

# **Erfolgreich entwickeln mit Python und InterSystems IRIS**

Stefan Wittmann – Product Manager  
InterSystems





# Agenda



---

01 Python und InterSystems IRIS

---

02 Setup

---

03 Interoperabilität

---

04 Debuggen

---

05 Virtuelle Umgebungen

---

- Python hat eine stark wachsendes und innovatives Ökosystem, optimiert für
  - Data Engineering
  - Data Analytics
  - Data Visualization








**DANGER**  
BEWARE OF  
ALLIGATORS  
AND SNAKES  
DO NOT ENTER  
THE WATER  
DO NOT FEED  
THE WILDLIFE



 **DANGER**

**BEWARE OF  
ALLIGATORS  
AND SNAKES**



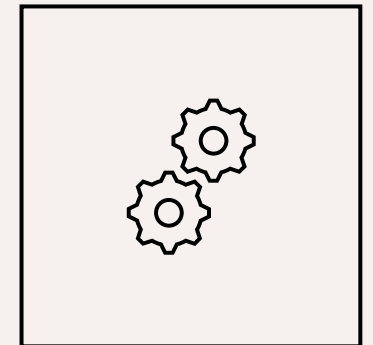
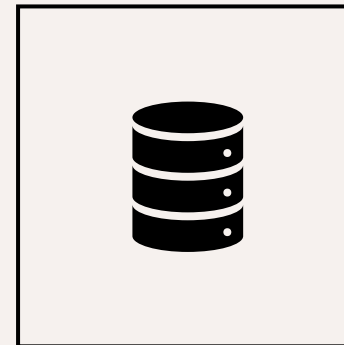
**DO NOT ENTER  
THE WATER**

**DO NOT FEED  
THE WILDLIFE**

# InterSystems IRIS Python Support



- Python SDKs
- Optionen
  - pyODBC
  - DB-API
  - Native
  - SQLAlchemy
    - alembic
  - External Language Server
  - PEX





# pyODBC

## Python ODBC Bridge

- Ein Python Modul, welches die DB-API Spezifikation umsetzt und die Datenbank über ODBC anspricht.
- Nicht mehr empfohlen → DB-API
  - Nur zu verwenden, wenn man auf InterSystems IRIS 2022.1 oder frühere Versionen zugreifen muss
- Installation (Windows und Unix):

```
pip install pyodbc
```

# DB-API



- Das Standard-Interface um mit relationalen Datenbanken zu interagieren

## So installiert man den InterSystems DB-API Treiber

- <https://pypi.org/project/interSystems-irispthon/>

- Führen Sie folgenden Befehl aus

```
pip install intersystems-irispthon
```

- Falls frühere Versionen manuell installiert wurden, muss einmal ein Update über PyPI erzwungen werden:

```
pip install intersystems-irispthon==5.1.2
```



# DB-API



Eine Verbindung aufbauen

```
1  import iris
2  connection_string = "localhost:1972/USER"
3  username = "demo"
4  password = "demo"
5  connection = iris.connect(connection_string, username, password)
6  cursor = connection.cursor()
7
8  try:
9      pass # do something with DB-API calls
10 except Exception as ex:
11     print(ex)
12 finally:
13     if cursor:
14         cursor.close()
15     if connection:
16         connection.close()
```

# DB-API



Eine SQL Abfrage ausführen

```
20 # Prepare and execute a SQL
21 cursor.execute("SELECT * FROM Sample.Person WHERE ID = 1")
22
23 # Fetch next row of a query result set
24 row = cursor.fetchone()
25
26 # Retrieve all column values of a row
27 values = row[:]
28 print(values)
29
30 # Close the cursor object
31 cursor.close()
```

# DB-API



## Weiter Beispiele

```
34 # Using query parameters with question mark style
35 sql = "SELECT * FROM Sample.Person WHERE id = ? and name = ?"
36 params = [1, 'Jane Doe']
37 cursor.execute(sql, params)
38
39 #Using query parameters with named style
40 sql = "SELECT * FROM Sample.Person WHERE firstname = :fname AND lastname = :lname"
41 params = {'fname' : 'John', 'lname' : 'Doe'}
42 cursor.execute(sql, params)
43
44 #Fetching result set rows in batch
45 cursor.fetchmany(100)
46
47 #Position the cursor at a specific row
48 cursor.scroll(3, 'absolute')
```

# Umfassende Dokumentation



[https://docs.intersystems.com/iris20251/csp/docbook/DocBook.UI.Page.cls?KEY=BPYNAT\\_pyapi](https://docs.intersystems.com/iris20251/csp/docbook/DocBook.UI.Page.cls?KEY=BPYNAT_pyapi)

