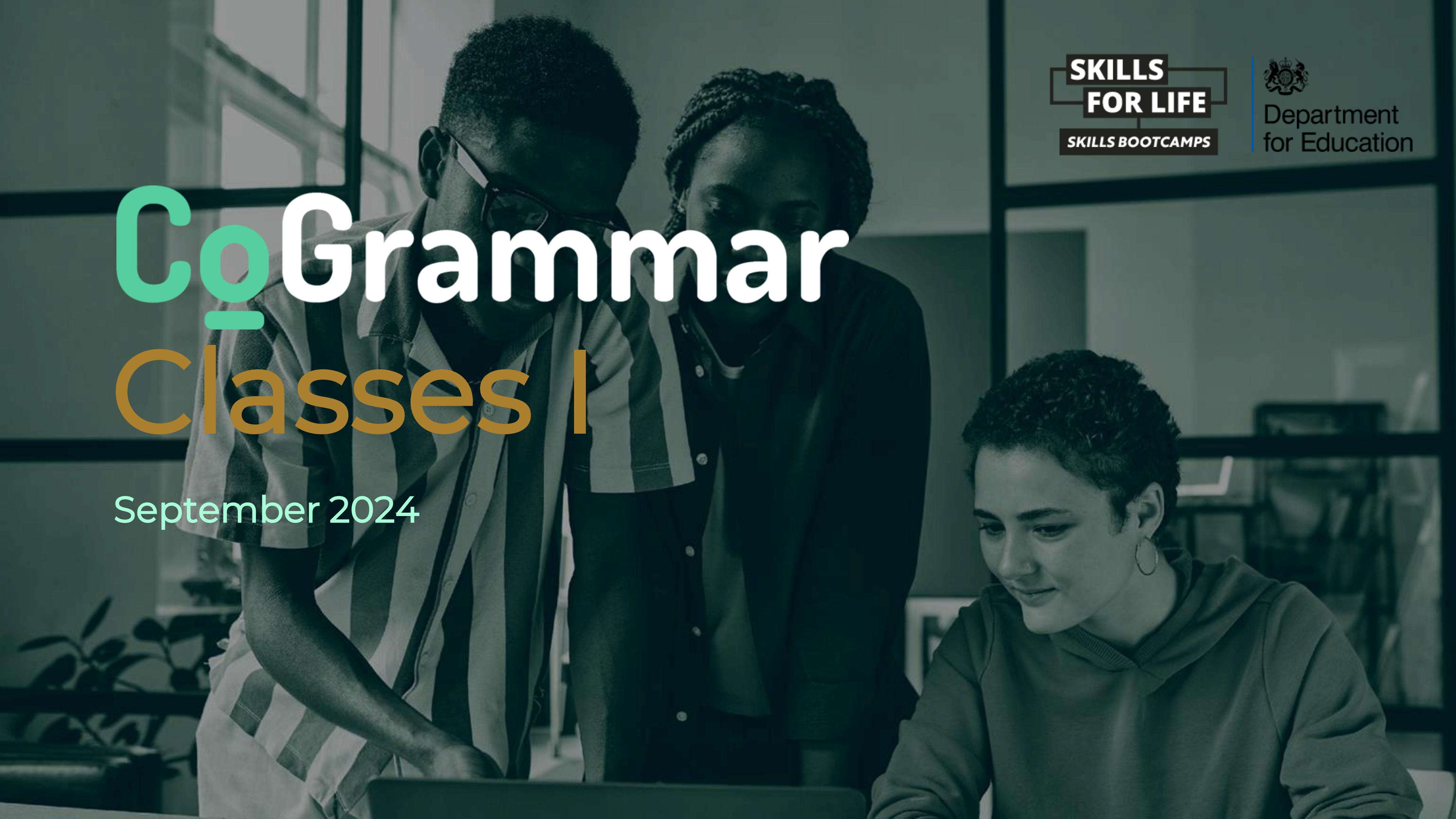
Welcome to this CoGrammar Lecture: Classes I

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

CoGrammar



SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar

Classes I

September 2024

Software Engineering Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- 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. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: Questions

Software Engineering Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support

- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting

- We would love your **feedback** on lectures: Feedback on Lectures

Skills Bootcamp Progression Overview

To be eligible for a certificate of completion, students must fulfil three specific criteria. These criteria ensure a high standard of achievement and alignment with the requirements for the successful completion of a Skills Bootcamp.

✓ Criterion 1 - Meeting Initial Requirements

Criterion 1 involves specific achievements within the first two weeks of the program.
To meet this criterion, students need to:

- Attend a minimum of 7-8 hours per week of guided learning (lectures, workshops, or mentor calls) within the initial two-week period, for a total minimum of 15 guided learning hours (GLH), by no later than 15 September 2024.
- Successfully complete the Initial Assessment by the end of the first 14 days, by no later than 15 September 2024.

CoGrammar

Skills Bootcamp Progression Overview

✓ Criterion 2 - Demonstrating Mid-Course Progress

Criterion 2 involves demonstrating meaningful progress through the successful completion of tasks within the first half of the bootcamp.
To meet this criterion, students should:

- Complete 42 guided learning hours and the first half of the assigned tasks by the end of week 7, no later than 20 October 2024.

