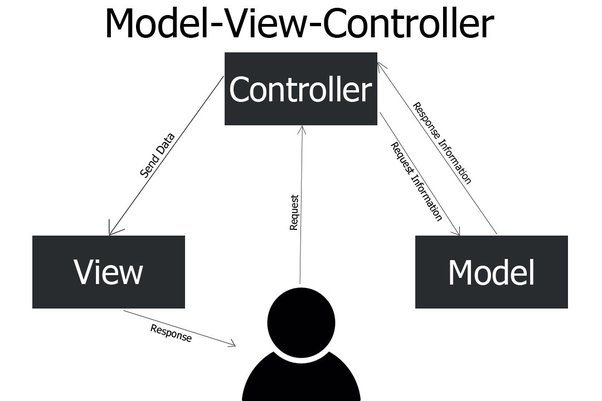
**[Q7. Is Django an MVC or an MVT framework?](https://www.quora.com/Is-Django-an-MVC-or-an-MVT-framework" \t "_blank)**

Before understanding whether Django is MVC or MVT, or what MVT is for that matter, you need to understand **what MVC is.**



Simply put, It is a traditional Software Design Pattern that puts a layer of abstraction between the user and the software to show only the relevant details to the end user.

**MVC stands for Model-View-Controller**, which are the three blocks of this architecture.

**The Model** defines the business logic of the software. It manipulates and holds the states of the application. It contains and updates the data that should be used by the application, or a relevant view.You can think of it as an interaction medium between the database and the application to make things simple.

**The View** represents all that is visible to the user, be it a graph, UI components, table, or simply a web page holding certain information. How different views interact and what data should each view hold is defined by the Controller, or in the case of Django, the view files.

**The Controller** is responsible for accepting inputs from the user(ex : URL in browser), and direct those inputs to the relevant view or model. Basically, the Controller defines how the application responds to the user requests.



**What is MVT and how does it differ from MVC?**

MVT stands for Model-View-Template which is a similar design pattern again defined by three parts.

**The Model**again defines the business logic as previously.

**The View** is responsible for deciding what Data should be displayed to the user.

**The Template** is the standard HTML file mixed with Django Template Language (DTL) (or Jinja2, whichever you feel suitable), which helps renders the data provided by the view into the HTML file. For example, you had a list of fruits which you want to render in HTML, you would use a for loop to render each <li> component. DTL can help you achieve this task easily.

MVT isn’t a traditional design architecture like MVC, and actually, Django hides most of the details from you ! MVT is MVC hidden under a layer of abstraction of Django itself, so you get to work on the relevant stuff and not waste your time defining long controllers for each task.

When a user makes an HTTP request to the server, MVC calls the controller to either make changes to the view via the Model or asks the model to return a view.

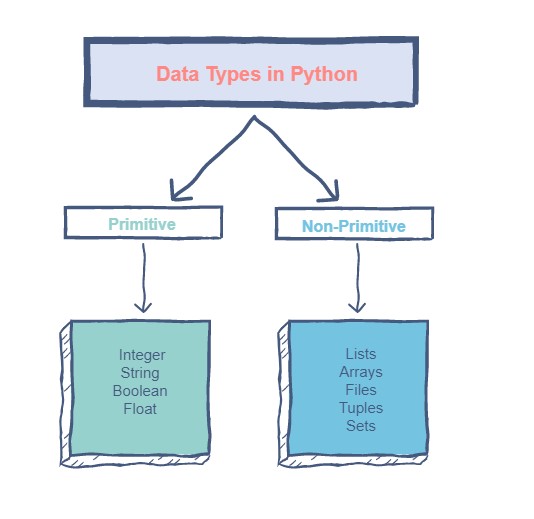
In MVT, an HTTP request calls upon the view directly to fetch the relevant template along with the required data. The data is filled in the template (using DTL) and returned to the user.

Also in MVT, unlike MVC, specific code to fetch data from the database and mapping them to URLs do not have to be written. All these activities are handled by the framework itself, you just need to tell it what models to present and Django takes care of the rest.

