Лекция 6: Работа со строками

Автор: Сергей Вячеславович Макрушин, 2022 г.

e-mail: s-makrushin@yandex.ru (mailto:s-makrushin@yandex.ru)

V 0.3 14.10.2022

Разделы:

-

- к оглавлению

In [68]:
```
# загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v1.css")
HTML(html.read().decode('utf-8'))
```

Out[68]:

Форматирование строк

-

- к оглавлению

0. Конкатенация.

Грубый способ форматирования, в котором мы просто склеиваем несколько строк с помощью операции сложения:

In [2]:
```
name = "Петя"
age = 20
print("Меня зовут " + name + ". Мне " + str(age) + " лет.")
```

```
Меня зовут Петя. Мне 20 лет.
```

Склеивание строк из множества компонент с помощью оператора `+` неэффективный подход,

вместо него рекомендуется использовать функцию `join` или f-строки.

In [3]:
```
strings = [f'Строка номер {i}' for i in range(10)]
strings
```

Out[3]:
```
['Строка номер 0',
 'Строка номер 1',
 'Строка номер 2',
 'Строка номер 3',
 'Строка номер 4',
 'Строка номер 5',
 'Строка номер 6',
 'Строка номер 7',
 'Строка номер 8',
 'Строка номер 9']
```

In [4]:
```
# неэффективный способ:
res_s = ''
for s in strings:
    res_s = res_s + ' ' + s
res_s
```

Out[4]:
```
' Строка номер 0 Строка номер 1 Строка номер 2 Строка номер 3 Строка номер 4 Строка но
мер 5 Строка номер 6 Строка номер 7 Строка номер 8 Строка номер 9'
```

In [5]:
```
# эффективное решение задачи склеивания многих строк:
' '.join(strings)
```

Out[5]:
```
'Строка номер 0 Строка номер 1 Строка номер 2 Строка номер 3 Строка номер 4 Строка ном
ер 5 Строка номер 6 Строка номер 7 Строка номер 8 Строка номер 9'
```

In [6]:
```
# если не нужен разделитель:
''.join(strings)
```

Out[6]:
```
'Строка номер 0Строка номер 1Строка номер 2Строка номер 3Строка номер 4Строка номер 5С
трока номер 6Строка номер 7Строка номер 8Строка номер 9'
```

1. %-форматирование.

Способ, который перешел в Python из языка C и распространенный в Python 2.X . Передавать значения в строку можно через списки и кортежи , а также и с помощью словаря. Во втором случае значения помещаются не по позиции, а в соответствии с именами.

In [7]:
```
name = "Дмитрий"
age = 25
print("Меня зовут %s. Мне %d лет." % (name, age))
print("Меня зовут %(name)s. Мне %(age)d лет." % {"name": name, "age": age})
```

```
Меня зовут Дмитрий. Мне 25 лет.
Меня зовут Дмитрий. Мне 25 лет.
```

2. Template-строки.

Этот способ появился в Python 2.4, как замена %-форматированию, но популярным так и не стал. Поддерживает передачу значений по имени и использует $-синтаксис как в PHP.

In [8]:
```
from string import Template
name = "Дмитрий"
age = 25
s = Template('Меня зовут $name. Мне $age лет.')
print(s.substitute(name=name, age=age))
```

```
Меня зовут Дмитрий. Мне 25 лет.
```

3. Форматирование с помощью метода format().

Этот способ появился в Python 3 в качестве замены %-форматированию. Он также поддерживает передачу значений по позиции и по имени. На данный момент устарел из-за появления похожего но существенно более удобного механизма f-строк.

In [9]:
```
name = "Дмитрий"
age = 25
print("Меня зовут {}. Мне {} лет.".format(name, age))
print("Меня зовут {name} Мне {age} лет.".format(age=age, name=name))
```

```
Меня зовут Дмитрий. Мне 25 лет.
Меня зовут Дмитрий Мне 25 лет.
```

4. f-строки.

Форматирование, которое появилось в Python 3.6 . Этот способ похож на форматирование с помощью метода format(), но удобнее, гибче, читабельней и быстрей.

f-строки делают очень простую вещь - они берут значения переменных, которые есть в текущей области видимости, и подставляют их в строку. В самой строке вам лишь нужно указать имя этой переменной в фигурных скобках.

In [10]:
```
name = "Дмитрий"
age = 25
print(f"Меня зовут {name} Мне {age} лет.")
```

```
Меня зовут Дмитрий Мне 25 лет.
```

Форматирование в f-строках

- к оглавлению

f-строки также поддерживают расширенное форматирование чисел:

In [12]:
```
from math import pi
print(f"Значение числа pi: {pi:.2f}")
```

```
Значение числа pi: 3.14
```

In [13]:
```
pi
```

Out[13]:
```
3.141592653589793
```

С помощью f-строк можно форматировать дату без вызова метода strftime():

In [14]:
```
from datetime import datetime as dt
now = dt.now()
print(f"Текущее время {now:%d.%m.%Y %H:%M}")
```

```
Текущее время 15.10.2022 09:52
```

In [15]:
```
# Пример использования продвинутого форматирования:

print('0123456789'*4)

def f(a,b,c,d):
    return f'|{a: >3}|{b:_>8.2f}|{c:=8}|{d:0>8}|'

print(f(1, 59.06, 453, 1))
print(f(5, 159.00, 123.453, 111))
print(f(15, -159.10, -12,10000))
print(f(105, -1059.10, 1200, 1111111))
```

```
01234567890123456789012345678901234567890123456789
|  1|___59.06|     453|00000001|
|  5|__159.00| 123.453|00000111|
| 15|_-159.10|-     12|00010000|
|105|-1059.10|    1200|01111111|
```

