Lightning Talks

Stefan Keller

From Code to DB: How to make Pythons and Elephants dance together!

Swiss Python Summit

PostgreSQL

From Code to Database Queries: How to Make Pythons and Elephants Dance Together

Lightning Talk

Prof. Stefan Keller

Institute for Software, FH OST Campus Rapperswil, ost.ch/ifs

From Code to Database Queries: How to Make Pythons and Elephants Dance Together

Introduction

- Connecting Python applications to databases like PostgreSQL
  - is a common yet complex task,
  - given the power of modern SQL
  - and the choice of existing libraries.

- This talk evaluates top 7 Python software libraries
  - that make it easy to connect Python to databases like PostgreSQL,
  - using 7 criteria such as lightweightness, Pythonic style, type-safety, SQL-like query building, result handling, SQL dialect support (especially, but not only PostgreSQL), Pandas integration.
  - (Not considered: code synchronization and schema evolution support.

The top 7 Python software libraries

- SQLAlchemy Core (requires psycog2 for PG)
  - GitHub stars: ~7,000+. First released in 2005
  - A widely-used SQL toolkit and Object-Relational Mapper (ORM) that provides full control over SQL expressions and database management.
- PyPika (complements psycog2 for PG)
  - GitHub stars: ~2,200. Released in 2017
  - PyPika is a pure SQL query builder focused on providing expressive query generation.
- Records (requires psycog2 for PG)
  - GitHub stars: ~3,000. Released in 2016
  - A simple wrapper for making database queries, emphasizing straightforward execution and fetching of results without needing an ORM.
- Pony ORM (requires psycog2 for PG)
  - GitHub stars: ~3,500. Released in 2009
  - An ORM that allows Pythonic syntax for db queries, including support for native Python generators to simplify query logic.
- Databases (with SQLAlchemy; can use psycog2)
  - GitHub stars: ~4,400 . Released in 2018
  - An asynchronous database library built to work seamlessly with SQLAlchemy, widely used in async Python applications.
- Peewee (requires psycog2 for PG)
  - GitHub stars: ~10,000. Released in 2010
  - A small, lightweight ORM known for its simplicity and expressiveness while offering many advanced ORM features.
- Tortoise ORM (requires asyncpg for PG)
  - GitHub stars: ~7,500. Released in 2018.
  - A fully asynchronous ORM inspired by Django, designed for compatibility with async frameworks like FastAPI.

Comparison

Feature	SQLAlchemy Core	PyPika	Records	Pony ORM	Databases	Peewee	Tortoise ORM
Lightweightness	Moderate	High	High	Moderate	Moderate	High	Moderate
Pythonic Style	High	High	High	High	High	High	High
Typing Support	Partial	Partial	No	Yes	Yes	Partial	Yes (full support)
Query Building	Yes	Yes	No	Yes (via Python comprehensions)	Yes (via Core)	Yes (via models)	Yes (via models)
Result Handling	`ResultProxy`	N/A (query only)	`RecordCollection`	`Entity` objects	Async Rows	`Model` instances	`Model` instances
SQL Dialect Support	Multiple (PostgreSQL)	Multiple (PostgreSQL)	Multiple (PostgreSQL)	Multiple (PostgreSQL, MySQL, SQLite, Oracle)	Multiple (PostgreSQL)	Multiple (PostgreSQL, MySQL, SQLite)	Multiple (PostgreSQL, MySQL, SQLite)
Pandas Integration	Good	Limited	Good	Limited	Good	Good (using `.dicts()` or `.tuples()`)	Good (requires conversion)

Summary Ranking by Feature

Feature	1st Place	2nd Place	3rd Place	4th Place	5th Place	6th Place	7th Place
Lightweightness	PyPika	Records	Peewee	Tortoise ORM	Databases	SQLAlchemy Core	Pony ORM
Pythonic Style	PyPika	Pony ORM	Databases	Peewee	Records	Tortoise ORM	SQLAlchemy Core
Typing Support	Tortoise ORM	Pony ORM	Databases	Peewee	PyPika	SQLAlchemy Core	Records
Query Building	SQLAlchemy Core	PyPika	Pony ORM	Tortoise ORM	Peewee	Databases	Records
Result Handling	Records	Databases	Peewee	Tortoise ORM	SQLAlchemy Core	Pony ORM	PyPika
SQL Dialect Support	SQLAlchemy Core	Pony ORM	Peewee	Databases	Tortoise ORM	PyPika	Records
Pandas Integration	Records	SQLAlchemy Core	Databases	Peewee	Tortoise ORM	PyPika	Pony ORM

