Welcome to this CoGrammar Q&A:

OOP Revision

The session will start shortly...

Questions? Drop them in the chat.

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** throughout this session, should you wish to ask any follow-up questions.
- 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

Enhancing Accessibility: Activate Browser Captions

Why Enable Browser Captions?

- Captions provide **real-time text for spoken content**, ensuring inclusivity.
- Ideal for individuals in noisy or quiet environments or for those with **hearing impairments**.

How to Activate Captions:

1. **YouTube or Video Players:**
   - Look for the CC (Closed Captions) icon and click to enable.
2. **Browser Settings:**
   - Google Chrome: Go to *Settings > Accessibility > Live Captions* and toggle ON.
   - Edge: Enable captions in *Settings > Accessibility*.

Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:

Ian Wyles
Designated Safeguarding Lead

Simone Botes

Nurhaan Snyman

Rafiq Manan

Ronald Munodawafa

Tevin Pitts

Scan to report a safeguarding concern

or email the Designated Safeguarding Lead:
Ian Wyles
safeguarding@hyperiondev.com

CoGrammar HyperionDev

Polls

CoGrammar

Poll

1. **Which of the following correctly describes the key difference between @classmethod and @staticmethod?**
   a) @classmethod modifies instance attributes, while @staticmethod cannot modify any attributes
   b) @classmethod receives the class (cls) as the first argument, while @staticmethod does not
   c) @staticmethod can only be called inside the class, while @classmethod can be called externally
   d) @staticmethod is always inherited by child classes, while @classmethod is not

Poll

2. **Which of the following is NOT a valid purpose of a special method?**

   a) Allowing an object to be called like a function

   b) Enabling mathematical operations like addition (+) for custom objects

   c) Automatically running the method after every attribute change

   d) Defining how an object is represented as a string

Poll

3. **Which of the following is true about Abstract Base Classes (ABC) in Python?**
   a) ABC classes can be instantiated directly
   b) All subclasses of an ABC must implement the abstract methods
   c) ABC classes don’t allow inheritance
   d) ABC classes automatically define all methods without the need for child classes to implement them

Poll

4. **Which of the following statements is true regarding method overriding in Python?**

   a) The parent class method can never be accessed once it's overridden

   b) You can call the parent class method from the subclass using super()

   c) Method overriding is only allowed when using multiple inheritance

   d) You must explicitly use the override() keyword to override methods

Learning Outcomes

- Identify the Components of a Class
- Demonstrate the Use of Inheritance
- Remember the purpose of special methods in Python.
- Apply special methods to create well-structured Python classes.
- Create a Python class that implements at least three special methods.

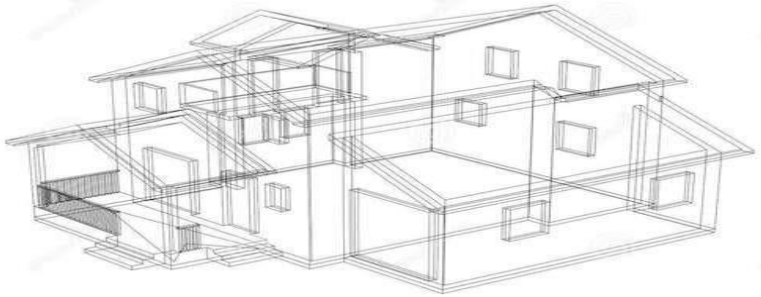
Learning Outcomes

- Describe and utilise polymorphism with the use of
  - Operator Overloading
  - Method Overriding
  - Method Overloading and
  - Duck Typing.


SKILLS
FOR LIFE
SKILLS BOOTCAMPS
Department
for Education
CoGrammar
Classes

Classes and Objects

- A class is a blueprint or template for creating objects. It defines the attributes and methods that all objects of that class will have.



- An object is an instance of a class. Objects are created based on the structure defined by the class.

Attributes

- Attributes are values that define the characteristics associated with an object.
- They define the state of an object and provide information about its current condition.

Methods (Behaviours)

- Methods, also known as behaviours, define the actions or behaviours that objects can perform.
- They encapsulate the functionality of objects and allow them to interact with each other and the outside world.

Constructor

- A constructor is a special method that gets called when an object is instantiated. It is used to initialise the object's attributes.

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

Creating a Class

- __init__ ( ) method is called when the class is instantiated.

```
class Student:

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

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

Class Instantiation

- When you instantiate a class, you create an instance or an object of that class.

