some things to know about

https://listserv.pomona.edu/scripts/wa.exe?SUBED1=CSCOLLOQ&A=1

https://listserv.pomona.edu/scripts/wa.exe?SUBED1=CSALL&A=1

csci54 – discrete math & functional programming introduction (to everything)

discrete math & functional programming

- discrete math is concerned with structures that can be counted
- functional programming requires solving problems by just applying functions to arguments, rather than by updating state.
- “thinking functionally" is very similar to "thinking mathematically

```
maxInt [x] = x
maxInt (x:xs) = max x (maxInt xs)
```

Calendar, course website

https://cs.pomona.edu/classes/cs54

structure of the class

- basic weekly structure:
  - Monday: lecture + quiz
  - Wednesday: lecture + quiz + test (no test today)
  - Wednesday/Thursday: small group meetings
  - Thursday night: turn in small group assignment
  - Sunday night: weekly problem set due
- a low-stakes quiz or test each lecture day
- an in-class final during week 15

small groups

- groups of ~4-6 that meet with an assigned TA for an hour once a week on either Thursday or Friday.
  - more on this, including first-pass on group formation, later this lecture.
- weekly low-stakes small group assignment on Gradescope
- due Thursday night
- no extensions (make sure your group knows who's turning in!)
- first group assignment due tomorrow

weekly assignments

- available on gradescope
- due in gradescope Sunday night
- coding and/or written
- will initially be done in groups of 2 with all members from the same small group. about halfway through the semester will add the option to work alone.
- if you need an extension let me know
- first problem set due this Sunday (done individually)

grading

- If you don’t like your grade on a homework or test…
  - Or if you have lingering questions…
- Meet with me or a TA to talk through what you tried!
  - “Oral retakes” like this can regain points
  - Within two weeks of the test!
- Even if you didn’t finish the assignment…
  - It’s worth talking through so you can stay on pace

Some rules and regulations

- don't look for solutions to problem set questions on the web (including ChatGPT or copilot), from students who have taken the class previously, or other sources not specifically distributed for this class this semester.
- don't share (non-publicly-available) materials from the class with anyone not in the class this semester
- but please discuss concepts with other people involved with the class this semester: me, the TAs, other students!
  - do not look at other people's (working) code
  - write code/proofs/etc on your own: do not, for example, take a photo of code/notes on the whiteboard. If you understand the idea, you should be able to write the code/proof/etc on your own without referring to notes.

resources

- Course webpage: syllabus, calendar, logistics, resources, problem sets, lecture notes
- Gradescope: for turning in assignments
- Slack: discussion
  - for questions/comments potentially of interest to the entire class and that do not contain private/confidential information
  - email me directly about things specific to you (e.g. grades)
- readings:
  - calendar at https://cs.pomona.edu/classes/cs54 includes relevant chapter/sections

resources – readings

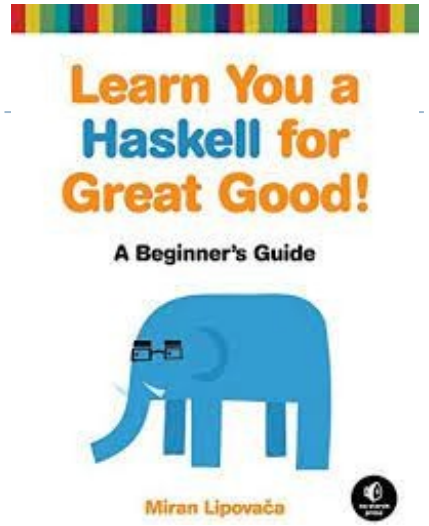
- "Learn you a Haskell for Great Good!" by Miran Lipovača and some GitHub contributors
  - https://learnyouahaskell.github.io

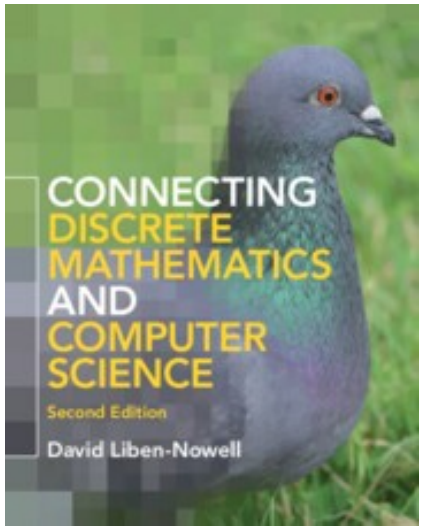
"Connecting Discrete Mathematics and Computer Science" by David Liben-Nowell

- https://cs.carleton.edu/faculty/dln/book/

Some other books, videos, etc listed on course webpage

- Let us know if you run across anything you find helpful for learning the material!





resources

- me (Prof. Osborn)
  - Edmunds 113
  - office hours: Mo/Fr 9:30-11:30 am
- TAs:
  - Lenny Raybukh
  - Emmett Levine
  - Harrison Brown
  - Emma Gandonou
  - Ruben Millan Fabian
- QSC: pomona.mywconline.com, qsc@pomona.edu

small groups

- ~4-6 students in each group
  - assigned using results of survey (we'll do this in a minute)
  - same group for entire semester, responsible for choosing a 1-hour weekly meeting time on Wednesday or Thursday
  - first meeting tomorrow

  Email me directly if there is a problem with your group assignment (besides "I was hoping to team with so-and-so") and I will try to fix any issues discreetly

grading

- 30% in-class tests
- 10% in-class quizzes
- 20% problem sets
- 10% group work
- 30% final exam

- Questions?

- First quiz!

Introduction to Haskell – getting set up

- Problem 2 on week01-group (due Thursday) asks each of you to set up Haskell on whatever machine you plan on doing your assignments on.
- The week01-ps asks you to write a few functions in Haskell and for each of you to turn it in (individually) on gradescope.

> - This assignment must be turned in individually on Gradescope. However **for this assignment only** you may collaborate as much as you want with the other students in your small group.
> - **If you ever have any questions about what is an acceptable amount of collaboration (or an acceptable use of ChatGPT or similar programs!), you must discuss with the professor in advance!!!**

Introduction to Haskell – getting set up

- Problem 2 on week01-group (due Thursday) asks each of you to set up Haskell on whatever machine you plan on doing your assignments on.
- The week01-ps asks you to write a few functions in Haskell and for each of you to turn it in (individually) on gradescope.
- Start the interpreter with ghci -W for more help getting code right
- Two ways of interacting with Haskell in this class:
  - type commands in the interpreter
  - edit a file and run in the interpreter

Haskell basics

- Some things will feel familiar:

```
ghci> 2 + 15
ghci> 49 * 100
ghci> 1892 - 1472
ghci> 5 / 2
ghci> True && False
ghci> True && True
ghci> False || True
ghci> not False
ghci> not (True && True)
ghci> 1 == 0
ghci> 5 /= 5
```

Haskell basics

- Some things may feel familiar-ish but are worth thinking about a little more

- Defining functions

```
ghci> add1 x = x + 1
ghci> addxy x y = x + y
```

- Conditionals

```
ghci> cap n = (if n > 100 then 100 else n)
```

Introduction to Haskell – getting set up

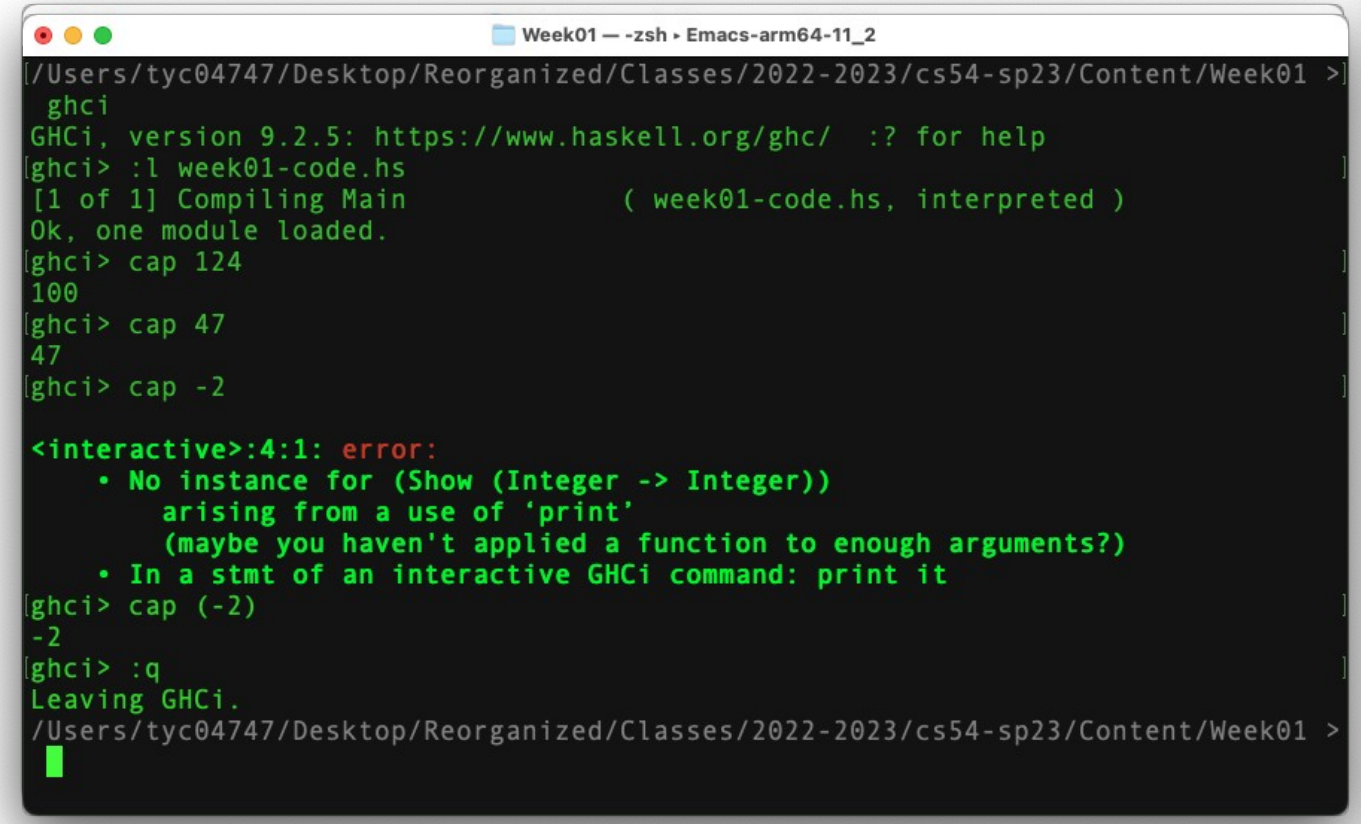
- Problem 2 on week01-group (due Thursday) asks each of you to set up Haskell on whatever machine you plan on doing your assignments on.
- The week01-ps asks you to write a few functions in Haskell and for each of you to turn it in (individually) on gradescope.

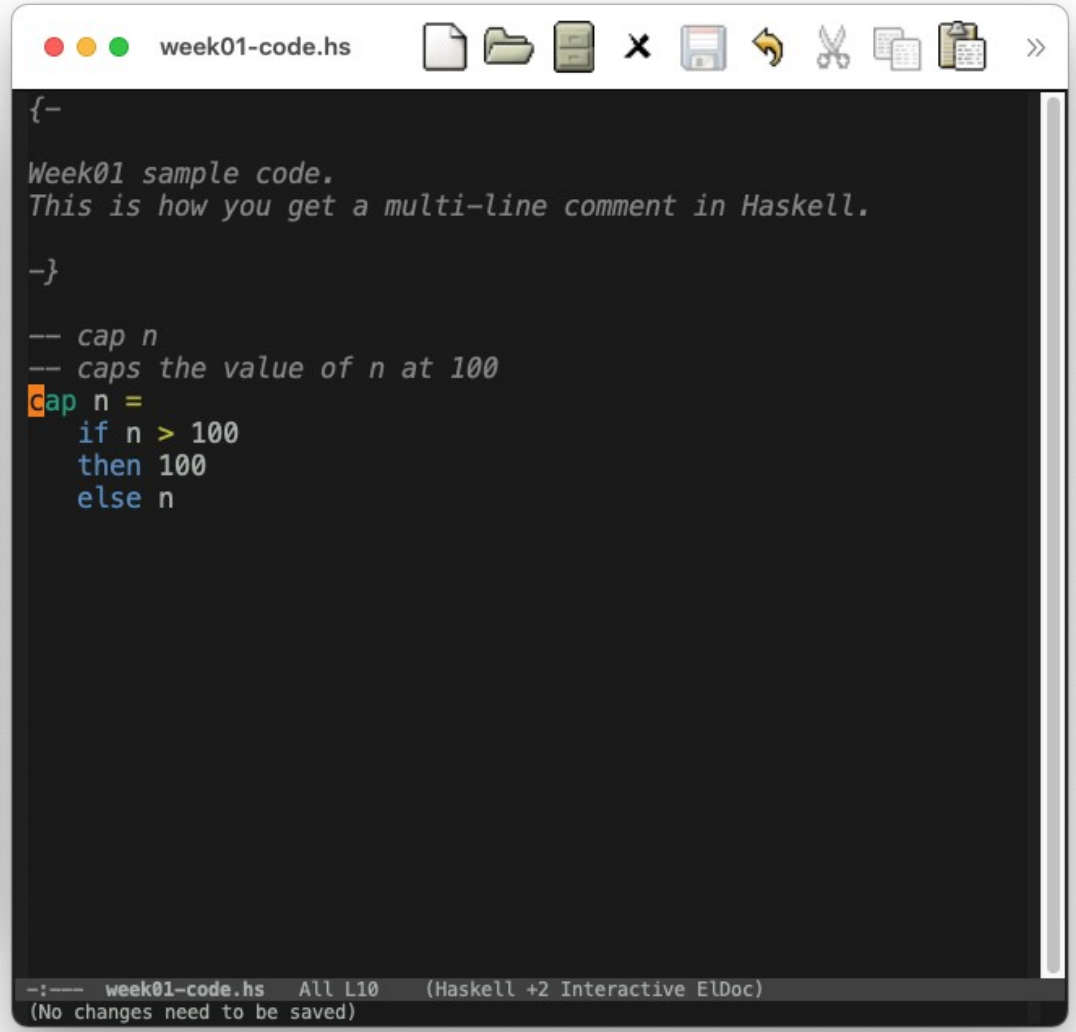
- Interacting with Haskell:
  - type commands in the interpreter
  - edit a file and run in the interpreter

- For the week01-ps assignment due Sunday:
  - Haskell
  - editor for writing/modifying code: VSCode, emacs

Running Haskell code

- sample interaction






```
ghci>  :l <filename>
ghci>  :q
```

Practice questions

```
cap n =
   if n > 100
   then 100
   else n
```

- Write a function cap' that not only caps the upper limit at 100, but additionally evaluates to 0 if n is less then or equal to 0.

- Write a function pow that takes two parameters n and k and returns n to the kth power. (assume that k is guaranteed to be a non-negative integer. do not use the ** operator)

Lists

- We'll use **lists** extensively in this class, and will spend most of next Monday talking about them
- For now, these are the essentials:
- A list is a kind of "pair" structure, combining the front of the list (the "head") and the remainder of the list (the "tail")
  - You can think of it like a snake---"a snake is just the head of the snake plus a littler snake attached to it"
  - OK, maybe a weird metaphor

Lists

- There is one “special” list called *nil* or *empty list*: []
- Every other list is built using :, pronounced *cons*
  - Cons is just that “head plus rest” construction
- 1:[] is a one-element list containing the number 1
- True:(False:[]) is a two-element list with the bools True and False, in that order.
- We can build up lists using `element : other_list`

List Patterns

- Constructors like [] and : are used to create values
  - Other constructors you've seen: True, 2, 5.0
- Each constructor has a corresponding *pattern* when a value of that type appears in a binding position:

- `opposite False = True`
- `opposite True = False`

```
count [] = 0
count (_elt:lst) = 1 + count lst
```

For this week, our "list recursion pattern" is exactly like count above.

Notes on Lists

- Lists in Haskell, unlike Python, *only contain one type of thing*
  - So a list of integers [Int] is a different type from [Char]
  - The type of : is a → [a] → [a]; in other words, it takes an object of some type and a list containing *that type of thing* and creates a new list of the same type.
- We can construct lists with square braces: [1, 2, 3]
  - But again: not [1, “hello”, False]
- [1,2,3] is a short-hand for 1:(2:(3:[]))