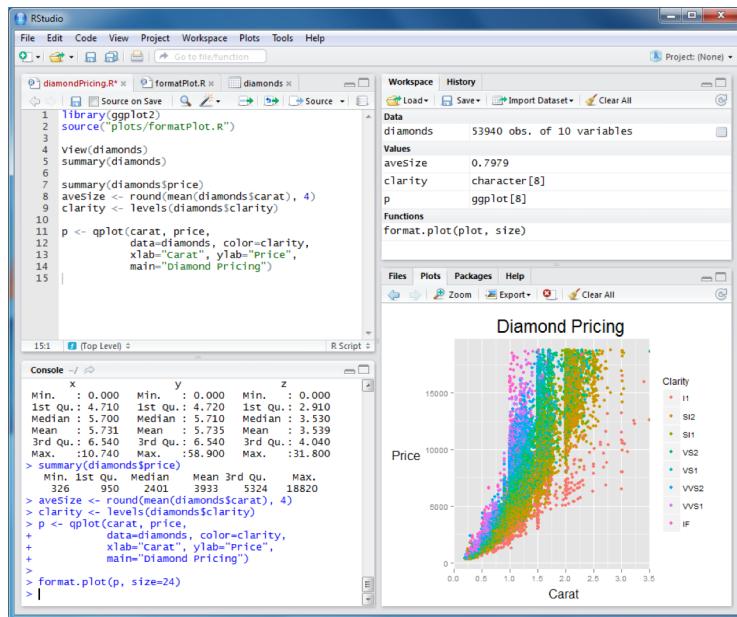


COMPSCI 220

Programming Methodology

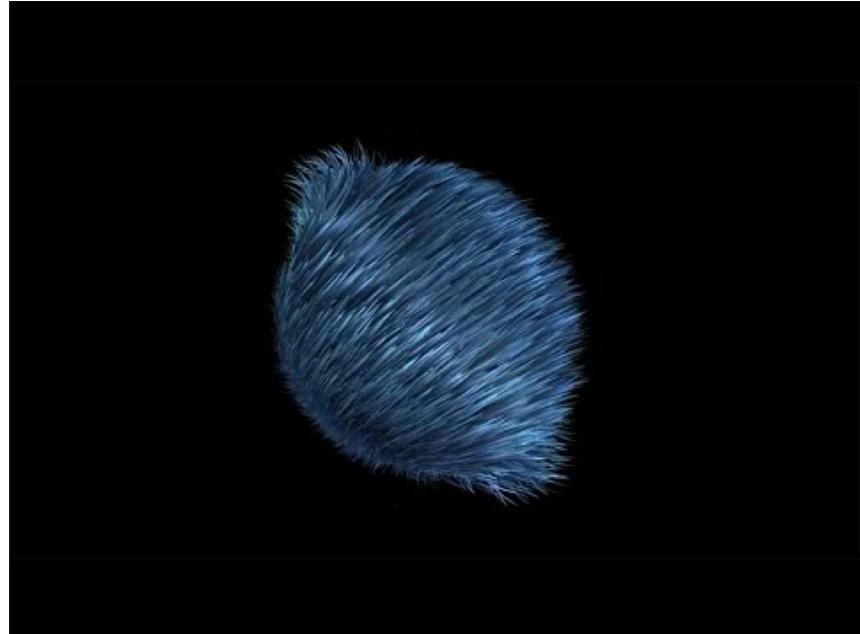
Slides credit: Arjun Guha

Different Tasks => Different Languages

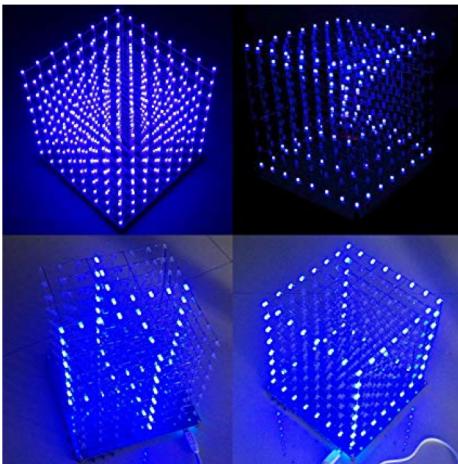


RStudio interface showing R code and a scatter plot:

```
1 library(ggplot2)
2 source("plots/formatPlot.R")
3
4 view(diamonds)
5 summary(diamonds)
6
7 summary(diamonds$price)
8 athesize <- round(mean(diamonds$carat), 4)
9 clarity <- levels(diamonds$clarity)
10
11 p <- qplot(carat, price,
12             data=diamonds, color=clarity,
13             xlab="Carat", ylab="Price",
14             main="Diamond Pricing")
15
15.1 [Top Level] R Script
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259.1 [Top Level] R Script
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
329.1 [Top Level] R Script
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
349.1 [Top Level] R Script
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
369.1 [Top Level] R Script
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
389.1 [Top Level] R Script
390
391
392
393
394
395
396
397
398
399
399.1 [Top Level] R Script
399.2 [Top Level] R Script
400
401
402
403
404
405
406
407
408
409
409.1 [Top Level] R Script
410
411
412
413
414
415
416
417
418
419
419.1 [Top Level] R Script
420
421
422
423
424
425
426
427
428
429
429.1 [Top Level] R Script
430
431
432
433
434
435
436
437
438
439
439.1 [Top Level] R Script
440
441
442
443
444
445
446
447
448
449
449.1 [Top Level] R Script
450
451
452
453
454
455
456
457
458
459
459.1 [Top Level] R Script
460
461
462
463
464
465
466
467
468
469
469.1 [Top Level] R Script
470
471
472
473
474
475
476
477
478
479
479.1 [Top Level] R Script
480
481
482
483
484
485
486
487
488
489
489.1 [Top Level] R Script
490
491
492
493
494
495
496
497
498
499
499.1 [Top Level] R Script
500
501
502
503
504
505
506
507
508
509
509.1 [Top Level] R Script
510
511
512
513
514
515
516
517
518
519
519.1 [Top Level] R Script
520
521
522
523
524
525
526
527
528
529
529.1 [Top Level] R Script
530
531
532
533
534
535
536
537
538
539
539.1 [Top Level] R Script
540
541
542
543
544
545
546
547
548
549
549.1 [Top Level] R Script
550
551
552
553
554
555
556
557
558
559
559.1 [Top Level] R Script
560
561
562
563
564
565
566
567
568
569
569.1 [Top Level] R Script
570
571
572
573
574
575
576
577
578
579
579.1 [Top Level] R Script
580
581
582
583
584
585
586
587
588
589
589.1 [Top Level] R Script
590
591
592
593
594
595
596
597
598
599
599.1 [Top Level] R Script
600
601
602
603
604
605
606
607
608
609
609.1 [Top Level] R Script
610
611
612
613
614
615
616
617
