Шпаргалка: циклы

Кратко

Что	Как	О чём помнить

Цикл	**Схема:**
	`for <переменная> in <имя списка>:`
	`  <тело цикла>`
	Пример:
	`for movie in best_movies:`
	`  print(movie)`
		Цикл состоит из двух частей: **условия** и **тела**.
		В условии указывают **внутреннюю переменную** и список, который нужно обработать. Имя переменной образуют от имени списка, в единственном числе.
		Элементы списка по очереди передаются в переменную.
		Тело цикла — это код, который выполняется с переменной, пока цикл перебирает список. Все строки тела отбивают четырьмя пробелами от начала строки.
		Обработку одного элемента списка называют **итерацией** цикла.

Диапазон чисел	Функция `range()` **Схема:** `range(<первое число>, <последнее число>)` **Пример:** `three = range(0, 3)` В диапазон войдут 0, 1 и 2	Последнее число диапазона **не входит** в последовательность. Если последовательность начинается с нуля, можно указать только конец диапазона. Диапазон может включать в себя отрицательные числа. Можно указать шаг последовательности. Например: `numbers = range(1, 101, 5)`. Тут шаг = `5`. Диапазон можно перебирать в цикле. Тогда внутреннюю переменную называют `i`.
Обратный перебор	Функция `reversed()` **Схема:** `reversed(<диапазон или цикл>)` **Примеры:** `reversed(range(1, 6))`; `reversed(lst)`	

Срез списка	**Схема:** `<список>[<индекс начала>:<индекс конца>:<величина шага>]`
	Пример:
	`buttons = ['Отели', 'Авиа', 'Ж/д', 'Автобусы', 'Туры']`
	`print(buttons[0:3:1])`
	Напечатает: ['Отели', 'Авиа', 'Ж/д']
		Последний индекс среза в него **не входит**.
		Если шаг равен единице, его не указывают.
		Если индексы начала и конца среза совпадают с индексами первого и последнего элемента списка, их не указывают.

Обратный срез списка	**Схема:** `<список>[<индекс начала>:<индекс конца>:-1]` **Пример:** `numbers[::-1]`	

Вложенный цикл	**Схема:**
	`for <переменная внешнего цикла> in <список>:`
	`  for <переменная вложенного цикла> in <переменная внешнего цикла>:`
	`  <тело вложенного цикла>`
	`  <тело внешнего цикла>`
	Пример:
	`for floor_rooms in hotel_floors:`
	`  for room in floor_rooms:`
	`  print('Убрать', room)`
		Условие вложенного цикла отбивают четырьмя пробелами от начала строки.
		Тело обивают ещё четырьмя пробелами.

Цикл с условием	**Схема:**
	`for <переменная> in <список>:`
	`  if <условие>:`
	`  <тело цикла>`
	`  else <условие>:`
	`  <тело цикла>`
	Пример:
	`for locker in lockers:`
	`  if locker == 3:`
	`  print('А вот и мой чемодан!')`	

Ключевые слова `break` **и** `continue`	**Схема:**
	`for <переменная> in <список>:`
	`  if <условие>:`
	`  <ключевое слово>`
	`  <тело цикла>`
	Пример:
	`for i in sheep:`
	`  if i == 3:`
	`  continue`
	`  print(i, 'овечка')`
		Используй `continue` для пропуска итерации и `break` для остановки цикла.
		Их нужно использовать с вложенным условием `if`. Ключевое слово пишется с новой строки после него, его отделяют +4 пробела.
		Если ключевое слово находится во вложенном цикле, на внешний цикл оно не влияет.

Подробно

Что такое цикл

В тестировании часто встречаются задачи, когда нужно выполнять серии одинаковых операций. Например, перебрать все строки таблицы из базы данных. Обрати внимание: этот процесс не бесконечен. Работа продолжается **при условии**, что остались необработанные строки.

Пример. Кинотеатр проводит фестиваль культового кино. Есть список с фильмами 1994-го года:

```
best_movies = ['Побег из Шоушенка', 'Криминальное чтиво', 'Форрест Гамп', 'Леон', 'Король Лев']
```

Нужно напечатать их названия на афише:

```
Побег из Шоушенка
Криминальное чтиво
Форрест Гамп
Леон
Король Лев
```

Можно вручную прописать вывод каждого элемента, но это долго и неэффективно:

```
print(best_movies[0])
print(best_movies[1])
...
```

Чтобы решить задачу, нужно: взять первый элемент списка `best_movies`, напечатать его через `print()`, потом взять следующий элемент... И продолжать так, пока список не кончится.