```
luke = Student("Luke Skywalker", 23, True)
```

Creating and Calling Methods

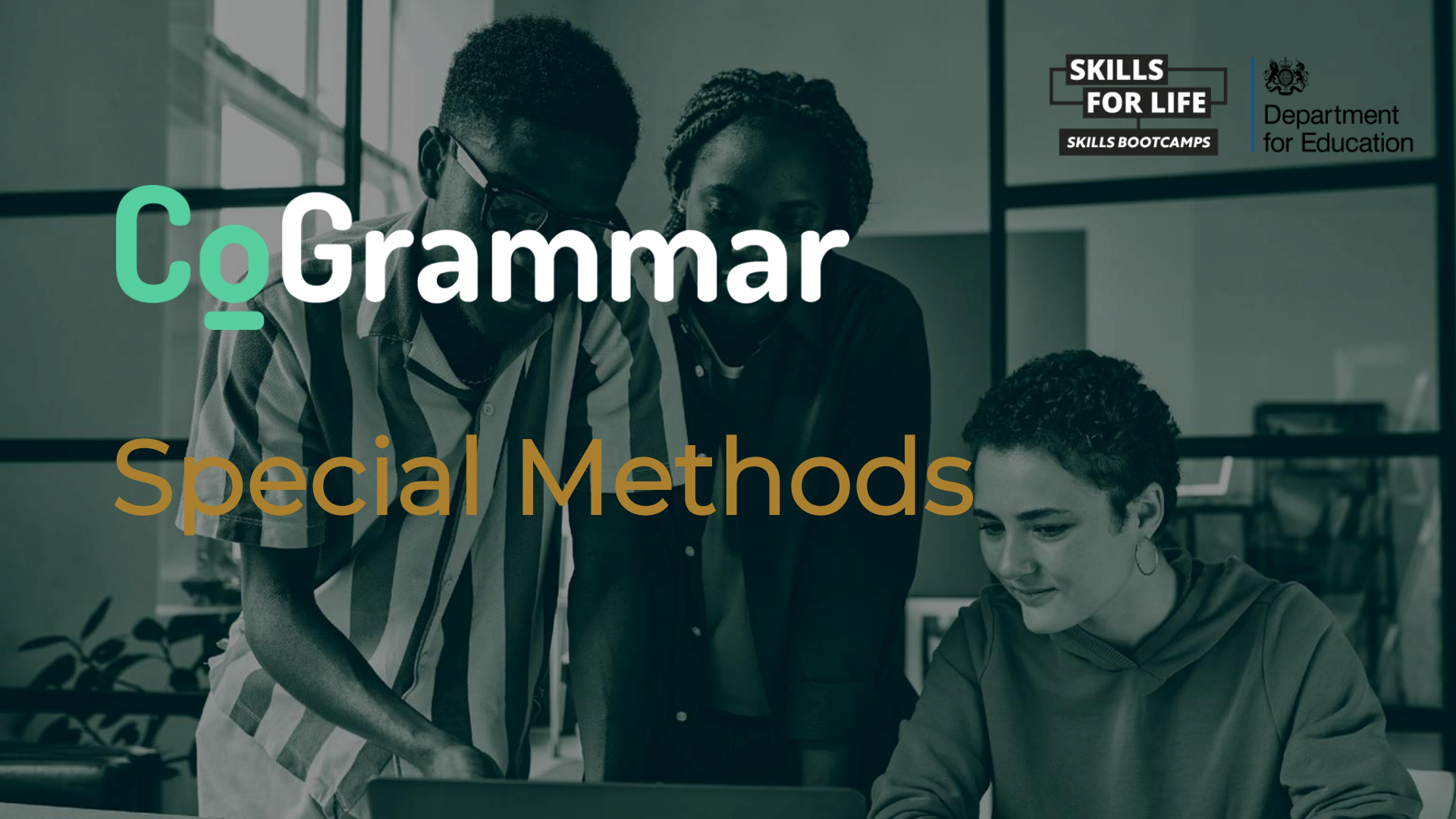
- change_location( ) method is called below:

```
class House:

    def __init__(self, location):
        self.location = location

    def change_location(self, new_location):
        self.location = new_location

house = House("London")
house.change_location("Manchester")
```


SKILLS
FOR LIFE
SKILLS BOOTCAMPS
Department
for Education
CoGrammar
Special Methods

What are Special Methods?

- Special methods in Python are predefined methods that allow developers to define how objects of a class should behave in certain situations.

Objects as Strings

CoGrammar

Objects As Strings

- You have probably noticed when using print() that some objects are represented differently than others.
- Some dictionaries and list have {} and [] in the representation and when we print an object we get a memory address <__main__.Person object at 0x000001EBCA11E650>
- We can set the string representations for our objects to whatever we like using either __repr__() or __str__()

__repr__()

- This method returns a string for an official representation of the object.
- __repr__() is usually used to build a representation that can assist developers when working with the class.
- This representation will contain extra information in the method about the object that is not meant for the user.

__repr__()

