Digital Methods

HIST 1030: Digital Methods in Humanistic Research
Spring 2027

When	Tuesday/Thursday, 9:00–9:50 am; recitation 10:00–10:50 am
Where	118D Mervis; recitation 3516 Posvar
Instructor	James Pickett
Email	pickettj@pitt.edu
Office	3908 Posvar
Office hours	Mondays 10:30-11:30am; Tuesdays 2:30-3:30pm
Website	https://dh.bactriana.org/

Last updated November 12, 2026. The course website is the current version of this syllabus.

Syllabus 2
Course Description 13
Learning Outcomes 15
Course Format and Materials 17
(Un)Grading 20
Research Projects 24
University-Required Syllabus Policies 27
Appendix: AI Policies for an AI Course 28
Appendix: Getting Set Up 32
Appendix: Copyright 33
Appendix: Past Projects 36

Syllabus

Homework listed under a session is assigned that day and due before the next session, unless it says otherwise. All assignments are submitted on Canvas. For what class sessions, homework, and capstone coding assignments are like, see Course Format and Materials.

Each homework assignment is marked in one of three ways (see AI Policies for an AI Course):

- *(AI off)*: on your honor, no AI. These are short, and they exist so that you can do the AI-assisted work better.
- *(AI required)*: directing and checking an AI model is the assignment. Say what you did in the comments of what you hand in.
- No mark: your call. If you use AI, say so in a comment (see Saying What You Did).

Your Worm Journal gets at least one short entry for every class session. Where a session lists a **Journal** prompt, that prompt is your entry.

Part One: What Does a Computer Know About a Text?

Core questions: What is a text, to a computer? What do we add to a text when we mark it up? What is an AI model doing when it "reads" one?

Session 1 — Tue, Jan 12: A Text, a Tree, and a Machine

In class:

- What is (computational) digital humanities? A look at a few past projects
- How this course works: (Un)Grading, the Worm Journal, AI on and AI off
- Introduction to XML: hierarchy, well-formedness, elements and attributes, using Lord of the Rings quotes

Homework:

- Work through Getting Set Up, and bring a paper notebook and a pen to the next class.
- Read An even gentler introduction to XML.
- XML exercise 1 *(AI off)*: copy the text of one letter by either Oscar Wilde or Anton Chekhov and mark it up in XML, using whatever markup you consider appropriate. Add a short paragraph explaining three of your tagging decisions.

Journal: Read Can humanities undergrads learn to code?, written in 2012 by two undergraduate instructors in this course's ancestor. What in it is still true now that AI can write code? What is not?

Session 2 — Thu, Jan 14: Human Markup, Machine Markup

In class:

- Briefing: how large language models work (prediction, training, context), and the difference between deterministic and stochastic systems

- Exercise: an AI model marks up the same letter you did. Compare the two, and ask why its markup looks the way it does
- Types of markup: descriptive, presentational, procedural
- A standing rule for the semester: the model's own report of what it did is never verification
- Introduction to the research project

Homework:

- XML exercise 2 *(AI required)*: have an AI model mark up a short text of your choice (any genre, any language; ideally something you might want for a project). Then correct its markup, and submit your corrected version with a punch list of what you changed and why.
- Read You aren't a beginning XML developer anymore, and use it as a checklist against the AI's markup.
- Read Research Projects and Copyright.
- Complete the questionnaire on research interests (Canvas).

Journal: Choose one project from Past Projects. What was its research question? What did its makers have to mark up in order to ask it? Name one thing you would steal for a project of your own.

Recitation this week: Setup clinic. One past project, from first idea to finished site.

Part Two: Nuts and Bolts

Core questions: Where do files live, and how does a computer find them? What is a program doing when it runs? How is instructing a program different from instructing an AI?

Session 3 — Tue, Jan 19: Files, Paths, and the Command Line

In class:

- Paths: absolute and relative locations, the working directory, and why moving a file breaks everything that pointed to it
- Coding without computers: path tracing (draw a project's folder tree on paper and predict where relative paths lead)
- The command line: a command is a program plus what you hand it, and an error message is information
- Read the command: annotate commands supplied by an AI in plain English, asking what each acts on and whether it can be undone

Homework:

- Command line practice *(AI required)*: in the practice folder provided, ask an AI model how to carry out the listed file tasks. Before running anything, annotate each command in plain English and mark any that cannot be undone. Then run them and record what happened.

Session 4 — Thu, Jan 21: Algorithms, Loops, and Functions

In class:

- Coding without computers: the literal-minded research assistant. Instruct a "robot" to alphabetize a stack of cards, then instruct an "AI" that fills every gap with a plausible guess. How do the failures differ?
- Loops and conditionals in plain language
- Functions as contracts: what a function takes, what it returns, and when it fails. Function cards, chained into a pipeline

Homework:

- Function cards *(AI off)*: write three function cards for small tasks a project of yours might need (name, takes, returns, fails when).

Recitation this week: Forming project teams; team norms, meeting times, and communication.

Part Three: Modeling a Text

Core questions: What is this document made of? Which of its features matter for your question, and why? How do you turn that analysis into rules a computer can enforce?

Session 5 — Tue, Jan 26: Document Analysis

In class:

- Document analysis without angle brackets: Kubla Khan
- Every document contains nearly infinite information. What you model depends on what you want to know
- Coding without computers: conditional logic as editorial policy. Draft if/then rules for a fuzzy editorial decision, trade them, and apply another team's rules literally

Homework:

- Document analysis, stage 1 *(AI off)*: choose a small text, ideally from your project's corpus. In one to two pages of prose, with no code, describe its structures and features, say which you would model, and say why.

Project: Teams (or solo researchers) submit a one-paragraph sketch: your text or texts, your question, and who is on the team.

Session 6 — Thu, Jan 28: Schemas: Rules for a Document

In class:

- What is a schema? Relax NG by way of a pasta recipe
- Coding without computers: the type sorting game. What is a string, a number, a date, a list? What are you allowed to do with each?
- Reading a schema aloud

Homework:

- Read Introduction to Relax NG.

- Relax NG exercise 1 *(AI off)*: write a Relax NG schema for the XML you produced in XML exercise 1 or 2 (you may modify your XML). Upload both files.

Journal: Explore one of these archives: the Blake Archive, the Rossetti Archive, Melville's Marginalia, or the Perseus Digital Library. You cannot see its schema, but you can infer it. From what the site lets you do, name three things its makers must have decided to model, and one thing they evidently did not.

Recitation this week: Assembling a corpus; first document analysis of your project's texts.