Overall Ranking

#	Library	Strengths	Weaknesses
1	**SQLAlchemy Core**	Comprehensive query building, excellent SQL dialect support, and good Pandas integration. Supports schema evolution with Alembic.	Moderate in terms of lightweightness; typing support is only partial.
2	**PyPika**	Extremely lightweight, highly Pythonic, with a fluent, readable interface for query building.	Lacks direct result handling and relies on adapters for executing queries. Does not support schema evolution.
3	**Records**	Simple to use, great for result handling, and has strong integration with Pandas.	Limited to executing raw SQL queries; no typing or query-building. Does not support schema evolution.
4	**Pony ORM**	Pythonic style and query building using Python comprehensions; extensive SQL dialect support. Supports built-in schema evolution.	Moderate in terms of lightweightness and lacks Pandas integration.
5	**Databases**	Asynchronous support, supports type annotations, and integrates SQLAlchemy Core for query building and for schema evolution.	Requires handling async results for Pandas integration, and it is not as lightweight compared to others.
6	**Peewee**	Lightweight and Pythonic with model-based query construction; moderate Pandas integration. Supports built-in schema evolution.	Lacks full typing support and is less comprehensive in terms of query building compared to SQLAlchemy Core.
7	**Tortoise ORM**	Full type annotation support, async capabilities, and model-based query building. Supports built-in schema evolution.	Limited Pandas integration; not as lightweight as some other options.

SQLAlchemy Core

```
# Define the 'author' table structure (matches your PostgreSQL table)
author_table = Table('author', metadata,
    Column('id', Integer, primary_key=True),
    Column('first_name', String(255)),
    Column('last_name', String(255))
)

# Create a connection to the database
with engine.connect() as connection:
    # Create a select query to fetch all rows from the 'author' table
    query = select([author_table]).where(author_table.c.last_name == 'Werner')

    # Execute the query
    result = connection.execute(query)

    # Fetch and print all results
    authors = result.fetchall()

    for author in authors:
        print(f"ID: {author.id}, First Name: {author.first_name}, Last Name: {author.last_name}")
```

PyPika

```
# Define the 'author' table using PyPika
author = Table('author')

# Create a query to select all rows from the 'author' table
query = Query.from_(Author).select('*').where(Author.last_name == 'Werner')

# Execute the query
cursor.execute(sql_query)

# Fetch and print all results
authors = cursor.fetchall()

for author in authors:
    print(f"ID: {author[0]}, First Name: {author[1]}, Last Name: {author[2]}")
```

Records

```
# SQL query to select all authors with last name 'Werner'
query = "SELECT * FROM author WHERE last_name = :last_name"

# Execute the query and pass 'Werner' as the parameter for 'last_name'
rows = db.query(query, last_name='Werner')

# Iterate through the results and print each author's details
for row in rows:
    print(f"ID: {row['id']}, First Name: {row['first_name']},
            Last Name: {row['last_name']}")
```

Feedback to me, stefan.keller@ost.ch

What's your experience?

Pony ORM

```
# Define the Author entity corresponding to the 'author' table
class Author(db.Entity):
    id = PrimaryKey(int, auto=True)
    first_name = Required(str)
    last_name = Required(str)

# Use db_session to manage transactions
@db_session
# Query to fetch all authors with last_name 'Werner'
def get_authors_by_last_name(last_name):
    authors = select(a for a in Author if a.last_name == last_name)[:]

    for author in authors:
        print(f"ID: {author.id}, First Name: {author.first_name},
                Last Name: {author.last_name}")

# Call the function
get_authors_by_last_name('Werner')
```

Databases

```
# Function to connect to the database
async def connect_to_db():
    await database.connect()

# Function to query all authors
async def get_authors_with_last_name(last_name):
    query = "SELECT id, first_name, last_name FROM author WHERE last_name = last_name"  # raw SQL style
    #query = author_table.select().where(author_table.c.last_name == last_name)  # SQLAlchemy style
    results = await database.fetch_all(query=query)

    for author in results:
        print(f"ID: {author['id']}, First Name: {author['first_name']}, Last Name: {author['last_name']}")

# Main asynchronous function to connect, query (and disconnect)
async def main():
    await connect_to_db()  # Connect to the database
    await get_authors_with_last_name('Werner')()  # Fetch and display all authors with last name Werner

asyncio.run(main())
```

Peewee

```
# Define the Author model corresponding to the 'author' table
class Author(Model):
    id = IntegerField(primary_key=True)
    first_name = CharField(max_length=255)
    last_name = CharField(max_length=255)

    class Meta:
        database = db  # This model uses the 'library' database

# Query the database to get all authors
def get_all_authors():
    authors = Author.select().where(Author.last_name == 'Werner')

    for author in authors:
        print(f"ID: {author.id}, First Name: {author.first_name}, Last Name: {author.last_name}")

# Call the function to fetch and display all authors
get_all_authors()
```

Tortoise ORM

```
# Define the Author model corresponding to the 'author' table
class Author(models.Model):
    id = fields.IntField(pk=True)
    first_name = fields.CharField(max_length=255)
    last_name = fields.CharField(max_length=255)

    class Meta:
        table = "author"  # This model maps to the 'author' table

# Function to query all authors with last_name "Werner"
async def get_authors_by_last_name(last_name: str):
    authors = await Author.filter(last_name=last_name)

    # Iterate through the results and print each author's details
    for author in authors:
        print(f"ID: {author.id}, First Name: {author.first_name}, Last Name: {author.last_name}")

# Main function to run the database operations
async def main():
    await init()  # Step 1: Connect to the database
    await get_authors_by_last_name("Werner")  # Query all authors

asyncio.run(main())
```

Octoprobe, Testing with HIL

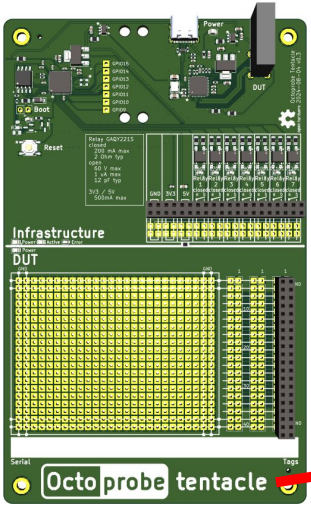
New FancyCam

Octo probe

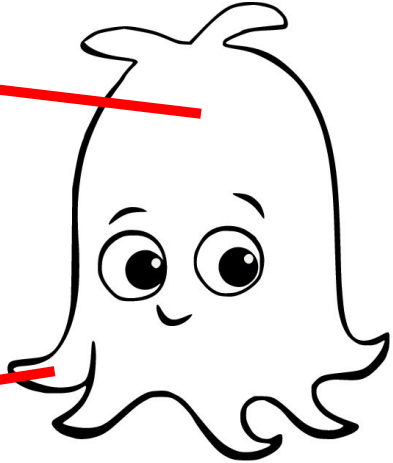
- Your task:
  - Write driver in C for micropython firmware
  - On github, accept PullRequests from community
  - Test matrix:
    - 2 FancyCam HW
    - 5 CPU boards
    - 2 Micropython versions
    - => 20 combinations to test!

Testing HIL (HW in the loop)

