

# CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding  
Oct 2 2019, Lecture 16



# Agenda

---

- Agenda:
  - Access Control in OOP
  - Data structures and algorithms

# Public, private and protected access modifiers

---

- Classical Objected-Oriented languages, such as C++ and Java controls the access to class resources by public, private and protected keywords
  - **Public** - accessible from outside the class
  - **Private** - they can be handled only from within the class
  - **Protected** - accessible from within the class and also available to its child classes

# Public, private and protected access modifiers

---

- Python **doesn't have** any mechanism that effectively restricts access to any instance variable or method
- All members in a Python are **public** by default. Any member can be accessed from outside the class environment
- Python prescribes a convention of prefixing the name of the variable/method with single or double underscore to emulate the behavior of protected and private access specifiers

# Public, private and protected access modifiers

- public by default

```
1 class Employee:
2     def __init__(self, name, sal):
3         self.name = name
4         self.salary = sal
5
6 e1 = Employee("carl", 10000)
7 print(e1.name)
8
```

access x

```
/Users/xinyi/anaconda/envs/torch/bin/python /Users/xinyi/Courses/cs1340/week6/access.py
carl
```

Process finished with exit code 0

- use one \_ for protected

```
1 class Employee:
2     def __init__(self, name, sal):
3         self._name = name # protected attribute
4         self._salary = sal # protected attribute
5
6 e1 = Employee("carl", 10000)
7 print(e1._salary)
8
```

access x

```
/Users/xinyi/anaconda/envs/torch/bin/python /Users/xinyi/Courses/cs1340/week6/access.py
10000
```

Process finished with exit code 0

# Public, private and protected access modifiers

- Use two underscore `__` for private

```
1 class Employee:
2     def __init__(self, name, sal):
3         self.__name = name    # private attribute
4         self.__salary = sal   # private attribute
5
6 e1 = Employee("carl", 10000)
7 print(e1.__salary)
8
```

access x

```
/Users/xinyi/anaconda/envs/torch/bin/python /Users/xinyi/Courses/cs1340/week6/access.py
Traceback (most recent call last):
  File "/Users/xinyi/Courses/cs1340/week6/access.py", line 7, in <module>
    print(e1.__salary)
AttributeError: 'Employee' object has no attribute '__salary'

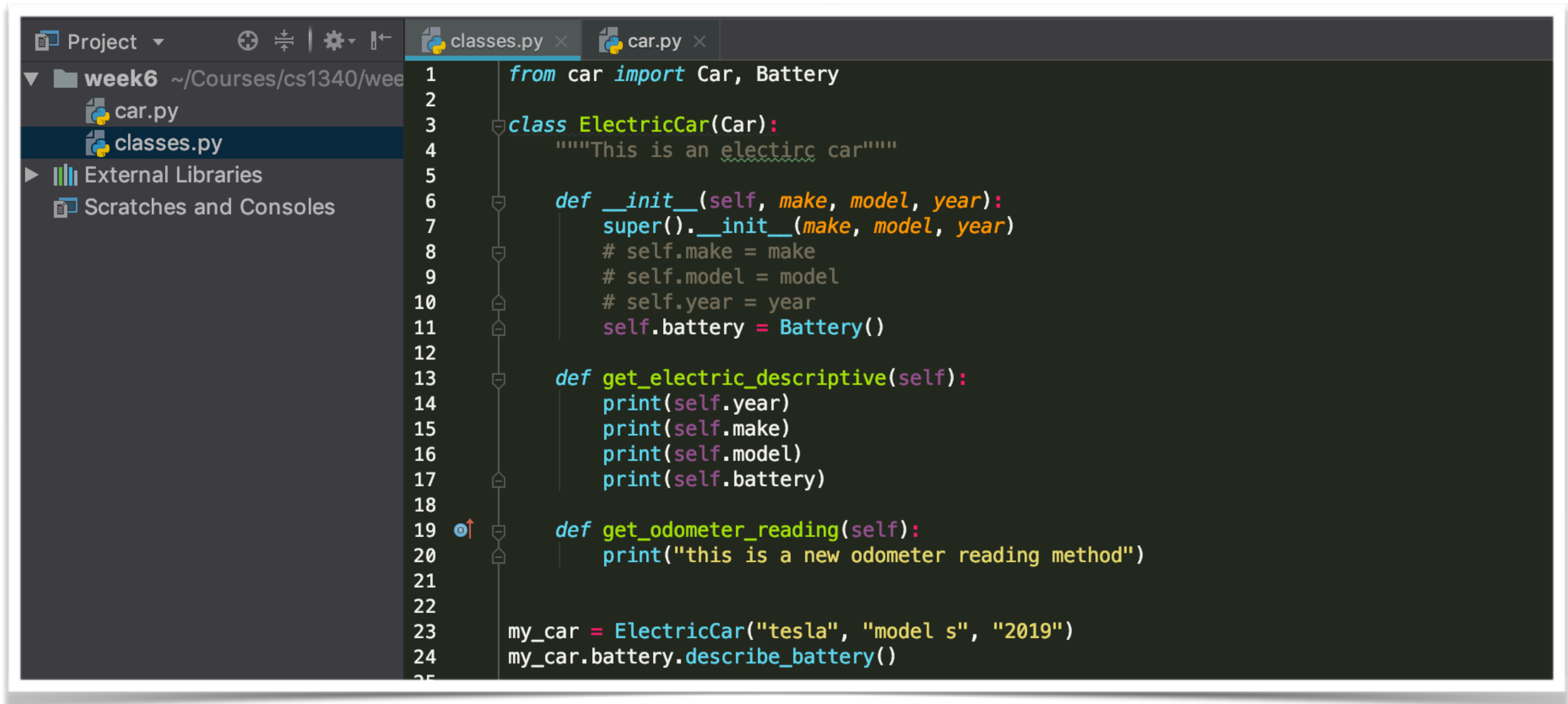
Process finished with exit code 1
```

