>
</table>

To create an object of a user-defined data type, we call its constructor, using the name of the data type, followed by the constructor’s arguments.

We use a variable name to identify the object to be associated with the method we intend to call.

In any data-type implementation, it is worthwhile to include an operation that converts an object’s value to a string.
User-Defined Data Types

<table>
<thead>
<tr>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color(r, g, b)</td>
</tr>
<tr>
<td>$c$.getRed()</td>
</tr>
<tr>
<td>$c$.getGreen()</td>
</tr>
<tr>
<td>$c$.getBlue()</td>
</tr>
<tr>
<td>str($c$)</td>
</tr>
</tbody>
</table>

To create an object of a user-defined data type, we call its constructor, using the name of the data type, followed by the constructor’s arguments.

We use a variable name to identify the object to be associated with the method we intend to call.

In any data-type implementation, it is worthwhile to include an operation that converts an object’s value to a string.

We use the following form of the \texttt{import} statement to import a data type $\texttt{XYZ}$ defined in a file $\texttt{xyz.py}$:

\begin{verbatim}
from xyz import XYZ
\end{verbatim}
User-Defined Data Types

Program: alberssquares.py

- Command-line input: \texttt{r1} (int), \texttt{g1} (int), \texttt{b1} (int), \texttt{r2} (int), \texttt{g2} (int), and \texttt{b2} (int)

- Standard draw output: Albers' squares using colors (\texttt{r1}, \texttt{g1}, \texttt{b1}) and (\texttt{r2}, \texttt{g2}, \texttt{b2})

\begin{verbatim}
/terminal
~/workspace/ipp/programs
$ python3 alberssquares.py 9 90 166 100 100 100
/terminal
~/workspace/ipp/programs
$ python3 alberssquares.py 0 174 239 147 149 252
/terminal
~/workspace/ipp/programs
$ python3 alberssquares.py 110 110 110 145 160 156
\end{verbatim}
Program: alberssquares.py
User-Defined Data Types

Program: alberssquares.py

- Command-line input: \( r_1 \) (int), \( g_1 \) (int), \( b_1 \) (int), \( r_2 \) (int), \( g_2 \) (int), and \( b_2 \) (int)
User-Defined Data Types

Program: `alberssquares.py`

- Command-line input: $r1$ (int), $g1$ (int), $b1$ (int), $r2$ (int), $g2$ (int), and $b2$ (int)
- Standard draw output: Albers’ squares using colors $(r1, g1, b1)$ and $(r2, g2, b2)$
User-Defined Data Types

Program: alberssquares.py

- Command-line input: \( r_1 \) (int), \( g_1 \) (int), \( b_1 \) (int), \( r_2 \) (int), \( g_2 \) (int), and \( b_2 \) (int)
- Standard draw output: Albers’ squares using colors \((r_1, g_1, b_1)\) and \((r_2, g_2, b_2)\)

```
$ python3 alberssquares.py 9 90 166 100 100 100
```

```
> ~/workspace/ipp/programs
```

```
$ python3 alberssquares.py 0 174 239 147 149 252
```

```
$ python3 alberssquares.py 110 110 110 145 160 156
```
Program: alberssquares.py

- Command-line input: \( r1 \) (int), \( g1 \) (int), \( b1 \) (int), \( r2 \) (int), \( g2 \) (int), and \( b2 \) (int)
- Standard draw output: Albers’ squares using colors \( (r1, g1, b1) \) and \( (r2, g2, b2) \)

```
> ~/workspace/ipp/programs
$ python3 alberssquares.py 9 90 166 100 100 100
```

```
> ~/workspace/ipp/programs
$ python3 alberssquares.py 0 174 239 147 149 252
```
User-Defined Data Types

