Introduction to Computer Programming
Lecture 5.1:

Classes in Python

Hemma Philamore

Department of Engineering Mathematics

What are classes?

Usually we think of variables and functions as separate entities.

Variables

Functions

B = ”hello”

C = (4, 5, 9)

D = [3, 5, 6, 7]

print(B)

type(C)

len (D)

What are classes?

Variables have functions that can be executed using a dot (.).

The functions that can be used depend on the variable type.

Variables

Functions

b = "Hello"

c = (4,5,9)

d = [3, 5, 6, 7]

Functions for **tuple**:
c.count()
c.index()

Functions for **list**:
d.count()
d.index()
d.append()
d.extend()
d.insert()
d.remove()
d.pop()
d.clear()
d.sort()
d.reverse()

b.find("e")

c.count(4)

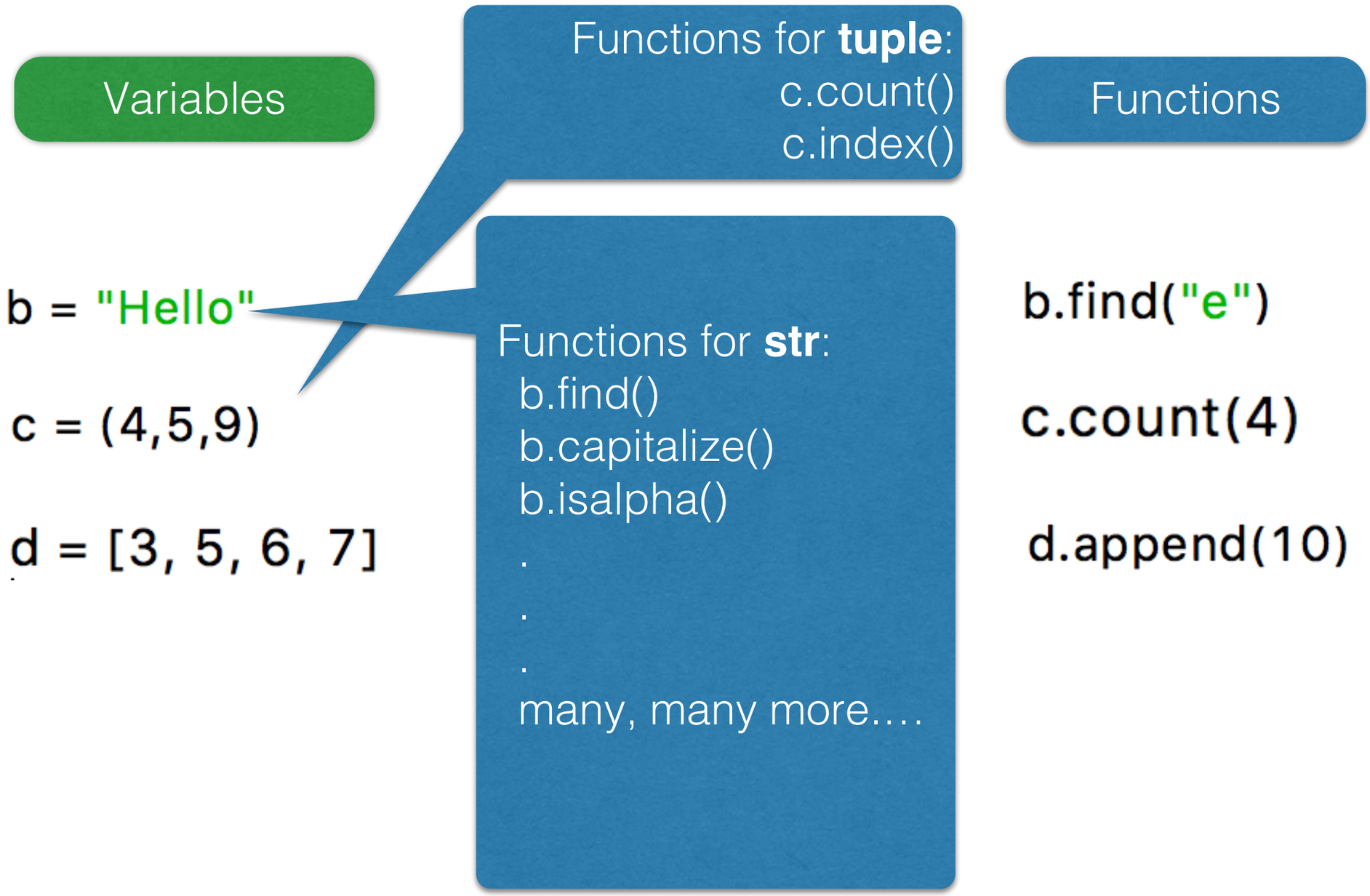
d.append(10)