CoGrammar

Skills Bootcamp Progression Overview

✓ Criterion 3 - Demonstrating Post-Course Progress

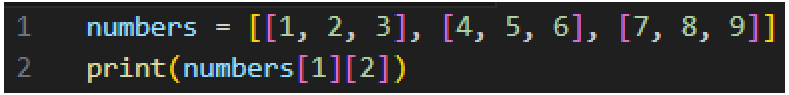
Criterion 3 involves showcasing students’ progress after completing the course.
To meet this criterion, students should:

- Complete all mandatory tasks before the bootcamp's end date. This includes any necessary resubmissions, no later than 22 December 2024.
- Achieve at least 84 guided learning hours by the end of the bootcamp, 22 December 2024.

CoGrammar

Poll

What is the output of the following code?



```python
numbers = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(numbers[1][2])
```

a. 3
b. 9
c. 6

Poll

What is the purpose of the return statement in a function?

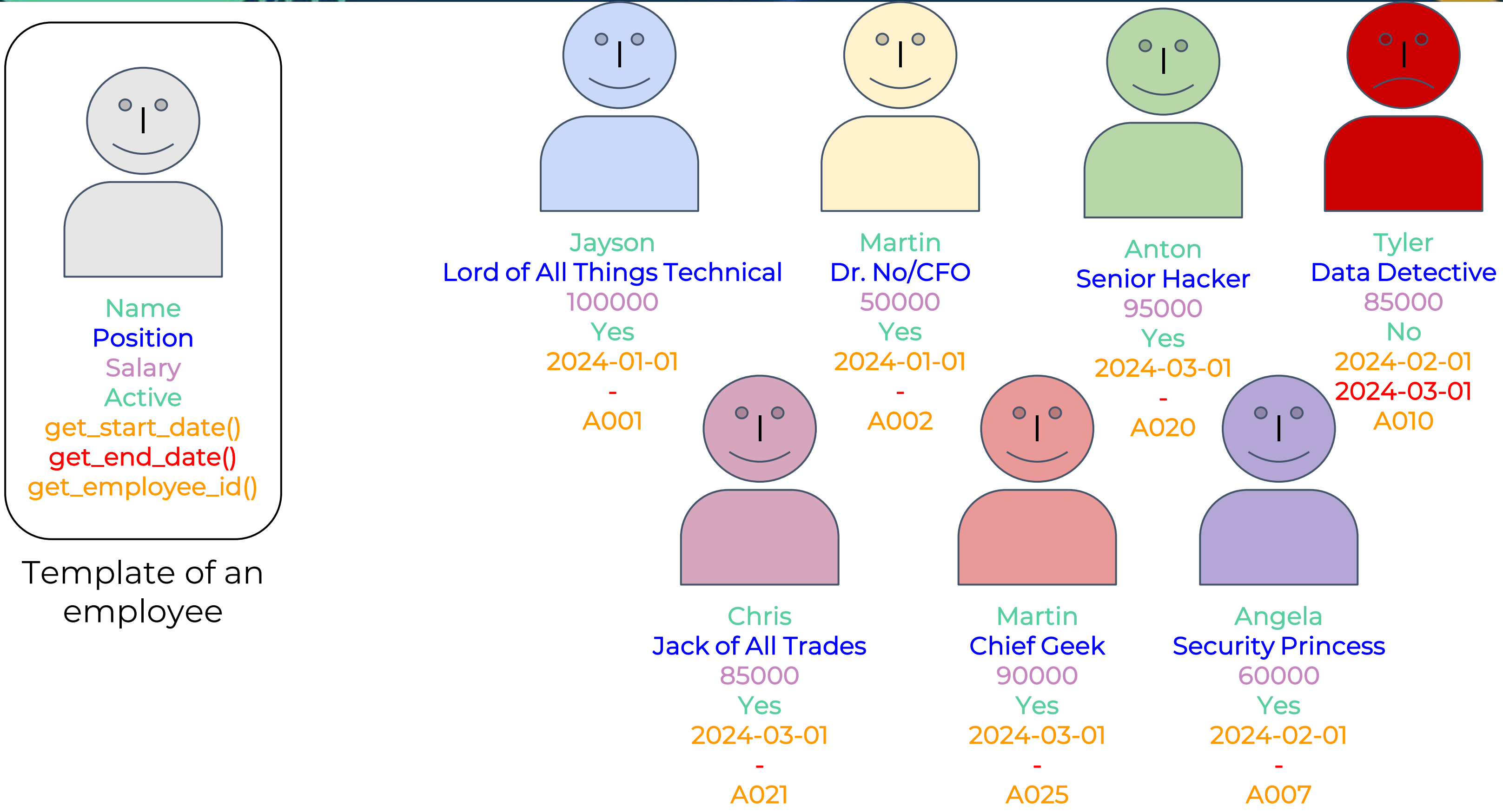
a. To end the execution of the program
b. To send a value back to the caller of the function
c. To print the result of the function

CoGrammar

Learning Objectives & Outcomes

- Explain the Difference Between Procedural and Object-Oriented Programming (OOP) paradigms
- Define a Class with Attributes and Methods
- Implement the __init__ Method for Object Initialization
- Access Object Members Using the Dot Operator
- Implement Encapsulation Using Private Attributes and Public Methods
- Explain the Role of self in Methods

