### issubclass() **and** isinstance() **in Python OOP**

Both **issubclass()** and **isinstance()** are **built-in Python functions** used for checking relationships between classes and objects.

## ****1.**** issubclass()

### ****Purpose:**** Checks if a class is a ****subclass**** of another class.

### ****Syntax:**** issubclass(subclass, superclass)

* Returns True if subclass is derived from superclass.
* Returns False otherwise.

**Example:**

class Animal:

 pass

class Dog(Animal):

 pass

class Cat:

 pass

print(issubclass(Dog, Animal)) # True (Dog is a subclass of Animal)

print(issubclass(Cat, Animal)) # False (Cat is not a subclass of Animal)

print(issubclass(Dog, object)) # True (All classes in Python inherit from object)

## ****2.**** isinstance()

### ****Purpose:**** Checks if an ****object**** is an ****instance**** of a specific class (or its subclass).

### ****Syntax:**** isinstance(object, class\_or\_tuple)

* Returns **True** if object is an **instance** of the given class or **any of the classes in a tuple**.
* Returns **False** otherwise.

**Example:**

dog = Dog()

cat = Cat()

print(isinstance(dog, Dog)) # True (dog is an instance of Dog)

print(isinstance(dog, Animal)) # True (Dog is a subclass of Animal)

print(isinstance(cat, Animal)) # False (Cat is not a subclass of Animal)

print(isinstance(42, int)) # True (42 is an integer)

print(isinstance(3.14, (int, float))) # True (3.14 is either an int or float)

## ****Key Differences:****



**Python OOP: getattr(), setattr(), hasattr(), and delattr() Built-in Functions**

These four built-in functions help in dynamically accessing and modifying **attributes of objects (or classes)** in Python.

**1.** **getattr()**

**Purpose:** Retrieves the value of an attribute from an object.

**Syntax:** getattr(object, attribute\_name [, default\_value])

* If the attribute exists, returns its value.
* If the attribute **does not exist**, it raises an **AttributeError**, unless a **default value** is provided.

### ****Example:****

### class Person:

###  def \_\_init\_\_(self, name, age):

###  self.name = name

###  self.age = age

### p = Person("Alice", 30)

### print(getattr(p, "name")) # Alice (Gets the attribute 'name')

### print(getattr(p, "age")) # 30 (Gets the attribute 'age')

### print(getattr(p, "gender", "Not Specified")) # "Not Specified" (Default value if attribute is missing)

**2. setattr()**

**Purpose:** Sets or modifies an attribute of an object dynamically.

**Syntax:** setattr(object, attribute\_name, value)

* If the attribute exists, it **updates** it.
* If the attribute does **not exist**, it **creates** a new one.

**Example:**

setattr(p, "age", 35) # Modifies existing attribute 'age'

print(p.age) # Output: 35

setattr(p, "gender", "Female") # Creates a new attribute 'gender'

print(p.gender) # Output: Female

**3. hasattr()**

**Purpose:** Checks if an object has a specific attribute.

**Syntax:** hasattr(object, attribute\_name)

* Returns True if the attribute exists.
* Returns False if the attribute does not exist.

### ****Example:****

print(hasattr(p, "name")) # True (p has 'name' attribute)

print(hasattr(p, "salary")) # False ('salary' does not exist)

**4. delattr()**

**Purpose:** Deletes an attribute from an object.

**Syntax: delattr(object, attribute\_name)**

* If the attribute exists, it gets deleted.
* If the attribute does not exist, it raises an AttributeError.

### ****Example:****

delattr(p, "age") # Deletes 'age' attribute

print(hasattr(p, "age")) # False (age is deleted)

# print(p.age) # AttributeError: 'Person' object has no attribute 'age'

**Comparison Table**



**Real-World Use Case**

Dynamically setting and getting attributes can be useful when working with **JSON data, configurations, or metaprogramming**.

**Example:** Using These Functions in a Dynamic Configuration

class Config:

 pass

config = Config()

**# Setting attributes dynamically**

setattr(config, "debug", True)

setattr(config, "version", "1.0.0")

**# Retrieving attributes dynamically**

print(getattr(config, "debug")) # True

print(getattr(config, "version")) # "1.0.0"

**# Checking existence**

print(hasattr(config, "debug")) # True

**# Deleting an attribute**

delattr(config, "debug")

print(hasattr(config, "debug")) # ❌ False

### ****Special Attributes in Python Classes:**** \_\_doc\_\_****,**** \_\_dict\_\_****,**** \_\_name\_\_

Python provides several **special attributes** (also called **dunder attributes**) that store useful metadata about a class or object. Here’s what \_\_doc\_\_, \_\_dict\_\_, and \_\_name\_\_ do:

## ****1.**** \_\_doc\_\_ ****→ Class or Function Documentation String****

### ****Purpose:**** Stores the ****docstring**** of a class, function, or module. If no docstring is provided, it returns None.

### ****Example:****

class Person:

 """This class represents a person."""

 def \_\_init\_\_(self, name):

 self.name = name

print(Person.\_\_doc\_\_) # "This class represents a person."

* **Use Case:** Helps in generating documentation, debugging, and introspection.

## ****2.**** \_\_dict\_\_ ****→ Dictionary of Attributes****

### ****Purpose:**** Stores all attributes of an object or class in a dictionary format.

### ****Example (Class Level**** \_\_dict\_\_****):****

class Person:

 species = "Human" # Class attribute

 def \_\_init\_\_(self, name, age):

 self.name = name

 self.age = age

print(Person.\_\_dict\_\_)

**Output (Truncated for readability):**

{'\_\_module\_\_': '\_\_main\_\_', 'species': 'Human', '\_\_init\_\_': <function Person.\_\_init\_\_ at 0x7f3c...>}

* This shows **all class attributes and methods**.

**Example (Instance Level \_\_dict\_\_):**

p = Person("Alice", 30)

print(p.\_\_dict\_\_)

**Output:** {'name': 'Alice', 'age': 30}

* This contains **only instance-specific attributes**, not class attributes.
* **Use Case:** Useful for **serialization (e.g., JSON), debugging, and inspecting objects**.

**3. \_\_name\_\_**

**Purpose:** Returns the **name of a class, function, or module**.

**Example** (For Classes & Functions):

class Person:

 pass

def greet():

 pass

print(Person.\_\_name\_\_) # "Person"

print(greet.\_\_name\_\_) # "greet**"**

**Example (For Modules):**

import math

print(math.\_\_name\_\_) # "math"

* **Use Case:** Used in **introspection, logging, and debugging**.

**Summary Table**



These attributes make Python **more introspective and dynamic**!