Session 7 — Tue, Feb 2: Mixed Content and Harder Cases

In class:

- The Dungeon Crawl: Relax NG patterns for harder cases (mixed content, empty elements, choice, order)
- How much markup is enough? Two projects in one field with two philosophies: TEI and OpenITI mARkdown

Homework:

- Read Relax NG content models.
- Document analysis, stage 2: write a schema that formalizes your stage 1 analysis, mark up the text according to it, and validate. Upload the XML and the schema.

Session 8 — Thu, Feb 4: The AI's Schema, and Yours

In class:

- An AI model can write a schema for your document in thirty seconds. Have it do so from the text alone, then set its schema beside your prose analysis. What did it skip? What did it get wrong?

Homework:

- **Capstone Coding Assignment 1** *(AI off)*: XML, paths, and schemas. Handwritten and scanned, no notes, and longer than a weekly exercise: it pulls together the unit.
- Schema critique *(AI required)*: a written comparison of the AI's schema and yours for the same document. Where they differ, say which is right for your research question and why.
- Read the regular expressions quick start.
- Scan and submit your Worm Journal so far (first check-in).

Recitation this week: The Text Encoding Initiative: what other people have considered worth modeling, and what to borrow. Project meetings.

Part Four: Pattern Matching

Core questions: How do you describe a pattern precisely enough for a machine to find it? How do you know that a pattern matches what you think it matches?

Session 9 — Tue, Feb 9: Regular Expressions

In class:

- Regex, the superpowered search-and-replace language; the difference between a language and a program
- Walkthrough: Sonnet 60, and puzzles on a word list
- Outside in: tag the largest structures first

Homework:

- Regex exercise 1 *(AI off)*: turn a plain-text edition of Shakespeare's sonnets into XML with search and replace, and write up your process.
- Read the first half of Regex tips (stop at "Using regex in XSLT").

Journal: Read Stephen Ramsay, The Hermeneutics of Screwing Around. Ramsay distinguishes searching from browsing. Where does asking an AI fall? Does it change his argument?

Session 10 — Thu, Feb 11: Almost Right: Regex with AI

In class:

- Three candidate patterns for the same job, one yours and two from an AI: test them, rank them, and explain the ranking (The Scarlet Letter)
- Adversarial testing: build an input designed to break a pattern
- Autotagging a text toward your own schema

Homework:

- Regex exercise 2 *(AI required)*: autotag *The Blithedale Romance*. For each step, have an AI model propose the pattern, then test it against the text. Your write-up records every place the AI's pattern failed, how you found out, and what you changed.

Recitation this week: Project meetings: autotagging your own corpus. Optional session on version control, for teams that want it.

Part Five: Asking Questions of a Document

Core questions: How do you ask a document a question? How do you know that the answer is right?

Session 11 — Tue, Feb 16: Paths Through a Tree

In class:

- Paths again: from folders to document trees
- XPath: paths and axes, with fables.xml and Bad Hamlet
- Coding without computers: draw the tree, and walk an expression through it with your finger

Homework:

- Read What can XPath do for me?

- XPath exercise 1 *(AI off)*

Session 12 — Thu, Feb 18: Predicates and Functions

In class:

- Predicates and functions
- Develop and test (!) path expressions one step at a time
- How to look things up, and how to check what an AI tells you

Homework:

- Read The XPath functions we use most.
- XPath exercise 2 *(AI off)*

Recitation this week: XPath workshop: harder problems, worked together (XPath exercise 4).

Session 13 — Tue, Feb 23: When the Expression Looks Right

In class:

- Diagnose and fix: expressions that look right and are not (the `.` in a predicate; `//stage[last()]` versus `(//stage)[last()]`; a sibling axis that returns nothing)
- Bad Hamlet as a trap: its markup is deliberately damaged, which is exactly what an AI model does not expect
- For fun: string functions, and Hamlet rewritten

Homework:

- XPath exercise 3 *(AI required)*: put each task to an AI model before writing anything yourself. Then check every answer against the document. Report which answers were wrong, how you could tell, and the corrected expression.

Session 14 — Thu, Feb 25: Rules That Catch Mistakes

In class:

- Schematron: editorial rules written as XPath, which a computer can enforce (the Best Stooge Ever election, where percentages must add up to 100)
- Checking an AI's markup at scale

Homework:

- **Capstone Coding Assignment 2** *(AI off)*: trace the tree by hand. Handwritten and scanned, no notes.
- Read the Schematron tutorial.
- Schematron exercise: write three rules for your project's markup that your Relax NG schema cannot express, and show each one catching a mistake.

Recitation this week: Project meetings: teams trade schemas and review each other's models.

Part Six: Directing the Machine

Core questions: What should a reader see? How do you say what you want precisely enough for an AI to build it? How do you know that what it built is right?

Session 15 — Tue, Mar 2: Saying What You Want

In class:

- How large language models work, revisited: now that you have used them for seven weeks
- The context window field trip: give a model a document that is too long, and document what it does to the middle
- Coding without computers: specification poker. Write a short plain-language specification, trade it, and watch another team satisfy it as unhelpfully as its wording allows

Homework (due Thursday, March 18):

- Reading view specification *(AI off)*: in plain language, with a hand-drawn sketch, specify what a reader of your project's text should see. No code.
- Read Thinking in algorithms.
- Scan and submit your Worm Journal so far (second check-in).

Journal: Read the American Historical Association's Guiding Principles for Artificial Intelligence in History Education. Where does this course follow them? Where does it not, and who is right?

Project: By the recess, your team should have its corpus assembled, a draft schema, and a sample of the text marked up.

Recitation this week: Project meetings: checkpoint before the recess.

No class Thursday, March 4 through Tuesday, March 16 (spring recess is March 7–14)

Session 16 — Thu, Mar 18: The Target: HTML and CSS

In class:

- XML and its relatives: HTML and TEI
- A web page as a tree: structure in HTML, appearance in CSS
- The do's and don'ts of AI assistance for web design
- Exercise: an AI model generates a page. Change it until it is yours

Homework:

- HTML/CSS exercise *(AI required)*: have an AI model generate a first page for your project's site. Submit the page with a punch list of everything you changed and why.
- Read XSLT basics.

Journal: Read the first chapter of Don Norman, *The Design of Everyday Things* (on Canvas), with attention to affordances and signifiers. How should those two ideas shape the interface of your project?

Recitation this week: Thinking about user experience.

Session 17 — Tue, Mar 23: XSLT: Trees In, Trees Out

In class:

- XPath refresher
- Templates and push processing. Coding without computers: which template fires?
- A first stylesheet, written together: an outline of Bad Hamlet

Homework:

- XSLT exercise 1 *(AI off)*: finish the outline of acts and scenes by hand. Above each template, add a comment saying in plain English what it does and why it is there.
- Read XSLT, part 2: conditionals and push and pull and Developing an XSLT stylesheet.

