

Wissenschaftliche Tools in der Hochschullehre mit bwJupyter

Jasmin Hörter, Paul Hoger,
Stephanie Hofmann



Dieses Dokument ist lizenziert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (<https://creativecommons.org/licenses/by-sa/4.0/legalcode>).

Ausgenommen von der Lizenz sind die Logos sowie das zitierte Bildmaterial.



bwJupyter - der JupyterHub für die Hochschullehre in BW

- Landesdienst bwJupyter als **webbasierte interaktive Entwicklungsumgebung** seit März 2025 für alle Hochschulen in Baden-Württemberg verfügbar
- **MWK-BW Projekt** im Rahmen von “Hochschulen in der digitalen Welt”
- Kooperation von **KIT** und **Uni Stuttgart**
- aktuell ca. 2500 Nutzenden von 33 Hochschulen



<https://hub.bwjupyter.de>

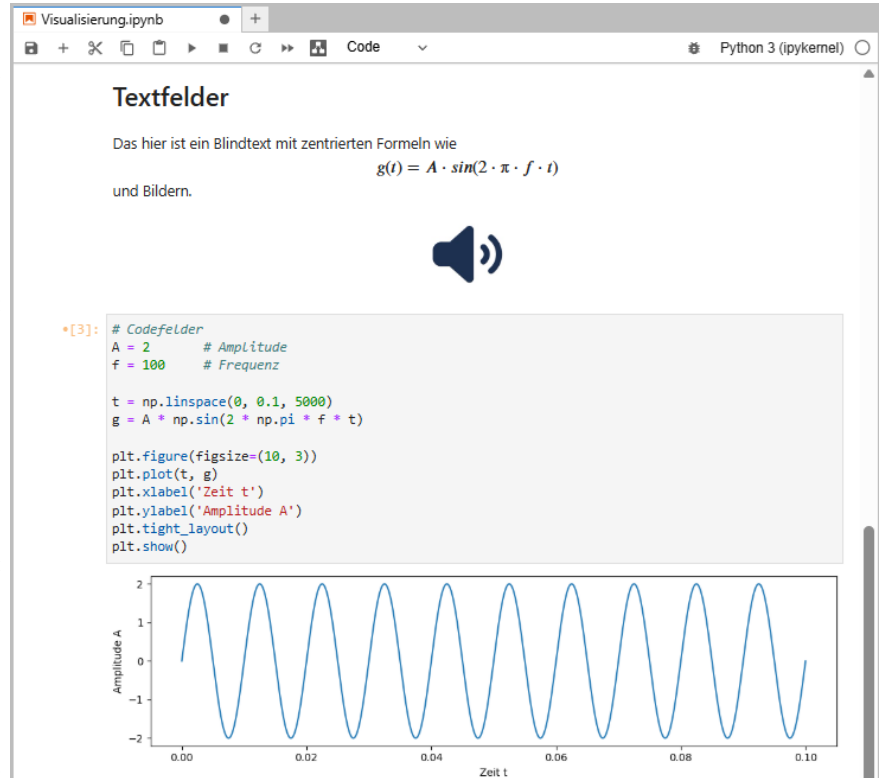
Jupyter Notebooks (JNB)

- interaktive Dokumente mit
 - Code
 - Text (Markdown & HTML)
 - Formeln (LaTeX)
 - Visualisierungen
- verschiedene Programmiersprachen unterstützt:
 - Julia
 - Python
 - R

Textfelder

Codefelder

Ausgabe



JNB in der Lehre - verschiedene Einsatzszenarien

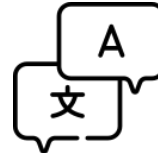
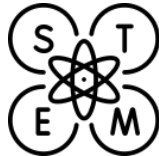
JNB in **Dozierendenhand**



JNB in **Studierendenhand**



verschiedene **Fachrichtungen**

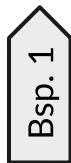


JNB in der Lehre - Vielseitigkeit der JNB

- offene Aufgabenformate
- exploratives, authentisches wissenschaftliches Arbeiten
- Programmierkenntnisse notwendig
- geschlossene Aufgabenformate
- angeleitetes Arbeiten
- für Lerngruppen mit unterschiedlichen Programmierkenntnissen einsetzbar
- Unterstützung des Lernprozesses auf verschiedenen Ebenen möglich

JNB als Werkzeug

JNB als Lernumgebung



Bsp. 1: JNB als Computational Essay

compessay.ipynb

Python 3 (ipykernel)


Open in...

