


Python im Unterricht

Gert-Ludwig Ingold

 `git clone https://github.com/gertingold/lit2018.git`

Über mich

- ▶ seit 1994 Theoretischer Physiker an der Universität Augsburg
- ▶ seit 2010 auch Vorlesungen zum Programmieren
`gertingold.github.io`

- ▶ 2015–2017 Erasmus+-Projekt iCSE4school



Co-funded by the
Erasmus+ Programme
of the European Union

Anwendung von SageMath im gymnasialen Unterricht

- ▶ Uniwersytet Śląski, Kattowitz
- ▶ Simula Research Laboratory, Oslo
- ▶ Universität Augsburg
- ▶ Gymnasien in Chorzów und Warschau
- ▶ EDU-RES Chorzów

- ▶ 2017–2019 Erasmus+-Projekt Jupyter@edu



Co-funded by the
Erasmus+ Programme
of the European Union

Anwendung von Jupyter Notebooks an Universitäten

- ▶ Uniwersytet Śląski, Kattowitz
- ▶ Universität Augsburg
- ▶ European University Cyprus, Nikosia
- ▶ Universidade Portucalense, Porto
- ▶ EDU-RES Chorzów

Welche Programmiersprache?

- ▶ Eignet sich die Programmiersprache für Anfänger?
- ▶ Eignet sich die Programmiersprache zur Verwendung in anderen Fächern, z.B. im naturwissenschaftlichen Unterricht?
- ▶ Eignet sich die Programmiersprache auch für reale Anwendungen in Wissenschaft und Industrie?

Welche Programmiersprache?



- ▶ Eignet sich die Programmiersprache für Anfänger? ✓
- ▶ Eignet sich die Programmiersprache zur Verwendung in anderen Fächern, z.B. im naturwissenschaftlichen Unterricht? ✓
- ▶ Eignet sich die Programmiersprache auch für reale Anwendungen in Wissenschaft und Industrie? ✓

Welche Programmiersprache?



- ▶ Eignet sich die Programmiersprache für Anfänger? ✓
- ▶ Eignet sich die Programmiersprache zur Verwendung in anderen Fächern, z.B. im naturwissenschaftlichen Unterricht? ✓
- ▶ Eignet sich die Programmiersprache auch für reale Anwendungen in Wissenschaft und Industrie? ✓

Python 2 oder Python 3?

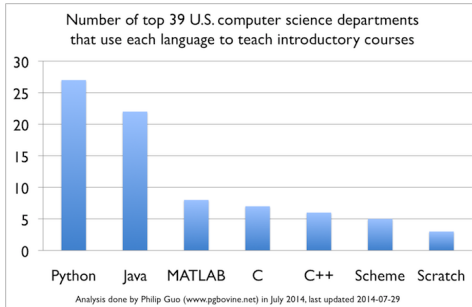
Welche Programmiersprache?



- ▶ Eignet sich die Programmiersprache für Anfänger? ✓
- ▶ Eignet sich die Programmiersprache zur Verwendung in anderen Fächern, z.B. im naturwissenschaftlichen Unterricht? ✓
- ▶ Eignet sich die Programmiersprache auch für reale Anwendungen in Wissenschaft und Industrie? ✓

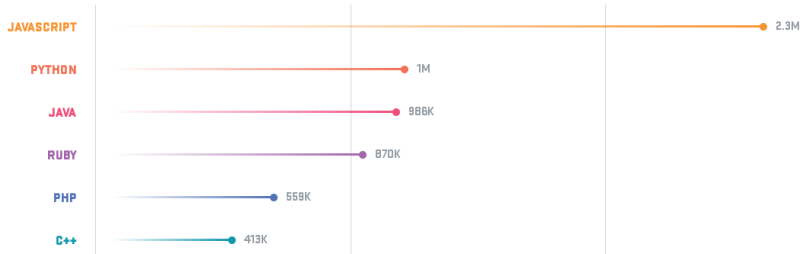
~~Python 2 oder~~ Python 3!

Was andere denken



<http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext>

populärste Programmiersprachen auf Github 2017 (octoverse.github.com)



Warum Python?