Для выполнения таких операций придумали **циклы**. Это программные конструкции, которые повторяют определённые действия, пока выполняется заданное условие.

Как написать цикл

Цикл для афиши будет выглядеть вот так:

```
best_movies = ['Побег из Шоушенка', 'Криминальное чтиво', 'Форрест Гамп', 'Леон', 'Король Лев']

for movie in best_movies: # Это условие цикла
    print(movie) # А это тело цикла
```

Посмотри на две строчки кода после объявления списка `best_movies`. Это и есть цикл. Он состоит из двух частей: **условия** и **тела**.

Условие цикла

Сначала нужно сообщить программе, что сейчас начнётся цикл. В Python цикл объявляют ключевыми словами `for` и `in`. После них задают **условие цикла** и ставят двоеточие.

```
for переменная in список_элементов: # Общая схема
```

Например, для фильмов это выглядит так:

```
for movie in best_movies:
```

В условии цикла:

- После `for` — имя переменной, в которую будут по очередь передаваться элементы списка. Эту переменную называют **внутренней переменной** цикла.
- После `in` — имя списка, который нужно обработать.

Что такое внутренняя переменная

Внутренняя переменная — это такой «держатель» для элемента, с которым прямо сейчас циклу нужно что-то сделать.

Когда первый элемент списка обработан, переменной присваивается значение следующего элемента. И так — элемент за элементом — цикл обрабатывает список.

Обрати внимание: переменная `movie` впервые появилась в условии цикла. До этого её не было в коде. Имя переменной дают при написании условия.

Как назвать внутреннюю переменную

> 📌 Внутреннюю переменную можно назвать как угодно. Но традиционно её имя образуют от имени списка, в единственном числе.

Если список называется `musicians`, то переменную называют `musician`. Если `friends` — то `friend`. Это удобно и помогает не запутаться.

После условия идёт переход на новую строку. Здесь пишут **тело цикла**.

Тело цикла

Это код, который будет воспроизводиться при каждой итерации цикла. То есть — с каждым элементом списка.

Условие говорит программе, с чем цикл будет работать. А тело поясняет — как именно.

Все строки тела цикла отбиваются от начала строки четырьмя пробелами:

```
for переменная in список_элементов:
    # Тут тело цикла: код, который выполняется для каждого элемента
    # Здесь можно обработать переменную, объявленную в условии цикла

for movie in best_movies:
    print(movie) # Например, сказали циклу
                 # напечатать переменную
```

По этим отступам Python понимает, где начинается и кончается тело цикла.

❗ Это строгое техническое условие. Без отступов в блоках кода программа не заработает.

Принцип работы цикла

Цикл берёт значение первого элемента из списка `best_movies` и передаёт его в переменную `movie`. Затем выполняется код в теле цикла: печатается содержимое переменной `movie`.

Первый элемент, который напечатает цикл, — `Побег из Шоушенка`. Затем начнётся новый «круг», уже со следующим элементом списка. В переменную поступит `Криминальное чтиво`. Цикл его напечатает.

Так будет продолжаться, пока цикл не переберёт весь список:

```
Побег из Шоушенка # Результат первого круга цикла
Криминальное чтиво # Результат второго круга цикла
Форрест Гамп # И так далее
Леон
Король Лев
```

Каждый такой «круг» называют **итерацией цикла**.

Когда список закончится — программа выйдет из цикла. Запустится код, который написан после него.

