Шпаргалка: списки

Кратко

Списки: общее

Что	Как	О чём помнить
Создать список	Схема: `<имя списка> = [<элемент>, <элемент>, <элемент>` Пример: `buttons = ['Отели', 'Туры']`	Нельзя называть переменную `list`. Список может хранить элементы разных типов и другие списки. Чтобы создать пустой список, оставь квадратные скобки пустыми: `buttons = []`.
Список списков	Схема: `<имя списка> = [<элемент>, [<элемент>, <элемент>], [<элемент>]]` Пример: `example = [1, 2, ['три', 'четыре']]`	Это список, в котором хранятся другие списки. Можно создать его сразу таким, а можно добавить вложенные списки с помощью методов `append()` и `insert()`. У него те же параметры, что и у обычного списка. При подсчёте длины вложенный список считается за один элемент.
Вычислить длину списка	Функция `len()` Схема: `len(<имя списка>)` Пример: `count = len(russian_alphabet)`	
Получить элемент списка по индексу	Схема: `<имя списка>[индекс]` Пример: `print(russian_alphabet[0])` Напечатает: а	У элементов в списке есть порядковые номера — **индексы**. Индекс указывается после переменной, в квадратных скобках. **Отсчёт индексов начинается с** `0`.
Найти индекс элемента через длину списка	Схема: `len(<имя списка>) - x` Пример: `count = len(russian_alphabet) - 1` `print(count)` Напечатает: я	Так можно находить элементы с конца списка. Индекс последнего элемента равен длине списка минус один элемент, предпоследнего — минус два и так далее.

Операции со списками

Что	Как	О чём помнить
Добавить элемент в конец списка	Метод `append()` **Схема:** `<имя списка>.append(<элемент>)` **Пример:** `hobbits.append('Сэм')`	Добавляет элемент в конец списка.
Добавить элемент по индексу	Метод `insert()` **Схема:** `<имя списка>.insert(<индекс>, <элемент>)` **Пример:** `hobbits.insert(2, 'Смеагол')`	Метод требует на вход два параметра: индекс, **по которому** нужно вставить элемент, и сам элемент.
Удалить элемент по значению	Метод `remove()` **Схема:** `<имя списка>.remove(<элемент>)` **Пример:** `append() hobbits.append('Сэм')`	Удаляет элемент из списка по значению. Если в списке несколько одинаковых элементов, метод удалит первый из них.

Удалить элемент по индексу	Метод `pop()`
	Схема: `<имя списка>.pop(<индекс>)`
	Примеры:
	`hobbits.pop(-2)`
	
	`hobbits.pop()`
		Удаляет элемент из списка и возвращает его. Можно подхватить удаляемое значение и присвоить его переменной или сохранить в другом списке.
		Если не указывать индекс, удалит последний элемент.

Проверить вхождение элемента в список	Оператор `in` **Схема:** `print(<элемент> in <имя списка>)` **Пример:** `print('Смеагол' in hobbits)`	Оператор всегда используют вместе с функцией `print()`. Если элемент содержится в списке, код выведет `True`, если не содержится — `False`.
Сложить списки	Оператор `+` **Схема:** `<имя списка> = <список 1> + <список 2>` **Пример:** `fellowship = hobbits + new_members`	При сложении длина списка увеличивается на то количество элементов, которое было прибавлено.

Поверхностное копирование	Метод `copy()` **Схема:** `<имя копии> = <имя списка>.copy()` **Пример:** `new_hobbits_list = hobbits.copy()`	Этот метод создаёт новый составной объект. В него он вставляет **ссылки** на объекты оригинального списка. Использовать этот метод нужно осторожно. Если копировать сложный составной объект и изменить копию, изменения затронут оригинал.
Полное копирование	Метод `deepcopy()` **Схема:** `<имя копии> = <имя списка>.deepcopy()` **Пример:** `new_hobbits_list = hobbits.deepcopy()`	Чтобы воспользоваться методом, его нужно импортировать: `from copy import deepcopy`. При полном копировании создаётся новый составной объект, в него вставляются **копии** объектов оригинального списка. После полного копирования можно делать с новым списком что угодно. Изменения не затронут оригинал.

