Последовательности и циклы

Умение работать с последовательностями и циклами позволяет разработчику эффективно обрабатывать много данных и выполнять одинаковые задачи много раз. Это делает программы быстрее и помогает автоматизировать повторяющуюся работу.

Последовательности

Примеры последовательностей

```
# Последовательность типа list (список); # значения элементов
списка - числа типа float. movie_ratings = [4.7, 5.0, 4.3, 3.8]

# Последовательность типа list (список); # значения
элементов списка - строки (str). movies = ['Матрица',
'Хакеры', 'Трон']

# Последовательность типа строка (str);
# элементы этой последовательности - символы, составляющие строку. name_movie = 'Джонни Мнемоник'

# Последовательности могут состоять из элементов разных типов.
# Например, одновременно элемент типа str и float. movie_info = ['Трон', 4.7]
# Элементами последовательности могут быть другие последовательности. movies_info = [['Трон', 4.7],
['Хакеры', 5.0], ['Матрица', 4.5]]
```

Адресация элементов

Доступ к значениям элементов последовательности можно получить через их индекс — целое число, обозначающее позицию элемента. Отсчёт индексов начинается с нуля.

```
# Чтобы получить значение определённого элемента,
# его индекс следует указать в квадратных скобках после имени последовательности.
print(movie_ratings[2])
# Вывод в терминал: 4.3.

print(name_movie[0])
# Вывод в терминал: Д

# При попытке обратиться несуществующему элементу интерпретатор вернет ошибку.
print(movies[10])
# Вывод в терминал: IndexError: list index out of range.
```

Сравнение последовательностей

При сравнении последовательностей поочерёдно сравниваются элементы с одинаковыми индексами, и при первом же различии определяется результат сравнения. Результатом сравнения будет логическое значение True или False.

```
print([1, 2, 3] < [1, 2, 4])
# Вывод в терминал: True
```

При сравнении строк принимаются во внимание символы и их регистр. У всех текстовых символов есть числовая кодировка:

- спецсимволы, знаки препинания и цифры: начинается с 32
- латиница — заглавные буквы: A =65
- латиница — строчные буквы: a =97
- кириллица — заглавные буквы: А =1040
- кириллица — строчные буквы: а =1072

```
print('Слон' < 'слон')
# Вывод в терминал: True
# Элемент с индексом 0 первой строки - это буквенный символ в верхнем регистре,
# его числовой код меньше той же буквы в нижнем регистре.

print('1Ь' > 'fЬ')
# Вывод в терминал: False
# Элемент с индексом 0 первой строки - цифра. Её числовой код меньше, чем у букв.
```

Длина последовательности при сравнении не имеет значения. Как только Python найдёт неравные элементы в последовательности, отношение элементов и определит результат сравнения.

```
print('a' < 'ADC')
# Вывод в терминал: False

print([1, 2, 4] > [1, 2, 3, 4])
# Вывод в терминал: True
```

Коды символов можно найти в таблицах кодировок Unicode. В Python эти коды можно получить через функцию ord().

```
print(ord('Ф'))
# Вывод в терминал: 1060
```

Операторы < > <= >= поддерживаются только при сравнении последовательностей одного типа и однотипных элементов в этих последовательностях.

Наименьшее и наибольшее значение элементов последовательности

В последовательностях, содержащих элементы одного типа, можно найти элементы с наименьшим и наибольшим значением. Для этого применяют встроенные методы min() и max().

```
my_string = '12345'
print(max(my_string))
# Вывод в терминал: 5

my_list = ['abc', 'Abc']
print(min(my_list))
# Вывод в терминал: Abc
```

Слияние (конкатенация)

Конкатенация (объединение двух последовательностей в одну) выполняется с помощью оператора + Объединять возможно только последовательности одного типа.

Результатом конкатенации будет новый объект.

```
first_baggage_list = ['Диван', 'Чемодан', 'Саквояж', 'Картина']

second_baggage_list= ['Корзина', 'Картонка', 'Маленькая собачонка']

full_baggage_list = first_baggage_list + second_baggage_list
print(full_baggage_list)
```

Повторение

Если умножить последовательность на число, то создастся новый объект, в котором элементы исходной последовательности будут повторяться указанное число раз.

```
pump = 'насос' * 4
print(pump)
# Вывод в терминал: насоснасоснасоснасос
```

Количество элементов последовательности

Чтобы узнать количество элементов последовательности — применяют функцию len(). Отсчёт индексов начинается с нуля.

```
movies = ['Матрица', 'Хакеры', 'Трон', 'Тихушники', 'Сеть']
print(len(movies))
# Вывод в терминал: 5
```

Проверка наличия элемента в последовательности

Для этого применяют оператор in, он возвращает True или False.

```
movie_ratings = [4.7, 5.0, 4.3, 3.1]
print(4.7 in movie_ratings)
# Вывод в терминал: True
```

С помощью оператора in можно проверить и отсутствие элемента в последовательности:

```
full_baggage_list = ['Диван', 'Чемодан', 'Саквояж', 'Картина', 'Корзина', 'Картонка']

# Если собачонки нет...
if 'Маленькая собачонка' not in full_baggage_list:
    # ...устраиваем скандал:
    print('— Товарищи! Где собачонка?')
```

Срезы

Из последовательности можно извлечь набор элементов по определённым условиям; результат такой операции называется срез.

Чтобы из последовательности sequence получить срез, в который войдут элементы, расположенные один за другим, применяют синтаксис sequence[start:end], где start и end — индексы элементов, определяющие диапазон, из которого будет взят срез.

```
name_movie = 'Джонни Мнемоник'
print(name_movie[2:10])
# Вывод в терминал: онни Мне
```

Для получения среза можно указать только начальный или конечный индекс элемента, тогда второй границей среза будет конец или начало исходной последовательности.

```
name_movie = 'Джонни Мнемоник'
# Взять срез с седьмого элемента и до конца последовательности.
print(name_movie[7:])
# Вывод в терминал: Мнемоник

name_movie = 'Джонни Мнемоник'
# Взять срез от начала последовательности до шестого элемента (не включая шестой).
print(name_movie[:6])
# Вывод в терминал: Джонни
```

Чтобы получить срез из элементов, расположенных с определённым шагом, применяют синтаксис sequence[start:end:step]

```
movies = ['Матрица', 'Хакеры', 'Трон', 'Тихушники', 'Сеть']
print(movies[0:5:2])
# Вывод в терминал: ['Матрица', 'Трон', 'Сеть']
```

Можно получить срез, где последовательность элементов будет инвертирована. Для этого сначала указывается конечный индекс исходной последовательности, потом начальный, а шаг указывается отрицательным числом:

```
movies = ['Матрица', 'Хакеры', 'Трон', 'Тихушники', 'Сеть']
# Шаг должен иметь отрицательное зачение print(movies[5:0:-1])
# Вывод в терминал: ['Сеть', 'Тихушники', 'Трон', 'Хакеры']

# Можно не указывать границы, а задать только отрицательное значение шага print(movies[::-1])
# Вывод в терминал: ['Сеть', 'Тихушники', 'Трон', 'Хакеры', 'Матрица'] # Инвертирован весь список
```

Циклы

Каждое повторное выполнение действия или ряда действий называют итерацией, а для работы с итерациями применяют циклы.

В Python существует два типа циклов: for и while.

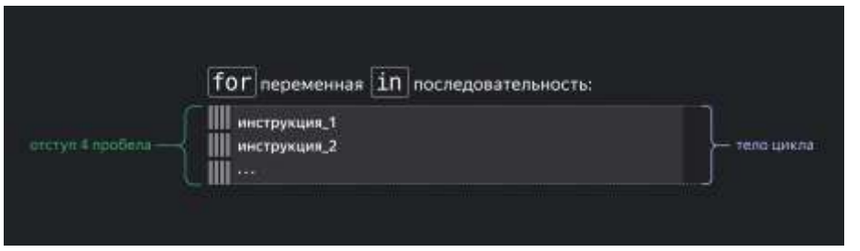
Цикл for

Используется, когда есть блок кода, который нужно выполнить фиксированное и заранее известное количество раз. Этот цикл всегда используется в сочетании с итерируемым объектом, таким как список или диапазон.

При объявлении цикла for указывают:

- переменную, в которую на каждой итерации будет передаваться очередной элемент последовательности;
- имя последовательности, которую нужно перебрать.

Синтаксис цикла for:



Пример использования:

```
movie_ratings = [4.7, 5.0, 4.3, 3.1]

# Переменная rating поочерёдно принимает элементы последовательности movie_ratings.
for rating in movie_ratings:
    # Теперь переменную rating можно обработать в теле цикла.
    if rating > 4.7:
        print('Фильм крут')
```

Итерация по диапазону чисел

Объект типа Range (с англ. «диапазон») — это числовой диапазон, заданный по определённым правилам. Он итерируемый, и с ним можно работать так же, как и с коллекциями.
Объект типа Range создаётся при вызове функции range(); границы создаваемого диапазона передаются в аргументах:

```
simple_range = range(1, 10)
print(type(simple_range))
# Вывод в терминал: <class 'range'>
```

Значение, определяющее верхнюю границу диапазона, не попадает в возвращаемую последовательность: range(110 создаст диапазон от 1 до 9

У функции range() может быть от одного до трёх аргументов, например:

- range(15 вернёт последовательность целых чисел от 0 до 14 включительно;

- range(315 вернёт последовательность целых чисел от 3 до 14 включительно;
- range(3152 вернёт последовательность целых чисел от 3 до 14 с шагом 2 это будет 3 5791113
  А вот range(153 -2 вернёт 151311975

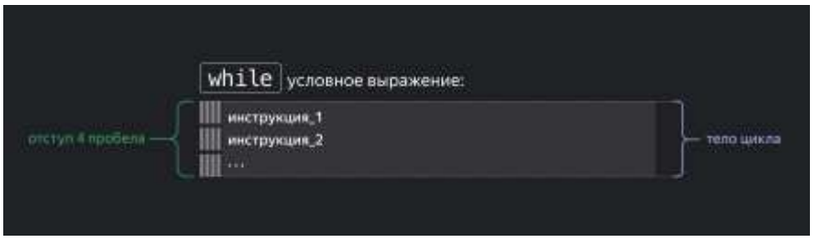
Чтобы из объекта Range получить конкретные значения — нужно явно обратиться к нужному элементу:

```
simple_range = range(1, 10, 2)
print(simple_range[3])
# Вывод в терминал: 7
```

Цикл while

Этот цикл используется в ситуациях, когда количество итераций неизвестно. Цикл будет выполнять вложенный в него код до тех пор, пока (англ. while) будет соблюдаться условие, заданное при его объявлении. Например, «выполняй цикл, пока не наступит полночь!».

Синтаксис цикла while:



Пример использования:

```
from random import choice

movies = ['Матрица', 'Хакеры', 'Трон', 'Тихушники', 'Сеть']
movie = choice(movies)
print('Какой фильм мы будем смотреть?')
print(movies)
answer = input()  # Ожидаем ввод названия фильма от пользователя.
while answer != movie: # Если не угадал,
    answer = input('Попробуй еще: ')  # то ждём ввода нового названия.
print('Молодец, угадал!')
```