Deep Dive into Python

Pre-requisites

Hope you have gone through the self-learning content for this session on the PRISM portal.

By the End of this Session:

- Learn the concept of Object-Oriented Programming.
- Create object blueprints in Python using Classes.
- Understand the various components of a class – methods and attributes.
- Use Inheritance to establish hierarchy and code reusability within Python classes.

What have we learned so far?

- How to work with functions in Python.
- Positional vs. Keyword Arguments.
- Combining multiple functions together into Modules.
- Importing and using modules in Python.
- Use of Lambda functions.
- Difference between Named and Anonymous Functions.
- Performing file manipulations using File Handling.
- Catching and handling errors in Python using Error Handling.

Poll Time

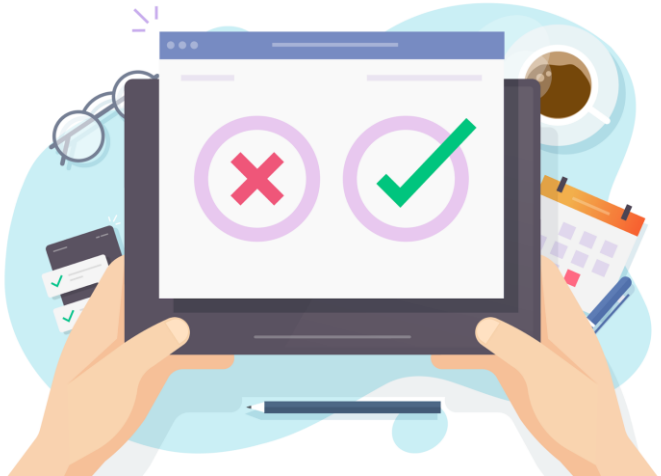
Q. Which of the following options represents the different file handling modes in Python?

a. read, write, delete

b. open, close, read, write

c. read, write, append

d. read, write, execute



Poll Time

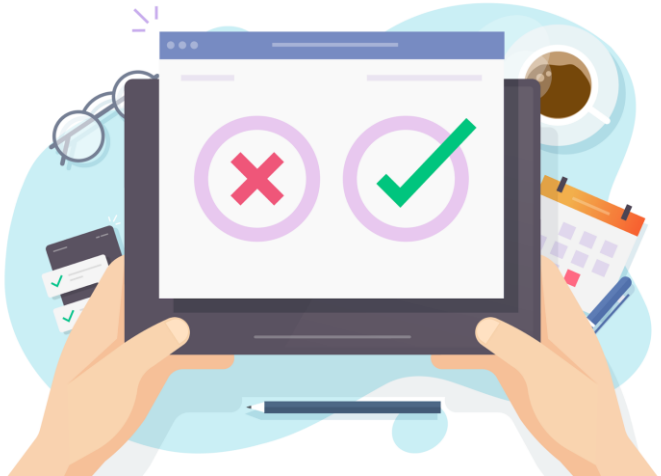
Q. Which of the following options represents the different file handling modes in Python?