Program: alberssquares.py

- Command-line input: \(r_1\) (int), \(g_1\) (int), \(b_1\) (int), \(r_2\) (int), \(g_2\) (int), and \(b_2\) (int)
- Standard draw output: Albers’ squares using colors \((r_1, g_1, b_1)\) and \((r_2, g_2, b_2)\)

```
$ python3 alberssquares.py 9 90 166 100 100 100
```

```
$ python3 alberssquares.py 0 174 239 147 149 252
```

```
$ python3 alberssquares.py 110 110 110 145 160 156
```
User-Defined Data Types

```python
from color import Color
import stddraw
import sys

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

if __name__ == '__main__':
    main()
```
User-Defined Data Types

```python
from color import Color
import stddraw
import sys

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

if __name__ == '__main__':
    main()
```
User-Defined Data Types

Program: luminance.py

• Command-line input: \(r_1\) (int), \(g_1\) (int), \(b_1\) (int), \(r_2\) (int), \(g_2\) (int), and \(b_2\) (int)

• Standard output: whether the two colors are compatible

```
~/workspace/ipp/programs
$ python3 luminance.py 0 0 0 0 0 255
(0, 0, 0) compatible with (0, 0, 255)? False

$ python3 luminance.py 0 0 0 255 255 255
(0, 0, 0) compatible with (255, 255, 255)? True
```
User-Defined Data Types

Program: luminance.py

---

**Program:** luminance.py

---
Program: luminance.py

- Command-line input: $r_1$ (int), $g_1$ (int), $b_1$ (int), $r_2$ (int), $g_2$ (int), and $b_2$ (int)
Program: luminance.py

• Command-line input: $r_1$ (int), $g_1$ (int), $b_1$ (int), $r_2$ (int), $g_2$ (int), and $b_2$ (int)

• Standard output: whether the two colors are compatible
**User-Defined Data Types**

**Program:** luminance.py

- Command-line input: \( r_1 \) (int), \( g_1 \) (int), \( b_1 \) (int), \( r_2 \) (int), \( g_2 \) (int), and \( b_2 \) (int)
- Standard output: whether the two colors are compatible

```bash
$ python3 luminance.py 0 0 0 0 0 255
(0, 0, 0) compatible with (0, 0, 255)? False
$ python3 luminance.py 0 0 255 255 255 255
(0, 0, 0) compatible with (255, 255, 255)? True
$ ```
User-Defined Data Types

```python
from color import Color
import stdio
import sys

def luminance(c):
    r = c.getRed()
    g = c.getGreen()
    b = c.getBlue()
    if r == g and r == b:
        return r
    return 0.299 * r + 0.587 * g + 0.114 * b

def toGray(c):
    y = int(round(luminance(c)))
    gray = Color(y, y, y)
    return gray

def areCompatible(c1, c2):
    return abs(luminance(c1) - luminance(c2)) >= 128.0

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

if __name__ == '__main__':
    _main()
```
from color import Color
import stdio
import sys

def luminance(c):
    r = c.getRed()
    g = c.getGreen()
    b = c.getBlue()
    if r == g and r == b:
        return r
    return 0.299 * r + 0.587 * g + 0.114 * b

def toGray(c):
    y = int(round(luminance(c)))
    gray = Color(y, y, y)
    return gray

def areCompatible(c1, c2):
    return abs(luminance(c1) - luminance(c2)) >= 128.0

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

if __name__ == '__main__':
    _main()
User-Defined Data Types

- `/list` (a new `w`-by-`h` picture `pic`)
- `pic(filename)` (a new picture `pic` initialized from `filename`)
- `pic.save(filename)` (save `pic` to `filename`)
- `pic.width()` (the width of `pic`)
- `pic.height()` (the height of `pic`)
- `pic.get(col, row)` (the color of pixel (col, row) in `pic`)
- `pic.set(col, row, c)` (set the color of pixel (col, row) in `pic` to `c`
### User-Defined Data Types

<table>
<thead>
<tr>
<th>Picture Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Picture(w, h)</td>
<td>a new (w)-by-(h) picture (pic)</td>
</tr>
<tr>
<td>Picture(filename)</td>
<td>a new picture (pic) initialized from (filename)</td>
</tr>
<tr>
<td>pic.save(filename)</td>
<td>save (pic) to (filename)</td>
</tr>
<tr>
<td>pic.width()</td>
<td>the width of (pic)</td>
</tr>
<tr>
<td>pic.height()</td>
<td>the height of (pic)</td>
</tr>
<tr>
<td>pic.get(col, row)</td>
<td>the color of pixel ((col, row)) in (pic)</td>
</tr>
<tr>
<td>pic.set(col, row, c)</td>
<td>set the color of pixel ((col, row)) in (pic) to (c)</td>
</tr>
</tbody>
</table>
User-Defined Data Types

Program: grayscale.py

- Command-line input: filename (str)
- Standard draw output: a grayscale version of the image with the given filename

