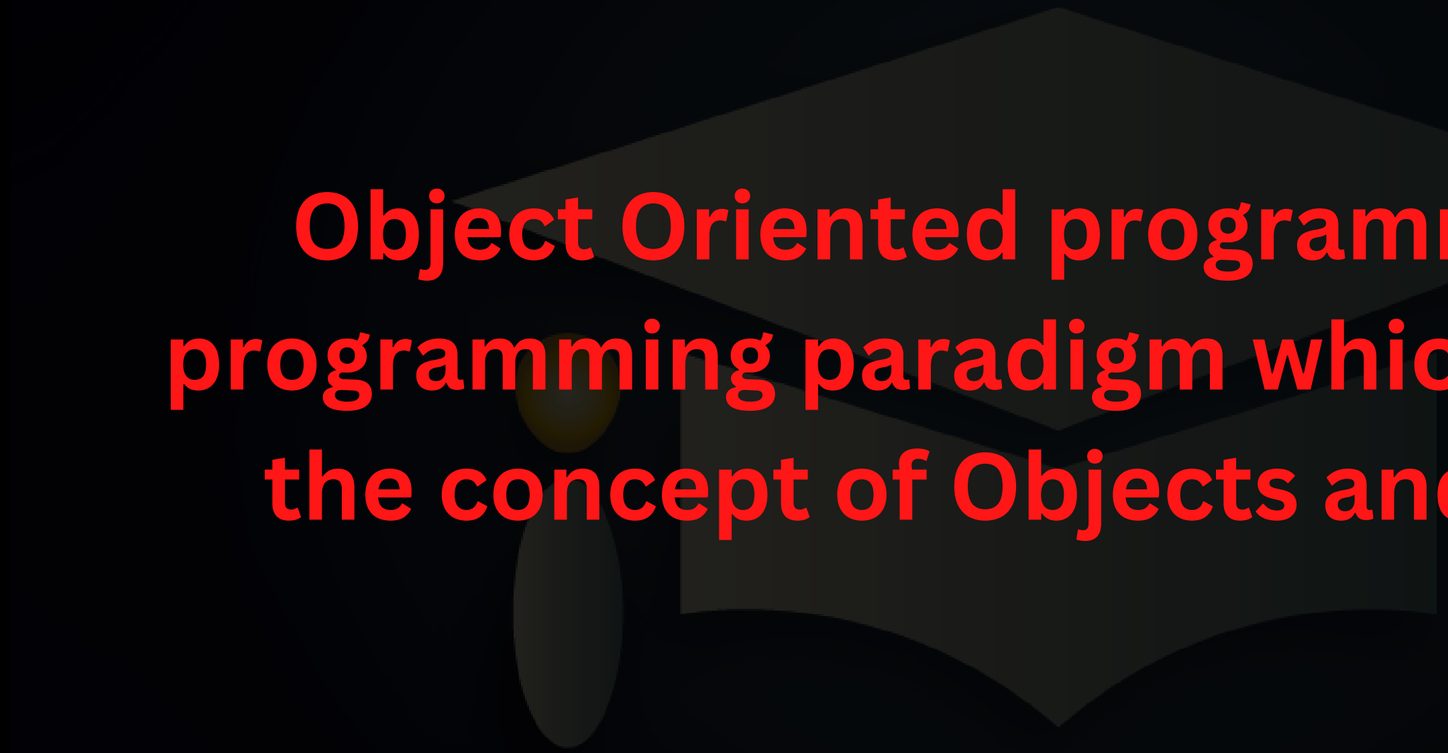
Papers Dock

PYTHON

9618

OBJECT ORIENTED PROGRAMMING

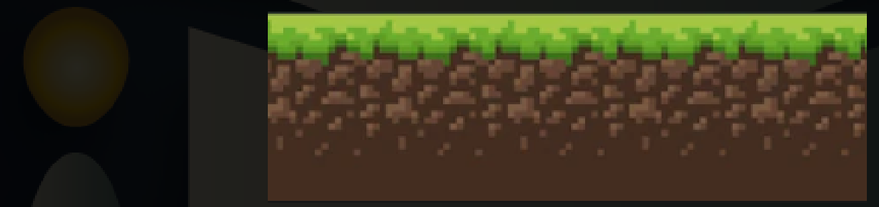
What is meant by OOP



Object Oriented programming is a programming paradigm which based on the concept of Objects and Classes

Objects

The Data and the Functions that belongs to single entity can be grouped in single object



Each Object Hold Its Specific Information

Attributes

Player Name, x-position, y-position, Width, Height

Variables

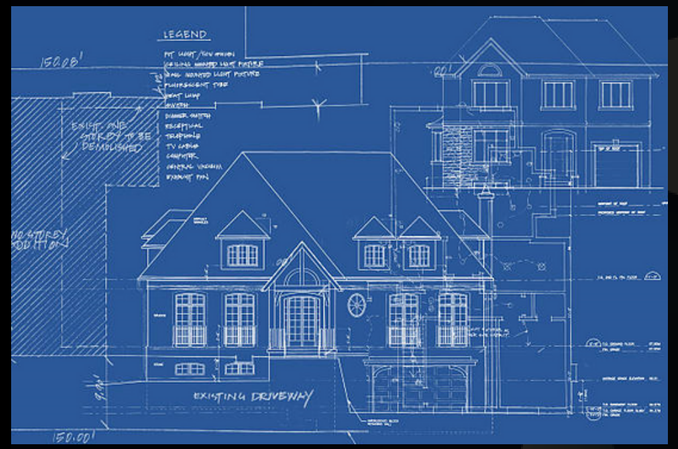
Methods

collision, movement, Score-calculation

Functions

An object not only contains attributes but also some action. Those actions are basically functions

What is the difference between Class and Object



Class

Object

Putting the data (attributes) and methods together as a single unit is called Encapsulation

