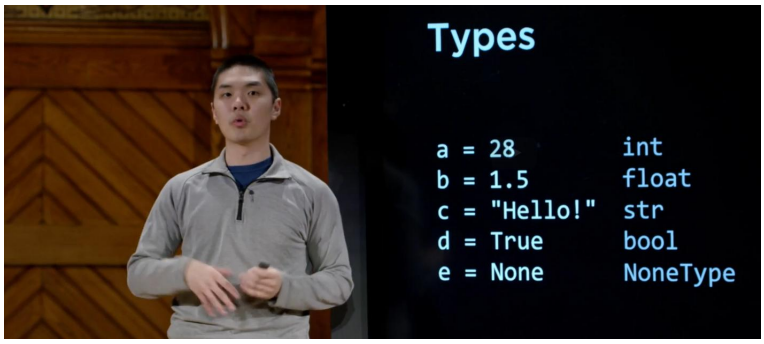
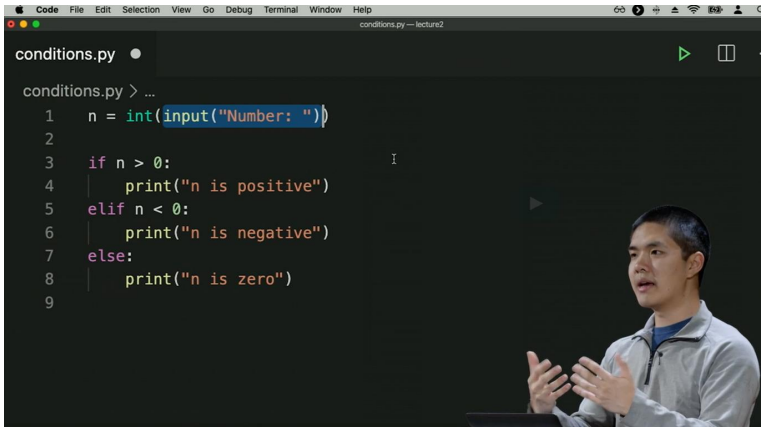
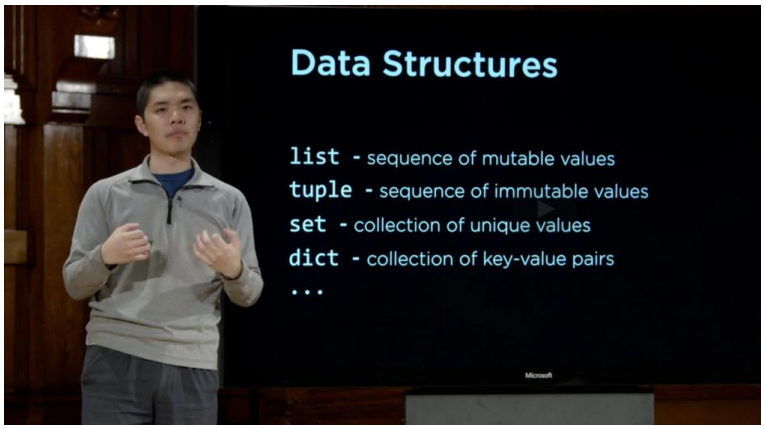
Module 2 - Python



And also, we have a special type in Pythoncalled the none type, which only has one possible value, this capital N, none. And none as a value we'll use whenever we want to represent the lack of a value somewhere. So if we have a function that is not returning anything, it is really returning none, effectively.

Another way that's quite popular in later versions of Python 3is a method known as using **f strings**, short for formatted strings. And in order to use f strings in Python, it's going to be a similar but slightly different syntax. Instead of just having a string in double quotation marks, we'll put the letter f before the string. And inside of the string, I can now say hello comma-- and then if in a formatted string, if I want to plug in the value of a variable, I can do so by specifying it in curly braces. So what I'll say here is, inside of curly braces, name. And so what's going on here is I am telling this formatted string to substitute right here the value of a variable.





Well, this input function doesn't care what you type in.It's always going to give you back a string.

And in Python, there are a couple of different ways to create a comment. But the simplest way is just to use the pound sign or the hashtag. As soon as you include that, everything after that for the remainder of the line is a comment.