Подробная информация по возможностям расширенного форматирования:

http://www.python-course.eu/python3_formatted_output.php (http://www.python-course.eu/python3_formatted_output.php)

- Возможности расширенного форматирования в f-строках и у функции format() совпадают.
- В параметре `<Формат>` (после двоеточия) указывается значение, имеющее следующий синтаксис: `[ [<Заполнитель>] <Выравнивание>] [<Знак>] [ #] [0] [<Ширина>] [,] [. <Точность>] [ <Преобразование>]`

In [48]:
```
# пример:
i1 = 3
s1 = "string"
f"'{i1:10}' '{s1:3}'"
```

Out[48]:
```
"'         3' 'string'"
```

По умолчанию значение внутри поля выравнивается по правому краю. Управлять выравниванием позволяет параметр <Выравнивание>. Можно указать следующие значения:

- < - по левому краю;
- > - по правому краю;
- ^ - по центру поля;
- = - знак числа выравнивается по левому краю, а число по правому краю.

In [18]:
```
val = 3
print(f"_0123456789_\n'{val:<10}'\n'{val:>10}'\n'{val:^10}'\n'{-val:=10}'")
```

```
_0123456789_
'3         '
'         3'
'    3     '
'-        3'
```

Пространство между знаком и числом по умолчанию заполняется пробелами, а знак положительного числа не указывается. Чтобы вместо пробелов пространство заполнялось нулями, необходимо указать нуль перед шириной поля. Такого же эффекта можно достичь, указав ноль в параметре `<Заполнитель>` . В этом параметре допускаются и другие символы, которые будут выводиться вместо пробелов:

In [19]:
```
print(f"'{-val:=010}'\n'{val:=010}'")
```

```
'-000000003'
'0000000003'
```

In [20]:
```
print(f"'{-val:0=10}'\n'{val:_=10}'")
```

```
'-000000003'
'_________3'
```

In [14]:
```
print(f"'{val:*<10}'\n'{val:+>10}'\n'{val:.^10}'")
```

```
'3*********'
'+++++++++3'
'....3.....'
```

Управлять выводом знака числа позволяет параметр `<Знак>` . Допустимые значения:

- `'+'` - задает обязательный вывод знака как для отрицательных, так и для положительных чисел;
- `'-'` - вывод знака только для отрицательных чисел (значение по умолчанию);
- `' '` - вставляет пробел перед положительным числом. Перед отрицательным числом будет стоять минус.

In [15]:
```
print(f"'{val:+}' '{-val:+}' '{val:-}' '{-val:-}'")
```

```
'+3' '-3' '3' '-3'
```

In [21]:
```
print(f"'{val: }' '{-val: }'")
```

```
' 3' '-3'
```

Для целых чисел в параметре `<Преобразование>` могут быть указаны следующие опции:

- `b` - двоичное значение:
- `c` - преобразует целое число в соответствующий символ;
- `d` - десятичное значение;
- `n` - аналогично опции d, но учитывает настройки локали. Например, выведем большое число с разделением тысячных разрядов пробелом;
- `o` - восьмеричное значение;
- `x` - шестнадцатеричное значение в нижнем регистре;
- `X` - шестнадцатеричное значение в верхнем регистре.

In [22]:
```
print(f"'{val:b}' '{val:#b}'")
```

```
'11' '0b11'
```

In [23]:
```
f"'{10:c}'"
```

Out[23]:
```
"'\n'"
```

In [24]:
```
print(f"'{10:c}'")
```

```
'
'
```

In [25]:
```
val_100 = 100
print(f"'{val_100:#x}'")
```

```
'0x64'
```

In [26]:
```
import locale
locale.setlocale(locale.LC_NUMERIC, 'Russian_Russia.1251')
```

Out[26]:
```
'Russian_Russia.1251'
```

In [27]:
```
val_big = 1_000_000_000
print(val_big)
print(f"{val_big:n}")
```

```
1000000000
1 000 000 000
```

In [28]:
```
print(f"{val_big:,d}")
```

```
1,000,000,000
```

In [25]:
```
val_200 = 200
print(f"'{val_200:o}' '{val_200:x}' '{val_200:X}'")
```

```
'310' 'c8' 'C8'
```

Для вещественных чисел в параметре `<Преобразование>` могут быть указаны следующие опции:

- `f` и `F` - вещественное число в десятичном представлении. Задать количество знаков после запятой позволяет параметр `<Точность>`;
- `e` - вещественное число в экспоненциальной форме (буква "e" в нижнем регистре);
- `E` - вещественное число в экспоненциальной форме (буква "e" в верхнем регистре);
- `n` - аналогично опции g, но учитывает настройки локали;
- `%` - умножает число на 100 и добавляет символ процента в конец. Значение отображается в соответствии с опцией f.

