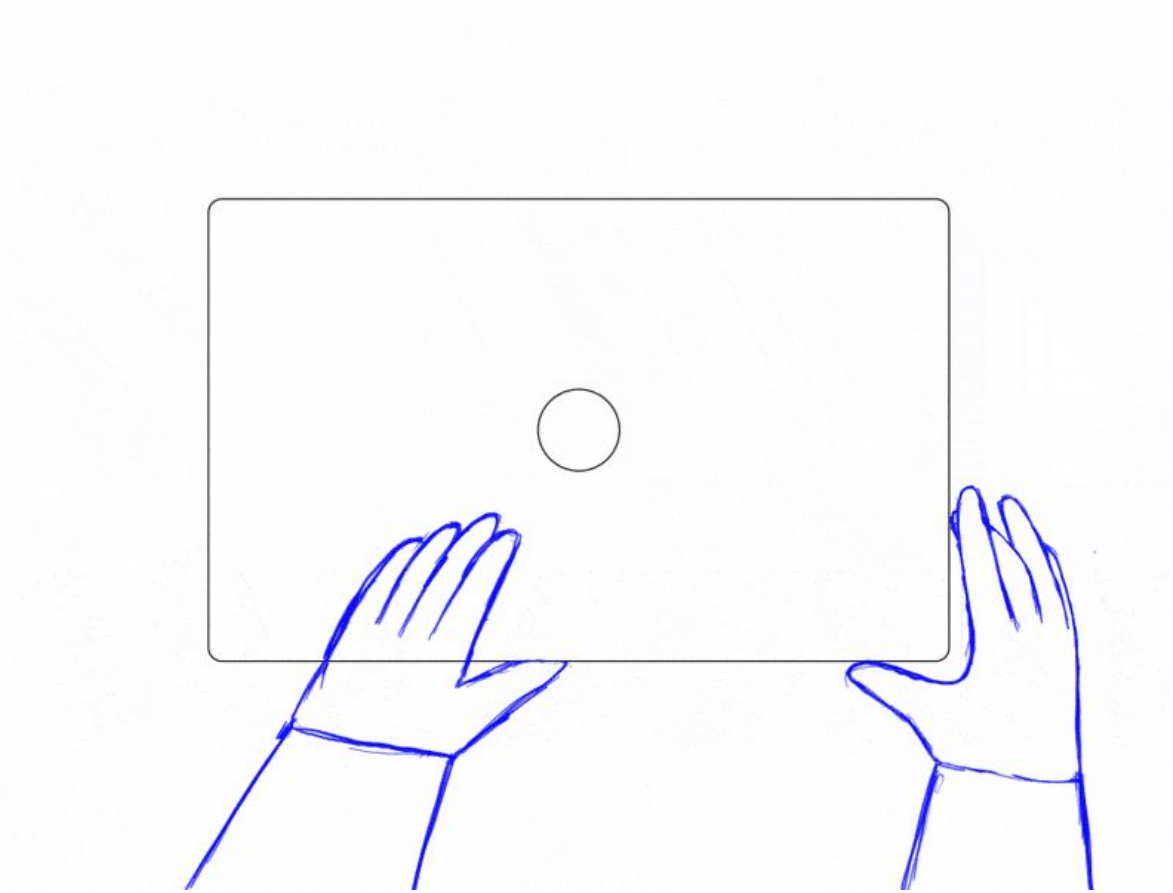
Моя IT Школа

сертифицированные IT курсы

Коллекции. Кортежи. Множества



Проверка домашнего задания

У вас есть словарь, где ключ – название продукта. Значение – список, который содержит цену и кол-во товара.
Выведите через ‘’–’’ название – цену – количество.
С клавиатуры вводите название товара и его кол-во. n – выход из программы. Посчитать цену выбранных товаров и сколько товаров осталось в изначальном списке.