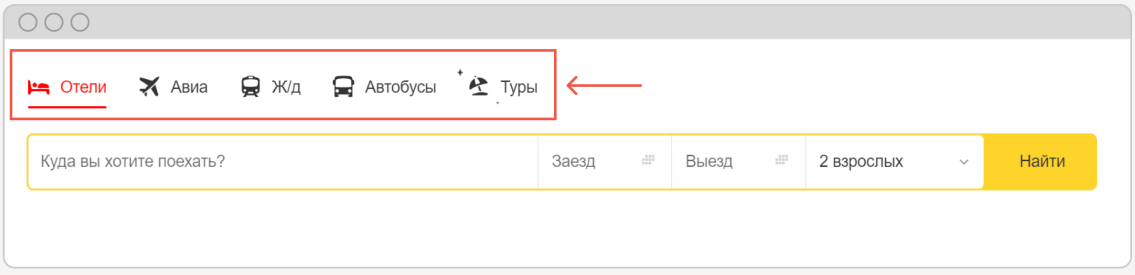
Подробно

В переменной можно хранить не только числа и строки, но и списки.

> ! **Список** — это последовательность чисел, строк или других значений. Например, числа от 0 до 10. Или все русские слова на букву «а».

Зачем нужен список. Список позволяет хранить в одной переменной сразу несколько значений. Так код получается лаконичнее, да и работать с ним удобнее.

Пример. Представь, что тебе нужно хранить названия кнопок на веб-странице Яндекс Путешествий: «Отели», «Авиа», «Ж/д», «Автобусы», «Туры».



Если каждую кнопку сохранить в отдельной переменной, код будет выглядеть так:

```
# Объявляем все кнопки по очереди

hotels_button = 'Отели' # По одной переменной на каждую кнопку
avia_button = 'Авиа'
rzd_button = 'Ж/д'
buses_button = 'Автобусы'
tours_button = 'Туры'
```

Чтобы облегчить код, можно сохранить все кнопки в одном списке:

```
buttons = ['Отели', 'Авиа', 'Ж/д', 'Автобусы', 'Туры']
# Так и выглядит список
```

Как создать список

Чтобы создать список, нужно:

1. Написать имя переменной, которая будет содержать список. Например, `buttons`.

Обрати внимание: имя `list` в Python зарезервировано. Так не стоит называть переменную. А вот `lst` или `new_list` — уже допустимо.

2. Поставить знак приравнивания `=`.
3. Поставить квадратные скобки `[]`. Именно они говорят, что перед тобой список.
4. Внутри квадратных скобок через запятую перечислить элементы списка.

```
buttons = ['Отели', 'Авиа', 'Ж/д', 'Автобусы', 'Туры']
```

Список с кнопками состоит из строк, поэтому каждый элемент в кавычках.

Что можно хранить в списке

📌 Список может состоять из элементов разных типов.

Python неважно, элементы каких типов помещать в список. Числа, строки и даже другие списки можно хранить вместе.

Пример списка с разными типами данных:

```
countdown2 = ['пять', 4, 3, 2, [1*1, 1*0]]

# [1*1, 1*0] — это список внутри списка.
# В четвёртом уроке ты научишься создавать такие же.
```

Пример списка из чисел:

```
countdown = [5, 4, 3, 2, 1, 0]
```

Можно сделать список из выражений. Тогда в нём будут храниться значения, которые программа вычислит прямо в списке.

