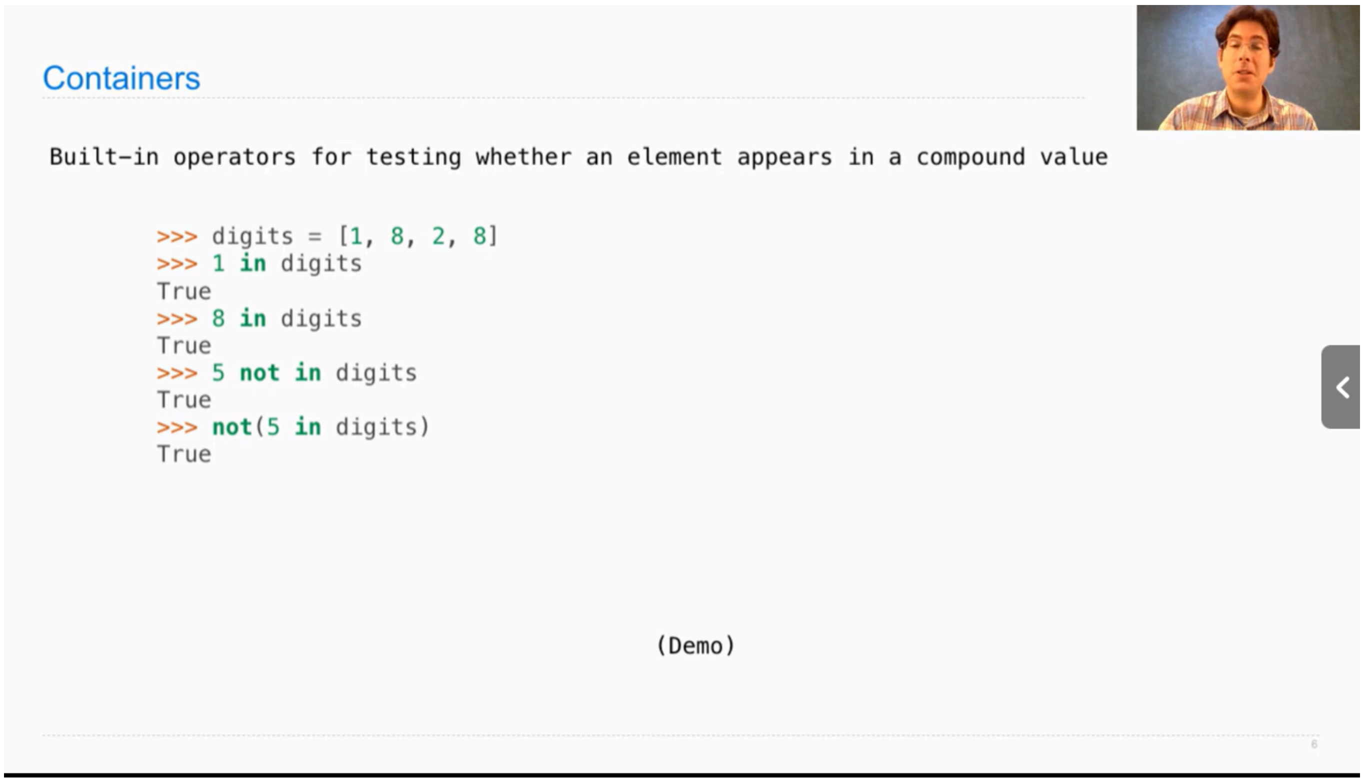
Lecture10 containers

Lists

a built-in data structure in python

```
digits = [1, 8, 2, 8]

digits[0] # 1

>>>[2, 7] + digits * 2
or
>>>add([2, 7], mul(digits, 2))
[2, 7, 1, 8, 2, 8, 1, 8, 2, 8]

pairs = [[10, 20], [30, 40]]
pairs[0][1] = 20
```