Диапазоны от и до

Часто нужно обработать последовательность целых чисел. Например, узнать сумму целых чисел от 1 до 100 или перечислить номера вагонов поезда. Можно вручную создать список из нужных чисел. Но это не всегда удобно.
Список из двадцати чисел уже получается длинным:

```
twenty_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Вместо списка можно использовать функцию `range()`.

Функция `range()`

В функцию передают два целых числа: начало и конец диапазона. Первое число должно быть меньше второго. Например, 1 и 1 000. Функция создаёт последовательность. В неё включаются все целые числа в диапазоне, кроме последнего.

Получается, значения все те же, что и в списке — если прописывать их вручную.

Пример. Нужно задать последовательность целых чисел от 0 до 2. Для этого в `range()` передают числа 0 и 3:

```
three = range(0, 3)
# Последовательность будет включать числа 0, 1 и 2
# Тройка в последовательность не войдёт
```

❌ В последовательность **не входит последнее число диапазона.**

Представь, что последнее число держит красный знак «СТОП». Когда последовательность доходит до соседнего числа, то видит его и останавливается.

Ещё пример. Вот последовательность из двадцати чисел:

```
twenty = range(1, 21)
# Последовательность чисел от 1 до 20
# 21 тоже на борт не возьмут
```

Это всё равно что написать:

```
twenty_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Но выглядит аккуратнее и создаётся проще.

Особенности диапазона

У функции `range()` есть четыре особенности. Они упрощают работу с ней.

1. **Если последовательность начинается с нуля, можно указать только конец диапазона.**

Когда функция получает одно число, она автоматически создаёт диапазон от 0 до указанного числа.

Пример. Последовательность от 0 до 5 — это `range(6)`.

```
five = range(0,6) # Последовательность включает 0, 1, 2, 3, 4 и 5

also_five = range(6) # Такая же последовательность

print(five == also_five) # Проверили логическим оператором

# Будет напечатано: True
```

2. **Диапазон может включать в себя отрицательные числа.**

Функция работает не только с положительными, но и с отрицательными числами.
Пример. Можно создать последовательность от -3 до 2.

```
around_zero = range(-3, 3)
# Последовательность включает числа -3, -2, -1, 0, 1 и 2
```

3. **Можно указать шаг последовательности.**

Необязательно включать в последовательность все числа подряд.

По умолчанию шаг равен единице. Но можно указать любой другой. Например, шаг, равный двум. Тогда последовательность будет включать первое число и каждое третье.

Чтобы указать величину шага, передают три значения: левую границу, правую и шаг.

Пример:

```
step_size_2 = range(1, 7, 2)
# Указали начало последовательности (1), её конец (7) и шаг = 2
# step_size_2 будет включать числа 1, 3, 5
```

4. **Диапазон можно перебирать в цикле.**

Последовательность `range()` обрабатывают в цикле, как обычный список.

Пример. Можно напечатать циклом последовательность `around_zero`.

```
around_zero = range(-3, 3)

# Вместо списка в цикл передаётся переменная around_zero
# В ней хранится range() от -3 до 3
for i in around_zero:
    # Перебрать все числа в диапазоне от -3 до 3 и напечатать их:
    print(i)

# Будет напечатано:
# -3
# -2
# -1
# 0
# 1
# 2
```

Внутреннюю переменную в цикле назвали `i`. Такое имя всегда дают переменной, в которую поступают элементы последовательности.

> 📢 Для последовательности внутреннюю переменную цикла традиционно называют `i`. Но её можно назвать как угодно.

Задать диапазон можно сразу в условии цикла. Не обязательно создавать промежуточную переменную. Например, уберём из кода `around_zero`:

```
# Цикл переберёт все числа в диапазоне от -3 до 3 и напечатает их:
for i in range(-3, 3):
    print(i)

# Результат будет тот же
```

Обратный перебор

Нередко нужно перебрать список или диапазон чисел в обратном порядке. То есть начать цикл с последнего элемента и двигаться к первому.

Для этого можно использовать:

- функцию `reversed()`;
- или обратный срез для списка.

Функция `reversed()`

Функцию `reversed()` используют, когда нужно пройтись по последовательности в обратном порядке. Например, посчитать не от 1 до 12, а от 12 до 1 — как обратный отсчёт.

Она переворачивает список или диапазон чисел. Например, `reversed(buttons)` — прочитает список кнопок от последней к первой. А `reversed(range(6))` пройдётся по диапазону от 5 к 0.

Пример с диапазоном. Вот симулятор новогодних курантов: цикл запускает обратный отсчёт до Нового года.

```
for i in reversed(range(1, 13)):
    print(i, 'бомм!')

print('С Новым годом!')
# Будет напечатано:
# 12
# 11
# <тут цикл итерирует значения от 10 до 4>
# 3
# 2
# 1
```

Обрати внимание: `reversed()`, как и `range()` использовали сразу в условии цикла.

