Classes and Objects

Thomas Schwarz, SJ

Classes and Objects

- Allows programmer to create their own data structures
  - Data Layout
  - Methods (functions working on the data structure)

Classes and Objects

- Object Oriented programming is its own style:
  - Imperative programming manipulates the state of memory
    - Breaks up tasks through procedures and functions
  - Object Oriented Programming
    - Creates objects that interact with each other
    - Objects are defined with a user-defined data type

Classes and Objects

- Each object maintains its own state
    - Objects manipulate themselves and other objects through methods

Classes and Objects

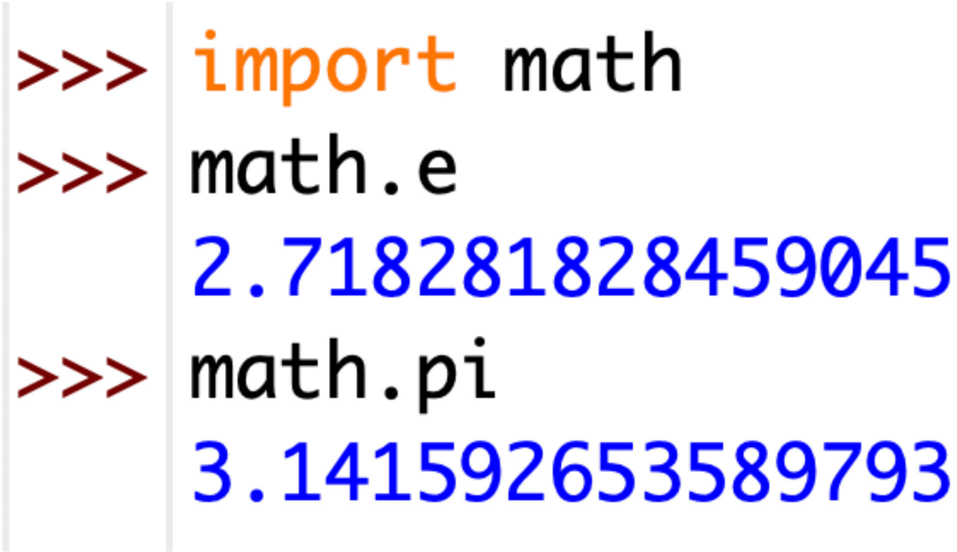
- Running Example:
  - Rational Numbers
  - Rational numbers have numerator and denominator
    - Divide by the largest common divisor
    - And if there is a negative sign, let it be at the denominator

Defining Objects

- Objects consists of fields and methods
    - Fields contain values
        - Two types:
            - Class variables aka Class fields
                - Belong to all objects of the class
            - Object variables aka Object fields

Defining Objects

- Example:
    - class math has a class field e
            - `math.e`
        - This is Euler's constant
            - math.pi
        - This is pi
- There is only one such thing
- Therefore it is a class variable



```
>>> import math
>>> math.e
2.718281828459045
>>> math.pi
3.141592653589793
```

Defining Objects

- Example:
    - `class Rational`
    - Each object has their own denominator and numerator
    - These are object variables, because they differ between objects
    -

Defining Objects

- Methods also belong either to a class or to an object
    - class methods vs. object methods
- An object method will also use the current object

Defining Objects

- To create an object of type class, **“instantiation”**: we define and use an initializer called `__init__()`
- The initializer can have arguments.
- If we create object variables and methods, we use the keyword `self` to refer to the object.

class Rational

- A rational number has an denominator and a numerator
- We give them name: den and num
- We use the key word class and the colon to define the class

```
class Rational:
```

class Rational

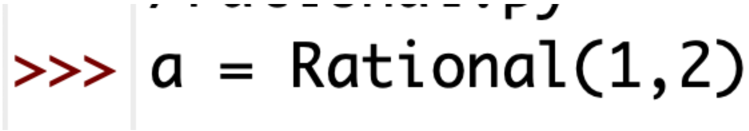
- We then add the “initializer”
  - You can think of this as a sort of constructor
  - This method will run whenever we create an object

```
class Rational:
    def __init__(self, a, b):
        self.num = a
        self.den = b
```

Defining Objects

- The init function is one of many special functions
  - Characterized by two underscores before and after the function name
  - These are known as ***dunder*** functions
- You create an object by assigning the class name