Решение

Моя Школа

```
goods = {"Apple": [4.5, 10],
         "Orange": [6.2, 5],
         "Pineapple": [10.0, 1],
         "Mango": [7.5, 2],
         "Banana": [3.8, 10]}

for key, value in goods.items():
    print(key, '-', value[0], '-', value[1])
cost = 0
while True:
    good = input("What? (n - nothing) ")
    if good == 'n' or good not in goods.keys():
        break
    qty = int(input("How many? "))
    if qty > goods[good][1]:
        print("We don't have so much.")
        continue
    cost += goods[good][0] * qty
    goods[good][1] -= qty
print("Price:", cost)
print('----------------------------------')
for key, value in goods.items():
    print(key, '-', value[0], '-', value[1])
```

```
Apple - 4.5 - 10
Orange - 6.2 - 5
Pineapple - 10.0 - 1
Mango - 7.5 - 2
Banana - 3.8 - 10
What? (n - nothing) Apple
How many? 5
What? (n - nothing) Banana
How many? 6
What? (n - nothing) n
Price: 45.3
----------------------------------
Apple - 4.5 - 5
Orange - 6.2 - 5
Pineapple - 10.0 - 1
Mango - 7.5 - 2
Banana - 3.8 - 4

Process finished with exit code 0
```

План занятия

Моя IT Школа
сертифицированные IT курсы

1. Кортежи (tuple) в Python
2. Кортежи и списки
3. Операции с кортежами
4. Множества
5. Методы множеств
6. Неизменяемое множество (frozenset)

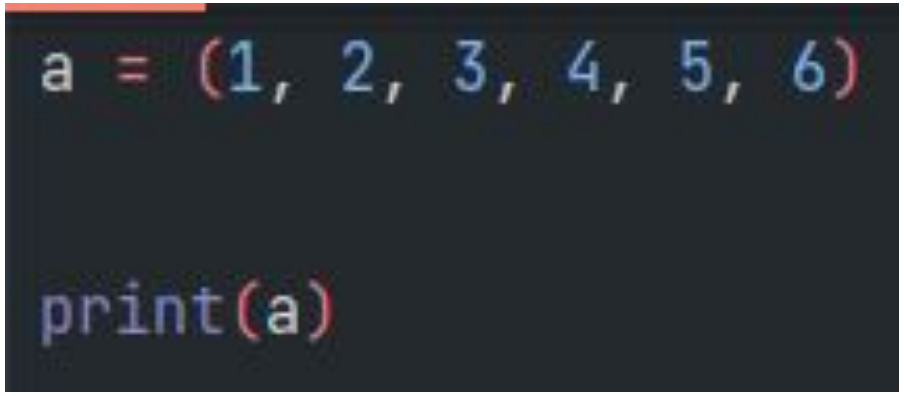
Коллекции. Кортежи. Множества

Моя IT Школа

Кортежи (tuple) в Python – это те же списки за одним исключением. Кортежи неизменяемые структуры данных. Так же как списки они могут состоять из элементов разных типов, перечисленных через запятую. Кортежи заключаются в круглые, а не квадратные скобки.

Обратите внимание на запятую (,) в объявлении кортежа a. Если ее не указать при создании объекта с одним элементом?

Python предположит, что вы по ошибке добавили лишнюю пару скобок (это ни на что не влияет), но тип данных в таком случае — это не кортеж. Поэтому важно не забывать использовать запятую при объявлении кортежа с одним элементом.



Зачем нужны кортежи, если есть списки?

- Кортеж защищен от изменений, как намеренных (что плохо), так и случайных (что хорошо).
- Меньший размер.

```
a = (1, 2, 3, 4, 5, 6)
b = [1, 2, 3, 4, 5, 6]

print(a.__sizeof__())

print(b.__sizeof__())
```

```
72
136

Process finished with exit code 0
```

Коллекции. Кортежи. Множества

Моя Школа

Из кортежа можно извлекать элементы и брать срезы:

```
a = (1, 2, 3, 4, 5, 6)
print(a[0:3])
print(a[:3])
print(a[1:])
print(a[2::2])
print(a[::2])
```

Однако изменять его элементы нельзя.

```
    a[1] = 11
TypeError: 'tuple' object does not support item assignment
```

Также у типа **tuple** нет методов для добавления и удаления элементов.