Octo probe

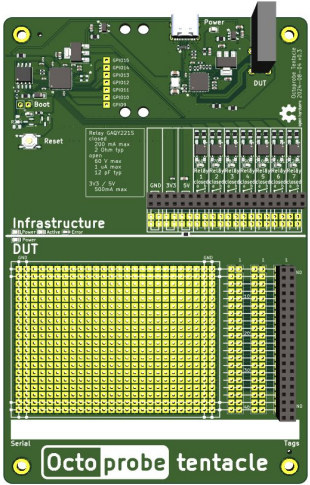


pytest



Step by step

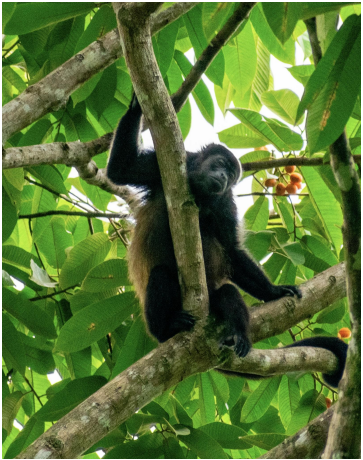
- 7 Tentacles: 2 FancyCam, 5 CPU
- test server: install octoprobe
- write pytests
- self hosted runner within github
- github action which triggers on PR



Intro to Monkey-Patching

Intro to Monkey-patching

Daniel Szoke



What is Monkey-patching? 🐒

“Monkey patching in Python refers to **dynamically modifying or extending a class or module at runtime**, allowing you to **change its behavior**.”

- ChatGPT

What is Monkey-patching? 🐒

“Monkey patching in Python refers to **dynamically modifying or extending a class or module at runtime**, allowing you to **change its behavior**.”

- ChatGPT

Let’s see an example of how to monkeypatch to capture exceptions!

Imagine a server framework…

```
import my_server_framework


@my_server_framework.route('/')
def index():
    1 / 0  # Uh, oh!
    return 'Hello, World!'
```

Somewhere in the server framework code

```
# Somewhere in my_server_framework

handlers: dict[str, Callable] = {}

# @my_server_framework.route registers handlers in dict

def request handler(path):
   handler = handlers[path]
   handler()  # <---- The function registered to the path
```

Now let’s patch in error SDK

```
# error SDK
import my_server_framework.request_handler

def patch_request_handler():
  old_request_handler = my_server_framework.request_handler

  def wrapper(*args, **kwargs):
      try:
          return old_request_handler(*args, **kwargs)
      except Exception as e:
          capture_exception(e)
          raise e

  my_server_framework.request_handler = wrapper  # <---- 🐵

patch_request_handler()
```

Yay! Error got captured :)

```
import my server_framework
import error_sdk

error_sdk.init()

@my server framework.route('/')
def index():
  1 / 0  # Uh, oh!
  return 'Hello, World!'
```

Jython + Mypy

Jython with mypy

Who here likes Python?

Who uses type annotations?

```python
def f(a: str, b: str, c: int) -> str:
    return (a + b) * c
```

Who uses static type checking?

```python
def f(a: str, b: str, c: int) -> str:
    return (a + b) * c

f(1, 2, 3)
```

Who has to interact with Java systems?

Jython:
Python 2.7 interpreter in Java

```
from java.lang import System

System.out.println("Hello, world")
```

```
~/jython-lt $ jython scripts/hello.py
Hello, world
```

Jython with mypy

```python
# Python 3
def f(a: str, b: str, c: int) -> str:
    return (a + b) * c
```

```python
# Python 2
def f(a, b, c):
    # type: (str, str, int) -> str
    return (a + b) * c
```

Jython with mypy

```
pip install 'mypy[python2]<0.980'
```

```
def f(a: str, b: str, c: int) -> str:
    return (a + b) * c
```

```
scripts/script.py:6: error: invalid syntax
Found 1 error in 1 file (errors prevented further checking)
```

Jython with mypy

```
pip install 'mypy[python2]<0.980'
```

```
import random

random.choices(["a", "b", "c"], k=3)
```

```
scripts/script.py:3: error: Module has no attribute "choices"; maybe "choice"?
Found 1 error in 1 file (checked 1 source file)
```

Java Packages

```
export JYTHONPATH=lib/commons-collections4-4.5.0-M2.jar
```

```
from org.apache.commons.collections4 import ListUtils

numbers = [9, 6, 2, 6, 1, 6, 5]

for batch in ListUtils.partition(numbers, 2):
    print(batch)
```

Type stubs

```
cat jython-stubs/org/apache/commons/collections4/__init__.pyi
```

```
from typing import TypeVar


T = TypeVar("T")

class ListUtils:
    @staticmethod
    def partition(items: list[T], size: int) -> list[list[T]]:
        """Returns consecutive sublist of a list, each of the sam
```

Type stubs

```
for batch in ListUtils.partition(numbers, 2):
    print(batch)
```

```
org.apache.commons.collections4.ListUtils

@staticmethod
def partition(items: list[T],
              size: int) -> list[list[T]]
```

Returns consecutive sublist of a list, each of the same size (the final list may be smaller).

