Data Science Bootcamp

Hyperiondev

Python Objects, Classes, Modules and Reusing Third-Party Software

WELCOME TO THE EVENT HANDLING TASK

Your Lecturer for This Session

Christiaan Joubert

Lecture – Housekeeping

- ❑ The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
- ❑ No question is daft or silly - **ask them!**
- ❑ There are Q/A sessions midway and at the end of the session, should you wish to ask any follow-up questions.
- ❑ You can also submit questions here: hyperiondev.com/sbc4-ds-questions
- ❑ For all non-academic questions, please submit a query: hyperiondev.com/support
- ❑ Report a safeguarding incident: hyperiondev.com/safeguardreporting
- ❑ We would love your feedback on lectures: https://hyperionde.wufoo.com/forms/zsgv4m40ui4i0g/

Lecture – Code Repo

Go to: github.com/HyperionDevBootcamps

Then click on the “**C4_DS_lecture_examples**” repository, do view or download the code.

Objectives

1. Understand the concept of object-oriented programming
2. Learn how to use classes

What is Object-Oriented Programming?

- A form of programming that models real-world interactions of physical objects.
- Relies on classes and objects over functions and logic.
- Powerful tool for abstraction.

Why use OOP?

- Imagine that you want to find the average of a student’s grades.
- While the code to find grades, sum them up and average them is easy, it can sometimes look a bit vague.
- It would be nice to simply have a student.get_average_grades().

Objects in Python

- Without knowing it, you have actually been using objects in Python.
- For example: string.split() - this uses the split() method present in the string object.
- Imagine needing to call split(string, delimiter) - not as powerful of a notation!

OOP Components

- **Class**
  - Different to an object.
  - Think of an object as a house - the class is the blueprint.
- **Properties**
  - Data contained in classes.
  - For example, a student has a name, grade, ID, etc. These are properties of a student.
  - Comes in the form of variables that you can access (e.g. student.name).

Class Properties

- Most often in Python, this comes in the form of a built-in method.
- These can be accessed using the "." e.g. string.upper() - this calls the upper() method present in the string object.
- FUN/USEFUL FACT: You can actually see all of the properties an object using dir().

Class Instantiation

```
my_student = Student("Luke Skywalker", 23, "Male")
```

- Class takes in three values: a name, age and gender.

Creating a Class

- __init__ function is called when class is instantiated.

```
class Student():

    def __init__(self, name, age, gender):
        self.age = age
        self.name = name
        self.gender = gender
```

Creating Methods within a Class

- Within the class, you define a function.
- First parameter is always called self - this references the object itself.
- Let's say you want to average all grades that a student achieved with a single call:

```
def average_grades(self):
    return sum(self.grades) / len(self.grades)
```

Class Variables vs. Instance Variables

- Class variable: static, value will never change.
- Instance variable: assigned at instantiation, can change.

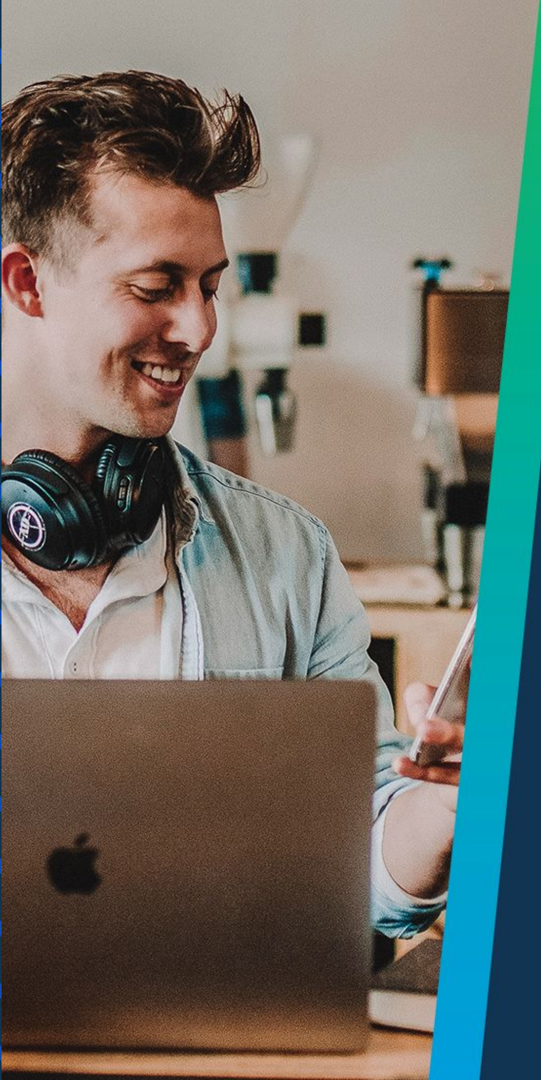
```
class DataScienceStudent:
    bootcamp = "Data Science"
    def __init__(self, name):
        self.name = name

my_ds_student = DataScienceStudent("Me")
print(my_ds_student.bootcamp) # class variable
print(my_ds_student.name) # instance variable
```

Hyperiondev

Q & A Section

Please use this time to ask any questions relating to the topic, should you have any.



Hyperiondev

Thank You for Joining Us