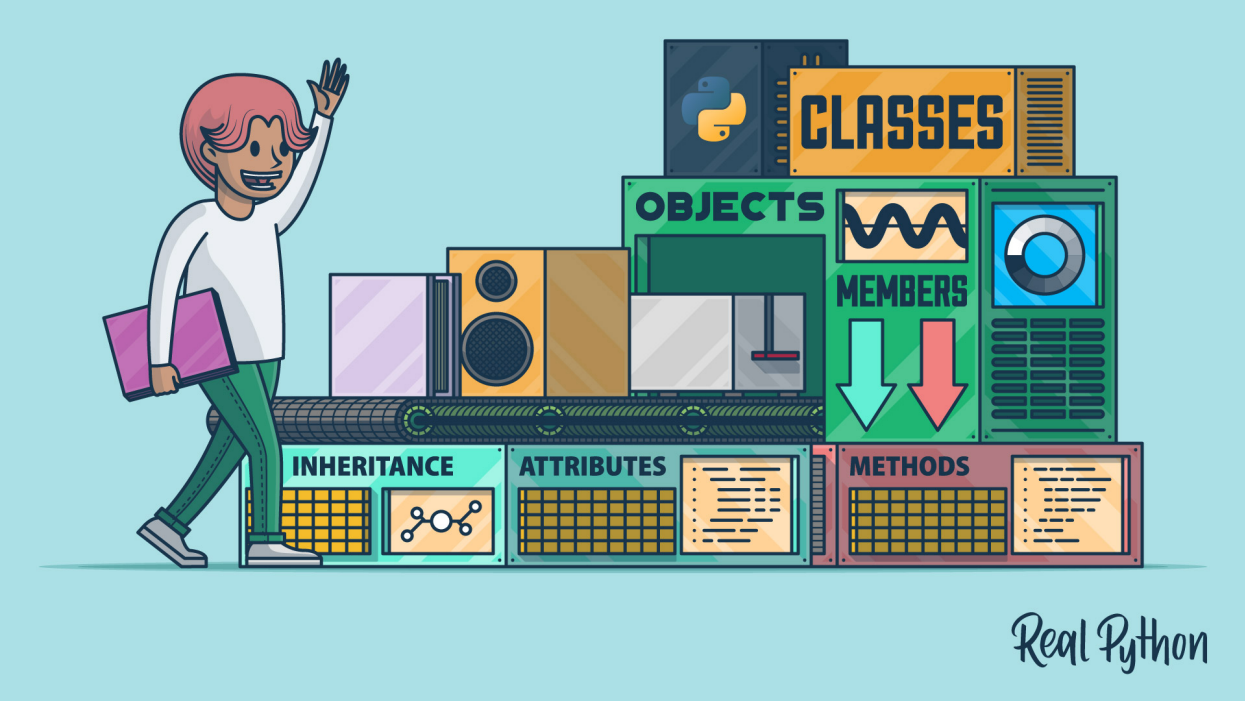
INHERITANCE AND INTERNALS: OBJECT-ORIENTED CODING IN PYTHON



MULTI-PART COURSE

- Course parts:
  - **A: Class Concepts**
  - **B: Inheritance and Internals**
  - C: Design and Guidance

IN THIS COURSE, YOU WILL LEARN ABOUT:

1. Class inheritance
2. Multi-level inheritance
3. Multiple inheritance
4. More special-methods
5. Some classes in the standard library
6. Abstract base classes

VERSIONS

Note:

- Code samples were tested using:
  - Python 3.11.4

OVERVIEW

- The previous course introduced you to declaring classes, their attributes, and their methods
- A class definition can be based on another class
  - Use hierarchical structure to match real-world structure
  - Code re-use
- Classes can inherit from classes that inherit
- Classes can inherit from multiple classes
- Partially define a class to describe a promised interface
- Some structures native to other programming languages have been implemented as classes in Python instead

NEXT UP...

Inheritance

TABLE OF CONTENTS

INHERITANCE

- Creates a hierarchy of class definitions
- Child classes gain all aspects of their parent
- Child classes can override an ancestor's definitions
- Useful for:
  - Expressing actual hierarchical relationships
  - Code re-use

