KAZAKH BRITISH TECHNICAL UNIVERSITY

BACKEND FRAMEWORK (DJANGO)

Lesson 2

1. RESTful API
2. API correct endpoints
3. HTTP request
4. Data exchange format in APIs
5. Protocols
6. HTTP response status codes

Python

Python is…

- Dynamic
- Interpreted
- Object-Oriented
- Exceptional
- Comfortable
- Readable
- Community

Interactive Shell

```
$ python
>>> print "Hello, world!"
Hello, world!
>>>
$ python3
>>> print("Hello, world!")
Hello, world!
>>>
```

Comments

```
# Best. Comment. Ever.
```

Booleans and Null

```
True

False

None
```

Strings

- `'Hello, world!’`
- `"Hello, world!”`
- `"""Hello,`
  `world!"""`
- `u"Hëllö, wörld!”`

String Operations

```
"foo" + "bar"

"foo"[0]

"foo"[:1]

"foo".upper()

"{0}: {1}".format("foo", "bar")

"{foo}: {bar}".format(foo=42, bar=1138)

len(“foo")
```

Sequence Operation

```
[...][0]

[...][-1]

[...][:1]  # same as [...][0:1]

[...].append(7)

[...].pop()

len([...])
```

Dictionaries

```
{'key1': 'value1', 'key2': 'value2'}
```

Dictionary Operations

```
{...}['key1']

{...}.get('key2')

{...}.keys()

{...}.values()

{...}.items()

len({...})
```

Assignment & Comparison

```
foo = ‘bar’

foo == ‘baz'

foo != ‘baz'

foo is None

foo is not None
```

Flow Control

```
if expression:
     ...
elif expression:
     ...
else:
     ...
```

Flow Control

```
for item in sequence:

    if expression:
        continue
    else:
        break
```

Functions

```
def foo():
   return 42


def foo(bar):
   return bar


def foo(bar, baz=“fit”):
   return (bar, baz)
```

Classes

```
class Foo(object):
    def __init__(self, bar):
        self.bar = bar
```

Docstrings

```
"Modules can have docstrings."

class Foo(object):

   "Classes can have docstrings too."

   def __init__(self, bar):
      self.bar = bar
```

Exceptions

```
try:
    raise Exception("OH NOES!”)

except:
    log_error()
    raise

else:
    do_something_more()

finally:
    clean_up()
```

Namespaces

```
import logging

from datetime import timedelta

from decimal import Decimal as D


from models import Product
```

Style: PEP-8

- Four-space indents
- lower_case_methods
- CamelCaseClasses
- Line breaks around
  78-79 chars

Installing Packages

- easy_install: easy_install package
- pip: pip install package

Installing Packages

- Installed packages go into a site-packages directory in your Python lib
- But different programs may need different versions of packages…
- So we have virtual environments!

Virtual Environments

- virtualenv
- Creates an isolated Python environment with its own site-packages
- Install whatever you want without fouling anything else up

Activate the Virtual Environment

```
# Mac/Linux/etc...

$ virtualenv myenv
$ source myenv/bin/activate


# Windows

> python virtualenv myenv
> myenv/Scripts/activate.bat
```

What is Django?

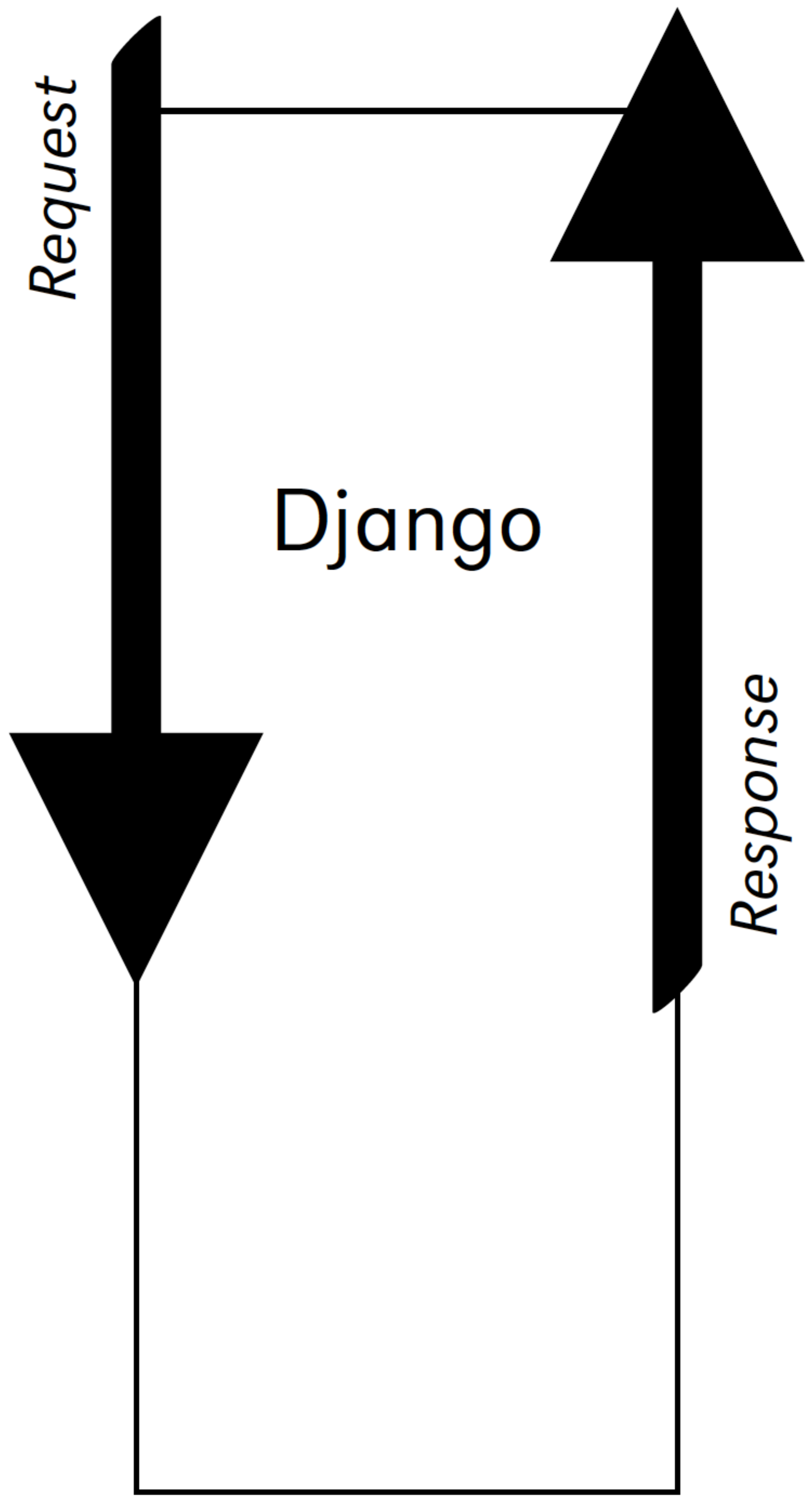
Django?

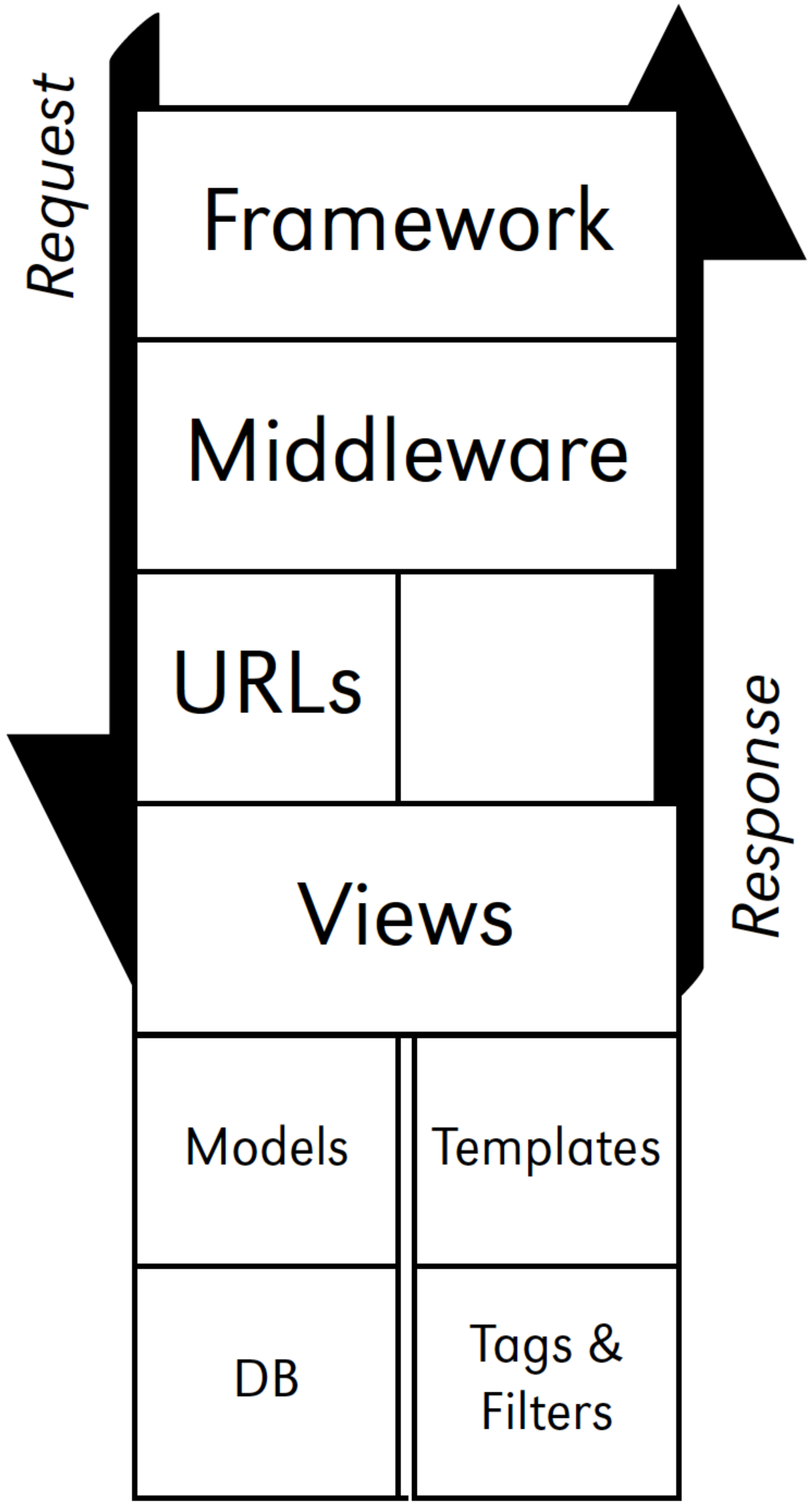
What is Django?

- High-level framework for rapid web development
- Complete stack of tools
- Data modelled with Python classes
- Production-ready data admin interface, generated dynamically
- Elegant system for mapping URLs to Python code
- Generic views' to handle common requests

Django Components

- Think MTV instead of MVC
- Models - Django ORM
- Templates - Django Template Engine
- Views - Python function, Request in Response out
- URL Patterns - Regular expression based


Request
Django
Response


Request
Framework
Middleware
URLs
Views
Response
Models
Templates
DB
Tags & Filters

Defining Requirements

- requirements.txt

```
# Create requirements.txt for current env
$ pip freeze > requirements.txt
```

```
# Install all modules from requirements.txt file recursive
$ pip install -r requirements.txt
```

Starting a Project

```
# Mac/Linux/etc...

$ pip install django
$ django-admin startproject demo
$ cd demo
$ python manage.py migrate
$ python manage.py runserver

# Windows

> pip install django
> python Scripts/django-admin.py startproject demo
> cd demo
> python manage.py migrate
> python manage.py runserver
```

URLs

- Map URLs in requests to code that can be executed
- Regular expressions!
- Subsections of your site can have their own urls.py modules

Views

- Code that handles requests
- Other frameworks often call these “controllers”
- Basically a function that:
  - gets a request passed to it
  - returns text or a response

Questions?