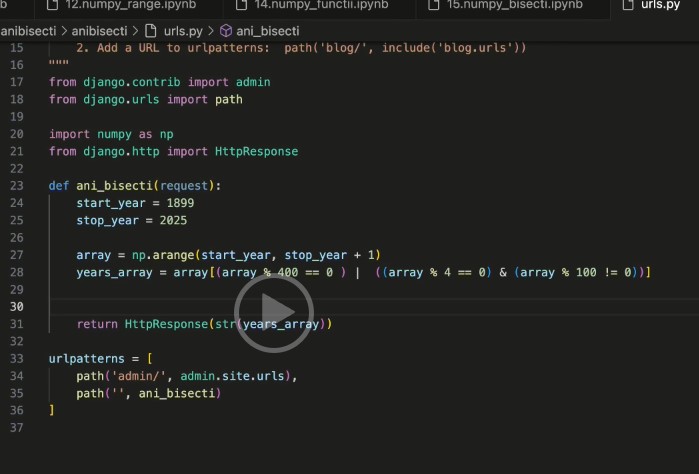
1. Folder -> code .. 🡪 python manage.py runserver

Data Analysis and Processing using Python 1/7

7Mar2025

Urls.py



Learned:

from django.contrib import admin

from django.urls import path

def ani\_bisecti(request, start\_year = 1899, stop\_year = 2025):

def F. (**request**, …):

li\_years = "\n".join(map(lambda x: f"<li>{x}</li>" ,years\_array))

you create a list, then can return it

return HttpResponse(f"<ol>{li\_years} </ol>")

return HttpResponse(f”http-can-go-here **{**list\_example**}** ”

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', ani\_bisecti), # se apeleaza functia cu https://127.0.0.1:8000

    path('<int:start\_year>/<int:stop\_year>', ani\_bisecti), # se apeleaza cu https://127.0.0.1:8000/1899/2025]

can return the input that is given at end of url via path3

from django.contrib import admin

from django.urls import path

import numpy as np

from django.http import HttpResponse

def ani\_bisecti(request, start\_year = 1899, stop\_year = 2025):

    # Construirea range-ului: ex: anii de la 1899 la 2025

    array = np.arange(start\_year, stop\_year + 1)

    # filtrarea anilor bisecti

    years\_array = array[(array % 400 == 0 ) |  ((array % 4 == 0) & (array % 100 != 0))]

    # fiecare an bisect va fi pus intre <li> </li>

    li\_years = "\n".join(map(lambda x: f"<li>{x}</li>" ,years\_array))

    return HttpResponse(f"<ol>{li\_years} </ol>")

urlpatterns = [

    path('admin/', admin.site.urls),

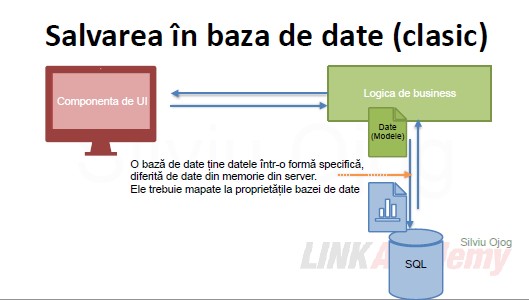
    path('', ani\_bisecti), # se apeleaza cu https://127.0.0.1:8000

    path('<int:start\_year>/<int:stop\_year>', ani\_bisecti), # se apeleaza cu https://127.0.0.1:8000/1899/2025

Data Analysis and Processing using Python 2/7

14.3.2025

Data Analysis and Processing using Python 2of7.pdf



Model (JSON) 🡨 Controller (JavaScript) 🡪 View (html/css) – Model View Controller

Django este un cadru / framework pt crearea de aplicatii web

Pg 27 – instalare Django

Pip install Django

python -m django

Documentatie: <https://docs.djangoproject.com/en/3.0/>

django-admin startproject denumire\_aplicatie OR python -m startproject app\_name

