Today

- Programming focused
- Python lists: indexing, operators
- R vs Python: object referencing behavior
- Data structures overview
- C programming, compiling
- C for repetition structure
- C arrays

Python: `for`

The list iterator checks for an end condition each iteration, which could change, so it's actually sentinel control

```
x = [0, 1]
for item in x:
    x.append(item)
```

lengthen x

This makes an infinite loop!
because the index of item never reaches the end

Py

Python: `for`

The list iterator is sentinel control

Effectively this:

```
x = [0, 1]
i = 0
while i < len(x):
    item = x[i]
    x.append(item)
    i = i + 1
```

never false

Contrast with counter control

```
x = [0, 1]
i = 0
n = len(x)
while i < n:
    item = x[i]
    x.append(item)
    i = i + 1
```

n calc ahead

Py

Python: `for`

Take away:

Mostly we can think of it as counter control even when it's implemented as sentinel control

Python defensive programming:
don't do anything in the loop that would mess with the iterator

Py

List comprehensions

Mini algorithms embedded in a list

```
mylist = list(range(1, 11))
print(mylist)
double_mylist = [2 * x for x in mylist]
print(double_mylist)
```

Py

List indexing

Index from zero

`mylist[0]`	1st element
`mylist[3]`	4th element
`mylist[1][2]`	3rd element of 1st element

List indexing

Ranges include first index, exclude last index

`mylist[0:4]`	1st to 4th element
`mylist[:4]`	same (first 4 elements)
`mylist[-2:]`	last 2 elements
`mylist[1][1:3]`	element 2-3 of 1st element

Py

List indexing

Non-contiguous elements
Use list comprehension

```
[mylist[i] for i in [0, 3, 4]]
```

Py

List operators

They don't do element by element math

Expand or combine lists

```
x = [2] * 10
print(x)
```

```
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

Py

String operators

Lists and strings are from a broader category of objects known as sequences

Operators have same behavior

```
w1 = "Hello"
w2 = "Kitty"
w1 + " " + w2

'Hello Kitty'
```

Py

Sequences are iterable

```
rainbow = "My little pony"
for character in rainbow:
    print(character)
```

Py

Object referencing behavior

See code

Py

Core data structures

C	R	Python
scalar	(scalar)[2]	scalar
array (1D+)	vector	list
(struct)[1]	matrix	tuple
	array	string
	list	dictionary
	data.frame	set
		numpy scalar
		numpy.ndarray
		pandas.DataFrame

[1] Build any data structure [2] Vector of length 1

C programming

See code

Py