Declaration Of Methods and Attributes

There are two declaration methods in OOP

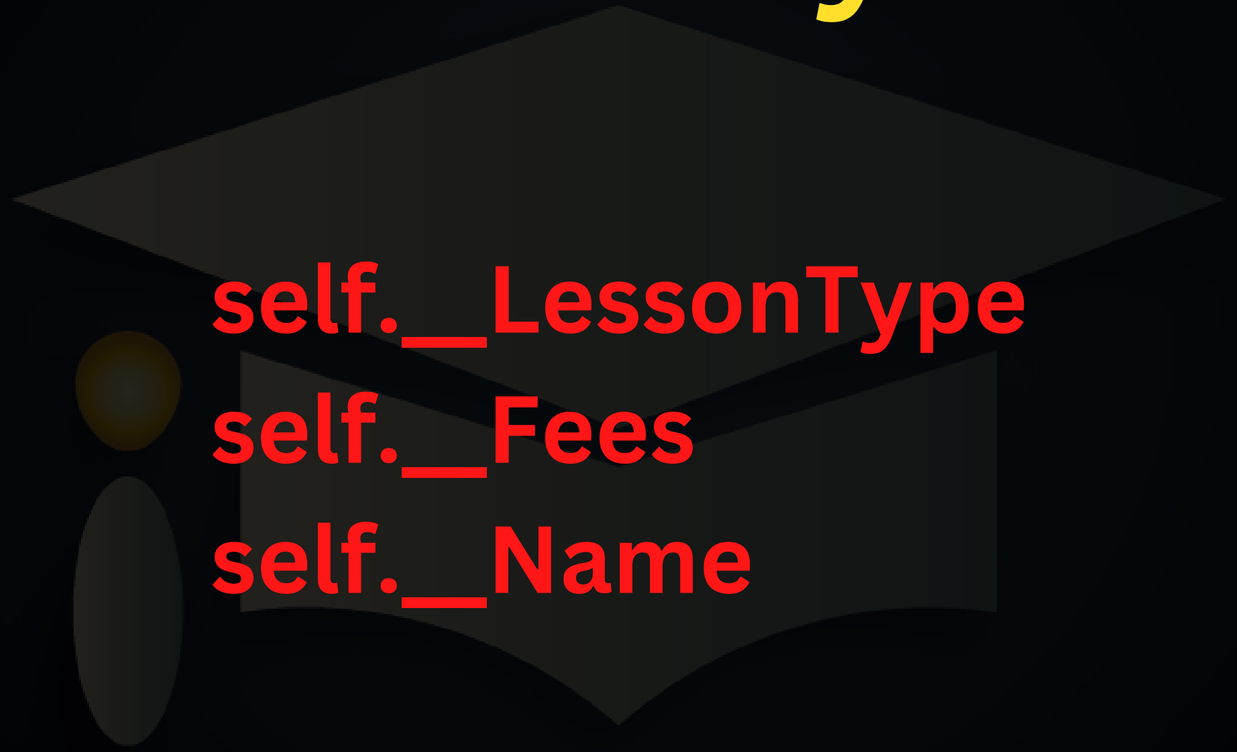
Public Private

The Attributes which are usually declared as PRIVATE can only be accessed inside the class

The Methods which are usually declared as PUBLIC can be accessed outside the class

NOTE : IN PYTHON YOU ARE SUPPOSE TO DECLARE ATTRIBUTES AS COMMENTS

How To Declare Attribute As Private In Python



self.__LessonType
self.__Fees
self.__Name

Double UnderScore is used to make the attribute Private

What is a Constructor ?

Constructor are Functions which are used for initializing the attributes / Properties of a class

def __init__():

Constructor In Python

QUESTION

P1

name : "Ghost"
age : 21

Object 1

Player

name :
age :
getAge ()
displayName ()

Class

P2

name : "Ninja"
age : 19

Object 2

Defining a class

```
class Player():
    #PRIVATE name : STRING
    #PRIVATE age : INTEGER
    def __init__(self, name, age):

        self.__name = name
        self.__age = age

    # METHODS
    def GetAge(self):
        return self.__age

    def DisplayName(self):
        return self.__name
```

Creating Objects

```
P1 = Player("Ghost", 21)
P2 = Player("Ninja", 19)

print(P1.DisplayName())
```

Practice Question

Picture	

Description : STRING
Width : INTEGER
Height : INTEGER
FrameColour : STRING
	// stores a description of the picture
	// stores the width e.g. 30
	// stores the height e.g. 40
	// stores the colour e.g. black

Constructor()
 	
GetDescription()
GetHeight()
GetWidth()
GetColour()
SetDescription()
	// takes all four values as parameters and sets them to the private attributes
	// returns the description of the picture
	// returns the height
	// returns the width
	// returns the frame colour
	// takes the new description as a parameter and writes the value to description

```
class Picture():
    #PRIVATE Description : STRING
    #PRIVATE Width : INTEGER
    #PRIVATE Height : INTEGER
    #PRIVATE FrameColour : STRING
    def __init__(self, Des, wid, Hei, Frame):
        self.__Description = Des
        self.__Width = int(wid)
        self.__Height = int(Hei)
        self.__FrameColour = Frame

    # GETTERS

    def GetDescription(self):
        return self.__Description

    def GetHeight(self):
        return self.__Height

    def GetWidth(self):
        return self.__Width

    def GetColour(self):
        return self.__FrameColour

    # SETTERS

    def SetDescription(self, newdesc):
        self.__Description = newdesc
```

TreasureChest	

question : STRING
answer : INTEGER
points : INTEGER
	// stores the question
	// stores the answer
	// stores the maximum possible number of points available for this chest

constructor()	// takes question, answer and points as parameters and creates an instance of an object
getQuestion()	// returns the question
checkAnswer()	// takes the user's answer as a parameter and returns True if it is correct, otherwise returns False
getPoints()	// takes the number of attempts as a parameter and returns the number of points awarded

(a) Create a new program.

Write program code to declare the class `TreasureChest`.

Do **not** write any other methods.

The attributes are private.

If you are using the Python programming language, include attribute declarations using comments.

```
class TreasureChest():
    #PRIVATE question : STRING
    #PRIVATE answer : INTEGER
    #PRIVATE points : INTEGER

    def __init__(self, ques, ans, point):
        self.__question = ques
        self.__answer = ans
        self.__points = point
```

(c) The main program repeats each question until the user inputs the correct answer. The number of points awarded depends on the number of attempts before the user gives the correct answer.

(i) The class `TreasureChest` has a method `getQuestion()` that returns the question.

Write the method `getQuestion()`.

(ii) The class `TreasureChest` has a method `checkAnswer()` that takes the user's answer as a parameter. It returns `True` if the answer is correct and `False` otherwise.

Write the method `checkAnswer()`.

```
def getQuestion(self):
    return self.__question

def checkAnswer(self, userans):
    if userans == self.__answer:
        return True
    else:
        return False
```

(iii) The class `TreasureChest` has a method `getPoints()` that takes the number of attempts as a parameter.

- If the number of attempts is 1, it returns the value of `points`.
- If the number of attempts is 2, it returns the integer value of `points` divided by 2 (`DIV 2`).
- If the number of attempts is 3 or 4, it returns the integer value of `points` divided by 4 (`DIV 4`).
- If the number of attempts is not 1 or 2 or 3 or 4, it returns `0` (zero).

For example, a question is worth 100 points and the user took 2 attempts to give the correct answer. The user is awarded 50 points (100 DIV 2).

Write the method `getPoints()`.

```
def getPoints(self, attempts):

    if attempts == 1:
        return int(self.__points)
    elif attempts == 2:
        temp = self.__points // 2
        return int(temp)
    elif attempts == 3 or attempts == 4:
        temp = self.__points // 4
        return int(temp)
    else:
        return 0
```

```
class TreasureChest():
    #PRIVATE question : STRING
    #PRIVATE answer : INTEGER
    #PRIVATE points : INTEGER

    def __init__(self, ques, ans, point):
        self.__question = ques
        self.__answer = ans
        self.__points = point

    def getQuestion(self):
        return self.__question

    def checkAnswer(self, userans):
        if userans == self.__answer:
            return True
        else:
            return False

    def getPoints(self, attempts):

        if attempts == 1:
            return int(self.__points)
        elif attempts == 2:
            temp = self.__points // 2
            return int(temp)
        elif attempts == 3 or attempts == 4:
            temp = self.__points // 4
            return int(temp)
        else:
            return 0
```

Balloon	
`Health : INTEGER`	The health of the balloon
`Colour : STRING`	The colour of the balloon
`DefenceItem : STRING`	The item the balloon uses to defend itself
`Constructor()`	Initialises the defence item and colour using the parameters Initialises health to 100
`ChangeHealth()`	Takes the change as a parameter and adds this to the health
`GetDefenceItem()`	Returns the defence item of the object
`CheckHealth()`	If the health is 0 or less, it returns `TRUE`, otherwise it returns `FALSE`

(a) The constructor takes the name of the defence item and the balloon's colour as parameters and sets these to the attributes. The health is initialised to `100`.

Write program code to declare the class `Balloon` and its constructor. Do not write any other methods.