Пример списка из выражений:

```
# сохраним в списках вторую и третью строки таблицы умножения
pithagoras_2 = [
    2*1, 2*2, 2*3, 2*4, 2*5, 2*6, 2*7, 2*8, 2*9
]
pithagoras_3 = [
    3*1, 3*2, 3*3, 3*4, 3*5, 3*6, 3*7, 3*8, 3*9
]
print(pithagoras_2)
print(pithagoras_3)
```

Если запустить код, он выведет:

```
[2, 4, 6, 8, 10, 12, 14, 16, 18]
[3, 6, 9, 12, 15, 18, 21, 24, 27]
```

Пустые списки

Если заранее не ясно, что будет храниться в списке, его можно создать пустым. А потом уже наполнить значениями. Пустой список создают через пустые квадратные скобки.

Пример:

```
5_km_list = []
10_km_list = []
half_marafon_list = []
marafon_list = []
```

Длина списка

Длина списка равна количеству включённых в него элементов. Для подсчёта элементов списка есть стандартная функция `len()`.

Пример. В переменной `russian_alphabet` сохранили список из букв алфавита:

```
russian_alphabet =
['а','б','в','г','д','е','ё','ж','з','и','й','к','л','м','н','о','п','р','с','т','у','ф','х','ц','ч','ш','щ','ъ','ы','ь','э','ю','я']
```

Посчитали длину `russian_alphabet`:

```
count = len(russian_alphabet)
print(count)

# Будет напечатано: 33
```

Индекс элемента

У каждого элемента списка есть порядковый номер — **индекс**. Зная индекс, можно получить значение элемента списка.

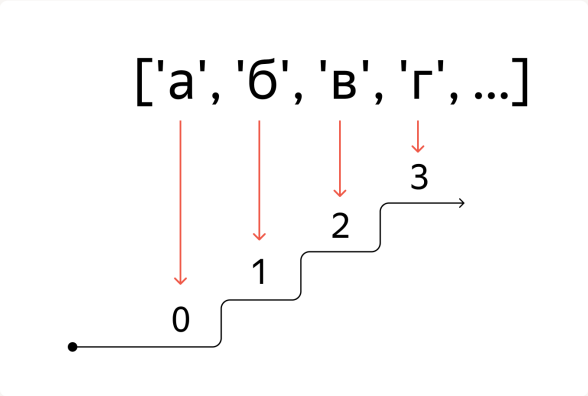
Пример. Тебе нужно вывести на экран первую букву алфавита. Для этого укажи индекс `0` после переменной в квадратных скобках.

```
russian_alphabet =
['а','б','в','г','д','е','ё','ж','з','и','й','к','л','м','н','о','п','р','с','т','у','ф','х','ц','ч','ш','щ','ъ','ы','ь','э','ю','я']
```

```
print(russian_alphabet[0])  # Напечатать значение элемента с индексом 0
# Будет напечатано: а
```

> ❗ У первого элемента списка индекс всегда нулевой.

Чтобы лучше запомнить, представь лестницу:



Как найти индекс элемента через `len()`

Не всегда удобно перебирать индексы всех элементов списка, чтобы найти значение последнего элемента. В этом случае можно использовать функцию `len()`. Индекс первого элемента — `0`. Значит, индекс последнего элемента будет равен длине списка минус один элемент.

Пример. У буквы я в списке `russian_alphabet` индекс будет равен `len(russian_alphabet) - 1`.

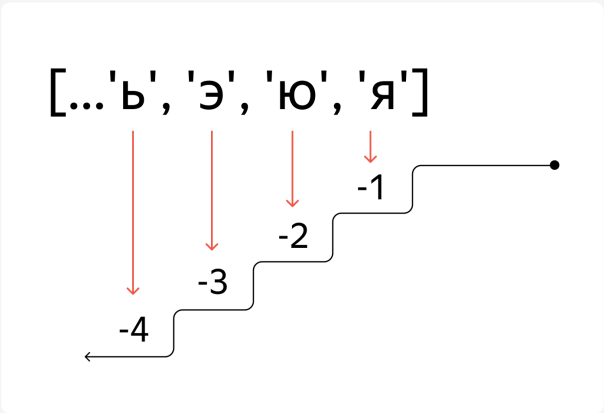
```
count = len(russian_alphabet)
print(russian_alphabet[count - 1]) # Будет напечатано: я
```

Так же можно обратиться и к предпоследним элементам. Например, индекс буквы `ю` находится как `len(russian_alphabet) - 2`.

Отрицательные индексы

Обратиться к элементам можно с помощью **отрицательных индексов**.

Отрицательные индексы — это отсчёт элементов с конца списка. У последнего элемента индекс `-1`, у предпоследнего `-2` и так далее.



Пример:

```
print(russian_alphabet[-2]) # Будет напечатано: ю
print(russian_alphabet[-4]) # Будет напечатано: ь
```

Добавить элемент: метод `append()`

Включить новый элемент **в конец** списка позволяет метод `append()`. Например, `numbers.append(5)` добавит элемент `5` в конец списка `numbers`.

Обрати внимание на запись `numbers.append(5)`. Так вызывается метод:

- То, к чему его нужно применить, — переменная `numbers`.
- Через точку сам метод — `numbers.append()`.
- В скобках — элемент, который нужно добавить. Получается `numbers.append(5)`.

Пример. С помощью `append()` добавили в список `hobbits` ещё один элемент.

```
# создали список из одного элемента — хоббита Фродо
hobbits = ['Фродо']

# добавили ещё одного хоббита
hobbits.append('Сэм')

# вдвоём кольцо станет нести легче

print(hobbits) # Будет напечатано: ['Фродо', 'Сэм']
```

Добавить элемент на указанную позицию: метод `insert()`

Чтобы поместить новый элемент в конкретное место, используют метод `insert()`. Он добавляет элементы по указанному индексу.

Метод принимает на вход два параметра:

- индекс, **куда** нужно вставить новый элемент;
- сам элемент.

Пример:

```
# в первом списке hobbits четверо героев: ['Фродо', 'Сэм', 'Мерри', 'Пиппин']

# добавили к ним ещё одного
hobbits.insert(2, 'Смеагол')

# посчитали длину списка
print(len(hobbits))

# Будет напечатано: 5
```

Удалить элемент по значению: метод `remove()`

Удалить элемент по имени может метод `remove()`. Он ищет в списке значение, которое ты указываешь в скобках, и убирает его из списка.

Пример. Удалили из списка `hobbits` элемент `Смеагол`.

```
# удалили странного хоббита
hobbits.remove('Смеагол')

# так-то лучше
print(hobbits)

# Будет напечатано: ['Фродо', 'Сэм', 'Мерри', 'Пиппин']
```

Обрати внимание: метод удаляет не все элементы с подходящим значением, а только **первый** из них. Он не идёт дальше первого вхождения.

```
# список с повторяющимися элементами
few_numbers = [1, 2, 1, 2]

# исключили элемент 2
few_numbers.remove(2)

# проверили состав списка
print(few_numbers) # Будет напечатано: [1, 1, 2]
```

Если указанного значения нет в списке, появится ошибка: `ValueError: list.remove(x): x not in list`.

Удалить элемент по индексу: метод `pop()`

Метод `pop()` удаляет элемент по указанному индексу, а также возвращает удалённый элемент. Например, `numbers.pop(4)` удалит из списка `numbers` элемент с индексом `4` и вернёт его.

Что значит «возвращает элемент».

Метод `pop()` как бы говорит программе:

- извлеки элемент из списка;
- верни его в ту часть кода, где меня вызвали;
- в список обратно не возвращай.

В результате элемент вынимается из списка, повисает ненадолго в коде, а затем стирается из него.

Пример. Удалили элемент `Мерри` из `hobbits`:

```
hobbits = ['Фродо', 'Сэм', 'Мерри', 'Пиппин']

lst = hobbits.pop(-2)
# В одно действие создали новый список, удалили Мерри из hobbits и добавили в merry
```

```
print(hobbits)
# Будет напечатано: ['Фродо', 'Сэм', 'Пиппин']

print(lst)
# Будет напечатано: Мерри
```

Для наглядности в первой строчке кода создали новую переменную. Ей присвоили значение, которое метод `pop()` удалил из списка `hobbits` и вернул в код. Так Мерри в одно действие покинул хоббитов и оказался в отдельном списке.

Возвращаемое значение можно присвоить переменной или передать на вход методам `append()` и `insert()`.

Иногда нужно перенести элемент из одного списка в другой. Тогда его удаляют через `pop()` и передают на вход методам, которые добавляют элементы.

Например, `lst_2.append(lst_1.pop(0))`.

Эта строчка:

1. удалит из списка `lst_1` первый элемент;
2. добавит его в конец `lst_2`.

В `pop()` можно не передавать индекс.

Индекс элемента можно не указывать. Тогда `pop()` по умолчанию удалит и вернёт последний элемент.

Пример:

```
hobbits = ['Пиппин', 'Фродо', 'Сэм']

hobbits.pop()   # 'Сэм'

print(hobbits)  # ['Пиппин', 'Фродо']
```

Метод `pop()` и ошибки

Python подскажет, если что-то пойдёт не так.

- Применение метода к пустому списку вызовет ошибку: `IndexError: pop from empty list`. В переводе с английского — «извлечение из пустого списка».
- Если указать индекс, которого нет в списке, программа тоже выдаст ошибку. Её текст будет таким: `IndexError: pop index out of range`. Переводится как «извлекаемый индекс вне диапазона».

Как проверить, есть ли элемент в списке

Пригодится оператор **проверки вхождения элемента в список** — `in`. Его используют вместе с функцией `print()`. В скобках после неё пишут имя элемента, затем ставят оператор `in` и указывают имя списка.

Если элемент содержится в списке, код выведет `True`, если не содержится — `False`.

Пример. Нужно проверить, есть ли в списке `few_numbers` элемент со значением `2`.

```
# объявили список
few_numbers = [1, 2, 1, 2]

# запустили проверку
print(2 in few_numbers)

# Будет напечатано: True
```

📢 Регистр букв имеет значение. Обращай на него внимание, иначе проверка не будет валидной.

Пример. Удалили Смеагола из `hobbits` и проверили вхождение этого элемента в список.

```
hobbits = ['Фродо', 'Смеагол', 'Сэм', 'Мерри', 'Пиппин']

# Удалили Смеагола
hobbits.pop(1)

print('Смеагол' in hobbits)
# Будет напечатано: False

# Достаточно ошибиться в одной букве, чтобы проверка не сработала
print('ФродО' in hobbits)
# Будет напечатано: False  (буква «Д» заглавная)
```

С помощью булевой переменной тоже можно проверить, есть ли элемент в списке.

Пример. Нужно проверить, есть ли `Сэм` в списке `hobbits`.

```
# Присвоили значение булевой переменной:
is_in_hobbits_list = 'Сэм' in hobbits

print(is_in_hobbit_list)
# Будет напечатано: True
```

Сложение списков

К списку можно прибавить другой список. Можно сказать, это ещё один способ добавить элементы, но сразу несколько.

Например, был список с числами от 1 до 3. К нему прибавили список с числами от 4 до 6. Получился список с числами 1-6. То есть сначала часть элементов сохранили в одном списке, ещё часть — в другом. Потом объединили их и получили финальный набор.

Как сложить списки

Для этого используют оператор `+`.

Пример. В список к хоббитам следует добавить ещё несколько представителей Братства кольца. Для этого новых героев нужно сохранить в новом списке. Затем — создать переменную и присвоить ей значение суммы списков.

```
# стартовый набор
hobbits = ['Фродо', 'Сэм', 'Мерри', 'Пиппин']

# новый набор
new_members = ['Гэндальф', 'Арагорн']

# объединение списков
fellowship = hobbits + new_members

print(fellowship)

# Будет напечатано: ['Фродо', 'Сэм', 'Мерри', 'Пиппин', 'Гэндальф', 'Арагорн']
```

При сложении длина списка увеличивается на то количество элементов, которое было прибавлено. Например, было 5 элементов, к ним добавили ещё 2. Длина нового списка — 7.

Список списков

Список может состоять из любых объектов — даже из списков. Список списков — это список, в котором хранятся другие списки.

Пример. Создали новый список с орлами и добавили его в `hobbits`.

```
hobbits = ['Фродо', 'Сэм']

# по одному орлу на хоббита
eagles = ['Орёл 1', 'Орёл 2']

# добавили орлов в список
hobbits.insert(2, eagles)

print(hobbits)
# Будет напечатано: ['Фродо', 'Сэм', ['Орёл 1', 'Орёл 2']]
```

Теперь `hobbits` — это список списков.

У списка списков все те же параметры, что и у обычного списка. Например, у него тоже можно определить длину.

Но тут есть одна деталь, о которой стоит помнить:

Вложенный список в списке списков считается за один элемент.

Копирование списков

Работа со списком может сильно его изменить. Например, в `buttons` хранятся кнопки с веб-страницы Яндекс Путешествий. В список внесли несколько изменений:

```
buttons = ['Отели', 'Авиа', 'Ж/д', 'Автобусы', 'Туры']

buttons.pop() # удалили последний элемент
buttons.append('Ещё') # добавили новую кнопку
buttons.insert(2, 'Пароходы') # добавили ещё кнопку
buttons.pop(0) # удалили первую кнопку

# Так-то лучше
print(buttons)

# Будет напечатно: ['Авиа', 'Пароходы', 'Ж/д', 'Автобусы', 'Ещё']
```

Теперь список сильно отличается от себя прежнего. Если не сохранить первую версию списка, то вернуться к ней уже не получится. Она потеряется в коде.

Поэтому перед операциями со списками используют копирование. Так можно создать клон списка и менять его, а не оригинал.

Как копировать список

В Python есть два вида копирования:

- **поверхностное** копирование,
- **полное**, или глубокое копирование.

Они почти ничем не отличаются. Различия видно, только когда работаешь со сложными составными объектами. Такими как списки списков.

Поверхностное копирование: метод `copy()`

Копировать список можно с помощью метода `copy()`.

```
# список списков
hobbits = ['Фродо', 'Сэм', ['Орёл 1', 'Орёл 2']]

# скопировали его
new_hobbits_list = hobbits.copy()

# логическим оператором проверили равенство списков:
print(new_hobbits_list == hobbits)
# Будет напечатано: True

# Всё получилось —  списки идентичны

print(new_hobbits_list)
# Будет напечатано: ['Фродо', 'Сэм', ['Орёл 1', 'Орёл 2']]
```

Этот способ копирования создаёт новый составной объект. В него метод вставляет **ссылки** на объекты, которые хранятся в оригинале.

📌 Метод `copy()` отсылает к оригинальным объектам, а не создаёт их копии.

Что будет, если добавить новый элемент в список-копию

Копия списка `hobbits` хранится в переменной `new_hobbits_list`. Посмотри, что произойдёт, если добавить в `new_hobbits_list` ещё один элемент:

```
new_hobbits_list.append('Бильбо')

print(new_hobbits_list)
# Будет напечатано: ['Фродо', 'Сэм', ['Орёл 1', 'Орёл 2'], 'Бильбо']

# при этом оригинальный список не изменился
print(hobbits)
# Будет напечатано: ['Фродо', 'Сэм', ['Орёл 1', 'Орёл 2']]
```

Оригинальный список не изменился. Новый элемент добавлен в сам список. Он не затронул уже существующий вложенный список, на который ссылается копия. Поэтому элементы оригинального списка не изменились.

Но если добавить новый элемент во вложенный список, изменения затронут оригинал.

Что будет, если добавить новый элемент во вложенный список

К примеру, в список добавили ещё одного орла. Посмотри, что поменяется:

```
new_hobbits_list[2].append('Орёл 3') # Добавили орла для Бильбо

print(new_hobbits_list)
# Будет напечатано: ['Фродо', 'Сэм', ['Орёл 1', 'Орёл 2', 'Орёл 3'], 'Бильбо']

# в этот раз оригинальный список поменяется
print(hobbits)
# Будет напечатано: ['Фродо', 'Сэм', ['Орёл 1', 'Орёл 2', 'Орёл 3'], 'Бильбо']
```

Изменения произошли сразу в двух местах: в оригинальном списке `hobbits` и в скопированном `new_hobbits_list`.

Вложенный список с орлами — это изменяемый объект, который существует в оригинальном списке `hobbits`. Копия ссылается на него. Когда в копию добавили нового орла, содержание списка изменилось. Поэтому список поменялся и в оригинале.

Полное копирование: метод `deepcopy()`

При полном копировании тоже создаётся новый составной объект. Но в него вставляются уже не ссылки, а **копии** объектов из оригинального списка.

Для полного копирования используют метод `deepcopy()`.

📌 Метод `deepcopy()` создаёт копии оригинальных объектов.

✅ После полного копирования можно делать с новым списком что угодно. Изменения не затронут оригинал.

В отличие от знакомых тебе методов и функций, `deepcopy()` Python не умеет вызывать самостоятельно. Чтобы использовать, его нужно импортировать. Для этого добавляют команду `from copy import deepcopy`. На языке программы она звучит так: «Из библиотеки `copy` достань метод `deepcopy`. Скоро я буду его использовать».

Пример:

```
from copy import deepcopy # импортировали метод deepcopy из модуля copy

# создали заново список
hobbits = ['Фродо', 'Сэм', ['Орёл 1', 'Орёл 2']]

# сделали полную копию
new_hobbits_list = deepcopy(hobbits)

print(hobbits, new_hobbits_list)
# Будет выведено: ['Фродо', 'Сэм', ['Орёл 1', 'Орёл 2']], ['Фродо', 'Сэм', ['Орёл 1', 'Орёл 2']]

# теперь изменили копию
new_hobbits_list.append('Бильбо')
new_hobbits_list[2].append('Орёл 3')

# проверили содержимое списков
print(new_hobbits_list)
# Будет выведено: ['Фродо', 'Сэм', ['Орёл 1', 'Орёл 2', 'Орёл 3'], 'Бильбо']

print(hobbits)
# Будет выведено: ['Фродо', 'Сэм', ['Орёл 1', 'Орёл 2']]
```

Обрати внимание: метод `deepcopy()` вызывается как функция. Перед ним не нужно писать имя списка и ставить точку.

Какой метод когда использовать

Пора закрепить, когда нужно использовать `copy()`, а когда `deepcopy()`.

Метод `copy()` работает быстрее, чем `deepcopy()`. Потому что он не копирует объекты, а только отсылает к ним. **Если ты знаешь, что в списке не будет других списков или словарей**, используй `copy()`.

Например, список кнопок на сайте будет состоять из неизменяемых объектов — названий кнопок:

```
new_buttons = ['Создать', 'Удалить', 'Переместить']
```

Названия кнопок — это обычные элементы строкового типа. Их можно удалить, но нельзя изменить. Такой список можно копировать через `copy()`.

Но как только в твоём списке **появляются вложенные списки или словари**, копируй через `deepcopy()`. Тогда копия будет полностью независима от оригинала.

Например, в списке `users` хранятся списки пользователей и наборы их реквизитов:

```
users = ['user_a', ['Иванов Петр Андреевич', 'г. Москва', '+79154567788'], 'user_b', ['Петрова Анна Сергеевна', 'г. Ярославль', '+79104859513']]
```

В списке есть неизменяемые объекты — ники пользователей. И есть изменяемые — вложенные списки с реквизитами. Тут используй только `deepcopy()`. Если ты поменяешь реквизиты в копии, они останутся прежними в оригинальном списке.