- But if you are using Python 2. Python performs name mangling of private variables. Every member with double underscore will be changed to `_object._class__variable`. If so required, it can still be accessed from outside the class, but the practice should be refrained.



# Importing Classes

- Store classes in modules and import the classes you need into your main program



```
1  from car import Car, Battery
2
3  class ElectricCar(Car):
4      """This is an electric car"""
5
6      def __init__(self, make, model, year):
7          super().__init__(make, model, year)
8          # self.make = make
9          # self.model = model
10         # self.year = year
11         self.battery = Battery()
12
13     def get_electric_descriptive(self):
14         print(self.year)
15         print(self.make)
16         print(self.model)
17         print(self.battery)
18
19     def get_odometer_reading(self):
20         print("this is a new odometer reading method")
21
22
23 my_car = ElectricCar("tesla", "model s", "2019")
24 my_car.battery.describe_battery()
```



# The Python standard library

- The *Python standard library* is a set of modules included with every Python installation
- You can use any function or class in the standard library by including a simple import statement

```
1  import datetime
2
3  # This class method creates a datetime object with the current date and time
4  now = datetime.datetime.today()
5
6  print(now.year)
7  print(now.hour)
8  print(now.minute)
9
10 long_ago = datetime.datetime(1999, 3, 14, 12, 30, 58)
11
12 print(long_ago)
13 print(long_ago < now)
```

std x

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week6/std.py
2019
15
22
1999-03-14 12:30:58
True
```

# The Python standard library

- math

```
1  import math
2
3  # These are constant attributes, not functions
4
5  print(math.pi)
6  print(math.e)
7
8  # Round a float up or down
9  print(math.ceil(3.3))
10 print(math.floor(3.3))
11
12 # Natural logarithm
13 print(math.log(5))
14
15 # Square root
16 print(math.sqrt(10))
17
18 # Trigonometric functions
19 print(math.sin(math.pi/2))
20 print(math.cos(0))
21
```

std ×

/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week6/std.py

```
3.141592653589793
2.718281828459045
4
3
1.6094379124341003
3.1622776601683795
1.0
1.0
```

# The Python standard library

- OrderedDict

```
1  from collections import OrderedDict
2
3  favorite_languages = OrderedDict()
4
5  favorite_languages["Alice"] = "Python"
6  favorite_languages["Sarah"] = "C"
7  favorite_languages["Jake"] = "Ruby"
8  favorite_languages["Phil"] = "Python"
9
10 for name, language in favorite_languages.items():
11     print(name + "'s favorite language is " + language)
12
13 # print("\n")
14
```

```
favorite_language x
/Users/xinyi/anaconda/envs/torch/bin/python /Users/xinyi/Courses/cs1340/week6/favorite_language.py
Alice's favorite language is Python
Sarah's favorite language is C
Jake's favorite language is Ruby
Phil's favorite language is Python
```

# The Python standard library

- Unordered Dict before Python 3.6