Use your language appropriate constructor.

All attributes should be private. If you are writing in Python include attribute declarations using comments.

```
class Balloon() :
    # PRIVATE HEALTH : INTEGER
    # PRIVATE COLOR : STRING
    # PRIVATE DEFENcEITEM ; STRING

    def __init__(self, color, defitem) :
        self.__defenceitem = defitem
        self.__color = color
        self.__health = 100
```

(b) The get method `GetDefenceItem()` returns the defence item of the object.

Amend your program code to include the get method `GetDefenceItem()`.

(c) The object's method `ChangeHealth()` takes an integer number as a parameter and adds this to the health attribute of the object.

Amend your program code to include the method `ChangeHealth()`.

(d) The object's method `CheckHealth()` returns `TRUE` if the health of the object is 0 or less (no health remaining) and returns `FALSE` otherwise (health remaining).

Amend your program code to include the method `CheckHealth()`.

```
def getdefenceitem(self) :
    return self.__defenceitem

def changehealth(self, newhealth) :
    self.__health = self.__health + newhealth

def checkhealth(self) :
    if self.__health <= 0 :
        return True
    else :
        return False
```

(e) Amend the main program to:

- take as input a defence item and colour from the user
- create a new balloon with the identifier `Balloon1` using the data input.

```
userdefence = input ("enter the defence item: ")
usercolor = input ("enter the color: ")

Balloon1 = Balloon(usercolor, userdefence)
```

(f) The function `Defend()`:

- takes a balloon object as a parameter
- takes as input the strength of an opponent from the user
- uses the `ChangeHealth()` method to subtract the strength input from the object's health
- outputs the defence item of the balloon
- checks the health of the object and outputs an appropriate message if it has no health remaining, or if it has health remaining
- returns the amended balloon object.

Write program code to declare the function `Defend()`.

```
def defend (balloonobject) :
    oppstrength = int(input("enter the strength: "))
    balloonobject.changehealth(-oppstrength)

    temp = balloonobject.getdefenceitem()
    print("your defence item is :", temp)

    check = balloonobject.checkhealth()
    if check == True :
        print("no health left")
    else :
        print("health left")

    return balloonobject
```

(g) **(i)** Amend the main program to call the function `Defend()`.

(ii) Test your program using the following inputs:

- balloon defence method `"Shield"`
- balloon colour `"Red"`
- strength of opponent `50`

Take a screenshot to show the output.

```
Balloon1 = Defend(Balloon1)
```

```
C:\Users\muham\PycharmProjects\pythonProject5\venv\Scripts\
Enter the Defence Item: Shield
Enter the Colour: Red
Enter the Strength: 50
Your Defence is: Shield
Health Remaining and your defence worked
```

5 A tennis club is developing a program to store details of the lessons it offers. The programmer has designed the class `Lesson` for the details of the lessons.

The following class diagram shows the design for the `Lesson` class.

Lesson

```
LessonType : STRING // initialised in constructor to the parameter
                    // value passed to the constructor
Instructor : STRING // initialised in constructor to the parameter
                    // value passed to the constructor
```

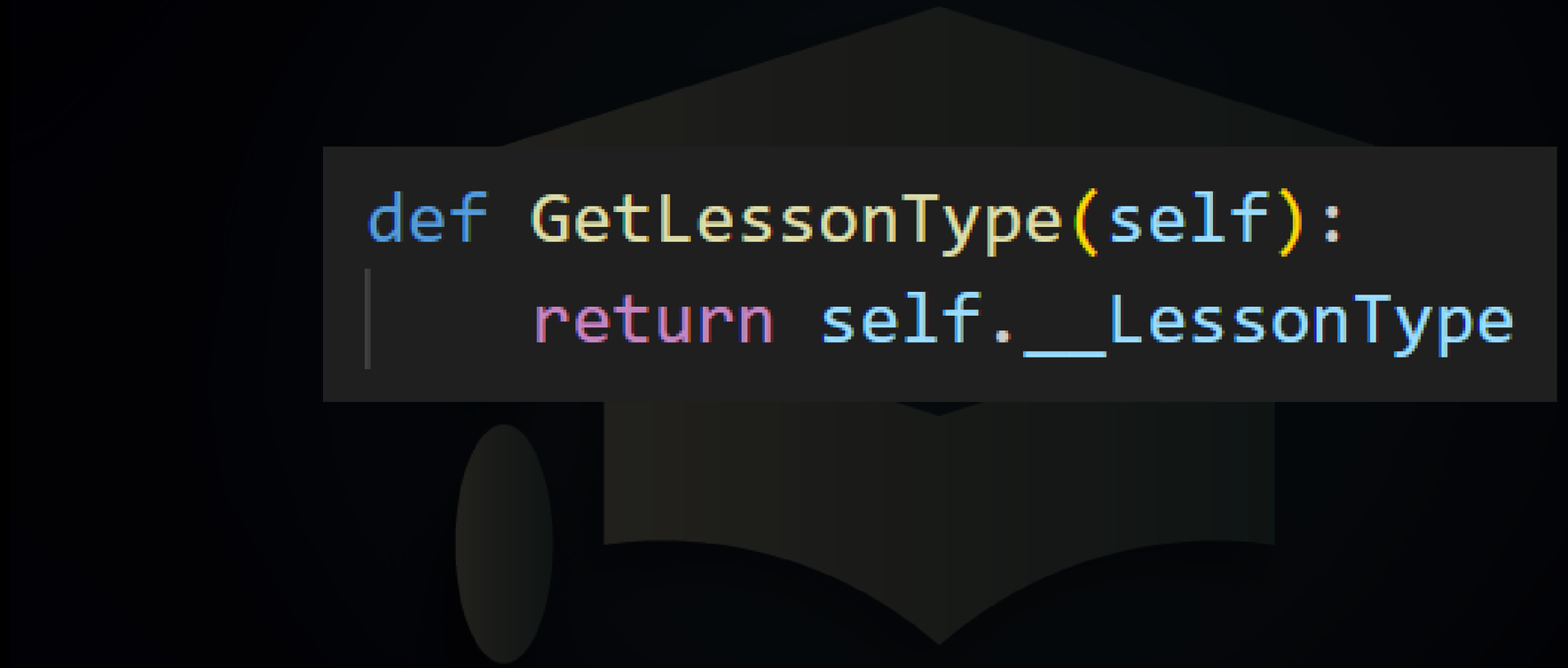
```
Constructor()       // method used to create and initialise an
                    // object
GetLessonType()     // returns LessonType value
GetInstructor()     // returns Instructor value
GetFee()            // returns the cost of a lesson
SetLessonType()     // sets the LessonType to the parameter value
SetInstructor()     // sets the Instructor to the parameter value
```

(a) Write **program code** for the `Constructor()` method.

Use the appropriate constructor method for your chosen programming language.

```
class Lesson():
    #PRIVATE LessonType : STRING
    #PRIVATE Instructor : STRING

    def __init__(self, LessonType, Instructor):
        self.__LessonType = LessonType
        self.__Instructor = Instructor
```

(b) Write **program code** for the `GetLessonType()` method.


def GetLessonType(self):
return self.__LessonType

(c) `Fee` is the cost that a customer will pay for a lesson.

The method `GetFee()` validates the parameter value. The method is sent a parameter value that represents the skill level of the customer: beginner, intermediate or advanced.

