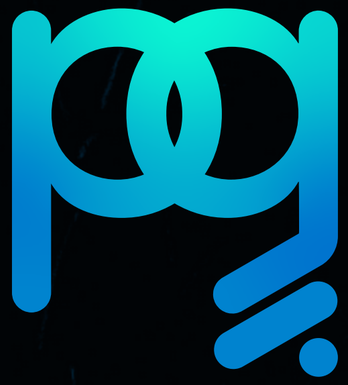


playgrounds

Learn
Play
Discover
Grow

Providing interactive education hubs, yield simulation environments, and advanced data analytics tools for the Olympus and Klima ecosystems.

playgrounds

Get to Know us

We are a cross DAO team working on enhancing the learning experience for our respective communities

Olympus + Klima

Playgrounds started as a simple simulation tool, but has grown into something much more impactful. We are enhancing the shared mental model of both communities

Get to Know us

We are engineers, scientists, designers, and educators with a shared vision on the future of DeFi and Web3

playgrounds

+

Protean Labs

Together with Protean Labs, we are building interactive learning hubs, advanced simulation environments, and powerful yet accessible on-chain data analytics tools for the entire DeFi landscape

playgrounds

Problem

There is a great chasm between natives and non-natives

We seek to bridge this gap.

01	**Olympus and Klima mechanics are dense**	We need to simplify the fundamentals and lower the learning curve
02	**Difficult onboarding process for non-natives**	We need a friendlier system to simplify new user onboarding
03	**Paradox of Choices**	There are many information sources. We need a reliable source of truth

playgrounds

Solution

Learn.Play.Discover.Grow

- Simplify fundamental concepts
- Gamify the learning experience
- Provide yield simulation environments
- Create tools for insight and discovery

SOLUTION 01

Breakdown complex but important protocol mechanics

SOLUTION 02

Provide an advanced but approachable simulation environment to enhance understanding

SOLUTION 03

Reward curiosity with beautiful POAPs / NFT's

SOLUTION 04

Provide advanced yet accessible data exploration tools for meaningful on-chain data analysis, insight generation and story telling

playgrounds

Our Journey

Our vision is our NorthStar: Provide an interactive learning hub and advanced simulation environment for the Olympus and Klima ecosystems

Our mission and how we'll do it:

- Identify the needs of our communities
- Curate the content and learning experience for our users
- Build impactful solutions and generate value for our DAOs

Phase 1 Olympus Playgrounds

Launched Olympus Playgrounds on 10/20/2021

Phase 2 Klima Playgrounds

Launch Klima Playgrounds in Q1 2021

Phase 3 Playgrounds v2 and PG Analytics v1

Launch V2 infrastructure in Q2 2022 and PG Analytics V1

Phase 4 Olympus and Klima Playgrounds V2

Launch Olympus and Klima Playgrounds v2 along with Classrooms MVP

Playgrounds Features

Learning: teach concepts of Olympus, Playgrounds, and greater DeFi

Simulations: allow users to develop strategies and estimate returns on investments

Playgrounds Analytics: allow transformation and visualization of on-chain activities

Learning Hub

- DeFi explainers
- Ecosystem guides
- Playground guides
- Classrooms

Simulation Environments

- Protocol interaction simulator
- Leverage Simulator
- Market dynamics and yield management Simulator

Playgrounds Analytics

- Model based data visualization
- Accessible dashboards
- Dynamic document generation
- Personal metrics analytics tools