print(b, c, d)

What are classes?

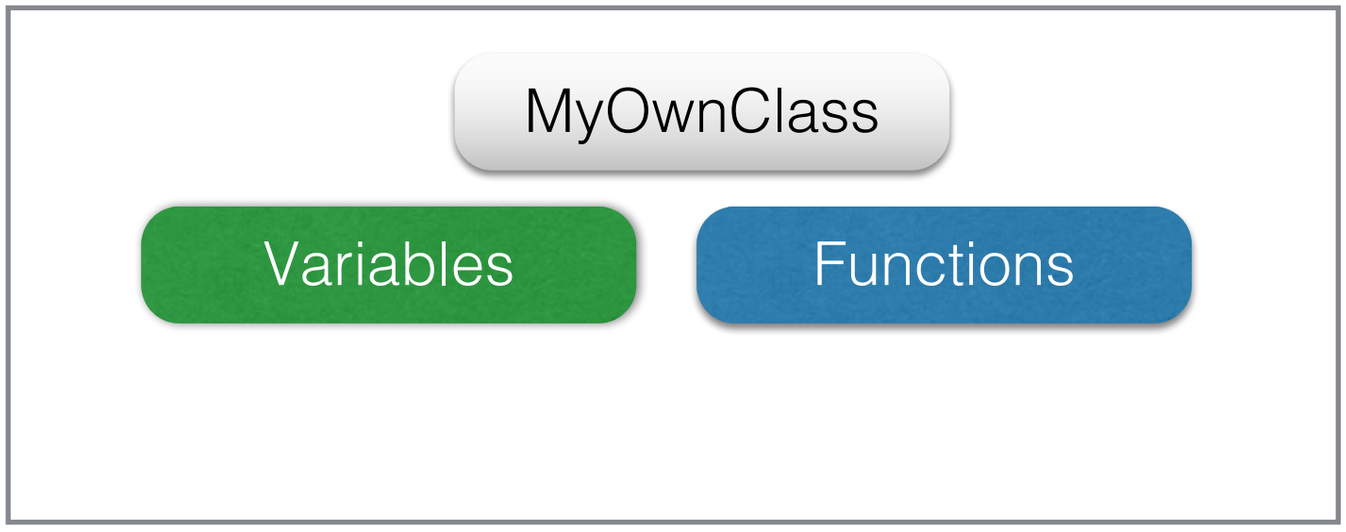
Variables have functions that can be executed using a dot (.).

The functions that can be used depend on the variable type.



What are classes?