```
~/workspace/ipp/programs
$ python3 grayscale.py mandril.jpg
```
User-Defined Data Types

Program: grayscale.py

- Command-line input: `filename` (str)
- Standard draw output: a grayscale version of the image with the given filename
User-Defined Data Types

Program: grayscale.py

- Command-line input: filename (str)
Program: grayscale.py

- Command-line input: filename (str)
- Standard draw output: a gray scale version of the image with the given filename
Program: grayscale.py

- Command-line input: filename (str)
- Standard draw output: a gray scale version of the image with the given filename

```bash
~$ /workspace/ipp/programs
$ python3 grayscale.py mandril.jpg
```
User-Defined Data Types

```python
from picture import Picture
import luminance
import stddraw
import sys

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

if __name__ == '__main__':
    main()
```
User-Defined Data Types

from picture import Picture
import luminance
import stddraw
import sys

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

if __name__ == '__main__':
    main()
User-Defined Data Types
User-Defined Data Types

Program: fade.py
Program: fade.py

- Command-line input: sourceFile (str), targetFile (str), and n (int)
User-Defined Data Types

Program: fade.py

- Command-line input: sourceFile (str), targetFile (str), and n (int)
- Standard draw output: over the course of n frames, gradually replaces the image from sourceFile with the image from targetFile

```bash
$ python3 fade .py mandril .jpg darwin .jpg 5
```
Program: fade.py

- Command-line input: `sourceFile` (str), `targetFile` (str), and `n` (int)
- Standard draw output: over the course of `n` frames, gradually replaces the image from `sourceFile` with the image from `targetFile`

```bash
$ python3 fade.py mandril.jpg darwin.jpg 5
```
User-Defined Data Types

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

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

def _blend (c1 , c2 , alpha ):
    r = (1 - alpha ) * c1. getRed () + alpha * c2. getRed ()
    g = (1 - alpha ) * c1. getGreen () + alpha * c2. getGreen ()
    b = (1 - alpha ) * c1. getBlue () + alpha * c2. getBlue ()
    return Color ( int (r), int (g), int (b))

if __name__ == ' __main__ ':
    main ()
```
from color import Color
from picture import Picture
import stddraw
import sys

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

def _blend(c1, c2, alpha):
    r = (1 - alpha) * c1.getRed() + alpha * c2.getRed()
    g = (1 - alpha) * c1.getGreen() + alpha * c2.getGreen()
    b = (1 - alpha) * c1.getBlue() + alpha * c2.getBlue()
    return Color(int(r), int(g), int(b))

if __name__ == '__main__':
    main()
User-Defined Data Types

InStream

InStream(filename)  a new input stream in, initialized from filename (defaults to standard input)

in.isEmpty()  is in empty?

in.readInt()  read a token from in, and return it as an integer

in.readString()  read a token from in, and return it as a string

OutStream

OutStream(filename)  a new output stream out that will write to filename (defaults to standard output)

out.write(x)  write x to out

out.writeln(x)  write x to out, followed by a newline

out.writef(fmt, arg1, ...)  write the arguments arg1, ..., to out as specified by the format string fmt
User-Defined Data Types

