Discussion 12

Interpreters, SQL

Mingxiao Wei
mingxiaowei@berkeley.edu

April 20, 2023

From last time ... 👀

"Come up with a random question"

- tuple or tuple
- How do you cut your sandwiches?
- Who is your favorite fictional character?
- How many distinct cat breeds are there?
- Does pineapple belong on pizza?
- What is your favorite food
- What is your favorite thing to cook?
- is water wet?
- Best class or clubs to join
- What is your favorite color?
- cafe with the best chai latte?
- What classes are you taking next semester?
- but it is required 0_o - THAT WAS A MISTAKE I'M SORRY 🥲
- why are you mcb
  - Tbh this to me is more like "why am I cs" bc I came to Cal thinking that I'd only do MCB but then I got interested in cogsci and took 61a and was like hmm maybe cs is also fun so here I am :)
- What is the hardest class at Cal?
- favorite cs class
- What is your favorite class out of all the classes you have taken at Berkeley?
- What is one place you would go to if you had to go on vacation for a whole month
- DeNero vs Farid FACE OFF.. who is your favorite professor and why?
- What would you do if everything was legal for 24 hours?
- pancakes or waffles?
- What is your favorite course you've taken at UC Berkeley?
- Plan for summer
- What's your favorite flavor of ice cream?

* I'm not too sure if some questions are meant for me or just a random question you came up with... but if it is and I didn't answer it please just ask it again in today's attendance form!

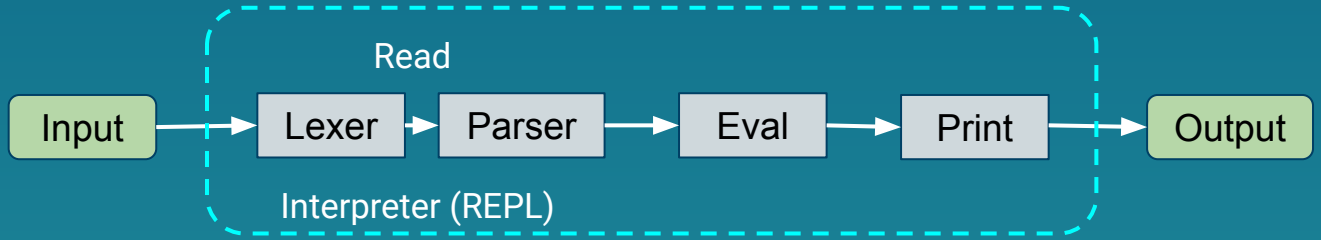
Logistics

- Homework 09 due today 04/20
- `(scheme)` 👀
  - Checkpoint 1 (part 1) due tomorrow 04/21
  - Checkpoint 2 (part 2 & 3) due next Tue 04/25
  - The entire project due next Fri 04/28
  - Submit everything by next Thu 04/27 for 1 extra credit!
  - Go to OH/project parties!
  - Recommended to work with a partner - pls don't just split the work, but try to sit down and work on the project together!
- (optional) Scheme Contest - create cool art with Scheme!! 🧑‍🎨
- Reminder about homework 08 recovery (Ed #2782)
- One more week left - let's keep it up 💪
  - Which… also means that we only have two more sections :')

Interpreters

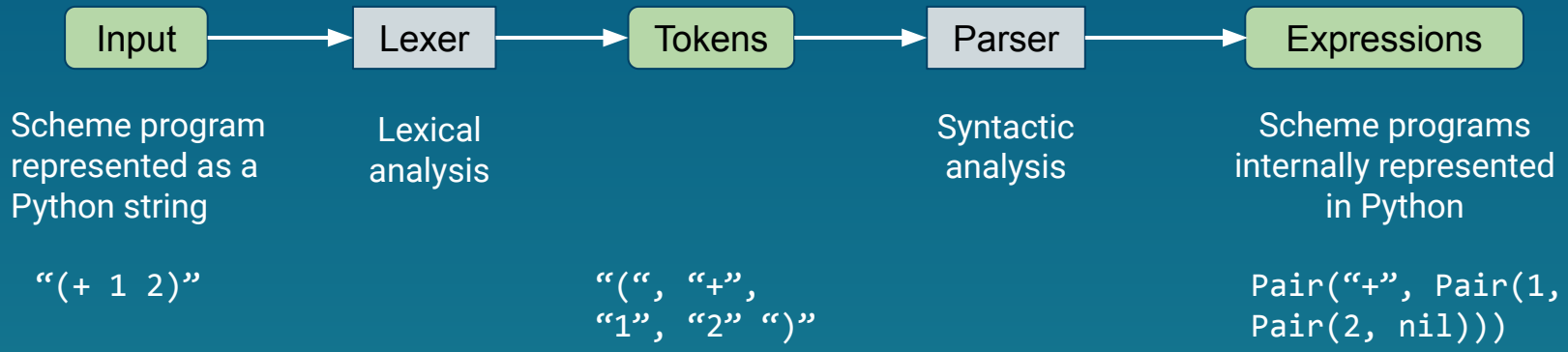
Interpreters

- **<u>Interpreter</u>** - a program that allows one to interact with the computer in a certain language
  - Think of it as a translator
  - Two languages at work
    - The language being interpreted (e.g., Scheme)
    - The language that performs the underlying interpretation (e.g., Python)
- Most interpreters use a REPL (Read-Eval-Print Loop)




```
scm> (+ 1 2)
3
```

Read stage


Input
Lexer
Tokens
Parser
Expressions
Scheme program represented as a Python string
Lexical analysis
Syntactic analysis
Scheme programs internally represented in Python
“(+ 1 2)”
“(“, “+”, “1”, “2” “)”
Pair(“+”, Pair(1, Pair(2, nil)))

Internal Representations

Expressions

How are scheme programs represented in the output of the read stage?

Scheme	Python
Numbers	`int` or `float` values
Symbols	String (`str`) values
Booleans (`#t`, `#f`)	`bool` values (`True`, `False`)
Combinations (anything that's not a primitive value - lists, call expressions, special forms)	`Pair` objects
`nil`	The `nil` object

No evaluation at the read stage - everything is a either primitive value (number, boolean, string) or a `Pair` object containing primitive values

The pair class

```
class Pair:
    """Represents the built-in pair data structure in Scheme."""
    def __init__(self, first, rest):
        self.first = first
        if not scheme_valid_cdrp(rest):
            raise SchemeError("cdr can only be a pair")
        self.rest = rest
    def map(self, fn):
        """Maps fn to every element in a list, returning a new Pair"""
        assert isinstance(self.rest, Pair) or self.rest is nil
        return Pair(fn(self.first), self.rest.map(fn))
```

Similar to a linked list - `first` is the value at the current node and `rest` is another `Pair` object or `nil` (empty Pair)

`rest` is not optional

To apply a one-argument function to every element in a Pair, do `pair_object.map(fn)`

The `nil` class/object

```
class nil:
    """Represents the special empty pair nil in Scheme."""
    def map(self, fn):
        return nil
    def __getitem__(self, i):
         raise IndexError('Index out of range')
    def __repr__(self):
        return 'nil'
```

```
nil = nil() # this hides the nil class *forever*
```

- `nil` represents the empty `Pair` - similar to `Link.empty`
- `nil` is an object - use `pair is nil` to check if a `Pair` object is empty

Convert a Scheme combination to a `Pair` object

- Each element in the Scheme combination corresponds to one node/element in the `Pair` object
- Length of Scheme combination = length of `Pair`
- Nested combination → nested `Pair`

“(+ 1 2)” → `Pair(“+”, Pair(1, Pair(2, nil)))`

“(+ 1 (* 2 3))” → `Pair(“+”, Pair(1, Pair(_____, nil)))`

Convert a Scheme combination to a `Pair` object

- Each element in the Scheme combination corresponds to one node/element in the `Pair` object
- Length of Scheme combination = length of `Pair`
- Nested combination → nested `Pair`

```
"(+ 1 2)"          →  Pair("+", Pair(1, Pair(2, nil)))

"(+ 1 (* 2 3))"    →  Pair("+", Pair(1, Pair(______, nil)))
```

↓

```
Pair("*", Pair(2, Pair(3, nil)))
```

Convert a Scheme combination to a `Pair` object

- Each element in the Scheme combination corresponds to one node/element in the `Pair` object
- Length of Scheme combination = length of `Pair`
- Nested combination → nested `Pair`

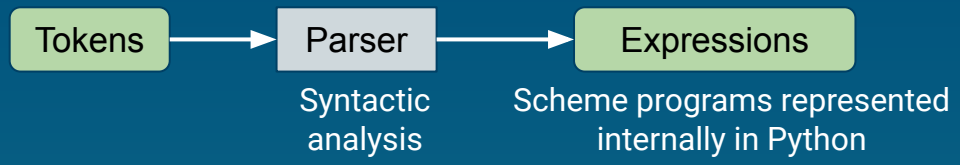
“(+ 1 2)” → `Pair(“+”, Pair(1, Pair(2, nil)))`

