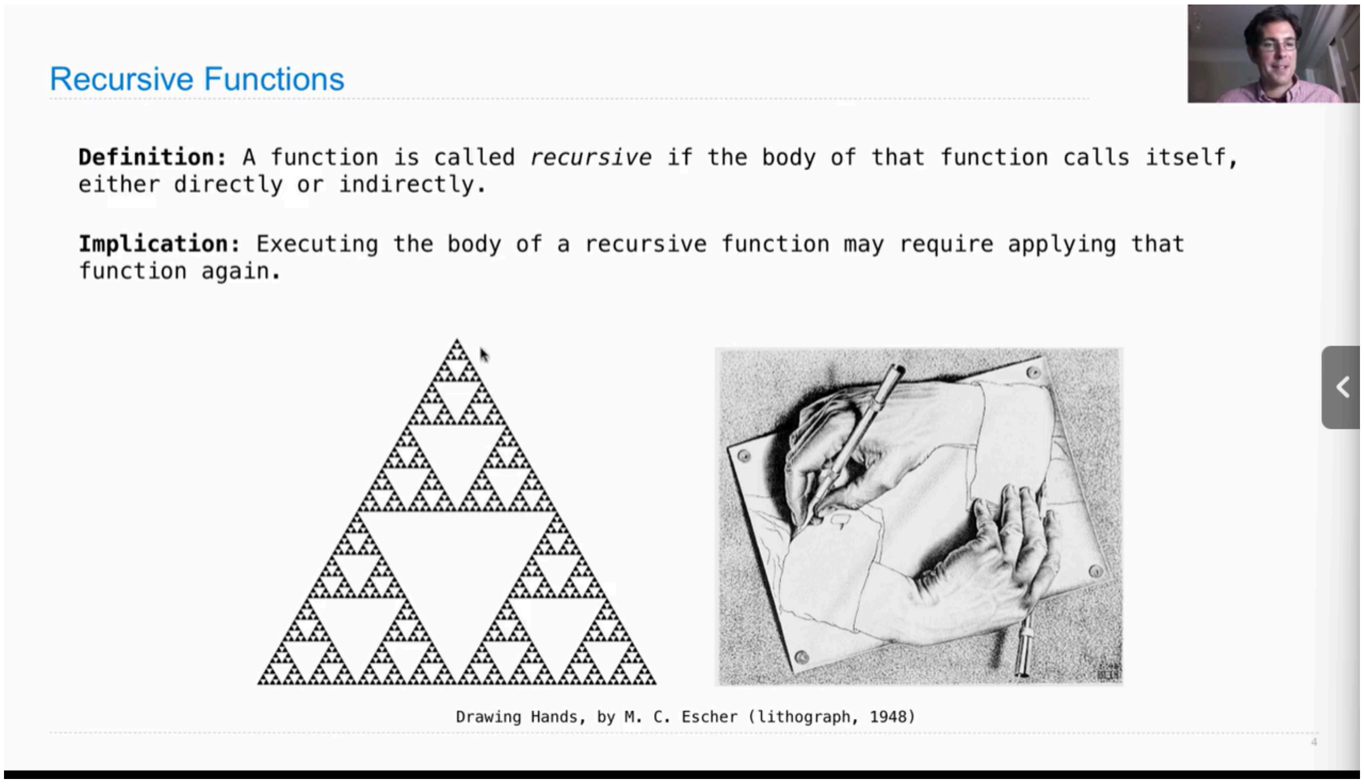
# Lecture8 recursion

## Recursive Functions



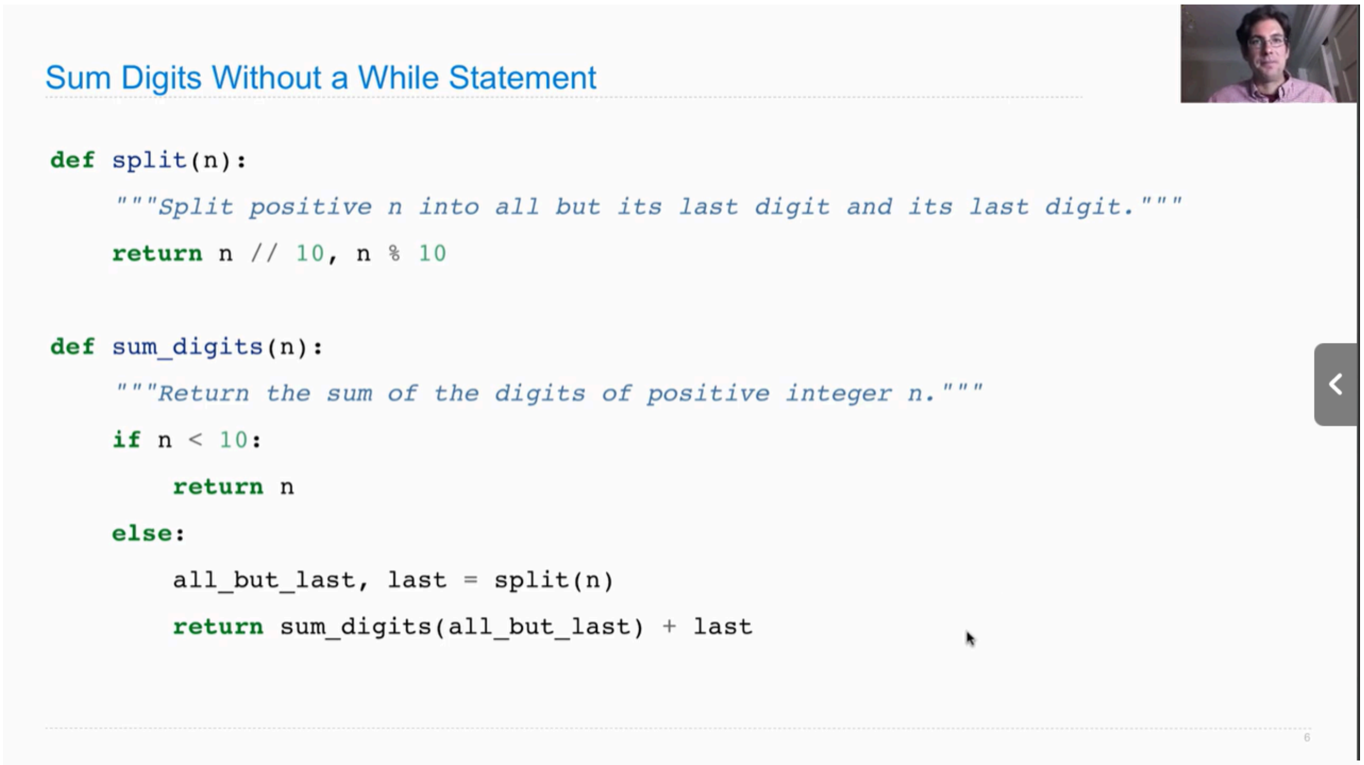
### Recursive Functions

**Definition:** A function is called *recursive* if the body of that function calls itself, either directly or indirectly.

