DISCUSSION 05

Trees

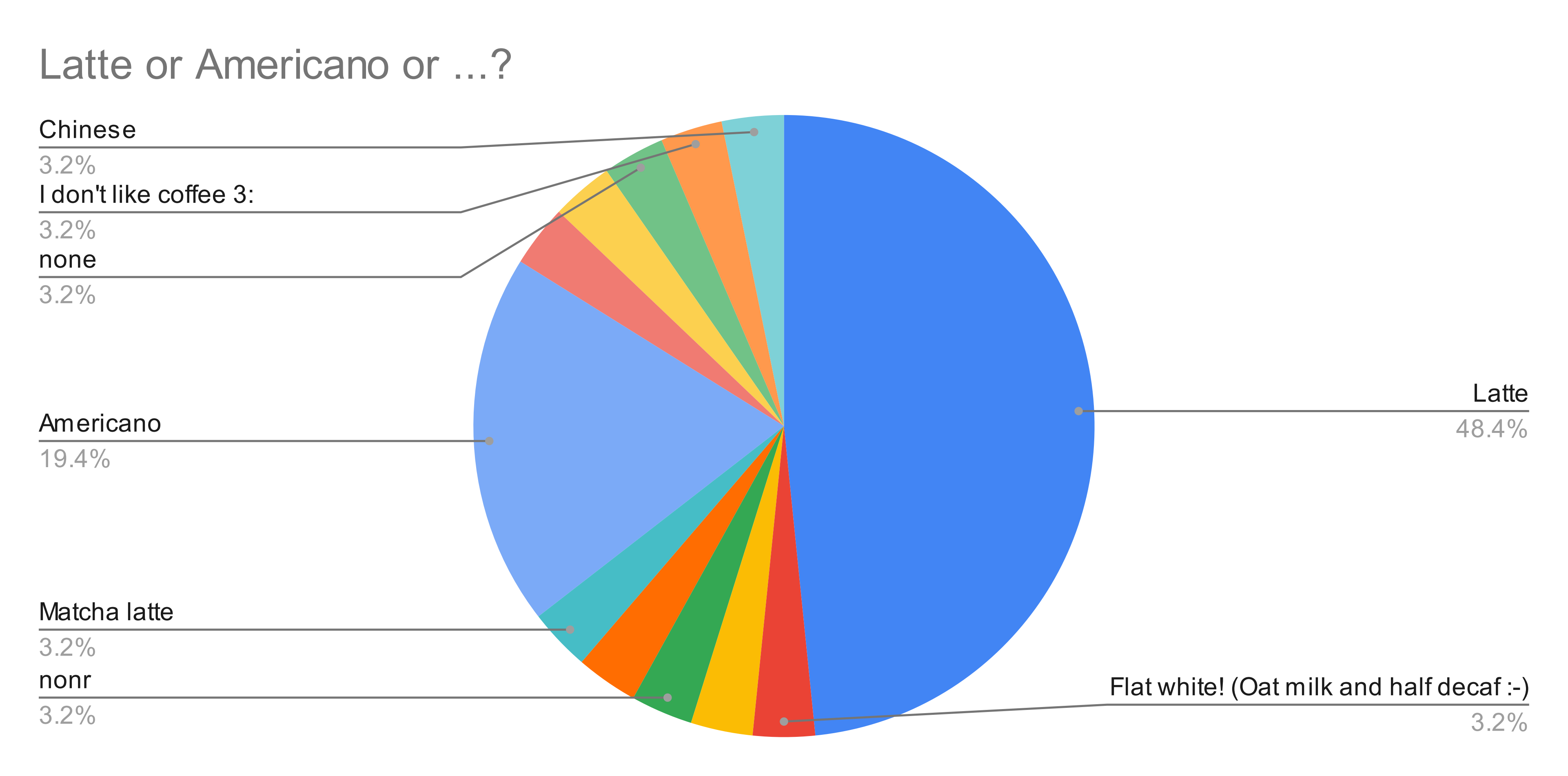
Mingxiao Wei
mingxiaowei@berkeley.edu

Feb 23, 2023

LOGISTICS 🏡

- CATS 🐈
  - Try out the game here: cats.cs61a.org
  - The whole project due tomorrow 02/24
  - Submit everything by today 02/23 for one extra point!
- NO homework due this week - homework 04 due next Thu 03/02

FROM LAST TIME...



