CoGrammar

SE PORTFOLIO SESSION 11

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

Software Engineering Lecture Housekeeping

- The use of disrespectful language is prohibited if asking a question. This is a supportive, learning environment for all – please engage accordingly! **(FBV: Mutual Respect.)**
- No question is ‘silly’ – **ask away!**
- 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 Open Classes. You can submit these questions here: Open Class Questions

Software Engineering Lecture 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]()

In Python, what happens when a method is defined in both the base class and derived classes with the same name?

A. The method in the derived class overrides the one in the base class
B. An error is raised, and the code fails to run
C. The method in the base class takes precedence
D. Python randomly selects which method to call at runtime

How can you check whether an object is an instance of a particular class or its subclasses in Python?

A. Using the isinstance() function
B. Using the typeof() keyword
C. Using the classof() method
D. Using the check_class() function

Recap of Week 11: OOP

Classes

- We can create classes in python using the ‘class’ keyword

Inheritance

- We can inherit attributes and behaviour from classes that we have already made to add and extend to their functionality.

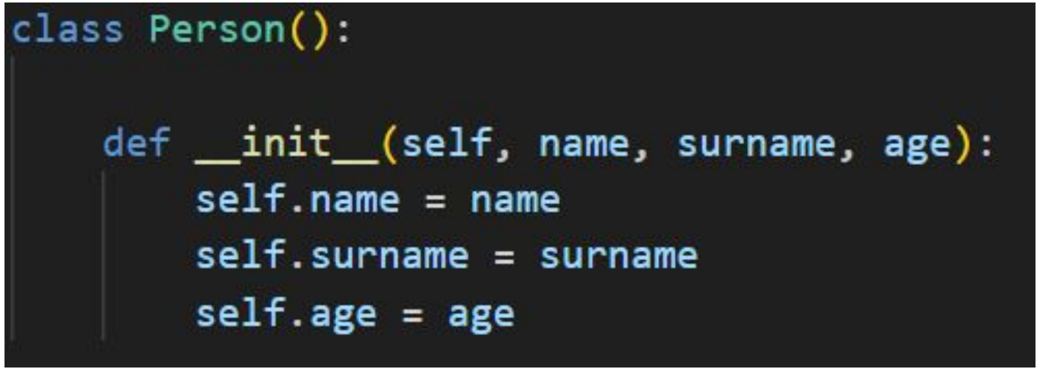
Method Overriding

- We can override methods from the base class to change the functionality of the method to match its class.

