Class 9: Functional Programming

6.102 — Software Construction
Spring 2024

Warmup

Exercise:  yellkey.com/**type**

Nanoquiz:  yellkey.com/**east**

Open `catalog.ts`...

- Run its tests with `npm test`
- Do TODO #1: replace loops & ifs with map/filter/reduce
- Tests should still pass

Nanoquiz

- This quiz is just for you and your own brain:
  - closed-book, closed-notes
  - nothing else on your screen
- Lower your laptop screen when you're done


yellkey.com/**east**

Map, Filter, Reduce

Exercise:  yellkey.com/**type**

Do the TODOs in `catalog.ts` to convert loops & ifs with map/filter/reduce

- Step-by-step refactoring: run `npm test` after doing each TODO

TODO #1: filter a list of integers

Map, Filter, Reduce

Exercise:  yellkey.com/**type**

Do the TODOs in `catalog.ts` to convert loops & ifs with map/filter/reduce

- Step-by-step refactoring: run `npm test` after doing each TODO

TODO #1: filter a list of integers

TODO #2: `map()` and `concat()` to make a list of strings

Map, Filter, Reduce

Exercise:  yellkey.com/**type**

Do the TODOs in `catalog.ts` to convert loops & ifs with map/filter/reduce

- Step-by-step refactoring: run `npm test` after doing each TODO

TODO #1: filter a list of integers

TODO #2: `map()` and `concat()` to make a list of strings

TODO #3: use `reduce()` to count a list

Map, Filter, Reduce

Exercise:  yellkey.com/**type**

Do the TODOs in `catalog.ts` to convert loops & ifs with map/filter/reduce

- Step-by-step refactoring: run `npm test` after doing each TODO

TODO #1: filter a list of integers

TODO #2: `map()` and `concat()` to make a list of strings

TODO #3: use `reduce()` to count a list

TODO #4: use `reduce()` to join a list

Map, Filter, Reduce

Exercise:  yellkey.com/**type**

Do the TODOs in `catalog.ts` to convert loops & ifs with map/filter/reduce

- Step-by-step refactoring: run `npm test` after doing each TODO

TODO #1: filter a list of integers

TODO #2: `map()` and `concat()` to make a list of strings

TODO #3: use `reduce()` to count a list

TODO #4: use `reduce()` to join a list

TODO #5: use `flatMap()` to flatten nested lists

Iterables and generators

Sometimes you're not working with arrays.

TODO #6

- remove the `skip` from the range test
- implement `range2()` as a generator function
  - don't use any arrays in the body of `range2()`

Iterables and generators

Sometimes you're not working with arrays.

TODO #6

- remove the `skip` from the range test
- implement `range2()` as a generator function
  - don't use any arrays in the body of `range2()`

TODO #7

- remove the `skip` from the map/filter test
- implement `map()` and `filter()` as generator functions
  - again, don't use arrays

Iterables and generators

Sometimes you're not working with arrays.

TODO #6

- remove the `skip` from the range test
- implement `range2()` as a generator function
  - don't use any arrays in the body of `range2()`

TODO #7

- remove the `skip` from the map/filter test
- implement `map()` and `filter()` as generator functions
  - again, don't use arrays

Finally, update `numberedMajors()` to use `Iterable`:
use `range2()`, the new `filter()`, and return `Iterable` instead of `Array`

Reduce is all you need

Let's use `reduce` to implement `map`

Suppose `arr: Array<T>` and `f: T -> U`

Which of the following is equivalent to **`arr.map(f)`** ? *(pick all good choices)*

```
(A)    arr.reduce( f, [])

(B)    arr.reduce( (a: Array<U>, t: T) => a.concat([f(t)]), [])

(C)    arr.reduce( (a: Array<U>, t: T) => [f(t)].concat(a), [])

(D)    arr.reduce( (a: Array<U>, t: T) => { a.push(f(t)); return a;}, [])
```

Reduce is all you need

Now let's use `reduce` to implement `filter`

Suppose `arr: Array<T>` and `f: T -> boolean`

Which of the following is equivalent to **`arr.filter(f)`**? *(pick all good choices)*

```
(A)    arr.reduce( (a: Array<T>, t: T) => f(t) ? a.concat([t]) : a, [])

(B)    arr.reduce( (a: Array<boolean>, t: T) => a.concat([f(t)]), [])

(C)    arr.reduce( (a: Array<T>, t: T) => {
               if (f(t)) {
                   a.push(t);
               }
       }, [])

(D)    arr.reduce( (a: Array<T>, t: T) => f(t) ? t : undefined, [])
```