Manage.py – modul in care interactionezi cu proiectul Django

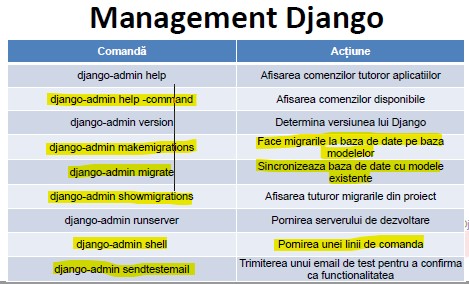
url.py – stocarea tuturor url-urilor proiectului

wsgi.py = web server gateway interface – legatura proiectului Django cu web-serverul / hosting-ul

python manage.py runserver – ruleaza serverul

oprirea portului: kill -9 $(lsof -t -i:8000)

djagon-admin – crearea proiectelor, migrarea basei, crearea utilizatorilor (accesat prin python -m Django sau manage.py)

Previziunile sunt mereu gresite. Dezvoltarea dureaza mereu mai mult.

admin.py – inregistram modelele (??obiectele) pentru admin panel

migrations.py – contine migratiile bazei de date

models.py – toate modelele (obiectele aplicatiei)

views.py – fiecare view primeste o cerere HTTP, o proceseaza si o returneaza (ATP aici accesam functiile)

tests.py – teste de verificare pentru aplicatie

## Django-admin startapp Instructions

!! o aplicatie trebuie activate:

INSTALLED\_APPS = [‘ ‘, ‘blog’,]

!! In Django, Controller-url == framework-ul !!

Model – Model – Stratul de date

View – Controller – Gestionarea cererilor primate din interfata

Template – View – Producerea continutului de raspuns

1. django-admin startproject \_\_proj\_name\_\_ - NOT mkdir
2. python manage.py startapp \_\_app\_name\_\_ - cd into folder, then
3. add \_\_app\_name\_\_ to INSTALLED\_APPS in settings.py (1x/proj)
4. urls.py (proj) 🡪 add path – ex: path(‘/short\_url‘, function\_name) + function

from django.conf.urls import include

urlpatterns = [

path('docxcloner/', include('docxcloner.urls')),]

this will link urls.py (proj) to the views.py (app) from which you can call a F.

client request goes into url which goes to views and via a context to a template

from django.http import HttpResponse

def zile\_pana\_la\_lansare(request):

    return HttpResponse("Mai sunt 100 zile")

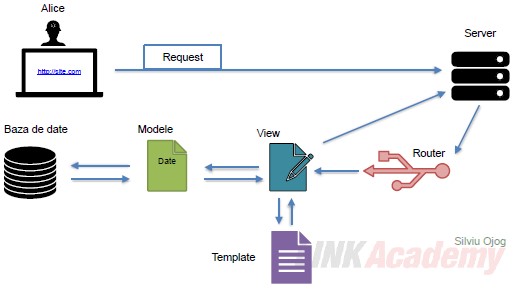
OR

    context = {

        'data' : f'{day} - {luni[month-1]} - {year}'

    }

    return render(request, "data.html", context)



Urlpatterns = toate rutile pe care le poate recunoaste aplicatia (vezi Router)

urls.py va fi initial de forma:

urlpatterns = [path(‘admin/’, adminsite.urls)], apoi rute pt differite sectiuni:

path(‘users/’, views.all\_users)

Documentatie view: <https://ccbv.co.uk/> -- generic views provide abstract classes implementing common web development tasks -- power comes at the expense of simplicity

1. ~~take the function from urls.py (app) and go to views.py (project) == lansator.views~~
2. import the function from views.py (app) INTO urls.py (app) – why for modularization

from lansator.views import zile\_pana\_la\_lansare

and to urlpatterns in urls.py (proj):

urlpatterns = [

    path('admin/', admin.site.urls),

    path('lansare/', zile\_pana\_la\_lansare),

1. in app folder mkdir templates 🡪 index.html ; then, in views.py (app) – add a F. which contains a request, & returns the index.html

def racheta\_template\_view(request):

    return render(request, "index.html")

