Django Review

*args and **kwargs

A Python function can accept arguments without specifying the actual argument names.

```
def fun(*args, **kwargs):
    print("Positional arguments:")
    for x in args:
        print(x)
    print("Named arguments:")
    for key in kwargs:
        print(f"{key} =", kwargs[key])
fun(5, "second", today="5/9/2023", size=10)
```

`*args` contains positional arguments.
`**kwargs` is a *dictionary* of named arguments (**k**ey **w**ord **args**) and values. The names can be anything.

*args and **kwargs

The help for many Django methods looks like this:

```
Question.objects.create(*args, **kwargs)
```

this means the create() method accepts any arguments, such as:

```
poll = Question.objects.create(
    name="Who will be next U.S. president?",
    pub_date=timezone.now()
    )
```

**kwargs must be the last parameter

It should be the last parameter in a function signature.

```
def myfun(x, **kwargs):
    print("x=", x)    # required param
    print("Optional arguments:")
    for key in kwargs:
        print(key, "=", kwargs[key] )


myfun("hi", id=219241, name="ISP")
```

Django Page Templates

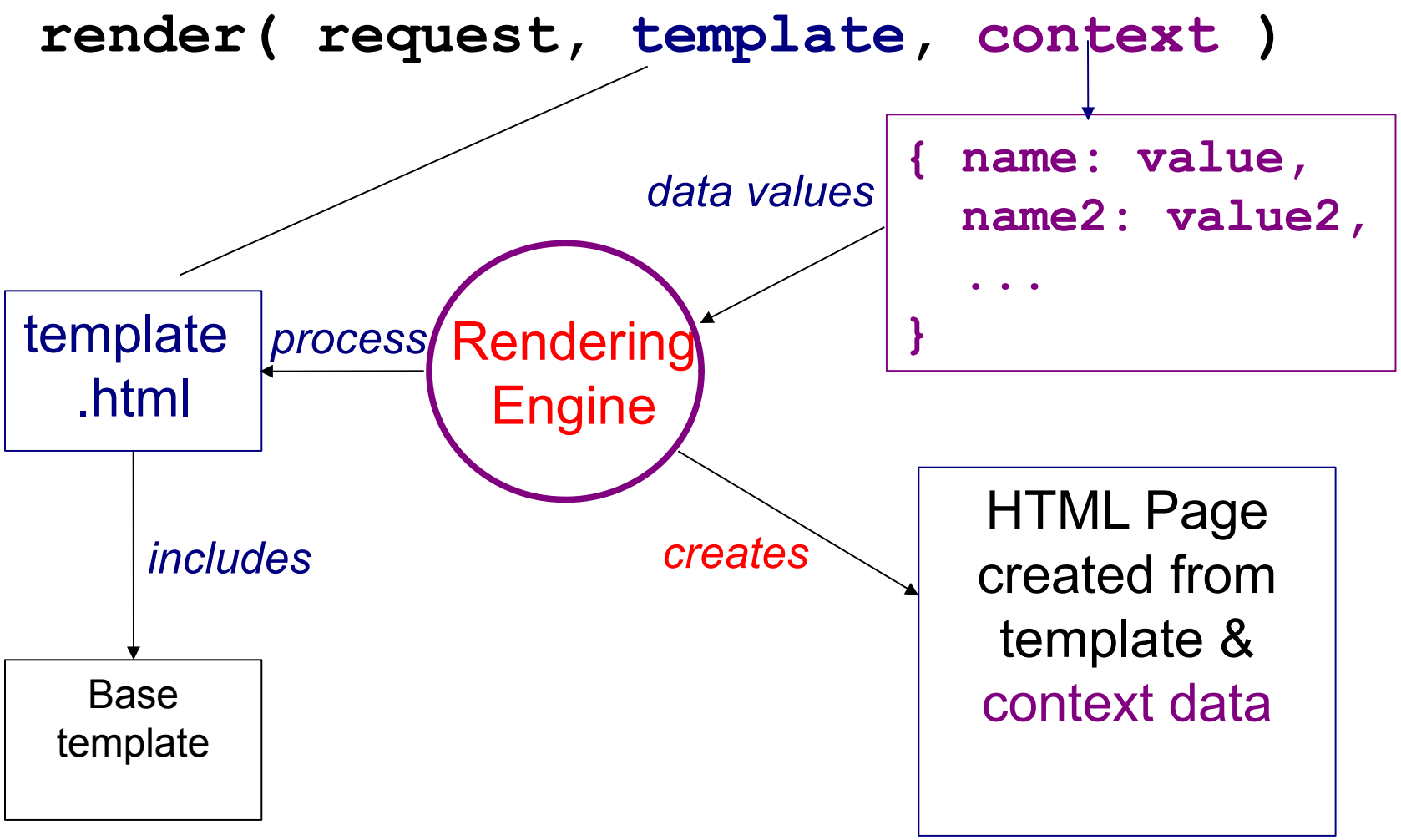
In a **template**, you put *variables* inside {{ ... }}

```
<p>
Q{{question.id}} is
        "{{question.question_text}}"
</p>
<!-- a template can invoke a method -->
{{question.was_published_recently}}
```

```
Q1 is "What is your favorite food?"
True
```

Rendering a Template

A "rendering engine" processes the template.



Explicitly invoke rendering

In a view method:

```
from django.template import loader
template =
    loader.get_template('polls/details.html')

# context = key-values to use in template
context = {'question': question, ...}
html = template.render(context, request)

return HttpResponse(html)
```

Shortcut for rendering

```
from django.shortcuts import render

context = {'question': question, ...}

# render returns an HttpResponse object
return render(request,
              'polls/detail.html',
              context)
```

Template can access `request` data

A **template** can also access vars from the **`request`**.

```
{% if user.is_authenticated %}
    <p>Welcome, {{ user.get_username }}.</p>
{% else %}
    <p>Welcome, web surfer.</p>
{% endif %}
```

`user` refers to **`request.user`**

`user.get_username` refers to
`request.user.get_username()`

Code Should be Easy to Read

Instead of:

```
return render(request,'template.html',
    {'question': "who are you?", ...} )
```

add an ***explanatory variable***

```
context = {'question': "who are you?"}
return render(request, 'template.html',
                context )
```

In a "view" what is request?

A Django "view" function looks like this:

```
from django.http import HttpRequest,
                        HttpResponse
from django.template import loader

def detail(request: HttpRequest, question_id):
    questions = Question.objects.all()[0:10]
    context = {'question_list':questions}
    template = \
            loader.get_template('some_file')

    return HttpResponse(
        template.render(context, request ) )
```

What is HttpResponse?

What does `HttpResponse represent`?

```
from django.http import HttpResponse
from django.template import loader

def detail(request, question_id):
    questions = Question.objects.all()[0:10]
    context = {'question_list':questions}
    template = \
             loader.get_template('some_file')

    return HttpResponse(
        template.render(context, request ) )
```

URL Dispatching

Each "app" can have a `urls.py` to match request URLs and dispatch them to a "view".

```
from django.urls import path

# app_name is used to define a namespace
# (used for "reverse mapping")
app_name = 'polls'

url_patterns = [
    path('', views.index, name='index'),
    path('<int:question_id>/',
              views.detail, name='detail'),
    path('<int:question_id>/vote/',
              views.vote, name='vote'),
    ]
```

Dispatch these URLs

Which view will handle each of these requests?

1) `http://localhost:8000/polls/`
2) `http://localhost:8000/polls/4/`
3) http://localhost:8000/polls/8/vote?username=nok
4) http://localhost:8000/polls/8/vote/summary

```
# URL mapping for /polls/ app
url_patterns = [
    path('', views.index, name='index'),
    path('<int:question_id>/',
              views.detail, name='detail'),
    path('<int:question_id>/vote/',
              views.vote, name='voting'),
    ]
```

Mapping from View to URL

Example: add a link to the polls index page.

How to "build" a URL inside a template?

BAD TEMPLATE CODE:

```
<a href="/polls/index">Back to Polls index</a>
```

app_name view name

GOOD TEMPLATE CODE:

```
<a href="{% url 'polls:index' %}">
Back to Polls index
</a>
```

Note that {%...%} is processed inside "..."

Why is the 2nd code better than the 1st code?

Mapping from View to URL

If a view URL requires parameters, include them in the `{% url %}` .

```
<!-- question details template -->
<a href="{% url 'polls:voting' 4 %}">
    Vote for Poll 4
</a>
```

app_name view name parameter

Reverse Dispatch

Sometimes a view controller wants to redirect the user to a different URL.

```
from django.http import HttpResponseRedirect

def vote(request, question_id):
    question = Question.objects.get(id=question_id)
    // TODO save the vote for this question
    ...
    // Show all votes for this question
    _____Redirect to polls/{id}/results_________
    return ???
```

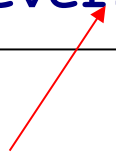
How to redirect the browser to this page?

reverse() for Reverse Dispatch

Redirect uses info from the urls.py files to construct the URL the user should go to.

```
from django.http import HttpResponseRedirect

def vote(request,question_id):
    q = Question.objects.get(id=question_id)
    ## TODO get user's choice and add +1 to votes
    ...
    # Redirect browser to page of vote results
    HttpResponseRedirect(
        reverse('polls:results', args=(q.id,) ) )
```



Get the URL that matches the named route

Thorough Testing is Needed!

Python code is interpreted

There is no compiler to catch errors (as in Java).
So, you need to **test every path of execution**.

```
NameError at /polls/1/vote/
name 'reverse' is not defined
```

Programmer forgot (in views.py):

```
from django.urls import reverse
```

but this error is not detected until `reverse`() is encountered at run-time.

Summary: Use names for app urls

All web app frameworks need a way to do this:

1. Include link to a URL **in an HTML template**

```
{% url 'app_name:view_name' args %}
```

2. Redirect user to another page **in a view (code)**

```
HttpResponseRedirect(
    reverse('app_name:view_name',
    args=(...)))
```

GET and POST

GET is used to request a web resource, such as a web page.

GET /polls/1/

What is POST used for?

(*Semantic meaning of POST*)

1. Send data to the application, such as from a form.

Your name: <input type="text" name="username" />

some text

2. To create a resource on the server.

One view for both GET and POST

One view can handle both.

Use **`request.method`** to determine which method.

```
def detail(request, question_id):
    question = Question.objects.get(id=question_id)

    if request.method == 'GET':
        # render and return the details template

    elif request.method == 'POST':
        # handle user's vote
        choice = request.POST['choice']

        # after a POST, always redirect somewhere
        return redirect('polls:results', args=(...))
```

Exploring Models - Django shell

Django interactive Python shell is in Tutorial Part 2.

```
python manage.py shell  [ -i python ]

>>> from polls.models import Question, Choice
>>> q = Question.objects.get(id=1)
>>> q.question_text
"What is your favorite programming language?"

>>> choices = q.choice_set.all( )
```

You should know how to use the Django shell.

Domain Model

Is a model of the concepts and objects that are important to your "model" for the "domain" of your application.

"Domain Model" for KU Polls includes:

```
Question
Choice
votes
question text
choice text
```

Draw a UML Domain Class Diagram

Show:

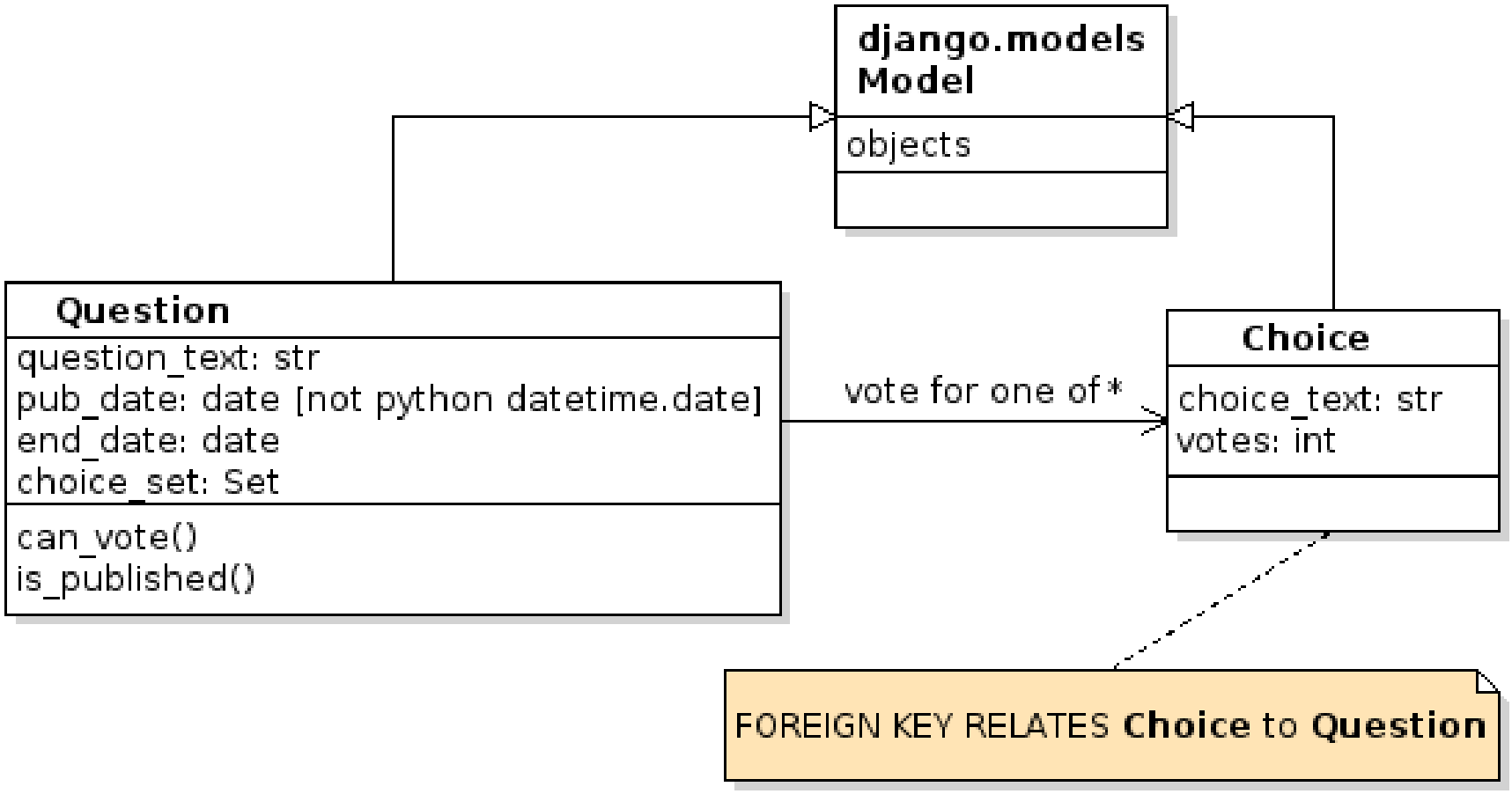
1. Classes
2. Important "domain" attributes of each class, but not non-domain variables like id.
3. Relationships between classes with multiplicity.

"Domain Model" includes:

```
Question
Choice
votes
question text
choice text
```


Question
question_text
choice for
1
*
Choice
choice_text
votes: int

UML for Software Model (not D.M.)



Persistence Operations: CRUD

All Persistence Frameworks provide a way to...

- Create (save) an entity to the database
- Retrieve an object, by id or by field value (query)
- retrieve all objects
- Update object data in database
- Delete an entity (object) from database

How does Django do these?

Try out Persistence

Try persistence operations: save(), get(), delete()

```
>>> c = Choice(choice_text = "Fortran")
>>> c.votes = 1
# Foreign Key.  You have to find this separately.
>>> c.question_id = 1
>>> c.save()

>>> for choice in q.choice_set.all():
...     print(choice)

# Now the output includes "Fortran"

# TODO: delete "Pascal" from poll. First, find it
pascal = q.choice_set.get(choice_text="Pascal")
pascal.delete()
```

Testing

Testing

Django Unit Tests extend TestCase class.

```
public class QuestionModelTest(TestCase):
    def test_create_question(self):
        question = Question(question_text="this is a test")
        self.assert
```

Wrong Name!

In Tutorial, name is "`QuestionModel`**`Tests`**".

It should be "`xxxTest`" (no "s")!

Don't use plural for your test classes.

What is a django.test.TestCase ?

```
>>> from django.test import TestCase
>>> help(TestCase)
class TestCase(TransactionTestCase)
    ...
    Method resolution order:
        TestCase
        TransactionTestCase
        SimpleTestCase
        unittest.case.TestCase
        builtins.object
```

Running Tests

```
cmd>  python manage.py test polls
```

Criticisms:

- Django test code is in same directory as production code.
- Should have separate "test" files for each target, don't bundle them into one file `(tests.py)`
- `tests.py` is poor name. Test what? Don't use plural (no "s")!

Design: Low Coupling

Good software design strives for low coupling.

Especially, low or no coupling between unrelated parts.

What features of Django reduce coupling?

1. Django divides a project into self-contained "apps"

2. {% url 'name' %} reduces coupling between URLS and templates

3. ???

Design: Portability and Reuse

Good software design enables portability and code reuse.

A framework itself is both portable and reusable (we use it to create our own web app)!

How does Django enable us to move or reuse our own web application code?

Django and Git

When you commit your Django project to Git,

what files should you **not commit**?

> Add them to `.gitignore`

> If you don't know what to put in .gitignore, create a repo on Github and ask Github to create a `.gitignore` file for you.

> What is `*.pyc` ? What is `*.py[cod]` ?

Is Django a Web Server?

[ ] Yes

[ ] No

Django is Not a Web Server

But I can type: `manage.py runserver`

and it works *right out of the box*.
How to you explain ***that***?

Web Developer

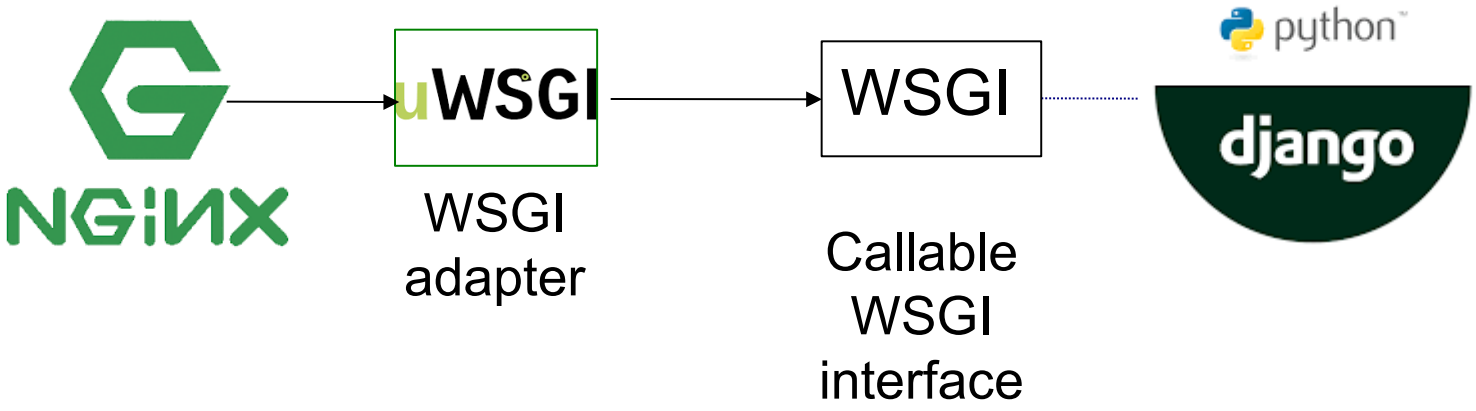
Django includes a "light-weight" HTTP server

Intended for development only.

Not suitable for production (Tutorial, part 1).

Django uses WSGI interface

WSGI (Web Server Gateway Interface) is a standard interface for *communication* between a Python web app and a web server.



You can run Django in any web server that:

- supports WSGI or has an *adapter* for WSGI interface