



Object Oriented Programming

A Level Student Booster

Which of the following statements is **NOT** an advantage of using Object-Oriented Programming (OOP)?

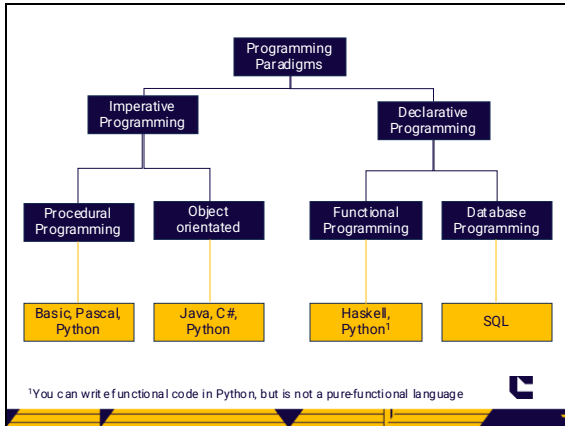
- A) OOP allows the creation of multiple object instances, which can reduce the overall amount of code and decrease the likelihood of errors.
- B) OOP ensures that attributes can be made private to prevent accidental changes from other parts of the program.
- C) OOP can lead to fewer mistakes since objects encapsulate their own data, unlike procedural programming where values are passed and returned.
- D) OOP uses global variables extensively, which increases memory usage.



Intended learning outcomes:

- Define and differentiate between objects and classes in object-oriented programming (OOP), highlighting their key components and relationships.
- Explain the role of constructors in class instantiation and demonstrate the creation of a class with attributes and methods.
- Instantiate objects from a class, modify their attributes, and employ methods to manage and manipulate object states effectively.
- Design and construct a class hierarchy that effectively demonstrates the use of inheritance and polymorphism.





OOP subject knowledge

Topics covered in this PDE:

- Objects
- Attributes
- Methods
- Classes
- Instantiation
- Constructors
- Inheritance
- Polymorphism
- UML Diagrams

The OOP Paradigm



How can objects represent the world around you in a program?

Object-oriented programming
1 minute intro



What is an object?



Activity 1

Isaac Computer Science Student Activity Booklet Object Oriented Programming

Activity 1: Pet object Attributes

Produce a formal definition of the pet object in the table below. You should include the data type of each attribute, and an example of the data, so that you are clear about the nature of each attribute.

Attribute	Data Type	Example

Complete **Activity 1** of your workbook.
3 minutes



OOP Definitions

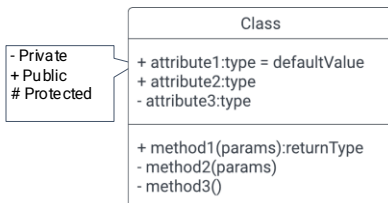
- **Class:** A template/blueprint for objects, specifying attributes and methods.
- **Attribute:** A variable that is associated with a class, representing the characteristics or properties of that object.
- **Method:** Is a subprogram defined within a class, designed to operate on or manipulate the attributes of that class.



What is a class?



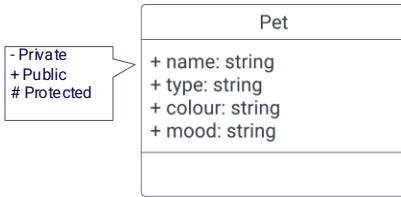
Activity 2



- Complete the top and middle sections of this class diagram in **Activity 2** of your workbook.
5 minutes

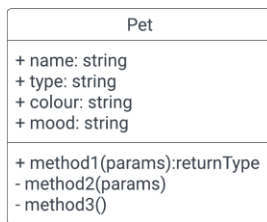


Class Identifiers and Attributes



Getter & Setter Methods

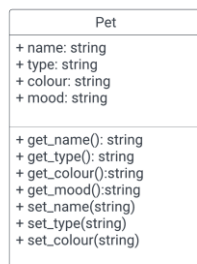
Activity 3



- Complete the final section of the class diagram in **Activity 3** of your workbook. **5 minutes**

Activity 3 – Solution

Activity 3



OOP Definitions

- **Encapsulation** is where the attributes are set to be private. These attributes can only be read and amended using get and set methods.
- An **object** is an instance of a class with actual attribute values.
- **Instantiation** is the creation of an object from a class. The instantiated class will have an identifier.



Encapsulation

- An object-oriented program puts all the data (**attributes**) and the code that can be carried out on that data (**methods**) together in one place called an **object**

```
# Attributes
private name

# Methods
public procedure set_name(given_name)
    name = given_name
Endprocedure
```



Constructors

- A class is a blueprint for an object, they contain attributes and methods, but no actual data.
- A class constructor is a special method automatically called upon to initialise the object, with the given values.

```
class Pet
    private name
    # Constructor method
    public procedure new (given_name)
        name = given_name
    Endprocedure

    public function get_name()
        return name
    Endfunction

    public procedure set_name(given_name)
        name = given_name
    Endprocedure
Endclass
```



Instantiation

- An **object** is a specific instance of a class. It contains concrete data for the attributes defined by its class.

```
# Instantiation
my_pet = new Pet('Romeo')

# Calling methods
my_pet.set_name('Juliet')
print(my_pet.get_name())
```

- Instantiation:** The creation of an object from a class.



Activity 4


tiny.cc/


Activity4

```
1: class Pet:
2:
3:     # Constructor
4:     def __init__(self, given_name, given_type, given_colour):
5:         self.name = given_name
6:         self.type = given_type
7:         self.colour = given_colour
8:
9:     def get_name(self): # getter method
10:        return self.name
11:
12:    def set_name(self, new_name): # setter method
13:        self.name = new_name
14:
15:    def describe(self):
16:        pass
17:
18:    ...
19: 1. Create a new pet object a cat called Romeo
20: 2. Use the setter method to change it's name to Juliet
21: 3. Use the getter method to check the change has been successful
22: 4. Complete the describe method, to return the object's attributes as a string
23: ...
```

- Complete **Activity 4** of your workbook.
10 minutes



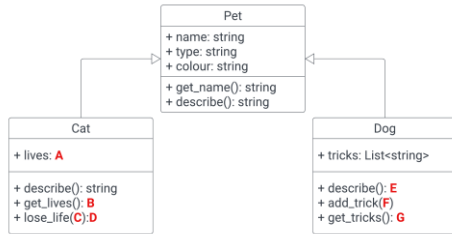
OOP Definitions

- Inheritance:** is where the sub class/child class inherits ALL the methods and attributes of a superclass/parent class. However, it can override the attributes and methods of the super class/ parent class.
- Overriding:** occurs when a subclass method supersedes a base class method. This can also apply to attributes.



Inheritance

Activity 5

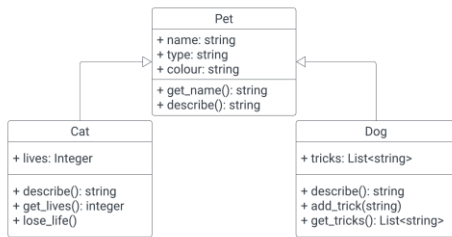


- Complete **Activity 5** of your workbook.
5 minutes



Activity 5 – Solutions

Activity 5



Superclasses & Subclasses

```

class Pet
  private name
  public procedure new(given_name)
    name = given_name
  Endprocedure
Endclass

class Cat inherits Pet
  private lives
  public procedure new(given_name)
    super.new(given_name)
    lives = 9
  Endprocedure
Endclass
  
```



Activity 6



```

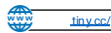
1 class Pet:
2     def __init__(self, given_name, given_type, given_colour):
3         self.name = given_name
4         self.type = given_type
5         self.colour = given_colour
6
7     def get_name(self):
8         return self.name
9
10    def describe(self):
11        return "I am a " + str(self.colour) + " " + str(self.type) + " called " + str(self.name)
12
13    # Define a new subclass 'Cat' which inherits from the 'Pet' superclass
14    class Cat(Pet):
15        # constructor
16        def __init__(self, given_name, given_colour):
17            # Calling the __init__ method of the superclass 'Pet'
18            super().__init__(given_name, 'cat', given_colour)
19            # Assigning the attribute 'lives'
20            self.lives = 9
21
22    # The describe method is also present in the Pet class this is an example of method overriding,
23    # which is a form of polymorphism.
24    def describe(self):
25        return "Meow, I am a " + str(self.colour) + " " + str(self.type) + " called " + str(self.name)
26
27    # Method to get the current number of lives of the Cat object.
28    def get_lives(self):
29        return self.lives
30
31    # Method to decrement the 'lives' attribute of the Cat object by 1.
32    def lose_a_life(self):
33        self.lives -= 1

```

- Complete **Activity 6** of your workbook.
10 minutes



Activity 7



```

1 class Pet:
2     def __init__(self, given_name, given_type, given_colour):
3         self.name = given_name
4         self.type = given_type
5         self.colour = given_colour
6
7     def get_name(self):
8         return self.name
9
10    def describe(self):
11        return "I am a " + str(self.colour) + " " + str(self.type) + " called " + str(self.name)
12
13    # Define a new subclass 'Cat' which inherits from the 'Pet' superclass
14    class Cat(Pet):
15        # constructor
16        def __init__(self, given_name, given_colour):
17            # Calling the __init__ method of the superclass 'Pet'
18            super().__init__(given_name, 'cat', given_colour)
19            # Assigning the attribute 'lives'
20            self.lives = 9
21
22    # The describe method is also present in the Pet class this is an example of method overriding,
23    # which is a form of polymorphism.
24    def describe(self):
25        return "Meow, I am a " + str(self.colour) + " " + str(self.type) + " called " + str(self.name)
26
27    # Method to get the current number of lives of the Cat object.
28    def get_lives(self):
29        return self.lives
30
31    # Method to decrement the 'lives' attribute of the Cat object by 1.
32    def lose_a_life(self):
33        self.lives -= 1
34
35    pet_name = input("Enter the name of your pet ")
36    pet_type = input("What type of animal is " + str(pet_name) + "?")
37    pet_colour = input("What colour is " + str(pet_name) + "?")
38
39    if pet_type.lower() == "cat":
40        my_cat = Cat(pet_name, pet_colour) # Instantiation
41        print(my_cat.describe()) # This will call the describe method defined in Cat, not in Pet.
42        print(my_cat.get_lives())
43        my_cat.lose_a_life()
44        print(my_cat.get_lives())

```

- Complete **Activity 7** in your workbook.
10 minutes



Activity 7 Solution



```

35 pet_name = input("Enter the name of your pet ")
36 pet_type = input("What type of animal is " + str(pet_name) + "?")
37 pet_colour = input("What colour is " + str(pet_name) + "?")
38
39 if pet_type.lower() == "cat":
40     my_cat = Cat(pet_name, pet_colour) # Instantiation
41     print(my_cat.describe()) # This will call the describe method defined in Cat, not in Pet.
42     print(my_cat.get_lives())
43     my_cat.lose_a_life()
44     print(my_cat.get_lives())

```

Powered by trinket

Enter the name of your pet John
What type of animal is John? Cat
What colour is John? Ginger
Meow, I am a Ginger Cat called John
9
8



Which of the following statements is **NOT** an advantage of using Object-Oriented Programming (OOP)?

- A) OOP allows the creation of multiple object instances, which can reduce the overall amount of code and decrease the likelihood of errors.
- B) OOP ensures that attributes can be made private to prevent accidental changes from other parts of the program.
- C) OOP can lead to fewer mistakes since objects encapsulate their own data, unlike procedural programming where values are passed and returned.
- D) OOP uses global variables extensively, which increases memory usage.

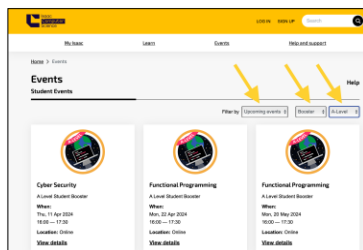


Intended learning outcomes:

- Define and differentiate between objects and classes in object-oriented programming (OOP), highlighting their key components and relationships.
- Explain the role of constructors in class instantiation and demonstrate the creation of a class with attributes and methods.
- Instantiate objects from a class, modify their attributes, and employ methods to manage and manipulate object states effectively.
- Design and construct a class hierarchy that effectively demonstrates the use of inheritance and polymorphism.



A-level Student Booster



Isaac CS Gameboards



Created: 12/04/2024
Last visited: 12/04/2024
Stage: Difficulty
A Level: P2, C2

Inheritance and polymorphism

By: Me

Assign / Unassign



Created: 12/04/2024
Last visited: 12/04/2024
Stage: Difficulty
A Level: P1

Encapsulation

By: Me

Assign / Unassign



Created: 12/04/2024
Last visited: 12/04/2024
Stage: Difficulty
A Level: P1, C1

OOP Fundamentals

By: Me

Assign / Unassign



Thank you

