Installing Python

Is Python 3 Installed?

- **Inside your "terminal" for Mac, or "command prompt" for windows.**
  - **Type "python3 --version" for Mac or "python --version" for windows**

```
ericroby@Erics-MacBook-Pro ~% python3 --version

Python 3.11.2
```

Could be a different version for you, requirement for FastAPI is version 3.7

© CODINGWITHROBY

Python Integrated Development Environment

WHAT IS AN INTEGRATED DEVELOPMENT ENVIRONMENT

© CODINGWITHROBY

What is an IDE?

- IDEs are simply a source code editor. This means an IDE will help and assist in writing software!
- Many of them have terminals and other useful build automation tools embedded within.
- IDEs have debugger tools that allow you to dive deep within the code to find bugs and create solutions

© CODINGWITHROBY

Why do developers use IDEs?

- IDEs allow developers to begin work quickly and efficiently due to all the tools that come out of the box
- IDEs also parse all code, and helps find bugs before they are committed by human error
- The GUIs of IDEs is created with one goal in mind, help developers be efficient.

© CODINGWITHROBY

Python Integrated Development Environment

WHAT IS AN INTEGRATED DEVELOPMENT ENVIRONMENT

© CODINGWITHROBY

What is an IDE?

- IDEs are simply a source code editor. This means an IDE will help and assist in writing software!
- Many of them have terminals and other useful build automation tools embedded within.
- IDEs have debugger tools that allow you to dive deep within the code to find bugs and create solutions

© CODINGWITHROBY

Why do developers use IDEs?

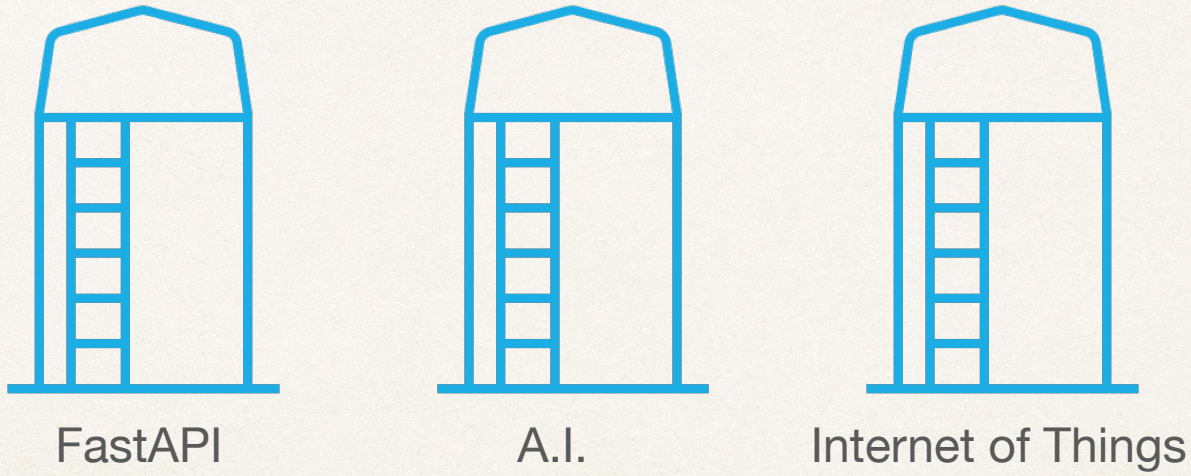
- IDEs allow developers to begin work quickly and efficiently due to all the tools that come out of the box
- IDEs also parse all code, and helps find bugs before they are committed by human error
- The GUIs of IDEs is created with one goal in mind, help developers be efficient.

© CODINGWITHROBY

Python Virtual Environments

Python Virtual Environments

- **A virtual environment is a Python environment that is isolated from those in other Python environments.**



© CODINGWITHROBY

How to Install Dependencies

- Pip is the Python package manager.
  - Pip is used to install and update packages.
  - You will want to make sure you have the latest version of pip installed.

```
ericroby@Erics-MacBook-Pro ~% python3 -m pip --version
 pip 22.3.1
```


© CODINGWITHROBY

Setting Up Virtual Environment

- I started by creating a brand-new folder / directory called “FastAPI”
- Within our pip we will be checking what dependencies we already have installed.
- Python already has an included dependency for virtual environments called venv.
- Creating a new FastAPI environment as a virtual environment
- Activating our virtual environment to install our dependencies in

© CODINGWITHROBY

How To Get The Most Out Of This Course

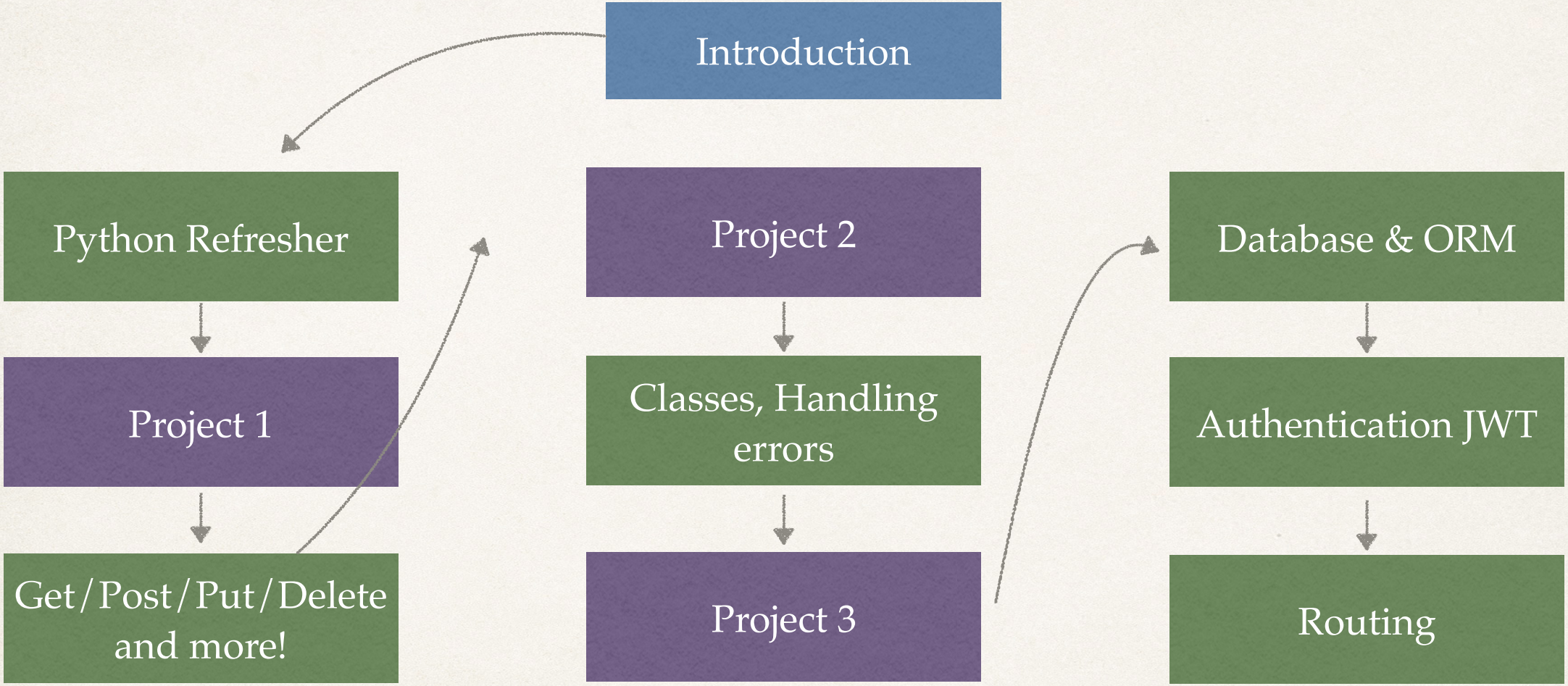
How To Get The Most Out Of This Course

Watch all the videos	Control the speed of videos to be your pace
Take the exercises	Pause videos to take exercises before seeing solutions
Search & Practice	Advance on your own and search for additional information
Ask Questions	Ask in Q&A section!

© CODINGWITHROBY

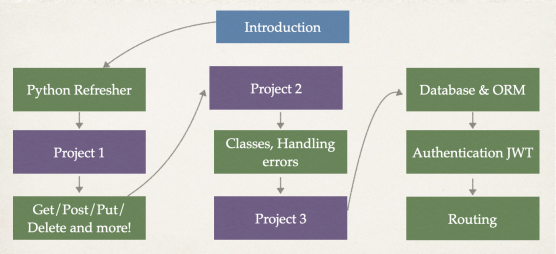
Course Content

Course Content!



© CODINGWITHROBY

Course Content!



Project 3.5

DB & Data Migration

Project 4

Unit & Integration Testing

Project 5

Full Stack Development

Git / GitHub Setup

Deployment

© CODINGWITHROBY

FASTAPI OVERVIEW

© CODINGWITHROBY

WHAT IS FASTAPI?

- FastAPI is a Python web-framework for building modern APIs
  - Fast (Performance)
  - Fast (Development)
- Key Notes:
  - Few Bugs
  - Quick & Easy
  - Robust
  - Standards

© CODINGWITHROBY

WHAT DOES THIS ALL MEAN FOR YOU?

- FastAPI is a web-framework for building modern RESTful APIs

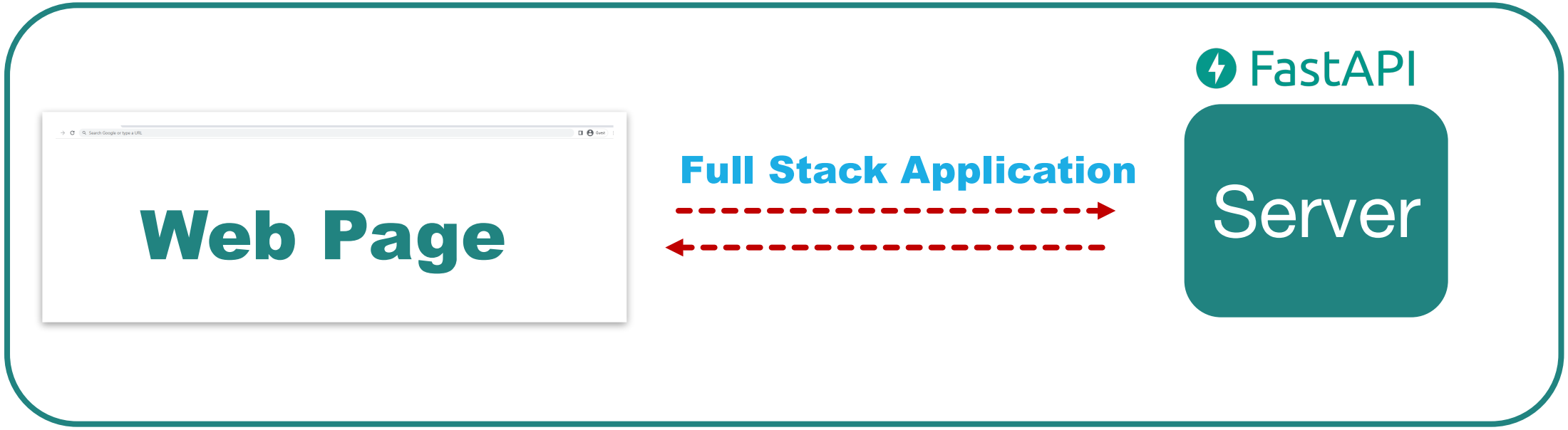
FastAPI

Official documentation

https://fastapi.tiangolo.com/

© CODINGWITHROBY

WHERE DOES FASTAPI FIT WITHIN AN APPLICATION ARCHITECTURE?



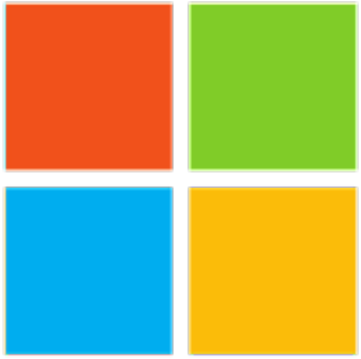
Handles all business logic for the application

© CODINGWITHROBY

WHO HAS USED FASTAPI?

Netflix

Uber



Microsoft

© CODINGWITHROBY

CAN’T I JUST WRITE EVERYTHING MYSELF?

- FAQ: Why do I need a web framework?
  - You may be able to write everything yourself, but why reinvent the wheel?
  - Web-frameworks allow a simplified way for rapid development.
  - Includes many years of development, which allows you to have a secure and fast application! ☺

© CODINGWITHROBY

Books Project

What will we be creating?

- Creating and enhancing books to learn the basics of FastAPI

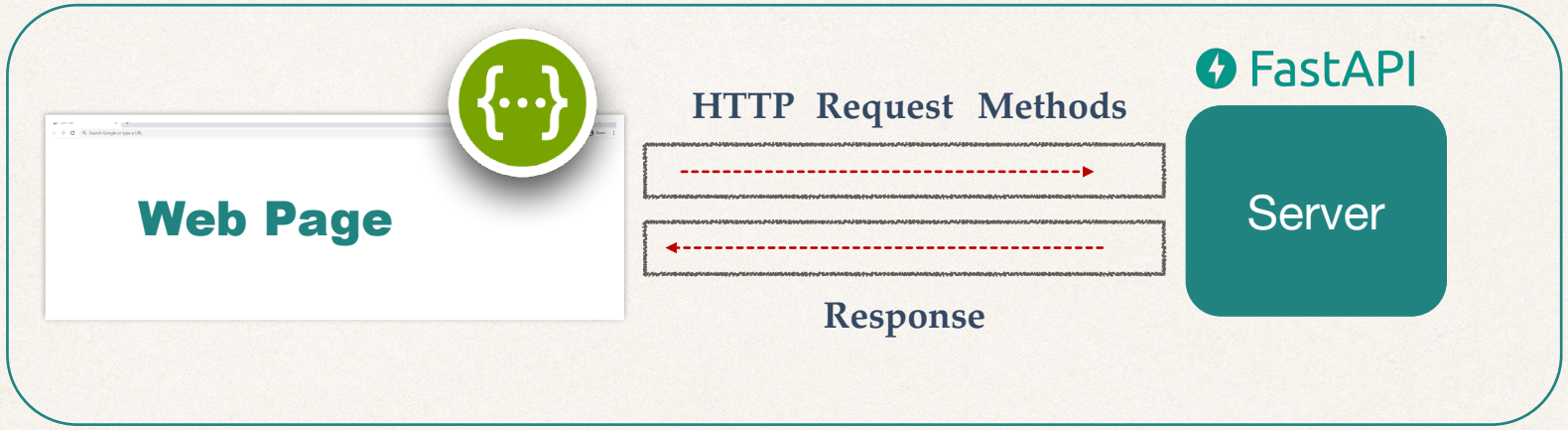
```
BOOKS = {
        {'title': 'Title One', 'author': 'Author One', 'category': 'science'},
        {'title': 'Title Two', 'author': 'Author Two', 'category': 'science'},
        {'title': 'Title Three', 'author': 'Author Three', 'category':
        'history'},
        {'title': 'Title Four', 'author': 'Author Four', 'category': 'math'},
        {'title': 'Title Five', 'author': 'Author Five', 'category': 'math'},
```

CRUD Operations

Create | Read | Update | Delete

© CODINGWITHROBY

Request and Response



CRUD Operations

- Create
- Read
- Update
- Delete

© CODINGWITHROBY

HTTP Request Methods

CRUD		HTTP Request Methods
Create	→	POST
Read	→	GET
Update	→	PUT
Delete	→	DELETE

© CODINGWITHROBY

GET HTTP Request Method

Creating a FastAPI application

File: books.py

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/api-endpoint")
async def first_api():
        return {'message': 'Hello Eric!'}
```

Lets dive into the first function

© CODINGWITHROBY

Dive in

Read

Get

API Endpoint:

```
@app.get("/api-endpoint")
async def first_api():
        return {'message': 'Hello Eric!'}
```

URL : 127.0.0.1:8000/api-endpoint

Response:

```
{
        "message": "Hello Eric!"
}
```


© CODINGWITHROBY

Start FastAPI Application

Read

Get

API Endpoint:

```
@app.get("/api-endpoint")
async def first_api():
        return {'message': 'Hello Eric!'}
```

Run FastAPI application:

```
ericroby@Erics-MacBook-Pro ~% uvicorn books:app --reload
```

URL : 127.0.0.1:8000

© CODINGWITHROBY

Change Endpoint for Books

Read

Get

```
BOOKS = {
     {'title': 'Title One', 'author': 'Author One', 'category': 'science'},
     {'title': 'Title Two', 'author': 'Author Two', 'category': 'science'},
     {'title': 'Title Three', 'author': 'Author Three', 'category': 'history'},
     {'title': 'Title Four', 'author': 'Author Four', 'category': 'math'},
     {'title': 'Title Five', 'author': 'Author Five', 'category': 'math'},
}
```

```
return BOOKS
```

URL : 127.0.0.1:8000/api/books

© CODINGWITHROBY

Path Parameters

What are Path Parameters

- Path Parameters are request parameters that have been attached to the URL
- Path Parameters are usually defined as a way to find information based on location
- Think of a computer file system:
  - You can identify the specific resources based on the file you are in

```
/Users/codingwithroby/Documents/python/fastapi/section1
```


© CODINGWITHROBY

Path Parameters

REQUEST:

URL : 127.0.0.1:8000/books

Read

Get

```
@app.get("/books")
async def read_all_books():
        return BOOKS
```


© CODINGWITHROBY

Path Parameters

REQUEST:

URL : 127.0.0.1:8000/books/book_one

Read

Get

```
@app.get("/books/{dynamic_param}")
async def read_all_books(dynamic_param):
        return {'dynamic_param': dynamic_param}
```

RESPONSE:

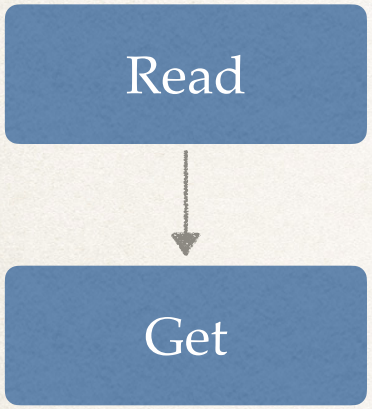
```
{
        "dynamic_param": "book_one"
}
```


© CODINGWITHROBY

Order Matters with Path Parameters

REQUEST:

URL : 127.0.0.1:8000/books/mybook




```
@app.get("/books/mybook")
async def read_all_books():
        return {'book_title': 'My Favorite Book'}

@app.get("/books/{dynamic_param}")
async def read_all_books(dynamic_param):
        return {'dynamic_param': dynamic_param}
```


© CODINGWITHROBY

Path Parameters

```
BOOKS = {
    {'title': 'Title One', 'author': 'Author One', 'category': 'science'},
    {'title': 'Title Two', 'author': 'Author Two', 'category': 'science'},
    {'title': 'Title Three', 'author': 'Author Three', 'category': 'history'},
    {'title': 'Title Four', 'author': 'Author Four', 'category': 'math'},
    {'title': 'Title Five', 'author': 'Author Five', 'category': 'math'},
}
```

Read

Get

REQUEST:

URL : 127.0.0.1:8000/books/title%20four = title four

```
@app.get("/books/{book_title}")
async def read_book(book_title: str):
        for book in BOOKS:
                if book.get('title').casefold() ==
                book_title.casefold():
```

RESPONSE:

```
{
        "title": "Title Four",
        "author": "Author Four",
        "category": "math"
}
```


© CODINGWITHROBY

Query Parameters

What are Query Parameters

- **Query Parameters are request parameters that have been attached after a "?"**
- **Query Parameters have name=value pairs**
- **Example:**
  - 127.0.0.1:8000/books/?category=math

© CODINGWITHROBY

Query Parameters

REQUEST:

URL : 127.0.0.1:8000/books/?category=science

```
@app.get("/books/")
async def read_category_by_query(category: str):
        books_to_return = []
        for book in BOOKS:
            if book.get('category').casefold() == category.casefold():
                books_to_return.append(book)

        return books_to_return
```


© CODINGWITHROBY

Query Parameters

REQUEST:

URL : 127.0.0.1:8000/books/author%20four/?category=science

```
@app.get("/books/{book_author}/")
async def read_category_by_query(book_author: str, category: str):
        books_to_return = []
        for book in BOOKS:
            if book.get('author').casefold() == book_author.casefold() and
                    book.get('category').casefold() == category.casefold():
                books_to_return.append(book)

        return books_to_return
```


© CODINGWITHROBY

POST HTTP Request Method

What is the POST Request Method

- **Used to create data**
- **POST can have a body that has additional information that GET does not have**
- **Example of Body:**

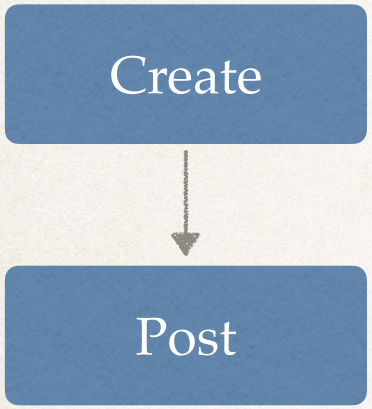
```
{“title”:”Title Seven”, “author”:”Author Two”, “category”:
```


© CODINGWITHROBY

POST Request Method

REQUEST:

URL : 127.0.0.1:8000/books/create_book




```
@app.post("/books/create_book")
async def create_book(new_book=Body()):
        BOOKS.append(new_book)
```


© CODINGWITHROBY

PUT HTTP Request Method

What is the PUT Request Method

- **Used to update data**
- **PUT can have a body that has additional information (like POST) that GET does not have**
- **Example of Body:**

```
{“title”:”Title Six”, “author”:”Author Two”, “category”:
```


© CODINGWITHROBY

PUT Request Method

Update

Put

REQUEST:

URL : 127.0.0.1:8000/books/update_book

```
@app.put("/books/update_book")
async def update_book(updated_book=Body()):
        for i in range(len(BOOKS)):
            if BOOKS[i].get('title').casefold() == updated_book.get('title').casefold():
                BOOKS[i] = updated_book
```


© CODINGWITHROBY

DELETE HTTP Request Method

What is the DELETE Request Method

- **Used to delete data**

Delete

Delete

REQUEST:

URL : 127.0.0.1:8000/books/delete_book/{book_title}

```
@app.delete("/books/delete_book/{book_title}")
async def delete_book(book_title: str):
        for i in range(len(BOOKS)):
            if BOOKS[i].get('title').casefold() == book_title.casefold():
               BOOKS.pop(i)
                    break
```


© CODINGWITHROBY

Books 2 Project

Project 2

- Project two will still be focused on creating Book API Endpoints
- Continued Education includes:
  - GET, POST, PUT, DELETE Request Methods
- New Information will include:
  - Data Validation, Exception Handling, Status Codes, Swagger Configuration, Python Request Objects

© CODINGWITHROBY

Creating a new books project

- We will create a new class called Book
- We will be using these books throughout this project:

```
class Book:
    id: int
    title: str
    author: str
    description: str
    rating: int

def __init__ (self, id, title, author, description, rating):
    self.id = id
    self.title = title
    self.author = author
    self.description = description
    self.rating = rating
```


© CODINGWITHROBY

HTTP Request Methods

CRUD		HTTP Request Methods
Create	→	POST
Read	→	GET
Update	→	PUT
Delete	→	DELETE

© CODINGWITHROBY

Pydantics

What is Pydantics

- Python library that is used for data modeling, data parsing and has efficient error handling.
- Pydantics is commonly used as a resource for data validation and how to handle data coming to our FastAPI application.

© CODINGWITHROBY

We will be implementing Pydantics

- Create a different request model for data validation
- Field data validation on each variable / element

```
class BookRequest(BaseModel):
    id: int
    title: str = Field(min_length=3)
    author: str = Field(min_length=1)
    description: str = Field(min_length=1, max_length=100)
    rating: int = Field(gt=0, lt=5)
```


© CODINGWITHROBY

BookRequest and Book Conversion

- We will convert the Pydantics Request into a Book
- Here is an example within an upcoming Post Request Method

```
@app.post("/create-book")
async def create_book(book_request: BookRequest):
        new_book = Book(**book_request.dict())
        BOOKS.append(new_book)
```

** operator will pass the key/value from BookRequest() into the Book() constructor

© CODINGWITHROBY

Status Codes

What are Status Codes?

- An HTTP Status Code is used to help the Client (the user or system submitting data to the server) to understand what happened on the server side application.
- Status Codes are international standards on how a Client/Server should handle the result of a request.
- It allows everyone who sends a request to know if their submission was successful or not.

© CODINGWITHROBY

Status Codes:

1xx	Information Response: Request Processing.
2xx	Success: Request Successfully complete
3xx	Redirection: Further action must be complete
4xx	Client Errors: An error was caused by the client.
5xx	Server Errors: An error occurred on the server.

© CODINGWITHROBY

2xx Successful Status Codes:

200: OK → Standard Response for a Successful Request. Commonly used for successful Get requests when data is being returned.

201: → The request has been successful, creating a new resource. Used when a POST creates an entity.

204: No → The request has been successful, did not create an entity nor return anything. Commonly used with PUT requests.

© CODINGWITHROBY

4xx Client Errors Status Codes:

400: Bad	Cannot process request due to client error. Commonly used for invalid request methods.
401: Unauthorized	Client does not have valid authentication for target resource
404: Not	The clients requested resource can not be found
422: Unprocessable Entity	Semantic Errors in Client Request

© CODINGWITHROBY

5xx Server Status Codes:

500: Internal Server Error → Generic Error Message, when an unexpected issue on the server happened.

© CODINGWITHROBY

Project 3

Project 3

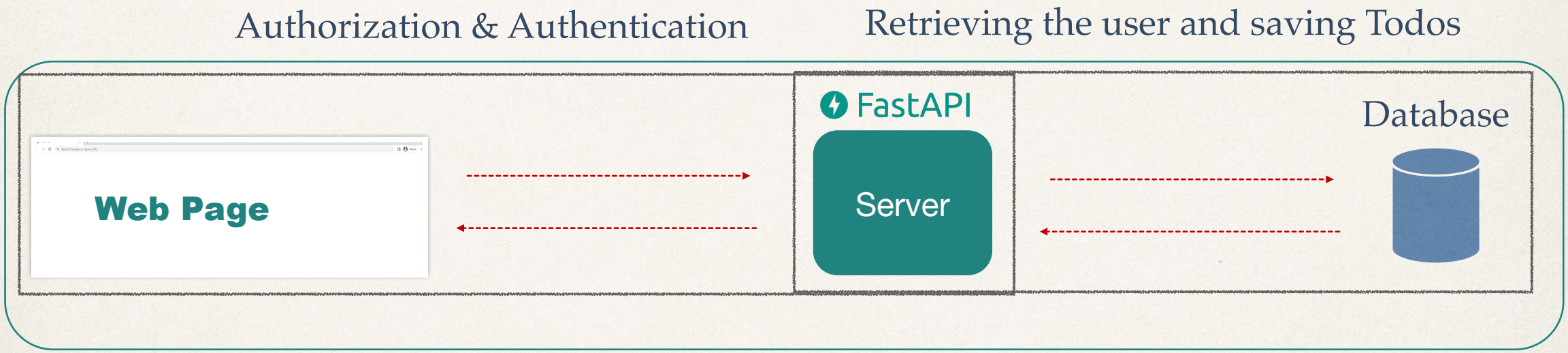
- Project three we will be switching our focus to TODOS instead of BOOKS
- New Information will include:
  - Full SQL Database
  - Authentication
  - Authorization
  - Hashing Passwords

Continued learning from other projects

© CODINGWITHROBY

Creating a Todo Table

- We will create new Todo Table Models for our application
- We will be using these Todos to save records throughout this project



© CODINGWITHROBY

SQL Database Introduction

What is a database?

- Organized collection of structured information of **data**, which is stored in a computer system.
- The data can be easily accessed
- The data can be m
- The data can be co
- Many databases use a structured query language (SQL) to modify and write data

What is Data?

© CODINGWITHROBY

What is a database?

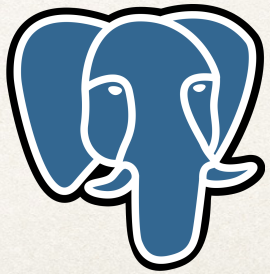
- Data can be related to just about any object.
- For example, a user on an application may have:
  - Name
  - Age
  - Email
  - Password

All of this is Data

© CODINGWITHROBY

What is a database?

- A database is a collection of data
- Since data, on its own, is just data. A database allows management of this data
- Databases are organized in how data can be retrieved, stored and modified
- There are many types of Database Management Systems (DBMS)



© CODINGWITHROBY

What is a SQL?

- Pronounced either as S-Q-L or “See Quel”
- Standard language for dealing with relational databases
- SQL can be used to do different things with database records:
  - Create
  - Read
  - Update
  - Delete

© CODINGWITHROBY

What is Alembic?

What is Alembic?

- Lightweight database migration tool for when using SQLAlchemy
- Migration tools allow us to plan, transfer and upgrade resources within databases
- Alembic allows you to change a SQLAlchemy database table after it has been created
- Currently SQLAlchemy will only create new database tables for us, not enhance any.

© CODINGWITHROBY

How does Alembic work?

- Alembic provides the creation and invocation of change management scripts
- This allows you to be able to create migration environments and be able to change data how you like
- In this section we will learn how we can use Alembic for our project

© CODINGWITHROBY

How does Alembic work?

- We already have some data within our database
- Let's take a look at our current users table
- Create a new column on our data that already exists

ID	email	first_name	.....	phone_number
1	codingwithroby@email.com	Eric	<Other Columns>	(1) 111-111-1111

© CODINGWITHROBY

How does Alembic work?

- Alembic is a powerful migration tool that allows us to modify our database schemes
- As our application evolves, our database will need to evolve as well
- Alembic helps us be able to keep modifying our database to keep up with rapid development requirements
- It works on tables that already have data. This allows us to be able to continually create additional content that works within our application!

© CODINGWITHROBY

Alembic?

What is Alembic?

- Lightweight database migration tool for when using SQLAlchemy
- We need to install Alembic into our project

```
ericroby@FastAPI-Project ~%     pip install alembic
```


© CODINGWITHROBY

What is Alembic?

Alembic Command	Details
alembic init <folder name>	Initializes a new, generic environment

© CODINGWITHROBY

What is Alembic?

Alembic Command	Details
alembic init <folder name>	Initializes a new, generic environment
alembic revision -m <message>	Creates a new revision of the environment

© CODINGWITHROBY

What is Alembic?

Alembic Command	Details
alembic init <folder name>	Initializes a new, generic environment
alembic revision -m <message>	Creates a new revision of the environment
alembic upgrade <revision #>	Run our upgrade migration to our database

© CODINGWITHROBY

What is Alembic?

Alembic Command	Details
alembic init <folder name>	Initializes a new, generic environment
alembic revision -m <message>	Creates a new revision of the environment
alembic upgrade <revision #>	Run our upgrade migration to our database
alembic downgrade -1	Run our downgrade migration to our database

© CODINGWITHROBY

How does Alembic work?

- After we initialize our project with alembic, two new items will appear in our directory
    - alembic.ini
    - alembic directory
- These are created automatically by alembic so we can upgrade, downgrade and keep data integrity of our application

© CODINGWITHROBY

Alembic.ini file

- File that alembic looks for when invoked
- Contains a bunch of configuration information for Alembic that we are able to change to match our project

© CODINGWITHROBY

Alembic Directory

- Has all environmental properties for alembic
- Holds all revisions of your application
- Where you can call the migrations for upgrading
- Where you can call the migrations for downgrading

© CODINGWITHROBY

Alembic Revisions

Alembic Revision?

- Alembic revision is how we create a new alembic file where we can add some type of database upgrade
- When we run:

```
ericroby@FastAPI-Project ~%
alembic revision -m “create phone number col on users table”
```

- Creates a new file where we can write the upgrade code
- Each new revision will have a Revision Id

© CODINGWITHROBY

Alembic Upgrade?

- Alembic upgrade is how we actually run the migration

```
def upgrade() -> None:
    op.add_column('users', sa.Column('phone_number', sa.String(),
    nullable=True))
```

- Enhances our database to now have a new column within our users tables called 'phone_number'
- Previous data within database does not change

© CODINGWITHROBY

Alembic Upgrade?

- To run the the upgrade migration:

```
ericroby@FastAPI-Project ~%
alembic upgrade <revision id>
```

- This will successfully implement the change within the upgrade functionality

© CODINGWITHROBY

Alembic Downgrade?

- Alembic downgrade is how we revert a migration

```
def downgrade() -> None:
    op.drop_column('users', 'phone_number')
```

- Reverts our database to remove the last migration change.
- Previous data within database does not change unless it was on the column 'phone_number' because we deleted it.

© CODINGWITHROBY

Alembic Downgrade?

- To run the the downgrade migration:

```
ericroby@FastAPI-Project ~%
alembic downgrade -1
```

- Will revert the last migration

© CODINGWITHROBY

FastAPI Testing

What is Testing

- It is a way for us to make sure our application is working as intended.
- Part of the Software Development Lifecycle (SDLC) that aims to identify:
  - Bugs
  - Errors
  - Defects
- Meets user requirements and specifications.
- Ensuring Software is high quality, reliable, secure and user friendly.

© CODINGWITHROBY

What is Testing

Manual Testing

Unit Testing

Integration Testing

© CODINGWITHROBY

Unit Testing

- Involves testing individual components or units of software in isolation from the rest of the application.
- Validates that each unit of the software performs as designed.
- Unit = Testable part of the application.
- Developers write unit tests during the development phase.
- Tests are automated and executed by a testing framework (Pytest).
- Benefit is to identify bugs early in the development process.

© CODINGWITHROBY

Integration Testing

- Focuses on testing the interactions between different units or components of the piece of software.
- The scope is broader than unit testing, as we are testing multiple units together.
- Helps identify problems for the entire solution.
- Example: Call an API endpoint and make sure the correct solution is returned.

© CODINGWITHROBY

Pytest

- Popular testing framework for Python
- Known for simplicity, scalability and ability to handle both unit and integration tests.
- Top reasons to use Pytest:
  - Simple & Flexible - Native Assertions
  - Fixtures - Feature setup and teardown
  - Parameterized Testing - Run same tests with different data

© CODINGWITHROBY

What is Testing

Both

Manual Testing

Unit Testing

Integration Testing

© CODINGWITHROBY

Getting Started With Testing

First Step

- Create a new directory on our project call ‘test’.
- Inside our test directory create a new file called:

```
__init__.py
```

- Inside our test directory create a new file called:

```
test_example.py
```


© CODINGWITHROBY

test_example.py

- Pytest will run all tests automatically that sit within files that have the name ‘test’ in them.
- For our demo, all tests will be in test_example.py so Pytest can find them easily.
- When we write tests for our application, we will create new tests from a new file that matches naming convention of project.
- Example: todos.py will be tested by test_todos.py

© CODINGWITHROBY

Create our first Unit Test

- Write our first assertion test.
  - Assertion = Statement that checks if a condition is true.
  - If condition is true = test passes.
  - If condition is false = test fails.

test_example.py:

```
def test_equal_or_not_equal():
        assert 3 == 3
```

Terminal:

```
ericroby@FastAPI-Project ~% pytest
```

Terminal:

```
Test/test_example.py

============ 1 passed ============
```


© CODINGWITHROBY

Create our first Unit Test

```
def test_equal_or_not_equal():
        assert 3 == 3
```

Pass

```
def test_equal_or_not_equal():
        assert 3 == 2
```

Fail

```
def test_equal_or_not_equal():
        assert 3 != 1
```

Pass

```
def test_equal_or_not_equal():
        assert 3 != 3
```

Fail

© CODINGWITHROBY

Pytest Basics

Pytest Basics

- Validate Integers

```
def test_equal_or_not_equal():
        assert 3 == 3
```

Pass

- Validate Instances

```
def test_is_instance():
        assert isInstance('this is a string', str)
        assert not isInstance('10', int)
```

Pass

© CODINGWITHROBY

Pytest Basics

- Validate Booleans

```
def test_boolean():
    validated = True
        assert validated is True
        assert ('hello' == 'world') is False
```

Pass

- Validate Types

```
def test_type():
        assert type('Hello' is str)
        assert type('World' is not int)
```

Pass

© CODINGWITHROBY

Pytest Basics

- Validate Greater Than & Less Than

```
def test_greater_and_less_than():
        assert 7 > 3
        assert 4 < 10
```

Pass

- Validate Types

```
def test_list():
        num_list = [1, 2, 3, 4, 5]
        any_list = [False, False]
        assert 1 in num_list
        assert 7 not in num_list
        assert all(num_list)
        assert not any(any_list)
```

Pass

© CODINGWITHROBY

Pytest Objects

Pytest Objects

- Create our Python Object

```
import pytest

class Student:
    def __init__(self, first_name: str, last_name: str, major: str, years:
int):
            self.first_name = first_name
            self.last_name = last_name
```

```
def test_person_initialization():
    p = Student('John', 'Doe', 'Computer Science', 3)
    assert p.first_name == 'John', 'First name should be John'
    assert p.last_name == 'Doe', 'Last name should be Doe'
    assert p.major == 'Computer Science'
    assert p.years == 3
```


© CODINGWITHROBY

Pytest Objects

- Create our Python Object

```python
import pytest

class Student:
    def __init__(self, first_name: str, last_name: str, major: str, years: int):
        self.first_name = first_name
        self.last_name = last_name
        self.major = major
        self.years = years
```

```python
@pytest.fixture
def default_employee():
    return Student('John', 'Doe', 'Computer Science', 3)
```

```python
def test_person_initialization(default_employee):
    assert default_employee.first_name == 'John', 'First name should be John'
    assert default_employee.last_name == 'Doe', 'Last name should be Doe'
    assert default_employee.major == 'Computer Science'
    assert default_employee.years == 3
```


© CODINGWITHROBY

Setup Test Dependencies & DB

Test Database

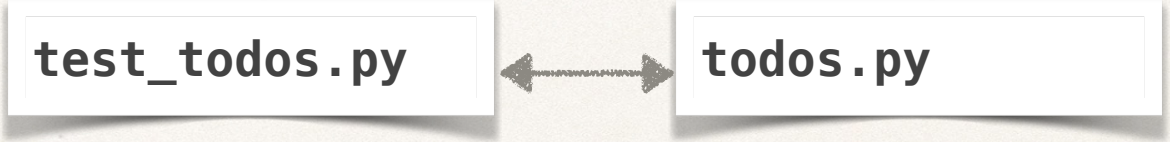
- Create a fake database that can store data.
- Create testing dependencies that are separate from our production dependencies.
- This way we can do integration testing to make sure our entire project is working correctly when we run our tests.
- App is live = production dependencies
- App is testing = testing dependencies

Separate shared dependencies in a different file later on.

© CODINGWITHROBY

Testing dependencies

- Create a new file called:



- Create new database engine for our testing environment

```
from sqlalchemy import create_engine

SQLALCHEMY_DATABASE_URL = “sqlite:///./testdb.db”
```


© CODINGWITHROBY

Testing dependencies

- Add the new engine:

```
from sqlalchemy import create_engine
from sqlalchemy.pool import StaticPool


SQLALCHEMY_DATABASE_URL = “sqlite:///./testdb.db”
engine = create_engine(
    SQLALCEMY_DATABASE_URL,
    connect_args={“check_same_thread”: False},
    Poolclass = StaticPool,
)
```


© CODINGWITHROBY

Testing dependencies

- Add TestingSessionLocal and Base

```
from sqlalchemy import create_engine
from sqlalchemy.pool import StaticPool
from ..database import Base


SQLALCHEMY_DATABASE_URL = “sqlite:///./testdb.db”
engine = create_engine(
    SQLALCEMY_DATABASE_URL,
    connect_args={“check_same_thread”: False},
    Poolclass = StaticPool,
)
TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base.metadata.create_all() = bind=engine
```


© CODINGWITHROBY

Testing dependencies

- Setup get_testing_db dependency

```
def override_get_db():
        db = TestingSessionLocal()
        try:
            yield db
        finally:
            db.close()

app.dependency_overrides[get_db] = override_get_db
```


© CODINGWITHROBY

Testing dependencies

- Mock our current logged in user for testing.

```
def override_get_db():
        db = TestingSessionLocal()
        try:
                yield db
        finally:
                db.close()

def override_get_current_user():
    return {'username': 'codingwithroby', 'id': 1, 'user_role': 'admin'}

app.dependency_overrides[get_db] = override_get_db

app.dependency_overrides[get_current_user] = override_get_current_user
```


© CODINGWITHROBY

FastAPI

- Normal:

```
app = FastAPI()
```

- Wrap it in testing client:

```
client = TestClient(app)
```


© CODINGWITHROBY

JSON WEB TOKEN (JWT) OVERVIEW

© CODINGWITHROBY

WHAT IS A JSON WEB TOKEN?

- JSON Web Token is a self-contained way to securely transmit data and information between two parties using a JSON Object.
- JSON Web Tokens can be trusted because each JWT can be digitally signed, which in return allows the server to know if the JWT has been changed at all
- JWT should be used when dealing with authorization
- JWT is a great way for information to be exchanged between the server and client

© CODINGWITHROBY

JSON WEB TOKEN STRUCTURE

- A JSON Web Token is created of three separate parts separated by dots (.) which include:
  - Header : (a)
  - Payload : (b)
  - Signature : (c)

```
aaaaaaaa.bbbbbbbb.cccccccc
```


© CODINGWITHROBY

JWT HEADER

- A JWT header usually consist of two parts:
  - (alg) The algorithm for signing
  - “typ” The specific type of token
- The JWT header is then encoded using Base64 to create the first part of the JWT (a)

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
aaaaaaaa.bbbbbbbb.cccccccc
```


© CODINGWITHROBY

JWT PAYLOAD

- A JWT Payload consists of the data. The Payloads data contains claims, and there are three different types of claims.
  - Registered
  - Public
  - Private
- The JWT Payload is then encoded using Base64 to create the second part of the JWT (b)

```
{
  "sub": "1234567890",
  "name": "Eric Roby",
  "given_name": "Eric",
  "family_name": "Roby",
  "email": "codingwithroby@gmail.com"
  "admin": true
}
```

```
aaaaaaaa.bbbbbbbb.cccccccc
```


© CODINGWITHROBY

JWT SIGNATURE

- A JWT Signature is created by using the algorithm in the header to hash out the encoded header, encoded payload with a secret.
- The secret can be anything, but is saved somewhere on the server that the client does not have access to
- The signature is the third and final part of a JWT (c)

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

```
aaaaaaaa.bbbbbbbb.cccccccc
```


© CODINGWITHROBY

JWT EXAMPLE

JWT Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

JWT Payload

```
{
  "sub": "1234567890",
  "name": "Eric Roby",
  "given_name": "Eric",
  "family_name": "Roby",
  "email": "codingwithroby@gmail.com"
  "admin": true
}
```

JWT Signature

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  learnonline)
```

JSON Web Token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkVyaWMgUm9ieSIsImdpdmVuX25hbWUiOiJFcmljIiwiZmFtaWx5X25hbWUiOiJSb2J5IiwiZW1haWwiOiJjb2Rpbmd3aXRocm9ieUBnbWFpbC5jb20iLCJhZG1pbiI6dHJ1ZX0.i8yqzOz-nWr5hwXqRYP18W9igUPoKMZiBZW315tK5g8

© CODINGWITHROBY

https://jwt.io

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkVyaWMgUm9ieSIsImdpdmVuX25hbWUiOiJFcmljIiwiZmFtaWx5X25hbWUiOiJSb2J5IiwiZW1haWwiOiJjb2Rpbmd3aXRocm9ieUBnbWFpbC5jb20iLCJhZG1pbiI6dHJ1ZX0.i8yqzOz-nWr5hwXqRYP18W9igUPoKMZiBZW315tK5g8
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


```
{
  "alg": "HS256",
  "typ": "JWT"
}
```


PAYLOAD: DATA


```
{
  "sub": "1234567890",
  "name": "Eric Roby",
  "given_name": "Eric",
  "family_name": "Roby",
  "email": "codingwithroby@gmail.com",
  "admin": true
}
```


VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  learnonline
) ☐ secret base64 encoded
```

Signature Verified

SHARE JWT

© CODINGWITHROBY

https://jwt.io

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkVya
WMgUm9ieSIsImdpdmVuX25hbWUiOiJFcmljIiwi
ZmFtaWx5X25hbWUiOiJSb2J5IiwiZW1haWwiOiJ
jb2Rpbmd3aXRocm9ieUBnbWFpbC5jb20iLCJhZG
1pbiI6dHJ1ZX0.i8yqzOz-
nWr5hwXqRYP18W9igUPoKMZiBZW315tK5g
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "Eric Roby",
  "given_name": "Eric",
  "family_name": "Roby",
  "email": "codingwithroby@gmail.com",
  "admin": true
}
```

VERIFY SIGNATURE

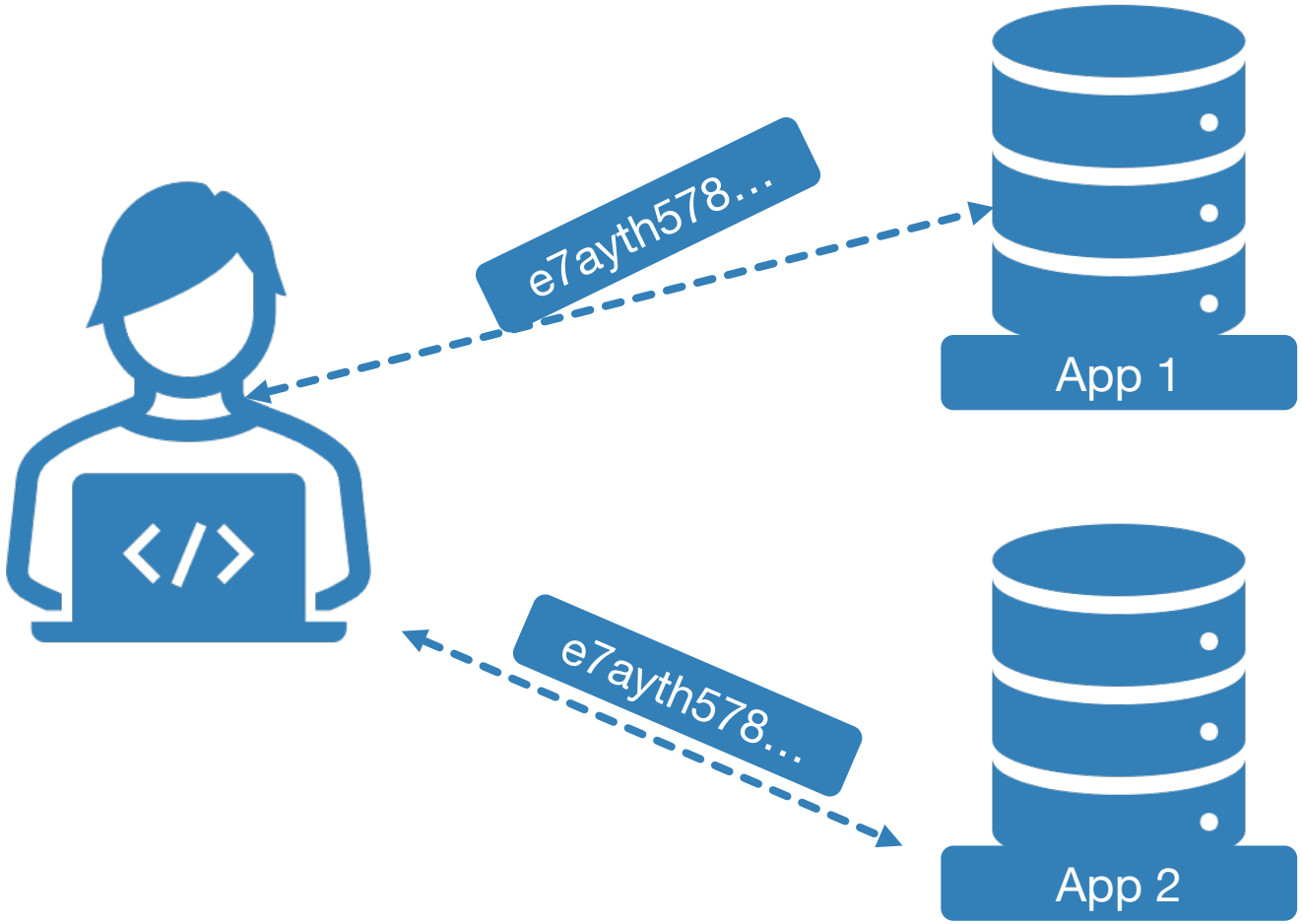
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  learnonline
) ☐ secret base64 encoded
```

⊗ Invalid Signature

SHARE JWT

© CODINGWITHROBY

JWT PRACTICAL USE CASE



© CODINGWITHROBY

PRODUCTION DATABASE INTRODUCTION

© CODINGWITHROBY

PRODUCTION DATABASE

- This section will go over installing a production

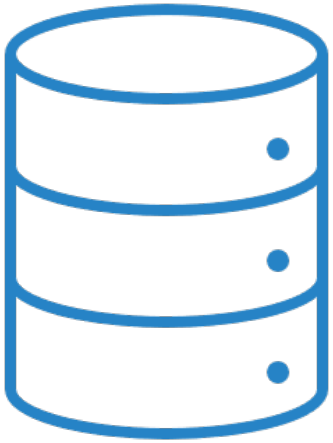
Before jumping in, lets take about the difference between SQLite and production DBMS

either one

© CODINGWITHROBY

PRODUCTION DBMS VS SQLITE3

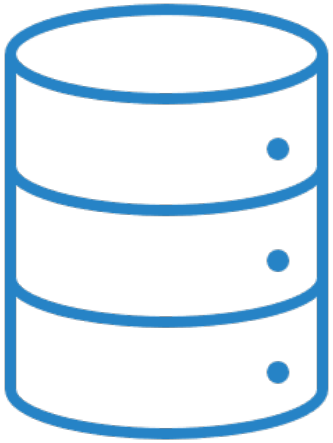
- SQLite3 strives to provide local data storage for individual applications and devices.
- SQLite3 emphasizes economy, efficiency and simplicity.
- For most small / medium applications, SQLite3 will works perfectly.
- SQLite3 focuses on different concepts than a production Database Management System.



© CODINGWITHROBY

PRODUCTION DBMS VS SQLITE3

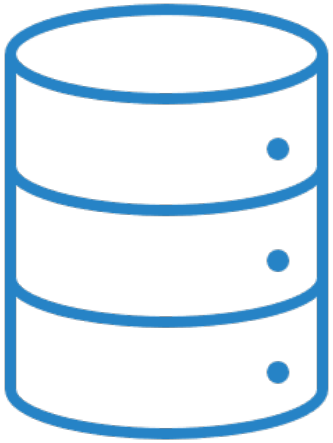
- MySQL & PostgreSQL focuses on a big difference compared to SQLite3.
- These production DBMS focuses on scalability, concurrency and control.
- If you application is going to have 10s of thousands of users, it may be wise to switch to a production DBMS
- If you application is only you, and a few others, SQLite3 will work great!



© CODINGWITHROBY

PRODUCTION DBMS KEY NOTES:

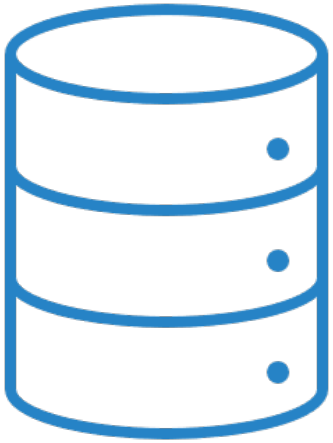
- SQLite3 runs in-memory, which allows development of a SQLite3 data to be easy, as it is part of your application!
- Production DBMS run on their own server and port. Which means you need to make sure the database is running, and have authentication linking to the DBMS
- (SQLite3) For deployment you can deploy a SQLite3 database along with the application
- (Prod DBMS) For deployment you will need to also deploy the database separate from the application

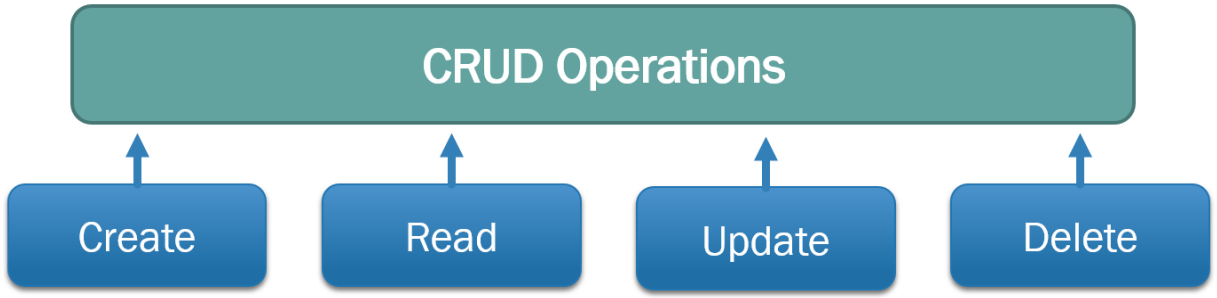


© CODINGWITHROBY

OVERVIEW OF SECTION (FOR BOTH MYSQL & POSTGRESQL)

- We will install the production DBMS
- Setup the tables and data within the production DBMS
- Connect the production DBMS to our application
- Push data from application to our production DBMS!





© CODINGWITHROBY

BASIC SQL QUERIES

© CODINGWITHROBY

INSERTING DATABASE TABLE (TODOS)

Id (PK)	title	description	priority	complete	owner (FK)

© CODINGWITHROBY

DELET THIS IS THE USERS TABLE

Id (PK)	email	username	first_name	last_name	hashed_passwor d	is_active

© CODINGWITHROBY

DELET THIS IS THE USERS TABLE

Id (PK)	email	username	first_name	last_name	hashed_passwor d	is_active
1	codingwithroby @gmail.com	codingwithroby	Eric	Roby	123abcEqw!..	1
2	exampleuser12 @example.com	exampleuser	Example	User	Gjjjd!2!..	1

© CODINGWITHROBY

INSERTING DATABASE TABLE (TODOS)

```
INSERT INTO todos (title,
description, priority, complete)

VALUES ('Go to store', 'To pick up
eggs' 4, False);
```

Id (PK)	title	description	priority	complete
1	Go to store	To pick up eggs	4	0

© CODINGWITHROBY

INSERTING DATABASE TABLE (TODOS)

```
INSERT INTO todos (title,
description, priority, complete)

VALUES ('Haircut', 'Need to get
length 1mm' 3, False);
```

Id (PK)	title	description	priority	complete
1	Go to store	To pick up eggs	4	0
2	Haircut	Need to get length 1mm	3	0

© CODINGWITHROBY

STARTING DATABASE TABLE (TODOS)

Id (PK)	title	description	priority	complete
1	Go to store	To pick up eggs	4	0
2	Haircut	Need to get length 1mm	3	0

© CODINGWITHROBY

STARTING DATABASE TABLE (TODOS)

Id (PK)	title	description	priority	complete
1	Go to store	To pick up eggs	4	0
2	Haircut	Need to get length 1mm	3	0
3	Feed dog	Make sure to use new food brand	5	0

© CODINGWITHROBY

STARTING DATABASE TABLE (TODOS)

Id (PK)	title	description	priority	complete
1	Go to store	To pick up eggs	4	0
2	Haircut	Need to get length 1mm	3	0
3	Feed dog	Make sure to use new food brand	5	0
4	Water plant	Inside and Outside plants	4	0

© CODINGWITHROBY

STARTING DATABASE TABLE (TODOS)

Id (PK)	title	description	priority	complete
1	Go to store	To pick up eggs	4	0
2	Haircut	Need to get length 1mm	3	0
3	Feed dog	Make sure to use new food brand	5	0
4	Water plant	Inside and Outside plants	4	0
5	Learn something new	Learn to program	5	0

© CODINGWITHROBY

STARTING DATABASE TABLE (TODOS)

Id (PK)	title	description	priority	complete
1	Go to store	To pick up eggs	4	0
2	Haircut	Need to get length 1mm	3	0
3	Feed dog	Make sure to use new food brand	5	0
4	Water plant	Inside and Outside plants	4	0
5	Learn something new	Learn to program	5	0
6	Shower	You have not showered in days	5	0

© CODINGWITHROBY

DELETE

Id (PK)	title	description	priority	complete	owner
3	Feed dog	Make sure to use new food brand	5	0	2
4	Water plant	Inside and Outside plants	4	0	2
6	Shower	You have not showered in days	5	0	2

© CODINGWITHROBY

SELECT SQL QUERIES

```
SELECT * FROM todos;
```

Select ALL columns and rows

Id (PK)	title	description	priority	complete
1	Go to store	To pick up eggs	4	0
2	Haircut	Need to get length 1mm	3	0
3	Feed dog	Make sure to use new food brand	5	0
4	Water plant	Inside and Outside plants	4	0
5	Learn something new	Learn to program	5	0
6	Shower	You have not showered in days	5	0

© CODINGWITHROBY

SELECT SQL QUERIES

```
SELECT title FROM todos;
```

Select just title from columns

title
Go to store
Haircut
Feed dog
Water plant
Learn something new
Shower

© CODINGWITHROBY

SELECT SQL QUERIES

```
SELECT description FROM todos;
```

Select just description from columns

description
To pick up eggs
Need to get length 1mm
Make sure to use new food brand
Inside and Outside plants
Learn to program
You have not showered in days

© CODINGWITHROBY

SELECT SQL QUERIES

```
SELECT title, description FROM
todos;
```

Select title, description from columns

title	description
Go to store	To pick up eggs
Haircut	Need to get length 1mm
Feed dog	Make sure to use new food brand
Water plant	Inside and Outside plants
Learn something new	Learn to program
Shower	You have not showered in days

© CODINGWITHROBY

SELECT SQL QUERIES

```
SELECT title, description, priority FROM
todos;
```

Select title, description and priority from columns

title	description	priority
Go to store	To pick up eggs	4
Haircut	Need to get length 1mm	3
Feed dog	Make sure to use new food brand	5
Water plant	Inside and Outside plants	4
Learn something new	Learn to program	5
Shower	You have not showered in days	5

© CODINGWITHROBY

WHERE SQL QUERIES

WHERE Clause

© CODINGWITHROBY

WHERE SQL QUERIES

```
SELECT * FROM todos WHERE priority=5;
```

Select ALL rows & columns WHERE priority = 5

Id (PK)	title	description	priority	complete
3	Feed dog	Make sure to use new food brand	5	0
5	Learn something new	Learn to program	5	0
6	Shower	You have not showered in days	5	0

