



# Class and Instance Variables

In this lesson, we'll learn about the concepts of instance variables and class variables, and how to use class variables correctly.

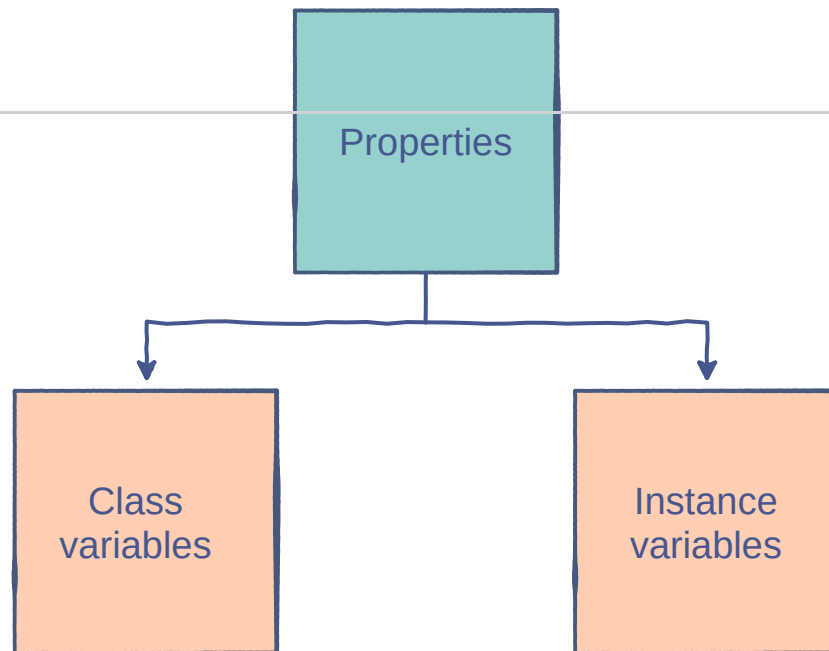
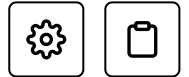
## We'll cover the following



- Definitions
  - Class Variables
  - Instance Variables
- Defining Class Variables and Instance Variables
  - Wrong Use of Class Variables
- Using Class Variables Smartly
  - Explanation

In Python, properties can be defined into two parts:

- **class variables**
- **instance variables**



## Definitions #

### Class Variables #

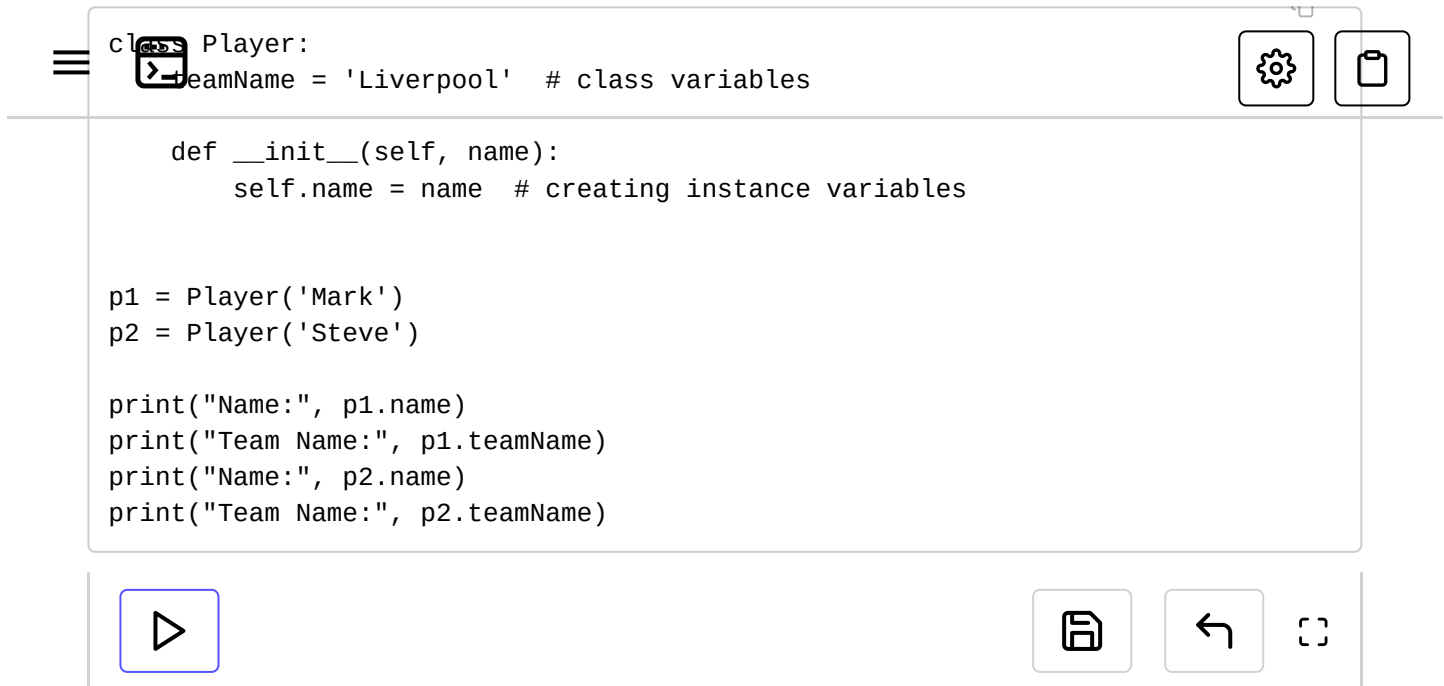
The **class variables** are shared by all instances or objects of the classes. A change in the class variable will change the value of that property in *all the objects* of the class.

### Instance Variables #

The **instance variables** are unique to each instance or object of the class. A change in the instance variable will change the value of the property in that *specific object only*.

## Defining Class Variables and Instance Variables #

Class variables are defined **outside** the initializer and instance variables are defined inside the initializer.



```
class Player:
    teamName = 'Liverpool' # class variables

    def __init__(self, name):
        self.name = name # creating instance variables

p1 = Player('Mark')
p2 = Player('Steve')

print("Name:", p1.name)
print("Team Name:", p1.teamName)
print("Name:", p2.name)
print("Team Name:", p2.teamName)
```

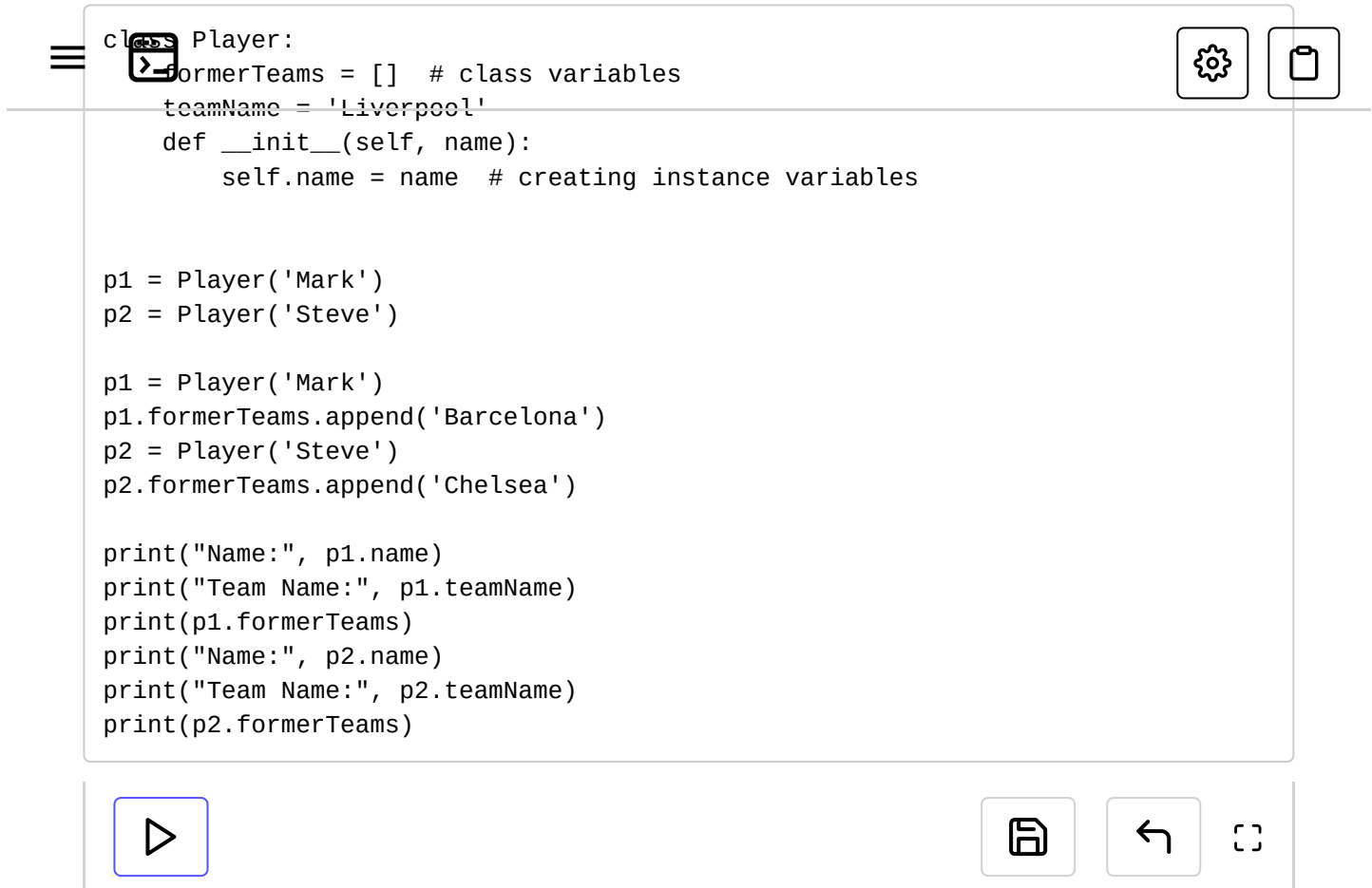
The image shows a Python IDE window with a menu icon on the left and settings and clipboard icons on the right. The code is written in a light blue font on a white background. Below the code editor, there is a toolbar with a play button, a save icon, a back arrow, and a full-screen icon.