**Implication:** Executing the body of a recursive function may require applying that function again.

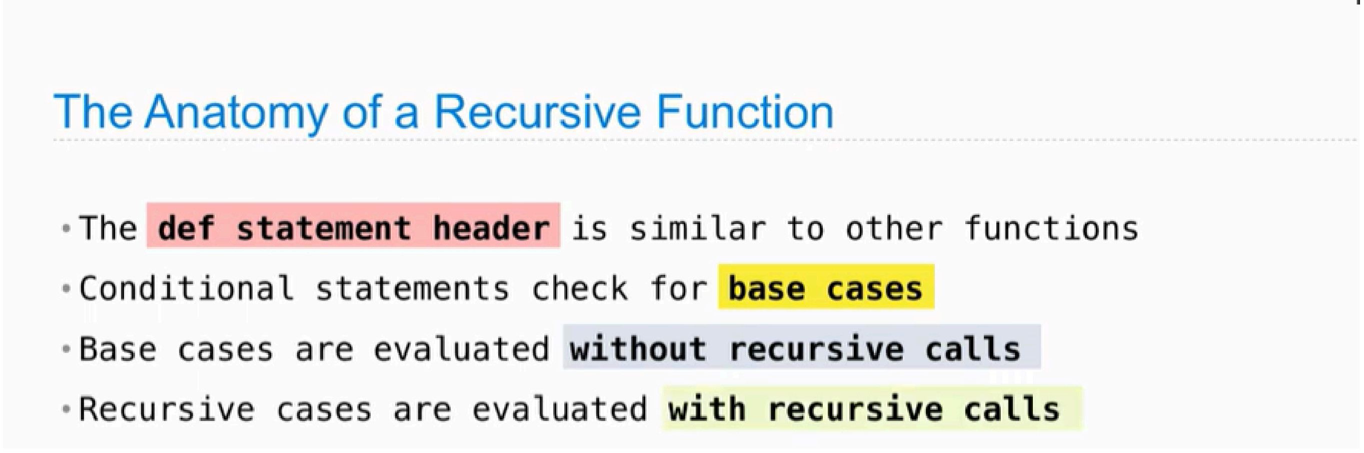
Drawing Hands, by M. C. Escher (lithograph, 1948)

```
def splict(n):
    if n < 10: return n
    else return split(n // 10)
```