Playgrounds Analytics

by

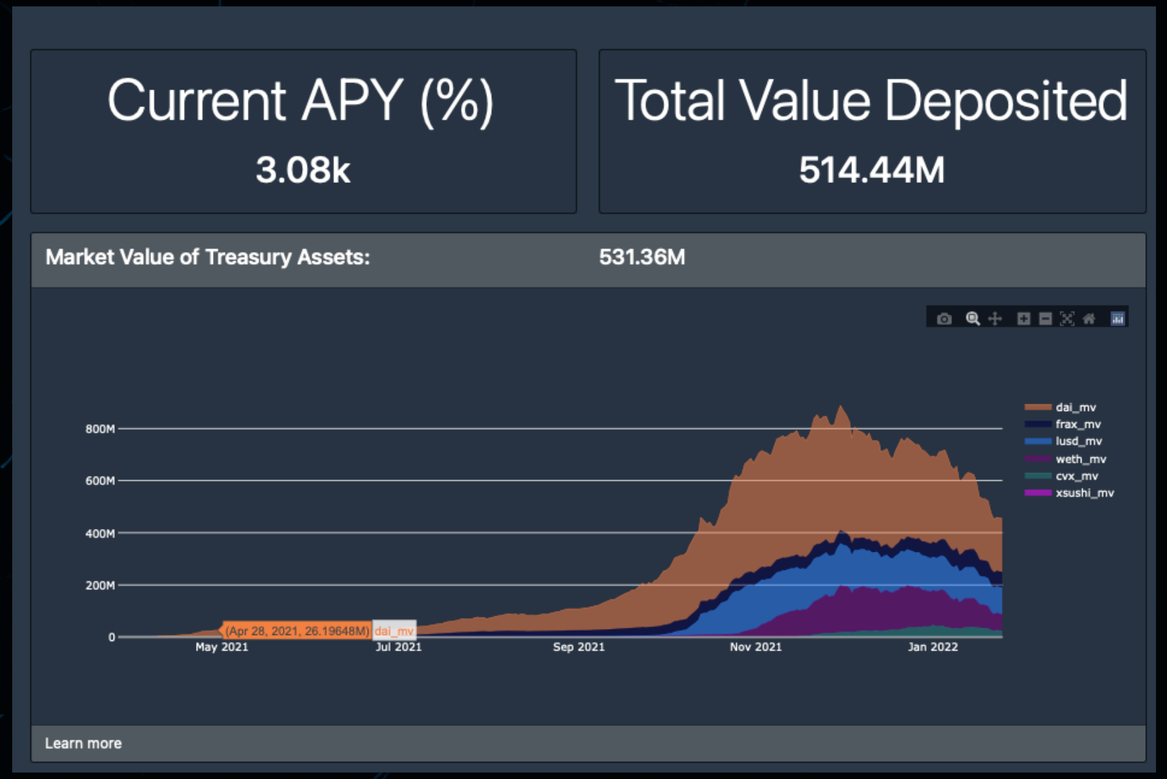
Protean Labs

playgrounds

PG Analytics quick overview

PG Analytics is the on-chain data analytics as a service component of Playgrounds.

By leveraging our experience in data engineering, data science, and our expertise with the graph network, along with our in-house on-chain data transformation tool called subgrounds, you can now build powerful analytics tools and data dashboards in pure python



playgrounds

EVM
Eth, Polygon, Arbitrium

Substrate
Kusama, Polkadot

Cosmwasm
Terra, Osmosis, Cosmos

Anchor
Solana

Data in it's raw form
- Opaque
- Entangled
- Encoded
- Non-human readable

The Graph

- Indexes on-chain data from any of these chains into specified data models
- Converts raw on-chain data into human readable data models

Subgrounds

- Enables interaction and interoperability of indexed data
- Enables model based transformations
- Provide powerful wrappers for Plotly and dash data visualizations

Playgrounds Analytics

- Optimizes the creation and dissemination of knowledge that was mined on-chain
- Leverages Subgrounds for personalized visualization

Transformed data
- Transparent
- Meaningful
- Human readable
- Story telling

Playgrounds Analytics: Data Infrastructure

Subgrounds summary

- Enables manipulations and reflect the domain in which they are defined.
- Highly extensible, modular, and provides continuity with existing data analytics tools
- Minimally verbose and significantly reduces on-chain analytics learning curve
- Entirely based on subgraph schemas made available through The Graph
- Built in Plotly wrappers enable model based transformation and visualization of on-chain data
- Provides accessible dashboards which can either be auto generated or customized to varying degrees