In **line 2**, we have created a class variable and in **line 5**, we have created an instance variable.

## Wrong Use of Class Variables #

It is imperative to use class variables properly since they are shared by all the class objects and can be modified using any one of them. Below is an example of wrongful use of class variables:





```
class Player:
    formerTeams = [] # class variables
    teamName = 'Liverpool'
    def __init__(self, name):
        self.name = name # creating instance variables

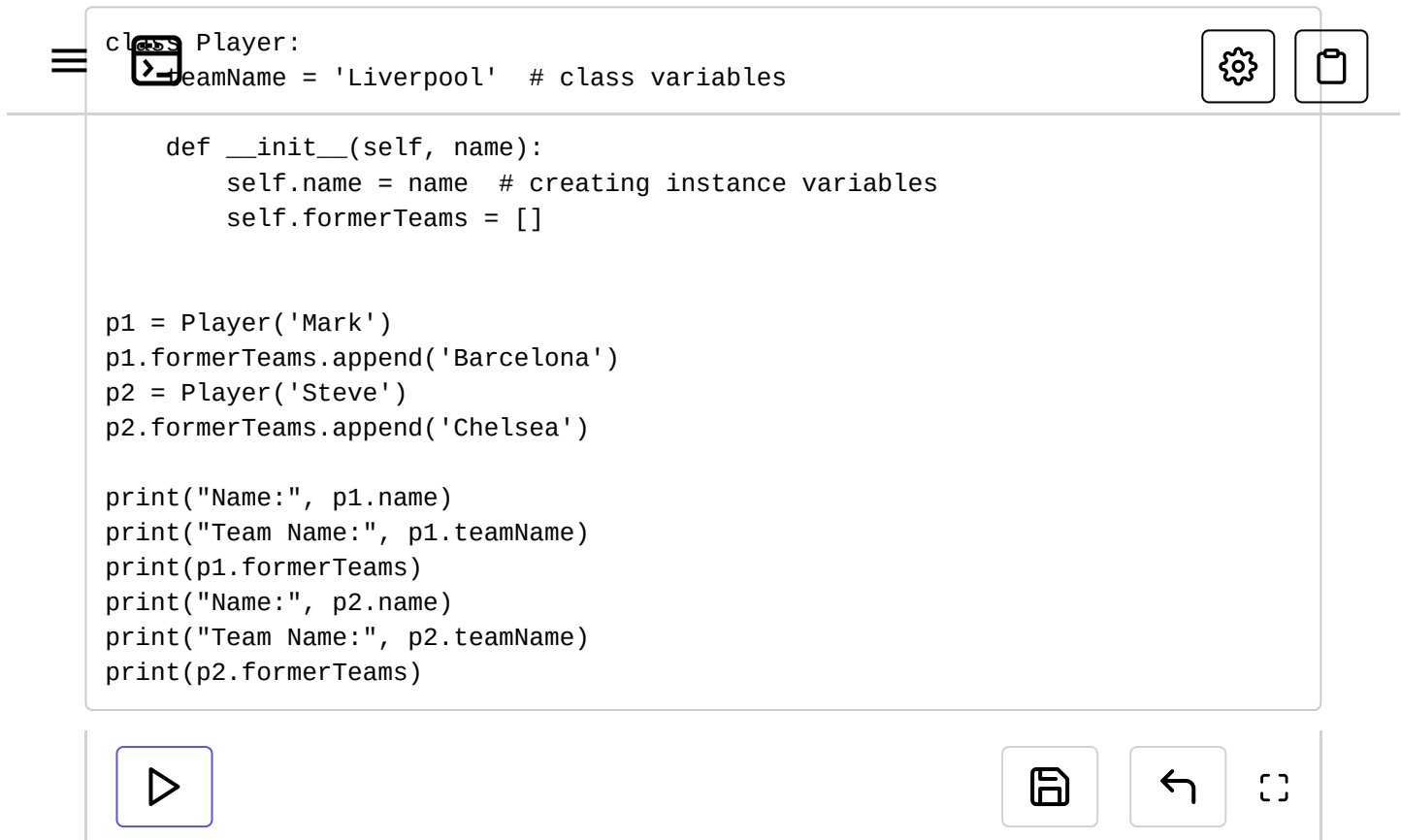
p1 = Player('Mark')
p2 = Player('Steve')

p1 = Player('Mark')
p1.formerTeams.append('Barcelona')
p2 = Player('Steve')
p2.formerTeams.append('Chelsea')

print("Name:", p1.name)
print("Team Name:", p1.teamName)
print(p1.formerTeams)
print("Name:", p2.name)
print("Team Name:", p2.teamName)
print(p2.formerTeams)
```

