

LECTURE 4

OBJECT-ORIENTED PROGRAMMING

OPERATOR OVERLOADING

MCS 275 Spring 2023

Emily Dumas

LECTURE 4: OPERATOR OVERLOADING

Reminders and announcements:

- Syllabus update coming soon to account for strike
- Project 1 due date will be postponed, details TBA

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.

Need to review these? See:

- [MCS 260 Lecture 25](#)
- [MCS 260 Fall 2021 OOP sample code](#)

We DO NOT assume knowledge of: Subclasses, inheritance, operator overloading.

REVIEW OF SOME KEY CONCEPTS

- **class** — A structure that allows data (attributes) and behavior (methods) to be bundled together, e.g.
`Point2`.
- **instance** or **object** — A value belonging to a class, e.g.
`p = Point2(5, 7)`
- **attribute** — data stored in an object, e.g. `p.x`
- **method** — function that is part of an object (uses and/or modifies its attributes), e.g. `p.radius()`

SPECIAL METHODS / OVERLOADING

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
$A == B$	<code>A.__eq__(B)</code>
$A + B$	<code>A.__add__(B)</code>
$A - B$	<code>A.__sub__(B)</code>
$A * B$	<code>A.__mul__(B)</code>
A / B	<code>A.__truediv__(B)</code>
$A ** B$	<code>A.__pow__(B)</code>

List of many more in the [Python documentation](#).

MORE SPECIAL METHODS

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

LIVE CODING

Let's work on

- `Point2` — point in the plane (a location in 2D)
- `Vector2` — vector in the plane (e.g. the displacement between two points)

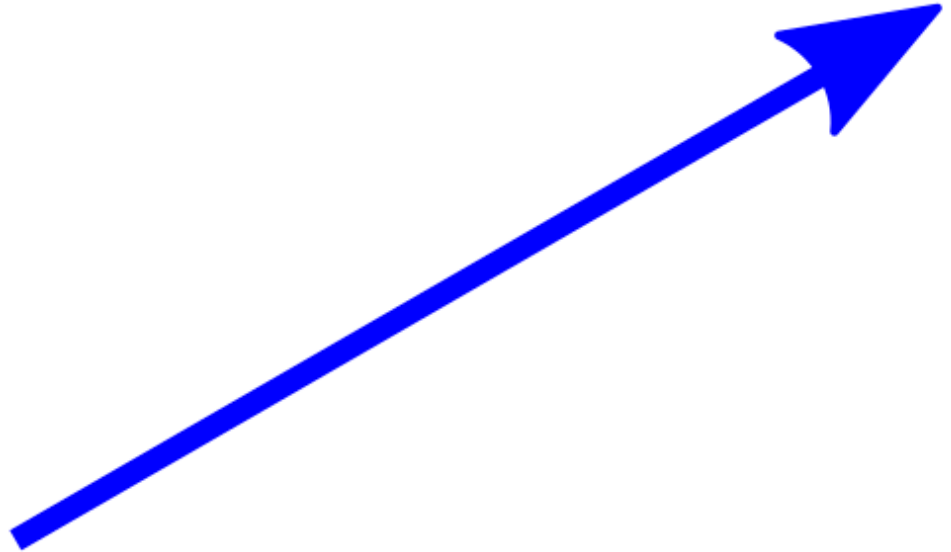
Difference of two `Point2`s is a `Vector2`.

Can multiply a `Vector2` by a float or add it to a `Point2`.

`Point2` **plus** `Vector2` is a `Point2`.

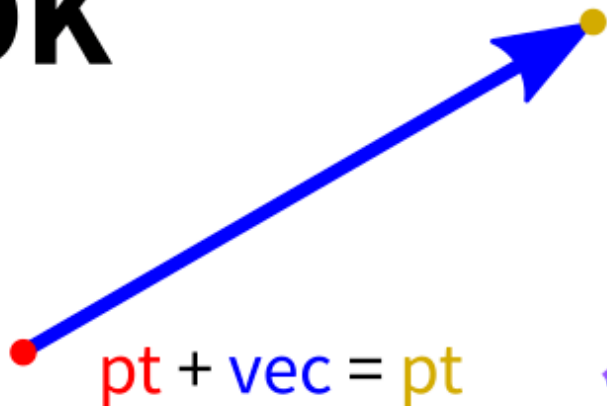


Point
(where?)

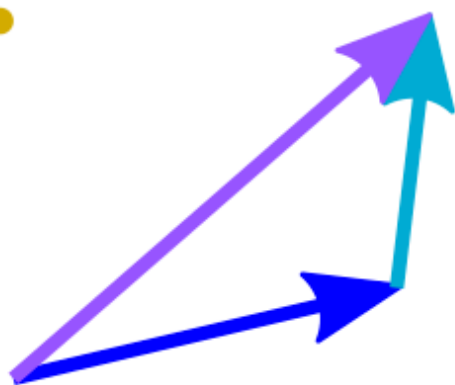


Vector
(which way, how far?)

OK



$$\text{pt} + \text{vec} = \text{pt}$$
$$\text{pt} - \text{pt} = \text{vec}$$



$$\text{vec} + \text{vec} = \text{vec}$$



$$1.5 \times \text{vec} = \text{vec}$$

Meaningless



$$\text{pt} + \text{pt} = ?$$



$$1.5 \times \text{pt} = ?$$

LANGUAGE FEATURES USED

- `isinstance(obj, classname)` -- returns bool indicating whether `obj` is an instance of the named class (or subclass thereof)
- `NotImplemented` -- Special value 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

Overloading is best used when a function or operator has a clear, natural meaning for a class.

If used too much or in unintuitive ways, it makes programs harder to understand.

SINGLETONS

When a class is designed so that it only ever has one instance, the class (or the only instance of it) is called a **singleton**.

We've seen two of these so far:

- `None`, the only instance of `NoneType`
- `NotImplemented`, the only instance of `NotImplementedType`

REFERENCES

- I discussed overloading in [MCS 260 Fall 2021 Lecture 26](#).
- See *Lutz*, Chapter 30 for more information about overloading.
- *Lutz*, Chapters 26-32 discuss object-oriented programming.

REVISION HISTORY

- 2022-01-19 Previous course source material
- 2023-01-23 Update for spring 2023

