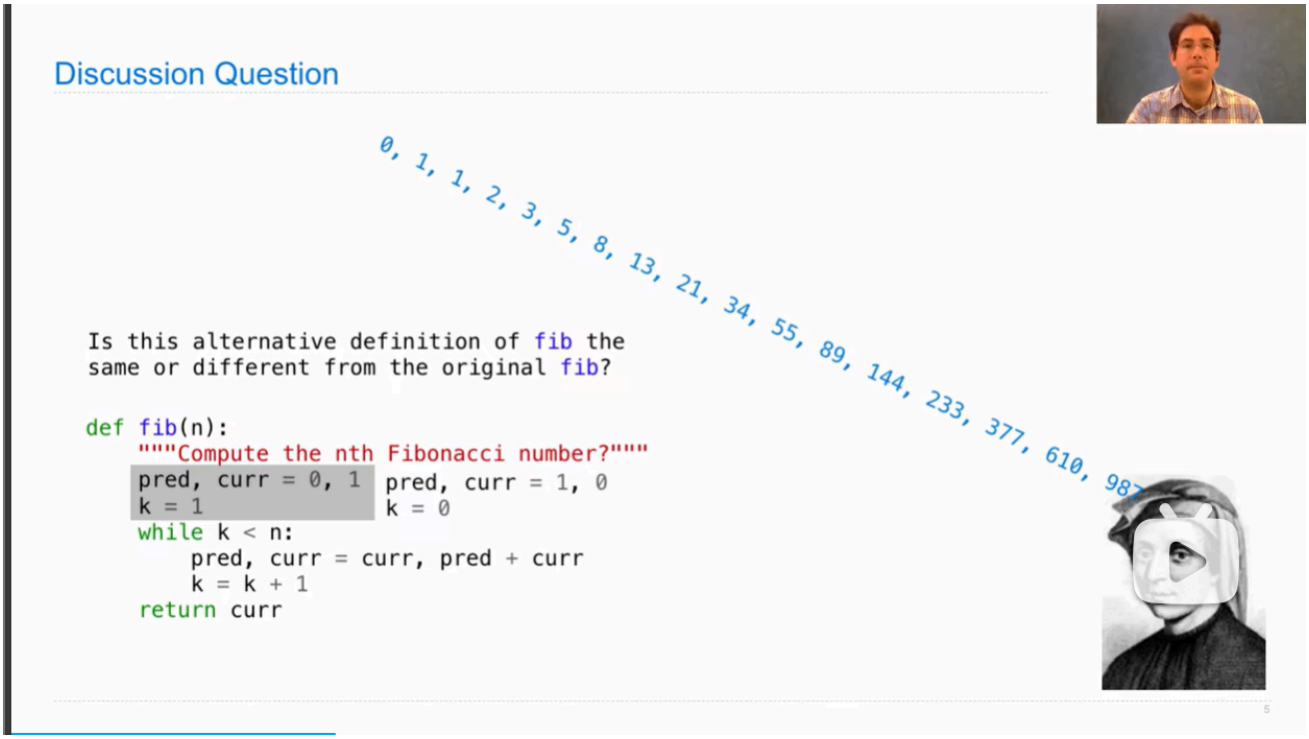
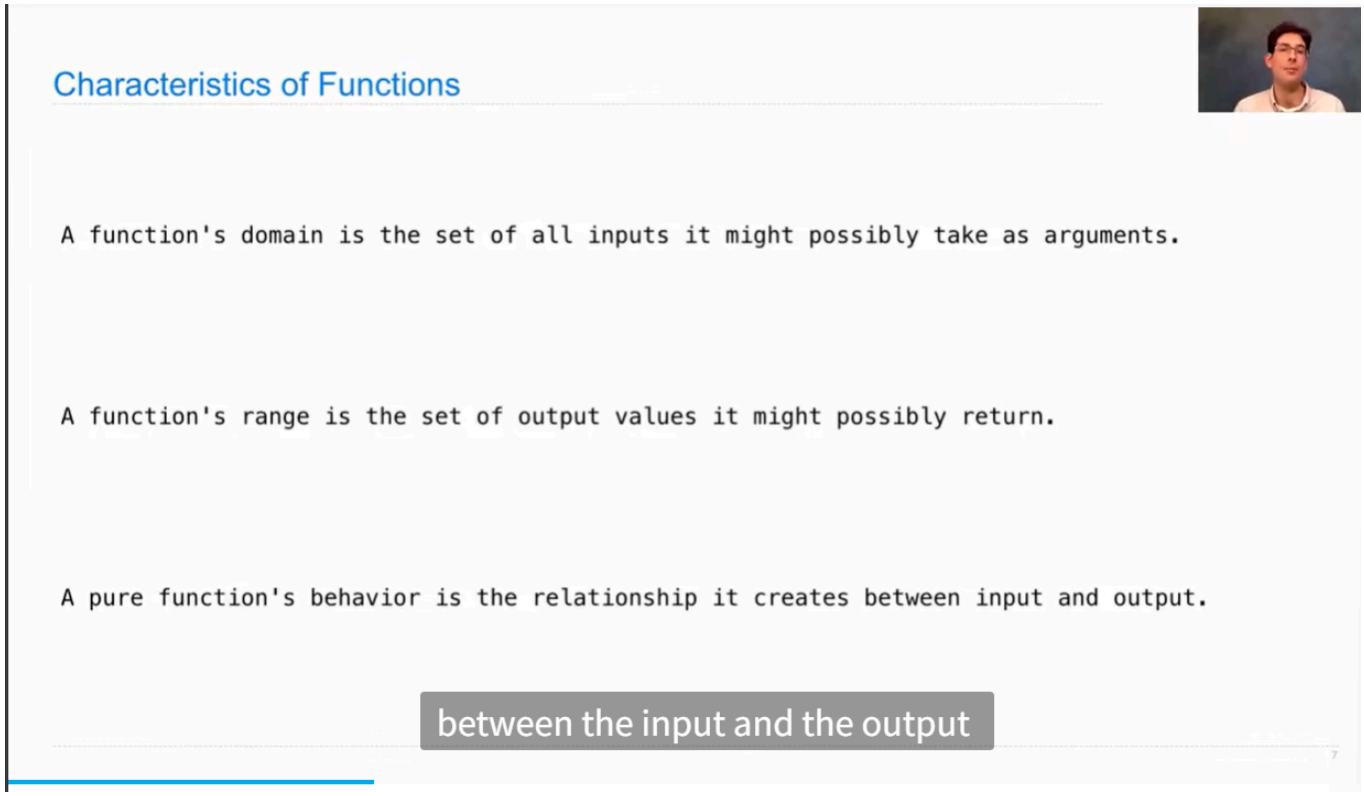
homework questions are not straight forward, project questions are not straight forward, even if you understand what's going on, you could spent a long time..... it's much better to go to office hour to get some help

