Python: Цикл While

Программы, которые мы пишем на курсе, становятся сложнее и объемнее. Они еще далеки от реальных программ, хотя уже заставляют напрячься.

В этом уроке мы переходим к одной из самых сложных базовых тем в программировании — **циклам**.

Прикладные программы помогают управлять сотрудниками, финансами и могут развлекать. Несмотря на различия, они выполняют заложенные в них алгоритмы, которые похожи. **Алгоритм** — это последовательность действий, которая приводит к ожидаемому результату.

Представим, что у нас есть книга, и мы хотим найти в ней конкретную фразу. Саму фразу мы помним, но не знаем, на какой она странице. Нам придется последовательно просматривать страницы до тех пор, пока не найдем нужную. Этот процесс и называется алгоритмом.

Алгоритм включает логические проверки и перебор страниц. Количество страниц, которое придется посмотреть, заранее неизвестно. Но сам процесс просмотра повторяется одинаковым образом. Чтобы выполнять повторяющиеся действия, нужны циклы. Каждый повтор называется **итерацией**.

Напишем функцию с простым циклом, который будет `n` раз выводить на экран строку `'Hello!'`:

```
def print_hello(n):
    counter = 0
    while counter < n:
        print('Hello!')
        counter = counter + 1

print_hello(2)
# => Hello!
# => Hello!
```

Теперь проанализируем пример функции с циклом, который выводит на экран числа от одного до числа-аргумента:

```
print_numbers(3)
# => 1
# => 2
# => 3
```

Эту функцию невозможно реализовать уже изученными средствами, так как количество выводов на экран заранее неизвестно. А с циклами проблем не будет:

```
def print_numbers(last_number):
    # i сокращение от index (порядковый номер)
    # используется по общему соглашению во множестве языков
```

```
    # как счетчик цикла
    i = 1
    while i <= last_number:
        print(i)
        i = i + 1
    print('finished!')

print_numbers(3)
# => 1
# => 2
# => 3
# => finished!
```

https://replit.com/@hexlet/python-basics-loops-while

Цикл `while` состоит из трех элементов:

Каждое выполнение тела называется **итерацией**. В примере выше `print_numbers(3)` вызвал три итерации, на каждой из которых была выведена на экран переменная `i`. Конструкция читается так: «делать то, что указано в теле цикла, пока истинно условие `i <= last_number`».

Разберем работу этого кода для вызова `print_numbers(3)`:

```
# Инициализируется i
i = 1

# Предикат возвращает true, поэтому выполняется тело цикла
```

```
while 1 <= 3
# print(1)
# i = 1 + 1

# Закончилось тело цикла, поэтому происходит возврат в нача
while 2 <= 3
# print(2)
# i = 2 + 1

# Закончилось тело цикла, поэтому происходит возврат в нача
while 3 <= 3
# print(3)
# i = 3 + 1

# Предикат возвращает false, поэтому выполнение переходит з
while 4 <= 3

# print('finished!');
# На этом этапе i равен 4, но он нам уже не нужен
# Функция завершается
```

Процесс, который порождает цикл, должен остановиться. За это отвечает программист.

Обычно задача сводится к введению переменной — **счетчику цикла**. Сначала он инициализируется — ему задается начальное значение. В нашем примере это строчка `i = 1`. Затем в условии цикла проверяется, не достиг ли счетчик своего предельного значения.

Предельное значение в примере определяется аргументом функции. Если условие цикла не

выполнено, то тело не выполняется и интерпретатор двигается дальше — работает с инструкциями после цикла.

Если условие цикла истинно, то выполняется тело, в котором находится элемент остановки — изменение счетчика. Обычно его делают в конце тела, и это изменение — место, где нельзя обойтись без переменной. В примере выше за изменение отвечает строчка `i = i + 1`.

На этом моменте новички много ошибаются. Например, можно забыть увеличить счетчик или неправильно проверить его в предикате. Это приведет к зацикливанию — цикл будет работать бесконечно и программа никогда не остановится. В таком случае ее нужно завершить принудительно.

```
def print_numbers(last_number):
    i = 1
    # Этот цикл никогда не остановится
    # и будет печатать всегда одно значение
    while i <= last_number:
        print(i)
    print('finished!')
```

В некоторых случаях бесконечные циклы полезны. Мы не будем рассматривать такие ситуации, но покажем, как выглядит этот код:

```
while True:
  # Что-то делаем
```

Без циклов невозможно обойтись, когда алгоритм решения задачи требует повторения каких-то действий и количество этих операций заранее неизвестно.

Задание

Модифицируйте функцию `print_numbers()` так, чтобы она выводила числа в обратном порядке. Для этого нужно идти от верхней границы к нижней. То есть счётчик должен быть инициализирован максимальным значением, а в теле цикла его нужно уменьшать до нижней границы.

Пример вызова и вывода:

```
print_numbers(4)
```

```
4
3
2
1
finished!
```

▶ Упражнение не проходит проверку — что делать? 😶

▶ В моей среде код работает, а здесь нет 🤨

- ▶ Мой код отличается от решения учителя 🤔
- ▶ Прочитал урок — ничего не понятно 🙄

Полезное

- Цикл while

Определения

- Цикл While — инструкция для повторения кода, пока удовлетворяется какое-то условие.