```python
class Student:
    def __init__(self, full_name, student_number):
        self.full_name = full_name
        self.student_number = student_number

    def __repr__(self):
        # Including memory address and internal state, useful for debugging
        return (f"<Student(name={self.full_name!r}, "
                f"S_Number={self.student_number!r}, id={hex(id(self))})>")


new_student = Student("Percy Jackson", "PJ323423")

print(new_student)
# Output: <Student(name='Percy Jackson', S_Number='PJ323423', id=0xc303747f50)>
```

__str__()

- This method returns a user-friendly representation for your object when the str() function is called.
- When your object is used in the print function it will automatically try to cast your object to a string and will then receive the representation returned by __str__()

__str__()

```python
class Student:

    def __init__(self, full_name, student_number):
        self.full_name = full_name
        self.student_number = student_number

    def __str__(self):
        return (f"Full Name:\t{self.full_name}\n"
                f"Student Num: \t{self.student_number}")


new_student = Student("Percy Jackson", "PJ323423")

print(new_student)
# Output:   Full Name:      Percy Jackson
#           Student Num:    PJ323423
```

Container-Like Objects

CoGrammar

Container-Like Objects

- A container-like object is any object that can hold or store other objects.
- Using special methods we can also incorporate the behaviour that we see in container-like objects.
- E.g. When we try to get an item from a list the special method __getitem__(self,key) is called. We can then override the default behaviour of the method to return the result we desire.
- For this example, object[0] will call object.__getitem__(self.key) where key = 0

Implementing Container-Like Behaviour

```python
class ContactList:

    def __init__(self):
        self.contact_list = []

    def add_contact(self, contact):
        self.contact_list.append(contact)

    def __getitem__(self, key):
        return self.contact_list[key]


contact_list = ContactList()
contact_list.add_contact("Test Contact")
print(contact_list[0]) # Output: Test Contact
```

Container-Like Objects: Special Methods

- Some special methods for container-like objects are:
  - len(object) -> __len__(self)
  - object[key]-> __getitem__(self, key)
  - object[key] = item -> __setitem__(self, key, item)
  - item in object -> __contains__(self, item)
  - variable = object(parameter) -> __call__(self, parameter)
  - iter(object) or ‘for item in object’ -> __iter__(self)
  - next(iterator) -> __next__(self)

Comparators

CoGrammar

Comparators

- We will use these methods to set the behaviour when we try to compare our objects to determine which one is smaller or larger or are they equal.
- E.g. When trying to see if object x is greater than object y. The method x__gt__(y) will be called to determine the result. We can then set the behaviour of __gt__() inside our class.
- x > y -> x.__gt__(y)

Comparators

```python
class Student:

    def __init__(self, fullname, student_number, average):
        self.fullname = fullname
        self.student_number = student_number
        self.average = average