Introduction



Name
Position
Salary
Active
get_start_date()
get_end_date()
get_employee_id()

Template of an employee

Jayson
Lord of All Things Technical
100000
Yes
2024-01-01
-
A001

Martin
Dr. No/CFO
50000
Yes
2024-01-01
-
A002

Anton
Senior Hacker
95000
Yes
2024-03-01
-
A020

Tyler
Data Detective
85000
No
2024-02-01
2024-03-01
A010

Chris
Jack of All Trades
85000
Yes
2024-03-01
-
A021

Martin
Chief Geek
90000
Yes
2024-03-01
-
A025

Angela
Security Princess
60000
Yes
2024-02-01
-
A007

Building Blocks of Object-Oriented Programming: Classes

CoGrammar

What is Object-Oriented Programming?

- Definition of OOP:
  - A programming paradigm based on the concept of "objects"
  - Objects contain attributes (data) and methods(behaviours)
  - OOP organizes software design around data, or objects, rather than functions and logic
- Key Principles of OOP:
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Abstraction

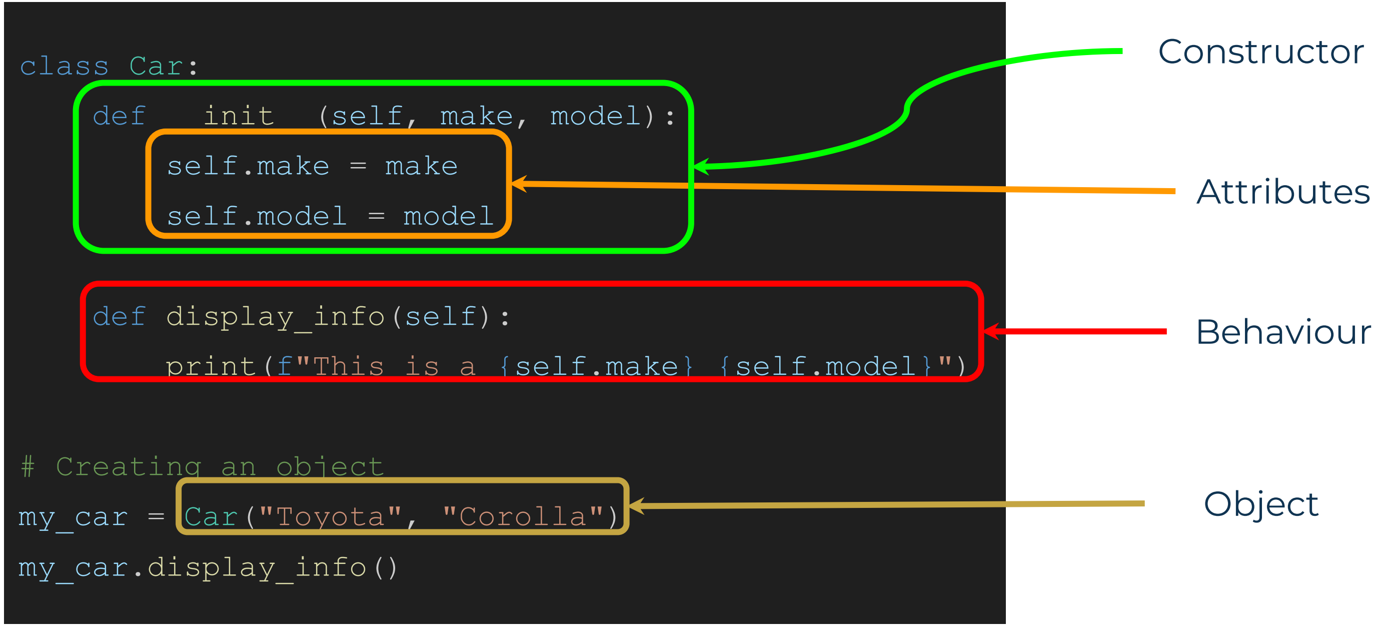
The Four Pillars of Object-Oriented Programming

- Encapsulation
  - Bundling data and methods that operate on that data
  - Hiding internal details and providing a public interface
- Inheritance
  - Creating new classes based on existing classes
  - Promotes code reuse and establishes a hierarchy
- Polymorphism
  - Objects of different classes can be treated as objects of a common base class
  - Allows for flexible and extensible code
- Abstraction
  - Simplifying complex systems by modeling classes based on real-world entities (methods or processes)
  - Focusing on essential features while hiding unnecessary details

Building Blocks of OOP: Classes

- Definition of Class:
  - Blueprint for creating objects
  - Defines attributes (data) and methods (behaviors)
  - Attributes are the properties or characteristics of an object
  - Methods are the functions or operations that an object can perform

Defining a Class in Python



```
class Car:
    def   init  (self, make, model):
        self.make = make
        self.model = model

    def display_info(self):
        print(f"This is a {self.make} {self.model}")


# Creating an object
my_car = Car("Toyota", "Corolla")
my_car.display_info()
```