a. read, write, delete

b. open, close, read, write

c. read, write, append

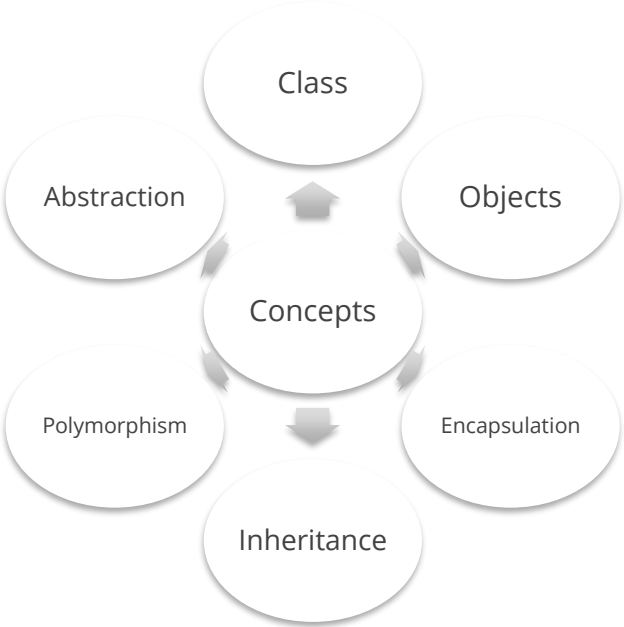
d. read, write, execute

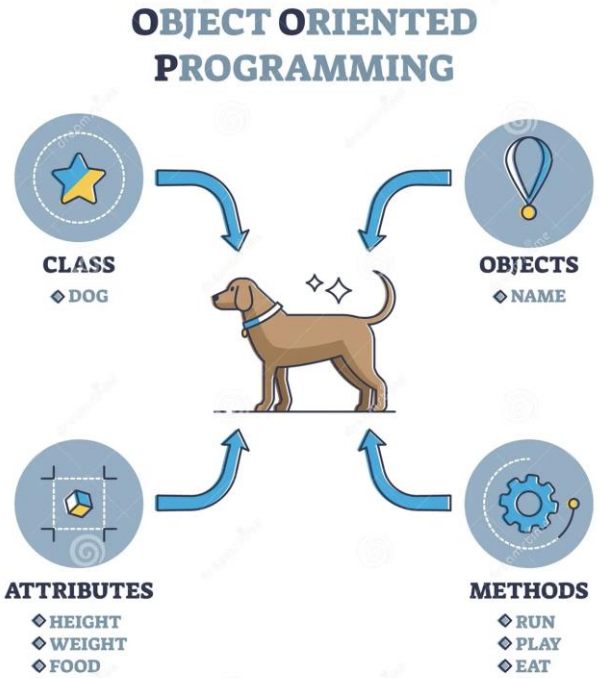


Introduction to OOP and Classes

Introduction to Object-Oriented Programming

- Paradigm in programming.
- Organizes code around objects and data.





Understanding Classes and Objects in Python

What is a Class?

- A blueprint or template for creating objects.
- Defines the structure and behavior of objects.

Defining a Class

python Copy code

```python
class MyClass:
    # Class attributes and methods go here
```

Understanding Classes and Objects in Python

What is an Object?

- An instance of a class.
- Represents a real-world entity or concept.

Creating Objects

python Copy code

```python
# Instantiating objects
obj1 = MyClass()
obj2 = MyClass()
```

Pop Quiz

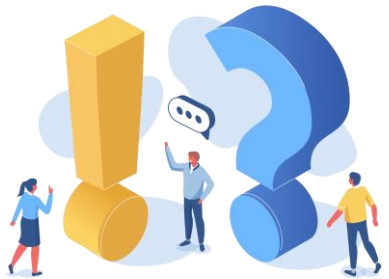
Q. Which of the following statements is true regarding classes in Python?

a. Classes are used to define loops in Python programs

b. Classes are data types used to store numerical values

c. Classes provide a blueprint for creating objects with attributes and methods

d. Classes can only have attributes and cannot have methods



Pop Quiz

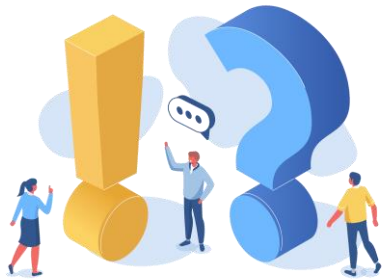
Q. Which of the following statements is true regarding classes in Python?

a. Classes are used to define loops in Python programs

b. Classes are data types used to store numerical values

c. Classes provide a blueprint for creating objects with attributes and methods

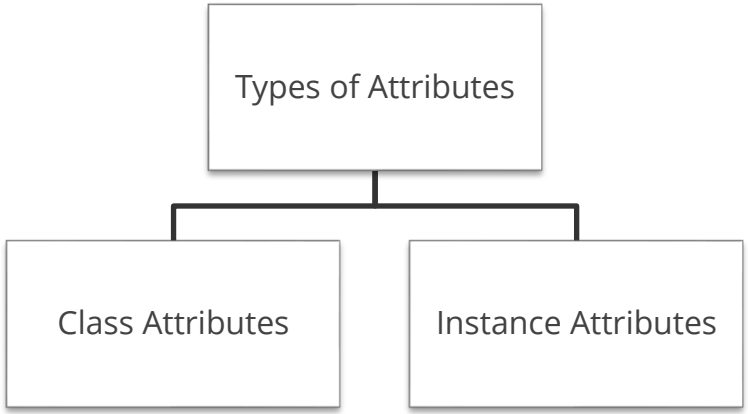
d. Classes can only have attributes and cannot have methods



Variables and Methods in Classes

Class Variables vs. Instance Variables

In Python classes, variables are known as "attributes" or "class variables." They are used to store data that belongs to the class and is shared among all instances (objects) of that class.



