

Python Fundamentals

Classes

Austin Bingham
🐦 @austin_bingham
austin@sixty-north.com



Presenter

Robert Smallshire
🐦 @robsmallshire
rob@sixty-north.com



pluralsight 
hardcore developer training



Classes in Python

You can get a **long way** with Python's **builtin** types.

But when they're **not right** for the job, you can use classes to create **custom** types.

Structure and Behavior

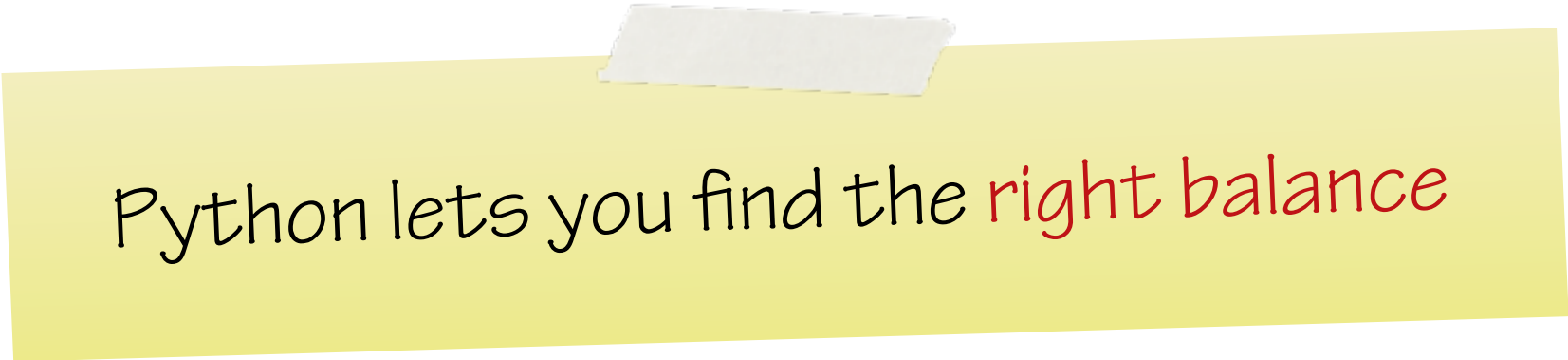
Classes define the **structure** and **behavior** of objects.

An object's **class** controls its **initialization**.

Classes are a Tool

Classes make **complex**
problems **tractable**.

Classes can make **simple**
solutions **overly complex**.



Python lets you find the *right balance*



class

used to define new classes

By convention, class names use

CamelCase

Methods

Method – A function defined within a class

Instance methods – functions which can be called on objects

self – the first argument to all instance methods



`__init__()`

instance method for initializing new objects

Initialization

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

self is similar to
this in C++ or Java.

Why _number?

1. Avoid **name clash** with `number()`
2. By convention, implementation details start with **underscore**

Public!

Private!

Protected!



We're all consenting adults here.



Class Invariants

Truths about an object that
endure for its **lifetime**.

seat letters

rows

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

Seat letter **I** omitted to
avoid confusion with **1**



Law of Demeter

The principle of
least knowledge.

Only **talk** to your
friends.



Moment of Zen

Complex is better
than complicated.

```
class Flight:  
  
    def aircraft_model(self):  
        return self._aircraft.model()
```

Many moving parts



Moment of Zen

Complex is better
than complicated.

