

CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding
Sep 25 2019, Lecture 13



Announcement

- **SMU CS Department Student Advisory Board** is looking for members if you are a CS major/minor.
 - representatives of the CS student body
 - help convey ideas/concerns/etc to faculty.
- The application can be accessed at this link: <https://forms.gle/PrZLcCkFtxh11eFF7> and is due **Friday, September, 27th at 5pm!**
- If you have questions, you can email advisory board graduate chair, Yasamin Fozouni yfozouni@smu.edu or undergraduate chair, Sunjoli Aggarwal sunjolia@smu.edu.

Announcement

- Homework 1 graded
- If your lab session is on Monday, its due 9:59am, if on Tuesday its due 12:59pm, right before the start of the lab (not midnight)
- Quiz this Friday (Sep 27)
 - take ~10mins at the beginning of the class, topics from week1 to week4 (review the slides)
- Midterm Exam will be on Oct 18 (Friday, after fall break).

Agenda

- Agenda:
 - Common errors in homework 1
 - Review concepts about exception
 - Object Oriented Programming

Homework 1

- Define a boolean type variable
 - `is_vegetarian = True`
- Define a string
 - Use `'''` or `"`, Use `()` to define a tuple

Exceptions

- When an error occurs, or exception as we call it, Python will normally stop and generate an error message
- We could handle these errors gracefully using try-except block
- Use try-except to prevent crash (crash is very bad and not user friendly)

Object Oriented Programming

- Object-oriented programming (OOP) is one of the most effective approaches to writing software
- In OOP, you write *classes* that represent real-world things and situations
- You create *objects* based on these *classes*
- When you write a *class*, you define the general behavior that a whole category of objects can have

Object Oriented Programming

- Procedural Oriented Programming
 - program is divided into small parts called functions
 - follows top down approach
 - no access specifier
- Object Oriented Programming
 - program is divided into small parts called objects
 - follows bottom up approach
 - have access specifiers like private, public, protected etc.
- **Understanding object-oriented programming will help you see the world as a programmer does**

Object Oriented Programming

- Python is an object oriented programming language
- Almost everything in Python is an object, with its properties (attributes) and methods.
 - int, string, file_object, etc
- A Class is like an object constructor, or a "blueprint" for creating objects
- Making an object from a class is called *instantiation*, and you work with *instances* of a class (one class definition, multiple instances)

Object Oriented Programming

- Creating and Using a Class

```
1 class Dog():
2     """ A simple attempt to model a dog"""
3
4     def __init__(self, name, age):
5         """Initialize name and age attributes. """
6         self.name = name
7         self.age = age
8
9     def sit(self):
10        """Simulating a dog sitting in response to a command"""
11        print(self.name.title() + "is now sitting")
12
13    def roll_over(self):
14        """Simulating rolling over in response to a command"""
15        print(self.name.title() + "rolled over!")
16
```

- Use the keyword **class** to define a class
- A function that is part of a class is called method
 - Everything you learnt about function apply to method

Object Oriented Programming

```
1 class Dog():
2     """ A simple attempt to model a dog"""
3
4     def __init__(self, name, age):
5         """Initialize name and age attributes. """
6         self.name = name
7         self.age = age
8
9     def sit(self):
10        """Simulating a dog sitting in response to a command"""
11        print(self.name.title() + "is now sitting")
12
13    def roll_over(self):
14        """Simulating rolling over in response to a command"""
15        print(self.name.title() + "rolled over!")
16
```

- `__init__` method
 - The `__init__` method is a special method Python runs automatically whenever we create a new instance based on the Dog class

Object Oriented Programming

```
1 class Dog():
2     """ A simple attempt to model a dog"""
3
4     def __init__(self, name, age):
5         """Initialize name and age attributes. """
6         self.name = name
7         self.age = age
8
9     def sit(self):
10        """Simulating a dog sitting in response to a command"""
11        print(self.name.title() + "is now sitting")
12
13    def roll_over(self):
14        """Simulating rolling over in response to a command"""
15        print(self.name.title() + "rolled over!")
16
```

- The **self** parameter is required in the method definition, and it must come first before other parameters
- **self** is a reference to the instance itself.
- **self** is passed automatically, so we don't need to pass it
- variables prefixed with **self** is available to every method in the class

Object Oriented Programming

```
1 class Dog():
2     """ A simple attempt to model a dog"""
3
4     def __init__(self, name, age):
5         """Initialize name and age attributes. """
6         self.name = name
7         self.age = age
8
9     def sit(self):
10        """Simulating a dog sitting in response to a command"""
11        print(self.name.title() + "is now sitting")
12
13    def roll_over(self):
14        """Simulating rolling over in response to a command"""
15        print(self.name.title() + "rolled over!")
16
```

- The Dog class has two other methods: sit() and roll_over()
 - don't need additional information like __init__()
 - only one parameter **self**

Object Oriented Programming

- Creating and Using a Class

```
1 class Dog():
2     """ A simple attempt to model a dog"""
3
4     def __init__(self, name, age):
5         """Initialize name and age attributes. """
6         self.name = name
7         self.age = age
8
9     def sit(self):
10        """Simulating a dog sitting in response to a command"""
11        print(self.name.title() + "is now sitting")
12
13    def roll_over(self):
14        """Simulating rolling over in response to a command"""
15        print(self.name.title() + "rolled over!")
16
17    my_dog = Dog("shadow", 6)
18    my_dog.sit()
19    my_dog.roll_over()
20
21
```

classes ×

/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week5/classes.py

Shadowis now sitting

Shadowrolled over!

Process finished with exit code 0

Object Oriented Programming

- Access attributes and methods
 - Use Dot notation
 - `my_dog.name` <- this is accessing the name
 - `my_dog.age` <- this is accessing the age
 - `my_dog.sit()` <- this is accessing the sit() method

Object Oriented Programming

- Creating multiple instances

```
1 class Dog():
2     """ A simple attempt to model a dog"""
3
4     def __init__(self, name, age):
5         """Initialize name and age attributes. """
6         self.name = name
7         self.age = age
8
9     def sit(self):
10        """Simulating a dog sitting in response to a command"""
11        print(self.name.title() + " is now sitting")
12
13    def roll_over(self):
14        """Simulating rolling over in response to a command"""
15        print(self.name.title() + " rolled over!")
16
17    my_dog = Dog("shadow", 6)
18    my_dog.sit()
19    my_dog.roll_over()
20
21    your_dog = Dog("lucy", 3)
22    your_dog.sit()
23    your_dog.roll_over()
24
25
```

classes x

```
/Users/xinyi/anaconda/envs/mlern/bin/python /Users/xinyi/Courses/cs1340/week5/classes.py
```

```
Shadow is now sitting
```

```
Shadow rolled over!
```

```
Lucy is now sitting
```

```
Lucy rolled over!
```

```
Process finished with exit code 0
```


Object Oriented Programming

- Another example

```
1 class Car():
2     """A simple attempt to represent a car."""
3
4     def __init__(self, make, model, year):
5         """Initialize attributes to describe a car."""
6         self.make = make
7         self.model = model
8         self.year = year
9
10
11     def get_descriptive_name(self):
12         """Return a neatly formatted descriptive name."""
13         long_name = str(self.year) + " " + self.make + " " + self.model
14         return long_name.title()
15
16
17 my_new_car = Car("bmw", "m3", 2016)
18 print(my_new_car.get_descriptive_name())
19
```

Car > get_descriptive_name()

classes x

/Users/xinyi/anaconda/envs/mlern/bin/python /Users/xinyi/Courses/cs1340/week5/classes.py
2016 Bmw M3

Process finished with exit code 0

Object Oriented Programming

- The objects we have encountered so far
 - string objects, `"hello".upper()`
 - list objects, `[3, 4, 5, 10].append(11)`
 - dictionaries, `a_dict = {"name": "cart", "age": 18},`
`a_dict.keys()`

Demo



Object Oriented Programming

- Working with the classes and Instances (objects)
- Add a new attribute and method to our Car class

```
1 class Car():
2     """A simple attempt to represent a car."""
3
4     def __init__(self, make, model, year):
5         """Initialize attributes to describe a car."""
6         self.make = make
7         self.model = model
8         self.year = year
9
10
11     def get_descriptive_name(self):
12         """Return a neatly formatted descriptive name."""
13         long_name = str(self.year) + " " + self.make + " " + self.model
14         return long_name.title()
15
16
17 my_new_car = Car("bmw", "m3", 2016)
18 print(my_new_car.get_descriptive_name())
19
```

Car > get_descriptive_name()

classes ×

/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week5/classes.py
2016 Bmw M3

Process finished with exit code 0

Object Oriented Programming

- Setting a Default value for an attribute

```
1  class Car():
2      """A simple attempt to represent a car."""
3
4      def __init__(self, make, model, year):
5          """Initialize attributes to describe a car."""
6          self.make = make
7          self.model = model
8          self.year = year
9          self.odometer_reading = 0
10
11
12      def get_descriptive_name(self):
13          """Return a neatly formatted descriptive name."""
14          long_name = str(self.year) + " " + self.make + " " + self.model
15          return long_name.title()
16
17      def read_odometer(self):
18          """Print a statement showing the car's mileage"""
19          print("This car has " + str(self.odometer_reading) + " miles on it.")
20
21  my_new_car = Car("bmw", "m3", 2016)
22  print(my_new_car.get_descriptive_name())
23  my_new_car.read_odometer()
```

Object Oriented Programming

- Modifying an Attribute's Value Directly
 - Use the Dot operation to access the attribute directly and then modify its value

```
21 my_new_car = Car("bmw", "m3", 2016)
22 print(my_new_car.get_descriptive_name())
23 my_new_car.read_odometer()
24
25 my_new_car.odometer_reading = 23
26 my_new_car.read_odometer()
```

Object Oriented Programming

- Modifying an Attribute's Value Through a Method (preferred way)
- You can do some validations about the incoming new value.

```
21 def update_odometer(self, mileage):  
22     """  
23     Set the odometer reading to the given value.  
24     Reject the change if it attempts to roll the odometer back.  
25     :param mileage: <int> - the new value of mileage  
26     :return: None  
27     """  
28     if mileage >= self.odometer_reading:  
29         self.odometer_reading = mileage  
30     else:  
31         print("You can't roll back an odometer!")
```