No Holds Barred
Web Framework Battle

ANONYMIZED

No Holds Barred
Web Framework Battle

ANONYMIZED

About this talk

- Compare some Python web frameworks
- Compare them in ways that make sense for me
- My preferences may not be your preferences

Addressing the question of

Bias

Quick review of my Python web work

Django

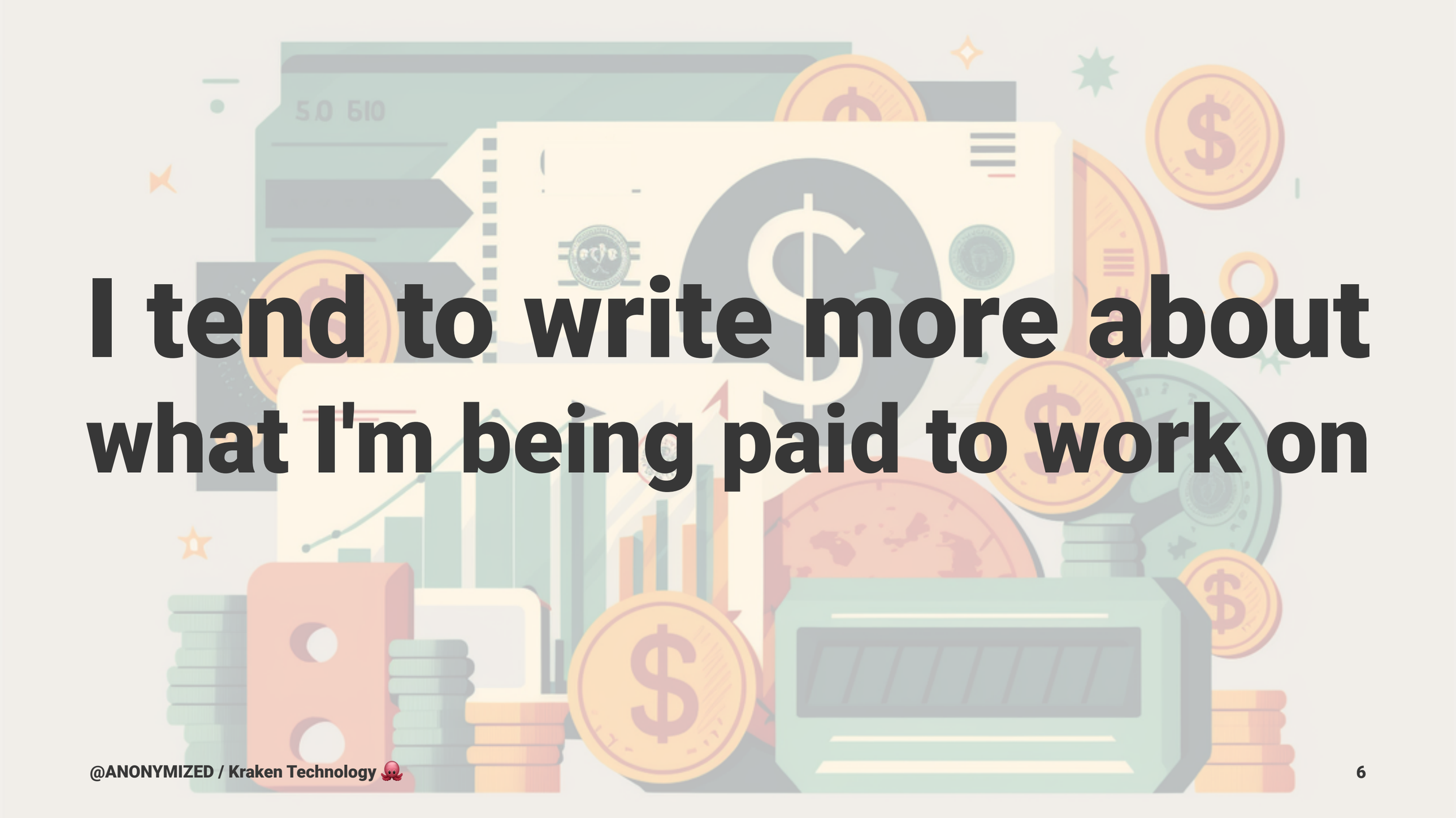
- 233 articles
- Countless open source contributions
- Oodles of production projects

FastAPI / Flask / Pyramid

- 3 FastAPI Articles
- 5 Flask Articles
- 8 Pyramid Articles
- About 15 production projects

Zope/Plone

- 74 Plone articles
- 38 Zope articles
- 2 production projects



I tend to write more about what I'm being paid to work on

Yes, I've written a lot about Django

- After 14 years I know the pain points of Django
- I've used Flask for 12 years
- I've used FastAPI for 3 years
- I did Zope/Plone for 4 years @ NASA 🚀
- I've used Pyramid off-and-on for 17 years
- I used Tornado a few times in 2014

Which frameworks am I covering?

