

Classes



Austin Bingham

COFOUNDER - SIXTY NORTH

@austin_bingham



Robert Smallshire

COFOUNDER - SIXTY NORTH

@robsmallshire

Overview



What is a class?

How class relates to type

Define new classes

Instance methods

- Adding to classes
- The self argument

Overview



Initializers

- Compare and contrast with constructors
- Establishing and enforcing invariants

Collaborating classes

Decomposing problems

Overview



Separating interface and implementation

Combine programming paradigms

– *Everything* is an object

Nominal typing and duck typing

Inheritance

Types and Classes

```
>>> type(5)
<class 'int'>
>>> type("python")
<class 'str'>
>>> type([1, 2, 3])
<class 'list'>
>>> type(x*x for x in [2, 4, 6])
<class 'generator'>
>>>
```

Classes

Define the structure and behavior of objects.

Act as a template for creating new objects.

Classes control an object's initial state, attributes, and methods.

Object-oriented Programming



Classes can make complex problems tractable.

But they can make simple problems unnecessarily complex.

Python lets you strike the right balance between functions and classes.

Defining Classes

Defining a Class

```
class MyClassName:
```

```
#
```

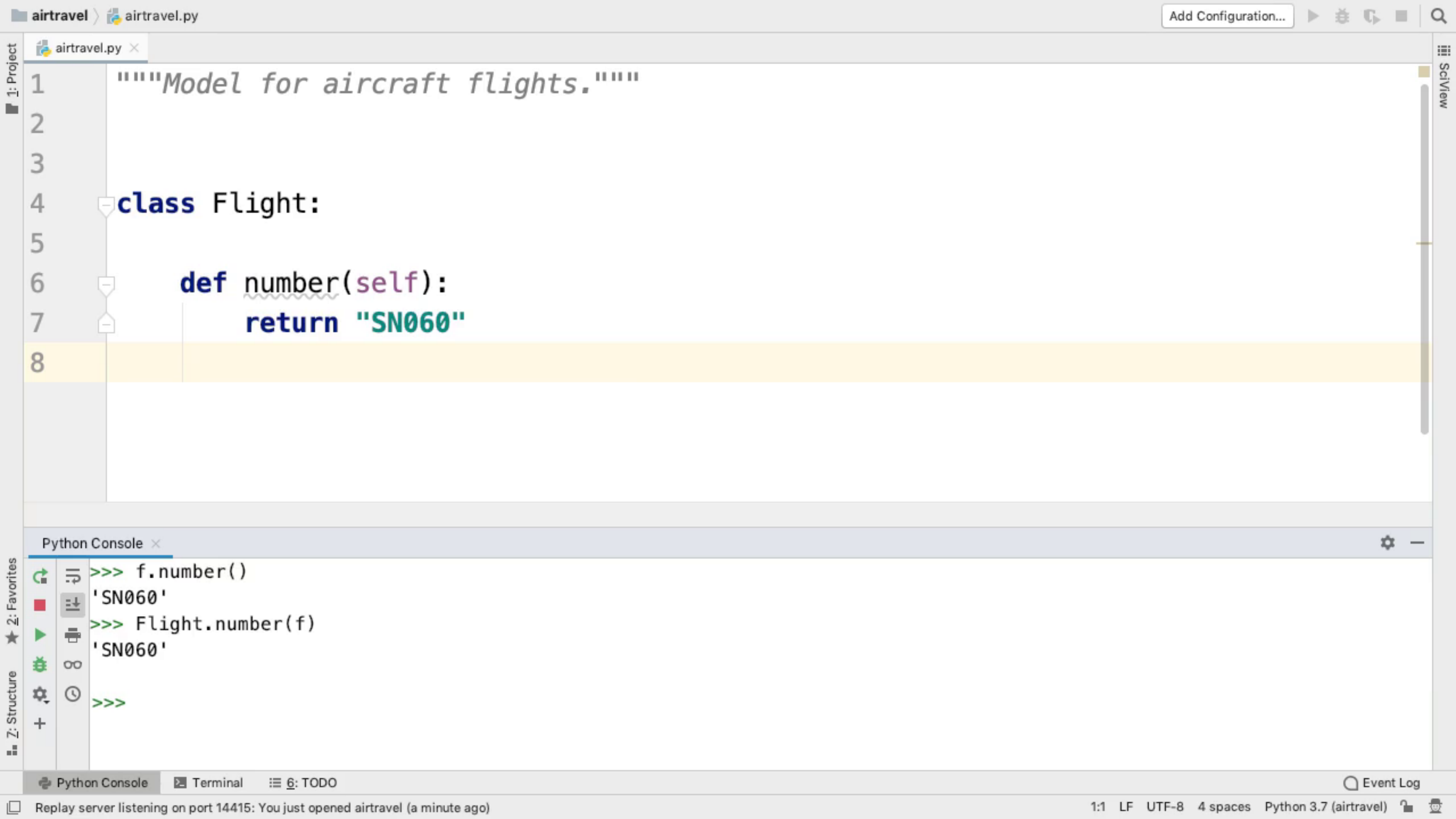


```
# By convention, class
```

```
# names use CamelCase
```

```
# . . .
```





```
__init__()
```

Instance method for initializing new objects

`__init__()` is an initializer,
not a constructor.

`self` is similar to `this` in
Java, C#, or C++.

Why `_number` ?



Avoid **name clash** with `number()`

By convention, implementation details start with **underscore**

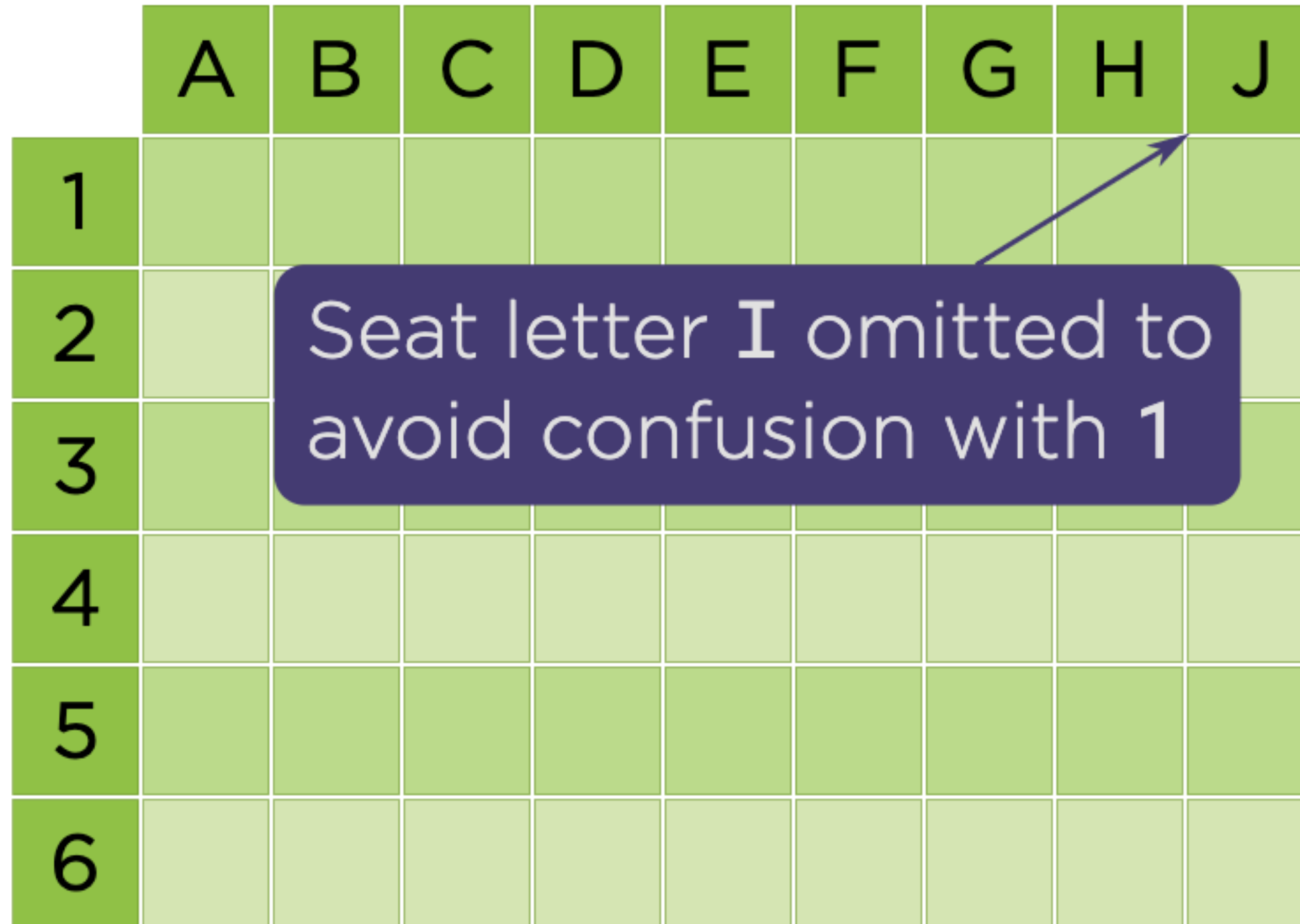
Class invariants

Truths about an object that endure for its lifetime.

Seating Plan

	A	B	C	D	E	F	G	H	J
1									
2									
3									
4									
5									
6									

Seat letter I omitted to avoid confusion with 1





The Law of Demeter

The principle of **least knowledge**.

Only **talk** to your **friends**.