DATA ABSTRACTION

👾

DATA ABSTRACTION - INTRO

- Treat code as "objects"
  - no need to know about the actual implementation (e.g., how information is stored and calculated)
  - just need to know what it does
- One can assume that the functions work as described
- A data abstraction consists of two types of functions:
  - Constructors: build and return the abstract data type
  - Selectors: retrieve information from the data type

RATIONAL NUMBERS REVISIT

- Constructor: `rational(n, d)` returns a rational number $x = \frac{n}{d}$ using some underlying representation, which we, as users, do not need to know anything of
- Selectors:
  - `numer(x)` returns the numerator of x
  - `denom(x)` returns the denominator of x
- Arithmetic Operations:

```python
def mul_rational(x, y):
    return rational(numer(x) * numer(y), \
                    denom(x) * denom(y))
```

  - We can manipulate rational numbers using their constructor/selectors without knowing their implementation
  - Side note: the `\` is used to indicate that the expression continues on the next line

RATIONAL NUMBERS REVISIT

There are many ways to implement the rational number data abstraction. Below are two examples:

```
def rational(n, d):
    return [n, d]
def numer(x):
    return x[0]
def denom(x):
    return x[1]
```

```
def rational(n, d):
    return {'n': n, 'd': d}
def numer(x):
    return x['n']
def denom(x):
    return x['d']
```

No matter which one we use, the rational number data abstraction has the same, correct behavior from the users' end.

As programmers, we can design the underlying implementation however we want as long as it behaves as expected

DON'T BREAK THE ABSTRACTION BARRIER!

Parts of the program that...	**Treat rationals as...**	**Using...**
Use rational numbers to perform computation	whole data values	`add_rational`, `mul_rational` `rationals_are_equal`, `print_rational`
Create rationals or implement rational operations	numerators and denominators	`rational`, `numer`, `denom`
Implement selectors and constructor for rationals	two-element lists	list literals and element selection
Implementation of lists		

Source: lecture 13 slides

DON'T BREAK THE ABSTRACTION BARRIER!