Пример со списком:

```
# Можно обратить вспять обычный список:
seasons = ['зима', 'весна', 'лето', 'осень']

for season in reversed(seasons):
    # Внутреннюю переменную цикла назвали season
    print(season)

# Будет напечатано:
# осень
# лето
# весна
# зима
```

Срез списка

Срез (на англ. «slice» — кусочек) создаёт подмножество внутри списка. Скажем, был список от 1 до 20. Из него выбрали кусочек — от 5 до 10. Это и будет срез.

Его используют, когда нужно поработать с некоторыми из элементов. Например, с каждым вторым или с первого по двадцатый.

Срез не извлекает элементы из списка, а просто копирует их.

Как создать срез

Для этого нужно:

- Указать имя списка. Например, `buttons`.
- Поставить квадратные скобки `[]`.
- В скобках передать аргументы через двоеточие. Аргументы в таком порядке: индекс начала, индекс конца, величина шага.

В итоге запись выглядит так: `<список>[<индекс начала>:<индекс конца>:<величина шага>]`.

Например, можно сделать срез знакомого тебе списка с кнопками:

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

print(buttons[1:4:2])
# Индекс начала — индекс второго элемента
# Индекс конца — индекс последнего элемента списка
# Указали индекс «Туры», но элемент не попадёт в срез
# Шаг = 2

# Будет напечатано: ['Авиа', 'Автобусы']
```

❌ Последний индекс среза в него **не входит**.

В этом срез похож на функцию `range()`. И тут последний элемент последовательности как бы держит красный знак «СТОП». Поэтому срез его не включает.

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

print(buttons[0:3:1])
# Указали индекс первого элемента 0, индекс последнего 3 и шаг = 1
# Индексу 3 соответствует элемент «Автобусы»
# Но срез остановится на «Ж/д», «Автобусы» в него не войдут

# Будет напечатано: ['Отели', 'Авиа', 'Ж/д']
```

Значения среза по умолчанию

Чтобы создать срез, необязательно передавать все три аргумента: и начало, и конец, и величину шага. Некоторые из них Python может задать сам.

1. **Если шаг равен единице, его не указывают.**

Для Python `print(buttons[1:3])` и `print(buttons[1:3:1])` — это одно и то же.

Программа не заметит разницы, а ты облегчишь свой код.

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

print(buttons[1:3])
# Пусть срез начинается с «Авиа» и кончается «Ж/д»
# Шаг автоматически = 1

# Будет напечатано: ['Авиа', 'Ж/д']
```

2. **Если индексы начала и конца среза совпадают с индексами первого и последнего элемента списка, их тоже не указывают.** Это значения по умолчанию.

Например, срез `buttons` с шагом = 2 выглядит так: `print(buttons[::2])`. Тут прописана только величина шага. Индексы начала и конца пропущены, но программа сама их заполнит.

Заметь, индексов нет, а двоеточия на своих местах.

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

print(buttons[::2])
# Указали только величину шага
# Срез будет включать первый элемент и каждый третий

# Будет напечатано: ['Отели', 'Ж/д', 'Туры']

print(buttons[1:])
# Указали начало среза 1, чтобы исключить первый элемент списка
# Шаг автоматически = 1

# Будет напечатано: ['Авиа', 'Ж/д', 'Автобусы', 'Туры']

print(buttons[1]) # А вот это уже не срез
# Это вызов значения элемента по индексу
# За двоеточиями в срезе стоит внимательно следить
```

```
# Будет напечатано: Авиа
```

Обратный срез списка

Величина шага в срезе может быть отрицательным числом. Например, `numbers[::-1]`. Если напечатать такой срез, код выведет на экран весь список в обратном порядке.

Когда нужно пройти весь список в обратном направлении, не указывай индексы. Только отрицательный шаг.

```
numbers = [1, 2, 3, 4, 5]

reversed_numbers = numbers[::-1]

print(reversed_numbers)
# Будет напечатано: [5, 4, 3, 2, 1]

print(numbers[::-1]) # Можно и без промежуточной переменной
# Результат будет тот же
# Будет напечатано: [5, 4, 3, 2, 1]
```

Вложенные циклы

С помощью циклов автоматизируют задачи, которые завязаны на повторении однообразных действий. Но есть задачи, которые нельзя решить обычным циклом.