Session 18 — Thu, Mar 25: Specify, Generate, Diagnose

In class:

- The workflow: your specification, an AI's draft stylesheet, your diagnosis
- Four ways an AI's stylesheet goes wrong: pulling where it should push; templates that never fire; namespaces that fail silently; valid HTML that misrepresents the document
- Coding without computers: the debugging autopsy. A stylesheet and its wrong output, on paper. Find the bug by reading
- Specification poker, round two: this time the genie is an AI

Homework:

- XSLT exercise 2 *(AI required)*: write a specification for a reading view of skyrim.xml (tables of characters and factions, and a styled narrative). Have an AI model draft the stylesheet. Diagnose it against the four failure types, fix it, and annotate every template.

Recitation this week: Project meetings: a first reading view of your own XML.

Session 19 — Tue, Mar 30: Reading Views as Arguments

In class:

- One document, two treatments: modal XSLT and Shakespeare's sonnets
- Prompt archaeology: given a finished reading view, reconstruct the specification that would produce it
- A provocation: do we need to transform at all? (CETEIcean)

Homework:

- **Capstone Coding Assignment 3** *(AI off)*: reading a stylesheet. Handwritten and scanned, no notes.
- XSLT exercise 3: build a reading view of your own project's XML from the specification you wrote over the recess. Teammates may share XML, but each of you submits your own stylesheet. Be ready to explain it.

Part Seven: From Tree to Table

Core questions: What counts as an observation? What is a row? What does a text lose, and gain, when it becomes a table?

Session 20 — Thu, Apr 1: Rows Are Arguments

In class:

- The same skills, a different product: sometimes a transformation makes a dataset, not a page
- Coding without computers: tidy data. Rows are observations and columns are variables. What is a row in Hamlet?
- Markup stays the source of truth; the table is derived from it

Homework:

- Tree to table exercise 1: produce a table of the speeches in Bad Hamlet, one row per speech, as a CSV file. In a paragraph, defend your decision about what counts as a row and which columns it needs.

Recitation this week: Walkthroughs: explain one template of your stylesheet to a TA. Python setup clinic.

Session 21 — Tue, Apr 6: Tables at Work

In class:

- Opening your table: first in a spreadsheet, then in Python
- People and places as tables: a network is a table of connections, and a map is a table of coordinates
- Checking a table against its source

Homework:

- Tree to table exercise 2 *(AI required)*: extract a dataset from your project's XML. Submit the table, a definition of its rows and columns, and an independent check: counts from the table set beside counts from XPath.

Journal: Read Erin Davis, Sweet & slim, greasy & grim: the physical traits that define men & women in literature. What table sits underneath this essay? What was a row? What would you want to check before believing it?

Part Eight: Analysis and Visualization

Core questions: What can counting tell you about a text, and what can it not? What should a chart argue? How do you check a script that you could not have written yourself?

Session 22 — Thu, Apr 8: Directing Python

In class:

- Not learning Python, but directing it: reading a script by its function signatures, its loops, and its types
- Frequency lists and concordances
- Old friends in a new setting: XPath and regular expressions inside Python, now over a whole corpus

Homework:

- Python exercise *(AI required)*: the verification ladder. Have an AI model write a script that answers one counting question about your corpus. Verify it three ways: by reading it, by testing it on a case where you already know the answer, and by building an input designed to break it. Compare what each method caught.
- Scan and submit your Worm Journal so far (third check-in).

Recitation this week: Project meetings: what will your project count, compare, or draw?

Session 23 — Tue, Apr 13: A Menu of Methods

In class:

- Short demonstrations, each starting from a table: counting and comparing; networks; maps; stylometry (whose voice is this?); topic models
- At the scale of a library: detecting reused text across a corpus (KITAB)
- Teams choose one method to try on their own data

Homework:

- Menu exercise *(AI required)*: try your team's chosen method on your own table. Put the same task to two AI models, or to one model twice, and compare what comes back. State what your result shows and what it does not.

Journal: Read the item that matches your team's method, and say what it suggests for your project: counting, Mining the Dispatch (the introduction); networks, the Background section of An introduction to network analysis and Cytoscape for XML coders; stylometry, José Binongo, Who wrote the 15th book of Oz?

Session 24 — Thu, Apr 15: What Should a Chart Argue?

In class:

- Coding without computers, then with one: the SVG coordinate space, and a bar chart drawn by hand (Best Stooge Ever)
- From coordinates to a grammar of graphics: describing a chart by mapping data to position, color, and size
- A chart is an argument: what does yours claim?

Homework:

- Visualization exercise: make one chart from your project's table. In a paragraph, say what it argues, and name one way it could mislead.

Journal: Watch Hans Rosling, The best stats you've ever seen. What does his presentation suggest that visualization can contribute to an argument? Are you persuaded?

Recitation this week: Project meetings: your site, and preparing for the showcase.

Part Nine: The Research Project

Core questions: What did you find out? How do you know? Can you explain how your project works?

Session 25 — Tue, Apr 20: Project Studio

In class:

- A working session for project teams, with the instructional team on hand
- Be ready to explain your project's code to a TA: what it does, why it is there, and how you know it works

Recitation this week: Project meetings: final checks.

Session 26 — Thu, Apr 22: Project Showcase

In class:

- Teams present their research projects

Project: Projects and final reports are due during finals week (April 24–30), on dates to be announced.

Course Description

This is a research course. You will learn to use computational methods to pursue your own independent research in the humanities: history, literature, philosophy, linguistics, and more. How can coding help you see things in a text that might otherwise be invisible? The course is not tied to any one topic, or even to any one discipline: the research questions come from you, and the methods we use range just as widely. By the end of one semester, students with no prior coding experience (our target constituency) will be seasoned researchers in the digital humanities.

The field has recently gone through a transformative and exciting development. AI puts within reach all sorts of things that, until very recently, a beginner could not have attempted in a single semester, and we are going to take full advantage of that. AI is one of the major tools of this course, and learning to use it well is one of the things you will take away from it. (For how and when we use it, see AI Policies for an AI Course.)

"Digital humanities" is a capacious rubric, and "coding" does not narrow it down much either. One thing we will spend a lot of time thinking about is texts. That does not mean we study texts exclusively, but a great deal of humanistic research works with texts in one way or another. A major goal of this course is to figure out how to model a text, so that a computer can work with it in ways that matter to a humanist, and then how to extract information from it. Much of that modeling happens in XML and its related technologies, which let you make explicit, structured claims about what a text is and what is in it.

Even though you will spend a lot of time coding (and directing AI to code), this is a humanities research course before it is a coding course: the computer is the means, not the end. It is fitting, then, that the centerpiece of the course is a research project, the topic of which is entirely driven by your interests, and to which you will begin applying your new skills immediately.

