InnoDom

Циклы

Цикл

это **процесс повторения** какого-либо действия или обход по какому-нибудь итерированному объекту.

Итерируемый объект – это объект, у которого можно брать элементы по одному (строки, списки, кортежи, множества, словари и др).
Итерация – это шаг или организация обработки данных, таким образом, когда действие происходит многократно.

Циклы

For

- цикл, срабатывающий **определенное** количество раз;
- цикл с **параметром**;
- цикл, с **известным** количеством итераций

While

- цикл, работающий по какому-то **условию**;
- цикл с **условием**;
- цикл, с **неизвестным** количеством итераций

Цикл for

Позволяет проходится только по **итерируемым** объектам, для создания цикла используется следующая схема:

```
for i in (итерируемый объект):
   действие
```

`i` - это элемент **объекта** на каждой итерации. Переменная `i` существует только в цикле и её принято называть **локальной** переменной цикла. Называться она может как угодно, но принято называть в зависимости от того, что хранит цикл.

Цикл for

Создание цикла:

```
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in list1:
    print(i)
```

Цикл проходится по 1 элементу из списка list1, итераций по списку 10, т.к 10 элементов в списке. Результат - это вывод каждого числа по отдельности.

Цикл for

Обход по **кортежу** и **множеству** схожи с обходом по списку:

```
tuple1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
for i in tuple1:
    print(i)

set1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
for i in set1:
    print(i)
```

Цикл for

У словарей обход можно делать по **ключам**, по **значениям** и по парам **ключ: значение** одновременно:

```
dict1 = dict(zip(range(10, 21), range(1, 11)))
for key in dict1.keys():
    print(key)
for val in dict1.values():
    print(val)
for key, val in dict1.items():
    print(key, val)
```

Цикл for

не всегда нужно взаимодействовать с элементами итерируемого объекта, можно просто произвести какое-нибудь действие несколько раз. В этом помогает функция `range()`. Благодаря ей можно смоделировать ситуацию повторения, хоть и проходимся по итерируемому объекту, который возвращает функция `range()`.

```
for i in range(1, 11):
    print('-_-')
```

Цикл while

Есть задачи, которые трудно решить через `for`, но легко через **`while`**.
Схема выглядит так:
while условие:
    действие

Создание цикла:

```
i = 0
while i < 10:
    print(i)
    i += 1
```

Циклы и индексы

Помимо обхода итеративных объектов через цикл for, можно ещё использовать обход по **индексам**.

```
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in range(0, len(list1), 1):
    print(list1[i])
```

Мы создали цикл от 0 до длины массива (второй аргумент функции не включается, а нумерация по индексам начинается с 0) с шагом 1 (по умолчанию 1, писать не обязательно) и выводим элементы по индексу. Такие ситуации спасают, когда нужно работать с соседними от индекса i-элементами в списке.

Циклы и индексы

Мы создали цикл, длину отняли на 1, чтобы посмотреть следующий элемент в цикле, иначе на последней итерации сработает исключение `IndexError` (такого элемента не существует) и выводим на экран текущий элемент и следующий в списке.

```
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in range(0, len(list1)-1, 1):
    print(list1[i]), list[i+1]
```

Циклы и индексы

Благодаря индексам, можно быстро обойти список в обратном порядке.

```
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in range(len(list1)-1, -1, -1):
    print(list1[i])
```

Операторы break и continue

В любой момент выполнения программы, при определённом условии, можно **пропустить** текущую итерацию, для используется оператор **`continue`**.
Чтобы **закончить** выполнение цикла, используется оператор **`break`**.

Например, когда `i` становится равным **7**, цикл **заканчивает** своё выполнение:

```
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in list1:
    if i == 7:
        break
    print(i)
```

Операторы break и continue

Например, когда `i` становится равным **5**, итерация **пропускается**, а цикл продолжает работать дальше:

```
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in list1:
    if i == 5:
        continue
    print(i)
```

Операторы break и continue

`break` и `continue` также работают в цикле `while`

Также можно создавать зацикливание или **бесконечный цикл**. А ограничивать работу цикла операторами `break` и `continue`.

Создание **бесконечного** цикла:

```
while True:
    print("-_0")
```

Операторы break и continue

Цикл можно **ограничить**. Например, цикл заканчивает работать после 100 итерации:

```
i = 0
while True:
    if i < 100:
        print("-_0")
    else:
        break
     i+=1
```

Оператор else в циклах

В конце любого цикла, т.е. после *последней итерации*, можно выполнять какое-либо действие. Для этого в цикле существует блок `else`.

```
for i in range(1, 10):
    print('o_0')
else:
    print('9_0')
```

Оператор else в циклах

Вложенные циклы используются тогда, когда нужно пройтись по многоуровневым спискам либо повторить действие в действии.

```
list1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9, 10]]
for i in list1:
    for j in i:
        print(j, end=" ")
    print()
```

! Будьте осторожны, вложенность всегда сказывается на быстродействии вашего приложения.

Пользовательский ввод элементов в список

Пример пользовательского ввода чисел в список:

```
list 1 = []
for i in range(int(input("Введите длину списка: "))):
    list1.append(int(input(f'Введите {i+1} элемент списка: ')))
print(list1)
```