## Списки

Иногда необходимо объединять несколько объектов в одну структуру. Например, когда вы идете в магазин и вам дают список продуктов для покупки. Как было бы удобно объединить покупки в единый объект?

В данном разделе будет изучено:

- Что такое последовательные типы?
- Что такое список и как с ним работать?
- Методы обработки списков

## Список – это

Итак, предположим, вы пишите продукты, которые собираетесь купить. Ваш список имеет вид:

1. Хлеб 1 шт
2. Яйца 10 шт
3. Молоко 1 шт
4. Кофе 1 шт

Наименование товара в данном случае будет строкой, а количество штук товара, которое вы намереваетесь купить, число. В настоящий момент было изучено, как задавать переменные. Используя эти знания, список покупок будет выглядеть так:

>>item1 = ‘хлеб’
>>item2 = ‘яйца’
>>item3 = ‘молоко’
>>item4 = ‘кофе’

Если еще и записывать отдельно количество, то это будет еще ряд переменных, в которых будут записаны значения чисел, обозначающих требуемое количество продукта.

Однако это неудобно. Удобнее было бы объединить весь список продуктов вместе и записать в одну переменную. Для этого существует тип данных list (список). Списки относятся к последовательному типу данных (то есть хранит последовательность элементов) и представляет собой набор отдельных элементов, отделенных друг от друга запятыми. Список обозначается квадратными скобками [ ]. Для нашего примера с покупками список будет иметь вид:

>>shopping_list = [‘хлеб’, ‘яйца’, ‘молоко’, ‘кофе’]

Как видим из примера, shopping_list состоит из набора строк, обозначающими наименование отдельных товаров из списка покупок. Однако, так же необходимо записать не только наименования товаров, но и их количество. Возможно создать отдельный список:

>>quantity = [1, 10, 1, 1]

До этого мы рассмотрели два списка, которые состояли из объектов одного типа данных. Список shopping_list содержит только строки, а quantity только целые числа. Можно

вспомнить, что python динамически типизированный язык. Поэтому список может содержать элементы разных типов:

>>bread = [‘хлеб’, 1]

## Задание

1. Создайте список, который содержит наименование товара «торт» и цифру 1.

## Список списков

Мы выяснили, что элементами списков могут быть любой тип. На самом деле список может содержать другие списки.

Вернемся к нашему списку покупок. Мы хотим, чтобы внутри списка хранились наименования товаров и их необходимое количество. Для этих целей можно создать список, который будет включать в себя списки:

>> shopping_list = [[‘хлеб’, 1], [‘яйца’, 10], [‘молоко’, 1], [‘кофе’, 1]]

## Задание

1. Нам написали дополнительный список товаров, которые необходимо купить в отделе бытовой химии. Он включает в себя стиральный порошок в количестве 1 штуки и средство для мытья посуды так же 1 штуку. Создайте новый список household_chemicals, который будет содержать список, состоящий из списков, в которых первым элементом будет наименование товара, а вторым количество.

## Метод Zip

Вернемся к списку покупок:

1. Хлеб 1 шт
2. Яйца 10 шт
3. Молоко 1 шт
4. Кофе 1 шт

Предположим, что у нас есть два отдельных списка, один включает в себя наименование товаров, второй – их количество:

>>shopping_list = [‘хлеб’, ‘яйца’, ‘молоко’, ‘кофе’]
>>quantity = [1, 10, 1, 1]

Если бы мы хотели создать список списков, в котором каждое наименование товара было бы вместе с необходимым количеством, мы могли бы использовать команду zip. zip принимает два (или более) списка в качестве входных данных и возвращает объект, содержащий список пар (кортежей). Каждая пара содержит по одному элементу из каждого из входных списков. Однако, команда print не покажет нам результат исполнения функции zip.

