Getters, Setters, and Properties in Python

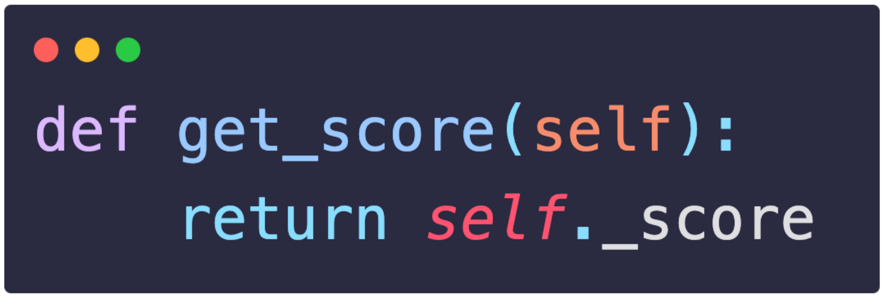
<u>Getters and Setters</u>

- ✦ **Getters** are methods used to get the value of an object's attribute.
- ✦ A common naming convention for getters is **get_attribute**. For example: get_name.
- ✦ **Setters** are methods used to modify the value of an object's attribute.
- ✦ A common naming convention for setters is **set_attribute**. For example: set_name.
- ✦ Setters usually take one value as argument, the value that will be assigned to the attribute.
- ✦ Getters and setters are very closely related to the principle of encapsulation, since they provide controlled access to an object's attributes.
- ✦ They serve as intermediaries to avoid accessing the data directly, which could result in unexpected or harmful modifications.
- ✦ They are commonly used to:
    - Validate input data before setting or updating the value of an attribute.
    - Perform additional actions when an attribute is accessed or modified, such as calculations.
    - Hide the internal representation of data by using intermediary methods.

<u>Properties</u>

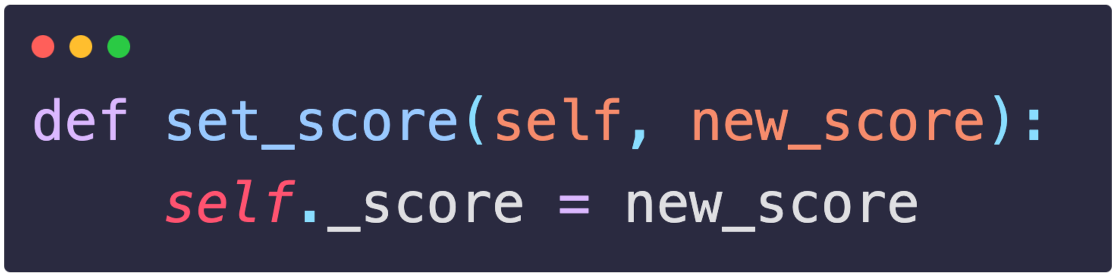
- ✦ Properties are the "pythonic" way of working with getters and setters in Python.
- ✦ Properties make code more readable because they allow you to access and modify attributes as if they were regular attributes, while still providing the benefits of using getters and setters as intermediaries to protect the data.
- ✦ By using properties, you can change the internal implementation of a class without breaking existing code that works with the class.
- ✦ When you use properties, you don't need to call getters and setters explicitly, but they do run behind the scenes.
- ✦ Properties can be used to create calculated attributes. For example: the area of a circle.
- ✦ There are two alternative syntaxes for defining properties: property() and @property.
- ✦ The @property decorator syntax is usually recommended because it's more compact, easier to read, and doesn't add new identifiers to the namespace.

Fig. 1 Getter (Example)



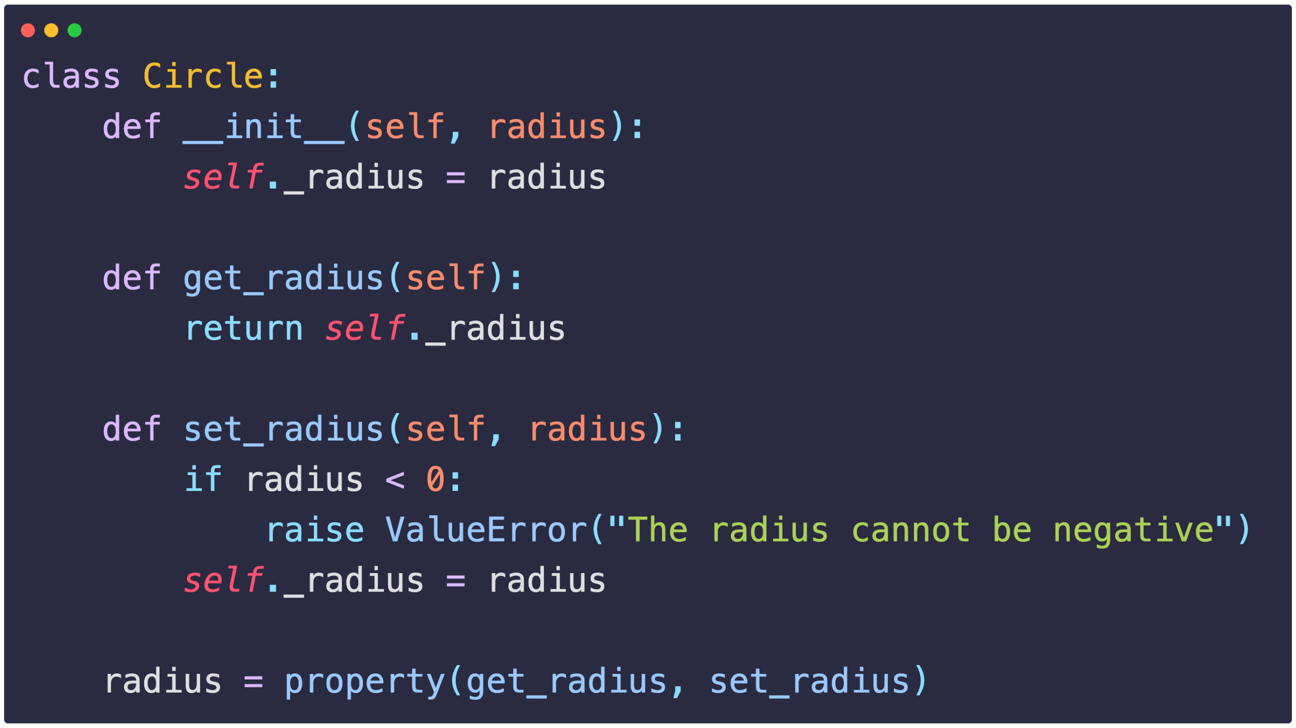
```python
def get_score(self):
    return self._score
```

Fig. 2 Setter (Example)



```python
def set_score(self, new_score):
    self._score = new_score
```

Fig. 3 Property using the property() syntax (Example)



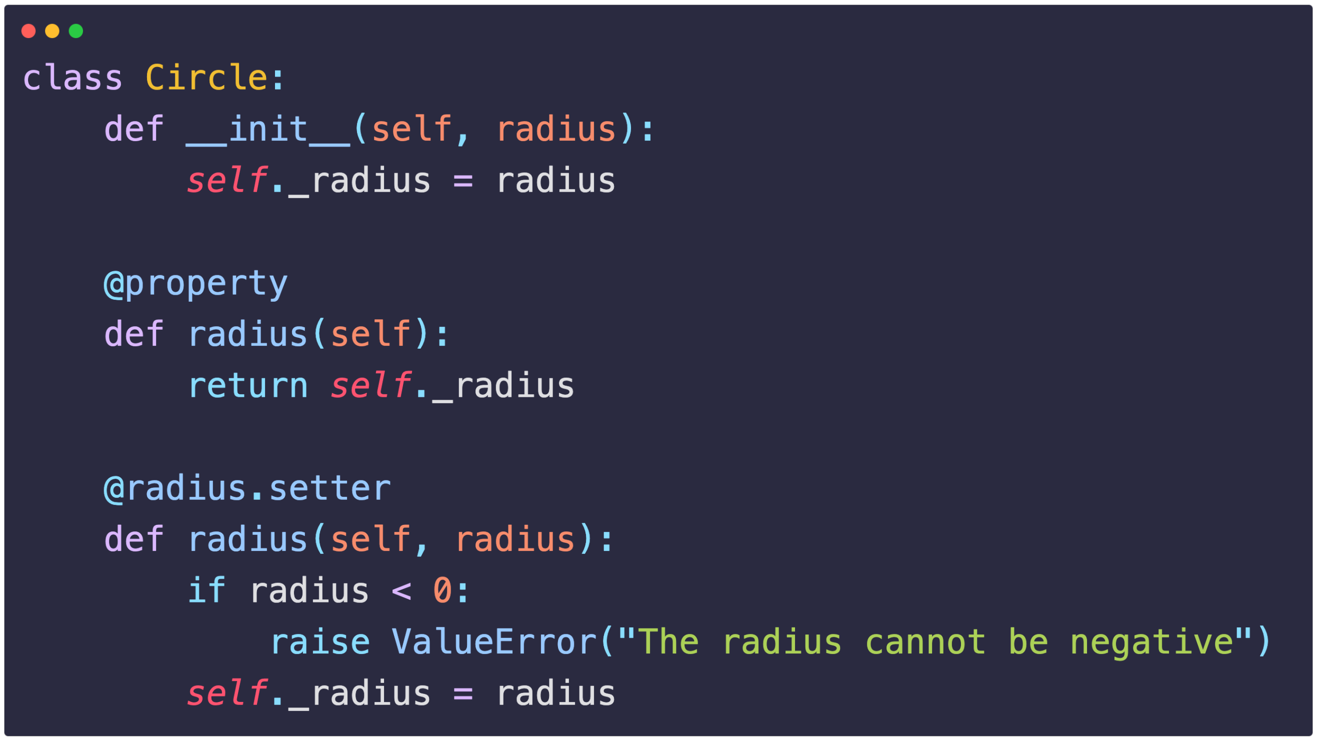
```python
class Circle:
    def __init__(self, radius):
        self._radius = radius

    def get_radius(self):
        return self._radius

    def set_radius(self, radius):
        if radius < 0:
            raise ValueError("The radius cannot be negative")
        self._radius = radius

    radius = property(get_radius, set_radius)
```

Fig. 4 Property using the @property syntax (Example)



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

    @property
    def radius(self):
        return self._radius

    @radius.setter
    def radius(self, radius):
        if radius < 0:
            raise ValueError("The radius cannot be negative")
        self._radius = radius
```

Note: ValueError is an exception that can be “raised when an operation or function receives an argument that has the right type but an inappropriate value” (source).