playgrounds

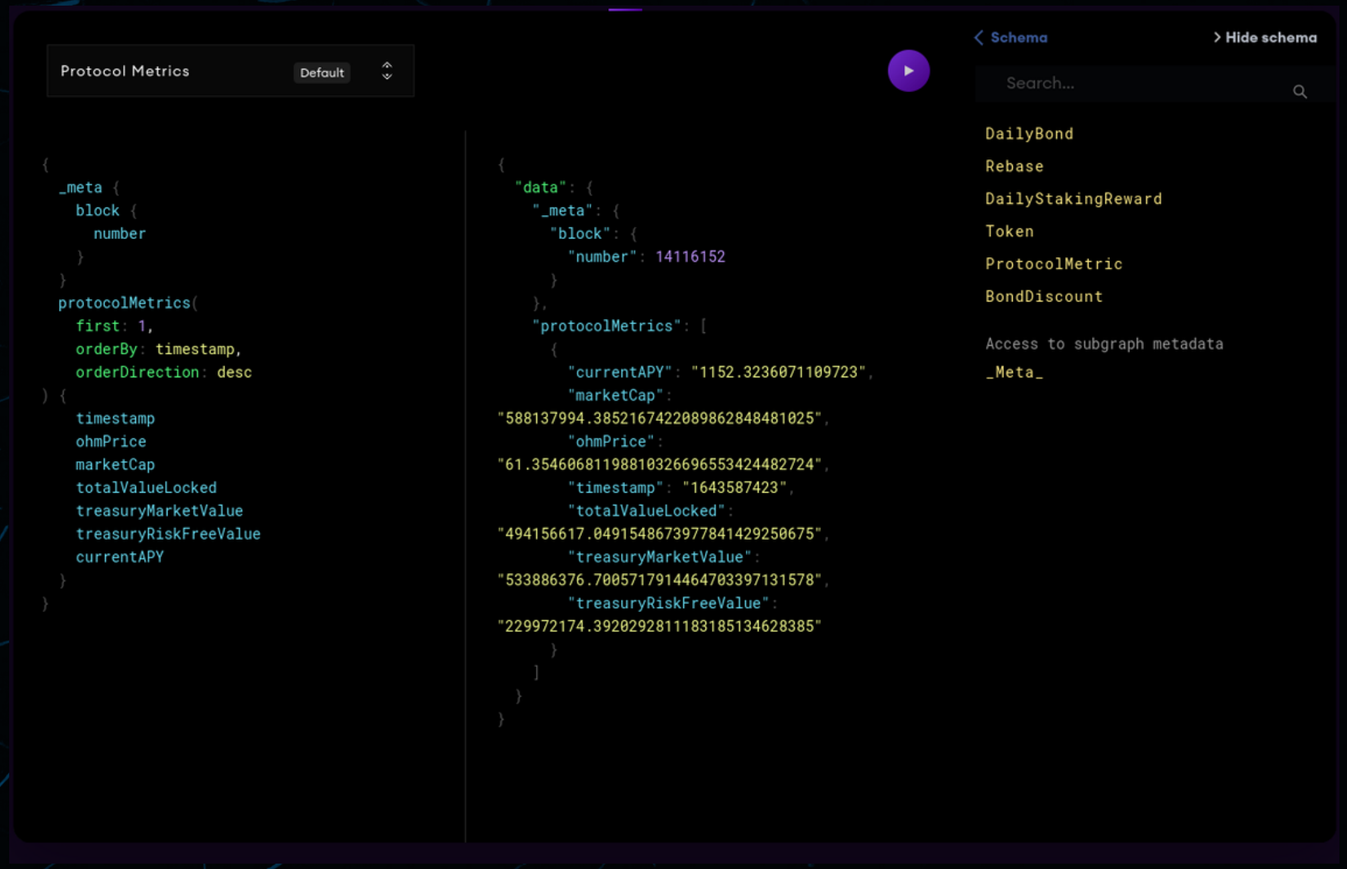
Subgrounds

The Graph (TL;DR)

The Graph is a decentralized blockchain indexing services.

The indexing service is analogous to an ETL (Extract, Transform and Load) service where developers:

1. Write subgraphs (ETL recipe),
2. Deploy those subgraphs to a graph-node (which performs the actual ETL).
3. Query the data through a GraphQL API served by the graph node whose schema is defined as part of the subgraph.



Subgraphs

Three components to a subgraph:

1. The schema: Defines the API that will be queryable once the ETL process is started. Written in GraphQL.
2. The subgraph manifest: Defines the subgraph’s data sources (i.e.: smart contracts) and data source templates, as well as the events and/or function calls that a graph node should listen for and the handler(s) to trigger whenever such events or/and function calls are detected. Written in YAML.
3. The handlers: Arbitrary code that is executed whenever their respective trigger (e.g.: smart contract event or function call) defined in the manifest is detected by the graph node. Written in assembly script.