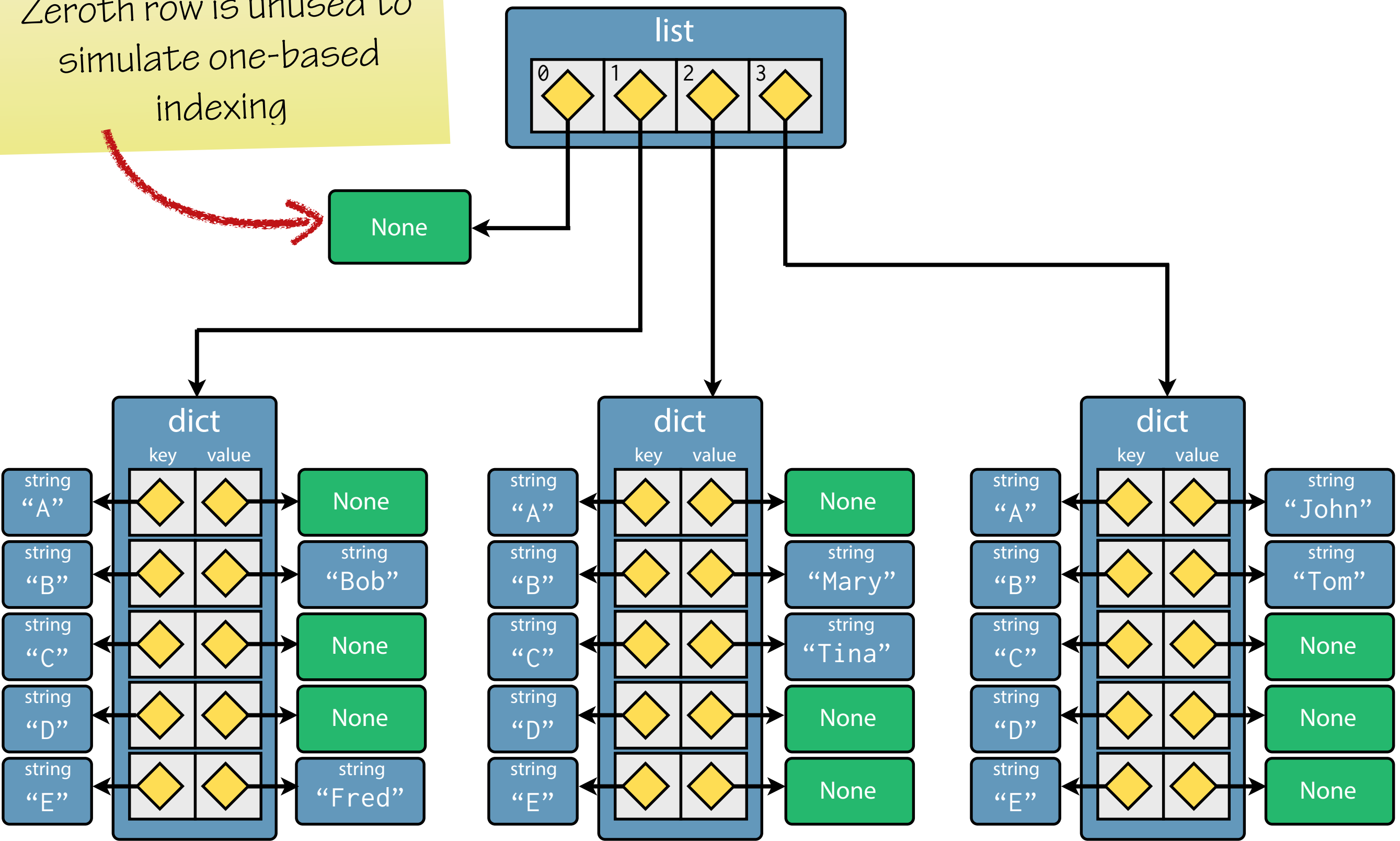
```
class Flight:  
  
    def aircraft_model(self):  
        return self._aircraft.model()
```

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



Seating Data Structure

Zeroth row is unused to simulate one-based indexing



```
rows, seats = self._aircraft.seating_plan()
self._seating = [None] + [ {letter:None for letter in seats} for _ in rows ]
```

Unpack seating plan



```
rows, seats = self._aircraft.seating_plan()
self._seating = [None] + [ {letter:None for letter in seats} for _ in rows ]
```

```
rows, seats = self._aircraft.seating_plan()  
self._seating = [None] + [ {letter:None for letter in seats} for _ in rows ]
```



Use first entry to account for offset

```
rows, seats = self._aircraft.seating_plan()  
self._seating = [None] + [ {letter:None for letter in seats} for _ in rows ]
```



One entry for each row in the aircraft

```
rows, seats = self._aircraft.seating_plan()  
self._seating = [None] + [ {letter:None for letter in seats} for _ in rows ]
```

Discard the row numbers



```
rows, seats = self._aircraft.seating_plan()  
self._seating = [None] + [ {letter:None for letter in seats} for _ in rows ]
```

Dictionary comprehensions



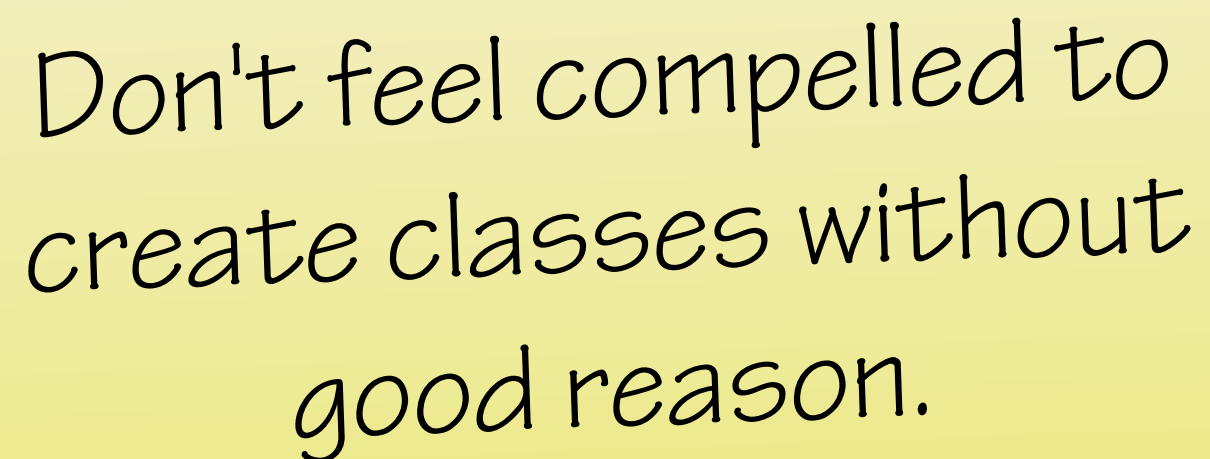
```
rows, seats = self._aircraft.seating_plan()
self._seating = [None] + [ {letter:None for letter in seats} for _ in rows ]
```

List comprehension



New requirement:

Boarding card printer



Don't feel compelled to
create classes without
good reason.

Tell! Don't ask.

Tell objects what to do.

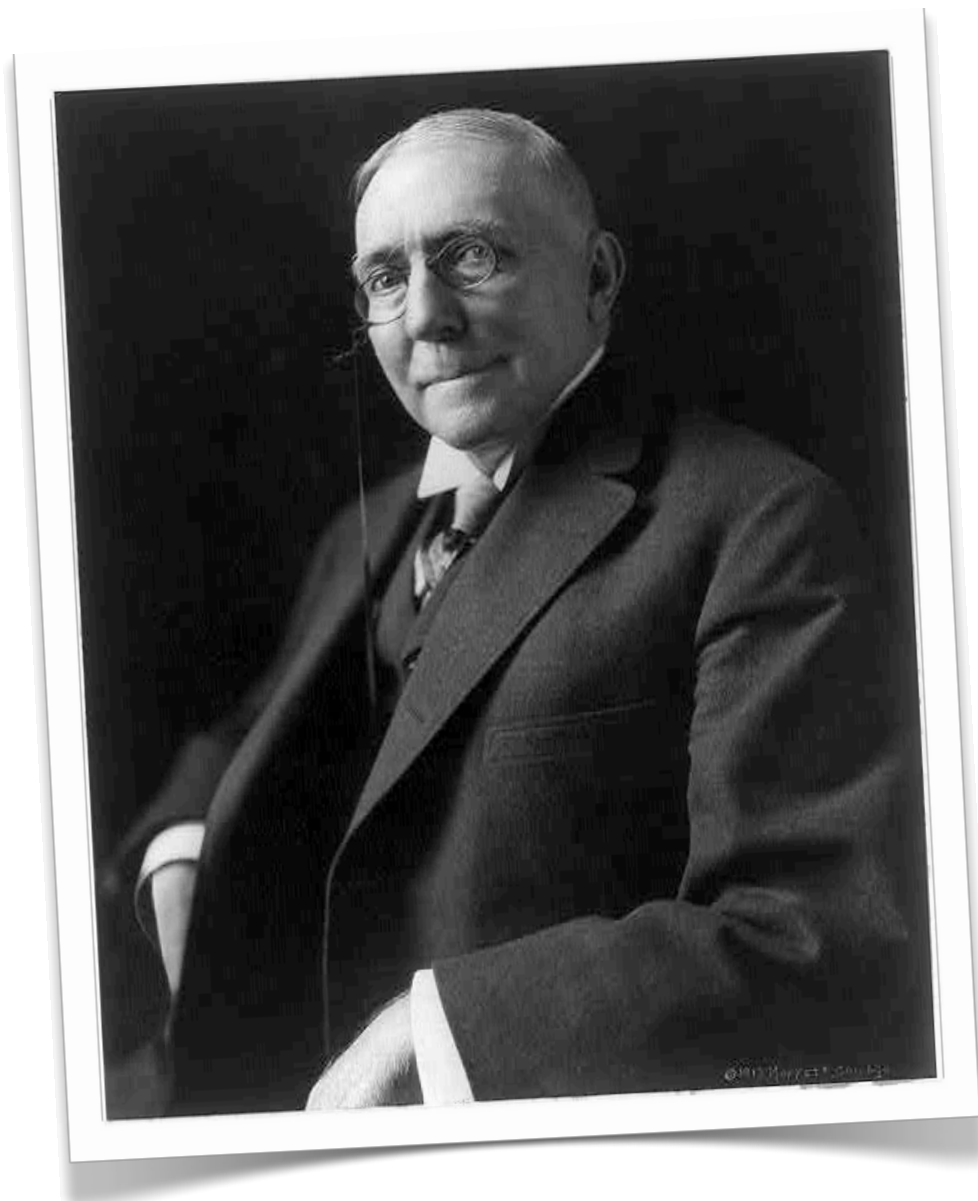
Don't ask for their state.

Polymorphism

Using objects of **different types**
through a **common interface**.

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.”

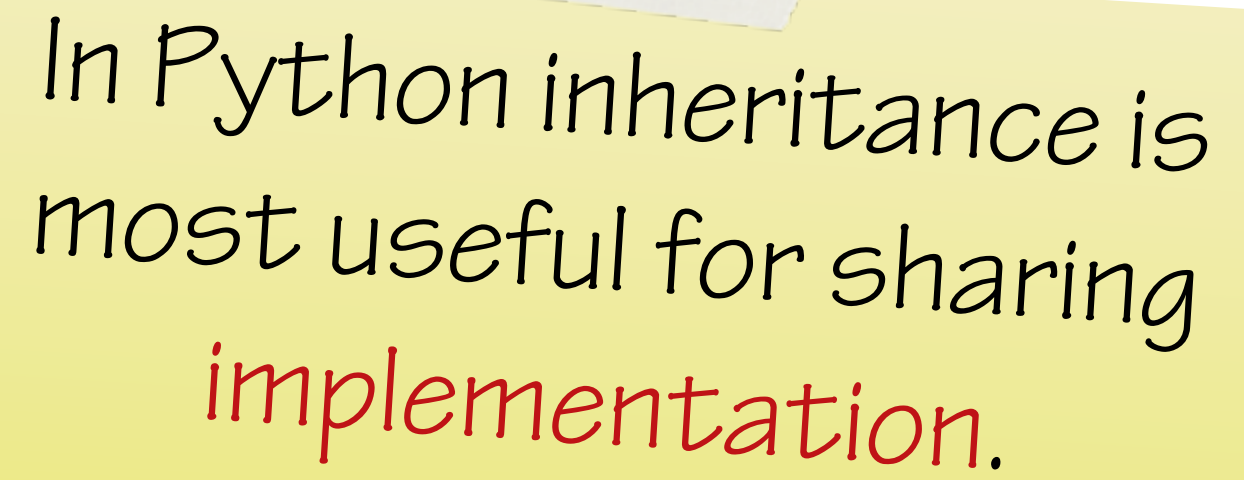


- James Whitcomb Riley

An objects fitness for purpose is determined at the *time of use*.

