WORKSHEET

A1.8 Inheritance, polymorphism and overriding

Programming techniques and object-oriented programming · A level · OCR H446 1.2.4, AQA 7517 4.1.2.3, Eduqas A500QS 1.4 · about 25 min

Name _____ Class _____ Date _____

What this lesson is about

Subclasses and super, overriding, polymorphism, and abstract, virtual and static methods, with robot behaviours as subclasses.

- Inheritance
- Overriding
- Polymorphism
- Abstract, virtual and static methods
- Drawing inheritance

Questions

5 marks in all

1. What does this program print? [1 mark]

```
class Robot:
    def __init__(self, name):
        self.name = name
    def greet(self):
        return "I am " + self.name

class Rover(Robot):
    def greet(self):
        return super().greet() + " and I rove"

class Drone(Robot):
    pass

for r in [Rover("R1"), Drone("D1")]:
    print(r.greet())
```

2. What is polymorphism? [1 mark]

- ☐ **A** The same method call behaving differently depending on the class of the object it is called on
- ☐ **B** A class inheriting from two superclasses
- ☐ **C** Hiding an object's attributes
- ☐ **D** Creating many objects from one class

3. What is true of an abstract class? [1 mark]

- ☐ **A** It cannot be instantiated, and its abstract methods must be overridden by subclasses
- ☐ **B** It has no attributes
- ☐ **C** It cannot have subclasses
- ☐ **D** All its methods are static

4. A static method is called as Motor.clamp(120). What can it not do? [1 mark]

- ☐ **A** Use the attributes of a particular object
- ☐ **B** Take parameters
- ☐ **C** Return a value
- ☐ **D** Be called from another method

5. Inheritance models which relationship between a subclass and its superclass? Answer with two words. [1 mark]

The task: behaviours by inheritance

The starter has the superclass `Behaviour`, with a `name` attribute and an `act()` method. Write three subclasses from the class diagram. Each constructor calls `super().__init__` with the behaviour's name, and each overrides `act()` so that it does its job and then prints `<name> done`: - `Chirp(note)`: name `chirp`; stores `note` (a frequency in Hz) and its `act` plays that note twice, 0.2 seconds each time. - `Square(size)`: name `square`; stores `size` (cm) and its `act` drives a square with sides of `size` cm, turning right 90 degrees at each corner. - `Spin()`: name `spin`; its `act` turns right a full 360 degrees. Then make the list `routine = [Chirp(523), Square(20), Spin(), Chirp(784)]` and call `act()` on each object in order with one loop. Do not use `isinstance` or `type`: polymorphism chooses the method.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

class Behaviour:
    def __init__(self, name):
        self.name = name

    def act(self):
        print(self.name, "does nothing")

routine = [Behaviour("chirp"), Behaviour("square")]
for behaviour in routine:
    behaviour.act()
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a1-8-inheritance-and-polymorphism/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Add a `Pause(seconds)` subclass that does not override `act`. What does it print, and why?
2. Make `Behaviour` abstract with Python's `abc` module and `@abstractmethod`. What happens now when you try `Behaviour("test")`?
3. Add a `Polygon(sides, size)` subclass, then make `Square` a subclass of `Polygon`. How much of `Square` is left?