Introduction to Programming in Python

Object-oriented Programming: Using Data Types

Outline

1. Methods
2. Basic and Collection Data Types
3. User-defined Data Types

Methods

Methods

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

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

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 as `<object>.<name>(<argument1>, <argument2>, ...)`

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 as `<object>.<name>(<argument1>, <argument2>, ...)`

Example

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

```
>>> x = "ALICE"
>>> y = "bob"
>>> x.isupper()
True
>>> y.isupper()
False
>>> x.islower()
False
>>> y.islower()
True
```

Basic and Collection Data Types

Basic and Collection Data Types

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

```
>>> 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 and Collection Data Types

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

```
>>> 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']
```

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

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

Basic and Collection Data Types

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

>>> 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']
```

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

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

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

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

Basic and Collection Data Types

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

>>> 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']
```

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

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

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

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

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

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

Basic and Collection Data Types

Basic and Collection Data Types

potentialgene.py

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

Basic and Collection Data Types

potentialgene.py

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

>_ ~/workspace/ipp/programs

```
$ _
```

Basic and Collection Data Types

potentialgene.py

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

~/workspace/ipp/programs

```
$ python3 potentialgene.py ATGCGCCTGCGTCTGTACTAG
```

Basic and Collection Data Types

potentialgene.py

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

~/workspace/ipp/programs

```
$ python3 potentialgene.py ATGCGCCTGCGTCTGTACTAG
True
$ _
```

Basic and Collection Data Types

potentialgene.py

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

~/workspace/ipp/programs

```
$ python3 potentialgene.py ATGCGCCTGCGTCTGTACTAG
True
$ python3 potentialgene.py ATGCGCTGCGTCTGTACTAG
```

Basic and Collection Data Types

potentialgene.py

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

~/workspace/ipp/programs

```
$ python3 potentialgene.py ATGCGCCTGCGTCTGTACTAG
True
$ python3 potentialgene.py ATGCGCTGCGTCTGTACTAG
False
$ _
```

Basic and Collection Data Types

Basic and Collection Data Types

</> potentialgene.py

```
import stdio
import sys

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

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

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

User-defined Data Types

User-defined Data Types

color.Color	
`Color(r = 0, g = 0, b = 0)`	constructs a color `c` given its red, green, and blue components
`c.getRed()`	returns the red component of `c`
`c.getGreen()`	returns the green component of `c`
`c.getBlue()`	returns the blue component of `c`
`c.luminance()`	returns the luminance of `c`
`c.toGray()`	returns the grayscale equivalent of `c`
`c.isCompatible(d)`	returns `True` if `c` is compatible with `d`, and `False` otherwise
`str(c)`	returns a string representation of `c`

User-defined Data Types

User-defined Data Types

To create an object of a data type, we call its constructor (a method with the same name as the data type)

```
<name> = <type>(<argument1>, <argument2>, ...)
```

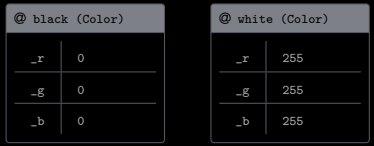
User-defined Data Types

To create an object of a data type, we call its constructor (a method with the same name as the data type)

```
<name> = <type>(<argument1>, <argument2>, ...)
```

Example

```
black = Color(0, 0, 0)
white = Color(255, 255, 255)
```



User-defined Data Types

User-defined Data Types

Printing an object `o` corresponds to printing the value returned by the function call `str(o)`

User-defined Data Types

- Printing an object `o` corresponds to printing the value returned by the function call `str(o)`
- The function call `str(o)` translates to the method call `o.__str__()`

User-defined Data Types

Printing an object `o` corresponds to printing the value returned by the function call `str(o)`

The function call `str(o)` translates to the method call `o.__str__()`

We use the following form of the `import` statement to import a data type `XYZ` defined in a file `xyz.py`

```
from xyz import XYZ
```

User-defined Data Types

User-defined Data Types

`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

albersssquares.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

```
$ _
```

User-defined Data Types

albersquares.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 albersquares.py 255 165 0 255 69 0
```

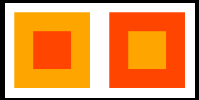
User-defined Data Types

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 255 165 0 255 69 0
```



User-defined Data Types

albersssquares.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 albersssquares.py 255 165 0 255 69 0
$ _
```

User-defined Data Types

User-defined Data Types

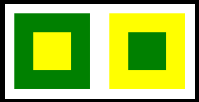
```
>_ ~/workspace/ipp/programs
$ _
```

User-defined Data Types

```
>_ ~/workspace/ipp/programs
$ python3 alberssquares.py 0 128 0 255 255 0
```

User-defined Data Types

```
>_ ~/workspace/ipp/programs
$ python3 alberssquares.py 0 128 0 255 255 0
```



User-defined Data Types

```
>_ ~/workspace/ipp/programs
$ python3 alberssquares.py 0 128 0 255 255 0
$ _
```

User-defined Data Types

User-defined Data Types

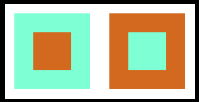
```
>_ ~/workspace/ipp/programs
$ _
```

User-defined Data Types