# Iteration Example

- the fibonacci sequence



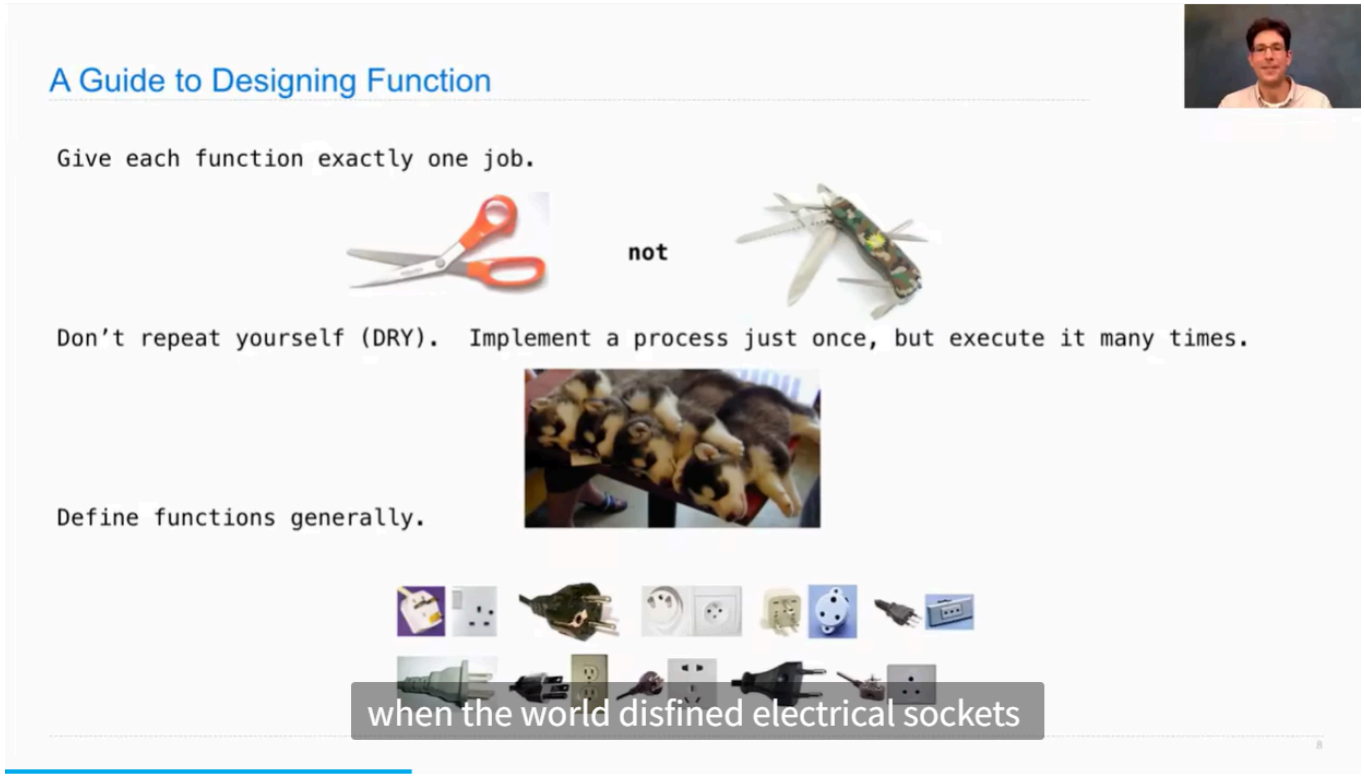

# Designing Functions



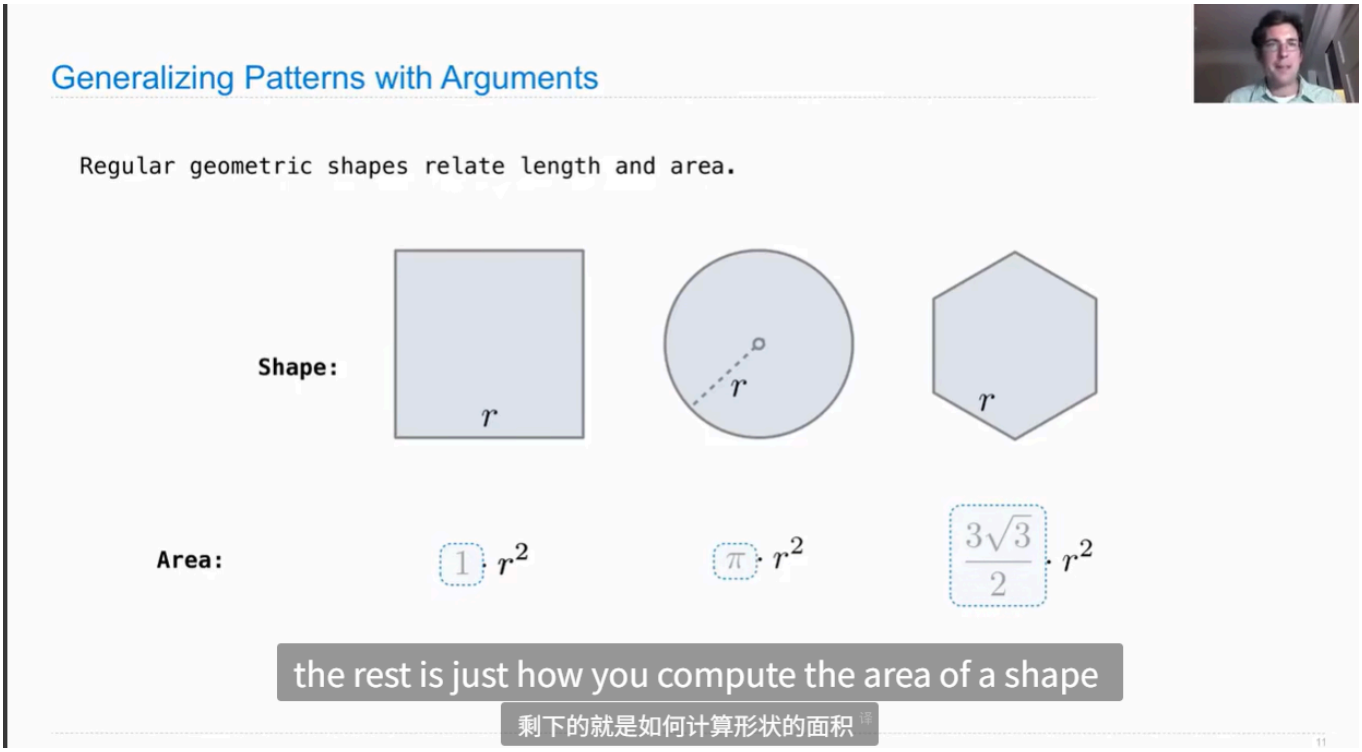

why do we discuss the domain and the range of the functions here?
i dont see the purpose.

so it is just trytin to explain what functions are for.