A class can be seen as combing values and corresponding functions in one entity!




```
>>> a = 1.45
>>> type(a)
<class 'float'>
```

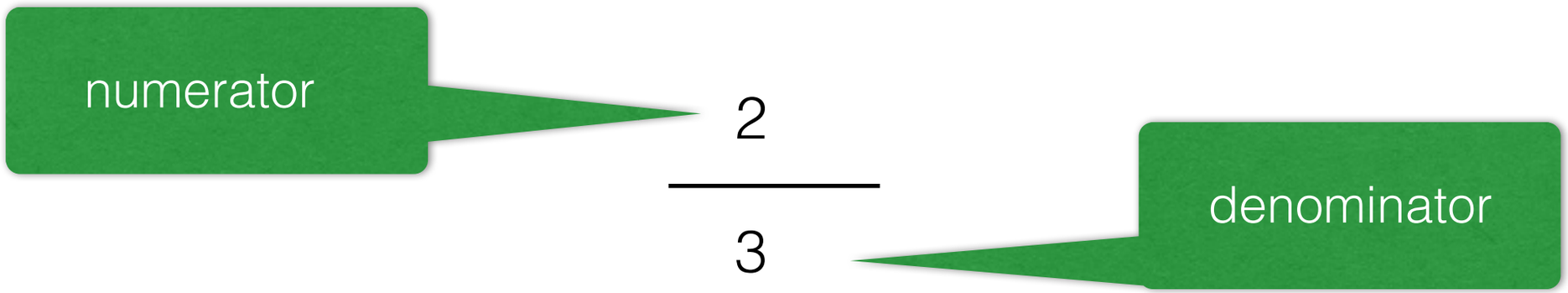
```
>>> b = [5,6,7,1]
>>> type(b)
<class 'list'>
```

Python is special: All build-in data types are “classes”!

Disclaimer: Note that the idea of classes (object orientated programming) is much more complex (inheritance, encapsulation, polymorphism, etc.)

Let's make our own class

Example: We want to build a <u>class</u> that represents fractions.



Possible functions:

class specific functions

- get float value
- print out nicely
- get numerator
- get denominator
- get fraction from float
- mathematical functions (addition, multiplication, …)

Let's make our own class

Class name

Convention: begin with capital letter

Keyword 'class'

parentheses

colon

Indentation

```
class MyFraction():
    """ Constructor """
    def __init__(self, num, den):
        """ Class attributes. """
        self.num = num
        self.den = den
```

Constructor:

Runs automatically when a class object is created

Let's make our own class

Constructor:

- Must be named __init__ (note the two underscores (_) before and after
- Must be defined on 2nd line of class
- Must take self as an input parameter

1st argument is 'self' (required)

Input Arguments when a class object is created (optional)

```
class MyFraction():
    """ Constructor """
    def __init__(self, num, den):
        """ Class attributes. """
        self.num = num
        self.den = den
```

Class attributes:

Variables belonging to the class

Let’s make our own class

self. behaves like the pronoun **'my’**
('my', means someone totally different when said by someone else)

Inside the class: **self.num (my** num , **my** den)
Outside of the class: **my_fraction.num**

```
class MyFraction():
    def __init__(self, num, den):
        self.num = num
        self.den = den

fraction = MyFraction(1, 2)

print(fraction.num)
```

1

Let's make our own class

Methods:
Functions belonging to the class

```
import math

class MyFraction():
    """ Constructor """
    def __init__(self, num, den):
        """ Class attributes.
        self.num = num
        self.den = den
        self.normalize()

def normalize(self):
    gcd = math.gcd(self.num, self.den)
    self.num = int(self.num / gcd)
    self.den = int(self.den / gcd)
```

NOTE : Class methods can be called inside of other methods, including __init__

1st argument is 'self'

Attribute names begin with 'self.'

self. behaves like the pronoun **'my'**

Inside the class, we are talking about **my** num , **my** den → **self.num**

Outside of the class → **my_fraction.num**
('my', means someone totally different when said by someone else.)

```
1  import math
2
3  class MyFraction():
4      """ Constructor """
5      def __init__(self, num, den):
6          """ Class attributes. """
7          self.num = num
8          self.den = den
9          self.normalize()
10
11     def normalize(self):
12         gcd = math.gcd(self.num, self.den)
13         self.num = int(self.num/gcd)
14         self.den = int(self.den/gcd)
15
16 my_fraction = MyFraction(4, 8)
17
18 print(my_fraction.num)
19 print(my_fraction.den)
```

Normalized values of num and den

```
1
2
```

```python
import math

class MyFraction():
    """ Constructor """
    def __init__(self, num, den):
        """ Class attributes. """
        self.num = num
        self.den = den
        self.normalize()

    def normalize(self):
        gcd = math.gcd(self.num, self.den)
        self.num = int(self.num/gcd)
        self.den = int(self.den/gcd)

    def eval(self):
        return(self.num / self.den)

my_fraction = MyFraction(4, 8)

print(my_fraction.eval())

x = 1
print(eval('x + 1'))
```

```
0.5
2
```

Re-definition of built-in function for this class

Class method **.eval()**

Built in function **eval(…)** still works normally

We could define a class method **print**….

```
1  import math
2
3  class MyFraction():
4      """ Constructor """
5      def __init__(self, num, den):
6          """ Class attributes. """
7          self.num = num
8          self.den = den
9          self.normalize()
10
11     def normalize(self):
12         gcd = math.gcd(self.num, self.den)
13         self.num = int(self.num/gcd)
14         self.den = int(self.den/gcd)
15
16     def eval(self):
17         return(self.num / self.den)
18
19     def print(self):
20         print(" " + str(self.num) + "\n---\n" + " " + str(self.den) + "\n")
21
22 my_fraction = MyFraction(4, 8)
23
24 my_fraction.print()
25
```

```
 1
---
 2
```

…but it would be annoying to need to remember to write **<item_to_print>.print()** when we are already used to writing print as **print(<item to print>).** Instead…

```python
import math

class MyFraction():
    """ Constructor """
    def __init__(self, num, den):
        """ Class attributes. """
        self.num = num
        self.den = den
        self.normalize()

    def normalize(self):
        gcd = math.gcd(self.num, self.den)
        self.num = int(self.num/gcd)
        self.den = int(self.den/gcd)

    def eval(self):
        return(self.num / self.den)

    def __float__(self):
        return(self.num / self.den)

    def __str__(self):
        return (" " + str(self.num) + "\n---\n" + " " + str(self.den) + "\n")


A = MyFraction(3, 4)

print(A)

print( float(A) + 5.0 )

print("hello world")
```

```
 3
---
 4

5.75
hello world
```

Double underscore (__) used to *overwrite* built-in functions for this class, when called normally

print function uses a string – ***__str__*** will be called

__float__ will be called

print can be used as normal for regular operations

```python
import math

class MyFraction():
    """ Constructor """
    def __init__(self, num, den):
        """ Class attributes. """
        self.num = num
        self.den = den
        self.normalize()

    def normalize(self):
        gcd = math.gcd(self.num, self.den)
        self.num = int(self.num/gcd)
        self.den = int(self.den/gcd)

    def eval(self):
        return(self.num / self.den)

    def __float__(self):
        return(self.num / self.den)

    def __str__(self):
        return (" " + str(self.num) + "\n---\n" + " " + str(self.den) + "\n")

    def __add__(self, other):
        CommonDen = self.den * other.den
        CommonNum = self.num*other.den + other.num*self.den
        return MyFraction(CommonNum, CommonDen)

    def __mul__(self, other):
        return MyFraction(self.num*other.num , self.den*other.den)

A = MyFraction(1, 2)
B = MyFraction(3, 4)

print(A+B)

print(A*B)
```

__***add***__ will be called

__***mul***__ will be called

```
 5
---
 4

 3
---
 8
```

Summary

- **Class** : a “classification” of an object.
  e.g.“person” or “image.”
- **Object** : a particular *instance* of a class.
  e.g. "Hemma" is an instance of “Person.”
- **Attributes** : *Variables* that belong an object.
  e.g. person's name, height, and age.
- **Methods** : *Functions* belong to an object i.e. actions that an object can do.
  e.g. run, jump, sit.
- Class names should begin with a capital letter e.g. Fish, Person