Constructor

Attributes

Behaviour

Object

CoGrammar

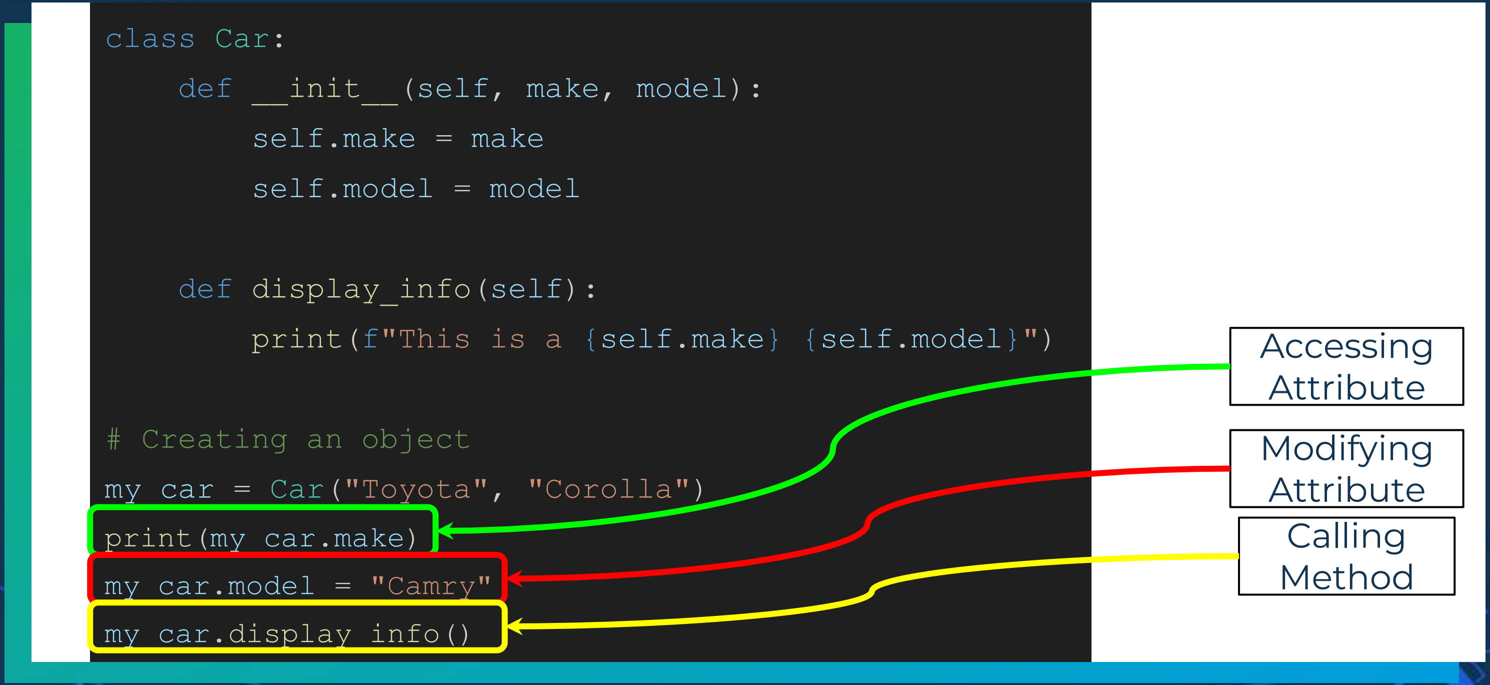
Interacting with Objects: The Dot Operator

CoGrammar

Using the Dot Operator

- Accessing attributes: `object.attribute`
- Calling methods: `object.method()`
- Modifying attributes: `object.attribute = new_value`

Dot Operator Examples



```
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def display_info(self):
        print(f"This is a {self.make} {self.model}")

# Creating an object
my_car = Car("Toyota", "Corolla")
print(my_car.make)
my_car.model = "Camry"
my_car.display_info()
```