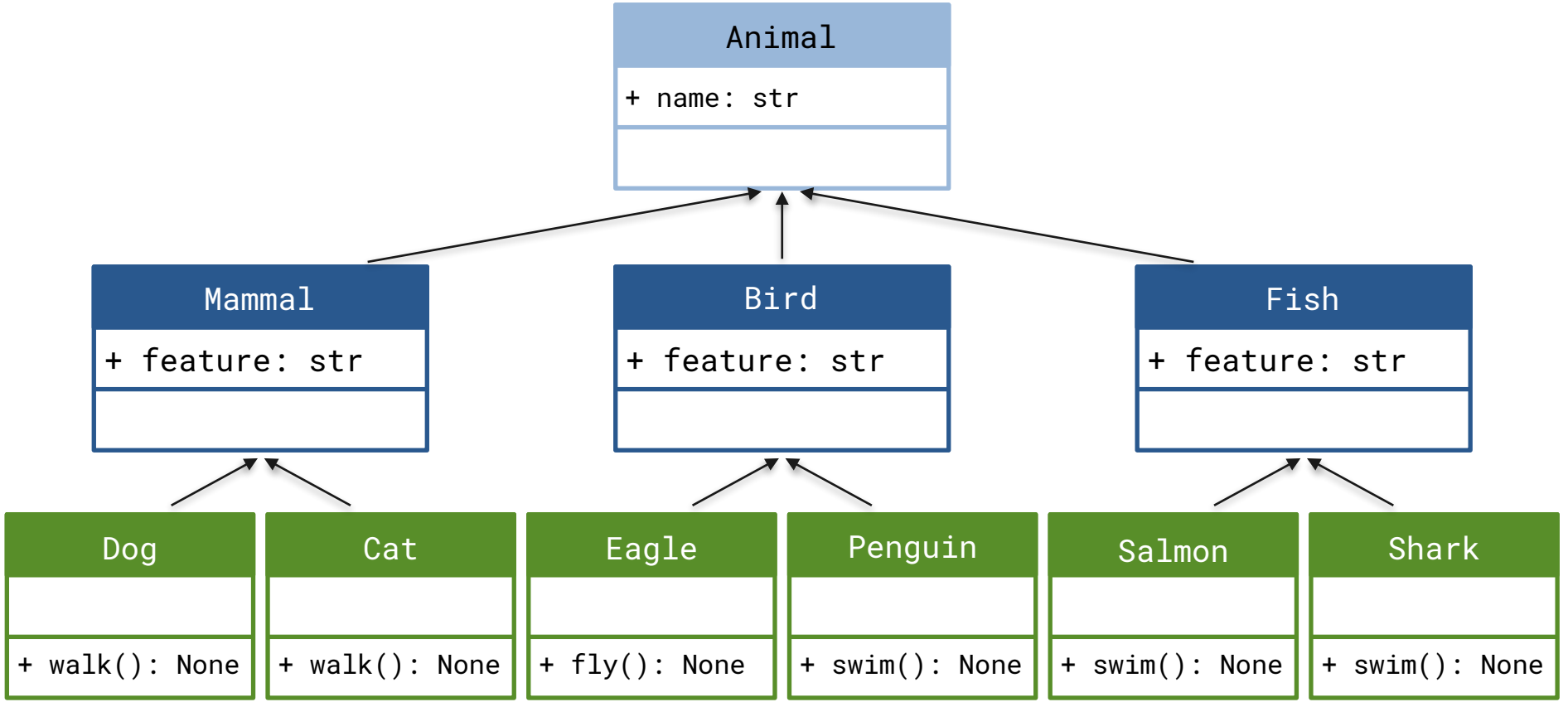
INHERITANCE TERMINOLOGY

- The class you inherit from can be known as:
  - Parent
  - Superclass (or just super)
  - Base class
- A class that inherits can be known as:
  - Child class
  - Derived class
  - Subclass
  - Extending the class
- There are many object-oriented programming languages out there and the terminology bleeds between languages

INHERITANCE TERMINOLOGY

- Overriding a method means disregarding parent's code
  - Writing a method in the child class with the same name as the parent
- Extending a method means including its parent's code
  - Using `super()` in the child's method of the same name

MULTI-LEVEL HIERARCHIES



MULTI-LEVEL HIERARCHIES

```
class Animal:
    def __init__(self, name):
        self.name = name
```

```
class Mammal(Animal):
    feature = "Mammary glands"

class Bird(Animal):
    feature = "Feathers"

class Fish(Animal):
    feature = "Gills"
```

```
class Dog(Mammal):
    def walk(self):
        print("The dog is walking")
...

class Eagle(Bird):
    def fly(self):
        print("The eagle is flying")

class Penguin(Bird):
    def swim(self):
        print("The penguin is swimming")
...

class Salmon(Fish):
    def swim(self):
        print("The salmon is swimming")
```

NEXT UP...

