Software Engineering Bootcamp

Hyperiondev

Object-Oriented Programming: Classes & Objects

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: 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/zsgv4m40ui4i0g/

Github Repository - Lecture Examples

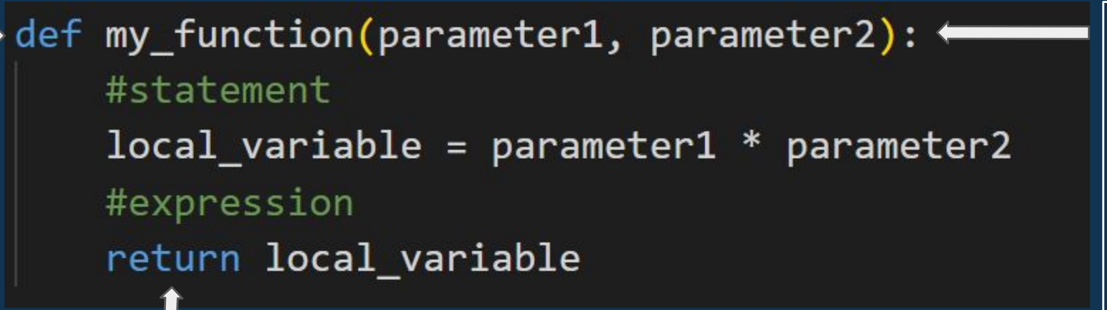
https://github.com/HyperionDevBootcamps/C4_SE_lecture_examples

Recap on Creating Your Own Functions

What is a Function?

- Reusable and Organised block of code.
- The general syntax of a function:

def-tells Python you are defining a function



```
def my_function(parameter1, parameter2):
    #statement
    local_variable = parameter1 * parameter2
    #expression
    return local_variable
```

Parameters can take **required positional input** or **optional keyword** input (default values)

return - if your function returns a value, then use this keyword to return it.

Parameters - The defined input of a function.

Hyperiondev

Calling Functions

- Declare a variable to store the return value
- Give arguments to the parameters of the function

```
answer = my_function(1, 9)
```

Arguments - The values passed to parameters.

- To display the output of the function you need to call print on the variable.

```
# Print the output of the function for the 'answer' instance
print(answer)
```

Objectives

1. What is object-oriented programming and why we use it.
2. Understand the concept of object-oriented programming
   a. Classes and their properties
   b. Class instantiation - Objects
   c. Methods within classes
3. 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 single line of code such as student.get_average_grades().

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).

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!

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().

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
```

Hyperiondev

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("Luke Skywalker", 23, "Male")
```

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

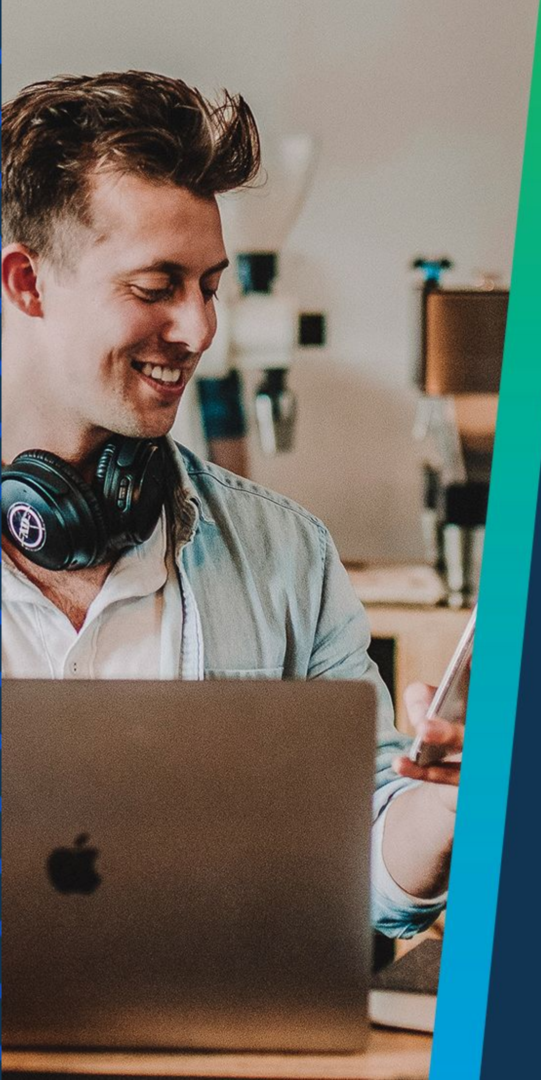
```
class SoftwareEngineeringStudent:
    bootcamp = "Software Engineering"
    def __init__(self, name):
        self.name = name

my_se_student = SoftwareEngineeringStudent("Me")
print(my_se_student.bootcamp) # class variable
print(my_se_student.name) # instance variable
```

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