# A Guide to Designing Function



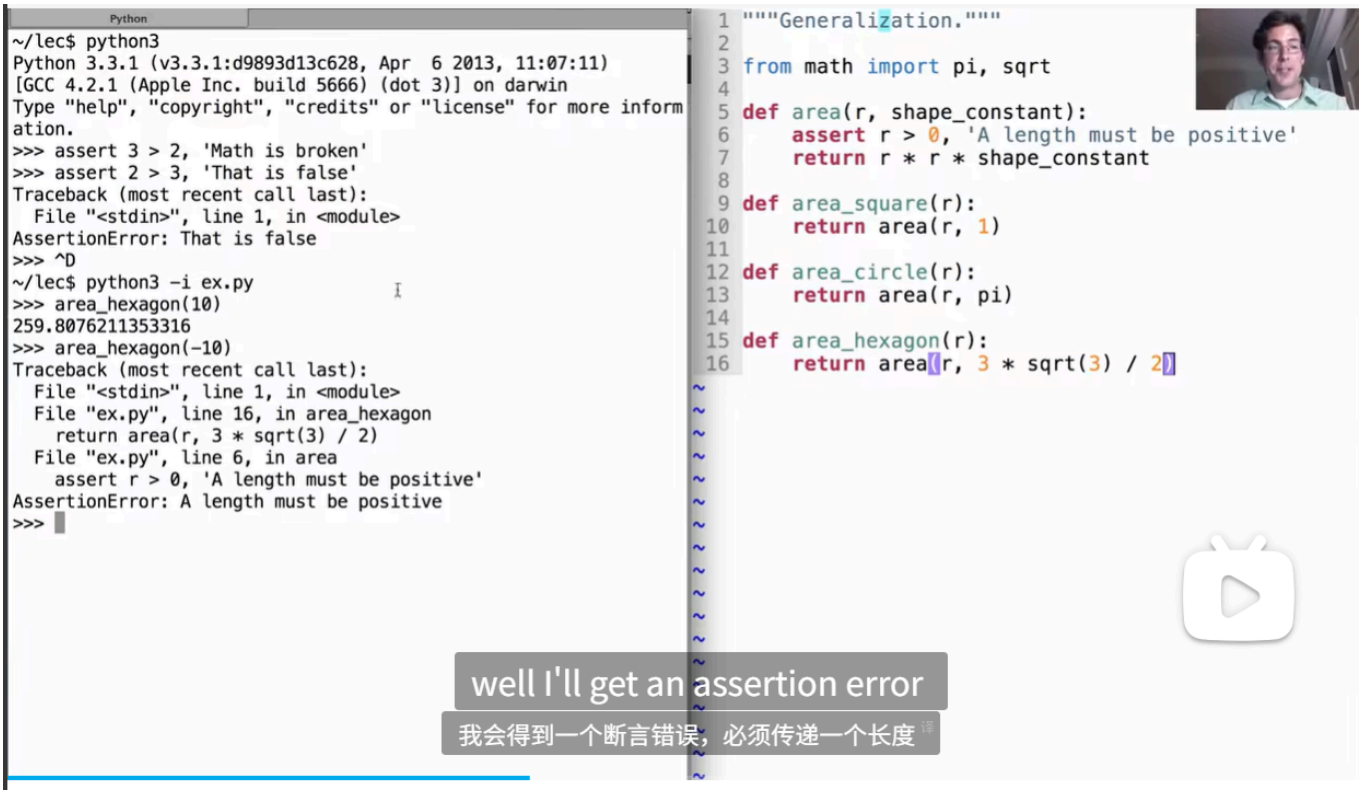
# Higher-Order Functions



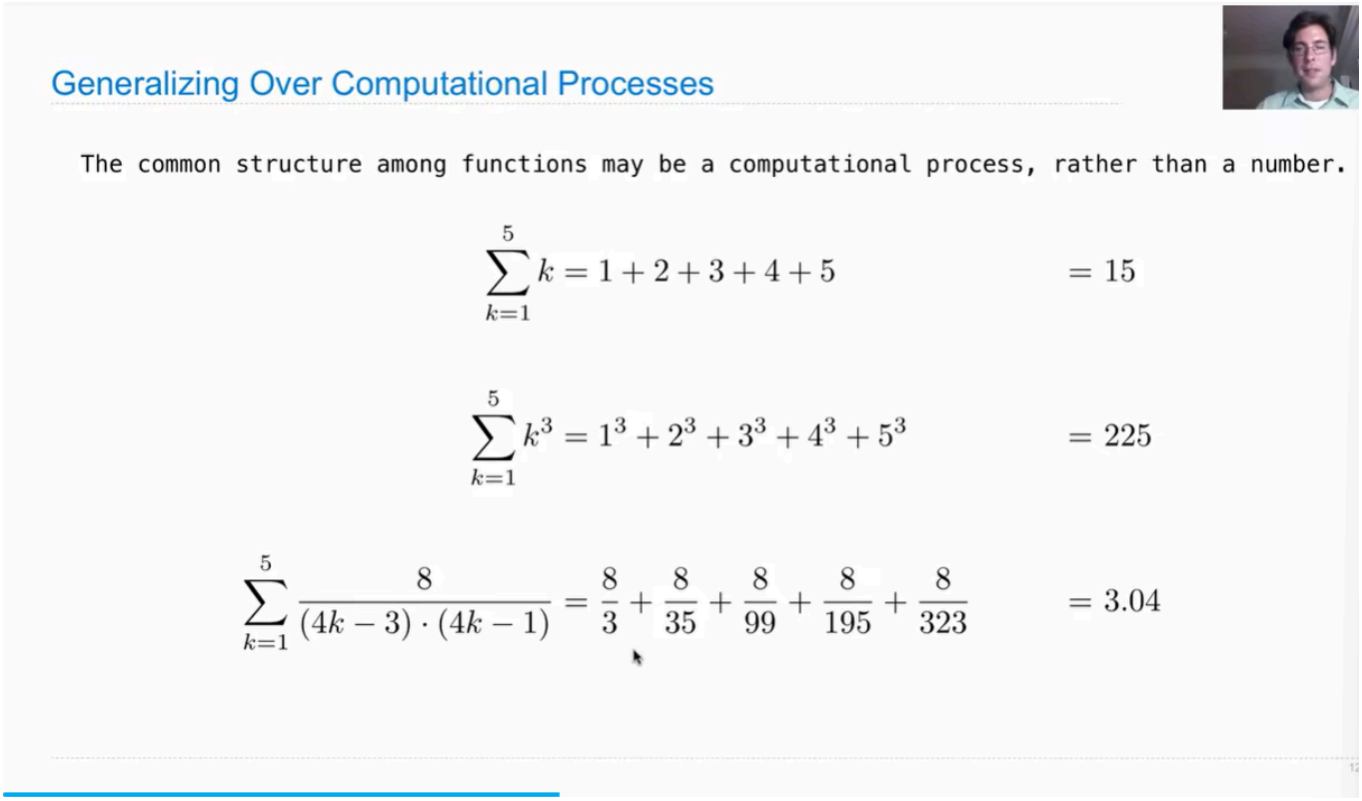

> try to find the similar structure and share the same implementation?