In the example above, while the instance variable `name` is unique for each and every object of the `Player` class, the class variable, `formerTeams`, can be accessed by any object of the class and is updated throughout. We are storing all players currently playing for the same team, but each player in the team may have played for different former teams. To avoid this issue, the correct implementation of the example above will be the following:





```
class Player:
    teamName = 'Liverpool' # class variables

    def __init__(self, name):
        self.name = name # creating instance variables
        self.formerTeams = []

p1 = Player('Mark')
p1.formerTeams.append('Barcelona')
p2 = Player('Steve')
p2.formerTeams.append('Chelsea')

print("Name:", p1.name)
print("Team Name:", p1.teamName)
print(p1.formerTeams)
print("Name:", p2.name)
print("Team Name:", p2.teamName)
print(p2.formerTeams)
```

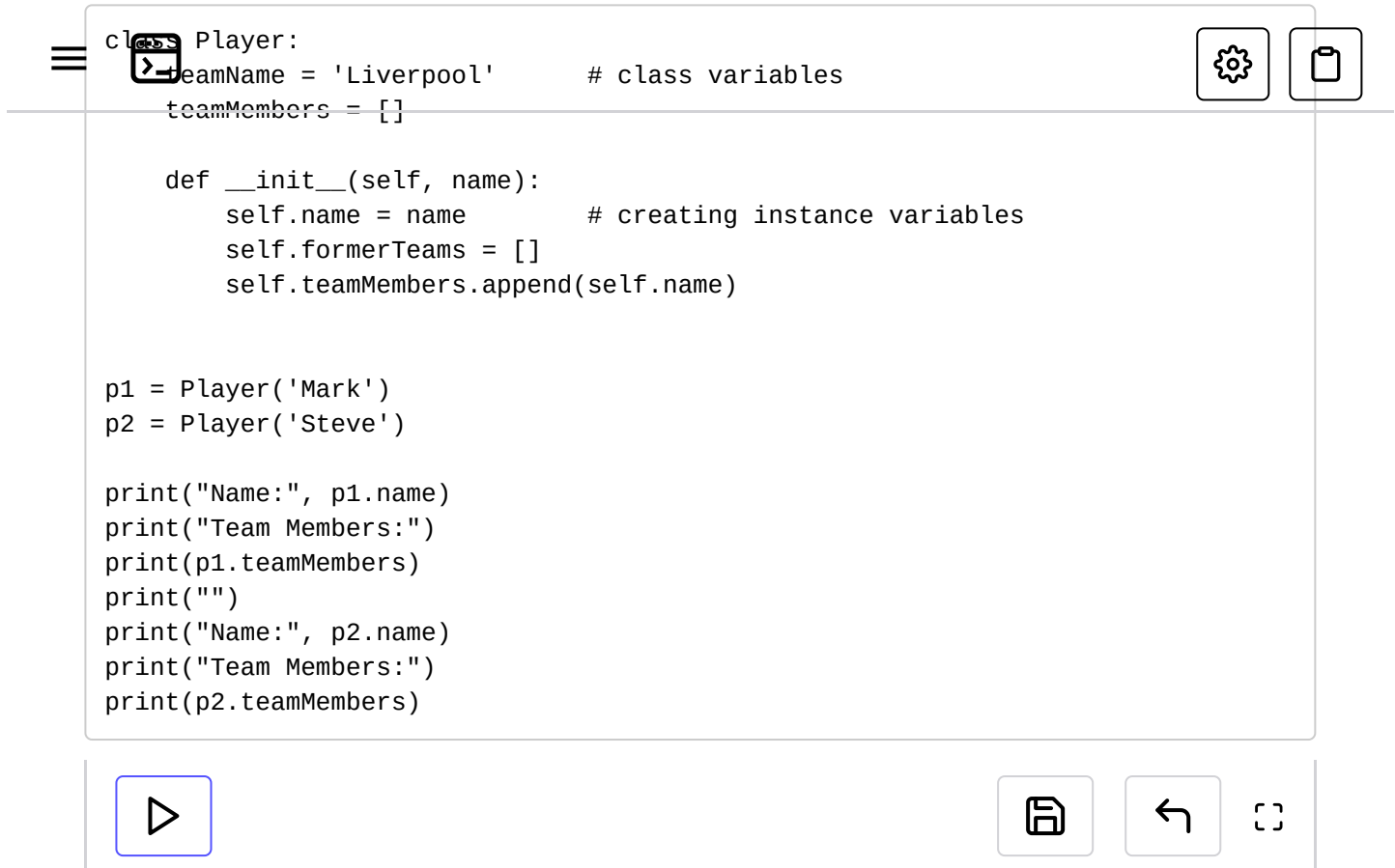
The image shows a Python IDE window with a menu icon on the top left, a settings gear icon, and a clipboard icon on the top right. The code defines a `Player` class with a class variable `teamName` and an `__init__` method. It then creates two instances, `p1` and `p2`, and prints their attributes and the `formerTeams` list.

Now the property `formerTeams` is unique for each `Player` class object and can only be accessed by that unique object.

