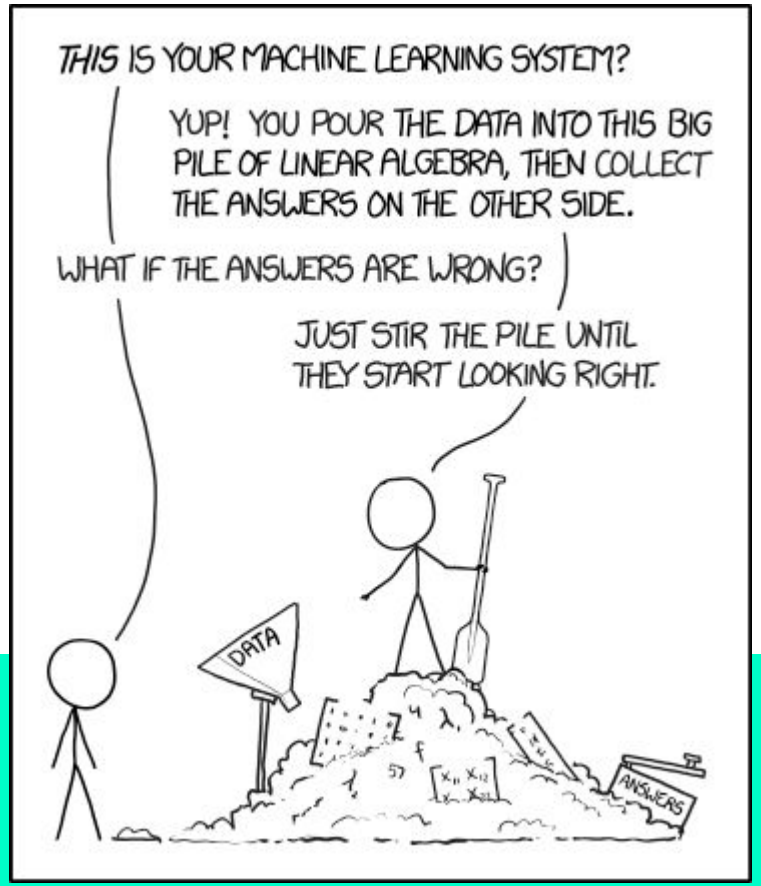


Lab 9 - Functors

ReadMe

Reminders

- Lab 9 due Sunday, December 2nd at 8pm on Canvas
- Project 5 due Friday, December 7th at 8pm
- ❤️ Please fill out my teaching evaluation!

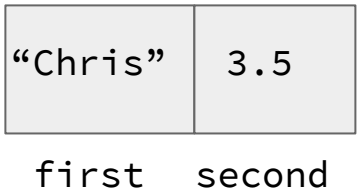
Agenda

- Discuss pairs, maps, functors
- Coding demo
- Worksheet
- Work on lab

std::pair

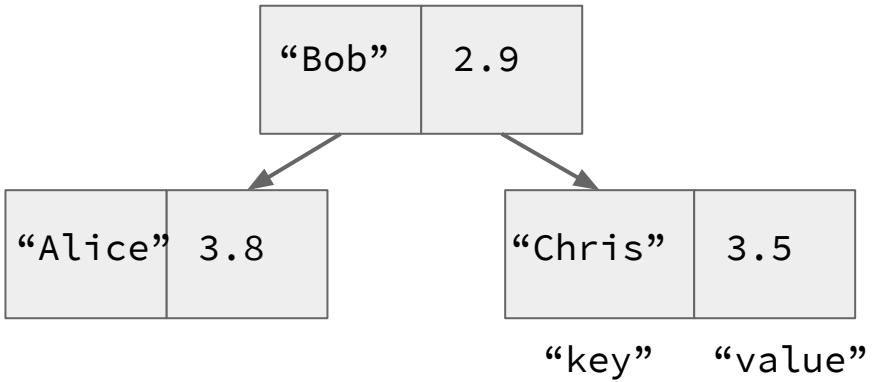
A pair is what it sounds like-- it's a pair of elements. We can make a pair with any two object types we want-- we just have to let std::pair know which one we want. We can make a std::pair<int, int>, a std::pair<std::string, double>, even a std::pair<Duck, std::map>!

```
std::pair<string, double> my_pair;
my_pair.first = “Chris”;
my_pair.second = 3.5;
```



STD::MAP

```
map<string, double> people_gpa;
```

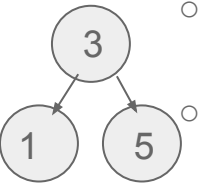


A map is a BST whose elements are of type std::pair. Each std::pair element will contain two values-- a "key", and a "value". These things are called "key" and "value" because we search for elements in the tree/ map that match the "key" we're looking for, and we hope to find a "value" that corresponds with that "key" in the map.

