CSE216
Programming Abstraction

Instructor: Zhoulai Fu

State University of New York, Korea

Course materials and Info available here:
https://github.com/zhoulaifu/23_cse216_spring

Some slides are taken from SBU. Thanks!
Some writings and code are generated by ChatGDP. Wow :-)

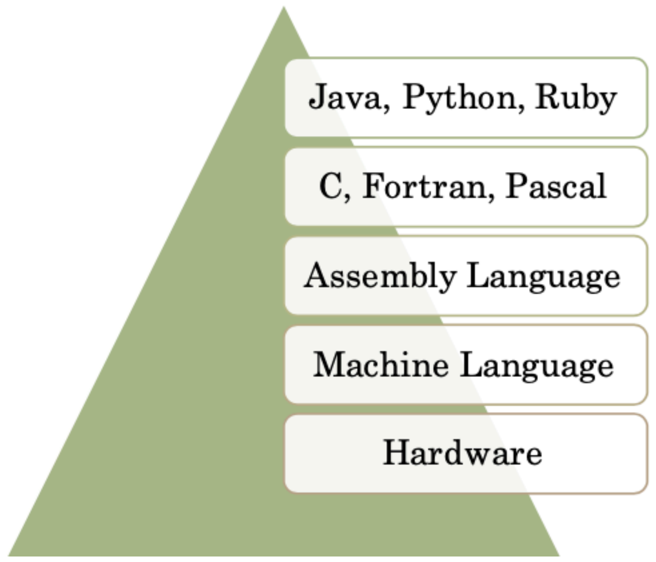
What is Programming Abstraction

- Programming abstraction is a technique used in computer programming **to simplify the complexity** of writing and understanding code. It involves creating higher-level abstractions that **hide low-level implementation details**, allowing programmers to focus on solving problems at a higher level of abstraction.

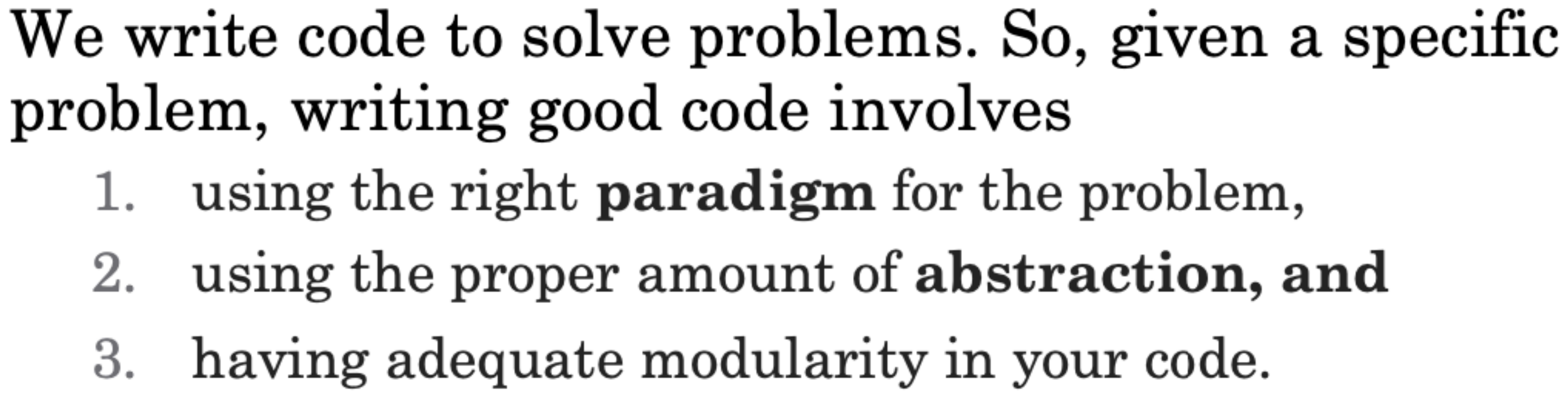
- Abstractions in programming can take many forms, including **data structures, functions, and classes.** For example, a programmer can use a class in object-oriented programming to group together related functions and data, creating a higher-level abstraction that makes it easier to work with the program.

Abstraction in Language Evolution

- Most programming languages are *high-level* languages, where the phrase "high-level" indicates a higher degree of abstraction.
- The second word of this course's name – **abstraction** – is among the most important ideas in programming.
- It refers to the degree to which the language's features are separated away from the details of a particular computer's architecture and/or implementation at *lower* levels.



Why abstraction?


We write code to solve problems. So, given a specific problem, writing good code involves

1. using the right **paradigm** for the problem,
2. using the proper amount of **abstraction, and**
3. having adequate modularity in your code.

- Programming abstraction is essential in software engineering because it allows programmers to manage the complexity of large software projects. By providing high-level abstractions, it makes it easier for developers to work collaboratively, and it allows for easier maintenance and testing of software code.

This course

- In this course, we work with three programming languages: **OCaml, Java, and Python**. However, the course **does not solely focus on these individual programming languages**. If you approach the course with a narrow focus on the syntax of each language, you may find it more challenging than necessary.

- Instead, the course **emphasizes the underlying concepts that are common to all programming languages.** We examine the programming paradigms that have emerged. Each paradigm has its own strengths and weaknesses, and our goal is to **gain a deep understanding of the various ways of thinking about programming**. This will enable us to determine, based on a given scenario, which language and paradigm to use to write efficient and effective code.

Course outcomes

An understanding of programming paradigms and tradeoffs.

An understanding of functional techniques to identify, formulate, and solve problems.

An ability to apply techniques of object-oriented programming in the context of software development.

Meet the Instructor

Education

- B.Sc, M.Sc, Ecole Polytechnique, France
- M.Eng. Telecom Paris, France
- Ph.D. INRIA (National CS Lab), France

Teaching & Research

- University of California Davis, United States
- IT University of Copenhagen, Denmark
- SUNY Korea

TA

- Unknown 1
- Unknown 2

Team

You	TA	Instructor	ChatGPT
Lectures	Office hours	Office hours	Not do homework
Homework	Answer questions	Lectures	Answer questions
Ask questions		Grading	
		Answer questions	

Practical matters

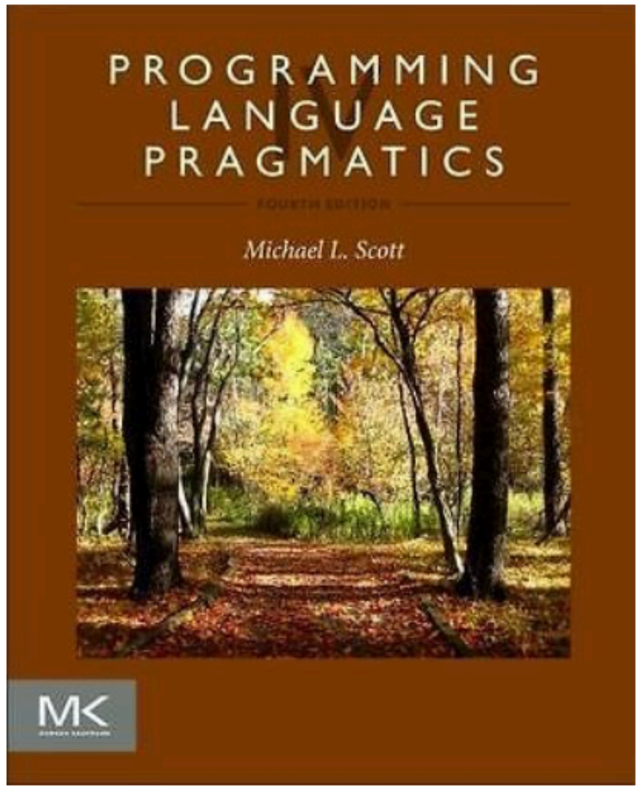
- COVID
- Reference books
- Schedule
- Exams and grading

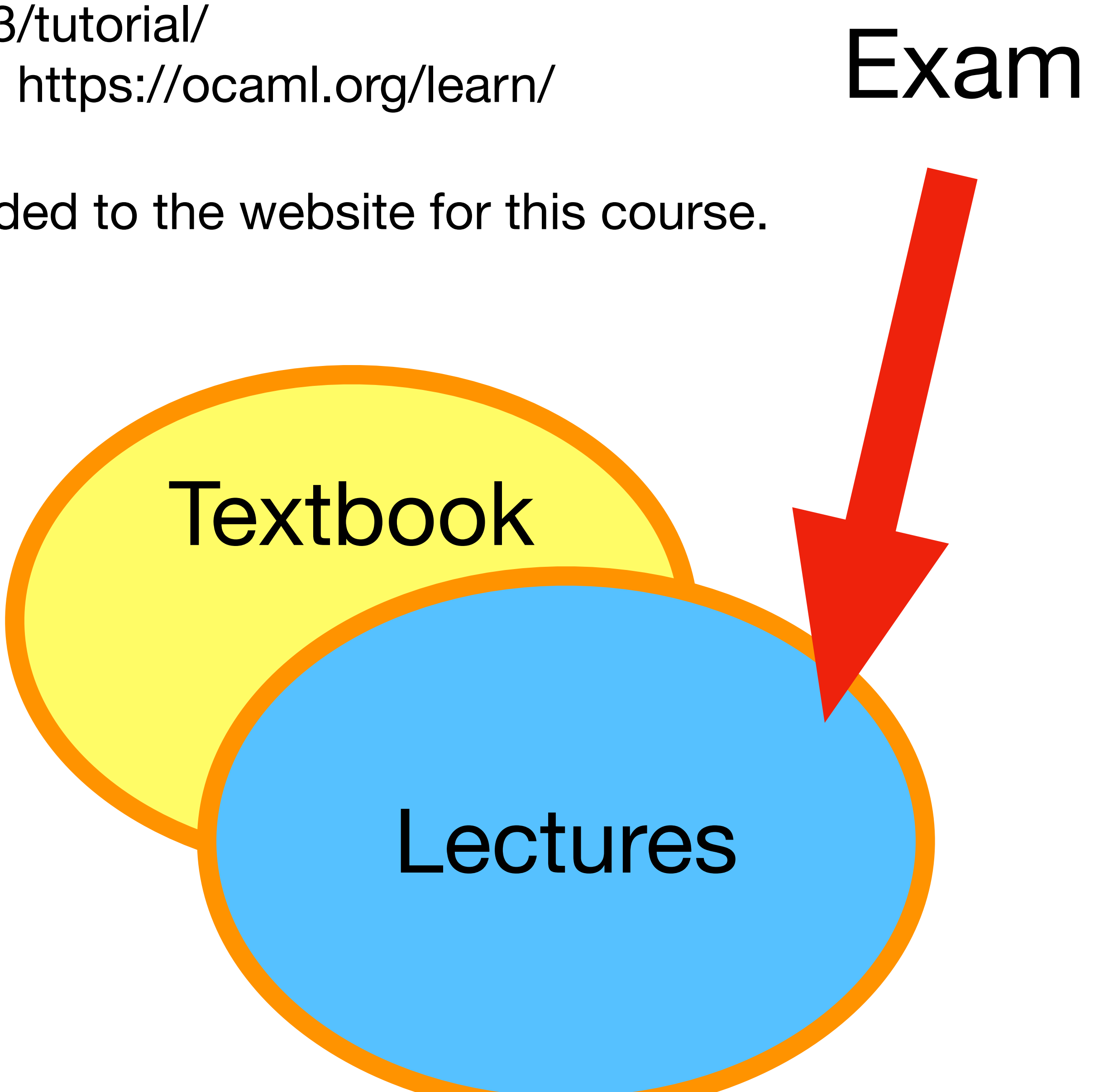
Covid is gone, but

- Inform instructor immediately of the date of a positive test. Your absences will be excused
- Follow government guidelines including a 7-day quarantine.
- Return to the class after quarantine. Negative test not needed.

Reference books and reading material

- Michael L. Scott. Programming Language Pragmatics.
- For details pertaining to specific programming languages, the recommended material will mostly be from the following:
  Python tutorial: https://docs.python.org/3/tutorial/
  The official OCaml learning material from https://ocaml.org/learn/
- Other reading material (if used) will be added to the website for this course.

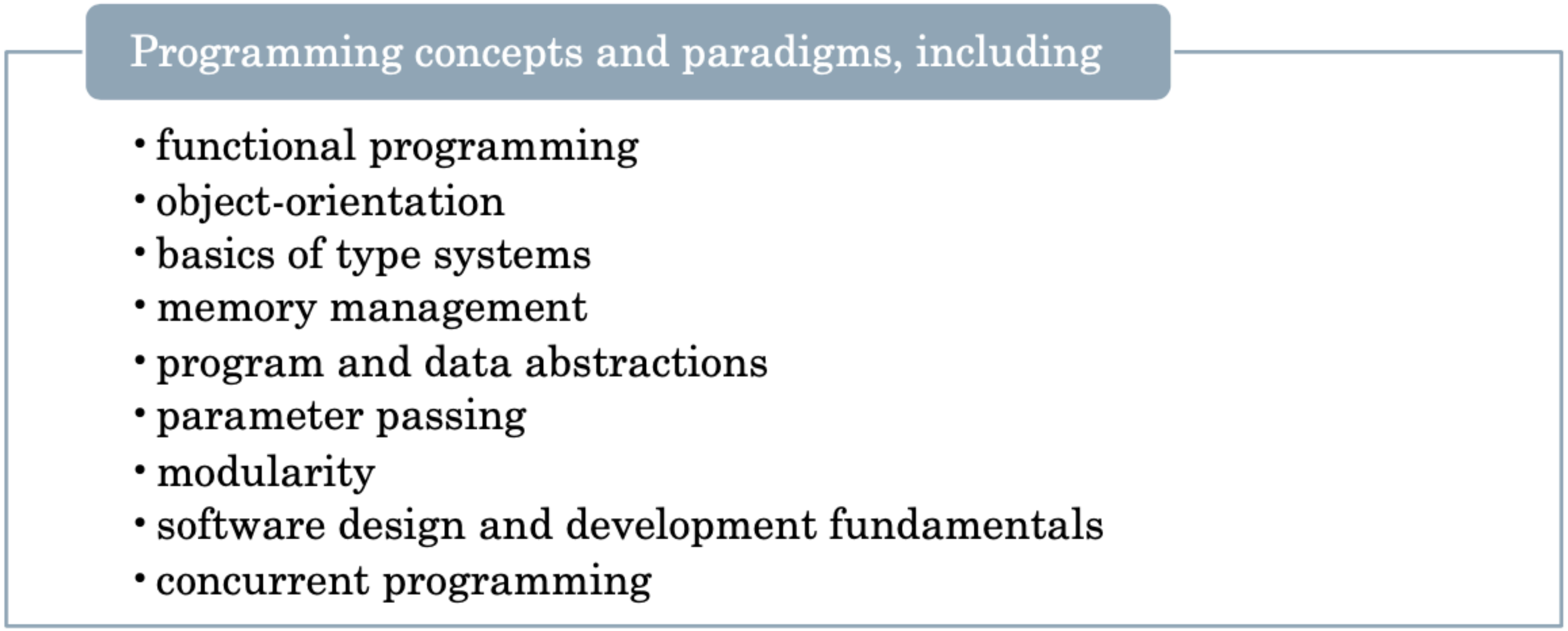




Schedule

- Lectures: Monday and Wednesday 2:00 - 3:20pm, at B203
- Recitation: Wednesday 3:30 - 4:25pm, at B203
- Office hours: Monday 5:00 - 6:00pm and Wednesday 8:00 - 9:00pm, at B424
- TA office hours: TBA
- course website: https://github.com/zhoulaifu/23_cse216_spring

Course outline



Programming concepts and paradigms, including

- functional programming
- object-orientation
- basics of type systems
- memory management
- program and data abstractions
- parameter passing
- modularity
- software design and development fundamentals
- concurrent programming

Grading

- Attendance: 5%
- take-home assignment (homework): 25%
- In-class assignments (quiz): 10%
- Midterms: 30%
- Final exam: 30%
- Students with regular participation get 1% bonus

Cont.

- In-class assignments take place in recitation classes (Wednesday). Format: Paper-based, open-notes. No Internet.
- In-class assignments are basic exercises.
- Take-home assignments take place at the end of each “chapter”, usually on Thursday
- Take-home assignments are harder and may need reflection, but you can ask for help.
- Exam format = In-class assignment format.

Questions so far?