>> shopping_list_with_quantity = zip(shopping_list, quantity)
>>print(shopping_list_with_quantity)
В результате выполнения кода выше в консоль будет выведено следующее:

```
<zip object at 0x7f1631e86b48>
```

Эта строка обозначает расположение созданного zip объекта в памяти. Однако, у нас есть операция преобразования типов. Ранее мы сталкивались с такой функций, как str(), которая переводила тип данных в строку. Такая же функция существует для преобразования типа данных в список. Эта функция list(). В нашем примере можно использовать данную функцию, чтобы увидеть результат выполнения функции zip:

>> shopping_list_with_quantity = zip(shopping_list, quantity)
>>print(list(shopping_list_with_quantity))

Результатом исполнения этого кода будет:

```
[('хлеб', 1), ('яйца', 10), ('молоко', 1), ('кофе', 1)]
```

**Задание**

1. Используйте zip для создания новой переменной с именем names_and_dogs_names, которая объединяет имена и имена собак в zip-объект.

Names=[‘Ben’, ‘Holly’, ‘Ann’]

dogs_names= [‘Sharik’, ‘Gab’, ‘Beethoven’]

2. Создайте новую переменную с именем list_of_names_and_dogs_names, вызвав list () для names_and_dogs_names. Выведите в консоль новую переменную.

## Пустые списки

Список не обязательно должен содержать что-то! Вы можете создать пустой список следующим образом:

>>empty_list = []

Зачем необходим пустой список?

Обычно, такие списки создаются с целью заполнения его позже на основе новых данных.

## Метод append

Данный метод позволяет добавлять элементы в конец списка. Например, нам нужно купить еще и шоколадку, и мы хотим отразить это в списке покупок. Для этого сделаем следующее:

>>shopping_list = [‘хлеб’, ‘яйца’, ‘молоко’, ‘кофе’]
>> shopping_list.append(‘шоколадка’)

>>print (shopping_list)

Этот код выведет следующее:

```
['хлеб', 'яйца', 'молоко', 'кофе', 'шоколадка']
```

Этот метод работает и с пустыми списками. Например,

```
>>empty_list = []
>>empty_list.append(1)
>>print(empty_list)
```

Этот код выведет:

```
[1]
```

Важно помнить, что .append () идет после списка. Это отличается от функций наподобие print, которые были изучены раньше.

**Задание**

1. Мария работает в цветочном магазине. Она ведет учет заказов в списке, который называется orders.

Создайте список orders и используйте print, чтобы проверить заказы, которые он получил сегодня.

orders = ['маргаритки', 'васильки'].

2. Мария только что получила новый заказ на тюльпаны. Используйте append, чтобы добавить эту строку в orders.

3. Пришел еще один заказ! Используйте append, чтобы добавлять «розы» к заказам.

4. Используйте print, чтобы просмотреть заказы, которые получила сегодня Мария.

**Оператор + при работе со списками**

Ранее было рассмотрено, как работает оператор + с числовыми и строковыми типами данных. Для списков данный оператор так же определен. + позволяет объединять несколько списков вместе. Например, пекарня имеет следующий ассортимент:

```
>>items_sold = ['cake', 'cookie', 'bread']
```

Предположим, пекарня хочет начать продавать печенье и пирожные:

```
>>items_sold_new = items_sold + ['biscuit', 'tart']
>>print(items_sold_new)
['cake', 'cookie', 'bread', 'biscuit', 'tart']
```

В этом примере мы создали новую переменную items_sold_new, которая содержала как оригинальные проданные товары, так и новые. Мы можем проверить исходный items_sold и убедиться, что он не изменился:

```
>>print(items_sold)
['cake', 'cookie', 'bread']
```

Так же, как и в случае со строками и числами, + можно применять только к спискам. Если мы введем этот код:

```
>>my_list = [1, 2, 3]
>>my_list + 4
```

мы получим следующую ошибку:

```
TypeError: can only concatenate list (not "int") to list
```

Если мы хотим добавить один элемент с помощью +, мы должны поместить его в список с помощью скобок ([]):

```
>>my_list + [4]
```

**Задание**

1. Мария все еще обновляет свой список заказов. Она только что получила заказы на «сирень» и «ирис».

Используйте +, чтобы создать новый список с именем new_orders, который объединяет заказы с двумя новыми заказами.

```
orders = ['маргаритка', 'лютик', 'львиный зев', 'гардения', 'лилия']

# broken_prices = [5, 3, 4, 5, 4] + 4
```

2. Удалите # перед списком broken_prices. Если вы запустите этот код, вы получите сообщение об ошибке
3. Исправьте команду, чтобы она выполнялась без ошибок.