Inheritance

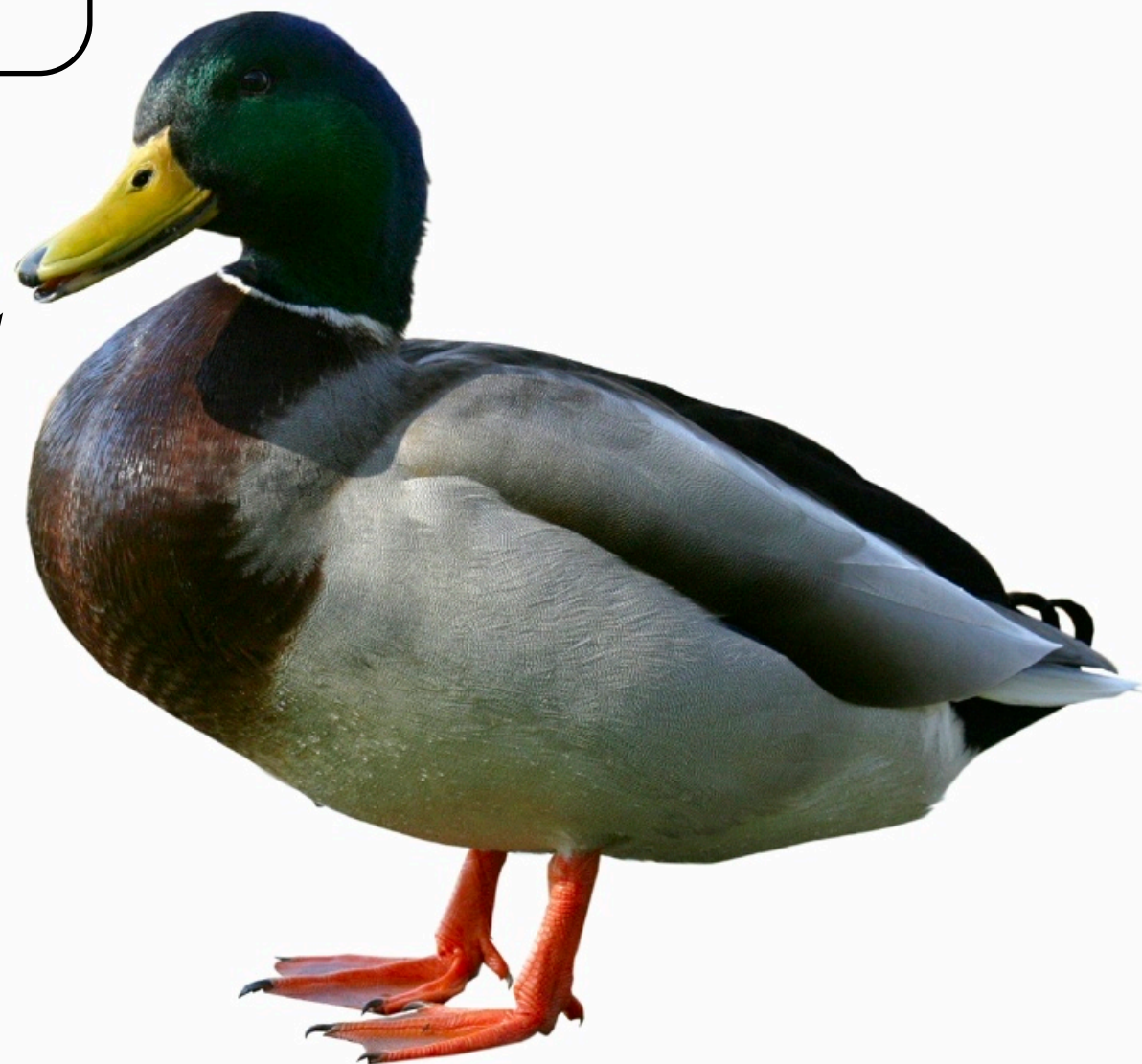
A sub-class can **derive** from a base-class, **inheriting** its behavior and making behavior **specific** to the sub-class.



In Python inheritance is most useful for sharing **implementation**.

Loose coupling is great!

wink-wink, nudge-nudge





Classes Summary

- All types in Python have a 'class'
- Classes define the structure and behavior of an object
- Class is determined when object is created
 - normally fixed for the lifetime
- Classes are the key support for Object-Oriented Programming in Python
- Classes defined using the `class` keyword followed by CamelCase name
- Class instances created by calling the class as if it were a function
- Instance methods are functions defined inside the class
 - Should accept an object instance called `self` as the first parameter
- Methods are called using `instance.method()`
 - Syntactic sugar for passing `self` instance to method
- The optional `__init__()` method initialized new instances
 - If present, the constructor calls `__init__()`
 - `__init__()` is not the constructor
- Arguments passed to the constructor are forwarded to the initializer



Classes Summary

- **Instance attributes** are created simply by assigning to them
- **Implementation details** are denoted by a leading underscore
 - There are no public, protected or private access modifiers in Python
- **Accessing implementation details can be very useful**
 - Especially during development and debugging
- **Class invariants should be established in the initializer**
 - If the invariants can't be established raise exceptions to signal failure
- **Methods can have docstrings, just like regular functions**
- **Classes can have docstrings**
- **Even within an object method calls must be preceded with `self`**
- **You can have as many classes and functions in a module as you wish**
 - Related classes and global functions are usually grouped together this way
- **Polymorphism in Python is achieved through duck typing**
- **Polymorphism in Python does not use shared base classes or interfaces**
- **Class inheritance is primarily useful for sharing implementation**
- **All methods are inherited, including special methods like the initializer**

python **Exception Handling – Summary**

- Strings support slicing, because they implement the *sequence* protocol
- Following the Law of Demeter can reduce coupling
- We can nest comprehensions
- It can sometimes be useful to discard the current item in a comprehension
- When dealing with one-based collections it's often easier just to waste one list entry.
- Don't feel compelled to use classes when a simple function will suffice
- Comprehensions or generator expression can be split over multiple lines
- Statements can be split over multiple lines using backslash
 - Use this feature sparingly and only when it improves readability
- Use “Tell! Don’t ask.” to avoid tight coupling between objects