Instances in Python

✦ An **instance** is a specific, concrete object created from a class.
✦ Even if they are created from the same class blueprint, instances have their own, individual set of attributes and behaviors.
✦ They store data as **instance attributes**, which represent the object's state.
✦ Each instance maintains its own attribute values independently; changes to one instance do not affect others.
  - Example: If a Player class defines a *score* instance attribute, a player_1 instance could have a score of 71 while a player_2 instance could have a score of 110. The values of their *score* attributes are independent. If one of them changes, the other one will not be affected.

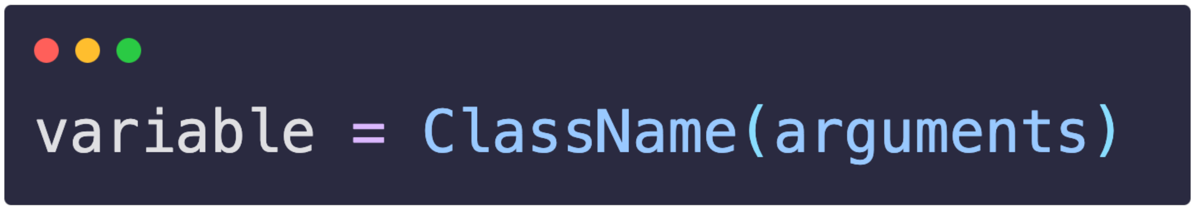
✦ The values of instance attributes are initially assigned in the __init__() method, which runs automatically when the object is created.
✦ You can change the data stored in instance attributes.
✦ When you make an instance, store it in a variable to use it later in your program.
✦ *self* is used as a general way to refer to the instance of a class within the class itself.
✦ With *self*, you can have access to the attributes and methods of the current instance.

The __init__() Method:

✦ Commonly referred to as a "constructor" (although it's a bit different from traditional constructors in other programming languages, like C++ or Java, because it's not responsible for allocating memory for the object).
✦ It's called automatically when an object of the class is created, to initialize its attributes.
✦ To define this special method in the class body, write the *def* keyword followed by __init__() (with two underscores before and after the word *init*), a list of formal parameters within the parentheses, and finally a colon at the end of the first line (See Fig. 3).
✦ The formal parameters of this method allow you to pass arguments when you create an instance of the class. This way, you can customize their initial values in the program (See Fig. 2 and Fig. 4).
✦ Within the __init__() method, you can set initial values for instance attributes, to ensure each new instance starts with those specified values. (See Fig. 5).
✦ You can set default values for the parameters of __init__() by writing parameter=value in the list of formal parameters.

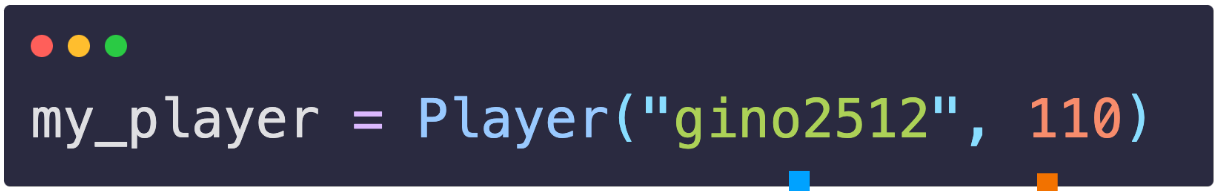
✦ Parameters with default arguments must be located at the end of the list of parameters. Otherwise, an error will be thrown.

Fig. 1 Create an Instance (General Syntax)



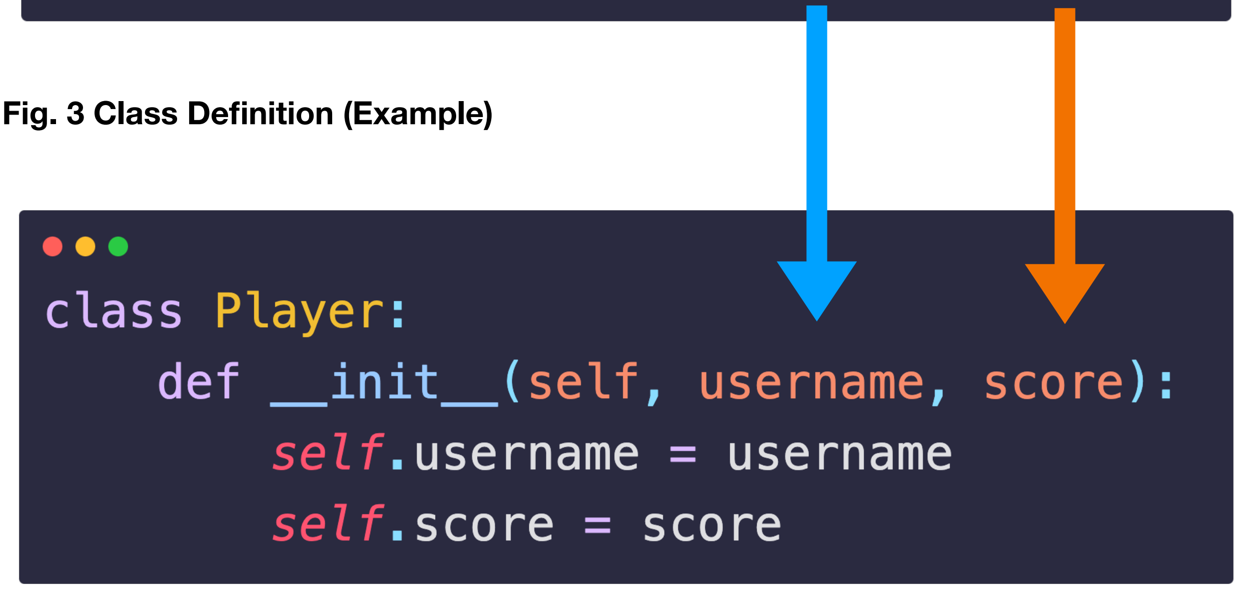
```python
variable = ClassName(arguments)
```