Accessing and Modifying Class Variables

Class attributes are defined at the class level and are shared among all instances of the class. They are accessed using the class name and remain the same for all objects of that class.

python Copy code

```python
class Circle:
    # Class attribute
    pi = 3.14

    def __init__(self, radius):
        # Instance attribute
        self.radius = radius
```

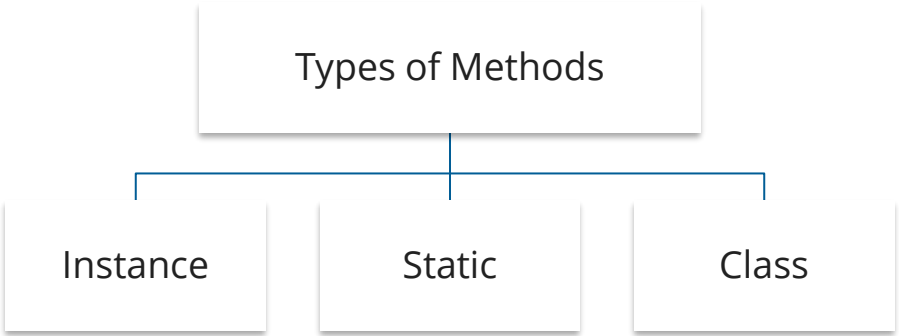
python Copy code

```python
# Accessing class attribute
print(Circle.pi)
```

Methods in Classes

What are methods?

- Functions defined inside a class.
- Operate on class and instance data.



Creating and Calling Methods

python Copy code

```python
class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2
```

python Copy code

```python
# Creating object
circle = Circle(5)

# Calling instance method
result = circle.area()
```

Poll Time

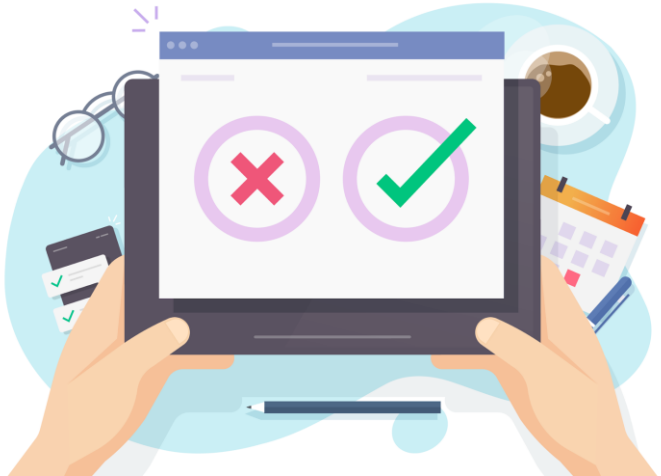
Q. Which of the following statements regarding attributes of classes in Python is true?

a. Class attributes are specific to each instance of the class

b. Instance attributes are specific to each instance of the class

c. Class attributes are defined inside instance methods of the class

d. Instance attributes represent data shared among all objects of the class



Poll Time

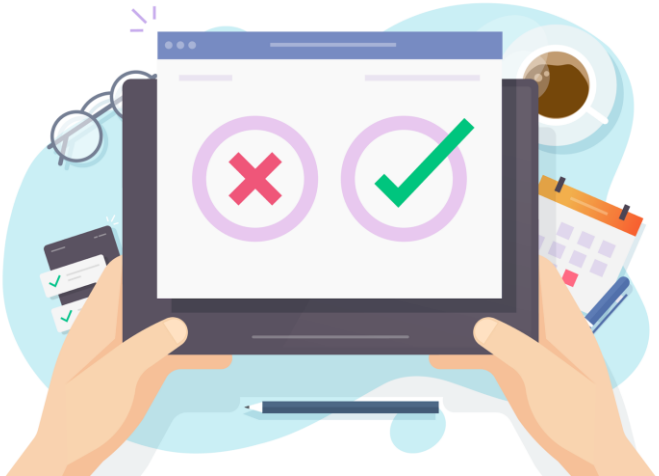
Q. Which of the following statements regarding attributes of classes in Python is true?

a. Class attributes are specific to each instance of the class

b. Instance attributes are specific to each instance of the class

c. Class attributes are defined inside instance methods of the class

d. Instance attributes represent data shared among all objects of the class




Q&A

Constructor, Destructor, and Special Methods

Constructor Method: init()

- A special method in classes.
- Automatically called when an object is created.
- Initializes object attributes.

Example: Constructor in Class

python Copy code

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Creating Objects with the Constructor

python Copy code

```python
# Creating instances of Person class
person1 = Person("Alice", 30)
person2 = Person("Bob", 25)
```

Destructor Method: del()

- A special method in classes.
- Automatically called when an object is destroyed.
- Performs cleanup operations before the object is removed from memory.