```
15 # Since Python 3.6, dict are ordered
16 favorite_fruits = {}
17 favorite_fruits["Bob"] = "apple"
18 favorite_fruits["Ethan"] = "banana"
19 favorite_fruits["Alice"] = "orange"
20 favorite_fruits["Carl"] = "pear"
21
22 for name, fruit in favorite_fruits.items():
23     print(name + "'s favorite fruit is " + fruit)
```

favorite\_language ×

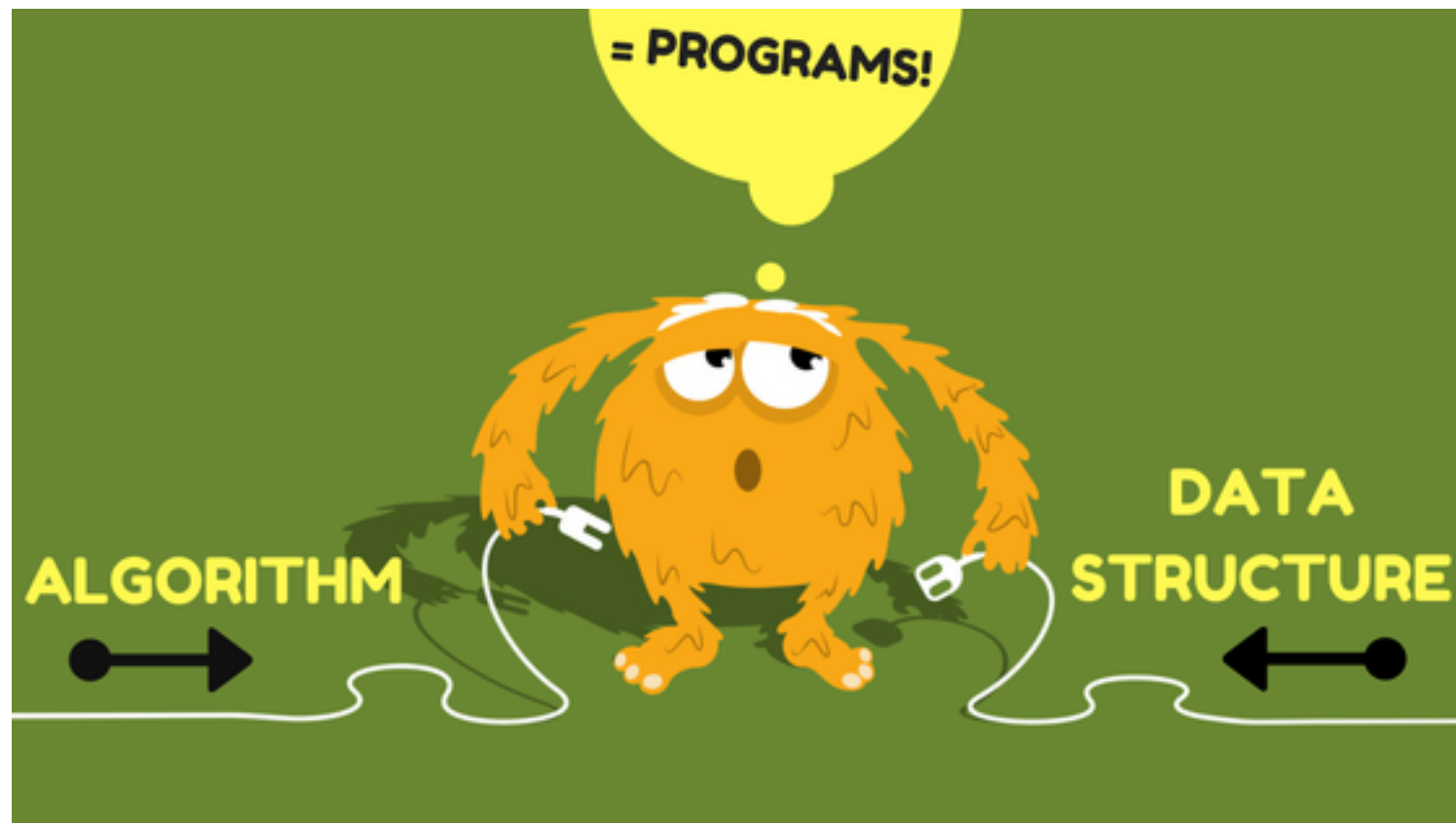
/Users/xinyi/anaconda/envs/torch/bin/python /Users/xinyi/Courses/cs1340/week6/favorite\_language.py

```
Carl's favorite fruit is pear
Bob's favorite fruit is apple
Alice's favorite fruit is orange
Ethan's favorite fruit is banana
```

# Data Structure and Algorithm

---

- Programs = Data Structures + Algorithms



# Data Structure and Algorithm

---

- While trying to describe problems, we often realize they involve some kind of data we have to manipulate. It can be numbers, letters or something more complicated.
- Python comes with some predefined ways of representing data while writing our programs.
  - `int`
  - `string`
  - `boolean`
  - `float`

# Data Structure and Algorithm

---

- Simple data types are great when dealing with individual values, but what happens when we need to perform operations on our data collectively?
- *Data Structures* allow us to group data together and describe the attributes and actions that can be performed on a particular instance of the data.
- Simple built in data structures in Python
  - *list*
  - *tuple*
  - *dict*

# Data Structure and Algorithm

---

- Once we have modeled our real world data using *data structures*.
- We have to describe to the computer, step by step, instructions for solving a particular problem. This is often referred to as an *algorithm*.





# Data Structure and Algorithm

---

- Common data structures
  - Array (list in Python)
  - Linked-list
  - Queue
  - Stack
  - Trees
  - Graph
  - Hash-Tables
  - ...

# Data Structure and Algorithm

- Multiple dimensional list
  - 2-D list

	Column 0	Column 1	Column 2
Row 0	<b>x[0][0]</b>	<b>x[0][1]</b>	<b>x[0][2]</b>
Row 1	<b>x[1][0]</b>	<b>x[1][1]</b>	<b>x[1][2]</b>
Row 2	<b>x[2][0]</b>	<b>x[2][1]</b>	<b>x[2][2]</b>

- 3-D list