The parameter will be a character:

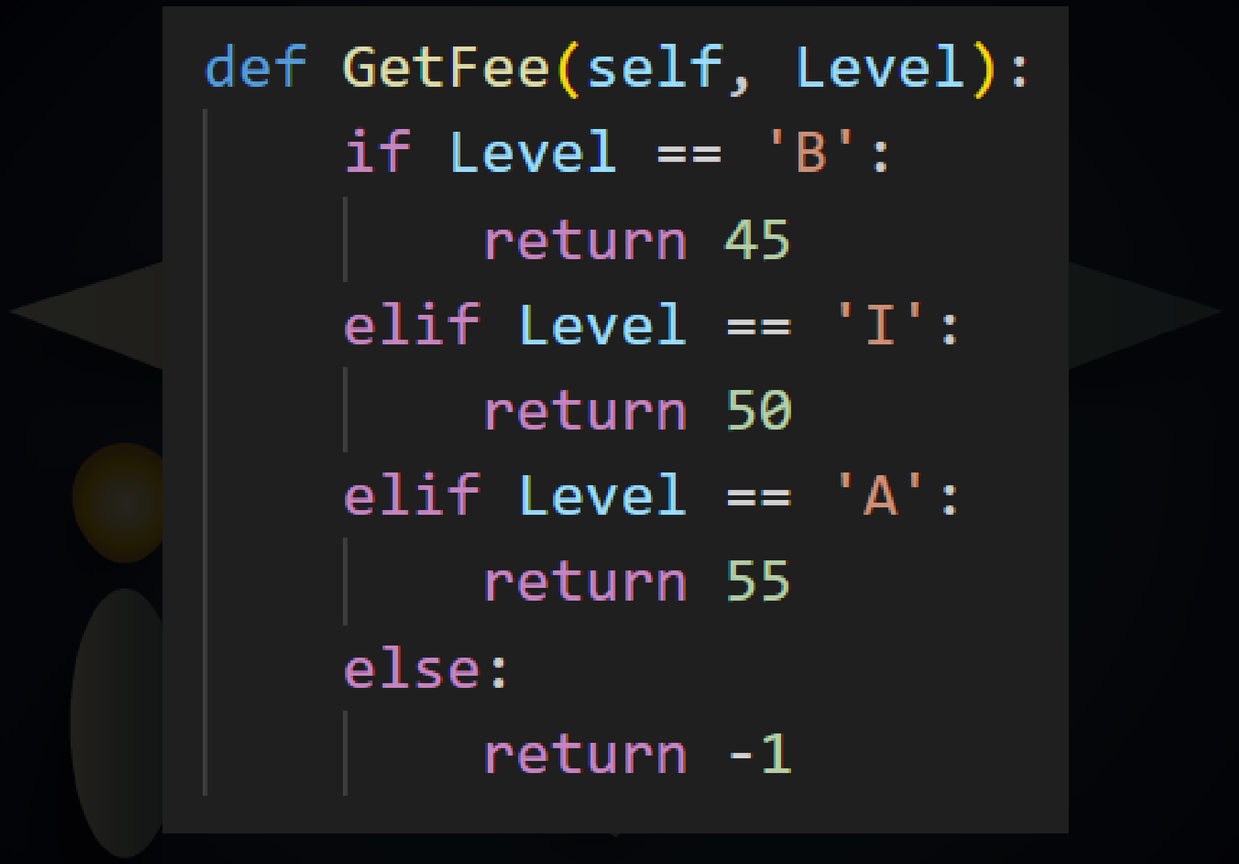
- `'B'` for beginner
- `'I'` for intermediate
- `'A'` for advanced.

The method must check the parameter value is a valid character (`'B'`, `'I'` or `'A'`) and return the correct fee. It must return `-1` if it is not a valid character.

The fees are:

- $45 for a beginner
- $50 for an intermediate
- $55 for an advanced.

Write **program code** for the `GetFee()` method.


def GetFee(self, Level):
if Level == 'B':
return 45
elif Level == 'I':
return 50
elif Level == 'A':
return 55
else:
return -1

(d) The tennis club only offers nine different types of lesson. The lesson objects are stored in a 1D array.

Write **pseudocode** to declare an array `LessonArray` to store the nine lesson objects.

(e) The tennis club has the lesson 'Improve Your Serve' that has David as the instructor.

Write **program code** to create the lesson 'Improve Your Serve' as an instance of the class `Lesson`. The object needs to be stored in the third element of the array `LessonArray`.

```
#DECLARE LessonArray : ARRAY [0 : 8] OF Lesson
LessonArray = [" "] * 9

LessonArray[2] = Lesson("Improve Your Serve", "David")

LessonArray[2].SetLessonType("OOP Class")
print(LessonArray[2].GetLessonType())

tempfees = LessonArray[2].GetFee('B')
print("Your Fees Is: ", tempfees)
```

Concept Of Inheritance

Part Time Employee

Attributes

Name : String

Age : Integer

Hourlyrate: Integer

Methods

GetName()

DailyWage()

Full Time Employee

Attributes

Name : String

Age : Integer

MonthlyRate: Integer

Methods

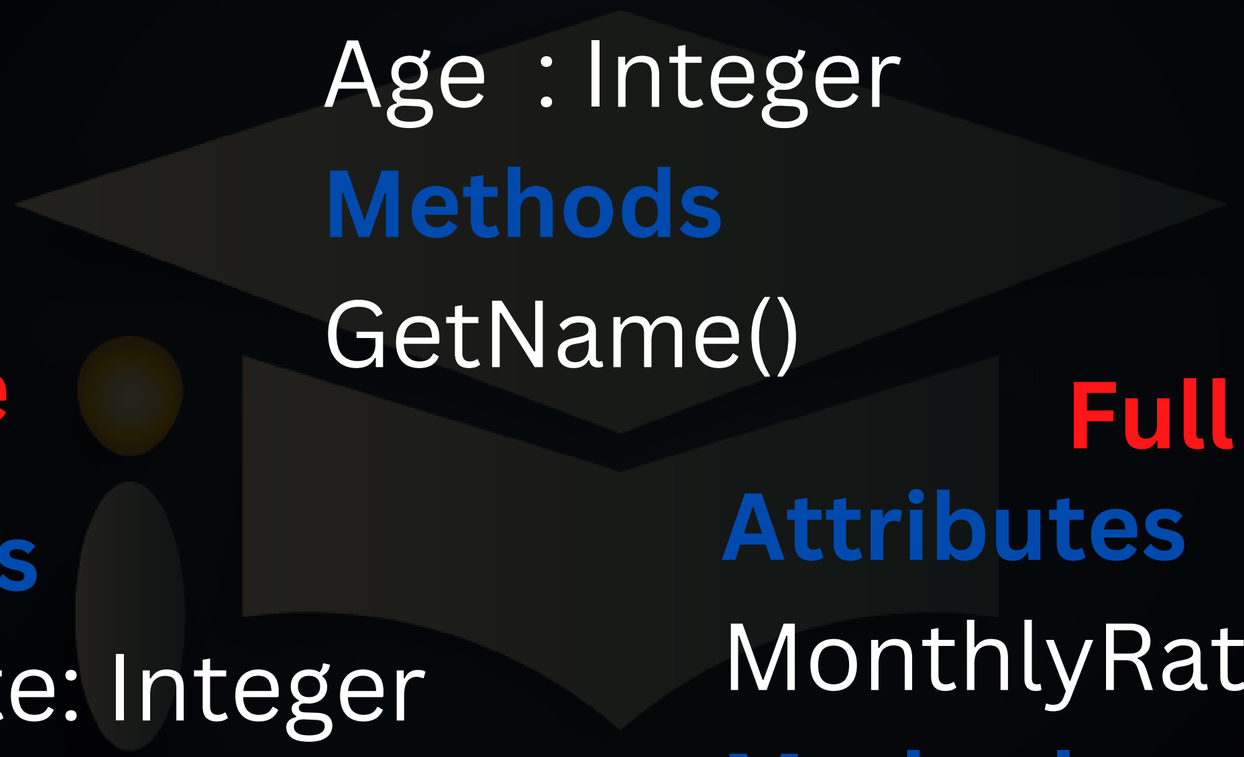
GetName()