In [29]:
```
flt_lst_1 = [30, 18.6578145, -2.5]
print(f"'{flt_lst_1[0]:f}' '{flt_lst_1[1]:f}' '{flt_lst_1[2]:f}'")
```

```
'30.000000' '18.657815' '-2.500000'
```

In [30]:
```
flt_lst_2 = [18.6578145, -2.5]
print(f"'{flt_lst_2[0]:.7f}' '{flt_lst_2[1]:.2f}'") #.format()
```

```
'18.6578145' '-2.50'
```

In [31]:
```
flt_lst_3 = [3000, 18657.81452]
print(f"'{flt_lst_3[0]:e}' '{flt_lst_3[1]:e}'")
```

```
'3.000000e+03' '1.865781e+04'
```

In [32]:
```
flt_lst_4 = [3000, 18657.81452]
print(f"'{flt_lst_4[0]:E}' '{flt_lst_4[1]:E}'")
```

```
'3.000000E+03' '1.865781E+04'
```

In [33]:
```
flt_lst_5 = [0.086578, 0.000086578]
print(f"'{flt_lst_5[0]:%}' '{flt_lst_5[1]:.4%}'")
```

```
'8.657800%' '0.0087%'
```

Операции в f-строках

Они поддерживают базовые арифметические операции. Да, прямо в строках:

In [34]:
```
x = 10
y = 5
print(f"{x} x {y} / 2 = {x * y / 2}")
```

```
10 x 5 / 2 = 25.0
```

Позволяют обращаться к значениям списков по индексу:

In [35]:
```
planets = ["Меркурий", "Венера", "Земля", "Марс"]
print(f"Мы живим на планете {planets[2]}")
```

```
Мы живим на планете Земля
```

А также к элементам словаря по ключу:

In [36]:
```
planet = {"name": "Земля", "radius": 6378000}
print(f"Планета {planet['name']}. Радиус {planet['radius']/1000} км.")
```

```
Планета Земля. Радиус 6378.0 км.
```

Причем вы можете использовать как строковые, так и числовые ключи. Точно также как в обычном Python коде:

In [38]:
```
digits = {0: 'ноль', 'one': 'один'}
print(f"0 - {digits[0]}, 1 - {digits['one']}")
```

```
0 - ноль, 1 - один
```

Вы можете вызывать в f-строках методы объектов:

In [39]:
```
name = "Дмитрий"
print(f"Имя: {name.upper()}")
```

```
Имя: ДМИТРИЙ
```

А также вызывать функции:

In [40]:
```
print(f"13 / 3 = {round(13/3)}")
```

```
13 / 3 = 4
```

In [44]:
```
with open('my_out.txt', 'wt') as f:
    print(f"13 / 3 = {round(13/3)}", file=f)
```

In [47]:
```
with open('my_out.txt') as f:
    for line in f:
        print(line)
```

```
13 / 3 = 4
```

Регулярные выражения

- к оглавлению

Регулярные выражения (regular expressions, regexp) — формальный язык, используемый в программах, работающих с текстом, для поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании шаблонов, содержащих метасимволы (символы-джокеры, англ. wildcard characters).

Для поиска используется строка-образец (pattern, по-русски её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска.

- Например, регулярное выражение (шаблон) `Иван*` (`*` - здесь wildcard character) соответствует строкам любым строкам, начинающимся с `Иван` , например: `Иван` , `Ивану` , `Иванов` .

Современные языки высокого уровня поддерживают работу с регулярными выражениями либо на уровне языка либо на уровне библиотек (обычно базовых, поставляющихся с языком).

Регулярные выражения предназначены для выполнения сложного поиска или замены в строке. В языке Python использовать регулярные выражения позволяет встроенный модуль `re` .

Материалы по теме:

https://docs.python.org/3.7/howto/regex.html (https://docs.python.org/3.7/howto/regex.html)

https://www.python-course.eu/re.php (https://www.python-course.eu/re.php)

https://www.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html (https://www.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html)

https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285 (https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285)

https://regexone.com (https://regexone.com)

In [49]:
```
import re # Библиотека для работы с регулярными выражениями
```

Синтаксис регулярных выражений

- к оглавлению

Создать откомпилированный шаблон регулярного выражения позволяет функция `compile()`. Функция имеет следующий формат:

```
<Шаблон> = re.compile(<Регулярное выражение>[, <Модификатор>])
```

In [51]:
```
# Пример регулярного выражения
p = re.compile(r"[а-яё]{0,6}", re.I) # Шаблон, соответствующий строке, начинающейся на б
p.match('тикввввbtik').group(0) # Возвращаем результат поиска подстроки по указанному ша
```

Out[51]:
```
'тиквв'
```

Составление регулярных выражений

Специальные символы регулярных выражений

Внутри регулярного выражения символы `.` `^` `$` `*` `+` `?` `{` `}` `[` `]` `\` `|` `(` `)` - имеют специальное значение.

- Если эти символы требуется выводить как есть, то их следует экранировать с помощью слэша `\`.
- Некоторые специальные символы теряют свое особое значение, если их разместить внутри квадратных скобок `[]`. В этом случае экранировать их не нужно.