Code

Kamerabasierte Artikelerkennung in dm-Filialen

I. Beschreibung des realen Problems

Im Einzelhandel gibt es viele Routinetätigkeiten, die viel Zeit und Personal in Anspruch nehmen, wie z.B. die Regalpflege oder Inventur. Jedoch wird die Anzahl der zur Verfügung stehenden Fachkräfte durch den demografischen Wandel immer geringer, weshalb man immer mehr versucht, die Mitarbeiter durch technische Hilfsmittel zu unterstützen. Unser Ziel war es daher, optimale Positionen für Kameras zu finden, die in einer dm-Filiale fehlende Artikel in Regalen feststellen können, damit man zeigerndest Artikel wieder auffüllen kann. Dies sollte für 90-100% der Artikel möglich sein.



Zentral waren bei uns folgende Fragen:

- Wo sind die optimalen Kamerapositionen, wenn man annimmt, dass die Kameras auf einer Höhe positioniert werden?
- Wie kann man das nun auf die dritte Dimension übertragen, um die jeweils notwendige Höhe und Anordnung der Kameras zu bestimmen?

II. Die Daten

Die Produkte in den Filialregalen sind in einer Tabelle gegeben und nach `aisle` (Gang), `side` (linke oder rechte Seite des Gangs) und `shelf` (einzelne Regalabschnitte in einer Regalreihe) sortiert. Jede Zeile entspricht einem Artikel im Regal und für alle vier Ecken des Sichtfensters des Artikels sind die Koordinaten `x`, `y` und `z` angegeben. Am Anfang eines Gangs sitzt immer der Ursprung, von dem aus die `x`-Achse den Gang entlang verläuft, während die `y`-Achse orthogonal zu ihr liegt und die `z`-Achse nach oben zeigt. Der Datensatz beinhaltet 6 Gänge mit jeweils zwei Seiten, in denen 5253 verschiedene Artikel Platz finden.

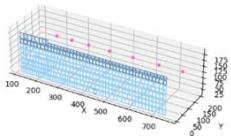
+ 10 cells hidden

III. Modell 1: Tracing Mapping Algorithm

```
### Tracing Mapping Algorithm / x-Richtung, darauf aufbauend z-Richtung ###

Auch wenn diese Funktion nur für eine eindimensionale Positionierung der Kamera optimal ist, so kann man sie trotzdem ebenfalls auf die Gänge anwenden. Die dabei entstandenen Ergebnisse benötigen sehr wenige Kameras, haben aber oft in den oberen oder unteren Regalböden eine geringere Abdeckung.
```

```
<[30] # Erstelle einen Zähler, um die Anzahl an benötigten Kameras zu erfassen
counter = 0
covs = []
# Gruppieren die Produkte nach Reihe und Seite und führe dafür jeweils das:
for w in df.groupby(['aisle', 'side']):
    # Führe die Methode 'set_cams' aus, dabei werden die Daten aus dem entsprechenden Gang und von der entsprechenden Seite ausgeführt. Als Datenname für das Bild werden ebenfalls Gang und Seite verwendet. Die Länge der Rückgabe, also die Anzahl an Kameras, wird dem Zähler hinzugefügt
    cams, cov = set_cams(df[w['aisle']][w['side']] & (df['x'] > 0), f'{w["aisle"]}_{w["side"]}')
    counter += len(cams)
    covs += cov
# Schliessendlich wird die Anzahl an Kameras auf der Konsole ausgegeben.
print(f"Es werden insgesamt {counter} Kameras benötigt.")
print(f" Insgesamt werden {covs/len(df)} der Produkte abgedeckt.")
Finished calculating products.
Es sind 75.89842955898411 % aller Produkte durch die Kameras abgedeckt.
```



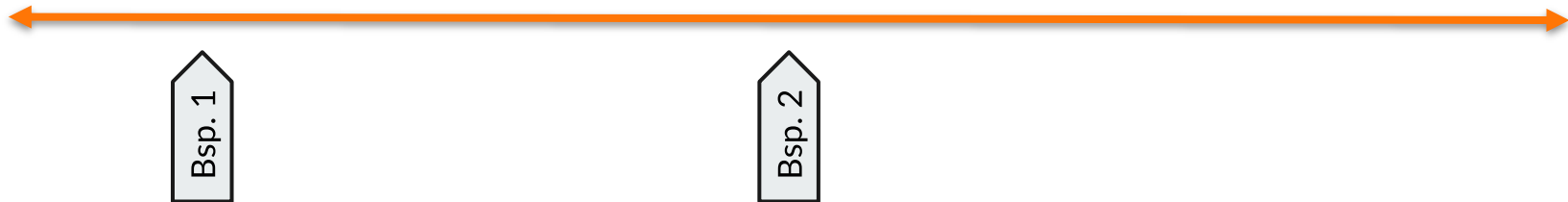
JNB in der Lehre - Vielseitigkeit der JNB

- offene Aufgabenformate
- exploratives, authentisches wissenschaftliches Arbeiten
- Programmierkenntnisse notwendig

- geschlossene Aufgabenformate
- angeleitetes Arbeiten
- für Lerngruppen mit unterschiedlichen Programmierkenntnissen einsetzbar
- Unterstützung des Lernprozesses auf verschiedenen Ebenen möglich

JNB als Werkzeug

JNB als Lernumgebung



Bsp. 2.1: JNB als Übungsblatt

Fibonacci.ipynb

+

Code

Python 3 (ipykernel)

Fibonacci-Zahlen

Die Fibonacci-Zahlen sind eine Folge von Zahlen, bei der jede Zahl die Summe der beiden vorhergehenden ist. Formal wird sie definiert durch:

$$F(0) = 0, \quad F(1) = 1, \quad F(n) = F(n-1) + F(n-2) \quad \text{für } n \geq 2$$

Aufgabe 1

Vervollständige den folgenden Python-Code, um die n -te Fibonacci-Zahl iterativ zu berechnen. Ersetze dazu die markierten Lücken im Code:

```
[ ]: import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator

# Funktion zur Berechnung der Fibonacci Zahlen
def fibonacci(n):
    # ... Ersetze diese Lücke!

# Scatter Plot
plt.scatter(range(20), [fibonacci(n) for n in range(20)])
plt.gca().xaxis.set_major_locator(MultipleLocator(1))
plt.show()

...
```


Bsp. 2.1: JNB als interaktives Skript

56_DiscussionMC.ipynb
Python 3 (ipykernel)

56 Discussion of Monte Carlo

- The MC method suffers from the curse of dimensionality, according to our definition. MC works, though, for the right class of functions, e.g. $g \in F_d^1$.
- Another specialty of MC is that we have an error equality

$$\varepsilon_{RMS} = \left(\mathbb{E} \left[\left(Qg - \bar{Q}_N g \right)^2 \right] \right)^{\frac{1}{2}} = \frac{\sqrt{\text{Var}(g)}}{\sqrt{N}}$$

This formula has to be taken with a grain of salt. In the frequentist interpretation of randomness, it means that we need to repeat the MC experiment often enough to actually observe the rate $\frac{1}{2}$.

- The major advantage of MC (and MLMC) is that in high dimensions, it is the only possibility, because we simply cannot construct a grid. A non-trivial grid has at least 2^d points. Because $2^{10} \approx 10^3$, in 100 dimensions we need $2^{100} \approx (10^3)^{10} = 10^{30}$ grid points.

EXERCISE 56.1)

In this exercise, we revisit Exercise 6.1, in which we studied the convergence of (quasi) Monte-Carlo. Go over the exercise again and think about how your answers (especially regarding task (d)) differs from your answer when you started the course. Discuss which method should be used for which problem to mitigate the curse of dimensionality.

a) In this section we compare different random numbers to approximate the value of π . What does the code do exactly?

```
def generateSamples (N): ...
```

All points

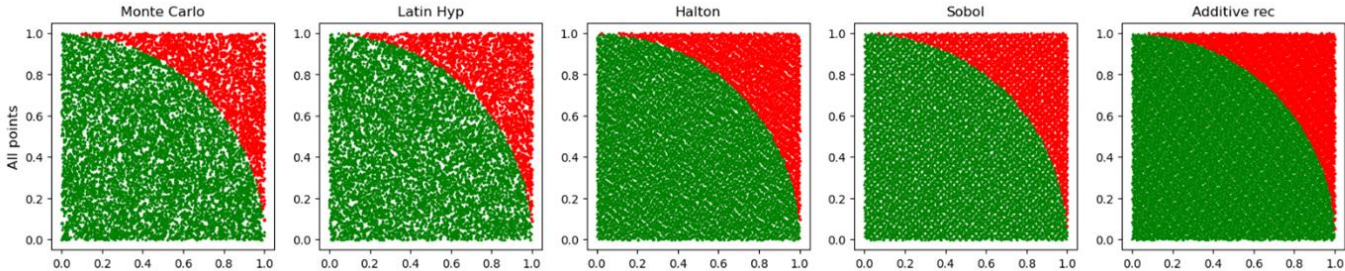
Monte Carlo

Latin Hyp

Halton

Sobol

Additive rec



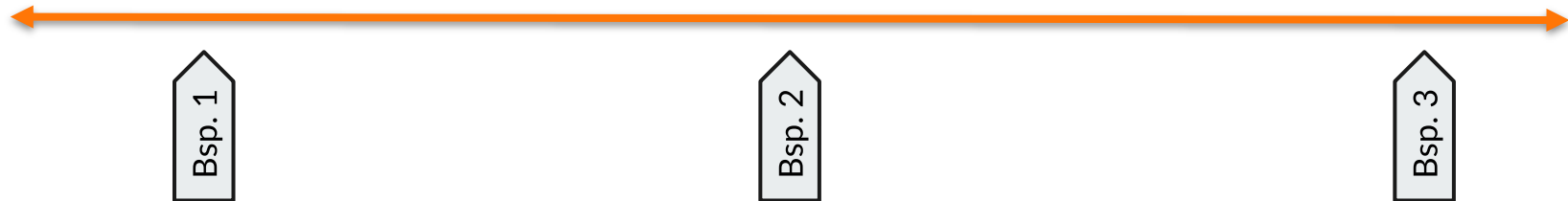
JNB in der Lehre - Vielseitigkeit der JNB

- offene Aufgabenformate
- exploratives, authentisches wissenschaftliches Arbeiten
- Programmierkenntnisse notwendig



- geschlossene Aufgabenformate
- angeleitetes Arbeiten
- für Lerngruppen mit unterschiedlichen Programmierkenntnissen einsetzbar
- Unterstützung des Lernprozesses auf verschiedenen Ebenen möglich

JNB als Werkzeug

JNB als Lernumgebung



Bsp. 3: JNB als geführte Lernumgebung


Arbeitsblatt 5 | Bewertung des Spracherkenners

Auf Arbeitsblatt 4 haben wir ein erstes funktionierendes Modell für einen Spracherkennung erstellt, indem das Muster eines neuen Sprachsignals mit vorhandenen Referenzmustern abgeglichen wurde. Auf diesem Arbeitsblatt wollen wir bewerten, wie gut der Spracherkennung ist. Wir probieren den Spracherkennung dafür systematisch aus und lesen Sprachsignale ein, deren Wortbedeutung und somit deren Klassenzugehörigkeit bereits bekannt ist. So können wir überprüfen, wie gut die Wortbedeutung über unser entwickeltes Verfahren erkannt wird. Hierfür nutzen wir eine Strategie aus dem Bereich des überwachten Maschinellen Lernens und teilen in einem ersten Schritt unsere Daten in Trainings- und Testdaten auf. Die Trainingsdaten nutzen wir um Referenzmuster zu bilden. Die Testdaten werden zur Bewertung des Spracherkenners genutzt.

1) Die Trainings- und Testdaten

Im Hintergrund haben wir die Daten bereits in Trainings- und Testdaten unterteilt und die Testdaten zurückbehalten. Wir lesen nun sowohl Trainings- als auch Testdaten ein. Im folgenden Codefeld werden die Referenzmuster, die wir bereits auf Arbeitsblatt 3 aus den Trainingsdaten ermittelt haben, sowie deren jeweilige Klassenzugehörigkeit geladen. Außerdem werden die Testdaten eingelesen und wie in Arbeitsblatt 3 die zu den Signalen zugehörigen Muster gebildet. Die Klassenzugehörigkeit ist zwar auch bei den Testdaten bekannt, für die Klassifizierung wird diese

Bsp. 3: JNB als geführte Lernumgebung




d) Dreiklangabfolge unterteilen

Gib im folgenden Codefeld die Anzahl der Zeitblöcke und die Länge der Zeitblöcke ein, um herauszufinden, ob die fehlende Unterteilung in Blöcke der einzige Grund war, warum die Frequenzbestimmung kein zufriedenstellendes Ergebnis geliefert hat. Führe anschließend das Codefeld aus und bewege den Schieberegler um dir die Frequenzen der verschiedenen Zeitblöcke anzusehen.

```
[ ]: # Aufteilung der Dreiklangabfolge in Blöcke
number_of_blocks = None


# Angabe in Sekunden
length_of_blocks = None

# Hier nichts ändern
block_frequencies = print_frequencies_of_blocks(number_of_blocks, length_of_blocks, triad_sequence)
check_blocks(number_of_blocks, length_of_blocks, triad_sequence)
```



e) Dreiklänge bestimmen

Im letzten Aufgabenteil wurde die Fourier-Transformation einzeln auf jeden Dreiklang angewendet, sodass die Frequenzen den Amplitudenspektren entnommen werden können. Um welche drei Dreiklänge handelt es sich? Führe das Codefeld aus und wähle den Dreiklang aus.

 Hier findest du die Übersicht der Dreiklänge.

```
[ ]: # Hier nichts ändern, nur ausführen!
check_unknown_triad_sequence(block_frequencies)
```

2) Frequenzen in einem Sprachsignal

Du hast kennengelernt, wie man ein komplexes akustisches Signal innerhalb einzelner Zeitblöcke hinsichtlich seiner Frequenzzusammensetzung analysiert. Diese Vorgehensweise wird nun bei einem realen Sprachsignal angewendet. Hierbei wird das Sprachsignal allerdings nicht nur in 3 Zeitblöcke sondern in 44 Zeitblöcke unterteilt, sodass möglichst viele zeitliche Änderungen in der Frequenzzusammensetzung des Sprachsignals erfasst

JNB in der Lehre - Benefits

Einsatz von JNB in unterschiedlichen Lernszenarien kann dazu beitragen...

- die **Beteiligung** zu steigern,
- das **Verständnis** zu fördern,
- die **Leistung** zu verbessern,
- eine praxisnahe **Vorbereitung auf das Berufsleben** zu ermöglichen.

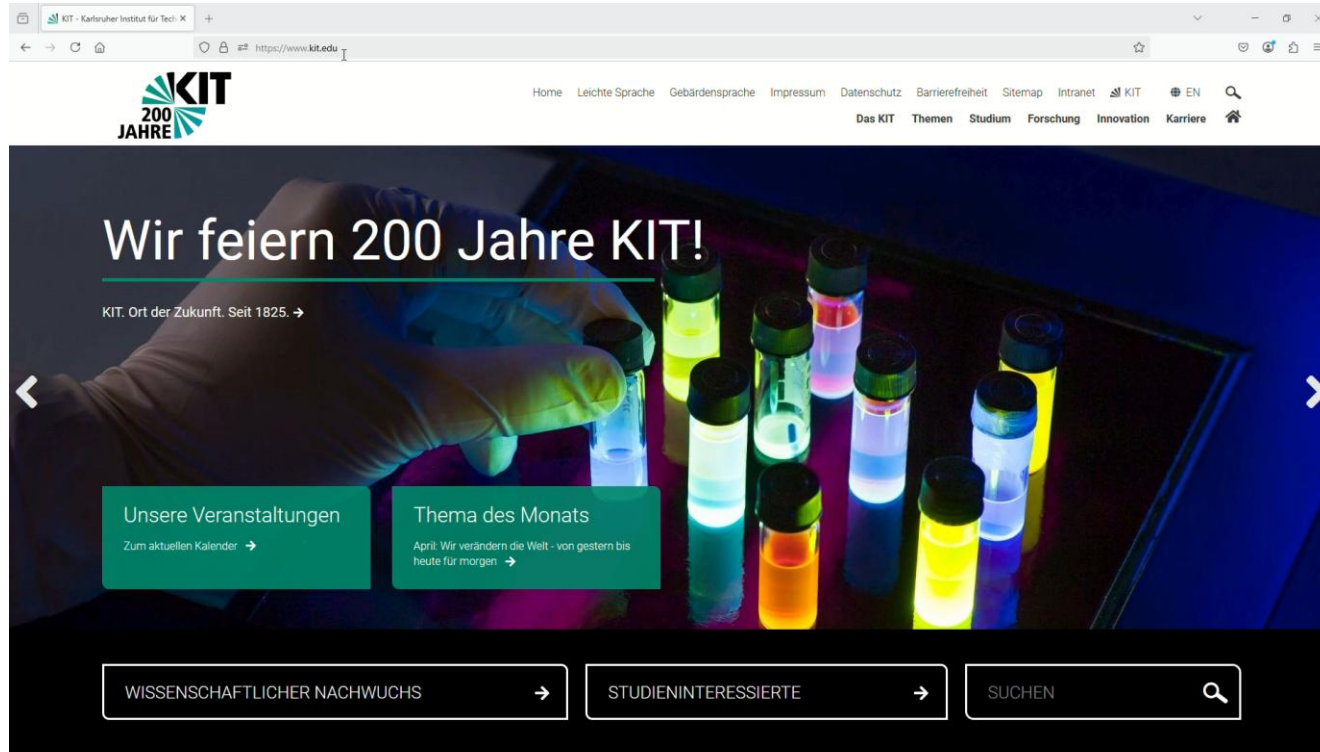
JupyterHub: Bereitstellung der Entwicklungsumgebung



Bisheriger Workflow:

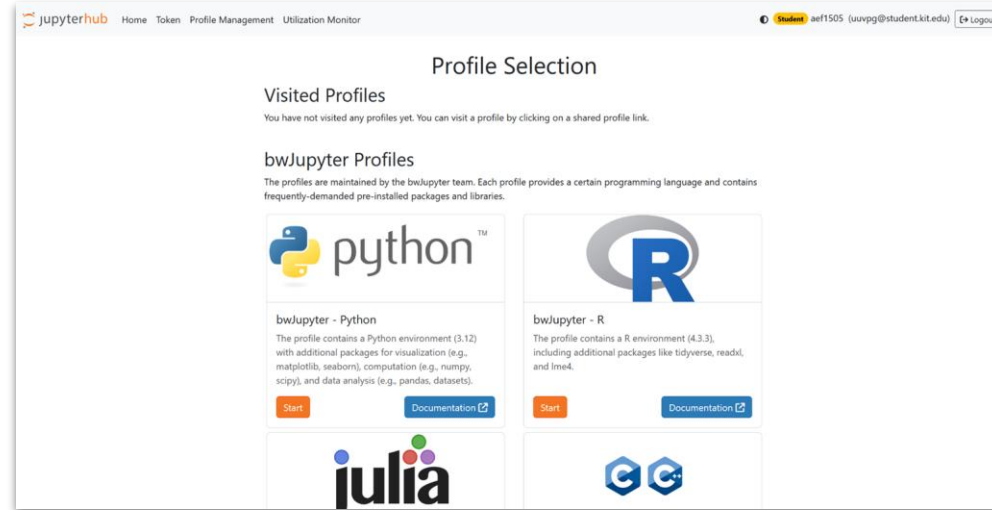
1. Python-Interpreter (Programmiersprache) auf dem PC installieren.
2. Ausführungsumgebung "Jupyter Lab" installieren.
3. Beim Lernmanagementsystem (LMS) anmelden und die Notebooks herunterladen (z.B. Übungen).
4. Benötigte Bibliotheken und Abhängigkeiten nachinstallieren.
5. Jupyter Lab starten.
6. Mit der Bearbeitung beginnen.

JupyterHub: Anmeldeprozess

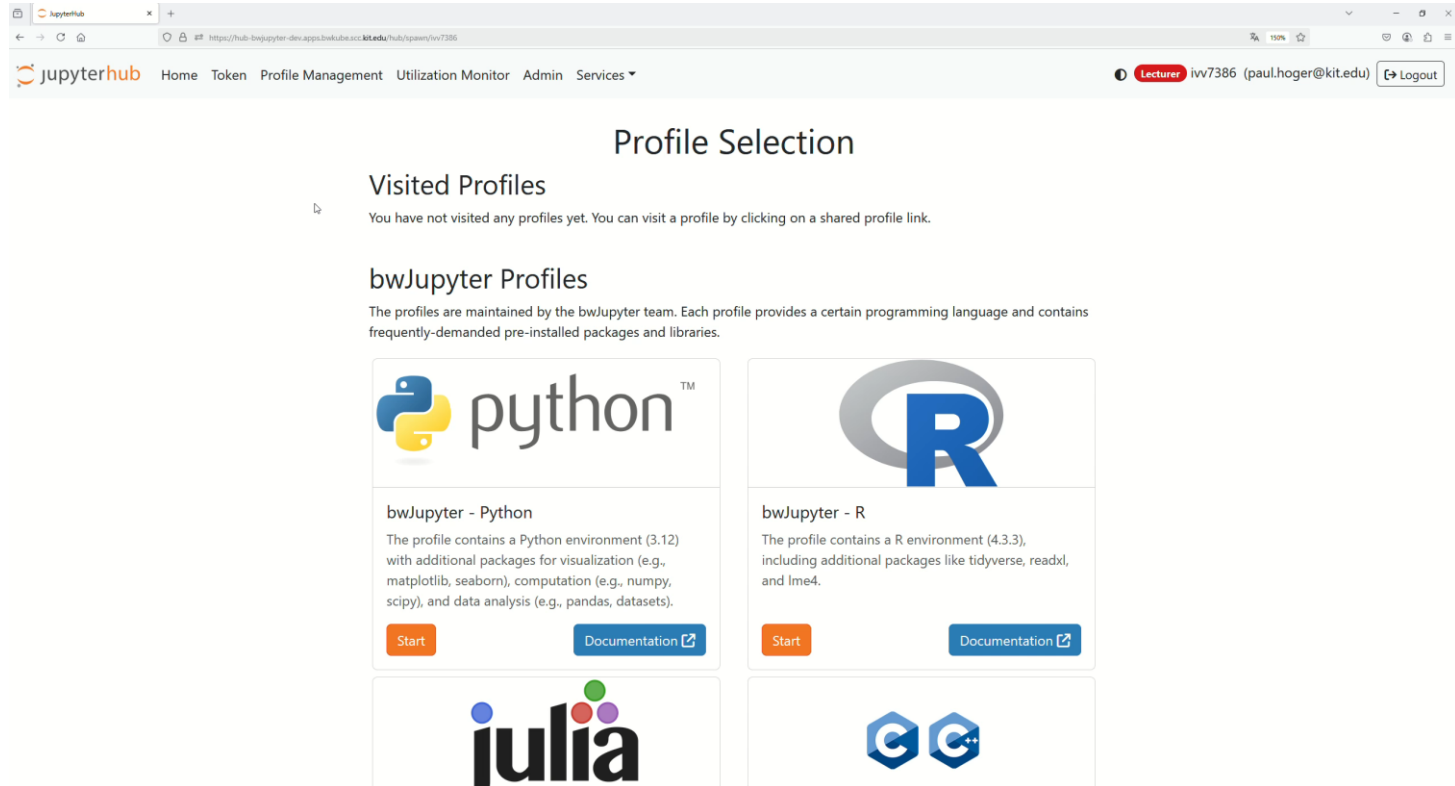


JupyterHub: Einführung in Profile

- JupyterHub nutzt sog. **Profile**
 - Ausführungsumgebung für Jupyter Notebooks
 - Bereitstellung der Rechenkapazitäten
- **bwJupyter-Profile**
 - Für die **Programmiersprachen** Python, R, Julia, C/C++, Java
 - Vorinstallierte Bibliotheken und Module
- Dozierende können eigene Profile anlegen
 - Vorhandenes Profil auswählen oder eine **eigene Umgebung** angeben
 - Nutzung einer **Grafikkarte** möglich
 - Jupyter Notebooks **hochladen**
 - Profil über einen Link mit Studierenden teilen



JupyterHub: Ein eigenes Profil anlegen



JupyterHub: Bereitstellung der Entwicklungsumgebung



Bisheriger Workflow:

1. Python-Interpreter (Programmiersprache) auf dem PC installieren.
2. Ausführungsumgebung "Jupyter Lab" installieren.
3. Beim Lernmanagementsystem (LMS) anmelden und die Notebooks herunterladen (z.B. Übungen).
4. Benötigte Bibliotheken und Abhängigkeiten nachinstallieren.
5. Jupyter Lab starten.
6. Mit der Bearbeitung beginnen.

Workflow mit bwJupyter:

1. Dozierende legen ein Profil an und laden die Notebooks hoch.
2. Studierende öffnen den geteilten Link.

Zusammenfassung:

- + Leicht zugänglich
- + Geräte- und Plattformunabhängig
- + Ressourcenverfügbarkeit
- + Geteiltes Dateisystem
- + Eigene Umgebungen

Ausblick

- **Integration** in die Lernmanagementsysteme (LMS)
- **Übungsbetrieb** (Erstellen, Ausgeben, Bearbeiten, Einsammeln, Korrigieren, Feedback verteilen)
über bwJupyter abdecken
- Entwicklung und Bereitstellung von **Begleitmaterialien, Vorlagen und Beispielen**



Vielen Dank für Ihre Aufmerksamkeit!

bwJupyter - der JupyterHub für die Hochschullehre in BW

- Jupyter Notebooks in der Lehre einsetzen
- Einfacher Anmeldeprozess ohne manuelle Freischaltung
- Nutzung bereitgestellter Rechenressourcen
- Eigene Profile für Dozierende



<https://bwjupyter.de>



support@bwjupyter.de (Support, Fragen, **Feedback**)



Dieses Dokument ist lizenziert unter einer Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (<https://creativecommons.org/licenses/by-sa/4.0/legalcode>).

Ausgenommen von der Lizenz sind die Logos sowie das zitierte Bildmaterial.

Registrieren Sie sich jetzt!



<https://hub.bwjupyter.de>