Type stubs

```
for batch in ListUtils.partition(numbers):
    print(batch)
```

```
(.venv) ~/jython-lt $ mypy scripts/script.py jython-stubs/
scripts/script.py:5: error: Missing positional argument "size" in call to "partition" of "ListUtils"  [call-arg]
Found 1 error in 1 file (checked 7 source files)
```

Testing with Mocks

```python
class ListUtils:
    @staticmethod
    def partition(items: list[T], size: int) -> list[list[T]]:
        batches = []
        batch = []
        for item in items:
```

```python
from org.apache.commons.collections4 import ListUtils
def test_partition() -> None:
    chars = list("Python")
    batches = ListUtils.partition(chars, 2)
    assert batches == [["P", "y"], ["t", "h"], ["o", "n"]]
```

```
(.venv) ~/jython-lt $ pytest scripts/test.py --quiet
.
1 passed in 0.01s
```

Don't use os.path

Don't use `os.path` !

Because since many years is a better way..

timon@python-summit.ch

Swiss Python Summit

timon.erhart@ost.ch

IFS | Institut für Software

What is `os.path`?

In [10]:

```
import os

path = os.getcwd()
path = os.path.join(path, "file.txt")
print(path)
with open(path, "w") as f:  # touch path
    pass
os.rename(path, os.path.join(
    os.path.dirname(path),"file2.txt"))
path = os.path.join(os.path.dirname(path), "file2.txt")
print(path)
print(
    os.path.exists(path)
    )
os.unlink(path)  # remove file
```

```
/home/erti/LEHRE_repos/ploting-with-python-slides/file.txt
/home/erti/LEHRE_repos/ploting-with-python-slides/file2.txt
True
```

There is a better way!

- use `pathlib` (standard library!)
- object oriented

- since 3.14 (2014)

Why people using it still?

python concat path

Alle Videos Bilder News Bücher Web Maps ⋮ Mehr Suchfilter

GeeksforGeeks
https://www.geeksforgeeks.org › ... · Diese Seite übersetzen ⋮

Python | os.path.join() method

08.10.2024 — os.**path**.join() takes multiple **path** components as arguments and concatenates them into a single **path**. It ensures the correct **path** separator is ...

Stack Overflow
https://stackoverflow.com › questions › platform-indep... ⋮

Platform independent path concatenation using "/" , "\"?

In **python** I have variables base_dir and filename. I would like to **concatenate** them to obtain fullpath. But under windows I should use \ and for POSIX /.

6 Antworten · Top-Antwort: You want to use os.path.join() for this. The strength of using this rat…

Concatenate path and filename - **python** - Stack Overflow 14. Nov. 2016
Python os.**path**.join() and "+" in string **concatenation** 6. Feb. 2022
File **Path Concatenation** - **python** - Stack Overflow 29. Juli 2021
connecting multiple strings to **path** in **python** with slashes 6. Juni 2018
Weitere Ergebnisse von stackoverflow.com

Python Docs
https://docs.python.org › library · Diese Seite übersetzen ⋮

os.path — Common pathname manipulations

Join one or more **path** segments intelligently. The return value is the **concatenation** of **path** and all members of ***paths**, with exactly one directory separator ...

Example

In [11]:

```
from pathlib import Path

path = Path() # os.getcwd()
path = path / "file.txt" # os.path.join
path.write_text("")  # touch
print(
    path.exists()# os.path.exists
    )
path = path.rename("file2.txt") # os.rename
path.unlink()  # os.unlink
```

```
True
```

In [12]:

```
# old
import glob

for f in glob.glob(os.path.join(os.getcwd(), "*.txt")):
    print(f, type(f))  # string

# new
for file in Path().glob("*.txt"):
    print(f, type(f))
```

```
/home/erti/LEHRE_repos/ploting-with-python-slides/requirements.txt <class 'str'>
/home/erti/LEHRE_repos/ploting-with-python-slides/requirements.txt <class 'str'>
```

Further reads

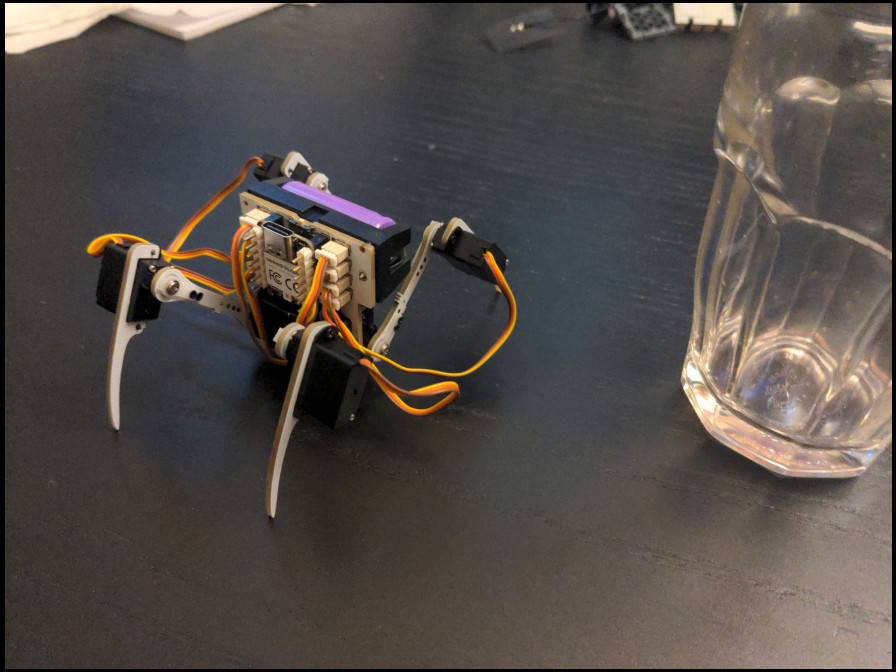
- The official doks
  - **https://docs.python.org/3/library/pathlib.html**
  - come with an nice comparison with `os.path` (##corresponding-tools)
- Python 3 Module of the Week
  - **https://pymotw.com/3/pathlib/index.html**

