COMPSCI 105 S1 2017
Principles of Computer Science

Classes 2

Exercise

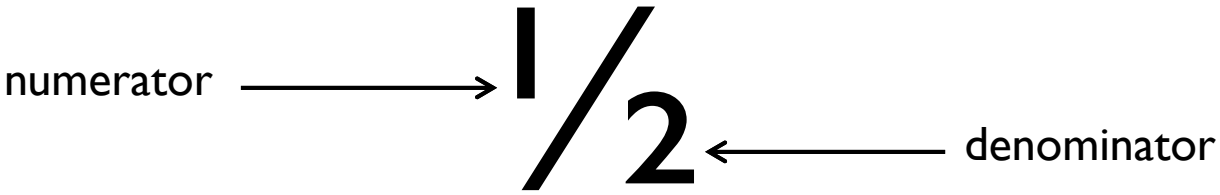
- Q1: Write a method named slope_from_origin() which returns the slope of the line joining the origin to the point. For example, Point(4,10).slop_from_origin() returns 2.5

```
p = Point(4, 10)
print(p.slop_from_origin()) #2.5
```

$$m = \frac{y_2 - (y_1)}{x_2 - x_1}$$

Example: Fractions

- Write a class to represent fractions in Python
  - create a fraction
  - add
  - subtract
  - multiply
  - divide
  - text representation