Containers



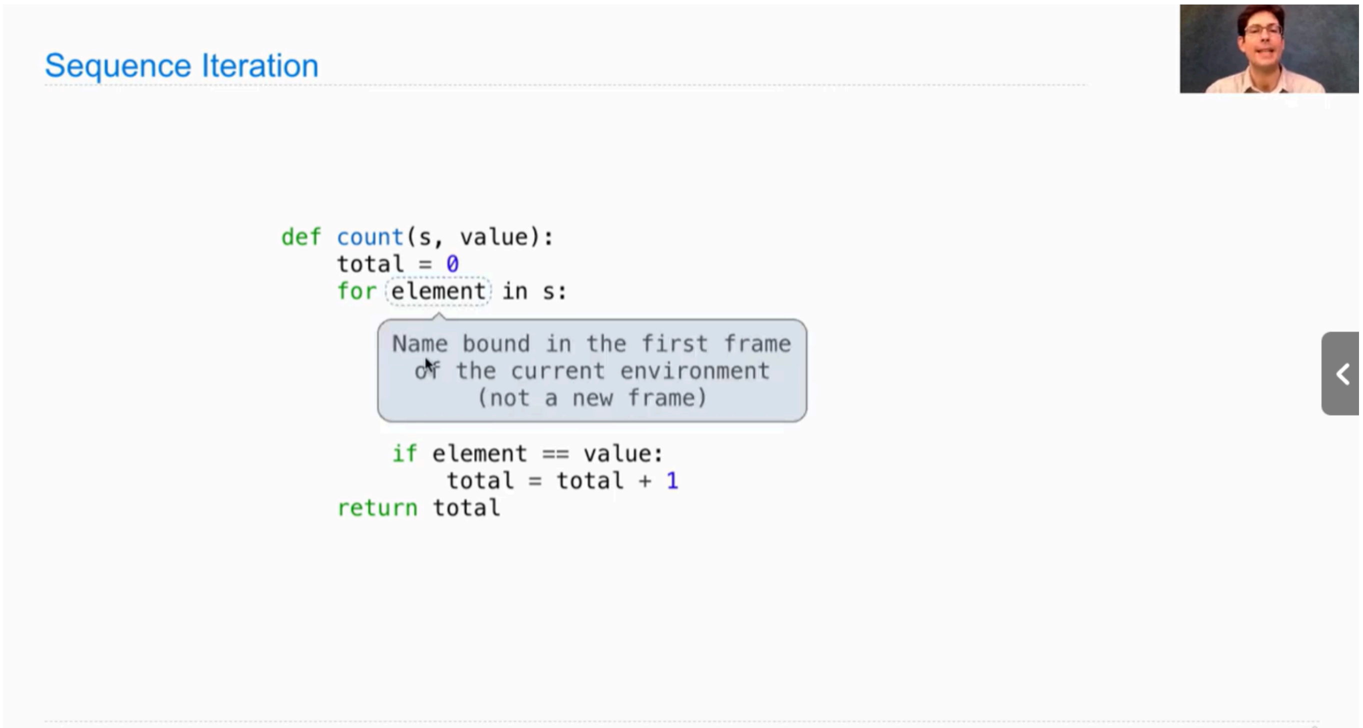
Containers

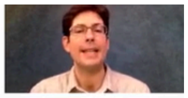
Built-in operators for testing whether an element appears in a compound value

```
>>> digits = [1, 8, 2, 8]
>>> 1 in digits
True
>>> 8 in digits
True
>>> 5 not in digits
True
>>> not(5 in digits)
True
```

(Demo)