    def __gt__(self, other):
        return self.average > other.average

student1 = Student("Peter Parker", "PP734624", 88)
student2 = Student("Tony Stark", "TS23425", 85)
print(student1 > student2) # Output: True
```

Other Comparators

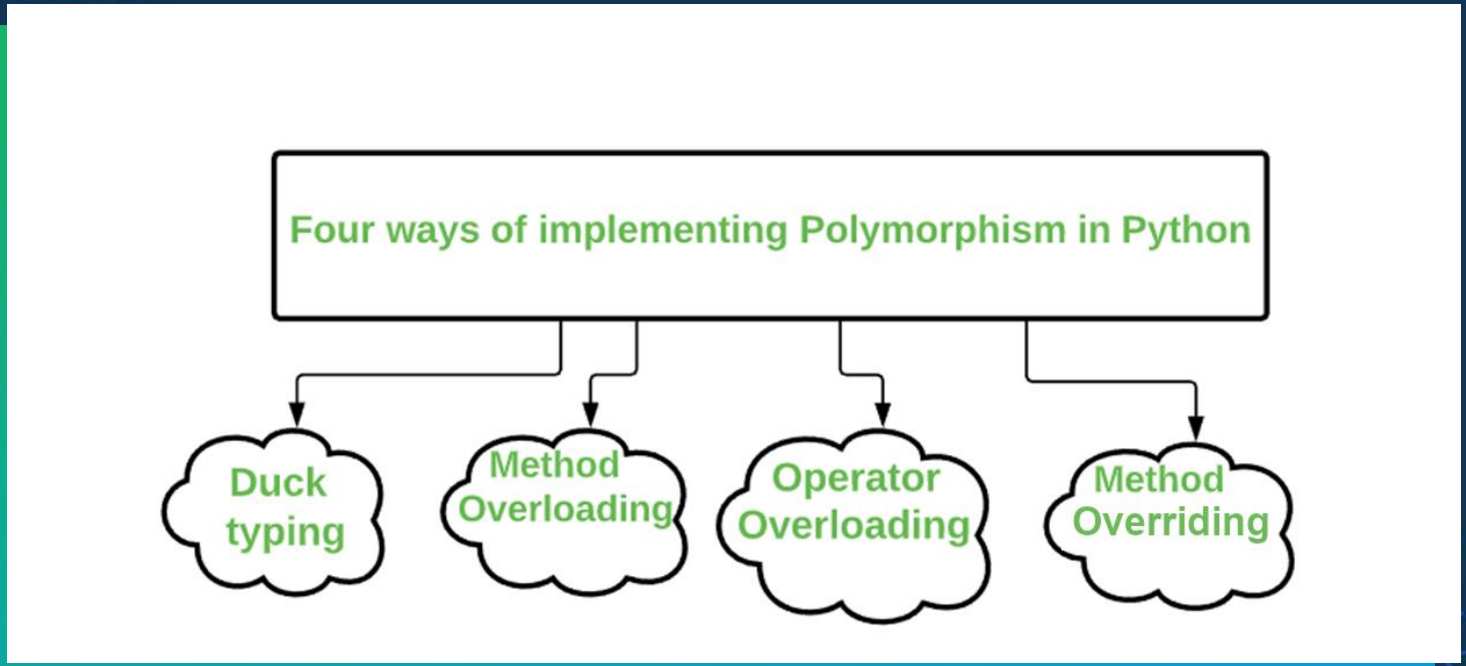
- Commonly Used Special Methods for Comparison:
  - __eq__(self, other): Behaviour for equality (==)
  - __ne__(self, other): Behaviour for inequality (!=)
  - __lt__(self, other): Behaviour for less-than (<)
  - __le__(self, other): Behaviour for less-than-or-equal (<=)
  - __gt__(self, other): Behaviour for greater-than (>)
  - __ge__(self, other): Behaviour for greater-than-or-equal (>=)


SKILLS FOR LIFE
SKILLS BOOTCAMPS
Department for Education
CoGrammar
Polymorphism

What is Polymorphism?

- Polymorphism refers to the ability of different objects to respond to the same message or method call in different ways.
- This allows objects of different classes to be treated as objects of a common superclass.

Implementing Polymorphism



Operator Overloading

CoGrammar

Poly: Operator Overloading

- Special methods allow us to set the behaviour for mathematical operations such as +, -, *, /, **
- Using these methods we can determine how the operators will be applied to our objects.
- x + y -> x.__add__(y)
- E.g. When trying to add two of your objects, x and y, together python will try to invoke the __add__() special method that sits inside your object x. The code inside __add__() will then determine how your objects will be added together and returned.

Operators for Overloading

- Commonly Used Special Methods for Operator Overloading:
  - __add__(self, other): Behaviour for the (+) operator.
  - __sub__(self, other): Behaviour for the (-) operator.
  - __mul__(self, other): Behaviour for the (*) operator.
  - __pow__(self, other): Behaviour for the (**) operator.
  - __truediv__(self, other): Behaviour for the (/) operator.
  - __eq__(self, other): Behaviour for the (==) operator.

Special Methods And Math

```python
class MyNumber:

    def __init__(self, value):
        self.value = value

    def __add__(self, other):
        return MyNumber(self.value + other.value)

num1 = MyNumber(10)
num2 = MyNumber(5)
num3 = num1 + num2
print(num3.value) # Output: 15
```

Method Overriding

CoGrammar

Poly: Method Overriding

- We can override methods in our subclass to either extend or change the behaviour of a method in the base class.
- To apply method overriding you simply need to define a method with the same name as the method you would like to override.
- To extend functionality of a method instead of completely overriding we can use the super() function.

Method Overloading

CoGrammar

Poly: Method Overloading

- In Python, method overloading can be achieved by using default values for function parameters as one possible option.
- You can also use the *args and **kwargs concept to receive a varying parameter list.

Duck Typing

CoGrammar

Duck Typing

- Duck typing is where the type or class of an object is less important than the methods or properties it possesses.
- The term "duck typing" comes from the saying, "If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.“

```python
class Dog:
    def speak(self):
        return "Woof!"

# Function that expects an object with a speak method
def make_sound(animal):
    return animal.speak()

# Using duck typing
dog = Dog()

print(make_sound(dog))  # Outputs: Woof!
```

Let’s take a short break

CoGrammar

Demo Time!

CoGrammar

Conclusion and Recap

CoGrammar

Lesson Recap

- Why OOP is Essential in Programming
- Implementing a Class
- Demonstration of Inheritance and Polymorphism
- Special Methods
  - Implement special behaviours into our classes to allow them to interact with built-in python methods.
- Polymorphism
  - An idea where different objects can respond to the same message or method call in different ways.

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar