

WELCOME TO ODSC WEBINAR



ODSC UPCOMING EVENTS

ODSC KICKSTART BOOTCAMP 2021

San Francisco

November 15th – 18th, 2021



ODSC WEST 2021

San Francisco

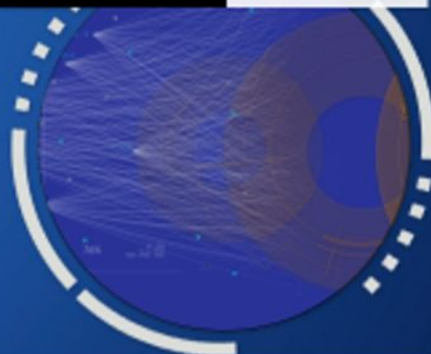
November 16th – 18th, 2021



WEST AI EXPO AND DEMO HALL 2021

Free Event

November 16th – 17th, 2021



IJUNGLE: ANOMALY DETECTION FOR REALLY BIG DATA

Free Webinar

November 23rd, 2021



DATA VISUALIZATION WITH SEABORN BY TEDDY PETROU

Live Training

December 1st & 8th, 2021



WEBCRAWLING AND SOCIAL MEDIA MINING FOR TEXT ANALYSIS AND NLP BY MINERVA SINGH PHD

Live Training

December 14th, 2021





NOVETTA
Part of Accenture Federal Services

Embracing Literate Programming without Compromise: nbdev for Collaborative Projects

Zachary Mueller
Deep Learning Software Engineer



Who am I?

Software Design and Development
major with minors in Computer Science
and Environmental Science

Open Source contributor

A Core Developer of fast.ai libraries

Deep Learning Software Engineer at
Novetta

Maintainer of the AdaptNLP project



What will we be talking about?

- ## What is Literate Programming and nbdev?
- ## How do we integrate this with small software packages?
- ## What are the struggles with only using nbdev?
- ## Solutions towards this problem
- ## What we have done at Novetta
- ## Where is the future heading?

What *is* literate programming?

“Standard” programming

Segregated code, tests, and documentation

Each are developed separately

Time is spent keeping all these ideas aligned together

Documentation is generally done last or as an afterthought, leading to likely lapses in developers’ train of thought, and ideas and understandings during development may not be in the forefront of their mind

Development is normally coupled with metrics such as coverage testing

“Literate” programming

Also called exploratory programming

Revolves around the idea of centralizing code, testing, and documentation to a single source and developed in tandem

Gives the developer a sense of flow and purpose, as each idea is surrounded by its tests and documentation

Is propelled through sandbox environments such as Jupyter Notebooks

Why is literate programming important?

- Faster training and pickup by users
 - By exploring each notebook, new users and teams can quickly understand the mindset of the project and see how the code operates
- Centralized location for all parts of the Software Engineering Pipeline
 - In the notebooks are the relevant source code, documentation, and tests for each aspect of the library
 - This not only helps readability, but also in-turn additionally speeds up the pickup, as mentioned earlier
- Programming with a documentation-focused mindset
 - By giving sandbox creativity, the programmer is allowed to put any thoughts and notes into these notebooks, without compromising the clarity of the end source-code

What and How Are These Sandbox Environments Being Used Today?

What is a sandbox environment? A “use and throw away” mindset

An interactive platform where code is typed in, and immediate feedback is received

Different than “standard” programming, as code is run as iteratively as needed

```
In [1]: # An example print statement in python
        print(f'The sum of 1 and 2 is: {1+2}')
```

The sum of 1 and 2 is: 3

Most data scientists are extensively familiar with this environment, and utilize it daily to build models and perform experiments

Code is quickly tested in this sandbox environment to ensure their idea works, before being thrown into some arbitrary python file, which was the real destination it needed to be

Often the original files are not named well, and get lost to time, such as “Untitled_Notebook_27a”

Usually they are not made with the foresight of someone may want to use them again, and are often messy and hard to follow

Utilizing This Environment as a Developer

Allow for a creative flow

Code, tests, and documentation are all sourced from a single interactive environment (Jupyter Notebooks)

Test-driven development is at the forefront of projects, providing best practices to flourish through an interactive environment

The goal is to take this familiarity, and let this be the foundation for strong software engineering without sacrificing usability and maintainability

```
In [1]: # default_exp math
```

Basic Math

Simplistic math functionalities

```
In [2]: #hide
from fastcore.test import test_eq
```

Below we have some simple functions that perform arithmetic:

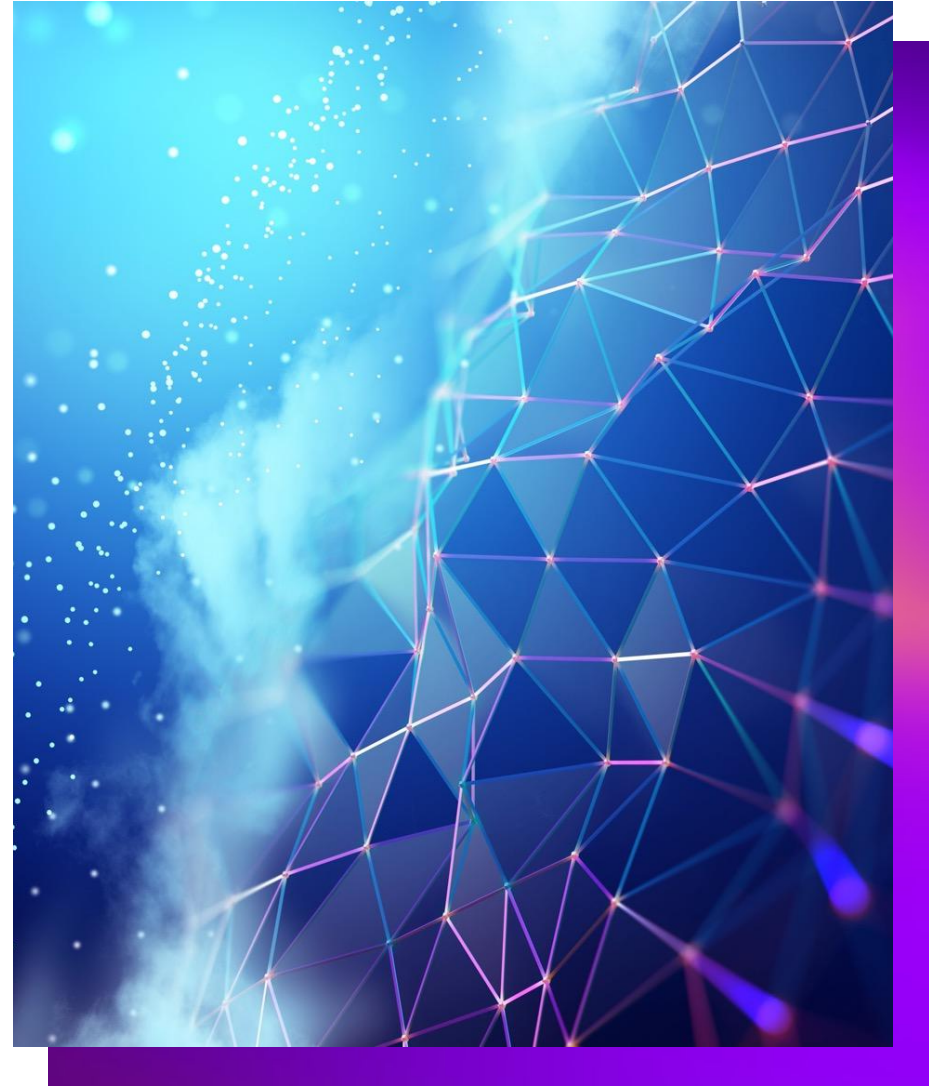
```
In [3]: #export
def addition(a:int, b:int) -> int:
    "Performs the addition of `a` and `b`"
    return a+b
```

```
In [5]: test_eq(addition(1,2), 3) # Testing positive numbers
```

```
In [ ]: test_eq(addition(-12, 7), 5) # Testing negative numbers
```

An example of a Jupyter Notebook

**How do we
integrate
this idea into
small
software
packages?**



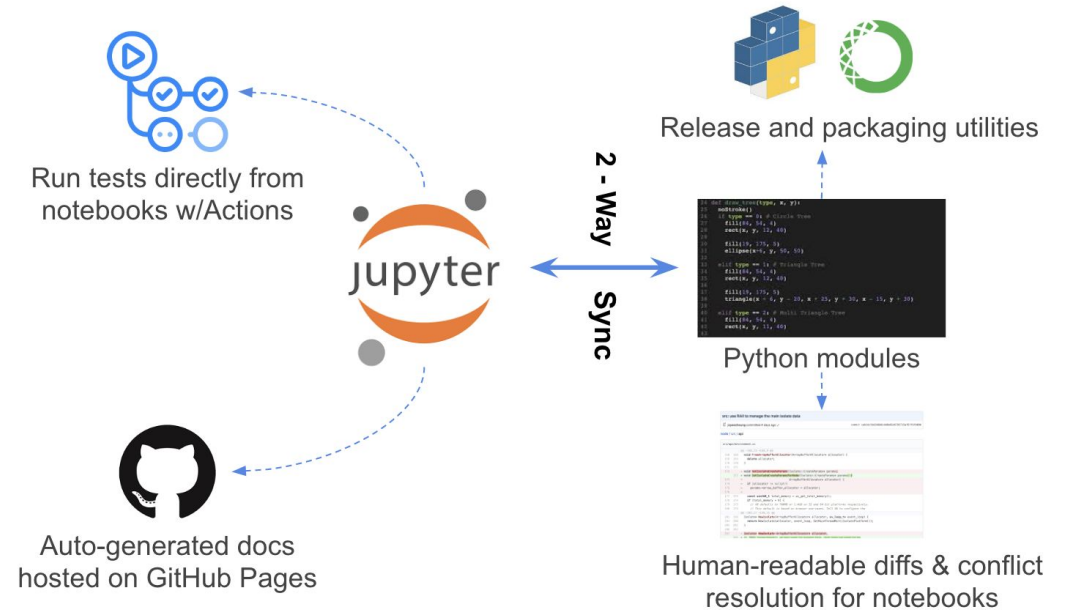
nbdev

A “true” literate coding environment



Leverages on the idea of exploratory programming by centralizing around the Jupyter Notebook

In a single file, we contain:










- Code writing
- Documentation
- Tests



nbdev - What does it look like?

 jupyter 00_core Last Checkpoint: Last Tuesday at 5:14 PM (unsaved changes)  Logout

File Edit View Insert Cell Kernel

```
# AUTOGENERATED! DO NOT EDIT! File to edit: 00_core.ipynb (unless otherwise specified).

__all__ = ['addition', 'multiplication']

# Cell
def addition(a:int,b:int) -> int:
```

Basic math

Simplistic math functionalities

Table of Contents

- [addition](#)
- [multiplication](#)

addition [\[source\]](#)

```
addition ( a : int , b : int )
```

Performs addition of `a` and `b`

```
.....
return a + b
```

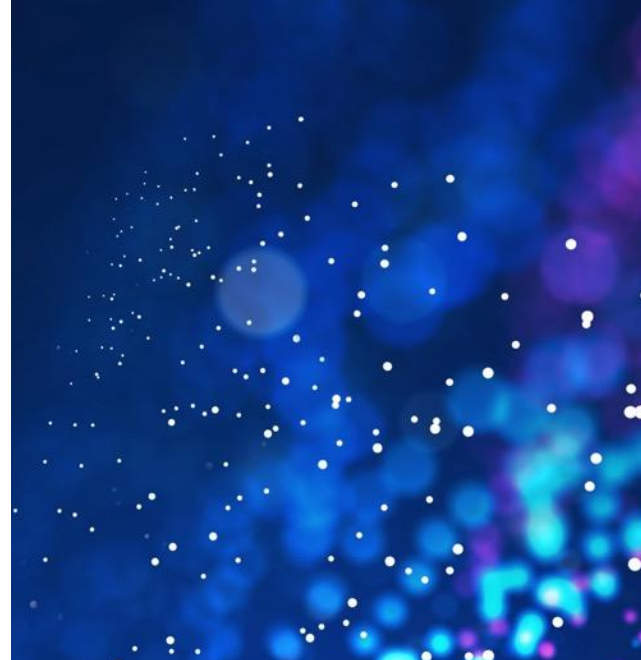

GitHub Actions - Your CI/CD Best Friend

- Through a single Action, we can centralize our **tests, deployment, and quality control** automatically each time code is pushed to GitHub
- They are automated workflows hooked into GitHub itself
 - Can run on any number of events, such as pushing code, merging pull requests, or on software releases
- If leveraged well, they act as a second person checking and reviewing your code in the background as you work

GitHub Actions - Your CI/CD Best Friend

- nbdev leverages GitHub Actions by providing checks for us:
 - Checking no metadata was left in the original notebooks
 - Source code and notebook code is aligned
 - All notebook tests* are run to ensure everything is working properly
 - Done on every commit
- At Novetta we've taken this even further, by having GH Actions check our Docker deployments, testing REST APIs, and provide aggressive version control to our software

**That's cool and all,
but where is this
being used?**



Walk with fastai





nbdev and Its Struggles in the Industry

nbdev and Its Struggles in the Industry

Setup Issues

Existing projects must be “rebuilt” into the nbdev framework

Without an automated tool to do so, even small projects can take > 20+ hours to complete

Documentation needs to be completely re-written to match the framework, if you want to maintain an exploratory style

Integration Issues

Testing in notebooks is very different from testing with standard testing frameworks like pytest, so a choice must be made of tests in the notebook, or externally

Code usually doesn't follow industry standards, such as PEP-8 (Efficiency vs Industry)

Jupyter Notebook review in GitHub is a mess, and there is no decent solution for when reviews scale in size

Viabale Solutions

- **lib2nbdev**
- **docments + nbverbose**
- **nbagile**
- **nbdev + PyTest**

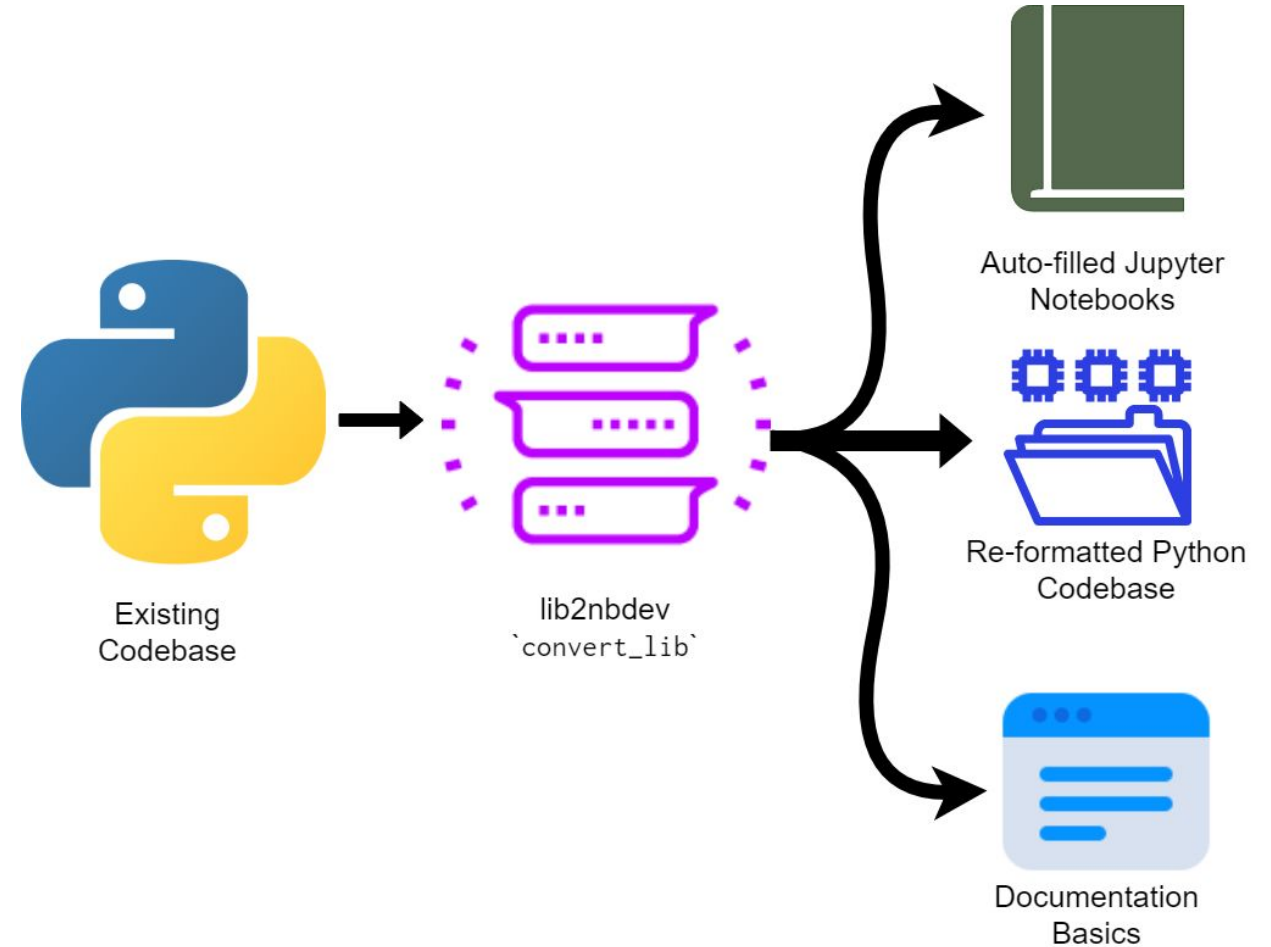


lib2nbdev

A single-use library for rapid conversion of “standard” python packages into the nbdev framework

I love that this much effort was put into something that's a "one-off". Converting the codebase might be a one-off task, but doing it by hand is a put-off. Without this library, I don't think I would've even considered it.

Now, minutes after starting, I'm ready to bring down the power of literate programming on my convoluted codebase.



documents and nbverbose

Typically in nbdev, we aim for “one-line documentation strings

```
def addition(  
    # The first number to add  
    a:int,  
    # The second number  
    # Which gets added to the first number  
    b:(int,float)  
) -> (int,float): # The sum of `a` and `b`  
    "Adds two numbers together"  
    return a + b
```

nbverbose

addition

[\[source\]](#)

```
addition(a:int, b:(<class 'int'>, <class 'float'>)=2)
```

Adds two numbers together

Parameters:

- **a** : <class 'int'>
The first number to be added
- **b** : (<class 'int'>, <class 'float'>), optional
The second number to be added

```
(int,float), The sum of `a` and `b`  
""  
return a+b
```


nbagile

Building upon everything prior, an
in-development
to the Agile

- Automate documentation
- “Number of patches making than a”
- Support sites

```
def addition(a,b) -> (int,float):  
    """Adds two numbers together  
  
    Parameters  
    -----  
    a : int  
        The first number to add  
    b : (int, float)  
        The second number to add  
  
    Returns  
    -----  
    (int, float)  
        The sum of a and b  
    """  
    return a+b
```

dev

r

tion

```
def addition(  
    # The first number to add  
    a:int,  
    # The second number  
    # Which gets added to the first number  
    b:(int,float)  
    ) -> (int,float): # The sum of `a` and `b`  
    "Adds two numbers together"  
    return a + b
```

nbdev + PyTest

Integration solutions can be simpler than these complex wrappers

A small patch to nbdev which allows for good library reporting

- Coverage testing
- Cell-by-Cell Failures.
- Extraction of tests into a “tests” folder

What is the best setup for success?

- Have teams start with nbdev, it is a strong foundation regardless of any additions
 - Through either starting from scratch from a [template](#)
 - Or converting an existing library with [lib2nbdev](#)
- Integrate documents if standard documentation styles and nbdev become too cumbersome
- If the coding fashion of documents is too hard to navigate, begin looking into nbagile as it is being developed



How do we use nbdev?

AdaptNLP and nbdev

- An Open-Source library allowing for easy access to state-of-the-art models and practices in the NLP domain, such as Named Entity Recognition and Sentiment Analysis
- When I took over the project, I wanted to use nbdev
- After hours of frustration, I wrote lib2nbdev to convert the library for me
- Afterwards it was quick house-keeping with the documentation, and the library was cleaned up

Improvements we saw:

- New users and coworkers could quickly utilize this new library and contribute to it, as it was easy to learn and follow the “flow” of the library
- Proper documentation was at the forefront, rather than an afterthought sparking numerous tutorials to be written
- Teams were hesitant to integrate it, but after learning the process they found it extremely helpful



What is nbdev's future? Does it have a future?

nbdev and the future

With the rise of live collaborative notebook viewing, teams can work on nbdev-built frameworks simultaneously without tripping over each other and have live feedback

Notebooks will become a means to an end to get thoughts onto paper, and then extracting these out into a more standard python-package setup.

We're already seeing trickles of this application too, though mostly manual

Testing will be more advanced, such as functions and applications requiring multiprocessing (which is hard to accomplish in Jupyter)

A clever reporting scheme to give better feedback on library performance



A Demo

Questions?