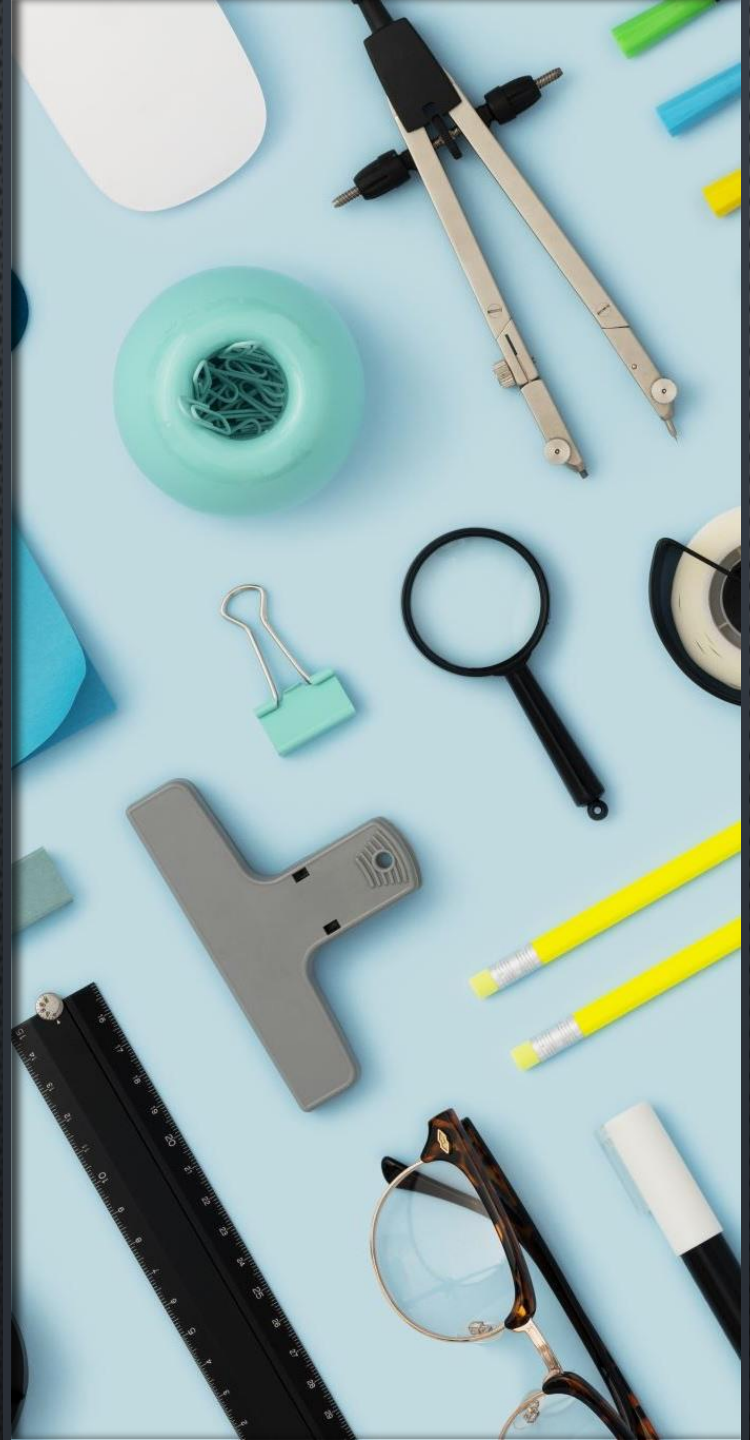
# WELCOME TO CSCI E-33A WEB PROGRAMMING W/ PYTHON AND JAVASCRIPT SECTION 3

TF FOR THIS SECTION: GLENN LANGDON

LANGDON@CS50.HARVARD.EDU

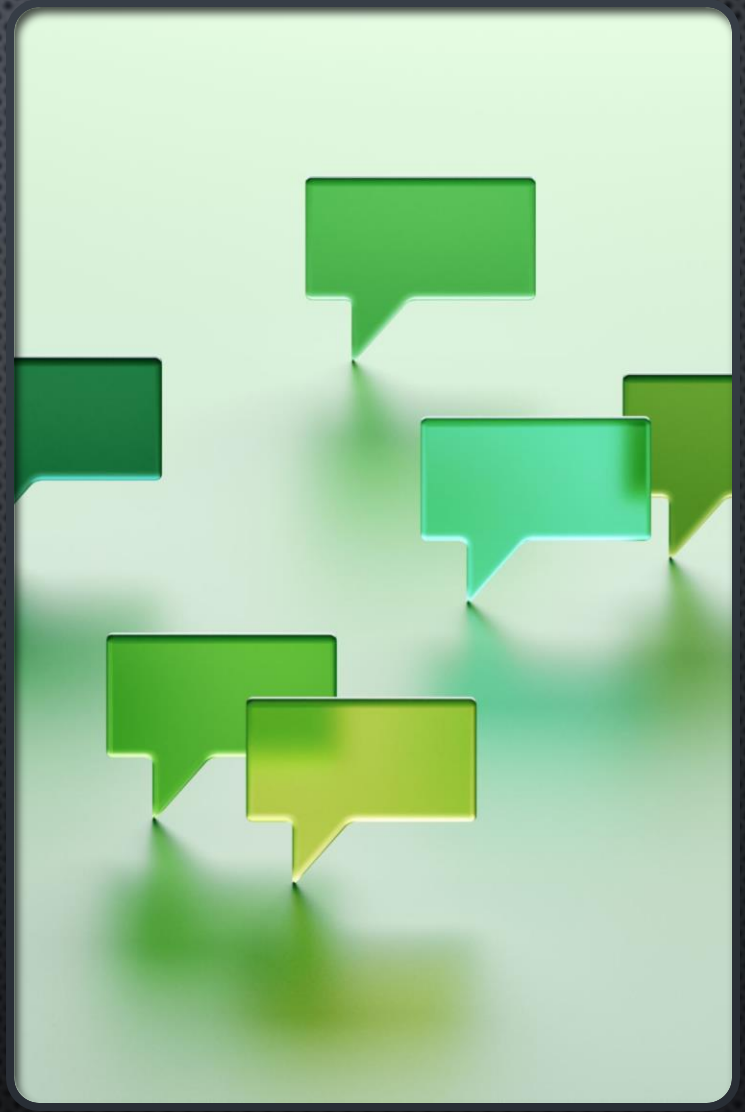


# PROGRAM FOR TODAY

- OOP IN PYTHON
- DJANGO MODELS
- QUERYING DJANGO MODELS
- EXPLORE A DJANGO APP W/ A MODEL

# QUESTIONS?

# IN THE CHAT IS A GITHUB REPO LINK WITH THE CODE WITH WHICH I'M WORKING IN THIS SECTION




# WHAT IS OOP?

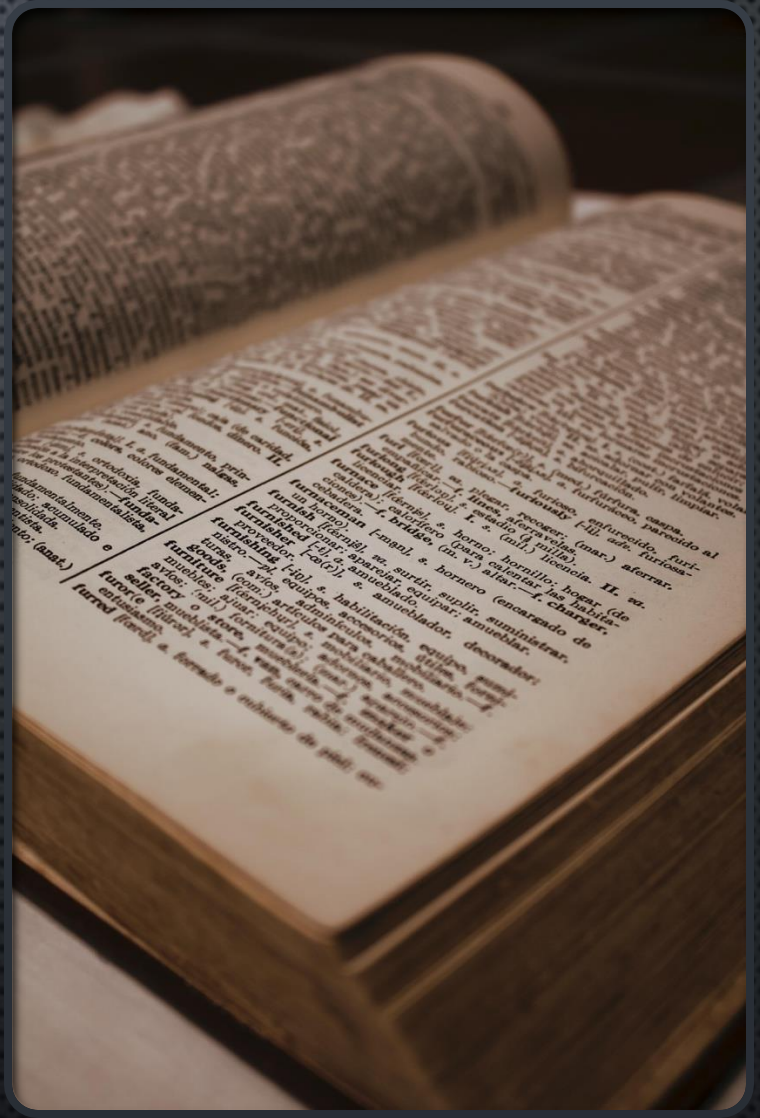
- A LANGUAGE THAT PROVIDES FEATURES, SUCH AS PROGRAMMER-DEFINED TYPES AND METHODS, IN WHICH DATA CAN BE ORGANIZED INTO CLASSES AND METHODS
- WHY? BECAUSE WE CAN BREAK DOWN COMPLEX PROBLEMS AS REAL-WORLD OBJECTS. IMPROVES CODE ORGANIZATION, REUSABILITY, AND SCALABILITY.

# How does this relate to web programming?

- In Django we'll model objects (people, products, things) as classes and manipulate instances of those classes with methods (functions associated with class instances).
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Abstraction
- OOP programming paradigm is transferrable to other modern languages like JavaScript, Ruby and Java

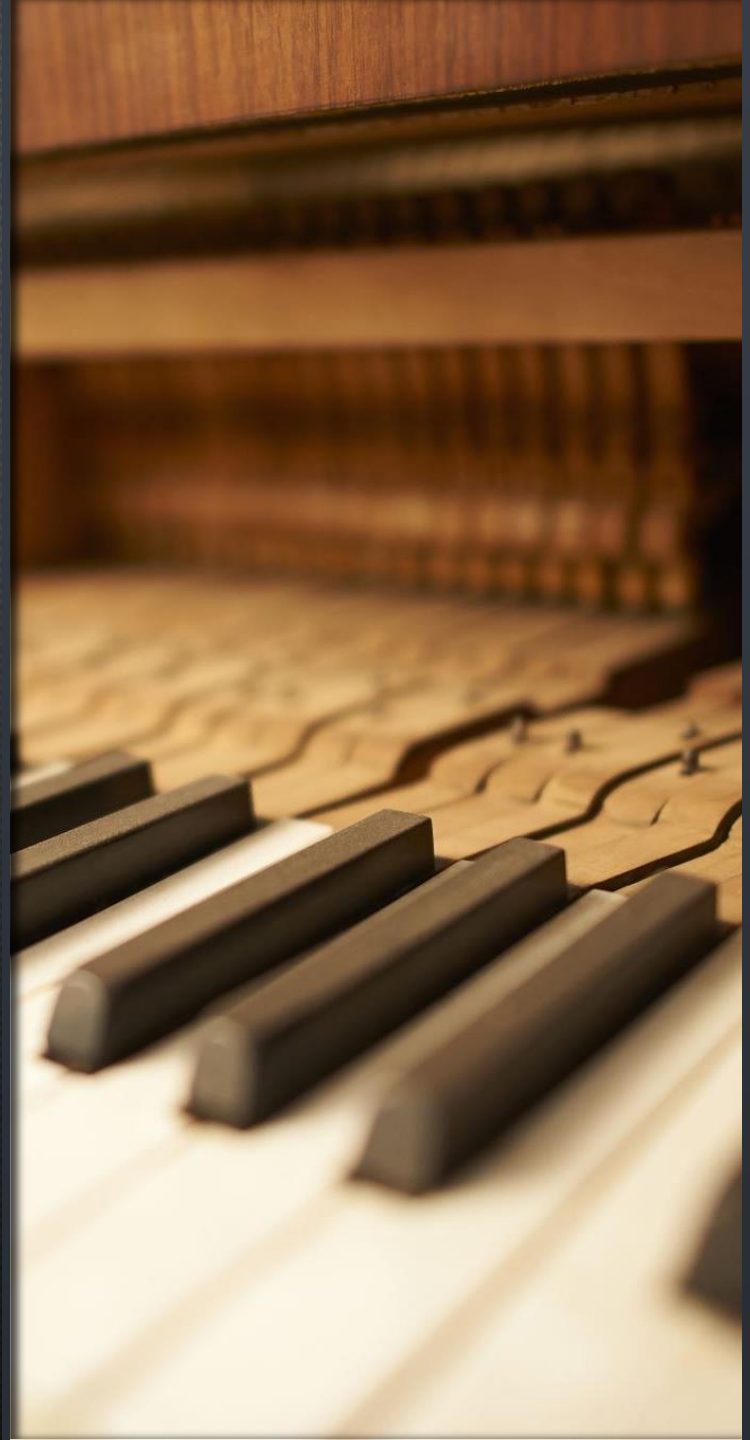
# BASIC TERMINOLOGY OF OOP IN PYTHON

- CLASS: TEMPLATE OR A BLUEPRINT FOR WHAT AN OBJECT WILL LOOK LIKE WHEN IT'S CREATED
- OBJECT: DATA, PLUS CODE THAT WORKS ON THE DATA OVER TIME
- INSTANTIATION: PROCESS OF CREATING AN OBJECT FROM A CLASS
- FUNCTION: INDEPENDENT OF A CLASS WHICH CAN BE USED ANYWHERE IN YOUR CODE AND YOU DON'T NEED AN OBJECT TO USE IT.
- METHOD: A FUNCTION DEFINED WITHIN A CLASS
- INSTANCE VARIABLE: ANY VARIABLE WHOSE VALUE MAY VARY FROM OBJECT TO OBJECT. IN A PYTHON CLASS, ALWAYS BE PRECEDED BY *SELF*.