**Range I**

Иногда нам необходимо создавать списки, состоящие из последовательностей. Например, необходимо создать список вида:

```
>>my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Набор такого числа чисел занимает определенное время. Кроме того, чем больше чисел мы набираем, тем больше вероятность того, что может произойти опечатка.

Python дает нам простой способ создания этих списков с помощью функции, называемой range. range принимает число на вход и генерирует числа, начиная с 0 и заканчивая числом перед входным параметром. Итак, если нам нужны числа от 0 до 9, мы используем диапазон (10), потому что 10 на 1 больше 9:

```
>>my_range = range(10)
```

Как и в случае с zip, функция диапазона возвращает объект, который мы можем преобразовать в список:

```
>>range(0, 10)
>>print(list(my_range))
```

## Задание

1. Измените list1 так, чтобы это был диапазон, содержащий числа, начинающиеся с 0 и до 9, но не включая 9.

list1=[1,8]

2. Создайте диапазон с именем list2 с числами от 0 до 7.

## Range II

Мы можем использовать range для создания более сложных последовательностей.

По умолчанию диапазон создает список, начинающийся с 0. Однако, если мы прописать в качестве входных параметров 2 числа, то можно задать начало и конец создаваемой последовательности. Например, диапазон (2, 9) будет генерировать числа, начиная с 2 и заканчивая 8 (не включая 9):

```
>> my_list = range(2, 9)
>>>print(list(my_list))
[2, 3, 4, 5, 6, 7, 8]
```

С одним или двумя аргументами range создаст список последовательных чисел (т.е. каждое число на единицу больше предыдущего). Если мы используем третий аргумент, мы можем создать список, который «пропускает» числа. Например, диапазон (2, 9, 2) даст нам список, в котором каждое число на 2 больше предыдущего:

```
>>> my_range2 = range(2, 9, 2)
>>> print(list(my_range2))
[2, 4, 6, 8]
```

Мы можем пропустить столько чисел, сколько захотим! В этом примере мы начнем с 1 и пропустим 10 между каждым числом, пока не дойдем до 100:

```
>>my_range3 = range(1, 100, 10)
>>print(list(my_range3))
[1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
```

Наш список останавливается на 91, потому что следующим числом в последовательности будет 101, что больше 100 (наша точка остановки).

Значения аргументов функции range можно обозначить, как (stert, stop, step) (начало, конец, шаг).

## Задание

1. Измените функцию диапазона, создавшую list1, так, чтобы она:

• Начиналась с 5

• Разница между каждым элементом составляет 3 единицы.

• Заканчивается на 15

list1 = диапазон (6, 15, 2)

2. Создайте объект диапазона с именем list2, который:

• Начинается с 0

• Разница между каждым элементом составляет 5 единиц.

• Заканчивается до 40

## Заключение

В данному занятии было изучено:

• Как создать список

• Как создать список списков с помощью zip-архива.

• Как добавлять элементы в список с помощью .append () или +

• Как использовать диапазон для создания списков целых чисел

## Задания

1. Мария вводит данные клиентов для своего бизнеса в области веб-дизайна. Вы поможете ей организовать свои данные.

Начните с превращения этого списка имен клиентов в список с именем first_names. Обязательно вводите имена в таком порядке:

• Эйнсли

• Бен

• Чани

• Депак

2. Создайте пустой список под названием age.

3. Возраст Депака - 42 года. Используйте .append (), чтобы прибавить 42 к age.

4. Марии нужен список возрастов для всех клиентов. Создайте новый список с именем all_ages, который добавляет возраст со следующим списком, содержащим возрасты Эйнсли, Бена и Чани:

[32, 41, 29]

Убедитесь, что all_ages начинается с возраста Эйнсли, Бена и Чани и заканчивается возрастом Депака (хранится в age)!

5. Создайте новую переменную name_and_age, которая объединяет first_names и all_ages с помощью zip.

6. Создайте диапазон с именем ids с идентификационным номером для каждого клиента. Поскольку клиентов 4, значения id должны изменяться от 0 до 3.