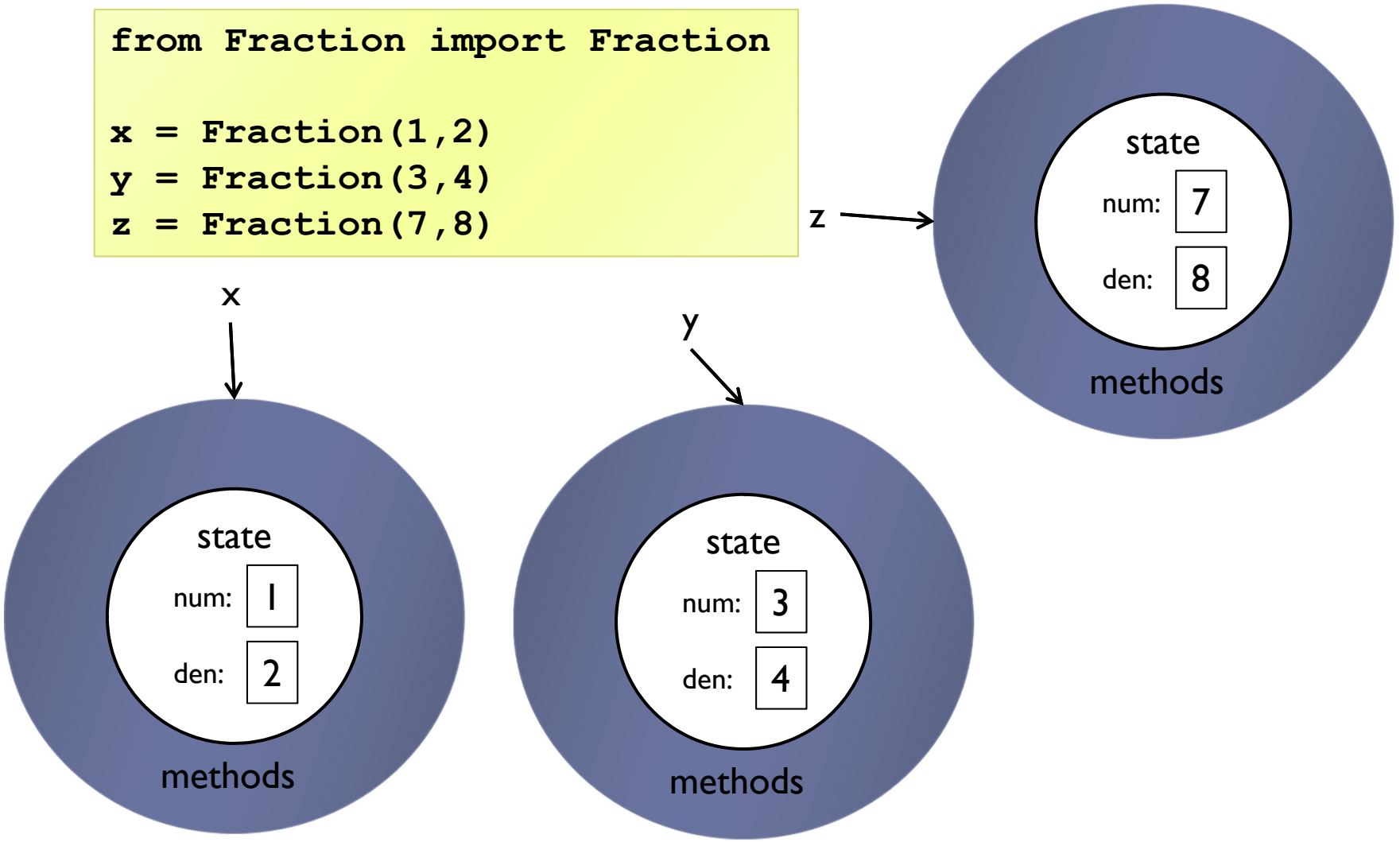


Example05.py

Model of objects in memory

```
from Fraction import Fraction

x = Fraction(1,2)
y = Fraction(3,4)
z = Fraction(7,8)
```



Constructor

- All classes must have a constructor
  - The constructor for a Fraction should store the numerator and the denominator

```
class Fraction:
    def __init__(self, top, bottom):
        self.num = top              #numerator
        self.den = bottom       #denominator
```

Using the Fraction class

- So far, we can create a Fraction

```
>>> x = Fraction(3, 4)
```

- We can access the state variables directly
  - Although not generally good practice to do so

```
>>> x.num
3
>>> x.den
4
```

- What else can we do with Fractions?
  - Nothing yet. We need to write the functions first!

Lecture 04

Overriding default behaviour

- All classes get a number of methods provided by default
  - Since default behaviour is not very useful, we should write our own versions of those methods
    - __repr__
    - __str__

Aside: Use of string formatting syntax

- Often we want to use a string that combines literal text and information from variables
  - Example:

```
name = 'Andrew'
greeting = 'Hello ' +  name + '. How are you?'
```

- We can use string formatting to perform this task
  - Use curly braces within the string to signify a variable to be replaced

```
my_name = 'Andrew'
greeting = 'Hello {name}.  How are you?'.format(name=my_name)
```

  - We can put the argument position in the curly braces

```
first = 'Andrew'
second = 'Luxton-Reilly'
greeting = 'Hello {0} {1}'.format(first, second)
```

__repr__

Fraction.py

- The __repr__ method produces a string that unambiguously describes the object
  - All classes should have a __repr__ function implemented
  - Ideally, the representation could be used to create the object
    - For example, a fraction created using Fraction(2, 3) should have a __repr__ method that returned 'Fraction(2, 3)'

```
class Fraction:
def __init__(self, top, bottom):
    self.num = top
     self.den = bottom

def __repr__(self):
    return 'Fraction({0}, {1})'.format(self.num, self.den)
```

  - Using the object

```
>>> x = Fraction(2, 3)
>>> x
```

Fraction(2, 3)

Without the __repr__ method

```
class Fraction:
    def __init__(self, top, bottom):
        self.num = top
        self.den = bottom
```

```
>>> x = Fraction(2, 3)
>>> x
```

```
<__main__.Fraction object
at 0x02762290>)
```

__str__

- The __str__ method returns a string representing the object
  - By default, it calls the __repr__ method
  - The __str__ method should focus on being human readable
  - We should implement a version with a natural representation:

```
def __str__(self):
        return str(self.num) + '/' + str(self.den)
```

- After we have implemented the method, we can use the print function to print the object

```
>>> x = Fraction(3, 4)
>>> print(x)
3/4
```

Without the __str__ method

```
class Fraction:
    def __init__(self, top, bottom):
        self.num = top
        self.den = bottom
```

```
>>> x = Fraction(2, 3)
>>> print(x)
```

```
<__main__.Fraction object
at 0x02714290>
```

Exercise 1

- Write the __repr__ method for the Square class created earlier.
- Would it be useful to implement a __str__ method?
- What would you choose to produce as output from a __str__ method?

str and repr

- What is the difference between the __str__ and __repr__ methods of a Python object?
  - In short __repr__ goal is to be unambigous and __str__ is to be readable.
  - The official Python documentation says:
    - __repr__ is used to compute the "official" string representation of an object and
    - __str__ is used to compute the "informal" string representation of an object.
  - The print statement and str() built-in function uses __str__
  - The repr() built-in function uses __repr__ to display the object.

```
>>> s1 = Square(10)

>>> str(s1)
'10 x 10 Square'

>>> repr(s1)
'Square(10)'
```

```
>>> print(s1)
10 x 10 Square

>>> s1
Square(10)
```

Exercise 2

- Consider the Circle class which we developed previously:
  - Modify the constructor with default values of 0 for the radius
  - Write the __repr__ method
  - Write the __str__ method

```
>>> c1 = Circle(10)

>>> str(s1)
'A circle with a radius of 10cm'

>>> repr(s1)
'Circle(10)'
```

  - Write a method named get_diameter() which returns the diameter of the circle.