Возникает резонный вопрос. Зачем в язык программирования был введен этот тип данных, по - сути представляющий собой неизменяемый список? Дело в том, что иногда надо защитить список от изменений. Преобразовать же кортеж в список, если это потребуется, как и выполнить обратную операцию легко с помощью встроенных в Python функций **list**() и **tuple**():

```
a = (10, 2.13, "square", 89, 'C')
b = [1, 2, 3]
c = list(a)
d = tuple(b)
```

Кортежи могут содержать списки, также как списки быть вложенными в другие списки.

```
nested = (1, "do", ["param", 10, 20])
```

Как вы думаете, можем ли мы изменить список ["param", 10, 20] вложенный в кортеж nested?

Список изменяем, кортеж – нет. Если вам кажется, что нельзя, то вам кажется неправильно. На самом деле можно:

```
nested = (1, "do", ["param", 10, 20])
nested[2][1] = 15
print(nested)
```

Примечание. Выражения типа nested[2][1] используются для обращения к вложенным объектам. Первый индекс указывает на позицию вложенного объекта, второй – индекс элемента внутри вложенного объекта. Так в данном случае сам список внутри кортежа имеет индекс 2, а элемент списка 10 – индекс 1 в списке.

Объединение кортежей

```
x = (1, 2, 3, 4)
y = (5, 6, 7, 8)

# Объединение двух кортежей для формирования нового кортежа
z = x + y
print(z)
```

Разрешается объединять только определенные типы данных. Так, попытка соединить кортеж и список закончится ошибкой.

Умножение кортежей

Операция умножения приводит к тому, что кортеж повторяется несколько раз.

```
x = (1, 2, 3, 4)
z = x * 2
print(z)
```

Функции кортежей

В отличие от списков у кортежей нет методов, таких как append(), remove(), extend(), insert() или pop() опять-таки из-за их неизменяемости. Но есть другие:

count() и **len()**

count() возвращает количество повторений элемента в кортеже.

```
a = (1, 2, 3, 4, 5, 5)

print(a.count(5), len(a))
```

Функции кортежей

Функция **max**() возвращает самый большой элемент последовательности,
а **min**() — самый маленький.
Эти функции можно использовать и для кортежей со строками.

```
a = (1, 2, 3, 4, 5, 5)
print('max ', max(a), 'min ', min(a))
```

Задание №1

Создайте кортеж из случайных 10 чисел. Найдите его максимальный минимальный элемент.

Решение

Моя IT Школа

```python
import random

a = [random.randint(0, 100) for i in range(10)]
b = tuple(a)
print(b)
print('max ', max(a), 'min ', min(a))
```

Задание №2

Заполните один кортеж десятью случайными целыми числами от 0 до 5 включительно. Также заполните второй кортеж числами от -5 до 0. Объедините два кортежа с помощью оператора +, создав тем самым третий
кортеж. С помощью метода кортежа count() определите в нем количество
нулей. Выведите на экран третий кортеж и количество нулей в нем.

Решение

Моя Школа

```python
import random
a = tuple([random.randint(0, 5) for _ in range(10)])
b = tuple([random.randint(-5, 0) for _ in range(10)])
c = a + b
print(c, '\n Количество нулей:', c.count(0))
```

Задание №3

Вывести данные кортежа без скобок, через запятую.

Обязательно: элементы кортежа – строки.

Решение

Моя Школа

```
a = ('one', 'two', 'three')

c = ','.join(a)

print(c)
```

Задание №4

Даны два кортежа:
A = (13, 5, 43, 49, 67, 32, 12, 98, 6, 10, 34, 20, 55, 68, 14, 60, 14)
B = (53, 21, 4, 23, 76, 3, 43, 12, 54, 342, 21)
Необходимо определить:
1) Сумма элементов какого из кортежей больше и вывести соответствующее
сообщение на экран (Сумма больше в кортеже - ..)
2) Вывести на экран порядковые номера минимальных элементов этих кортежей

Решение

Моя Школа

```
A = (13, 5, 43, 49, 67, 32, 12, 98, 6, 10, 34, 20, 55, 68, 14, 60, 14)
B = (53, 21, 4, 23, 76, 3, 43, 12, 54, 342, 21)
s_A = sum(A)
s_B = sum(B)
if s_A > s_B:
    print("Сумма больше в кортеже - A")
else:
    print("Сумма больше в кортеже - B")

print('min A', min(A), 'Номер - ', A.index(min(A)))
print('min B', min(B), 'Номер - ', B.index(min(B)))
```

Множества

Множества в Python – это структура данных, которые содержат неупорядоченные элементы. Элементы также не является индексированным. Как и список, множество позволяет внесение и удаление элементов. Однако, есть ряд особенных характеристик, которые определяют и отделяют множество от других структур данных:

- Множество не содержит дубликаты элементов;
- Элементы множества являются неизменными (их нельзя менять), однако само по себе множество является изменяемым, и его можно менять;
- Так как элементы не индексируются, множества не поддерживают никаких операций среза и индексирования.

Создание множеств

Существует два пути, следуя которым, мы можем создавать множества в Python.
Мы можем создать множество путем передачи всех элементов множества внутри фигурных скобок **{}** и разделить элементы при помощи запятых (**,**). Множество может содержать любое количество элементов и элементы могут быть разных типов, к примеру, целые числа, строки, кортежи, и т. д. Однако, множество не поддерживает изменяемые элементы, такие как списки, словари, и так далее.

Рассмотрим пример создания множества в Python:

```
num_set = {1, 2, 3, 4, 5, 6}
print(num_set)
```

Создание множеств

Только что мы создали множество чисел. Мы также можем создать множество из строк. Например:

```
string_set = {"Nicholas", "Michelle", "John", "Mercy"}
print(string_set)
```

Возможно вы обратили внимание на то, что элементы в выдаче выше находятся в другом порядке, отличном от того, как мы добавляли их в множество. Это связано с тем, что элементы множества **находятся в произвольном порядке**. Если вы запустите тот же код еще раз, возможно вы получите выдачу с элементами, которые каждый раз будут находиться в другом порядке.

Создание множеств

Мы также можем создать множество с элементами разных типов.
Например:

```
mixed_set = {2.0, "Nicholas", (1, 2, 3)}
print(mixed_set)
```

Все элементы в упомянутом выше множестве принадлежат разным типам.

Создание множеств

Мы также можем создать множество из списков. Это можно сделать, вызвав встроенную функцию Python под названием set().
Например:

```
num_set = set([1, 2, 3, 4, 5, 6])
print(num_set)
```

Создание множеств

Как упоминалось ранее, множества не содержат дубликаты элементов. Предположим, наш список содержит дубликаты элементов, как показано ниже:

```
num_set = set([1, 2, 3, 1, 2])
print(num_set)
```

Множество удалило дубликаты и выдало только по одному экземпляру элементов. Это также происходит при **создании множества** с нуля. Например:

```
num_set = {1, 2, 3, 1, 2}
print(num_set)
```

Создание множеств

Создание пустого множества подразумевает определенную хитрость. Если вы используете пустые фигурные скобки {} в Python, вы скорее создадите пустой словарь, а не множество.
Например:

```
x = {}
print(type(x))
```

```
<class 'dict'>


Process finished with exit code 0
```

Как показано в выдаче, тип переменной x является словарем.

Создание множеств

Чтобы создать пустое множество в Python, мы должны использовать функцию set() без передачи какого-либо значения в параметрах, как показано ниже:

```
x = set()
print(type(x))
```

```
<class 'set'>


Process finished with exit code 0
```

Выдача показывает, что мы создали множество.