Multiple Inheritance

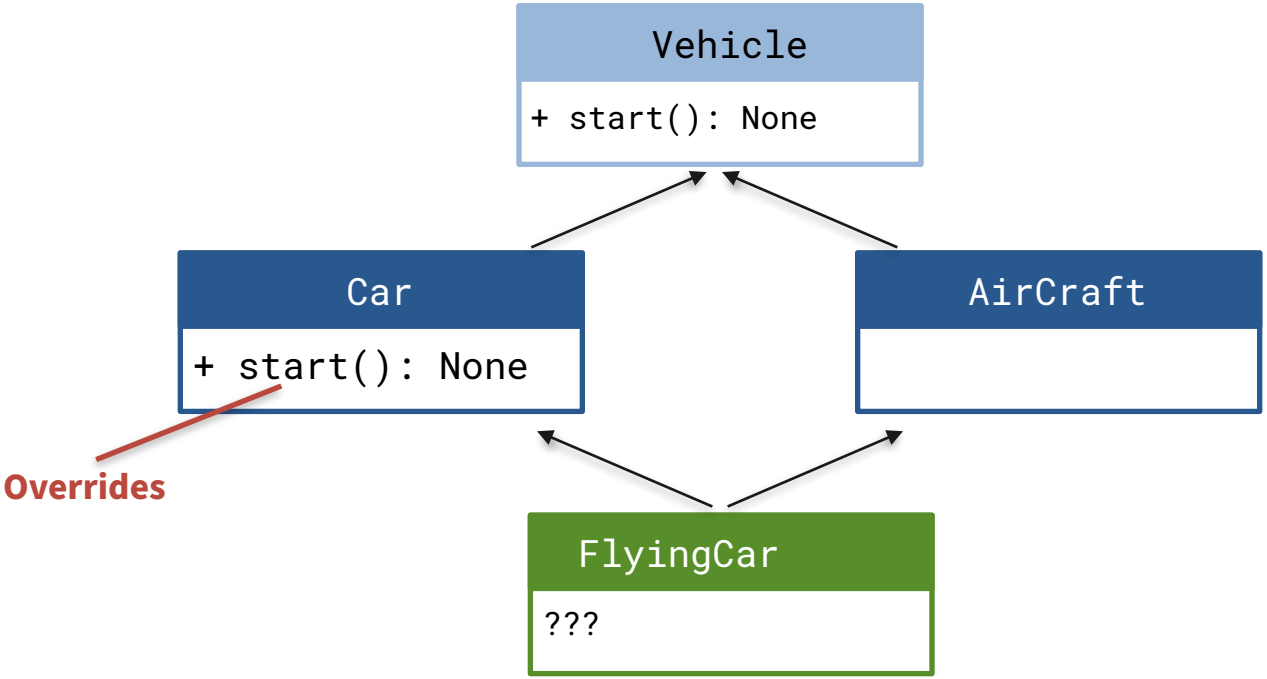
TABLE OF CONTENTS

MULTIPLE INHERITANCE

- A child class can inherit from more than one parent
- Gains all the members of both parents

THE DIAMOND PROBLEM

- Multiple inheritance mechanisms need to resolve the “Diamond Problem”



METHOD RESOLUTION ORDER (MRO)

- MRO is Python's solution to the "Diamond Problem"
- Class members are discovered in the following order:
  - Current class
  - Leftmost superclass
  - Next listed superclass, left-to-right
  - Superclasses of inherited classes
  - Object class
- Inheritance definition order of the class statement is the MRO

MIXINS

- A class that doesn’t declare any attributes
- You inherit from it in order to add its methods to a child class
- Like a utility module but adds functionality to a class instead
- Never instantiated directly
- Common in frameworks

CLASS INTERNALS (TANGENT)

- Classes and instances store their writable members in a dictionary
- Dictionary is named `.__dict__`
- Built-in `vars()` function displays the contents of `.__dict__`
- “Writable”:
  - Instance object: the attributes
  - Class: class attributes and methods

NEXT UP...

More class internals

TABLE OF CONTENTS

DUNDER-GET AND DUNDER-SET

- Can use `@property` and `@.setter` to perform side-effects of getting and setting a value
- Part of the descriptor protocol
- Special methods give you finer control:
  - `.__set_name__()`
  - `.__get__()`
  - `.__set__()`

SLOTS

- Dictionaries have overhead
- Every class has `.__dict__` by default
- A lighter weight class can be defined using `.__slots__`
- Tuple specifying the attributes of the class
- Disallows the class dictionary
    - Raises a `AttributeError` if you attempt to use `.__dict__`