Пример. Представь трёхэтажный отель. На каждом этаже — четыре номера. Чтобы постояльцы жили в чистоте и уюте, номера нужно регулярно убирать.

Ты управляешь отелем и сообщаешь клининговой службе, какие номера нуждаются в уборке. Последовательность шагов при уборке такая:

```
Убрать Номер 1
Убрать Номер 2
Убрать Номер 3
Убрать Номер 4
Подняться на этаж выше
Убрать Номер 5
# и так далее
...
```

Можно создать один список и сохранить в нём все номера:

```
hotel_rooms = ['Номер 1', 'Номер 2', ...]
```

Но если так сделать, разделение номеров по этажам не будет учитываться. Из-за этого придётся вручную высчитывать, сколько номеров на этаже уже убрано и когда можно подниматься на следующий.

Поэтому следует создать три списка, по числу этажей в отеле: `rooms_floor1`, `rooms_floor2`, `rooms_floor3`.

```
rooms_floor1 = ['Номер 1', 'Номер 2', 'Номер 3', 'Номер 4']
rooms_floor2 = ['Номер 5', 'Номер 6', 'Номер 7', 'Номер 8']
rooms_floor3 = ['Номер 9', 'Номер 10', 'Номер 11', 'Номер 12']
```

Затем нужно создать список `hotel_floors`. В нём будут храниться списки номеров для каждого этажа отеля.

```
hotel_floors = [rooms_floor1, rooms_floor2, rooms_floor3]
```

Теперь есть список списков, который хранит другие списки. А раз есть вложенные списки, есть и **вложенные циклы**.

Цикл для задания «убрать все номера в отеле по порядку» будет выглядеть так:

```
rooms_floor1 = ['Номер 1', 'Номер 2', 'Номер 3', 'Номер 4']
rooms_floor2 = ['Номер 5', 'Номер 6', 'Номер 7', 'Номер 8']
rooms_floor3 = ['Номер 9', 'Номер 10', 'Номер 11', 'Номер 12']

hotel_floors = [rooms_floor1, rooms_floor2, rooms_floor3]

for floor_rooms in hotel_floors: # Запустили внешний цикл по списку hotel_floors
    # тело внешнего цикла
    for room in floor_rooms: # Создали вложенный цикл для хождения по списку номеров, который записан в переменную
        # тело вложенного цикла
        print('Убрать', room) # Вывели номер номера на каждом шаге вложенного цикла
    # тело внешнего цикла
    print('Подняться на этаж выше') # Вывели сообщение о переходе на следующий этаж на каждом шаге внешнего цикла
```

Разберём код по кусочкам:

1. Сначала объявили три списка, в каждом из которых хранятся номера одного из этажей. Сами этажи сохранили в переменной `hotel_floors`.
2. Запустили **внешний цикл** `for` для главного списка с этажами: `for floor_rooms in hotel_floors`.

