UC Berkeley EECS
Lecturer
Michael Ball

Wrap Up

Announcements

- All remaining assignments including Ants Due 12/5
- All can be submitted as w/a partner
- Everyone gets 3 homework + lab drops
- HW10, 11, 12 will be largely effort based.
  - If you get a 4/8, you'll get 8/8
- Slip Days basically do not matter. ☺
- Ants Updates:
  - Everyone gets Checkpoint 1 points for submitting the assignment.
  - Max Score = 41/54 – I'll scale the points
  - Phase 4 is optional, will become EC of some sort.
  - No early EC, but can get EC for doing some of Phase 4 + EC

Final Exam Information

- Final Exam format is slightly TBD
- Will be entirely **remote**
- We will use Zoom proctoring.
  - I will email everyone a Zoom link early finals week
- Will likely be multiple choice + short answer via Gradescope
  - SQL is in scope, Trees are in scope
  - These topics will be in less depth than normal.
  - think conceptual questions, self-checks, environment diagrams, pick the right line of code.
- Exam is typically 96 points over 3 hours.
  - I'll likely design a ~60-point exam (completable in 2 hours) but you'll have 3.
  - Closed internet, but open hand-written notes

Announcements

- Final Exam, Friday 8-11am slot
- http://howamidoing.c88c.org/
  - Not fully updated, but will be next week.
- Sum everything up! (Everything except lecture self-checks are in howamidoing.)
  - » Basically everyone is getting 20/20 on the lecture self-checks. That's the point. ☺
  - » Correctness *does* count! Resubmit them, if you haven't gotten them right. (No late penalty!) Lateness does not matter
  - » There are 27 self-checks which means you could have skipped 7 lectures
  - » There's a few "bonus" ones on Gradescope, just from different semesters that are optional practice.

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

SQL: CREATE and INSERT and UPDATE

CREATE TABLE

- SQL often used interactively
  - Result of select displayed to the user, but not stored
- Can create a table in many ways
  - Often may just supply a list of columns without data.
- Create table statement gives the result a name
  - Like a variable, but for a permanent object

```
CREATE TABLE [name] AS [select statement];
```

SQL: creating a named table

```
CREATE TABLE cones AS
    select 1 as ID, "strawberry" as Flavor, "pink" as Color,
3.55 as Price union
    select 2, "chocolate", "light brown", 4.75 union
    select 3, "chocolate", "dark brown", 5.25 union
    select 4, "strawberry", "pink",5.25 union
    select 5, "bubblegum", "pink",4.75 union
    select 6, "chocolate", "dark brown", 5.25;
```

Notice how column names are introduced and implicit later on.

Inserting new records (rows)

```
INSERT INTO table(column1, column2,...)
       VALUES (value1, value2,...);
```

```
sqlite> insert into cones(ID, Flavor, Color, Price) values (7, "Vanila", "White", 3.95);
sqlite> select * from cones;
ID|Flavor|Color|Price
1|strawberry|pink|3.55
2|chocolate|light brown|4.75
3|chocolate|dark brown|5.25
4|strawberry|pink|5.25
5|bubblegum|pink|4.75
6|chocolate|dark brown|5.25
7|Vanila|White|3.95
sqlite>
```

```
cones.append((7, "Vanila", "White", 3.95))
cones
```

ID	Flavor	Color	Price
1	strawberry	pink	3.55
2	chocolate	light brown	4.75
3	chocolate	dark brown	5.25
4	strawberry	pink	5.25
5	bubblegum	pink	4.75
6	chocolate	dark brown	5.25
7	Vanila	White	3.95

- A database table is typically a shared, durable repository shared by multiple applications

UPDATING new records (rows)

```
UPDATE table SET column1 = value1, column2 =
value2 [WHERE condition];
```

- If you don't specify a WHERE, you'll update all rows!

UC Berkeley EECS
Lecturer
Michael Ball

SQL: Aggregations

Groupinng and Aggregatations

- The `GROUP BY` clause is used to group rows returned by SELECT statement into a set of summary rows or groups based on values of columns or expressions.
- Apply an aggregate function, such as SUM, AVG, MIN, MAX or COUNT, to each group to output the summary information.

```
cones.group('Flavor')
```

Flavor	count
bubblegum	1
chocolate	3
strawberry	2

```
sqlite> select count(Price), Flavor from cones group by Flavor;
count(Price)|Flavor
1|bubblegum
2|chocolate
2|strawberry
```

```
cones.select(['Flavor', 'Price']).group('Flavor', np.mean)
```

Flavor	Price mean
bubblegum	4.75
chocolate	5.08333
strawberry	4.4

```
sqlite> select avg(Price), Flavor from cones group by Flavor;
avg(Price)|Flavor
4.75|bubblegum
5.0|chocolate
4.4|strawberry
```

UNIQUE / Distinct values

```
select DISTINCT [columns] from [table] where [condition] order by [order];
```

```
sqlite> select distinct Flavor, Color from cones;
strawberry|pink
chocolate|light brown
chocolate|dark brown
bubblegum|pink
sqlite>
```

```
In [8]: cones.groups(['Flavor', 'Color']).drop('count')
```

Out[8]:

Flavor	Color
bubblegum	pink
chocolate	dark brown
chocolate	light brown
strawberry	pink

```
In [7]: np.unique(cones['Flavor'])
Out[7]: array(['bubblegum', 'chocolate', 'strawberry'], dtype='<U10')
```

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

SQL: Joins

Joining tables

- Two tables are joined by a comma to yield all combinations of a row from each
  - `select * from sales, cones;`

```
create table sales as
    select "Baskin" as Cashier, 1 as TID union
    select "Baskin", 3 union
    select "Baskin", 4 union
    select "Robin", 2 union
    select "Robin", 5 union
    select "Robin", 6;
```

Cashier	TID
Baskin	1
Robin	2
Baskin	3
Baskin	4
Robin	5
Robin	6

```
sales.join('TID', cones, 'ID')
```

TID	Cashier	Flavor	Color	Price
1	Baskin	strawberry	pink	3.55
2	Robin	chocolate	light brown	4.75
3	Baskin	chocolate	dark brown	5.25
4	Baskin	strawberry	pink	5.25
5	Robin	bubblegum	pink	4.75
6	Robin	chocolate	dark brown	5.25

```
sqlite> select * from sales, cones;
Baskin|1|1|strawberry|pink|3.55
Baskin|1|2|chocolate|light brown|4.75
Baskin|1|3|chocolate|dark brown|5.25
Baskin|1|4|strawberry|pink|5.25
Baskin|1|5|bubblegum|pink|4.75
Baskin|1|6|chocolate|dark brown|5.25
Baskin|3|1|strawberry|pink|3.55
Baskin|3|2|chocolate|light brown|4.75
Baskin|3|3|chocolate|dark brown|5.25
Baskin|3|4|strawberry|pink|5.25
Baskin|3|5|bubblegum|pink|4.75
Baskin|3|6|chocolate|dark brown|5.25
Baskin|4|1|strawberry|pink|3.55
Baskin|4|2|chocolate|light brown|4.75
Baskin|4|3|chocolate|dark brown|5.25
Baskin|4|4|strawberry|pink|5.25
Baskin|4|5|bubblegum|pink|4.75
Baskin|4|6|chocolate|dark brown|5.25
Robin|2|1|strawberry|pink|3.55
Robin|2|2|chocolate|light brown|4.75
Robin|2|3|chocolate|dark brown|5.25
Robin|2|4|strawberry|pink|5.25
Robin|2|5|bubblegum|pink|4.75
Robin|2|6|chocolate|dark brown|5.25
Robin|5|1|strawberry|pink|3.55
Robin|5|2|chocolate|light brown|4.75
Robin|5|3|chocolate|dark brown|5.25
Robin|5|4|strawberry|pink|5.25
Robin|5|5|bubblegum|pink|4.75
Robin|5|6|chocolate|dark brown|5.25
Robin|6|1|strawberry|pink|3.55
Robin|6|2|chocolate|light brown|4.75
Robin|6|3|chocolate|dark brown|5.25
Robin|6|4|strawberry|pink|5.25
Robin|6|5|bubblegum|pink|4.75
Robin|6|6|chocolate|dark brown|5.25
```

Inner Join

```
SELECT * FROM sales, cones WHERE cone_id =cones.id;
```

When column names conflict we write: `table_name.column_name` in a query.

```
sqlite> SELECT * FROM cones, sales WHERE cone_id=cones.id;
Id|Flavor|Color|Price|Cashier|id|cone_id
1|strawberry|pink|3.55|Baskin|3|1
1|strawberry|pink|3.55|Robin|6|1
2|chocolate|light brown|4.75|Baskin|1|2
2|chocolate|light brown|4.75|Baskin|4|2
2|chocolate|light brown|4.75|Robin|5|2
3|chocolate|dark brown|5.25|Robin|2|3
```

Putting It All Together:

- Which of our cashiers sold the highest value of ice cream?
- First we need to find which cones were sold by whom, then we SUM() the results!

```
sqlite> SELECT Cashier, SUM(Price) as 'Total Sold' FROM
sales, cones WHERE sales.cone_id = cones.id GROUP BY
Cashier;
Cashier|Total Sold
Baskin|13.3
Robin|13.8
```

SQL: using named tables - FROM

```
SELECT "delicious" as Taste, Flavor, Color FROM cones
      WHERE Flavor is "chocolate" UNION
SELECT "also tasty", Flavor, Color FROM cones
      WHERE Flavor is not "chocolate";
```

```
sqlite> select "delicious" as Taste, Flavor, Color from cones where Flavor is "chocolate" union
   ...> select "other", Flavor, Color from cones where Flavor is not "chocolate";
Taste|Flavor|Color
delicious|chocolate|dark brown
delicious|chocolate|light brown
other|bubblegum|pink
other|strawberry|pink
sqlite>
```

Queries within queries

- Any place that a table is named within a select statement, a table could be computed
  - As a sub-query

```
select TID from sales where Cashier is "Baskin";

select * from cones
    where ID in (select TID from sales where Cashier is "Baskin");

sqlite> select * from cones
   ...>     where ID in (select TID from sales where Cashier is "Baskin");
ID|Flavor|Color|Price
1|strawberry|pink|3.55
3|chocolate|dark brown|5.25
4|strawberry|pink|5.25
```

Summary

```
SELECT  <col spec> FROM <table spec> WHERE <cond spec>
    GROUP BY <group spec> ORDER BY <order spec> ;
```

```
INSERT INTO table(column1, column2,...)
       VALUES (value1, value2,...);
```

```
CREATE TABLE name ( <columns> ) ;
```

```
CREATE TABLE name AS  <select statement> ;
```

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

SQL: Joins

Joining tables

- Two tables are joined by a comma to yield all combinations of a row from each
  - `select * from sales, cones;`

```
create table sales as
    select "Baskin" as Cashier, 1 as TID union
    select "Baskin", 3 union
    select "Baskin", 4 union
    select "Robin", 2 union
    select "Robin", 5 union
    select "Robin", 6;
```

Cashier	TID
Baskin	1
Robin	2
Baskin	3
Baskin	4
Robin	5
Robin	6

```
sales.join('TID', cones, 'ID')
```

TID	Cashier	Flavor	Color	Price
1	Baskin	strawberry	pink	3.55
2	Robin	chocolate	light brown	4.75
3	Baskin	chocolate	dark brown	5.25
4	Baskin	strawberry	pink	5.25
5	Robin	bubblegum	pink	4.75
6	Robin	chocolate	dark brown	5.25

```
sqlite> select * from sales, cones;
Baskin|1|1|strawberry|pink|3.55
Baskin|1|2|chocolate|light brown|4.75
Baskin|1|3|chocolate|dark brown|5.25
Baskin|1|4|strawberry|pink|5.25
Baskin|1|5|bubblegum|pink|4.75
Baskin|1|6|chocolate|dark brown|5.25
Baskin|3|1|strawberry|pink|3.55
Baskin|3|2|chocolate|light brown|4.75
Baskin|3|3|chocolate|dark brown|5.25
Baskin|3|4|strawberry|pink|5.25
Baskin|3|5|bubblegum|pink|4.75
Baskin|3|6|chocolate|dark brown|5.25
Baskin|4|1|strawberry|pink|3.55
Baskin|4|2|chocolate|light brown|4.75
Baskin|4|3|chocolate|dark brown|5.25
Baskin|4|4|strawberry|pink|5.25
Baskin|4|5|bubblegum|pink|4.75
Baskin|4|6|chocolate|dark brown|5.25
Robin|2|1|strawberry|pink|3.55
Robin|2|2|chocolate|light brown|4.75
Robin|2|3|chocolate|dark brown|5.25
Robin|2|4|strawberry|pink|5.25
Robin|2|5|bubblegum|pink|4.75
Robin|2|6|chocolate|dark brown|5.25
Robin|5|1|strawberry|pink|3.55
Robin|5|2|chocolate|light brown|4.75
Robin|5|3|chocolate|dark brown|5.25
Robin|5|4|strawberry|pink|5.25
Robin|5|5|bubblegum|pink|4.75
Robin|5|6|chocolate|dark brown|5.25
Robin|6|1|strawberry|pink|3.55
Robin|6|2|chocolate|light brown|4.75
Robin|6|3|chocolate|dark brown|5.25
Robin|6|4|strawberry|pink|5.25
Robin|6|5|bubblegum|pink|4.75
Robin|6|6|chocolate|dark brown|5.25
```

Inner Join

```
select * from sales, cones where TID=ID;
```

```
sales.join('TID', cones, 'ID')
```

TID	Cashier	Flavor	Color	Price
1	Baskin	strawberry	pink	3.55
2	Robin	chocolate	light brown	4.75
3	Baskin	chocolate	dark brown	5.25
4	Baskin	strawberry	pink	5.25
5	Robin	bubblegum	pink	4.75
6	Robin	chocolate	dark brown	5.25

```
sqlite> select * from sales, cones where TID=ID;
Baskin|1|1|strawberry|pink|3.55
Baskin|3|3|chocolate|dark brown|5.25
Baskin|4|4|strawberry|pink|5.25
Robin|2|2|chocolate|light brown|4.75
Robin|5|5|bubblegum|pink|4.75
Robin|6|6|chocolate|dark brown|5.25
sqlite>
```

SQL: using named tables - `from`

```
select "delicious" as Taste, Flavor, Color from cones
       where Flavor is "chocolate" union
select "other", Flavor, Color from cones
       where Flavor is not "chocolate";
```

```
sqlite> select "delicious" as Taste, Flavor, Color from cones where Flavor is "chocolate" union
   ...> select "other", Flavor, Color from cones where Flavor is not "chocolate";
Taste|Flavor|Color
delicious|chocolate|dark brown
delicious|chocolate|light brown
other|bubblegum|pink
other|strawberry|pink
sqlite>
```

Queries within queries

- Any place that a table is named within a select statement, a table could be computed
  - As a sub-query

```
select TID from sales where Cashier is "Baskin";

select * from cones
    where ID in (select TID from sales where Cashier is "Baskin");

sqlite> select * from cones
   ...>     where ID in (select TID from sales where Cashier is "Baskin");
ID|Flavor|Color|Price
1|strawberry|pink|3.55
3|chocolate|dark brown|5.25
4|strawberry|pink|5.25
```

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Review and Wrap Up

CS88 Head TAs

Head Teaching Assistants

Matt Au [he/him]

Office Hours: Tuesday 5pm - 7pm, Thursday 5pm - 6pm

mattau@berkeley.edu

Did you know Plentea gives free glass jars with their boba? I like boba + boba u4's.

Shreya Kannan [she/her]

Office Hours: Tuesday 11am - 12pm

shreyakannan@berkeley.edu

Hi everyone, I'm super thrilled to meet y'all this semester! In no particular order, I prefer musicals over concerts, cold over warm weather, dancing over singing, and sushi over tacos. Feel free to talk to me about great restaurants, cool study spots, and, of course, CS 88 :)

CS88 TAs

Teaching Assistants

Anjali Gurajapu [she/her]

Office Hours: Wednesday 4pm - 5pm

agura@berkeley.edu

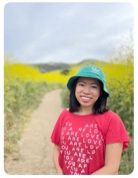
Hi! I'm Anjali, and I'm a third year studying Chemistry and Data Science. In my free time, I've been drawing, baking, and trying to befriend the stray cat that sleeps on my porch. I'm excited to be a TA this semester, and looking forward to meeting you all! :)

Chi Tsai [she/her]

Office Hours: Tuesday 12pm - 1pm

chitsai@berkeley.edu

Hi! I'm Chi, and I'm from the LA area. I enjoy cooking and trying new food, playing minecraft, planning upcoming vacations, and watching Brooklyn 99 (Terry loves yogurt). Come talk to me about anything, happy to meet you!

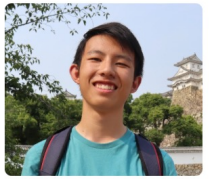


Jessica Lin [she/her]

Office Hours: Monday 10am - 11am

linjessica@berkeley.edu

Hi friends! I'm a second year CS major from Southern California. I enjoy dancing, reading, doing crosswords, and making random peace signs. Feel free to reach out to me for anything (:



Lukas Chang [he/him]

Office Hours: Monday 11am - 12pm

lukasc@berkeley.edu

Hi everyone I'm Lukas, a 3rd year CS major from the south bay area. This is my third semester TAing for CS88 and I'm excited to meet you all! A little about me—in my free time I love making/listening to music, thrifting, and watching anime. I hope I can share my love for CS with you all!

Minnie Chen [she/her]

Office Hours: Tuesday 6pm - 7pm

chn_minnie_22@berkeley.edu

Hi! I'm Minnie, a 4th year Civil Engineering major and minoring in EECS and Sustainability. I like to cook when I'm not too lazy and enjoy active sports when I'm not too tired or lazy :D I'm trying to get back into reading so please let me know if you have any book reccs! I also recently studied abroad last semester, so feel free to ask me about that. Excited to meet everyone!

Tommy Joseph [he/him]

Office Hours: Tuesday 1:30pm - 2:30pm

tommy11jo@berkeley.edu

Hi I'm a third-year CS major from Southern California, and this is my third time TAing for CS 88. Right now, some things I like are bread, blogs, and tv shows (highly recommend The Wire and Silicon Valley). Feel free to reach out about anything

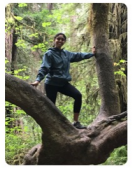
CS88 Tutors

Amit Sant [he/him]

Office Hours: Thursday 4pm - 5pm, Friday 12pm - 1pm

amitsant2000@berkeley.edu

Hello, my name is Amit, and I am a third year CS major at UC Berkeley. My hobbies include gapping League of Legends esports, chess, anime, and osu! Aside from that I love to code, teach, and work on stuff related to making our planet somewhat more livable.



Hetal Shah [she/her]

Office Hours: Wednesday 3pm - 4pm, Friday 11am - 12pm

hetal.shah@berkeley.edu

Hello! I am a 3rd year CS major from Southern California. I love the beach and the sun. In my free time, I like working out, reading, hiking, and traveling. Feel free to reach to me!

Karim Kaylani [he/him]

karimkaylani@berkeley.edu

Hi everyone! My name is Karim and I'm a 2nd year CS major from Southern California. I'm super passionate about all things music whether it's playing guitar, collecting vinyls, or going to shows. I also love taking film photos, hiking, wordle, and ghibli movies. I'm very excited to be a tutor this semester and meet you all, never hesitate to reach out to me about anything anytime! :)

Kevin Gu [he/him]

Office Hours: Tuesday 1pm - 2pm, Thursday 6pm - 7pm

kevinjgu@berkeley.edu

Hi everyone! I'm Kevin, a 5th year Master of Information and Data Science student in the UC Berkeley School of Information. I'm so excited to be a tutor and am looking forward to meeting all of you during office hours! Outside of academics, I love to listen to music (mainly classical, but I can appreciate any genre), take walks, travel (unfortunately not during Covid, but in general), and learn languages (just not enough time to do it, but am willing to practice speaking French), so feel free to reach out to me regarding any of these. :D

Sebastian Zhao [he/him]

sebbyzhao@berkeley.edu

Hiya! I'm a first year intended CS and DS major from Erie, Pennsylvania (a small town right below the lake). I'm a huge fan of cooking, digital art, video games, and manga. Feel free to talk to me about anything, I am a huge believer in conversations about random stuff!

COME JOIN COURSE STAFF!

THANK YOU!

THANK YOU!

The One Big Thing...

Abstraction

- Detail removal

  "The act of leaving out of consideration one or more properties of a complex object so as to attend to others."

- Generalization

  "The process of formulating general concepts by abstracting common properties of instances"

- Technical terms: Compression, Quantization, Clustering, Unsupervised Learning

Henri Matisse "Naked Blue IV"

The Power of Abstraction, Everywhere!

- **Examples:**
  - **Functions (e.g., sin x)**
  - **Hiring contractors**
  - **Application Programming Interfaces (APIs)**
  - **Technology (e.g., cars)**
- **Amazing things are built when these layer**
  - **And the abstraction layers are getting deeper by the day!**

We only need to worry about the interface, or specification, or contract NOT how (or by whom) it's built

Above the abstraction line

Abstraction Barrier (Interface)
(the interface, or specification, or contract)

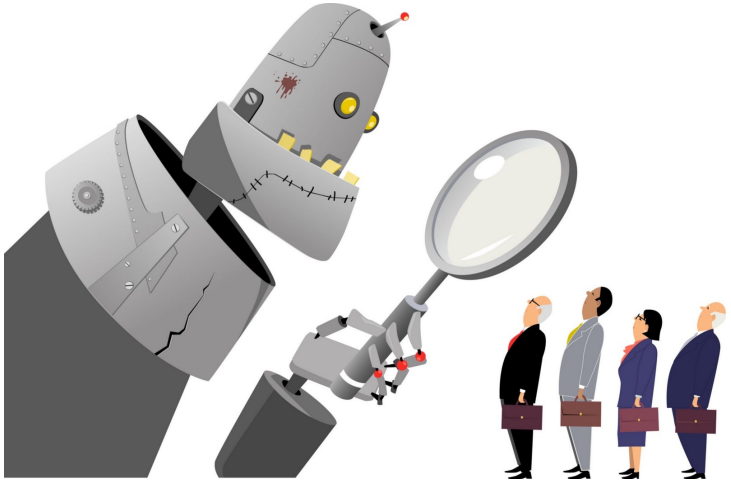
Below the abstraction line

This is where / how / when / by whom it is actually built, which is done according to the interface, specification, or contract.

Abstraction: Pitfalls

- **Abstraction is not universal without loss of information (mathematically provable). This means, in the end, the complexity can only be "moved around"**

- **Abstraction makes us forget how things actually work and can therefore hide bias. Example: AI and hiring decisions.**



- **Abstraction makes things special and that creates dependencies. Dependencies grow longer and longer over time and can become unmanageable.**

Is Your Brain Full Yet?

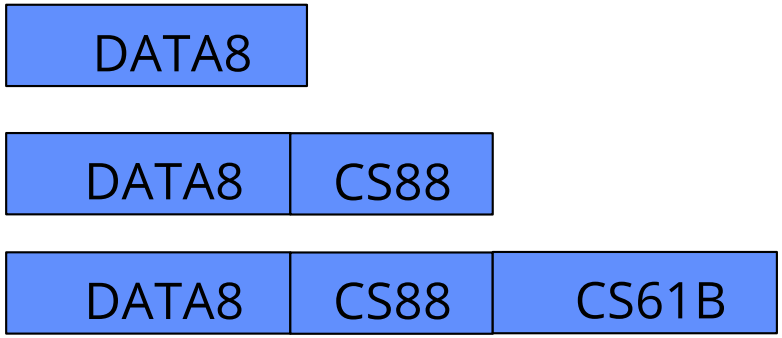
Python Tools:

- Data type: values, literals, operations,
- Expressions, Call expression
- Variables
- Assignment Statement, Tuple assignment
- Sequences: tuple, list
- Dictionaries
- Function Definition Statement
- Conditional Statement
- Iteration: list comp, for, while
- Lambda function expr.

- Higher Order Functions
  - Functions as Values
  - Functions with functions as argument
  - Assignment of function values
  - Function factories – create and return functions
- Higher order function patterns
  - Map, Filter, Reduce
- Recursion
- Abstract Data Types
- Mutation
- Class & Inheritance
- Exceptions
- Iterators & Generators
- SQL / Declarative Programming

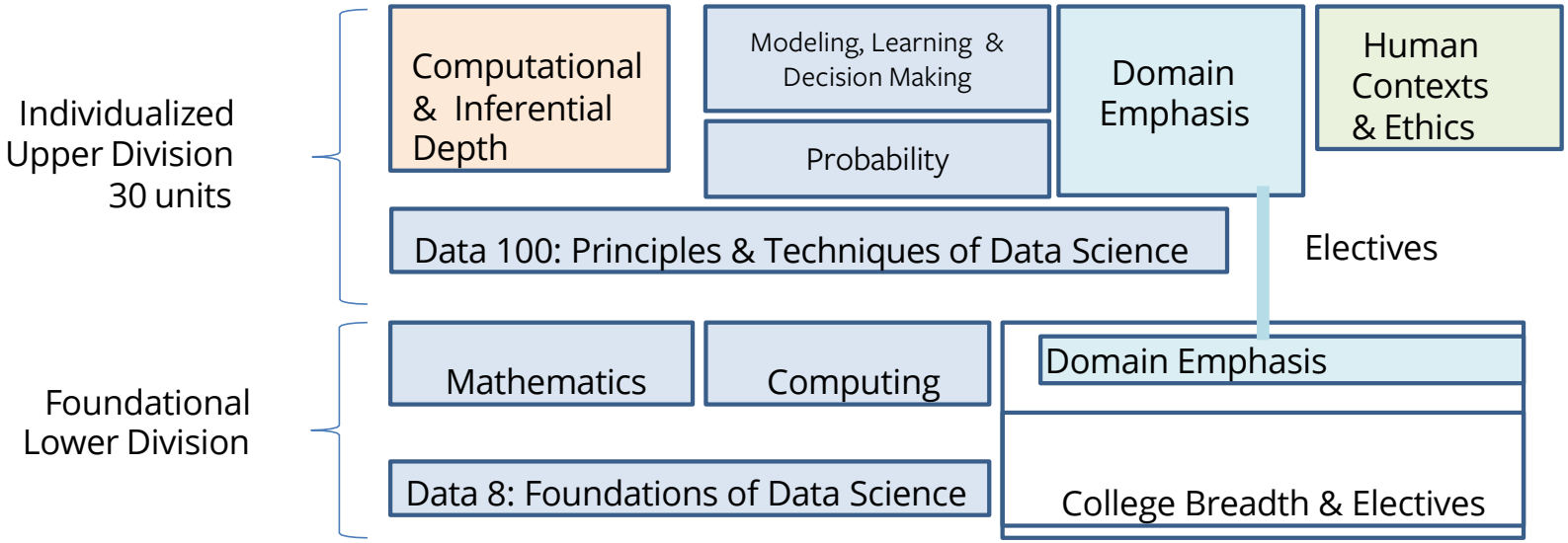
Keep on Programming

Where to go from here?



CS Major Options (But this hard to declare:

The Data Science Major



Data Science Info: https://data.berkeley.edu/2020-cal-week (Recordings)
There's also a minor!

LOTS of Courses to Follow Up!

- Explore all the DS connectors
  - Data 88E (Econ)
  - Stat 88
- Data 100:
- Data 101: Data Engineering
- Data 102
- Data 103
- Data 104
- INFO – Course Schedule Varies!
  - INFO C103 in Spring: "History of Information"
  - Some Database courses, web development, etc.

CS Courses

- CS61B: (conventional) data structures, statically typed production languages.
- CS61C: computing architecture and hardware as programmers see it.
- CS70: Discrete Math and Probability Theory.
- CS170, CS171, CS172, CS174: “Theory”—analysis and construction of algorithms, cryptography, computability, complexity, combinatorics, use of probabilistic algorithms and analysis.
- CS161: Security
- CS162: Operating systems.
- CS164: Implementation of programming languages
- CS168: Introduction to the Internet
- CS160, CS169: User interfaces, software engineering
- CS176: Computational Biology

CS Courses Part 2

* CS182, CS188, CS189: Neural networks, Artificial intelligence, Machine Learning
- CS184: Graphics
- CS186: Databases
- CS191: Quantum Computing
- CS195: Social Implications of Computing
- EECS 16A, 16B: Designing Information Systems and Devices
- EECS 126: Probabilty and Random Processes
- EECS149: Embedded Systems
- EECS 151: Digital Design
- CS194: Special topics. (E.g.) computational photography and image manipulation, cryptography, cyberwar.
- Plus graduate courses on these subjects and more.
- And please don't forget CS199 and research projects.

EE Courses Are There Too

- EE105: Microelectronic Devices and Circuits.
- EE106: Robotics
- EE118, EE134: Optical Engineering, Photovotalaic Devices.
- EE120: Signals and Systems.
- EE123: Digital Signal Processing.
- EE126: Probability and Random Processes.
- EE130: Integrated Circuit Devices.
- EE137A: Power Circuits.
- EE140: Linear Integrated Circuits (analog circuits, amplifiers).
- EE142: Integrated Circuits for Communication.
- EE143: Microfabrication Technology.
- EE147: Micromechanical Systems (MEMS).
- EE192: Mechatronic Design.

And there's lots more to Python!

What can you do with Python?

- Almost anything!
- Webapp backends
- Web scraping
- Natural Language Processing
- Data analysis
- Machine Learning
- Scientific computing
- Games
- Procedural generation - L Systems, Noise, Markov

What can you do with Python?

- Almost anything! Thanks to libraries!
- Webapp backends (Flask, Django)
- Web scraping (BeautifulSoup)
- Natural Language Processing (NLTK)
- Data analysis (Numpy, Pandas, Matplotlib)
- Machine Learning (FastAi, PyTorch, Keras)
- Scientific computing (SciPy)
- Games (Pygame)
- Procedural generation - L Systems, Noise, Markov

Peer Resources: Join Piazza!

- https://bit.ly/datapiazza – Data 001 Piazza
- https://bit.ly/eecspiazza -- EECS 101 Piazza

Ask Is Anything!

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

THANK YOU!

(Again!)