Колекції

Рядки

Робота з текстовими даними у Python реалізована через ***str***-об'єкти або ***рядки***.

Рядок — це незмінна впорядкована послідовність символів у деякому кодуванні. За замовчуванням використовується кодування UTF-8, але можна працювати майже з усіма відомими таблицями кодування символів. Для того, щоб створити змінну типу "***рядок***", необхідно певний набір символів взяти в лапки.

Варіант 1. Одинарні лапки (апостроф) 'some text'

Варіант 2. Подвійні лапки "some text".

Різні варіанти використання лапок обумовлені тим, що при використанні одинарних лапок можна в рядку вказати подвійні й навпаки.

```
str1 = ‘Авто “BMW”’

str2 = “Об’єкт”
```

Впорядкована послідовність означає, що до елементів рядка можна звертатися за індексом:

```
s = "Hello world!"

print(s[0])   # H

print(s[-1])  # !
```

Важливо! Індексація всіх послідовностей в Python починається з 0!

Незмінна послідовність означає, що якщо рядок вже створено, то змінити його неможливо, можна тільки створити новий.

```
s = "Hello world!"
```

```
s[0] = "Q" # Тут буде викликано виключення (помилка) TypeError
```

Для об'єднання рядків використовують оператор «**+**»

```
s1, s2 = 'Hello ', 'world'

s = s1 + s2  # Hello world
```

Для багаторазового повторення використовують оператор «*****»

```
S = 'SPAM' * 5  # SPAMSPAMSPAMSPAMSPAM
```

Досить часто необхідно отримати не один якийсь символ (за індексом), а деякий набір символів за певними простими правилами. Наприклад: перші 5 символів, останні 3 символи, кожен другий символ. У таких завданнях замість перебору в циклі набагато зручніше використовувати так званий ***зріз*** (`***slice***, slicing`). ***Зріз*** – отримання з даного рядка набору з його символів. Зріз рядка також є рядком. Отримання одного символу рядка є найпростішим варіантом зрізу.

Задати ***зріз*** можна одним із двох варіантів:

```
str[start: stop]str[start: stop: step]
```

Для рядку ***str*** береться зріз від символу з індексом start до символу з індексом stop (не включаючи його), з кроком step (тобто будуть взяті символи з індексами `start`, `start + step`, `start + 2 * step і т. д.`). Також при записі зрізу деякі, а можливо й усі параметри, можуть бути опущені (знаки двокрапки в записі все одно залишаються). У випадку відсутності деяких параметрів, їхнє значення встановлюється за замовчуванням, а саме: `start = 0`, `stop = кількості символів рядка`, `step = 1`.

У випадку, якщо параметри start і stop мають від'ємні значення, то нумерація відбувається з кінця (`кількість символів рядка + від'ємний індекс`). Якщо параметр step має від'ємне значення, то зріз береться справа наліво. Приклади зрізів:

```
str = 'ABCDEFGHIJ'

str[1]  # 'B'
```

```
str[1:4]  # 'BCD'

str[:4]  # 'ABCD'

str[4:]  # 'EFGHIJ'

str[-4:]  # 'GHIJ'

str[1:-1]  # 'BCDEFGHI'

str[1:-1:2]  # 'BDFH'

str[::-1]  # 'JIHGFEDCBA'
```

Кілька основних функцій для роботи з рядками

Для того, щоб дізнатися кількість символів у рядку (довжину рядка), необхідно скористатися функцією ***len(рядок)***

```
len('Привіт!')  # 7
```

Для отримання коду символу можна скористатися функцією ***ord(c)***, за якою повертається ціле число, яке відповідає коду цього символу.

```
ord('a'), ord('€')  # 97, 8364
```

Зворотною до функції ***ord()*** є функція ***chr(n)***, за якою для цілого числа n повертається символ (рядок, що є єдиним символом), для якого n є його кодом.

```
chr(97), chr(8364)  # a, €
```

Кілька основних методів для роботи з рядками

`str.find(substr [, start [,end]])`. Повертає найменший індекс, за яким знаходиться початок підрядка substr в зрізі `str[start:end]`. Тобто знаходиться перше входження підрядка в рядку. Значення, що повертається, є індексом рядка `str`. Якщо підрядок не знайдено, то повертається значення `-1`.

```
'habrahabr'.find('r')  # 3
```

```
'habrahabr'.find('r',4)  # 8

'habrahabr'.find('Abr')  # -1
```

`str.rfind(substr [, start [,end]])`. Повертає найбільший індекс, за яким знаходиться початок підрядка substr в зрізі `str[start:end]` (необов'язкові параметри start та `end` інтерпретуються як нотації зрізу). Тобто знаходиться останнє входження підрядка в рядку. Значення, що повертається, є індексом рядка str. Якщо підрядок не знайдено, то повертається значення `-1`

```
'habrahabr'.rfind('abr')  # 6
```

`str.index(substr [, start [,end]])`. Повертає найменший індекс, за яким знаходиться початок підрядка substr в зрізі str[start:end] (необов'язкові параметри `start` та `end` інтерпретуються як нотації зрізу). Тобто знаходиться перше входження підрядка в рядку. Значення, що повертається, є індексом рядка `str`. Якщо підрядок не знайдено, то виникає виняток `ValueError`.

`str.rindex(substr [, start [,end]])`. Повертає найбільший індекс, за яким знаходиться початок підрядка `substr` в зрізі `str[start:end]` (необов'язкові параметри `start` та end інтерпретуються як нотації зрізу). Тобто знаходиться останнє входження підрядка в рядку. Значення, що повертається, є індексом рядка `str`. Якщо підрядок не знайдено, то виникає виняток `ValueError`

`str.startswith(prefix[, start[, end]])`. Повертає `True`, якщо зріз `str[start:end]` (необов'язкові параметри `start` та end інтерпретуються як нотації зрізу) починається з префікса `prefix`, , інакше – `False`.

`str.endswith(suffix[, start[, end]])`. Повертає `True`, якщо зріз `str[start:end]` закінчується на суфікс suffix, інакше – `False`.

`str.count(substr [, start [,end]])`. Повертає кількість входжень підрядка sub в зріз `str[start:end]` (необов'язкові параметри s`tart` та `end` інтерпретуються як нотації зрізу) без самоперетинів.

`str.upper()`. Повертає копію рядка, в якому всі літери, записані в нижньому регістрі, будуть приведені до верхнього регістру.

`str.lower()`. Повертає копію рядка, в якому всі літери, записані в верхньому регістрі, будуть приведені до нижнього регістру.

`str.swapcase()`. Повертає копію рядка, в якому всі літери, записані в верхньому регістрі, будуть приведені до нижнього регістру, а нижньому – до верхнього.

`str.title()`. Повертає копію рядка, в якому перша літера кожного слова буде приведена до верхнього регістру, а всі інші – до нижнього.

`str.capitalize()`. Повертає копію рядка, в якому перший символ, якщо він є літерою, буде приведений до верхнього регістру, а всі інші літери до нижнього.

`str.replace(old, new[, count])`. Повертає копію рядка, в якому всі входження підрядка old будуть замінені на новий підрядок `new`. Якщо задано параметри count, то буде виконано не більше ніж `count` замін

`str.strip([chars])`. Повертає копію рядка з вилученими початковими та кінцевими символами, вказаними в рядку `chars`. Якщо параметр `chars` відсутній або `None`, то вилучаються пропуски.

`str.split(sep=None, maxsplit=-1)`. Повертає список слів, які отримуються розбиттям рядка, за **роздільником** рядком `sep` з`tr.join(iterable)`. Повертає рядок, який є результатом конкатенації всіх рядків з `iterable`. Під час конкатенації між рядковими елементами `iterable` буде розміщений рядок `str`.

`str.isalpha()`. Повертає `True`, якщо рядок є непорожнім і складається лише з алфавітних символів, інакше – `False`.

`str.isdecimal()`. Повертає `True`, якщо рядок є непорожнім і складається лише з десяткових цифр (десяткових символів), інакше – `False`.

`str.isdigit()`. Повертає `True`, якщо рядок є непорожнім і складається лише з цифр, інакше – `False`

`str.islower()`. Повертає `True`, якщо рядок містить принаймні одну літеру й усі літери записані в нижньому регістрі, інакше – `False`.

`str.isupper()`. Повертає `True`, якщо рядок містить принаймніодну літеру й усі літери записані в верхньому регістрі, інакше – `False`.

Списки

Список (list) – це структура даних для зберігання елементів (об'єктів) не обов'язково одного типу. Список є змінюваним типом даних. Списки записуються як перелік елементів, розділених комою та взятих у квадратні дужки: `[1, 2, 3, 'Hello']`.

Для задання порожнього списку можна скористатися однією знаступних команд:

```
a = []

a = list()
```

Список можна отримати з елементів об'єкта, що може ітеруватися (діапазон, рядок, словник, множина, кортеж, файл тощо), використавши функцію **list([iterable]).**

До списків, як і до рядків, можна застосовувати операції «**+**» та «*****» і вибір елементів списку за допомогою ***зрізів***. Задати зріз можна одним із двох варіантів:

```
item[start: stop].

item[start: stop: step].
```

Також `«»` використовують для ***розпакування*** списку. ***Розпакування (unpacking)*** є розкладанням колекції (кортежу, списку тощо) на окремі значення.

```
lst1 = [1, 2, 3]

print(lst1)  # [1, 2, 3]

print(*lst1)  # 1 2 3
```

Оскільки списки – змінювані, то елементи списку можна змінювати чи видаляти за індексом.

```
b = [1, 3, 5, 7, 9]

b[1:3] = [11, 12]  # [1, 11, 12, 7, 9]

del b[2:4]  #  [1, 11, 9]
```

Функції списків

Щоб порахувати довжину списку, ми використовуємо вбудовану функцію ***len().***

Вбудована функція ***sum()*** приймає в якості параметра список чисел і обчислює суму його елементів. Вбудовані функції ***min()*** і ***max()*** приймають в якості параметра список і знаходять мінімальний і максимальний елементи відповідно.

Функція `sorted()` повертає відсортовану копію списку

Методи списків

`list.append(x)`**.** Додає елемент x в кінець списку list.

`list.extend(iterable)`**.** Розширює наявний список list за рахунок додавання до нього всіх елементів з iterable.

`list.insert(n, x)`. Вставляє в список list елемент x в позицію n (індекс елемента, після якого буде вставлений елемент).

list.remove(x). Вилучає перше входження елемента x зі списку list.

`list.pop([n])`. Вилучає з списку list елемент з позиції n та повертає його, як результат виконання функції. Якщо використовувати метод без параметра, то буде вилучений останній елемент списку.

`list.clear()`. Очищує список list (вилучає всі елементи зі списку)

`list.index(x[, start[, end]])`**.** Повертає індекс першого входження елемента x в зрізі list[start: end]

`list.count(x)`**.** Повертає кількість входжень елемента x в список.

`list.sort(key=None, reverse=False)`**.** Відсортовує елементи списку (аргументи методу можуть бути використані для налаштування сортування). За замовчуванням сортування відбувається за зростанням. Для сортування в зворотному порядку використовуйте параметр reverse = True. В результаті сортування змінюється сам список.

`list.reverse()`. Змінює порядок розташування елементів у списку на зворотний. Змінюється сам список.

`list.copy()`**.** Повертає копію списку.

Генератор списку (List Comprehensions)

У Python є синтаксична конструкція, яка дозволяє в один рядок заповнювати списки простими або складними значеннями. Називається вона – ***генератори списків,*** або ***List Comprehensions***.

Усі генератори списків будуються за однаковим шаблоном, який має такий вигляд:

`список = [вираз for елемент in колекція]`

Наприклад,

```
a = [i**2 for i in range(8)]
```

Можливий варіант з умовою:

список = `[вираз for елемент in колекція if умова]`

```
a = [i**2 for i in range(8) if i % 2 == 0]
```

Кортежі

Кортеж (tuple) - це незмінна структура даних, яка за своєю будовою дуже схожа на список. Інколи навіть кажуть, що кортеж – це незмінюваний список. Так само, як і список, кортеж може містити елементи різних типів. Кортеж записується як перелік елементів, розділених комою та взятих в круглі дужки: `(1, 3, 5, 'Hello')`.

Існує кілька причин, коли варто використовувати кортежі замість списків. Першою причиною є можливість захисту даних від випадкової зміни (захист від дурня). Якщо ми отримали набір даних і є необхідність опрацьовувати його без зміни даних, то це як раз той випадок, коли доцільно використати кортеж.

Другою причиною є те, що кортежі в пам'яті займають менший об'єм у порівнянні зі списками. Третьою причиною є приріст продуктивності, який пов'язаний з тим, що кортежі працюють швидше, ніж списки (наприклад, операції перебору елементів). Четвертою причиною є можливість використання кортежів в якості ключа у словнику.

Для задання порожнього кортежу можна скористатися однією з наступних команд:

```
a = ()
```

```
b = tuple()
```

Кортеж можна отримати з елементів об’єкта, що може ітеруватися (діапазон, рядок, словник, множина, кортеж, файл і т.д.), використавши функцію ***tuple([iterable])***

До кортежів, як і до списків, можна застосовувати операції «***+***» та «*******» та вибір елементів кортежів за допомогою ***зрізів***. Задати зріз можна одним із двох варіантів:

```
tuple[start: stop].

tuple[start: stop: step].
```

Можна перевірити приналежність деякого елемента до кортежу, використовуючи оператор ***in*** (значення in ім’я_кортежу).

Але, як вже було сказано, змінювати елементи кортежу не можна!

Методи кортежів

Враховуючи незмінюваність кортежів, вони мають лише методи: `tuple.index(x[, start[, end]])` та `tuple.count(x)`, призначення яких аналогічне до призначення однойменних методів списків.

Множини

Множина `(set)` - це структура даних, що містить невпорядкований набір унікальних елементів. Використання множин є доцільним у тому випадку, коли присутність елемента в наборі важливіша порядку слідування елементів та того, скільки разів цей елемент там зустрічається. Множина може містити елементи різних типів, проте ці елементи можуть бути лише незмінюваних типів даних: числа, рядки, кортежі.

Сама множина є змінюваним типом даних, тому до множин можна додавати нові та видаляти наявні елементи. Як і у випадку математичних множин, у мові Python передбачено виконання операцій над множинами: об'єднання, перетину, різниці, симетричної різниці.

На відміну від масивів, де елементи зберігаються у вигляді послідовного списку, у множинах порядок зберігання елементів невизначений. Це дозволяє виконувати операції типу "перевірити приналежність елемента множині" швидше, ніж просто перебираючи всі елементи множини.

Множини записуються, як перелік елементів, розділених комою та взятих у фігурні дужки: `{1, 2, 3, 'Hello'}`

Задання порожньої множини виконується з використанням функції ***`set()`***:

```
a = set()
```

Використання «{}» призведе до створення порожнього словника:

```
s={}

type(s)  # <class 'dict'>
```

Проте можна задати множину, перерахувавши її елементи, взяті в фігурні дужки «`{}`»:

```
s1 = {1, 2, 3}
```

Множину можна отримати з елементів об'єкта, що може ітеруватися (діапазон, список, рядок, словник, кортеж, файл і т.д.), використавши функцію ***`set([iterable])`***. Проте варто пам'ятати, що до множини будуть включені лише унікальні елементи.

Можна перевірити приналежність деякого елемента до множини, використовуючи оператор ***`in`*** (значення in ім'я_множини).

Порівняння множин

Множини можна порівнювати між собою. Порівняння множин зводиться до перевірки, чи є множини рівними або чи є певна множина підмножиною іншої.

`set == other`. Перевірка, чи є множини `set` та other рівними. Повертає `True`, якщо всі елементи множини `set` належать множині `other`, і всі елементи множини `other` належать множині `set`, інакше – `False`.

`set != other`. Перевірка, чи є множини `set` та other не рівними. Повертає `True`, якщо принаймні один елемент множини `set` не належить множині `other`, або принаймні один елемент множини other не належить множині `set`, інакше – `False`

`set <= other`. Перевірка, чи є множина set підмножиною множини other. Поверта , якщо всі елементи множини `set` належать множині `other`, інакше – `False`.

`set < other`. Повертає True, якщо всі елементи множини set належать множині other, але не всі елементи множини other належать множині `set`, інакше – `False`.

`set.isdisjoint(other)`. Повертає True, якщо множини set і other не мають спільних елементів, інакше – False

`set.issubset(other)`. Перевірка, чи є множина set підмножиною множини other. Аналогічно до `set <= other`.

`set.issuperset(other)`. Перевірка чи є множина other підмножиною множини set. Аналогічно до set >= other

Методи множин

`set.add(x)`. Додає елемент `x` до множини `set`

set.remove(x). Вилучає елемент `x` із множини `set`. Якщо такого елемента в множині немає, то виникає виняток `KeyError`.

`set.discard(x)`. Вилучає елемент `x` із множини `set`. Якщо такого елемента в множині немає, то нічого не відбувається

`set.pop()`. Вилучає «перший» елемент з множини `set` та повертає його значення, як результат виконання функції. Так як множина – це невпорядкований набір, то не можна точно передбачити, який з елементів буде взятий як перший. Якщо множина порожня, то виникає виняток `KeyError`

`set.clear()`. Очищує множину `set` (вилучає всі елементи з множини)

Операції з множинами

`set.union(*other)` або `set | other | ...` Повертає об'єднання множин `set` і `other`. Множина-результат буде містити як елементи множини `set`, так і елементи множини `other`

`set.intersection(*other)` або `set & other & ...` Повертає перетин множин `set` і `other`. Множина-результат буде містити елементи, які належать як множині `set`, так і множині `other`

`set.difference(*other)` або `set - other - ...` Повертає різницю множин `set` і `other`. Множина-результат буде містити елементи множини `set`, які не належать

множині `other`

`set.symmetric_difference(*other)` або **`set ^ other`**. Повертає симетричну різницю множин `set` і `other`. Множина-результат буде містити елементи, які належать множинам `set` та `other`, але не належать обом множинам

`set.update(*other)` або **`set |= other`**. Додає до множини `set` всі елементи множини `other`. Множина set буде містити об'єднання множин `set` і `other`

`set.intersection_update(*other)` або **`set &= other`**. Вилучає з множини `set` всі елементи, які не входять до множини `other`. Множина `set` буде містити перетин множин `set` і `other`.

`set.difference_update(*other)` або **`set -= other`**. Вилучає з множини `set` всі елементи, які входять до множини `other`. Множина `set` буде містити різницю множин `set` і `other`.

`set.symmetric_difference_update(other)` або **`set ^= other`**. Множина set буде містити симетричну різницю множин `set` і `other`

Словники

Словник (dict) – це структура даних, призначена для зберігання довільних об'єктів з доступом за довільним ключем. Дані в словнику зберігаються в форматі ключ=значення. Ключі в межах словника мають бути унікальними, тобто двох однакових ключів в словнику бути не може. Ключ повинен мати незмінюваний тип даних: ціле або дійсне число, рядок, кортеж.

Словник записується як перелік пар ***ключ : значення***, розділених комою та взятих у фігурні дужки: `{'A1':2, 'A2':3}`

Словник є змінюваним типом даних: в нього можна додавати нові елементи з довільними ключами і вилучати вже існуючі елементи.

Для задання порожнього словника можна скористатися однією з наступних команд:

```
a = {}

b = dict()
```

Задання словника з наперед заданим набором елементів:

```
a = {'A1':2, 'A2':3}

b = dict(id1=4, id2=8)
```

Враховуючи те, що елементом словника є пара ключ=значення, то, як такого, доступу до елемента словник не має. В словнику передбачена можливість доступу до значення елемента словника за його ключем. Для того, щоб звернутися до значення елемента словника, необхідно вказати ім'я змінної словника та в квадратних дужках ключ необхідного елемента ***(ім'я_словника[ключ])***.

```
e = {'A1':2, 'A2':3}

print(e['A2'])96  # 3
```

При спробі доступу за неіснуючим у словнику ключем виникає виняток KeyErrror.

Можна перевірити приналежність деякого ключа до словника, використовуючи оператор ***in*** ***(ключ in ім'я_словника)***.

Щоб додати елемент до словника, потрібно вказати ім'я змінної словника, в квадратних дужках — новий ключ і виконати присвоєння нового значення ***(ім'я_словника[новий_ключ]=значення)***.

```
e['A3']=4
```

Для видалення елемента зі словника можна скористатися командою `del (del ім'я_словника[ключ])`.

```
del e['A1']
```

Методи словників

`dict.fromkeys(iterable [, value])`. Створює новий словник, ключами якого будуть елементи з `iterable` і з однаковим для всіх значенням `value`.

```
d.fromkeys(['a', 'b', 'c'],12)  # { 'a': 12, 'b': 12, 'c': 12}}
```

`dict.update([other])`. Доповнює словник `dict` парами (ключ=значення) зі словника `other`, якщо ключ вже присутній в словнику, то його значення оновлюється.

`dict.copy()`. Повертає копію словника `dict`.

`dict.get(key[, default])`. Повертає значення зі словника `dict` за ключем `key`. У випадку відсутності елемента з ключем `key` повертається значення `default`.

`dict.setdefault(key[, default])`. Повертає значення зі словника `dict` за ключем `key`. У випадку відсутності елемента з ключем `key` повертається значення `default`, і до словника додається елемент з ключем `key` і значенням `default`.

`dict.keys()`. Повертає ключі елементів словника `dict` у вигляді об’єкта перегляду словника, що забезпечують динамічний перегляд записів словника.

`dict.values()`. Повертає значення елементів словника `dict` у вигляді об’єкта перегляду словника, що забезпечують динамічний перегляд записів словника.

`dict.items()`. Повертає ключі та значення елементів словника `dict` у вигляді об’єкта перегляду словника, що забезпечують динамічний перегляд записів словника. Елементи словника подаються в вигляді кортежів (ключ, значення).

`dict.pop(key[, default])`. Вилучає зі словника `dict` елемент з ключем key та повертає його значення, як результат виконання функції. У випадку відсутності елемента з ключем key повертається значення `default`. Якщо `default` не вказаний і елемент з ключем key відсутній, то з’являється виняток `KeyError`.

`dict.popitem()`. Вилучає і повертає пару (ключ — значення) зі словника `dict`, як результат виконання функції. Пари повертаються в порядку `LIFO (last-in first-out)`. Якщо словник порожній, то виникає виняток `KeyError`.

`dict.clear()`. Очищує словник `dict` (вилучає всі елементи зі словника).