📘 Python + Django CRUD Notes

🔹 Python Basics

1. Introduction to Python

- Python is a high-level, interpreted programming language.
- It supports object-oriented, functional, and procedural programming.
- Features simple syntax close to English, making it beginner-friendly.
- Applications: Web Development (Django, Flask), Data Science, Machine Learning, Automation, Scripting, and more.

Example:

```
print("Hello, Python!")
```

---

2. Comments in Python

- **Comments** are non-executable lines in the program. Used to explain code.
- Improves code readability.

Types:

- Single-line → starts with `#`
- Multi-line → enclosed in triple quotes `""" ... """`

Example:

```
# This is a single-line comment
print("Hello")  # Inline comment

"""
This is a
```

multi-line comment
"""

---

3. Variables

- Variables are containers for storing data.
- Python is **dynamically typed** → no need to declare type.
- Variable name must start with letter or underscore.

Example:

```
x = 10       # integer
name = "John" # string
price = 99.5  # float
is_valid = True # boolean
```

---

4. Data Types

- Built-in types:
  - Numeric → `int`, `float`, `complex`
  - Sequence → `str`, `list`, `tuple`
  - Set → `set`, `frozenset`
  - Mapping → `dict`
  - Boolean → `True`, `False`

Examples:

```
# Numbers
age = 25          # int
pi = 3.14         # float
z = 2 + 3j        # complex

# String
text = "Python"
```

```
# List (mutable)
fruits = ["apple", "banana", "cherry"]

# Tuple (immutable)
colors = ("red", "green", "blue")

# Set (unique values)
numbers = {1, 2, 3, 3}

# Dictionary (key-value pairs)
student = {"name": "John", "age": 21}
```

---

5. Conditional Statements

- Used to make decisions in code.
- Executes different blocks of code based on conditions.

Example:

```
x = 20
if x > 10:
    print("Greater than 10")
elif x == 10:
    print("Equal to 10")
else:
    print("Less than 10")
```

---

6. Loops

- **For Loop** → iterate over sequence or range.
- **While Loop** → repeat until condition is false.

For Loop Example:

```
for i in range(5):
    print("Iteration:", i)
```

While Loop Example:

```
n = 1
while n <= 5:
    print("Count:", n)
    n += 1
```

---

7. Functions

- Functions = block of code that performs a task.
- Helps reusability and clean structure.

Syntax:

```
def function_name(parameters):
    # body
    return value
```

Example:

```
def greet(name):
    return "Hello, " + name

print(greet("Valluvan"))
```

---

8. Modules

- A module = Python file with reusable functions, variables, or classes.
- Built-in modules (math, random, os, sys) or custom modules.

Built-in module example:

```
import math
print(math.sqrt(16))  # 4.0
print(math.factorial(5))  # 120
```

Custom module example:

```
# mymodule.py
def add(x,y):
    return x+y
```

```
# main.py
import mymodule
print(mymodule.add(5,3)) # 8
```

📘 Django CRUD Notes

---

1. Introduction to CRUD in Django

- **CRUD** → Create, Read, Update, Delete.
- Used in almost all applications (student management, blogs, e-commerce).
- Django provides an easy way with **Models**, **Forms (ModelForms)**, **Views**, **Templates**, **URLs**, and **Static Files (CSS/JS)**.

---

2. Setup

```
# Create virtual environment
python -m venv venv
venv\Scripts\activate     # Windows
source venv/bin/activate  # Mac/Linux

# Install Django
pip install django

# Create project and app
django-admin startproject myproject
cd myproject
python manage.py startapp students
```

---

3. Register App in settings.py

myproject/settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'students',   # register your app here
]
```

---

4. Model

students/models.py

```
from django.db import models

class Student(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    age = models.IntegerField()

    def __str__(self):
        return self.name
```

Run Migrations:

```
python manage.py makemigrations
python manage.py migrate
```

---

5. ModelForm

students/forms.py

```
from django import forms
from .models import Student

class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ['name', 'email', 'age']
```

---

6. Views (CRUD)

students/views.py

```
from django.shortcuts import render, redirect
from .models import Student
from .forms import StudentForm

# Read
def student_list(request):
    students = Student.objects.all()
    return render(request, 'student_list.html', {'students': students})

# Create
def add_student(request):
    if request.method == "POST":
        form = StudentForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('student_list')
    else:
        form = StudentForm()
    return render(request, 'add_student.html', {'form': form})

# Update
def edit_student(request, id):
    student = Student.objects.get(id=id)
    if request.method == "POST":
        form = StudentForm(request.POST, instance=student)
        if form.is_valid():
            form.save()
            return redirect('student_list')
    else:
        form = StudentForm(instance=student)
    return render(request, 'edit_student.html', {'form': form})

# Delete
def delete_student(request, id):
    student = Student.objects.get(id=id)
    student.delete()
    return redirect('student_list')
```

---

7. URLs

myproject/urls.py

```
from django.contrib import admin
from django.urls import path
from students import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.student_list, name='student_list'),
    path('add/', views.add_student, name='add_student'),
    path('edit/<int:id>/', views.edit_student, name='edit_student'),
    path('delete/<int:id>/', views.delete_student, name='delete_student'),
]
```

---

8. Templates

student_list.html

```
{% load static %}
<link rel="stylesheet" href="{% static 'css/style.css' %}">

<h2>Students List</h2>
<a href="{% url 'add_student' %}">Add New Student</a>
<ul>
{% for s in students %}
  <li>{{ s.name }} - {{ s.email }} - {{ s.age }}
      <a href="{% url 'edit_student' s.id %}">Edit</a>
      <a href="{% url 'delete_student' s.id %}">Delete</a>
  </li>
{% endfor %}
</ul>
```

add_student.html

```
{% load static %}
<link rel="stylesheet" href="{% static 'css/style.css' %}">

<h2>Add Student</h2>
<form method="POST">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Save</button>
</form>
```

edit_student.html

```
{% load static %}
<link rel="stylesheet" href="{% static 'css/style.css' %}">

<h2>Edit Student</h2>
<form method="POST">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Update</button>
</form>
```

---

9. Static Files & CSS Setup

Step 1: Create static folder

```
myproject/
│── manage.py
│── myproject/
│   ├── settings.py
│   ├── urls.py
│── students/
│   ├── models.py
│   ├── views.py
│   ├── forms.py
│── templates/
│   ├── student_list.html
│   ├── add_student.html
│   ├── edit_student.html
│── static/
   └── css/
      └── style.css
```

Step 2: Configure static files in settings.py

```
STATIC_URL = 'static/'
STATICFILES_DIRS = [
   BASE_DIR / "static"
]
```

Step 3: Example CSS file (static/css/style.css)

```
body {
   font-family: Arial, sans-serif;
   background-color: #f9f9f9;
   margin: 20px;
}
```

```
h2 {
    color: #333;
}

a {
    margin-right: 10px;
    text-decoration: none;
    color: blue;
}

button {
    background: green;
    color: white;
    padding: 5px 10px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}
```

🔑 Key Points

- CRUD = backbone of Django applications.
- ModelForm reduces repetitive code.
- App must be registered in settings.
- URLs directly mapped to views.
- Static files allow us to add CSS/JS for styling.
- Python basics + Django CRUD + Static/CSS → complete foundation for beginners.