***Conclusion - MVT is an abstraction over MVC to save developers from writing trivial stuff and focus on the important parts***

What are Premitive & non-premitive Data Types in Python

Primitive data types: Data types which are pre-defined and supported by the programming language.

Non-primitive data types: Data types which are derived from the primitive data types and offer increased functionality.​​

During the request phase, before calling the view,

Django applies middleware in the order it’s defined in MIDDLEWARE, top-down.

You can think of it like an onion: each middleware class is a “layer” that wraps the view,

which is in the core of the onion. If the request passes through all

the layers of the onion (each one calls get\_response to pass the request in to the next layer),

all the way to the view at the core, the response will then pass through

every layer (in reverse order) on the way back out.

If one of the layers decides to short-circuit and return a response

without ever calling its get\_response, none of the layers of the onion

inside that layer (including the view) will see the request or the response.

The response will only return through the same layers that the request passed in through

Django Request-Response Cycle?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

WSGI is the Web Server Gateway Interface.

It is a specification that describes how a web server

communicates with web applications, and how web

applications can be chained together to process one request.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**The Complete -- Data Flow Operation**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

When a user sends a request to your application, a WSGI handler is created that:

1.imports the settings.py file as well as Django's exception classes.

2.Loads all middleware classes found in the MIDDLEWARE CLASSES

or MIDDLEWARES tuple in settings.py (depending on Django version)

3.URL Router (URL Dispatcher)

URL Router receives the request from the request middleware and extracts the URL Path.

The URL router will try to match the request path to the available URL patterns based on the URL path.

4.Views-We are now in the business logic layer Views.

5. the request is forwarded to context processors,

which add context data to help Template Renderers render

the template and provide the HTTP response.

6. Response Middlewares

The request will be sent back to the Response middlewares to be processed.

Before delivering the request back to the client,

response middlewares will process it and add or modify the header and body information

(Browser). After that, the browser will analyze the information and present it to the user.

**1. What is difference between .py and .pyc file in python**

\_\_\_\_\_\_\_\_\_

.py

\_\_\_\_\_\_\_\_\_

1.These files store the source code of the Python programs

2.Speed of the .py file is lower while generating output

3..py files store the human-readable high-level python code

4.Data in this file is interpreted by the interpreter of python

\_\_\_\_\_\_\_\_\_

.pyc

\_\_\_\_\_\_\_\_\_

These files store the bytecode of the Python programs

Speed of the .pyc file is higher than the .py file as it contains the compiled codes.

.pyc files store the machine-readable low-level intermediate code

The data in this is the interpreted data which is ready to run on a virtual machine

.pyc file are not generated for all the files that are run - instead it is generated only for files that imported

**2. Explain working of PVM**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The role of Python Virtual Machine (PVM) is to convert the byte code instructions into machine code

so that the computer can execute those machine code instructions and display the final output.

To carry out this conversion, PVM is equipped with an interpreter.

The interpreter converts the byte code into machine code and sends that machine code to the computer processor for execution.

Since interpreter is playing the main role, often the Python Virtual Machine is also called an interpreter.

**3. what is difference between static, instance and abstraction method**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. static method can be called without object of class, but it also associates with class

2. instance method required object of class to call the method, it also associated with object

3. abstract method - it is defined in base class but do not have any implementation,

we cannot create object of abstract class.

**What is Lock in Multithreading? – it is class in threading module which is used for synchronization in race condition.**

The condition occurs when one thread tries to modify a shared resource at the same time that another thread is modifying that resource – t​his leads to garbled output, which is why threads need to be synchronized. The threading module of Python includes locks as a synchronization tool.

A Lock object cannot be acquired again by any thread unless it is released by the thread which is accessing the shared resource.

Lock class in present in Threading module and it Contains 2 methods.

* 1. Acquired Lock
  2. Release Lock

**What is GIL- Global Interpreter lock in python?- one thread to hold the control of the Python interpreter**

The Python Global Interpreter Lock or GIL, in simple words, is **a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter**. This means that only one thread can be in a state of execution at any point in time.