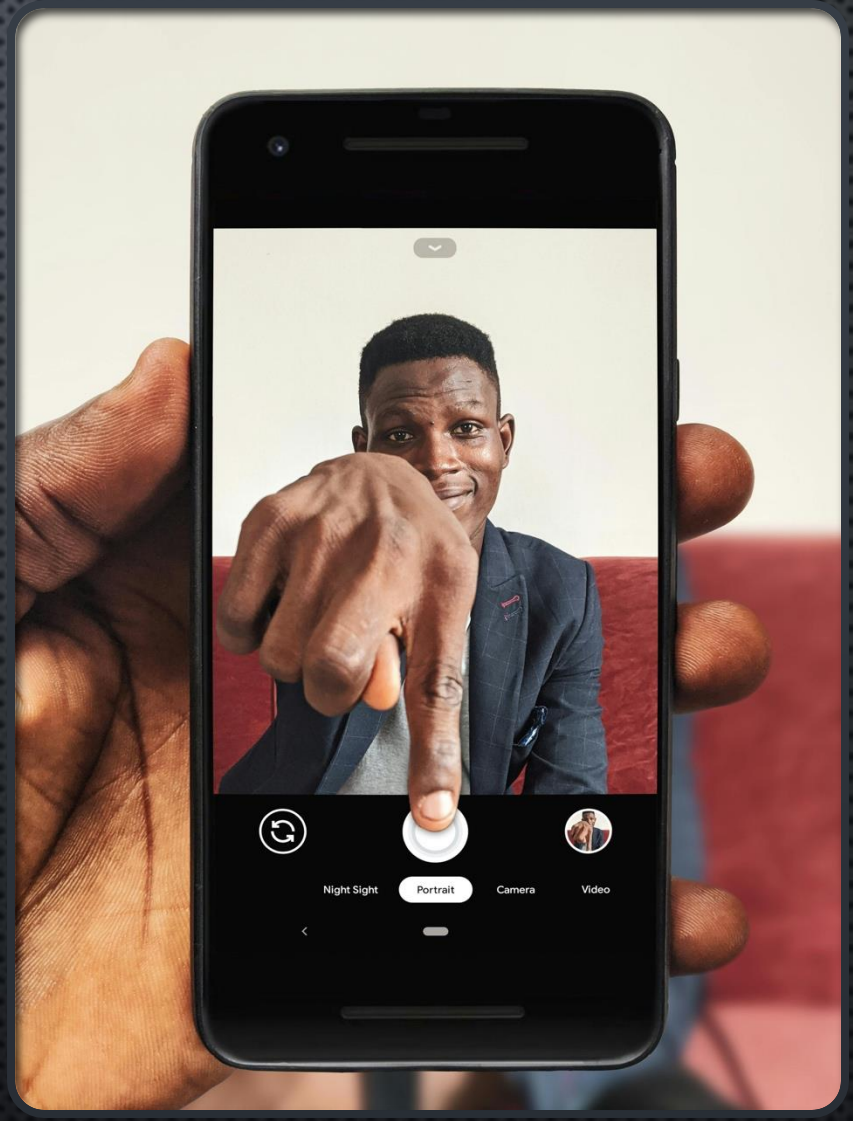
# HERE'S AN EXAMPLE

- I WANT TO RETRIEVE INFORMATION ABOUT PIANOS
  - FINISH
  - SIZE
  - PRICE
  - PLUS OTHER ATTRIBUTES
- I WANT SPECIFIC INFORMATION ON A CERTAIN BRAND OF PIANO, ITS DETAILS, AND ITS UNIQUE FEATURES
- SO HOW WOULD I DO THAT?
  - HINT: OOP IN PYTHON
- AND HERE'S HOW I'D DO IT…



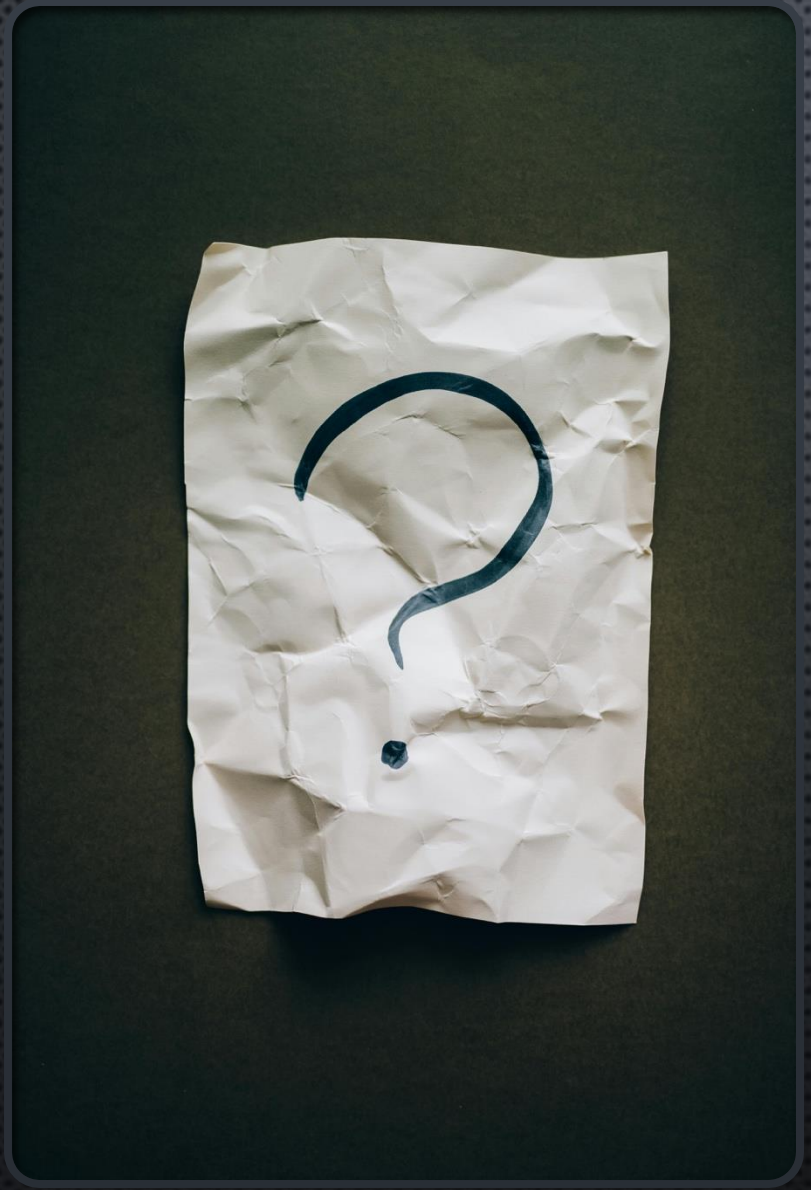
# A FEW WORDS ABOUT *SELF*?