The screenshot shows the InterSystems documentation website for the IRIS Data Platform 2025.1. The page is titled "Using the Python DB-API". The left sidebar contains a navigation menu with categories like "Using InterSystems External Servers", "Java", ".NET", and "Native SDK for Python". The "Native SDK for Python" category is expanded, showing sub-items like "About...", "Using the Native SDK for Python", "Introduction to the Native SDK for Python", "Calling Database Methods and Functions from Python", "Controlling Database Objects from Python", "Accessing Global Arrays with Python", "Managing Transactions and Locking with Python", "Using the Python DB-API", "Native SDK for Python Quick Reference", and "Native SDK for Node.js". The main content area has a search bar and a "AskMe (beta)" button. The article text describes the InterSystems Python DB-API driver as a fully compliant implementation of the PEP 249 version 2.0 Python Database API specification. It lists required implementation features and provides a table of contents for the article, including "Usage", "PEP 249 Implementation Reference", and "SQLType enumeration values". A "Note: DB-API Driver Installation" section at the bottom states that the DB-API is available when you install InterSystems IRIS. The right sidebar includes a "Feedback" button and a "Microsoft To" button.

InterSystems Documentation

Learning Documentation Community Open Exchange Global Masters Certification Partner Directory

InterSystems IRIS Data Platform 2025.1 / External Language Development / Native SDK for Python / Using the Native SDK for Python / Using the Python DB-API

Search InterSystems IRIS 2025.1 Documentation

AskMe (beta)

InterSystems IRIS Data Platform 2025.1

- Using InterSystems External Servers
- Java
- .NET
- Native SDK for Python
  - About...
  - Using the Native SDK for Python
    - Introduction to the Native SDK for Python
    - Calling Database Methods and Functions from Python
    - Controlling Database Objects from Python
    - Accessing Global Arrays with Python
    - Managing Transactions and Locking with Python
    - Using the Python DB-API
    - Native SDK for Python Quick Reference
  - Native SDK for Node.js

Class Reference

View this item as PDF

Download all PDFs

## Using the Python DB-API

The InterSystems Python DB-API driver is a fully compliant implementation of the [PEP 249 version 2.0](#) Python Database API specification. The following sections list all required implementation features, indicate the level of support for each one, and describe all InterSystems-specific features in detail:

- [Usage](#)

Describes how to make a connection to InterSystems IRIS and get a **Cursor** object.
- [PEP 249 Implementation Reference](#)

Lists all PEP 249 requirements and provides implementation details in the following subsections:

  - [Globals](#) lists values for required global constants *apilevel*, *threadsafety*, and *paramstyle*.
  - [Connection Object](#) describes **Connection** methods [connect\(\)](#), [close\(\)](#), [commit\(\)](#), [rollback\(\)](#), and [cursor\(\)](#).
  - [Cursor Object](#) describes the following **Cursor** members:
    - Attributes [arraysize](#), [description](#), and [rowcount](#).
    - Standard methods [callproc\(\)](#), [close\(\)](#), [execute\(\)](#), [executemany\(\)](#), [fetchone\(\)](#), [fetchmany\(\)](#), [fetchall\(\)](#), [nextset\(\)](#), [scroll\(\)](#), [setinputsize\(\)](#), and [setoutputsize\(\)](#).
    - InterSystems extension methods [isClosed\(\)](#) and [stored\\_results\(\)](#).
- [SQLType enumeration values](#)

Lists valid SQLType enumeration constants.

**Note:**  
**DB-API Driver Installation**

The DB-API is available when you install InterSystems IRIS. If you do not have the InterSystems DB-API driver (for example, if you are connecting from a host on which

Is this page helpful?

Contents

- Usage
- PEP 249
- Implementation
- Reference
- SQLType enumeration values

Feedback

Microsoft To



# SQLAlchemy

## *Auf der Roadmap*



- SQLAlchemy ist ein Object-Relational Mapping (ORM) Implementierung und basiert auf DB-API
- Entwickler können ein Datenmodell in Python erstellen und mit einer Datenbank synchronisieren
- Nicht nur für Datenmodellentwicklung relevant
  - Viele Module verwenden SQLAlchemy zur Persistenz
- Installieren Sie den DB-API Treiber und den SQLAlchemy IRIS Dialekt
  - (SQLAlchemy wird implizit als Abhängigkeit installiert)