```
>_ ~/workspace/ipp/programs
$ python3 alberssquares.py 127 255 212 210 105 30
```

User-defined Data Types

```
>_ ~/workspace/ipp/programs
$ python3 alberssquares.py 127 255 212 210 105 30
```



User-defined Data Types

```
>_ ~/workspace/ipp/programs
$ python3 alberssquares.py 127 255 212 210 105 30
$ _
```

User-defined Data Types

User-defined Data Types

</> alberssquares.py

```
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

User-defined Data Types

picture.Picture	
`Picture(file)`	constructs a picture `p` from an image (`.jpg` or `.png`) file
`Picture(width = 512, height = 512)`	constructs a picture `p` given its dimensions in pixels
`p.save(file)`	saves `p` to the file with the given name
`p.width()`	returns the width of `p` in pixels
`p.height()`	returns the height of `p` in pixels
`p.get(x, y)`	returns the color of `p` at the location `(x, y)`
`p.set(x, y, c)`	sets the color of `p` at the location `(x, y)` to `c`

User-defined Data Types

User-defined Data Types

grayscale.py

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

User-defined Data Types

grayscale.py

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

>_ ~/workspace/ipp/programs

```
$ _
```

User-defined Data Types

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 ../data/mandrill.png
```

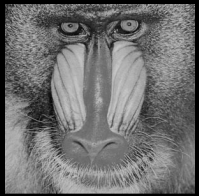
User-defined Data Types

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 ../data/mandrill.png
```



User-defined Data Types

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 ../data/mandrill.png
$ _
```

User-defined Data Types

User-defined Data Types

</> grayscale.py

```
from picture import Picture
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 = pixel.toGray()
            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

fade.py

Command-line input	*sourceFile* (str), *targetFile* (str), and *n* (int)
Standard draw output	replaces the image from *sourceFile* with the image from *targetFile* in *n* frames

User-defined Data Types

fade.py

Command-line input	*sourceFile* (str), *targetFile* (str), and *n* (int)
Standard draw output	replaces the image from *sourceFile* with the image from *targetFile* in *n* frames

```
>_ ~/workspace/ipp/programs
$ _
```

User-defined Data Types

fade.py

Command-line input	*sourceFile* (str), *targetFile* (str), and *n* (int)
Standard draw output	replaces the image from *sourceFile* with the image from *targetFile* in *n* frames

>_ ~/workspace/ipp/programs

```
$ python3 fade.py ../data/mandrill.png ../data/darwin.png 5
```

User-defined Data Types

fade.py

Command-line input	*sourceFile* (str), *targetFile* (str), and *n* (int)
Standard draw output	replaces the image from *sourceFile* with the image from *targetFile* in *n* frames

~/workspace/ipp/programs

```
$ python3 fade.py ../data/mandrill.png ../data/darwin.png 5
```






User-defined Data Types

fade.py

Command-line input	*sourceFile* (str), *targetFile* (str), and *n* (int)
Standard draw output	replaces the image from *sourceFile* with the image from *targetFile* in *n* frames

>_ ~/workspace/ipp/programs

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

User-defined Data Types

User-defined Data Types

</> fade.py

```
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(1000)
    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

User-defined Data Types

instream.InStream

`InStream(fileOrUrl = None)`	constructs an input stream `i` from a file/URL or standard input if the argument is empty
`i.isEmpty()`	returns `True` if `i` is empty, and `False` otherwise
`i.readInt()`	returns a token from `i` as an integer
`i.readAllInts()`	returns the remaining tokens from `i` as a list of integers
`i.readFloat()`	returns a token from `i` as a float
`i.readAllFloats()`	returns the remaining tokens from `i` as a list of floats
`i.readBool()`	returns a token from `i` as a boolean
`i.readAllBools()`	returns the remaining tokens from `i` as a list of booleans
`i.readString()`	returns a token from `i` as a string
`i.readAllStrings()`	returns the remaining tokens from `i` as a list of strings
`i.hasNextLine()`	returns `True` if `i` has a next line, and `False` otherwise
`i.readLine()`	returns a line of tokens from `i` as a string
`i.readAllLines()`	returns the remaining lines of tokens from `i` as a list of strings
`i.readAll()`	returns the remaining tokens from `i` as a string

User-defined Data Types

User-defined Data Types

outstream.OutStream	
`OutStream(file = None)`	constructs an output stream `o` from a file or standard output if the argument is empty
`o.writeln(x = "")`	writes `x` followed by newline to `o`
`o.write(x = "")`	writes `x` to `o`
`o.writef(fmt, *args)`	writes each element of `args` to `o` according to the format specified by the string `fmt`

User-defined Data Types

User-defined Data Types

cat.py

Command-line input	input file or web page names and output file name
File output	copies the contents of the input files or web pages to the output file

User-defined Data Types

cat.py

Command-line input	input file or web page names and output file name
File output	copies the contents of the input files or web pages to the output file

>_ ~/workspace/ipp/programs