Fig. 2 Create an Instance (Example)



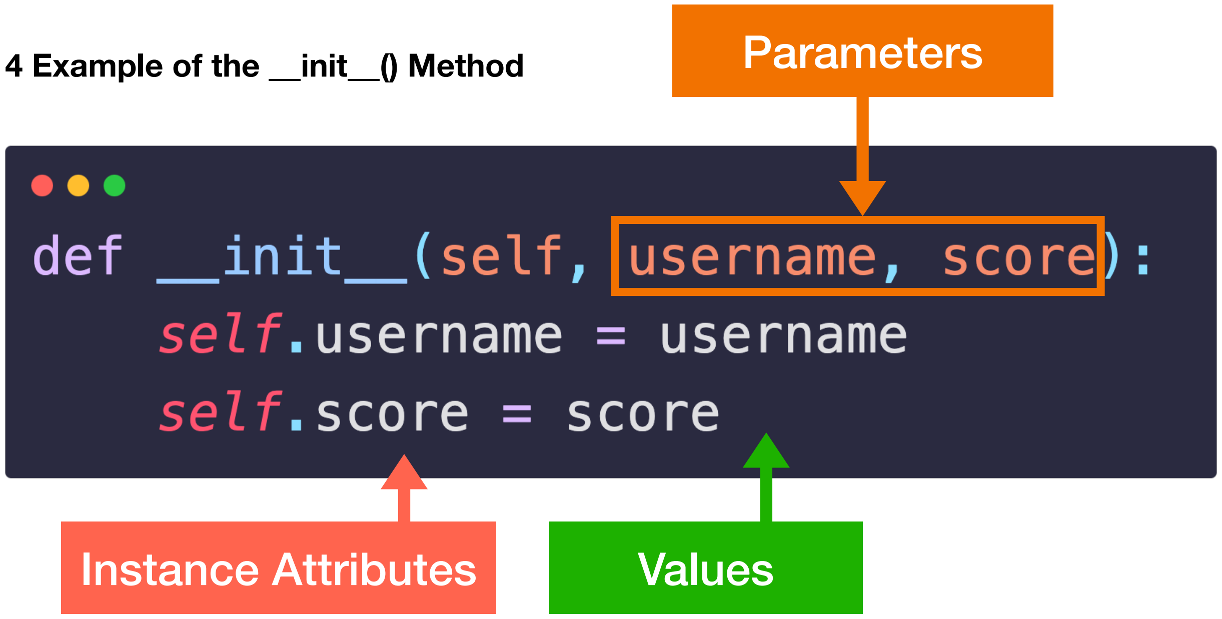
```python
my_player = Player("gino2512", 110)
```

Fig. 3 Class Definition (Example)



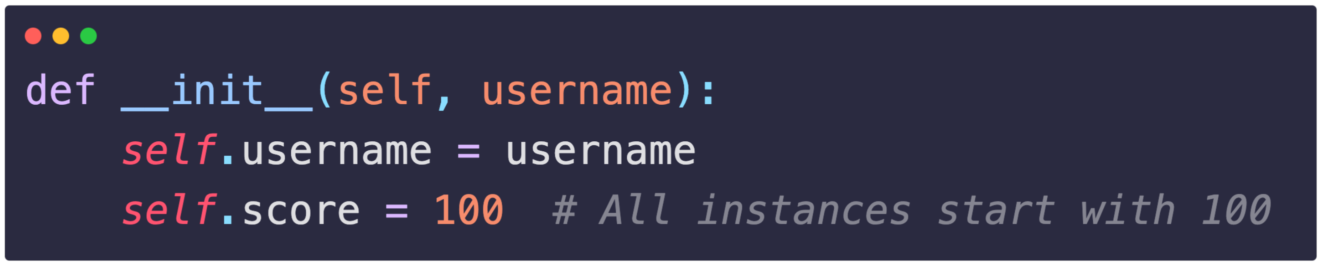
```python
class Player:
    def __init__(self, username, score):
        self.username = username
        self.score = score
```

Fig. 4 Example of the __init__() Method



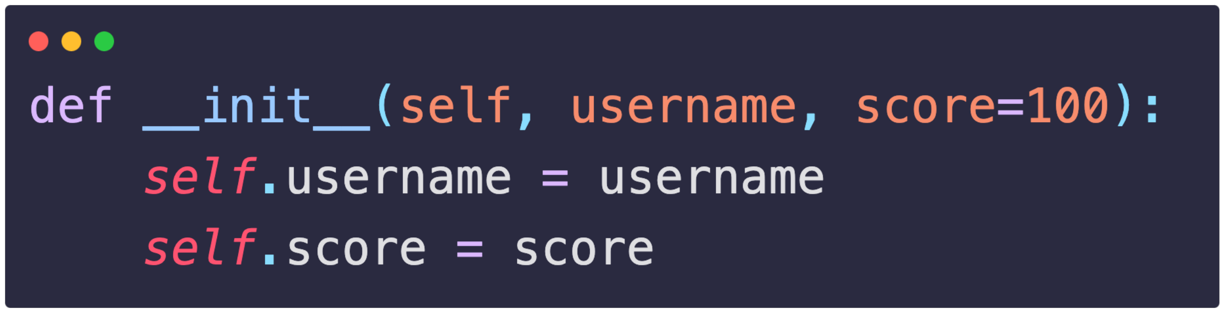
```python
def __init__(self, username, score):
    self.username = username
    self.score = score
```

Fig. 5 Setting Default Initial Values for Instance Attributes (Example)



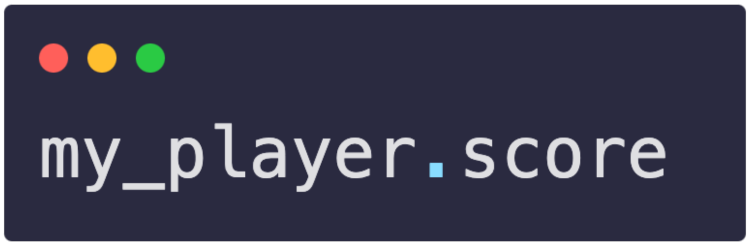
```python
def __init__(self, username):
    self.username = username
    self.score = 100  # All instances start with 100
```

Fig. 6 Default Argument (Example)



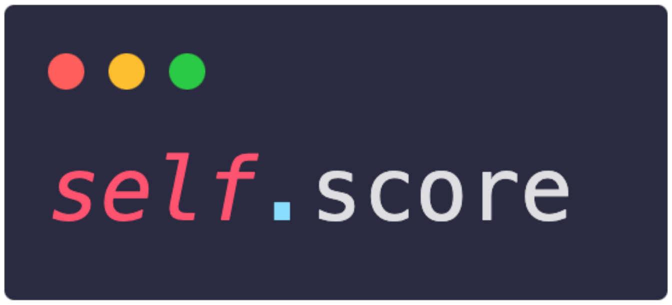
```python
def __init__(self, username, score=100):
    self.username = username
    self.score = score
```

Fig. 7 Access Instance Attribute (Outside the Class, with my_player defined) (Example)



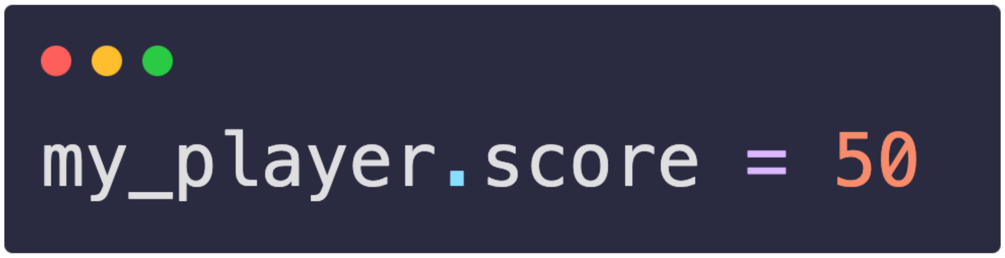
```python
my_player.score
```

Fig. 8 Access Instance Attribute (Inside the Class) (Example)



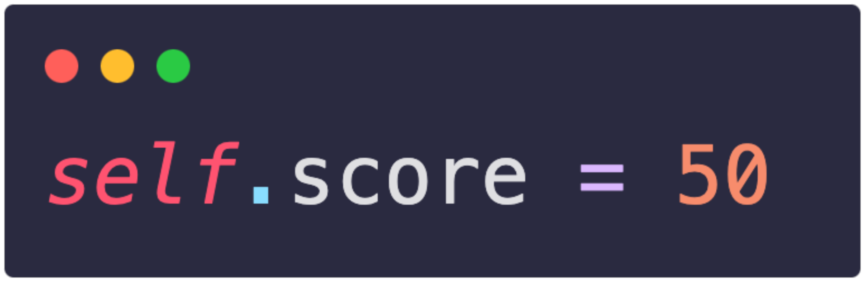
```python
self.score
```

Fig. 9 Modify Instance Attribute (Outside the Class) (Example)



```python
my_player.score = 50
```

Fig. 10 Modify Instance Attribute (Inside the Class) (Example)



```python
self.score = 50
```