ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

# Урок 2:Структуры данных

Уровень: начинающий
Курс: введение Python

# Содержание

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Добро пожаловать на урок Структуры данных! Вы изучите:
- Тип структуры данных: списки, кортежи, наборы, словари, составные структуры данных;
- Операторы: членство, тождественность;
- Встроенные функции или методы.

# Список



- **Структуры данных - это контейнеры, которые организуют и группируют типы данных различным образом. Список является одной из наиболее распространенных и основных структур данных в Python.**
- Как можно видеть здесь, вы можете создать список в квадратных скобках. Списки могут содержать любое сочетание и соответствие типов данных, которые вы видели до сих пор.

```
list_of_random_things = [1, 3.4, 'a string', True]
```

# Обращение к элементу списка



Это список из 4 элементов. Все упорядоченные контейнеры (как списки) индексируются в Python с помощью начального индекса 0. Следовательно, чтобы извлечь первое значение из приведенного выше списка, мы можем написать:

```
>>> list_of_random_things[0]
1
```

# Индексация элементов

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Может показаться, что вы можете получить последний элемент с помощью следующего кода, но на самом деле это не сработает:

```
>>> list_of_random_things[len(list_of_random_things)]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-34-f88b03e5c60e> in <module>()
----> 1 lst[len(lst)]

IndexError: list index out of range
```

# Индексация элементов


- Однако вы можете извлечь последний элемент, уменьшив индекс на 1. Для этого вам нужно сделать следующее:

```
>>> list_of_random_things[len(list_of_random_things) - 1]
True
```

# Индексация элементов


- Также вы можете индексировать с конца списка, используя отрицательные значения, где -1 - последний элемент, -2 предпоследний элемент и.т.д.

```
>>> list_of_random_things[-1]
True
>>> list_of_random_things[-2]
a string
```

# Срезы и слайсы со списками



- Вы видели, что мы можем извлечь более одного значения из списка за один раз, используя **срезы**. При использовании срезов важно помнить, что **нижний** индекс является **включающим,** а **верхний** индекс **исключающим**. Таким образом, это:

```
>>> list_of_random_things = [1, 3.4, 'a string', True]
>>> list_of_random_things[1:2]
[3.4]
```

- вернет только **3.4** в списке. Обратите внимание, это отличается от простого индексирования одного элемента, поскольку вы получаете обратно список с этим индексированием. Двоеточие означает, что нам следует идти от начального значения слева от двоеточия, за исключением элемента справа.

# Срезы и слайсы со списками



- Если вы знаете, что хотите начать с начала списка, также можете пропустить это значение.

```
>>> list_of_random_things[:2]
[1, 3.4]
```

- или чтобы вернуть все элементы в конец списка, мы можем бросить последний элемент.

# Срезы и слайсы со списками



```
>>> list_of_random_things[1:]
[3.4, 'a string', True]
```

- Этот тип индексации работает точно так же для строк, где возвращаемое значение будет строкой.

# Операторы **in** и **not in**?



- Вы видели, что мы можем также использовать операторы **in** и **not in** для возврата **true,** если элемент присутствует в нашем списке, или если одна строка является подстрокой другой.

```
>>> 'this' in 'this is a string'
True
>>> 'in' in 'this is a string'
True
>>> 'isa' in 'this is a string'
False
>>> 5 not in [1, 2, 3, 4, 6]
True
>>> 5 in [1, 2, 3, 4, 6]
False
```

# Переменность и порядок


- **Переменность** заключается в том, можем ли мы изменить объект после его создания. Если объект (например, список или строка) можно изменить (как список), то он именуется **переменным**. Однако, если объект нельзя изменить, создав совершенно новый объект (например, строки), объект считается **неизменяемым**.

```
>>> my_lst = [1, 2, 3, 4, 5]
>>> my_lst[0] = 'one'
>>> print(my_lst)
['one', 2, 3, 4, 5]
```

# Переменность и порядок


- Как показано выше, можете заменить 1 на «один» в приведенном выше списке. Это потому, что списки **изменяемые**.
- Однако следующее не работает:

```
>>> greeting = "Hello there"
>>> greeting[0] = 'M'
```

- Это потому, что строки **неизменяемые**. Это означает, что для изменения этой строки вам потребуется создать совершенно новую строку.

# Переменность и порядок


- Нужно помнить о двух вещах для каждого типа данных, которые вы используете:
  - Являются ли они **переменными**?
  - Являются ли они **упорядоченными**?
- **Порядок** означает, может ли положение элемента в объекте использоваться для доступа к элементу. **Строки и списки упорядочены.** Мы можем использовать порядок для доступа к части списка и строки.

# Переменность и порядок


- Однако, в следующих разделах вы увидите некоторые типы данных, которые будут неупорядоченными. Для каждой из следующих структур данных, которые вы видите, важно понять, как вы индексируете их, являются ли они изменяемыми и упорядочены ли они. Знание этого о структуре данных действительно полезно!
- Кроме того, вы увидите, что у каждого из них разные методы, поэтому то, почему вы будете использовать одну структуру данных против другой, во многом зависит от этих свойств и от того, что вы легко можете с этим сделать!

# Тест: индексирование списка


- Имена: сохраните имена нескольких своих друзей в списке с именем names. Выведите имя каждого друга, обратившись к каждому элементу списка (по одному за раз).
- . Сообщения: начните со списка, использованного в первом упражнении, но вместо вывода имени каждого человека выведите сообщение. Основной текст всех сообщений должен быть одинаковым, но каждое сообщение должно включать имя адресата. (Например, “Привет Harry”)

# Зачем нам нужны списки?


- Давайте поговорим о том, зачем нам нужна такая структура данных как список, или когда ее использовать. Мы возьмем пример из Уолл-стрит для обсуждения.
- Компании, зарегистрированные на фондовой бирже NASDAQ, получают биржевой код или аббревиатуры для названия каждой компании. Например, биржевым кодом для Alphabet, Inc. будем GOOGL.

# Зачем нам нужны списки?



- Теперь представьте, что у вас есть акции для одной компании, скажем, Microsoft, вы хотите распечатать биржевой код вашей акции. Поскольку это одно значение, вы можете сохранить его в переменной **Microsoft** и присвоить ему значение MSFT. Например:
  - **microsoft = MSFT**
- Ведь это удобно! Итак, теперь, когда вы хотите распечатать биржевой код для компании, в которой вы владеет акциями, вы используете команду печати.

```
print(microsoft)
>>> MSFT
```

# Полезные функции для списков I

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

1. **len ()** возвращает количество элементов в списке.
2. **max ()** возвращает наибольший элемент списка. Как определяется наибольший элемент, зависит от типа объектов в списке. Наибольшим элементом в списке чисел является наибольшее число. Наибольший элемент в списке строк - это элемент, который появился бы последним, если бы список был отсортирован по алфавиту. Это работает, поскольку функция max определяется через оператора сравнения «больше чем». Функция max не определена для списков, которые содержат элементы разных несопоставимых типов.

# Полезные функции для списков II



3. **min ()** возвращает наименьший элемент в списке. Min это противоположность max, который возвращает наибольший элемент в списке.
4. **sorted ()** возвращает копию списка в порядке от наименьшего к наибольшему, оставляя список неизменным.

# Тест: методы списка



1. Создайте список с числовыми элементами
2. Выведите минимальный и максимальный элемент этого списка
3. Отсортируйте и выведите список