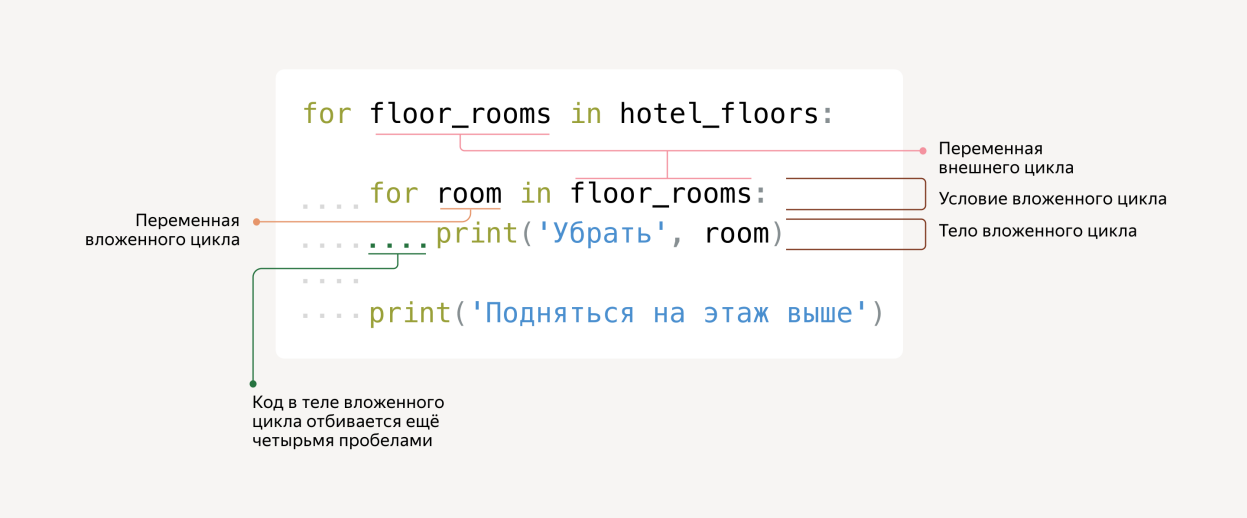
Переменная, в которую будут поступать все этажи, — `floor_rooms`. Это тоже список, который содержит номера конкретного этажа одного из списков `room_floor`.

3. Дальше **в теле внешнего цикла** написали ещё один цикл, но уже для переменной `floor_rooms`. Это **вложенный цикл**: `for room in floor_rooms`.

В новую переменную `room` записали по очереди значение каждого номера на этаже.

4. **В теле вложенного цикла** сделали вывод нужного номера на экран: `print('Убрать', room)`.
5. Перешли в тело внешнего цикла. Там написали код: `print('Подняться на этаж выше')`.

Его программа будет исполнять каждый раз, как закончит вложенный цикл.



Когда все номера на этаже выведены на экран, вложенный цикл завершается. Тогда код возвращается обратно в тело основного цикла этажей. Там он печатает сообщение «Подняться на этаж выше». И сразу попадает на следующий шаг внешнего цикла.

```
Убрать Номер 1
Убрать Номер 2
Убрать Номер 3
Убрать Номер 4
Подняться на этаж выше
Убрать Номер 5
Убрать Номер 6
Убрать Номер 7
Убрать Номер 8
Подняться на этаж выше
Убрать Номер 9
Убрать Номер 10
Убрать Номер 11
Убрать Номер 12
Подняться на этаж выше
```

Цикл с условием

Иногда нужно перебрать все элементы последовательности, а произвести операции — только с некоторыми из них. Например, найти в таблице базы данных определённое значение и вывести его на экран.

Тогда следует перебрать все элементы и найти те, которые подходят по критериям. Например, все чётные числа.

Для таких задач можно использовать ветвление кода с помощью конструкции `if...else`. Её можно применить внутри цикла.

Пример. Твои вещи лежат в первом шкафчике, в ячейке номер три. Объявим список `lockers` с номерами ячеек и циклом найдём среди них нужную.

```
lockers = [1, 2, 3, 4, 5]

for locker in lockers:
    # Тело цикла
    if locker == 3: # Вложенная конструкция if...else
        print('А вот и мой чемодан!') # Когда ячейка найдена, код печатает сообщение
```

В теле цикла код перебирает номера ячеек. При этом он сравнивает номер текущей ячейки с нужным. Когда цикл дойдёт до ячейки с номером три, он напечатает сообщение.

Ключевые слова `break` и `continue` в циклах

Когда в цикле есть условие `if`, программа ищет значения, с которыми нужно что-то сделать. Например, напечатать. Для этого она перебирает всю последовательность — элемент за элементом.

Есть проблема: программа продолжает работать, даже когда уже нашла нужное.

Пример. Представь, что ты ищешь ячейку со своим чемоданом, но не знаешь её номер. Тебе нужно пройти циклом все ячейки в камере хранения. Всего их 999.

Твои вещи по-прежнему лежат в ячейке номер три. Ты доходишь до третьей ячейки, понимаешь: «Ага, вот и она!». Но не открываешь её, а идёшь дальше. Обходишь шкаф за шкафом, пока не переберёшь их все.

Именно так обычно действует цикл:

```
lockers = range(1, 1000) # Диапазон с номерами ячеек в камере хранения

for i in lockers:
    if i == 3: # Вложенное условие
        print('А вот и мой чемодан!') # Когда оно выполнится,
                                      # код напечатает сообщение

    print('Осматриваю', str(i), 'ячейку')
    # А эта строка напечатает все значения переменной i
    # Код в любом случае обработает весь диапазон
    # Но так будет видно, сколько лишней работы он проделает
```

Код выведет:

```
Осматриваю 1 ячейку
Осматриваю 2 ячейку
А вот и мой чемодан!
Осматриваю 3 ячейку
Осматриваю 4 ячейку
Осматриваю 5 ячейку
...
<Тут цикл итерирует значения от 5 до 996>
...
Осматриваю 997 ячейку
Осматриваю 998 ячейку
Осматриваю 999 ячейку
```

В Python есть инструмент, который позволяет оптимизировать работу цикла. Это **ключевые слова** `continue` и `break`. С ними цикл не будет обрабатывать заведомо ненужные элементы.

Пропуск итерации цикла: `continue`

Ключевое слово `continue` используют вместе с вложенным условием `if`, чтобы пропустить итерацию цикла.

Например. Создали список `sheep` и сохранили там числа от 1 до 5. Нужно напечатать циклом все значения, кроме третьего.

Для этого:

- создали вложенное условие `if i == 3`;
- **в теле условия** написали `continue`.

```
sheep = [1, 2, 3, 4, 5]

for i in sheep: # Внешний цикл
    if i == 3: # Вложенное условие: если переменная = 3,
        continue # итерацию нужно пропустить
    print(i, 'овечка')

print('А куда пропала одна овечка?') # Эта строка не относится к циклу
```

Код напечатает такое сообщение:

```
1 овечка
2 овечка
4 овечка
5 овечка
А куда пропала одна овечка?
```

Пояснение. Когда в переменную `i` попало значение `3`, условие `if i == 3` выполнилось. Слово `continue` дало команду пропустить эту итерацию.

Выход из цикла: `break`

Слово `break` прерывает выполнение цикла.

Например. В переменной `new_sheep` хранятся числа от 1 до 10. Нужно обработать циклом первые четыре значения и вывести их на экран. Затем цикл следует остановить.

Чтобы код выполнил эту задачу:

- создали вложенное условие `if i == 5:`;
- в теле условия написали слово `break`.

```
new_sheep = range(1, 11) # Диапазон чисел от 1 до 10

for i in new_sheep:
```

```
    if i == 5:
        break
    print(i, 'овечка')

print('А где все остальные?')
```

В итоге код напечатает:

```
1 овечка
2 овечка
3 овечка
4 овечка
А где все остальные?
```

Пояснение. Когда в переменную `i` попало значение `5`, условие `i == 5` выполнилось. Слово `break` дало команду прекратить выполнение цикла.

⏳ Использование ключевых слов избавляет программу от ненужной работы. А тебе — экономит время.

Слова `break` и `continue` во вложенном цикле

Когда работаешь с вложенными циклами, обращай внимание на расположение ключевых слов.

Если использовать ключевое слово во вложенном цикле, оно будет относиться только к нему. И на работу внешнего цикла не повлияет.

Пример. Внешним циклом нужно посчитать овец. Их будет всего четыре. Значения будут передаваться в переменную `i`.

Вложенный цикл должен посчитать коров. Их всего девять. Значения будут передаваться в переменную `j`. Но коров с третьей по девятую на выгул не повели. Поэтому цикл следует остановить на втором элементе.

```
for i in range(1, 5): # Внешний цикл
    print('Овца', i) # Тело внешнего цикла

    for j in range(1, 10): # Вложенный цикл
        print('Корова', j) # Тело вложенного цикла

        if j == 2: # Условие вложенного цикла
            break # Ключевое слово для остановки цикла
```

Получится такое сообщение:

```
Овца 1 # Результат внешнего цикла
Корова 1 # Результат вложенного цикла
Корова 2 # Вложенный цикл не итерируется дальше второго значения
Овца 2
Корова 1
Корова 2
```

```
Овца 3 # А внешний цикл не остановился на втором значении
Корова 1
Корова 2
Овца 4 # И пошёл по диапазону дальше
Корова 1
Корова 2
```