Циклы

Представим, что у нас есть список сортов мороженного, которое продается в киоске. Перед нами стоит задача вывести в консоль все эти сорта мороженного по одному. Тогда код будет выглядеть следующим образом:

```
>>ice_cream = ['клубничное', 'фисташковое', 'шоколадное', 'малиновое', 'блю курасава', 'лимонное', 'ванильное']
>>print(ice_cream[0])
>>print(ice_cream[1])
>>print(ice_cream [2])
>>print(ice_cream [3])
>>print(ice_cream [4])
>>print(ice_cream [5])
>>print(ice_cream [6])
```

Это кажется неэффективным и громоздким. В Python (и большинстве других языках программирования) существует более простой способ перебора (итерации) каждого элемента в списке. Мы можем использовать циклы! Цикл - это способ многократного повторения программного кода.

В этом занятии будет объяснено:

• Циклы, которые позволяют перебирать каждый элемент списка, обозначаемые ключевым словом for.

• Циклы, которые повторяют набранный код, пока не выполнится какое-либо условие, обозначаемые ключевым словом while.

• Циклы, которые создают новые списки, называемые составления списков.

Создание цикла for

Возможно вывести каждый элемент списка с помощью цикла for. Цикл for позволяет нам выполнять действие с каждым элементом в списке. Использование каждого элемента списка называется повторением или итерацией (типы данных, к которым применимы циклы, называются итеративными).

Создадим цикл для вывода сортов мороженного из предыдущего примера:

```
>>ice_cream = ['клубничное', 'фисташковое', 'шоколадное', 'малиновое', 'блю курасава', 'лимонное', 'ванильное']
>>for item in ice_cream:
>>    print(item)
```

Общий синтаксис создания цикла for:

```
>>for <temporary variable> in <list variable>:
>>    <action>
```

В нашем примере с мороженным item была временной переменной, ice_cream - переменной списка, а print (item) - действием, выполняемым над каждым элементом в списке.

Наша временная переменная может иметь любое имя, и ее не нужно определять заранее. Каждый из следующих фрагментов кода делает то же самое, что и наш пример:

```
>>for i in ice_cream:
>>    print(i)
>>for ice_cream_sort in ice_cream:
>>    print(ice_cream_sort)
```

Обратите внимание, что во всех этих примерах оператор print имеет отступ. Все, что находится на том же уровне отступа после объявления цикла for, включается в цикл for и запускается на каждой итерации.

Если мы забудем сделать отступ, мы получим ошибку IndentationError.

Задание

1.Запустите код ниже. Вы должны получить ошибку IndentationError, потому что строка печати (игра) не имеет отступа.

```
board_games = ['Settlers of Catan', 'Carcassone', 'Power Grid', 'Agricola', 'Scrabble']
sport_games = ['football', 'football - American', 'hockey', 'baseball', 'cricket']
for game in board_games:
print(game)
```

2.Сделайте отступ в строке 6, чтобы при запуске кода не возникала ошибка IndentationError.

3. Напишите цикл, который выводит каждый вид спорта в sport_games.

Использование range в циклах

Ранее с помощью цикла for перебирались элементы существующего списка.

Иногда существует необходимость перебора не конкретного списка, а выполнения определенного действия несколько раз. Например, если мы хотим вывести в консоль «ВНИМАНИЕ!» сообщение три раза, мы хотели бы сказать что-то вроде:

```
>>for i in <a list of length 3>:
>>    print("ВНИМАНИЕ!")
```

Обратите внимание, что необходимо перебирать список длиной 3, но нам все равно, что в списке. Чтобы создать эти списки длины n, мы можем использовать функцию range. Range принимает на вход число n и возвращает список от 0 до n-1. Например:

```
>>zero_thru_five = range(6)
>># zero_thru_five is now [0, 1, 2, 3, 4, 5]
>>zero_thru_one = range(2)
>># zero_thru_one is now [0, 1]
```

Итак, простой способ выполнить наше «ВНИМАНИЕ!» пример будет:

```
>>for i in range(3):
>>   print("ВНИМАНИЕ!")
```

Задание

1. Используйте функцию диапазона в цикле for, чтобы распечатать обещание 5 раз.
promise = "I will not chew gum in class"

Бесконечные циклы

В примерах выше осуществлялся перебор элементов таких списков, у которых есть фиксированную длину (начало и конец). Однако давайте рассмотрим этот пример:

```
>>my_favorite_numbers = [4, 8, 15, 16, 42]
>>for number in my_favorite_numbers:
>>        my_favorite_numbers.append(1)
```

Что здесь происходит? Каждый раз, когда мы входим в цикл, мы добавляем 1 в конец списка, элементы которого мы перебираем. В результате мы никогда не доходим до конца списка! Он продолжает расти!

Цикл, который никогда не завершается, называется бесконечным циклом. Это очень опасно для кода!

Программа, которая попадает в бесконечный цикл, часто становится полностью непригодной для использования. Необходимо избегать бесконечных циклов. Но если вы случайно наткнетесь на один из них, вы можете завершить цикл, нажав Ctrl + c для завершения программы.