```
$ _
```

User-defined Data Types

cat.py

Command-line input	input file or web page names and output file name
File output	copies the contents of the input files or web pages to the output file

~/workspace/ipp/programs

```
$ cat ../data/in1.txt
```

User-defined Data Types

cat.py

Command-line input	input file or web page names and output file name
File output	copies the contents of the input files or web pages to the output file

>_ ~/workspace/ipp/programs

```
$ cat ../data/in1.txt
This is
$ _
```

User-defined Data Types

cat.py

Command-line input	input file or web page names and output file name
File output	copies the contents of the input files or web pages to the output file

>_ ~/workspace/ipp/programs

```
$ cat ../data/in1.txt
This is
$ cat ../data/in2.txt
```

User-defined Data Types

cat.py

Command-line input	input file or web page names and output file name
File output	copies the contents of the input files or web pages to the output file

~/workspace/ipp/programs

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

User-defined Data Types

cat.py

Command-line input	input file or web page names and output file name
File output	copies the contents of the input files or web pages to the output file

~/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
```

User-defined Data Types

cat.py

Command-line input	input file or web page names and output file name
File output	copies the contents of the input files or web pages to the output file

~/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
$ _
```

User-defined Data Types

cat.py

Command-line input	input file or web page names and output file name
File output	copies the contents of the input files or web pages to the output file

~/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
```

User-defined Data Types

cat.py

Command-line input	input file or web page names and output file name
File output	copies the contents of the input files or web pages to the output file

~/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

User-defined Data Types

</> cat.py

```
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

User-defined Data Types

split.py

Command-line input	*filename* (str) and *n* (int)
File output	splits *filename*.*csv* by field into *n* files named *filename*1.*txt*, *filename*2.*txt*, etc

User-defined Data Types

split.py

Command-line input	*filename* (str) and *n* (int)
File output	splits *filename*.*csv* by field into *n* files named *filename*1.*txt*, *filename*2.*txt*, etc

>_ ~/workspace/ipp/programs

```
$ _
```

User-defined Data Types

split.py

Command-line input	*filename* (str) and *n* (int)
File output	splits *filename*.*csv* by field into *n* files named *filename*1.*txt*, *filename*2.*txt*, etc

~/workspace/ipp/programs

```
$ head -5 ../data/ip.csv
```

User-defined Data Types

split.py

Command-line input	*filename* (str) and *n* (int)
File output	splits *filename*.*csv* by field into *n* files named *filename*1.*txt*, *filename*2.*txt*, etc

~/workspace/ipp/programs

```
$ head -5 ../data/ip.csv
www.princeton.edu,128.112.128.15
www.cs.princeton.edu,128.112.136.35
www.math.princeton.edu,128.112.18.11
www.cs.harvard.edu,140.247.50.127
www.harvard.edu,128.103.60.24
$ _
```

User-defined Data Types

split.py

Command-line input	*filename* (str) and *n* (int)
File output	splits *filename*.*csv* by field into *n* files named *filename*1.*txt*, *filename*2.*txt*, etc

~/workspace/ipp/programs

```
$ head -5 ../data/ip.csv
www.princeton.edu,128.112.128.15
www.cs.princeton.edu,128.112.136.35
www.math.princeton.edu,128.112.18.11
www.cs.harvard.edu,140.247.50.127
www.harvard.edu,128.103.60.24
$ python3 split.py ../data/ip 2
```

User-defined Data Types

split.py

Command-line input	*filename* (str) and *n* (int)
File output	splits *filename*.*csv* by field into *n* files named *filename*1.*txt*, *filename*2.*txt*, etc

~/workspace/ipp/programs

```
$ head -5 ../data/ip.csv
www.princeton.edu,128.112.128.15
www.cs.princeton.edu,128.112.136.35
www.math.princeton.edu,128.112.18.11
www.cs.harvard.edu,140.247.50.127
www.harvard.edu,128.103.60.24
$ python3 split.py ../data/ip 2
$ _
```

User-defined Data Types

split.py

Command-line input	*filename* (str) and *n* (int)
File output	splits *filename*.*csv* by field into *n* files named *filename*1.*txt*, *filename*2.*txt*, etc

~/workspace/ipp/programs

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

User-defined Data Types

split.py

Command-line input	*filename* (str) and *n* (int)
File output	splits *filename*.*csv* by field into *n* files named *filename*1.*txt*, *filename*2.*txt*, etc

~/workspace/ipp/programs

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

User-defined Data Types

split.py

Command-line input	*filename* (str) and *n* (int)
File output	splits *filename.csv* by field into *n* files named *filename*1.*txt*, *filename*2.*txt*, etc

~/workspace/ipp/programs

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

User-defined Data Types

split.py

Command-line input	*filename* (str) and *n* (int)
File output	splits *filename*.*csv* by field into *n* files named *filename*1.*txt*, *filename*2.*txt*, etc

~/workspace/ipp/programs

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

User-defined Data Types

User-defined Data Types

split.py

```
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()
```