# SQLAlchemy



## Ein einfacher Einstieg

```
#import sqlalchemy
from sqlalchemy import create_engine

#create an engine with the IRIS dialect and a connection string
engine = create_engine('iris://demo:demo@localhost:9091/USER',echo=False)

#SQL statements get wrapped in text sequences
from sqlalchemy import text

#connect to the database and execute a static query
with engine.connect() as conn:
    result = conn.execute(text("SELECT 'Hello World'"))
    print(result.all())
```

# SQLAlchemy



## Transaktionen und Sessions

```
#engine.begin starts a transaction block with an auto-commit
with engine.begin() as conn:
    result = conn.execute(text("SELECT x, y FROM some_table"))
    for row in result:
        print(f"x: {row.x}  y: {row.y}")
```

#Session starts an ORM session with fine-grained commit/rollback controls

```
from sqlalchemy.orm import Session
```

```
print("Listing rows where y > 3")
stmt = text("SELECT x, y FROM some_table WHERE y > :y ORDER BY x, y")
with Session(engine) as session:
    result = session.execute(stmt, {"y": 3})
    for row in result:
        print(f"x: {row.x}  y: {row.y}")
```

# SQLAlchemy



## Einstieg in das ORM Modellieren

```
from typing import List, Optional
from sqlalchemy import String, ForeignKey
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column, relationship

class Base(DeclarativeBase):
    pass

class User(Base):
    __tablename__ = "user_account"

    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(30))
    fullname: Mapped[Optional[str]]

    def __repr__(self) -> str:
        return f"User(id={self.id!r}, name={self.name!r}, fullname={self.fullname!r})"
```



# SQLAlchemy



## Das Modell synchronisieren und Daten persistieren

```
engine = create_engine('iris://demo:demo@localhost:9091/USER', echo=True)
Base.metadata.create_all(engine)

with Session(engine) as session:
    spongebob = User(
        name="spongebob",
        fullname="Spongebob Squarepants"
    )
    sandy = User(
        name="sandy",
        fullname="Sandy Cheeks"
    )
    patrick = User(name="patrick", fullname="Patrick Star")

    session.add_all([spongebob, sandy, patrick])

    session.commit()
```

# SQLAlchemy



## Daten abfragen und aufräumen

```
from sqlalchemy import select

#this is still within the context of our session!

stmt = select(User).where(User.name.in_(["spongebob", "sandy"]))

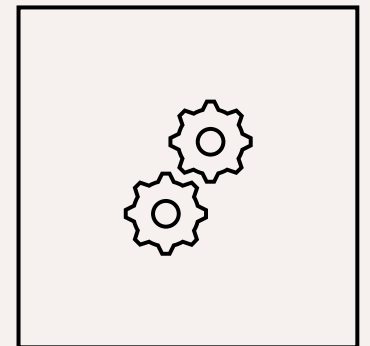
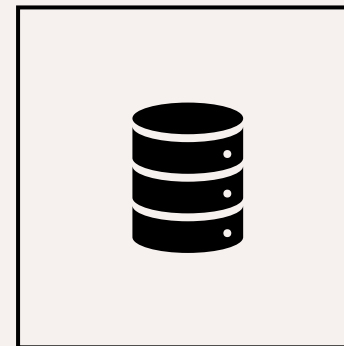
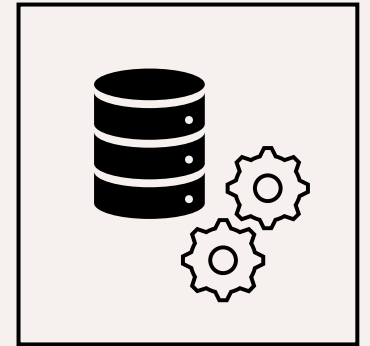
for user in session.scalars(stmt):
    print(user)

session.delete(spongebob)
session.delete(sandy)
session.delete(patrick)
```

# InterSystems IRIS Python Support



- Embedded Python
- Optionen
  - Das "language" tag einer Methode auf "Python" setzen
  - Ein Python Modul mit ObjectScript Code laden
  - Python Ausdrücke in Interoperabilität Produktionen verwenden
  - IoP
- Ein Python .py Skript verwenden



# Wie verwendet man Embedded Python?



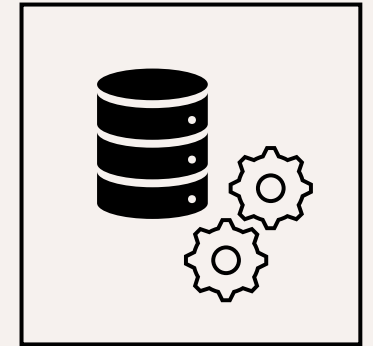
## Die Hauptoptionen

- Das language Tag verwenden:

```
Method Foo() As %String [Language = python]
```