And so list can definitely quite powerful anytime you need to store elements in order, a list is definitely a useful tool that Python gives to you.

```
print(f"The set has {len(s)} elements.")
```

functions.py

```
def square(x):
    return x * x

for i in range(10):
    print(f"The square of {i} is {square(i)}")
```

functions.py | squares.py

```
from functions import square

for i in range(10):
    print(f"The square of {i} is {square(i)}")
```

functions.py | squares.py

```
import functions

for i in range(10):
    print(f"The square of {i} is {functions.square(i)}")
```

总结:

classes.py > Flight > __init__

```python
class Flight():
    def __init__(self, capacity):
        self.capacity = capacity
        self.passengers = []

    def add_passenger(self, name):
        if not self.open_seats():
            return False
        self.passengers.append(name)
        return True

    def open_seats(self):
        return self.capacity - len(self.passengers
```

```python

flight = Flight(3)

people = ["Harry", "Ron", "Hermione", "Ginny"]
for person in people:
    success = flight.add_passenger(person)
    if success:
        print(f"Added {person} to flight successfully.")
    else:
        print(f"No available seats for {person}")
```

I can say if not self.open_seats. This is equivalent to me saying in this case, like, if self.open_seats equals equals zero, meaning there are no open seats, a more Pythonic way, so to speak, of expressing this idea is just saying if not self.open_seats.
... So that is a brief look at **object-oriented programming**.

decorators.py

decorators.py > announce

```python
def announce(f):
    def wrapper():
        print("About to run the function...")
        f()
        print("Done with the function.")
    return wrapper
```

decorators.py

decorators.py > ...

```python
def announce(f):
    def wrapper():
        print("About to run the function...")
        f()
        print("Done with the function.")
    return wrapper

@announce
def hello():
    print("Hello, world!")
```

Now, there are a couple of final examples that are just worth taking a look at just to give you some exposure to some of the other features that are available in Python. One thing that will be coming up soon is the idea of **decorators**. And just as we can take a value in Python like a number and modify the value, decorators are a way in Python of taking a function, and modifying that function, adding some additional behavior to that function. And the idea of a decorator is a decorator is going to be a function that takes a function of input and returns a modified version of that function as output. So unlike other programming languages where functions just exist on their own and they can't be passed in as input or output to other functions, in Python, a function is just a value like any other. You can pass it as input to another function. You can get it as the output of another function. And this is known as a **functional programming paradigm**, where functions are themselves values.

So let's create a function that modifies another function by announcingthat the function is about to run and that the function has completed run, just to demonstrate. So this Announce function will take, as input, a function f. And it's going to return a new function. And usually, this function wraps up this function f with some additional behavior, and for that reason, is often called a **wrapper function**. So we may call this wrapper to say that, all right, whatis my wrapper function going to do? It's first going to print about to run the function just to announce that we're about to run the function. That's what I want my Announce decorator to do. Then let's actually run the function f. And then let's print done with the function. So what my Announce decorator is doing is it's taking the function f and it's creating a new function that just announces, via a print statement, before and after the function is done running. And then at the end, we'll return this new function, which is the wrapper function. So this right here is what we might call a decorator, a function that takes a function, modifies it by adding some additional capabilities to it, and then gives us back some output (左图).

So now here, I can define a function called Hello that just prints "hello,world!". And then to add a decorator, I use the at symbol. I can say at announce to say add the Announce decorator to this function. And then I'll just run the function Hello. And we'll see what happens. I'll run Python decorators.py. And I see about to run the function, then "hello, world," then done with the function.

- So again, why did that work?It's because our Hello function that just printed 'hello, world" is wrapped inside of this Announce decorator, where what the Announce decorator does, is it takes our Hello function of input and gets us a new function that first prints an alert warning that we're about to run the function, actually runs the function, and then prints another message.
- So, this is just a bit of a simple example.But there's a lot of power in decorators for being able to very quickly take a function and add capability to it. You might imagine in a web application, if you only want certain functions to be able to run, if a user is logged in, you can imagine writing a decorator that checks to make sure that a user is logged in, and then just using that decorator on all of the functions that you want to make sure only work when a user so happens to be logged in. So decorators are a very powerful tool that web application frameworks like Django can make use of just to make the web application development process a little bit easier as well.

总结:

lambda.py

```python
people = [
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Cho", "house": "Ravenclaw"},
    {"name": "Draco", "house": "Slytherin"}
]

people.sort()

print(people)
```

```
workspace@Brian-MBP lecture2 % python lambda.py
Traceback (most recent call last):
  File "lambda.py", line 7, in <module>
    people.sort()
TypeError: '<' not supported between instances of 'dict' and 'dict
'
workspace@Brian-MBP lecture2 %
```

lambda.py

```python
people = [
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Cho", "house": "Ravenclaw"},
    {"name": "Draco", "house": "Slytherin"}
]

def f(person):
    return person["name"]

people.sort(key=f)

print(people)
```

```
workspace@Brian-MBP lecture2 % python lambda.py
[{'name': 'Cho', 'house': 'Ravenclaw'}, {'name': 'Draco', 'house':
 'Slytherin'}, {'name': 'Harry', 'house': 'Gryffindor'}]
workspace@Brian-MBP lecture2 %
```

But in the trace-back, you'll see that the line of code that it's catching onis people.sort. Somehow, people.sort is causing a type error because it's trying to use less than to compare two dictionaries. And what this appears to mean is that Python doesn't know how to sort these dictionaries. It doesn't know, does Harry belong before or after Cho because it doesn't know how to compare these two elements. And so if I want to do something like this, then I need to tell the sort function how to sort these people.

