6.009
Fundamentals of Programming

Week 8 Lecture:
Custom Types

Adam Hartz
hz@mit.edu

6.009: Goals

Our goals involve helping you develop as a programmer, in multiple aspects:

- **Programming:** Analyzing problems, developing plans
- **Coding:** Translating plans into Python
- **Debugging:** Developing test cases, verifying correctness, finding and fixing errors

So we will spend time discussing (**and practicing!**):

- High-level design strategies
- Ways to manage complexity
- Details and "goodies" of Python
- A mental model of Python's operation
- Testing and debugging strategies

The Power of Abstraction

Thinking about complicated systems is *complicated*.

The Power of Abstraction

Thinking about complicated systems is *complicated*.
Thinking about simpler systems is often simpler.

The Power of Abstraction

Thinking about complicated systems is *complicated*.
Thinking about simpler systems is often simpler.

Framework for thinking about complicated systems:

- **Primitives**
- Means of **Combination**
- Means of **Abstraction**
- Recognizing meaningful **Patterns**

The Power of Abstraction

Thinking about complicated systems is *complicated*.
Thinking about simpler systems is often simpler.

Framework for thinking about complicated systems:

- **Primitives**
- Means of **Combination**
- Means of **Abstraction**
- Recognizing meaningful **Patterns**

Example: Python *procedures*

- Primitives: `+`, `*`, `==`, `!=`, ...
- Combination: `if`, `while`, `f(g(x))`, ...
- Abstraction: `def`

The Power of Abstraction

Thinking about complicated systems is *complicated*.
Thinking about simpler systems is often simpler.

Framework for thinking about complicated systems:

- **Primitives**
- Means of **Combination**
- Means of **Abstraction**
- Recognizing meaningful **Patterns**

Example: Python *procedures*

- Primitives: `+`, `*`, `==`, `!=`, ...
- Combination: `if`, `while`, `f(g(x))`, ...
- Abstraction: `def`

Example: Python *types*

- Primitives: `int`, `float`, `str`, ...
- Combination: `list`, `set`, `dict`, ...
- Abstraction: `class`

Custom Types

Python provides a means of creating custom *types*:
the `class` keyword.

Today:

- Extending our notional machine to include classes
- What is `self`?
- Examples of creating new types and integrating them into Python

Example: 2-D Vectors

```
class Vec2D:
    pass
```

Example: 2-D Vectors

```
class Vec2D:
    pass

v = Vec2D()
```

Example: 2-D Vectors

```
class Vec2D:
    pass

v = Vec2D()

v.x = 3
v.y = 4
```

Example: 2-D Vectors

```
class Vec2D:
    pass

v = Vec2D()

v.x = 3
v.y = 4

def mag(vec):
    return (vec.x**2 + vec.y**2) ** 0.5
```

Example: 2-D Vectors

```
class Vec2D:
    pass

v = Vec2D()

v.x = 3
v.y = 4

def mag(vec):
    return (vec.x**2 + vec.y**2) ** 0.5

print(mag(v))
```

Example: 2-D Vectors

```
class Vec2D:
    ndims = 2

    def mag(vec):
        return (vec.x**2 + vec.y**2) ** 0.5
```

Example: 2-D Vectors

```
class Vec2D:
    ndims = 2

    def mag(vec):
        return (vec.x**2 + vec.y**2) ** 0.5

v = Vec2D()
v.x = 3
v.y = 4
print(v.x)
print(v.ndims)
```

Example: 2-D Vectors

```
class Vec2D:
    ndims = 2

    def mag(vec):
        return (vec.x**2 + vec.y**2) ** 0.5

v = Vec2D()
v.x = 3
v.y = 4
print(v.x)
print(v.ndims)

print(Vec2D.mag(v))
```

Example: 2-D Vectors

```
class Vec2D:
    ndims = 2

    def mag(vec):
        return (vec.x**2 + vec.y**2) ** 0.5

v = Vec2D()
v.x = 3
v.y = 4
print(v.x)
print(v.ndims)

print(Vec2D.mag(v))

print(v.mag())
```

Example: 2-D Vectors

```
class Vec2D:
    ndims = 2

    def mag(self):
        return (self.x**2 + self.y**2) ** 0.5

v = Vec2D()
v.x = 3
v.y = 4
print(v.x)
print(v.ndims)

print(Vec2D.mag(v))

print(v.mag())
```

Example: 2-D Vectors

```
class Vec2D:
    ndims = 2

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def mag(self):
        return (self.x**2 + self.y**2) ** 0.5
```

Example: 2-D Vectors

```
class Vec2D:
    ndims = 2

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def mag(self):
        return (self.x**2 + self.y**2) ** 0.5

v = Vec2D(3, 4)
```

Example: 2-D Vectors

```
class Vec2D:
    ndims = 2

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def mag(self):
        return (self.x**2 + self.y**2) ** 0.5

v = Vec2D(3, 4)

print(v.mag())
```

Integrating More Closely With Python

Python offers ways to integrate things more tightly into the language: "magic" methods or "dunder" methods. For example:

- `print(x)` is translated implicitly to `print(x.__str__())`
- `abs(x)` is translated implicitly to `x.__abs__()`
- `x + y` is translated implicitly to `x.__add__(y)`
- `x - y` is translated implicitly to `x.__sub__(y)`
- `x[y]` is translated implicitly to `x.__getitem__(y)`
- `x[y] = z` is translated implicitly to `x.__setitem__(y, z)`

For a full list, see:
`https://docs.python.org/3/reference/datamodel.html`, Section 3.3

Let's add a couple of these to `Vec2D` and see the effects.

The Rest of Today: More Examples

We'll see how many we have time for...

- Polynomial
- Linked List
- Memoized Function