Example: Destructor in Class

python Copy code

```python
class FileHandler:
    def __init__(self, filename):
        self.filename = filename

    def open_file(self):
        self.file = open(self.filename, 'r')

    def read_data(self):
        return self.file.read()

    def __del__(self):
        self.file.close()
```

Special Methods (Magic/Dunder Methods)

__str__	• String representation
__add__	• Addition operator overloading
__len__	• Length of an object
__eq__	• Equality operator overloading
__lt__, __gt__	• Comparison operators overloading

Poll Time

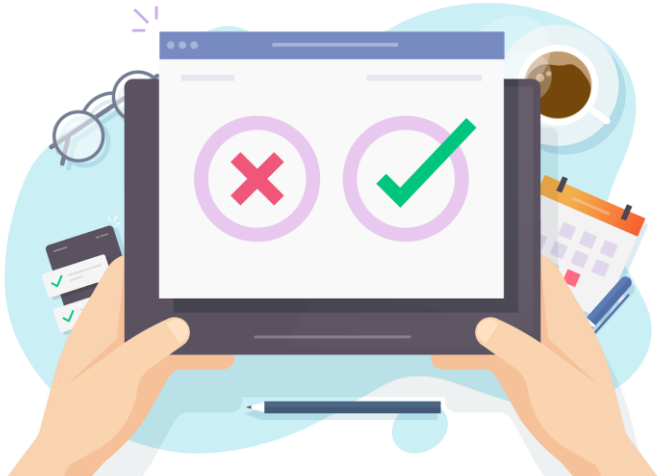
Q. Which of the following statements is true regarding the constructor in Python classes?

a. The constructor is used to initialize object attributes and is called when an object is destroyed

b. The constructor is automatically called when an object is created and is used to initialize object attributes

c. The constructor is used to perform cleanup operations before the object is removed from memory

d. The constructor is used to modify the default behavior of classes and objects



Poll Time

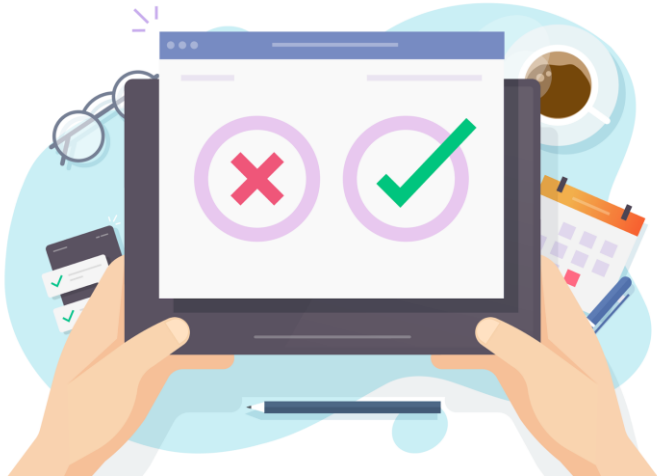
Q. Which of the following statements is true regarding the constructor in Python classes?

a. The constructor is used to initialize object attributes and is called when an object is destroyed

b. The constructor is automatically called when an object is created and is used to initialize object attributes

c. The constructor is used to perform cleanup operations before the object is removed from memory

d. The constructor is used to modify the default behavior of classes and objects



Inheritance and Subclasses

Inheritance: Creating Subclasses

- OOP concept that allows creating new classes from existing ones.
- Inherits attributes and methods of the parent class.

Terminology

Superclass / Parent Class

- The class being inherited from

Subclass / Child Class

- The new class being created

Single Inheritance and Superclasses

python Copy code

```python
# Parent Class
class Animal:
    def __init__(self, name):
        self.name = name

    def make_sound(self, sound):
        return f"{self.name} makes {sound}"

# Child Class (Inherits from Animal)
class Dog(Animal):
    def __init__(self, name, breed):
        # Call parent class constructor
        super().__init__(name)
        self.breed = breed
```

python Copy code

```python
# Creating instance of Dog
dog1 = Dog("Buddy")

# Calling overridden method
result = dog1.make_sound("loudly")
# Output: "Buddy barks loudly"
```

Overriding Methods in Subclasses

python Copy code

```python
class Cat(Animal):
    def make_sound(self, sound):
        # Override parent class method
        return f"{self.name} says {sound} meow"
```

python Copy code

```python
# Creating instance of Cat
cat1 = Cat("Whiskers")

# Calling overridden method
result = cat1.make_sound("softly")
# Output: "Whiskers says softly meow"
```

Pop Quiz

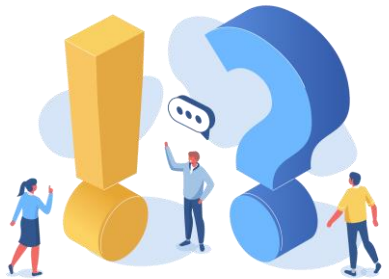
Q. Which of the following statements is true regarding class inheritance in Python?