- And so in order to do that, one way I could do this is by defining a function that tells the sort function how to do the sorting, what to look at when sorting. So if I want to sort by people's name, let me define a functionthat I'll just call f, that takes a person as input and returns that person's name by looking up the name field inside of the dictionary. And now I can sort people by their name by saying sort key equals f. What this means is sort all the people. And the way to sort them, the way you know how to compare them, is by running this function where this function takes a person and gives us back their name. And this will sort everyone by name. Now, if I run Python lambda.py, you will see that I firstget Cho, then Draco, then Harry in alphabetical order by name。
- Whereas if instead I had tried to sort people by their house by changing my function that I'm using to sort and then rerun this, now I see that its first Harry who is in Gryffindor, then Ravenclaw, then Slytherin. So we get the houses in alphabetical order instead. But the reason I show this is because this function is so simple and is only used in one place. (下图左)

Python actually gives us an easier way to represent a very short, one-line function using **lambda expression**. And this is a way of including the function just as a single value on a single line. I can say instead of defining a function called f, I can get rid of all of this and just say, sort by this key, a lambda, which is a function that takes a person and returns the person's name. So we say person as the input, colon person name as the output. This is a condensed way of saying the same thing we saw a moment ago, of defining a function, giving it a name, and then passing in the name here. (下图右)

lambda.py

```python
people = [
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Cho", "house": "Ravenclaw"},
    {"name": "Draco", "house": "Slytherin"}
]

def f(person):
    return person["house"]

people.sort(key=f)

print(people)
```

```
workspace@Brian-MBP lecture2 % python lambda.py
[{'name': 'Harry', 'house': 'Gryffindor'}, {'name': 'Cho', 'house'
: 'Ravenclaw'}, {'name': 'Draco', 'house': 'Slytherin'}]
workspace@Brian-MBP lecture2 %
```

lambda.py

```python
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Cho", "house": "Ravenclaw"},
    {"name": "Draco", "house": "Slytherin"}
]

people.sort(key=lambda person: person["name"])

print(people)
```

总结:

exceptions.py > ...

```python
import sys

x = int(input("x: "))
y = int(input("y: "))

try:
    result = x / y
except ZeroDivisionError:
    print("Error: Cannot divide by 0.")
    sys.exit(1)

print(f"{x} / {y} = {result}")
```

```
workspace@Brian-MBP lecture2 % python exceptions.py
x: 5
y: 0
Traceback (most recent call last):
  File "exceptions.py", line 4, in <module>
    result = x / y
ZeroDivisionError: division by zero
workspace@Brian-MBP lecture2 %
```

未加 try...except

```
workspace@Brian-MBP lecture2 % python exceptions.py
x: 5
y: 0
Error: Cannot divide by 0.
workspace@Brian-MBP lecture2 %
```

Well, when I do that, I get an exception. I get a zero division error, which is an error that happens whenever you try to divide by zero. What I'd like to happen though in this case is not for my program to display kind of a messy error and a trace-back like this, but to handle the exception gracefully, so to speak. To be able to catch when the user does something wrong and report a nicer looking message instead. And so how might I go about doing that. Well, one thing I can do here is instead of just saying result equals x over y, I can say try to do this, try to set result equal to x divided by y, I can say try to do this, try to set result equal to x divided by y,and then say except if a zero division error happens. Then let's do something else. Let's print error cannot divide by zero, and then exit the program. How do you exit the program? It turns out there's a module in Python called sys. And if I import the sys module, I can say sys.exit1 to mean exit the program with a status code of one, where a status code of one generally means something went wrong in this program.
So now let's try it, now if I try five and zero, press Return, I get an error. Cannot divide by zero-- no long exception that's going to look complicated to the user. It's no longer messy. I've been able to handle the exception gracefully.

So those are some of the key features now with this Python programming language, this language that gives us the ability to define these functions, and loops, and conditions in very convenient ways, to create classes where we can begin to build objects that are able to perform various different types of tasks. And next time using Python, we'll be able to design web applications such that users are able to make requests to our web applications and get some sort of response back.

总结: