Week2

February 8, 2018

1 Week 2

1.1 Programming Paradigms and OOP

Ayal Gussow, 02/08/2018

2 Basic Outline

- What are programming paradigms?
- Object-Oriented Programming
- In-Class / HW Assignment

3 What does "programming paradigm" mean?

- Really, it's a convenience for the programmer
- It's how the programmer structures and writes a program
- For our purposes, the paradigm does not effect what your program does

4 Let's see an example: procedural

- Conceptually simple: instructions happen, in order
- Data are stored in variables
- An emphasis on modular sub-routines that accept data as arguments (scope)

```
In [ ]: location = "Bethesda"
        zip_code = 20892
        elevation = 71.10

        print(
            "My location is ",
            location,
            ". My elevation is ",
            elevation,
```

```
            ". My zip code is ",
            zip_code,
            ".",
            sep=""
        )
```

```
In [1]: """
        This module contains functions for checking whether a number is even.
        """
        def is_even(n):
            """Returns true if n is even, False otherwise"""
            assert isinstance(n, int), "This function requires an integer!"
            return not n % 2
```

```
In [2]: # Note that it's repeatable, so:
        #     1) Assertion is always run
        #     2) Less likely to get error
        #     3) If error, easy to fix in one place

        my_number = 10

        if is_even(my_number):
            print("It's even!")
        else:
            print("It's odd!")
```

```
It's even!
```

5 A more complex example: people

- Let's say you're writing an NBA video game. How would you store people?

```
In [3]: # Let's write some code for an NBA video game
        lebron = {
            'first_name': 'LeBron',
            'last_name': 'James',
            'nickname': "King James",
            'team': 'Cleveland Cavaliers'
        }
        print(lebron)
```

```
{'first_name': 'LeBron', 'last_name': 'James', 'nickname': 'King James', 'team': 'Cleveland Cava
```

```
In [4]: # What if we want a function to print player name?
        def print_player_name(player):
            """Print the player's name. The player parameter must be a dictionary."""
```

```
            print(player["first_name"], ' "', player["nickname"], '" ', player["last_name"],
                  sep="")

        lebron = {
            'first_name': 'LeBron', 'last_name': 'James', 'nickname': "King James", 'team': 'Cle
        }

        print_player_name(lebron)
```

```
LeBron "King James" James
```

6 Last weeks assignment was also a good example

```
people = [
    {'name': 'Charlie', 'age': 35},
    {'name': 'Alice', 'age': 30},
    {'name': 'Eve', 'age': 20},
    {'name': 'Gail', 'age': 30},
    {'name': 'Dennis', 'age': 25},
    {'name': 'Bob', 'age': 35},
    {'name': 'Fred', 'age': 25}]
```

7 Procedural languages

7.0.1 Note: The programming paradigm is part of the language

- Procedural languages: C, Fortran, GO, Python (multiple paradigms)

7.0.2 Downsides:

- Can get very messy:
  - remembering attributes
  - function expectations
  - remembering which functions apply to which data
  - creating new instances
- OOP stores everything under one unified roof

8 OOP Conceptually

8.1 Encapsulation!

- Bundling of data with the methods that operate on that data.
- Hide the values or state of data inside the class with public methods to access the values.

9 OOP Conceptually: NBAPlayer class

Attributes

1. first_name
2. last_name
3. nickname

10 OOP Conceptually: NBAPlayer class

Functions

1. print_name

11 Terminology

- Class - the overall structure description (e.g. "Player").
- Object / Instance - an instance of the class (e.g. "LeBron").
- Constructors / Destroyers
- "self" - a reference to the class instance.

```
In [5]: class NBAPlayer:
            """This class represents an NBA player"""
            def __init__(self,
                         first_name,
                         last_name,
                         nickname):
                """Initialize an instance of the player class"""
                self.first_name = first_name
                self.last_name = last_name
                self.nickname = nickname
```

```
In [10]: class NBAPlayer:
             """This class represents an NBA player"""
             def __init__(self,
                          first_name,
                          last_name,
                          nickname):
                 """Initialize an instance of the player class"""
                 self.first_name = first_name
                 self.last_name = last_name
                 self.nickname = nickname

         lebron = NBAPlayer(
                     "LeBron",
                     "James",
                     "King James"
         )
```

```
        # print(lebron.nickname)
        # print(nickname)
        lebron.nickname = "Bron"
        print(lebron.nickname)
```

```
Bron
```

```
In [15]: class NBAPlayer:
             """This class represents an NBA player"""
             def __init__(self,
                          first_name,
                          last_name,
                          nickname):
                 """Initialize an instance of the player class"""
                 self.first_name = first_name
                 self.last_name = last_name
                 self.nickname = nickname

             def __len__(self):
                 return 3

             def print_name(self):
                 """Print the player's name."""
                 print(self.first_name, ' "', self.nickname, '" ', self.last_name, sep="")

         lebron = NBAPlayer("LeBron", "James", "King James")

         len(lebron)
```

```
Out[15]: 3
```

```
In [16]: lebron = NBAPlayer("LeBron", "James", "King James")
         chris = NBAPlayer("Chris", "Paul", "CP3")

         lebron.print_name()
         chris.print_name()
```

```
LeBron "King James" James
Chris "CP3" Paul
```

12 Terminology

- Class - the overall structure description (e.g. "NBAPlayer").
- Object / Instance - an instance of the class (e.g. "LeBron").
- Constructors / Destroyers

- "self" - a reference to the class instance.
- Hidden variables and name mangling with "__"

```
In [19]: class NBAPlayer:
             """This class represents an NBA player"""
             def __init__(self,
                          first_name,
                          last_name,
                          nickname,
                          team):
                 """Initialize an instance of the player class"""
                 self.first_name = first_name
                 self.last_name = last_name
                 self.nickname = nickname
                 self.__team = team

         lebron = NBAPlayer(
                     "LeBron",
                     "James",
                     "King James",
                     "Cleveland Cavaliers"
         )

         print(lebron.__team)


         ---------------------------------------------------------------------------

         AttributeError                            Traceback (most recent call last)

         <ipython-input-19-655c1159a1d1> in <module>()
          19 )
          20
        ---> 21 print(lebron.__team)


         AttributeError: 'NBAPlayer' object has no attribute '__team'
```

- Bundling of data with the methods that operate on that data.
- Hide the values or state of data inside the class with public methods to access the values.

```
In [22]: class NBAPlayer:
             """This class represents an NBA player"""
             def __init__(self,
                          first_name,
                          last_name,
                          nickname,
                          team):
```

```
            """Initialize an instance of the player class"""
            self.first_name = first_name
            self.last_name = last_name
            self.nickname = nickname
            self.__team = team
lebron = NBAPlayer("LeBron", "James", "King James", "Cleveland Cavaliers")

dir(lebron)
```

```
Out[22]: ['_NBAPlayer__team',
          '__class__',
          '__delattr__',
          '__dict__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__gt__',
          '__hash__',
          '__init__',
          '__init_subclass__',
          '__le__',
          '__lt__',
          '__module__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__setattr__',
          '__sizeof__',
          '__str__',
          '__subclasshook__',
          '__weakref__',
          'first_name',
          'last_name',
          'nickname',
          'print_profile']
```

13 Terminology

- Class - the overall structure description (e.g. "NBAPlayer").
- Object / Instance - an instance of the class (e.g. "LeBron").
- Constructors / Destroyers
- "self" - a reference to the class instance.

- Hidden variables and name mangling with "__"
- Abstraction

```
In [ ]: class NBAPlayer:
            """This class represents an NBA player"""
            def __init__(self,
                         first_name,
                         last_name,
                         nickname):
                """Initialize an instance of the player class"""
                self.first_name = first_name
                self.last_name = last_name
                self.nickname = nickname

            def print_name(self):
                """Print the player's profile."""
                print(self.first_name, ' "', self.nickname, '" ', self.last_name, sep="")

            def print_sport(self):
                """Print the sport this player plays in."""
                print("Basketball!")

        lebron = NBAPlayer("LeBron", "James", "King James")

        lebron.print_name()
        lebron.print_sport()
```

```
In [ ]: class Player:
            """This is an abstract class for a sports player."""
            def __init__(self,
                         first_name,
                         last_name,
                         nickname):
                """Initialize an instance of the player class"""
                self.first_name = first_name
                self.last_name = last_name
                self.nickname = nickname

            def print_name(self):
                """Print the player's profile."""
                print(self.first_name, ' "', self.nickname, '" ', self.last_name, sep="")

            def print_sport(self):
                pass
```

```
In [ ]: class NBAPlayer(Player):
            """This class represents an NBA player."""
            def print_sport(self):
```

```
            print("Basketball!")

        lebron = NBAPlayer("LeBron", "James", "King James")

        lebron.print_name()
        lebron.print_sport()
```

```
In [ ]: class NFLPlayer(Player):
            """This class represents an NFL player."""
            def print_sport(self):
                print("Football!")

        nick = NFLPlayer("Nick", "Foles", "Saint Nick")

        nick.print_name()
        nick.print_sport()
```

14 Terminology

- Class - the overall structure description (e.g. "NBAPlayer").
- Object / Instance - an instance of the class (e.g. "LeBron").
- Constructors / Destroyers
- "self" - a reference to the class instance.
- Hidden variables and name mangling with "__"
- Abstraction
- Polymorphism

```
In [ ]: class Player:
            """This is an abstract class for a sports player."""
            def __init__(self,
                         first_name,
                         last_name,
                         nickname,
                         stats,
                        ):
                """Initialize an instance of the player class"""
                self.first_name = first_name
                self.last_name = last_name
                self.nickname = nickname
                self.stats = stats

            def print_name(self):
                """Print the player's profile."""
                print(self.first_name, ' "', self.nickname, '" ', self.last_name, sep="")

            def print_player_profile(self):
                pass
```

```
In [ ]: class NBAPlayer(Player):
            """This class represents an NBA player."""
            def print_sport(self):
                print("Basketball!")

            def print_player_profile(self):
                self.print_name()
                print("PPG:", self.stats["PPG"], "and FT:", self.stats["FT"])

        lebron = NBAPlayer(
                    "LeBron",
                    "James",
                    "King James",
                    {
                        "PPG":27,
                        "FT": 74
                    }
        )

        lebron.print_player_profile()
```

```
In [ ]: class NFLPlayer(Player):
            """This class represents an NFL player."""
            def print_sport(self):
                print("Football!")

            def print_player_profile(self):
                self.print_name()
                print("Completion rate: ", self.stats["completions"] / self.stats["attempts"])

        nick = NFLPlayer(
                    "Nick",
                    "Foles",
                    "Saint Nick",
                    {
                        "completions":27.0,
                        "attempts": 1386
                    }
        )

        nick.print_player_profile()
```

15 Polymorphism in sklearn!

```
model.fit(training_data)
model.predict(testing_data)
```

16 Class vs Objects

```
In [ ]: class NBAPlayer(Player):
            """This class represents an NBA player."""
            league_name = "Basketball"

            def print_sport(self):
                print("Basketball!")
                print(self.league_name)

            def print_player_profile(self):
                self.print_name()
                print("PPG:", self.stats["PPG"], "and FT:", self.stats["FT"])

        NBAPlayer.league_name
```