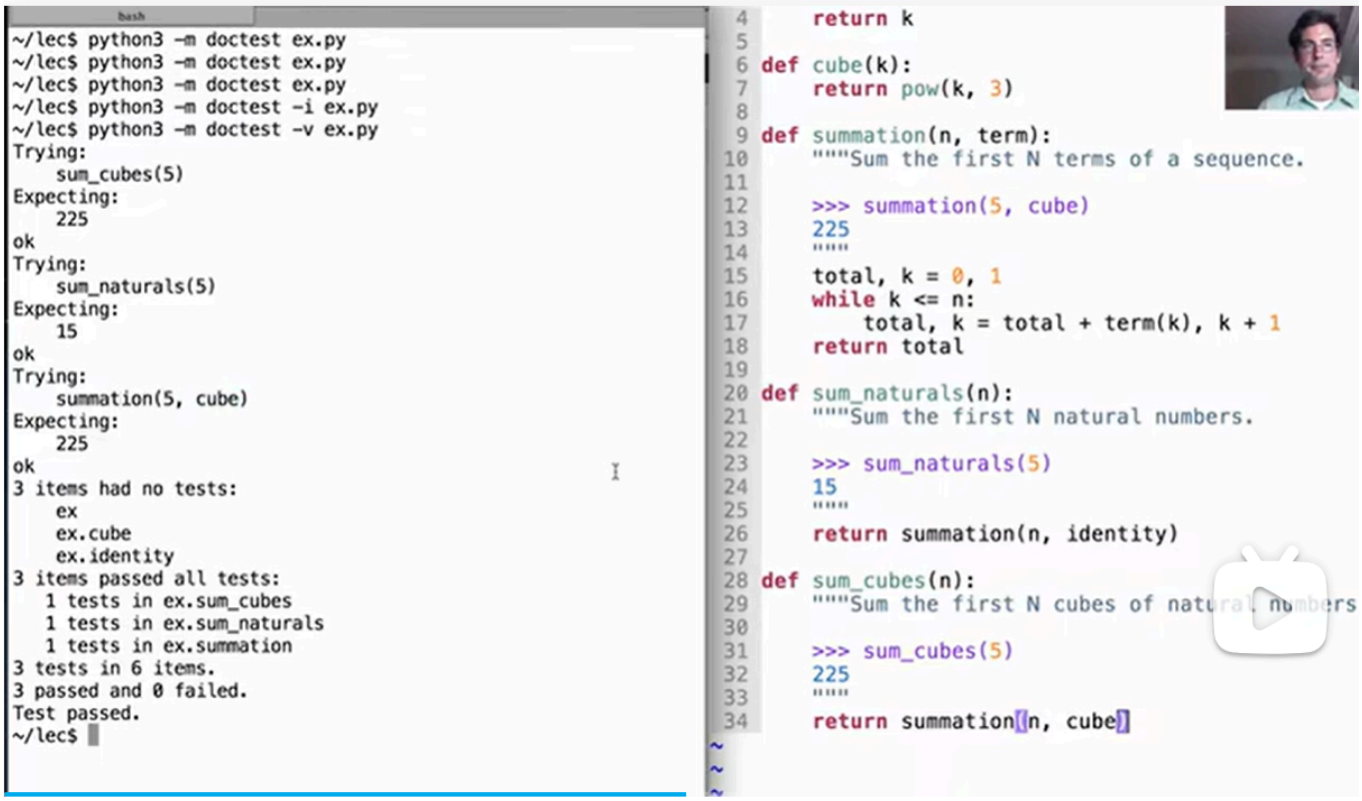
- assert

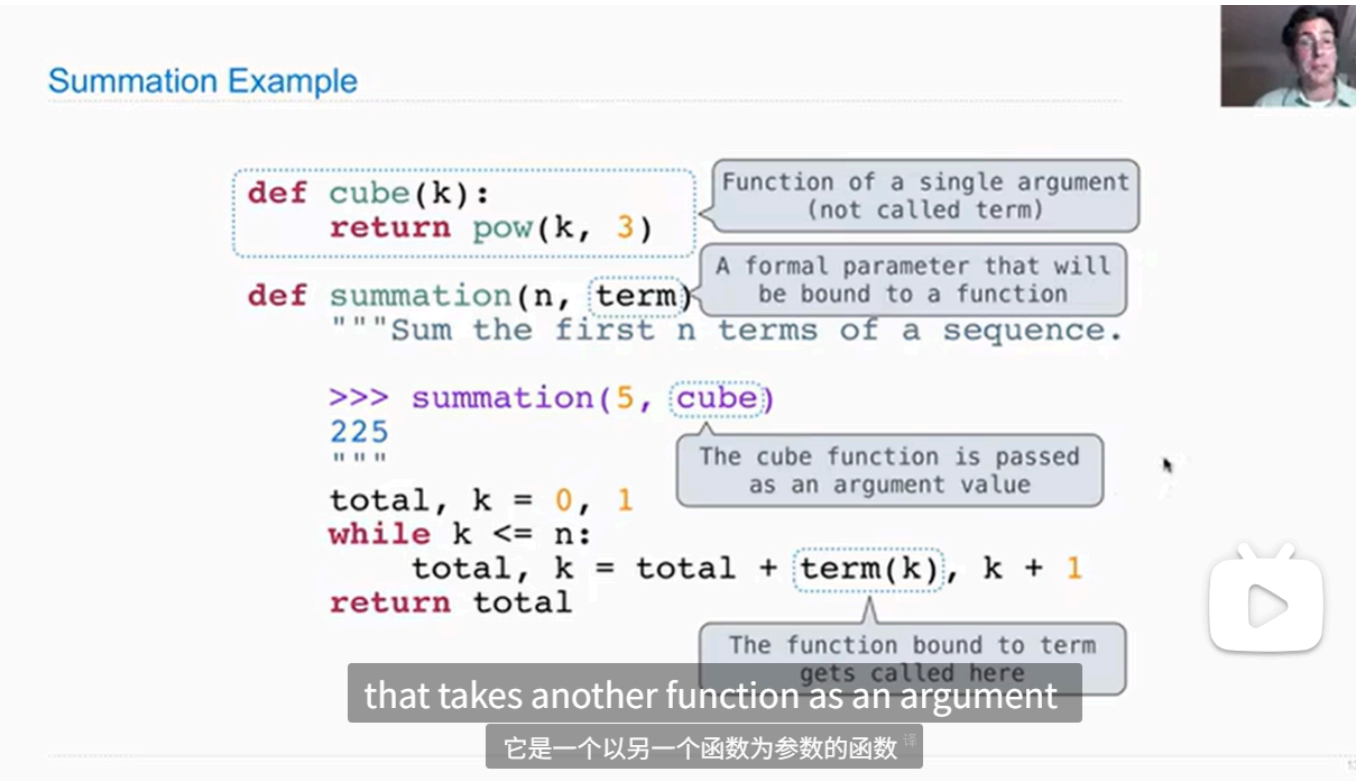
```
assert (condition), 'error message'
```