В квадратных скобках `[]` можно указать символы, которые могут встречаться на этом месте в строке. Можно перечислять символы подряд или указать их диапазон через тире. Например:

- `[09]` - соответствует цифре 0 или 9
- `[0-9]` - соответствует одной цифре от 0 до 9
- `[абв]` - соответствует букве "а", "б" или "в"
- `[а-г]` - соответствует букве "а", "б", "в" или "г"
- `[а-я]` - соответствует любой букве от "а" до "я", кроме буквы "ё" (т.к. "ё" находится вне непрерывного дипаозона символов русского алфавита)
- `[а-яё]` - соответствует любой букве от "а" до "я"
- `[АБВ]` - соответствует букве "А", "Б" или "В"
- `[А-ЯЁ]` - соответствует любой букве от "А" до "Я"
- `[а-яА-ЯёЁ]` - соответствует любой русской букве в любом регистре
- `[0-9а-яА-ЯёЁa-zA-Z]` - любая цифра и любая буква независимо от регистра и языка

Примеры:

In [36]:
```
# ОШИБОЧНЫЙ шаблон для проверки корректной даты:
p = re.compile(r'[0-3][0-9].[01][0-9].[12][09][0-9][0-9]')
```

In [53]:
```
date_ = "29,10.2016"
res = p.match(date_)
res
```

Out[53]:
```
<re.Match object; span=(0, 0), match=''>
```

In [54]:
```
bool(res)
```

Out[54]:
```
True
```

In [55]:
```
# Проверка на соответствие заданному шаблону:
if res:
    print('Введено правильно') # выдаст неверный ответ!
else:
    print('Введено неправильно')
```

```
Введено правильно
```

Так как точка не экранирована, то выведет "Введено правильно", т.к. точка означает любой символ, кроме перевода строки

In [56]:
```
bool(p.match("33.10.2016"))
```

Out[56]:
```
True
```

In [57]:
```
bool(p.match("ва.10.2016"))
```

Out[57]:
```
True
```

In [58]:
```
# Шаблон корректной даты с экранированием точки
# и более жесткими (но все-таки допускающими неверные значения!) ограничениями на запись
p2 = re.compile(r'(([0-2][0-9])|([3][01]))\.[01][0-9]\.[12][09][0-9][0-9]')
```

In [59]:
```
bool(p2.match("29,10.2016"))
```

Out[59]:
```
False
```

In [60]:
```
bool(p2.match("33.10.2016"))
```

Out[60]:
```
False
```

In [61]:
```
bool(p2.match("11.10.2016"))
```

Out[61]:
```
True
```

In [62]:
```
# Некорректная дата:
bool(p2.match("00.10.2016"))
```

Out[62]:
```
True
```

In [63]:
```
# Некорректная дата:
bool(p2.match("01.19.2016"))
```

Out[63]:
```
True
```

In [47]:
```
# Некорректная дата:
bool(p2.match("31.02.2016"))
```

Out[47]:
```
True
```

In [64]:
```
# Еще один вариант указать точку в шаблоне - заключить ее в квадратные скобки:
p3 = re.compile(r'[0-3][0-9][.][01][0-9][.][12][09][0-9][0-9]')
```

In [65]:
```
bool(p3.match("29,10.2016"))
```

Out[65]:
```
False
```

In [66]:
```
bool(p3.match("33.10.2016"))
```

Out[66]:
```
True
```

Если после первой скобки указать ^ , то все указанные в квадратных скобках символы должны отсутствовать на данной позиции шаблона.

In [67]:
```
p=re.compile(r'[^09][^0-9][^a-яA-ЯёЁa-zA-Z]') # Первый символ - не 0 и не 9; второй симв

if p.search('1a2'):
    print('Строка соответствует шаблону')
else:
    print('Строка не соответствует шаблону')


if p.search('04Q'):
    print('Строка соответствует шаблону')
else:
    print('Строка не соответствует шаблону')
```

```
Строка соответствует шаблону
Строка не соответствует шаблону
```

Вместо перечисления символов можно использовать стандартные классы:

- . - любой символ, кроме перевода строки (если точка не экранирована и не заключена в квадратные скобки)
- \d - соответствует любой цифре (эквивалентно [0-9])
- \w - соответствует любой букве, цифре или символу подчеркивания ([a-zA-Za-яЁA-ЯЁ0-9_])
- \s - любой пробельный символ (пробел, перевод строки, табуляция и т.д.)
- \D - не цифра (эквивалентно [^0-9])
- \W - не буква, не цифра и не символ подчеркивания (эквивалентно [^a-zA-Za-яЁA-ЯЁ0-9_])
- \S - не пробельный символ
- \b - обозначение левой или правой границы слова (где слово трактуется как последовательность букв или цифр)

In [68]:
```
p = re.compile(r'[\d][\D][\s][\S]')

if p.search('1a 2'):
    print('Строка соответствует шаблону')
else:
    print('Строка не соответствует шаблону')


if p.search('1a2 '):
    print('Строка соответствует шаблону')
else:
    print('Строка не соответствует шаблону')
```

```
Строка соответствует шаблону
Строка не соответствует шаблону
```

Квантификаторы

С помощью квантификаторов задается количество вхождений символа в строку. Указывается после символа, к которому относится разрешенное количество повторений:

- `{n}` - n вхождений символа в строку. Например. шаблон `r"[0-9]{2}"` соответствует двум вхождениям любой цифры
- `{n,}` - n или более вхождений символа в строку. Например. шаблон `r"[0-9]{2,}"` соответствует двум и более вхождениям любой цифры
- `{n,m}` - не менее n и не более m вхождений символа в строку. Числа указываются через запятую без пробела.
  - Например, шаблон `r"[0-9]{2,4}"` соответствует от двух до четырех вхождениям любой цифры
- `*` - ноль или большее число вхождений символа в строку. Эквивалентно комбинации `{0,}`
- `+` - одно или большее число вхождений символа в строку. Эквивалентно комбинации `{1,}`
- `?` - ни одного или одно вхождение символа в строку. Эквивалентно комбинации `{0,1}`.

In [69]:
```
p = re.compile(r'[0-9]{4}') #Вместо r'[0-9][0-9][0-9][0-9]'

if p.search('111'):
    print('Строка соответствует шаблону')
else:
    print('Строка не соответствует шаблону')

if p.search('1111'):
    print('Строка соответствует шаблону')
else:
    print('Строка не соответствует шаблону')
```

```
Строка не соответствует шаблону
Строка соответствует шаблону
```

Все квантификаторы являются "жадными": при поиске соответствия ищется самая длинная подстрока, соответствующая шаблону, и не учитываются более короткие соответствия. Например:

In [70]:
```
s="<b>Text1</b>Text2<b>Text3</b>"
p = re.compile(r'<b>.*</b>')
p.findall(s)
```

Out[70]:
```
['<b>Text1</b>Text2<b>Text3</b>']
```

Для ограничения жадности необходимо указать символ `?` после квантификатора:

In [71]:
```
# В этом случае будут искаться самые короткие подстроки^
p = re.compile(r'<b>.*?</b>')
p.findall(s)
```

Out[71]:
```
['<b>Text1</b>', '<b>Text3</b>']
```

Если необходимо получить только содержимое тегов, то нужный фрагмент шаблона следует разместить внутри круглых скобок

In [72]:
```
p = re.compile(r'<b>(.*?)</b>')
p.findall(s)
```

Out[72]:
```
['Text1', 'Text3']
```

Круглые скобки также часто используются для группировки фрагментов внутри шаблона. По умолчанию все фрагменты в скобках выводятся в результат. Чтобы избежать вывода конкретного фрагмента, следует после его открывающей круглой скобки разместить символы ?:

In [73]:
```
s = 'test tent text contest'
p = re.compile(r'[a-z]+((st)|(xt))') # Выводятся только фрагменты, которые заключены в к
print(p.findall(s))

p = re.compile(r'([a-z]+((st)|(xt)))') # За счет внешних скобок выводятся и целые слова,
print(p.findall(s))

p = re.compile(r'([a-z]+((?:st)|(?:xt)))') #С помощью ?: во всех вложенных фрагментах на
p.findall(s)
```

```
[('st', 'st', ''), ('xt', '', 'xt'), ('st', 'st', '')]
[('test', 'st', 'st', ''), ('text', 'xt', '', 'xt'), ('contest', 'st', 'st', '')]
```

Out[73]:
```
[('test', 'st'), ('text', 'xt'), ('contest', 'st')]
```

Привязка к началу строки или подстроки

- `^` - привязка к началу строки или подстроки. Зависит от модификаторов M (или MULTILINE) и S (или DOTALL)
- `$` - привязка к концу строки или подстроки. Зависит от модификаторов M (или MULTILINE) и S (или DOTALL)
- `\A` - привязка к началу строки (не зависит от модификатора)
- `\Z` - привязка к концу строки (не зависит от модификатора).

(модификаторы рассматриваются ниже)

In [74]:
```
p = re.compile(r'^help') # Привязка к началу строки, т.е. проверка на соответствие начал
s = 'help '
print(p.findall(s))
s = ' help'
print(p.findall(s)) # Строка начинается с пробела, поэтому не соответствует регулярному
```

```
['help']
[]
```

In [75]:
```
p = re.compile(r'help$') #Привязка к концу строки (проверка на соответствие конца строки
s = 'help '
print(p.findall(s))
s = ' help'
print(p.findall(s))
```

```
[]
['help']
```

При одновременном использовании привязок к началу и к концу строки, мы говорим, что ищем строку с данным количеством текстовых символов, которые соответствуют шаблону

In [213]:
```
p=re.compile(r'[a-zA-Za-яА-Я]{3}') #Без привязок
p.findall('2qwer')
```

Out[213]:
```
['qwe']
```

In [76]:
```
import re
p=re.compile(r'^[a-zA-Za-яА-Я]{3}$') #С привязками
print(p.findall('qwer'))
p=re.compile(r'^[a-zA-Za-яА-Я]{4}$')
print(p.findall('qwer'))
```

```
[]
['qwer']
```

Модификаторы

- к оглавлению

При определении регулярного выражения в параметре `<Модификатор>` могут быть указаны следующие флаги (или их комбинация через оператор `|`:

- L или LOCALE- учитываются настройки текущей локали
- I или IGNORECASE- поиск без учета регистра.
- M или MULTILINE- поиск в строке, состоящей из нескольких подстрок, разделенных символом новой строки ("\n"). Символ ^ соответствует привязке к началу каждой подстроки, а символ $ соответствует позиции перед символом перевода строки
- S или DOTALL- метасимвол "точка" будет соответствовать любому символу, включая символ перевода строки {\n). По умолчанию метасимвол "точка" не соответствует символу перевода строки. Символ ^ будет соответствовать привязке к началу всей строки, а символ $ - привязке к концу всей строки.
- X или VERBOSE - Включает многословные (подробные) регулярные выражения, которые могут быть организованы более ясно и понятно. Если указан этот флаг, пробелы в строке регулярного выражения игнорируются, кроме случаев, когда они имеются в классе символов (напр. в квадратных скобках) или им предшествует неэкранированный бэкслеш; это позволяет организовать регулярные выражения более ясным образом. Этот флаг также позволяет помещать в регулярные выражения комментарии, начинающиеся с '#', которые будут игнорироваться движком.

In [77]:
```
p = re.compile(r'[a-z]{3}', re.I) #Поиск без учета регистра
s = 'QWE'
p.findall(s)
```

Out[77]:
```
['QWE']
```

In [78]:
```
p = re.compile(r'''\d{1,2} #Ищем цифру или 2 цифры
[a-z]? #Ищем 0-1 букву
[\w]+ #Ищем непробельные символы (более 1)
''', re.X|re.I)
p1 = re.compile(r'\d{1,2}[a-z]?[\w]+', re.I) # То же самое в строку: менее наглядно
s = 'sss40A2'
s1 = '444a_banana293'
s2 = '4 banana'
p.findall(s), p1.findall(s1), p.findall(s2)
```

Out[78]:
```
(['40A2'], ['444a_banana293'], [])
```

Основные методы

- к оглавлению

- `re.match()` - Этот метод ищет по заданному шаблону в начале строки. Возвращает первое вхождение подстроки в виде объекта SRE_Match object, из которого:
  - можно получить результирующую подстроку с помощью функции group
  - индексы начальной и конечной позиции с помощью функций start() и end(), соответственно.
- `re.search()` - Этот метод ищет по заданному шаблону во всей строке. Возвращает первое вхождение подстроки в виде объекта SRE_Match object, из которого:
  - можно получить результирующую подстроку с помощью функции group
  - индексы начальной и конечной позиции с помощью функций start() и end(), соответственно.
- `re.findall()` - Этот метод возвращает список всех найденных совпадений (подстрок).
- `re.split()` - Этот метод разделяет строку по заданному шаблону. Первый аргумент функции - регулярное выражение, обозначающее разделитель, второй аргумент - исходная строка.
- `re.sub()` - Этот метод ищет шаблон в строке и заменяет его на указанную подстроку.
- `re.compile()` - собирает регулярное выражение в отдельный объект, который может быть использован для поиска. Это также избавляет от переписывания одного и того же выражения.

re.match(pattern, string)

In [79]:
```
result = re.match(r'Analytics', 'AV Analytics Vidhya AV')
print(result)
```

```
None
```

In [80]:
```
result = re.match(r'AV', 'AV Analytics Vidhya AV')
print(result)
print(result.group())
print("Индекс начальной позиции найденной подстроки = {}, \
Индекс конечной позиции найденной подстроки = {}".format(result.start(), result.end()))
```

```
<re.Match object; span=(0, 2), match='AV'>
AV
Индекс начальной позиции найденной подстроки = 0, Индекс конечной позиции найденной по
дстроки = 2
```

re.search(pattern, string)

In [82]:
```
result = re.search(r'Analytics', 'AV Analytics Vidhya AV')
print(result.group())
f"Индекс начальной позиции найденной подстроки = {result.start()}, Индекс конечной позиц
```

```
Analytics
```

Out[82]:
```
'Индекс начальной позиции найденной подстроки = 3, Индекс конечной позиции найденной п
одстроки = 12'
```

In [476]:
```
result = re.search(r'AV', 'AV Analytics Vidhya AV')
print(result)
print(result.group(0))
print("Индекс начальной позиции найденной подстроки = {}, \
Индекс конечной позиции найденной подстроки = {}".format(result.start(), result.end()))
```

```
<re.Match object; span=(0, 2), match='AV'>
AV
Индекс начальной позиции найденной подстроки = 0, Индекс конечной позиции найденной по
дстроки = 2
```

Метод search() ищет по всей строке, но возвращает только первое найденное совпадение.

re.findall(pattern, string)

In [83]:
```
result = re.findall(r'AV', 'AV Analytics Vidhya AV')
print(result)
```

```
['AV', 'AV']
```

re.split(pattern, string, [maxsplit=0])

In [84]:
```
result = re.split(r't', 'Analytics')
print(result)
```

```
['Analy', 'ics']
```

С помощью модуля re можно задать несколько шаблонов и знаков, по которым нужно разбить одну строку

In [85]:
```
result = re.split(r'\.|\\|_', r'User\Homework_Ivanov.docs')
print(result)
```

```
['User', 'Homework', 'Ivanov', 'docs']
```

re.sub(pattern, repl, string)

In [86]:
```
result = re.sub(r'India', 'the World', 'AV is largest Analytics community of India')
print (result)
```

```
AV is largest Analytics community of the World
```

In [87]:
```
result = re.sub(r'\.|\'| |,|=|-', '_', 'We don\'t.need=no-education')
print(result)
```

```
We_don_t_need_no_education
```

re.compile(pattern, repl, string)

In [89]:
```
pattern = re.compile(r'\.|\'| |,|=|-')
s = 'We don\'t.need=no-education'
pattern.sub('_', s), pattern.split(s)
```

Out[89]:
```
('We_don_t_need_no_education', ['We', 'don', 't', 'need', 'no', 'education'])
```

Сегментация текста

- к оглавлению

Рассмотрим следующие популярные задачи сегментации текста:

- Определение границ предложения
- Токенизация

Определение границ предложений

- к оглавлению

Проблема определения границы предложения

Как автоматически определять границы предложений? Обычно определяются по точке, но:

- Точка - имеет много значений:
  - граница предложения
  - сокращение: “Dr.”, “U.S.A.”
  - Разделитель в числах 3.14
  - ...
- Предложение может заканчиваться не только точкой:
  - ./!/?/!?/!!!/... и т.д.
  - разбор предложений с прямой и косвенной речью
  - разбор предложений со списками
  - на конце предложения может вообще не быть знаков препинания

Определение границ предложений

Рассмотрим задачу разрешения многозначности точки:

- Задача сводится к классификации точки на два класса: конец предложения или нет. Есть два подхода:
  - Подход основанный на правилах (rule-based). Пример правил:
    - перед точкой и после нее стоят цифры -> не к.п.
    - слово перед точкой есть в словаре сокращений -> не к.п.
    - ... правил может быть очень много и постоянно будут находится новые исключения.
  - Подход основанный на машинном обучении (machine learning):
    - решение задачи классификации одним из методов ML

Решение задач NLP (natural language processing, обработки текста на естественном языке) на основе машинного обучения:

Среди методов, применяемых в рамках подхода, выделяют методы обучения с учителем (supervised), методы обучения без учителя (unsupervised), методы частичного обучения с учителем (bootstrapping).

- ML не требует ручного труда по составлению правил и сокращает время разработки систем
- ML обчно предполагает наличие подходящего размеченного корпуса текстов, что не всегда возможно. Создание такого корпуса в любом случае требует значительных объемов ручного труда.
- модели (классификаторы) непрозрачны для понимания, т. к. не имеют явной лингвистической интерпретации

In [96]:
```
with open('phm.txt ') as f:
    lines = [l for l in f]
print(len(lines))
print(lines[0])
```

```
3
Постгуманизм – рациональное мировоззрение, основанное на представлении, что эволюция ч
еловека не завершена и может быть продолжена в будущем. Эволюционное развитие должно п
ривести к становлению постчеловека – гипотетической стадии эволюции человеческого вид
а, строение и возможности которого стали бы отличными от современных человеческих в ре
зультате активного использования передовых технологий преобразования человека. Постгум
анизм признаёт неотъемлемыми правами совершенствование человеческих возможностей (физи
ологических, интеллектуальных и т. п.) и достижение физического бессмертия. В отличие
от трансгуманизма, под определением постгуманизма также понимается критика классическо
го гуманизма, подчёркивающая изменение отношения человека к себе, обществу, окружающей
среде и бурно развивающимся технологиям, но окончательно разница между транс- и постгу
манизмом не определена и остаётся предметом дискуссий.
```

In [97]:
```
lines[0].split(".")
```

Out[97]:
```
['Постгуманизм – рациональное мировоззрение, основанное на представлении, что эволюция
человека не завершена и может быть продолжена в будущем',
 ' Эволюционное развитие должно привести к становлению постчеловека – гипотетической с
тадии эволюции человеческого вида, строение и возможности которого стали бы отличными
от современных человеческих в результате активного использования передовых технологий
преобразования человека',
 ' Постгуманизм признаёт неотъемлемыми правами совершенствование человеческих возможно
стей (физиологических, интеллектуальных и т',
 ' п',
 ') и достижение физического бессмертия',
 ' В отличие от трансгуманизма, под определением постгуманизма также понимается критик
а классического гуманизма, подчёркивающая изменение отношения человека к себе, обществ
у, окружающей среде и бурно развивающимся технологиям, но окончательно разница между т
ранс- и постгуманизмом не определена и остаётся предметом дискуссий',
 '\n']
```

In [98]:
```
import nltk # Natural Language Toolkit - русский язык поддерживается только в некоторых
```

In [99]:
```
# Загрузка модулей и наборов данных NLTK:
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml (http
s://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml)
```

Out[99]:
```
True
```

In [100]:
```
from nltk.tokenize import sent_tokenize
sent_tokenize(lines[0])
```

Out[100]:
```
['Постгуманизм — рациональное мировоззрение, основанное на представлении, что эволюция человека не завершена и может быть продолжена в будущем.',
 'Эволюционное развитие должно привести к становлению постчеловека — гипотетической стадии эволюции человеческого вида, строение и возможности которого стали бы отличными от современных человеческих в результате активного использования передовых технологий преобразования человека.',
 'Постгуманизм признаёт неотъемлемыми правами совершенствование человеческих возможностей (физиологических, интеллектуальных и т.',
 'п.)',
 'и достижение физического бессмертия.',
 'В отличие от трансгуманизма, под определением постгуманизма также понимается критика классического гуманизма, подчёркивающая изменение отношения человека к себе, обществу, окружающей среде и бурно развивающимся технологиям, но окончательно разница между транс- и постгуманизмом не определена и остаётся предметом дискуссий.']
```

Даже сегментация строки на предложения очень зависит от языка!

In [101]:
```
# Так выглядит подключение токинизаторов для других языков в NLTK
# (к сожалению в nltk.tokenize нет модуля для русского языка):
spanish_tokenizer = nltk.data.load('tokenizers/punkt/spanish.pickle')
spanish_tokenizer.tokenize('Hola amigo. Estoy bien.')
```

Out[101]:
```
['Hola amigo.', 'Estoy bien.']
```

В каком виде лучше представлять результат сегментации?

Простая модель: список сегментов.

- если применяется несколько способов сегментации?
- как искать в тексте исходное расположение сегмента?

Более общий способ представления результата анализа текстов - **модель аннотаций**.