a. Class inheritance is used to create new instances of a class

b. Inheritance allows one class to inherit attributes and methods from another class

c. Parent classes can access attributes and methods of their subclasses

d. Subclasses cannot override methods inherited from the parent class



Pop Quiz

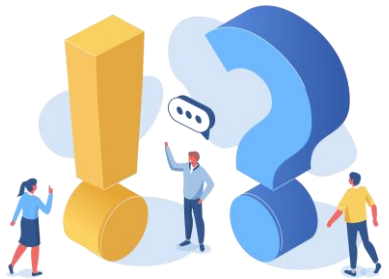
Q. Which of the following statements is true regarding class inheritance in Python?

a. Class inheritance is used to create new instances of a class

b. Inheritance allows one class to inherit attributes and methods from another class

c. Parent classes can access attributes and methods of their subclasses

d. Subclasses cannot override methods inherited from the parent class




Q&A

Summary

- Classes help us in creating blueprints for Python objects.
- Classes contains attributes and methods.
- Attributes can be classified into class and instance attributes.
- Methods can be classified into class, static, and instance.
- Inheritance allows one class to inherit attributes and methods from another class.

Activity 1

Pre-requisites:

- Python 3.x – preferably Python 3.8
- Jupyter Notebook

Scenario:

Imagine you are building a program to manage a zoo. The zoo has different types of animals, and you want to use Python classes to represent and manage them.

- Create a base class called **Animal** with the following attributes and methods:
    1. Attributes: name (string), species (string), age (integer), sound (string)
    2. Method: make_sound() - Print the sound the animal makes.
- Create **Two** subclasses that inherit from the **Animal** class:
    1. Elephant: Additional attribute - trunk_length (float)
    2. Penguin: Additional attribute - can_swim (boolean)
- Implement the **__init__()** constructor in each subclass to initialize the attributes inherited from the **Animal** class and their subclass-specific attributes.
- Implement the **make_sound()** method in each subclass to display a unique sound for each animal type.

Next Session:
Polymorphism, Encapsulation, and Abstraction in Python

THANK YOU!

Please complete your assessments and review the self-learning content for this session on the **PRISM** portal.

knowledgehut
upGrad

Polymorphism, Encapsulation, and Abstraction in Python

Pre-requisites

Hope you have gone through the self-learning content for this session on the PRISM portal.

By the End of this Session:

- Learn essential concepts of OOP.
- Use Polymorphism to override functions and operators.
- Use Encapsulation to restrict the access to data within classes.
- Use Abstraction to represent essential features while hiding details.

Recap

Poll Time

Q. Which of the following is an example of a special method in Python?

a. def add_numbers(a, b): return a + b

b. def __init__(self, name): self.name = name

c. def multiply_numbers(a, b): return a * b

d. def display_name(self): print(self.name)

Poll Time

Q. Which of the following is an example of a special method in Python?

a. def add_numbers(a, b): return a + b

b. def __init__(self, name): self.name = name

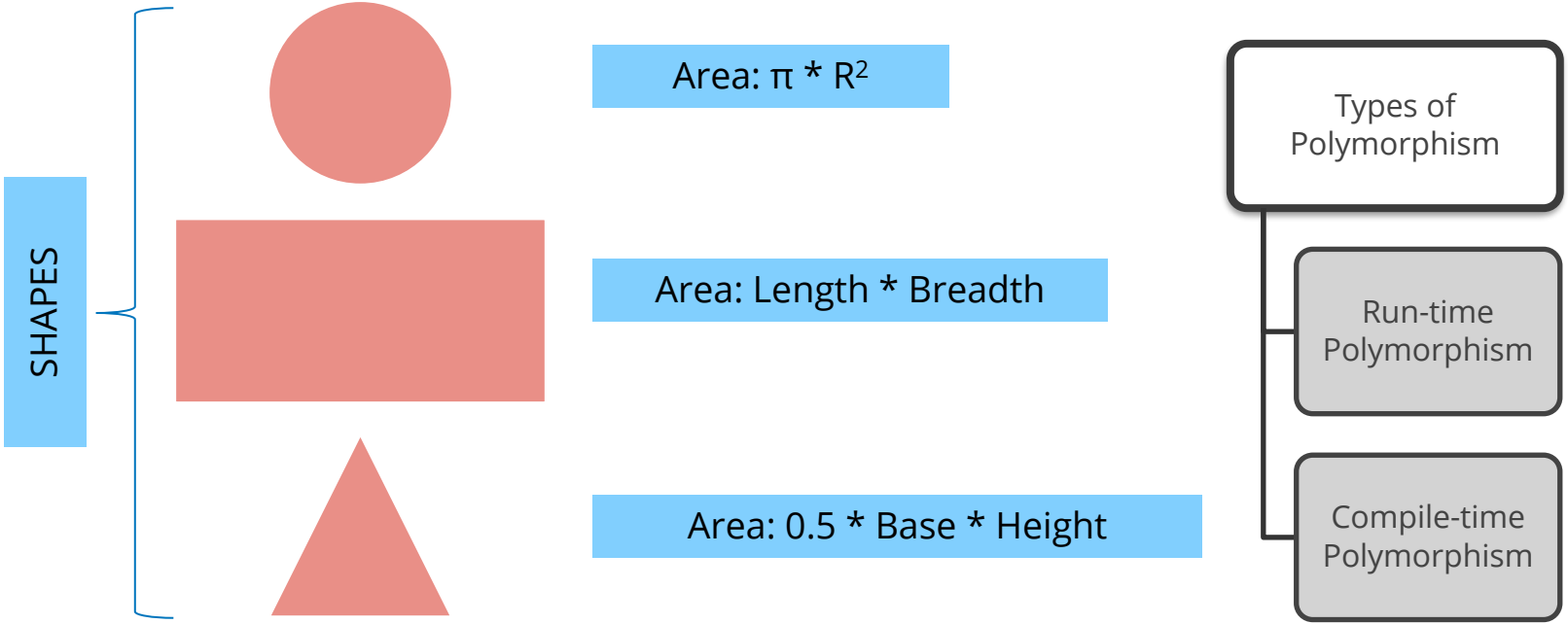
c. def multiply_numbers(a, b): return a * b