- REFERENCE TO THE INSTANCE OF THE CLASS BEING CREATED (OR MODIFIED)
- ALLOWS EACH OBJECT TO HAVE ITS OWN ATTRIBUTES AND METHODS
- IN THE __INIT__ METHOD, 'SELF' IS USED TO BIND ATTRIBUTES TO THE SPECIFIC INSTANCE, ACCESSIBLE THROUGHOUT THE OBJECT'S LIFETIME.

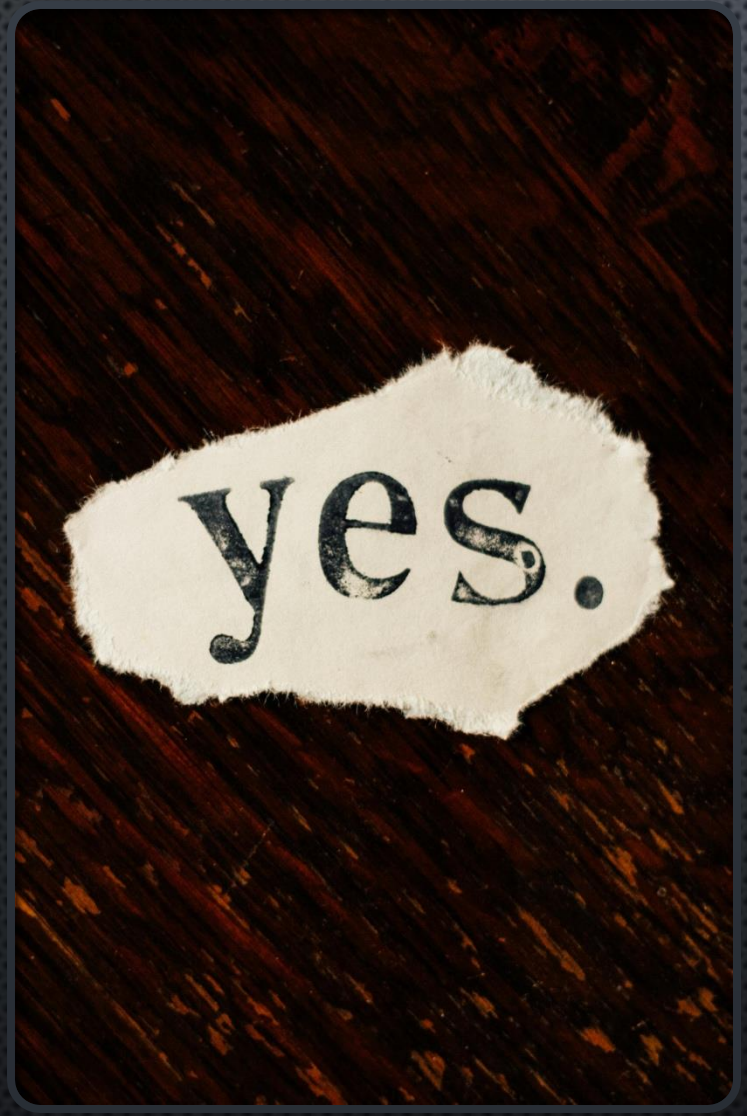


# Inheritance can be useful

- Inheritance in OOP is the ability to create a class that builds on (extends) an existing class.
- When creating large programs, you may want to create a general class, with other classes that offer more specific behavior.
- In this manner, it's not necessary to repeat the functionality of the super class, that functionality can be inherited. No need to repeat the same functionality in every class.
- One can refer to this relationship as a base class to a subclass



# BUT AM I GOING TO BE DOING OOP IN THIS COURSE?

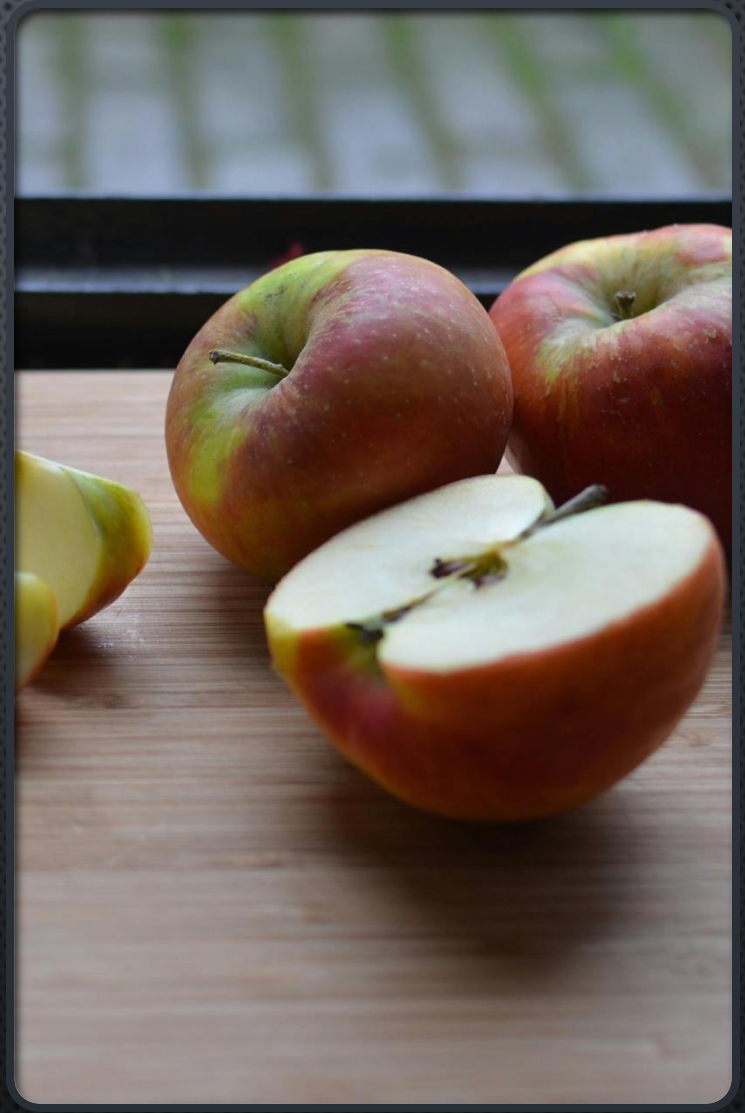


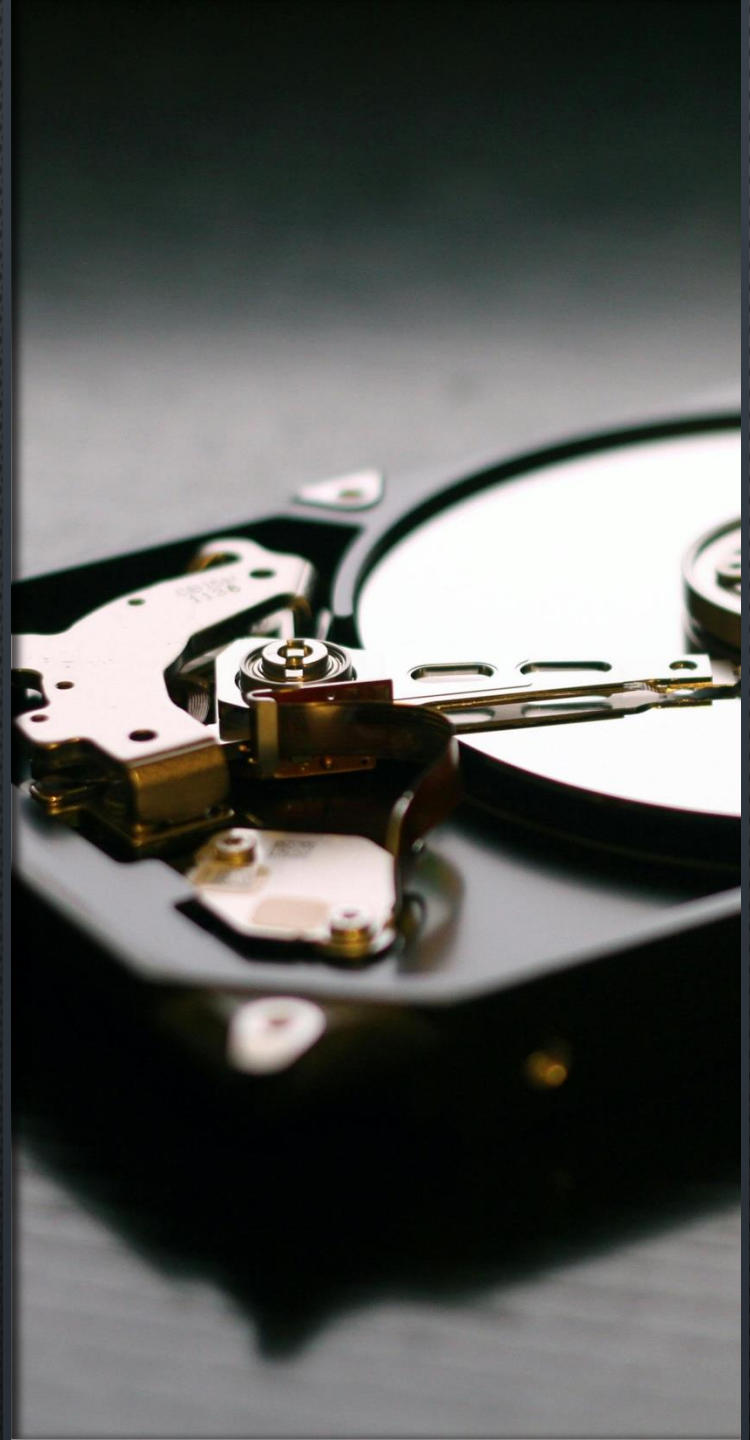

# IN A WORD…

- You'll see a lot of classes in Django
- Not the least of which will be found in classes modeled for a database where Django is going to be doing a lot of the heavy lifting for us with respect to SQL abstraction
- Understanding the basics of python OOP is important for a deeper understanding of how Django models and forms work.

# THIS WEEK WE GET TO THE CORE OF WEB APPLICATION DEVELOPMENT

The ability to manipulate and retrieve data over the web using HTTP (Hyper Text Transfer Protocol)



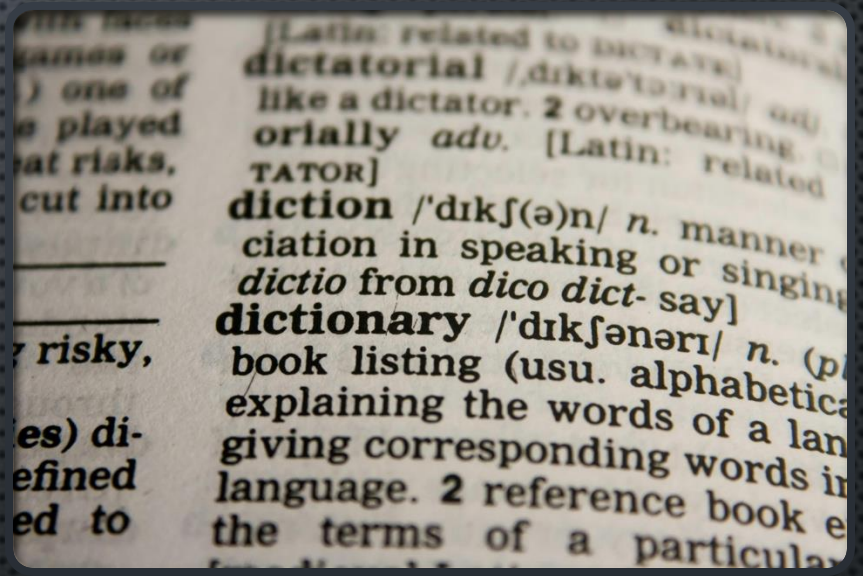


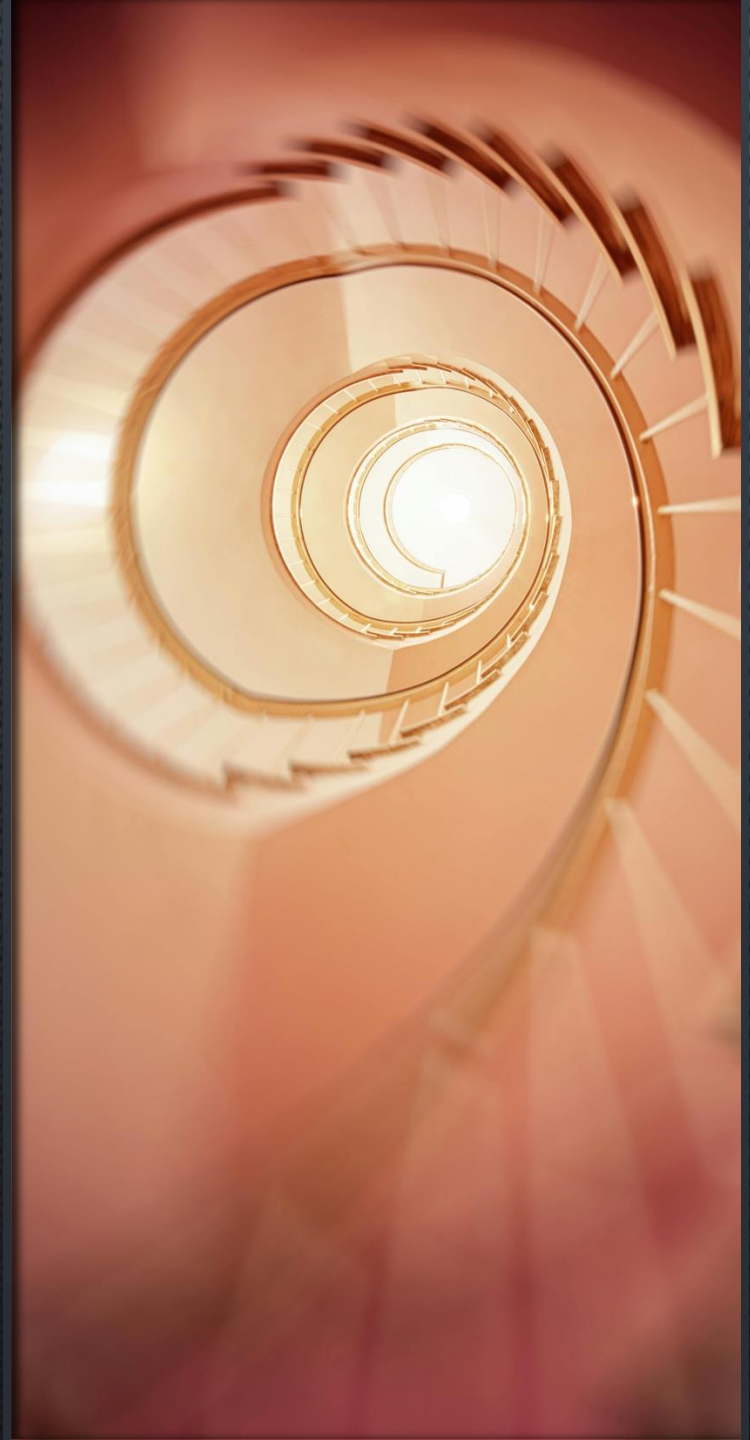
# DATA STORAGE

- In your previous project, we simulated a database and gave you methods to access that data
  - Util.list_entries()
  - Util.save_entry()
  - Util.get_entry()
- This week Brian introduced databases, and how we manipulate the data within them
- models are how we are going to store and manipulate data in Django and knowing how to build and manipulate them will be central to the rest of your work in Django.

# DATABASE TERMINOLOGY

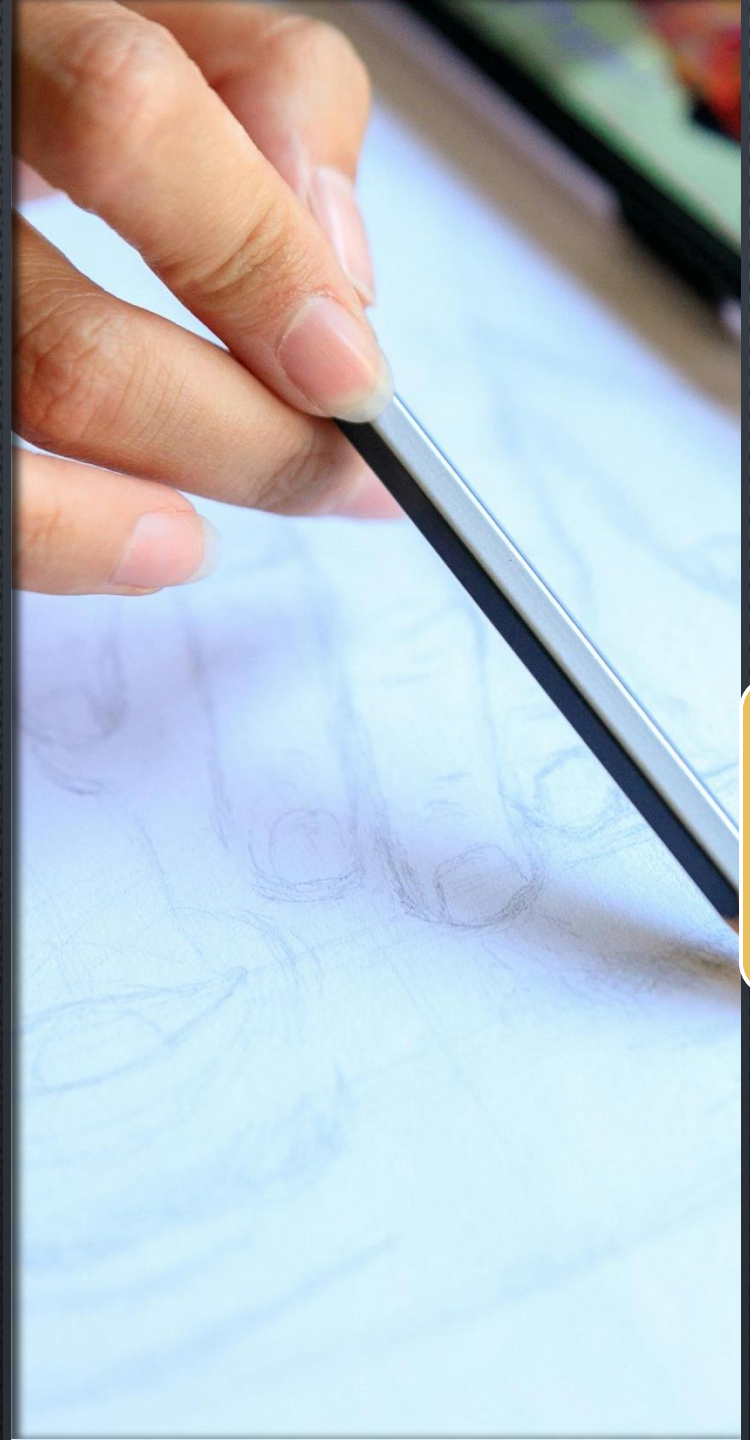
- DATABASE: A STRUCTURE THAT CAN STORE INFORMATION ABOUT MULTIPLE TYPES OF ENTITIES, THE ATTRIBUTES OF THE ENTITIES, AND THE RELATIONSHIPS AMONG THE ENTITIES.
- ENTITY: A PERSON, PLACE, OBJECT, EVENT, OR IDEA FOR WHICH YOU WANT TO STORE DATA (SIMILAR TO A PYTHON CLASS BUT DIFFERENT FUNCTION)
- WHY WE USE DATABASES:
  - REDUNDANCY
  - ACCESSING RELATED DATA
  - SECURITY FEATURES





# Django Models

- Django abstracts SQL tables into models through its Object-Relational Mapping (ORM) system.
- This allows you to interact with databases using Python classes instead of raw SQL queries.



# MORE PRECISELY:

Each model is a python class that subclasses django.db.models.Model

Each attribute of the model represents a column in a relation

Django gives us an API with which to query these models

# Django models as classes

- Each class represents a table in the database
- Each class attribute corresponds to a column in that table
- Every instance of the model corresponds to a row in the table
- Django maps the class to a table and attributes to the table's columns

```python
from django.db import models

class Piano(models.Model):
    brand = models.CharField(max_length=100)
    finish = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=10, decimal_places=2)
```

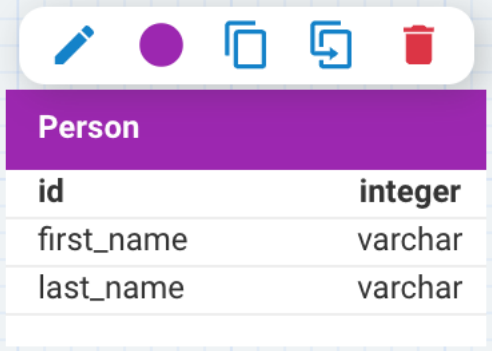
```python
from django.db import models


class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

# The above Person model creates this database table

```sql
CREATE TABLE myapp_person (
    "id" bigint NOT NULL PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

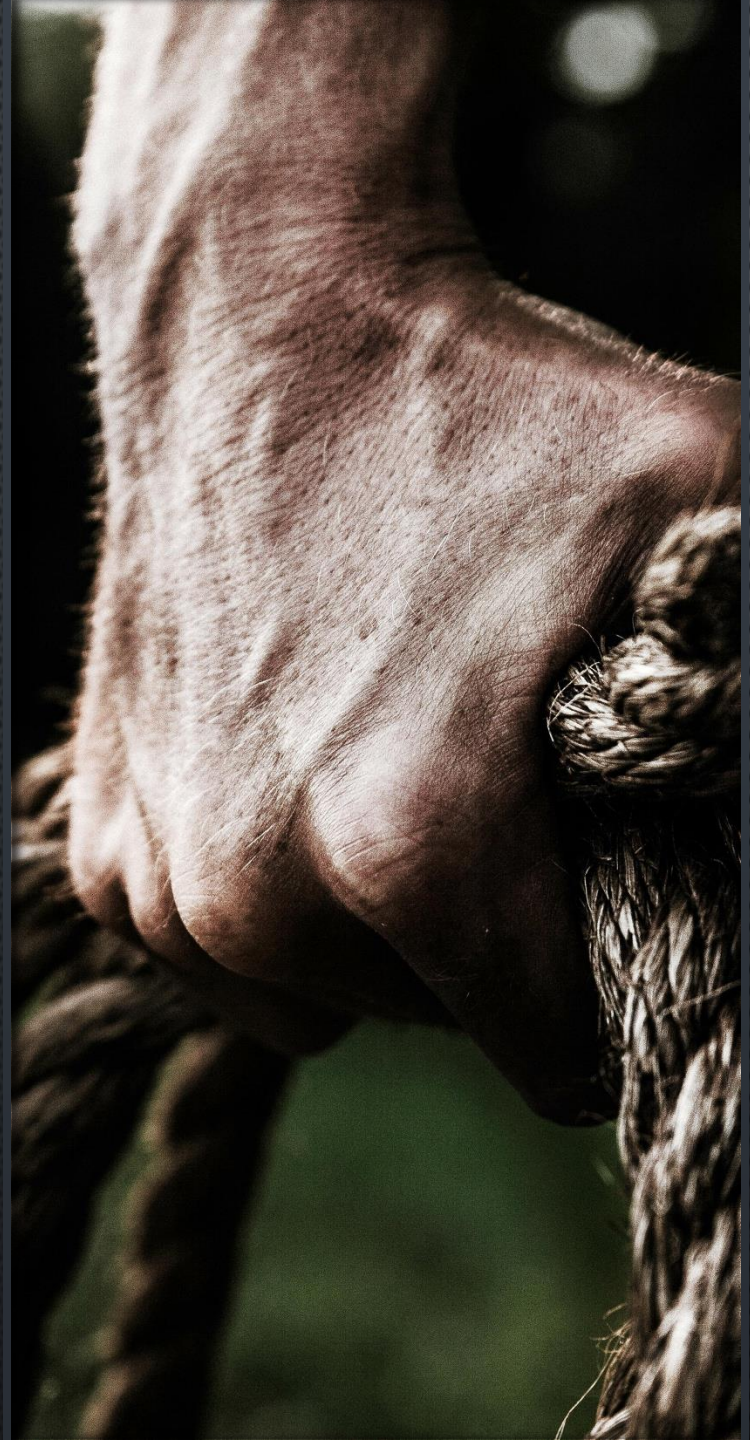
Markup

```
1
2  Person {
3      id integer pk increments
4      first_name varchar
5      last_name varchar
6  }
7
8
```



# Django's ORM abstracts away the complexities of SQL

- You define models using Python classes that inherit from 'django.db.models.Model
- Each class represents a table, where Django automatically generates the corresponding SQL. (Django does the heavy lifting!)
- Django ORM allows you to create, retrieve, update, and delete records using Python instead of SQL
- Django models handle relationships like one-to-many through foreign keys and related fields, abstracting the SQL JOIN operation
- Django's migration system automatically generates and applies database schema changes based on your model definitions.



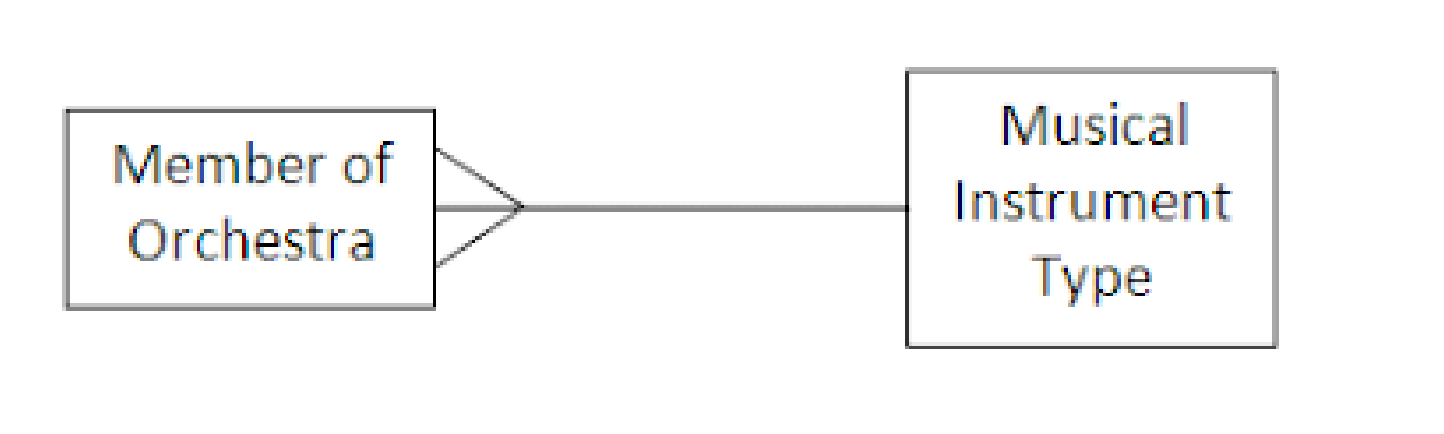
# The real power of relational databases resides in the relationships between tables