Доступ к элементам множеств

Python не предоставляет прямой способ получения значения к отдельным элементам множества. Однако, мы можем использовать цикл для итерации через все элементы множества.
Например:

```
months = set(["Jan", "Feb", "March", "Apr", "May", "June", "July", "Aug", "Sep", "Oct", "Nov", "Dec"])

for m in months:
    print(m)
```

Доступ к элементам множеств

Мы также можем проверить наличие элемента во множестве при помощи in, как показано ниже:

```
months = set(["Jan", "Feb", "March", "Apr", "May", "June", "July", "Aug", "Sep", "Oct", "Nov", "Dec"])

print("May" in months)
```

Код возвращает **True**, а это означает, что элемент был найден во множестве. Аналогичным образом, при поиске элемента, который отсутствует во множестве, мы получим **False.**

Добавление элементов во множество

Python позволяет нам вносить новые элементы во множество при помощи функции add(). Например:

```
months = set(["Jan", "March", "Apr", "May", "June", "July", "Aug", "Sep", "Oct", "Nov", "Dec"])

months.add("Feb")
print(months)
```

Удаление элемента из множеств

Python позволяет нам удалять элемент из множества, но не используя индекс, так как множество элементов не индексированы. Элементы могут быть удалены при помощи обоих методов discard() и remove().
Помните, что метод discard() не будет выдавать ошибку, если элемент не был найден во множестве. Однако, если метод remove() используется и элемент не был найден, возникнет ошибка.

Давайте продемонстрируем как удалять элемент при помощи метода discard():

```
num_set = {1, 2, 3, 4, 5, 6}
num_set.discard(3)
print(num_set)
```

Удаление элемента из множеств

Аналогично, метод remove() может использоваться следующим образом:

```
num_set = {1, 2, 3, 4, 5, 6}
num_set.remove(3)
print(num_set)
```

Удаление элемента из множеств

Теперь попробуем удалить элемент, которого нет во множестве. Сначала используем метод discard():

```
num_set = {1, 2, 3, 4, 5, 6}
num_set.discard(7)
print(num_set)
```

Удаление элемента из множеств

Теперь посмотрим, что выйдет из использования метода remove() по аналогичному сценарию:

```
num_set = {1, 2, 3, 4, 5, 6}
num_set.remove(7)
print(num_set)
```

```
    num_set.remove(7)
KeyError: 7

Process finished with exit code 1
```

Выдача показывает, что метод выдал ошибку KeyError, так как мы пытались удалить элемент, которого нет во множестве.

Удаление элемента из множеств

С методом pop(), мы можем удалить и вернуть элемент. Так как элементы находятся в произвольном порядке, мы не можем утверждать или предсказать, какой элемент будет удален.

```
num_set = {1, 2, 3, 4, 5, 6}
num_set.pop()
print(num_set)
```

Удаление элемента из множеств

Метод Python под названием clear() поможет удалить все элементы во множестве.
Например:

```
num_set = {1, 2, 3, 4, 5, 6}
num_set.clear()
print(num_set)
```

Результатом является пустой set() без каких-либо элементов внутри.

Объединение множеств

Предположим, у нас есть два множества, A и B. Объединение этих двух множеств — это множество со всеми элементами обеих множеств. Такая операция выполняется при помощи функции Python под названием union().

Рассмотрим пример:

```
months_a = set(["Jan", "Feb", "March", "Apr", "May", "June"])
months_b = set(["July", "Aug", "Sep", "Oct", "Nov", "Dec"])

all_months = months_a.union(months_b)
print(all_months)
```

Объединение множеств

При выполнении операции объединения, дубликаты игнорируются, так что только один из двух элементов дубликатов будет отображаться. Например:

```
x = {1, 2, 3}
y = {4, 3, 6}
z = {7, 4, 9}

output = x.union(y, z)

print(output)
```

Объединение множеств

Оператор | может также использоваться при поиске объединения двух или более множеств.
Например:

```
months_a = set(["Jan", "Feb", "March", "Apr", "May", "June"])
months_b = set(["July", "Aug", "Sep", "Oct", "Nov", "Dec"])

print(months_a | months_b)
```

Объединение множеств

Если вы хотите создать объединение из более двух множеств, разделите названия множеств при помощи оператора | . Взглянем на пример:

```
x = {1, 2, 3}
y = {4, 3, 6}
z = {7, 4, 9}

print(x | y | z)
```

Пересечение множеств

Предположим, у вас есть два множества: A и B. Их пересечение представляет собой множество элементов, которые являются общими для A и для B.
Операция пересечения во множествах может быть достигнута как при помощи оператора &, так и метода intersection().
Рассмотрим пример:

```
x = {1, 2, 3}
y = {4, 3, 6}

print(x & y)  # Результат: 3
```

Разница между множествами

Предположим, у вас есть два множества: A и B. Разница между A и B (A — B) — это множество со всеми элементами, которые содержатся в A, но не в B. Соответственно, (B — A) — это множество со всеми элементами в B, но не в A.

```
A = {1, 2, 3}
B = {4, 3, 6}

print(A - B)  # Результат: 1,2
print(B - A)  # Результат: 4,6
```

Методы множеств

Python содержит огромное количество встроенных методов, включая следующие:

Метод **copy()**
Этот метод возвращает копию множества.
Например:

```
string_set = {"Nicholas", "Michelle", "John", "Mercy"}
x = string_set.copy()


print(x)
```

Методы множеств

Метод isdisjoint()

Этот метод проверяет, является ли множество пересечением или нет. Если множества не содержат общих элементов, метод возвращает True, в противном случае — False.

Например:

```
names_a = {"Nicholas", "Michelle", "John", "Mercy"}
names_b = {"Jeff", "Bosco", "Teddy", "Milly"}

x = names_a.isdisjoint(names_b)
print(x)
```

Оба множества не имеют общих элементов, что делает выдачу True.

Методы множеств

Метод **len()**

Этот метод возвращает длину множества, которая является общим количеством элементов во множестве.

Пример:

```
names_a = {"Nicholas", "Michelle", "John", "Mercy"}

print(len(names_a))  # Результат: 4
```

frozenset

Единственное отличие set от frozenset заключается в том, что set - изменяемый тип данных, а frozenset - нет.

```
my_set = frozenset([1, 2, 3, -10, 40])
print(type(my_set))
```

```
<class 'frozenset'>


Process finished with exit code 0
```

Задание №1

Проверить, есть ли в последовательности дубликаты

Решение

Моя IT Школа

```python
# Создаем список с дубликатами lst
lst = [0, 0, 1, 2, 3, 4, 5, 5, 6, 7]

# На основе списка создаем множество st
# Помним про основное свойство множеств - они не могут содержать дубликатов
# Поэтому если lst содержит дубликаты, то при создании множества на его основе дубликаты будут удалены
st = set(lst)

# А значит количество элементов в списке и во множестве будет различаться
# Сравниваем количество элементов с помощью встроенного метода len()
print(len(st) == len(lst))
# Длины не равны, значит в изначальном списке были дубликаты
```

Задание №2

1. Создать множество.
2. Создать неизменяемое множество.
3. Выполнить операцию объединения созданных множеств.
4. Выполнить операцию пересечения созданных множеств.

Решение

Моя IT Школа

```python
# 1. Создаем изменяемое множество
st = {'it', 'is', 'set', 1}

# 2. Создаем неизменяемое множество
frozen_st = frozenset({'it', 'is', 'frozen', 'set', 2})

# 3. Выполняем операцию объединения созданных множеств
# Результатом объединения будет множество, содержащее все элементы обоих множеств(без дубликатов)
union = st | frozen_st
# Результат: {'frozen', 1, 2, 'set', 'it', 'is'}

# 4. Выполняем операцию пересечения созданных множеств
# Результатом пересечения будет множество, содержащее элементы, присутствующие одновременно в обоих множествах
intersection = st & frozen_st
# Результат: {'it', 'set', 'is'}
```

Домашнее задание

1. Создайте кортеж с цифрами от 0 до 9 и посчитайте сумму
2. Выведите статистику частности букв в кортеже

```
long_word = ( 'т', 'т', 'а', 'и', 'и', 'а', 'и’,
              'и', 'и', 'т', 'т', 'а', 'и', 'и',
              'и', 'и', 'и', 'т', 'и’)
```

3. Допишите скрипт для расчета средней температуры

```
# Постарайтесь посчитать количество дней на основе week_temp.
# Так наш скрипт сможет работать с данными за любой период

week_temp = (26, 29, 34, 32, 28, 26, 23)
sum_temp =
days =
mean_temp = sum_temp / days
print(int(mean_temp))
```