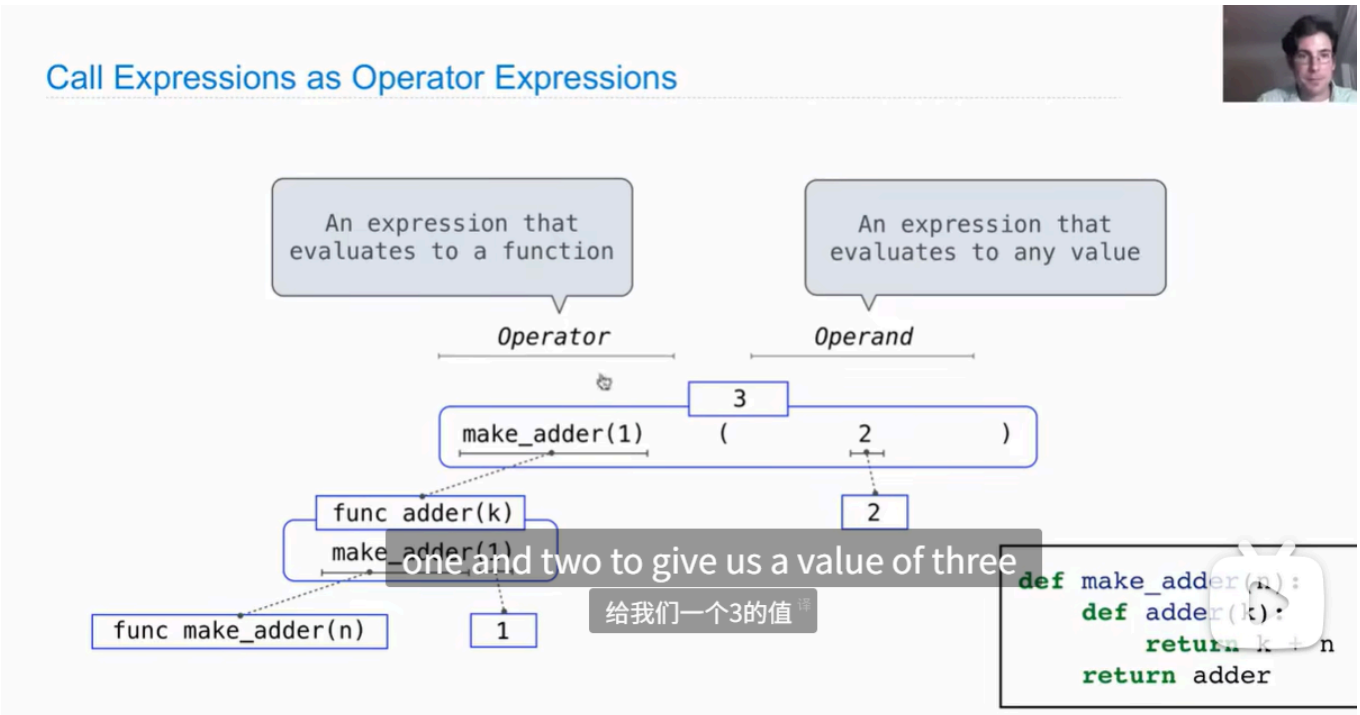
# Generalization of the process

