Introduction to Programming in Python

Object-oriented Programming: Defining Data Types

Outline

Basic Elements of a Data Type

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

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

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

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

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

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`

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, ie, 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

Basic Elements of a Data Type

Instance variables implement the values of a data type

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

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

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

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

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`

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

Examples of Data Types

Examples of Data Types

Stopwatch	
`Stopwatch()`	Constructs a new stopwatch
`elapsedTime()`	Returns the elapsed time (in seconds) since creation

Examples of Data Types

Examples of Data Types

Program: `timeops.py`

Examples of Data Types

Program: `timeops.py`

- Command-line input: n (int)

Examples of Data Types

Program: `timeops.py`

- Command-line input: n (int)
- Standard output: computes the sum $1^{0.5} + 2^{0.5} + ... + n^{0.5}$ using `math.sqrt(x)` and `math.pow(x)` to calculate the $\sqrt{x}$, and writes a comparison of the performance characteristics of the two functions

Examples of Data Types

Program: `timeops.py`

- Command-line input: n (int)
- Standard output: computes the sum $1^{0.5} + 2^{0.5} + ... + n^{0.5}$ using `math.sqrt(x)` and `math.pow(x)` to calculate the $\sqrt{x}$, and writes a comparison of the performance characteristics of the two functions

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

$ python3 timeops.py 10000000
math.sqrt() is 2.05 times faster than math.pow()
$
```

Examples of Data Types

Examples of Data Types

</> timeops.py

```
from stopwatch import Stopwatch
import math
import stdio
import sys

def main():
    n = int(sys.argv[1])
    watch1 = Stopwatch()
    total = 0.0
    for i in range(1, n + 1):
        total += math.sqrt(i)
    time1 = watch1.elapsedTime()
    watch2 = Stopwatch()
    total = 0.0
    for i in range(1, n + 1):
        total += math.pow(i, 0.5)
    time2 = watch2.elapsedTime()
    stdio.writef('math.sqrt() is %.2f times faster than math.pow()\n', time2 / time1)

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

Examples of Data Types

Examples of Data Types

</> stopwatch.py

```
import stdio
import sys
import time

class Stopwatch:
    def __init__(self):
        self.creationTime = time.time()

    def elapsedTime(self):
        return time.time() - self.creationTime

def _main():
    n = int(sys.argv[1])
    watch = Stopwatch()
    primes = 0
    for i in range(2, n + 1):
        j = 2
        while j <= i / j:
            if i % j == 0:
                break
            j += 1
        if j > i / j:
            primes += 1
    time = watch.elapsedTime()
    stdio.writef('pi(%d) = %d computed in %.5f seconds\n', n, primes, time)

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

Examples of Data Types

Examples of Data Types

Histogram	
`Histogram(n)`	constructs a new histogram from the integer values in $0, 1, \ldots, n-1$
`addDataPoint(i)`	adds an occurrence of integer i to the histogram
`draw()`	draw the histogram to standard draw

Examples of Data Types

Examples of Data Types

Program: `bernoulli.py`

Examples of Data Types

Program: `bernoulli.py`

- Command-line input: *n* (int), *p* (float), and *trials* (int)

Examples of Data Types

Program: `bernoulli.py`

- Command-line input: *n* (int), *p* (float), and *trials* (int)
- Standard draw output: performs *trials* experiments, each of which counts the number of heads found when a coin with bias *p* is flipped *n* times, and draws the results

Examples of Data Types

Program: `bernoulli.py`

- Command-line input: *n* (int), *p* (float), and *trials* (int)
- Standard draw output: performs *trials* experiments, each of which counts the number of heads found when a coin with bias *p* is flipped *n* times, and draws the results

```
>_ ~/workspace/ipp/programs
$ python3 bernoulli.py 50 0.5 1000000
```

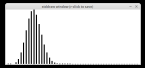
Examples of Data Types

Program: `bernoulli.py`

- Command-line input: *n* (int), *p* (float), and *trials* (int)
- Standard draw output: performs *trials* experiments, each of which counts the number of heads found when a coin with bias *p* is flipped *n* times, and draws the results