CoGrammar

Recap of Week 11: OOP

Defining a Class for Inheritance

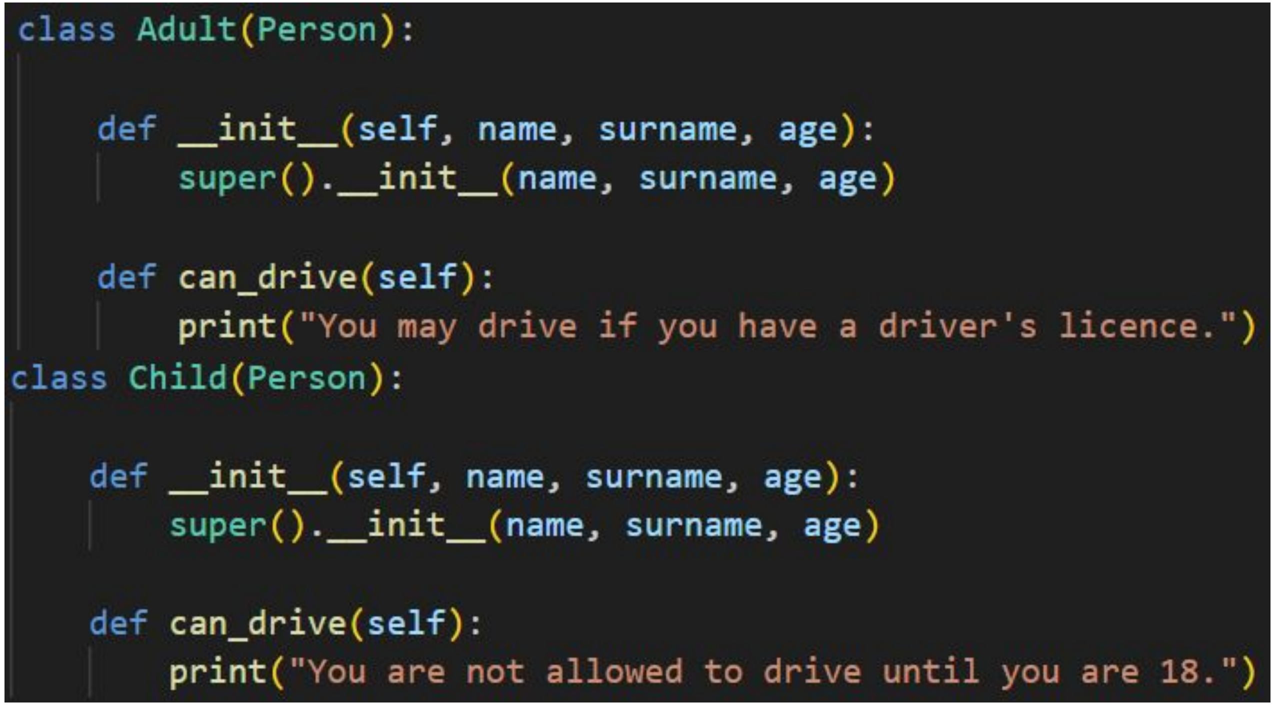


```python
class Person():

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

CoGrammar

Recap of Week 11: OOP

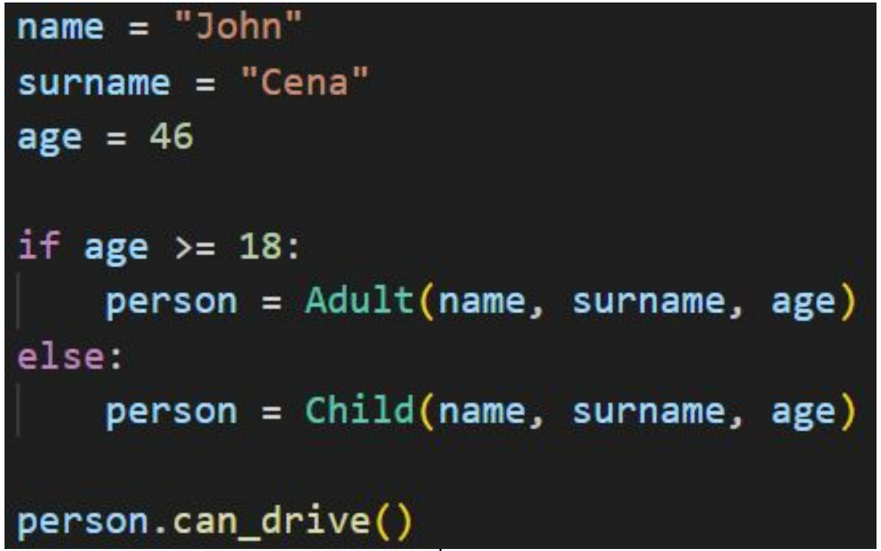
Inheriting from a Parent Class



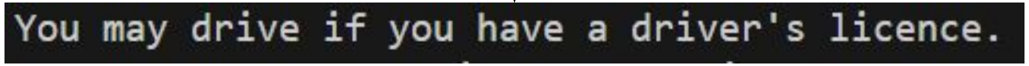
```python
class Adult(Person):

    def __init__(self, name, surname, age):
        super().__init__(name, surname, age)

    def can_drive(self):
        print("You may drive if you have a driver's licence.")
class Child(Person):

    def __init__(self, name, surname, age):
        super().__init__(name, surname, age)

    def can_drive(self):
        print("You are not allowed to drive until you are 18.")
```

Recap of Week 11: OOP

Creating Class Instances



