

# CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding  
Sep 23 2019, Lecture 12



# Announcement

---

- Lab 1 is due today or tomorrow (no late submission)
- Lab 2 posted
- Quiz this Friday (Sep 27)
  - may take ~10mins at the beginning of the class
- Midterm Exam will be on Oct 18 (Friday, after fall break).

# Agenda

---

- Agenda:
  - Lab 2
  - Review concepts about file handling
  - Exceptions
  - Object Oriented Programming

# Python Input/Output

---

- Instead of hard code the value of variables into the source code.
- To Allow flexibility we might want to take the input from the user. We use `input()` function in python

`input([prompt])`

Where **prompt** is the string we wish to display on the screen, it is optional

# Python File Handling

---

- The key function for working with files is the `open()` function.
- The `open()` function takes two parameters: *filename* and *mode*
- There are four different modes for opening a file
  - `"r"` - Read-Default value. Opens a file for reading, error if the file does not exist
  - `"a"` - Append- Opens a file for appending, creates the file if it does not exist
  - `"w"` - Write- Opens a file for writing, create the file if it does not exist
  - `"x"` - Create -Creates the specified file, returns an error if the file exists

# Python File Handling

---

- With Statement
  - The keyword **with** closes the file once access to it is no longer needed
  - **with** syntax:

Indentation(4 spaces) → **with open(filename) as the\_file:**  
                                  **contents = the\_file.read()**  
                                  **print(contents)**

                                  ...

First line with less indentation is considered to be outside of the with code block

Colon

The diagram illustrates the Python 'with' statement syntax. It shows a code block with three lines: 'with open(filename) as the\_file:', 'contents = the\_file.read()', and 'print(contents)'. An orange arrow points from the text 'Indentation(4 spaces)' to the first line of the code block. Another orange arrow points from the text 'Colon' to the colon at the end of the first line. A third orange arrow points from the text 'First line with less indentation is considered to be outside of the with code block' to the first line of the code block. Below the code block, there are three red dots '...'.

# 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)

# Exceptions

- An example, pay attention to the exit code

```
1 print(x)
2 print("This line will not print, because an error occurred")
3
```

exceptions ×

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week4/exceptions.py
Traceback (most recent call last):
  File "/Users/xinyi/Courses/cs1340/week4/exceptions.py", line 1, in <module>
    print(x)
NameError: name 'x' is not defined

Process finished with exit code 1
```

```
1 try:
2     print(x)
3 except:
4     print("An exception occurred")
5     print("We could do something else")
6
7 print("This line will print")
8
```

exceptions ×

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week4/exceptions.py
An exception occurred
We could do something else
This line will print

Process finished with exit code 0
```



# Exceptions

- Python has a lot of built in Errors
  - NameError
  - ZeroDivisionError
  - FileNotFoundError
  - ...

```
1  try:
2      print(x)
3  except NameError:
4      print("An exception occurred")
5      print("We could do something else")
6
7  print("This line will print")
8
```

exceptions ×


```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week4/exceptions.py
```

```
An exception occurred
We could do something else
This line will print
```

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

# Exceptions

- ZeroDivisionError

```
1 x = 18
2 y = 0
3 try:
4     z = x / y
5 except ZeroDivisionError:
6     print("You can not divide a number by 0")
7     
8 print("This line will print")
9
```

exceptions ×

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week4/exceptions.py
```

```
You can not divide a number by 0
```

```
This line will print
```

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

# Exceptions

- FileNotFoundError

```
1 filename = "program.txt"
2
3 try:
4     with open(filename) as the_file:
5         contents = the_file.read()
6         print(contents)
7 except FileNotFoundError:
8     print("File not found, please check your filename is correct")
9
10 print("This line will print")
```

exceptions ×

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week4/exceptions.py
File not found, please check your filename is correct
This line will print

Process finished with exit code 0
```

# Exceptions

- Use the wrong Error will not catch the exception (when you are not sure what error may raise, just leave it empty)
- FileNotFoundError will not catch a NameError

```
1  try:
2      print(x)
3  except FileNotFoundError:
4      print("An exception occurred")
5      print("We could do something else")
6
7  print("This line will NOT print")
8
```

exceptions ×

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week4/exceptions.py
Traceback (most recent call last):
  File "/Users/xinyi/Courses/cs1340/week4/exceptions.py", line 2, in <module>
    print(x)
NameError: name 'x' is not defined

Process finished with exit code 1
```

# Exceptions

- Many exceptions

```
1  try:
2      print(x)
3  except FileNotFoundError:
4      print("An exception occurred")
5      print("We could do something else")
6  except NameError:
7      print("There is a name error ")
8
9  print("This line will print")
10
```

exceptions ×

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week4/exceptions.py
```

```
There is a name error
```

```
This line will print
```

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

# Exceptions

- Else
  - You can use `else` keyword to define a block of code to be executed if no errors were raised

```
1  try:
2      print("Hello")
3  except:
4      print("Something went wrong")
5  else:
6      print("Nothing went wrong")
7
8  print("This line will print")
9
```

else

exceptions x

/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week4/exceptions.py

Hello

Nothing went wrong

This line will print

Process finished with exit code 0

# Exceptions

- finally block
  - The finally block, if specified, will be executed regardless if the try block raise an error or not.

```
1  try:
2      print(x)
3  except:
4      print("Something went wrong")
5  finally:
6      print("The 'try except' is finished")
-
```

exceptions ×

```
/Users/xinyi/anaconda/envs/mlern/bin/python /Users/xinyi/Courses/cs1340/week4/exceptions.py
Something went wrong
The 'try except' is finished

Process finished with exit code 0
```

# Demo

---





# 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

- 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

# Demo

---