- ▶ Eigenschaften
- ▶ Verfügbarkeit
- ▶ Umfeld

Warum Python?

- ▶ Eigenschaften
 - ▶ niedrige Lernbarriere durch expressiven und gut lesbaren Code
 - ▶ keine Fixierung auf ein Programmierparadigma
 - ▶ objektorientiertes Programmieren
 - ▶ Elemente der funktionalen Programmierung
 - ▶ interpretierte Sprache
 - ▶ schnelles Feedback
 - ▶ exploratives Lernen möglich
- ▶ Verfügbarkeit
- ▶ Umfeld

Warum Python?

niedrige Lernbarriere durch expressiven und gut lesbaren Code

Python

```
for n in range(5):  
    print(n, n**2)
```

- ▶ Einrückungen sind syntaktisch relevant
- ▶ keine Typdeklaration (*duck typing*)

Fortran

```
PROGRAM Squares  
DO n = 0, 4  
    PRINT '(2I4)', n, n**2  
END DO  
END PROGRAM Squares
```

C

```
#include <stdio.h>  
  
void main(){  
    int n;  
    for(n = 0; n < 5; n++){  
        printf("%4i %4i\n", n, n*n);  
    }  
}
```

Warum Python?

- ▶ Eigenschaften
- ▶ Verfügbarkeit
- ▶ Umfeld

Warum Python?

- ▶ Eigenschaften
- ▶ Verfügbarkeit
 - ▶ frei verfügbar für Windows, MacOS, Linux, ...
 - ▶ umfangreiche Distributionen frei verfügbar
z.B. Anaconda (www.anaconda.com), ~3 GB
vor allem für wissenschaftliche Anwendungen, Datenanalyse ...
enthält auch Entwicklungsumgebung Spyder
 - ▶ Python kann webbasiert über Jupyterhub zur Verfügung gestellt werden
- ▶ Umfeld

Warum Python?

- ▶ Eigenschaften
- ▶ Verfügbarkeit
- ▶ Umfeld
 - ▶ umfangreiche freie Programmbibliotheken, auch im wissenschaftlichen Bereich
 - ▶ Jupyter Notebook
 - ▶ webbasiertes Arbeiten mit Python
 - ▶ Möglichkeit, Lernmaterialien zu verteilen
 - ▶ Möglichkeit, die Arbeit im Unterricht zu dokumentieren
 - ▶ nbgrader
 - System zur Verteilung, dem Einsammeln und Korrigieren von Jupyter Notebooks

Das wissenschaftliche Ökosystem von Python



<https://www.scipy-lectures.org/>

- ▶ Matrizen als Objekte (ndarray)
- ▶ numerische Integration und Lösung von Differentialgleichungen
- ▶ lineare Algebra
- ▶ statistische Funktionen
- ▶ spezielle Funktionen
- ▶ graphische Darstellungen
- ▶ Bildbearbeitung
- ▶ maschinelles Lernen
- ▶ Bearbeitung großer Datensätze
- ▶ symbolische Mathematik
- ▶ und vieles mehr ...

Jupyter notebook



Taylorreihe

Die Taylorreihe einer unendlich oft differenzierbaren Funktion $f(x)$ an der Stelle x_0 ist durch

$$f(x) \approx \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n$$

gegeben. Im Folgenden sehen wir uns an, wie eine Funktion angenähert wird, wenn die obere Grenze der Summe durch einen variablen endlichen Wert ersetzt wird.

```
In [1]: from ipynbwidgets import interact, widgets
```

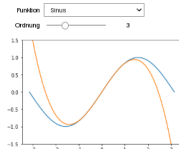
```
In [2]: from math import factorial
import numpy as np
import matplotlib.pyplot as plt
```

```
In [3]: class Taylorseries:
    def __init__(self, coeffs):
        self.coeffs = coeffs

    def __call__(self, x, order):
        p = np.poly1d(self.coeffs[:order+1])
        return p(x)

    maxorder = 10
    taylor_sin = Taylorseries([(n-1)**((n-1)/2)/factorial(n) if n % 2 else 0
                              for n in range(maxorder, -1, -1)])
    taylor_cos = Taylorseries([(n-1)**((n-1)/2)/factorial(n) if not (n % 2) else 0
                              for n in range(maxorder, -1, -1)])
```

```
In [4]: @interact(funcs=widgets.Dropdown(options=['sin', (np.sin, taylor_sin),
                                                'cos', (np.cos, taylor_cos)]),
                description='Funktion'),
        order=widgets.IntSlider(min=0, max=maxorder, value=0,
                                description='Ordnung')
    def __call__(self, funcs, order):
        f, ftaylor = funcs
        x = np.linspace(-np.pi, np.pi)
        plt.plot(x, f(x))
        plt.plot(x, ftaylor(x, order))
        plt.ylim(-1.5, 1.5)
        plt.show()
```



- ▶ läuft im Browser
- ▶ Notebookelemente:
 - ▶ Textzellen
 - ▶ Codezellen
 - ▶ Formeln
 - ▶ graphische Ausgabe
 - ▶ Widgets
- ▶ Umwandlung in verschiedene Formate (HTML, PDF, ...) möglich

Jupyter notebook



Taylorreihe

Die Taylorreihe einer unendlich oft differenzierbaren Funktion $f(x)$ an der Stelle x_0 ist durch

$$f(x) \approx \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n$$

gegeben. Im Folgenden sehen wir uns an, wie eine Funktion angewandt wird, wenn die obere Grenze der Summe durch einen variablen endlichen Wert ersetzt wird.

```
In [1]: from ipynbwidgets import interact, widgets
```

```
In [2]: from math import factorial
import numpy as np
import matplotlib.pyplot as plt
```

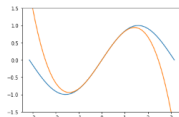
```
In [3]: class Taylorseries:
def __init__(self, coeffs):
    self.coeffs = coeffs

def __call__(self, x, order):
    p = np.polydiv(self.coeffs, order-1:1)
    return p(x)

maxorder = 10
taylor_sin = Taylorseries([(n-1)**((n-1)/2)/factorial(n) if n % 2 else 0
                           for n in range(maxorder, -1, -1)])
taylor_cos = Taylorseries([(n-1)**((n-1)/2)/factorial(n) if not (n % 2) else 0
                           for n in range(maxorder, -1, -1)])
```

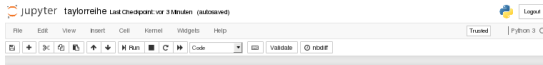
```
In [4]: @interact(functs=widgets.Dropdown(options=['sinus': (np.sin, taylor_sin),
                                                'cosinus': (np.cos, taylor_cos)),
                                                description='Funktion'),
               order=widgets.IntSlider(min=0, max=maxorder, value=0,
                                       description='Ordnung')
def g(functs, order):
    f, ftaylor = functs
    x = np.linspace(-np.pi, np.pi)
    plt.plot(x, f(x))
    plt.plot(x, ftaylor(x, order))
    plt.ylim(-1.5, 1.5)
    plt.show()
```

Funktion:
Ordnung:



- ▶ läuft im Browser
- ▶ Notebookelemente:
 - ▶ Textzellen
 - ▶ Codezellen
 - ▶ Formeln
 - ▶ graphische Ausgabe
 - ▶ Widgets
- ▶ Umwandlung in verschiedene Formate (HTML, PDF, ...) möglich

Jupyter notebook



Taylorreihe

Die Taylorreihe einer unendlich oft differenzierbaren Funktion $f(x)$ an der Stelle x_0 ist durch

$$f(x) \approx \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n$$

gegeben. Im Folgenden sehen wir uns an, wie eine Funktion angenähert wird, wenn die obere Grenze der Summe durch einen variablen endlichen Wert ersetzt wird.

```
In [1]: from ipynbwidgets import interact, widgets
```

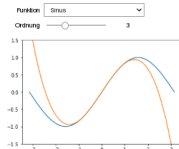
```
In [2]: from math import factorial
import numpy as np
import matplotlib.pyplot as plt
```

```
In [3]: class Taylorseries:
    def __init__(self, coeffs):
        self.coeffs = coeffs

    def __call__(self, x, order):
        p = np.poly1d(self.coeffs, order=1)
        return p(x)

    maxorder = 10
    taylor_sin = Taylorseries([(n-1)/(2*factorial(n)) if n % 2 else 0
                              for n in range(maxorder, -1, -1)])
    taylor_cos = Taylorseries([(n-1)/(2*factorial(n)) if not (n % 2) else 0
                              for n in range(maxorder, -1, -1)])
```

```
In [4]: @interact(funcs=widgets.Dropdown(options=['sinus': (np.sin, taylor_sin),
                                                'cosinus': (np.cos, taylor_cos)),
                                                description='Funktion'),
                order=widgets.IntSlider(min=0, max=maxorder, value=0,
                                         description='Ordnung')
    def g(funcs, order):
        f, ftaylor = funcs
        x = np.linspace(-np.pi, np.pi)
        plt.plot(x, f(x))
        plt.plot(x, ftaylor(x, order))
        plt.ylim(-1.5, 1.5)
        plt.show()
```



- ▶ läuft im Browser
- ▶ Notebookelemente:
 - ▶ Textzellen
 - ▶ Codezellen
 - ▶ Formeln
 - ▶ graphische Ausgabe
 - ▶ Widgets
- ▶ Umwandlung in verschiedene Formate (HTML, PDF, ...) möglich

Jupyter notebook

JupyterLab interface showing the top toolbar with menus (File, Edit, View, Insert, Cell, Format, Widgets, Help) and buttons for Trusted, Python 3, Run, Validate, and Notebook.

Taylorreihe

Die Tayloreihe einer unendlich oft differenzierbaren Funktion $f(x)$ an der Stelle x_0 ist durch

$$f(x) \equiv \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n$$

gegeben. Im Folgenden sehen wir uns an, wie eine Funktion angereichert wird, wenn die obere Grenze der Summe durch einen variablen endlichen Wert ersetzt wird.

```
In [1]: from ipywidgets import interact, widgets
```

```
In [2]: from math import factorial
import numpy as np
import matplotlib.pyplot as plt
```

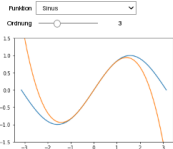
```
In [3]: class TaylorSeries:
def __init__(self, coeffs):
    self.coeffs = coeffs

def __call__(self, x, order):
    p = np.polydiv(self.coeffs[1:order-1:1])
    return p(x)

mxorder = 10
taylor_sin = TaylorSeries([(-1)**((n-1)/2)/factorial(n) if n % 2 else 0
                           for n in range(mxorder-1, -1, -1)])
taylor_cos = TaylorSeries([(1)**((n-1)/2)/factorial(n) if not (n % 2) else 0
                           for n in range(mxorder, -1, -1)])
```

```
In [4]: @interact(funcs=widgets.Dropdown(options={'Sims': (np.sin, taylor_sin),
                                                    'Kosinus': (np.cos, taylor_cos)),
                  description='Funktion'),
          order=widgets.IntSlider(min=0, max=order, value=0,
                                  description='Ordnung'))

def g(funcs, order):
    f, ftaylor = funcs
    x = np.linspace(-np.pi, np.pi)
    plt.plot(x, f(x))
    plt.plot(x, ftaylor(x, order))
    plt.gcf().axis([-1.5, 1.5])
    plt.show()
```



- ▶ läuft im Browser
- ▶ Notebookelemente:
 - ▶ Textzellen
 - ▶ Codezellen
 - ▶ **Formeln**
 - ▶ graphische Ausgabe
 - ▶ Widgets
- ▶ Umwandlung in verschiedene Formate (HTML, PDF, ...) möglich

Jupyter notebook



Taylorreihe

Die Taylorreihe einer unendlich oft differenzierbaren Funktion $f(x)$ an der Stelle x_0 ist durch

$$f(x) \approx \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n$$

gegeben. Im Folgenden sehen wir uns an, wie eine Funktion angenähert wird, wenn die obere Grenze der Summe durch einen variablen endlichen Wert ersetzt wird.

```
In [1]: from ipynbwidgets import interact, widgets
```

```
In [2]: from math import factorial
import numpy as np
import matplotlib.pyplot as plt
```

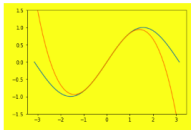
```
In [3]: class Taylorseries:
    def __init__(self, coeffs):
        self.coeffs = coeffs

    def __call__(self, x, order):
        p = np.polydiv(self.coeffs, order-1:1)
        return p(x)

    maxorder = 10
    taylor_sin = Taylorseries([(n-1)**((n-1)/2)/factorial(n) if n % 2 else 0
                               for n in range(maxorder, -1, -1)])
    taylor_cos = Taylorseries([(n-1)**((n-1)/2)/factorial(n) if not (n % 2) else 0
                               for n in range(maxorder, -1, -1)])
```

```
In [4]: @interact(funcs=widgets.Dropdown(options=['sin', (np.sin, taylor_sin),
                                                'cos', (np.cos, taylor_cos)]),
                 description='Funktion'),
        order=widgets.IntSlider(min=0, max=maxorder, value=0,
                                description='Ordnung')
    def __call__(self, order):
        f, ftaylor = funcs
        x = np.linspace(-np.pi, np.pi)
        plt.plot(x, f(x))
        plt.plot(x, ftaylor(x, order))
        plt.ylim(-1.5, 1.5)
        plt.show()
```

Funktion: sin
Ordnung: 3



- ▶ läuft im Browser
- ▶ Notebookelemente:
 - ▶ Textzellen
 - ▶ Codezellen
 - ▶ Formeln
 - ▶ graphische Ausgabe
 - ▶ Widgets
- ▶ Umwandlung in verschiedene Formate (HTML, PDF, ...) möglich

Jupyter notebook



Taylorreihe

Die Taylorreihe einer unendlich oft differenzierbaren Funktion $f(x)$ an der Stelle x_0 ist durch

$$f(x) \approx \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n$$

gegeben. Im Folgenden sehen wir uns an, wie eine Funktion angenähert wird, wenn die obere Grenze der Summe durch einen variablen endlichen Wert ersetzt wird.

```
In [1]: from ipynbwidgets import interact, widgets
```

```
In [2]: from math import factorial
import numpy as np
import matplotlib.pyplot as plt
```

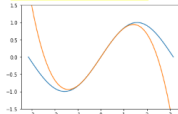
```
In [3]: class Taylorseries:
    def __init__(self, coeffs):
        self.coeffs = coeffs

    def __call__(self, x, order):
        p = np.poly1d(self.coeffs[:order+1])
        return p(x)

maxorder = 10
taylor_sin = Taylorseries([(n-1)**((n-1)/2)/factorial(n) if n % 2 else 0
                           for n in range(maxorder, -1, -1)])
taylor_cos = Taylorseries([(n-1)**((n-1)/2)/factorial(n) if not (n % 2) else 0
                           for n in range(maxorder, -1, -1)])
```

```
In [4]: @interact(funcs=widgets.Dropdown(options={'sin': (np.sin, taylor_sin),
                                                'cos': (np.cos, taylor_cos)}),
                  description='Funktion'),
        order=widgets.IntSlider(min=0, max=maxorder, value=0,
                                description='Ordnung')
def g(funcs, order):
    f, ftaylor = funcs
    x = np.linspace(-np.pi, np.pi)
    plt.plot(x, f(x))
    plt.plot(x, ftaylor(x, order))
    plt.ylim(-1.5, 1.5)
    plt.show()
```

Funktion: sin
Ordnung: 3



- ▶ läuft im Browser
- ▶ Notebookelemente:
 - ▶ Textzellen
 - ▶ Codezellen
 - ▶ Formeln
 - ▶ graphische Ausgabe
 - ▶ **Widgets**
- ▶ Umwandlung in verschiedene Formate (HTML, PDF, ...) möglich

Jupyter notebook



Taylorreihe

Die Taylorreihe einer unendlich oft differenzierbaren Funktion $f(x)$ an der Stelle x_0 ist durch

$$f(x) \approx \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n$$

gegeben. Im Folgenden sehen wir uns an, wie eine Funktion angenähert wird, wenn die obere Grenze der Summe durch einen variablen endlichen Wert ersetzt wird.

```
In [1]: from ipynbwidgets import interact, widgets
```

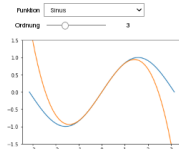
```
In [2]: from math import factorial
import numpy as np
import matplotlib.pyplot as plt
```

```
In [3]: class Taylorseries:
    def __init__(self, coeffs):
        self.coeffs = coeffs

    def __call__(self, x, order):
        p = np.poly1d(self.coeffs[:order+1])
        return p(x)

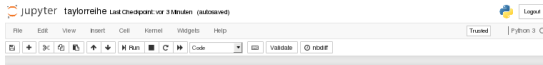
    maxorder = 10
    taylor_sin = Taylorseries([(n-1)**((n-1)/2)/factorial(n) if n % 2 else 0
                               for n in range(maxorder, -1, -1)])
    taylor_cos = Taylorseries([(n-1)**((n-1)/2)/factorial(n) if not (n % 2) else 0
                               for n in range(maxorder, -1, -1)])
```

```
In [4]: @interact(funcs=widgets.Dropdown(options=['sin', (np.sin, taylor_sin),
                                                'cos', (np.cos, taylor_cos)]),
                  description='Funktion'),
        order=widgets.IntSlider(min=0, max=maxorder, value=0,
                                description='Ordnung')
def g(funcs, order):
    f, ftaylor = funcs
    x = np.linspace(-np.pi, np.pi)
    plt.plot(x, f(x))
    plt.plot(x, ftaylor(x, order))
    plt.ylim(-1.5, 1.5)
    plt.show()
```



- ▶ läuft im Browser
- ▶ Notebookelemente:
 - ▶ Textzellen
 - ▶ Codezellen
 - ▶ Formeln
 - ▶ graphische Ausgabe
 - ▶ Widgets
- ▶ Umwandlung in verschiedene Formate (HTML, PDF, ...) möglich

Jupyter notebook



Taylorreihe

Die Taylorreihe einer unendlich oft differenzierbaren Funktion $f(x)$ an der Stelle x_0 ist durch

$$f(x) \approx \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n$$

gegeben. Im Folgenden sehen wir uns an, wie eine Funktion angenähert wird, wenn die obere Grenze der Summe durch einen variablen endlichen Wert ersetzt wird.

```
In [1]: from ipynbwidgets import interact, widgets
```

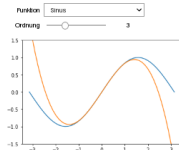
```
In [2]: from math import factorial
import numpy as np
import matplotlib.pyplot as plt
```

```
In [3]: class Taylorseries:
    def __init__(self, coeffs):
        self.coeffs = coeffs

    def __call__(self, x, order):
        p = np.poly1d(self.coeffs[:order+1])
        return p(x)

    maxorder = 10
    taylor_sin = Taylorseries([(n-1)**((n-1)/2)/factorial(n) if n % 2 else 0
                               for n in range(maxorder, -1, -1)])
    taylor_cos = Taylorseries([(n-1)**((n-1)/2)/factorial(n) if not (n % 2) else 0
                               for n in range(maxorder, -1, -1)])
```

```
In [4]: @interact(funcs=widgets.Dropdown(options=[('sin', (np.sin, taylor_sin),
                                                  'cosine': (np.cos, taylor_cos)),
                                                  ('sinus': (np.sin, taylor_sin),
                                                  'cosinus': (np.cos, taylor_cos))],
                                                  description='Funktion'),
                  order=widgets.IntSlider(min=0, max=maxorder, value=0,
                                                  description='Ordnung')
    def g(funcs, order):
        f, ftaylor = funcs
        x = np.linspace(-np.pi, np.pi)
        plt.plot(x, f(x))
        plt.plot(x, ftaylor(x, order))
        plt.ylim(-1.5, 1.5)
        plt.show()
```



- ▶ läuft im Browser
- ▶ Notebookelemente:
 - ▶ Textzellen
 - ▶ Codezellen
 - ▶ Formeln
 - ▶ graphische Ausgabe
 - ▶ Widgets
- ▶ Umwandlung in verschiedene Formate (HTML, PDF, ...) möglich

➔ Demo

nbgrader

System zum Verteilen, Einsammeln und Korrigieren von Jupyter Notebooks

Quelle:

<https://github.com/jupyter/nbgrader>

Dokumentation:

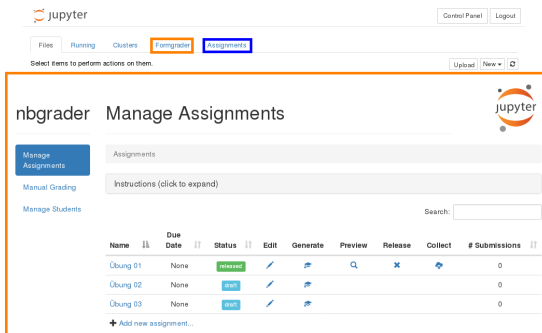
<http://nbgrader.readthedocs.io/en/stable/>



Jessica B. Hamrick

- ▶ Kommandozeilenbefehle für Verwaltung von Übungsaufgaben und Übungsteilnehmern
- ▶ Erweiterung für Jupyter Notebook für graphisches Benutzerinterface
- ▶ Verwendung mit Jupyterhub

nbgrader



The screenshot shows the nbgrader web interface. At the top, there's a Jupyter logo and navigation tabs: Files, Running, Clusters, Formgrader, and Assignments. The Assignments tab is active. Below the tabs, there's a search bar and a table of assignments. The table has columns for Name, Due Date, Status, Edit, Generate, Preview, Release, Collect, and # Submissions. Three assignments are listed: Übung 01, Übung 02, and Übung 03, all with a status of 'released'.

nbgrader Manage Assignments

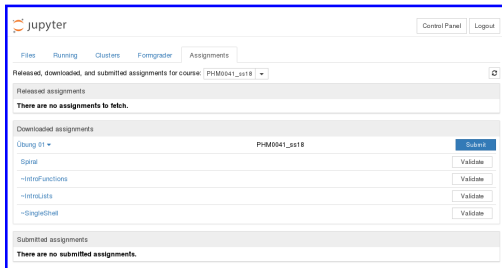
Assignments

Instructions (click to expand)

Search:

Name	Due Date	Status	Edit	Generate	Preview	Release	Collect	# Submissions
Übung 01	None	released	edit	generate	preview	release	collect	0
Übung 02	None	draft	edit	generate				0
Übung 03	None	draft	edit	generate				0

+ Add new assignment...



The screenshot shows the nbgrader web interface for a specific course. The Assignments tab is active. Below the tabs, there's a dropdown menu for the course, currently set to 'PHM0041_ss18'. The interface is divided into three sections: Released assignments, Downloaded assignments, and Submitted assignments. The Released assignments section shows 'There are no assignments to fetch.' The Downloaded assignments section shows a list of assignments with buttons for 'Submit', 'Validate', and 'Generate'. The Submitted assignments section shows 'There are no submitted assignments.'

Released, downloaded, and submitted assignments for course: PHM0041_ss18

Released assignments

There are no assignments to fetch.

Downloaded assignments

Assignment	PHM0041_ss18	Submit
Übung 01		Submit
Spiral		Validate
~IntroFunctions		Validate
~IntroLists		Validate
~SingleShell		Validate

Submitted assignments

There are no submitted assignments.


nbgrader

In []: ID: cell-14980c0124453639 Autograded answer

```
def diagonalsum(sidelength):  
    """Compute sum of diagonal elements on an integer spiral  
  
    sidelength: side length of the grid, needs to be odd  
    """  
    # YOUR CODE HERE  
    raise NotImplementedError()
```

 ID: cell-1e7eda4d39d41249 Read-only

Test your solution by executing the following cell. If there is no output, everything is fine. Otherwise you have to go hunting coding errors.

In []: Points: 5  ID: cell-659caf13848a35a3 Autograder tests

```
assert diagonalsum(7) == 261, "The example given above does not work out correctly"  
for n in (11, 101, 1001):  
    expected = (4*n**3+3*n**2+8*n-9)/6  
    assert diagonalsum(n) == expected, "Side length {} yields an incorrect result"
```

-
- Manually graded answer
- Autograded answer
- Autograder tests**
- Read-only

[→ Demo](#)