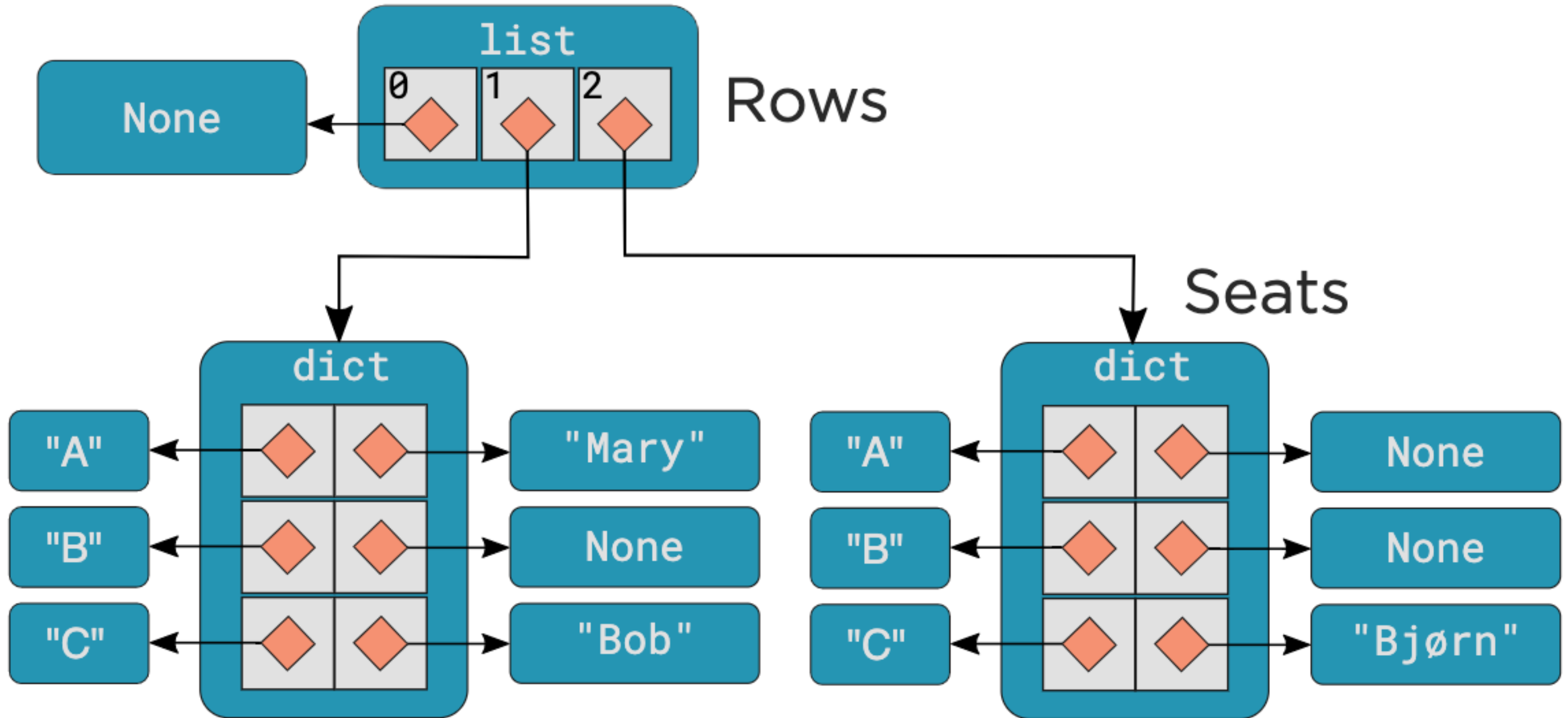
Moment of Zen

Complex is better than complicated

Many moving parts
Combined in a clever box
Are now one good tool



Seat Booking Data Structure



New Requirement: Print Boarding Passes

New Requirement



Print boarding cards in alphabetical order.

Separation of concerns : don't put this on the Flight class.

Remember that functions are objects, too !

Tell! Don't ask.

Tell other objects what to do instead of asking them their state and responding to it.

Polymorphism

Using objects of different types through a uniform interface.

It applies to both functions as well as more complex types.

Polymorphism with the Card Printer



`make_boarding_card()` did not rely on any concrete types.

Any other object that fit the interface would work in place of `console_card_printer()` .

Duck typing

"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck."

James Whitecomb Riley



An object's fitness for use is only determined at use.

This is in contrast to statically typed compiled languages.

Suitability is not determined by inheritance or interfaces.

Inheritance

Late Binding

1. **Nominally-typed languages** use inheritance for polymorphism.
2. Python uses **late binding**.
3. You can try **any method on any object**.

Inheritance in Python is primarily useful for sharing implementation between classes.

Thanks to duck-typing,
Python uses inheritance
less than many other
languages.

Summary



All types in Python have a class

Classes define the structure and behavior of objects

An object's class is set when it's created, and it's fixed for the object's lifetime

Classes are a key part of object-oriented programming in Python

Classes are defined with the `class` keyword