Legacy

This course is modeled on one pioneered by emeritus Pitt Professor David Birnbaum, and we draw extensively on the material on his website, Obdurodon. (What is an "Obdurodon," you ask? An extinct platypus with molars, apparently… don't worry about it.)

That course took shape before the age of AI, so we do not follow it to the letter. Its key insights, though, are as valid as ever, and Obdurodon remains an encyclopedia of information essential for this course. Just bear in mind that course-specific information on Obdurodon (due dates, classroom assignments, and the like) may relate to a previous semester.

Dr. Birnbaum summed up the impetus for this course as follows:

> Humanities students often do not realize (or even imagine) that 1) they are capable of learning to write useful and practical computer programs within the course of a semester even if they have no prior background in programming; 2) the ability to write one's own programs can be valuable for scholars in the humanities, especially because commercial software often does not

> address research needs in the humanities; and 3) practical computer programming, no less than reading, writing, and arithmetic, is a useful skill that is within the reach of any educated person regardless of academic specialization.

AI has only strengthened the case: what a humanities student can build in one semester is far greater now than when he wrote those words.

Learning Outcomes

Much of the subjectmatter in this course is driven by student interests, unbound by a shared commitment to any dingle topic or even discipline. What follows serves as a shared reference point for the feedback you receive on assignments and exercises, and as a basis for your final report, where you assess your own work (see (Un)Grading).

Research and Modeling

1. **Ask a research question** about a text, or a body of texts, that computational methods can help answer, and explain why it matters in your field.
2. **Analyze a document:** identify its structures and features, and decide which of them to model, justifying your choices in terms of your research question.
3. **Design a schema** that expresses your model, and explain what it allows and what it rules out.

Coding Literacy

4. **Explain core computational concepts** (paths, loops, conditionals, data types, and functions) in plain language, and demonstrate them on paper.
5. **Read code you did not write,** including XML (TEI or otherwise), regular expressions, XPath, XSLT, and Python, and explain what each part does and why it is there.
6. **Mark up a text,** both by hand and with AI-assistance, and check it against your schema.
7. **Extract information** from a structured text and turn it into something new: a reading edition, a table, a chart, a network, or a map.

Working with AI

8. **Explain in general terms how large language models work,** including the difference between deterministic and stochastic systems, and what follows for when to trust them.
9. **Specify a task** precisely enough for an AI tool to carry it out, and recognize when it has not; learn to build effective stylesheets and spec documents.
10. **Take responsibility for the code you use:** verify it independently, by reading it, by testing it on cases with known answers, and by trying to break it, never by taking the model's word for it.
11. **Look things up:** find answers you do not already have, in reference documentation, online, and from AI, and judge which of those answers to trust.

Research in Practice

12. **Present your research** in a form other people can see and use, including website design.
13. **Collaborate:** divide the work, communicate, credit each other's contributions, learn workflow and version control best practices.

14. **Reflect on your own learning** in writing, and in a way that feeds back into the research process.
15. **Discuss how digital methods,** and AI in particular, **are changing research in the humanities,** drawing on other digital humanities projects.

Course Format and Materials

Class sessions will very in format, moving between lecture, code walkthroughs, and hands-on exercises. (For how all of this adds up to a grade, see (Un)Grading; for when and how we use AI, see AI Policies for an AI Course.)

The Heart of the Course: Your Research Project

This entire course revolves around a research project on a topic of your own choosing, which you will start working on almost immediately. Most recitation sessions will be devoted to project work, sometimes in a structured manner with specific assignments meant to advance your project progress, and sometimes as discretionary time (in *addition* to your weekly project meetings, separate from class).

In Class

Typical in-class activities include:

- **Coding walkthrough**: Guided discussion of the features of complex code, aimed to increase general literacy.
- **Digital humanities lecture**: Conceptual coverage of computational methods and AI as a field of inquiry.
- **Coding without computers.** Quizzes and exercises on paper that drive home core concepts and coding literacy: functions, paths, loops, and the like (see Pen and Paper).
- **Project showcases.** We look at digital humanities projects, figure out how they were built, and consider the research questions behind them, and what they suggest for yours.
- **Hands-on work,** machines open, on code and on your project.

Recitations

There are two recitation sessions each week. Choose one; you are welcome to attend both, and / or are at liberty to switch at the last minute without notifying an instructor (though you should keep your team members posted). Many recitations are dedicated to project meetings, so do your very best to make sure that everyone on your project team attends the same one. Don't worry about registration: you are free to switch between them as you see fit.

Homework

Coding Assignments

Coding assignments in this course are a technique for learning, not, as in many other courses, a test of whether you have already learned something covered in class. A crucial skill in the digital humanities is figuring out how to do something you don't yet know how to do, so these assignments

will often ask you to write code you do not yet know how to write. If you are used to homework-as-test, this may feel disconcerting ("But they haven't told us how to do this!"). Professional developers look things up all the time, and learning how to find answers, in references, online, and with AI, is one of the goals of this course.

AI on or off. By default, you may use AI on coding assignments. When an assignment asks you not to, it will say so plainly. Either way, you are responsible for the code you submit (see AI Policies for an AI Course).

Getting stuck. There will be times when you don't get the result you want. In those cases, submit what you have, along with a description of what you tried, what you expected, what you got, and what you think went wrong. That counts as completing the assignment. Getting stuck is part of learning, and the TAs and I are happy to help unstick you once you have described the problem and your own attempts to solve it.

Comment your code. Every coding assignment must include at least one relevant, thoughtful comment, written directly in your file as a formal code comment (we'll show you how), not in a Canvas comment box. This builds the habit of reading, and explaining, the code you use.

Assignments and Feedback: Links to all assignments are on the syllabus. Some assignment descriptions live on Obdurodon, some on this site, and some on Canvas. Don't wait for an assignment to appear on Canvas to get started, even though Canvas is where you submit it. In some cases, we will post an answer key and discussion.

Reflection Assignments

Reflection assignments usually take the form of special entries in your Worm Journal. For instance, we might ask you to look at how other digital humanities projects pursued their research, and what lessons they hold for your own project. Or we might prompt you to think about how digital methods, and AI in particular, are changing our society, and especially research in the humanities.

A reflection does not have to be long, but it should show real thought. Don't summarize an article (which you could do by skimming the first paragraph), and don't just praise or condemn a project without getting specific about what works, what doesn't, and why. Some projects matter for their design, others for their research methodology; engage with whatever is most significant about the one in front of you.

Periodically, we will ask you to submit scans of your handwritten reflections to Canvas (so that you can keep your journal with you, and so that we have a joint record). You will need some way of turning paper into a PDF: a scanning app on your phone is fine, and most phones have one built in (see Getting Set Up).

Capstone Coding Assignments

At the end of each unit there is a capstone coding assignment: a longer piece of work that pulls together what the unit covered and asks you to show that you can use it. They are announced

on the syllabus. Most are take-home, and their format varies, but usually they are no AI and no notes. A capstone comes back with feedback on each part, but no grade. Its job is to show you where things stand, and you can draw on it when you assess your own work at the end of the semester (see (Un)Grading).

Help, Collaboration, and the Course Forum

Coding was never a solitary pursuit, and even less so now with "friends" like Claude. Collaboration is welcome. You can always ask for help: from the TAs, from me, and from your classmates. A lot of the time you will also be consulting online resources and AI.

Despite the image of the programmer as a solitary soul hunched over a glowing screen, programming is a social activity, and real programmers make liberal use of discussion forums to ask and answer questions. Ours lives on GitHub, which is where those conversations actually happen in the wild, and where the code we hand out lives. Everyone sets up a GitHub account in the first week (see Getting Set Up). You do not need to learn Git to use the forum: it is a web page with comment boxes on it.

That gets you two things. You can ask about course code in the place where the code is, rather than describing it from memory: comment on the line that is giving you trouble, and whoever answers is looking at exactly what you are looking at. And when you want help with your own work, you can @-mention an instructor on your own repository, so that we can read the actual file instead of a screenshot of it.

The forum is entirely optional but encouraged: don't be shy about asking questions there, and don't be shy about answering them, which is both good citizenship and satisfying. Your contributions are also evidence of engagement that you can point to in your final report.

What You Need

A computer of your own. We assume you have regular access to a computer and can bring one to every class: we do not meet in a computer classroom, so whatever you bring is your lab. If that is not the case, talk to me or one of the TAs. There are various solutions at Pitt.

Software. All of it is free to you. For installation, see Getting Set Up.

- **VS Code,** with GitHub Copilot (see Tools).
- **oXygen XML Editor,** through Pitt's license. Installation instructions and the license key are on Canvas.
- **Python** (via Anaconda) later in the semester, with details announced in class.

Reference. Obdurodon is the course's encyclopedia: tutorials, explanations, and many of the assignments.

(Un)Grading

This course does not grade your work in the usual way. Nobody will put an overall score on your assignments (though there will often be right and wrong answers to individual questions), and there is no weighted formula at the end of the semester. Instead, you decide your own final grade and make the case for it.

Why (Un)Grading?

This is an honors course. Students who choose it tend to want a challenge, and most of you are here because the material interests you. If you make a good-faith effort all semester, you have done what this course asks of you. Managing that effort through points, penalties, and percentages would add paperwork and anxiety (yours and mine) without adding much learning.

The course is built around a research project of your own design. Even the assigned homework leaves a lot of room for personal exploration: two students can complete the same assignment in very different ways and learn a great deal from each. A single scale for measuring all of that fits poorly.

Flexibility. You have other classes, jobs, families, and lives. Under this system you can decide to put in a B effort here and devote more time to something else, and that is a legitimate choice. You owe nobody an explanation for it except yourself, and, at the end of the semester, your final report.

AI. When an AI model can produce a working schema or stylesheet in thirty seconds, a grade on the finished file says very little about what you understand. What matters in this course is your judgment: what you asked for, how you checked it, and what you learned along the way. Points measure that badly. Your own account of your work shows it much better. (See AI Policies for an AI Course for more.)

There is also a pedagogical case. "Ungrading" is an established movement in higher education. Its proponents argue that grades are a poor form of feedback, that they steer students toward safe choices over interesting ones, and that students learn more when asked to reflect on their own learning (Blum, *Ungrading*; Stommel, "How to Ungrade"). Research on how well it works is still limited, though studies of students' experience in ungraded courses have mostly been encouraging (Hasinoff et al. 2024; Emerson 2026).

The Short Version

- **Your final grade is the grade you claim,** justified in a final report at the end of the semester.
- **This is the honor system.** It assumes that you will self-evaluate with integrity.
- **Your evidence:** your completion record, your Worm Journal, feedback on your capstone coding assignments, and the course's learning outcomes.

- **Feedback, not grades:** throughout the semester, the TAs and I respond to your work in writing with qualitative assessment.
- **I override a claimed grade at the end of the semester only in rare cases,** and never without written warning well in advance.
- **No need to email me about absences or missed work.** Account for them in your journal and your final report instead.

How Your Grade Is Determined

At the end of the semester you will submit a final report in which you claim a grade and explain why it is the right one. (The report has its own page: Final Report.) Except in the rare cases described below, the grade you claim is the grade you receive.

The point of the exercise is not to guess the "right answer" or anticipate what grade I have in mind for you. The decision, and the responsibility for it, rest on your shoulders: you are the one who knows how much you put into this course and what you got out of it.

What Goes Into Your Decision

Your Completion Record

The TAs and I keep track of which assignments you complete, both homework and in-class work. At the end of the semester you will receive a report summarizing it: for example, what percentage of the homework assignments you completed. This is information for you to interpret and weigh in your final report, not a formula for your grade.

The Worm Journal

You will keep a journal throughout the semester, written in ink pen in a paper notebook of your own.

Why "Worm"? In computing, WORM storage means "write once, read many": data can be written a single time and read as often as you like, but never altered or erased. A page written in ink works the same way, and needs no electricity. (Keeping one also makes you a bookworm.) In a course about digital technology, the journal is deliberately the oldest technology in the room.

What it is for. The journal is where you reflect on, describe, and articulate what you are learning. It is our window into what you are actually getting out of this course, which we cannot observe directly.

What it is not. It is not a log. We already track attendance and completed assignments, so there is no need to record them here. Write about ideas, problems, discoveries, frustrations, and connections, especially as they bear on your research project.

Form and content. We want the journal to be flexible and open-ended, so its form and content are largely up to you. The one firm minimum: at least one short entry for every class session.

Keep it with you. Bring the journal to every class. Some in-class activities use it, and we may ask to look at it during class. From time to time we will also ask you to scan and submit what you have written so far. The notebook itself always stays with you.

It is not private. Although we call it a journal, it is not a diary. The instructor and the TAs will read it, so do not put anything in it that you would not want us to see.

Capstone Feedback

The capstone coding assignments at the end of each unit come back with feedback on each part, but no grade. They are meant to show you where things stand, and you can draw on them when you make your case.

Learning Outcomes

The course's learning outcomes describe what you should be able to do by the end of the semester. They are the shared reference point for your final report and for the feedback you receive along the way.