```
name = "John"
surname = "Cena"
age = 46

if age >= 18:
    person = Adult(name, surname, age)
else:
    person = Child(name, surname, age)

person.can_drive()
```



```
You may drive if you have a driver's licence.
```

Quest for Lost Relics

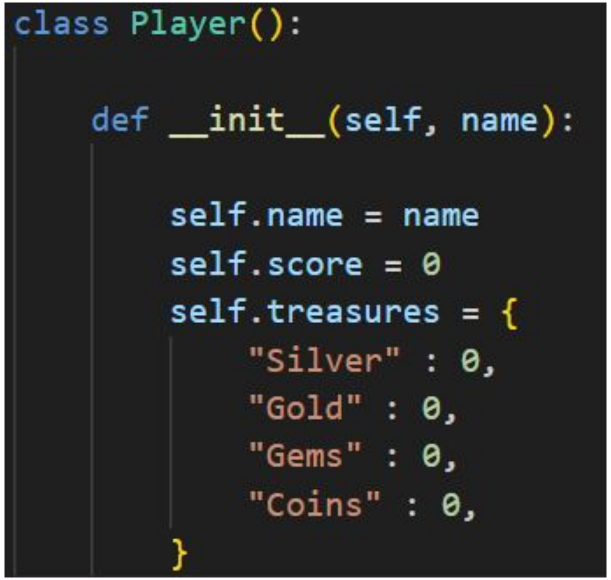
- **Background:** Your skills are required to create another game. You want to create a treasure-hunting game that will allow the player to collect treasure in some way, and reward the player based on the different treasures they collect.

- **Challenge:** Make use of class inheritance and abstraction to create a treasure-hunting game.

- **Objective:**
  - Create an abstract treasure class with common attributes and behaviours.
  - Implement inheritance to create specific types of treasures (e.g., gold, gems, artefacts).
  - Develop a treasure-hunt game where players collect various treasures with distinct properties.

Creating the Player Class

This is a basic set-up for a Player class. The player has attributes for their name, their current score, and how many of each treasure type they have found.

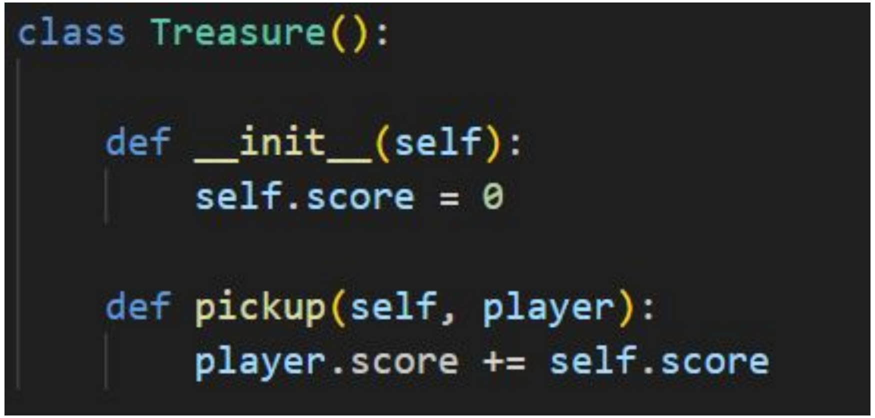


```python
class Player():

    def __init__(self, name):