## Sum Digits Without a While Statement

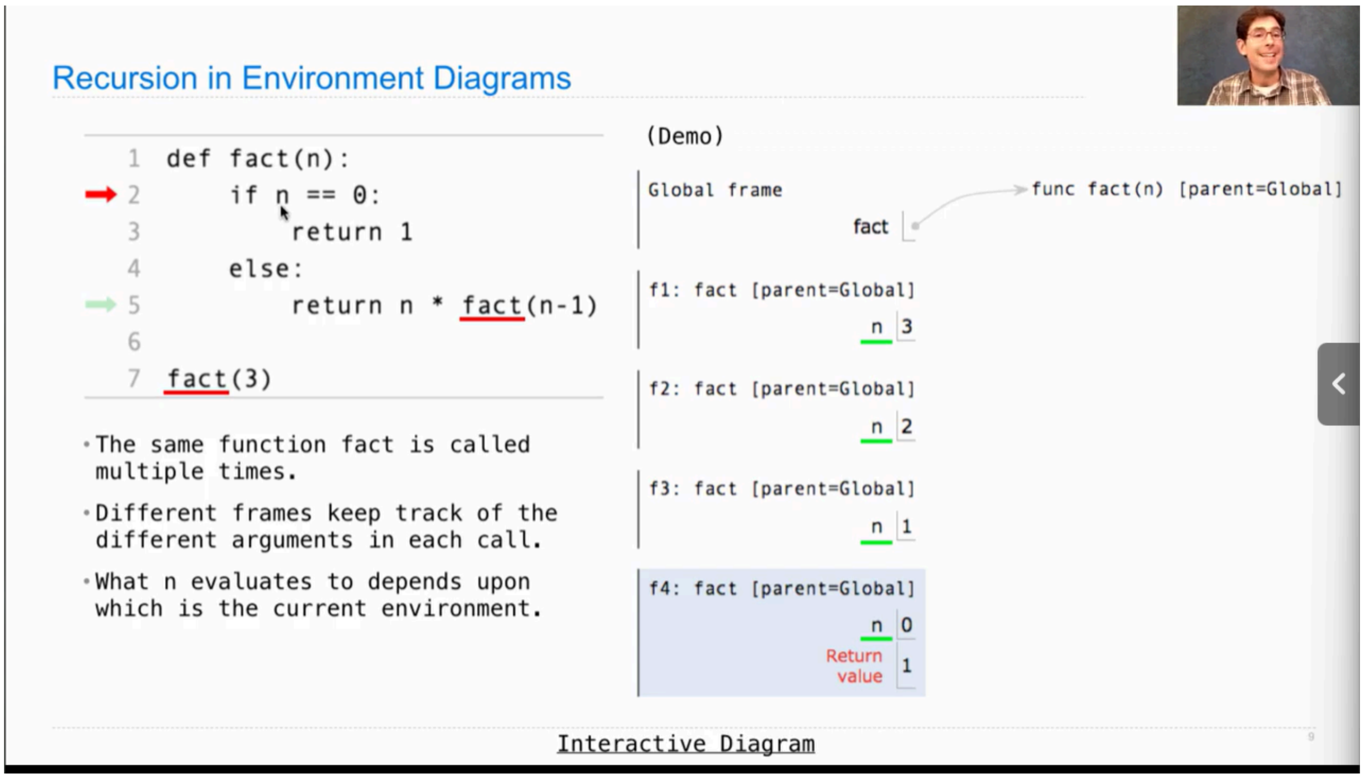
```python
def split(n):
    """Split positive n into all but its last digit and its last digit."""
    return n // 10, n % 10

def sum_digits(n):
    """Return the sum of the digits of positive integer n."""
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return sum_digits(all_but_last) + last
```