```
>_ ~/workspace/ipp/programs
$ python3 bernoulli.py 50 0.5 1000000
```

```
>_ ~/workspace/ipp/programs
$ python3 bernoulli.py 50 0.2 1000000
```



Examples of Data Types

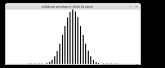
Program: `bernoulli.py`

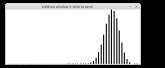
- Command-line input: *n* (int), *p* (float), and *trials* (int)
- Standard draw output: performs *trials* experiments, each of which counts the number of heads found when a coin with bias *p* is flipped *n* times, and draws the results

```
>_ ~/workspace/ipp/programs
$ python3 bernoulli.py 50 0.5 1000000
```

```
>_ ~/workspace/ipp/programs
$ python3 bernoulli.py 50 0.2 1000000
```

```
>_ ~/workspace/ipp/programs
$ python3 bernoulli.py 50 0.8 1000000
```





Examples of Data Types

Examples of Data Types

</> bernoulli.py

```
from histogram import Histogram
import stddraw
import stdrandom
import sys

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

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

Examples of Data Types

Examples of Data Types

</> histogram.py

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

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

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

    def draw(self):
        stddraw.setYscale(-1, max(self.freq) + 1)
        stdstats.plotBars(self.freq)

def _main():
    trials = int(sys.argv[1])
    histogram = Histogram(6)
    for t in range(trials):
        roll = stdrandom.uniformInt(0, 6)
        histogram.addDataPoint(roll)
    stddraw.setCanvasSize(500, 200)
    histogram.draw()
    stddraw.show()

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

Examples of Data Types

[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

Examples of Data Types

A data type `Turtle` for producing turtle graphics[1]

`Turtle`	
`Turtle(x0, y0, a0)`	constructs a new turtle at (x_0, y_0) facing a_0 degrees from the x-axis
`turnLeft(delta)`	instructs the turtle to turn left (conterclockwise) by *delta* degrees
`goForward(step)`	instructs the turtle to move forward distance *step*, drawing a line

[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

Examples of Data Types

Examples of Data Types

Program: `drunks.py`

Examples of Data Types

Program: `drunks.py`

- Command-line input: *n* (int), *steps* (int), and *stepSize* (float)

Examples of Data Types

Program: `drunks.py`

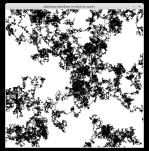
- Command-line input: *n* (int), *steps* (int), and *stepSize* (float)
- Standard draw output: creates *n* `Turtle` objects and has them take *steps* random steps, each of size *stepSize*

Examples of Data Types

Program: `drunks.py`

- Command-line input: *n* (int), *steps* (int), and *stepSize* (float)
- Standard draw output: creates *n* `Turtle` objects and has them take *steps* random steps, each of size *stepSize*

```
>_ ~/workspace/ipp/programs
$ python3 drunks.py 20 5000 .005
```



Examples of Data Types

Examples of Data Types

</> drunks.py

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

def main():
    n = int(sys.argv[1])
    steps = int(sys.argv[2])
    stepSize = float(sys.argv[3])
    turtles = stdarray.create1D(n, None)
    for i in range(n):
        x = stdrandom.uniformFloat(0.0, 1.0)
        y = stdrandom.uniformFloat(0.0, 1.0)
        theta = stdrandom.uniformFloat(0.0, 360.0)
        turtles[i] = Turtle(x, y, theta)
    stddraw.setPenRadius(0.0)
    for i in range(steps):
        for turtle in turtles:
            theta = stdrandom.uniformFloat(0.0, 360.0)
            turtle.turnLeft(theta)
            turtle.goForward(stepSize)
            stddraw.show(0.0)
    stddraw.show()

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

Examples of Data Types

Examples of Data Types

</> turtle.py

```
import math
import stddraw
import sys


class Turtle:
    def __init__(self, x, y, theta):
        self.x = x
        self.y = y
        self.theta = theta

    def turnLeft(self, theta):
        self.theta += theta

    def goForward(self, stepSize):
        xOld = self.x
        yOld = self.y
        self.x += stepSize * math.cos(math.radians(self.theta))
        self.y += stepSize * math.sin(math.radians(self.theta))
        stddraw.line(xOld, yOld, self.x, self.y)

def _main():
    n = int(sys.argv[1])
    turtle = Turtle(0.5, 0.0, 180.0 / n)
    stepSize = math.sin(math.radians(180.0 / n))
    stddraw.setPenRadius(0.0)
    for i in range(n):
        turtle.goForward(stepSize)
        turtle.turnLeft(360.0 / n)
    stddraw.show()

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