Notice: Are these pairs sorted by the key or the value?

Project 5 Machine Learning (Tips!)

- Binary Search Tree:
  - Inorder traversal means left first, middle second, right last. (Left, Self, Right) (1, 3, 5)
  - Preorder traversal means middle first, left next, right last. (Self, Left, Right) (3, 1, 5)



- Map:
  - A map is a BST, but its elements are pairs. The sort order is determined by comparing the “first” elements in the pairs.
  - [] means something special: if the element exists, return a reference to it. If it doesn’t, add it and return a reference to it. Completing the insert function first might help!
- Machine learning:
  - Train on training set to learn probabilities of each label given each word. Then use those probabilities to calculate probabilities of each label for each test post, selecting the label that is most probable for the post.
  - You only want to store UNIQUE words in a post

Recall: Functors

Functors are class-type objects that behave like functions. They "behave like functions" because they have overloaded () operators.

There are 2 main (really useful) types of functors--

Predicates, and Comparators

Predicates

A predicate is another special type of functor used to check a property of an object. Predicates return either "true" or "false" when passed a single object. For example, this functor can be instantiated with a private member variable "height". You can then pass "Person" objects into it and return whether Person "perry" is taller than "height."

```
class TestHeight {

public:

TestHeight(int height_in) : height(height_in) {}

bool operator()(const Person& perry) {

  return perry.getHeight() > height;

}

private:

int height;

};
```

Comparators

A comparator is a special type of functor used to compare objects. Comparators often follow a "less than pattern," and return true when the first object passed into the comparator is "less than" the second object passed into the comparator. This returns true whenever Duck d1 has a name that is less than Duck d2's name. This functor might be useful to us if we are trying to, for example, sort Ducks by name. If we wanted to use std::sort to sort our Ducks for us (get our Ducks in a row, if you will), we would need some way to tell std::sort what it means for a Duck to be "less than" another Duck.

```
class DuckNameLess {

public:

bool operator()(const Duck &d1, const Duck &d2) const {

    return d1.getName() < d2.getName();

}

};
```

But why functors?

Suppose we're developers for Rate My Professors and we want to implement features that:

- Select all the professors that teach a specific course
- Select all the professors that work at a specific university
- Select all the professors that have above a 3.4 rating
- Select all the professors that have a specific tag
- Select all the professors that have a chili pepper

We could do this by writing functions called select_all_for_eecs280, select_all_for_university, select_all_for_3.4, select_all_for_tag, select_all_for_chili_pepper… but what's wrong with that?

Let's solve this problem using a functor!

```
35 // HasGpaAbove is a predicate. It is initialized with a certain
36 // GPA. If a Professor with a GPA above that value is passed in,
37 // the () operator will return true. Else, it will return false.
38 class HasGpaAbove {
39 public:
40     HasGpaAbove(double gpa_in) : gpa(gpa_in) {}
41
42     bool operator()(const Professor& prof) const {
43         return prof.gpa > 3.4;
44     }
45
46 private:
47     double gpa;
48 }
```

```
template <typename Predicate>
void select_all(Professor profs[], int num, Predicate pred) {
    for(int i = 0; i < num; ++i) {
        if(pred(profs[i])) {
            cout << profs[i].name << " ";
        }
    }
}
```

Maybe we could use a functor like this one for each professor property we want to test, and we write some "select_all" function that will print all the professors for whom the predicate returns true-- this reduces code duplication by a lot!

Coding demo -- functors & maps

https://github.com/MelGeorge/rate_my_professor

Go to this link & clone the repository if you want to code along with me!