

Today

- Code: house finch & moving *Paramecium*
- Data structures: vectors & indexing
- Repetition structures:
 - counter control using **for**

house finch continued

Paramecium movement
continued

Data structures

- Scalar: a single number
- Vector: multiple numbers, 1 dimension

Data structures

- Scalar: a single number
- Vector: multiple numbers, 1 dimension

Vectors: indexing

MyVec

Element 1	6.115	← MyVec[1]
Element 2	7.726	
Element 3	8.352	← MyVec[3]
etc	6.289	
	1.087	
	7.344	
	2.911	
	3.209	
	5.290	
	4.445	
	2.505	
	4.541	
	5.568	
	6.873	
	5.208	
Element 16	3.631	← MyVec[16]

Each element is a slot in the computer's memory (RAM).

Vectors: indexing

MyVec

Element 1	6.115	← MyVec[0]
Element 2	7.726	
Element 3	8.352	← MyVec[2]
etc	6.289	
	1.087	
	7.344	
	2.911	
	3.209	
	5.290	
	4.445	
	2.505	
	4.541	
	5.568	
	6.873	
	5.208	
Element 16	3.631	← MyVec[15]

Python and C use
offset indexing

0 is first element

R: `for` repetition structure

Most programming languages have a specialized structure for **counter-controlled repetition** (usually called **for**)

```
for ( i in 1:n ) {  
    expression  
}
```


R: `for` repetition structure

Example

```
for ( i in 1:10 ) {  
  j <- i * 2  
  print(j)  
}
```

What does this do?

The 4 components of counter control using `while` or `for`

Counter

```
i <- 0  
while ( i < n ) {  
  expression  
  i <- i + 1  
}
```

Counter initialized to 0

Number of reps

Counter incremented by 1

Counter

```
for ( i in 1:n ) {  
  expression  
}
```

Counter initialized to 1

Number of reps

Counter increments by 1

R: `for` repetition structure

Correct

```
for ( i in 1:n ) {  
    expression  
}
```

Incorrect

```
i <- 1  
for ( i in 1:n ) {  
    expression  
    i <- i + 1  
}
```