

Teaching coding with nbgrader – lessons learned –

Gert-Ludwig Ingold
Universität Augsburg

Teaching situation

- ▶ programming language: Python

Dlaczego warto uczyć w języku python? ↗ youtu.be/CEmg3gdFlaE

Why teach Python? ↗ youtu.be/X18jt2lVmJY

- ▶ low entry barrier
 - ▶ expressive code
 - ▶ well suited for serious work
 - ▶ freely available, “batteries included
 - ▶ Anaconda distribution for coherent environment on current operating systems
- ▶ teaching programming to physicists and materials scientists since 2010, typically 10+ students
- ▶ next year: mandatory course for 100+ students

Coping with heterogeneity

- ▶ students without programming experience and (sometimes) code writing apprehension
- ▶ students with knowledge in another programming language
code often does not look very pythonic
example: loop over objects vs. loop over indices

students with programming experience should not be bored and should obtain an interesting result from their code

- ▶ *but* scientific applications are often considered as an extra mental burden

nbgrader

- ▶ tool that facilitates creating and grading assignments in the Jupyter notebook
- ▶ distributing and collecting assignments
- ▶ automatic grading by means of unit tests
- ▶ manual grading possible
- ▶ feedback can be given to students

- ▶ freely available
- ▶ open source
- ▶ latest version: 0.6.0 released end of August 2019

nbgrader on Github

github.com/jupyter/nbgrader/

The screenshot shows the GitHub repository page for jupyter/nbgrader. At the top, there's a navigation bar with links for Pull requests, Issues, Marketplace, and Explore. Below this, the repository name 'jupyter / nbgrader' is displayed, along with statistics: 41 Unwatch, 791 Star, and 215 Fork. The main content area shows the repository description: 'A system for assigning and grading notebooks' with a link to the documentation. Below the description are tags for nbgrader, jupyter, jupyter-notebook, jupyterhub, teaching, and grading. A progress bar indicates 2,753 commits, 6 branches, 0 packages, 19 releases, 56 contributors, and BSD-3-Clause license. The 'Branch: master' dropdown is set to 'master', and there are buttons for 'New pull request', 'Create new file', 'Upload files', 'Find File', and 'Clone or download'. The commit history table lists recent commits by jhamrick, including a merge pull request #1204 and updates to .github, appveyor, demos, paper, tools, .bowerrc, .coveragerc, and .gitignore.

Search or jump to... [Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)

jupyter / nbgrader [Unwatch](#) 41 [Star](#) 791 [Fork](#) 215

[Code](#) [Issues](#) 97 [Pull requests](#) 4 [Actions](#) [Projects](#) 0 [Wiki](#) [Security](#) [Insights](#)

A system for assigning and grading notebooks <https://nbgrader.readthedocs.io/>

[nbgrader](#) [jupyter](#) [jupyter-notebook](#) [jupyterhub](#) [teaching](#) [grading](#)

2,753 commits 6 branches 0 packages 19 releases 56 contributors BSD-3-Clause

Branch: master [New pull request](#) [Create new file](#) [Upload files](#) [Find File](#) [Clone or download](#)

jhamrick Merge pull request #1204 from jhamrick/force-generate ... Latest commit c2a3e79 23 days ago

.github	improve issue template to explain logic behind filling it out	2 years ago
appveyor	add appveyor ci files	3 years ago
demos	Include course id in grader config files	25 days ago
nbgrader	Merge pull request #1204 from jhamrick/force-generate	23 days ago
paper	Ensure consistency in capitalizing Jupyter Notebook	9 months ago
tools	Remove tests from releasing script	24 days ago
.bowerrc	Turn formgrader into a server extension	3 years ago
.coveragerc	Create new databases with most recent alembic version	2 years ago
.gitignore	Formatting; gitignore; added @check_notebook_dir	4 months ago

Github issues for nbgrader

report and discuss problems or new features

The screenshot shows the GitHub interface for the `jupyter/nbgrader` repository. At the top, there's a navigation bar with links for Pull requests, Issues, Marketplace, and Explore. Below this, the repository name is displayed along with statistics: 41 Unwatched, 791 Stars, and 215 Forks. A secondary navigation bar includes links for Code, Issues (97), Pull requests (4), Actions, Projects (0), Wiki, Security, and Insights. A search bar is present with the filter `is:issue is:open`. To the right of the search bar are filters for Labels (17) and Milestones (5), and a green 'New issue' button. The main content area lists 97 open issues. The first issue is a question about coding constraints, followed by a report of a failure with self-signed certificates, a request for formgrader support in a non-home directory, a feature request for PDF export, a report of unavailable formgrader services, a request for an import option, and a report of missing feedback forms for HTML submissions.

Search or jump to...

Pull requests Issues Marketplace Explore

jupyter / nbgrader

Unwatch 41 Star 791 Fork 215

Code Issues 97 Pull requests 4 Actions Projects 0 Wiki Security Insights

Filters 🔍 is:issue is:open Labels 17 Milestones 5 New issue

97 Open ✓ 512 Closed Author Labels Projects Milestones Assignee Sort

- Question (not an issue): will nbgrader work for questions with coding constraints?
#1221 opened 6 hours ago by dhawaljoh 1
- course_list fails using https with self-signed certificate
#1220 opened 23 hours ago by nickhine
- Getting formgrader to work in non-home directory
#1218 opened 5 days ago by timbers 1
- Option to export all feedback as PDF
#1216 opened 5 days ago by myedibleenso
- Course List Extension: There are no available formgrader services.
#1215 opened 6 days ago by sigurdurb
- Add import Students option in Formgrader app
#1214 opened 6 days ago by whosilwhatnow
- Feedback forms are not generated if student's submission contains a .html version of the notebook
#1211 opened 18 days ago by timbers 3

Pull requests for nbgrader

contribute to the code or the documentation

The screenshot shows the GitHub interface for the `jupyter/nbgrader` repository. The top navigation bar includes links for Pull requests, Issues, Marketplace, and Explore. The repository name `jupyter / nbgrader` is displayed, along with statistics: 41 Unwatch, 791 Star, and 215 Fork. Below this, the 'Pull requests' tab is selected, showing 4 open pull requests. A search filter `is:pr is:open` is applied. The list of pull requests includes:


- CourseDir.format_path: supports absolute paths in nbgrader_step** (1 comment, #1222, opened 2 hours ago by nthiery)
- Improve coverage of nbgraderformat** (1 comment, #1208, opened 22 days ago by jhamrick, labels: bugfix, maintenance)
- make path relative** (1 comment, #1177, opened on Aug 16 by phauslin, labels: bugfix, needs work)
- Langsupport: Validation refactoring and better language support for autograder test cells** (2 comments, #1173, opened on Aug 15 by sigurdurb, labels: enhancement, needs work)

A ProTip at the bottom states: "Exclude everything labeled bug with -label:bug."



nbgrader documentation

`nbgrader.readthedocs.io`

 nbgrader
stable


Search docs

USER DOCUMENTATION

- Interface highlights
- Installation
- The philosophy and the approach
- Creating and grading assignments
- Managing the database
- Managing assignment files
- Managing assignment files manually
- Autograding resources
- Frequently asked questions
- Advanced topics
- API library documentation

CONFIGURATION

- Customizing how the student version of an assignment looks
- The `nbgrader_config.py` file
- Configuration options
- Command line options

 Read the Docs v: stable ▾

[Docs](#) » nbgrader

[Edit on GitHub](#)

nbgrader

`nbgrader` is a tool that facilitates creating and grading assignments in the Jupyter notebook. It allows instructors to easily create notebook-based assignments that include both coding exercises and written free-responses. `nbgrader` then also provides a streamlined interface for quickly grading completed assignments.

For an overview and demonstration of nbgrader's core functionality, check out the talk on nbgrader given at SciPy 2017:



Jupyter notebook and Jupyterhub – pros and cons

Jupyter notebook

- ▶ notebook allows to guide students through a problem set
- ▶ problems related to out-of-order execution
- ▶ students might think that programming in Python necessarily implies working with a Jupyter notebook, they should know about IDEs (spyder, ...) and editors (vim, ...)

Jupyterhub

- ▶ easily accessible interface to problem sets
- ▶ no need to install Python on local computer
- ▶ consistent working environment for all students
- ▶ *but* maybe beginners should gather experience with Python on their own computer

Functions



```
In [ ]: 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()
```

- ▶ need to introduce functions very early in the course
but special aspects (no arguments, no return value, default arguments, keyword arguments, ...) not needed
- ▶ students get used to logically structured code early on
but they do not do it themselves
- ▶ include docstrings to make task well defined
students get used to the idea that a function contains a docstring
but they are not writing the docstring by themselves

Grading with tests

```
In [ ]: 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".format(n)
```

- ▶ test-driven development
 - but* tests are not developed by the students
- ▶ tests allow to give feedback through error messages
 - but* students tend to rely on this feedback instead of developing their own critical view on their code
- ▶ “Stupid mistakes” will be made not only at the beginning
 - ▶ “trivial” standard tests (are results returned, do they have the correct type, ...?)
 - ▶ + tests of specific functionality which should not disclose the solution

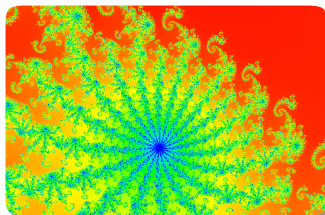
 Points: ID: Autograder tests 

```
result = juliaiter(-0.3+0.4j, -0.8+0.156j, 2, 100)
assert result is not None, 'Does your function return a result?'
assert type(result) == type(1), 'The result should be an integer.'
```

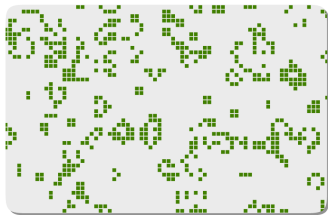
Examples of problem sets I

99	72	43	44	45	46	47	48	49	50	8
98	71	42	21	22	23	24	25	26	51	8
97	70	41	20	7	8	9	10	27	52	8
96	69	40	19	6	1	2	11	28	53	8
95	68	39	18	5	4	3	12	29	54	8
94	67	38	17	16	15	14	13	30	55	8
93	66	37	36	35	34	33	32	31	56	8

a problem from
Project Euler (projecteuler.net)



Julia set

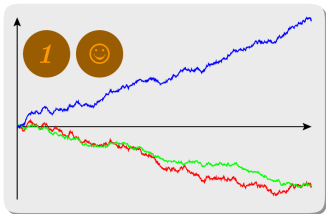


Conway's game of life

3.14159265358979323846264
3383279502884197169399375
105820974945973078164062
8620899867303432534211706
798214808551322306647093
84460955582237253594081
2848111745028410270193852

π to a few thousand digits

Examples of problem sets II



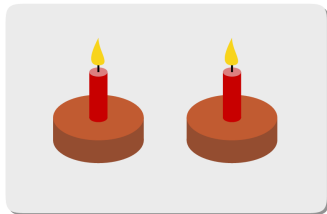
Parrondo paradoxon

$$(2x^4 + 10x^3)(5x^2 - x + 32)$$

symbolic manipulation of
polynomials

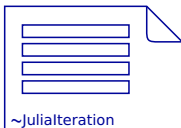
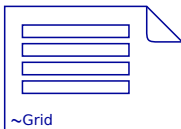
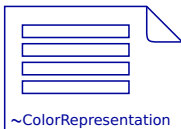


How many primes are times?



birthday problem

Branching paths



should not set the gridsize too large. It is a good idea to start e.g. with a 100×100 grid. Once your code is working, you can increase the number of grid points. For practical purposes, it makes sense to define a threshold for the absolute value of z beyond which you assume that the series for a given starting value z_0 diverges. A good value could be 2. You should also limit the number of iterations. Here, a good value could be 600. Feel free to play around with these values once your code works.

If you want to tackle the problem without any further help, you can jump over the following three points which are intended to give you some help by splitting the complete problem into smaller parts.

1. [Representation of a number by a color](#)
2. [Equally spaced numbers on a grid](#)
3. [Iteration prescription](#)

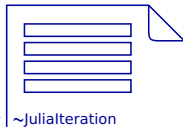
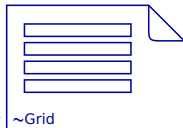
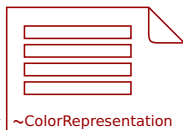
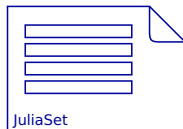
```
In [ ]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

def plot(data, ndim, cmap):
    """Create a 2d graphics from a list of data points

    data: list of ndim2 data between 0 and 1
    ndim: side length of grid
    cmap: matplotlib color map mapping interval from 0 to 1 to color

    """
```

Branching paths



```
def colorbar(cmapname):  
    gradient = np.linspace(0, 1, 256)  
    gradient = np.vstack((gradient, gradient))  
    fig = plt.imshow(gradient, aspect=0.05, extent=(0, 1, 0, 1), cmap=plt.  
    fig.axes.get_yaxis().set_visible(False)  
  
    def display_color(x, cmapname):  
        rgbcolor = cm.get_cmap(name=cmapname)(float(x), bytes=True, alpha=False)  
        print(rgbcolor)  
        return SVG('<svg height="100" width="100">  
                    <rect x="1" y="1" width="50" height="50" fill="rgb(  
                    </svg>'.format(*rgbcolor))
```

We start with one of the colormaps provided by the matplotlib library called hot:

In [3]: `colorbar('hot')`



In the following cell, you can change the argument of the function `display_color` between 0 and 1 and explore how the color changes.

In [4]: `display_color(0.9, 'hot')`

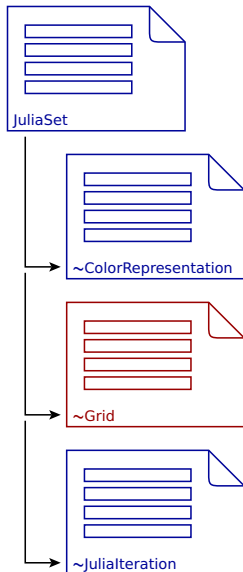
(255, 255, 156)

Out[4]:

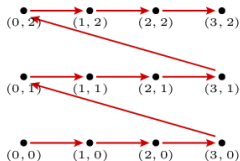


The matplotlib library provides many more colormaps, like e.g. [viridis](#). Take a look at [Colormaps in Matplotlib](#) for more information. Try out a couple of other colormaps.

Branching paths



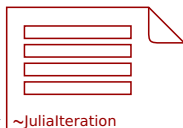
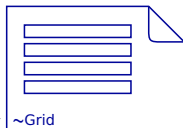
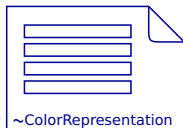
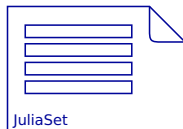
If the code for generating a one-dimensional grid is working, you are ready to extend it to two-dimensional grids. Walk the grid horizontally before making a vertical step like indicated in the following figure.



The output in this case should be the list [(0, 0), (1, 0), (2, 0), (3, 0), (0, 1), (1, 1), (2, 1), (3, 1), (0, 2), (1, 2), (2, 2), (3, 2)].

```
In [ ]: def grid2d(xmin, xmax, nxpts, ymin, ymax, nypts):  
    """Generate a list of coordinate tuples (x, y) on a two-dimensional grid.  
    The points are equally spaced in each of the two directions between  
    (and including) the respective bounds. The grid is first walked along  
    the horizontal (x) direction, then along the vertical (y) direction.  
  
    xmin: lower bound in horizontal direction  
    xmax: upper bound in horizontal direction  
    nxpts: number of points in horizontal direction  
    ymin: lower bound in vertical direction  
    ymax: upper bound in vertical direction  
    nypts: number of points in vertical direction  
    """
```


Branching paths



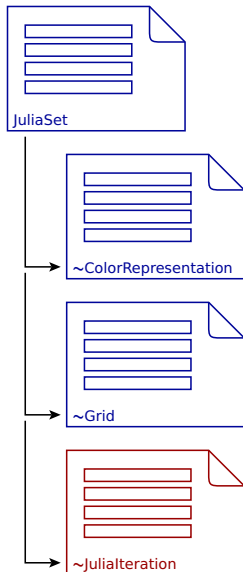
```
In [ ]: def juliaiter(z0, c, threshold, maxiter):  
        """Determine number of iterations needed to cross the threshold  
  
        z0:      initial value for z  
        c:       complex number in iteration prescription  
        threshold: threshold value to be crossed by |z|  
        maxiter: maximum of iterations  
        """  
        # YOUR CODE HERE  
        raise NotImplementedError()
```

Test your solution by executing the following two cells. Everything is fine if no error message is displayed.

```
In [ ]: result = juliaiter(-0.3+0.4j, -0.8+0.156j, 2, 100)  
        assert result is not None, 'Does your function return a result?'  
        assert type(result) == type(1), 'The result should be an integer.'
```

```
In [ ]: import math  
        maxiter = 100  
        threshold = 2  
        assert juliaiter(1, 0, threshold, maxiter) == maxiter, 'There is a problem e  
        z0 = 1.0000001  
        expected = int(math.log(math.log(threshold)/math.log(z0), 2))+1  
        assert juliaiter(z0, 0, threshold, maxiter) == expected, 'There is a problem  
        z0 = -0.3+0.4j  
        c = -0.8+0.156j  
        threshold = abs(z0**8+4*z0**6*c+2*z0**4*(3*c**2+c)+4*z0**2*(c**3+c**2)+c**4+  
        maxiter = 4  
        assert juliaiter(z0, c, threshold+1e-6, maxiter) == maxiter, 'Wrong number c  
        assert juliaiter(z0, c, threshold-1e-6, maxiter) == maxiter-1, 'Wrong number
```

Branching paths



- ▶ individual path through problem possible
- ▶ *but* notebooks are opened in new tabs, it is easy to lose track