For Statements



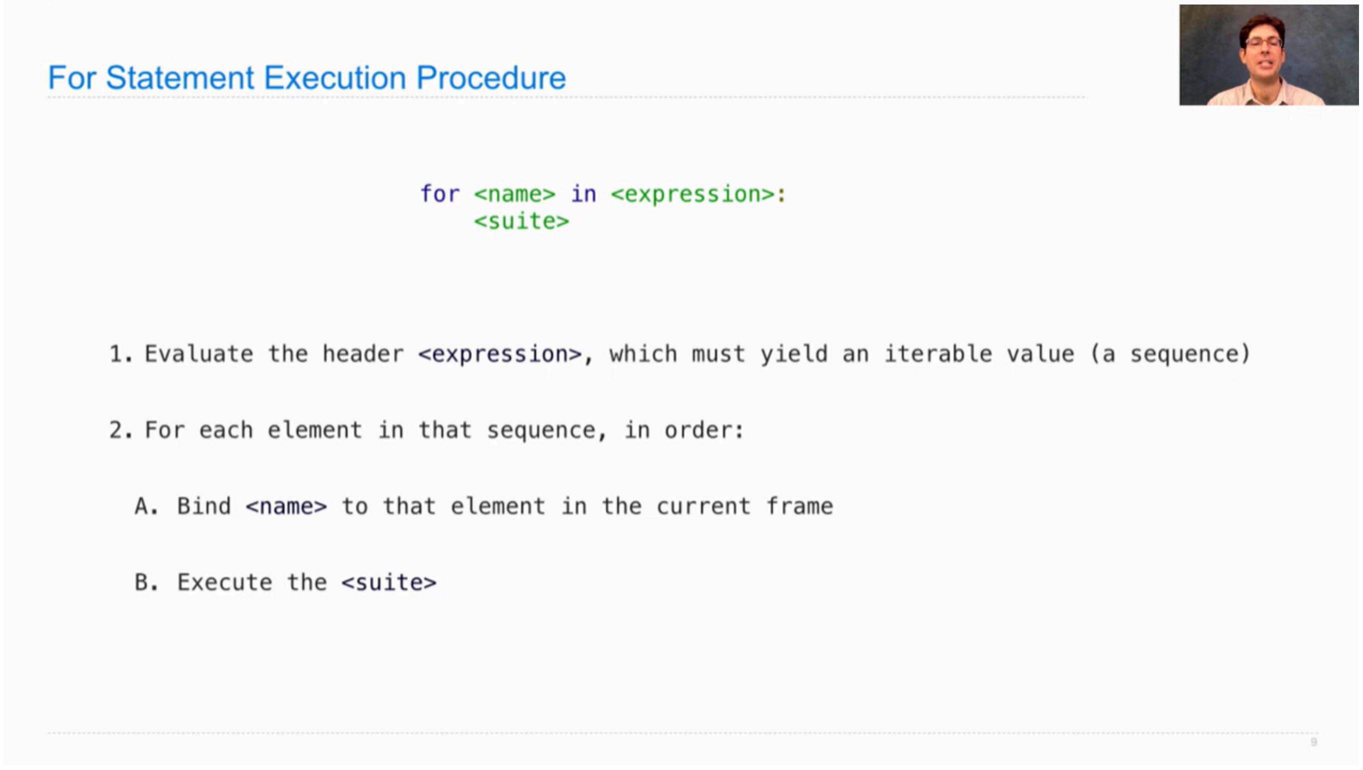


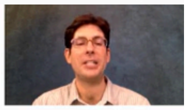
Sequence Iteration

```
def count(s, value):
    total = 0
    for element in s:
```

Name bound in the first frame of the current environment (not a new frame)

```
        if element == value:
            total = total + 1
    return total
```