Accessing Attribute

Modifying Attribute

Calling Method

Encapsulating Data: A Guide to Private Attributes

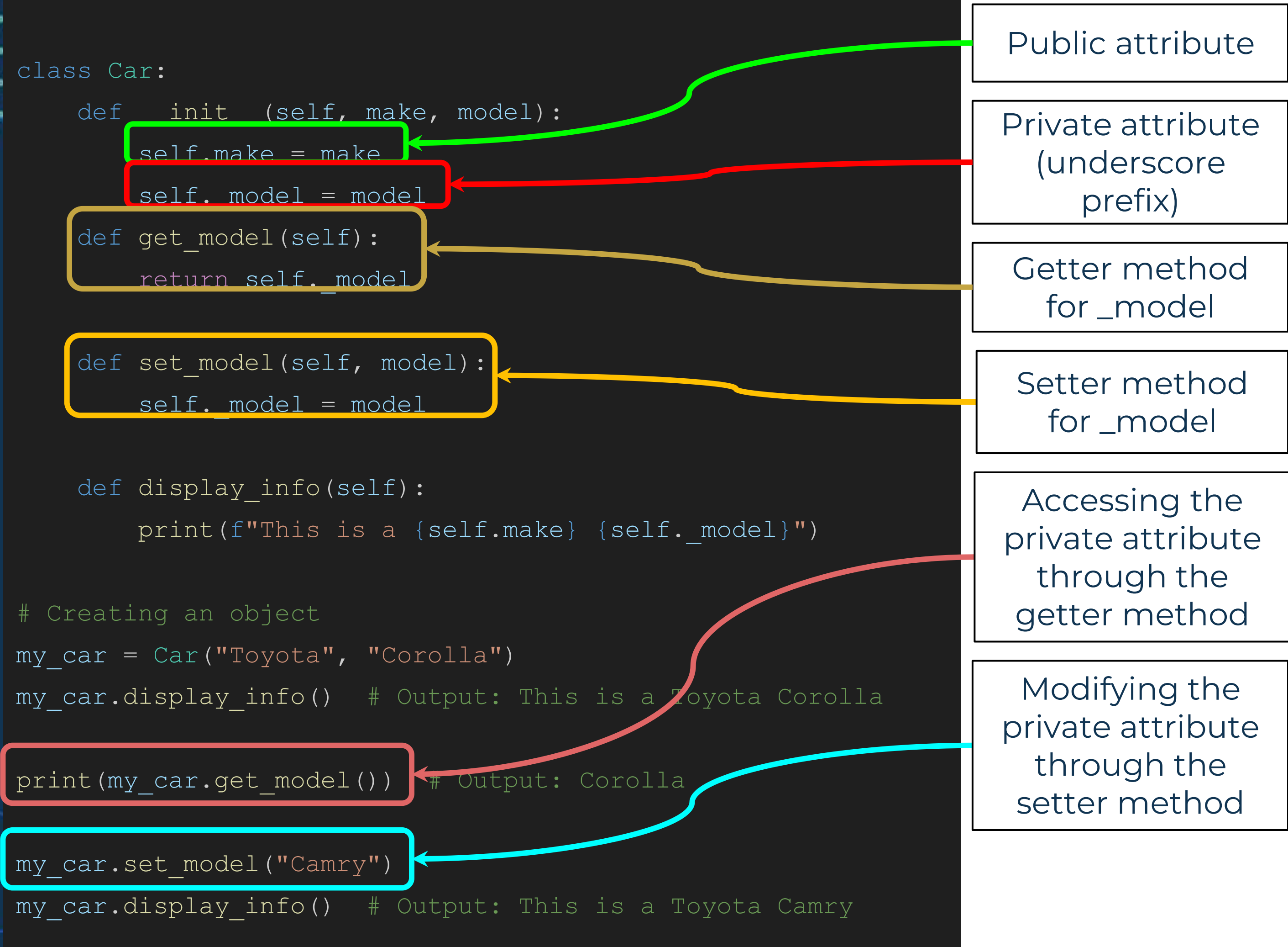
CoGrammar

Understanding Encapsulation

- What is Encapsulation?:
  - Bundling of data and methods that operate on that data
  - Restricting direct access to some of an object's components
- Why is it important?:
  - Data protection
  - Flexibility to change implementation

CoGrammar

Implementing Private Attributes in Python



```python
class Car:
    def __init__(self, make, model):
        self.make = make
        self._model = model
    def get_model(self):
        return self._model

    def set_model(self, model):
        self._model = model

    def display_info(self):
        print(f"This is a {self.make} {self._model}")

# Creating an object
my_car = Car("Toyota", "Corolla")
my_car.display_info()  # Output: This is a Toyota Corolla

print(my_car.get_model())  # Output: Corolla

my_car.set_model("Camry")
my_car.display_info()  # Output: This is a Toyota Camry
```

Public attribute

Private attribute (underscore prefix)

Getter method for _model

Setter method for _model

Accessing the private attribute through the getter method

Modifying the private attribute through the setter method

CoGrammar

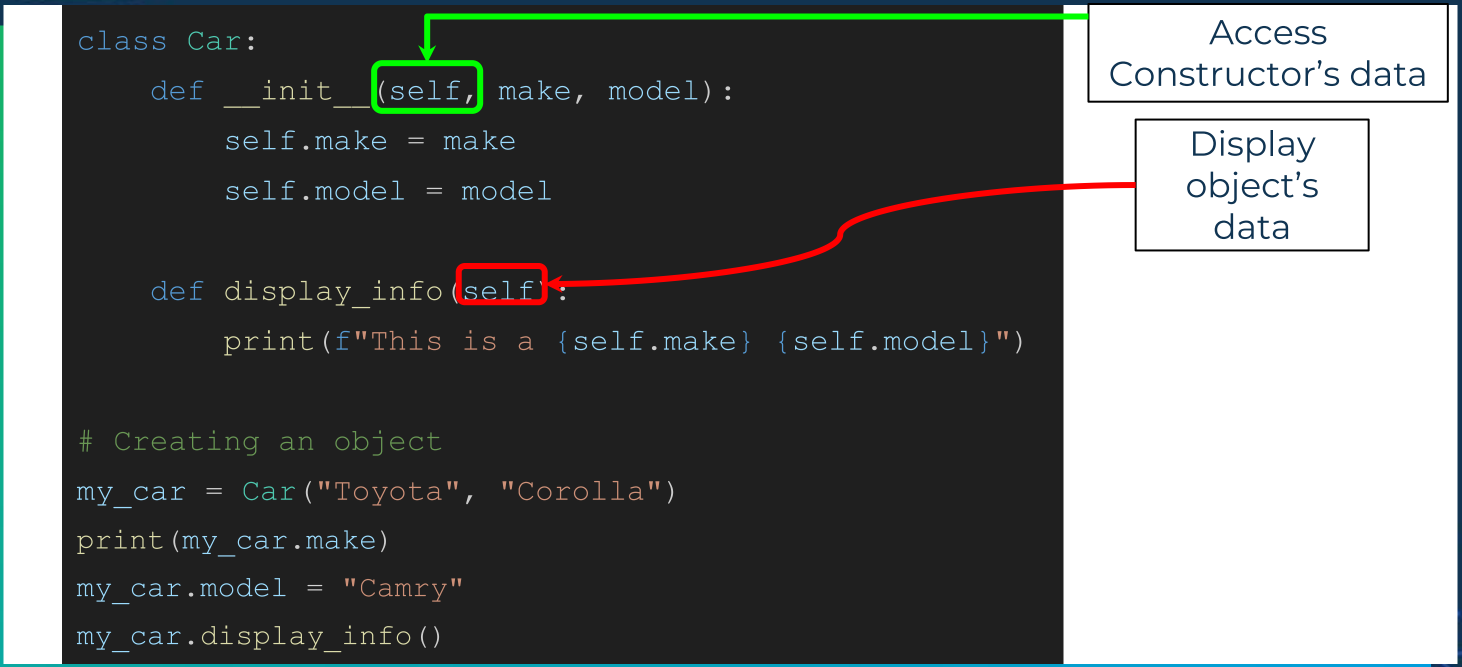
The Importance of self in Python Classes