- Reducing data redundancy
  - Normalization
- Data integrity and consistency
  - Foreign keys ensure that a value in one table corresponds to a valid value in another.
- Enables complex queries through table joins

# MODEL RELATIONSHIPS

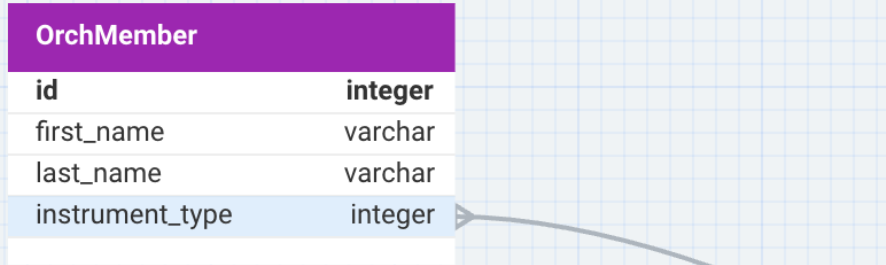
- One-to-One
- One-to-Many
- Many-to-Many (M2M)

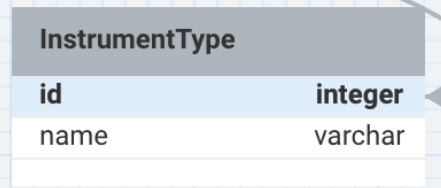
# ONE-TO-MANY




Markup

```
OrchMember {
    id integer pk increments
    first_name varchar
    last_name varchar
    instrument_type integer *> I
}

InstrumentType {
    id integer pk increments
    name varchar
}
```



# THE FOREIGN KEY ALWAYS GOES ON THE MANY SIDE OF THE RELATION

In other terms, it goes on the child of the parent table

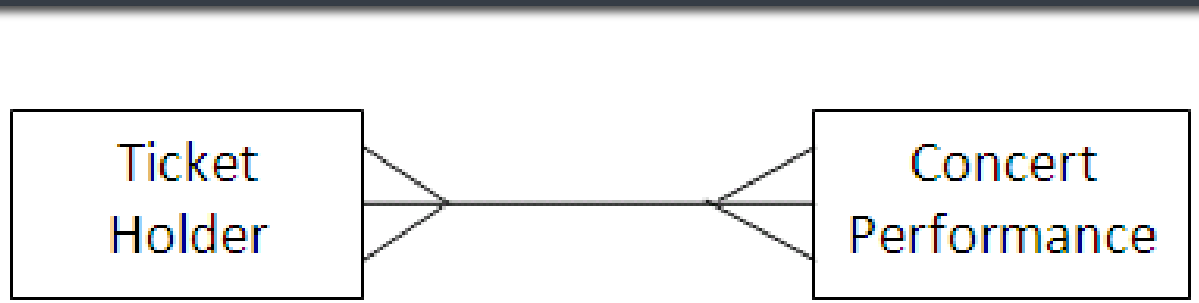
(A parent can have many children, but a child belongs to one parent)

# CAN YOU THINK OF OTHER ONE-TO-MANY RELATIONSHIPS?

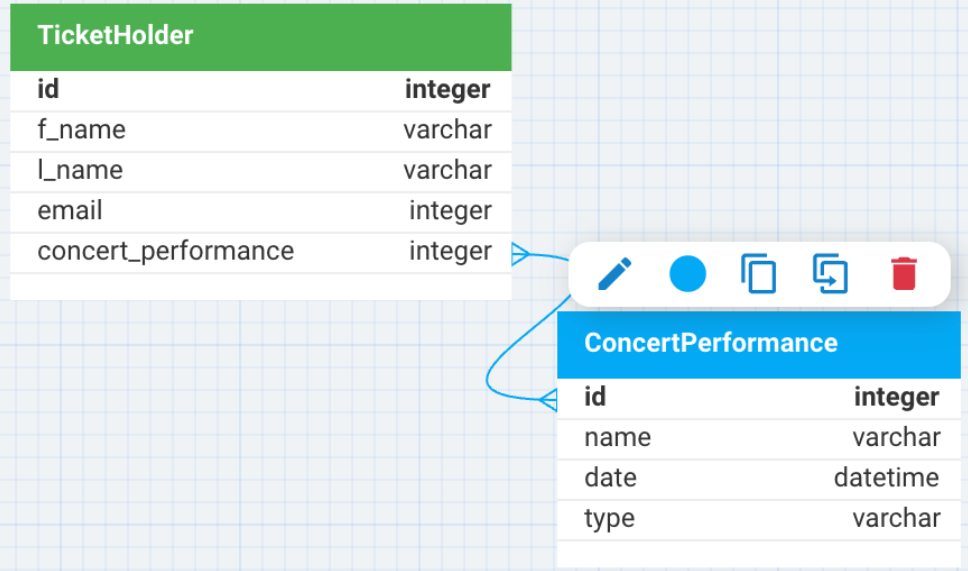
# ONE-TO-MANY RELATIONSHIP EXAMPLES

- A CUSTOMER REP HAS MANY CUSTOMERS, BUT A CUSTOMER BELONGS TO ONE CUSTOMER REP
- A CAR MANUFACTURE MAKES MANY BRANDS, BUT A BRAND BELONGS TO ONE MANUFACTURER.
- A BALLET COMPANY CAN HAVE MANY DANCERS, BUT A DANCER BELONGS TO ONE COMPANY. (USUALLY)
- A WEB SITE ARTICLE CAN HAVE MANY COMMENTS, BUT A COMMENT BELONGS TO ONE ARTICLE

# MANY-TO-MANY




```
13
14  TicketHolder {
15      id integer pk increments
16      f_name varchar
17      l_name varchar
18      email integer
19      concert_performance integer
20  }
21
22  ConcertPerformance {
23      id integer pk increments
24      name varchar
25      date datetime
26      type varchar
27  }
28
29
```



# CAN YOU THINK OF OTHER M2M RELATIONSHIPS?

# M2M EXAMPLES

- A PIZZA CAN HAVE MANY TOPPINGS, AND A TOPPING CAN GO ON MANY PIZZAS.
- A DOCTOR CAN HAVE MANY PATIENTS, AND A PATIENT CAN HAVE MANY DOCTORS.
- A STUDENT CAN HAVE MANY CLASSES, AND A CLASS CAN CONTAIN MANY STUDENTS.

# Let's look closely at the models.py file

Student
Advisor
Course

- Considering the 3 basic relationship paradigms
  - One-to-one
  - One-to-many
  - Many-to-many

- What is the logical relationship between students and advisors?
- What is the logical relationship between students and courses?

# HOW DO WE QUERY  IN DJANGO?

LET'S LOOK AT OUR STUDENT APP, OPEN THE DJANGO SHELL AND MANIPULATE THE MODEL.

# HERE'S AN EXAMPLE OF AN APP FOR A SCHOOL DATABASE WITH STUDENTS, ADVISORS AND COURSES

# THAT'S IT FOR TONIGHT!

HAVE A GREAT WEEK!