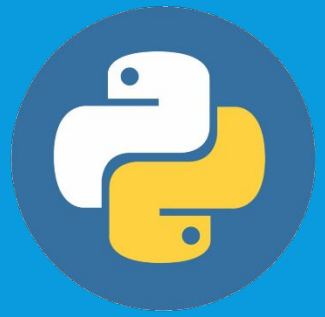
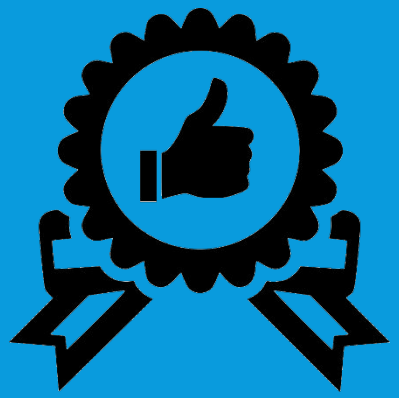
PYTHON FOR DUMMIES

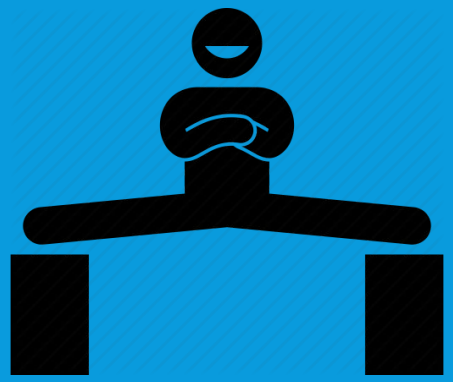
Basics of Django: Session #1



INTRODUCTION

- Python is one of the most popular open source programming languages today
- Easy to learn and easy to build more functions with less lines of code
- Flexible integration with many programming languages
- High availability of resources with an extensive library and built-in functions
- Robust available environments and frameworks





PYTHON & POPULAR FRAMEWORKS

- Django
- Flask
- Pyramid
- Falcon

Check out these links for more information!

DJANGO FOR WEB DEVELOPMENT

We will build a simple quiz app with Django during this session.

Before the session:

- Ensure you have Python installed

  If using VS Code, make sure you have the Python extension installed

- Set up your virtual environment that you'll be working in

  mkdir quiz-app

  cd quiz-app

  python3 –m venv virtualenv

  source virtualenv/bin/activate

- Install Django: python3 –m pip install Django. Click here for Windows

TOPICS TO BE COVERED

- Model-View-Template Architecture
- URL Routing
- Django's Object Relational Mapper (ORM)

Models

  Making queries

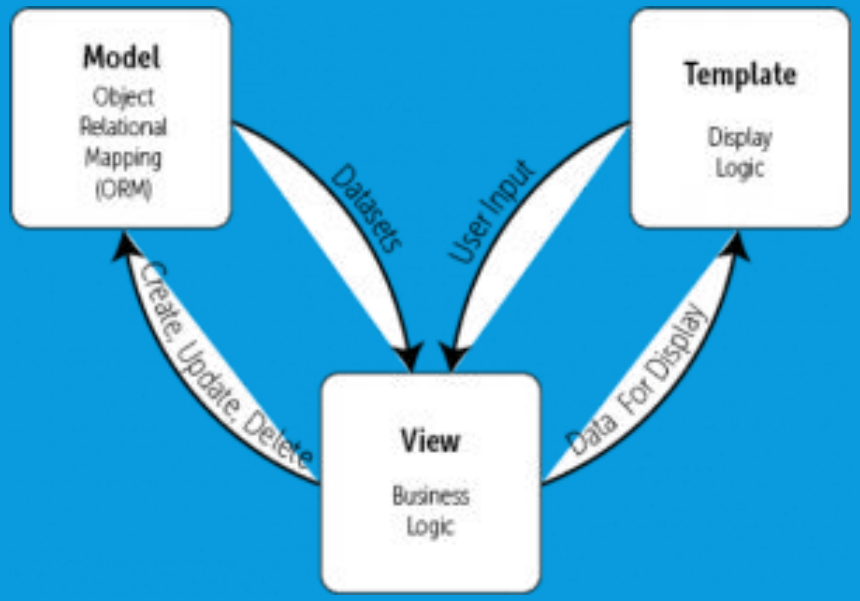
  Searching/Filtering

- Schema migrations
- Limitations

GENERAL STRUCTURE & FLOW

- A project is split into apps that handles its own function
- Model-View-Template architecture is used - the controller functionality is handled by Django
- The general flow of a Django project:
  - User's browser prompts for a URL
  - This request gets matched with a URL that is found in the urls.py files
  - Moves to the view that is associated with the URL, found in views.py
  - The particular view grabs data from model
  - The view returns this data to a template / returns an exception

M-V-T


Model
Object
Relational
Mapping
(ORM)
Template
Display
Logic
Datasets
User Input
Create, Update, Delete
Data For Display
View
Business
Logic

STARTING THE PROJECT

- Create the project and the apps:
- django-admin startproject quiz_app
- Optional: from quiz-app, run mv quiz_app/manage.py ./
- mv quiz_app/quiz_app/* quiz_app
- rm –r quiz_app/quiz_app (restructuring the layout of the project)

- **python3 manage.py runserver**

STARTING THE APP

- python3 manage.py startapp quizzes
- add 'quizzes' onto your INSTALLED_APPS on settings.py
- Make a static folder where we will keep our base css and images for our app
- E.g mkdir quizzes/static/
- mkdir quizzes/static/img/
- Add the attached img files onto the img folder
- touch quizzes/static/style.css

STARTING THE APP

- Make a base template called template.html that we can inherit for our other templates for our project
- mkdir quiz_app/templates
- touch quiz_app/templates/template.html

TEMPLATES

- Configure template engine with the TEMPLATES setting in settings.py
- DIRS: a list of directories to look for template source files
- Ensuring a D-R-Y Methodology throughout the app

STARTING THE APP

- Copy paste this code in template.html onto your template.html
- Copy paste this css code onto style.css in static folder
- Go to your settings.py and add "quiz_app/templates/" in DIRS

MODELS

- Each model maps to a table in the database
- Each field in the table is represented by an attribute in the model
- Each model is a class that subclasses Django.db.models.Model
- Each field is an instance of the appropriate Field class
- There are many built-in field types that Django supports

MODELS

- Copy paste this code for models.py

```python
from django.db import models

# Create your models here.
class Category(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField()
    img = models.FilePathField(path="/img")

#ForeignKey defines a many to one relationship
class Quiz(models.Model):
    category = models.ForeignKey(Category, on_delete = models.CASCADE)
    name = models.CharField(max_length=255)

class Question(models.Model):
    quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE) #many question instances in one quiz
    description = models.CharField(max_length=255)
class Answer(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE, related_name='answers') #many answers to a question
    text = models.CharField(max_length=255)
    is_answer = models.BooleanField(default=False)
```

MODELS

- To start creating our database, we start migration. This should create quizzes/migrations/0001_initial.py:
- python3 manage.py makemigrations quizzes
- Apply migrations and create database:
- python3 manage.py migrate quizzes

```
(virtualenv) Rupsis-MacBook-Air:quiz-app Rupsi$ python3 manage.py makemigrations
 quizzes
Migrations for 'quizzes':
  quizzes/migrations/0001_initial.py
    - Create model Category
    - Create model Quiz
    - Create model Question
    - Create model Answer
(virtualenv) Rupsis-MacBook-Air:quiz-app Rupsi$ python3 manage.py migrate quizze
s
Operations to perform:
  Apply all migrations: quizzes
Running migrations:
  Applying quizzes.0001_initial... OK
```

DJANGO'S ORM

- python3 manage.py shell
- from quizzes.models import Category, Quiz, Question, Answer
- **Create categories:**
- cat_one = Category(name= "Data Structures and Algorithms", description= "A quiz testing your knowledge on data structures such as arrays, linked lists…", img="img/data-structure.png")
- cat_one.save()
- cat_two = Category(name="Artificial Intelligence", description="A quiz testing your knowledge on various AI Components", img="img/ai.png")
- cat_two.save()

DJANGO'S ORM

- **Create quizzes:**
- c = Category.objects.get(pk=1)
- q_one = Quiz(category = c, name = "Arrays")
- q_one.save()
- **Even better:** d = Category.objects.get(pk=2)
- d.quiz_set.create(name="Stacks")
- **Create questions:** q = Quiz.objects.get(pk=1), q.question_set.create(description="What is the simplest type of array?")
- **Create answers:** a = Question.objects.get(pk=1), a.answers.create(text = "Linear Array", is_answer = True), a.answers.create(text="Multidimensional Array", is_answer = False)
- Click here for more information on model API reference

VIEWS

- A view returns a HttpResponse object containing the data or an exception
- For example, add this code to your views.py

```
def category_view(request):
    categories = Category.objects.all()
    main_categories = {
        'categories' : categories
        }
    return render(request, 'quizzes_view.html', main_categories)
```

URL ROUTING

- We need to create our app specific views by adding to templates for quizzes app
- mkdir quizzes/templates
- touch quizzes/templates/quizzes_view.html
- Repeat for categories_view, and question_view
- Copy the code found here onto your respective views
- Add an urls.py file to store all urls inside the quizzes app: touch quizzes/urls.py
- In your urls.py in quizzes, write this code fragment
- Go to your project urls (quiz_app > urls.py) and include this configuration right below admin path
- Python3 manage.py runserver

QUICK LOOK AT ADMIN INTERFACE

- Go to your localhost:8000/admin
- You will be prompted for user name and password, which we will create
- In quiz-app, run python3 manage.py createsuperuser
- Add username, email, and password
- You are now able to log in
- To be able to register and manage your models, go to your admin.py and include the models you want
- from .models import Question, Quiz
- admin.site.register(Question) and same with quiz

ADVANTAGES

- Scalable
- Readable
- Fast
- MVC Support
- Security features
- Community support

LIMITATIONS

- Use of regex to route URLS which can become convoluted
- Seems like a lot of stuff included for small projects
- All models are in one file (can become tightly coupled)

QUESTIONS?

REFERENCES

- https://docs.djangoproject.com/en/2.2/
- https://www.tutorialspoint.com/django/django_overview.htm