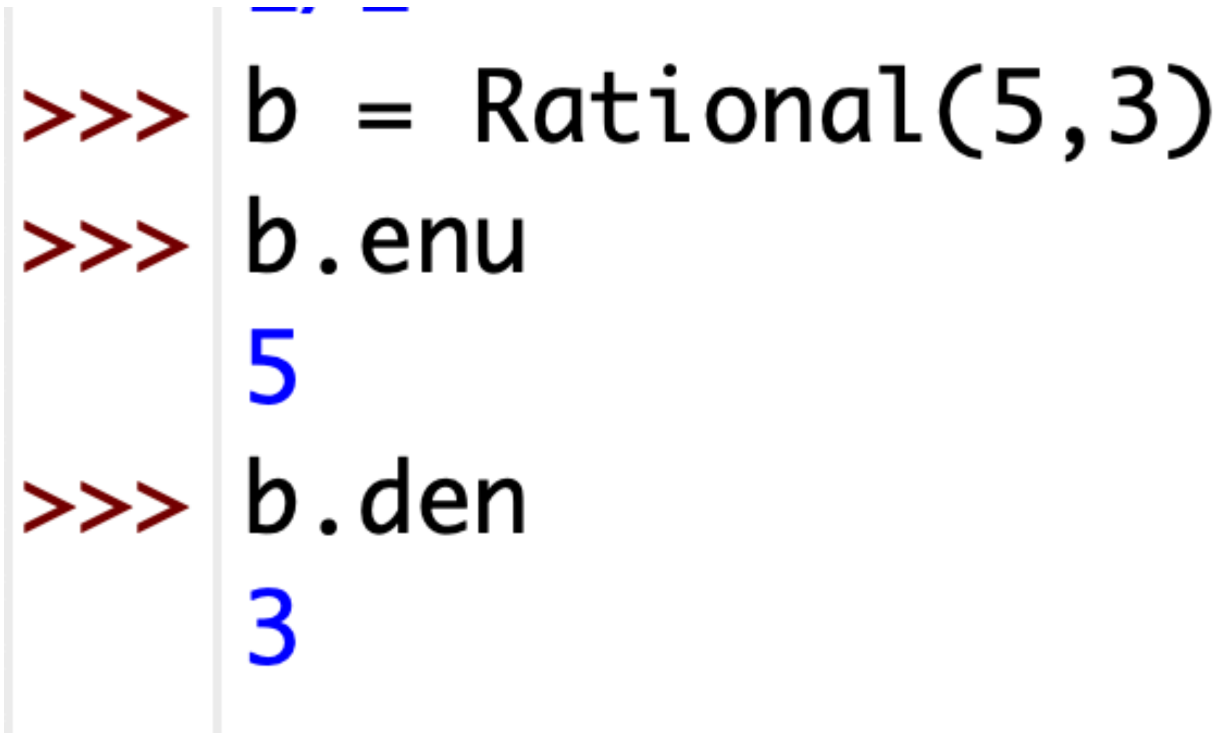
class Rational



```
>>> a = Rational(1,2)
```

Defining Objects

- You can access fields and methods using the dot notation



```
>>> b = Rational(5,3)
>>> b.enu
5
>>> b.den
3
```

class Rational

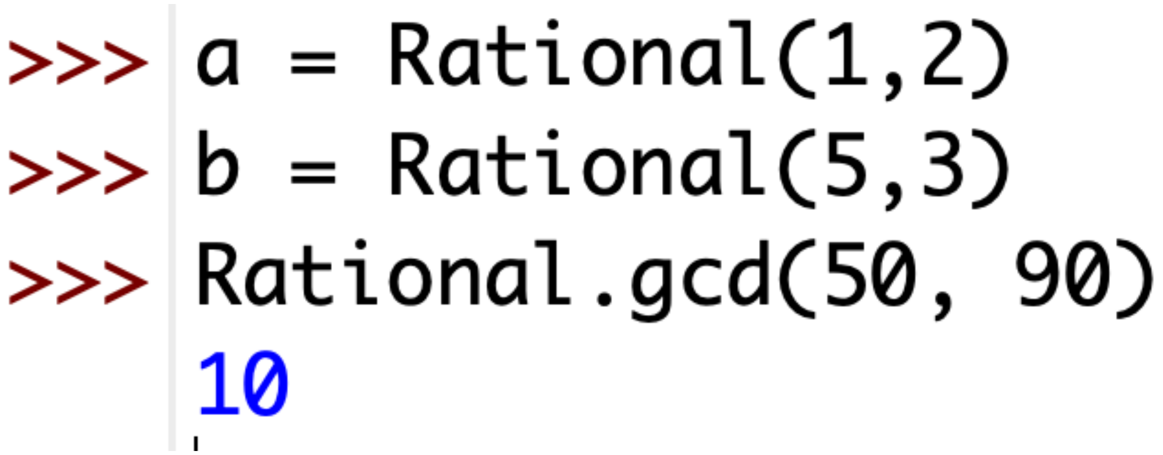
- We need to keep rational numbers neat
  - Factor out common factors of numerator and denominator
  - Use the gcd function with the Euclidean algorithms
  - This becomes a class method, because there is only one of them

class Rational

- We call a class function by using class name and the dot

```
class Rational:
    def __init__(self, a, b):
        self.num = a
        self.den = b

    def gcd(a, b):
        while(b):
            a, b = b, a % b
        return a
```



```
>>> a = Rational(1,2)
>>> b = Rational(5,3)
>>> Rational.gcd(50, 90)
10
```

class Rational

- We need to improve the initial definition
  - Both numerator and denominator are either positive or numerator is negative and denominator is positive
  - There is no common divisor between numerator and denominator

class Rational

- This leads to:
-

```
class Rational:
    def __init__(self, a, b):
        divisor = Rational.gcd(abs(a),abs(b))
        self.num = a//divisor
        self.den = b//divisor
        if self.den < 0:
            self.num = -self.num
            self.den = -self.den
```

Defining Objects

- The string and repr dunder
  - string dunder is called when we apply the str function on objects
  - repr dunder is used to show the state of an object
  - Both need to return a string
    - If there is no string dunder, then Python uses the repr dunder and vice versa

class Rational

- The two dunders

```
def __str__(self):
    return f'{self.num}/{self.den}'

def __repr__(self):
    return f'{self.num}/{self.den}'
```