## The Anatomy of a Recursive Function

- The **def statement header** is similar to other functions
- Conditional statements check for **base cases**
- Base cases are evaluated **without recursive calls**
- Recursive cases are evaluated **with recursive calls**

## Recursion in Environment Diagrams



### Recursion in Environment Diagrams

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)

fact(3)
```

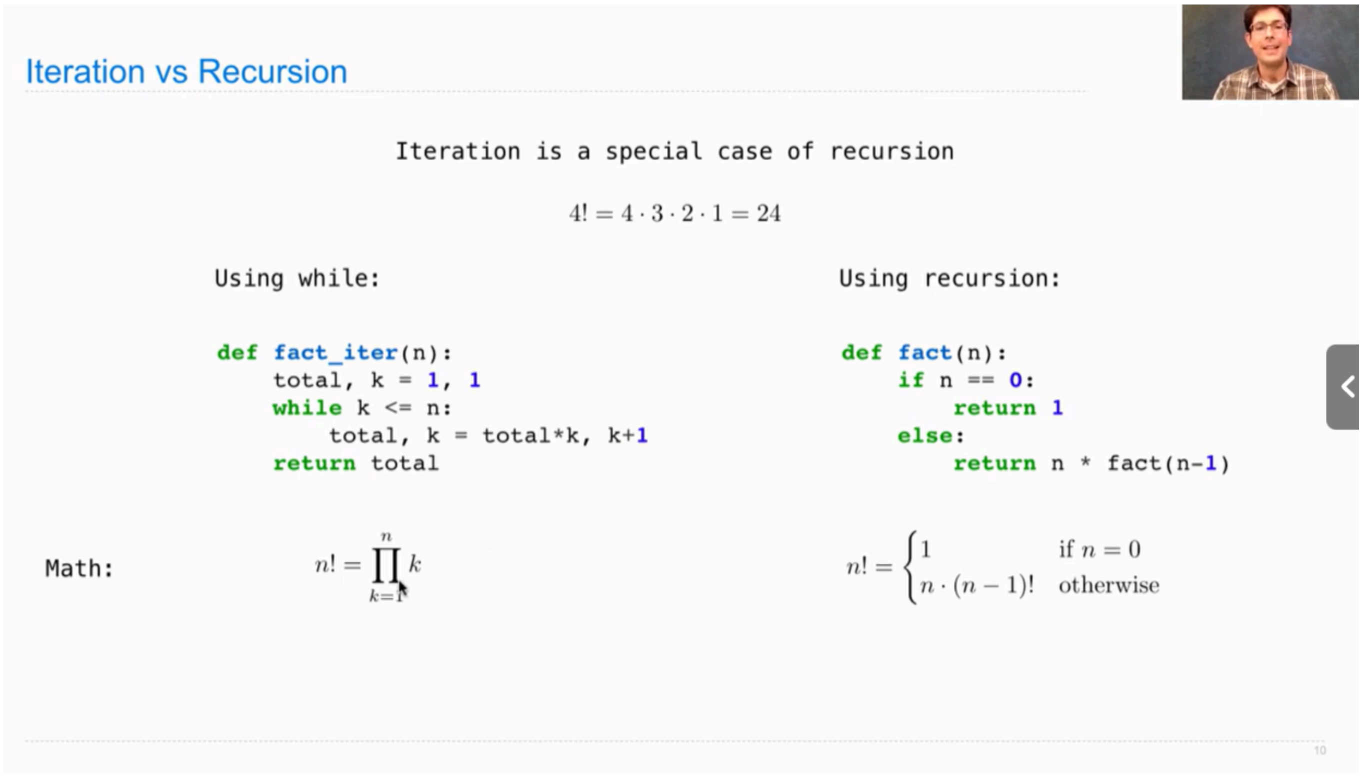
- The same function fact is called multiple times.
- Different frames keep track of the different arguments in each call.
- What n evaluates to depends upon which is the current environment.

(Demo)

```
Global frame
              fact  ──> func fact(n) [parent=Global]

f1: fact [parent=Global]
                 n  3

f2: fact [parent=Global]
                 n  2

f3: fact [parent=Global]
                 n  1

f4: fact [parent=Global]
                 n  0
      Return value  1
```

Interactive Diagram

## Iteration vs Recursion



### Iteration vs Recursion

Iteration is a special case of recursion

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$$

Using while:

```
def fact_iter(n):
    total, k = 1, 1
    while k <= n:
        total, k = total*k, k+1
    return total
```

Using recursion:

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)
```

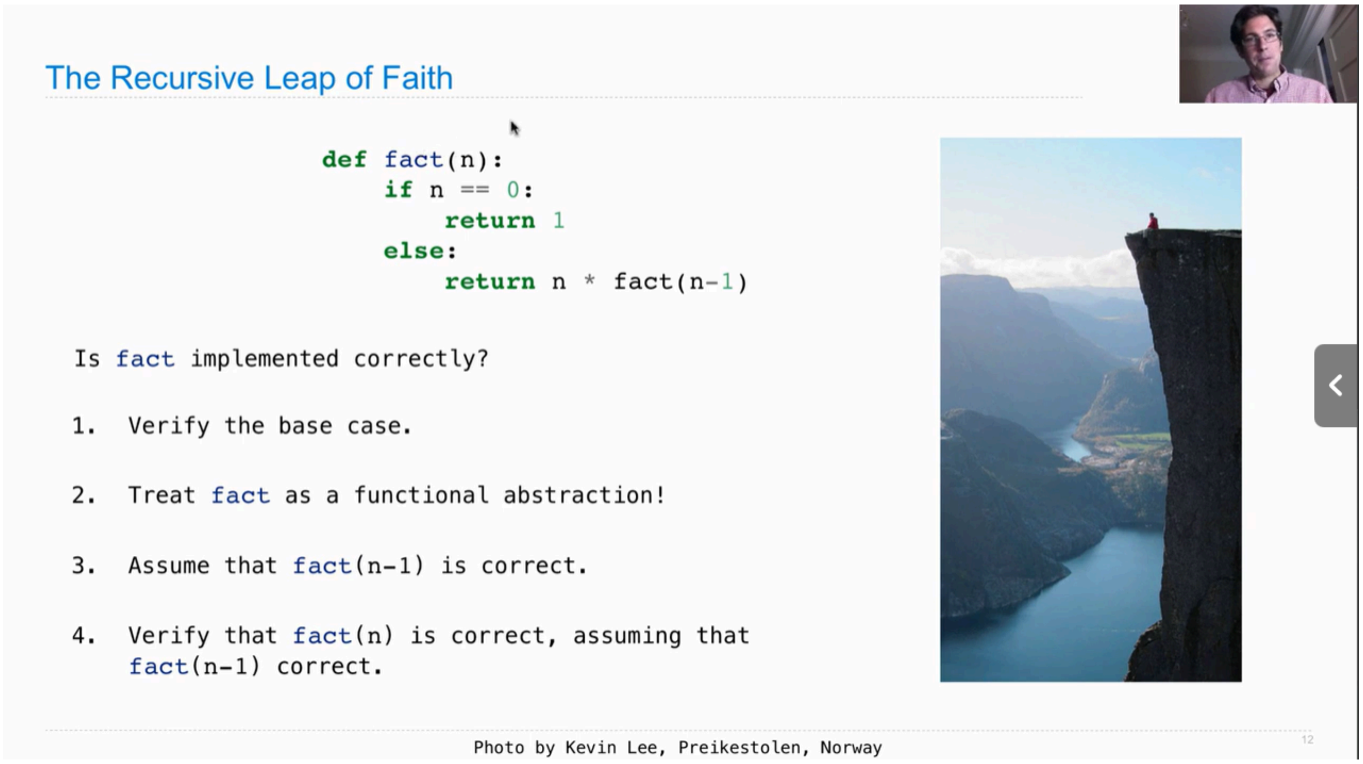
Math:

$$n! = \prod_{k=1}^{n} k$$

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

## The Recursive Leap of Faith

```
don't think about how it's implemented. just think about what it's supposed to do.
```



