LECTURE 4

OBJECT-ORIENTED PROGRAMMING

OPERATOR OVERLOADING

MCS 275 Spring 2022

David Dumas

LECTURE 4: OPERATOR OVERLOADING

Course bulletins:

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

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 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 / 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`	`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

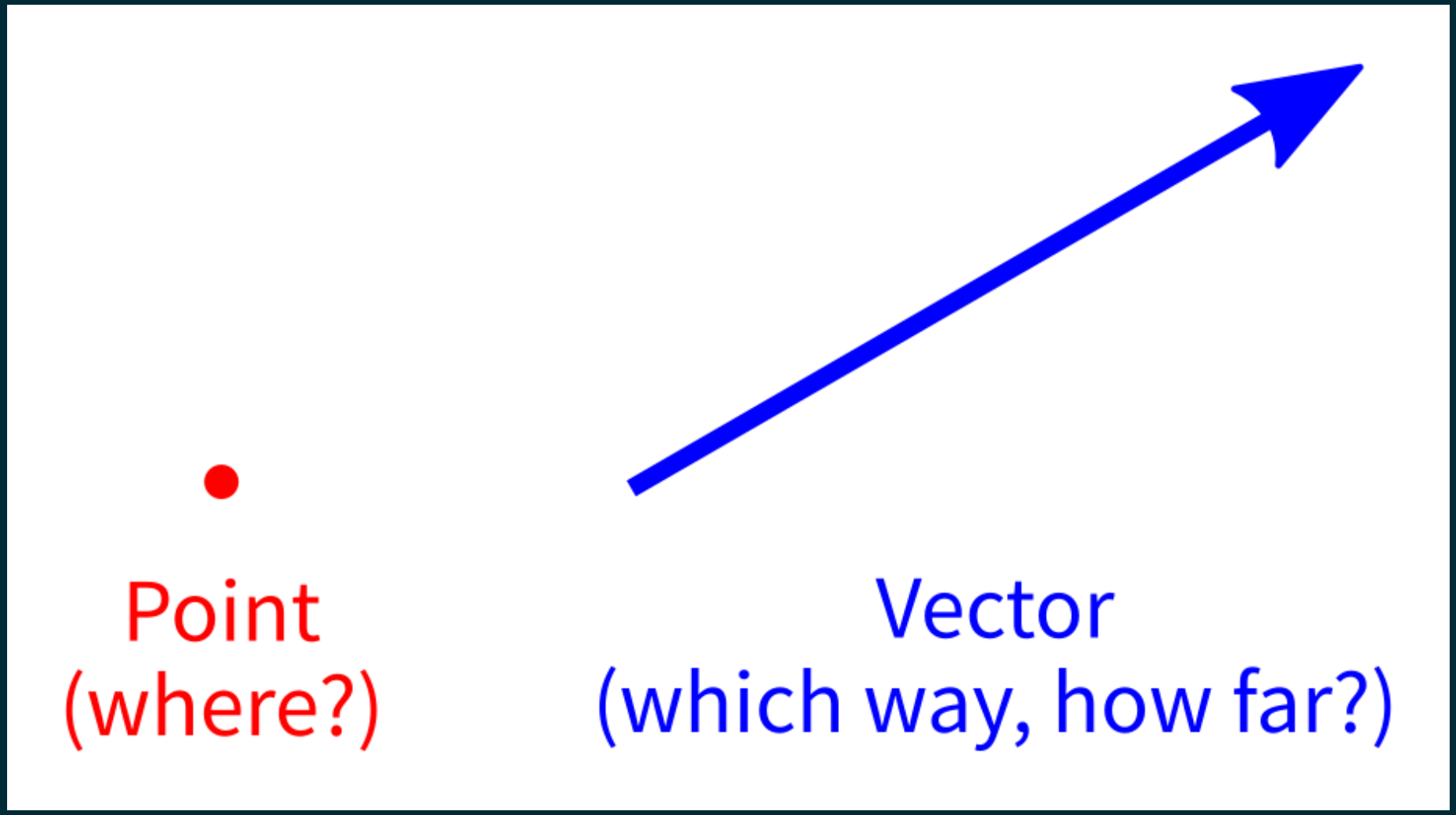
Let's build classes:

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

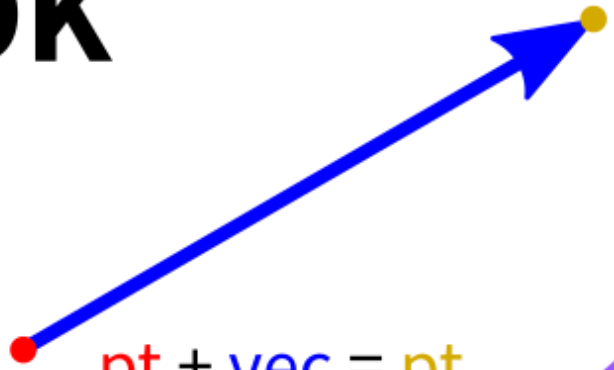
Difference of two `Point2`s is a `Vector2`.

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

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

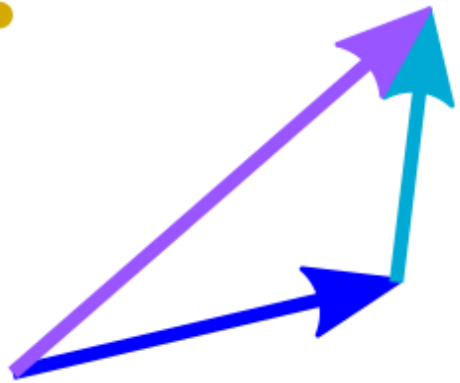

Point
(where?)
Vector
(which way, how far?)

OK



pt + vec = pt

pt - pt = vec



vec+vec=vec

1.5 × vec = vec

Meaningless

pt + pt = ?

1.5 × 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

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!

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 Initial publication