© CODINGWITHROBY

WHERE SQL QUERIES

```
SELECT * FROM todos WHERE title='Feed dog';
```

Select ALL rows & columns WHERE title= Feed dog

Id (PK)	title	description	priority	complete
3	Feed dog	Make sure to use new food brand	5	0

© CODINGWITHROBY

WHERE SQL QUERIES

```
SELECT * FROM todos WHERE id=2;
```

Select ALL rows & columns WHERE id= 2

Id (PK)	title	description	priority	complete
2	Haircut	Need to get length 1mm	3	0

© CODINGWITHROBY

UPDATE SQL QUERIES

UPDATE Clause

© CODINGWITHROBY

WHERE SQL QUERIES

```
UPDATE todos SET complete=True WHERE
title='Learn something new';
```

Update ALL rows & columns to now have complete = True WHERE id = 5

Id (PK)	title	description	priority	complete
5	Learn something new	Learn to program	5	1

1 == True

© CODINGWITHROBY

WHERE SQL QUERIES

```
UPDATE todos SET complete=True WHERE id=5;
```

Update ALL rows & columns to now have complete = True WHERE id = 5

Id (PK)	title	description	priority	complete
5	Learn something new	Learn to program	5	1

1 == True

© CODINGWITHROBY

DELETE SQL QUERIES

DELETE Clause

© CODINGWITHROBY

DELETE SQL QUERIES

```
DELETE FROM todos WHERE id=5;
```

Delete ALL rows and columns where id = 5

Id (PK)	title	description	priority	complete
1	Go to store	To pick up eggs	4	0
2	Haircut	Need to get length 1mm	3	0
3	Feed dog	Make sure to use new food brand	5	0
4	Water plant	Inside and Outside plants	4	0
6	Shower	You have not showered in days	5	0

© CODINGWITHROBY

DELETE SQL QUERIES

```
DELETE FROM todos WHERE complete=0;
```

Delete ALL rows and columns where complete = 0

Id (PK)	title	description	priority	complete

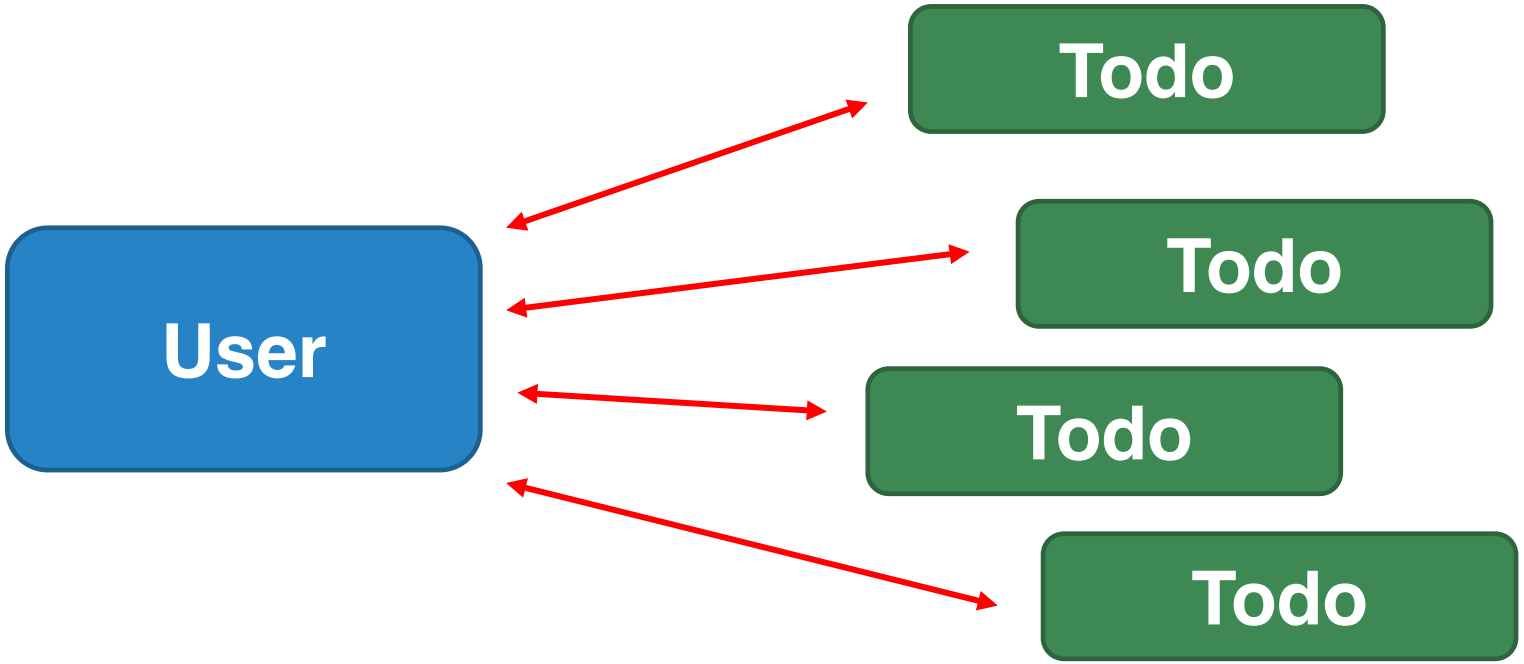
© CODINGWITHROBY

ONE TO MANY INTRODUCTION

© CODINGWITHROBY

WHAT IS A ONE TO MANY RELATIONSHIP

- A user can have many todos



© CODINGWITHROBY

WHAT IS A ONE TO MANY RELATIONSHIP

Users & Todos

Id (PK)	email	username	first_name	last_name	hashed_password	is_active

Id (PK)	title	description	priority	complete

© CODINGWITHROBY

WHAT IS A ONE TO MANY RELATIONSHIP

Users & Todos

- Id (PK)
- email
- username
- first_name
- last_name
- hashed_password
- is_active

- Id (PK)
- title
- description
- priority
- complete
- Owner (FK)

© CODINGWITHROBY

WHAT IS A ONE TO MANY RELATIONSHIP

Id (PK)	title	description	priority	complete

Add new column

Id (PK)	title	description	priority	complete	owner (FK)

© CODINGWITHROBY

WHAT IS A ONE TO MANY RELATIONSHIP

Users

Id (PK)	email	username	first_name	last_name	hashed_password	is_active
1	codingwithroby@gmail.com	codingwithroby	Eric	Roby	123abcEqw!..	1
2	exampleuser12@example.com	exampleuser	Example	User	Gjjjd!2!..	1

Todos

Id (PK)	title	description	priority	complete	owner (FK)
1	Go to store	To pick up eggs	4	0	1
2	Haircut	Need to get length 1mm	3	0	1
3	Feed dog	Make sure to use new food brand	5	0	2
4	Water plant	Inside and Outside plants	4	0	2
5	Learn something new	Learn to program	5	0	1
6	Shower	You have not showered in days	5	0	2

© CODINGWITHROBY

FOREIGN KEYS

© CODINGWITHROBY

WHAT IS A FOREIGN KEY?

- A foreign key (FK) is a column within a relational database table that provides a link betw
- A foreign key references
- Most relational database o link tables together to prese

Users

Id (PK)	email	username	first_name	last_name	hashed_password	is_active
1	codingwithroby@gmail.com	codingwithroby	Eric	Roby	123abcEqw!..	1
2	exampleuser12@example.com	exampleuser	Example	User	Gjjjd!2!..	1

Todos

Id (PK)	title	description	priority	complete	owner (FK)
1	Go to store	To pick up eggs	4	0	1
2	Haircut	Need to get length 1mm	3	0	1
3	Feed dog	Make sure to use new food brand	5	0	2
4	Water plant	Inside and Outside plants	4	0	2
5	Learn something new	Learn to program	5	0	1
6	Shower	You have not showered in days	5	0	2

© CODINGWITHROBY

FOREIGN KEYS

```
SELECT * FROM todos;
```

Select ALL columns and rows

Id (PK)	title	description	priority	complete	owner (FK)
1	Go to store	To pick up eggs	4	0	1
2	Haircut	Need to get length 1mm	3	0	1
3	Feed dog	Make sure to use new food brand	5	0	2
4	Water plant	Inside and Outside plants	4	0	2
5	Learn something new	Learn to program	5	0	1
6	Shower	You have not showered in days	5	0	2

© CODINGWITHROBY

FOREIGN KEYS

```
SELECT * FROM users;
```

Select ALL columns and rows

Id (PK)	email	username	first_name	last_name	hashed_password	is_active
1	codingwithroby@gmail.com	codingwithroby	Eric	Roby	123abcEqw!..	1
2	exampleuser12@example.com	exampleuser	Example	User	Gjjjd!2!..	1

© CODINGWITHROBY

- Each API request a user will have their ID attached
  - If we have the user ID attached to each request, we can use the ID to find their todos

Id (PK)	email	username	first_name	last_name	hashed_password	is_active
1	codingwithroby@gmail.com	codingwithroby	Eric	Roby	123abcEqw!..	1
2	exampleuser12@example.com	exampleuser	Example	User	Gjjjd!2!..	1

Id (PK)	title	description	priority	complete	owner (FK)
1	Go to store	To pick up eggs	4	0	1
2	Haircut	Need to get length 1mm	3	0	1
3	Feed dog	Make sure to use new food brand	5	0	2
4	Water plant	Inside and Outside plants	4	0	2
5	Learn something new	Learn to program	5	0	1
6	Shower	You have not showered in days	5	0	2

© CODINGWITHROBY

WHERE SQL QUERIES

```
SELECT * FROM todos WHERE owner=1;
```

Select ALL rows & columns WHERE owner = 1

Id (PK)	title	description	priority	complete	owner
1	Go to store	To pick up eggs	4	0	1
2	Haircut	Need to get length 1mm	3	0	1
5	Learn something new	Learn to program	5	0	1

© CODINGWITHROBY

WHERE SQL QUERIES

```
SELECT * FROM todos WHERE owner=2;
```

Select ALL rows & columns WHERE owner = 1

Id (PK)	title	description	priority	complete	owner
3	Feed dog	Make sure to use new food brand	5	0	2
4	Water plant	Inside and Outside plants	4	0	2
6	Shower	You have not showered in days	5	0	2

© CODINGWITHROBY

MYSQL INTRODUCTION

© CODINGWITHROBY

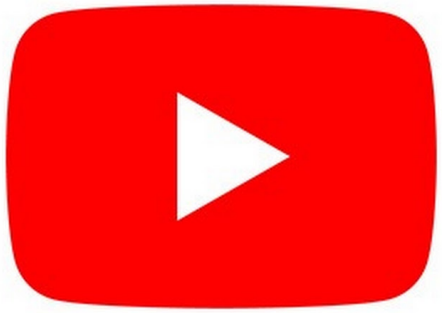
WHAT IS MYSQL

- Open-source relational database management system
- Requires a server
- Production ready
- Scalable
- Secure

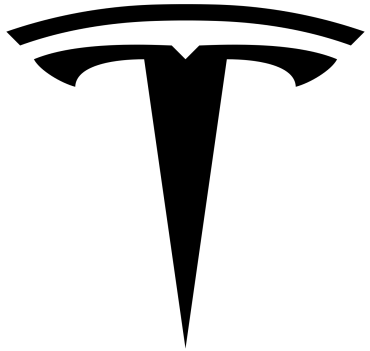
© CODINGWITHROBY

WHO USES/USED MYSQL?

Facebook



YouTube



Tesla

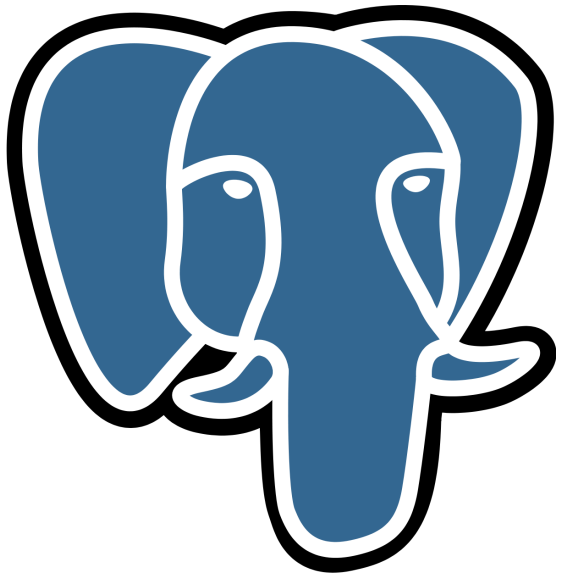
© CODINGWITHROBY

POSTGRESQL INTRODUCTION

© CODINGWITHROBY

WHAT IS POSTGRESQL

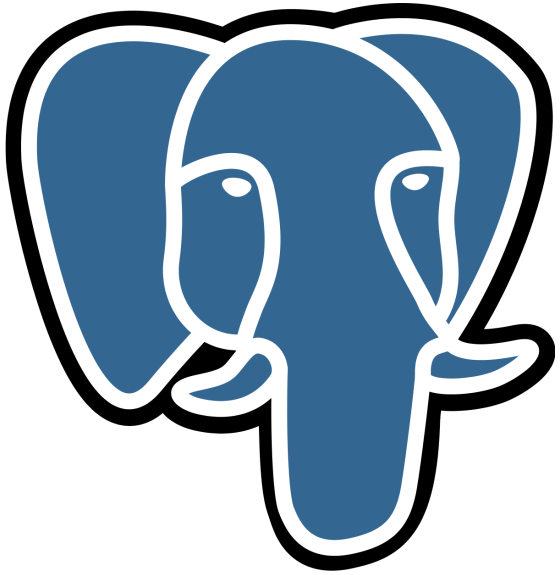
- Production ready
- Open-source relational database management system
- Secure
- Requires a server
- Scalable



© CODINGWITHROBY

WHAT WILL WE COVER?

- How to install PostgreSQL
  - Windows
  - Mac
- Setup SQL tables
- Connect PostgreSQL to our application



© CODINGWITHROBY

WHO USES/USED POSTGRESQL?

Appl
e

Reddit

Twitch

© CODINGWITHROBY

ROUTING INTRODUCTION

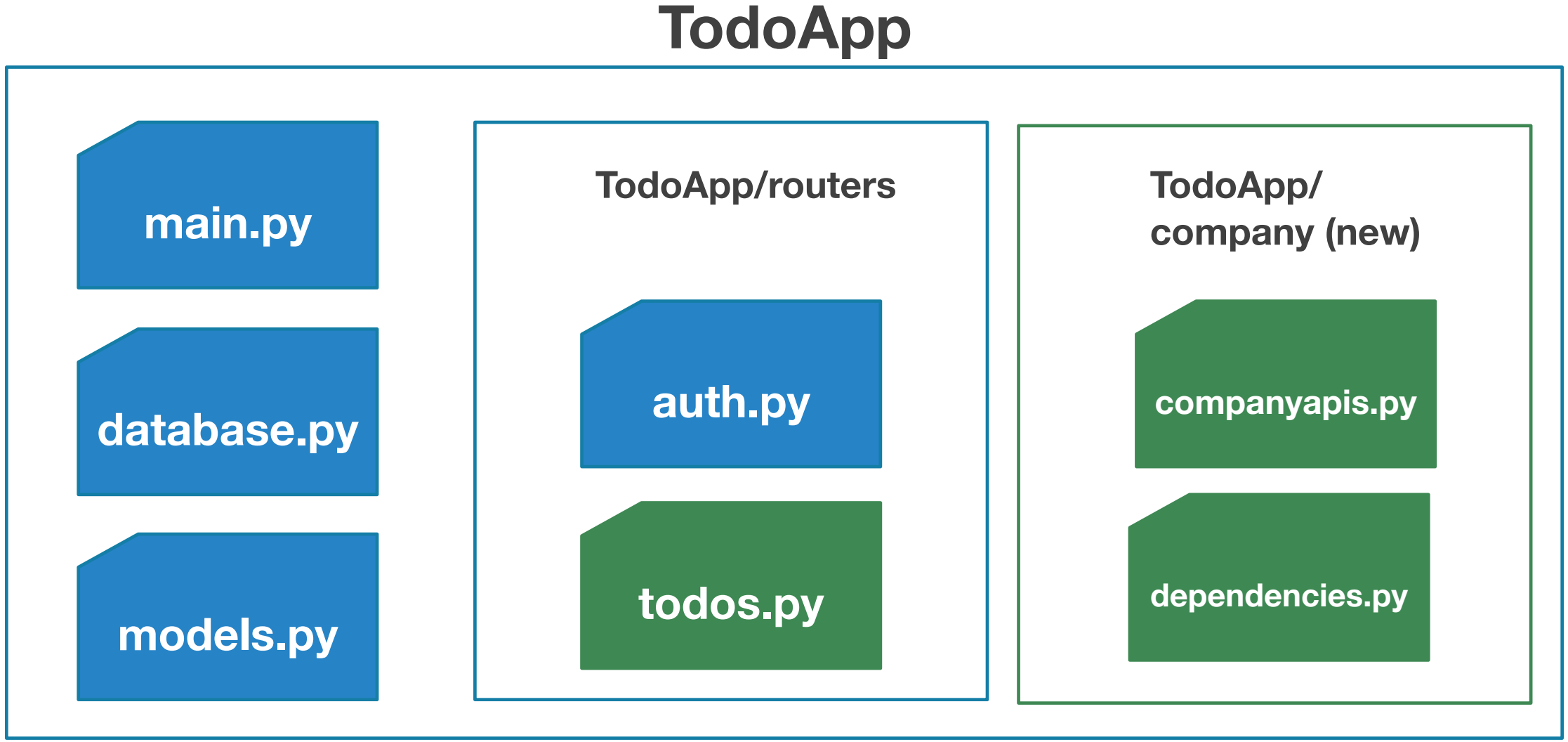
© CODINGWITHROBY

WHAT IS ROUTING

- Rare that you want your entire application to be on a single file
- Flexible tool to structure your application
- Scalable architecture
- Organize file structure

© CODINGWITHROBY

NEW PROJECT STRUCTURE



© CODINGWITHROBY

auth

POST	/auth/create/user	Create New User
POST	/auth/token	Login For Access Token

todos

GET	/todos/	Read All
POST	/todos/	Create Todo
GET	/todos/user	Read All By User
GET	/todos/{todo_id}	Read Todo
PUT	/todos/{todo_id}	Update Todo
DELETE	/todos/{todo_id}	Delete Todo

companyapis

GET	/companyapis/	Get Company Name
GET	/companyapis/employees	Number Of Employees

© CODINGWITHROBY

JINJA SCRIPTS

© CODINGWITHROBY

WHAT JINJA?

- Fast, expressive and extensible templating language
- Able to write code similar to Python in the DOM
- The template is passed data to render within the final document

© CODINGWITHROBY

WHAT ARE JINJA TEMPLATING TAGS AND SCRIPTS?

- Jinja tags allows developers to be confident while working with backend data, using tags that are similar to HTML.

```
<link rel="stylesheet" type="text/css" href="{{ url_for('static', path='/todo/css/base.css') }}">
```


© CODINGWITHROBY

WHAT ARE JINJA TEMPLATING TAGS AND SCRIPTS?

- Now image we have a list of *todos* that we retrieved from the database. We can pass the entire list of *todos* into the front-end and loop through each *todo* with this simple ‘for loop’ on the template.

todos.py

```
context:
  {
    "todos": todo_list
  }
```

Home.html

```
{% for todo in todos %}
    Do something with todo
{% endfor %}
```


© CODINGWITHROBY

WHAT ARE JINJA TEMPLATING TAGS AND SCRIPTS?

- We can also use Jinja templating language with *if else* statements. One thing that may stand out is the double brackets with *todos|length*

```
{% if todos %}
    Displaying: {{ todos|length }} Todos
{% else %}
    You don't have any todos :)
{% endif %}
```


© CODINGWITHROBY

LET’S LEARN GIT

© CODINGWITHROBY

WHAT IS GIT?

- Free and open-source distributed **version control system**
- Can handle small to large applications and projects
- Allows team members to use same files by distributed branches/ environments

© CODINGWITHROBY

GIT EXAMPLE?

Version 1

```
class MathFunctions

  def addition(a, b):
    return a + b
```

Simple Python Class

Addition Function

We need to add subtraction to MathFunctions

© CODINGWITHROBY

GIT EXAMPLE?

Version 2

```
class MathFunctions

  def addition(a, b):
    return a + b

  def subtraction(a, b):
    return a - b
```

Simple Python Class

Addition Function

Subtraction Function

New Function

New Version Of Code

© CODINGWITHROBY

GIT EXAMPLE?

Version 3

```
class MathFunctions

  def addition(a, b):
    return a + b

  def subtraction(a, b):
    return a - b

  def multiplication(a, b):
    return a * b
```

Simple Python Class

Addition Function

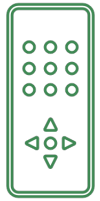
Subtraction Function

Multiple Function

© CODINGWITHROBY

GIT EXAMPLE?

Version Control



Version 1

```
class MathFunctions:

  def addition(a, b):
    return a + b
```

Commit 1

Version 2

```
class MathFunctions:

  def addition(a, b):
    return a + b

  def subtraction(a, b):
    return a - b
```

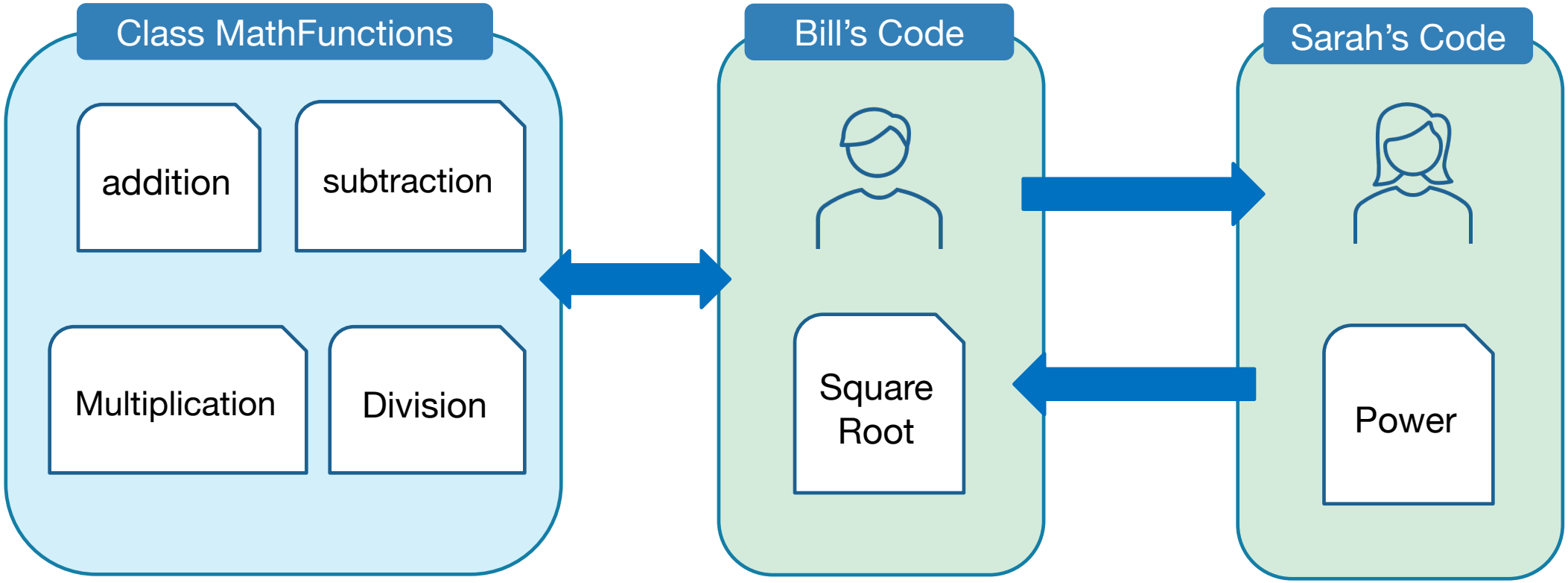
Commit 2

Version 3

```
class MathFunctions:

  def addition(a, b):
    return a + b

  def subtraction(a, b):
    return a - b

  def multiplication(a, b):
    return a * b
```

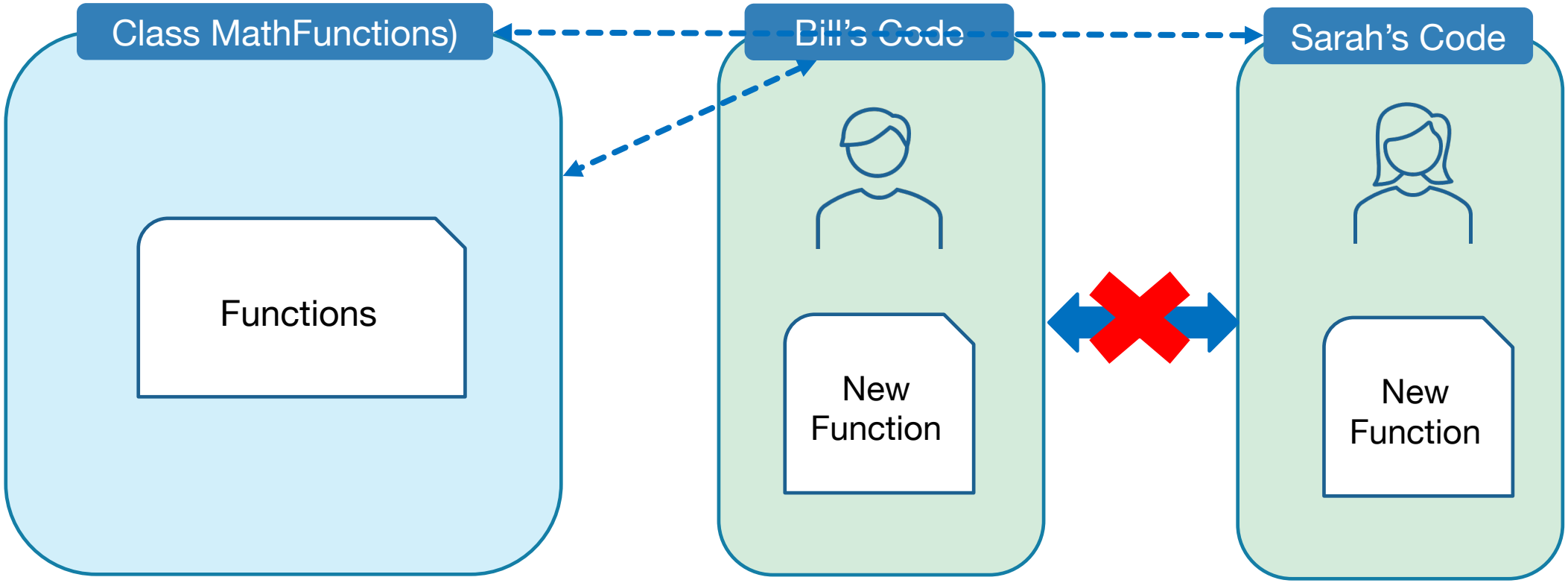

© CODINGWITHROBY

WHAT IS GIT?



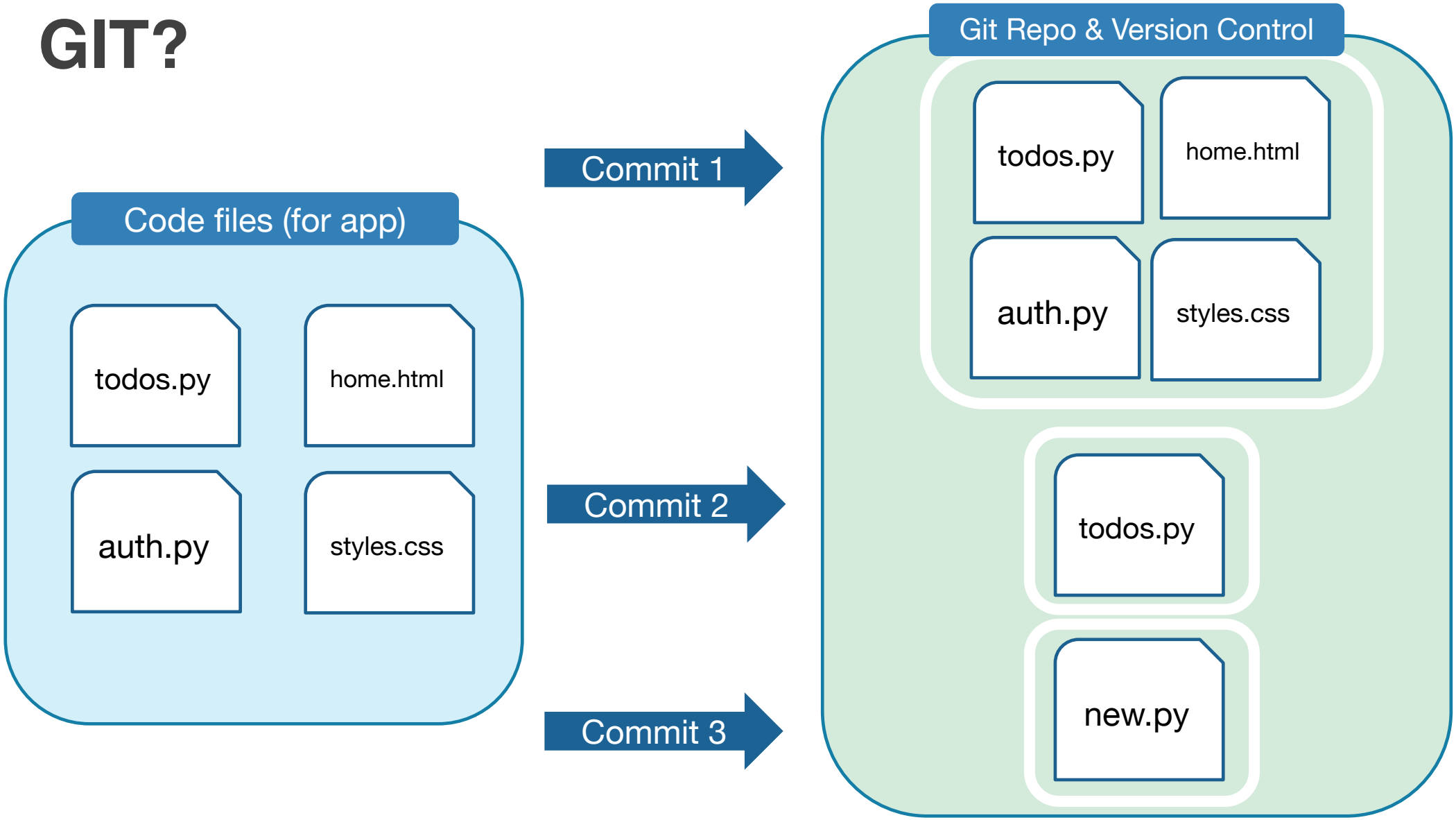
© CODINGWITHROBY

WHAT IS GIT?



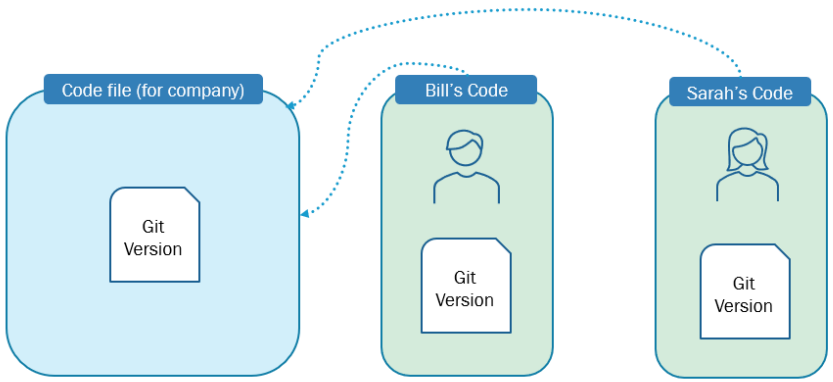
© CODINGWITHROBY

GIT?



© CODINGWITHROBY

WHAT IS GIT?



Free and open-source distributed version control system

Version 1

```
class MathFunctions:

  def addition(a, b):
    return a + b)
```

Version 2

```
class MathFunctions:

  def addition(a, b):
    return a + b)

  def subtraction(a, b):
    return a - b)
```

Version 3

```
class MathFunctions:

  def addition(a, b):
    return a + b)

  def subtraction(a, b):
    return a - b)

  def multiplication(a, b):
    return a * b)
```


© CODINGWITHROBY

WHAT IS GIT?

Free and open-source distributed version control system

- Track Changes
- Version Control
- Allows team members to use same files without needing to sync every time

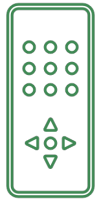
© CODINGWITHROBY

GIT BASICS

© CODINGWITHROBY

GIT EXAMPLE?

Version Control



Version 1

```
class MathFunctions:

  def addition(a, b):
    return a + b
```

Commit 1

Version 2

```
class MathFunctions:

  def addition(a, b):
    return a + b

  def subtraction(a, b):
    return a - b
```

Commit 2

Version 3

```
class MathFunctions:

  def addition(a, b):
    return a + b

  def subtraction(a, b):
    return a - b

  def multiplication(a, b):
    return a * b
```


© CODINGWITHROBY

GIT EXAMPLE?

Git Command	Details
git init	Initializes a new, empty repository

© CODINGWITHROBY

GIT EXAMPLE?

Git Command	Details
git init	Initializes a new, empty repository
git add .	Adds files from a non-staged area to a staging GIT area

© CODINGWITHROBY

GIT EXAMPLE?

Git Command	Details
git init	Initializes a new, empty repository
git add .	Adds files from a non-staged area to a staging GIT area
git commit –m “info here”	Move files from a staging area to a commit

© CODINGWITHROBY

GIT EXAMPLE?

Git Command	Details
git init	Initializes a new, empty repository
git add .	Adds files from a non-staged area to a staging GIT area
git commit –m “info here”	Move files from a staging area to a commit
git checkout <commit #>	Open up previous commit

© CODINGWITHROBY

GIT EXAMPLE?

Git Command	Details
git init	Initializes a new, empty repository
git add .	Adds files from a non-staged area to a staging GIT area
git commit –m “info here”	Move files from a staging area to a commit
git checkout <commit #>	Open up previous commit
git log	Shows committed Snapshots

© CODINGWITHROBY

GIT BASICS

python-git-basics

```
class main:

  print('Hello World')
```

GIT

```
git init
```

un-stashed

```
git add .
```

stashed

```
class main:

  print('Hello World')
```

```
git commit –m "First commit."
```

Commit 1 Complete

© CODINGWITHROBY

GIT BASICS

python-git-basics

```
class main:

  print('Hello World')

  print(2 + 2)
```

stashed

```
class main:

  print('Hello World')

  print(2 + 2)
```

un-stashed

```
git add .
```

```
git commit –m "Added addition functionality"
```

Commit 2 Complete

© CODINGWITHROBY

GIT BASICS

python-git-basics

```
git log
```

2 Different Commits

```
commit
aabbccddee112233445566
Author: Eric Roby
"Added Addition Function"

commit
aabbccddee112233445567
Author: Eric Roby
"First Commit"
```

```
git checkout
aabbccddee112233445567
```


© CODINGWITHROBY

GIT BRANCHES

© CODINGWITHROBY

WHAT IS A GIT BRANCH?

A pointer - to take a snapshot of a change. Branches can be merged into other branches

- A pointer of data change
- Isolation of feature development
- Linear development

© CODINGWITHROBY

GIT EXAMPLE?

Git Command	Details
git branch <branch name>	Create new isolated branch

© CODINGWITHROBY

GIT EXAMPLE?

Git Command	Details
git branch <branch name>	Create new isolated branch
git checkout branch <branch name>	Change root to branch selected

© CODINGWITHROBY

GIT EXAMPLE?

Git Command	Details
git branch <branch name>	Create new isolated branch
git checkout branch <branch name>	Change root to branch selected
git switch branch <branch name>	Same as above new command as of Git (2.23)

© CODINGWITHROBY

GIT BRANCHES?

All code is currently in the ***master branch***

```
git branch multiplication-branch
```

New branch

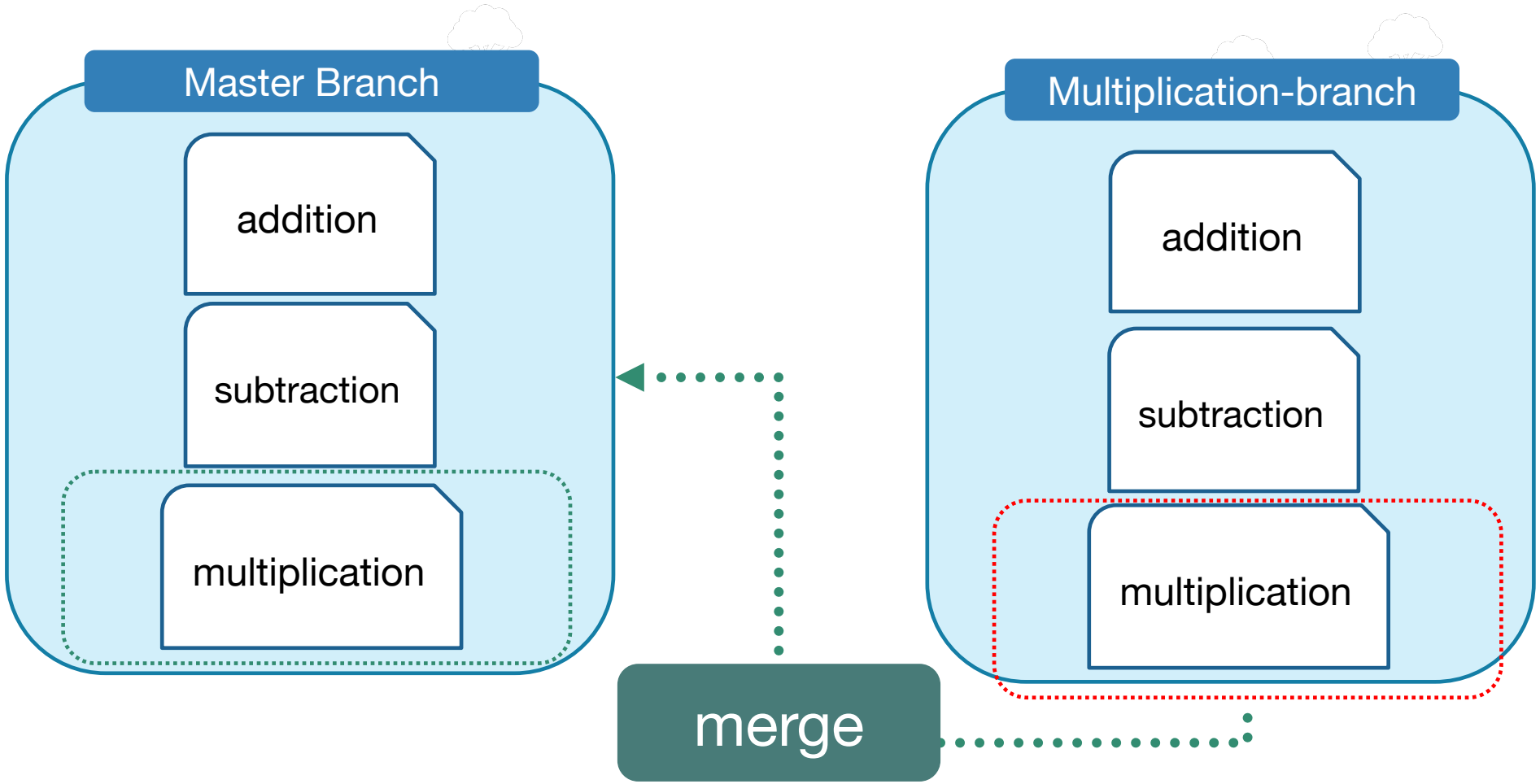
```
git checkout multiplication-branch
```

Checkout new branch

master branch ← → Multiplication-branch

© CODINGWITHROBY

GIT EXAMPLE?



© CODINGWITHROBY

GIT BRANCHES

python-git-basics

```
class MathFunctions:

  def addition(a, b):
    return a + b

  def subtraction(a, b):
    return a - b
```

detached-head

merge

master branch

```
git merge <commit id>
```

git checkout multiplication-branch

multiplication-branch

```
class MathFunctions:

  def addition(a, b):
    return a + b

  def subtraction(a, b):
    return a - b

  def multiplication(a, b):
    return a * b
```

```
git commit –m “adding multiplication function.”
```

master branch

```
git merge multiplication-branch
```


© CODINGWITHROBY

WHAT IS GITHUB?

© CODINGWITHROBY

WHAT IS GITHUB?

- Git Repository hosting service
- User friendly interface
- Large development platform

GitHub

© CODINGWITHROBY

WHAT IS GITHUB?

git

Free and open-source distributed version control system

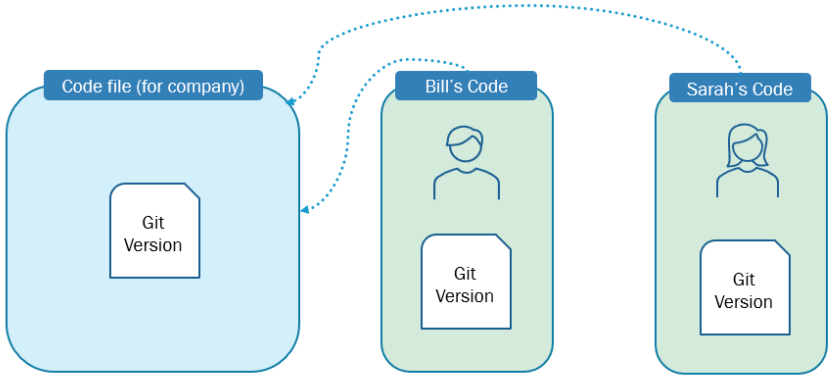
Top Git Repository hosting service available

© CODINGWITHROBY

WHAT IS GITHUB?

GitHub

All Git data is saved on Cloud. Can access version control and team's merges



Git

Version 1

```
class MathFunctions:

  def addition(a, b):
    return a + b)
```

Version 2

```
class MathFunctions:

  def addition(a, b):
    return a + b)

  def subtraction(a, b):
    return a - b)
```

Version 3

```
class MathFunctions:

  def addition(a, b):
    return a + b)

  def subtraction(a, b):
    return a - b)

  def multiplication(a, b):
    return a * b)
```


© CODINGWITHROBY

WHAT IS RENDER?

© CODINGWITHROBY

WHAT IS RENDER?

- Platform as a service (PaaS)
- Helps developers build, run and operate applications entirely on the cloud
- Developers can focus on coding and not have to worry about the infrastructure of their applications

© CODINGWITHROBY

RENDER PRICING?

- Pricing depends on many factors of how you want your application to perform online
- Free Trial
- Free Plan
- Businesses of all sizes use Render as their cloud PaaS provider.

© CODINGWITHROBY

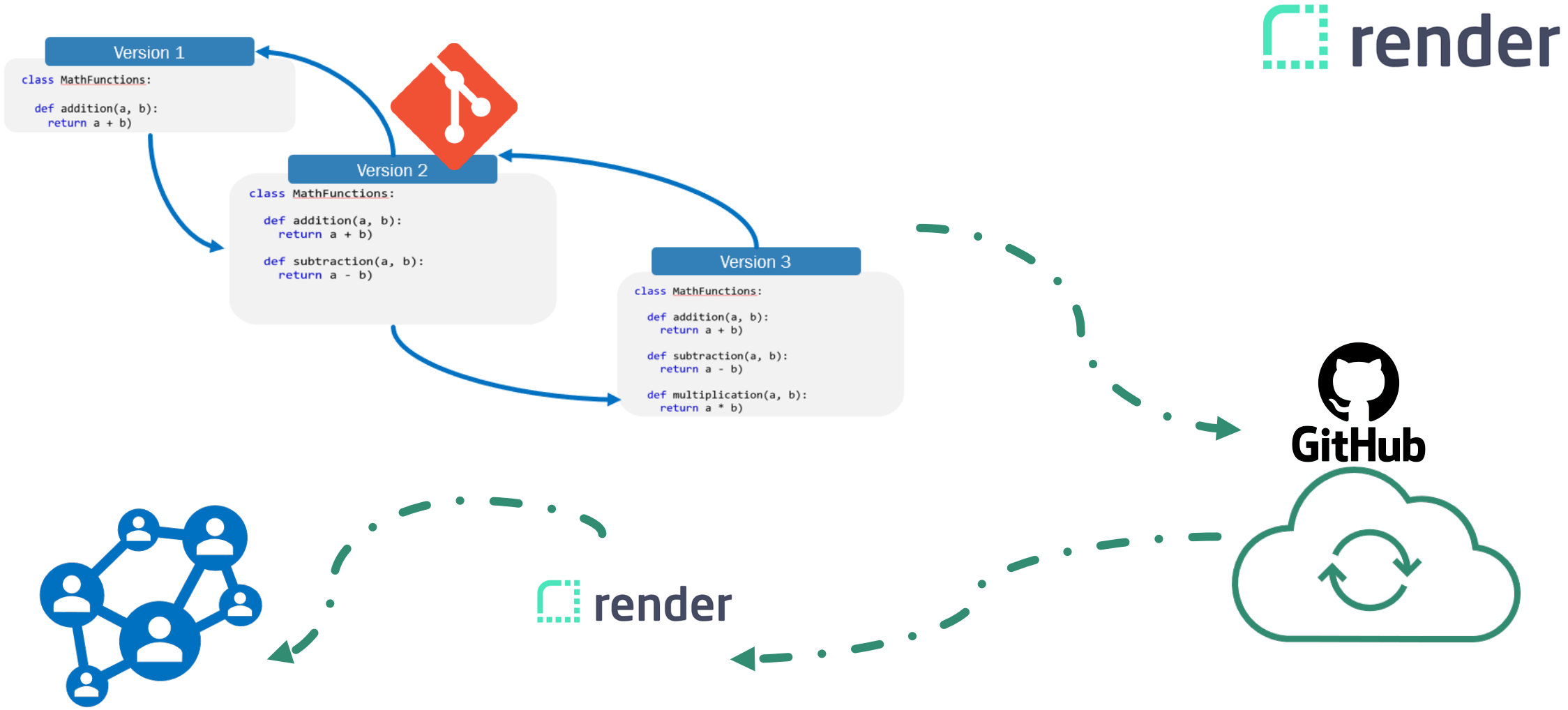
WHAT IS RENDER?

- Code deployment system
- Continuous Integration & Continuous Deployment (CI/CD)
- Load Balancing
- and more…

Focus on Code

© CODINGWITHROBY

WHAT IS RENDER?



© CODINGWITHROBY

DIFFERENT PLATFORMS?

render

aws

© CODINGWITHROBY

THANK YOU!!

© CODINGWITHROBY

CONTACT ME!

- codingwithroby@gmail.com
- Instagram: @codingwithroby

© CODINGWITHROBY

LET'S LEARN PYTHON

Assignment!

© CODINGWITHROBY

LET'S LEARN PYTHON

Variables!

© CODINGWITHROBY

LET'S LEARN PYTHON

Comments!

© CODINGWITHROBY

LET'S LEARN PYTHON

String

© CODINGWITHROBY

LET'S LEARN PYTHON

User Input!

© CODINGWITHROBY

LET'S LEARN PYTHON

Lists!

© CODINGWITHROBY

LET'S LEARN PYTHON

Sets & Tuples!

© CODINGWITHROBY

LET'S LEARN PYTHON

Boolean &

© CODINGWITHROBY

LET'S LEARN PYTHON

If Else!

© CODINGWITHROBY

LET'S LEARN PYTHON

Loops!

© CODINGWITHROBY

LET'S LEARN PYTHON

Dictionaries!

© CODINGWITHROBY

LET'S LEARN PYTHON

Functions!

© CODINGWITHROBY

Object Oriented Programming

What is Object Oriented Programming?

© CODINGWITHROBY

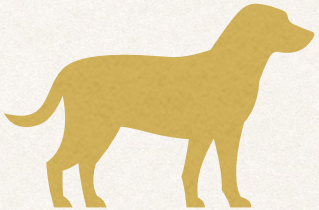
What is Object Oriented Programming?

- **Object Oriented Programming (OOP) -** is a programming paradigm based on the concepts of objects, which can contain data and code.
- OOP has benefits that include:
  - Scalability
  - Efficiency
  - Reusability

© CODINGWITHROBY

What are objects?

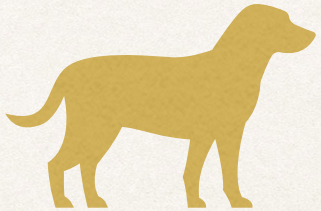
- There are objects around us in real life
- Looking out my window I see:
  - Trees
  - Houses
  - My dog, Milo.
- All of these are real objects that we could create.



© CODINGWITHROBY

What are objects?

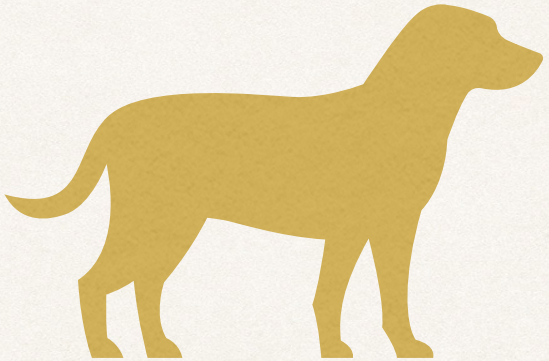
- Looking at these objects there are a two ways to define them:
  - Behavior
  - State
- Let's take a closer look at my dog, Milo.



© CODINGWITHROBY

Behavior

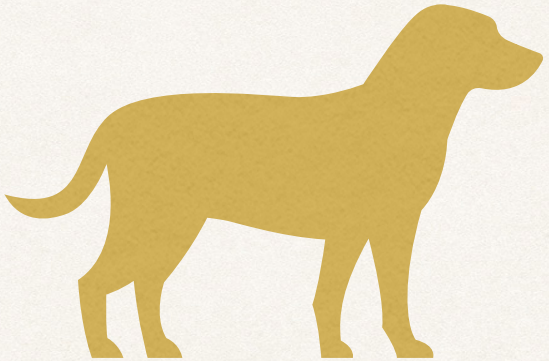
- Milo has many behaviors that a normal dog would have:
  - Barking
  - Eating
  - Drinking
  - Sleeping
  - Walking
  - Running



© CODINGWITHROBY

State

- Milo has individual pieces of state:
  - 4 legs
  - 2 ears
  - Goldendoodle
  - 5 years old
  - Yellow



© CODINGWITHROBY

Primitive Data Types

File: Main.py

```
legs: int = 4
arms: int = 2
type: str = 'Goldendoodle'
age: int = 5
color: str = 'Yellow'
```


© CODINGWITHROBY

Dog as an Object

File: Dog.py

```
class Dog:

        legs: int = 4
        ears: int = 2
        type: str = ‘Goldendoodle’
        age: int = 5
        color: str = ‘Yellow’
```


© CODINGWITHROBY

Dog Object

File: Main.py

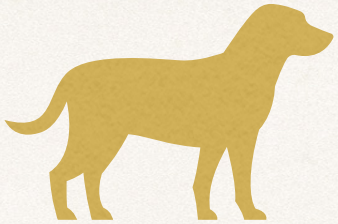
```
from Dog import *

dog = Dog()

dog.legs
dog.ears
dog.type
dog.age
dog.color
```

File: Dog.py

```
class Dog:

    legs: int = 4
    ears: int = 2
    type: str = 'Goldendoodle'
    age: int = 5
    color: str = 'Yellow'
```


© CODINGWITHROBY

Four Pillars



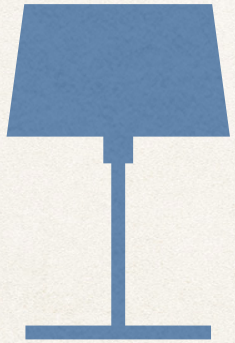
File: Main.py

```
from Dog import *

dog = Dog()

dog.legs;
dog.ears;
dog.type;
dog.age;
dog.color;
```

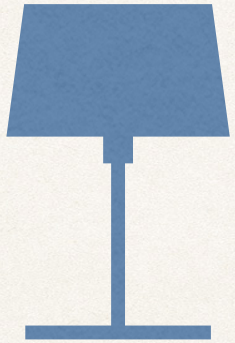
File: Dog.py

```
class Dog:

    legs: int = 4
    ears: int = 2
    type: str = 'Goldendoodle'
    age: int = 5
    color: str = 'Yellow'
```

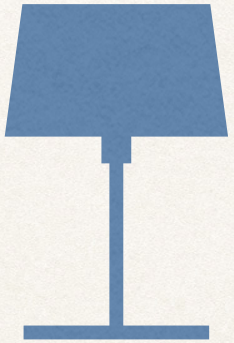
Encapsulation



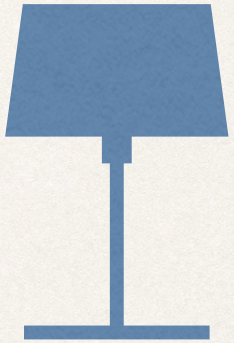
Abstraction



Inheritance



Polymorphism



© CODINGWITHROBY

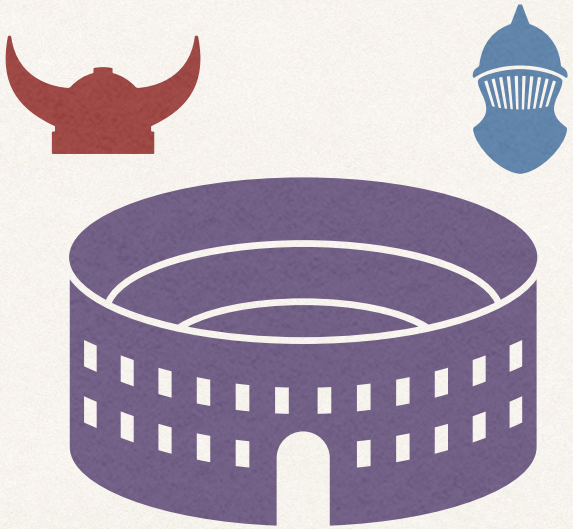
What are we creating?

What are we creating?

© CODINGWITHROBY

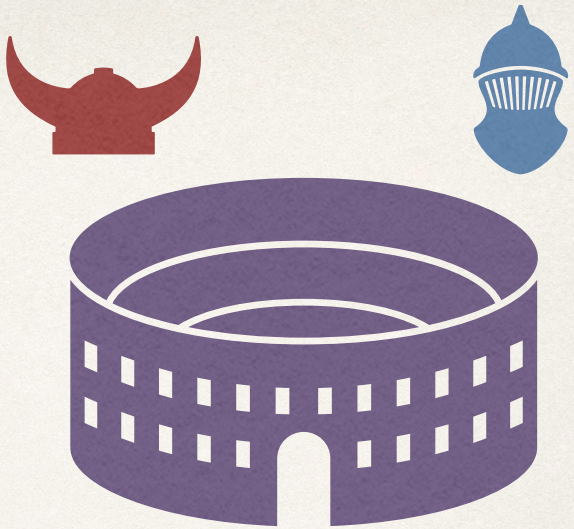
Drum roll….

- **Small game where we can create enemies that fight each other!**



© CODINGWITHROBY

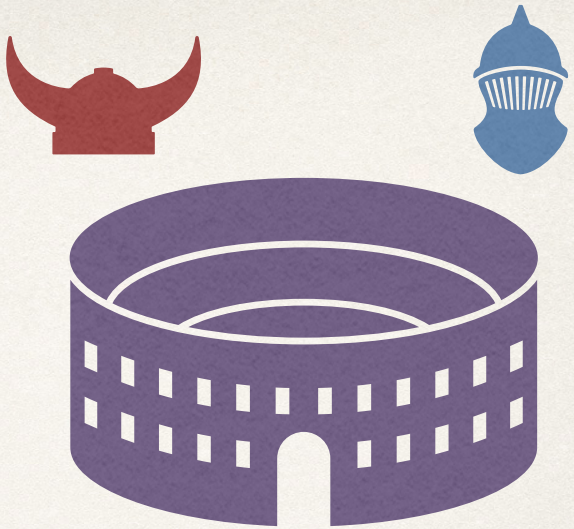
Acceptance Criteria



- Enemies that can fight one another
- Different types of enemies
  - Zombie
  - Ogre
- Each enemy has different powers, health points and attack damage

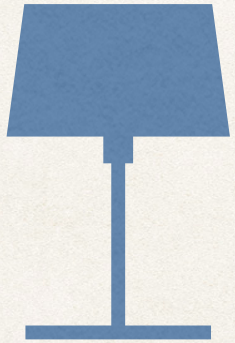
© CODINGWITHROBY

What are we creating continued

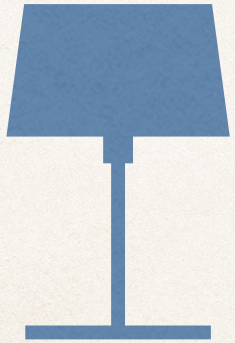


- We will be implementing the four pillars of OOP:

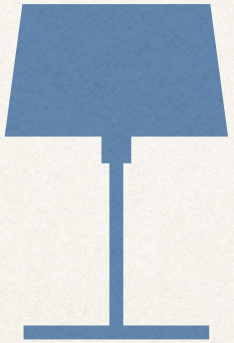
Encapsulation



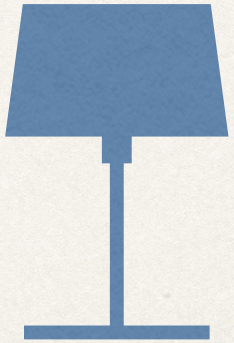
Abstraction



Inheritance

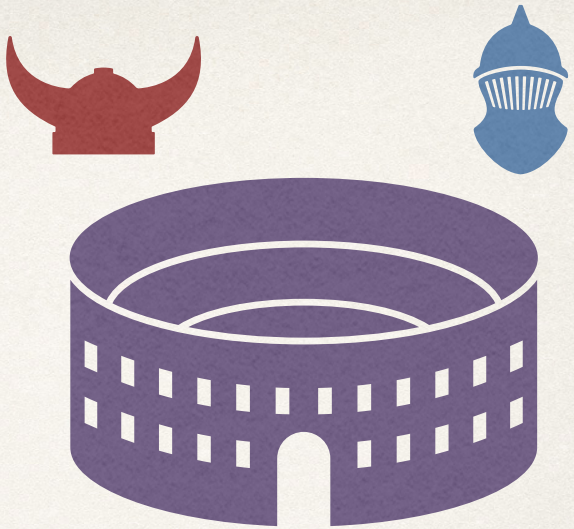


Polymorphism



© CODINGWITHROBY

What do we need to start?



- Enemy Object:
  - Name / Type of enemy
  - Health points
  - Attack damage

© CODINGWITHROBY

Enemy Object

File: Enemy.py

```
class Enemy:

                type_of_enemy: str
                health_points: int = 10
                attack_damage: int = 1
```


© CODINGWITHROBY

Enemy Object

File: Main.py

```python
from Enemy import *

enemy = Enemy()

print(f'{enemy.type_of_enemy} has {enemy.health_points} health points
    And can do attack of {enemy.attack_damage}')
```

```python
class Enemy:

    type_of_enemy: str
    health_points: int = 10
    attack_damage: int = 1
```


© CODINGWITHROBY

Enemy Object

File: Enemy.py

```python
class Enemy:

    type_of_enemy: str
    health_points: int = 10
    attack_damage: int = 1
```

File: Main.py

```python
from Enemy import *

enemy = Enemy()

print(f'{enemy.type_of_enemy} has {enemy.health_points} health points
    And can do attack of {enemy.attack_damage}')
```

Terminal:

```
'Enemy' object has no attribute 'type_of_enemy'
```


© CODINGWITHROBY

Enemy Object

```python
class Enemy:

    type_of_enemy: str
    health_points: int = 10
    attack_damage: int = 1
```

File: Main.py

```python
from Enemy import *


enemy = Enemy()

        enemy.type_of_enemy =
        'Zombie'

print(f'{enemy.type_of_enemy} has {enemy.health_points} health points
    And can do attack of {enemy.attack_damage}')
```


© CODINGWITHROBY

Enemy Object

File: Enemy.py

```
class Enemy:

        type_of_enemy: str
        health_points: int = 10
        attack_damage: int = 1
```

File: Main.py

```
from Enemy import *


enemy = Enemy()


enemy.type_of_enemy = 'Zombie'


print(f'{enemy.type_of_enemy} has {enemy.health_points} health points
    And can do attack of {enemy.attack_damage}')
```

Terminal

```
Zombie has 10 health points and can do attack of 1
```


© CODINGWITHROBY

Abstraction

What is abstraction?

© CODINGWITHROBY

What is Abstraction?

- Let’s think of flashlight
- A flashlight has a ‘on' and ‘off’ switch
- When you turn the switch ‘on’, you expect a beam of light to shoot out of the flashlight
- When the switch is ‘off’, you expect no light
- We do not know how the light inside the flashlight turns ‘on’ and ‘off’, we just know that when the flashlight is ‘on’ we expect a beam of light.

© CODINGWITHROBY

What is Abstraction?

- **Abstraction -** This means to hide the implementation and only show necessary details to the user.
- Let's say we add new code to allow the enemy to talk:

File: Enemy.py

```
class Enemy:

            type_of_enemy: str
            health_points: int = 10
            attack_damage: int = 1

            def talk(self):
                print('I am an Enemy!')
```


© CODINGWITHROBY

Now we can use Abstraction

```python
class Enemy:

        type_of_enemy: str
        health_points: int = 10
        attack_damage: int = 1

        def talk(self):
            print('I am an Enemy!')
```

File: Main.py

```python
from Enemy import *


  enemy = Enemy()

  enemy.type_of_enemy = 'Zombie'

  enemy.talk()

  print(f'{enemy.type_of_enemy} has {enemy.health_points} health points
      And can do attack of {enemy.attack_damage}')
```


© CODINGWITHROBY

Abstraction

- Make more methods for our Enemy.

File: Enemy.py

```
. . .

def talk():
    print(f'I am a {self.type_of_enemy}. Be prepared to fight.')


def walk_forward():
    print(f'{self.type_of_enemy} moves closer to you.')


def attack():
    print(f'{self.type_of_enemy} attacks for {self.attack_damage} damage')


. . .
```


© CODINGWITHROBY

Now we can use Abstraction

File: Main.py

```
from Enemy import *

  enemy = Enemy()

  enemy.type_of_enemy = 'Zombie'

  enemy.talk()

  enemy.walk_forward()

  enemy.attack()
```

Terminal

```
I am a Zombie be prepared to fight!

Zombie moves closer to you

Zombie attacks for 1 damage
```


© CODINGWITHROBY

Why use Abstraction?

- This allows users to not have to understand what the functionality is behind scenes.
- You can create simple and reusable code
- Allows for a better use of the DRY principle (Don’t repeat yourself)
- Enables Python objects to become more scalable.

© CODINGWITHROBY

Constructors

How to use constructors?

© CODINGWITHROBY

What is constructors?

- **Constructors -** Are used to create and initialize an object of a class with or without starting values.

```
from Enemy import *

  enemy = Enemy()
  enemy.type_of_enemy = 'Zombie'
  enemy.talk()
  enemy.walk_forward()
  enemy.attack()
```

Example of calling a constructor

© CODINGWITHROBY

Different types of constructors

- Default / Empty Constructors
- No Argument Constructors
- Parameter Constructors

© CODINGWITHROBY

Default / Empty Constructor

Default/Empty Constructor

File: Enemy.py

```
class Enemy:

        type_of_enemy: str
        health_points: int = 10
        attack_damage: int = 1

        def __init__(self):
            pass

        def talk(self):
            print('I am an Enemy!')
```

Example:

```
enemy = Enemy()
```

Automatically created if no constructor is found

"self" refers to the current class/ object

© CODINGWITHROBY

No Argument Constructor

File: Enemy.py

```
class Enemy:

        type_of_enemy: str
        health_points: int = 10
        attack_damage: int = 1

        def __init__(self):
            print('New enemy created with no starting values')

        def talk(self):
            print('I am an Enemy!')
```

Example:

```
enemy = Enemy()
```


© CODINGWITHROBY

Parameter Constructor

File: Enemy.py

Example:

```
enemy = Enemy('Zombie')
```

```
class Enemy:

        def __init__(self, type_of_enemy, health_points=10, attack_damage=1):
            self.type_of_enemy = type_of_enemy
            self.health_points = health_points
            self.attack_damage = attack_damage

        def talk(self):
            print('I am an Enemy!')
```

Example:

```
enemy = Enemy('Zombie', 15, 3)
```


© CODINGWITHROBY

Using parameters constructor

File: Main.py

```
from Enemy import *

enemy = Enemy('Zombie', 15, 3)

print(f'{enemy.type_of_enemy} has {enemy.health_points} health points
    And can do attack of {enemy.attack_damage}')
```

Terminal

```
Zombie has 15 health points and can do attack of 3
```


© CODINGWITHROBY

Encapsulation

What is encapsulation?

© CODINGWITHROBY

What is Encapsulation?

- **Encapsulation -** Bundling of data
- Let's take a closer look at the 3 attributes our Enemy currently has:

File: Enemy.py

```
class Enemy:
    def __init__(self, type_of_enemy, health_points=10, attack_damage=1):
        self.type_of_enemy = type_of_enemy
        self.health_points = health_points
        self.attack_damage = attack_damage

    def talk(self):
        print('I am an Enemy!')
```


© CODINGWITHROBY

What public class attributes mean?

File: Main.py

```
from Enemy import *


  zombie = Enemy('Zombie')

  zombie.type_of_enemy = 'Orc'

  zombie.talk()

  zombie.walk_forward()

  zombie.attack()
```


© CODINGWITHROBY

What is Encapsulation?

- Changes our public attributes to private

File: Enemy.py

```
class Enemy:

            def __init__(self, type_of_enemy, health_points=10, attack_damage=1):
                self.__type_of_enemy = type_of_enemy
                self.health_points = health_points
                self.attack_damage = attack_damage

            def talk(self):
                print('I am an Enemy!')
```

__ (double underscore)

© CODINGWITHROBY

What is Encapsulation?

File: Enemy.py

GETTERS & SETTERS

```
class Enemy:

            def __init__(self, type_of_enemy, health_points=10, attack_damage=1):
                self.__type_of_enemy = type_of_enemy
                self.health_points = health_points
                self.attack_damage = attack_damage

            def get_type_of_enemy(self):
                return self.__type_of_enemy

                                                        my):


            def talk(self):
                print('I am an Enemy!')
```


© CODINGWITHROBY

Using Getters & Setters

File: Main.py

```
from Enemy import *


  enemy = Enemy('Zombie', 10, 1)

  enemy.get_type_of_enemy()

  enemy.talk()

  enemy.walk_forward()

  enemy.attack()
```


© CODINGWITHROBY

Why use Encapsulation?

- Helps keep related fields and methods together
- Makes our code cleaner and easier to read
- Provides more flexibility to our code
- Provides more reusability with our code

© CODINGWITHROBY

Inheritance

What is inheritance?

© CODINGWITHROBY

What is Inheritance?

- **Inheritance -** Process of acquiring properties from one class to other classes
- Creates a hierarchy between classes

Parent Class

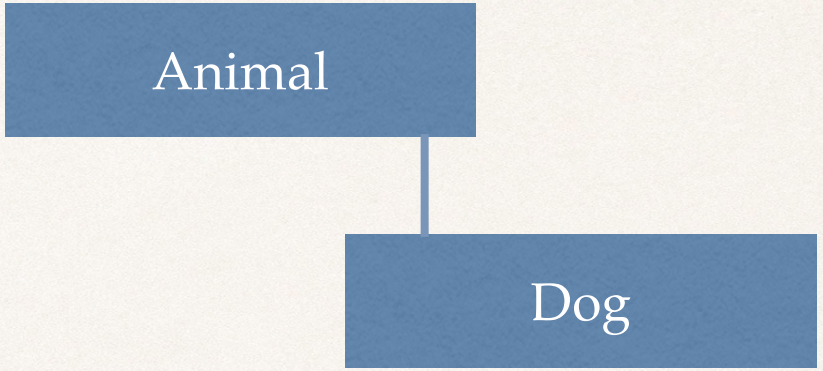
```
class Animal

        weight: int
        color: str
        age: int
        animal_type: str
```

Child Class

```
class Dog(Animal)

    ** All Animal Attributes **

        can_shed: bool
        domestic_name: str
```


© CODINGWITHROBY

Hierarchy thus far



© CODINGWITHROBY

What is Inheritance?

```
class Animal:

        weight: int
        color: str
        age: int
        animal_type: str

        def eat(self):
            print('Animal eating')

        def sleep(self):
            print('Animal sleeping')
```

```
class Dog(Animal):

        ** All Animal Attributes **

                can_shed: bool
                domestic_name: str

        ** All Animal Methods **
  }
```

```
dog = Dog()
dog.eat()
```

Animal eating

© CODINGWITHROBY

What is Inheritance?

```
class Animal:

        weight: int
        color: str
        age: int
        animal_type: str

        def eat(self):
            print('Animal eating')

        def sleep(self):
            print('Animal sleeping')
```

```
class Dog(Animal):

        ** All Animal Attributes **

                can_shed: bool
                domestic_name: str

        def talk(self):
            print('Bark!')

        ** All Animal Methods **
  }
```

```
dog = Dog()
dog.talk()
```

Bark!

```
animal = Animal()
animal.talk()
```


© CODINGWITHROBY

What is a method override?

- Our *Animal eat()* is pretty generic. What if we want to add our own *Dog eat()*?

```
class Animal:

        ** All Animal Attributes **

        def eat(self):
            print('Animal eating')

        def sleep(self):
            print('Animal sleeping')
```

```
class Dog(Animal):

        ** All Animal Attributes **

                can_shed: bool
                domestic_name: str

        def talk(self):
            print('Bark!')

        def eat(self):
            print('Chews on bone!')

        ** All Animal Methods **
  }
```

```
dog = Dog()
dog.eat()
dog.sleep()
```

Chews on bone!
Animal sleeping

© CODINGWITHROBY

What is method overriding?

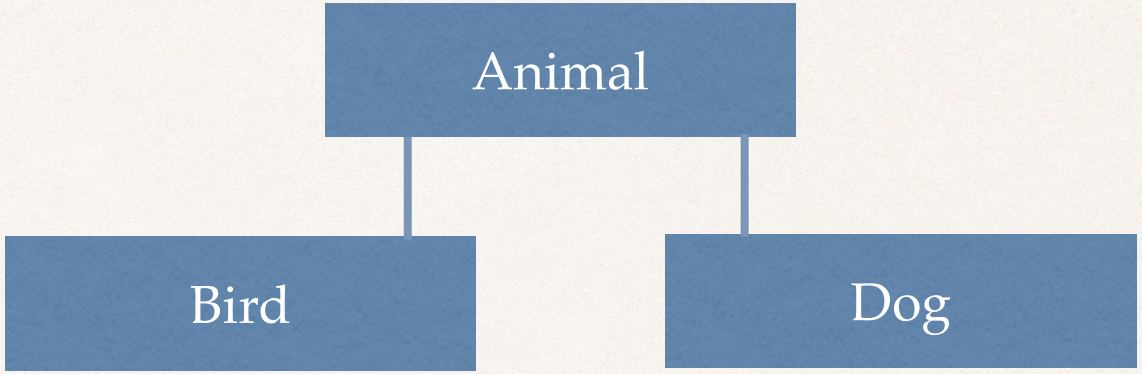
- As we can see from the last example, both Dog() and Animal() have the eat() functionality.
- Method overriding is when a child class has its own method already present in the parent class.
- When the child class does not have the same method, it will default to the parent method.

© CODINGWITHROBY

Let’s now add a Bird class

© CODINGWITHROBY

Hierarchy thus far



© CODINGWITHROBY

What is Inheritance?

```
class Animal:

        weight: int
        color: str
        age: int
        animal_type: str

        def eat(self):
       print('Animal eating')

        def sleep(self):
       print('Animal sleeping')
```

```
class Bird(Animal)

        ** All Animal Attributes **

        birdType: str

        def talk(self):
       print('Chirp!')

        def fly(self):
       print('Bird begins to soar!')

        ** All Animal Methods **
```

```
bird = Bird()
bird.talk()
bird.fly()
```

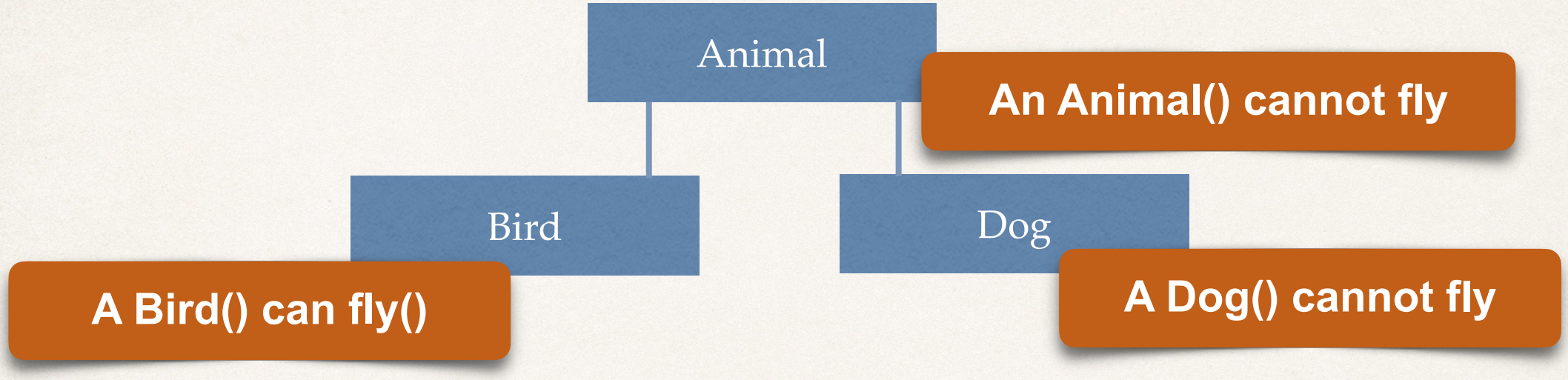
Chirp!

Bird begins to soar!

```
animal = Animal()
animal.talk()
animal.fly()
```


© CODINGWITHROBY

Hierarchy thus far



© CODINGWITHROBY

self vs super

self vs super

- **self** is used to refer to the current object that is created or being instantiated, while **super** is used to refer to the parent class.
- **self** is used when there is a need to differentiate between the instance variables & parameters with the same name, while **super** is used to call the parent class methods and/or constructors.

© CODINGWITHROBY

this vs super

```
class Person:

          def __init__(self, name, age):
             self.name = name
             self.age = age


class Student(Person):

          def __init__(self, name, age, degree):
             super().__init__(name=name, age=age)
             self.degree = degree
```

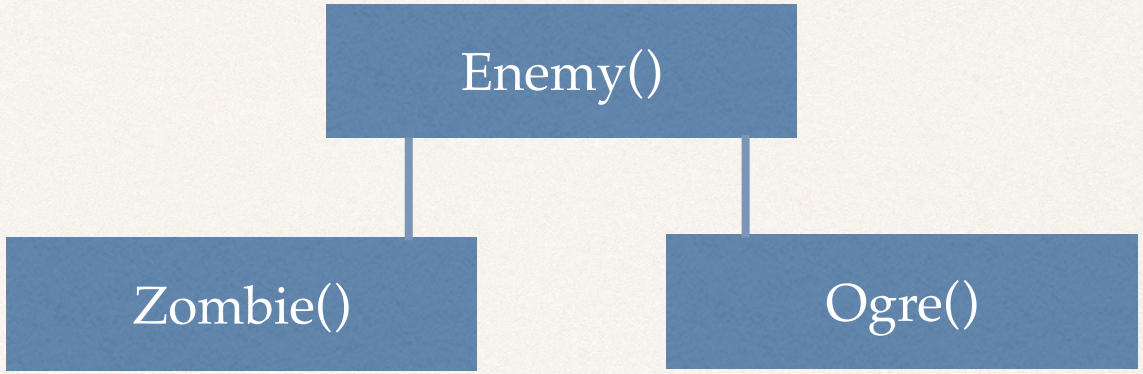
- Student has an extra property: degree
- Student uses **super** to call the Person constructor which assigns the properties **name** and **age**.
- Student uses **self** to reference the degree property which is different from the degree parameter / argument coming in.

© CODINGWITHROBY

How will we implement inheritance?

Parent and Child classes?

- Currently our only class is Enemy()
  - Enemy() is our Parent Class
- We will now create two children classes
  - Zombie()
  - Ogre()



© CODINGWITHROBY

Take a look at the Zombie Class

```
class Enemy:

  def __init__(self, type_of_enemy,    health_points,
  attack_damage):

     self.__type_of_enemy = type_of_enemy
     self.health_points = health_points
     self.attack_damage = attack_damage

  def get_type_of_enemy(self):
      return self.__type_of_enemy

  def talk(self):
      print('I am an Enemy!')

  def walk_forward(self):
      print(f'{self.__type_of_enemy} moves closer
  to you')
```

```
class Zombie(Enemy):

  def __init__(self, health_points,
  attack_damage):

       super().__init__(type_of_enemy='Zombie',
      health_points=health_points,
      attack_damage=attack_damage)


  def talk(self):
      print('*Grumbling…*')
```

Method Overriding

```
  def spread_disease(self):
      print('The zombie is trying to spread
  infection')
```

New Zombie only Method

Parent class is also called the Super class

© CODINGWITHROBY

Take a look at the Ogre Class

```
class Enemy:

  def __init__(self, type_of_enemy,    health_points,
  attack_damage):

     self.__type_of_enemy = type_of_enemy
     self.health_points = health_points
     self.attack_damage = attack_damage

  def get_type_of_enemy(self):
      return self.__type_of_enemy

  def talk(self):
      print('I am an Enemy!')

  def walk_forward(self):
      print(f'{self.__type_of_enemy} moves closer
  to you')
```

```
class Ogre(Enemy):

  def __init__(self, health_points,
  attack_damage):

       super().__init__(type_of_enemy='Ogre',
      health_points=health_points,
      attack_damage=attack_damage)


  def talk(self):
      print('Ogre is slamming hands all around.')
```

Method Overriding

Parent class is also called the Super class

© CODINGWITHROBY

Polymorphism

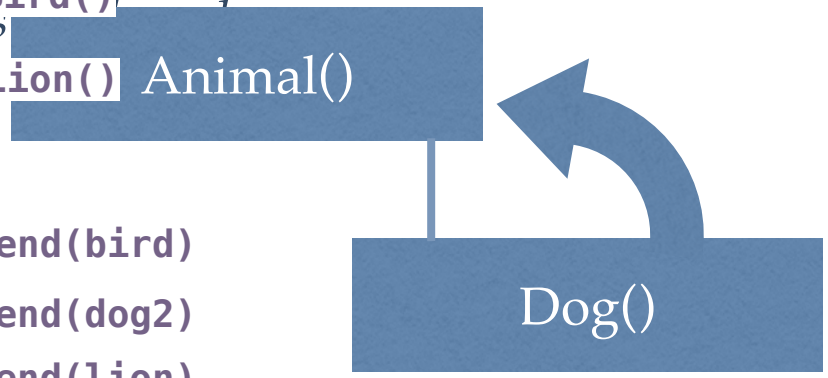
What is Polymorphism?

- **Polymorphism** means to have many forms
- Let's go back to our Animal examples:

© CODINGWITHROBY

What is Polymorphism?

```
zoo: Animal =[]

dog = Animal()
dog2 = Dog()
bird = Bird()
lion = Lion()

zoo.append(bird)
zoo.append(dog2)
zoo.append(lion)
```




```
class Dog(Animal):
        ** All Dog Methods **

class Bird(Animal):
        ** All Bird Methods **

class Lion(Animal):
        ** All Lion Methods **
```


© CODINGWITHROBY

What is Polymorphism?

```
zoo = Animal[]
dog = Animal()
dog2 = Dog()
bird = Bird()
lion = Lion()
zoo.append(dog)
zoo.append(dog2)
zoo.append(bird)
zoo.append(lion)
```

```
/* We already added some
                animals to our zoo
*/ VALIDATE THIS
```

```
for animal in zoo:
      animal.talk()
```

Bark!
Chirp!
Bark!
Roar!
Roar!

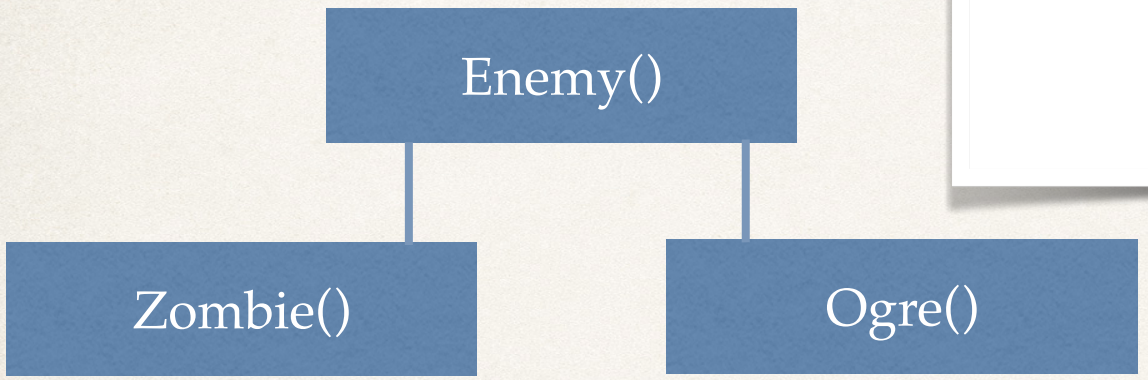
```
class Animal:

        def talk(self):
           print('Does not make a sound')


class Dog(Animal):

        def talk(self):
           print('Bark!')


class Bird(Animal):

        def talk(self):
           print('Chirp!')


class Lion(Animal):

        def talk(self):
           print('Roar!')
  }
```


© CODINGWITHROBY

How will we implement Polymorphism?

Starting off..

- Create a new battle function within our main.py file
  - Uses our enemy talk() and attack() methods



File: Main.py

```
def battle(Enemy e):
        e.talk()
        e.attack()
```


© CODINGWITHROBY

Now we can use Polymorphism

File: Main.py

```
def battle(Enemy e) {
        e.talk()
        e.attack()

zombie = Zombie(10, 1)

ogre = Ogre(20, 3)

battle(zombie)
battle(ogre)
```

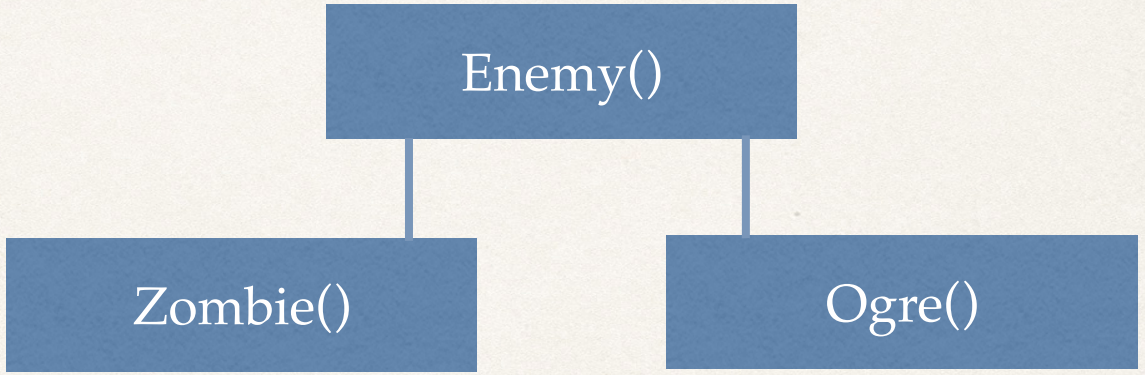
Grumbling…
Zombie attacks for 1 damage
Ogre is slamming his hands all around
Ogre attacks for 3 damage

© CODINGWITHROBY

Time for a battle!

Starting off..

- Before we start a battle:
  - Let's give our Enemies a special attack.
- If the Enemy does not have a special attack, we print "Enemy does not have a special attack."
- Add code into the:
  - Enemy (Parent)
  - Both Zombie and Ogre



© CODINGWITHROBY

Take a look at the Enemy Parent Class

```
class Enemy:

  def __init__(self, type_of_enemy,    health_points,
  attack_damage):

     self.__type_of_enemy = type_of_enemy
     self.health_points = health_points
     self.attack_damage = attack_damage

  def get_type_of_enemy(self):
      return self.__type_of_enemy

  def talk(self):
      print('I am an Enemy!')

  def walk_forward(self):
      print(f'{self.__type_of_enemy} moves closer
  to you')


  def special_attack(self):
      print('Enemy has no special attack')
```

- New **special_attack()** method that enemies can default to when they do not have a special attack.

© CODINGWITHROBY

Take a look at the Zombie & Ogre Class

```
class Zombie(Enemy):

  def __init__(self, health_points, attack_damage):

        super().__init__(type_of_enemy='Zombie',
       health_points=health_points,
       attack_damage=attack_damage)


  def talk(self):
      print('*Grumbling…*')

  def spread_disease(self):
      print('The zombie is trying to spread infection')

  def special_attack(self):
      did_special_attack_work = random.random() < 0.50
      if did_special_attack_work:
           self.health_points += 2
           print('Zombie regenerated 2 HP!')
```

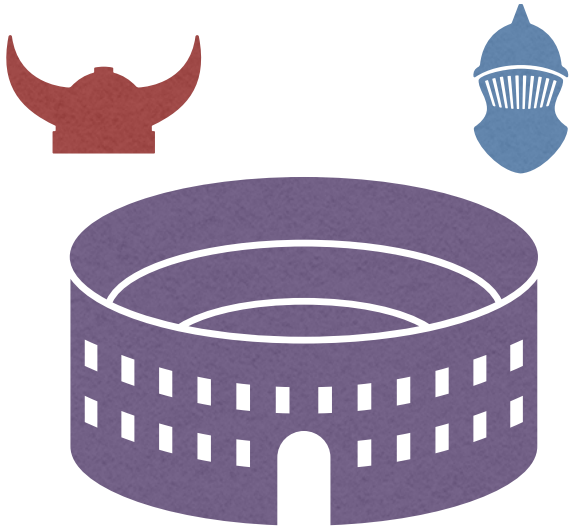
```
class Ogre(Enemy):

  def __init__(self, health_points, attack_damage):

        super().__init__(type_of_enemy='Ogre',
       health_points=health_points,
       attack_damage=attack_damage)


  def talk(self):
      print('Ogre is slamming hands all around.')

  def special_attack(self):
      did_special_attack_work = random.random() < 0.20
      if did_special_attack_work:
           self.attack_damage += 4
           print('Ogre attack has increased by 4!')
```


© CODINGWITHROBY

Now we can enhance our main file

File: Main.py

```
def battle(e1: Enemy, e2: Enemy) {
          e1.talk()
          e2.talk()

          while e1.health_points > 0 and e2.health_points > 0
             e1.special_attack()
             e2.special_attack()
             e2.attack()
             e1.health_points -= e2.attack_damage
             e1.attack()
             e2.health_points -= e1.attack_damage

          if e1.health_points > 0:
       print("Enemy 1 wins!")
          else:
       print("Enemy 2 wins!")


zombie = Zombie(10, 1)
ogre = Ogre(20, 3)

battle(zombie, ogre)
```



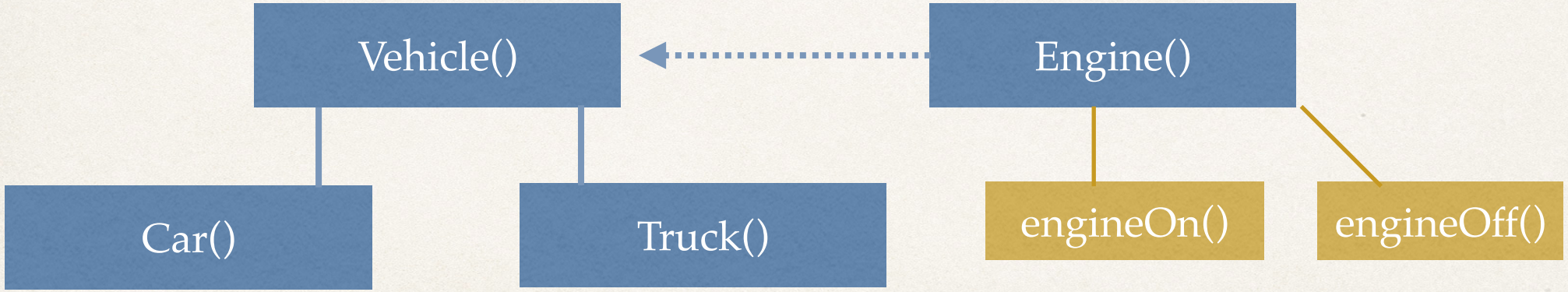
© CODINGWITHROBY

Composition

What is a composition?

- A way to create objects made up of other objects
- In composition, a class contains one or more objects of another class as instance variables.
- Provide layered functionality to the object
- Known as a HAS-A relationship

```
car.getEngine().engineOn()
```



© CODINGWITHROBY

What is composition

```python
class Engine:

        def __init__(self, engineType):
            self.engineType = engineType

        def startEngine(self):
            print("Engine is running")

        def stopEngine(self):
            print("Engine is off")
```

```python
class Vehicle:

        def __init__(self, type, forSale, engine):
            self.type = type
            self.forSale = forSale
            self.engine = engine
```


© CODINGWITHROBY

What is composition

```
engine = Engine("V6")
vehicle = Vehicle("Car", True, engine)
vehicle.engine.startEngine()
```

Engine is running

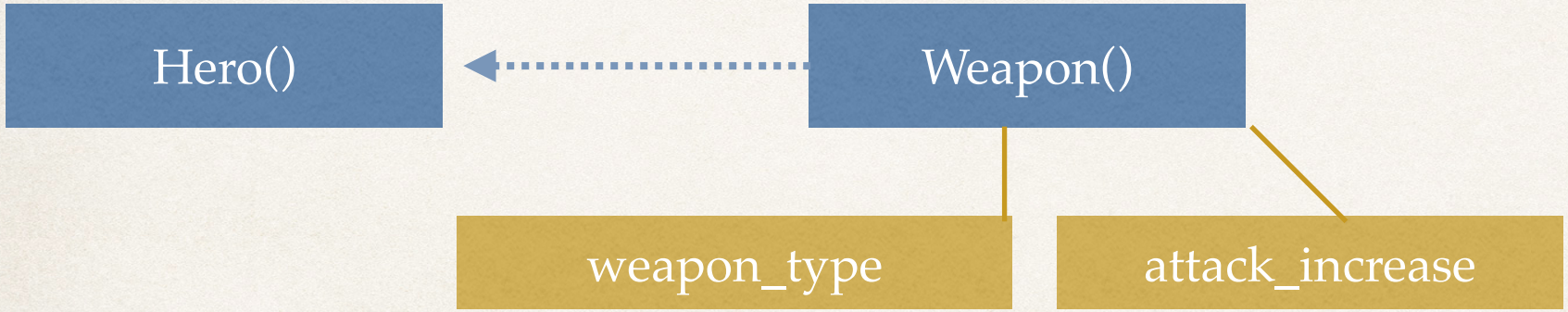
- Using composition here because the vehicle HAS-A engine.
- This is different from an interface IS-A relationship.
- A vehicle must have an engine, but an engine does not need to have a vehicle.

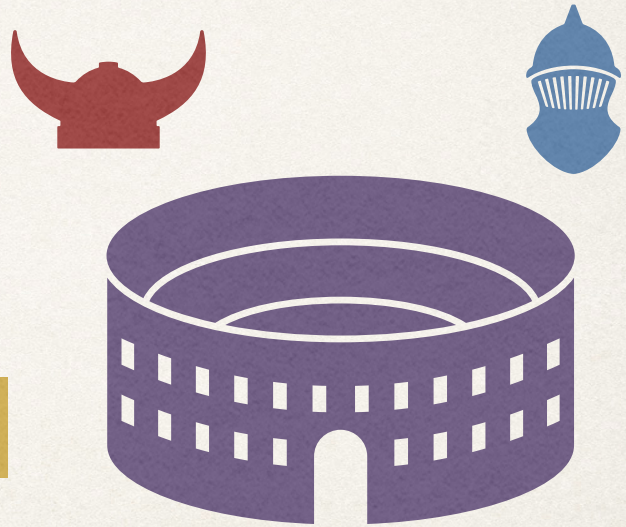
© CODINGWITHROBY

Implementation of Composition

Composition implementation?

- We will create a new Hero class
- We will create a new Weapon class
- Our Hero will have a HAS-A (composition) relationship with our Weapon class

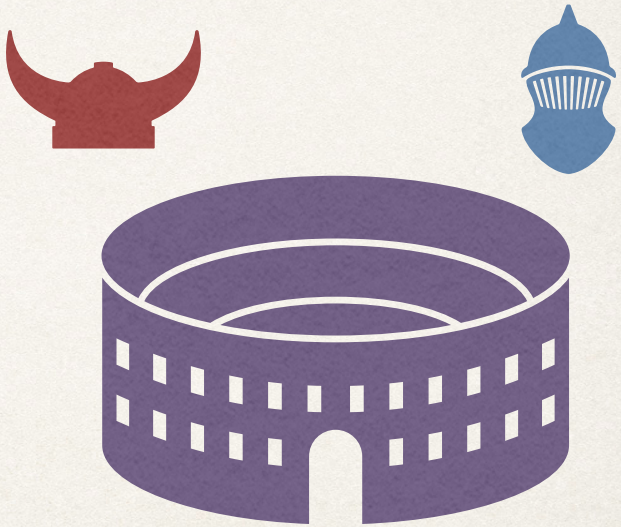




© CODINGWITHROBY

Composition implementation

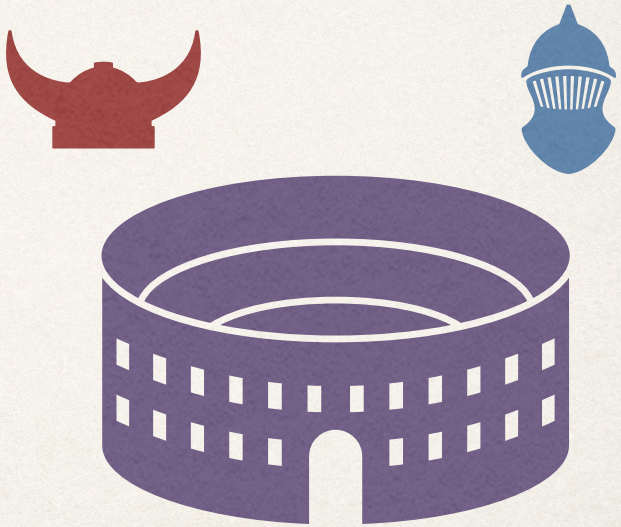
```
class Weapon:

          def __init__(self, weapon_type, attack_increase):
             self.weapon_type = weapon_type
             self.attack_increase = attack_increase
```



© CODINGWITHROBY

Composition implementation

```
class Hero:

        def __init__(health_points, attack_damage):
            self.health_points = health_points
            self.attack_damage = attack_damage
            self.is_weapon_equipped = False
            self.weapon: Weapon = None


        def equipWeapon(self)
            if self.weapon is not None and not self.is_weapon_equipped:
                    self.attack_damage += self.weapon.attack_increase
                    self.is_weapon_equipped = True
```

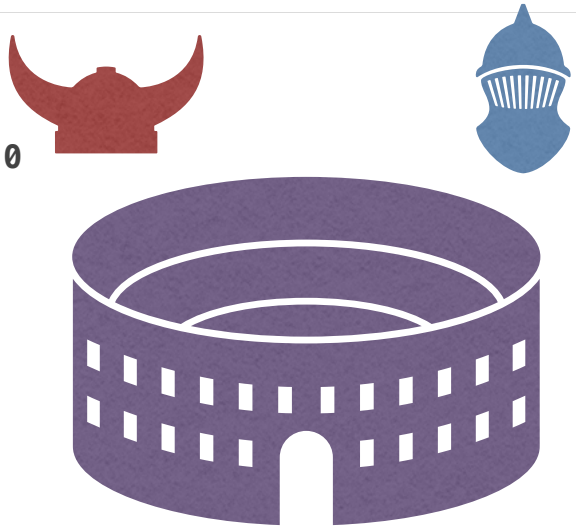


© CODINGWITHROBY

Now we can enhance our main file

File: Main.py

```
def hero_battle(hero: Hero, enemy: Enemy) {

          while hero.health_points > 0 and enemy.health_points > 0
             enemy.special_attack()
             enemy.attack()
             hero.health_points -= enemy.attack_damage
             hero.attack()
             enemy.health_points -= hero.attack_damage

          if hero.health_points > 0:
       print("Hero wins!")
          else:
       print("Enemy 2 wins!")


zombie = Zombie(10, 1)
hero = Hero(10, 1)
weapon = Weapon('Sword', 5)
hero.weapon = weapon
hero.equip_weapon()
hero_battle(hero, zombie)

battle(zombie, ogre)
```



© CODINGWITHROBY