Instances of a class are created by calling the class like a function

Summary



Instance methods are function defined within a class and must accept a self argument

Methods are called using the `instance.method()` syntax

Classes may have a `__init__()` method for initializing new instances

A class's constructor will call `__init__()` if it's present

`__init__()` is not, strictly speaking, a constructor

Arguments passed to the constructor are forwarded to `__init__()`

Summary



Instance attributes are created simply by assigning to them

Implementation details are conventionally prefixed with an underscore

Access to implementation details outside a class can be useful during development

Class invariants should be established within `__init__()`

Methods can have docstrings

Classes can have docstrings

Summary



Method calls on `self` within a method must be preceded with `self`

A module can contain as many classes and functions as you wish

Polymorphism in Python is achieved through duck-typing

Polymorphism in Python doesn't rely on shared base classes or interfaces

Class inheritance in Python is primarily useful for sharing implementation

All methods - including special methods - are inherited

Summary



Strings support slicing

The Law of Demeter can help reduce coupling

We can nest comprehensions

It can be useful to discard the item in a comprehension, conventionally using underscore

You can discard a collection element to simplify handling one-based indexing

Summary



Don't feel compelled to use classes if functions suffice

You can spread complex comprehensions over multiple lines

Statements can be split over lines with backslash; use this primarily to improve readability

The "Tell! Don't ask." approach to object-oriented design can reduce coupling