YearlySalary()

Some Properties and Functions are repeated

```
class PartTimeEmployee():
    #PRIVATE Name : String
    #PRIVATE Age : Integer
    #PRIVATE HourlyRate : Integer
    def __init__(self, NameP, AgeP, HourlyRateP):
        self.__Name = NameP
        self.__Age = AgeP
        self.__HourlyRate = HourlyRateP

    def GetName(self):
        return self.__Name

    def DailyWage(self, HoursWorked):
        temp = HoursWorked * self.__HourlyRate
        return temp
```

```
class FullTimeEmployee():
    # PRIVATE Name : String
    # PRIVATE Age : Integer
    # PRIVATE MonthlyRate : Integer
    def __init__(self, NameP, AgeP, MonthlyRateP):
        self.__Name = NameP
        self.__Age = AgeP
        self.__MonthlyRate = MonthlyRateP

    def GetName(self):
        return self.__Name

    def YearlySalary(self):
        temp = self.__MonthlyRate * 12
        return temp
```



Employee

Attributes

Name : String

Age : Integer

Methods

GetName()

Part Time

Attributes

Hourlyrate: Integer

Methods

DailyWage()

Full Time

Attributes

MonthlyRate: Integer

Methods

YearlySalary()

```
class Employee():
    # PRIVATE Name : String
    # PRIVATE Age : Integer
    def __init__(self, NameP, AgeP):
        self.__Name = NameP
        self.__Age = AgeP

    def GetName(self):
        return self.__Name
```

```
class PartTime(Employee):

    # PRIVATE HourlyRate : Integer
    def __init__(self, NameP, AgeP, HourlyRateP):
        super().__init__(NameP, AgeP)
        self.__HourlyRate = HourlyRateP

    def DailyWage(self, HoursWorked):
        temp = HoursWorked * self.__HourlyRate
        return temp
```

```
class FullTime(Employee):
    # PRIVATE MonthlyRate : Integer
    def __init__(self, NameP, AgeP, MonthlyRateP):
        super().__init__(NameP, AgeP)
        self.__MonthlyRate = MonthlyRateP

    def YearlySalary(self):
        temp = self.__MonthlyRate * 12
        return temp
```

```
worker = PartTime("Bhatti", 50, 20)
PermanentWorker = FullTime("Sheikh", 60, 3000)
print(PermanentWorker.GetName())
print(PermanentWorker.YearlySalary())
```

What Is Meant By Inheritance ?

. The derived class can use the properties from the Parent Class Without Redeclaring them

. The derived class can use the Methods from the Parent Class Without Redeclaring them

. Can extend the Properties from the Parent Class

. Can extend the Methods from the Parent Class

super().

super() is a built-in Python function that provides a way to access methods and attributes of a superclass from a subclass. It is used to call the implementation of a method that is defined in a superclass, which has been overridden in a subclass.

Polymorphism

polymorphism is the ability of a method to take on different behaviors depending on the object that calls it. This allows a single method name to be used across different classes, with each class providing its own implementation.

```python
class Vehicle:
    def drive(self):
        print("Driving the vehicle")

class Car(Vehicle):
    def drive(self):
        super().drive()
        print("Driving the car")

# Create an instance of the Car class
my_car = Car()

# Call the drive method on the Car instance
my_car.drive()
```

How to identify if you should use super(). or not

If you want to extend the methods used in Super Class then you should use super().Method

Additionally is the Key Word to Identify

3 A computer game is written using object-oriented programming.

The game has multiple characters that can move around the screen.

The class `Character` stores data about the characters. Each character has a name, a current X (horizontal) position and a current Y (vertical) position.

Character	
`Name : STRING`	stores the name of the character as a string
`XPosition : INTEGER`	stores the X position as an integer
`YPosition : INTEGER`	stores the Y position as an integer
`Constructor()`	initialises `Name`, `XPosition` and `YPosition` to its parameter values
`GetXPosition()`	returns the X position
`GetYPosition()`	returns the Y position
`SetXPosition()`	adds the parameter to the X position validates that the new X position is between 0 and 10000 inclusive
`SetYPosition()`	adds the parameter to the Y position validates that the new Y position is between 0 and 10000 inclusive
`Move()`	takes a direction as a parameter and calls either `SetXPosition` or `SetYPosition` with an integer value

(a) (i) Write program code to declare the class `Character` and its constructor.

Do **not** declare the other methods.

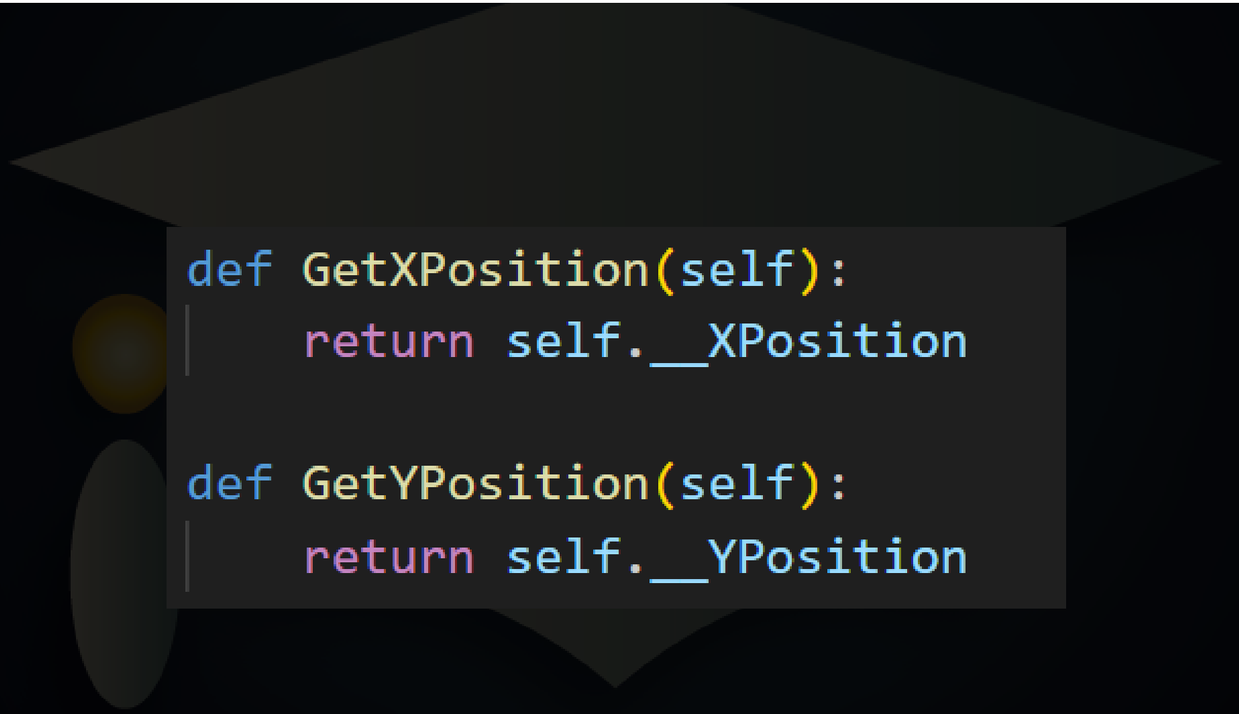
Use your programming language's appropriate constructor.