Why was the GIL chosen as the solution:  
Python supports C language in the backend and all the related libraries that python have are mostly written in C and C++. Due to GIL, Python provides a better way to deal with thread-safe memory management. Global Interpreter Lock is easy to implement in python as it only needs to provide a single lock to a thread for processing in python. The GIL is simple to implement and was easily added to Python. It provides a performance increase to single-threaded programs as only one lock needs to be managed.

# different ways to copy a list

# 1. using assignment operation # reffering

list1 = [1,2,3,4,5]

copy\_list = list1

print(copy\_list)

print(id(list1))

print(id(copy\_list))

print('\n')

# 2. using copy() function

list1 = [1,2,3,4,5]

copy\_list = list1.copy()

print(copy\_list)

print(id(list1))

print(id(copy\_list))

print('\n')

# 3. using list constructor

list1 = [1,2,3,4,5]

copy\_list = list(list1)

print(copy\_list)

print('\n')

# 4. using indexing method

list1 = [1,2,3,4,5]

copy\_list = list1[:] # cloning

print(copy\_list)

print('\n')

# 5. using list comprehension

list1 = [1,2,3,4,5]

copy\_list = [i for i in list1]

print(copy\_list)

print('\n')

# points to remember

# Assignment operation → Both lists will point to the same list object

# Indexing,list(),list comprehension,copy()

# Both lists(original and copied list) will point to different list-objects.

# 6. extend method

list1 = [10, 20, 30, 40, 50, 60, 70, 80, 90]

copy\_list = []

copy\_list.extend(list1)

print("Input List:", list1)

print("Copied List:", copy\_list)

print('\n')

# 7. append method

list1 = [10, 20, 30, 40, 50, 60, 70, 80, 90]

copy\_list = []

for ele in list1: copy\_list.append(ele)

print("Input List:", list1)

print("Copied List:", copy\_list)

print('\n')

# 8 deepcopy

import copy

list1 = [1,2,3,4,5]

copy\_list = copy.deepcopy(list1)

print(copy\_list)

print('\n')

# 9 shallow-copy

import copy

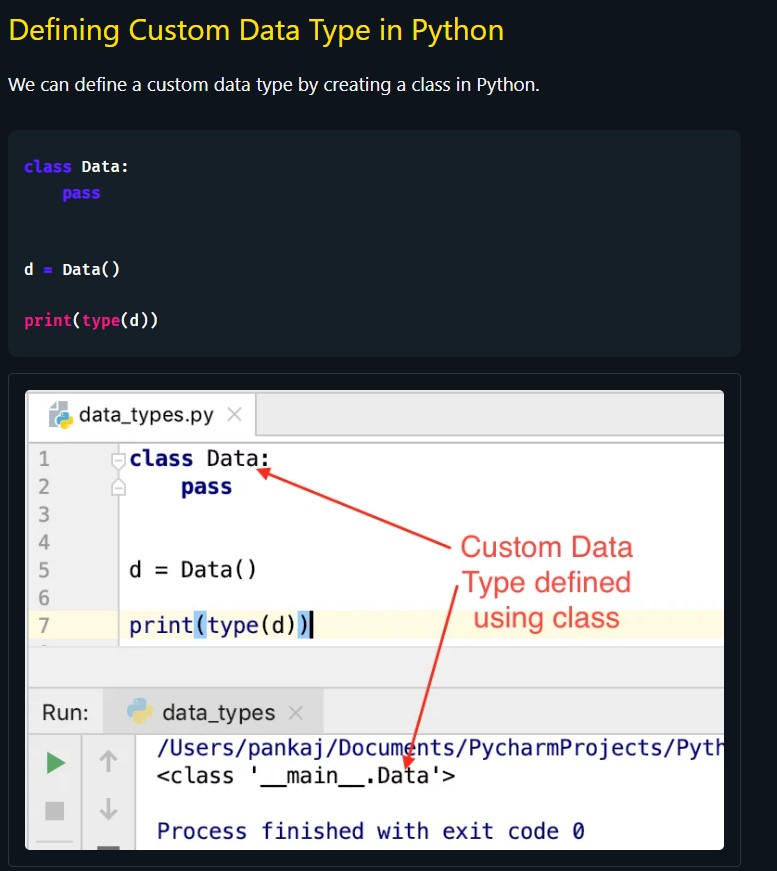
list1 = [1,2,3,4,5]

