course setup & programming basics

LING 430 Computational Linguistics Fall 2026
lecture 1b
Aug 27 2026

Siyu Liang
siyu.liang@rice.edu

agenda

- programming basics
- coding & LLM

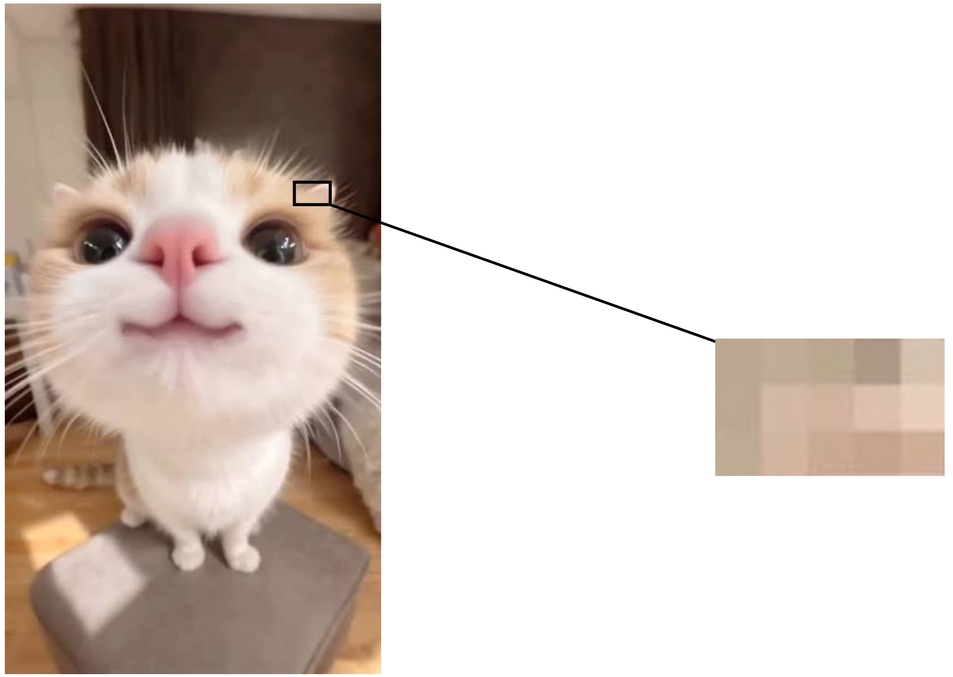
programming basics

operating system

- operating systems are **softwares that support basic functions**
  - e.g., how does at program get executed at the hardware level?
  - memory allocation, file locations, I/O
- no program can run without an OS

operating system

- understanding an OS requires understanding **low-level** software
  - less abstraction over hardware
- we will be programming with **high-level** languages
  - more abstraction over hardware
- why learn about system?
  - projects are pipelines that cross both
  - so you know things can be different



high level

low level

operating system

- older
  - Unix (AT&T's Bell labs)
  - MS-DOS (Microsoft)
- newer
  - Linux (open source, Linux community)
    - Unix-like
  - macOS (Apple)
    - Unix-based
  - Windows (Microsoft)
    - DOS-based
  - cloud (someone else's computer)
    - usually Unix-like

operating system

practical differences

- **command-line language** is different
- different **text file format**
- different **file path separator** (uses “/” instead of “\”)
  - technical note: “/” is forward-slash or slash, “\” is backward-slash or backslash
- different **applications** available

operating system

practical differences for us?

- not much, really
- when I demo something, it's mostly in a *nix system
  - you will be able to replicate demos on your system
  - although sometimes you will need to Google "How do I do X on Windows/Mac/Linux/…?"

programming language

- we will be using **Python**
- other languages you may have heard of: C/C++, C#, Java, JavaScript, Ruby, Haskell, Go, R, MATLAB, Julia, Objective-C, COBOL, Fortran…
- a particular programming language can be **more or less suited** for a task
  - **not true** of human/natural languages!
- some languages are more **popular** than others, regardless of how **well suited** they are for a given task

natural language vs formal language

- a **natural language** evolved
  - English, Thai, ASL, etc.
  - nobody designed it
- a **formal language** is designed
  - Python, arithmetic, chemical formulas
- interesting situations
  - conlangs: constructed languages, e.g. Klingon, Galach, etc.
  - some cases of state language policy, e.g. Modern Hebrew, Turkish, etc.
  - language regulator

Galach in Dune

Python basics

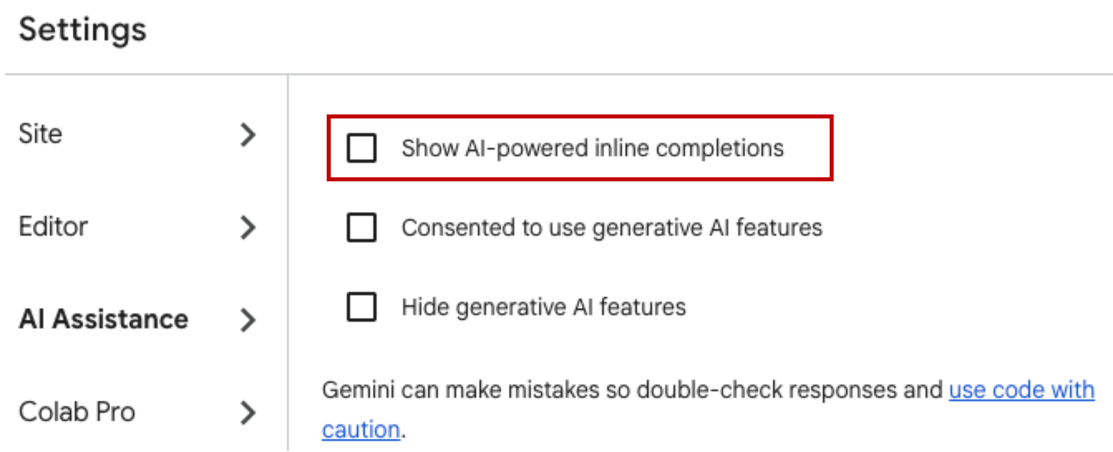
Python notebooks

- course website → scroll down → activity
- opens in your browser as a Google Colab file
  - easy and straightforward for us now
- we move to VSCode later
  - better practice and future work

Python notebooks

nota bene

- **before** typing in the Colab document
  - File → Save a copy in Drive
- the notebook opens read-only, straight from a public repository
- your copy goes to your Drive and stays yours
- no copy, no saved work!
- uncheck AI-powered inline completions if prompted



Python notebooks

- a **cell** holds **code**
- ▶️ or **shift** + **enter** runs a cell
- running it prints the result underneath
- you can run as many times as you like

variables

- sort of like a variable in algebra (and sort of not)
- using some sort of **alphanumeric name** to stand in for a **value**
- useful because typing out the full values EVERY time is tedious and liable to produce bugs
  - imagine typing out π to 10 digits every time you wanted to use it!
  - `pi = 3.1415926535`

assignment

- `word = "hola"`
- `word`
  - 'hola'
- name on the left, value on the right, = in between
- assignment is silent
  - nothing prints; type the name to see what's inside
- name it once, then use the name instead of the thing

assignment

= is not equals

- `count = 1`
- `count = count + 1  # fine here, nonsense in algebra`
- `count`
  - → 2
- `=` means "put this in that box"
  - putting the right-hand-side to the left-hand-side

operators

- `6 * 7`
  - → 42
- `84 / 2`
  - → 42.0
- `2 ** 10`
  - → 1024
- `count == 2`
  - → True