“(+ 1 (* 2 3))” → `Pair(“+”, Pair(1, Pair(Pair(“*”, Pair(2, Pair(3, nil))), nil)))`

↓

`Pair(“*”, Pair(2, Pair(3, nil)))`

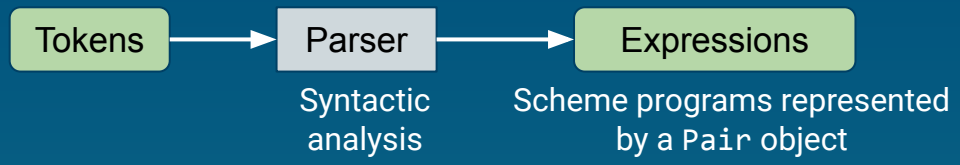
Syntactic analysis



- Identify the hierarchical structure of the program - may be nested
- Convert token to their internal representation in Python
  - Each `scheme_read` call consumes the input tokens for exactly one expression
- Recursive
  - Base case
    - primitive values (numbers, booleans, symbols)
    - Return the corresponding primitive value in Python
  - Recursive case
    - recursively call `scheme_read` to read the sub-expression and combine
    - Return the expression as a `Pair` object in Python

* In the Scheme project, this function is called `scheme_read`

Syntactic analysis



"(+ 1 (* 2 3))"

Tokenized by the lexer

"(", "+", "1", "(", "*", "2", "3", ")", ")"

Parser - syntactic analysis

Returns the expression

Pair("+", Pair(1, Pair(Pair("*", Pair(2, Pair(3, nil))), nil)))

scheme_read(["(", "+", "1", "(", "*", "2", "3", ")", ")"])

"+"

1

Pair("*", Pair(2, Pair(3, nil)))

scheme_read(["+"])

scheme_read(["1"])

scheme_read(["(", "*", "2", "3", ")"])

3

"*"

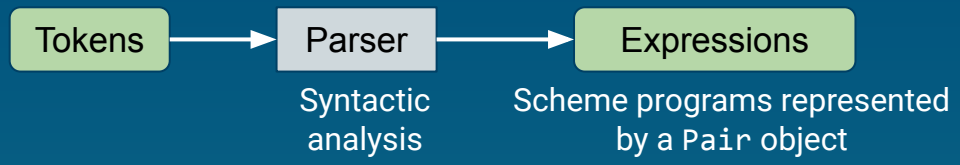
2

scheme_read(["*"])

scheme_read(["2"])

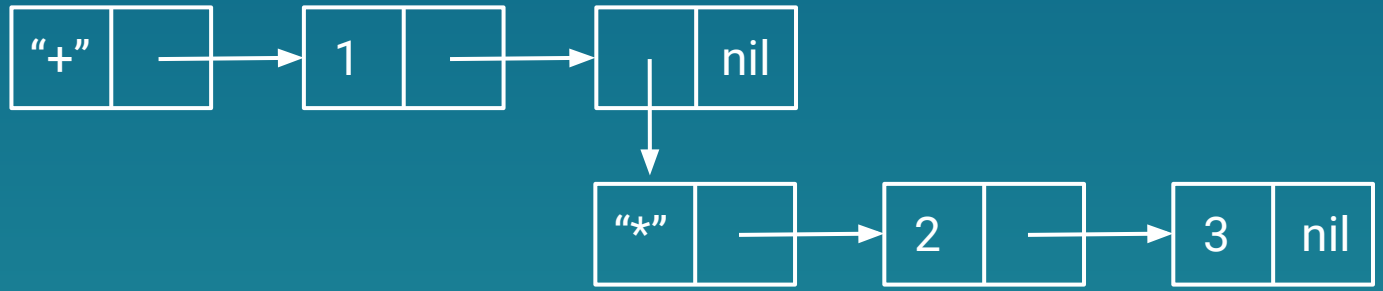
scheme_read(["3"])

Syntactic analysis

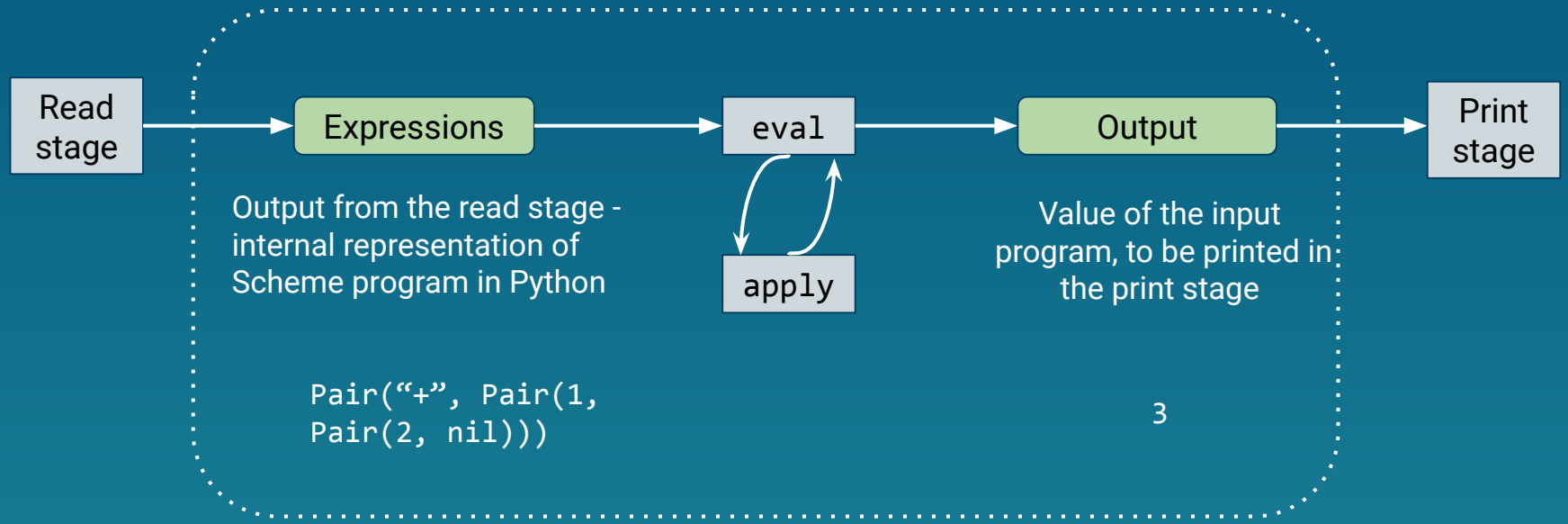



```
(+ 1 (* 2 3))
```

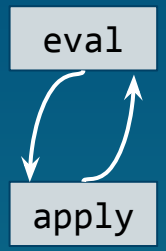
```
Pair(“+”, Pair(1, Pair(Pair(“*”, Pair(2, Pair(3, nil))), nil)))
```



Eval stage

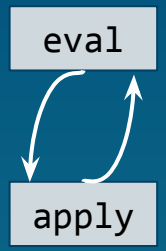

Read stage
Expressions
eval
Output
Print stage
Output from the read stage - internal representation of Scheme program in Python
apply
Value of the input program, to be printed in the print stage
Pair(“+”, Pair(1, Pair(2, nil)))
3

Eval/Apply



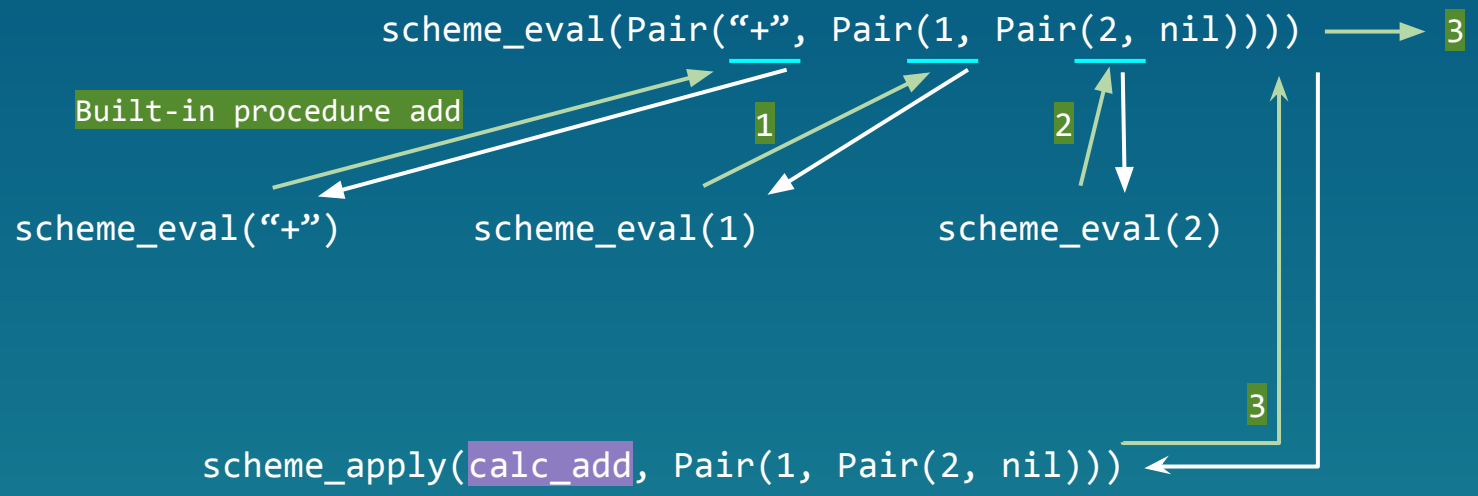
- `eval(exp)`
  - Takes in the output from the read stage as the input
  - Recursively evaluate the expression
    - Base case
      - Primitive values - return the value
      - Symbols - look up the variable name and return its value
    - Recursive case
      - Look at the first element in the expression
      - Special form → follow the corresponding evaluation rule
      - Call expression → recursively call `eval` on the operator and operands, then `apply` the operator to the operands

Eval/Apply



- `apply(op, args)`
  - Inputs
    - op - the evaluated operator, which is a function value
    - `args` - the evaluated operands as a `Pair` object
  - Apply the operator to its operands (recursively)
    - Base case
      - Built-in procedures
    - Recursive case
      - User-defined procedures - recursively call `eval` to evaluate the function body

Eval/Apply



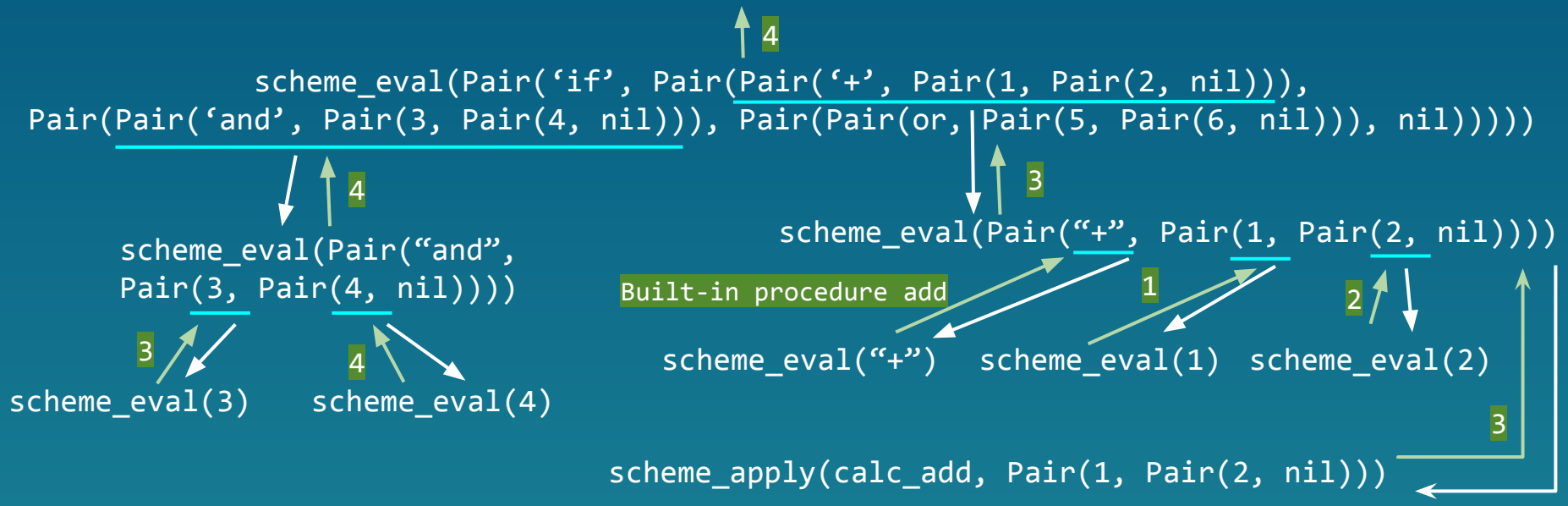
The corresponding Python function for procedures like this should take in a Pair object as its argument, and return the desired result

Counting Eval/Apply

- Eval - once for each complete expression (a complete parenthesis) and each primitive value (variable names, numbers, symbols, etc.) <u>that gets evaluated</u>
  - For special forms, not everything gets evaluated!
- Apply - once for each procedure that gets called
  - Special form keywords (`if`, `cond`, etc,) do not count as operator!