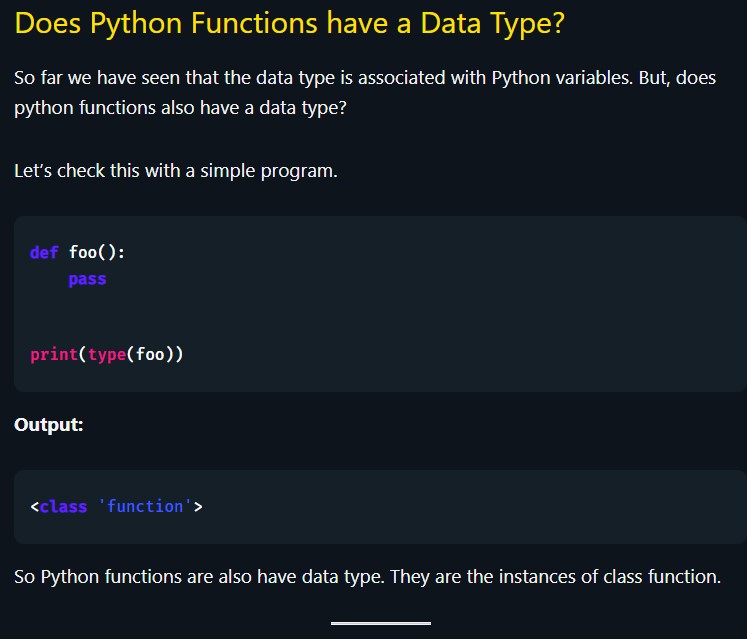
copy\_list = copy.copy(list1)

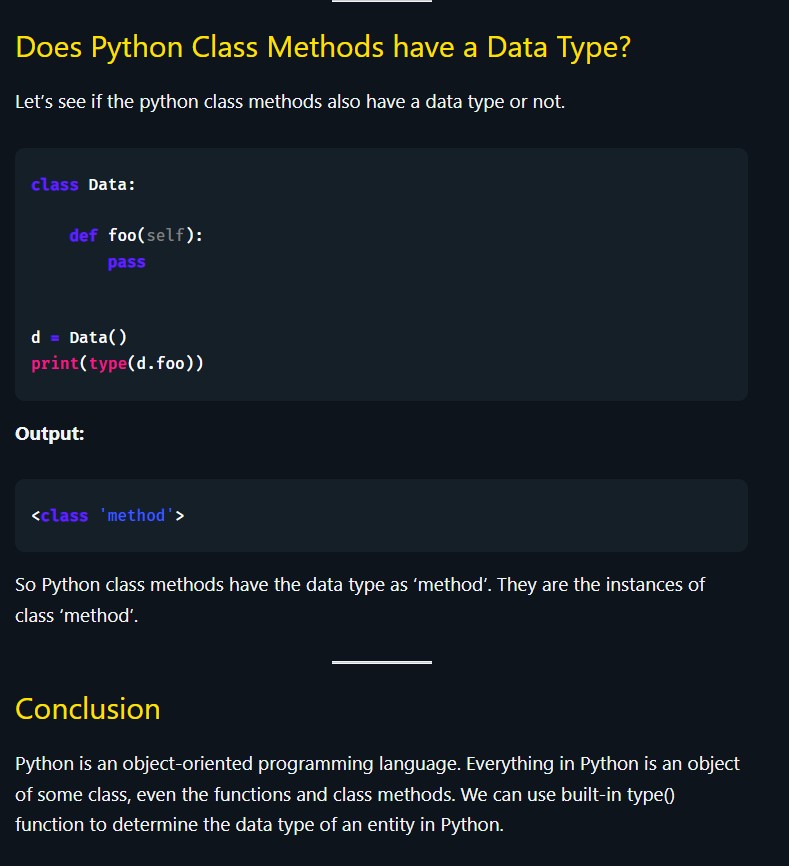
print(copy\_list)

print('\n')

How to Create Custom DATA TYPE in PYTHON?







Conclusion: -

1. Yes, we can create custom data types in PYTHON.
2. By defining Class, we can create custom data type.
3. Even the functions, methods are custom data type.

**What is Difference between Declarative and Classical Way of Mapping?**

**Which Databases are supported by Django??**

Django officially supports the following databases:

[PostgreSQL](https://docs.djangoproject.com/en/4.0/ref/databases/#postgresql-notes)

[MariaDB](https://docs.djangoproject.com/en/4.0/ref/databases/#mariadb-notes)

[MySQL](https://docs.djangoproject.com/en/4.0/ref/databases/#mysql-notes)

[Oracle](https://docs.djangoproject.com/en/4.0/ref/databases/#oracle-notes)

[SQLite](https://docs.djangoproject.com/en/4.0/ref/databases/#sqlite-notes)

There are also a number of [database backends provided by third parties](https://docs.djangoproject.com/en/4.0/ref/databases/#third-party-notes).

In addition to the officially supported databases, there are backends provided by 3rd parties that allow you to use other databases with Django:

[CockroachDB](https://pypi.org/project/django-cockroachdb/)

[Firebird](https://pypi.org/project/django-firebird/)

[Google Cloud Spanner](https://pypi.org/project/django-google-spanner/)

[Microsoft SQL Server](https://pypi.org/project/mssql-django/)

[TiDB](https://pypi.org/project/django-tidb/)

The Django versions and ORM features supported by these unofficial backends vary considerably. Queries regarding the specific capabilities of these unofficial backends, along with any support queries, should be directed to the support channels provided by each 3rd party project.

**01.What does the render Function Returns?**

The purpose of render() is to return an HttpResponse whose content is filled with the result of calling render\_to\_string() with the passed arguments.

02.Redirect?

***Returns an [HttpResponseRedirect](https://docs.djangoproject.com/en/4.0/ref/request-response/" \l "django.http.HttpResponseRedirect" \o "django.http.HttpResponseRedirect) to the appropriate URL for the arguments passed.***

The arguments could be:

A model: the model’s [get\_absolute\_url()](https://docs.djangoproject.com/en/4.0/ref/models/instances/" \l "django.db.models.Model.get_absolute_url" \o "django.db.models.Model.get_absolute_url) function will be called.

A view name, possibly with arguments: [reverse()](https://docs.djangoproject.com/en/4.0/ref/urlresolvers/#django.urls.reverse) will be used to reverse-resolve the name.

An absolute or relative URL, which will be used as-is for the redirect location.

By default issues a temporary redirect; pass permanent=True to issue a permanent redirect.

**03.In which module in Django Request comes from Browser?**

In urls.py module first request comes from browser.

What are marque tags in html?

What is marquee tag?

An HTML marquee is a scrolling piece of text displayed either horizontally across or vertically down your webpage depending on the settings. This is created by using HTML <marquees> tag. Note − The <marquee> tag deprecated in HTML5. Do not use this element, instead you can use JavaScript and CSS to create such effects.

What is sessionmaker and Mapper? (sqlalchemy)--🡪

Session class is defined using sessionmaker() – a configurable session factory method which is bound to the engine object created earlier. from sqlalchemy. orm import sessionmaker Session = sessionmaker(bind = engine) The session object is then set up using its default constructor as follows − session = Session()

Faker Module in Python?

Faker is a Python package that generates fake data for you. Whether you need to bootstrap your database, create good-looking XML documents, fill-in your persistence to stress test it, or anonymize data taken from a production service, Faker is for you.

Cmd-

pip install Faker

python code-

from faker import Faker

fake = Faker()

print (fake.email())

print(fake.country())

print(fake.name())

print(fake.text())

print(fake.latitude(), fake.longitude())

print(fake.url())