---

### InStream

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InStream(filename)</td>
<td>a new input stream \textit{in}, initialized from \textit{filename} (defaults to standard input)</td>
</tr>
<tr>
<td>in.isEmpty()</td>
<td>is \textit{in} empty?</td>
</tr>
<tr>
<td>in.readInt()</td>
<td>read a token from \textit{in}, and return it as an integer</td>
</tr>
<tr>
<td>in.readString()</td>
<td>read a token from \textit{in}, and return it as a string</td>
</tr>
</tbody>
</table>
## User-Defined Data Types

### InStream

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InStream(filename)</td>
<td>a new input stream in, initialized from filename (defaults to standard input)</td>
</tr>
<tr>
<td><code>in.isEmpty()</code></td>
<td>is in empty?</td>
</tr>
<tr>
<td><code>in.readInt()</code></td>
<td>read a token from in, and return it as an integer</td>
</tr>
<tr>
<td><code>in.readString()</code></td>
<td>read a token from in, and return it as a string</td>
</tr>
</tbody>
</table>

### OutStream

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutStream(filename)</td>
<td>a new output stream out that will write to filename (defaults to standard output)</td>
</tr>
<tr>
<td><code>out.write(x)</code></td>
<td>write x to out</td>
</tr>
<tr>
<td><code>out.writeln(x)</code></td>
<td>write x to out, followed by a newline</td>
</tr>
<tr>
<td><code>out.writeln(fmt, arg1, ...)</code></td>
<td>write the arguments arg1, ... to out as specified by the format string fmt</td>
</tr>
</tbody>
</table>
Program: cat.py

- Command-line input: `sys.argv[1 : n - 2]` files or web pages
- File output: copies them to the file whose name is accepted is `sys.argv[n - 1]`

```
$ cat ../data/in1.txt
This is a tiny test.
```

```
$ python3 cat.py ../data/in1.txt ../data/in2.txt out.txt
cat out.txt
This is a tiny test.
```
User-Defined Data Types

Program: `cat.py`

```
# Program: cat.py

# Command-line input:
# sys.argv[1] : n - 2] files or web pages

# File output: copies them to the file whose name is accepted
# sys.argv[n - 1]

# terminal

~/workspace/ipp/programs

cat ../data/in1.txt

This is

```

cat ../data/in2.txt

a tiny
test.

```

```

cat ../data/in1.txt ../data/in2.txt out.txt

cat out.txt

This is

```

a tiny
test.
```

Program: cat.py

- Command-line input: `sys.argv[1 : n - 2]` files or web pages
Program: `cat.py`

- Command-line input: `sys.argv[1 : n - 2]` files or web pages
- File output: copies them to the file whose name is accepted is `sys.argv[n - 1]`
Program: cat.py

- Command-line input: `sys.argv[1 : n - 2]` files or web pages
- File output: copies them to the file whose name is accepted is `sys.argv[n - 1]`

```bash
> ~/workspace/ipp/programs
$ cat ../data/in1.txt
This is
$ cat ../data/in2.txt
a tiny
test.
$ python3 cat.py ../data/in1.txt ../data/in2.txt out.txt
$ cat out.txt
This is
a tiny
test.
$ 
```
User-Defined Data Types

```python
from instream import InStream
from outstream import OutStream
import sys

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

if __name__ == '__main__':
    main()
```
User-Defined Data Types

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

if __name__ == '__main__':
    main()
```
User-Defined Data Types
Program: split.py
User-Defined Data Types

Program: split.py

- Command-line input: filename (str) and n (int)
User-Defined Data Types

**Program:** split.py

- Command-line input: `filename` (str) and `n` (int)
- File output: splits the file whose name is `filename.csv`, by field, into `n` files named `filename1.txt`, `filename2.txt`, etc.
User-Defined Data Types

Program: split.py

- Command-line input: `filename` (str) and `n` (int)
- File output: splits the file whose name is `filename.csv`, by field, into `n` files named `filename1.txt`, `filename2.txt`, etc.
from instream import InStream
from outstream import OutStream
import stdarray
import sys

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

if __name__ == '__main__':
    main()
from instream import InStream
from outstream import OutStream
import stdarray
import sys

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

if __name__ == '__main__':
    main()