Counting Eval/Apply

(if (+ 1 2) (and 3 4) (or 5 6))



Worksheet Q1, 2

SQL

Structured Query Language

SQL - Structured Query Language

- Declarative programming language - describe the **desired result** of computation instead of computations
- Used to query from a database (table, e.g.)
- Query interpreter is responsible for planning and performing computations to produce the desired result
  - Not in scope - take cs186 if you are interested in learning more about database :0
- By convention, keywords are capitalized (`SELECT`, `AS`, `UNION`, `WHERE`, etc.)
- Each query ends with a semicolon `;`
  - Line break does not matter as long as the query is terminated properly by a `;`

Creating a table

Column name

; denotes the end of a query

```
sql> SELECT "Ben" AS first, "Bitdiddle" AS last;
```

```
Ben|Bitdiddle
```

Entries

Output table

first	last
Ben	Bitdiddle

```
sql> SELECT "Ben" AS first, "Bitdiddle" AS last UNION
...> SELECT "Louis", "Reasoner";
Ben|Bitdiddle
Louis|Reasoner
```

Combine multiple rows together

first	last
Ben	Bitdiddle
Louis	Reasoner

Select values from a table

```
SELECT * FROM table;
```

Select all columns from `table` → print the entire table

```
SELECT col FROM table;
```

Select the column `col` from `table` (all rows included)

```
SELECT col_1, col_2, … FROM table;
```

Select the specified columns `col_1, col_2, …` from `table` (all rows included)

Select values from a table

```
sql> SELECT * FROM table1;
Alice|20|Dog
Bob|21|Cat
sql> SELECT Name FROM table1;
Alice
Bob
sql> SELECT Name, Pet FROM table1;
Alice|Dog
Bob|Cat
```

table1

Name	Age	Pet
Alice	20	Dog
Bob	21	Cat

Filtering on rows - WHERE

```
SELECT [columns] FROM [tables] WHERE [condition];
```

- Only rows that satisfy `[condition]` will be displayed
- Use =, >=, <=, != to compare; AND, OR to combine multiple conditions

```
sql> SELECT * FROM table2 WHERE Name = 'Alice';

Alice|20|Dog

sql> SELECT Name FROM table2

...>     WHERE Age >= 21 AND Pet = 'Cat';

Cindy
```

Column names treated as variable names, no quotes

Entries are string values, quoted

table2

Name	Age	Pet
Alice	20	Dog
Bob	21	Cat
Cindy	22	Fish

Ordering of output rows - ORDER BY

`SELECT [columns] FROM [tables] ORDER BY [criteria];`

- [criteria] is often the name of one column
  - If it contains multiple columns, sort by the first column first, use the second column to break tie, etc.
- Default - ascending order (alphabetical order for strings)
- Can also specify by `ORDER BY [criteria] DESC` or `ORDER BY [criteria] ASC`

```
sql> SELECT Name FROM table1 ORDER BY Age;
Alice
Bob
sql> SELECT Name FROM table1 ORDER BY Age DESC;
Bob
Alice
```

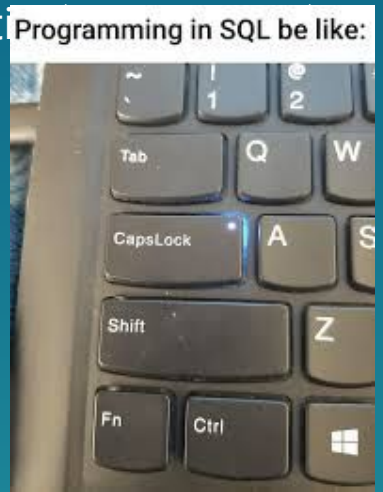
table1

Name	Age	Pet
Alice	20	Dog
Bob	21	Cat

Select syntax

```
SELECT [columns] FROM [tables] WHERE [condition] ORDER BY [criteria] LIMIT #;
```

- `WHERE` and `ORDER BY` are optional
- LIMIT is the upper bound of # of rows in the output
- We'll see more cool operations (aggregat
- SQL keywords are CAPITALIZED



Worksheet Q3-6

Attendance 🤠

go.cs61a.org/mingxiao-att