If you are writing in Python, include attribute declarations using comments.

```
class Character():
    #PRIVATE Name : STRING
    #PRIVATE XPosition : INTEGER
    #PRIVATE YPosition : INTEGER
    def __init__(self, NameP, XPositionP, YPositionP):
        self.__Name = NameP
        self.__XPosition = XPositionP
        self.__YPosition = YPositionP
```

(ii) The get methods `GetXPosition()` and `GetYPosition()` each return the relevant attribute.

Write program code for the get methods.



```
def GetXPosition(self):
    return self.__XPosition

def GetYPosition(self):
    return self.__YPosition
```

(iii) The set methods `SetXPosition()` and `SetYPosition()` each take a value as a parameter and add this to the current X or Y position.

If the new value exceeds 10 000, it is limited to 10 000.

If the new value is below 0, it is limited to 0.

Write program code for the set methods.

```
def SetXPosition(self, NewXPosition):
    self.__XPosition = self.__XPosition + NewXPosition

    if self.__XPosition > 10000:
        self.__XPosition = 10000
    elif self.__XPosition < 0:
        self.__XPosition = 0

def SetYPosition(self, NewYPosition):
    self.__YPosition = self.__YPosition + NewYPosition

    if self.__YPosition > 10000:
        self.__YPosition = 10000
    elif self.__YPosition < 0:
        self.__YPosition = 0
```

(iv) The method `Move()` takes a string parameter: `"up"`, `"down"`, `"left"` or `"right"`.

The table shows the change each direction will make to the X or Y position.

Use the appropriate method to change the position value.

Direction	Value change
up	Y position + 10
down	Y position − 10
left	X position − 10
right	X position + 10

Write program code for `Move()`.

```
def Move(self, Direction):
    if Direction == "up":
        self.SetYPosition(10)
    elif Direction == "down":
        self.SetYPosition(-10)
    elif Direction == "left":
        self.SetXPosition(-10)
    elif Direction == "right":
        self.SetXPosition(10)
```

(b) Write program code to declare a new instance of `Character` with the identifier `Jack`.

The starting X position is 50 and the starting Y position is 50, the character's name is Jack.

```
Jack = Character("Jack", 50, 50)
```

(c) The class `BikeCharacter` inherits from the class `Character`.

BikeCharacter	
`Constructor()`	takes `Name`, `XPosition` and `YPosition` as parameters calls its parent class constructor with the appropriate values
`Move()`	overrides the method `Move()` from the parent class by changing either the X position or the Y position by 20 instead of 10

(i) Write program code to declare the class `BikeCharacter` and its constructor.

Do **not** declare the other method.

Use your programming language's appropriate constructor.

If you are writing in Python, include attribute declarations using comments.

```
class BikeCharacter(Character):
    def __init__(self, NameP, XPositionP, YPositionP):
        super().__init__(NameP, XPositionP, YPositionP)
```

(ii) The method `Move()` overrides the method from the parent class.

The table shows the change each direction will make to the X or Y position.

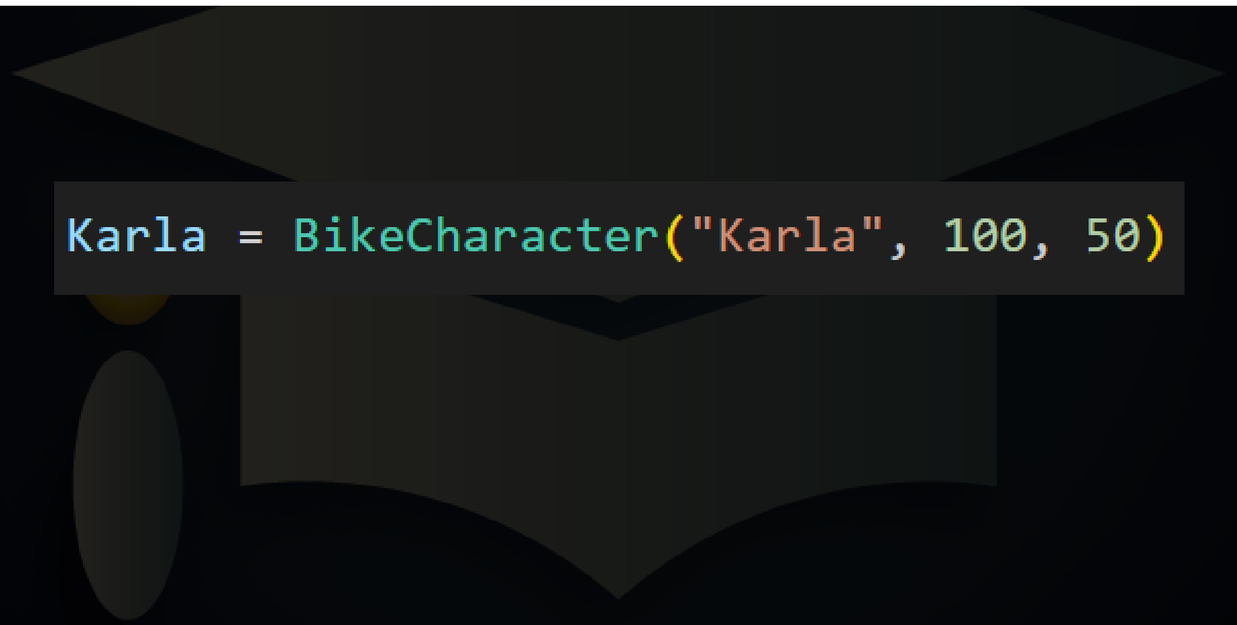
Direction	Value change
up	Y position + 20
down	Y position − 20
left	X position − 20
right	X position + 20

Write program code for `Move()`.

```
def Move(self, Direction):
    if Direction == "up":
        super().SetYPosition(20)
    elif Direction == "down":
        super().SetYPosition(-20)
    elif Direction == "left":
        super().SetXPosition(-20)
    elif Direction == "right":
        super().SetXPosition(20)
```

(d) Write program code to declare a new instance of `BikeCharacter` with the identifier `Karla`.

The starting X position is 100, the starting Y position is 50 and the character's name is Karla.



```
Karla = BikeCharacter("Karla", 100, 50)
```

(e) (i) Write program code to:

- take as input which of the two characters the user would like to move
- take as input the direction the user would like the character to move
- call the appropriate method to move the character
- output the character's new X and Y position in an appropriate format, for example:

```
"Karla's new position is X = 100 Y = 200"
```

All inputs require appropriate prompts and must be validated.

```
userCharacter = input("Chooses the character : Jack or Karla: ")
userDirection = input("Choose the direction : up down left right: ")

if userCharacter.lower() == "jack":
    Jack.Move(userDirection)
    xPos = Jack.GetXPosition()
    yPos = Jack.GetYPosition()
    print("Jack's new position is X =", xPos, "Y =", yPos)
elif userCharacter.lower() == "karla":
    Karla.Move(userDirection)
    xPos = Karla.GetXPosition()
    yPos = Karla.GetYPosition()
    print("Karla's new position is X =", xPos, "Y =", yPos)
```

(ii) Test your program twice with the following inputs.

Test 1: jack right

Test 2: karla down

Take a screenshot of the output.

Containment

Containment is a fundamental concept in Object-Oriented Programming (OOP) that refers to the ability to include one object inside another object.

1 A vending machine allows users to insert coins to purchase an item.

The user then enters the code for the item they would like the machine to dispense (give out). The user must re-enter the code until it is valid.

If the code is valid but the user has not inserted enough money for the item chosen, the machine waits for more coins to be inserted. The user then has to re-enter the code.

The user can press cancel at any time to return the money inserted into the machine.

(b) The vending machine is part of a program that is written using object-oriented programming (OOP). The vending machine makes use of two classes that are described in the following tables.

All attributes are declared as private.

foodItem	
`name : STRING`	`// the name of the item of food`
`code : STRING`	`// the code to be entered for that item to be` `// selected`
`cost : REAL`	`// the cost of the item`
`constructor(nameP, codeP,` `costP)`	`// creates an instance of foodItem` `// takes the name, code and cost as parameters`
`getCode()`	`// returns the code for the item`
`getCost()`	`// returns the cost of the item`
`getName()`	`// returns the name of the item`

vendingMachine	

items : ARRAY[0:3] OF foodItem
moneyIn : REAL
	// stores four items of type foodItem
	// stores the total money inserted by the
	// user, initialised to 0 in the constructor

constructor(item1, item2,
item3, item4)
 	
insertMoney()
 	
checkValid ()
 	
getItemName()
	// creates an instance of vendingMachine,
	// takes four objects of type foodItem as
	// parameters and stores them in array items
	
	// takes the value of the coin as a parameter
	// and adds it to moneyIn
	// takes a code as a parameter and checks it is
	// valid against the food item codes
	// takes the array index as a parameter and
	// returns the name of the food items

(i) Write **program code** to declare the class `vendingMachine`. You are only required to write program code for the attribute declarations and the constructor.

If you are writing in Python, include attribute declarations using comments.

Use your programming language's constructor method.

```
class vendingMachines():
    # PRIVATE Items : Array [0:3] OF foodItems
    # PRIVATE moneyIn : Real
    def __init__(self, item1, item2, item3, item4):
        self.__items = []
        self.__items.append(item1)
        self.__items.append(item2)
        self.__items.append(item3)
        self.__items.append(item4)
        self.__moneyIn = 0
```

(ii) The method `checkValid()` takes the food item code as a parameter. It checks the code against each element in `items` and returns:

- `-1` if the code is not valid
- `-2` if the code is valid, but the `moneyIn` is less than the cost of the item
- the index of the item, if the code is valid and the `moneyIn` is greater than or equal to the cost of the item.

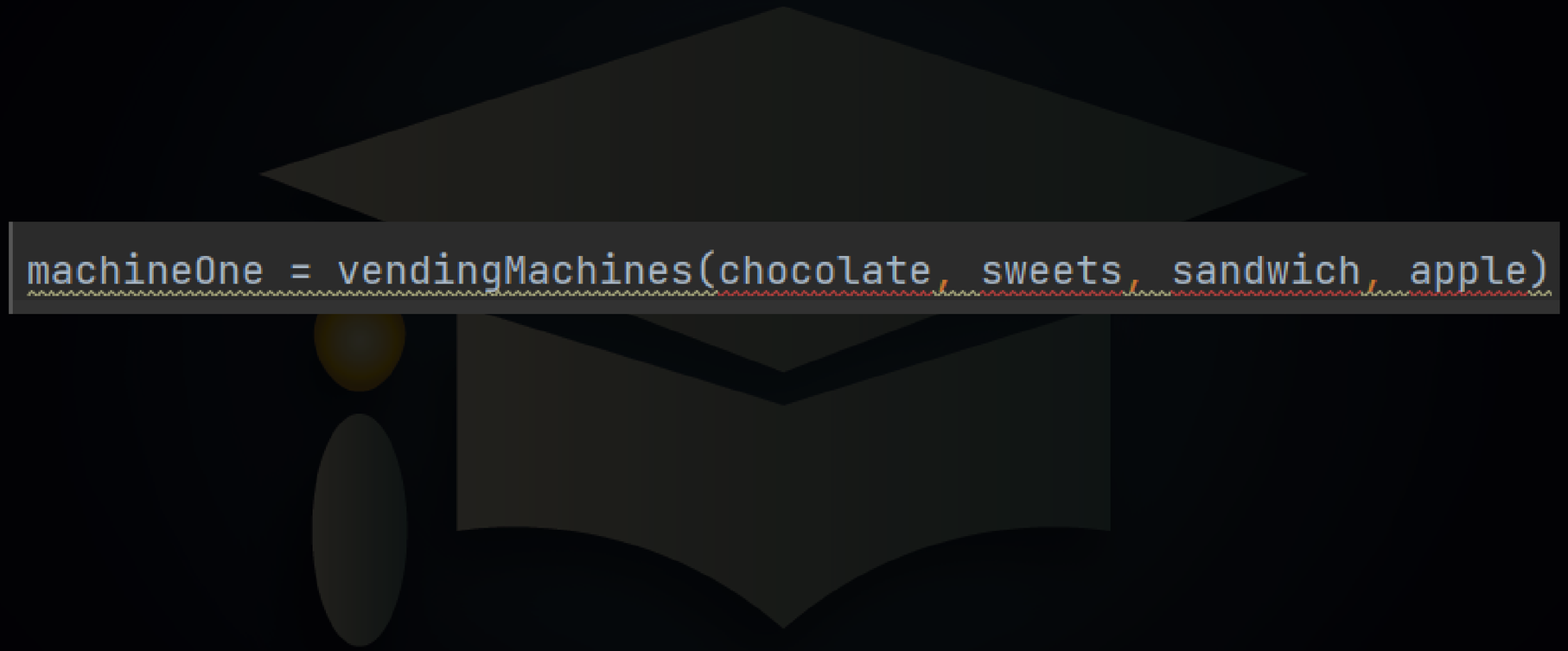
Write **program code** for the method `checkValid()`.

```
def checkValid(self, code):
    for x in range(0, 4):
        if self.__items[x].getCode == code:
            if self.__items[x].getCost <= self.__moneyIn:
                return x
            else:
                return -2
    return -1
```

(iii) Four objects of type `foodItem` are declared with the identifiers:

```
chocolate, sweets, sandwich, apple
```

Write **program code** to declare an instance of `vendingMachine` with the identifier `machineOne` and the objects: `chocolate, sweets, sandwich, apple`.


machineOne = vendingMachines(chocolate, sweets, sandwich, apple)

```
class vendingMachines():
    # PRIVATE Items : Array [0:3] OF foodItems
    # PRIVATE moneyIn : Real
    def __init__(self, item1, item2, item3, item4):
        self.__items = []
        self.__items.append(item1)
        self.__items.append(item2)
        self.__items.append(item3)
        self.__items.append(item4)
        self.__moneyIn = 0


    def checkValid(self, code):
        for x in range(0, 4):
            if self.__items[x].getCode == code:
                if self.__items[x].getCost <= self.__moneyIn:
                    return x
                else:
                    return -2
        return -1

machineOne = vendingMachines(chocolate, sweets, sandwich, apple)
```

Practice Question

3 Ejaz is creating a program that will allow the user to create quizzes. He is using object-oriented programming (OOP).

There are two classes: `QuestionClass` and `QuizClass`.

The class attributes and methods are in the following tables. All attributes are declared as private.