In this way you can call a request from the project & link it from the the app via urls.py (proj)

from lansator.views import

1. from lansator.views / views.py (app) import,

then b)

urlpatterns = [

    path("", racheta\_template\_view),

1. a) in views.py (app) – acum pot isa returnezi request-uri mai complexe utilizand render + context + \*.html

from django.shortcuts import render

from django.http import HttpResponse

def today\_view(request):

    context = {

        'data' : f'{day} - {luni[month-1]} - {year}'

    }

    return render(request, "data.html", context)

1. Apoi, in urls.py (proj)

from lansator.views (views.py (project)) import today\_view

urlpatterns = [

    path("azi", today\_view),

1. proj\_folder/templates: index.html
2. How to use context = { key/example : value}
3. In views.py (app) 🡪 f. with (request) + contex context = { key/example : value} + return render(request, “\*.html”, context

from django.shortcuts import render

def today\_view(request):

   context = { ' key/example ' : ‘ceva / inclusive o variabila’ }

return render(request, "data.html", context)

1. In urls.py (proj) 🡪 import F. from views.py(app) + add path

urlpatterns = [path("azi", today\_view), ]

# azi va fi part din url; URL = base\_url + /azi

1. in \*.html / in "data.html", folosesc {{ key/example }}

        Astazi este {{data}}

{{data}} va returna ‘ceva / inclusive o variabila’

1. cum sa pasam o valaore din browser/user in server
2. in views.py (app) cream o F. care poate sa accepte variabila din browser, impreuna cu un context care accepta o variabila (exemplu: {hex}

from django.shortcuts import render

def hex\_color\_view(request, hex):

    context = {        "culoare\_hex":f'#{hex}'    }

1. in templates\_folder (app) cream un \*.html (ex: hex\_color.html) care va afisa variabila luata de la utlizator (ex: {{ culoare\_hex}})

background-color:{{culoare\_hex}};

1. in urls.py (proj) 🡪 import F. from views.py(app) + add path (introdu un catra app din views-ul proiectului) – c) cheama functia de la a) (i.e., hex\_color\_view)

from culori/app.views import hex\_color view

1. in urls.py (proj) – numai proiectul are urls (app-urile au views)

from culori.views import hex\_color\_view

urlpatterns = [path('culoare/<hex>', hex\_color\_view), ]

!! <hex> fix acest parametru trebuie sa fie in context

!! Make sure you have installed the Django Template (to properly view html)

!! Settings 🡪 search settings: emmet 🡪 add item: item Django-html value html

## Data Analysis and Processing using Python 3/7

* how to install app & get a http response (subject matematica; notes I took Django 21 Mar 25.txt below) ; step-by-step instructions are slightly better

NA: mindamp of what is in each folder

<https://www.hostinger.com/tutorials/django-best-practices>

<https://www.doprax.com/tutorial/django> - tutorial-for-beginners-part-1/

create folder for project --> cd / ls

python -m django startproject matematica

move into folder

python manage.py runserver

sql - structured querry language

NO -- git clone (drag folder matematica into folder)

Use Github\_bash\_command.py

python manage.py startapp factorial

$cd factorial

$code . -- command allows you to r

in seetings.py in folder matematica (proj fld) --> installed\_apps

'factorial',

urls.py --> path('factorial/', factorial\_view),

from factorial.views import factorial\_view

views.py (in app: factorial) --> arhitecture

in app views.py --> from django.http import HttpResponse

def factorial\_view(request):

return HttpResponse("Aici va fi factorialul dvs")

http://127.0.0.1:8000/admin/

http://127.0.0.1:8000/factorial/

in factorial (app) --> templates folder --> factorial.html

django template --> **install extension Django Template** 🡪 h2: g2 settings emmet --> add item: item django-html value html (Emmet: Include Languages)

settings.py DEBUG = True (leaves supplementary info for dev. s, turn to False when deploying)

Django Temlate (.html)

for + Enter in body -- for in python bagat in html (need to be on Django Template)

form + TAB => form

input:text + TAB !! other options

16 Aug 2025 – add an application (copilot instructions):

Scaffold a new Django project & app

~~# 1.1 Create & activate a venv~~

~~python3 -m venv .venv~~

~~source .venv/bin/activate~~

~~# 1.2 Install Django & Matplotlib~~

~~pip install django matplotlib~~

~~# 1.3 Start project & app~~

~~django-admin startproject mysite .~~

python manage.py startapp investments

# 1.4 Register the app

1. In mysite/settings.py, add:

INSTALLED\_APPS += ["investments"]

1. in urls.py (proj) add path 🡪 ex: path(‘factorial/’, factorial\_view) + !! from XXX\_APP.views import factorial\_view

from factorial.views import factorial\_view

urlpatterns = [path('factorial/<n>/',factorial\_view),]

1. views.py (app)

from django.http import HttpResponse

def factorial\_view(request, n):

    try:

        n = int(n)

    except:

        return HttpResponse("Aici va fi factorialul dvs")

    if n < 0:

        return HttpResponse("Factorialul trebuie sa fie >= 0")

    return HttpResponse(f"Aici va fi factorialul de eafefe {n}")

1. views.py w/o factorial.html

def factorial\_view(request):

return HttpResponse("Aici va fi factorialul dvs")

1. views.py (app) with factorial.html🡪 def XXX\_template\_views (): -- am creat o F. care va pasa / Randa contextual catra o pagina XXX.html

from django.shortcuts import render

def factorial\_template\_view(request, n):

    context = {n':n, 'factorial':produs}

    return render(request, 'factorial.html', context)

views = contain F.s which pass requests (here, pass request into a html page)

conext = cel care paseaza – ex: paseaza cheia ‘n’ cu valoarea

render = methods which injects code (context) into html page / renders the request

1. In factorial.html 🡪 am pasat continutul context-ului in XXX.html utilizand {{}}

<body>

    <h1>Factorial de {{n}} va avea produsul {{factorial}}</h1>

1. Urls.py (proj) 🡪 add the path for views.py (app) with factorial.html (am important metoda XXX\_template\_view in urls.py)

urlpatterns = [

    path('template/<n>', factorial\_template\_view),]

!! can use: “path(‘product’/<variable>/, views.addproduct” – can call directly, w/o import + str:variable forces the variable to be a string, or int, or…

1. Note: Can pass list or dict in same way: (i.e., in XXX.html {{dict}} ; in conext: [‘lista’: list\_variable] + outside content list\_variable = [] OR dict = {}
2. Django template library 🡪 for + TAB – iterate in xxx.html

    {% for i in lista %}

    <p>{{i}}</p>

    {% endfor %}

!! requires lista below to exist (views.py):

def inmultire\_view(request, num):

    lista = []

    num = int(num)

    for i in range(num+1):

        lista.append(f"{i} \* {num} = {i \* num}")

return render(request, "XXX.html", context)

!! add valori to context for valori to work in XXX.html

## Data Analysis and Processing using Python 4 of 7

\*how to use include; how to receive a parameter & return it ; how to take the received parameter & process it, then return it – ATP: use case: receive a parameter x & a function f(x) = a\*x^2+b\*x+c

\*how to add a form (form:get + input:text) + how to pass the user\_input for processing

\*how to use try: except: continue (to prevent errors for inproper value)

\*how to stack options

\*#how to pass into context & then to render (h2 pass user input / parameter)

\*how to save output via pandas – df.to\_json(“things.json”)

1. (urls.py in proj) –URL mare

from django.urls import path, include

urlpatterns = [

    path('parola/', include('parola.urls')),] # parola.urls **==** app\_name.urls

1. create urls.py in app (URL mic)– i.e. import path + copy-paste urlpatterns = []

from ~~capitalizare~~.views import capitalizare\_view

1. in views.py (app) –

def capitalizare\_view(request, ~~text~~):

return HttpResponse~~(text.upper())~~

!! can pass dict / list into httpResponse, simply: after (request, var):

li\_years = "\n".join(map(lambda x: f"<li>{x}</li>" ,years\_array))

return HttpResponse(f"<ol>{li\_years} </ol>")

how to receive a parameter & return it

1) URL mare

urlpatterns = [

   path('parametri', parametri\_view), # ?text=hello

   path('<text>', capitalizare\_view), ]

1. URL mic

from .views import capitalizare\_view, parametri\_view

## URL MIC

urlpatterns = [

   path('parametri', parametri\_view), # ?text=hello

   path('<text>', capitalizare\_view), ]

1. Views.py

def parametri\_view(request, text): # text is from user

    print(request.GET)

    primit = request.GET.get('text')

        return HttpResponse(primit.upper()) # calls method upper on the result from dictionary (reguest.GET = dict => request.GET.get('text') == value

.get(‘text’) – calls dict (avoids error) & brings back the value corresponding to the key “text” (which is received from the user)

how to take the received parameter & process it, then return it

    OPTIUNI = CIFRE + LITERE\_MICI + LITERE\_MARI # how to combine a list vs list of lists if ,

1) views.py (app)

def alege\_view(request):

    print(request.GET) # a returnat lungimea introdusa prin formularul submit

    context = {}

    lungime = request.GET.get('lungime')

    if lungime:

        parola = ''

            parola += random.choice(CIFRE) # something to change based on user\_input

        context['parola'] = parola # this probably creates the key value pair

## opt: code to save the user\_input

    return render(request, "alege.html", context)

Note: code to save the user\_input using panda -- \*how to save output via pandas

## opt: code to save the user\_input

import pandas as pd

        df = pd.read\_csv("parole.csv", index\_col=0) # .csv must E, index\_col for formatting

        df.loc[len(df)] = parola # allows to keep adding to file

        df.to\_csv("parole.csv") # method to add to csv from panda

\*how to add a form; (connected to above)

2) alege.html 🡪 a) form:get 🡪 input:text (or others) + Submit button (input:submit)

    <h1>Aici va fi formularul dvs</h1>

    <form action="" method="get">

<input type="text" name="" id=""> # name=”diferit” => 127.0.0.1:8000/parola/alege?diferit=user\_input # user input in form goes is submitted as dict key value pair text:user\_input, but text is named diferit so diferit=user\_input

OR

        <input type="number" min=8 max=15 value=12 name="lungime" id="">

+        <input type="submit" name="" id="" value="Name of Submit Button">

OR <input type="text" name="lungime" id="">

3) view.py (app) – the user\_input type=”text” will be saved as lungime:user\_input & available for .get(‘lungime

def alege\_view(request):

    context = {}

    lungime = request.GET.get('lungime') # set via <input type="text" name="lungime" id="">

# GET = metoda HTTP

# .get() = metoda dictionarului request.GET (acest dict ia key\_value\_pair{lungime:user\_input}

# acum pot pasa acest user\_input in context si sa fac render

    lungime = request.GET.get('lungime')

    if lungime:

            parola += random.choice(CIFRE)

        context['parola'] = parola

    return render(request, "alege.html", context)

\*how to add if or for & display a parameter based on user input (req Django Template) – after ABOVE

    {% if parola %}

        <h2> {{parola}}</h2>

    {% endif %}

\*how to use try: except: continue (to prevent errors for inproper value)

Views.py (app)

def alege\_view(request):

    try:

        lungime = int(lungime)

    except:

        pass

\*how to stack options

    <form action="" method="get">

         <input type="checkbox" name="withupper" id="" checked>

        <label for="withupper">Cu litere mari?</label>

        <input type="checkbox" name="withdigits" id="" checked>

        <label for="withdigits">Cu numere?</label>

+

def alege\_view(request):

    lungime = request.GET.get('lungime')

    if request.GET.get('withupper'):

        OPTIONS += string.ascii\_uppercase

    if request.GET.get('withdigits'):

         OPTIONS += string.digits

    for i in range(lungime):

        parola += random.choice(OPTIONS)

      context['parola'] = parola # how to pass into context & then to render

    return render(request, "alege.html", context)

## Data Analysis and Processing using Python 6 of 7

-5/7 = not interesting

\*how to submit a value directly into context from user (via form)

\*how to utilize for in html linked to context

\*how to add a list + select

\*how to use if/else for method POST/GET (views.py app) – important

\*how to display the result of user\_input after processing / after request.POST

\*how to add a complex logic into views.py (app)

\*how to use enumerate module + h2 iterate through all objects in a class

numele\_aplicatie = 'joc'

fisierul = 'urls'

⬄ f’{numele\_aplicatie}.{fisierul} = ‘joc.urls'

urlpatterns = [

    path('admin/', admin.site.urls),

    path('joc/', include('joc.urls')),]

html –> within form+TAB -- type:input == type:button

<form action="" method="post">

    <select name="" id="">

    <option value="Rock"></option>

    <option value="Paper"></option>

    <option value="Scissors"></option>

    <button type="submit"></button> ## button:submit

    </select>

</form>

## post versus get sends browser address bar / hides in body of request

## (+see in terminal)

<form action="" method="get">

{% csrf\_token %}

    <select name="chosen" id="">

    <option value="1">Rock</option>

    <option value="2">Paper</option>

    <option value="3">Scissors</option>

    <button type="submit">Trimite</button>

\*how to utilize for in html linked to context – add here

    {% if client %}

        <p> Ai ales: {{client}} </p>

        <p> Serverul a ales: {{server}} </p>

        <p> Rezultatul este: {{rezultat}} </p>

    {% endif %}

Request.GET vs request.POST -- See min 33

HTML form = elements wrapped inside **<form>...</form> tags – allow user to input text, select options, manipulate controls, submit data to server – Django Forms**

* GET method appends form data to URL as query parameters – ex: ?q=data
* POST method sends data in body of HTTP request – used when request modifies data on server (ex: Updating a database

Steps: 1) form:get (in .html) +TAB 🡪 label: +TAB + input: … / text + input:submit OR -- !! needs return render(request, “name.html”)

2) form:post (in .html) +TAB 🡪 {% csrf\_token %}🡪 label: + input:text + input:submit

3) Views.py (app)

\*how to use if/else for method POST/GET -- important

def rock\_paper\_view (request):

    context = { 'pairs' : Optiuni.pairs()}

    if request.method == "POST":

        print("Metoda -- POST")

        print(request.POST)

client = request.POST.get("chosen")

## extended logic

    else:

        print("Metoda -- GET")

context = {context gol sau rezultatul alegerii user-ului}

    return render(request, "rock\_paper.html", context)

Why? Refreshing page calls GET versus click: Trimite calls POST method

4) ## extended logic

        if client and (client in map(str, Optiuni.values() )):

            alegere\_client, alegere\_server, rezultat\_joc = logica\_de\_joc(int(client))

            context.update({

                'client': alegere\_client,

                'server': alegere\_server,

                'rezultat' : rezultat\_joc,

            })

Exp: update the context based on the user input

5) implement if/then in html so that reaction to user\_input can be displayed

\*how to display the result of user\_input after processing / after request.POST

    {% if client %}

        <p> Ai ales: {{client}} </p>

        <p> Serverul a ales: {{server}} </p>

        <p> Rezultatul este: {{rezultat}} </p>

    {% endif %}

Exp: after the execution of request.POST (user has submitted data), the post-processing result of the server needs to be displayed (if it exists, therefore if client).