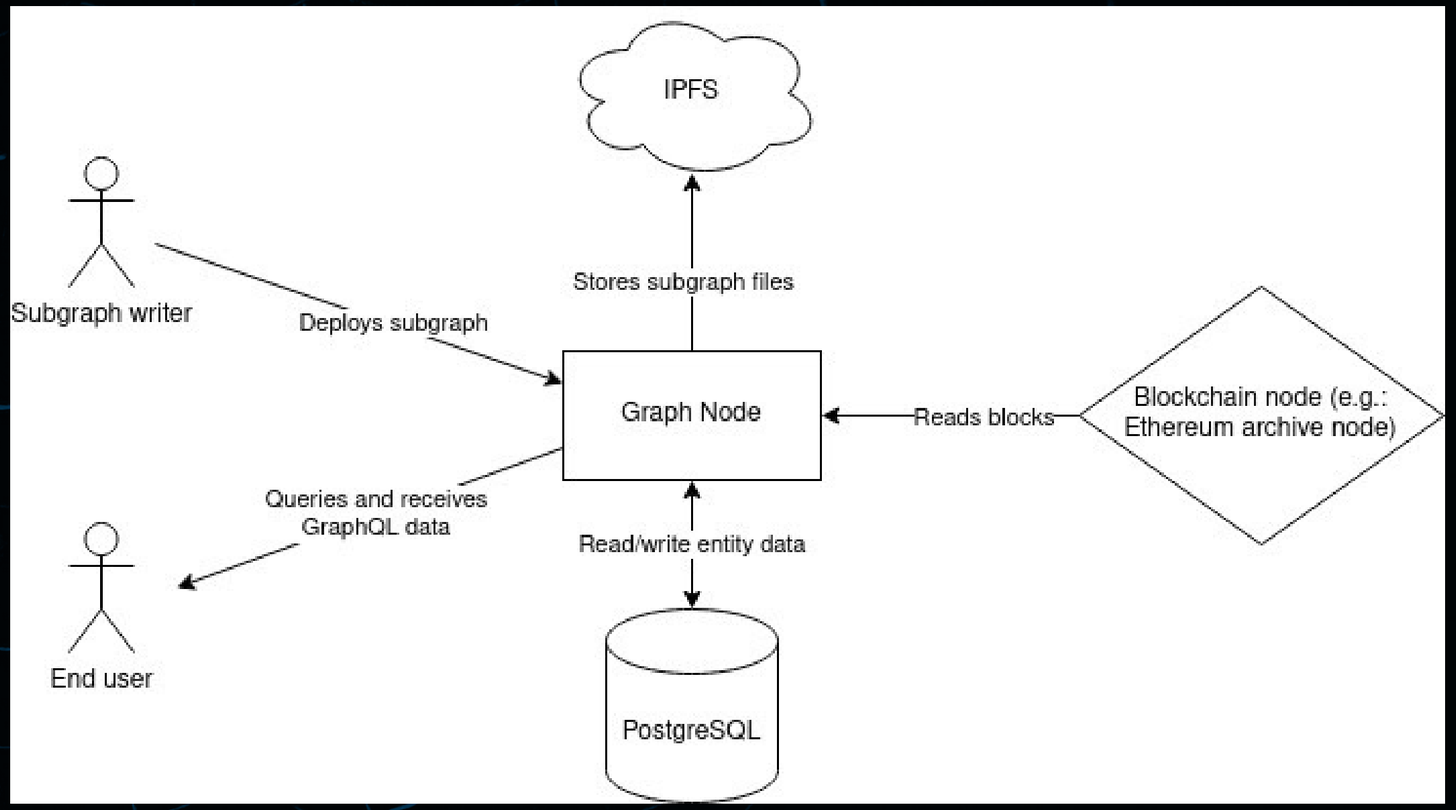
Subgraphs

Handlers can:

- Modify existing entities
- Create new entities
- Delete entities
- Instantiate data source templates
- Read **any** blockchain or smart contract data (but no mutable interactions!)

Graph nodes

- Receives subgraphs
- Initializes PostgreSQL database based on GraphQL schema (one entity type == one table)
- Reads blocks sequentially(!) and executes handlers when their associated events and/or function calls are detected
- Provides read-only GraphQL API to access stored entities



Accessing a graph node

Three options

The Graph’s hosted service

- Pros: Free, no infrastructure required, can index 24 different networks
- Cons: Sometimes fails during peak request hours, will be eventually discontinued

The decentralized service

- Pros: Decentralized, can query past versions of subgraphs
- Cons: Must pay a fee per query and for deployment, only supports limited number of networks

Self-hosted node

- Pros: Not dependent on others for availability, good for testing subgraphs
- Cons: Must have access to fully synced archive nodes (or equivalent), can add some management overhead

Subgrounds

Main features:

- Automatic GraphQL schema introspection and class generation
- Type-safe queries
- Automatic json data formatting into DataFrames
- SyntheticFields (more on this later)
- Plug-and-play functionality with Dash and Plotly

Subgrounds: Loading a subgraph

Subgrounds parses the subgraph's schema and generates classes that match the schema entities and their fields and arguments

```
from subgrounds.subgrounds import Subgrounds

sg = Subgrounds()
olympuspm = sg.load_subgraph('https://api.thegraph.com/subgraphs/name/drondin/olympus-protocol-metrics')

last_metric = olympuspm.Query.protocolMetrics(
  orderBy=olympuspm.ProtocolMetric.timestamp,
  orderDirection='desc',
  first=1
)

sg.query(last_metric.currentAPY)
```

✓ 11.5s

```
1151.6513086199332
```

```
type ProtocolMetric @entity {
  id: ID!
  timestamp: BigInt!
  ohmCirculatingSupply: BigDecimal!
  sOhmCirculatingSupply: BigDecimal!
  totalSupply: BigDecimal!
  ohmPrice: BigDecimal!
  marketCap: BigDecimal!
  totalValueLocked: BigDecimal!
  treasuryRiskFreeValue: BigDecimal!
  treasuryMarketValue: BigDecimal!
  nextEpochRebase: BigDecimal!
  nextDistributedOhm: BigDecimal!
  treasuryDaiRiskFreeValue: BigDecimal!
  treasuryFraxRiskFreeValue: BigDecimal!
  treasuryLusdRiskFreeValue: BigDecimal!
  treasuryWETHRiskFreeValue: BigDecimal!
  treasuryDaiMarketValue: BigDecimal!
  treasuryFraxMarketValue: BigDecimal!
  treasuryLusdMarketValue: BigDecimal!
  treasuryUstMarketValue: BigDecimal!
  treasuryXsushiMarketValue: BigDecimal!
  treasuryWETHMarketValue: BigDecimal!
  treasuryWBTCMarketValue: BigDecimal!
  treasuryCVXMarketValue: BigDecimal!
  treasuryOtherMarketValue: BigDecimal!
  treasuryLPValue: BigDecimal!
  treasuryStableBacking: BigDecimal!
  treasuryVolatileBacking: BigDecimal!
  treasuryTotalBacking: BigDecimal!
  currentAPY: BigDecimal!
```

Subgrounds: Type-safe queries

Invalid queries raise an exception before sending the query instead of making a useless roundtrip to the graph node

```
from subgrounds.subgrounds import Subgrounds

sg = Subgrounds()
olympuspm = sg.load_subgraph('https://api.thegraph.com/subgraphs/name/drondin/olympus-protocol-metrics')

last_metric = olympuspm.Query.protocolMetrics(
  orderBy=olympuspm.ProtocolMetric.timestamp,
  orderDirection='desc',
  first=1
)

sg.query(last_metric.potato)
```

⊗ 0.5s

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
File ~/Documents/Programming/subgrounds/subgrounds/subgraph.py:545, in FieldPath.__getattribute__(self, _Fi
    544 try:
--> 545   return super().__getattribute__(__name)
    546 except AttributeError:

AttributeError: 'FieldPath' object has no attribute 'potato'
```

Subgrounds: Formatting DataFrames

Querying into a DataFrame is seamless

```
from datetime import datetime

from subgrounds.subgraph import SyntheticField

# SyntheticField using Python operators
olympuspm.ProtocolMetric.mkt_cap_tvl_ratio = olympuspm.ProtocolMetric.marketCap / olympuspm.ProtocolMetric.totalValueLocked