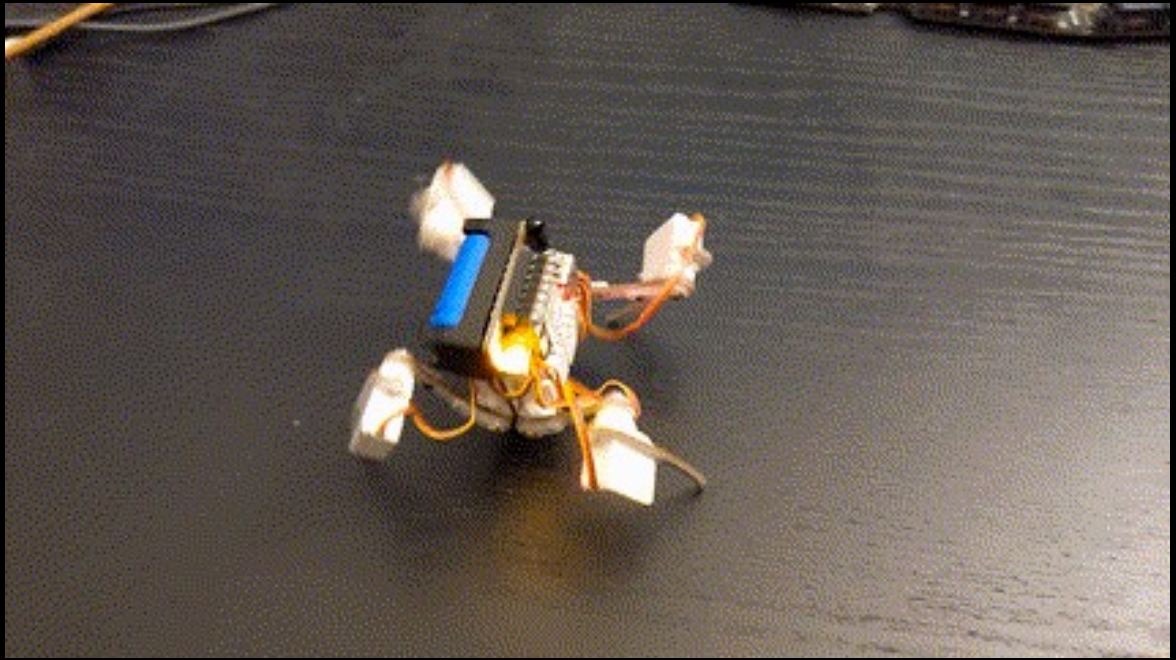
Radomir Dopieralski

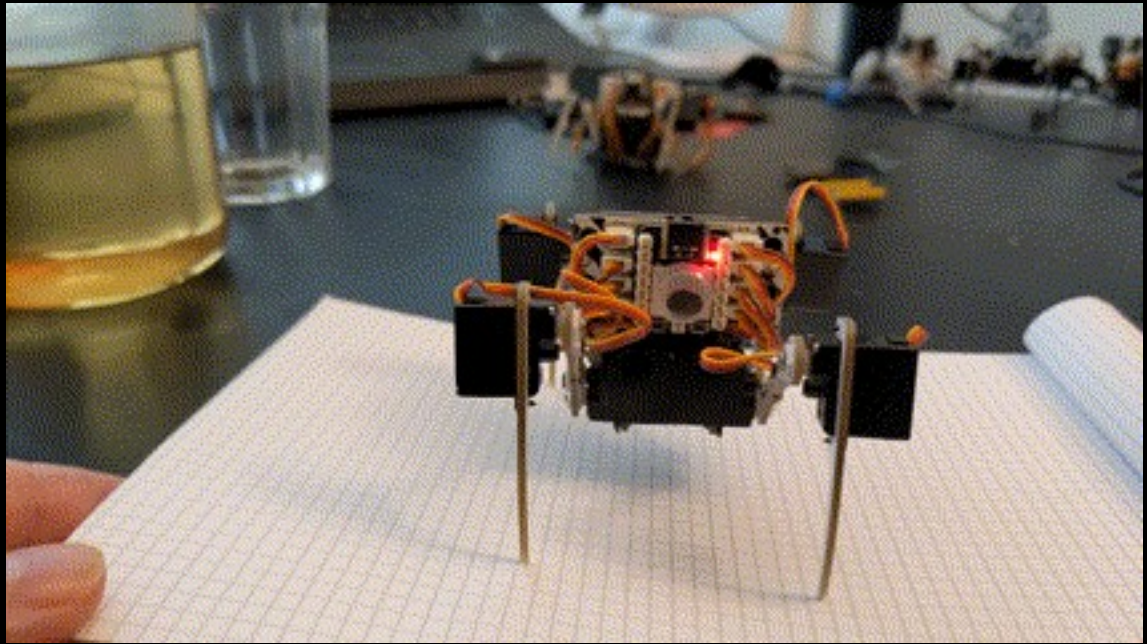
Robots and MicroPython

Robots and MicroPython

Radomir Dopieralski
@deshipu@fosstodon.org







Thank you!

https://deshipu.art/

Mypy rewritten in Rust

Jedi Autocompletion

2022

Mypy in Rust

- Passing 20% of Mypy's test suite
- Tests run 650x faster than Mypy

2024

Mypy in Rust

- Passing ~~20%~~ 90% of Mypy's relevant tests
- Tests run ~~650x~~ 230x faster than Mypy

Goal: ZubanLS

- A Python Language Server written in Rust
- **zubanls.com**
- **info@zubanls.com**
- Twitter/GitHub: **@ZubanLS**

zmypy

```
zmypy foobar --strict --warn-unreachable
```

- 0.9s vs 40s when type checking Mypy (no cache)
- Single Thread (optimizable)
- Supports most mypy flags and features
- Reducing false positives

Outlook

- Probably not Open Source (But fallback to Mypy)
- zmypy is available very soon
- Language Server hopefully in 2025
- I am very interested to have a discussion

Questions?

- **zubanls.com**
- **info@zubanls.com**
- Twitter/GitHub: **@ZubanLS**