Discussion 4

Mutability + Data Abstraction + Trees

Antonio Kam

`anto [at] berkeley [dot] edu`

Announcements

- Cats got released 🐈
  - Be on the lookout for question 7 - start early, and start often
  - Getting started videos

Comments from last section

- I'm hungry
    - I agree!
    - Question: who eats breakfast in the morning
    - who ate breakfast in the morning?
- Could you send out an email with answers to discussion questions so we can do on our own after.
    - Discussion solutions are always uploaded on the website! (both on cs61a.org and on teaching.rouxl.es), so you'll be able to do them on your own
- More recursion 👀
- can y'all put down more stuff please there were only 3 things worth noting 🙄

Temperature Check 🌡️

- Lists
  - List Slicing
  - List Comprehensions
- Mutability
- Data Abstractions
  - Trees

All slides can be found on `teaching.rouxl.es`

Mutability

List Mutation Functions (adding)

- `.append(element)`
  - Adds elements to the end of the list
  - All elements go in one new box (can get nested lists if the element passed in is a list)
- `.extend(iterable)`
  - Concatenates two lists together (typcially `iterable` is a list)
- `.insert(index, element)`
  - Inserts `element` at `index`
  - Does **not** replace elements - this operation instead makes the list longer.
- All these functions return `None` once you use them

List Mutation Functions (removing)

- `.remove(element)`
  - Removes first appearance of element in list
  - Errors if it's unable to remove an element
- `.pop(optional index)`
  - Removes and **returns** element at the given index
  - If index is not provided, it defaults to the last element in the list.

Mutating Lists

- List mutation functions can modify an existing list
- Slicing will create a new list
  - Examples later
- `a = a + b` will create a new list
- `a += b` does not create a new list
- Indexing into a list and changing the element at that list will mutate the list:
  - `a[0] = 7` will change the first element in `a` to be 7.

Identity vs Equality

- `is` will check whether 2 objects are the same thing (i.e. pointing to the same object)
- `==` will check if two objects have the same value

Identity vs Equality

- `is` will check whether 2 objects are the same thing (i.e. pointing to the same object)
- `==` will check if two objects have the same value

```python
a = [1, 2, 3]
b = [1, 2, 3]
a == b # True
a is b # False
```

Mutating Lists (Example)

```
lst1 = [1, 2, 3]
lst2 = lst1
lst3 = lst1[:]

test1a = lst1 == lst2
test1b = lst1 == lst3
test2a = lst1 is lst2
test2b = lst1 is lst3

lst1.append(3)
lst1 = lst1 + [4]
```

(For those reading the slides later, put this into tutor.cs61a.org)

Shallow Copy vs Deep Copy

- Shallow Copy
  - Only copies the first layer of a list
  - If we have a nested list, we only copy the arrow (not the list itself)
  - Create a new list where you copy over whatever is in the same box
- Deep Copy
  - Makes a complete copy of everything in a list
  - Very slow operation - no easy way to do this
- Python uses shallow copies (as do most languages) when copying lists!

Example: Shallow Copy vs Deep Copy

```
lst1 = [1, 2, [3, 4], 5]
lst2 = lst1[:]
```

(For those reading the slides later, put this into tutor.cs61a.org)

Worksheet!

Data Abstractions

What are Data Abstractions?

- Data abstractions are a super powerful way to let people treat code as objects, rather than knowing how the thing works itself
- Allows you to worry about how something works, rather than how something is implemented
- You'll see a lot of abstractions in other courses (Data 8, Data 100 are filled with abstractions of some sort)

What are Data Abstractions?

- Data abstractions have the following:
  - Constructors: Used to build the abstract data type
    - IMPORTANT: You do not need to know how the programmer decided to implement this!
  - Selectors: Used to interact with the data type

Example: Tree Data Abstraction

- Trees are recursive data structures (as in, trees contain more trees)
- Important terms:
  - Root Node
  - Branch(es)
    - This will be a list!
  - Leaf Node
  - Children
- Sort of looks like an upside-down tree compared to the real world
- Questions are generally solved using tree recursions

Tree ADT Implementation:

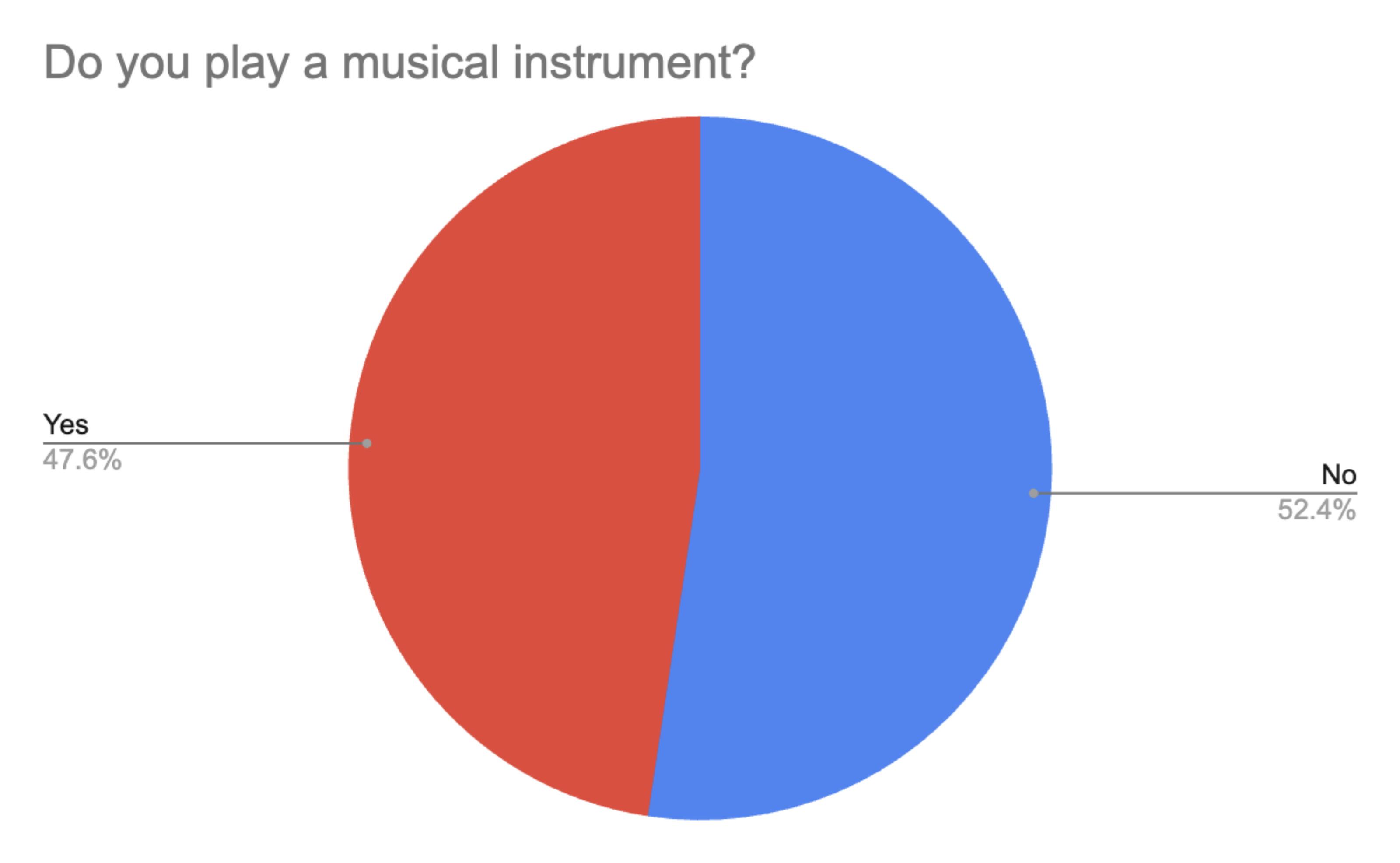
```python
def tree(label, branches=[]):
    """Construct a tree with the given label value and a list of branches."""
    return [label] + list(branches) # All items in branches must be trees!

def label(tree):
    """Return the label value of a tree."""
    return tree[0]

def branches(tree):
    """Return the list of branches of the given tree."""
    return tree[1:]

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

Tree Example:

```
t = tree(1,
      [tree(3,
          [tree(4),
           tree(5),
           tree(6)]),
      tree(2)])
```

Worksheet!

Results from last section (`links.rouxl.es/disc`)



Mental Health Resources

- CAPS:
  - If you need to talk to a professional, please call CAPS at 510-642-9494.
- After Hours Assistance
  - For any assistance after hours, details on what to do can be found at this link

Anonymous Feedback Form

`links.rouxl.es/feedback`

Thanks for coming! 🥳

Please give me feedback on what to improve!