        self.name = name
        self.score = 0
        self.treasures = {
            "Silver" : 0,
            "Gold" : 0,
            "Gems" : 0,
            "Coins" : 0,
        }
```

CoGrammar

Creating the Treasure Class

This will be our base class from which all our other treasures will inherit. All of our treasures will have a score attribute that they add to the user. They will also contain a method defining the behaviour when a player picks up the treasure.

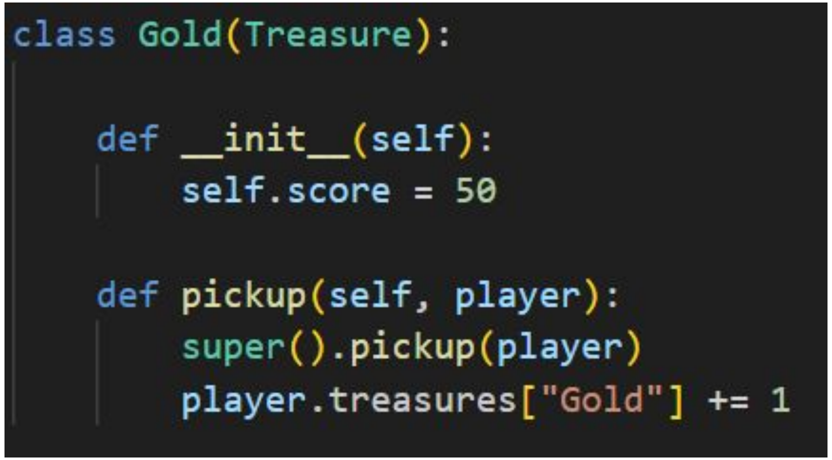


```
class Treasure():

    def __init__(self):
        self.score = 0

    def pickup(self, player):
        player.score += self.score
```

CoGrammar

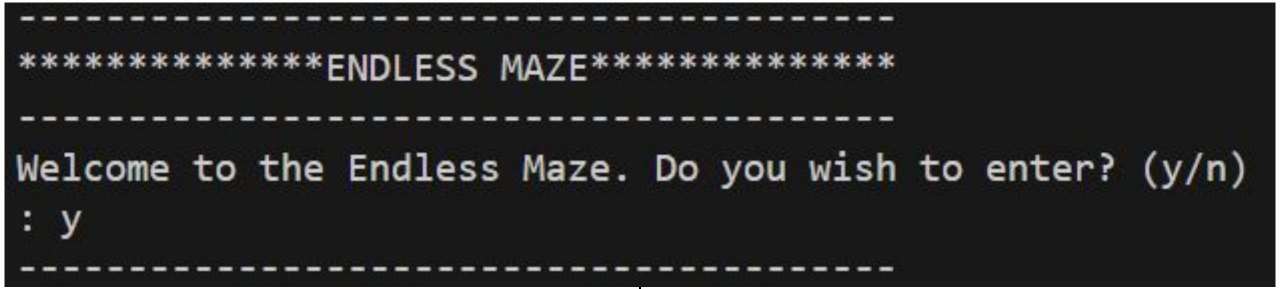
Inheriting from the Treasure Class

Now we can create one of our treasures by inheriting from our base Treasure class. We create the Gold class that has an attribute score of 50. The pickup() method is overridden; it still adds the score to the user, now it also increases the 'Gold' amount in the player's 'treasures' attribute.

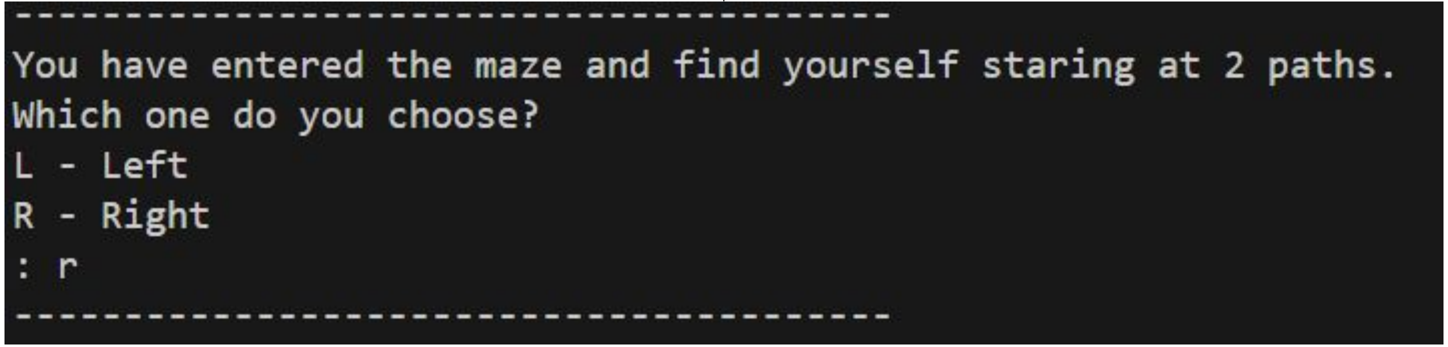


```python
class Gold(Treasure):

    def __init__(self):
        self.score = 50

    def pickup(self, player):
        super().pickup(player)
        player.treasures["Gold"] += 1
```

Output Example



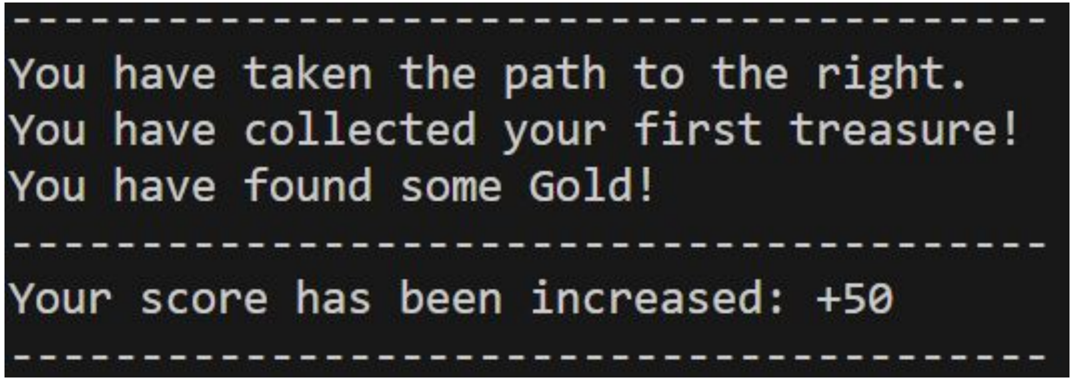
```
----------------------------------------
**************ENDLESS MAZE**************
----------------------------------------
Welcome to the Endless Maze. Do you wish to enter? (y/n)
: y
----------------------------------------
```



```
----------------------------------------
You have entered the maze and find yourself staring at 2 paths.
Which one do you choose?
L - Left
R - Right
: r
----------------------------------------
```

Collected Treasure Output Example

Here is an example of how we can display information about the outcome of the user's selected path.



```
----------------------------------------
You have taken the path to the right.
You have collected your first treasure!
You have found some Gold!
----------------------------------------
Your score has been increased: +50
----------------------------------------
```

ZooWonders

Make use of class inheritance and abstraction to create a treasure hunting game.

Important features:

1. **Menu:** Provide the user with a main menu where they can select to start a new game, view their high scores, select a profile, etc.

2. **Treasure Hunt:** Try to be creative in the way the player should hunt for treasure such as moving on a grid or selecting a room to enter from a list of rooms.

3. **Treasure:** Add different types of treasure that each provide the player with a different rewards.

4. **User Experience:** Try to make the game as fun as possible for the user. Remember to balance the game and not make it too difficult as it may deter some players.

Advanced Challenge:

- Add enemies to your games that the user can run into and provide the user with a way/ways to defeat enemies.

Summary

Quest for Lost Relics

★ Create a treasure-hunting game that allows the user to collect treasure.

Inheritance

★ Use inheritance to create different treasure types with different rewards and effects for the player.

Method Overriding

★ Method overriding can be used to add different functionality to each treasure type.

What is the primary purpose of inheritance in Python classes?

A. Code organisation and readability
B. Achieving multiple inheritance
C. Code reuse and extensibility
D. Implementing encapsulation

Which built-in function in Python can be used to check if a class inherits from another class or a tuple of classes?

A. issubclass()
B. inheritsfrom()
C. subclassof()
D. checkinherits()

Questions and Answers

Questions around the Case Study

CoGrammar

Thank you for joining