Строки

Списки

Кортежи

Словари

Множества

Строки **(string)**

- Строка-это последовательность букв

Для обозначения строки используются одинарные или двойные кавычки.

Для длинных строк более удобна другая запись – строка, заключенная в группы из трех одинарных или двойных кавычек.

Строки в языке Python неизменяемы

Управляющие последовательности

- Короткие строки могут содержать управляющие последовательности, кроме обратной косой черты (‘\’), символов перехода на новую строку и кавычек, в которые строка заключена.

• Последовательность	• Представляемый символ
\newline	Игнорируется (newline-символ новой строки)
\\	Символ косой черты
\”	Двойная кавычка
\t	Символ горизонтальной табуляции
\v	Символ вертикальной табуляции

Операции над строками

- Строки можно объединить (склеить) с помощью оператора +

  Пример:

  s = ‘Hello’+’A’

  Результат:

  s = ‘HelloA’

- Строки можно размножить с помощью оператора *

  Пример:

  s = ‘Word*3’

  Результат:

  s = ‘WordWordWord’

Индексы

- Первый символ строки имеет индекс 0
- Индексы могут иметь отрицательные значения для отсчета с конца

Подстрока

- Строка-последовательность символов с произвольным доступом.

Любой символ строки может быть получен по его индексу.

Подстрока может быть определена с помощью среза – двух индексов, разделенных двоеточием.

Пример:

s = ‘Hello’

[0:2]

Результат:

‘He’

Длина строки

- Встроенная функция len () возвращает длину строки

Пример:

s = ‘Monday begins on saturday’

Len (s)

Результат:

25

Списки

Список является упорядоченным множеством элементов, перечисленных в квадратных скобках.

Совсем необязательно, чтобы элементы списка были одного типа

Пример

```
s = [‘hello’, 100,  5]
```

Индексы

Как и для строк, для списков нумерация индексов начинается с нуля.

При использовании отрицательных индексов отсчет ведется с конца списка.

Срезы

Указав через двоеточие два индекса, вы можете получить подмножество элементов списка, называемое “срезом”. Получаемое значение является новым списком, содержащим все элементы исходного списка в том же порядке.

Нумерация элементов начинается с нуля

Изменение отдельных элементов списка

В отличие от строк существует возможность изменения отдельных элементов списка

Пример:

```
a=[‘Alla’, 100, 34]
   a[1]= a[1]+19
```

Результат:

```
a=[‘Alla’, 119, 34]
```

Длина списка

Встроенная функция len() также применима к спискам, как и к строкам

Добавление элементов в список

- Метод append добавляет один элемент в конец списка.

  Пример: a. append(‘new’)

  Результат: [‘Alla’, 119, 34, ‘new’]

Метод insert вставляет один элемент в список. Целочисленный аргумент является индексом первого элемента, позиция которого изменится.

Пример: a.insert(1, ‘new’)

Результат: [‘Alla’, 119, ‘new’ ,34, ‘new’]

Метод extend добавляет в конец элементы другого списка.

Пример: a.extend([‘two’, ‘elements’])

Результат: [‘Alla’, 119, ‘new’ ,34, ‘new’, ‘two’, ‘elements’]

Изменение элементов списка

a = [3, 8, 15, 43]

Замена нескольких элементов:

Пример: a[0:2] = [1,12]

Результат: [1, 12, 15, 43]

Удаление элемента:

Пример: a[0:2] = [ ]

Результат: [15, 43]

Вставка:

Пример: a[1:1] = [‘Hello’, 5]

Результат: [3, ‘Hello’, 5, 8, 15, 43]

Копия самого себя в начале:

Пример: a[:0]=a

Результат: [3, 8, 15, 43, 3, 8, 15, 43]

Удаление элементов из списка

Метод remove удаляет из списка первый элемент с указанным значением.

Пример: a.remove (‘new’)

Результат: [‘Alla’, 119, 34, ‘new’, ‘two’, ‘elements’]

Метод remove удаляет *только* один элемент.

В данном случае строка "new" присутствует в списке дважды, но a.remove("new") удалит только первую.

Применение операторов к спискам

С помощью
оператора + можно “склеивать” списки
Оператор * размножает элементы списка.

Расширенная запись списков

Одна из самых мощных особенностей языка Python — расширенная запись списков, которая позволяет легко преобразовать один список в другой, применяя к каждому элементу функцию.

Пример:

```
li = [1, 9, 8, 4]

li =[elem*2 for elem in li]
```

Результат:

```
[2, 18, 16, 8]
```

li — список, который вы преобразуете. Python пробегает по всем элементам li, временно присваивает каждый из них переменной elem, вычисляет значение выражения elem*2 и добавляет в конец списка, который вы в результате получаете

Задачи на тему « Списки»

Задача 1. Найти сумму элементов списка

```
l1 = [1, 99, 57896, 21, 35, 176]


res = 0
for i in l1:
    res += i

print(res)
```

Задача 2. Ввести элементы списка произвольной длины

```
sp = [int(i) for i in input('Введите элементы списка через пробел ').split()]
print(sp)
```

Кортежи (tuple)

Кортеж — это неизменяемый список.

Кортеж определяется так же, как и список, но элементы перечисляются в круглых скобках вместо квадратных.

Как и в списках, элементы в кортежах имеют определенный порядок. Точно так же нумерация элементов начинается с нуля.

К кортежам, как и к спискам можно применить операцию среза. Обратите внимание, что срез списка — новый список, а срез кортежа — новый кортеж.

Операции с кортежами

Нельзя добавлять элементы в кортеж

Нельзя удалять элементы из кортежа

Нельзя искать элементы в кортеже с помощью index

Однако, можно с помощью in

При совершении операций с кортежем (например +=) создается новый кортеж

Упаковка и распаковка в кортеж

t= 12345, 54321, ‘hello’

Пример упаковки в кортеж

x, y, z = t

Распаковка кортежа требует, чтобы слева стояло столько переменных, сколько элементов в кортеже

Пустые и одноэлементные кортежи

Пустой кортеж создается с помощью пустой пары скобок

Кортеж с одним элементом создается с помощью значения и следующей за ним запятой, просто значения недостаточно

Связь кортежа и списка

Кортеж может быть преобразован в список и наоборот. Встроенная функция tuple воспринимает список в качестве аргумента и возвращает кортеж с теми же самыми элементами, и функция list воспринимает кортеж в качестве аргумента и возвращает список.

В результате tuple “замораживает” список, а list его “размораживает”.

Задачи на кортежи

Задача 1. Перебор элементов кортежа

```
thistuple = ("помидор",  "огурец",  "лук")
for x in thistuple:
    print(x)
print(len(thistuple))
```

Задача 2. Создать кортеж из элементов строки

- s = '1,2,3,4,5,6'
- t = tuple(map(int, s.split(',')))
- print(t)

Словари

Словари в Python - неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами.

Элементы словаря состоят из двух компонентов. Первый называется «ключ», второй – «значение».

Простейшие примеры информации, которую в программе удобно хранить в виде словаря:

- имя человека и дата его дня рождения;
- номер авиарейса и аэропорт назначения;
- название государства и его столица

Ключом в словаре Python может быть любой так называемый «неизменяемый тип» данных, к которому относится число, символьная строка или кортеж (неизменяемый набор значений).

Создание словаря

Если на момент написания программы известны все элементы словаря, то последний создается так:

< имя словаря > = {< ключ 1 >: < значение 1 >, {< ключ 2 >: < значение 2 >, ...}

структура: «номер авиарейса : аэропорт назначения»):

```
R = {'ПЛ6553': 'Сочи', 'ЮТ381': 'Санкт-Петербург', 'ДР181': 'Волгоград', 'ДР157': 'Краснодар'}
```

Когда тип всех ключей – строковый и они не содержат пробелов, то для создания словаря удобно использовать функцию dict()

```
R = dict(ПЛ6553 = 'Сочи', ЮТ381 = 'Санкт-Петербург', ДР181 ='Волгоград', ДР157 = 'Краснодар')
```

Добавить элемент в уже существующий словарь можно, указав новый ключ и новое значение в виде:

R['АБ1234'] = 'Сургут'

Если при этом указать уже существующий ключ, соответствующее ему значение будет изменено на новое.

Если же все элементы словаря становятся известными в ходе выполнения программы (с использованием инструкции input() или после расчетов), то сначала надо описать словарь как пустой:

D = {}

```
for k in range(…):
 kl = input('Введите ключ очередного элемента словаря')
 zn = input('Введите значение очередного элемента словаря')
 R[kl] = zn
```

```
d = {'cat': 'кошка', 'dog': 'собака', 'bird': 'птица',
'mouse': 'мышь'}
print(d)
{'dog': 'собака', 'cat': 'кошка', 'mouse': 'мышь',
'bird': 'птица'}

R = {'ПЛ6553': 'Сочи', 'ЮТ381': 'Санкт-Петербург',
'ДР181':'Волгоград', 'ДР157': 'Краснодар'}


print(R['ЮТ381'])
```

Ключ элементов должен быть уникальным по отношению к другим.

Если **КЛЮЧ** в словаре повторяется, то будет использовано то значение с данным ключом, которое ближе к концу словаря

	Вид	Формат	Комментарий
`len()`	Функция	`len(<имя словаря>)`	Возвращает количество элементов словаря
`keys()`	Функция	`keys(<имя словаря>)`	Возвращает список ключей словаря
`values()`	Функция	`values(<имя словаря>)`	Возвращает список значений словаря
`del`	Оператор	`del<имя словаря>[<ключ>]`	Удаляет элемент словаря по его ключу
`clear()`	Метод	`<имя словаря>.clear()`	Удаляет все значения из словаря

Задачи на тему «Словари»

Задача 1.

Даны два словаря: dictionary_1 = {'a': 300, 'b': 400} и dictionary_2 = {'c': 500, 'd': 600}. Объедините их в один при помощи встроенных функций языка Python.

```
dictionary_1 = {'a': 100, 'b': 200}
dictionary_2 = {'x': 300, 'y': 200}
dictionary_3 = dictionary_1.copy()
dictionary_3.update(dictionary_2)
print(dictionary_3)
```

Для объединения двух словарей создадим третий словарь в виде копии первого.

Для этого используем встроенную функцию copy().

Далее к уже созданному словарю мы присоединяем второй словарь. Для этого мы используем встроенную функцию update().

Задача 2.

Дан словарь с числовыми значениями. Необходимо их все перемножить и вывести на экран.

```
m= {'data1': 375, 'data2': 567, 'data3': -37, 'data4': 21}
result = 1
for i in m:
    result = result * m[i]
print(result)
```

Задача 3

Создайте словарь, в котором ключами будут числа от 1 до 10, а значениями эти же числа, возведенные в куб.

```
my_dict = {i : i ** 3 for i in range(1, 11)}
print(my_dict)
```

Задача 4

Даны два списка одинаковой длины. Необходимо создать из них словарь таким образом, чтобы элементы первого списка были ключами, а элементы второго — соответственно значениями нашего словаря.

```
keys = ['red', 'green', 'blue']
values = ['#FF0000','#008000', '#0000FF']
color_dictionary = dict(zip(keys, values))
print(color_dictionary)
```

Функция zip, которая из двух списков создает один, состоящий из кортежей длиной в два элемента каждый. Первый элемент кортежа взят из первого списка, а второй, соответственно, из второго. Индексы обоих элементов кортежа совпадают.
После передачи такого списка в функцию dict получается требуемый словарь.

Задача 5.

Создайте словарь из строки 'pythonist' следующим образом: в качестве ключей возьмите буквы строки, а значениями пусть будут числа, соответствующие количеству вхождений данной буквы в строку.

```
str1 = 'pythonist'
my_dict = {i: str1.count(i) for i in str1}
print(my_dict)
```

Для решения данной задачи воспользуемся функцией count(), которая считает количество вхождений элемента в строку.

Множества

Множества (set) представлены как неупорядоченные коллекции уникальных и неизменяемых объектов. Коллекции, которые не являются ни последовательностями (как списки), ни отображениями (как словари). Хотя с последними у множеств много общего.

Пример set-ов в Python:

```
# множество натуральных чисел от 1 до 10
natural_num_set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
# множество приближений math.sqrt(2)
sqrt_approximation_set = {1.4142135623, 1.414213562,
1.41421356, 1.4142135, 1.414213}
# множество результатов какого-то голосования
vote_result_set = {('P', 76.69), ('G', 11.77), ('J', 5.65), ('S',
1.68), ('Y', 1.05),
```

Особенности set

Одно из основных свойств множеств заключается в уникальности каждого из их элементов. Посмотрим, что получится, если сформировать set из строчки с заведомо повторяющимися символами:

```
strange_app = set(‘Informatika')
print(strange_app)
```

результат {‘I', ‘n', ‘f', 'k’,’o’,’r’,’m’,’t’,’a’}