Note: can clarify logic in separate file – ex: logica\_joc.py – use of enum module (rol: stocarea constantelor cu context tematic)

import enum

class Optiuni(enum.Enum):

    Rock = 1

    Paper  = 2 ; Scissors = 3

# initializezi obiecte care mostenesc din Enum

    def \_\_str\_\_(self): # returns obj.name as opposed to address of obj

        return self.name

# print(Optiuni.Rock) => 1

    client = Optiuni(client) # store the user\_input value

\*how to add a complex logic into views.py (app) – min 105

Note: Test logic beforehand logica\_joc.py OR tests.py, then

Views.py(app)

def logica\_de\_joc(client:int):

    client = Optiuni(client)

    server = Optiuni(random.choice([1, 2, 3]) # call random module & feed the value to Optiuni(), which is a command that instantiates the Object of class Optiuni based on the provided value (1-3) & assigns it to the variable server

    if client == server:

        rezultat = "Egalitate"

    elif client.wins\_over(server):

        rezultat = "Castig"

    else:

        rezultat = "Pierdere"

    return client, server, rezultat

Exp: a F. is created to process the user\_input. However, it is required to check whether the user input should be processed based on whether the input was provided. Ex:

def rock\_paper\_view (request):

    if request.method == "POST":

        client = request.POST.get("chosen")

…

        if client: # if provided user input, then call logica\_de\_joc()

# & pass user\_input == client

           alegere\_client, alegere\_server, rezultat\_joc = logica\_de\_joc(int(client))

# update the context & return the page

            context.update({

                'client': alegere\_client,

                'server': alegere\_server,

                'rezultat' : rezultat\_joc,

            })

    return render(request, "rock\_paper.html", context)

\*h2 iterate through all objects in a class – min 141

a) initialize the objects – ex Rock = 1 b) define values -> return list of objects (in class Optiuni – return [iterate] c) call the method/F. via Optiuni.values()

class Optiuni(enum.Enum):

    Rock = 1, Paper  = 2, Scissors = 3, Lizar = 4, Spock = 5

    def \_\_str\_\_(self):

        return self.name

    @classmethod # w/o @classmethod OUT: <function Optiuni.values at 0x000001E6C6E77E20>

    def values(cls):

        return [op.value for op in Optiuni] # returns the !! values of all the objects of class Optiune – value is METHOD in Enum class & returns the value of the Enum member

# I defined the decorator & method

print(Optiuni.values()) # OUT: [1, 2, 3, 4, 5]

Note: if [op.name for op in Optiuni] # OUT Rock, Paper, …

    client = input("Introduceti o valoarea 1.Rock, 2.Paper, 3.Scissors\n")

    server = random.choice([1, 2, 3])

translated in views.py

    client = Optiuni(client) # instantiate an class Optiuni Object based on user\_input

    server = Optiuni(random.choice(Optiuni.values()  )) # same but via random choice from the values() / Method of the class Optiuni

classmethod() transforms a regular method into a class method – i.e. bound to the class & not an instance of the class – def values(): return doesn’t need the cls, but the decorator uses def values(cls): - i.e. the method receives the class (cls) as its first argument, rather than an instance(self) / object.

This means it can be called on the class itself rather than being called on instances of the class

\*h2 iterate through all object names & values in a class

class Optiuni(enum.Enum):

    Rock = 1, Paper  = 2, Scissors = 3, Lizar = 4, Spock = 5

    @classmethod

    def pairs(cls):

        return [(op.name, op.value) for op in Optiuni]

# print(Optiuni.Rock) – OUT Rock

# print(Optiuni.Rock.value) – OUT 1

In views.py (app)

    @classmethod

    def pairs(cls):

        return [(op, op.value) for op in Optiuni]

def rock\_paper\_view (request):

    context = { 'pairs' : Optiuni.pairs()}

min 153