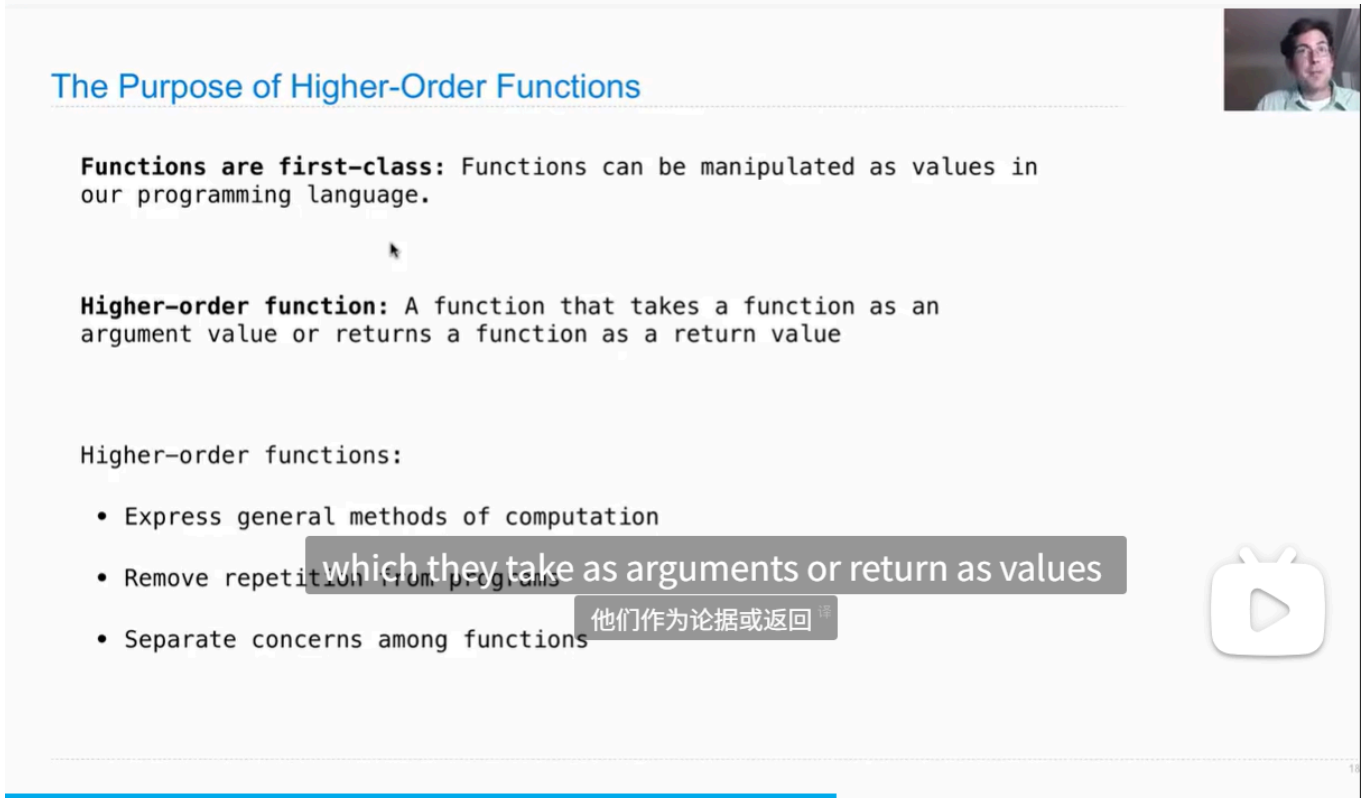




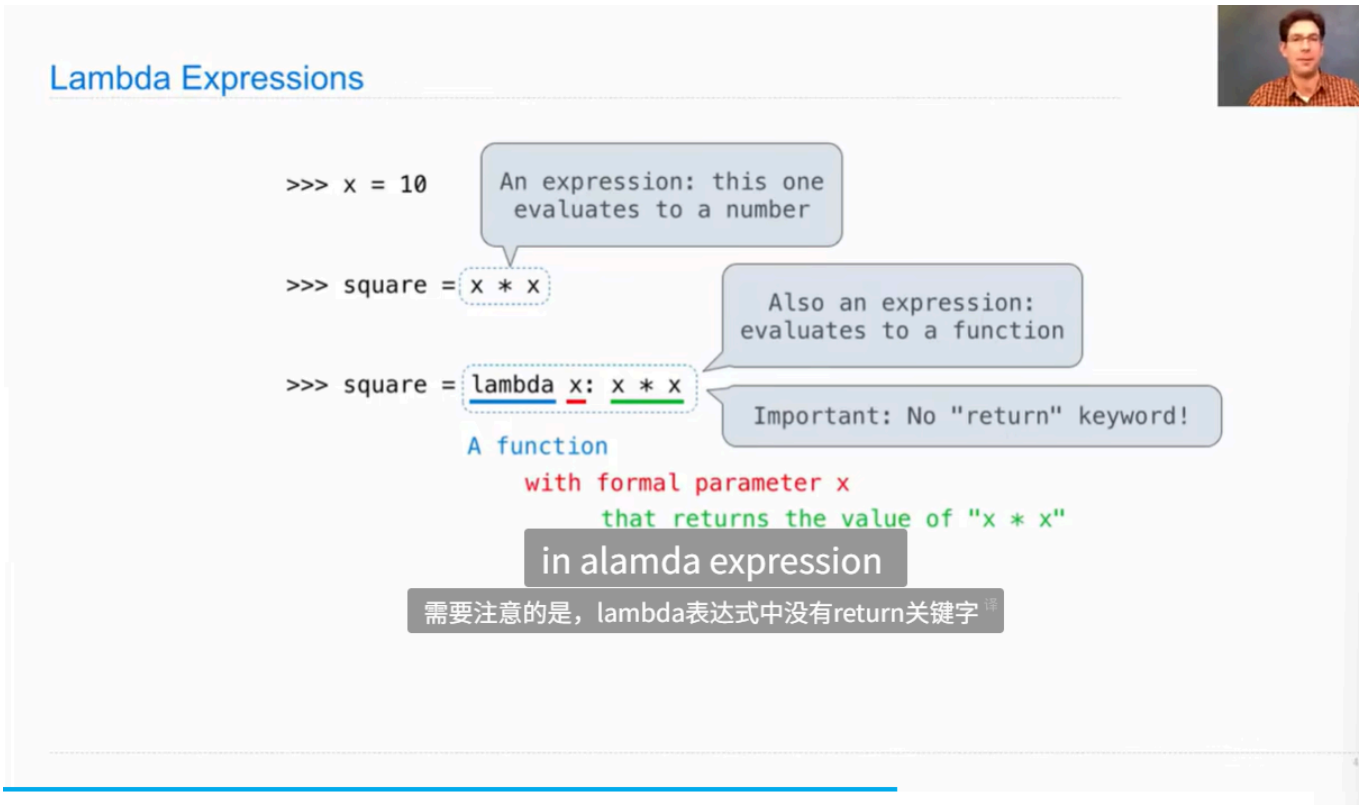

nesting functions???




