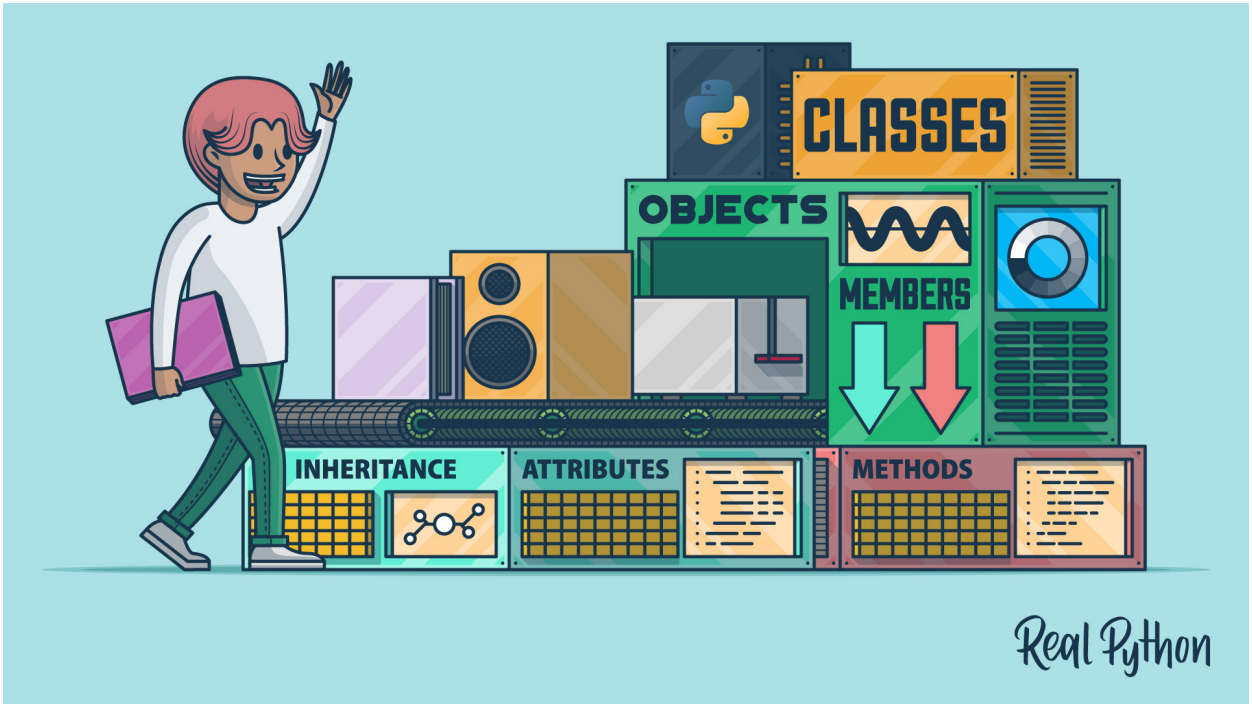
CLASS CONCEPTS : OBJECT-ORIENTED CODING IN PYTHON



MULTI-PART COURSE

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

IN THIS COURSE, YOU WILL LEARN ABOUT:

1. Why write object-oriented code
2. How to write classes
3. What attributes and methods are
4. How to use the descriptor protocol

VERSIONS

Note:

- Code samples were tested using:
  - Python 3.11.4

OVERVIEW

- Python is a mixed language, supporting:
  - Procedural coding
  - Functional coding
  - Object-oriented coding
- Object-oriented coding is the process of grouping data and the operations on that data together, structurally
- The `class` keyword allows you to define structures containing **attributes** and **methods**
- There are different several variations on attributes and methods that give you the power of abstracting your code

NEXT UP...

Why you use an object oriented approach?

TABLE OF CONTENTS

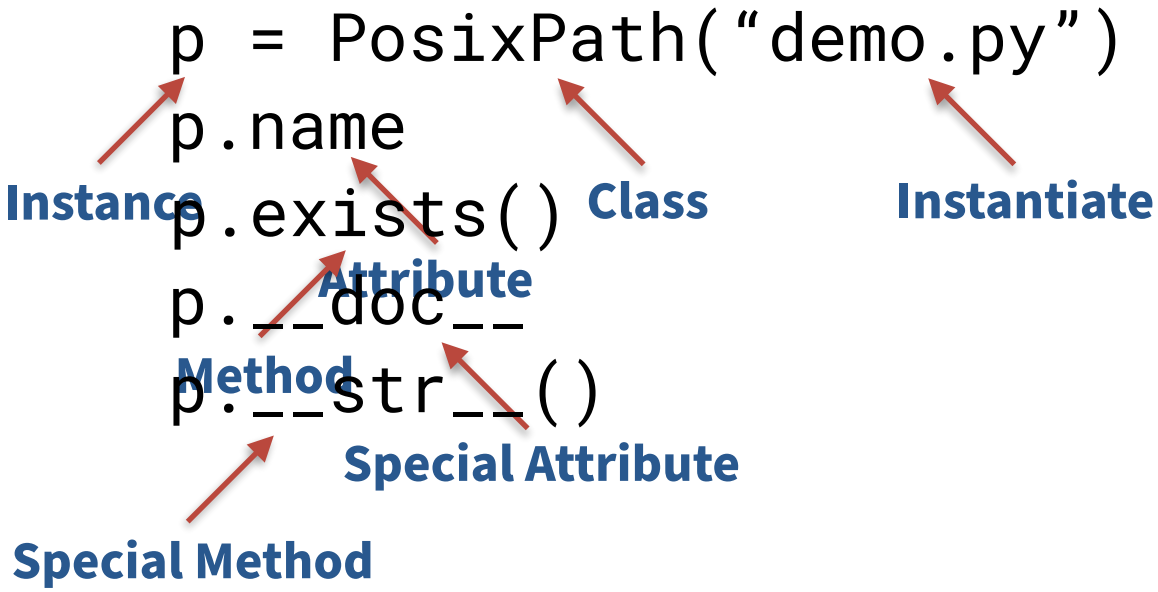
THE CASE FOR OBJECT-ORIENTED CODING

- Logical to associate data and operations on that data together
- Can be done by file grouping
  - Better if the compiler helps to enforce rules

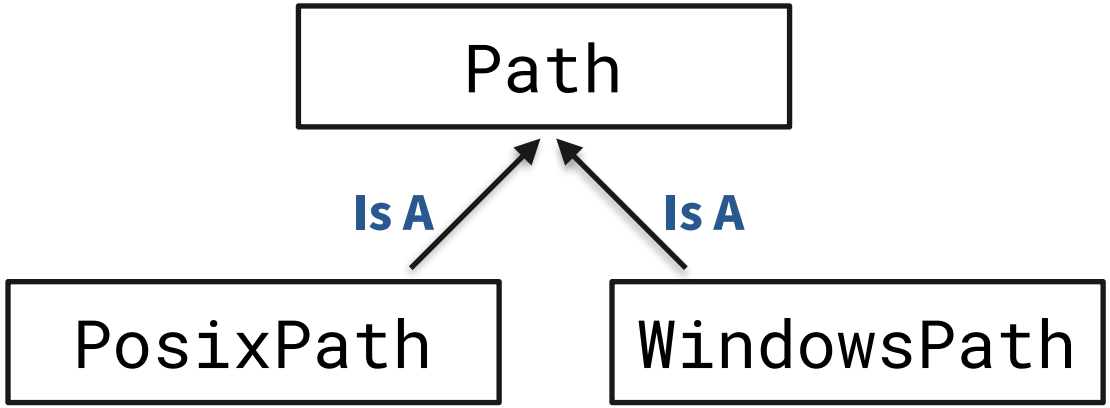
THE CASE FOR OBJECT-ORIENTED CODING

- Structure data for re-use
- “is a” relationships
- Inheritance
- Changes to “person” should effect “employee”

OBJECTS AND CLASSES



OBJECTS AND CLASSES



WHY USE CLASSES

- Code often relates to real-world things (people, places, machines, ...)
  - Modelling data actions together
- Code reuse
  - Inheritance structures help your code be DRY
- Code abstraction
  - Objects provide an interface (API) that can abstract away details
- Flexibility
  - Interfaces can be built for general use (duck-typing, polymorphism)
  - Example: file-like behavior

NEXT UP...

Your first class

TABLE OF CONTENTS

DUNDER INIT

- To instantiate a class you call it with parenthesis
- Arguments to the class constructor are usually stored as attributes
- You write a special `.__init__()` method that gets called when a class is instantiated
- All methods on an object require at least one argument
  - A reference to the object
  - Known as `self`

NAMING CONVENTIONS AND STYLE

- Python variables and attributes use `snake_case`
- Python class names use `PascalCase`
- Python has no concept of private, protected, or public
  - Uses convention instead
- Non-Public members of a class use an underscore prefix
  - A warning to other developers that they shouldn't rely on that member existing
  - Developer style varies on when to use non-public members

NAME MANGLING

- Dunder or special methods have leading double-underscores
- Python uses these for built-in parts of a class
- You can write your own
- Any member with two leading underscores gets renamed
  - “Hides” the member
  - Doesn’t make it un-callable, just not obvious
- `.__member` becomes `._ClassName__member`
- Works for both attributes and methods

NEXT UP...

Attributes

TABLE OF CONTENTS

CLASS vs INSTANCE ATTRIBUTES

- Classes support two kinds of attributes:
  - Instance attributes are on the object instance
  - Class attributes are on the class and shared across instances
- Instance attributes are on `self`
- Class attributes are directly on the class

CLASS vs INSTANCE ATTRIBUTES

- Class attributes can be accessed through an object instance
- Class attributes can only be modified through the Class reference
- Objects can have attributes assigned dynamically
  - Not just in the `.__init__()` or other class code
  - Attempting to assign a Class attribute through the object results in the creation of an instance attribute

NEXT UP...

Properties and descriptor-based attributes

TABLE OF CONTENTS

PROPERTIES

- The `@property` decorator allows you to create a method that is accessed like an attribute
- This means you can do calculation to return a value

DESCRIPTORS

- The `@property` decorator is for accessing a value
- You can also set a value by defining a companion method
- Decorate the method using the property name and `@.setter`
- Allows you to perform side-effects when a value is set
  - Perform calculations
  - Caching
  - Error checks

DESCRIPTORS AND PRIVATE ATTRIBUTES

- A common pattern you'll find in code:
  - Argument to `.__init__()` passes in initial value
  - Value is assigned to private attribute
  - `@property` method with the same name uses the private attribute
  - `@.setter` method with the same name sets the private attribute with error checking
- Some programmers are very strict about this and have no public attributes, only getter/setter methods

NEXT UP...

More on methods

TABLE OF CONTENTS

MULTIPLE TYPES OF METHODS

- There are three types of methods in Python:
  - Instance methods
  - Class methods
  - Static methods

INSTANCE METHODS

- The methods you've used so far are instance methods
- Require an instantiated object
- Typically used to perform operations on associated data
- First argument is always an instance of the object
    - By convention named `self`

CLASS METHODS

- Methods that operate on the class itself
- Created using the `@classmethod` decorator
- First argument is always a reference to the class
  - By convention named `cls`
- Used for:
  - Manipulating data common across all instances
  - Factories

STATIC METHODS

- Static methods require neither a class nor instance
- Created using the `@staticmethod` decorator
- No required arguments
- Typically used for:
  - Grouping data-less functions together (although this can be done equally well with a module)
    - Example: `Converter.km_to_miles(distance)`, `Converter.litres_to_gallons(amount)`
- Very seldom needed in Python and can always be achieved with a classmethod

NEXT UP...

Putting it all together

TABLE OF CONTENTS

PUTTING IT ALL TOGETHER

- So far you've learned about:
  - Declaring classes
  - Attributes
  - Instances
  - Instance, class, and static methods
  - Properties and descriptors

QUICK TANGENT

- A few quick things to know about the coming code:
  - When using a classmethod, the `cls` reference is a class
    - You can instantiate an object by calling it with parenthesis
  - You can use the `**kwargs` mechanism to instantiate a class
    - Key/value pairs in the dictionary become the arguments to the constructor
  - The `.__str__()` method gets called when you convert an object to a string
  - The `.__repr__()` method gets called when you "view" an object in the REPL
    - By convention this should output a string version of the code needed to instantiate the object

NEXT UP...

Summary

TABLE OF CONTENTS

A: Class Concepts

SUMMARY

- The `class` keyword allows you to declare the structure of set of attributes and methods that are used together
- You create an object by instantiating a class
- Each object gets its own data but can use the methods from the class
- The `.__init__()` method is called when a class has been initialized
  - The first argument is `self`, a reference to constructed object
  - You can assign arguments passed in from the constructor onto the object using dot-notation on `self`

SUMMARY

- Instances of an object have their own copy of attributes
  - Accessible using dot notation on the object
  - Or on `self` inside the object's methods
- Class attributes are stored on the class instead of the object
  - Shared across all object instances of that class
- Methods are like functions associated with a class
- Instance methods require at least one argument: `self`
- Class methods operate on the class rather than an instance
  - Require at least one argument: `cls`
- Static methods require neither a class nor an object

SUMMARY

- The descriptor protocol is a pattern you can use to change how attributes are accessed on an object
- The `@property` decorator allows you to have a method accessed like an attribute
- The `@.setter` decorator allows you to define a method called when an attribute's value is set

B: INHERITANCE AND INTERNALS

- A class definition can be based on another class by inheriting its properties
- You can have multiple levels of inheritance
- A class can inherit from multiple classes
- Inheritance is a way of re-using code and making data structures hierarchical

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 gra ありがとう matur nuwu ದನ್ಯವಾದಗಳ ಸ್ ಸូមអរគុណអ្នក Kamsahamnida ຂໍຂອບໃຈທ່ານ

Ngiyabonga paldies ačiū ви благодариме m aotr Te Gra zi xie ru ru Dhanyawaadh Welálin баярлалаа barka

Ahéhee' Dhanyabaad miigwetch manana تشکر از شما dziękuję obrigado ਤੁਹਾਡਾ ਧੰਨਵਾਦ mulţumesc 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