618
619
619.1 [Top Level] R Script
620
621
622
623
624
625
626
627
628
629
629.1 [Top Level] R Script
630
631
632
633
634
635
636
637
638
639
639.1 [Top Level] R Script
640
641
642
643
644
645
646
647
648
649
649.1 [Top Level] R Script
650
651
652
653
654
655
656
657
658
659
659.1 [Top Level] R Script
660
661
662
663
664
665
666
667
668
669
669.1 [Top Level] R Script
670
671
672
673
674
675
676
677
678
679
679.1 [Top Level] R Script
680
681
682
683
684
685
686
687
688
689
689.1 [Top Level] R Script
690
691
692
693
694
695
696
697
698
699
699.1 [Top Level] R Script
700
701
702
703
704
705
706
707
708
709
709.1 [Top Level] R Script
710
711
712
713
714
715
716
717
718
719
719.1 [Top Level] R Script
720
721
722
723
724
725
726
727
728
729
729.1 [Top Level] R Script
730
731
732
733
734
735
736
737
738
739
739.1 [Top Level] R Script
740
741
742
743
744
745
746
747
748
749
749.1 [Top Level] R Script
750
751
752
753
754
755
756
757
758
759
759.1 [Top Level] R Script
760
761
762
763
764
765
766
767
768
769
769.1 [Top Level] R Script
770
771
772
773
774
775
776
777
778
779
779.1 [Top Level] R Script
780
781
782
783
784
785
786
787
788
789
789.1 [Top Level] R Script
790
791
792
793
794
795
796
797
798
799
799.1 [Top Level] R Script
800
801
802
803
804
805
806
807
808
809
809.1 [Top Level] R Script
810
811
812
813
814
815
816
817
818
819
819.1 [Top Level] R Script
820
821
822
823
824
825
826
827
828
829
829.1 [Top Level] R Script
830
831
832
833
834
835
836
837
838
839
839.1 [Top Level] R Script
840
841
842
843
844
845
846
847
848
849
849.1 [Top Level] R Script
850
851
852
853
854
855
856
857
858
859
859.1 [Top Level] R Script
860
861
862
863
864
865
866
867
868
869
869.1 [Top Level] R Script
870
871
872
873
874
875
876
877
878
879
879.1 [Top Level] R Script
880
881
882
883
884
885
886
887
888
889
889.1 [Top Level] R Script
890
891
892
893
894
895
896
897
898
899
899.1 [Top Level] R Script
900
901
902
903
904
905
906
907
908
909
909.1 [Top Level] R Script
910
911
912
913
914
915
916
917
918
919
919.1 [Top Level] R Script
920
921
922
923
924
925
926
927
928
929
929.1 [Top Level] R Script
930
931
932
933
934
935
936
937
938
939
939.1 [Top Level] R Script
940
941
942
943
944
945
946
947
948
949
949.1 [Top Level] R Script
950
951
952
953
954
955
956
957
958
959
959.1 [Top Level] R Script
960
961
962
963
964
965
966
967
968
969
969.1 [Top Level] R Script
970
971
972
973
974
975
976
977
978
979
979.1 [Top Level] R Script
980
981
982
983
984
985
986
987
988
989
989.1 [Top Level] R Script
990
991
992
993
994
995
996
997
998
999
999.1 [Top Level] R Script
999.2 [Top Level] R Script
```



