Software Engineering Bootcamp

Hyperiondev

Workshop: Object-Oriented Programming

Workshop - 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: http://hyperiondev.com/sbc4-se-questions
- ❑ For all non-academic questions, please submit a query: www.hyperiondev.com/support
- ❑ Report a safeguarding incident: http://hyperiondev.com/safeguardreporting
- ❑ We would love your feedback on lectures: https://hyperionde.wufoo.com/forms/zsgv4m40ui4i0q/

Github Repository - Workshop Examples

https://github.com/HyperionDevBootcamps/C4_SE_lecture_examples

Objectives

1. What is Object-oriented programming?
2. Understand the concept of object-oriented programming
    a. Classes and their properties
    b. Class instantiation - Objects
    c. Methods within 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.

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, age, grade etc. These are properties of a student.
  - Comes in the form of variables that you can access (e.g. my_student.name).

Creating a Class

- __init__ function is called when class is instantiated.

```
class Student():
    # class variables
    college = "HyperionDev"
    def __init__(self, name, age, grades):
        # instance variables
        self.name = name
        self.age = age
        self.grades = grades
```

static, value will never change.

assigned at instantiation, can change.

Creating an object – Class Instantiation

- Objects are basically initialised versions of your blueprint
- They each have the properties you have defined in your constructor.

```
my_student = Student("Anne", 23, [80, 75, 91])
```

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

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_grade(self):
    return sum(self.grades) / len(self.grades)
```

```
class Student():

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

          def average_grades(self):
                    average = sum(self.grades)/len(self.grades)
                    print(f”The average for student {self.name} is {average}”)
def main():
          # initialise student object
          my_student = Student("Anne", 23, [80, 75, 91])
          # Call the method on the objects
          my_student.average_grades() # prints The average for student Anne is 82
main()
```

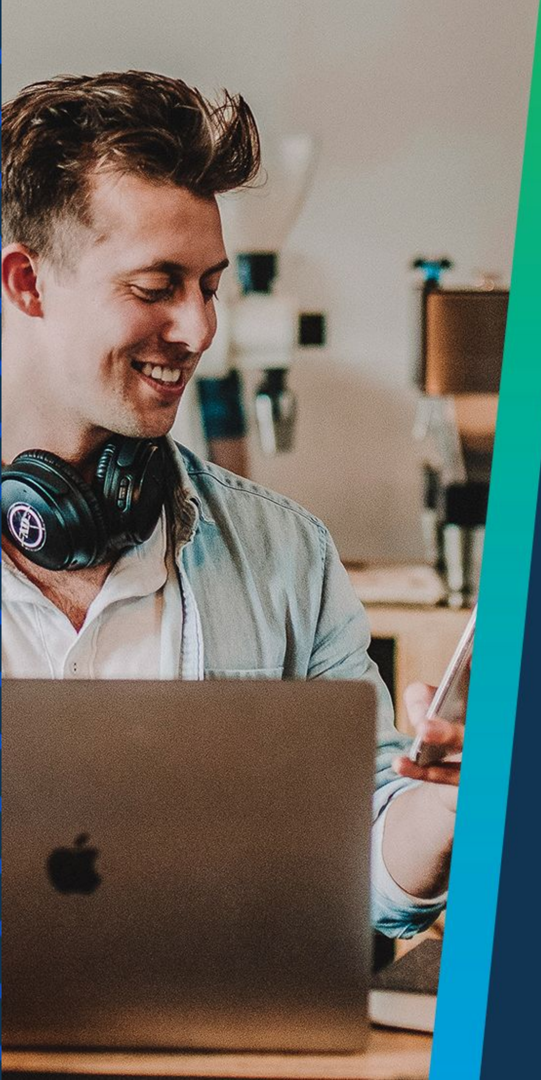
Hyperiondev

Let’s have look at an example in VS code!

Hyperiondev

Q & A Section

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



Hyperiondev

Thank you for joining us

Stay hydrated
Avoid prolonged screen time
Take regular breaks
Have fun :)