Introduction to SQLAlchemy

Data Boot Camp

Lesson 10.1

Class Objectives

By the end of today’s class, you will be able to:

- Connect to a SQL database by using SQLAlchemy.
- Perform basic SQL queries by using `engine.execute()`.
- Create Python classes and objects.
- Create, read, update, and delete data from a SQL database by using SQLAlchemy’s object-relational mapper (ORM).

WELCOME

Pair Programming Activity:

Looking into SQLAlchemy

In this activity, you'll be working in groups of two or three to research a few questions.

Suggested Time:

5 Minutes

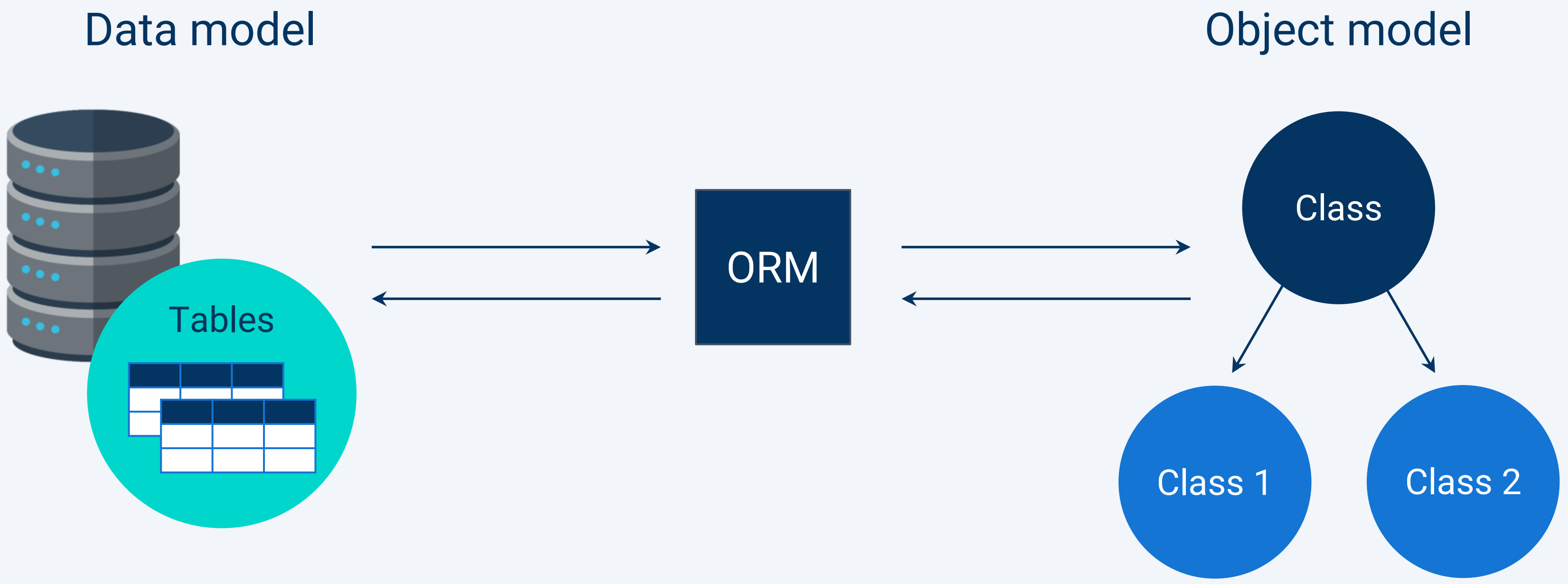
Activity: Looking into SQLAlchemy

Research the following questions:

- What is an ORM?
- What are the benefits of using an ORM?
- What are some of the disadvantages of using an ORM?

Time’s Up! Let’s Review.

Object-Relational Mapping (ORM)



Using An ORM

Advantages

- The ability to work across different SQL dialects by using the same basic Python query.
- The ability to create command line interfaces that allow users to construct SQL queries without needing to know the language.

Disadvantages

- ORMs are like a new dialect of a language, so you have to learn how to use them.
- They may reduce control or ability to optimize a query.

Questions?

Introduction to

SQLAlchemy

SQLAlchemy is a Python library designed to work with SQL databases.

Introduction to SQLAlchemy

SQLAlchemy bridges the differences among the various SQL dialects. A single Python script that uses SQLAlchemy can perform the same query across the different SQL dialects, such as:

- PostgreSQL
- SQLite
- MySQL

SQLAlchemy ORM Is Flexible

It’s possible to query a database using more SQL:

```
data = engine.execute("SELECT * FROM icecreamstore")
```

Or more Python:

```
players = session.query(BaseballPlayer)
for player in players:
    print(player.name_given)
```

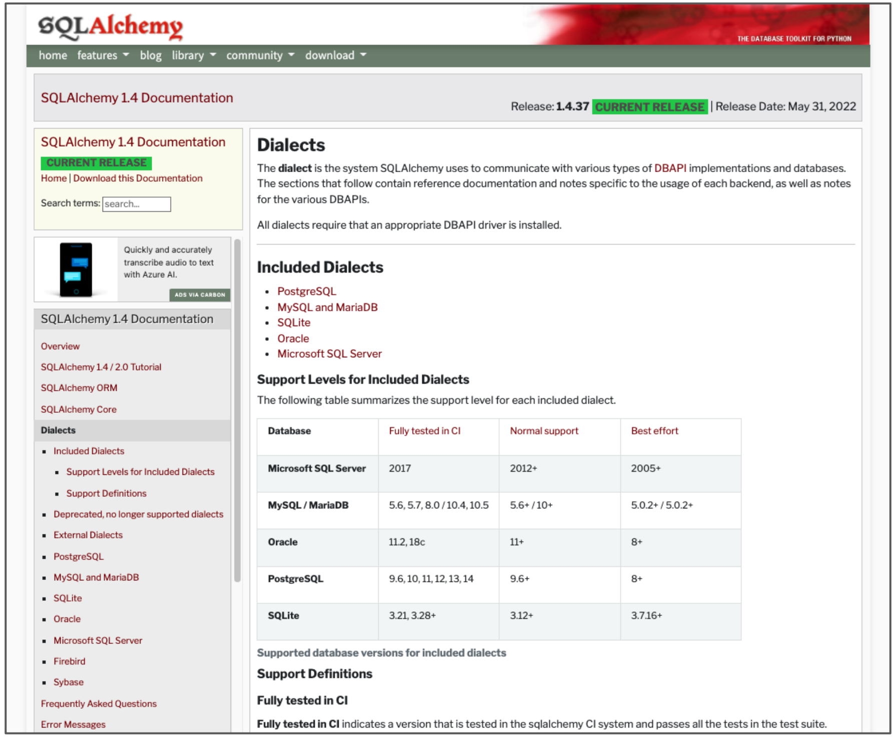
Introduction to SQLAlchemy

The SQLAlchemy documentation lists SQL dialects that are compatible with SQLAlchemy.

Complete documentation of the SQLAlchemy library is on the left side of the page.

Consult this documentation to clarify any questions you may have.

You should be able to fix most bugs this way.



SQLAlchemy 1.4 Documentation

Release: **1.4.37** CURRENT RELEASE | Release Date: May 31, 2022

SQLAlchemy 1.4 Documentation

CURRENT RELEASE

Home | Download this Documentation

Search terms: search...

Quickly and accurately transcribe audio to text with Azure AI.

ADS VIA CARBON

SQLAlchemy 1.4 Documentation

Overview

SQLAlchemy 1.4 / 2.0 Tutorial

SQLAlchemy ORM

SQLAlchemy Core

Dialects

- Included Dialects
  - Support Levels for Included Dialects
  - Support Definitions
- Deprecated, no longer supported dialects
- External Dialects
- PostgreSQL
- MySQL and MariaDB
- SQLite
- Oracle
- Microsoft SQL Server
- Firebird
- Sybase

Frequently Asked Questions

Error Messages

Dialects

The **dialect** is the system SQLAlchemy uses to communicate with various types of DBAPI implementations and databases. The sections that follow contain reference documentation and notes specific to the usage of each backend, as well as notes for the various DBAPIs.

All dialects require that an appropriate DBAPI driver is installed.

Included Dialects

- PostgreSQL
- MySQL and MariaDB
- SQLite
- Oracle
- Microsoft SQL Server

Support Levels for Included Dialects

The following table summarizes the support level for each included dialect.

Database	Fully tested in CI	Normal support	Best effort
Microsoft SQL Server	2017	2012+	2005+
MySQL / MariaDB	5.6, 5.7, 8.0 / 10.4, 10.5	5.6+ / 10+	5.0.2+ / 5.0.2+
Oracle	11.2, 18c	11+	8+
PostgreSQL	9.6, 10, 11, 12, 13, 14	9.6+	8+
SQLite	3.21, 3.28+	3.12+	3.7.16+

Supported database versions for included dialects

Support Definitions

Fully tested in CI

Fully tested in CI indicates a version that is tested in the sqlalchemy CI system and passes all the tests in the test suite.

Ice Cream Connection

Today you will only be working with SQLite databases.

SQLite is a SQL dialect that shares much of the same syntax as PostgreSQL, but it is entirely serverless.

How can a database be serverless?

Building a SQLAlchemy Connection

SQLite reads and writes directly to ordinary disk files, which can in turn be stored on a computer’s hard drive. This makes it much easier to use to perform tests and share between users.

If you do not have SQLite installed, run the following code within your terminal/Git Bash:

```
conda install -c anaconda sqlite
```

Instructor Demonstration

Building a SQLAlchemy Connection

Questions?

Activity: Ice Cream Connection

In this activity, you will be creating and connecting to a new database using SQLAlchemy.

Suggested Time:

10 Minutes

Activity: Ice Cream Connection

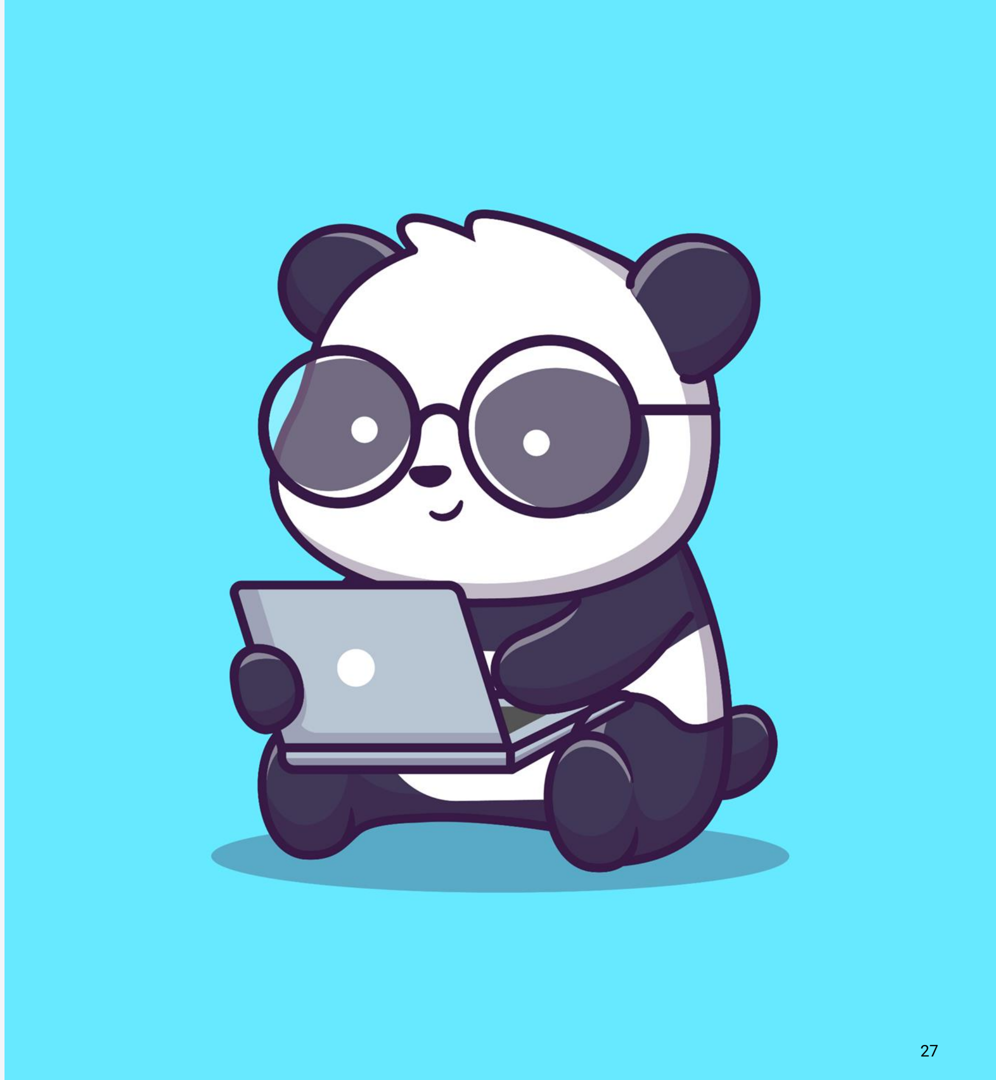
Instructions	Use the database path to create a SQLite engine.
	Use the engine to select all of the rows and columns from the table `icecreamstore`.
	Create a new query that finds the ice cream flavors that cost more than `2.0`.

Time’s Up! Let’s Review.

Questions?

Read All the SQL

One of the most impressive aspects of SQLAlchemy is how it integrates with pandas



SQLAlchemy and Pandas

Once we connect to our SQL database using SQLAlchemy …

```
# Create Engine
engine = create_engine(f"sqlite:///{database_path}")
conn = engine.connect()
```

… we can query directly using Pandas:

```
# Query All Records in the Database
data = pd.read_sql("SELECT * FROM Census_Data", conn)
```

Instructor Demonstration

SQLAlchemy and Pandas

Questions?

Activity: Read All the SQL

In this activity, you will query an external server by using Pandas and SQLAlchemy as you work to create new DataFrames based on U.S. Census data.

Suggested Time:

10 Minutes

Activity: Read All the SQL

Instructions	Create an engine to connect to the Census database.
	Query all the data from the `Census_Data` table, and load it into Pandas.
	Create an engine to connect to the zip database.
	Query all the data from the `Zip_Census` table, and load it in Pandas.
	Show the `.head()` of your newly imported data.
Bonus	Use Pandas's `merge` to combine the two DataFrames.

Time's Up! Let's Review.

Questions?

Preview SQLAlchemy with Classes

Preview SQLAlchemy with Classes

SQLAlchemy is not just for making SQL queries in Python.

It can also update a SQL database using Python classes.

Python classes are traditionally used to bundle data and functions together.

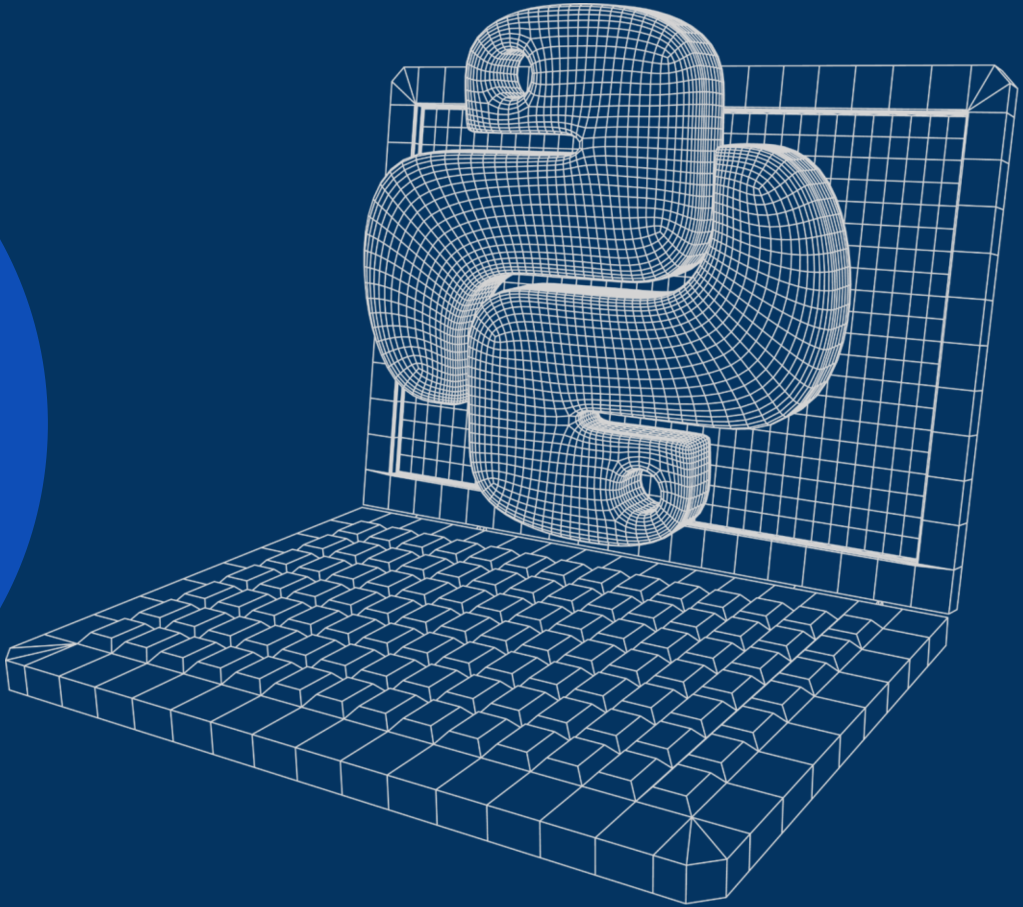
In SQLAlchemy, they are used to define structures.

```python
# Sets an object to utilize the default declarative base in SQLAlchemy
Base = declarative_base()

# Creates Classes which will serve as the anchor points for our Tables
class Dog(Base):
    __tablename__ = 'dog'
    id = Column(Integer, primary_key=True)
    name = Column(String(255))
    color = Column(String(255))
    age = Column(Integer)

class Cat(Base):
    __tablename__ = 'cat'
    id = Column(Integer, primary_key=True)
    name = Column(String(255))
    color = Column(String(255))
    age = Column(Integer)
```

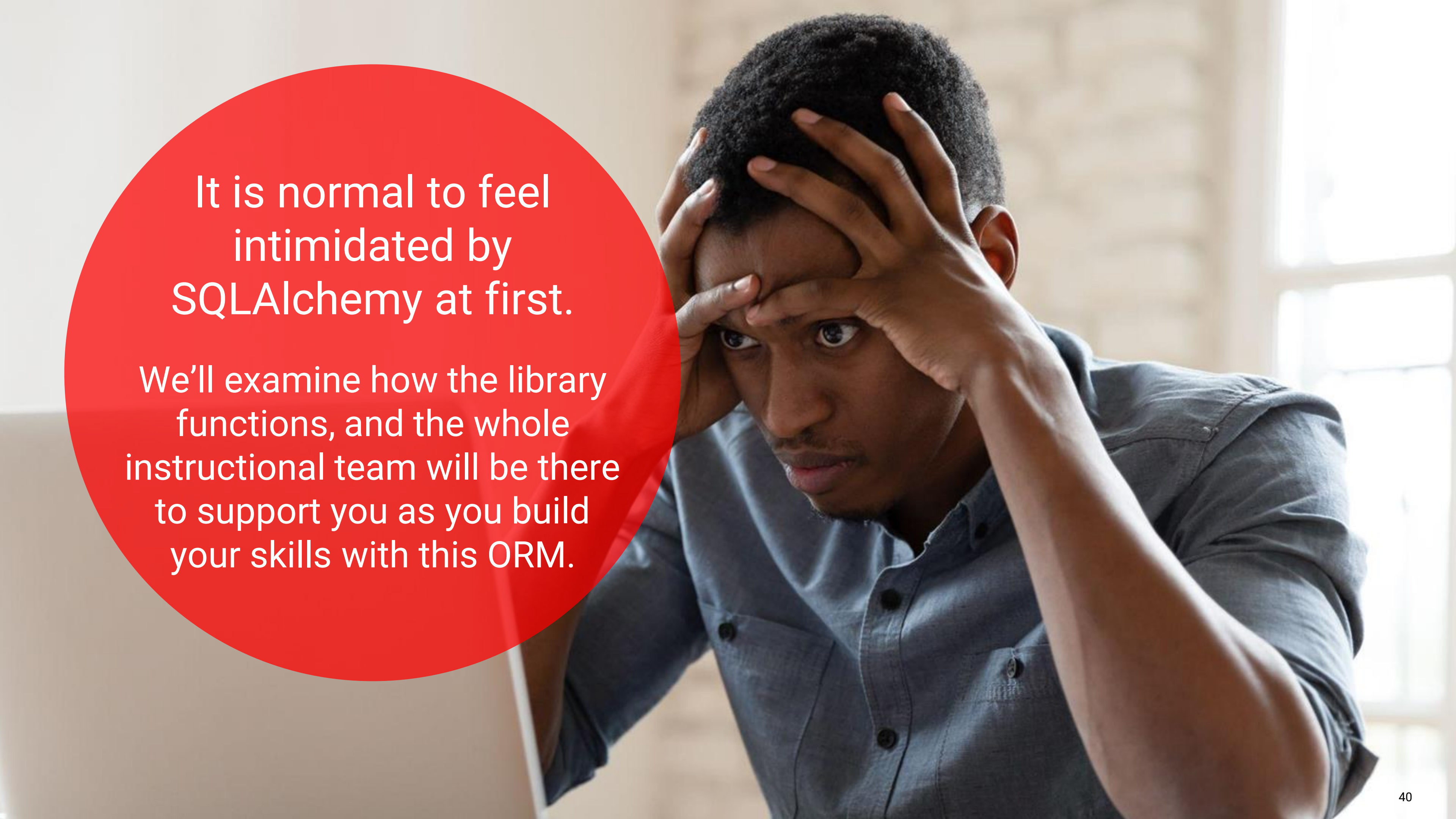
Classes are essentially blueprints for Python objects; they allow developers to create organized variables with keys, values, and methods on the fly.



In the case of SQLAlchemy, we can use classes to make a table blueprint and update the SQL schema.

Instructor Demonstration

Preview SQLAlchemy with Classes



It is normal to feel intimidated by SQLAlchemy at first.

We'll examine how the library functions, and the whole instructional team will be there to support you as you build your skills with this ORM.

Countdown timer

15:00

(with alarm)

Surfer Class

Time for a crash course in object-oriented programming.

Object-Oriented Programming (OOP)

Object-oriented programming (OOP) is a style of coding based around the concept of “objects.” These objects may contain data, often known as **attributes**, and functions, often known as **methods**.

OOP

Encapsulation

Object data (and often functions) can be neatly stored (or encapsulated).

Abstraction

Creating a simple model of something that is complex.

Inheritance

New classes can be created based on other classes (the `Person` class is parent to the `Student` and `Teacher` classes).

Polymorphism

Multiple object types can implement the same functionality.

Python is a class-based programming language.

Objects can be created according to user-created blueprints, allowing developers to rapidly create objects with a similar structure/purpose—just with different values.

Instructor Demonstration

A Schooling on Classes

Questions?

Activity: Surfer Class

In this activity, you will work on creating your own classes in Python.

Suggested Time:

15 Minutes

Activity: Surfer Class

Instructions	Create a class, `Surfer`, and initialize it with name, hometown, and rank-instance variables.
	Create an *instance* of a surfer.
	Then print the name, hometown, and rank of your surfer object.
Bonus	Create a `while` loop that will allow you to continuously create new instances of surfers using `input()`.
	Keep the loop going until the user indicates otherwise.

Time’s Up! Let’s Review.

Questions?

Surfer Class Extended

A Method to the Classes

Creating and attaching methods to Python classes is also easy to accomplish, allowing developers to attach regularly used functions to objects of similar types.

Add the Method

Adding methods to a class is very similar to the `__init__` method discussed earlier:

- define the function using `def`
- provide it with a name
- pass a list of parameters — including self — into the parentheses that follow.

Run the Method

To run the method in code, use the instance of a created object, and then, using dot notation, reference the method.

For example, `doggy.printHello()` would run the `printHello()` method for the `doggy` object.

A Method to the Classes

The `boast() method` contained within the `Expert` class takes in another object as a parameter and then prints out some statements based on its contents.

```
# Define the Expert class
class Expert():

    # A required function to initialize the class object
    def __init__(self, name):

        self.name = name

    # A method that takes another object as its argument
    def boast(self, obj):

        # Print out Expert object's name
        print("Hi. My name is", self.name)

        # Print out the name of the Film class object
        print("I know a lot about", obj.name)
        print("It is", obj.length, "minutes long")
        print("It was released in", obj.release_year)
        print("It is in", obj.language)
```

Instructor Demonstration

A Method to the Classes

Questions?

Activity: Surfer Class Extended

In this activity, you will rework your `Surfer` script as you add in methods to perform specific tasks.

Suggested Time:

10 Minutes

Activity: Surfer Class Extended

Instructions	Create a `Surfer` class that has `name`, `hometown`, `rank`, and `wipeouts` instance variables.
	Create a method called `speak` that prints "Hang loose, bruh!"
	Create a method called `biography` that prints the surfer's name and hometown.
	Create a method called `cheer` that will print "I totally rock man, no wipeouts!" if the surfer has no wipeouts. Otherwise, it prints “Bummer, bruh, keep on keeping on!”
	Create two surfer instances of the Surfer class, and run all the methods.
Bonus	Add a method to your class that prints out how many surfers are currently “shredding.”

Time's Up! Let's Review.

Questions?

Surfing SQL

Time to Code

Back to the SQL

Suggested Time:

20 Minutes

Questions?

Activity: Surfing SQL

In this activity, you will test your SQLAlchemy skills to turn your Python classes into SQL database tables.

Suggested Time:

20 Minutes

Activity: Surfing SQL

Instructions

Modify the `Surfer` class created during the previous activity so that it will function with SQLAlchemy. Use the following parameters:

- `__tablename__` should be "surfers".
- `surfer_id` should be an integer and the primary key.
- `name` should be a string capable of holding 255 characters.
- `hometown` should be a string capable of holding 255 characters.
- `rank` should be an integer.

Create a new class called `Board`, which will function with SQLAlchemy and meet the following parameters:

- `__tablename__` should be "surfboards".
- `id` should be an integer and the primary key.
- `surfer_id` should be an integer that references a surfer id in the "surfers" column.
- `board_name` should be a string capable of holding 255 characters.
- `color` should be a string capable of holding 255 characters.
- `length` should be an integer.

Pull a list of all of the surfers and surfboards already inside the database.

Push a new surfer and surfboard to the tables in the database.

Time’s Up! Let’s Review.

Questions?

The End