# **Магические методы \_\_getitem\_\_, \_\_setitem\_\_ и \_\_delitem\_\_**

[Смотреть материал на видео](https://www.youtube.com/watch?v=EAoiOwYQSuY&list=PLA0M1Bcd0w8zPwP7t-FgwONhZOHt9rz9E)

Я Сергей Балакирев и мы продолжаем знакомиться с магическими методами классов. Здесь речь пойдет о следующем их наборе:

* \_\_getitem\_\_(self, item) – получение значения по ключу item;
* \_\_setitem\_\_(self, key, value) – запись значения value по ключу key;
* \_\_delitem\_\_(self, key) – удаление элемента по ключу key.

Давайте разберемся для чего они нужны и как их можно использовать. Предположим, что мы создаем класс для представления студентов:

**class** Student:     **def** \_\_init\_\_(self, name, marks):         self.name = name         self.marks = list(marks)

Его экземпляр можно сформировать, следующим образом:

s1 = Student('Сергей', [5, 5, 3, 2, 5])

В объекте s1 имеется локальное свойство marks со списком студентов. Мы можем к нему обратиться и выбрать любую оценку:

**print**(s1.marks[2])

Но что если мы хотим делать то же самое, но используя только ссылку на объект s1:

**print**(s1[2])

Если сейчас запустить программу, то увидим сообщение об ошибке, что наш класс (объект) не поддерживает такой синтаксис. Как вы, наверное, уже догадались, поправить это можно с помощью магического метода \_\_getitem\_\_. Запишем его в нашем классе, следующим образом:

**def** \_\_getitem\_\_(self, item):         **return** self.marks[item]

Теперь ошибок нет и на экране видим значение 3. Однако, если указать неверный индекс:

**print**(s1[20])

то получим исключение IndexError, которое сгенерировал список marks. При необходимости, мы можем сами контролировать эту ошибку, если в методе \_\_getitem\_\_ пропишем проверку:

**def** \_\_getitem\_\_(self, item):         **if** 0 <= item < len(self.marks):             **return** self.marks[item]         **else**:             **raise** IndexError("Неверный индекс")

При запуске программы видим наше сообщение «Неверный индекс». Также можно сделать проверку на тип индекса:

**print**(s1['abc'])

для списков он должен быть целым числом. Поэтому дополнительно можно записать такую проверку:

**def** \_\_getitem\_\_(self, item):         **if** **not** isinstance(item, int):             **raise** TypeError("Индекс должен быть целым числом")          **if** 0 <= item < len(self.marks):             **return** self.marks[item]         **else**:             **raise** IndexError("Неверный индекс")

То есть, здесь возможны самые разные вариации обработки и проверки исходных данных, прежде чем обратиться к списку marks и вернуть значение.

Теперь давайте предположим, что хотели бы иметь возможность менять оценки студентов, используя синтаксис:

s1[2] = 4**print**(s1[2])

Сейчас, после запуска программы будет ошибка TypeError, что объект не поддерживает операцию присвоения, так как в классе не реализован метод \_\_setitem\_\_. Давайте добавим и его:

**def** \_\_setitem\_\_(self, key, value):         **if** **not** isinstance(key, int) **or** key < 0:             **raise** TypeError("Индекс должен быть целым неотрицательным числом")           self.marks[key] = value

Однако, если мы сейчас укажем несуществующий индекс:

s1[6] = 4

то операция присвоения новой оценки приведет к ошибке. Если предполагается использовать такую возможность, то реализовать ее можно, следующим образом:

**def** \_\_setitem\_\_(self, key, value):         **if** **not** isinstance(key, int) **or** key < 0:             **raise** TypeError("Индекс должен быть целым неотрицательным числом")           **if** key >= len(self.marks):             off = key + 1 - en(self.marks)             self.marks.extend([None]\*off)           self.marks[key] = value

Если индекс превышает размер списка, то мы расширяем список значениями None до нужной длины (с помощью метода extend), а затем, в последний элемент записываем переданное значение value. Теперь, при выполнении команд:

s1[10] = 4**print**(s1.marks)

Увидим список:

[5, 5, 3, 2, 5, None, None, None, None, None, 4]

То есть, он был расширен до 10 элементов и последним элементом записано 4. И так можно прописывать любую нужную нам логику при записи новых значений в список marks.

Наконец, последний третий магический метод \_\_delitem\_\_ вызывается при удалении элемента из списка. Если сейчас записать команду:

**del** s1[2]

то в консоли увидим сообщение: «AttributeError: \_\_delitem\_\_». Здесь явно указывается, что при удалении вызывается метод \_\_delitem\_\_. Добавим его в наш класс:

**def** \_\_delitem\_\_(self, key):         **if** **not** isinstance(key, int):             **raise** TypeError("Индекс должен быть целым числом")           **del** self.marks[key]

Теперь оценки успешно удаляются, если указан верный индекс.

Вот общие возможности данных магических методов. Теперь, если в программе вам потребуется реализовать подобную логику поведения экземпляров классов, я думаю, вы легко со всем справитесь.