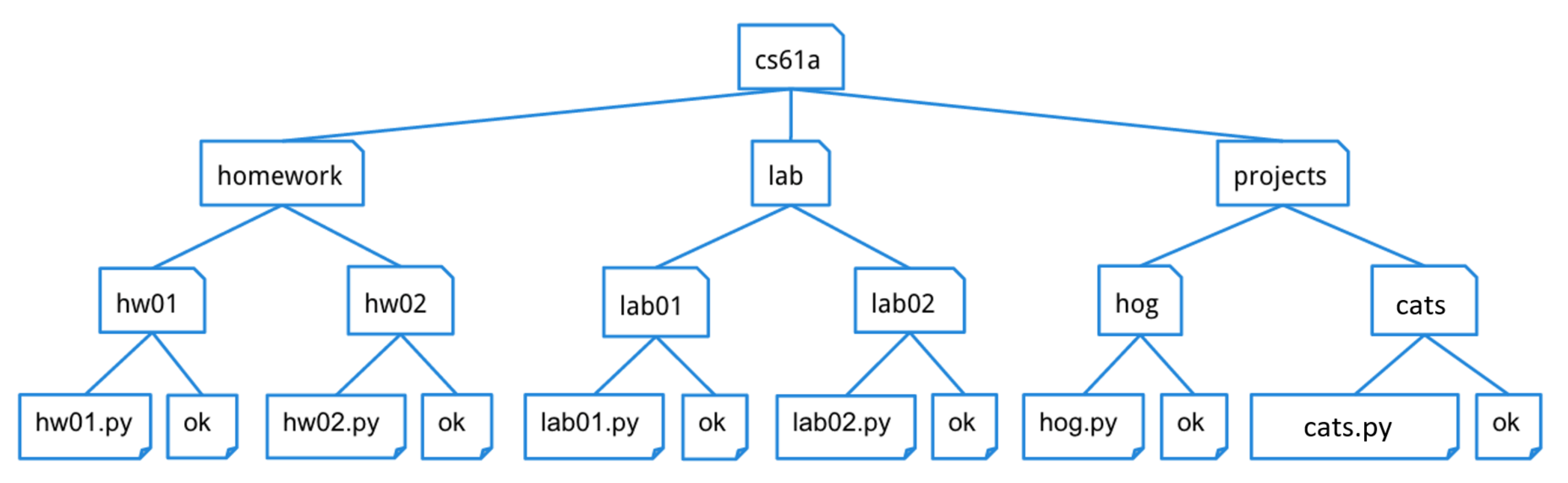
- Why?
    - Breaking the abstraction barrier is repeating other function's job
    - Once the underlying implementation changes, the code may not work anymore
- Examples of violating the abstraction barriers:
    - `mul_rational([1, 2], [3, 4])` should be `mul_rational(rational(1, 2), rational(3, 4))`

    -
      ```
      def divide_rational(x, y):
          return [ x[0] * y[1], x[1] * y[0] ]
      # should be:
      def divide_rational(x, y):
          return rational(numer(x) * denom(y), \
                          denom(x) * numer(y))
      ```
- Takeaway: call an existing function could do whenever you can!

TREES 🎄

TREES - INTRO

- A tree is a data structure that represents a hierarchy of information.
  - For example, a file system:



Trees in computer science are usually drawn upside down - the root at the top and the leaves at the bottom.

TREES - TERMINOLOGY

- node: any location within the tree (e.g., root node, leaf nodes, etc.)
- root: the node at the top of the tree
- label: the value in a node
- branches: <u>**a list of trees**</u> directly under the tree's root
- leaf: a tree with zero branches
- parent node: a node that has at least one branch.
- child node: a node that has a parent. A child node can only have one parent.
- depth: the number of edges between the root to the node. The root has depth 0.
- height: the depth of the lowest (furthest from the root) leaf.

TREES - DATA ABSTRACTION

- Constructor - `tree(label, branches=[])`
  - <u>creates and returns a tree object</u> with the given `label` value at its root node and list of `branches`
  - `branches` is optional and defaults to an empty list
- Selector:
  - `label(tree)` returns the value in the root node of `tree`.
  - `branches(tree)` returns the <u>list of branches</u> of `tree`.
- Convenience function:
  - `is_leaf(tree)` returns `True` if `tree`'s list of branches is empty, and `False` otherwise.

TREES - DATA ABSTRACTION IMPLEMENTATION

```
def tree (label, branches=[]):
    return [label] + list(branches)

def label (tree):
    return tree[0]

def branches (tree):
    return tree[1:]

def is_leaf (tree):
    return not branches(tree)
```

WORKING WITH DATA ABSTRACTION

- no mutation in data abstraction (yet) - to "update" an object, rather than resetting the attributes, create a new object with the updated attributes

```
t0 = tree(61, [tree('a'), tree('b')])

# update the label of t0 to be 100
t1 = tree(100, branches(t0))

# add one more branch to t0
t2 = tree(label(t0), branches(t0) + [tree('c')])
```

WORKSHEET Q1-Q6

ATTENDANCE! 🤠

go.cs61a.org/mingxiao-att

- The attendance form and slides are both linked on our section website!
- Please leave any anonymous feedback here go.cs61a.org/mingxiao-anon
- Please do remember to fill out the form by midnight today!!