CoGrammar

The self Keyword: A Closer Look

- What is `self`?
  - Reference to the instance of the class
  - First parameter in method definitions on class
- Why is it important?:
  - Allows access to instance attributes and methods
  - Distinguishes instance variables from local variables

Dot Operator Examples



```python
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def display_info(self):
        print(f"This is a {self.make} {self.model}")

# Creating an object
my_car = Car("Toyota", "Corolla")
print(my_car.make)
my_car.model = "Camry"
my_car.display_info()
```

Access Constructor's data

Display object's data

The Concept of Objects in OOP

CoGrammar

Objects: The Foundation of Python Programming

- What is an Object?
  - Instance of a class
  - Combination of data (attributes) and behavior (methods)
  - Representation of real-world entities in code
- Why is it important?:
  - Organize and structure code
  - Create reusable and modular code
  - Model real-world systems intuitively
  - Everything in Python is an Object

Key Aspects of Objects

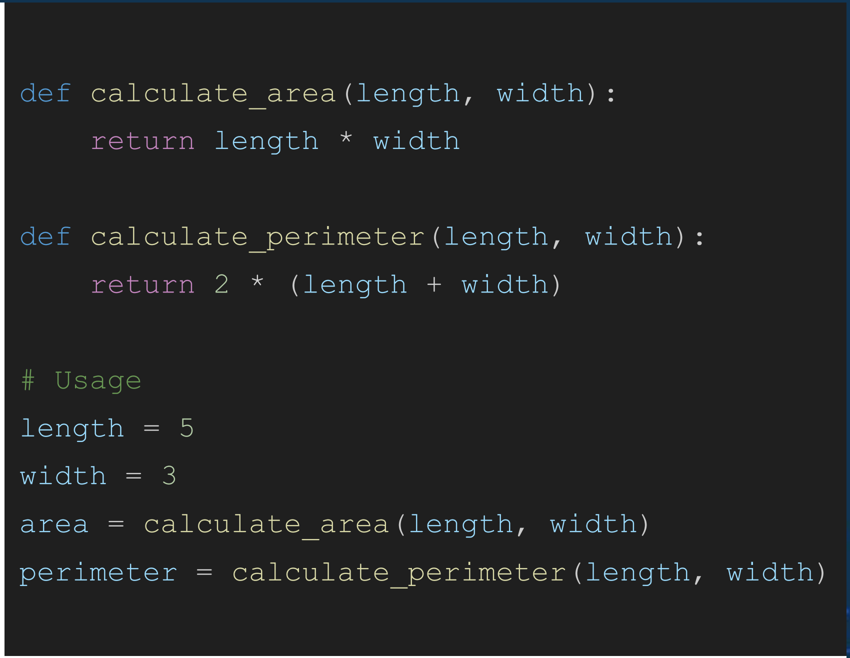
- Aspects
  - State: Data stored in the object (attributes)
  - Behavior: What the object can do (methods)
  - Identity: Each object is unique
  - Lifecycle: Objects are created, used, and destroyed
- Real-world Analogy: Car
  - Attributes: color, make, model, current speed
  - Methods: accelerate, brake, turn

Procedural vs Object-Oriented Programming

CoGrammar

Procedural Programming

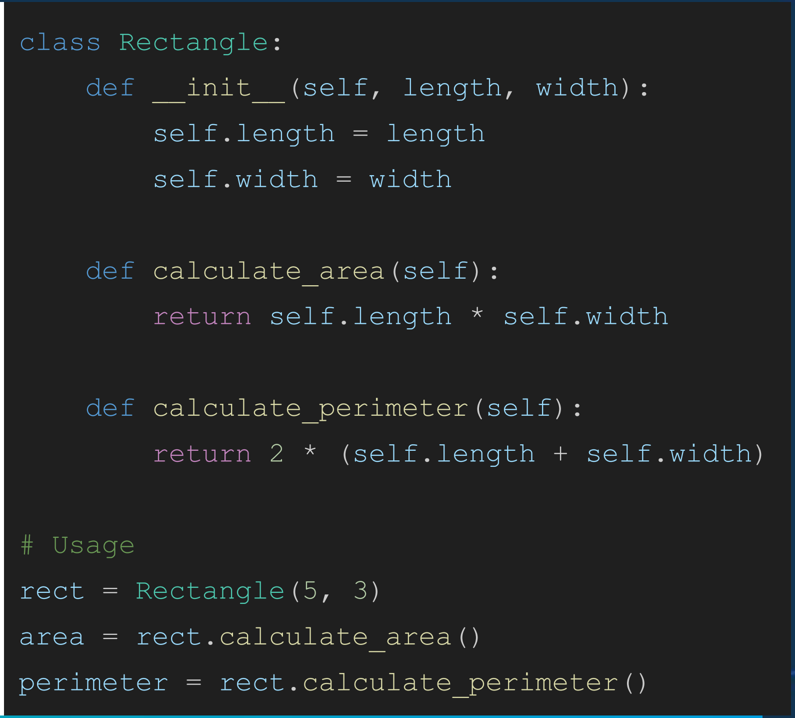
- Characteristics
    - Sequential execution of instructions
    - Functions operating on data
- Example:



```python
def calculate_area(length, width):
    return length * width

def calculate_perimeter(length, width):
    return 2 * (length + width)

# Usage
length = 5
width = 3
area = calculate_area(length, width)
perimeter = calculate_perimeter(length, width)
```

Object Oriented Programming

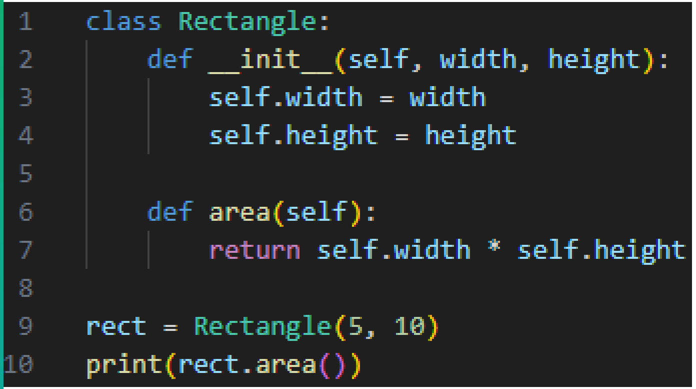
- Characteristics
  - Objects as combinations of data and behavior
  - Classes as blueprints for objects
- Advantages
  - Modularity
  - Reusability
  - Easier maintenance
- Example:



```python
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_area(self):
        return self.length * self.width

    def calculate_perimeter(self):
        return 2 * (self.length + self.width)

# Usage
rect = Rectangle(5, 3)
area = rect.calculate_area()
perimeter = rect.calculate_perimeter()
```

Poll

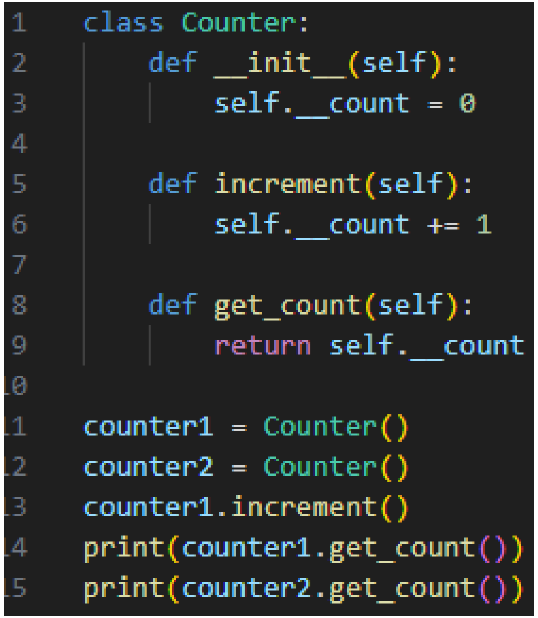
Which one of the following is true?



```python
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

rect = Rectangle(5, 10)
print(rect.area())
```

a. area is an attribute
b. self.width is an attribute
c. self.width is an behaviour
d. area is an behaviour

Poll



```python
class Counter:
    def __init__(self):
        self.__count = 0

    def increment(self):
        self.__count += 1

    def get_count(self):
        return self.__count

counter1 = Counter()
counter2 = Counter()
counter1.increment()
print(counter1.get_count())
print(counter2.get_count())
```

Given the following code, what will be the output of the print statements?

a. 0 0
b. 1 0
c. 1 1

Lesson Conclusion and Recap

Recap the key concepts and techniques covered during the lesson.