For Statement Execution Procedure

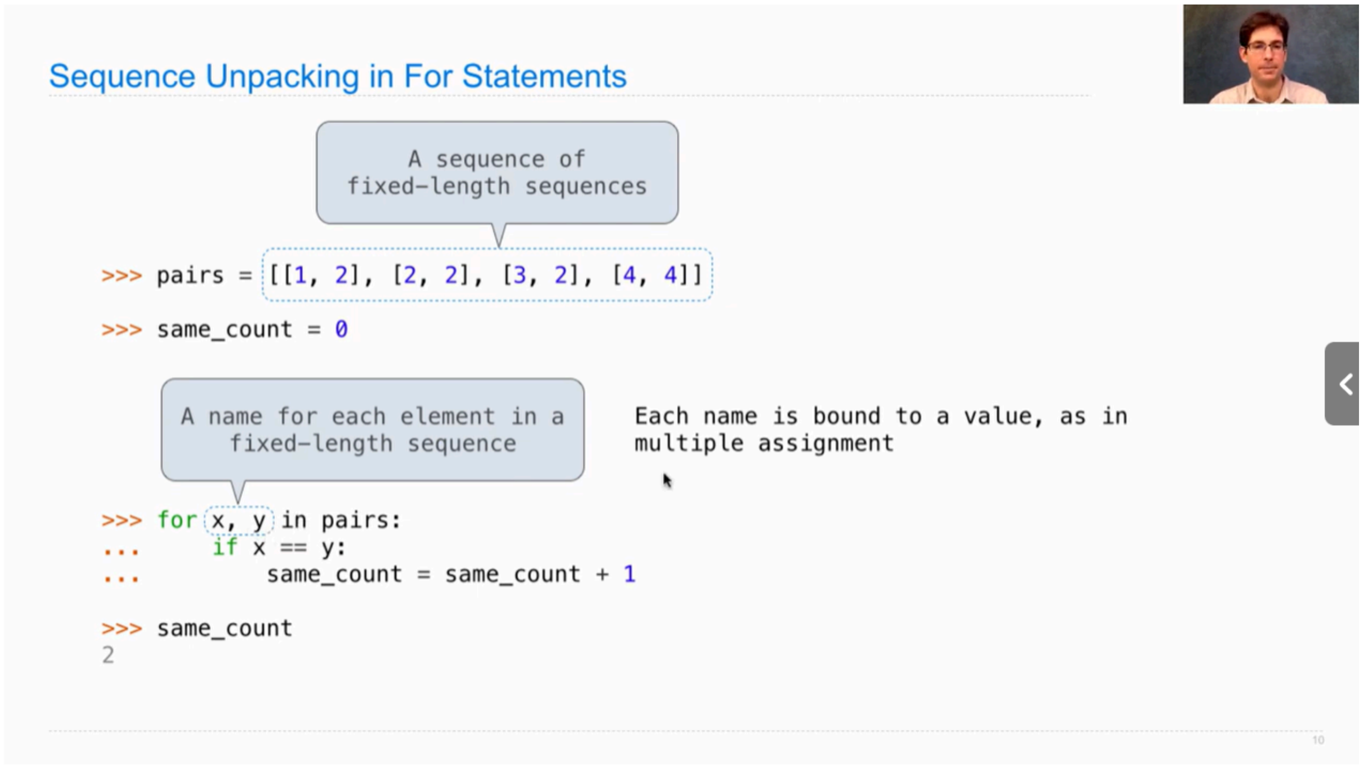
```
for <name> in <expression>:
    <suite>
```

1. Evaluate the header `<expression>`, which must yield an iterable value (a sequence)

2. For each element in that sequence, in order:

   A. Bind `<name>` to that element in the current frame

   B. Execute the `<suite>`



Sequence Unpacking in For Statements

A sequence of fixed-length sequences

```
>>> pairs = [[1, 2], [2, 2], [3, 2], [4, 4]]

>>> same_count = 0
```