Feedback Along the Way

When you submit journal scans or responses to prompted reflection assignments, I will respond in writing with my own qualitative assessment of how you are doing. This feedback does not determine your grade. It exists to help you assess yourself. It might say, for instance, that you need to do more to catch up, or that you are being too selective in what you devote attention to. If the evidence plainly supports a higher grade than the one you seem to be aiming for, I will tell you that too.

These check-ins happen at several points during the semester and serve the purpose of a midterm report: they come early enough for you to change course.

The TAs also give you written feedback on your work. They never grade it.

When I Override a Grade

As instructor of record, I am responsible for final grades, and I reserve the right to change a claimed grade. I expect to use it very rarely; almost every student will never need to think about it. It applies in two situations.

A claim not made in good faith. For instance: claiming a high grade after attending fewer than half the class sessions, without doing the assignments adequately, or without completing one of the course's foundations, such as the Worm Journal or the research project.

Circumstances beyond what a grade can fairly reflect. A serious illness or family emergency that disrupts a large part of the semester may call for a remedy available under university policy,

such as an incomplete (G grade) or a withdrawal. In those cases we will work out the right path together, with your academic advisor or the Dean of Students office.

An override for lack of good faith will never come as a surprise. Before it happens, there will be a documented record of concern: written feedback over the course of the semester telling you that your effort is falling short and what you need to do about it. If you have received no such warning, your claimed grade stands.

Absences and Missed Work

Please do not email me about absences, missed assignments, or extensions: this system already accounts for them, and there are no penalties to negotiate. If you miss class, catch up on what you missed and keep writing in your journal. How you managed the semester is something to reflect on there and to account for in your final report.

Emails about course content, ideas, and your research project are always welcome.

Frequently Asked Questions

Can I just give myself an A?
If an A is the grade you can honestly defend, yes.

How do I know what grade to claim?
Read the learning outcomes, look at your completion record and your journal, and pay attention to the written feedback you receive along the way. Then make the call.

Will I be compared to the rest of the class?
No. Your completion record describes your own work, not where you stand relative to anyone else.

Is the Worm Journal graded?
No. Nothing in this course receives a grade, not even the capstone assignments; your work gets feedback instead. The journal is your main evidence, and our main window into your learning.

Can I keep my journal on a computer?
No. Ink on paper. That is the point.

Are the TAs grading my work?
No. They give feedback, never grades.

Research Projects

There are not many courses where you can study whatever you want, however you want. This one comes close. The centerpiece of the course is a research project on a topic driven entirely by your own interests, and you will begin applying what you learn to it almost immediately. So think creatively, and think outside the box: what have you always wanted to know about a text, a body of work, or a corner of the past, that a computer might help you see?

What Most Projects Have in Common

None of these are hard rules, but most projects share a few features.

A text, or a body of texts, free of copyright restrictions. It could be a thousand pages. It could be one page. It depends on the material and on the question. (Copyright is trickier than it looks; see Copyright before you settle on a source.)

A focus on text. Much of this course is about modeling texts: What is a text? What are its features? What can we know about it, and how can computers help us understand it better? So most projects are text-centric in one way or another. That still leaves a lot of room. An exhibition catalogue for a show of paintings is a text. So is a set of song lyrics, a collection of letters, or a run of newspaper advertisements.

Non-English texts are strongly encouraged. Not everyone on a team needs to know the language or be able to read the script. Different people can do different parts of a project, and in this course that happens all the time: some team members are drawn to reading the text itself and the scholarship about it, while others gravitate toward the coding. Both are essential.

Beyond that, there is not much to say in general terms, because what you do will depend on your own interests and your own disciplines.

Teams

Working in a team is strongly encouraged, though not required. The sweet spot is three students, perhaps four. Five gets hard to coordinate; two is small. The expectations for what a good project looks like do not change with the size of the team, so there is a built-in incentive to aim for the sweet spot.

Mentors

Project mentor. Every project team has a mentor from the instructional team: either me or one of the undergraduate TAs. All projects meet weekly with their mentor. Many recitation sessions are dedicated to these meetings, so do your very best to make sure that everyone on your team attends the same recitation (see Recitations).

Subject-matter mentor. We strongly encourage, but do not require, each team to find a faculty member who works on something close to your project's topic. This person may know nothing about coding or AI, and that's fine: you are going to them for high-level, field-specific advice, above all about what makes an interesting research question in their field, regardless of method. Consider someone whose class you enjoyed, or someone you have worked with before; we are also happy to suggest names. These are busy people, so expect feedback once or twice in the semester, not weekly.

How a Project Unfolds

Every project is different, but most follow the flow of the semester.

Getting started. The first few weeks are about settling on a team, a regular meeting time, a general topic, and a text.

Analyzing and modeling the document. Every document contains a nearly infinite amount of information, if you think about it. What kind of paper is it on? What genre is it? What ink was used? What is its rhyme scheme? Who is mentioned, and where? You can model almost anything, so a big part of this course is thinking carefully about what you are actually interested in, because the way you model a text is inseparable from your research question. In coding terms, this is the work of designing a *schema*. Expect to spend much of the first half of the semester here.

Encoding. Next comes marking up your text according to your model. Here projects begin to diverge. Some spend a lot of their time working out ways to tag and model their text automatically, with tools like regular expressions.

Processing and publishing. In the later part of the semester, projects diverge further. Many produce some kind of digital edition, so that readers can encounter the same text you find compelling, enriched with everything you have encoded in it. Others focus on graphics that summarize what you have learned about the text: anything from charts and graphs to network analysis to digital mapping.

Throughout the semester there will be check-ins on your project: sometimes a submission, sometimes a journal assignment. See the syllabus for these.

Different Ways In

Projects from earlier semesters show just how many directions a project can take. A few examples:

- **Language itself:** Shakespeare's Neologisms, or Sevdah, an etymological study of Bosnian song and poetry.
- **Representation and theme:** Gender in the Iliad, Women in Folklore, and Motherhood and Power through Greek Mythology.
- **Documents of power and persuasion:** Constitutions of the Former Soviet Union, Presidential Propaganda, and Soviet Textbooks.

- **A single author or work:** Literary Analysis Engine for Oscar Wilde, and Perceptions of the Black Death in the Decameron.

For the full list, see Past Projects.

Your Project and Your Grade

You need to be able to explain the code behind your project: what it does, why it is there, and how you know it works. Comment it as you go, including where AI wrote it and what you checked (see AI Policies for an AI Course). At the end of the semester, your final report accounts for your project, including who on the team did what (see (Un)Grading).

University-Required Syllabus Policies

The University of Pittsburgh's required syllabus statements apply to this course and are part of this syllabus. They are available at https://teaching.pitt.edu/resources/syllabus-checklist/.

Appendix: AI Policies for an AI Course

Many AI policies are about what *not* to do. This course, by contrast, features AI as a core research tool: how it works, how it is changing digital research in the humanities, and above all how to use it to advance scholarship. Learning to use AI well is one of the main things you will take away from this course.

(A note on terms: on this page and in this course, "AI" is shorthand for generative AI, and in practice mostly for large language models, or LLMs, such as Claude, GPT, and Gemini: systems that produce text and code in response to prompts.)

Where This Course Comes From

This course has roots that go back well before generative AI existed. It descends from a course that David Birnbaum developed at Pitt, and many of its exercises, designed long before anyone could ask a chatbot for help, remain essential.

But the changes AI has brought to coding since then are profound. Most professional programmers now work with AI assistance. Some of what the original course asked of students, such as debugging long, complex programs by hand, is now something few working coders do unaided.

Thus, AI introduces exciting new possibilities for a course such as this, while also introducing a genuine pedagogical puzzle. You cannot direct a tool well, or tell when it has gone wrong, unless you understand the logic of what you are asking it to do. But the entry-level work through which people used to acquire that understanding, writing your first function and hunting down your own typo, is precisely the work AI now does for you. Economists have a name for this: the **missing junior loop**. Automate the apprentice's tasks and you cut the pipeline that produces the people who can judge the results (Catalini, Hui, and Wu 2026). Part of our task for the semester is to try to close that loop.

In the words of Ian Malcom in Michael Crichton's *Jurassic Park*:

> You know what's wrong with scientific power? It's a form of inherited wealth. Most kinds of power require a substantial sacrifice by whoever wants the power. There is an apprenticeship, a discipline lasting many years. Whatever it is you seek, you have to put in the time, the practice, the effort. You must give up a lot to get it. It has to be very important to you. And once you have attained it, it is your power. It can't be given away: it resides in you. It is literally the result of your discipline. But scientific power is like inherited wealth: attained without discipline. You read what others have done, and you take the next step. There is no discipline, no mastery. There is no humility before nature.

He meant it as an indictment of scientists who take the next step without the discipline that earned the previous one. So, as we wield this new, unearned technology, let's be careful not to let the dinosaurs out…

Contextual AI Usage

This course's stance toward AI is therefore contextual. In many parts of this course you are **expected to use AI, and to use it well.** In other parts **we will not use it at all.** That is not because AI is good or bad. It is because understanding certain things without it will let you use it much better.

If you can (or cannot) use AI on a given assignment or exercise, that is by design. The default is that AI is allowed. When it is not, you will know: either the exercise is handwritten, so AI is out of the picture anyway, or we will say plainly that we are asking you not to use it. (Pitt's Teaching Center describes a range of approaches to generative AI in courses; this course uses several of them, assignment by assignment.)

Pen and Paper

Counterintuitively, then, this course includes short in-class exercises done by hand, without a computer, let alone AI, on core concepts in coding and logic: What is a function? How does a computer think about paths? What is a for loop? The goal is to recognize each concept when you meet it, and, yes, to ask AI to implement it for you in other contexts.

A quite reasonable reaction might be: "I will never have to do this in Real Life, so why on earth are we being tormented this way?" The calculator rendered long division obsolete, and yet you still memorized your times tables: the calculator changed the purpose of that work, making it a means to mathematical reasoning rather than an end in itself. Coding is no different. If you want to tell AI what good code should do, rather than the other way around, you need to spend some time alone with pen, paper, and your own intellect. There is research behind this: among novice programmers, the ability to trace what a piece of code does, and to explain it in plain English, goes hand in hand with the ability to write it (Lister, Fidge, and Teague 2009). When AI does the writing, tracing and explaining are the skills you have left, and they are what tell you whether the writing is any good.

Responsibility

When you submit a piece of code, or use it in your project, **you are responsible for it**, even if AI did most of the actual coding. You answer for what it does, what it gets wrong, and what it breaks. Pitt's own standard for AI use makes the same point: you are accountable for anything you produce with AI, and for verifying its accuracy.

In this course, that means you should be able to read the code you use, understand it, and explain it: what each part does, why it is there, and how you know it works.

Does that mean sight-reading every bit of syntax? Not necessarily. Ironically, AI is very good at helping you decipher code, and we will teach you to do that better; and the stronger your

independent checks on a result, the less rides on your reading of every nut and bolt. But you do need to understand the code you use at the level of its logic. Call it coding literacy.

Test the Output, Not the Story

Understanding the code is half the job. The other half is testing what it produces, rather than taking the AI's word for it. This is easier said than done, and it will be a major focus of the course. One rule holds throughout: **never accept the model's own report of what it did as verification.** Asking an AI "did that work?" and believing the answer is not checking. Professional developers have learned this the hard way: in the same 2025 survey, more developers said they distrust the accuracy of AI tools than said they trust it (Stack Overflow 2025).

To put it pithily: **if it feels like magic when you get a cool result, that is not an acceptable outcome for this course.** Magic is what a result looks like when you cannot say how it happened, or how you would know if it were wrong.

Saying What You Did

There is no separate paperwork for AI use in this course. There are two habits instead.

In your code. Where AI wrote or substantially reworked something you hand in, say so in a comment, along with what you checked and what you changed. Every coding assignment requires real comments anyway (see Course Format and Materials); this is part of what they are for.

In your journal. Your Worm Journal is where you work out what you are learning: by hand, in ink, with no AI in the loop. When AI was involved, the questions worth sitting with are the ones that are easiest to skip in the moment. What were you actually trying to do? Where did the model turn out to be wrong, and how did you find out? What did you change, and why? What do you still not understand?

Together they show us your understanding and your learning, not just your output, which is why output is not what this course grades (see (Un)Grading).

Tools

Everything AI-related in this course can be done with tools that cost you nothing:

- **GitHub Copilot in VS Code,** for working directly with code. Copilot Free comes with VS Code, and verified students get the more capable Copilot Student plan at no cost through GitHub Education (GitHub Copilot plans). Verification can take a while, so apply in the first week (see Getting Set Up).
- **Claude for Education,** Pitt's university-provided Claude account, for a chat window and a second model to check the first against.

What you feed into these tools, especially copyrighted texts and archival images, raises questions of its own, covered on the course's Copyright page.

How These Principles Apply to the Instructor

I hold myself to the same standards.

- **Grading and feedback:** I will never, ever use AI for grading, or for feedback on your work. Not as a first pass, not as a suggestion, not at all. The same goes for the TAs.
- **Your work:** I will never put student work into any tool, AI or otherwise, that Pitt has not approved for use with University data.
- **Course materials:** I use AI in building this course, including some example code and some of the AI output you will be asked to critique. When the origin of something matters to an exercise, I will tell you.
- **Everywhere else:** I use AI to design websites like this one, and constantly in my own research and work.

Appendix: Getting Set Up

Do all of this in the first week of the semester. Some steps take a few days to go through, so don't leave them for later.

A Notebook and a Pen

For your Worm Journal: a paper notebook of your own, and a pen that writes in ink.

Scanning Your Journal

Periodically you will scan pages of your Worm Journal and submit them to Canvas as a PDF. Your phone can do this; you do not need a scanner.

A Computer

VS Code

GitHub and Copilot

Claude for Education

oXygen XML Editor

Running XSLT

Python

Appendix: Copyright

Your project will probably end up on the open web, and that makes copyright a real question for this course, not a technicality. This page sets out general principles and rules of thumb. It is a starting point, not an answer.

First, a Disclaimer

None of us are lawyers. Not me, not the TAs. Copyright is ultimately a legal question, and when you really get down to it, it is a question of risk: how likely is it that someone will object to what you have done, and that a court would side with them? Nobody on the instructional team can guarantee you anything.

What we can do is offer guidance, and so can the University Library System, whose librarians can help you understand your options and point you to resources (though, as their own guide notes, they do not give legal advice either). If a question really matters, ask early: the earlier you settle on a source, the cheaper it is to change your mind.

The Public Domain

A work in the **public domain** is no longer protected by copyright (or never was), and anyone may use it, in full, without permission. Most projects in this course work with public domain texts, and that is the safest place to be.

In the United States, as of January 1, 2027:

- **Published before 1932:** anything first published in the United States in 1931 or earlier is in the public domain. Each January 1, another year joins it.
- **Published 1932 to 1963:** many of these works are also in the public domain, because their copyright had to be renewed after 28 years and often was not. Finding out takes some research.
- **Published 1964 to 1977:** most of these are still under copyright, for 95 years from publication.
- **Published 1978 or later:** protected for the life of the author plus 70 years, so assume they are under copyright.
- **Never published** (letters, diaries, manuscripts): generally protected for the life of the author plus 70 years, so those by authors who died in 1956 or earlier are in the public domain.
- **Works of the US federal government** (but not necessarily state governments) are in the public domain no matter when they were made.

The standard reference for all of this, including the many special cases, is Peter Hirtle's chart, Copyright Term and the Public Domain in the United States (Cornell University Library).

Works from other countries are more complicated. US rules govern what you do in the United States, but a foreign work's status here can depend on its status in its home country. The good

news for many humanities projects: a medieval manuscript, a classical text, or a nineteenth-century novel is in the public domain everywhere.

The Version Matters

This is the most common trap. **An old text can be in the public domain while the version you are reading is not.**

- **Translations** are new works with their own copyright. Homer is in the public domain; a translation of Homer published in 1990 is not. A translation published before 1932 is. (Or make your own, which is a fine project in itself.)
- **Modern editions** can carry new copyright in everything the editor added: introductions, notes, critical apparatus, and sometimes editorial choices about the text itself.
- **Reprints** do not reset the clock on the original text, but check what else the reprint added.

So always record exactly which edition or translation you are using, and when it was published.

Images and Digitized Sources

In the United States, a faithful photograph or scan of a two-dimensional public domain work (a painting, a page of a manuscript) does not get a new copyright of its own: a federal court held that such exact reproductions lack the originality copyright requires (*Bridgeman Art Library v. Corel*, 1999). That decision is widely followed, but it does not bind every court, and other countries see things differently.

Separately, the library, museum, or website you get an image or text from may have **terms of use** that you agreed to by downloading it. Those are a contract, not copyright, but they still matter. Read them, and when in doubt, prefer sources that say plainly what you may do.

Analyzing Is Not the Same as Publishing

Studying a copyrighted text computationally is one thing; posting it on the web is another.

US courts have held that copying copyrighted books into a database so they can be searched and analyzed, rather than read, can be a fair use (*Authors Guild v. HathiTrust*, 2014, and *Authors Guild v. Google*, 2015). This kind of computational analysis is sometimes called **non-consumptive** research. So a project might be able to analyze copyrighted song lyrics and publish its *findings*: counts, patterns, charts, short quotations for discussion. What it should not do is republish the lyrics in full on its website.

Fair use is decided case by case, weighing four factors: the purpose of the use, the nature of the work, how much of it is used, and the effect on the market for the original. Pitt's Copyright and Intellectual Property Toolkit explains each. Note too that some databases and websites forbid automated downloading in their terms of use, whatever fair use might allow.

AI Tools and Your Sources

Uploading a text into an AI tool makes a copy of it. For public domain texts this raises no copyright question. For copyrighted material, the law around AI is still being worked out, so be more cautious: use the course's university-provided tools where you can, and do not upload copyrighted material you would not be comfortable sharing in any other way. (See AI Policies for an AI Course.)

Rules of Thumb

1. **Prefer public domain or openly licensed sources** (such as material under a Creative Commons license that permits reuse). Read the license: some forbid commercial use or modification.
2. **Know your version.** Record the edition or translation, its publisher, and its date.
3. **Publish results, not copyrighted texts.** Analysis, data derived from a text, and brief quotations are far safer than full reproduction.
4. **Respect terms of use** of the archives and websites you draw on.
5. **Always attribute.** Attribution is not permission, and permission is not attribution. You need to cite your sources either way.
6. **Ask early.** Bring copyright questions to your project mentor at the start, and to the library when you need more help: Pitt's Digital Scholarship Services can be reached at ULS-DigitalScholarshipServices@groups.pitt.edu.

Further Reading

- Copyright and Intellectual Property Toolkit, University Library System
- Copyright Term and the Public Domain in the United States, Cornell University Library
- Text and Data Mining under U.S. Copyright Law, Authors Alliance (2024)

Appendix: Past Projects

Research projects by students in earlier semesters of this course. For guidelines and advice on your own, see Research Projects.

Spring 2026

- Mapping Ancient Iran
- Soviet Textbooks
- Motherhood and Power through Greek Mythology
- Perceptions of the Black Death in the Decameron
- Literary Analysis Engine for Oscar Wilde

Fall 2025

- Shakespeare's Neologisms
- Gender in the Iliad
- Feminist Media Throughout History
- Presidential Propaganda

Fall 2024

- Constitutional Thinkers
- Representation in Shakespeare: Language and Appearances

Spring 2024

- Constitutions of the Former Soviet Union
- Sevdah: an etymological study of Bosnian song and poetry
- Women in Folklore

Earlier Semesters

- Projects hosted on Obdurodon (Pitt Oakland)
- Projects hosted on Newtfire