- Procedural vs. OOP: Procedural: Focuses on functions and sequential instructions, and OOP uses classes and objects to model real-world entities, encapsulating data and behaviour
- Understanding Objects: Objects are instances of classes with attributes (data) and methods (behaviour).They Help in modelling complex systems and promote modular code.
- Classes and Attributes: Classes: Blueprints for creating objects, Attributes: Data stored in objects (e.g., `name`, `age`), Methods: Functions that define object behaviour (e.g., `bark()`, `deposit()`).
- `__init__` Method and Dot Operator: Initialises object attributes, Dot Operator: Accesses and modifies object attributes and methods (e.g., `my_dog.name`, `my_dog.bark()`).
- Encapsulation and **`self`:** **Encapsulation: Private Attributes:** Hidden data accessed via methods. **Public Methods:** Provide controlled access to private data. **`self`:** Refers to the current instance, used to access attributes and methods.

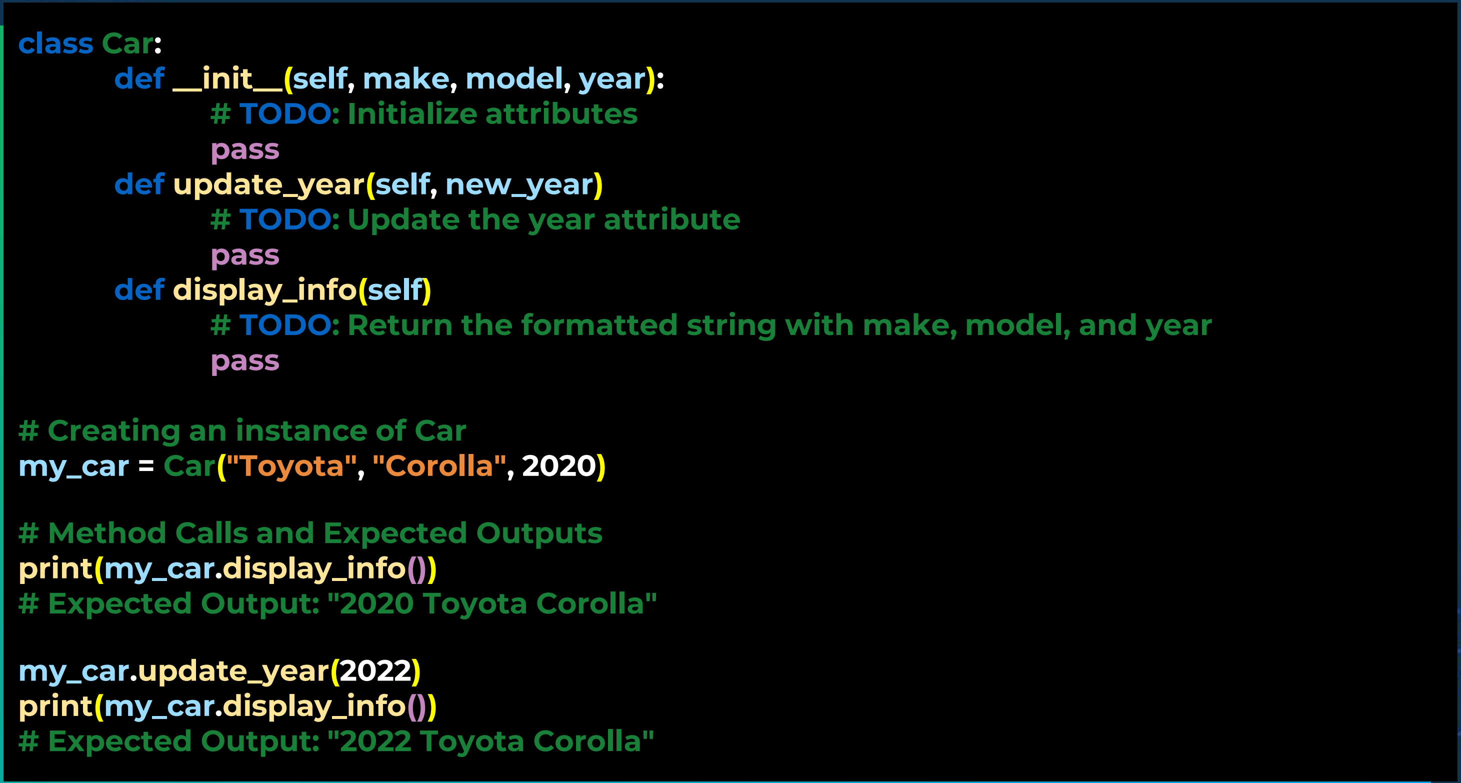
CoGrammar

Follow-up Activity

Task Overview:

- Define a `Car` class with attributes `make`, `model`, and `year`.
- Implement methods to update and display these attributes.

Follow-up Activity



```
class Car:
    def __init__(self, make, model, year):
        # TODO: Initialize attributes
        pass
    def update_year(self, new_year)
        # TODO: Update the year attribute
        pass
    def display_info(self)
        # TODO: Return the formatted string with make, model, and year
        pass

# Creating an instance of Car
my_car = Car("Toyota", "Corolla", 2020)

# Method Calls and Expected Outputs
print(my_car.display_info())
# Expected Output: "2020 Toyota Corolla"

my_car.update_year(2022)
print(my_car.display_info())
# Expected Output: "2022 Toyota Corolla"
```

Follow-up Activity

1. **Submission**: Just make sure that you have the output provided above. This is not tied to any of your tasks.
2. Use any method available to you. As long as you understand the process.

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar