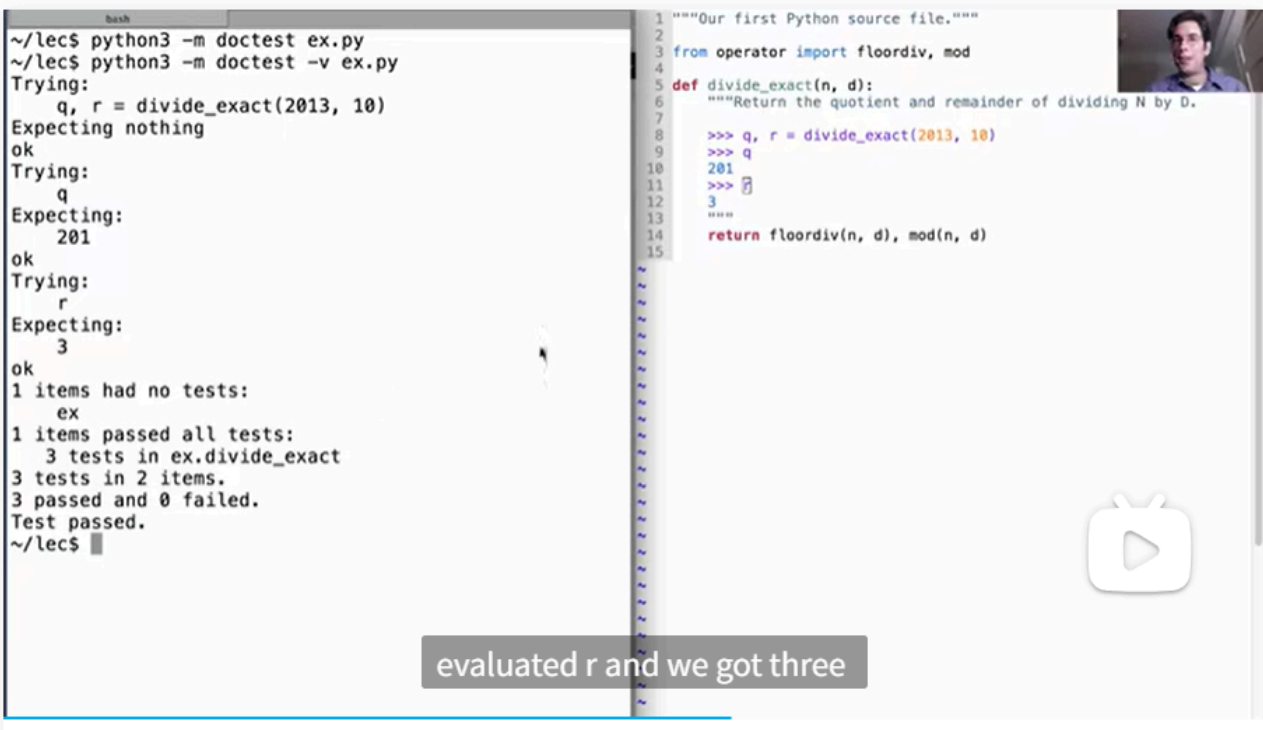


# Lecture 3.control

## RETURN & EFFECTS

### miscellaneous python features

- - - - `/(true div)`
  - `/(floor dive)`
  - `%(mod )`
  - return multiple values from function
    - `return n // d, n % d`
  - docstring
    - documentation about what a function does
  - besides describing what it does , we could also add some doctest



The screenshot shows a terminal window on the left and a code editor on the right. The terminal displays the execution of a Python script with doctests. The code editor shows the source code of the `divide_exact` function, which includes a docstring and a doctest. A small video inset in the top right corner shows a person speaking. A play button icon is visible in the bottom right corner of the code editor. A text box at the bottom of the terminal says "evaluated r and we got three".

```
bash
~/lec$ python3 -m doctest ex.py
~/lec$ python3 -m doctest -v ex.py
Trying:
  q, r = divide_exact(2013, 10)
Expecting nothing
ok
Trying:
  q
Expecting:
  201
ok
Trying:
  r
Expecting:
  3
ok
1 items had no tests:
  ex
1 items passed all tests:
  3 tests in ex.divide_exact
3 tests in 2 items.
3 passed and 0 failed.
Test passed.
~/lec$
```

```
1 """Our first Python source file."""
2
3 from operator import floordiv, mod
4
5 def divide_exact(n, d):
6     """Return the quotient and remainder of dividing N by D.
7
8     >>> q, r = divide_exact(2013, 10)
9     >>> q
10    201
11    >>> r
12    3
13    """
14     return floordiv(n, d), mod(n, d)
15
```

- default values of parameters
  - already learnt, so no more explanation.

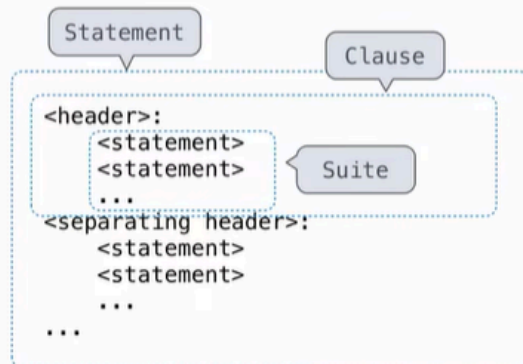
## Conditional Statements

## Statements



A **statement** is executed by the interpreter to perform an action

**Compound statements:**



The first header determines a statement's type

The header of a clause "controls" the suite that follows

def statements are compound statements

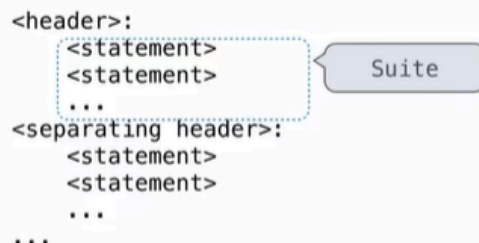
deaf statements as we saw

11

## Compound Statements



**Compound statements:**



A suite is a sequence of statements

To "execute" a suite means to execute its sequence of statements, in order

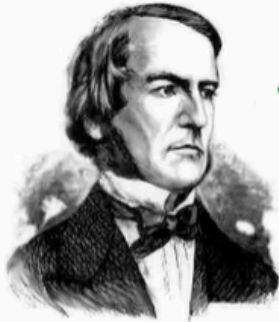
**Execution Rule for a sequence of statements:**

- Execute the first statement
- Unless directed otherwise, execute the rest

it's execute the first statement

12

## Boolean Contexts



George Boole

```
def absolute_value(x):  
    """Return the absolute value of x."""  
    if x < 0:  
        return -x  
    elif x == 0:  
        return 0  
    else:  
        return x
```

Two boolean contexts

False values in Python: False, 0, '', None (more to come)

True values in Python: Anything else (True)

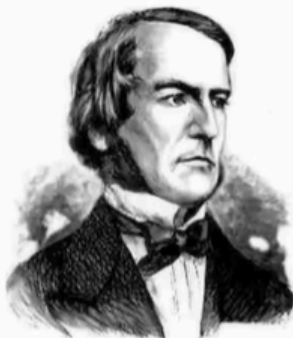
so george



15

## iteration

### While Statements



George Boole

(Demo)

```
1 i, total = 0, 0  
2 while i < 3:  
3     i = i + 1  
4     total = total + i
```

Global frame

i	<del>0</del> <del>1</del> <del>2</del>
total	<del>0</del> <del>1</del> <del>2</del> 3

#### Execution Rule for While Statements:

1. Evaluate the header's expression.
2. If it is a true value, execute the (whole) suite, then return to step 1.

and here's the really important part



example: find the prime factorization

```
"""Print the prime factors of n in non-decreasing order.

>>> prime_factors(8)
2
2
2
>>> prime_factors(9)
3
3
>>> prime_factors(10)
2
5
>>> prime_factors(11)
11
>>> prime_factors(12)
2
2
3
>>> prime_factors(858)
2
3
11
13
.....
while n > 1:
    k = smallest_prime_factor(n)
    n = n // k
    print(k)

def smallest_prime_factor(n):
    """Return the smallest k > 1 that evenly divides n."""
    k = 2
    while n % k != 0:
        k = k + 1
    return k
```



what we learn from this code?

at first, think of the problems in the simplest way , ignore all the complex structures, use the simplest sentences to organize your main document.

then, write the structures of the functions you used in your main document, but in the same way, so in this way, you don't need to organize your program at the very begining. in stead, you solve it in the middle.

more things, try to make your names easy to understand, like later prof changed k into smallest\_prime, this is not necessary, but it makes your program a better one, at least easier to be interpreted by human.

this is a way of abstraction when we are thinking about things, however, after we finished our program, we might realize that we don't need the functions, we might think we don't need it, cause obviously we can just fit it into the main document. however, when we fit it together, it does seems shorter, but more unlikely to be understand straightly.

but often using functions and their names makes everything clearer than trying to fit everything into a long body of a program.