d. def display_name(self): print(self.name)

Polymorphism

Introduction to Polymorphism

- A core principle of Object-Oriented Programming (OOP).
- Allows objects of different classes to be treated as objects of a common superclass.



Pop Quiz

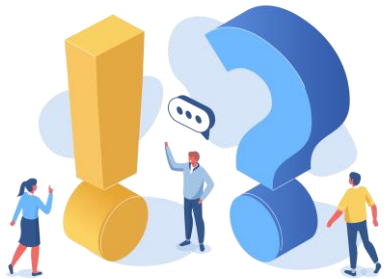
Q. Which of the following statements is true regarding polymorphism in functions in Python?

a. Polymorphism in functions is achieved through function overloading

b. Polymorphism in functions allows a single function to have multiple names

c. Polymorphism in functions allows a function to accept different types of arguments

d. Polymorphism in functions is applicable only to built-in functions



Pop Quiz

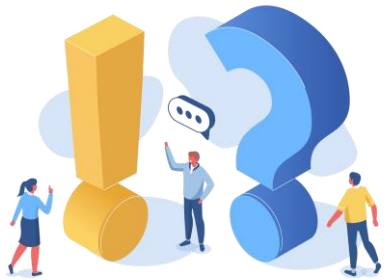
Q. Which of the following statements is true regarding polymorphism in functions in Python?

a. Polymorphism in functions is achieved through function overloading

b. Polymorphism in functions allows a single function to have multiple names

c. Polymorphism in functions allows a function to accept different types of arguments

d. Polymorphism in functions is applicable only to built-in functions



Compile-Time Polymorphism

- Achieved through method overloading.
- Multiple methods with the same name but different parameters in the same class.

Example: Compile-time Polymorphism

python Copy code

```python
class MathOperations:
    def add(self, a, b):
        return a + b

    def add(self, a, b, c):
        return a + b + c
```

Run-Time Polymorphism

- Achieved through method overriding.
- Subclasses provide a specific implementation of a method defined in the superclass.

Example: Run-Time Polymorphism

python Copy code

```python
class Shape:
    def area(self):
        pass   # Abstract method
```

```python
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2
```

```python
class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side ** 2
```

```python
# Polymorphic behavior
def calculate_area(shape_obj):
    return shape_obj.area()
```

Polymorphism in Operators

- Common operators (e.g., +, -, *, /) exhibit polymorphic behavior.
- They can perform different operations based on the data type of operands.

Example: Addition Operator

python Copy code

```python
num1 = 10
num2 = 20
result1 = num1 + num2   # Integer addition

str1 = "Hello"
str2 = "World"
result2 = str1 + str2   # String concatenation
```

Example: Multiplication Operator

python Copy code

```python
num = 5
result = num * 3    # Integer multiplication

str = "Hello "
result_str = str * 3   # String repetition
```

Polymorphism in Built-in Functions

- In Python many built-in can work with different data types and structures.
- They can accept various arguments and provide different functionalities based on the data types or structures they receive.