Задание

1. Предположим, у нас есть два списка студентов: student_period_A и student_period_B. Мы хотим объединить всех студентов в student_period_B.

Напишите цикл for, который проходит через каждого студента в student_period_A и добавляет его в конец student_period_B.

```
students_period_A = ["Alex", "Briana", "Cheri", "Daniele"]
students_period_B = ["Dora", "Minerva", "Alexa", "Obie"]
```

2. Внутри цикла for, после добавления student к student_period_B, выведите student.

3. Предположим, вы допустили опечатку в теле цикла for.

Внутри цикла for измените объект оператора добавления с student_period_B на student_period_A. В данном случае вами будет получен бесконечный цикл

Выйдите из бесконечного цикла! Затем избавьтесь от ошибки, которая вызвала бесконечный цикл.

Break

Мы часто хотим использовать цикл for для поиска некоторого значения в списке:

```
>>items_on_sale = ["blue_shirt", "striped_socks", "knit_dress", "red_headband",
"dinosaur_onesie"]
>># Мы хотим проверить наличие товара с ID "knit_dress":
>>for item in items_on_sale:
>>        if item == "knit_dress":
>>        print("Knit Dress is on sale!")
```

Этот код просматривает каждый элемент в items_on_sale и проверяет соответствие. После того, как мы обнаружим, что knit_dress находится в списке items_on_sale, нам не нужно просматривать остальную часть списка items_on_sale. Поскольку он состоит всего из 5 элементов, в этом случае не составляет большого труда перебирать весь список. Но что, если у items_on_sale после "knit_dress" будет 1000 наименований? Что, если бы после "knit_dress" было 100 000 предметов?

Вы можете остановить цикл for изнутри цикла, используя break. Когда программа встречает оператор break, управление возвращается коду вне цикла for. Например:

```
>>items_on_sale = ["blue_shirt", "striped_socks", "knit_dress", "red_headband",
"dinosaur_onesie"]

>>print("Checking the sale list!")
>>for item in items_on_sale:
>>        print(item)
>>        if item == "knit_dress":
>>        break
>>print("End of search!")
```

Это даст результат:

```
Checking the sale list!
blue_shirt
striped_socks
knit_dress
End of search!
```

Нам вообще не нужно было проверять "red_headband" или "dinosaur_onesie"!

Continue

Когда мы перебираем списки, мы можем захотеть пропустить некоторые значения. Допустим, мы хотим распечатать все числа в списке, если они не отрицательные. Мы можем использовать continue для перехода к следующему i в списке:

```
>>big_number_list = [1, 2, -1, 4, -5, 5, 2, -9]

>>for i in big_number_list:
>>        if i < 0:
>>                continue
>>        print(i)
```

Это даст результат:

```
1
2
4
5
2
```

Задание

1. У вас есть список пород собак, из числа которых вы можете взять себе питомца, dog_breeds_available_for_adoption. Используя цикл for, выполните итерацию по списку dog_breeds_available_for_adoption и распечатайте каждую породу собак.

```
dog_breeds_available_for_adoption = ['french_bulldog', 'dalmatian', 'shihtzu', 'poodle', 'collie'
]
dog_breed_I_want = 'dalmatian'
```

2. Внутри цикла for проверьте, совпадает ли текущее значение породы с dog_breed_I_want. Если да, то выведите "У них есть собака, которую я хочу!"
3. Добавьте оператор break, когда ваш цикл обнаружит dog_breed_I_want, чтобы не нужно было проверять остальную часть списка.

While циклы

Существует другой тип цикла, называемый циклом while. Цикл while выполняет код до тех пор, пока не будет достигнуто какое-либо условие.

Циклы while можно использовать для перебора элементов списков, как и для циклов for:

```
>>dog_breeds = ['bulldog', 'dalmation', 'shihtzu', 'poodle', 'collie']

>>index = 0
>>while index < len(dog_breeds):
>>        print(dog_breeds[index])
>>        index += 1
```

Каждый раз, когда выполняется условие цикла while (в данном случае index <len (dog_breeds)), выполняется код внутри цикла while.

Циклы while могут быть полезны, когда неизвестно, сколько итераций потребуется для выполнения условия.

Вложенные циклы

Мы видели, как можно перебирать элементы списка. Что, если у нас есть список, состоящий из нескольких списков? Как мы можем перебрать все отдельные элементы?

Предположим, мы отвечаем за научный класс, который разделен на три проектные группы:

```
>>project_teams = [["Ava", "Samantha", "James"], ["Lucille", "Zed"], ["Edgar", "Gabriel"]]
```

Если мы хотим получить каждого ученика, мы должны поместить один цикл внутрь другого:

```
>>for team in project_teams:
>>          for student in team:
>>                    print(student)
```

Это приводит к:

```
Ava
Samantha
James
Lucille
Zed
Edgar
Gabriel
```

Задание

Мы предоставили список sales_data, который показывает количество различных вкусов мороженого, проданных в трех разных местах вымышленного магазина. Мы хотим суммировать общее количество проданных сортов. Начните с определения переменной scoops_sold и установите ее равной нулю.

sales_data = [[12, 17, 22], [2, 10, 3], [5, 12, 13]]

2. Просмотрите список sales_data. Создайте цикл, перебирающий каждый список из списка sales_data .

3.В списке sales_data переберите значения внутри каждого вложенного списка из sales_data и прибавьте к своей переменной scoops_sold.

К концу у вас должна быть сумма всех чисел во вложенном списке sales_data.

4. Выведите значение scoops_sold.

Генераторы списков

Допустим, мы просмотрели определенный веб-сайт и получили следующие слова:

```
>>words = ["@coolguy35", "#nofilter", "@kewldawg54", "reply", "timestamp",
"@matchamom", "follow", "#updog"]
```

Мы хотим создать новый список, называемый именами пользователей, в котором будут все строки в словах с символом «@» в качестве первого символа. Мы знаем, что можем сделать это с помощью цикла for:

```
>>words = ["@coolguy35", "#nofilter", "@kewldawg54", "reply", "timestamp",
"@matchamom", "follow", "#updog"]
>>usernames = []
>>for word in words:
>>          if word[0] == '@':
>>                    usernames.append(word)
```

Сначала мы создали новый пустой список имен пользователей и, просматривая список слов, добавляли каждое слово, соответствующее нашему критерию. Теперь список имен пользователей выглядит так:

```
>>> print(usernames)
["@coolguy35", "@kewldawg54", "@matchamom"]
```

В Python есть удобное сокращение для создания подобных списков с одной строкой:

```
>>usernames = [word for word in words if word[0] == '@']
```

Это называется генератором списка. Результат будет тот же, что и в цикле for:

```
["@coolguy35", "@kewldawg54", "@matchamom"]
```

Понимание этого синтаксиса:

1. Принимает элемент в words

2. Назначает этот элемент переменной с именем word

3. Проверяет, есть ли слово [0] == '@', и если да, добавляет слово в новый список имен пользователей. В противном случае ничего не происходит.

4. Повторите шаги 1–3 для всех строк в словах.

Примечание: если бы мы не проводили никакой проверки (допустим, мы пропустили if word [0] == '@'), новый список был бы просто копией слов:

```
>>usernames = [word for word in words]
>>#usernames сейчас выглядит так ["@coolguy35", "#nofilter", "@kewldawg54", "reply",
"timestamp", "@matchamom", "follow", "#updog"]
```

Допустим, мы работаем со списком имен пользователей из последнего упражнения:

```
>>> print(usernames)
["@coolguy35", "@kewldawg54", "@matchamom"]
```

Мы хотим создать новый список со строкой «пожалуйста, следуй за мной!» добавляется в конец каждого имени пользователя. Мы хотим назвать этот новый список сообщениями. Мы можем использовать понимание списка, чтобы составить этот список из одной строки:

```
messages = [user + " please follow me!" for user in usernames]
```

Понимание этого списка:

1. Принимает строку в usernames.

2. Назначает эту строку переменной с именем user.

3. Добавляет «пожалуйста, следуйте за мной!» user

4. Добавляет это объединение в новый список под названием messages.

5. Повторяет шаги 1–4 для всех строк в `usernames`.

Теперь messages содержат следующие значения:

```
["@coolguy35 please follow me!", "@kewldawg54 please follow me!", "@matchamom please follow me!"]
```

Возможность создавать списки с измененными значениями особенно полезна при работе с числами. Допустим, у нас есть этот список:

>>my_upvotes = [192, 34, 22, 175, 75, 101, 97]

Мы хотим добавить 100 к каждому значению. Мы можем достичь этой цели одной строкой:

>>updated_upvotes = [vote_value + 100 for vote_value in my_upvotes]

Понимание этого списка:

1. Принимает число в my_upvotes

2. Присваивает это число переменной с именем vote_value.

3. Добавляет 100 к vote_value

4. Добавляет эту сумму в новый список updated_upvotes.

5. Повторяет шаги 1–4 для всех чисел в my_upvotes.

Теперь updated_upvotes содержит следующие значения:

```
[292, 134, 122, 275, 175, 201, 197]
```

Заключение

На этом уроке вы узнали

• как написать цикл for

• как использовать диапазон в цикле

• что такое бесконечные циклы и как их избежать

• как пропустить значения в цикле

• как написать цикл while

• как составлять списки из одной строки

Давайте еще попрактикуемся с этими концепциями!

Задание

1. Создайте список с именем single_digits, состоящий из чисел от 0 до 9 (включительно).

2. Создайте цикл for, который проходит через single_digits и выводит каждую из них.

3. Перед циклом создайте список под названием squares. Назначьте для начала пустой список.

4. Внутри цикла, который повторяет single_digits, добавьте значение квадрата каждого элемента single_digits к squares. Вы можете сделать это до или после вывода элемента.

5. После цикла for выведите squares.

6. Создайте список cubes, используя сравнение списков в списке single_digits. Каждый элемент cubes должен быть элементом single_digits, возведенным в куб.

7. Выведите cubes.