SLOTS

```
class Point:
    __slots__ = ("x", "y")
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

- Less memory and overhead
  - 528 bytes with `.__dict__`
  - 112 bytes with `.__slots__`

NEXT UP...

All the dunder

TABLE OF CONTENTS

OPERATIONS, CONVERTERS, AND COMPARISON

- Everything in Python is an object
- Special methods are called when you invoke:
  - Operation
    - Add, subtract, ... math, bitwise operations
  - Conversion
    - `str()`, `bool()`, ... casting to another type
  - Comparison
    - Equal, less-than, ...
- Override these methods to overload operations

... AND MORE

- Context managers
- Class creation and instantiation
- Containment, is-instance checks, length
- String formatting
- Object and attribute management
  - Get (multiple ways including not there), set, add, delete
- ... and others
  - Over 125 special-methods and attributes

ERRORS

- Different exceptions are raised for different issues with a class:
  - `AttributeError`
    - Member accessed / called doesn't exist on the object
  - `TypeError`
    - Operation not implemented
    - Calling `len()` on an integer
  - `NotImplementedError`
    - Un-implemented method

DRAGONS

- Some common mistakes with classes and instances:
  - Forgetting the self argument on a method
  - Using the class when you mean to use an instance
  - Confusing class and instance attributes
- Possible gotchas:
  - Using non-public members outside a class
  - Over use of multiple inheritance
  - Over use of operator overloading

NEXT UP...

Classes in the standard library

TABLE OF CONTENTS

DATA CLASSES

- Python 3.7 introduced the data class
- Shortcut for creating classes as data objects
  - Dictionary meets named tuple, but as a class
- Declared like a class but using the `@dataclass` decorator
- Type information included

ENUMERATION

- Enumeration, also known as an enum is a grouping of constants
- Built into some languages
- Python added a class based implementation in 3.4
- Inherit from `enum.Enum` and declare attributes
- Still a class so you can add your own methods

CALLABLES

- Everything in Python is an object
- When you invoke a function through parenthesis:
  - You are calling that function object
- The `.__call__()` method is invoked when you “call” an object
- Many “functions” in the Python standard library are actually callable classes
  - All of the conversion calls: `str()`, `int()`, `bool()`, etc.
  - Iteration calls: `range()`, `enumerate()`
  - 44 of the 72 built-in “functions” aren’t

NEXT UP...

Abstraction and interfaces

TABLE OF CONTENTS

ABSTRACT BASE CLASSES

- Define an interface for others to extend
  - Force the implementation of certain methods
  - Without a base implementation
- Built into other languages
- Python implements it as a class and a decorator
  - `abc.ABC` indicates a class contains abstract methods
  - `@abstractmethod` indicates the abstract method
  - “Implement” the abstract method with `pass`

COMMON INTERFACES

- Python supports duck-typing
- Polymorphism
- Special methods (again):
  - `.__iter__()` for use in loops
  - `.__getitem__()` for access using square brackets
  - `.__len__()` for responding to length
- Strings, lists, and tuples all implement this interface

demo/d20_poly.py

NEXT UP...

Summary

TABLE OF CONTENTS

SUMMARY

- In order to re-use code and better represent hierarchical data relations, classes can be built based on other classes
- This is known as inheritance
- A class that extends another class gets all its members by default
- A subclass can override values from the parent
- But still access the parent using `super()`
- Class hierarchies can be multi-level
- Child classes can have more than one parent, known as multiple inheritance
- Abstract base classes allow you to define a partially implemented class

SUMMARY

- Python's approach to operators, convertors, and comparison is to invoke special methods
- The descriptor protocol uses special methods to control how a attribute behaves
- The sequence protocol uses special methods to make objects work like lists
- The standard library uses classes to implement some features
- A dataclass is a shortcut for representing attributes
- Enum is a grouping of constants built using a base class in Python

C: DESIGN AND GUIDANCE

- Just because you can doesn't mean you should
- When to, and when not to, use object-oriented concepts
- **SOLID**: guidance on writing good object-oriented code

Thanks!

Dankie ju faleminderit faleminderit شكرا Grazias Շնորհակալություն Sağ ol eskerrik asko Дзякуй তোমাকে ধন্যবাদ hvala trugéré

благодаря Akeva Chezu ba gràcies Salamat zikomo 谢谢 hvala děkuji Tak dank u Dankon aitäh takk fyri salamat kiitos Merci

Grazas დიდი მადლობა Danke σας ευχαριστώ આભાર Mèsi poutèt ou Na gode Mahalo תודה Dhanyawaad köszönöm þakka þér

Daalụ terima kasih Go raibh maith agat grazie ありがとう matur nuwun ಧನ್ಯವಾದಗಳು សូមអរគុណអ្នក Kamsahamnida ຂໍຂອບໃຈທ່ານ

Ngiyabonga paldies ačiū ви благодариме misaotra Te ma kaih Grazzi xie Māuruuru Dhanyawaadh Welálin баярлалаа barka

Ahéhee' Dhanyabaad miigwetch manana تشکر از شما dziękuję obrigado ਤੁਹਾਡਾ ਧੰਨਵਾਦ mulțumesc спасибо tapadh leibh хвала

ďakujem hvala Waad ku mahadsan tahay Gracias Asante Tack Salamat rahmat நன்றி ధన్యవాదాలు ขอบคุณ tualumba teşekkür

ederim Спасибі آپ کا شکریہ rahmat cảm ơn bạn Diolch yn fawr אַ דאַנק Balika o ṣeun