```
print(f" Team: {teambd.name}")
print(f" captain: {teambd. captain}")
print ("Squad:")
for player in teambd. get_squad ():
    print (f"-{player}")
```

Django

1. Write down the step to create a django project named LibraryManagement System.

Solution:

```
pip install virtualenv
~~venv/~~
python -m virtaulenv venv
venv\scripts\activate
pip install django
django-admin startproject LibraryManagement .
python manage. py runserver
```

2. Create models named Book, Library.

*Book model will have attribute such as title, author, quantity, price, category.

* Library model will have attribute such as: book(foreign key), address.

Solution:

(I) python manage. py startapp Library

(II) Library(app) → models. py →

```
class Book (models. Model):
    title= models. CharField(max_length= 200, null=True,
                                      blank= True)
    author= models. CharField(max_length= 200, null=True,
                                        blank= True)
    price= models. IntergerField(null=True, blank=True)
    quantity = models. IntegerField (null= True, blank=True)
    category= models. charField (max_length= 300)

    def __str__(self):
        return self. title

class  Library(models. Model):
    book= models. Foreignkey(Book, on_delete=models.
                                          CASCADE)
    address= models. charField (max_length=200, null= True,
                                          blank= True).

    def __str__(self):
        return self. book. title
```

admin. py →

```
 from .models import *
 admin. site. register([Book, Library])
```

Terminal →

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

Read Data:

.views.py →

```
from .models import *

 def home(request):
     book= Book.objects.all()
     context= {'books': books}
     return render(request, ~~tep~~template_name='home.html',
                                  context = context)
```

~~Ɫ~~ Create a directory called "template".

settings.py →

```
TEMPLATE = [BASE_DIR, 'template']
```

home.html →

```
<body>
{% for book in books %}
        <h1> {{ book.title}} </h1>
        <h3> {{book.author}} </h3>

   {% endfor%}
</body>
```

urls.py →

```
from Library import views as ~~l_~~ l_views
urlspatterns=[
   path=('home/', l_views.home, name='home')
   ]
```

~~ER~~

Create

Quick steps:

$$\begin{tikzcd}
1.\ \boxed{\text{forms.py}} \arrow[d] \\
2.\ \boxed{\text{html for form}} \arrow[d] \\
3.\ \boxed{\text{views.py}} \arrow[d] \\
4.\ \boxed{\text{urls.py}}
\end{tikzcd}$$

*open a python file name "forms" in the app.

Step:1

forms.py →

```
from .models import *
from django.form import ModelForm

 class BookForm ( ModelForm):
     Class Meta:
         model= Book
          fields ='__all__'
```

Step:2

```
<form method='POST' encrypt ="multipart/form-data">
   {% csrf_token %}
   {{ form.as-p}}
 </form>
```

Step:3

views.py →

```
from .forms import *

def createBook(request):
    form= BookForm()
    if request.method == 'POST':
        form= BookForm(request.POST, request.FILES)

        if form.is_valid():
            form save()
            return redirect('home')

    context= {'form': form}
    return render(request, "BookForm.html", context)
```

Step:4

urls.py →

```
path('upload/', l_views.CreateBook, name='upload')
```

Update:

Quick steps:

1. $\boxed{\text{views.py}}$
2. $\boxed{\text{urls.py}}$

Step:1

views.py →

```
from .forms import *

def updateBook (request, id):
    book= Book.objects.get(PK=id)
    form= BookForm (instance=book)
    if request.method == 'POST':
        form=BookForm(request.POST, request.
                 FILES, instance=book)
        if form.is_valid():
            form.save()
            return redirect('home')
        :
        content = {'form': form}
        return render(request, "BookForm.html", context)
```

Step:2

urls.py →

```
path('update/<str:id>', l_view.updateBook, name='update')
```

~~#~~Delete:

Quick steps:

$$\begin{tikzcd} 1.\ \boxed{\text{html}} \arrow[d] \\ 2.\ \boxed{\text{views.py}} \arrow[d] \\ 3.\ \boxed{\text{urls.py}} \end{tikzcd}$$

Step:1

deleteBook.html →

```
<form method='POST', encrypt="multipart/
                               form-data">
{% csrf_token %}
~~{{f~~
<a href="{% url 'home' %} >
        NO </a>
  <input type='submit'>

  </form>
```

Step:2

view.py →

```
from forms import *
def Delete_Book(request, id):
    book = Book.objects.get(PK=id)
    if request.method == 'POST':
        if form.is_valid()
            book.delete()
            return redirect('home')
    return render(request, "deleteBook.html")
```

Step:3 urls.py →

```
path('delete/<str:id>', l_view.DeleteBook, name='delete')
```

MVT

MVT (Model-View-Template) is a software design pattern commonly used with Django, a high-level Python web framework. ~~It~~

How it works :-

The MVT (Model-View-Template) architecture in Django helps organize a web application into three parts. The Model handles the data and interacts with the database. The View controls what happens when a user makes a request - it gets data from the model and chooses what to show. The Template is the HTML page that shows the data to the user. When someone visits a website, Django sends the request to the view, the view gets data from the model, and then shows it using the template. This makes the code clean and easy to manage.

Example:

~~Let~~ Suppose, we are building a website that shows a list of restaurants.

Firstly, user open the website, ~~and goes~~ the View receives the request and decides what to show. It asks the model to get the needed data from the database, like a list of restaurants. Then, this data is sent to the Template, which is a web page layout. The ~~t~~ template ~~files~~ fills in the data ~~creates~~ and creates a full webpage. Finally, this page is sent back to the user's browser. This setup keeps the work divided - models handle data, views handle logic and templates handle what the user sees - making the website easier to manage.