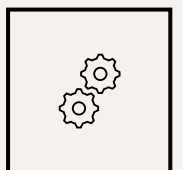
- Ein Python Modul mit ObjectScript Code laden

```
##class(%SYS.Python).Import()
```



- Einen neuen Prozess starten; den Python Interpreter starten und `import iris` ausführen

```
python -c "import iris; print(iris.system.Version.GetVersion())"
```





# Wie verwendet man Embedded Python?

Der language Tag

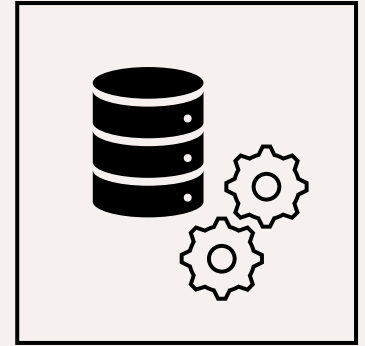
```
ClassMethod Get() As %Status [ Language = python ]
{

    import requests

    url = "https://httpbin.org/get"

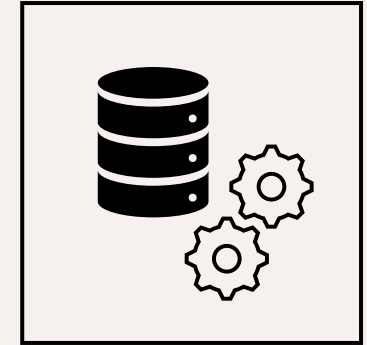
    # make a request
    response = requests.get(url)
    # get the json data from the response
    data = response.json()
    #iterate over the data and print the key-value pairs
    for key, value in data.items():
        print(key, ":", value)

}
```



# Wie verwendet man Embedded Python?

Das %SYS.Python Interface



```
ClassMethod Get() As %Status
{
    set status = $$$OK
    set url = "https://httpbin.org/get"

    set request = ##class(%SYS.Python).Import("requests")

    set response = request.get(url)

    set data = response.json()

    // Import built-in Python module
    set builtins = ##class(%SYS.Python).Import("builtins")

    For i = 0:1:builtins.len(data)-1 {
        Write builtins.list(data.items()).__getitem__(i).__getitem__(0),": ",builtins.li
    }
    quit status
}
```

# Wie verwendet man Embedded Python?



Ein Python Skript ausführen

```
1  import requests
2  import iris
3
4  def get(param_url: str = None):
5      url = "https://mockbin.io/bar" if param_url is None else param_url
6      # make a get request
7      response = requests.get(url)
8      # get json data
9      data = response.json()
10     # iterate over the data and print key-value pairs
11     for key, value in data.items():
12         print(f"{key}: {value}")
13
14     # store response in a global variable
15     iris.gref('^Response')[None] = response.text
16
17     return response.text
18
19 if __name__ == "__main__":
20     get("https://df584e11d7ee451f953beb31e96a9948.api.mockbin.io/")
```



# Embedded Python konfigurieren



## Flexible Python Runtime

- Sie können die Python Laufzeitumgebung die InterSystems IRIS verwenden soll konfigurieren
- Windows:
  - Zuerst muss Python installiert werden
- Auf Unix-basierten Betriebssystemen können Sie die Standard Umgebung überschreiben
- Passen Sie folgende Parameter in IRIS an:
  - PythonRuntimeLibrary
  - PythonRuntimeLibraryVersion

### PythonRuntimeLibrary

Specify the location of the runtime library to use when running Embedded Python.

#### Synopsis

```
[config] PythonRuntimeLibrary=library
```

### PythonRuntimeLibraryVersion

Specify the version number of the runtime library to use when running Embedded Python.