## Using Class Variables Smartly #

Class variables are useful when implementing properties that should be common and accessible to all class objects. Let's see an example of this:





```
class Player:
    teamName = 'Liverpool'      # class variables
    teamMembers = []

    def __init__(self, name):
        self.name = name        # creating instance variables
        self.formerTeams = []
        self.teamMembers.append(self.name)

p1 = Player('Mark')
p2 = Player('Steve')

print("Name:", p1.name)
print("Team Members:")
print(p1.teamMembers)
print("")
print("Name:", p2.name)
print("Team Members:")
print(p2.teamMembers)
```

The code defines a `Player` class with class variables `teamName` and `teamMembers`. The `__init__` method initializes instance variables `name`, `formerTeams`, and appends `self.name` to `self.teamMembers`. Two instances, `p1` and `p2`, are created with names 'Mark' and 'Steve' respectively. The code then prints the name and team members for both instances.

## Explanation #

- In the example above, we've defined a class variable `teamMembers`, which is a list that will be shared by all the objects of the class `Player`.
- This list, `teamMembers`, will contain names of all the instances created of the `Player` class.
- As you can see in **line 8**, whenever a new object is created, its name is appended in `teamMembers`.
- In **line 16** and **line 20**, we can see that `teamMembers` is accessed by `p1` and `p2` respectively and both produce the same output.

In the next lesson, we'll learn about implementing methods in a class.