WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

https://xkcd.com/208/

CS 2731
Introduction to Natural Language Processing

Session 2: Words and tokens

Michael Miller Yoder

August 27, 2025

University of Pittsburgh | School of Computing and Information

Overview: Text normalization

- Course logistics
- JupyterHub CRCD setup
- Words and corpora
- Morphemes
- Unicode
- Regular expressions
- Other text preprocessing
- Coding activity: preprocessing Airbnb listings

Course logistics

- Reading for today was Jurafsky & Martin sections 2-2.4, 2.6-2.7, 2.10
- I will release Homework 0 today unless we all get set up in class with CRCD JupyterHub fine
- Please remind me of your name before asking or answering a question (just this class session)

CRCD JupyterHub setup

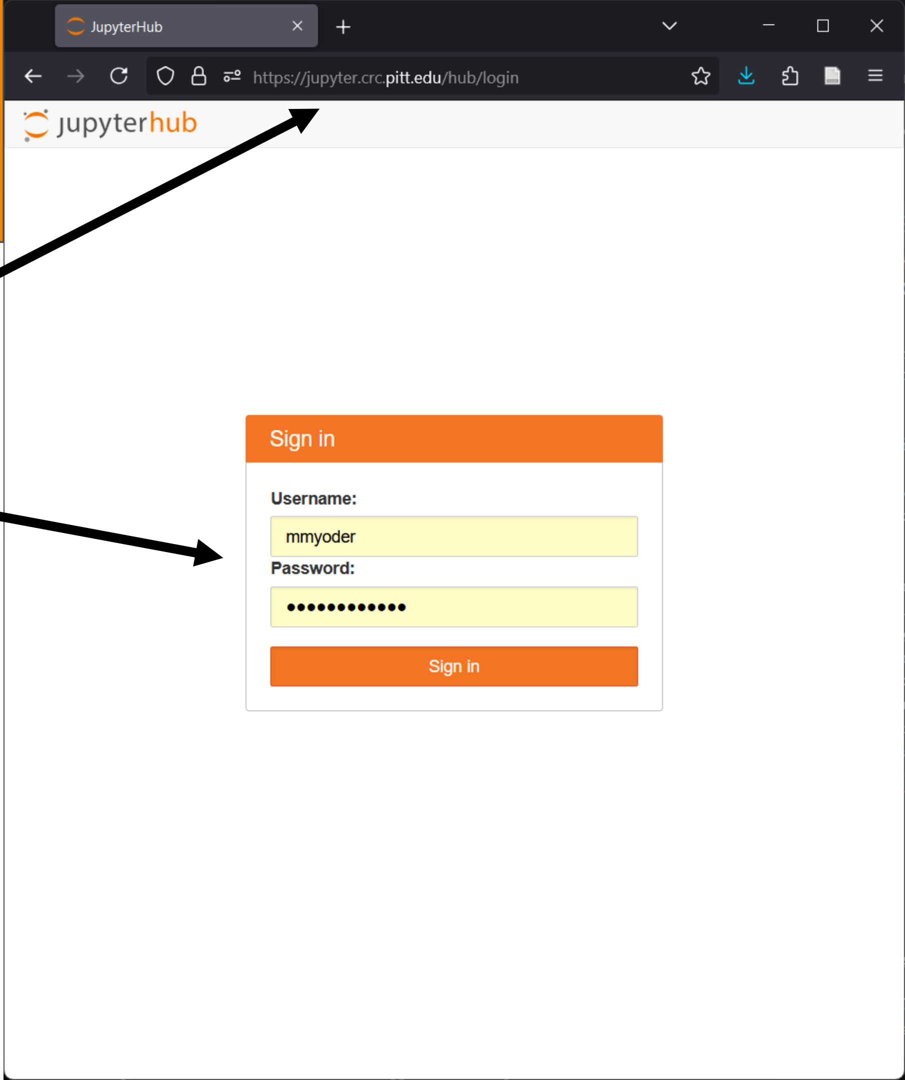
CRCD and JupyterHub

- CRCD (Center for Research Computing and Data) is a Pitt center providing computing services on various clusters
- They maintain a JupyterHub where people can run Jupyter Notebooks on their servers
- What we will be using the CRCD for:
    - Working through code examples in class
    - Writing code to submit as part of homework assignments
    - Running code and storing data for your projects (if you want to)

Logging in to your CRCD JupyterHub account

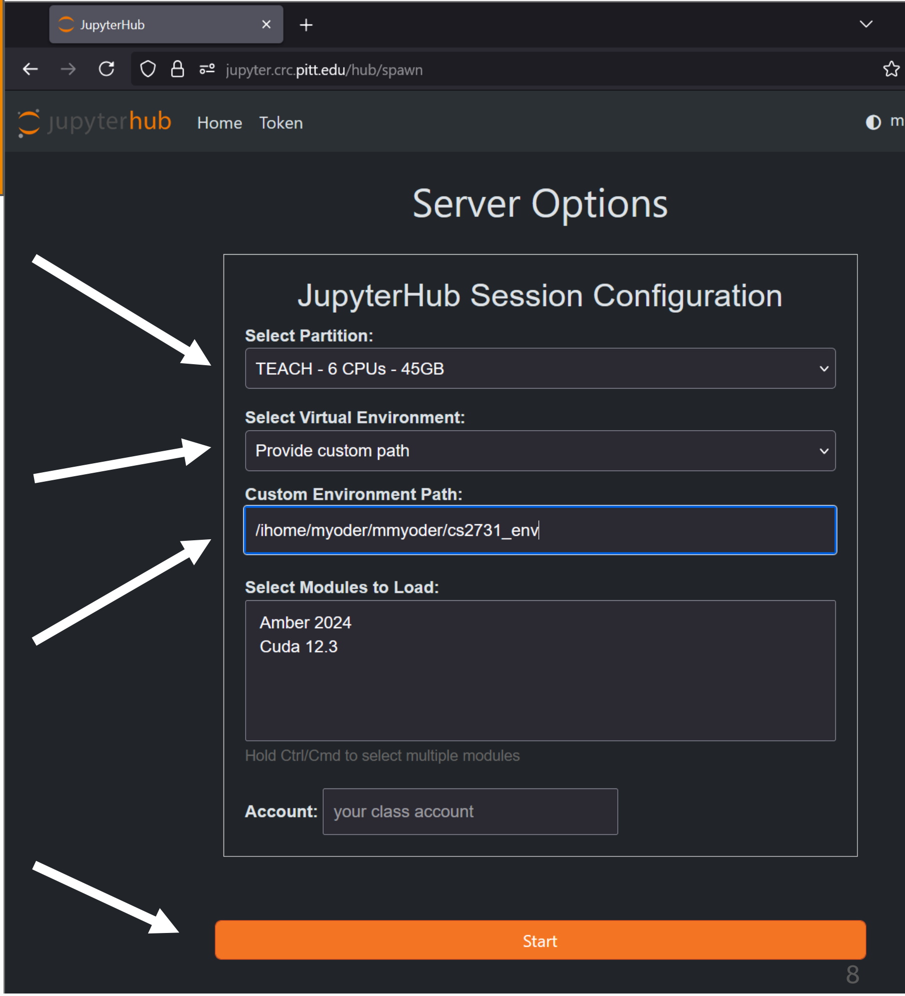
1. Go to **jupyter.crc.pitt.edu** in a web browser
2. Log in with your Pitt credentials

Note that if you are off-campus, you have to log in to the Pitt VPN first through the GlobalProtect app. Instructions: https://services.pitt.edu/TDClient/33/Portal/KB/ArticleDet?ID=293



Starting a Jupyter Notebook on the CRCD JupyterHub

1. Partition: **TEACH – 6 CPUs – 45 GB**
   We might use the GPU options later on in the course
2. Under **Select Virtual Environment**, select **Provide custom path**
3. **Custom Environment Path: /ihome/myoder/mmyoder/cs2731_env**
4. Click **Start**
5. Wait for the server to start up



Server Options

JupyterHub Session Configuration

Select Partition: TEACH - 6 CPUs - 45GB

Select Virtual Environment: Provide custom path

Custom Environment Path: /ihome/myoder/mmyoder/cs2731_env

Select Modules to Load:
Amber 2024
Cuda 12.3

Hold Ctrl/Cmd to select multiple modules

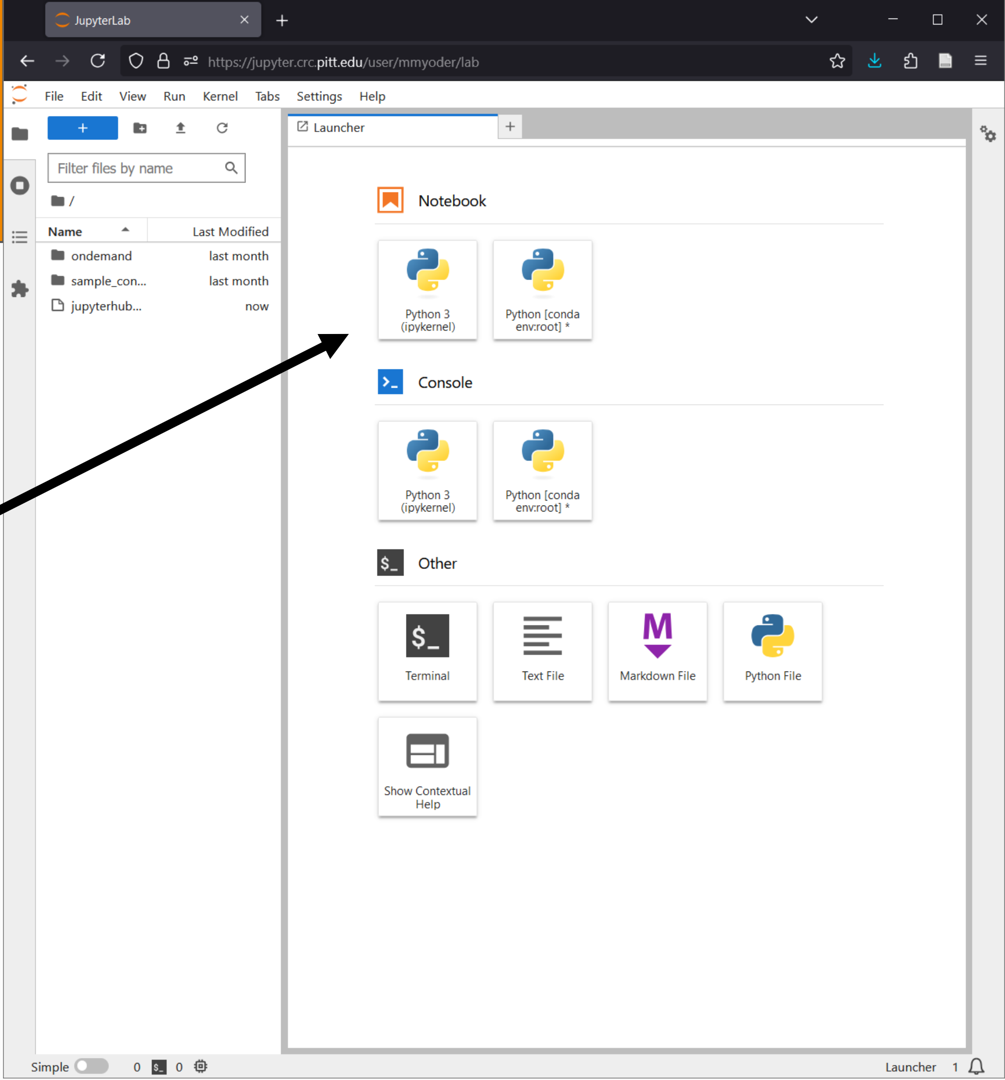
Account: your class account

Start

Welcome to your JupyterLab

Files are here

You can launch a new Jupyter Notebook by clicking Python 3 (ipykernel) under Notebook



Words and corpora

How many words in this phrase?

they lay back on the San Francisco grass and looked at the stars and their

- How many?
  - 15 tokens (or 14 if you count "San Francisco" as one)
  - 13 types (or 12) (or 11?)
- **Type**: a unique word in the vocabulary
- **Instance (token)**: an instance of a word type in running text
- **Lemma**: same stem, part of speech, rough word sense
  - cat and cats = same lemma
- **Wordform**: the full inflected surface form
  - cat and cats = different wordforms

Slide adapted from Jurafsky & Martin

How many words in a corpus?

Corpus: a (machine-readable) collection of texts

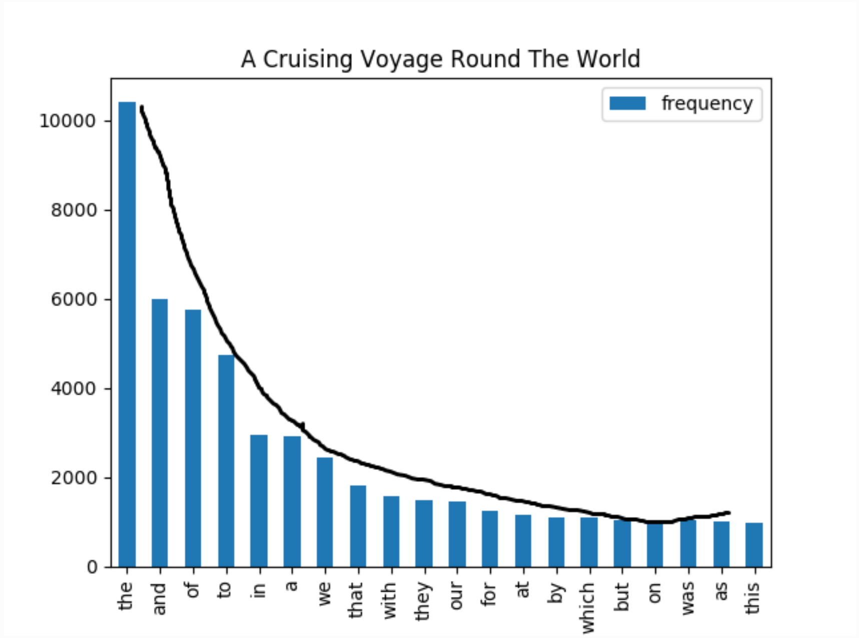
N = number of word instances

V = vocabulary = set of types, |*V*| is size of vocabulary

	Instances = N	Types = |V|
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million

Slide adapted from Jurafsky & Martin

Word frequencies: Heap's Law



The Lexical Learner blog

- Word (type) frequency is inversely proportional to word frequency rank

$$\textbf{frequency} \propto \frac{1}{(\textbf{rank} + b)^a}$$

- "Long tail" of infrequent words
- Similar to Zipf's Law

Corpora vary along dimensions like

- Texts don't appear out of nowhere!
- **Language**: 7097 languages in the world
- **Variety**, like African American Language varieties.
  - AAE Twitter posts might include forms like "*iont*" *(I don't)*
- **Code switching**, e.g., Spanish/English, Hindi/English:

  Por primera vez veo a @username actually being helpful! It was beautiful:)

  [For the first time I get to see @username actually being helpful! it was beautiful:)]

  dost tha or ra- hega ... dont wory ... but dherya rakhe

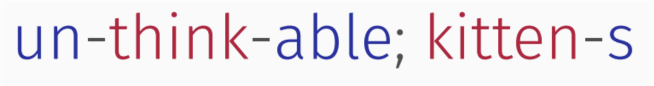
  [“he was and will remain a friend ... don’t worry ... but have faith”]

- **Genre:** newswire, fiction, scientific articles, Wikipedia
- **Author Demographics**: writer's age, gender, ethnicity, SES
- Corpus datasheets [Bender & Friedman 2018, Gebru+ 2020] ask about this information

Morphemes

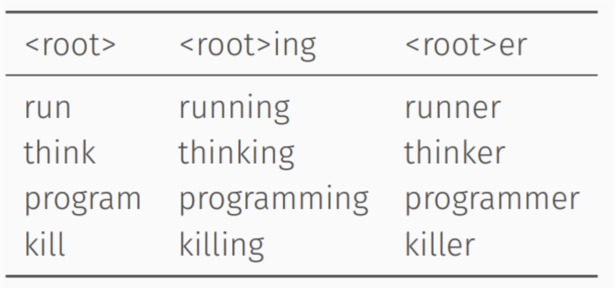
Morphemes

- Morphemes: small meaningful units that make up words
  - Roots: The core meaning-bearing units
  - Affixes: Parts that adhere to roots



un-think-able; kitten-s

- Affixes can add grammatical meaning (inflections, 2nd column) or modify semantic meaning (derivations, 3rd column)



<root>	<root>ing	<root>er
run	running	runner
think	thinking	thinker
program	programming	programmer
kill	killing	killer

Slide adapted from Jurafsky & Martin, David Mortensen

Dealing with complex morphology is necessary for many languages

- e.g., the Turkish word:
  Uygarlastiramadiklarimizdanmissinizcasina
  '(behaving) as if you are among those whom we could not civilize'

  Uygar 'civilized' + las 'become'
  + tir 'cause' + ama 'not able'
  + dik 'past' + lar 'plural'
  + imiz '1pl' + dan 'abl'
  + mis 'past' + siniz '2pl' + casina 'as if'

Slide adapted from Jurafsky & Martin

Unicode

Unicode

a method for representing written text in

- any character (more than 150,000!)
- any script (168 to date!)
- of the languages of the world
  - Chinese, Arabic, Hindi, Cherokee, Ethiopic, Khmer, N'Ko,...
  - dead ones like Sumerian cuneiform
  - invented ones like Klingon
  - plus emojis, currency symbols, etc.

Slide adapted from Jurafsky & Martin

ASCII: Some history for English

1960s American Standard Code for Information Exchange

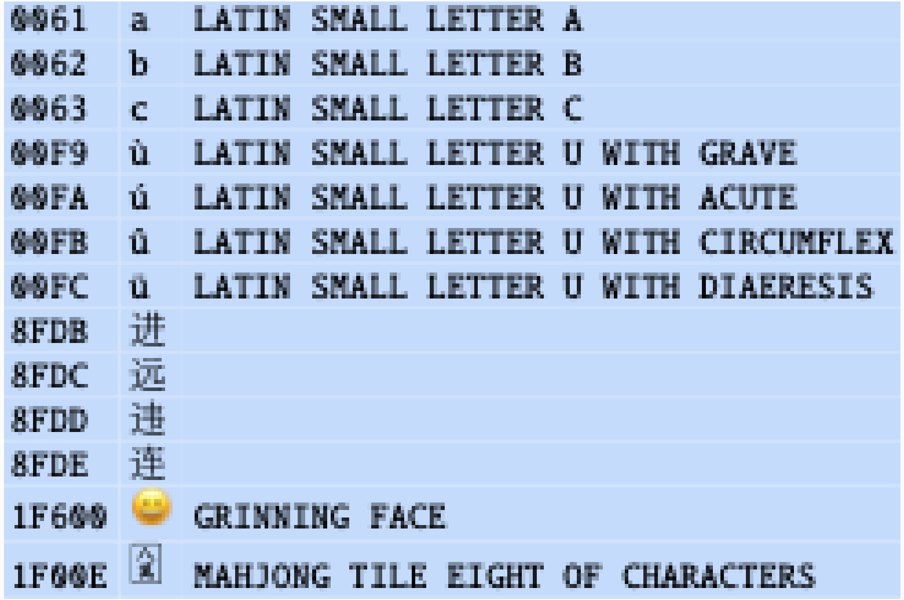
- 1 byte per character
- Set of letters without diacritical marks (such as accent marks, etc)
- Encodings for special characters used by teletypes, too

Slide adapted from Jurafsky & Martin

Code Points

- Unicode assigns a unique ID, a **code point**, to each of its 150,000 characters
- 1.1 million possible code points
  - 0 – 0x10FFFF
- Written in hex, with prefix "U+"
  - a is U+0061 which = 0x0061

Some code points



0061	a	LATIN SMALL LETTER A
0062	b	LATIN SMALL LETTER B
0063	c	LATIN SMALL LETTER C
00F9	ù	LATIN SMALL LETTER U WITH GRAVE
00FA	ú	LATIN SMALL LETTER U WITH ACUTE
00FB	û	LATIN SMALL LETTER U WITH CIRCUMFLEX
00FC	ü	LATIN SMALL LETTER U WITH DIAERESIS
8FDB	进	
8FDC	远	
8FDD	违	
8FDE	连	
1F600	😀	GRINNING FACE
1F00E	🀎	MAHJONG TILE EIGHT OF CHARACTERS

A code point has no visuals; it is **not** a glyph!
Glyphs are stored in **fonts**: **a** *a* a a
But all of them are U+0061, abstract "LATIN SMALL A"

Slide adapted from Jurafsky & Martin

Encodings and UTF-8

- We don't stick code points directly in files
- We store **encodings** of characters
- The most popular encoding is UTF-8
- Most of the web is stored in UTF-8

Slide adapted from Jurafsky & Martin

Variable Length Encoding

- **UTF-8** (Unicode Transformation Format 8)
- UTF-8 encoding of `hello` is :
  - 68 65 6C 6C 6F
- Code points ≥128 are encoded as a sequence of 2, 3, or 4 bytes
  - First few bits say if its 2-byte, 3-byte, or 4-byte

Tokenization

Why tokenize?

- Using a deterministic series of tokens means systems can be compared equally
  - Systems agree on the length of a string
- Eliminates the problem of unknown words

Slide adapted from Jurafsky & Martin

Space-based tokenization

- A very simple way to tokenize
- For languages that use space characters between words
    - Arabic, Cyrillic, Greek, Latin, etc., based writing systems
- Segment off a token between instances of spaces

Slide adapted from Jurafsky & Martin

Issues in Tokenization

- Can't just blindly remove punctuation:
  - m.p.h., Ph.D., AT&T, cap’n
  - prices ($45.55)
  - dates (01/02/06)
  - URLs (http://www.pitt.edu)
  - hashtags (#nlproc)
  - email addresses (someone@cs.colorado.edu)
- Clitic: a word that doesn't stand on its own
  - "are" in we're, French "je" in j'ai, "le" in l'honneur
- When should multiword expressions (MWE) be words?
  - New York, rock ’n’ roll

Slide adapted from Jurafsky & Martin

Tokenization in languages without spaces between words

- Many languages (like Chinese, Japanese, Thai) don't use spaces to separate words!
- How do we decide where the token boundaries should be?

Slide adapted from Jurafsky & Martin

Word tokenization in Chinese

- Chinese words are composed of characters called "hanzi" (or sometimes just "zi")
- Each one represents a meaning unit called a morpheme
- Each word has on average 2.4 of them.
- But deciding what counts as a word is complex and not agreed upon.

How to do word tokenization in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?

姚明	进入	总决赛
YaoMing	reaches	finals

5 words?

姚	明	进入	总	决赛
Yao	Ming	reaches	overall	finals

7 characters? (don't use words at all):

姚	明	进	入	总	决	赛
Yao	Ming	enter	enter	overall	decision	game

Slide adapted from Jurafsky & Martin

Word tokenization / segmentation

- In Chinese NLP it's common to just treat each character (zi) as a token.
  - So the **segmentation** step is very simple
- In other languages (like Thai and Japanese), more complex word segmentation is required.
  - The standard algorithms are neural sequence models trained by supervised machine learning.

Slide adapted from Jurafsky & Martin

Subword tokenization & BPE

Another option for text tokenization

- **Use the data** to tell us how to tokenize.
- **Subword tokenization** (because tokens can be parts of words as well as whole words)
- Many modern neural NLP systems (like BERT) use this to handle unknown words
- 2 parts:
  - A token learner that takes a raw training corpus and induces a vocabulary (a set of tokens).
  - A token segmenter that takes a raw test sentence and tokenizes it according to that vocabulary

Slide adapted from Jurafsky & Martin

Byte Pair Encoding [BPE, Sennrich+ 2016] token learner

Let vocabulary be the set of all individual characters

= {A, B, C, D,…, a, b, c, d….}

Repeat:

- Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
- Add a new merged symbol 'AB' to the vocabulary
- Replace every adjacent 'A' 'B' in the corpus with 'AB'.

Until *k* merges have been done.

Byte Pair Encoding (BPE) token learner

Iteratively merge frequent neighboring tokens to create longer tokens.

Start with all characters

Repeat:

- Choose most frequent neighboring pair ('A', 'B')
- Add a new merged symbol ('AB') to the vocabulary
- Replace every 'A' 'B' in the corpus with 'AB'.

Until *k* merges

Vocabulary

[A, B, C, D, E]

[A, B, C, D, E, AB]

[A, B, C, D, E, AB, CAB]

Corpus

A B D C A B E C A B

AB D C AB E C AB

AB D CAB E CAB

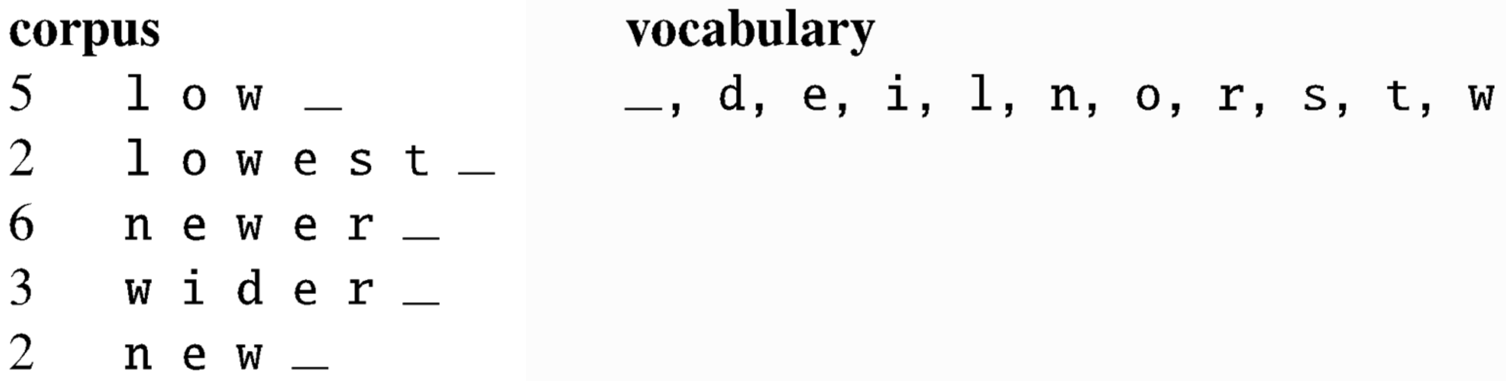
Slide adapted from Jurafsky & Martin

BPE token learner

Original (very fascinating 🙄) corpus:

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

Split on whitespace, add end-of-word tokens _



corpus	
5	l o w _
2	l o w e s t _
6	n e w e r _
3	w i d e r _
2	n e w _

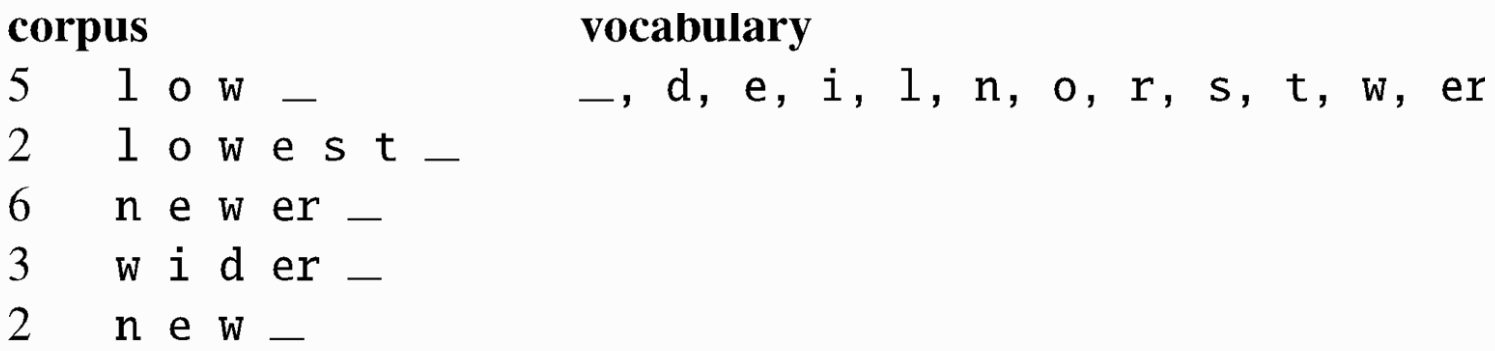
vocabulary

_, d, e, i, l, n, o, r, s, t, w

Slide adapted from Jurafsky & Martin

BPE token learner

- Merge e r to er



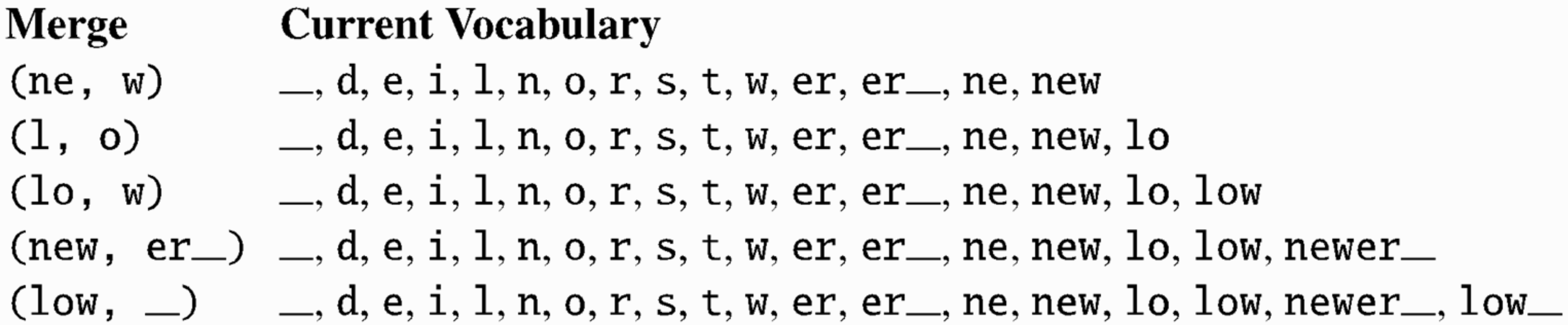
corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

- Merge er _ to er_
- Merge n e to ne

Slide adapted from Jurafsky & Martin

BPE token learner

The next merges are:



Merge	Current Vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

Slide adapted from Jurafsky & Martin

BPE token segmenter algorithm

- On the test data, run each merge learned from the training data:
  - Greedily, in the order we learned them
- So merge every e r to er, then merge er _ to er_, etc.
- Result:
  - Test set "n e w e r _" would be tokenized as a full word
  - Test set "l o w e r _" would be two tokens: "low er_"

Regular expressions (regex)

Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions: Disjunctions (OR)

- Letters inside square brackets []

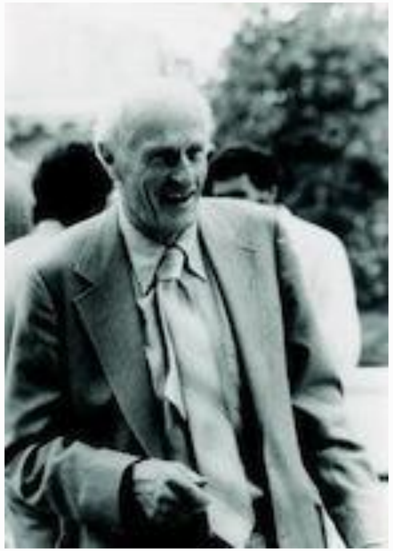
Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

- Ranges `[A-Z]` `[a-z]` `[0-9]`
- Negations `[^A-Z]`
  - Carat means negation only when first in []
- Sequence disjunctions with pipe |
  - `groundhog|woodchuck`

Slide adapted from Jurafsky & Martin

Regular Expressions wildcards: *+.

Pattern	Matches	
`oo*h`	0 or more of previous char	oh ooh oooh ooooh
`o+h`	1 or more of previous char	oh ooh oooh ooooh
`beg.n`	Any char	begin begun begun beg3n


Stephen C Kleene

Slide adapted from Jurafsky & Martin

Regular expression example

- Find all instances of the word “the” in a text.

```
the
```

- Misses capitalized examples

```
[tT]he
```

- Incorrectly returns "other" or "theology"

```
[^a-zA-Z][tT]he[^a-zA-Z]
```

Slide adapted from Jurafsky & Martin

Errors

The process we just went through was based on fixing two kinds of errors:

1. Matching strings that we should not have matched (there, then, other)

   False positives (Type I errors)

2. Not matching things that we should have matched (The)

   False negatives (Type II errors)

Slide adapted from Jurafsky & Martin

Simple Application: ELIZA

- Early NLP system that imitated a Rogerian psychotherapist [Weizenbaum 1966]
- Uses pattern matching to match phrases

  `“I need X”`

- and translates them into, e.g.

  `“What would it mean to you if you got X?`

Simple Application: ELIZA

Men are all alike.
IN WHAT WAY

They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED

Slide adapted from Jurafsky & Martin

How ELIZA works

```
.* I'M (depressed|sad) .* → I AM SORRY TO HEAR YOU ARE \1
.* all .* → IN WHAT WAY?
.* always .* → CAN YOU THINK OF A SPECIFIC EXAMPLE?/
```

Other text preprocessing (normalization)

Case folding (lowercasing)

- Applications like IR: reduce all letters to lowercase
  - Since users tend to use lowercase
  - Possible exception: upper case in mid-sentence?
    - e.g., *General Motors*
    - *Fed* vs. *fed*
    - *SAIL* vs. *sail*
- For sentiment analysis, MT, information extraction
  - Case is helpful (*US* versus *us* is important)

Lemmatization

Represent words as their **lemma**: their shared root, dictionary headword form:

- *am, are, is* → *be*
- *car, cars, car's, cars'* → *car*
- Spanish quiero (‘I want’), quieres (‘you want’)
  → querer ‘want'
- *He is reading detective stories*
  → *He be read detective story*

Stemming

- Reduce terms to stems, chopping off affixes crudely

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with

Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with

$$
\begin{aligned}
\text{ATIONAL} &\rightarrow \text{ATE} \quad \text{(e.g., relational} \rightarrow \text{relate)} \\
\text{ING} &\rightarrow \epsilon \quad \text{if stem contains vowel (e.g., motoring} \rightarrow \text{motor)} \\
\text{SSES} &\rightarrow \text{SS} \quad \text{(e.g., grasses} \rightarrow \text{grass)}
\end{aligned}
$$

Slide adapted from Jurafsky & Martin

Stopword removal

- Do we want to keep "function words" like *the*, *of*, *and*, *I*, *you*, etc?
- Sometimes no (information retrieval)
- Sometimes yes (authorship attribution)

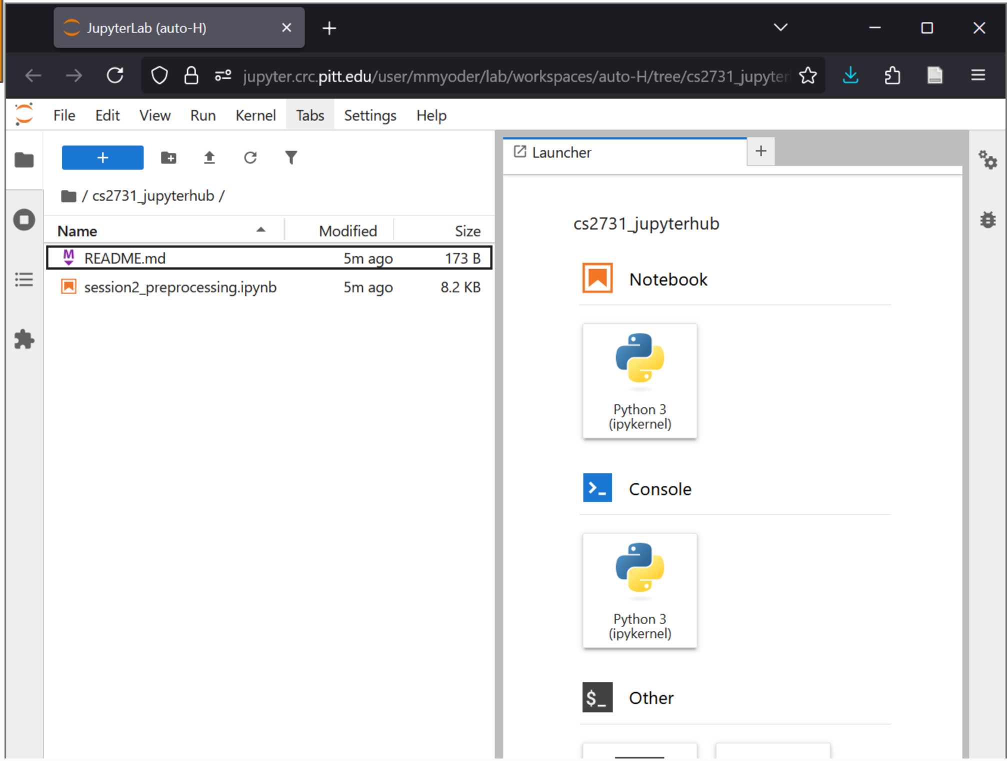
Conclusion: Text normalization

- Regular expressions match flexible sequences of characters and allow substitution of groups of characters
- Tokenization: splitting texts into sequences of words
    - Subword tokenization finds tokens based on frequencies of sequences of characters in data
- Lemmatization: normalizing words to their dictionary roots
- Stemming: chopping off affixes of words to reduce them to stems
- Stopwords are function words like “the”, “a”, “and”, “of”, etc that are often ignored in NLP applications

Coding activity: Preprocessing Airbnb listings

Load in-class notebooks

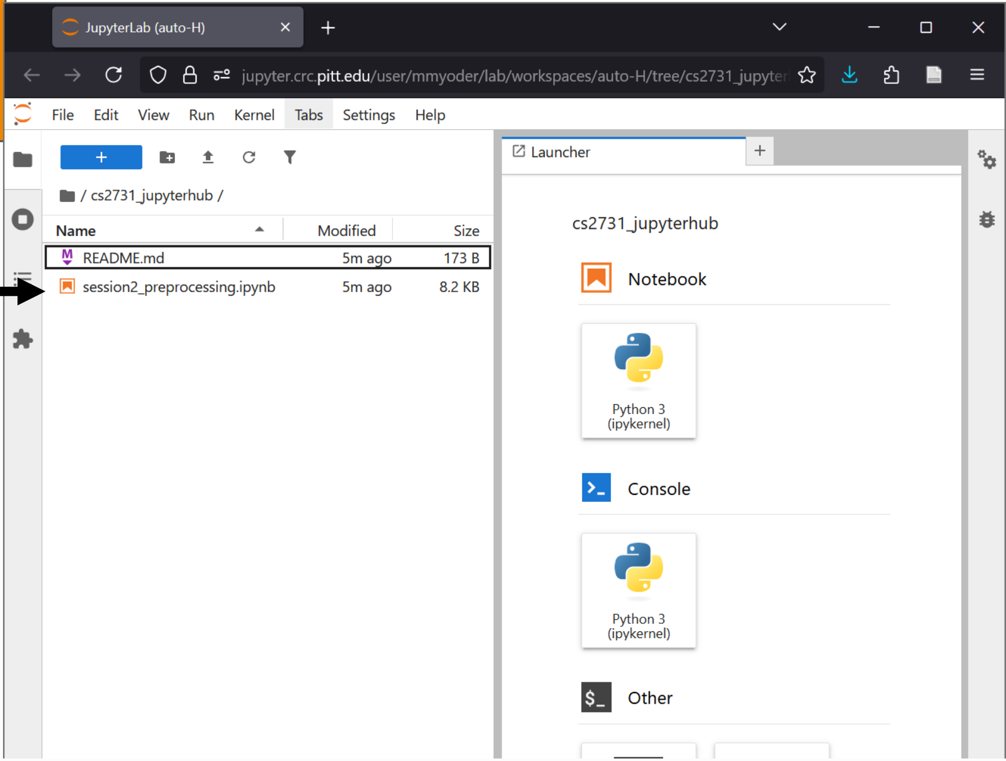
1. Go to this nbgitpuller link (also available on course website)
2. Log in with your Pitt username if necessary
3. Start a server with **TEACH – 6 CPUs, 48 GB**
4. Load custom environment at /ihome/myoder/mmyoder/cs2731_env
5. This should pull a folder (cs2731_jupyterhub) into your JupyterLab



Name	Modified	Size
README.md	5m ago	173 B
session2_preprocessing.ipynb	5m ago	8.2 KB

Open Jupyter notebook

1. Double-click **session2_preprocessing .ipynb** on the left panel to open the notebook



/ cs2731_jupyterhub /

Name	Modified	Size
README.md	5m ago	173 B
session2_preprocessing.ipynb	5m ago	8.2 KB

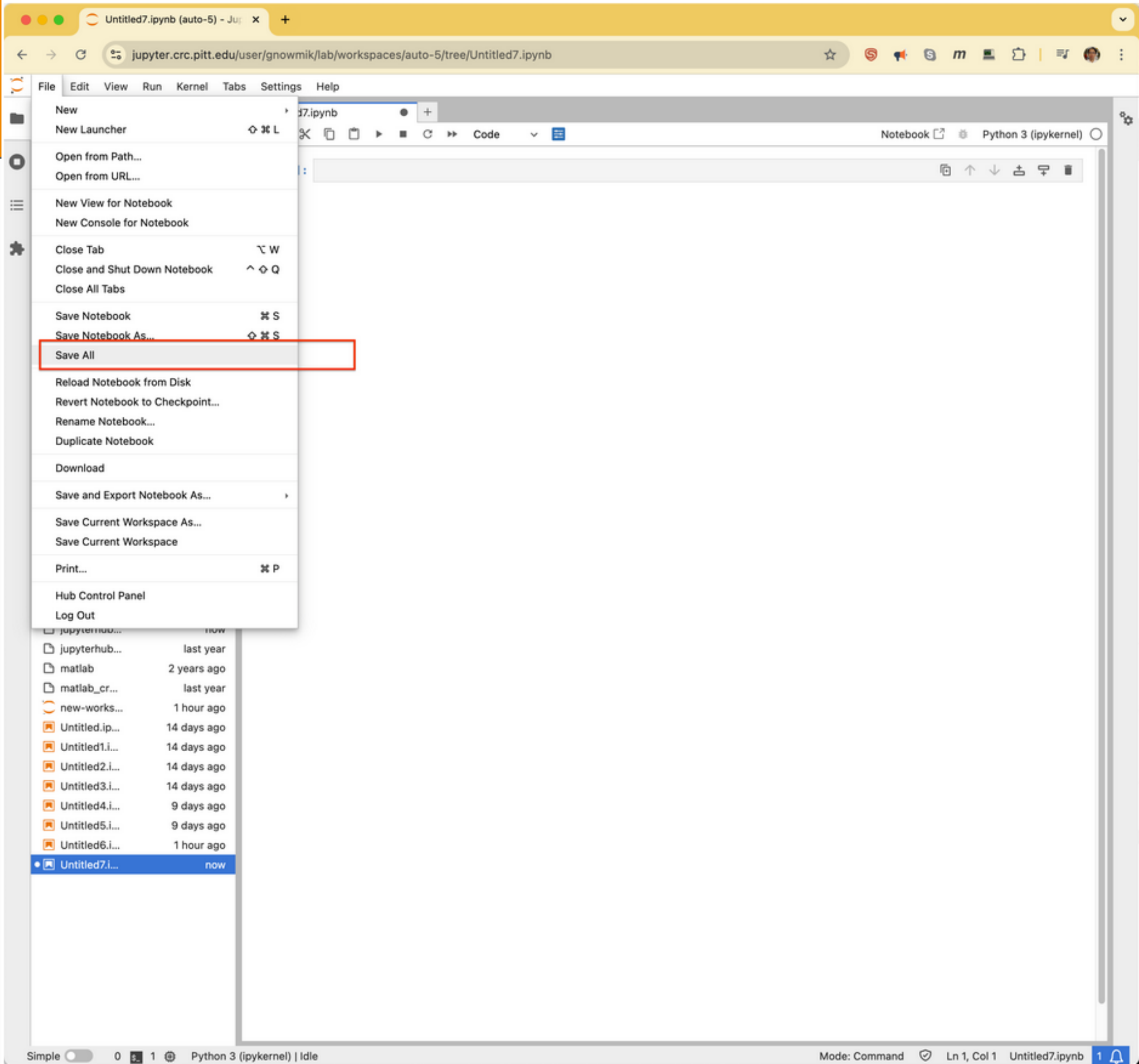
Jupyter Notebook basics

- Each block is called a “cell”
  - Has input and possibly output
  - Input can be Python code, Markdown or shell commands (after `!`)
- Modes
  - Command mode
    - Move, select, manipulate cells
    - Get into command mode by clicking anywhere outside of a cell
  - Edit mode
    - Edit content of a particular cell
- Running cells
  - Click “Run” button or do Ctrl+Enter (on Windows or Linux, Cmd+Enter on Mac) to run code or render Markdown
  - Any result will be shown in the output of the cell

Implementation

- Remove undesired text with regular expressions
- Lowercase
- Remove stopwords
- Tokenize with the NLTK package
- Stem the tokens with NLTK

Saving your work

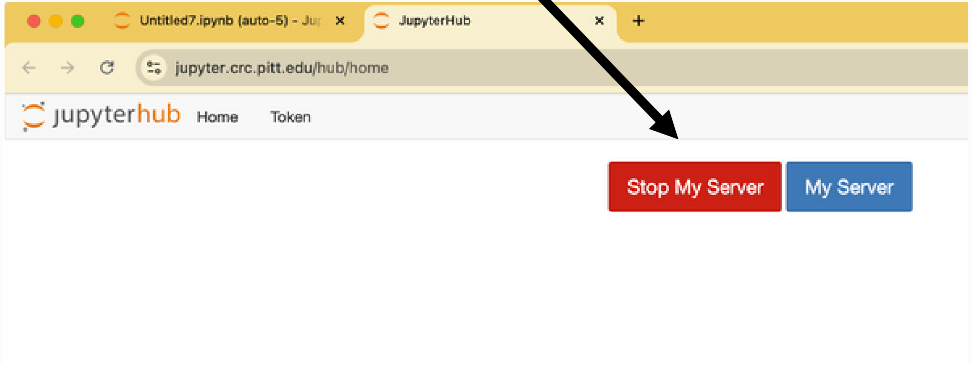


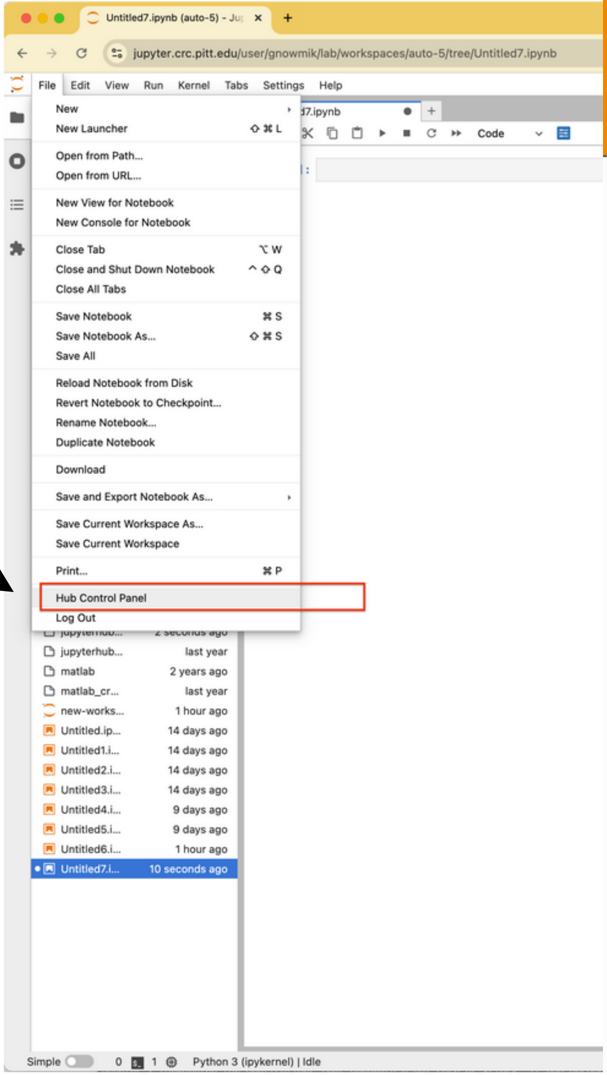
jupyter.crc.pitt.edu/user/gnowmik/lab/workspaces/auto-5/tree/Untitled7.ipynb

Ending your session

Be sure to save your work before ending the session

1. Select **File** > **Hub Control Panel**
2. Click **Stop My Server**





Untitled7.ipynb (auto-5) - Jup
jupyter.crc.pitt.edu/user/gnowmik/lab/workspaces/auto-5/tree/Untitled7.ipynb
File
Edit
View
Run
Kernel
Tabs
Settings
Help
New
New Launcher
Open from Path...
Open from URL...
New View for Notebook
New Console for Notebook
Close Tab
Close and Shut Down Notebook
Close All Tabs
Save Notebook
Save Notebook As...
Save All
Reload Notebook from Disk
Revert Notebook to Checkpoint...
Rename Notebook...
Duplicate Notebook
Download
Save and Export Notebook As...
Save Current Workspace As...
Save Current Workspace
Print...
Hub Control Panel
Log Out
jupyterhub... last year
matlab 2 years ago
matlab_cr... last year
new-works... 1 hour ago
Untitled.ip... 14 days ago
Untitled1.i... 14 days ago
Untitled2.i... 14 days ago
Untitled3.i... 14 days ago
Untitled4.i... 9 days ago
Untitled5.i... 9 days ago
Untitled6.i... 1 hour ago
Untitled7.i... 10 seconds ago
Code
Simple
Python 3 (ipykernel) | Idle

Questions?

Enjoy Labor Day holiday

No class on Monday