#### Synopsis

```
[config] PythonRuntimeLibraryVersion=version
```

# Interoperabilität und Python



- Alles bisher genannte
- Code Fragmente können in Python statt in ObjectScript definiert werden

Actions (0 selected) Select all ✕ ✂ 📄 🗑 ↑ ↓ 🚫 Disable 📄 New

<input type="checkbox"/> 2	<b>CODE</b>	/* DATE: 2018-07-09 TASK: TASK00000 NAME: Samantha Forish NOTE: Initial Development */			
<input type="checkbox"/> 3	<b>SET</b>	target.Separators	=	" ^-\\&"	
<input type="checkbox"/> 4	<b>SET</b>	target.{MSH}	=	source.{MSH}	
<input type="checkbox"/> 5	<b>SET</b>	target: target.{EVN}	source: source.{EVN}	key: ""	comment: Description language: Class default
<input type="checkbox"/> 6	<b>FOR EACH</b>	source.{PIDgrp()}	iterate using	k1	
<input type="checkbox"/> 7	<b>SET</b>	target.{PIDgrp(k1).PID}	=	source.{PIDgrp(k1).PID}	
<input type="checkbox"/> 8	<b>SET</b>	target.{PIDgrp(k1).NTE}	=	source.{PIDgrp(k1).NTE}	

Class default (objectscript)

Python

ObjectScript


- „Interoperability on Python“
  - Community Projekt
  - <https://openexchange.intersystems.com/package/interoperability-embedded-python>

# IoP

## Ein Community Projekt

- Auf OpenExchange gelistet
- Ermöglicht das Entwickeln von Interoperabilitätselementen in Python





### interoperability-embedded-python

by Guillaume Rongier

InterSystems does not provide technical support for this project. Please contact its developer for the technical assistance.

[Details](#) [Releases \(66\)](#) [Reviews \(15\)](#) [Issues](#) [Pull requests \(1\)](#) [Articles \(2\)](#)

[python](#)

Hack of PEX Python but for Embedded Python

#### What's new in this version

Enhancement :

- Support of async function in BP and BO.
- Support of multi sync call
- Trace option in log

Fix:

- Better support of retro-compatibility of grongnier.pex

status **stable**

pypl **v3.4.4**

downloads **2.2k/month**

license **MIT**

last commit **last friday**

Welcome to the **Interoperability On Python (IoP)** proof of concept! This project demonstrates how the **IRIS Interoperability Framework** can be utilized with a **Python-first approach**.

Documentation can be found [here](#).

5 ★  
15 reviews

0  
Awards

7.7k  
Views

1.2k  
IPM installs

[🔔 12](#) [🔖 12](#) [🔗](#) [GitHub](#)

#### Made with

[Docker](#) [Python](#) [IPM](#)

#### Install

[📦 zpm install pex-embedded-python](#)  
[📄 download archive](#)

[Repository](#) | [Documentation](#) | [License](#)

#### Version

3.1.0 | 26 Jul, 2024

#### Python package

iris-pex-embedded-python

#### Ideas portal

<https://ideas.intersystems.com/ideas/DP-I-143>

#### Category

Developer Environment

#### Works with

InterSystems IRIS

**First published**  
03 Jan, 2022

**Last edited**  
11 Feb, 2025

# Interoperabilität und Python



- Die Konzepte:
  - Python-zentrischer Ansatz
  - Business hosts, Adapter und Nachrichten werden in .py Skripten entwickelt
  - Properties und Parameters definieren
  - Callbacks können implementiert werden (OnProcessInput)
  - Ens.Director Methoden können aufgerufen werden, um Nachrichten zu versenden
  - Python Object müssen zum Persistieren „gewrapped“ werden
  - Auf Referenzen als Rückgabe kann zugegriffen werden
  - .py Skripte werden auf dem Server registriert und generieren dann die passenden Server-seitigen Module
  - Debugging wird unterstützt



# Roadmap – Python und Interoperabilität



Roadmap!

```
# Message type which can be persisted by the production framework.
class MyData(ProductionMessage):
    Name: str = Description(default="something", newcol=True, index=True)
    Amount: int = 10
```

```
class CustomInAdapter(InboundAdapter):
    Counter: int = IRISProperty("%Integer", default=0)
    testSetting: str = IRISProperty("%String", "Test Description", settings={"category": "category"})
    Tabnine | Edit | Test | Explain | Document
    def OnTask(self):
        try:
            msg = PythonData("Message ID: " + str(self.Counter), self.Counter)
            self.Counter = self.Counter + 1
            status, Output, _ = self.BusinessHost_ProcessInput(msg)
        except Exception as e:
            status = iris.system.Status.Error(5001, str(e))
        return status
```

# Roadmap – Python und Interoperabilität



Roadmap!

```
class CustomBS(BusinessService):
    testBSSetting: str = IRISProperty("%String", settings={"category": "BScategory"})
    target_host: str = IRISProperty("%String", default = "PythonMessage.customBP")
    ADAPTER = IRISParameter("PythonMessage.customInAdapter", description="Business adapter")
    Tabnine | Edit | Test | Explain | Document
    def OnProcessInput(self, input):

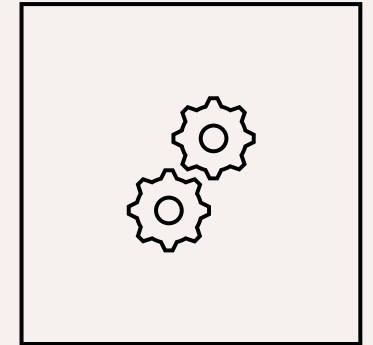
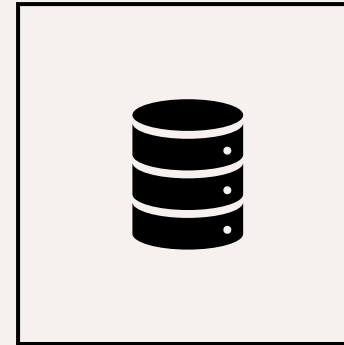
        try:
            # debug_host("172.22.31.148", 5547)
            time.sleep(2)
            persistentObject = MyData(input.Name, input.Amount)
            status = self.SendRequestAsync("PythonMessage.customBP", persistentObject)
            output = "*****Output from BS*****"
        except Exception as e:
            output = "*****SCRIPT ERROR in BS = *****" + str(e)
            status = iris.system.Status.Error(5001, str(e))
        return {"pOutput": output, "status": status}
```

# Python Code debuggen

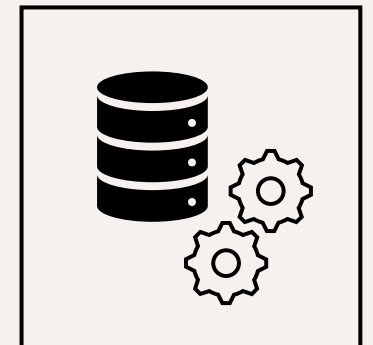


- Interaktives Debuggen wird unterstützt wenn Python in einem eigenen Prozess läuft!

- Zwei Szenarien
  - Lokales Debuggen
  - Remote Debuggen



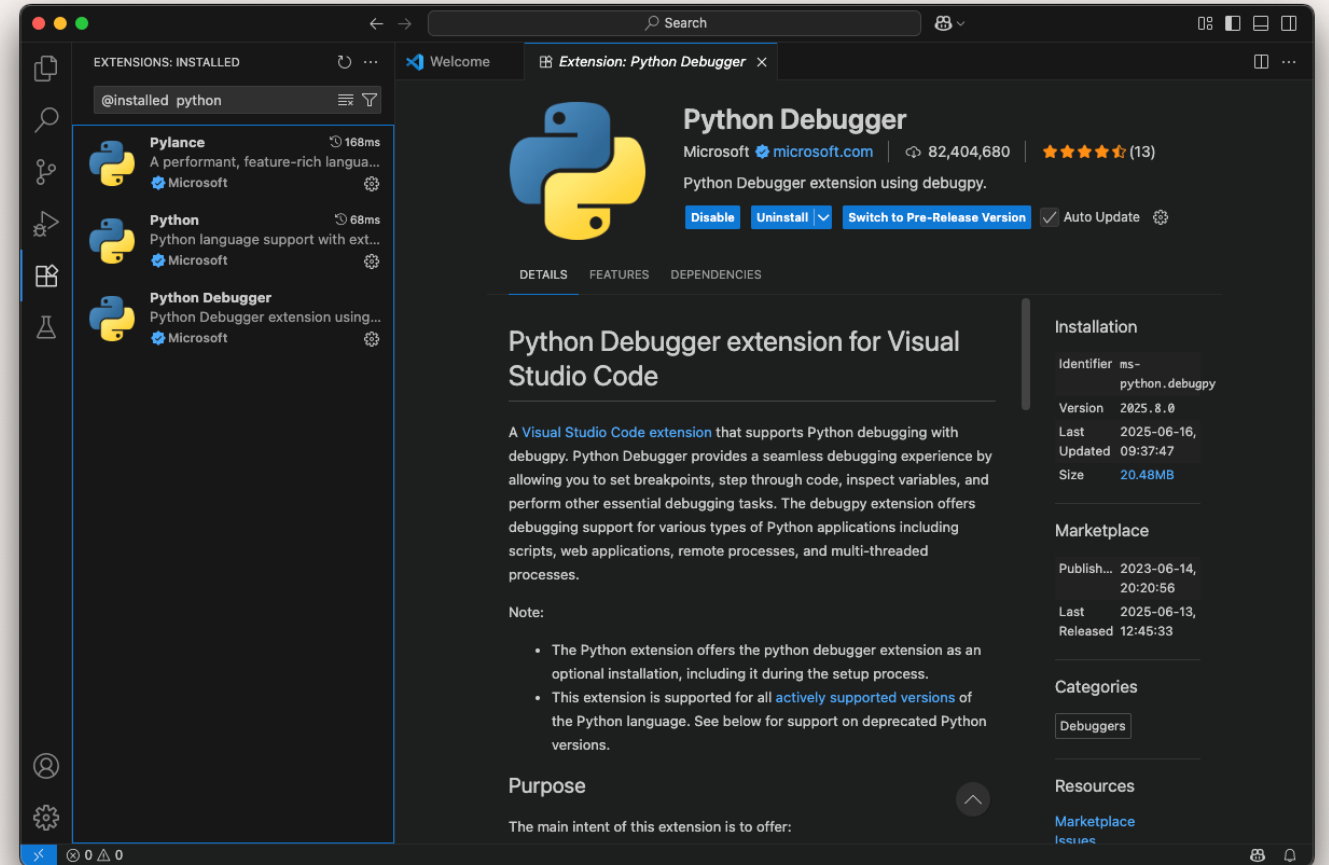
- Debuggen wird nicht unterstützt, wenn Python in einem von IRIS gestarteten Prozess läuft!



# Schnellstart

## Visual Studio Code

- Installieren Sie Visual Studio Code (und einen Python Interpreter)
- Installieren Sie die Python and Python Debugger Erweiterungen

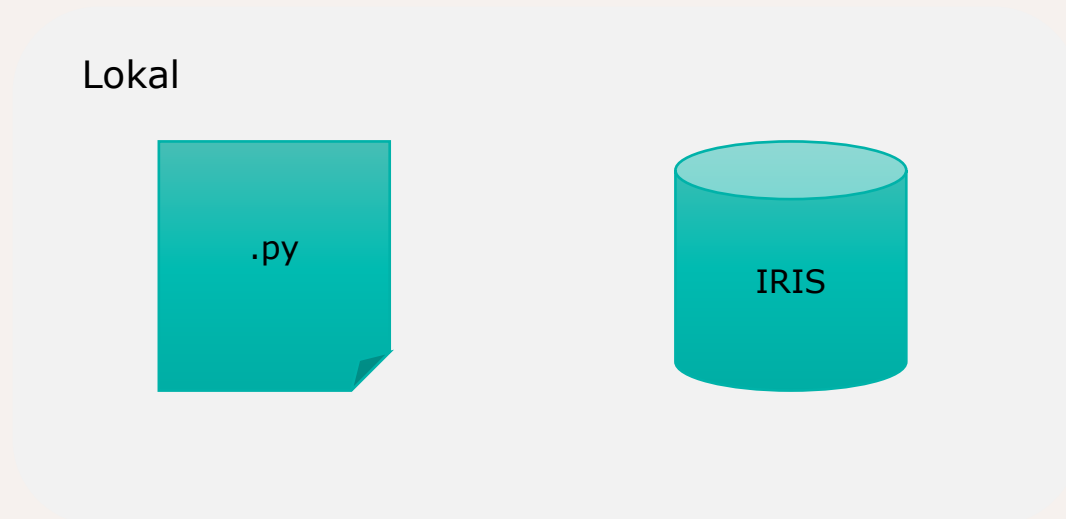


# Lokales Debuggen



Das Szenario

- Ein lokales Python Projekt
- Eine lokale InterSystems IRIS Instanz



# Lokales Debuggen



Die "launch.json" Datei konfigurieren

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python Debugger: Current File",
      "type": "debugpy",
      "request": "launch",
      "program": "${file}",
      "console": "integratedTerminal",
      "sudo": true,
      "env": {
        "IRISUSERNAME": "demo",
        "IRISPASSWORD": "demo",
        "IRISNAMESPACE": "USER",
        "PYTHONPATH": "/Users/wittmann/iris/lib/python"
      }
    }
  ]
}
```

# Lokales Debuggen



Die "launch.json" Datei konfigurieren

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python Debugger: Current File",
      "type": "debugpy",
      "request": "launch",
      "program": "${file}",
      "console": "integratedTerminal",
      "sudo": true,
      "env": {
        "IRISUSERNAME": "demo",
        "IRISPASSWORD": "demo",
        "IRISNAMESPACE": "USER",
        "PYTHONPATH": "/Users/wittmann/iris/lib/python"
      }
    }
  ]
}
```