## The Recursive Leap of Faith

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)
```

Is fact implemented correctly?

1. Verify the base case.

2. Treat fact as a functional abstraction!

3. Assume that fact(n-1) is correct.

4. Verify that fact(n) is correct, assuming that fact(n-1) correct.

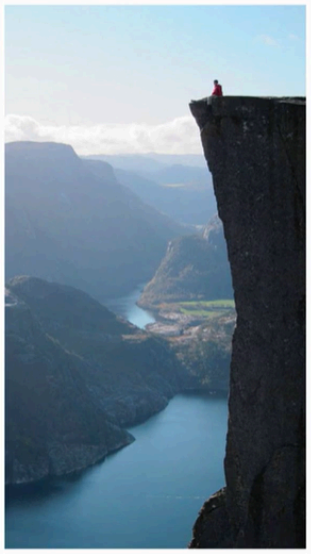
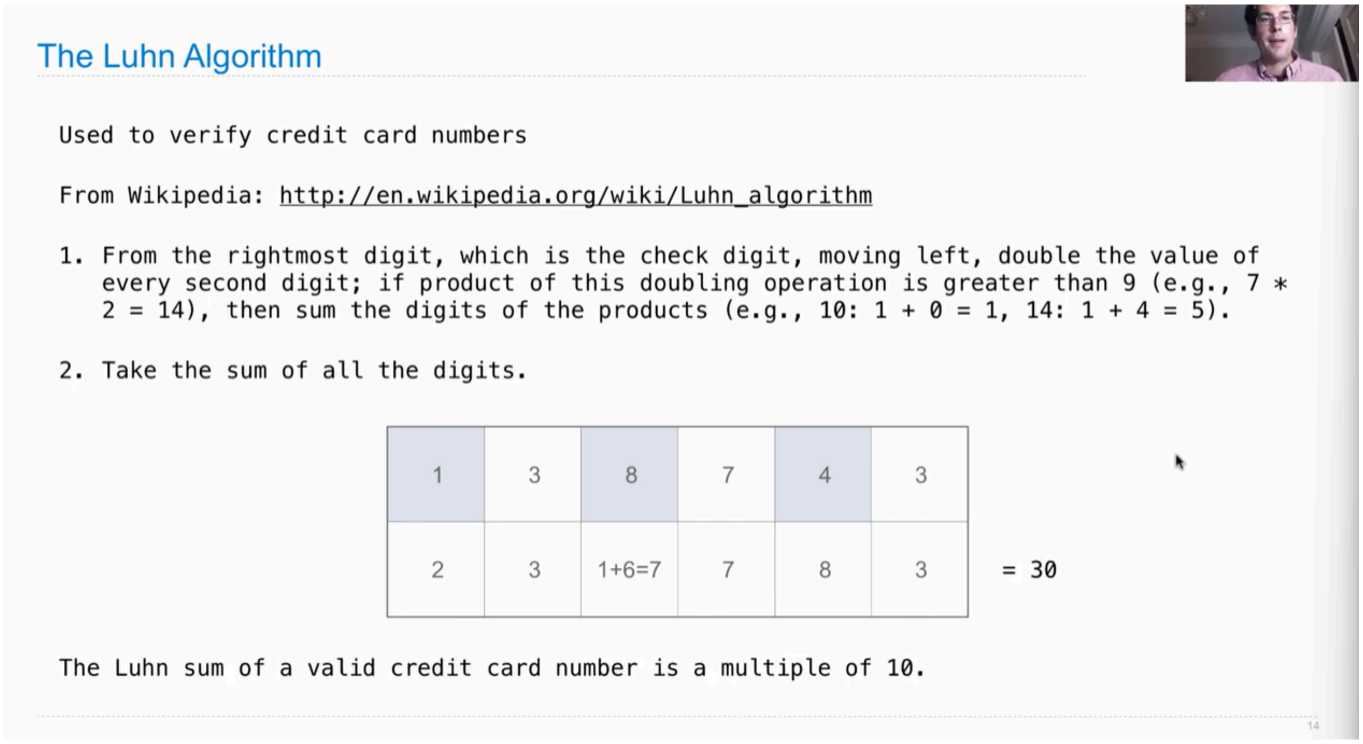


Photo by Kevin Lee, Preikestolen, Norway

## Mutual Recursion

```
happens when 2 functions call each other
```



## The Luhn Algorithm

Used to verify credit card numbers

From Wikipedia: http://en.wikipedia.org/wiki/Luhn_algorithm

1. From the rightmost digit, which is the check digit, moving left, double the value of every second digit; if product of this doubling operation is greater than 9 (e.g., 7 * 2 = 14), then sum the digits of the products (e.g., 10: 1 + 0 = 1, 14: 1 + 4 = 5).

2. Take the sum of all the digits.

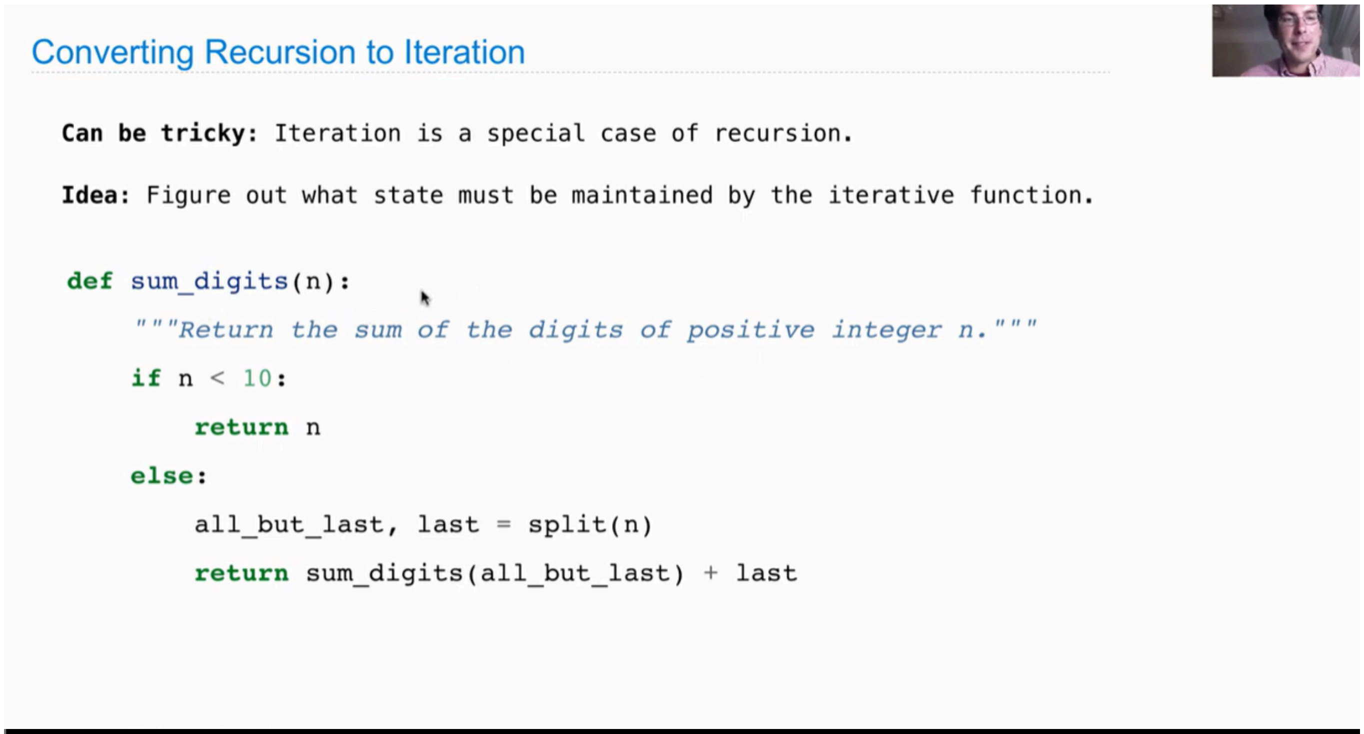
| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 3 | 8 | 7 | 4 | 3 | |
| 2 | 3 | 1+6=7 | 7 | 8 | 3 | = 30 |

The Luhn sum of a valid credit card number is a multiple of 10.

```
def split(n):
    return n // 10, n % 10

def sum_digits(n):
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return sum_digits(all_but_last) + last

def luhn_sum(n):
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return luhn_sum_double(all_but_last) + last

def luhn_sum_double(n):
    all_but_last, last = split(n)
    luhn_digit = sum_digits(2 * last)
    if n < 10:
        return luhn_digit
    else:
        return luhn_sum(all_but_last) + luhn_digit
```

## Recursion and Iteration