The ones I considered

- Django
- FastAPI
- Flask
- Pyramid
- Tornado

Which frameworks am I covering?

The ones I chose

- Django ✅
- FastAPI ✅
- Flask ✅
- Pyramid
- Tornado

Popularity and usage is at least an order of magnitude higher than other python frameworks.

I like all three frameworks

Django is awesome

FastAPI is awesome

Flask is awesome

You can win with any of these frameworks.

Mad respect to the authors and contributors.

Ready?

Let's dive in

Quick overview of the frameworks

Django

- Released: 2005
- Full-stack web framework
  - Web-serving, databases, HTML generation
- Bundled with ORM & form validation
- Admin tool driven by ORM & form systems
- Larger than the other frameworks

FastAPI

- Released: 2018
- Almost Microframework
  - Data validation: tightly integrated with pydantic
- Uses type annotations as a superpower
- Async from the start

Flask

- Released: April 1, 2010
- Microframework
  - Minimal core, lots of extensions
- Very popular with SaaS vendors for tutorials

Points of Comparison

- Developer experience
- Speed
- Async support
- Databases
- Governance
- Sweet spots

Developer Experience

- **Getting started**
- **Data validation**
- **Small projects**
- **Large projects**

Developer Experience

Getting started

Developer Experience

Getting started

Django 👎

```python
# urls.py
# Not including a bunch of setup
# No one builds Django projects this way
from django.urls import path
from django.views.generic import View

def index(request):
    return "Index Page"

def hello(request):
    return "Hello, World"

urlpatterns = [
    path('', index, name='home'),
    path('/hello', hello, name='home'),
]
```

FastAPI 👍

```python
# app.py
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def index():
    return {"Index": "Page"}

@app.get("/hello")
def hello():
    return {"Hello": "World"}
```

Flask 👍

```python
# app.py
# Flask can also return
# JSON by returning dict
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'Index Page'

@app.route('/hello')
def hello():
    return 'Hello, World'
```

Developer Experience:

Getting started

Django 👎

- No such thing as a quickstart in Django
- Tutorial is good, just takes time

FastAPI 👍 👎

- Easy to get started
- Quickstart needs work

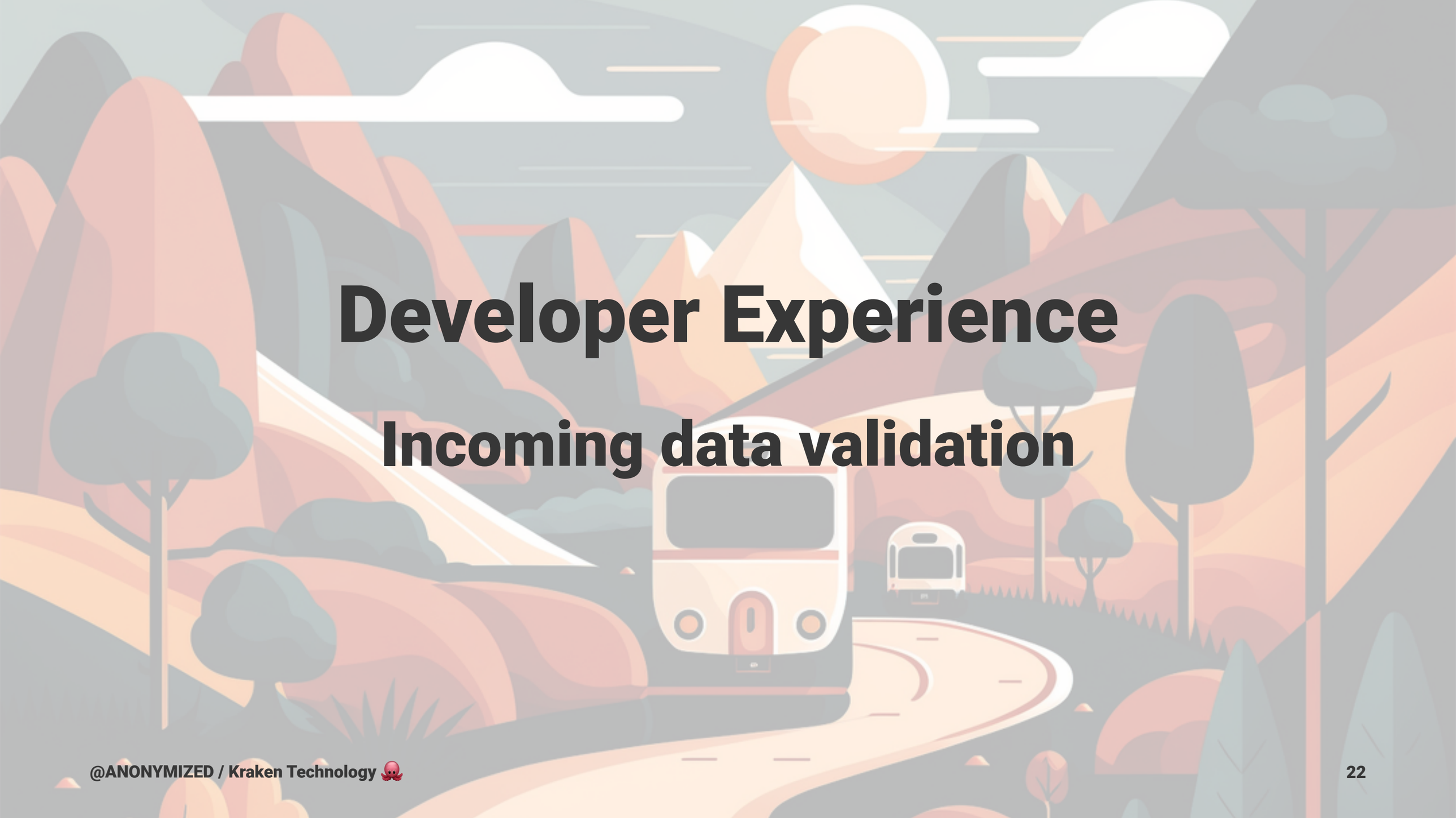
Flask 👍 👍

- Easy to get started
- Flask's quickstart is amazing
- Quickstart covers everything many small projects need
- Quickstart is technically a cheatsheet

Developer Experience:

Getting started

Winner: Flask



Developer Experience

Incoming data validation

Developer Experience

Incoming data validation

Django 👍

```python
from django import forms
from django.views.generic import CreateView
from django.contrib.auth.mixins import (
    LoginRequiredMixin
)

from .models import Cheese


class CheeseForm(forms.Form):
    name = forms.CharField(max_length=100)
    description = forms.CharField(
        max_length=300,
        required=False
    )
    age = forms.IntegerField(
        default=0
        min_value=0,
        description="Age in months"
    )


class CheeseCreateView(LoginRequiredMixin, CreateView):
    model = Cheese
    form_class = CheeseForm

    def form_valid(self, form):
        form.instance.creator = self.request.user
        return super().form_valid(form)
```

FastAPI 👍

```python
from fastapi import FastAPI
from pydantic import BaseModel, Field

app = FastAPI()


class Cheese(BaseModel):
    name: str
    description: str = Field(
        default=None,
        title="The description of the item",
        max_length=300)
    age: int = Field(
        default=0,
        gt=0,
        description="Age in months"
    )

@app.post("/cheeses/")
async def update_item(cheese: Cheese):
    return Cheese
```

Flask 👎

- No built-in validation
- flask-wtf is awesome, but no mention in core docs
- Documentation doesn't encourage data validation

Developer Experience

Incoming data validation

Django 👍

- Comes with built-in validation
- Powered by Django's Forms and ORM
- Edge cases require digging
- Encouraged, but not forced

FastAPI 👍 👍

- Comes with built-in validation
- Powered by the pydantic library, which is awesome
- Type annotations makes it intuitive
- Forces use of validation

Flask 👎

- No built-in validation
- Documentation doesn't encourage data validation
- flask-wtf is awesome, but no mention in core docs

Developer Experience

Data Validation

Winner: FastAPI

Developer Experience

Small projects

Developer Experience

Small projects

Django 👍👎

- Admin tool is great for small projects with SQL databases
- Overkill for projects without SQL databases
- Single file apps require advanced knowledge of Django

FastAPI 👍👍

- Light and easy to get started
- One file apps are easy to build
- Flexibility on databases is a virtue
- Built-in validation
- Built-in REST API is awesome

Flask 👍

- Light and easy to get started
- One file apps are easy to build
- Flexibility on databases is a virtue

Developer Experience

Small projects

Winner: FastAPI

Developer Experience

Large projects

Developer Experience

Large projects

Django 👍 👍

- "app" structure forced by Framework
- Described in core tutorial
- Each app supports a specific feature
- Each app has its own models, views, forms, etc.
- Borrowed from POSIX philosophy

Flask 👍

- Core docs
- Each blueprint supports a specific feature
- Each blueprint has its own models, views, forms, etc.
- In my experience, rarely used

FastAPI 👍 👎

- No guidance provided
- Framework design encourages separation of concerns

Developer Experience

Large projects

Winner: Django

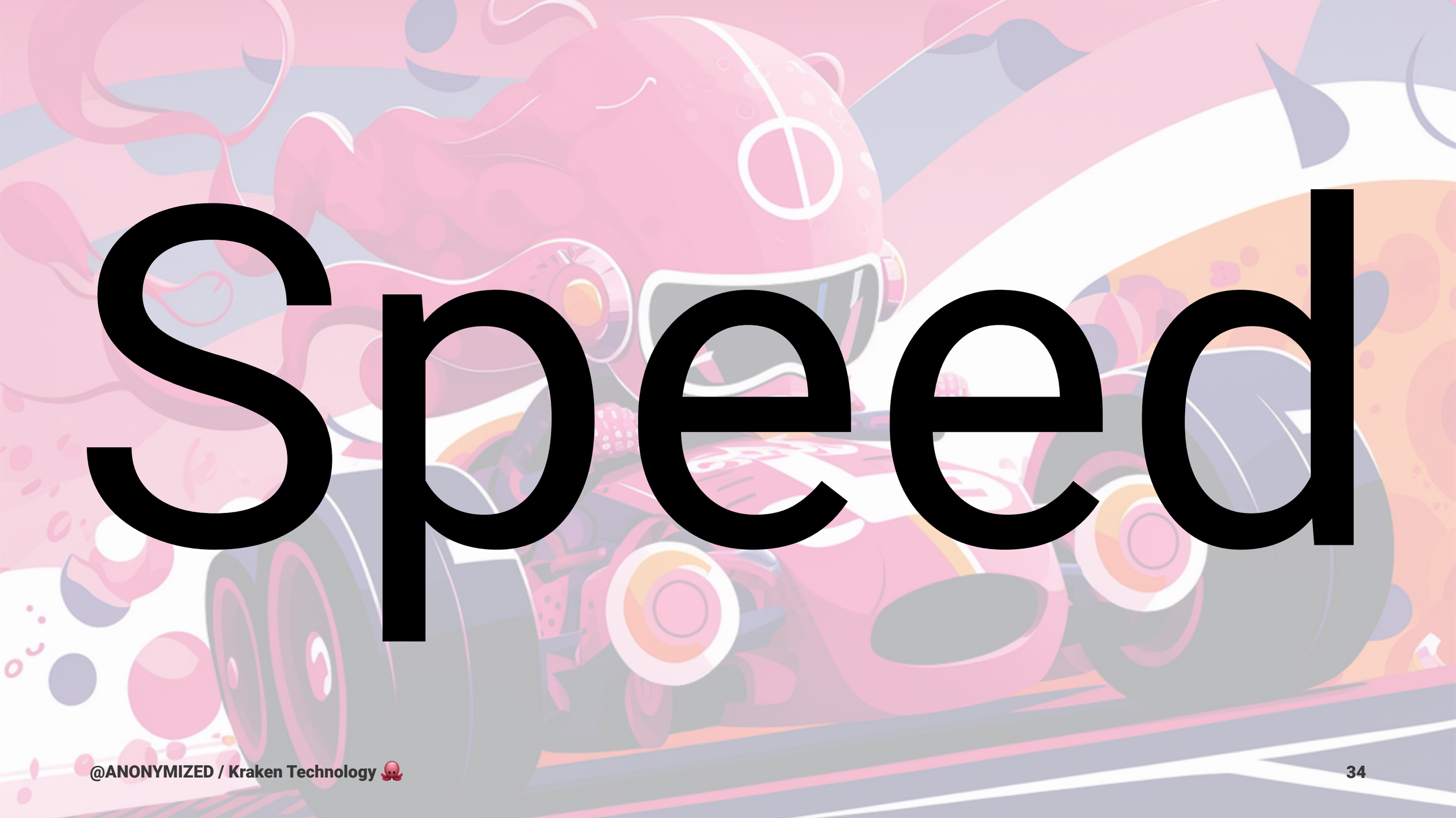
Developer Experience Summary

Category	Django	FastAPI	Flask
Getting started			✅
Data validation		✅	
Small projects		✅	
Large projects	✅		

Developer Experience

Aggregate

Winner: FastAPI


Speed

About speed metrics

- Often the database is the bottleneck real projects face
- Indexing and caching are often the best ways to improve performance rather than changing the framework

"Lies, Damn Lies and Statistics"

- Different benchmarks use different tools
- Conditions in tests are often misleading
- Example: Comparing **Django** with queries to **FastAPI/Flask** with no database access


Speed
JSON Serialization

Speed

JSON Serialization

Each response is a JSON serialization of a freshly-instantiated object that maps the key message to the value Hello, World!

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 28
Server: Example
Date: Wed, 17 Apr 2013 12:00:00 GMT

{"message":"Hello, World!"}
```

Speed

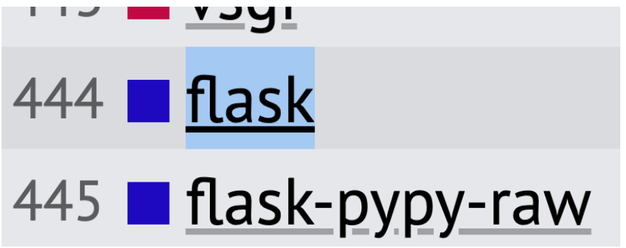
JSON Serialization

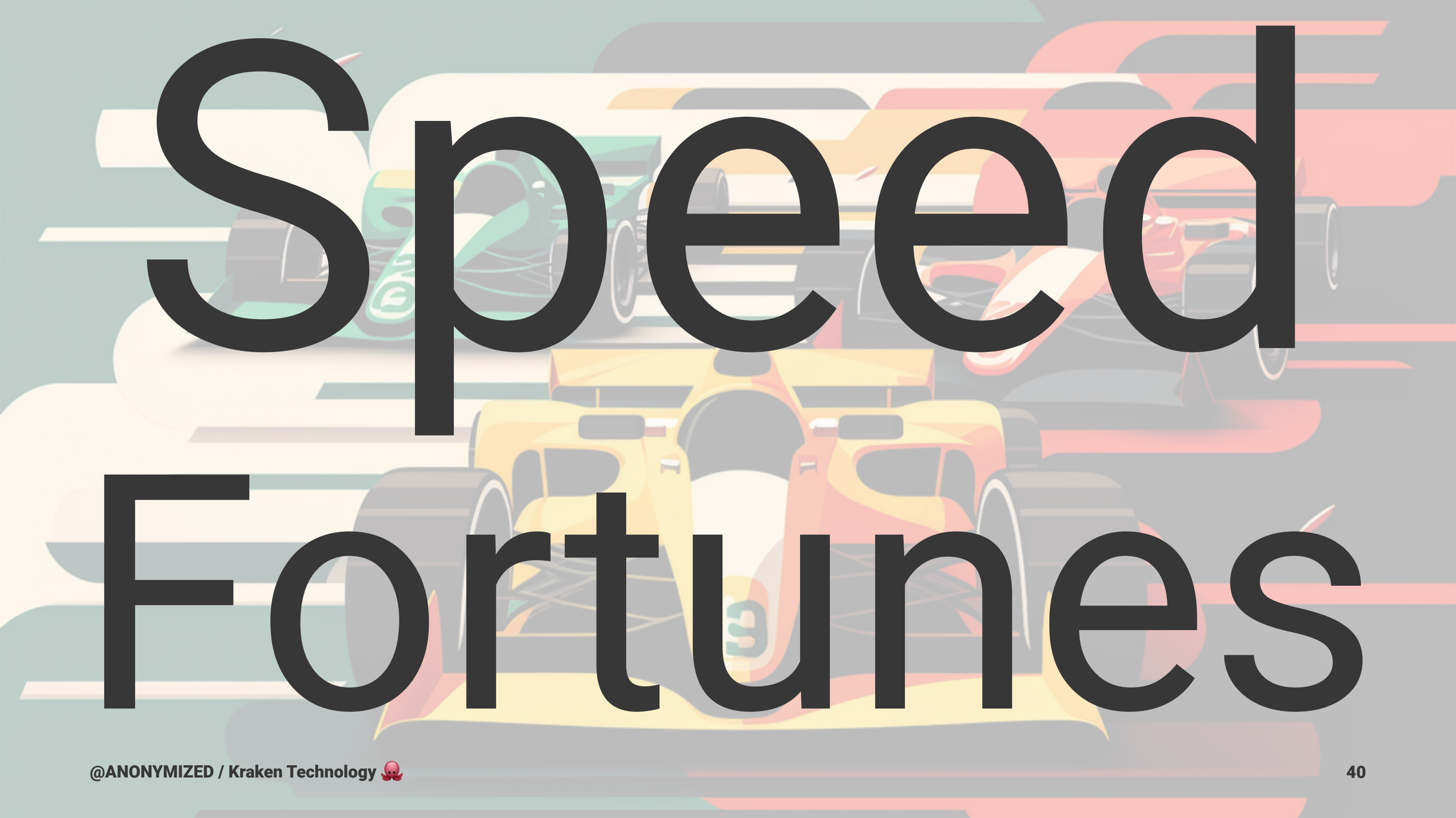
techempower.com/benchmarks/#section=data-r21&test=json

Django 👍

FastAPI 👍👍

Flask ❓ ❗




Speed
Fortunes

Speed

Fortunes

- The framework's ORM is used to fetch all rows from a database table containing an unknown number of Unix fortune cookie messages (the table has 12 rows, but the code cannot have foreknowledge of the table's size).
- An additional fortune cookie message is inserted into the list at runtime and then the list is sorted by the message text.
- The list is delivered to the client using a server-side HTML template.
- The message text must be considered untrusted and properly escaped and the UTF-8 fortune messages must be rendered properly.
- Whitespace is optional and may comply with the framework's best practices.