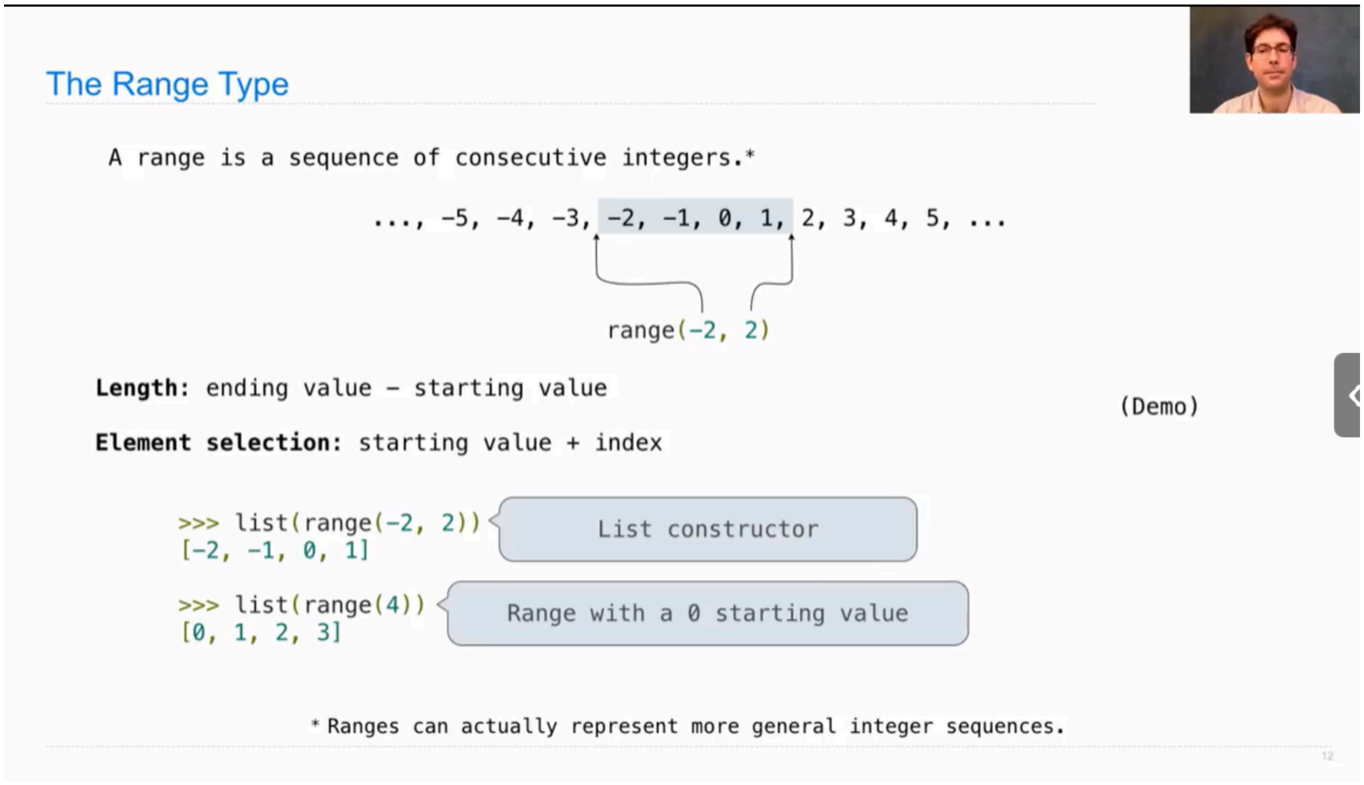
A name for each element in a fixed-length sequence

Each name is bound to a value, as in multiple assignment

```
>>> for x, y in pairs:
...     if x == y:
...         same_count = same_count + 1

>>> same_count
2
```

10

Ranges



The Range Type

A range is a sequence of consecutive integers.*

..., -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, ...

range(-2, 2)

Length: ending value - starting value

Element selection: starting value + index

(Demo)

```
>>> list(range(-2, 2))
[-2, -1, 0, 1]

>>> list(range(4))
[0, 1, 2, 3]
```

List constructor

Range with a 0 starting value

* Ranges can actually represent more general integer sequences.

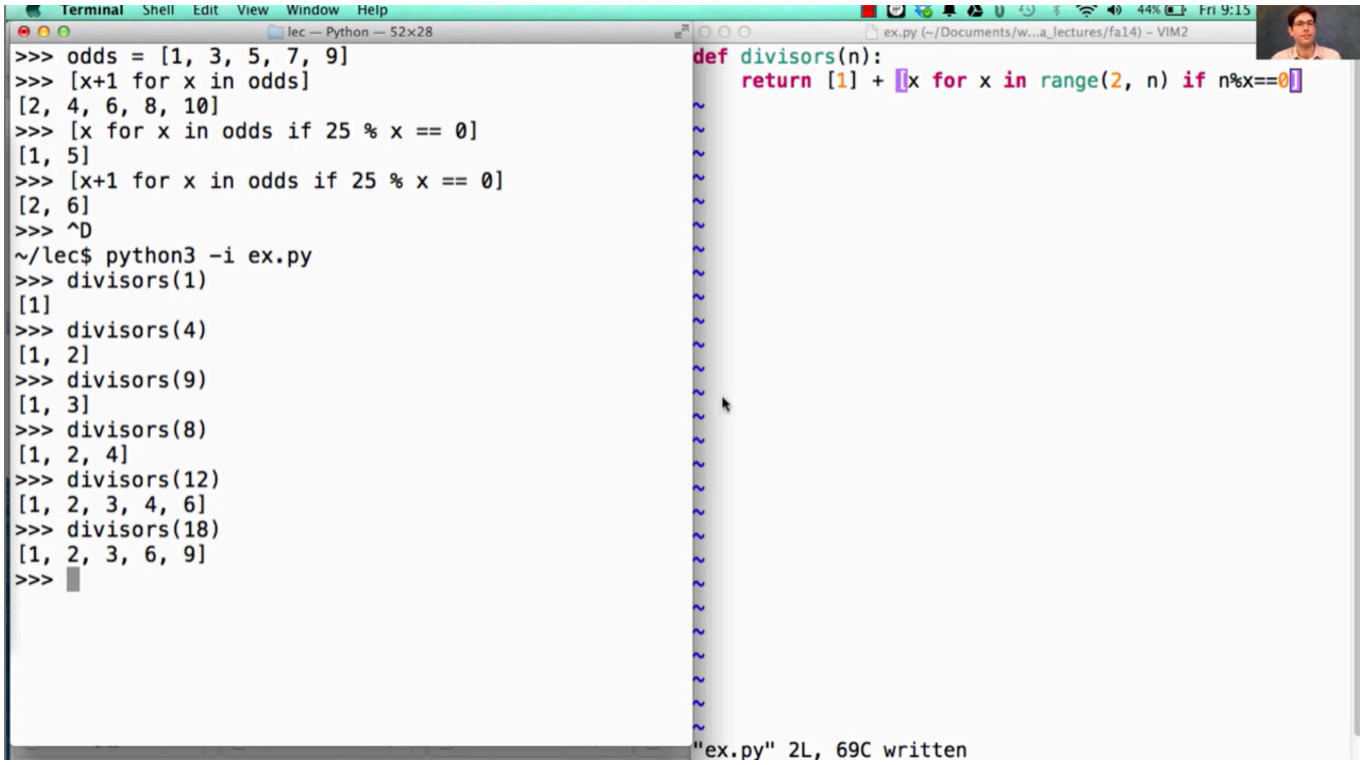
12

for iteration and recursion

```
Sum(recursively)

def mysum(L):
    if L == []:
        return 0
    else:
        return L[0] + mysum[1:]


def sum(n):
    total = 0
    for i in range(n + 1):
        totol += i
    return total

def sum(n):
    if n = 0: return 0
    return n + sum(n - 1)
```

List Comprehensions

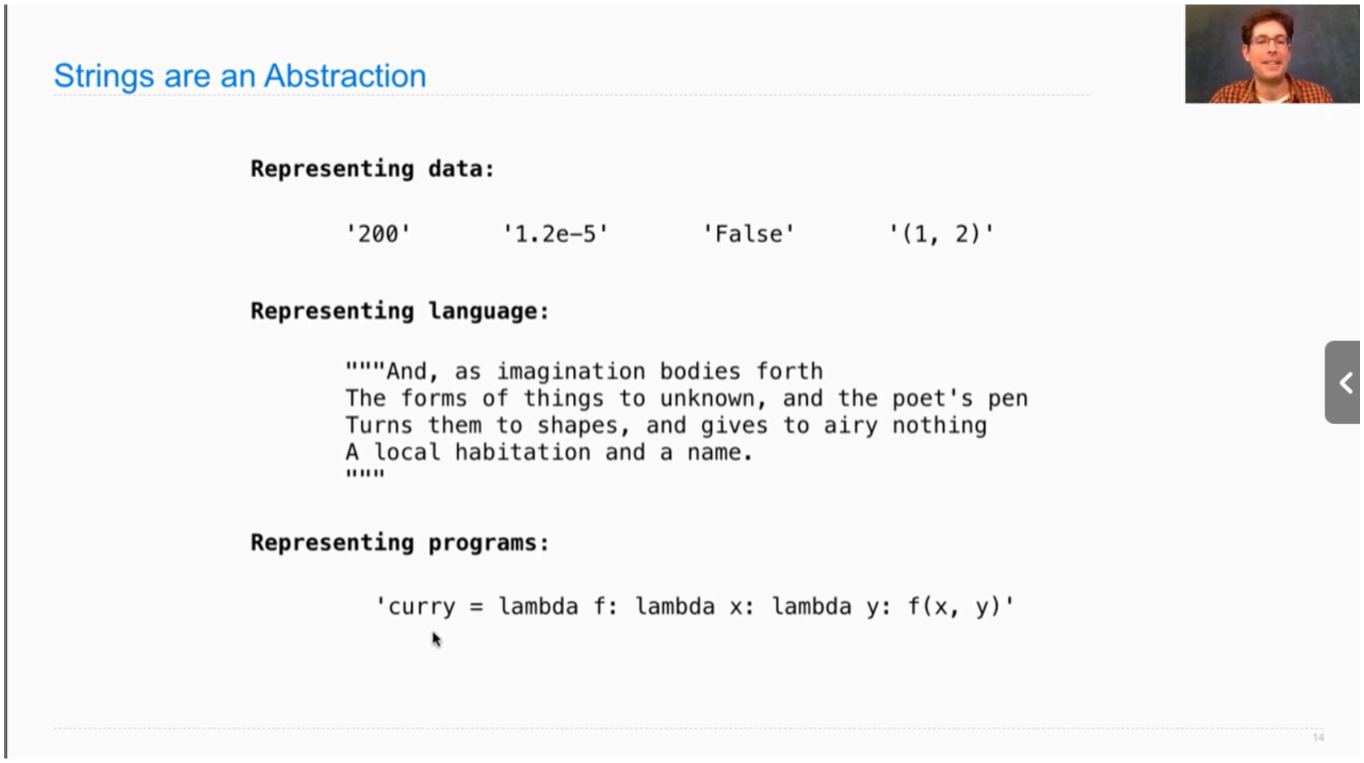
```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'm', 'n', 'o', 'p']
[letters[i] for i in [3, 4, 6, 8]]

['d','e','m','o']
```


Terminal Shell Edit View Window Help
lec — Python — 52×28
ex.py (~/Documents/w...a_lectures/fa14) - VIM2

```
>>> odds = [1, 3, 5, 7, 9]
>>> [x+1 for x in odds]
[2, 4, 6, 8, 10]
>>> [x for x in odds if 25 % x == 0]
[1, 5]
>>> [x+1 for x in odds if 25 % x == 0]
[2, 6]
>>> ^D
~/lec$ python3 -i ex.py
>>> divisors(1)
[1]
>>> divisors(4)
[1, 2]
>>> divisors(9)
[1, 3]
>>> divisors(8)
[1, 2, 4]
>>> divisors(12)
[1, 2, 3, 4, 6]
>>> divisors(18)
[1, 2, 3, 6, 9]
>>>
```

```
def divisors(n):
    return [1] + [x for x in range(2, n) if n%x==0]

"ex.py" 2L, 69C written
```

Strings


Strings are an Abstraction

Representing data:

```
'200'      '1.2e-5'      'False'      '(1, 2)'
```

Representing language:

```
"""And, as imagination bodies forth
The forms of things to unknown, and the poet's pen
Turns them to shapes, and gives to airy nothing
A local habitation and a name.
"""
```

Representing programs:

```
'curry = lambda f: lambda x: lambda y: f(x, y)'
```

Reversing a LIst(recursively)

```
def ReverseList(list):
    if len(list) == 1: return list
    else: return ReverseList(list[1:]) + [list[0]]
```

ATTENTION

in python, string and list are actually different things.
although they all use index to touch and operate, and almost the same in c++, but python doesn't see them as same things.