LECTURE 4

OBJECT-ORIENTED PROGRAMMING

SPECIAL METHODS AND OVERLOADING

MCS 275 Spring 2021
Emily Dumas

LECTURE 4: SPECIAL METHODS AND OVERLOADING

Course bulletins:

- At this point you **must** have read the syllabus.
- Discord open (link in the zoom chat or Blackboard).
- Worksheet 2 available.

OBJECT-ORIENTED PROGRAMMING

Today we're starting our unit on object-oriented programming (OOP).

We assume knowledge of: Class definitions, creating instances, accessing attributes, calling methods.

We DO NOT assume knowledge of: Subclasses, inheritance, special methods.

REVIEW OF SOME KEY CONCEPTS

- **class** -- A type in that combines attributes (data) and methods (behavior).
- **instance** or **object** -- A value whose type is a certain class (e.g. `"hello"` is an instance of `str`)
- **attribute** -- A variable local to an object, accessed as `objname.attrname`.
- **constructor** -- The method named `__init__` that is called when a new object is created.

SPECIAL METHODS

In Python, built-in operations are often silently translated into method calls.

e.g. `A+B` turns into `A.__add__(B)`

These special method names begin and end with two underscores (__). They are used to customize the way your classes work with built-in language features.

Using these to add special behavior for operators like `+`, `-`, `*` is called operator overloading.

OPERATOR EXAMPLES

Expression	Special method
`A==B`	`A.__eq__(B)`
`A+B`	`A.__add__(B)`
`A-B`	`A.__sub__(B)`
`A*B`	`A.__mul__(B)`
`A/B`	`A.__truediv__(B)`
`A**B`	`A.__pow__(B)`

List of many more in the Python documentation.

MORE SPECIAL METHODS

Expression	Actually calls
`str(A)`	`A.__str__()`
`len(A)`	`A.__len__()`
`abs(A)`	`A.__abs__()`
`bool(A)`	`A.__bool__()`
`A[k]`	`A.__getitem__(k)`
`A[k]=v`	`A.__setitem__(k,v)`

LIVE CODING

Let's build classes:

- `Point` -- point in the plane
- `Vector` -- vector in the plane

Difference of two `Point`s is a `Vector`.

Can multiply a `Vector` by a float or add it to a `Point`.

LANGUAGE FEATURES USED

- `isinstance(obj,classname)` -- returns bool indicating whether `obj` is an instance of the named class (or subclass thereof)
- `NotImplemented` -- Special symbol that operators should return if the operation is not supported

__ADD__ & __RADD__

In evaluating `A+B`, Python first tries

```
A.__add__(B)
```

but if that fails (returns `NotImplemented`), it will try

```
B.__radd__(A)
```

There are reflected versions of all the binary operations (e.g. `__rmul__`).

OVERLOADING DANGER

Given the very flexible overloading system in Python, it's easy to be too clever.

Overloading is best used when a function or operator has a clear meaning for a class, and when the operation is so frequently used that direct method calls would be cumbersome.

Avoid overloading when it makes code harder to understand!

Note: This is good advice, but wasn't actually discussed in Lecture 4.

REFERENCES

- I discussed overloading in MCS 260 Fall 2020 Lecture 24, and used this geometric object module as an example. Overloading is often, but not always, covered in MCS 260.
- See Lutz, Chapter 30 for more information about overloading.
- Lutz, Chapters 26-32 discuss object-oriented programming.

REVISION HISTORY

- 2021-01-22 Retrospective editing based on what was covered
- 2021-01-19 Initial publication