ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

Урок 3: Структуры данных

Уровень: начинающий
Курс: введение Python

Содержание

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Добро пожаловать на этот урок по структурам данных! Вы изучите:
- Типы структур данных: списки, кортежи, наборы, словари, составные структуры данных
- Операторы: членство, личность
- Встроенные функции или методы

Полезные функции для списков III

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- **join** метод
  - Соединение - это строковый метод, который принимает список строк в качестве аргумента и возвращает строку, состоящую из элементов списка, соединенных строкой разделителя.

```
new_str = "\n".join(["fore", "aft", "starboard", "port"])
print(new_str)
```

Полезные функции для списков III

- Output:

```
fore
aft
starboard
port
```

- В этом примере мы используем строку **«\n»** в качестве разделителя, чтобы между каждым элементом была новая строка. Мы также можем использовать другие строки в качестве разделителей с соединением. Здесь мы используем дефис.

Еще один пример метода **join**

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

```
name = "-".join(["García", "O'Kelly"])
print(name)
```

Еще один пример метода **join**

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Output:

```
García-O'Kelly
```

- Важно не забывать разделять каждый элемент списка, к которому вы присоединяетесь, запятой (,). Если вы забудете это сделать, это не вызовет ошибку, но также даст вам неожиданные результаты.

метод append(присоеденения)

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Полезный метод, именуемый присоединением, добавляет элемент в конец списка.

```
letters = ['a', 'b', 'c', 'd']
letters.append('z')
print(letters)
```

- Output:

```
['a', 'b', 'c', 'd', 'z']
```

Тест: методы списка

- Создайте список гостей и выведите каждое имя на новой строке
- Добавьте в конец списка несколько имен, и снова выведите его

Tuples(Кортежи)

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Кортеж - это еще один полезный контейнер. Это тип данных для неизменяемых упорядоченных последовательностей элементов. Они часто используются для сохранения сопутствующих частей информации. Рассмотрим пример с широтой и долготой:

```
location = (13.4125, 103.866667)
print("Latitude:", location[0])
print("Longitude:", location[1])
```

- Кортежи похожи на списки тем, что в них хранится упорядоченная коллекция объектов, доступ к которым осуществляется по индексам. Однако, в отличие от списков, кортежи неизменны - вы не можете добавлять или удалять элементы из кортежей или сортировать их на месте.
- Кортежи также можно использовать для назначения множества переменных в одну составную.

Tuples(Кортежи)


```
dimensions = 52, 40, 100
length, width, height = dimensions
print("The dimensions are {} x {} x {}".format(length, width, height))
```

- Круглые скобки необязательны при определении кортежей, программисты часто пропускают их, если круглые скобки не уточняют код.
- Во второй строке три переменные назначаются из содержимого измерений кортежа. Это называется распаковкой кортежа. Вы можете использовать распаковку кортежей для назначения информации из кортежа нескольким переменным без необходимости обращаться к ним по одному и делать несколько операторов назначения.

Tuples(Кортежи)

- Если нам не нужно использовать измерения напрямую, мы можем сократить эти две строки кода до одной строки, которая назначает три переменные за один раз!!

```
length, width, height = 52, 40, 100
print("The dimensions are {} x {} x {}".format(length, width, height))
```

Quiz: Tuples

- Создайте кортеж в формате имя, возраст, рост, вес. И выедите его следующим образом:

Его/ее зовут ‘имя’, ему/ей ‘возраст’, на данный момент его/ее рост и вес составляет ‘рост’, ‘вес’

Наборы

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Набор - это тип данных для изменяемых неупорядоченных коллекций уникальных элементов. Одним из применений набора является быстрое удаление дубликатов из списка.

```
numbers = [1, 2, 6, 3, 1, 1, 6]
unique_nums = set(numbers)
print(unique_nums)
```

- Это выведет:

```
{1, 2, 3, 6}
```

Наборы

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Наборы поддерживают оператор in, также, как и списки. Вы можете добавлять элементы в наборы, используя метод добавления, удалять элементы, используя метод удаления, как в списках. Хотя, когда вы извлекаете элемент из набора, нетипичный элемент удаляется. Помните, что наборы, в отличие от списков, неупорядоченные, поэтому «последний элемент» отсутствует.

```
fruit = {"apple", "banana", "orange", "grapefruit"}  # define a set

print("watermelon" in fruit)  # check for element

fruit.add("watermelon")  # add an element
print(fruit)

print(fruit.pop())  # remove a random element
print(fruit)
```

Наборы

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Output:

```
False
{'grapefruit', 'orange', 'watermelon', 'banana', 'apple'}
grapefruit
{'orange', 'watermelon', 'banana', 'apple'}
```

Операции с наборами

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

A | B **A.union(B)**	Возвращает множество, являющееся объединением множеств A и B.
A |= B **A.update(B)**	Добавляет в множество A все элементы из множества B.
A & B **A.intersection(B)**	Возвращает множество, являющееся пересечением множеств A и B.
A &= B **A.intersection_update(B)**	Оставляет в множестве A только те элементы, которые есть в множестве B.
A - B **A.difference(B)**	Возвращает разность множеств A и B (элементы, входящие в A, но не входящие в B).
A -= B **A.difference_update(B)**	Удаляет из множества A все элементы, входящие в B.
A ^ B **A.symmetric_difference(B)**	Возвращает симметрическую разность множеств A и B (элементы, входящие в A или в B, но не в оба из них одновременно)

Операции с наборами

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

A ^= B **A.symmetric_difference_update(B)**	Записывает в A симметрическую разность множеств A и B.
A <= B **A.issubset(B)**	Возвращает true, если A является подмножеством B.
A >= B **A.issuperset(B)**	Возвращает true, если B является подмножеством A.
A &= B **A.intersection_update(B)**	Оставляет в множестве A только те элементы, которые есть в множестве B.
A < B	Эквивалентно A <= B and A != B
A > B	Эквивалентно A <= B and A != B

Quiz: sets

- Создайте список и выведите количество уникальных элементов
- Даны два списка чисел. Посчитайте, сколько чисел содержится одновременно как в первом списке, так и во втором.

Словари и операторы тождественности

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Словарь - это изменяемый тип данных, который хранит сопоставления уникальных ключей к значениям. Вот словарь, который хранит элементы и их порядковые номера.

```
elements = {"hydrogen": 1, "helium": 2, "carbon": 6}
```

Словари и операторы тождественности

- В словарях могут быть ключи неизменяемого типа, например, целые числа или кортежи, а не только строки.
- Нет необходимости, чтобы все ключи были одинакового типа!
- Мы можем искать значения или вставлять новые значения в словарь, используя квадратные скобки, которые заключают ключ.

```
print(elements["helium"])   # print the value mapped to "helium"
elements["lithium"] = 3   # insert "lithium" with a value of 3 into the dict
```

Словари и операторы тождественности

- Мы можем проверить, находится ли значение в словаре, так же, как мы проверяем, находится ли значение в списке или наборе с помощью ключевого слова in.
- В словарях есть сопутствующий метод, который также полезен get.
- get ищет значения в словаре, но в отличие от квадратных скобок, get возвращает None (или значение по умолчанию по вашему выбору), если ключ не найден.

```
print("carbon" in elements)
print(elements.get("dilithium"))
```

Словари и операторы тождественности

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Output:

```
True
None
```

- Углерод находится в словаре, поэтому печатается True.
- Дилития нет в нашем словаре, поэтому get возвращает None, а затем печатает.
- Если вы ожидаете, что поиск иногда завершится неуспешно, get может быть лучшим инструментом, чем обычные квадратные скобки, потому что ошибки могут привести к отказу вашей программы.

Словари и операторы тождественности
Операторы тождественности

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

Keyword	Operator
`is`	evaluates if both sides have the same identity
`is not`	evaluates if both sides have different identities

- Вы можете проверить возврат ключом None с помощью оператора is.
- Вы можете проверить обратное с помощью оператора is not.

```
n = elements.get("dilithium")
print(n is None)
print(n is not None)
```

Словари и операторы тождественности
Операторы тождественности

- Output:

```
True
False
```

Тест: Определить словарь

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

Человек: используйте словарь для сохранения информации об известном вам человеке. Сохраните имя, фамилию, возраст и город, в котором живет этот человек. Словарь должен содержать ключи с такими именами, как first_name, last_name, age и city. Выведите каждый фрагмент информации, хранящийся в словаре.

Тест: больше о словарях

- Любимые числа: используйте словарь для хранения любимых чисел. Возьмите пять имен и используйте их как ключи словаря. Придумайте любимое число для каждого человека и сохраните его как значение в словаре. Выведите имя каждого человека и его любимое число. Чтобы задача стала более интересной, опросите нескольких друзей и соберите реальные данные для своей программы.

Когда мы используем словари

Словари нужно использовать в следующих случаях:

- Подсчет числа каких-то объектов. В этом случае нужно завести словарь, в котором ключами являются объекты, а значениями — их количество.
- Хранение каких-либо данных, связанных с объектом. Ключи — объекты, значения — связанные с ними данные. Например, если нужно по названию месяца определить его порядковый номер, то это можно сделать при помощи словаря Num['January'] = 1; Num['February'] = 2; ....
- Установка соответствия между объектами (например, “родитель—потомок”). Ключ — объект, значение — соответствующий ему объект.
- Если нужен обычный массив, но масимальное значение индекса элемента очень велико, и при этом будут использоваться не все возможные индексы (так называемый “разреженный массив”), то можно использовать ассоциативный массив для экономии памяти.

Составные структуры данных

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Мы можем включать контейнеры в другие контейнеры для создания составных структур данных. Например, данный словарь отображает ключи значений, которые также являются словарями!

```
elements = {"hydrogen": {"number": 1,
                         "weight": 1.00794,
                         "symbol": "H"},
            "helium": {"number": 2,
                       "weight": 4.002602,
                       "symbol": "He"}}
```

- Мы можем получить доступ к элементам в этом вложенном словаре.

```
helium = elements["helium"]  # get the helium dictionary
hydrogen_weight = elements["hydrogen"]["weight"]  # get hydrogen's weight
```

Составные структуры данных

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Вы также можете добавить новый ключ в словарь элементов.

```
oxygen = {"number":8,"weight":15.999,"symbol":"O"}  # create a new oxygen dic
elements["oxygen"] = oxygen  # assign 'oxygen' as a key to the elements dicti
print('elements = ', elements)
```

- Output:

```
elements =  {"hydrogen": {"number": 1,
                          "weight": 1.00794,
                          "symbol": 'H'},
             "helium": {"number": 2,
                        "weight": 4.002602,
                        "symbol": "He"},
             "oxygen": {"number": 8,
                        "weight": 15.999,
                        "symbol": "O"}}
```

Тест: добавление значений во вложенные словари

Создайте несколько словарей с информацией о человеке. Например, Peson1={‘name’:..., ‘age’:..., ‘job’:...}, Person2=..., и т.д.

Затем создайте еще один словарь People и в значения присвойте предыдущие словари (см. предыдущий слайд)

Поздравляем с завершением урока по структурам данных!

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Хорошее понимание структур данных является неотъемлемой частью программирования и анализа данных. Как аналитик данных вы будете работать с данными и кодом все время, поэтому четкое понимание того, какие типы данных и структуры данных доступны, когда использовать каждый из них, поможет вам разработать более эффективный код.

Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	`[ ]` or `list()`	`[5.7, 4, 'yes', 5.7]`
Tuple	Yes	No	`( )` or `tuple()`	`(5.7, 4, 'yes', 5.7)`
Set	No	Yes	`{}`* or `set()`	`{5.7, 4, 'yes'}`
Dictionary	No	No**	`{ }` or `dict()`	`{'Jun': 75, 'Jul': 89}`

Итоги

- В этом уроке мы рассмотрели четыре важные структуры данных в Python:
- ** Вы можете использовать фигурные скобки, чтобы определить такой набор: **{1, 2, 3}**. Однако, если вы оставите фигурные скобки пустыми: **{}** Python вместо этого создаст пустой словарь. Поэтому, чтобы создать пустой набор, используйте **set ()**.*
- *** Сам словарь является изменяемым, но каждый из его отдельных ключей должен быть неизменяемым.*