# Lambda Expressions


the question is, why would we use the lambda expression if we could just define functions?




lambda functions create simple functions

```
lambda expressions are not common in Python, but important in general,
```

```
escpecially in some programming languages
```

> so, in which way?

## differences between def and lambda

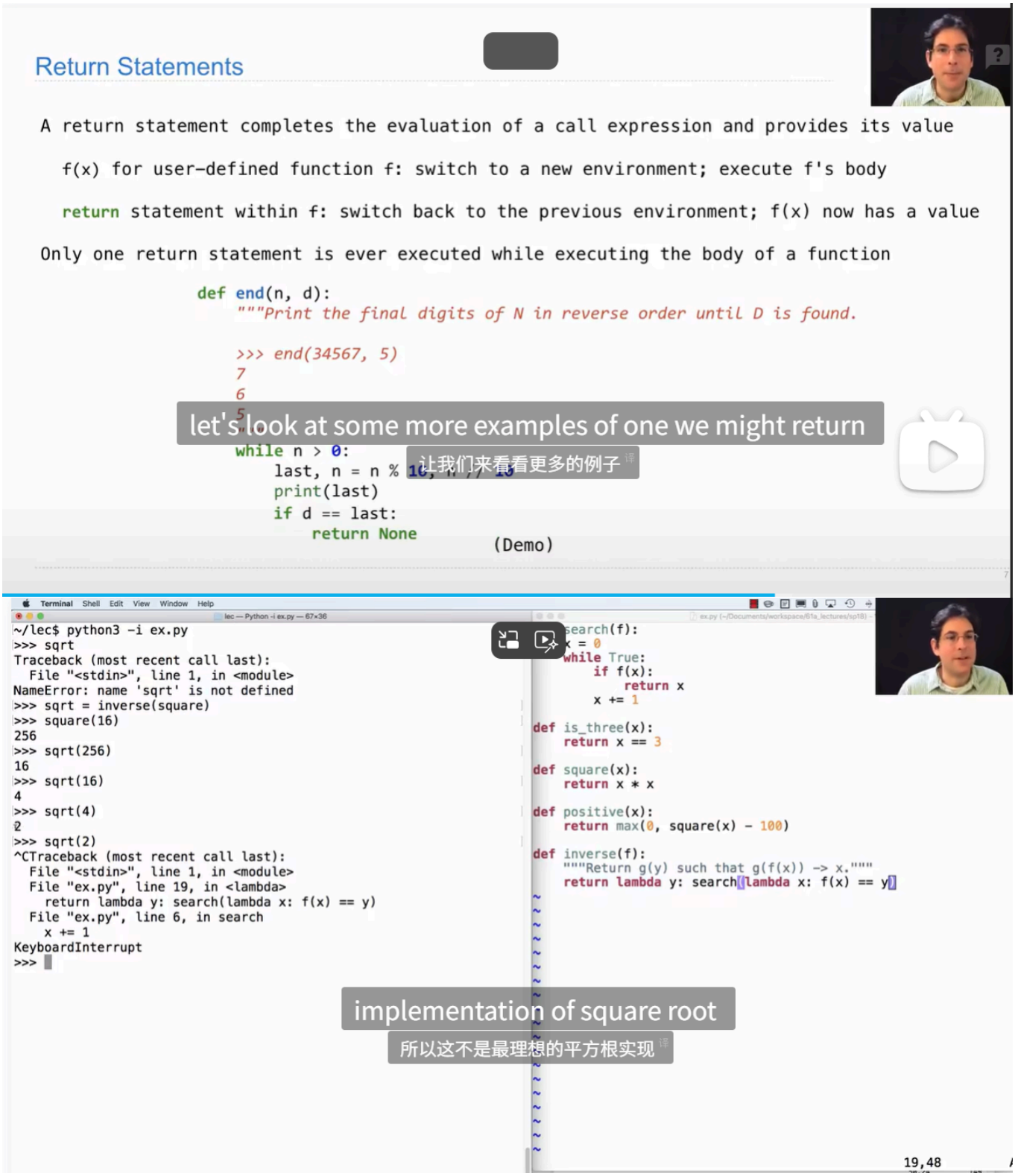
- only the def statement gives the function an intrinsic name
  this affects how we write the frame

  > however this is only a small difference

## Return statement

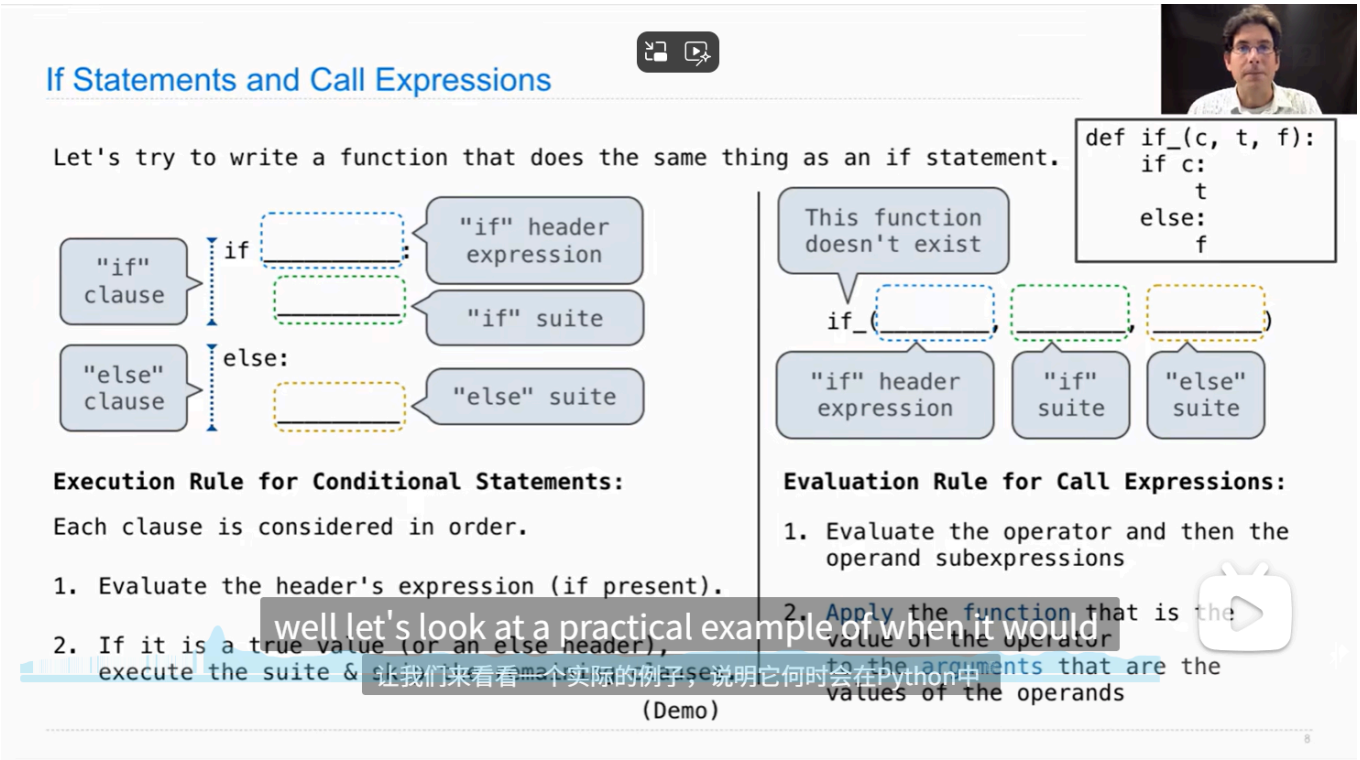
```
a return statement completes the evaluation of a call expression and provides
its value

return statement within f: switch back to the prevous environment
```

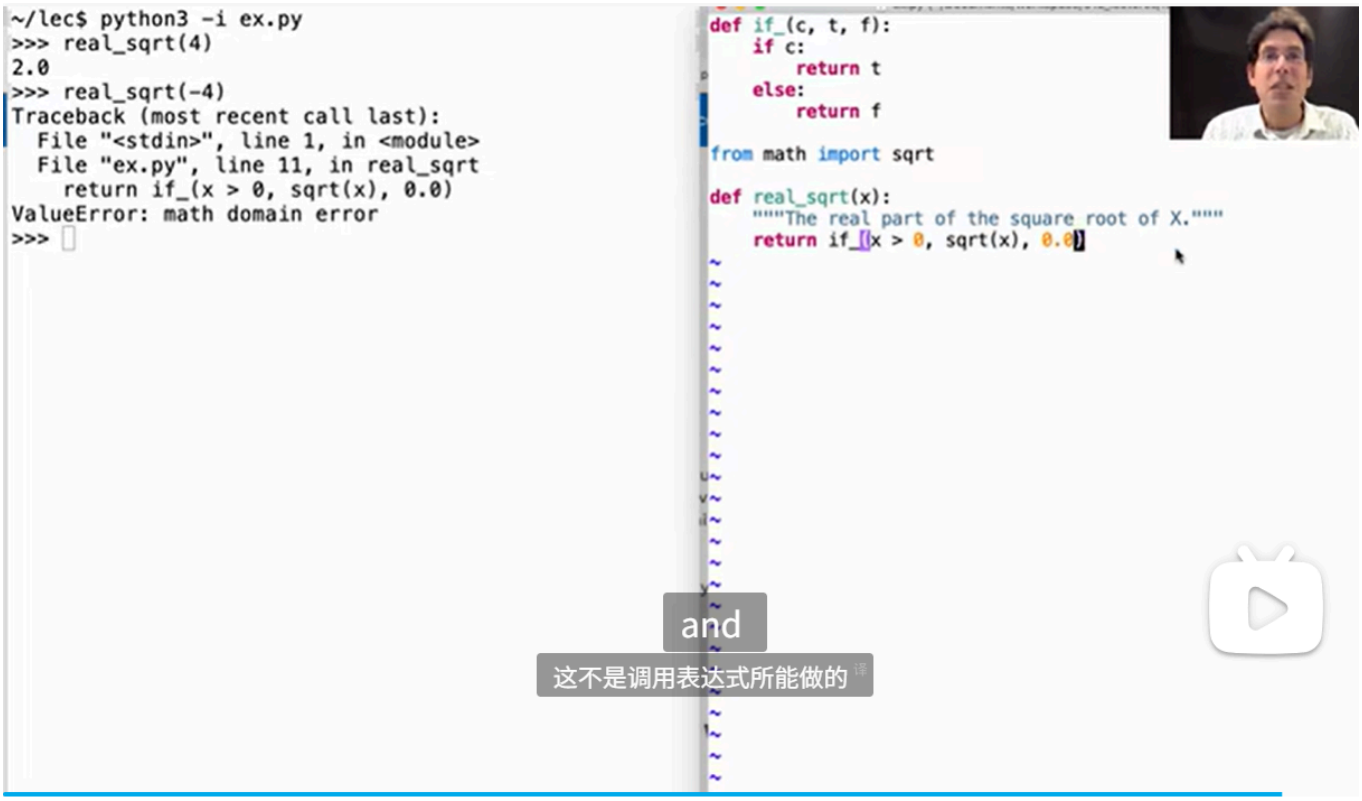



> wow i have to say that this is impressive

# Control




> the same question in Q5 of hw1




# Control Expressions

## Logical Operators

To evaluate the expression **<left> and <right>**:

1. Evaluate the subexpression **<left>**.
2. If the result is a false value **v**, then the expression evaluates to **v**.
3. Otherwise, the expression evaluates to the value of the subexpression **<right>**.

To evaluate the expression **<left> or <right>**:

1. Evaluate the subexpression **<left>**.
2. If the result is a true value **v**, then the expression evaluates to **v**.
3. Otherwise, the expression evaluates to the value of the subexpression **<right>**.

let's look at a couple of examples

让我们看几个例子

(Demo)

## Conditional Expressions

A conditional expression has the form

**<consequent> if <predicate> else <alternative>**

**Evaluation rule:**

1. Evaluate the **<predicate>** expression.
2. If it's a true value, the value of the whole expression is the value of the **<consequent>**.
3. Otherwise, the value of the whole expression is the value of the **<alternative>**.

字幕已关闭

```
>>> x = 0
>>> abs(1/x if x != 0 else 0)
0
```