QuestionClass	
`Question : STRING`	`// stores the question`
`Answer : STRING`	`// stores the correct answer`
`Difficulty : INTEGER`	`// stores the difficulty as an integer` `// from 0 (easy) to 10 (hard)`
`Constructor(QuestionP, AnswerP, DifficultyP)`	`// creates an instance of QuestionClass` `// sets the attributes to the parameter values`
`GetQuestion()`	`// returns the question`
`GetDifficulty()`	`// returns the difficulty level`
`GetAnswer()`	`// returns the answer`

QuizClass	
Questions : ARRAY[0:19] OF QuestionClass	// stores maximum 20 questions of // type QuestionClass
NumberOfQuestions : INTEGER	// stores the number of questions // in this quiz

Constructor()	// creates an instance of
	// QuizClass
	// initialises NumberOfQuestions
	// to 0

AddQuestion()	// adds the parameter question to
	// the array
	// increments NumberOfQuestions

GetQuestion()	// returns the next question to be // asked
CheckAnswer()	// takes an answer as a parameter // and returns TRUE if correct

(a) Write **program code** to define the class `QuizClass`. You are only required to write code for the attribute declarations and constructor.

If you are writing in Python, include attribute declarations using comments.

Use your programming language's constructor method.

```
class QuizClass():
    # PRIVATE Questions : Array [0:19] OF QuestionClass
    # PRIVATE NumberofQuestions : Integers
    def __init__(self):
        self.__NumberofQuestions = 0
        self.__Questions = []
        for x in range(20):
            self.__Questions.append(QuestionClass("", "", 0))
```

(b) The `QuizClass` method `AddQuestion()` takes a question object as a parameter and stores it in the next available location in the array `Questions`. It returns `TRUE` if it is successfully stored, and `FALSE` otherwise.

Write **program code** for the method `AddQuestion()`.

```
def AddQuestion(self, QuestionObject):

    if self.__NumberofQuestions < 20:
        self.__Questions[self.__NumberofQuestions] = QuestionObject
        self.__NumberofQuestions = self.__NumberofQuestions + 1
        return True
    else:
        return False
```

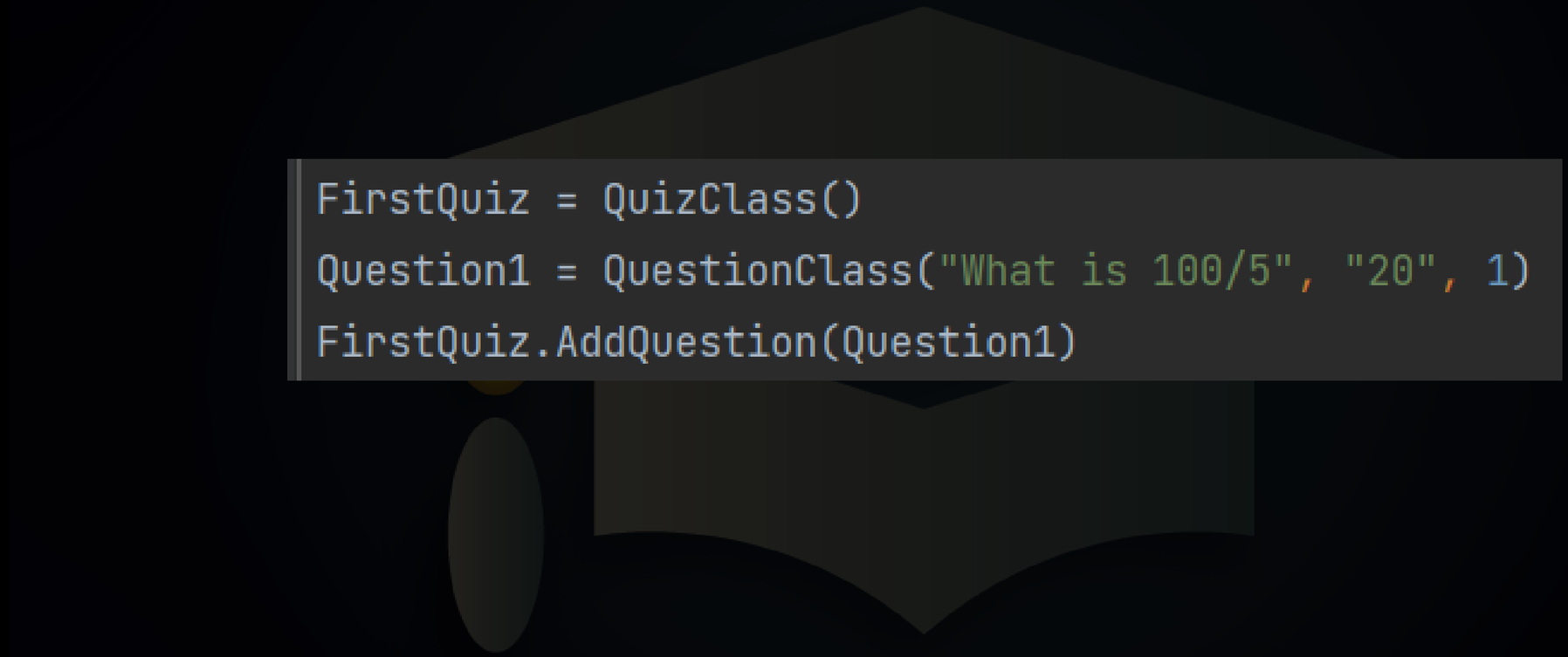
(c) The first quiz is created with the identifier `FirstQuiz`.

The first question in this quiz is: “What is 100 / 5 ?”.

The answer is “20” and the difficulty level is 1.

Write **program code** to:

- declare an instance of `QuizClass` with the identifier `FirstQuiz`
- declare an instance of `QuestionClass` with the identifier `Question1`
- add `Question1` to the array in `FirstQuiz` using `AddQuestion()`.



```
FirstQuiz = QuizClass()
Question1 = QuestionClass("What is 100/5", "20", 1)
FirstQuiz.AddQuestion(Question1)
```

```
class foodItem():
    #PRIVATE name : STRING
    #PRIVATE code : STRING
    #PRIVATE cost : REAL
    def __init__(self, nameP, codeP, costP):
        self.__name = nameP
        self.__code = codeP
        self.__cost = costP

    def getCode(self):
        return self.__code

    def getCost(self):
        return self.__cost

    def getName(self):
        return self.__name
```

```
class vendingMachine():
    #PRIVATE items : ARRAY [0 : 3] OF foodItem
    #PRIVATE moneyIn : REAL

    def __init__(self, item1, item2, item3, item4):
        self.__items = [0] * 4
        self.__items[0] = item1
        self.__items[1] = item2
        self.__items[2] = item3
        self.__items[3] = item4

        self.__moneyIn = 0
```

```
    def checkValid(self, code):
        for x in range(4):
            if self.__items[x].getCode() == code:
                if self.__moneyIn >= self.__items[x].getCost():
                    return x
                else:
                    return -2
        return -1

    def InsertMoney(self, paisa):
        self.__moneyIn = paisa

chocolate = foodItem("You", "1024", 150.5)
sweets = foodItem("Lab e Shiri", "2017", 12.2)
sandwich = foodItem("BBQ", "2132", 32.2)
apple = foodItem("red", "4321", 32.1)

machineOne = vendingMachine(chocolate, sweets, sandwich, apple)

machineOne.InsertMoney(100.5)
temp = machineOne.checkValid("2132")
print(temp)
```