```
iteration is a specail case of recursion
```

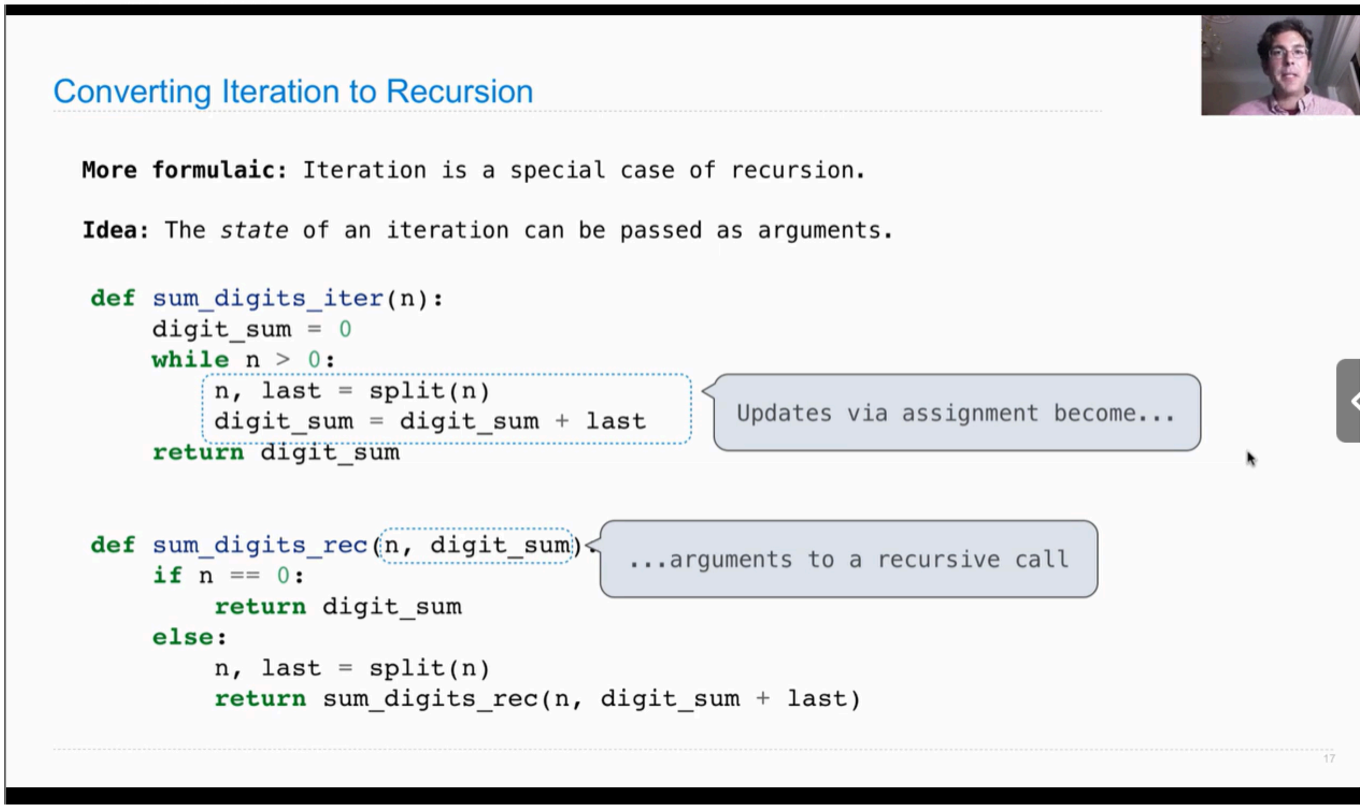


## Converting Recursion to Iteration

**Can be tricky:** Iteration is a special case of recursion.

**Idea:** Figure out what state must be maintained by the iterative function.

```
def sum_digits(n):
    """Return the sum of the digits of positive integer n."""
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return sum_digits(all_but_last) + last
```



## Converting Iteration to Recursion

**More formulaic:** Iteration is a special case of recursion.

**Idea:** The *state* of an iteration can be passed as arguments.

```
def sum_digits_iter(n):
    digit_sum = 0
    while n > 0:
        n, last = split(n)
        digit_sum = digit_sum + last
    return digit_sum
```

Updates via assignment become...

```
def sum_digits_rec(n, digit_sum):
    if n == 0:
        return digit_sum
    else:
        n, last = split(n)
        return sum_digits_rec(n, digit_sum + last)
```

...arguments to a recursive call
17

> why we need recursion?
> because it's more
>
> - abstract, which means the main body of the function could be simpler
> - straight forward, which means it's easier to be understood by other people.