Example: len() Function

```python
string_length = len("Hello")
list_length = len([1, 2, 3, 4, 5])
tuple_length = len((1, 2, 3))
dict_length = len({"a": 1, "b": 2})
```

Example: sum() function

```python
sum_of_list = sum([1, 2, 3, 4, 5])
sum_of_tuple = sum((1, 2, 3))
sum_of_set = sum({10, 20, 30, 40, 50})
sum_of_range = sum(range(1, 6))
```

Poll Time

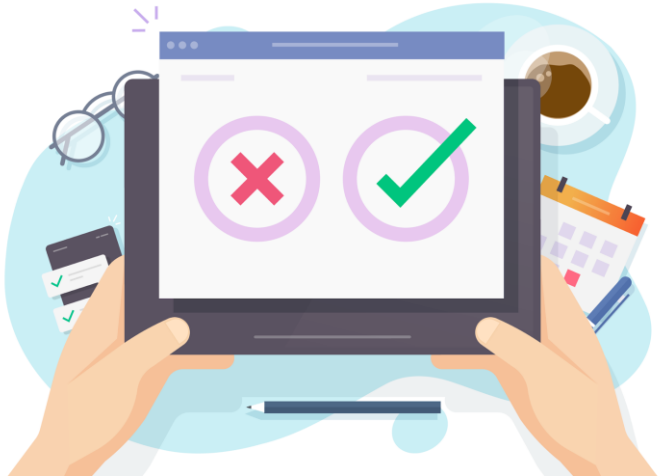
Q. Which of the following built-in functions in Python exhibits polymorphic behavior by accepting different data types and structures?

a. Input()

b. Type()

c. Len()

d. Range()



Poll Time

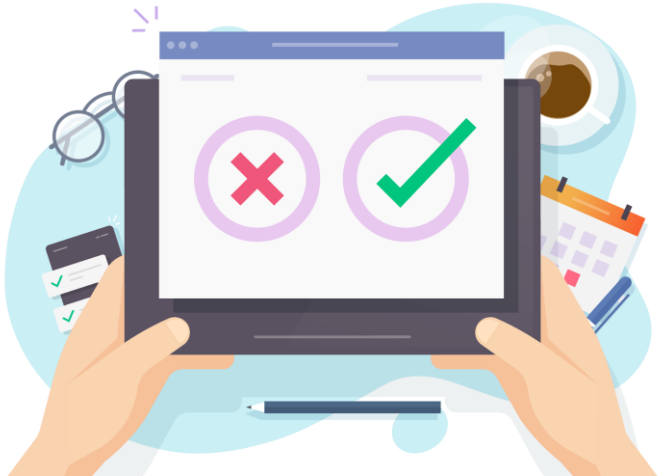
Q. Which of the following built-in functions in Python exhibits polymorphic behavior by accepting different data types and structures?

a. Input()

b. Type()

c. Len()

d. Range()




Q&A

Encapsulation and Abstraction

Introduction to Encapsulation

- Encapsulation is the bundling of data and methods that operate on that data within a single unit, known as a "class.“
- It helps hide the internal details and implementation of a class from the outside world.

Access Modifiers

Public

- Accessible from anywhere outside the class. (No restrictions on access)

Private

- Accessible within the class and its subclasses. (Use single underscore: _variable)

Protected

- Accessible only within the class. (Use double underscore: __variable)

Encapsulation in Python

python Copy code

```python
class BankAccount:
    def __init__(self, account_number, balance):
        self.__account_number = account_number    # Private attribute
        self.__balance = balance                  # Private attribute

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Insufficient balance!")

    def get_balance(self):
        return self.__balance
```

```python
# Creating an instance of BankAccount
account = BankAccount("123456", 1000)

# Performing encapsulated operations
account.deposit(500)
account.withdraw(200)
balance = account.get_balance()
```

Pop Quiz

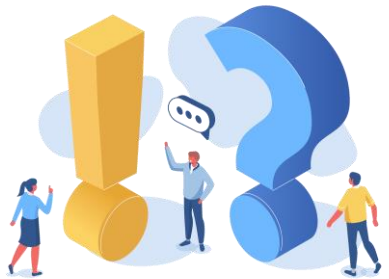
Q. Which of the following access control levels in Python encapsulation provides the highest level of restriction on class members?

a. Public

b. Protected

c. Private

d. Hidden



Pop Quiz

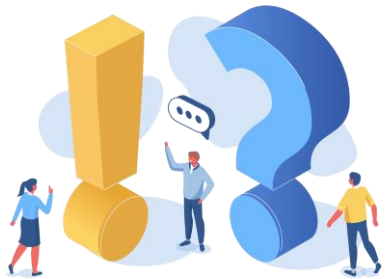
Q. Which of the following access control levels in Python encapsulation provides the highest level of restriction on class members?

a. Public

b. Protected

c. Private

d. Hidden



Abstraction in Python

- Abstraction focuses on representing essential features while hiding unnecessary details.
- Python supports abstraction through abstract classes and interfaces.