# SyntheticField using constructor
olympuspm.ProtocolMetric.datetime = SyntheticField(
  lambda t: str(datetime.fromtimestamp(t)),
  SyntheticField.STRING,
  olympuspm.ProtocolMetric.timestamp
)

sg.query_df([
  last365_metrics.datetime,
  last365_metrics.ohmPrice,
  last365_metrics.marketCap,
  last365_metrics.totalValueLocked,
  last365_metrics.mkt_cap_tvl_ratio
])
```

✓ 0.7s

	protocolMetrics_datetime	protocolMetrics_ohmPrice	protocolMetrics_marketCap	protocolMetrics_totalValueLocked	protocolMetrics_mkt_cap_tvl_ratio
0	2022-01-30 19:03:43	61.716801	5.916204e+08	4.971926e+08	1.189922
1	2022-01-29 19:02:01	62.127117	5.913815e+08	4.979960e+08	1.187523
2	2022-01-28 19:00:04	63.395624	5.989934e+08	5.037363e+08	1.189101
3	2022-01-27 19:04:00	62.890826	5.895667e+08	4.945564e+08	1.192112
4	2022-01-26 19:00:23	63.611329	5.913067e+08	4.968437e+08	1.190126
...	...	...	...	...	...
195	2021-10-28 00:36:51	1199.129678	4.036614e+09	3.693584e+09	1.092872
196	2021-10-26 23:28:03	1083.235421	3.592189e+09	3.268887e+09	1.098903
197	2021-10-25 22:47:10	1100.348136	3.589812e+09	3.271797e+09	1.097199
198	2021-10-24 22:10:09	1209.098495	3.887113e+09	3.549793e+09	1.095025
199	2021-10-23 21:15:41	1182.364633	3.743896e+09	3.413541e+09	1.096778

200 rows × 5 columns

Subgrounds: Synthetic Fields

Powerful construct to define transformations on subgraph entities pre-querying

<u>Analogous to SQL views</u>

Note: The code example follows from the previous slide's code

```
from subgrounds.subgrounds import Subgrounds

sg = Subgrounds()
olympuspm = sg.load_subgraph('https://api.thegraph.com/subgraphs/name/drondin/olympus-protocol-metrics')

last365_metrics = olympuspm.Query.protocolMetrics(
  orderBy=olympuspm.ProtocolMetric.timestamp,
  orderDirection='desc',
  first=365
)

sg.query_df([
  last365_metrics.timestamp,
  last365_metrics.ohmPrice,
  last365_metrics.totalSupply,
  last365_metrics.marketCap,
  last365_metrics.totalValueLocked
])
```

✓ 1.8s

	protocolMetrics_timestamp	protocolMetrics_ohmPrice	protocolMetrics_totalSupply	protocolMetrics_marketCap	protocolMetrics_totalValueLocked
0	1643587423	61.716801	1.049962e+07	5.916204e+08	4.971926e+08
1	1643500921	62.127117	1.043247e+07	5.913815e+08	4.979960e+08
2	1643414404	63.395624	1.036207e+07	5.989934e+08	5.037363e+08
3	1643328240	62.890826	1.028802e+07	5.895667e+08	4.945564e+08
4	1643241623	63.611329	1.020919e+07	5.913067e+08	4.968437e+08
...	...	...	...	...	...
307	1617064373	821.354482	7.359546e+04	5.416297e+07	4.796628e+07
308	1616977978	951.124564	7.202087e+04	6.122283e+07	5.450741e+07
309	1616918609	1069.482050	7.118761e+04	6.795022e+07	6.038699e+07
310	1616830340	1043.623462	7.063727e+04	6.573292e+07	5.801410e+07
311	1616743300	647.739952	6.974178e+04	4.021804e+07	3.390833e+07

312 rows × 5 columns

Subgrounds: Plug and Play visualization

```python
from subgrounds.plotly_wrappers import Figure, Scatter
from subgrounds.dash_wrappers import Graph

# Dashboard
app = dash.Dash(__name__)

app.layout = html.Div(
  html.Div([
    html.H4('Entities'),
    html.Div([
      Graph(Figure(
        subgrounds=sg,
        traces=[
          Scatter(x=last365_metrics.datetime, y=last365_metrics.marketCap, name='Market cap'),
          Scatter(x=last365_metrics.datetime, y=last365_metrics.totalValueLocked, name='TVL'),
        ]
      ))
    ])
  ])
)

if __name__ == '__main__':
  app.run_server(debug=True)
```

Demo

This presentation was made with love by