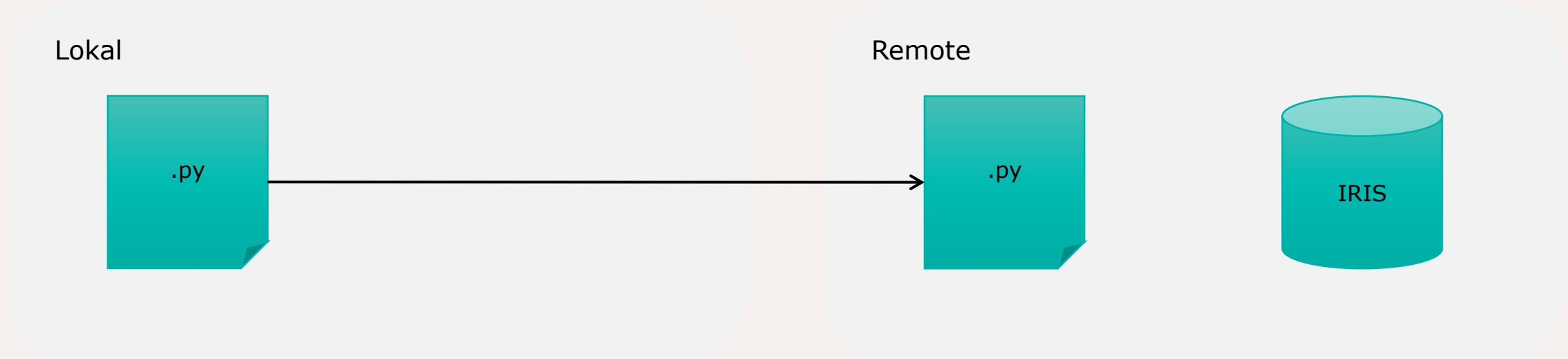
# Remote Debuggen



## Das Szenario

- Ein Remote Python Projekt
- Eine Remote InterSystems IRIS Instanz
- Zusätzliche wird benötigt:
  - Lokaler Zugriff auf das Python Projekt

1. Den Debugger auf dem Remote Rechner starten
2. Den lokalen Debugger starten und mit dem Remote Port verbinden



# Remote Debuggen



Die "launch.json" Datei konfigurieren

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python Debugger: Remote Attach",
      "type": "debugpy",
      "request": "attach",
      "connect": {
        "host": "localhost",
        "port": 5678
      },
      "pathMappings": [
        {
          "localRoot": "${workspaceFolder}",
          "remoteRoot": "/usr/irissys/mgr/python"
        }
      ]
    }
  ]
}
```

Der Remote Port

Der lokale Ordner mit dem Python Projekt

Der Remote Ordner mit dem Python Projekt



# Was sind “environments”?

- Umgebungen ermöglichen es ein Projekt abzuschirmen
  - Sich vor Änderungen in der globalen Umgebung schützen
  - Die globale Umgebung nicht „verschmutzen“
- Zwei Möglichkeiten
  - Virtuelle Umgebungen (venv) und Conda
  - Virtual Environments sind ein Bestandteil von der Python Laufzeitumgebung

# Virtual Environments



## Schnellstart

- Ein "virtual environment" erstellen

```
python -m venv my-folder
```

- Das "venv" aktivieren

Windows: `my-folder\Scripts\activate`

Mac/Unix: `source my-folder/bin/activate`

- Deaktivieren

```
deactivate
```



# Virtual Environments

## Pakete und hilfreiche Befehle

- Pakete werden in der aktivierten Umgebung verwaltet

```
python -m pip install sqlalchemy
```

```
python -m pip list
```

- Eine requirements.txt Datei erstellen

```
python -m pip freeze
```

```
python -m pip freeze > requirements.txt
```

- Pakete direkt von einer requirements.txt Datei installieren

```
python -m pip install -r requirements.txt
```

# Virtual Environments



## ***ROADMAP***

- Wir arbeiten an einer Option InterSystems IRIS mit einer “venv” Konfiguration zu starten
- Die zu aktivierende Umgebung wird in der Konfigurationsdatei definiert werden

# Take-aways



- DB-API statt pyODBC
- SQLAlchemy für aufwendigere Datenmodelle
- Language tag = python → Nur für simple Projekte
- Python Skripte für aufwendigere Projekte bevorzugen



# Vielen Dank

Gerne beantworte ich im Anschluss Ihre Fragen. Bitte sprechen Sie mich an.