Abstract Classes and Interfaces

- Abstract classes cannot be instantiated directly.
- They serve as blueprints for other classes, defining common attributes and methods.

python Copy code

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def perimeter(self):
        pass
```

python Copy code

```python
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2

    def perimeter(self):
        return 2 * 3.14 * self.radius
```

Importance of Abstraction

- Focus on essential aspects, reducing complexity.
- Encourages code reusability and modularity.
- Allows for flexible implementation and future changes.

Poll Time

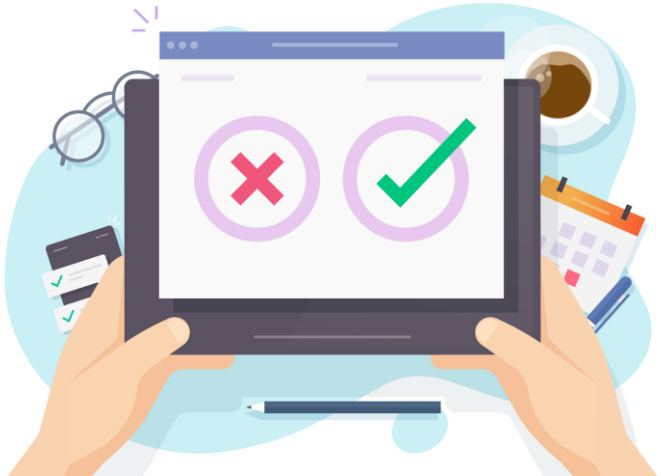
Q. Which of the following statements is true regarding abstraction in Python?

a. Abstraction hides the internal implementation of a class from the outside world

b. Abstraction can only be achieved through interfaces in Python

c. Abstract classes can be instantiated directly to create objects

d. Abstraction is not essential in Object-Oriented Programming



Poll Time

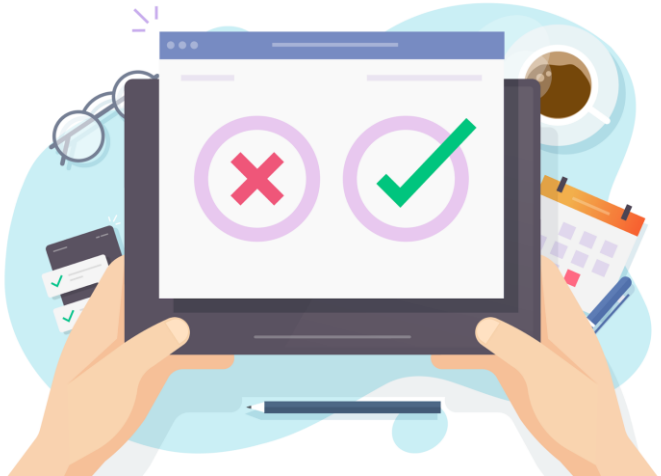
Q. Which of the following statements is true regarding abstraction in Python?

a. **Abstraction hides the internal implementation of a class from the outside world**

b. Abstraction can only be achieved through interfaces in Python

c. Abstract classes can be instantiated directly to create objects

d. Abstraction is not essential in Object-Oriented Programming




Q&A

Summary

- Inheritance, Polymorphism, Encapsulation, and Abstraction are the 4 key concepts in OOP.
- Polymorphism allows objects of different classes to be treated as objects of a common superclass.
- Encapsulation is the concept of bundling data and methods within a class to control access and prevent direct modification of data from outside the class.
- Abstraction hides the internal implementation of a class from the outside world.

Activity 1

Pre-requisites:

- Python 3.x – preferably Python 3.8
- Jupyter Notebook

Scenario:

Practice Polymorphism and Abstraction in Python by creating a program that demonstrates the concept of polymorphism using different shapes.

Instructions:

- Define an abstract class called **Shape** that contains two abstract methods: **area()** and **perimeter()**. These methods will represent the common functionalities of all shapes.
- Implement three different shapes (e.g., **Circle, Square, and Triangle**) as subclasses of the **Shape** class.
- Each subclass should override the **area()** and **perimeter()** methods to calculate the area and perimeter specific to that shape.

Activity 2

Pre-requisites:

- Python 3.x – preferably Python 3.8
- Jupyter Notebook

Scenario:

Use the classes defined in the last activity and perform the below instructions.

Instructions:

- Create a function called **print_shape_details()** that takes a shape object as an argument and prints its area and perimeter using polymorphism.
- In the **main()** function, create instances of each shape and call the **print_shape_details()** function with each shape object.

Session Feedback

Next Session:
Understanding and Summarizing Data – Descriptive Statistics

THANK YOU!

Please complete your assessments and review the self-learning content for this session on the **PRISM** portal.

knowledgehut
upGrad