Arduino-C



GLShader



New Languages Show Up At An Unprecedented Rate

The screenshot shows the Swift programming language page on the Apple Developer website. The header includes the Apple logo and navigation links for Developer, Technologies, Resources, Programs, Support, and Member Center. Below the header, there's a large orange banner featuring the Swift logo and the text "A new programming language for iOS and OS X.". Underneath the banner, there are three navigation links: Overview, Blog, and Resources. The main content section is titled "Introducing Swift" and contains a paragraph about Swift being an innovative new programming language for Cocoa and Cocoa Touch. It highlights the language's interactivity, conciseness, expressiveness, and speed. A code editor window is shown with some Swift code, and a "Run" button is visible.

The screenshot shows the Go Programming Language website. The header features the Go logo and navigation links for Documents, Packages, The Project, Help, Blog, and Search. The main content area has a "Try Go" section with a code editor containing a simple Go program. To the right, there's a text block about Go being an open-source language for building reliable software, followed by a large cartoon gopher character. Below the gopher is a "Download Go" section with a link to binary distributions for Linux, Mac OS X, and Windows. There are also "Featured video" and "Featured articles" sections.

The screenshot shows the Hack programming language website. The header includes the Hack logo and navigation links for INSTALL, TUTORIAL, COOKBOOK, DOCS, GITHUB, and HHVM. The main content features a large orange banner with the text "Programming productivity without breaking things". Below the banner is a code editor showing a snippet of Hack code. The footer contains sections for "What is Hack?", "Community" (links to Twitter, HHVM Blog, IRC Chat, and Hack Dev Day 2014), and a "PLAYLIST" section with a video thumbnail of a speaker.

The screenshot shows the Rust Programming Language website. The header includes the Rust logo and navigation links for Docs (Nightly), Docs (0.11.0), and Community. The main content area features a "Rust" logo and a brief description of Rust as a systems programming language that runs fast and prevents crashes. Below this is a "Show me more!" button. The "Featuring" section lists various Rust features like algebraic data types, pattern matching, closures, type inference, zero-cost abstractions, guaranteed memory safety, and concurrency. To the right, there's a code editor with a snippet of Rust code for a simple integer calculator, a "Run" button, and a "Recommended Version: nightly (Mac installer)" link. There are also "Install" and "Other Downloads" buttons.

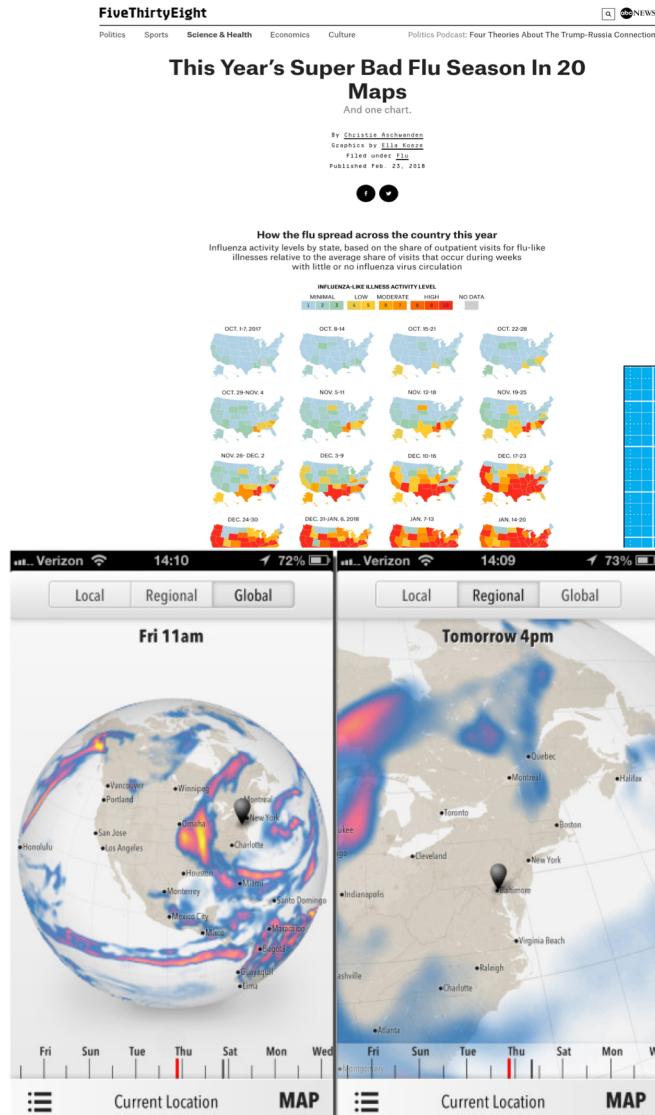
Our Approach in COMPSCI 220

- You will encounter many new languages and will need to master quite a few to succeed in computer science
- We don't have time to teach and learn all of them

Thus, we will learn:

- **Principles** that translate across any modern language, including Java, C++, Scala, Go, Swift
- How to reason about programs for **correctness**, ease of **debugging**, and **maintainability**

JavaScript Is Everywhere



The 220 web-based IDE: Ocelot

FILES ▶ RUN A TEST ■ STOP ⏪ DOWNLOAD > CONSOLE CANVAS HISTORY

jbiswas@umass.edu SIGN OUT

All Changes Saved

+ NEW

- <> assignment1.js
- <> assignment2.js
- <> classes.js
- <> errorTesting.js
- <> images.js
- <> lecture1.js
- <> lecture2.js
- <> lecture3.js
- <> mergeSort.js
- <> scoping.js
- <> sortReduce.js
- <> test.js
- <> test2.js
- <> testArray.js
- <> untitled.js
- <> imageDemo.js

```
1 let I = lib220.loadImageFromURL(
2   "https://people.cs.umass.edu/~joydeepb/robot.jpg");
3 I.show();
4
5 function scramble(img) {
6   for (let y = 0; y < img.height; ++y) {
7     for (let x = 0; x < img.width; ++x) {
8       let c = img.getPixel(x, y);
9       img.setPixel(x, y, [c[2], c[1], c[0]]);
10    }
11  }
12  return img;
13 }
14
15 scramble(I).show();
```



Starting program...
Program terminated normally.
8/22/2018, 9:34:08 PM EDT
Compiling...
Compilation successful.
Starting program...
Program terminated normally.

A Modern Take On JavaScript

COMPSCI 220-JS: Design to match other languages

1. JavaScript has been around for > 20 years

Many old ways of programming still exist

New features, abstractions, and capabilities have been continuously added

2. COMPSCI 220: An emphasis on modern JavaScript

Checks will disallow older features that have been superseded

Known bad coding practices are explicitly prohibited

3. Arbitrary JS code from the internet may not compile

Much of the internet uses older JS features, and often poor coding practices

You should not be using code from the internet for assignments anyway

4. Anything you write in Ocelot will work outside class

JS: Basic Syntax; Using the Console

```
let a = 42; // Declare a variable, and assign it a number.  
let b = 'Hello World'; // A string.  
let c = [0, 1, 4, 9]; // An array of numbers.  
const pi = 3.14159265359; // A constant.  
console.log('The value of pi is ' + pi.toString()); // Print to the console.  
console.log(c); // The console can display objects.  
let d; // Will not compile: need to initialize value.  
  
// The === operator in JS is equivalent to the == in other languages.  
if (1 === 1) {  
    console.log('All is good.');//  
}  
  
function factorial(x) {  
    let y = 1;  
    for (let z = 1; z <= x; ++z) {  
        y = y * z;  
    }  
    return y;  
}
```

Types in JavaScript

Java	JavaScript
<pre>int a = 2; float b = 1.3; double c = 1.3; String d = "Hello world";</pre>	<pre>let a = 2; // All numbers are doubles let b = 1.3; let c = 1.3; let d = "Hello world";</pre>

1. *Static types*: In Java, you have to specify the types of variables and methods. The Java compiler catches type errors before you run the program.
2. *Dynamic types*: In JavaScript, you do not have to specify the types in the program. The JavaScript evaluator catches type errors when the program is running.
3. **Note**: JavaScript has types.

```
> typeof 1
"number"
> typeof 2.3
"number"
> typeof "Hello, world!"
"String"
> typeof true
"boolean"
```

Note: all numbers have the type "number", which is equivalent to the type double in Java.

Some Small *Gotcha* Points About JS

1. Instead of "==" , use "===".

Instead of "!=" , use "!=="

JavaScript does not treat "==" and "!=" the same way as other languages.
We will not use this feature: remember, good programming practices.

2. The type of a variable can change

Do not deliberately use this feature. It hampers readability, and may lead to bugs.

```
let a = 2;  
a = "Hello world";
```

3. The value undefined has type "undefined". It is similar to null in Java.

Introduction to higher-order functions

We introduce several canonical *higher-order functions* over arrays.

- We derive map by abstracting out the differences between two first-order functions
- We generalize the type of map
- We derive filter in a similar way
- As an exercise, we derive map2

Three different functions

Problem Statement: Write a function that consumes an array of numbers and produces an array of numbers. Each element of the output array should be *the double of* the corresponding element in the input array.

Example:

```
doubleAll([10, 5, 2])  
// [20, 10, 4]
```

Type Signature:

```
doubleAll(  
  a: number[]): number[]
```

Problem Statement: Write a function that consumes an array of numbers and produces an array of numbers. Each element of the output array should be *the reciprocal of* the corresponding element in the input array.

Example:

```
recipAll([10, 5, 2])  
// [0.1, 0.2, 0.5]
```

Type Signature:

```
recipAll(  
  a: number[]): number[]
```

Problem Statement: Write a function that consumes an array of numbers and produces an array of numbers. Each element of the output array should be *zero if the corresponding element in the input is negative, or the same as the corresponding element if it is non-negative*.

Examples:

```
clipAll([-1, 5, -3])  
// [0, 5, 0]
```

Type Signature:

```
clipAll(  
  a: number[]): number[]
```

Solutions

```
// doubleAll(a: number[]): number[]
function doubleAll(a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    result.push(2 * a[i]);
  }
  return result;
}

// clipAll(a: number[]): number[]
function clipAll(a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    if (a[i] < 0) {
      result.push(0);
    } else {
      result.push(a[i]);
    }
  }
  return result;
}
```

```
// recipAll(a: number[]): number[]
function recipAll(a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    result.push(1 / a[i]);
  }
  return result;
}
```

1. Note: these functions produce a new array and do not update the input array in-place. (See the slides on Unit Testing for why.)
2. These three functions are almost identical. So, they have a lot of duplicated code. Code duplication is bad. How can we address this?

Step 1: Identify exactly what is different

```
// doubleAll(a: number[]): number[]  
function doubleAll(a) {  
  let result = [];  
  for (let i = 0; i < a.length; ++i) {  
    result.push(2 * a[i]);  
  }  
  return result;  
}
```

```
// recipAll(a: number[]): number[]  
function recipAll(a) {  
  let result = [];  
  for (let i = 0; i < a.length; ++i) {  
    result.push(1 / a[i]);  
  }  
  return result;  
}
```

Ignore clipAll for a moment.

What is different about about
doubleAll and recipAll?

The only difference between them is
what they do to each element of the
array.

Step 2: Abstract out the difference

```
// double(x: number): number
function double(x) {
  return 2 * x;
}

// doubleAll(a: number[]): number[]
function doubleAll(a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    result.push(double(a[i]));
  }
  return result;
}
```

```
// recip(x: number): number
function recip(x) {
  return 1 / x;
}

// recipAll(a: number[]): number[]
function recipAll(a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    result.push(recip(a[i]));
  }
  return result;
}
```

Definition: to abstract something out means to write a helper function that performs the operation.

At this point, the only difference between doubleAll and recipAll is which helper function they apply.

Step 3: Make the helper function an argument

```
// double(x: number): number
function double(x) {
  return 2 * x;
}
```

```
// recip(x: number): number
function recip(x) {
  return 1 / x;
}
```

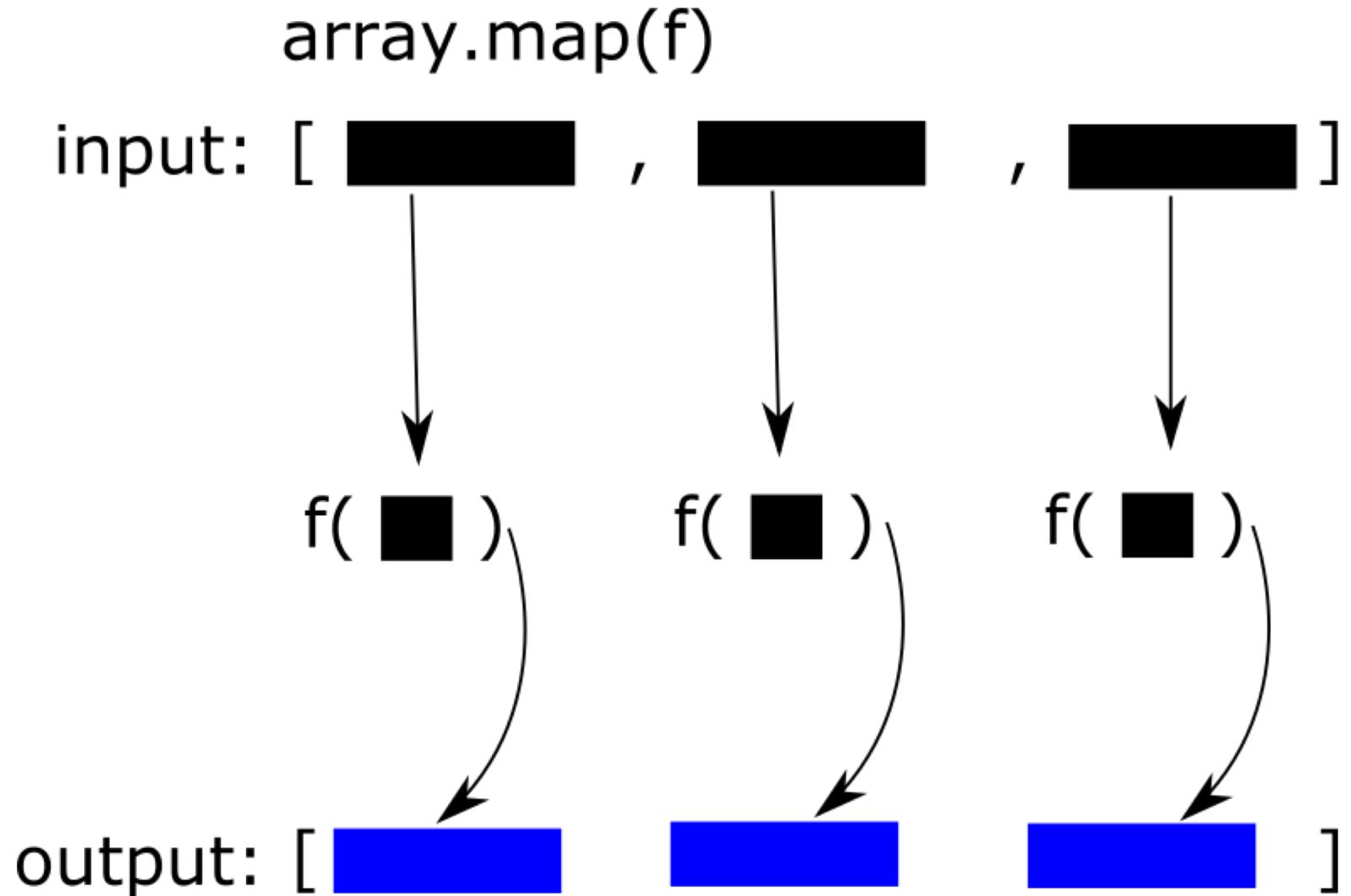
```
// map(f: (x : number) => number, a: number[]): number[]
function map(f, a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    result.push(f(a[i]));
  }
  return result;
}
```

A *higher-order function* is a function that takes another function as an argument.

```
// doubleAll(a: number[]): number[]
function doubleAll(a) {
  return map(double, a);
}
```

```
// recipAll(a: number[]): number[]
function recipAll(a) {
  return map(recip, a);
}
```

Map, pictorially



Can we write clipAll using map?

map applies the same function f to each element of the array. clipAll appears to do two different things to each element.

```
// clipAll(a: number[]): number[]
function clipAll(a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    if (a[i] < 0) {
      result.push(0);
    } else {
      result.push(a[i]);
    }
  }
  return result;
}
```

```
// map(f: (x : number) => number,
//       a: number[]): number[]
function map(f, a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    result.push(f(a[i]));
  }
  return result;
}
```

```
// clip(x: number): number
function clip(x) {
  if (x < 0) {
    return 0;
  } else {
    return x;
  }
}

// clipAll(a: number[]): number[]
function clipAll(a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    result.push(clip(a[i]));
  }
  return result;
}
```

At this point, it is easy to use map. But, it wasn't obvious to start.

Another example with map

Problem Statement: Write a function that consumes an array of string and produces an array of numbers. Each element of the output array should be *the length of the* corresponding string in the input array.

Example:

```
lengthAll(["xy", "x"])
// [2, 1]
```

Type Signature:

```
lengthAll(
  a: string[]): number[]
```

Can we write this using map? We said earlier that the argument f to map has the type (x: number) => number.

```
// length(s: string): number
function length(s) {
  return s.length;
}

// lengthAll(a: string[]): number[]
function lengthAll(a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    result.push(length(a[i]));
  }
  return result;
}
```

```
// map(f: (x : number) => number, a: number[]): number[]
function map(f, a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    result.push(f(a[i]));
  }
  return result;
}
```

Generalizing the type of map

```
// map<S,T>(f: (x : S) => T, a: S[]): T[]
function map(f, a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    result.push(f(a[i]));
  }
  return result;
}
```

The type of values produced by f are the same as the type of element in the output array.

The type of each element of a must be the same as the type of argument to f .

Three more functions

Problem Statement: Write a function that consumes an array of numbers and produces an array of the even numbers in the input array.

Example:

```
evens([10, 5, 2])  
// [10, 4]
```

Type Signature:

```
evens(  
  a: number[]): number[]
```

Problem Statement: Write a function that consumes an array of numbers and produces an array of the positive numbers in the input array.

Example:

```
odds([10, 5, 2])  
// [5]
```

Type Signature:

```
odds(  
  a: number[]): number[]
```

Problem Statement: Write a function that consumes an array of strings and produces an array of the non-empty strings in the input array.

Examples:

```
nonEmpty(["hi", ""])  
// ["hi"]
```

Type Signature:

```
nonEmpty(  
  a: string[]): string[]
```

Step 1. Write the functions and identify the differences

```
// evens(a: number[]): number[]
function evens(a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    let x = a[i];
    if (x % 2 === 0) {
      result.push(x);
    }
  }
  return result;
}
```

```
// odds(a: number[]): number[]
function odds(a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    let x = a[i];
    if (x % 2 === 1) {
      result.push(x);
    }
  }
  return result;
}
```

Note: We cannot write these using map, because the length of the output array is not the same as the length of the input array.

Step 2. Abstract out the differences

```
// isEven(x: number): boolean
function isEven(x) {
  return x % 2 === 0;
}

// evens(a: number[]): number[]
function evens(a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    let x = a[i];
    if (isEven(x)) {
      result.push(x);
    }
  }
  return result;
}
```

```
// isOdd(x: number): boolean
function isOdd(x) {
  return x % 2 === 0;
}

// odds(a: number[]): number[]
function evens(a) {
  let result = [];
  for (let i = 0; i < a.length; ++i) {
    let x = a[i];
    if (isOdd(x)) {
      result.push(x);
    }
  }
  return result;
}
```

At this point, the only difference between `evens` and `odds` is which helper function they apply.

Step 3: Make the helper function an argument

```
// isEven(x: number): boolean  
function isEven(x) {  
    return x % 2 === 0;  
}
```

```
// isOdd(x: number): boolean  
function isOdd(x) {  
    return x % 2 === 0;  
}
```

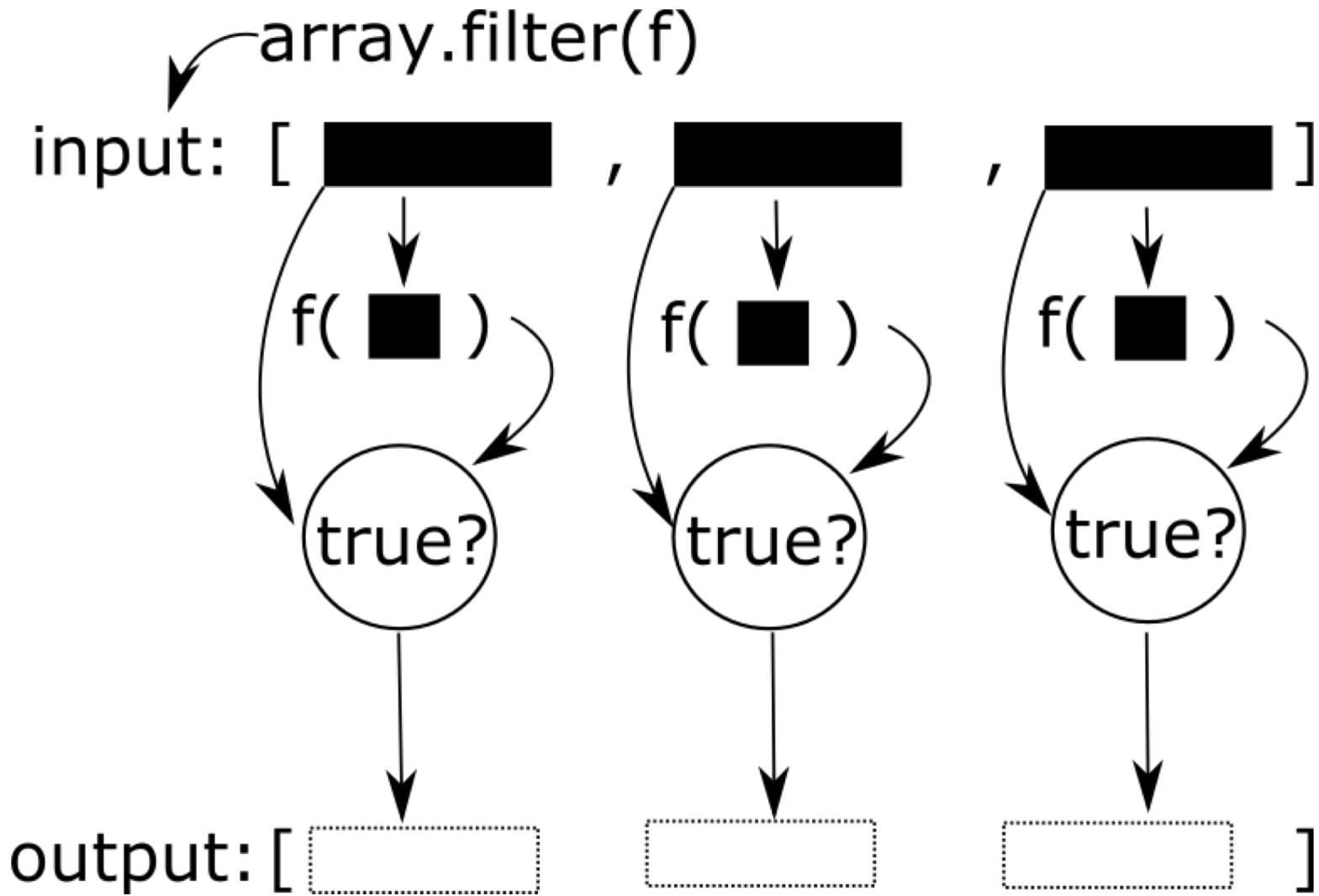
```
// filter<T>(f: (x : T) => boolean, a: T[]): T[]  
function filter(f, a) {  
    let result = [];  
    for (let i = 0; i < a.length; ++i) {  
        let x = a[i];  
        if (f(x)) {  
            result.push(x);  
        }  
    }  
    return result;  
}
```

Note that filter does not require the argument to *f* to be a function.

```
// evens(a: number[]): number[]  
function evens(a) {  
    return filter(isEven, a);  
}
```

```
// odds(a: number[]): number[]  
function odds(a) {  
    return filter(isOdd, a);  
}
```

Filter



Exercise: two more functions

Problem Statement: Write a function that consumes two array of numbers and produces an array of numbers, where each element is the sum of the corresponding numbers in the input array.

Problem Statement: Write a function that consumes two array of numbers and produces an array of numbers, where each element is the product of the corresponding numbers in the input array.

Try to derive a higher-order function that you can use to implement both of these functions systematically:

1. Write down some example inputs and outputs for each function.
2. Write the type signature of each function.
3. Implement each function without using any higher-order functions.
4. Abstract out the difference between them into a helper function.
5. Make the helper function an argument.

Built-in higher-order functions

The JavaScript standard library has several built-in higher-order functions.

The built-in map function

🔗 Syntax

```
var new_array = arr.map(function callback(currentValue[, index[, array]]) {  
    // Return element for new_array  
}[, thisArg])
```

🔗 Parameters

callback

Function that produces an element of the new Array, taking three arguments:

currentValue

The current element being processed in the array.

index | Optional

The index of the current element being processed in the array.

array | Optional

The array `map` was called upon.

thisArg | Optional

Value to use as `this` when executing `callback`.

🔗 Return value

A new array with each element being the result of the callback function.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

The built-in map function

⌚ Syntax

```
let | const new_array = arr.map(function callback(currentValue[, index[, array]]) {  
    // Return element for new_array  
}[, thisArg])
```

⌚ Parameters

callback

Function that produces an element of the new Array, taking three arguments:

currentValue

The current element being processed in the array.

index | Optional

The index of the current element being processed.

array | Optional

The array map was called upon.

thisArg | Optional

Value to use as `this` when executing callback.

Do not use these. The map function callback should not use these.

Recall: Emphasis on good programming practices

⌚ Return value

A new array with each element being the result of the callback function.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

How to Read the Documentation

1. Feel free to try out the many built-in functions

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/

2. Do not use 'var'

Use 'let' or 'const'

Why? 'var' has unusual scoping rules. Will see after lecture 4

3. Do not use the optional arguments

- Some are JavaScript's poetic license at re-implementing well-understood standard HOFs (e.g. map, reduce, filter)
- Some are just bad programming practices.

4. Remember: Check in Ocelot

!!! Does not run in Ocelot === 0 on assignment

More Built-In HOFs: forEach(), filter()

```
// Print all entries in array
console.log("Print all entries:");
[1, 2, 3, 4].forEach(
  function(x) {
    console.log(x);
  }
);

// Remember, HOFs take in functions: these can also be built-in functions.
[1, 2, 3, 4].forEach(console.log);

// Filter arrays
console.log("Filter array:");
console.log([1, 2, 3, 4, 5, 6, 7].filter(
  function(x) {
    return (x % 2 === 0);
  }
));
```

Dealing with errors

Unfortunately, if you program uses higher-order functions has a bug, you will get some inscrutable error messages from JavaScript.

Common Errors in Calling HOFs

1. Calling a function instead of passing it as a value

```
[1, 2, 3, 4].forEach(console.log(x)); // Incorrect  
[1, 2, 3, 4].forEach(function(x) { console.log(x); }); // Correct
```

2. Mixing up parameters passed to HOF, vs. parameters passed to function, passed to HOF

```
[1, 2, 3, 4].forEach(x, function() { console.log(x); });  
[1, 2, 3, 4].forEach(function(x) { console.log(x); });
```

3. Return type of function passed to HOF is incorrect

```
[1, 2, 3, 4].filter(function(x) { return '0'; });  
[1, 2, 3, 4].filter(function(x) { return false; });
```

4. Function passed to HOF accepts incorrect number of parameters

```
[1, 2, 3, 4].map(function() { return 0; });  
[1, 2, 3, 4].map(function(x) { return 0; });
```

5. Function passed to HOF accepts incorrect types of parameters

```
[1, 2, 3, 4].forEach(function(x) { console.log(x[0]); });  
[1, 2, 3, 4].forEach(function(x) { console.log(x); });
```

Dealing With Error Messages: THINK Before you Act!

```
function reciprocal(x) {  
  return 1/x;  
}  
console.log([1, 2, 3, 4].map(reciprocal(x)));
```

> x is not defined at Line 4

```
function reciprocal(x) {  
  return 1/x;  
}  
let x = 0;  
console.log([1, 2, 3, 4].map(reciprocal(x)));
```

> f is not a function at Line 201: in (anonymous function)
... Line 5

Dealing With Error Messages: THINK Before you Act!

```
function reciprocal(x) {  
    return 1/x;  
}  
function f(x) {  
}  
let x = 0;  
console.log([1, 2, 3, 4].map(reciprocal(x)));
```

```
> f is not a function at Line 201: in (anonymous function)  
... Line 7
```

```
function apply(x, f) {  
    let result = f(x);  
    return result;  
}  
apply(2, 3);
```

```
> f is not a function at Line 2: in apply  
... Line 5
```

Exercise: Fix this error

Question: Using map, write a new function called zeroClone() to create a new array of the same length as an array provided, but initialized all to zeroes.

```
function zeroClone(a) {  
  let a2 = a.map(function() { return 0; });  
  return a2;  
}  
console.log(zeroClone([1, 2, 3, 4]));
```

**function (anonymous) expected 0 arguments but received 1 argument
at Line 2: in (anonymous function)
... Line 201: in (anonymous function)
... Line 2: in zeroClone
... Line 5**