```
HTTP/1.1 200 OK
Content-Length: 1196
Content-Type: text/html; charset=UTF-8
Server: Example
Date: Wed, 17 Apr 2013 12:00:00 GMT

<!DOCTYPE html><html><head><title>Fortunes</title></head><body>...
```

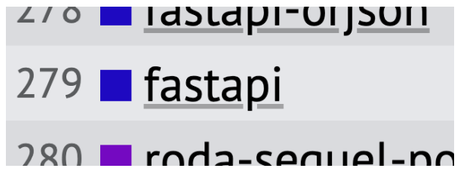
Speed

Fortunes

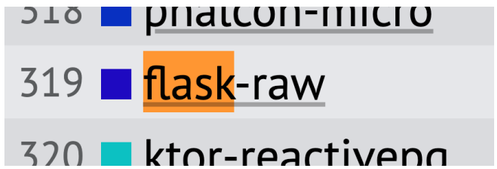
techempower.com/benchmarks/#section=data-r21&test=fortunen

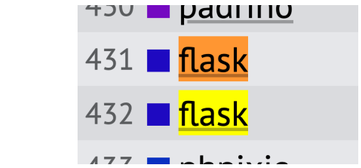
Django 👍

FastAPI 👍👍



Flask 👍 ❓






Speed
Composite

Speed

Composite

techempower.com/benchmarks/#section=data-r21&test=composite

Django 👍

FastAPI 👍

Flask 👍👍

Apples and Oranges

So many variables that it's easy to get lost in the weeds.

1. FastAPI is really fast with JSON serialization
2. Flask tends to have better composite scores
3. Django run as a single file app without middleware, context processors, etc is comparable to Flask and FastAPI

Speed Summary

Source: techempower.com/benchmarks/

Django 👎

- Usually slowest of the big three Python frameworks
- Removing critical components to increase speed is a fool's errand

FastAPI 👍

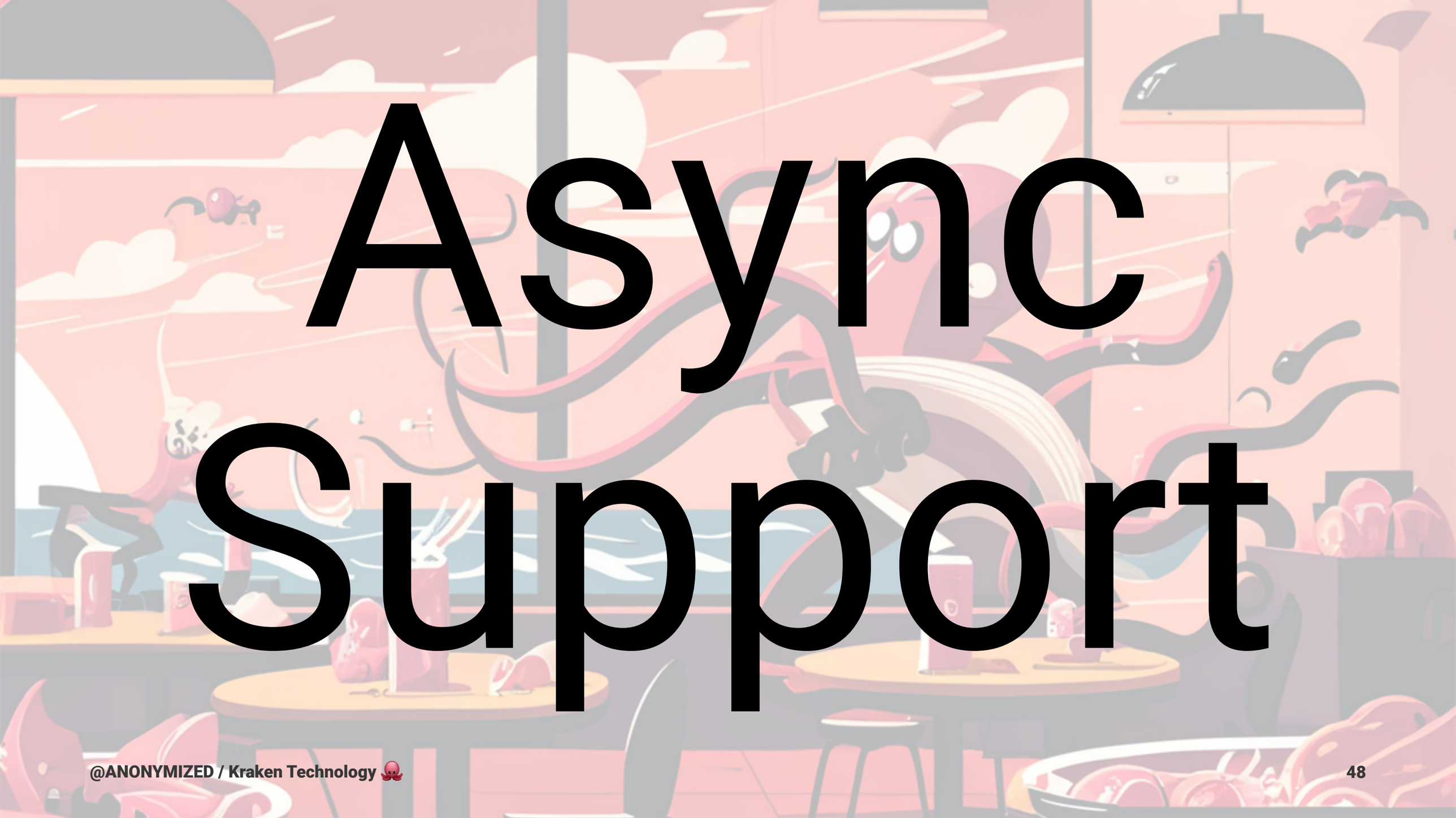
- Fastest JSON serialization
- Fastest async
- Will get a boost with advent of pydantic 2.0

Flask 👍

- Faster with templates & queries

Speed

Winner: Flask & FastAPI


Async
Support

Async Support

Why it matters

- Websockets
- Blocking I/O
  - File streaming
  - Slow APIs
  - Tarpits

Async Support

Django 👍

- Herculean effort to bring async support to Django
- Not for beginners
- Docs cover the fundamentals, but not much else
- Nifty `sync_to_async()` decorator
- Much of ecosystem is synchronous

FastAPI 👍 👍

- Designed from the outset to support async
- All of the framework is async by default
- Good documentation

Flask 👍

- Slow, hence recommendation to use Quart instead
- Modern Flask supports async
- Light documentation
- Much of Flask and ecosystem is synchronous

Async

Winner: FastAPI

Databases

Databases I like to use

- Relational Databases (SQL for short)
- DynamoDB, specifically single-table designs
- Firebase, especially in hackathons

Databases

Django 👍

- Designed for SQL databases
- Key/value stores used for caching
- Universality of SQL empowers the package ecosystem
- Using No-SQL is always a mistake

FastAPI 👍

- Use whatever you want
- Or skip the database entirely

Flask 👍

- Use whatever you want
- Or skip the database entirely

Databases

Winner: Tie

TODO

Add Governance section here


Sweet Spots

Sweet Spots

Django 👍👍

- Large projects
- Admin
- Third-party packages
- Django Templates + HTMX

FastAPI 👍👍

- REST/JSON APIs
- Small projects
- Async
- Elegant design

Flask

- Small projects

Sweet Spots

Winner: Django & FastAPI

Conclusion

(what do I use on new projects?)

I like all three frameworks

Django is awesome

FastAPI is awesome

Flask is awesome

You can win with any of these frameworks.

Mad respect to the authors and contributors.

There can only be one!

Django or FastAPI or Flask?

Category	Django	FastAPI	Flask
Developer Experience		✅	
Performance		✅	✅
Async Support		✅	
Databases	✅	✅	✅
Governance	✅		✅
Sweet Spots	✅	✅	

Grand Winner

Winner: FastAPI

Where FastAPI can improve:

- Quickstart
- Large projects
- Governance

About Me

ANONYMIZED

Code @ Kraken Technologies 🐙

Author, Coder, Leader

Husband of ANONYMIZED

Father

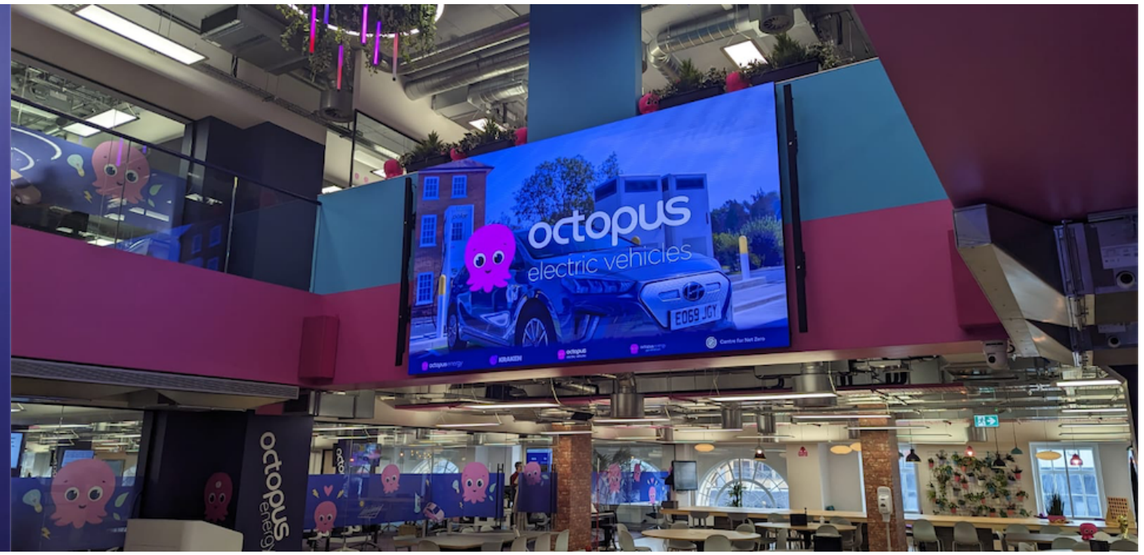
Superhero saving the planet

ANONYMIZED

Looking for work that

makes a difference?


Do the best you can until you know better.
When you know better, do better.
MAYA ANGELOU


octopus
electric vehicles
E069 JCY
octopus energy

Join me in the battle against climate change

Octopus Energy Group is hiring

octopus.energy/careers


Questions?