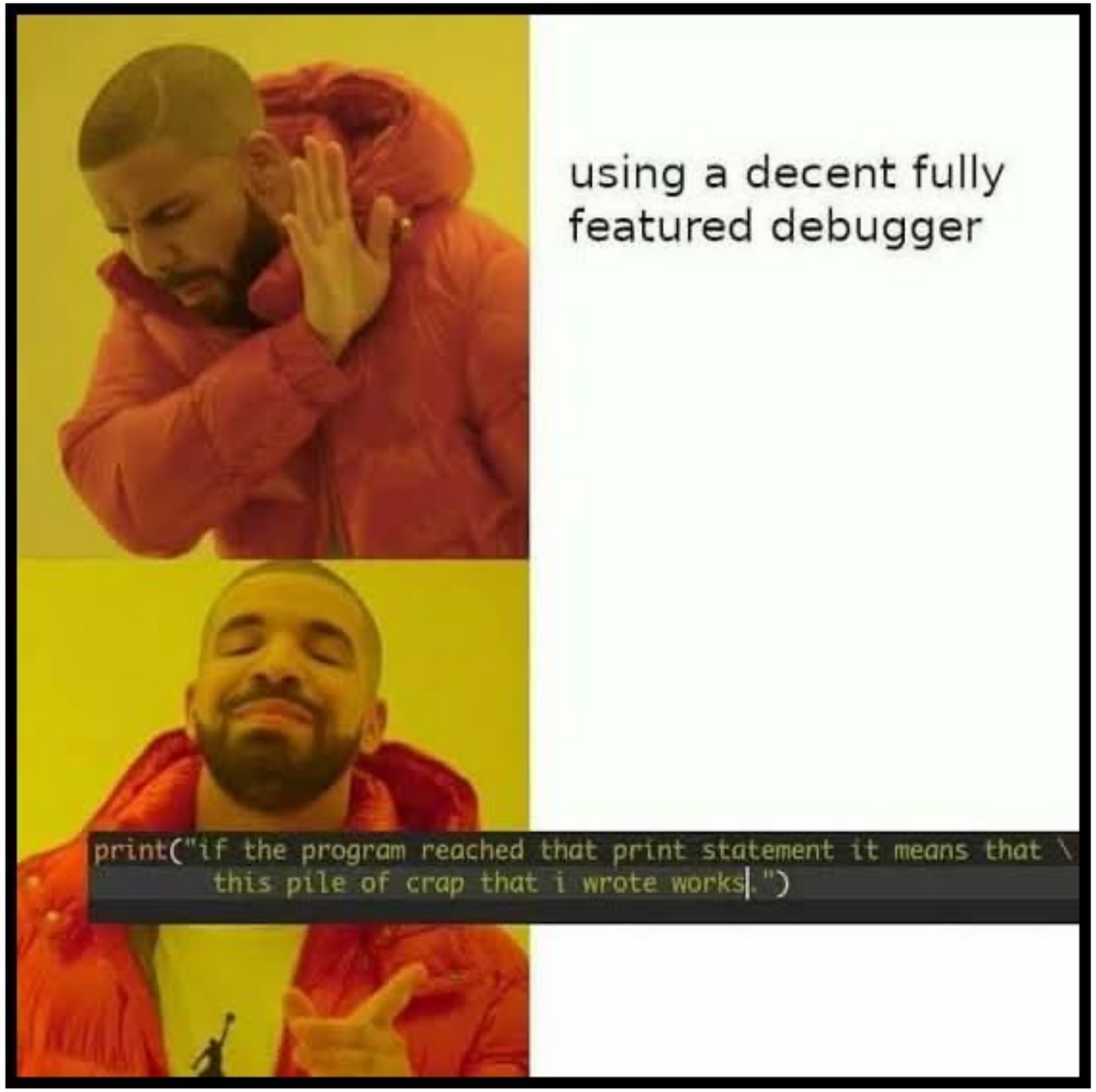
variable names

- rules
  - letters, digits, and underscore
  - cannot *start* with a digit/space/hyphen
- case-sensitive
  - `Word` and `word` are two different names
- Python's own keywords are off-limits
  - `class = "LING 430"` is a SyntaxError

- `2word = "hola”`
  - SyntaxError: invalid decimal literal
- `my word = "hola"`
  - SyntaxError: invalid syntax
- `word2 = "hola"  # fine`
- also note
  - `#` marks a comment that will be ignored during running

basic functions

- `print()`
  - display the **same** info inside the parenthesis
  - `print("hello, LING 430!")`
    - → hello, LING 430!



basic functions

- `len()`
  - return the **total number** of items in a collection, e.g., length of a text
  - `len("Odyssey")`
    - → 7

basic functions

- `len("Odyssey")`
  - **name**
    - what you want done: `len`
  - **parentheses**
    - () always, even when empty
  - **argument**
    - what you want it done to: `"Odyssey"`

basic functions

City with the longest name

- Longest place name
- `len("Krung Thep Maha Nakhon Amon Rattanakosin Mahintharayutthaya Mahadilok Phop Noppharat Ratchathani Burirom Udomratchaniwet Mahasathan Amon Phiman Awatan Sathit Sakkathattiya Witsanukam Prasit")`

basic functions

- `type()`
  - tells you **what kind of thing** something is
  - `type("Odyssey")`
    - → str (a string)
  - `type(7)`
    - → int (an integer)
- quotes matter!
  - `"7"` is text, `7` is a number

basic functions

everything has a type!

- `>>> "rose" * 3`
  - `'roseroserose’`
- `>>>"rose" + 3`
  - `TypeError`
- why?
  - "rose" is a string, in quotes
  - 3 is an integer, no quotes
  - some operations can work across types, some don’t

basic functions

gluing and repeating

- `"avada" + "kedavra"`
- → 'avadakedavra' concatenation
- `"ha" * 3`
  - → 'hahaha' repetition
- the same + adds numbers and glues text
  - the machine decides by looking at the types

basic functions

function vs method

- a **function**: something you do to a value
  - `len(word)`
  - `print(word)`
- a **method**: something the value does itself
  - `word.lower()`
  - `word.split()`
  - the dot means “ask this thing to do that to itself”

basic methods

- `.lower()` and .upper()
  - return the same text, with the case changed
  - `"Odyssey".lower()`
    - → 'odyssey'
  - `"Odyssey".upper()`
    - → 'ODYSSEY'

basic methods

- `.split()`
  - cut a text into a **list of pieces**, at the spaces
  - line = "Tell me, O Muse, of that ingenious hero"
  - `line.split()`
    - → ['Tell', 'me,', 'O', 'Muse,', 'of', 'that', 'ingenious', 'hero']
  - the opening of the *Odyssey*, Butler's translation

basic methods

- `['Tell', 'me,', 'O', 'Muse,', ...]`
  - `'me,'` with the comma attached is not a word
- `.split()` only knows about **spaces**, not words
- deciding what counts as a word is **tokenization**
  - we'll talk about it in week 4!

basic functions

keep one of each

- `set()`
  - throw away every repeated item
  - `set(words)`
    - → the **distinct** words
  - `len(set(words))`
    - → how many distinct words

tokens and types

- `words = "Rose is a rose is a rose".split()`
  - `len(words)` → **7**
    - tokens: every word, counted every time
  - `len(set(words))` → **4**
    - types: Rose, is, a, rose
- is **Rose** the same word as **rose**?
  - you'll have to decide
  - modern approaches usually say no

Python notebooks

order of cells

- `word = "hola"  # cell 1`
- `print(word)  # cell 2`
- run cell 2 first, and you get an error
  - `NameError: name 'word' is not defined`
- **order on the page ≠ order in memory**

Python notebooks

how to read an error

- `Traceback (most recent call last):`
- `  File "<stdin>", line 1, in <module>`
- `NameError: name 'word' is not defined`
- **the last line** is usually the actual complaint
  - before the colon: the kind of error
  - after it: the specifics

Python notebooks

the two errors you met today

- `NameError`
  - I have no such name in memory
- `TypeError`
  - these two kinds of thing do not combine
- an error is the machine telling you exactly what confused it

Python notebooks

general good practice

- before you trust a result, and always before you submit
  - **Runtime → Restart and run all**

checkpoint

- you opened the notebook and saved a copy to Drive
- you ran a cell and saw output
- you met a NameError and survived
- if any of these is missing, raise your hand or grab a neighbor now

coding & LLM

Ma et al. 2025 *Not Everyone Wins with LLMs*

- students did programming homework with an LLM assistant
- what predicted success
  - not familiarity with AI tools or how well they wrote prompts
  - **technical experience they already had**
- which means
  - these systems produce errors that look right
  - catching those needs the knowledge you are here to build
  - outsourcing the practice removes what makes the tool useful later

Parsons et al. 2024

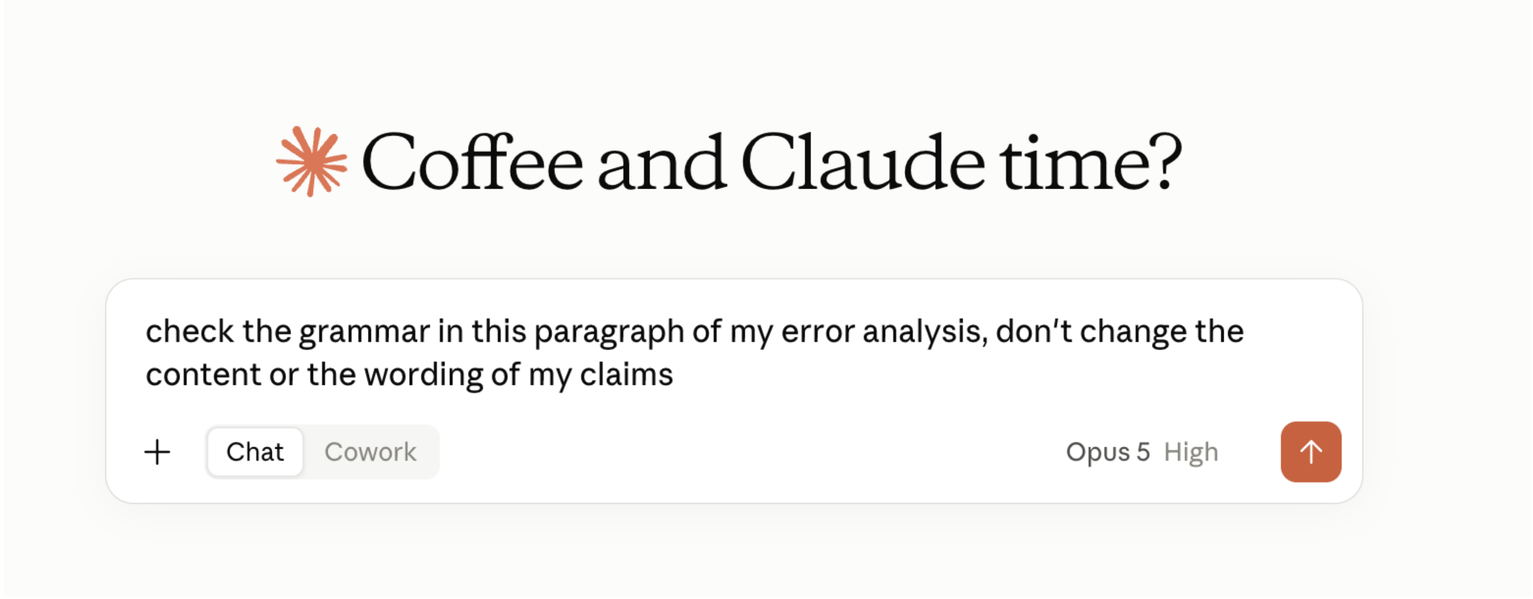
on the course website and Canvas

Parsons' AI Classroom Use Scale (2024)

Level	Level Name	Key Elements	Example Prompts
0	No AI Use	* No AI tools used for any aspect of written assignments. * All work is independently conceived and produced. * No AI assistance of any kind.	(No prompts used)
1	AI for Minor Grammar/Style Corrections	* Limited to basic grammar, spelling, and punctuation checks. * No alteration of content or meaning. * Primarily uses tools like Grammarly for surface-level edits.	"Check this paragraph for grammatical errors" "Improve sentence structure" "Suggest synonyms for [words]"
2	AI for Sentence-Level Refinement	* Focus on improving individual sentences. * No generation of complete paragraphs or sections. * AI suggestions are incorporated selectively by the student.	"Refine this sentence for impact" "Rephrase this sentence more academically" "Rewrite this sentence to improve clarity"
3	AI for Limited Idea Generation & Outlining	* AI used for brainstorming and initial ideas, but student develops content. * Creates a basic outline or structure, but student fills in details. * AI suggestions are a starting point, not the final product.	"Give me three thesis statements on [topic]" "Suggest three main points for [argument]" "Create a basic outline for an essay on [topic]"
4	AI for Feedback and Revision (Sentence Level)	* AI provides feedback on existing drafts. * Suggestions focus on sentence-level clarity, style, and flow. * Student retains control over content changes.	"Review this paragraph for clarity" "Suggest ways to improve flow" "Evaluate the argument and suggest improvements"
5	AI for Paraphrasing and Clarification	* AI used to simplify complex sentences or passages. * Maintains original expression; AI acts as a clarifying tool. * Student independently synthesizes information.	"Paraphrase this complex sentence" "Explain this passage in simpler terms" "Summarize the main arguments of this article"
6	AI for Structural Assistance (Outlining & Organization)	* AI helps organize thoughts and structure the argument. * AI supports logical flow and organization of ideas. * Content and ideas remain entirely the student's own.	"Suggest a logical order for these points" "Help organize this essay for smooth transitions" "Suggest a strong conclusion based on these points"
7	AI for Advanced Revision & Style Suggestions	* Extensive AI use for revision and stylistic enhancement. * All AI contributions are explicitly cited and acknowledged. * Student maintains overall control and authorship.	"Analyze this essay for clarity and style. Suggest improvements (cite all suggestions)" "Revise this essay for consistent tone (cite all suggestions)" "Check for logical fallacies (cite all suggestions)"
8	AI for Significant Idea Generation & Structure	* AI significantly contributes to idea generation and structural planning. * Requires explicit instructor approval. * All AI contributions are meticulously cited and documented.	"Brainstorm and outline a research paper on [topic] (cite all AI contributions)" "Develop a comprehensive research plan (cite all AI contributions)" "Generate potential arguments and counterarguments (cite all AI contributions)"

LLM guidelines

examples

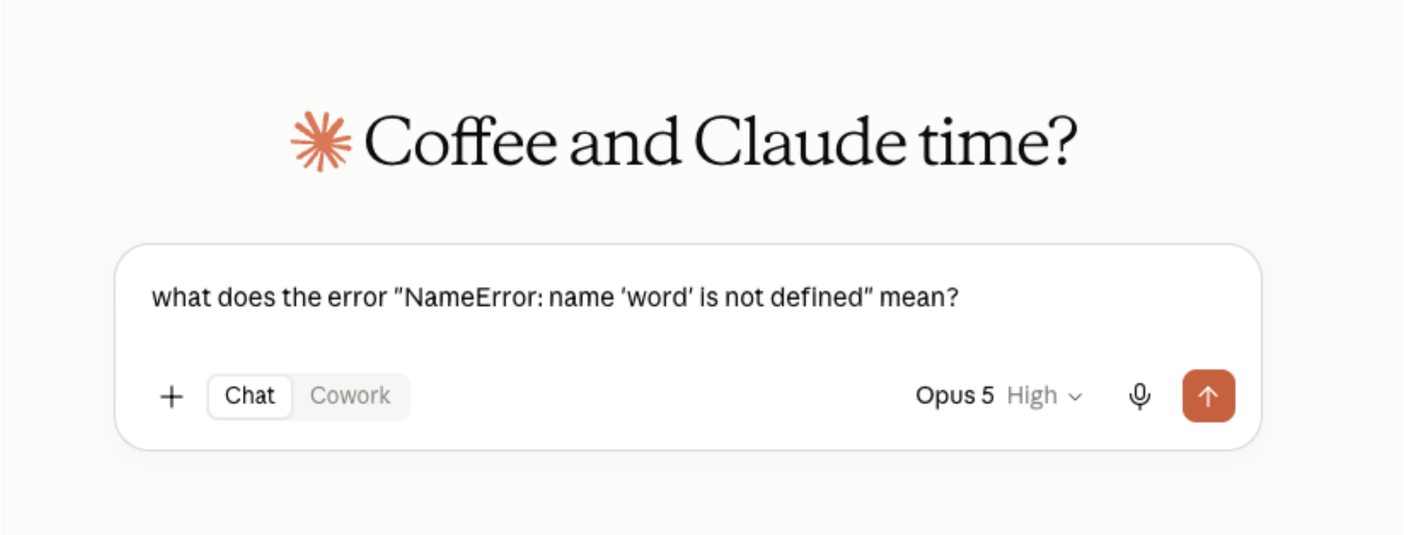


level 1-2

ok!

LLM guidelines

examples

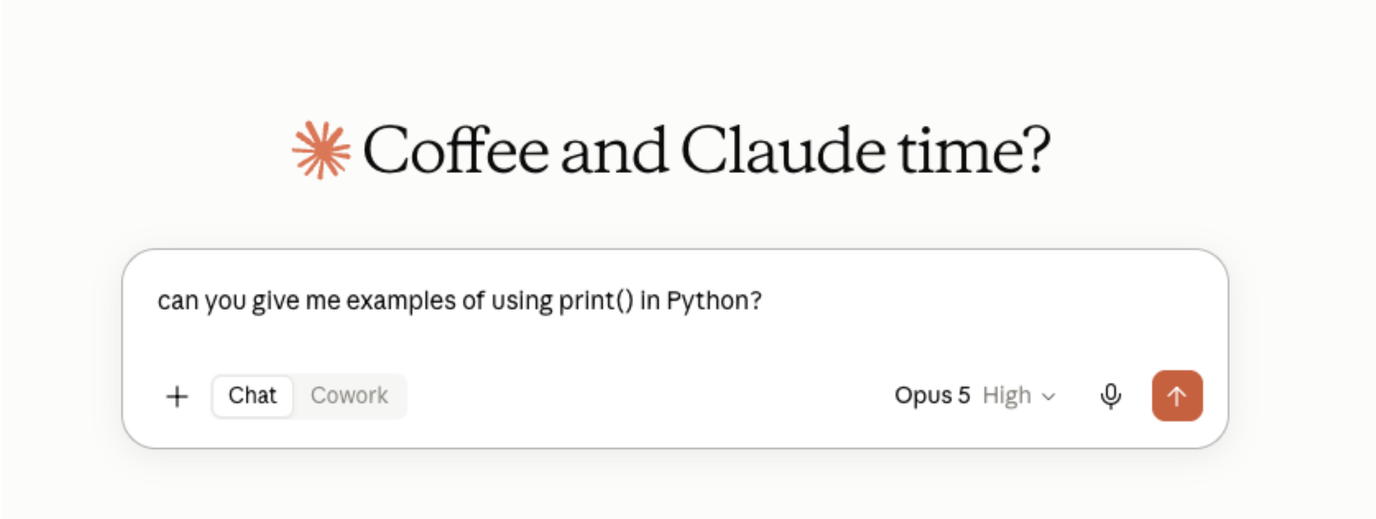


level 5

ok!

LLM guidelines

examples

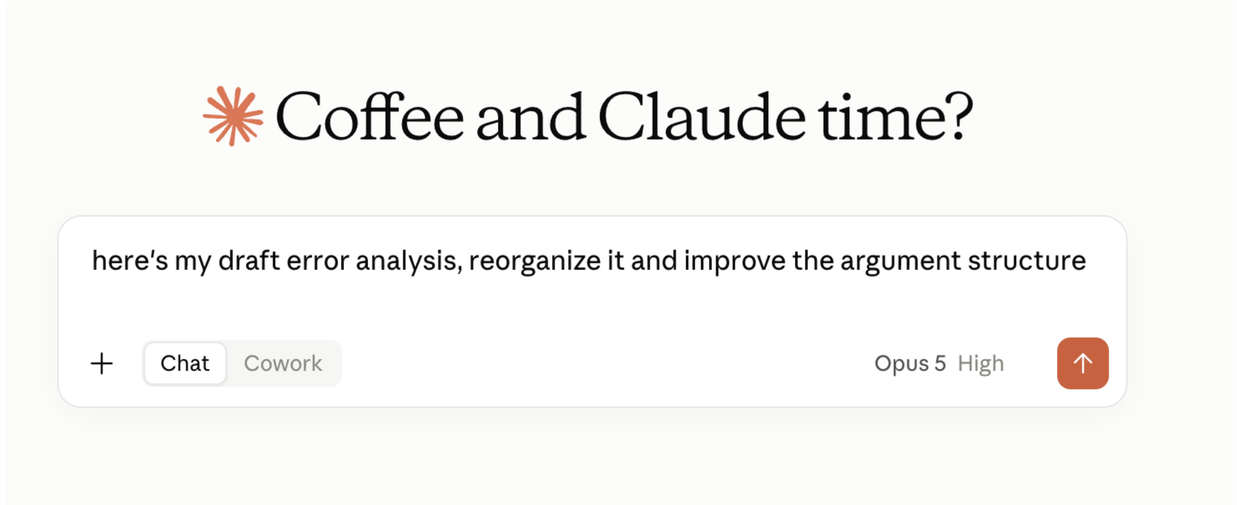


level 5

ok!

LLM guidelines

examples

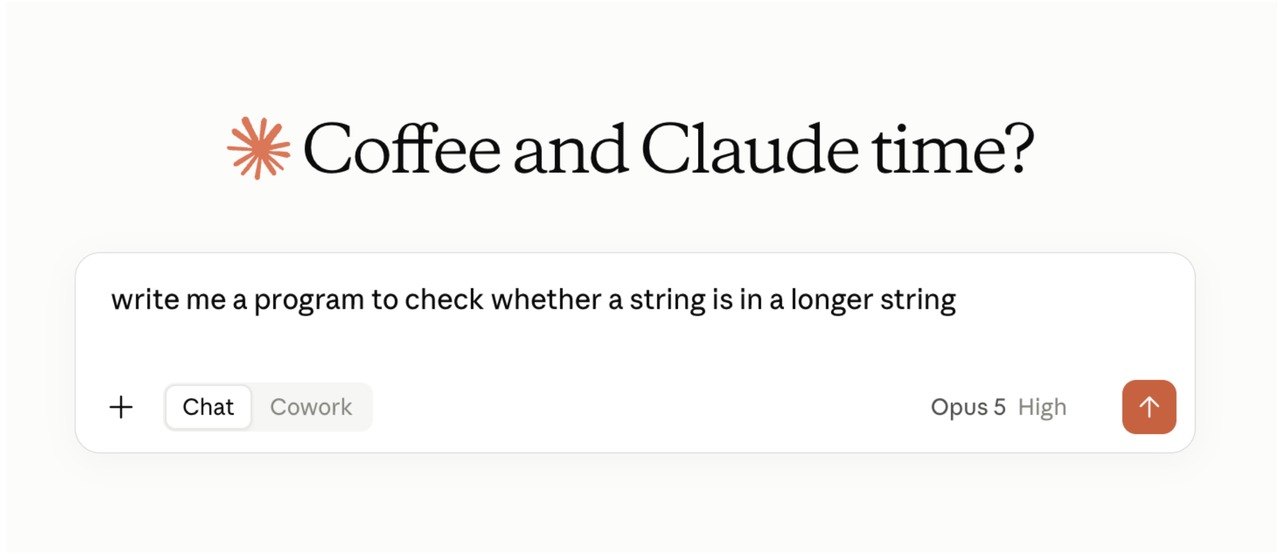


level 6

not ok!

LLM guidelines

examples

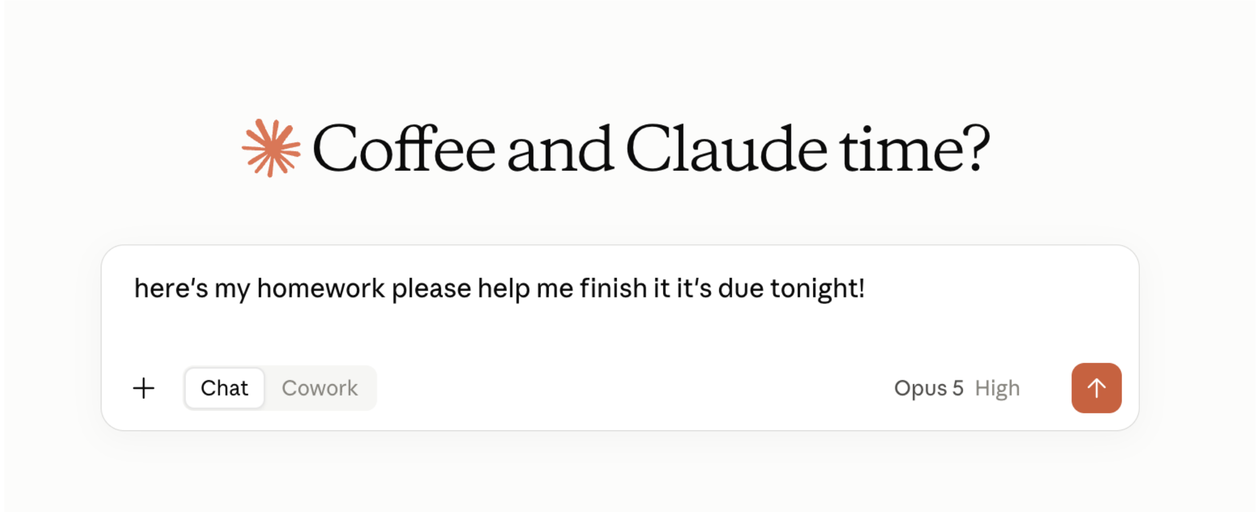


level 7-8

not ok!

LLM guidelines

examples



level 8

not ok!

LLM guidelines

- our line is between **5 and 6**
  - other courses might draw it elsewhere
  - disclose what you used, name the level, and be able to explain the result
- use it as a **tutor**
  - explain a concept, help you read an error message
  - check and verify in the actual documentation
    - https://docs.python.org/3/
- not as a **ghostwriter**
  - it should not write the code or the analysis you turn in

activity

activity

- finish the 3 cells at the end of the Python notebook
- raise your hand or ask a neighbor if you get stuck

next Tuesday

- read
  - NLTK Book Ch. 1, §§1–2
  - Think Python Ch. 2
- if they look easy to you
  - great!
- if you need time to understand the content
  - spend enough time so you could understand them
- HW 1 will be released on Tuesday
- Feel free to come say hi during office hours