Аннотация - в общем случае тройка:

- начало
- конец
- значение (не обязательно)

In [64]:
```
%pip install razdel
```

```
Collecting razdel
  Using cached razdel-0.5.0-py3-none-any.whl (21 kB)
Installing collected packages: razdel
Successfully installed razdel-0.5.0
Note: you may need to restart the kernel to use updated packages.
```

In [102]:
```
# разделение на основе правил
# https://github.com/natasha/razdel
from razdel import sentenize
```

In [103]:
```
sent_0 = list(sentenize(lines[0]))
sent_0
```

Out[103]:
```
[Substring(0,
           141,
           'Постгуманизм — рациональное мировоззрение, основанное на представлении, чт
о эволюция человека не завершена и может быть продолжена в будущем.'),
 Substring(142,
           421,
           'Эволюционное развитие должно привести к становлению постчеловека — гипотет
ической стадии эволюции человеческого вида, строение и возможности которого стали бы о
тличными от современных человеческих в результате активного использования передовых те
хнологий преобразования человека.'),
 Substring(422,
           590,
           'Постгуманизм признаёт неотъемлемыми правами совершенствование человеческих
возможностей (физиологических, интеллектуальных и т. п.) и достижение физического бесс
мертия.'),
 Substring(591,
           914,
           'В отличие от трансгуманизма, под определением постгуманизма также понимает
ся критика классического гуманизма, подчёркивающая изменение отношения человека к себ
е, обществу, окружающей среде и бурно развивающимся технологиям, но окончательно разни
ца между транс- и постгуманизмом не определена и остаётся предметом дискуссий.')]
```

In [104]:
```
sent_0[0].text
```

Out[104]:
```
'Постгуманизм — рациональное мировоззрение, основанное на представлении, что эволюция
человека не завершена и может быть продолжена в будущем.'
```

Токенизация

- к оглавлению

Токенизация - разбиение строки на подстроки, которые мы рассматриваем как интересующие нас группы символов (токены).

В NLP под токенизацией обычно понимают разбиение текста на слова, знаки препинания и т.д.

Многозначность определения токена:

- I'm - один токен или два?
- won't - один токен или два?
- т.к. - один токен или два?

Разрешение таких воп зависит от целей токенизации. Или (что хуже) от применяемой библиотеки.

In [105]:
```
# токенизация русскоязычного текста с помощью библиотеки razdel:
from razdel import tokenize
tokens = list(tokenize(lines[0]))
tokens[:5]
```

Out[105]:
```
[Substring(0, 12, 'Постгуманизм'),
 Substring(13, 14, '—'),
 Substring(15, 27, 'рациональное'),
 Substring(28, 41, 'мировоззрение'),
 Substring(41, 42, ',')]
```

In [106]:
```
[_.text for _ in tokens]
```

Out[106]:
```
['Постгуманизм',
 '–',
 'рациональное',
 'мировоззрение',
 ',',
 'основанное',
 'на',
 'представлении',
 ',',
 'что',
 'эволюция',
 'человека',
 'не',
 'завершена',
 'и',
 'может',
 'быть',
 'продолжена',
 'в',
```

Токинизатор tok-tok

Токинизатор tok-tok простой токинизатор общего назначения. Он рассматривает только одно предложение в строке. Таким образом, только последняя точка в предложении рассматривается как токен.

Tok-tok был протестирован и показал приемлемые результаты на следующих языках:

- English
- Persian
- Russian
- Czech
- French
- German
- Vietnamese
- Tajik
- ... и некоторых других.

Tok-tok принимает строку в кодировке UTF-8.

In [107]:
```
from nltk.tokenize.toktok import ToktokTokenizer
toktok = ToktokTokenizer()
toktok.tokenize(sent_0[0].text)
```

Out[107]:
```
['Постгуманизм',
 '_',
 'рациональное',
 'мировоззрение',
 ',',
 'основанное',
 'на',
 'представлении',
 ',',
 'что',
 'эволюция',
 'человека',
 'не',
 'завершена',
 'и',
 'может',
 'быть',
 'продолжена',
 'в',
 'будущем',
 '.']
```

Более сложные задачи сегментации текста:

Языки, где слова не разделяются пробелами:

- """两个月前遭受恐怖袭击的法国巴黎的犹太超市在装修之后周日重新开放，法国内政部长以及超市的管理者都表示，这显示了生命力要比野蛮行为更强大。 该超市1月9日遭受枪手袭击，导致4人死亡，据悉这起事件与法国《查理周刊》杂志社恐怖袭击案有关。 """ -> WordList(['两', '个', '月', '前', '遭受', '恐怖', '袭击', '的', '法国', '巴黎', '的', '犹太', '超市', '在', '装修', '之后', '周日', '重新', '开放', '，', '，', '法国', '内政', '部长', '以及', '超市', '的', '管理者', '都', '表示', '，', '，', '这', '显示', '了', '生命力', '要', '比', '野蛮', '行为', '更', '强大', '。', '，', '该', '超市', '1', '月', '9', '日', '遭受', '枪手', '袭击', '，', '，', '导致', '4', '人', '死亡', '，', '，', '据悉', '这', '起', '事件', '与', '法国', '，', '《', '查理', '周刊', '》', '，', '杂志', '，', '社', '，', '恐怖', '袭击', '，', '案', '，', '有关', '，', '。', ']')
- В немецком языке возможны слова типа Donaudampfschifffahrtskapitän (капитан рейса, выполняемого пароходом по Дунаю), по сути состоящего из слов Dona (Дунай), Dampfschiff (пароход), Fahrt (рейс) и Kapitän (капитан). Если не выполнять разделение таких слов на составляющие слова, то документ не будет найден по запросам, содержащим слова, входящие в "склеенные" слова.

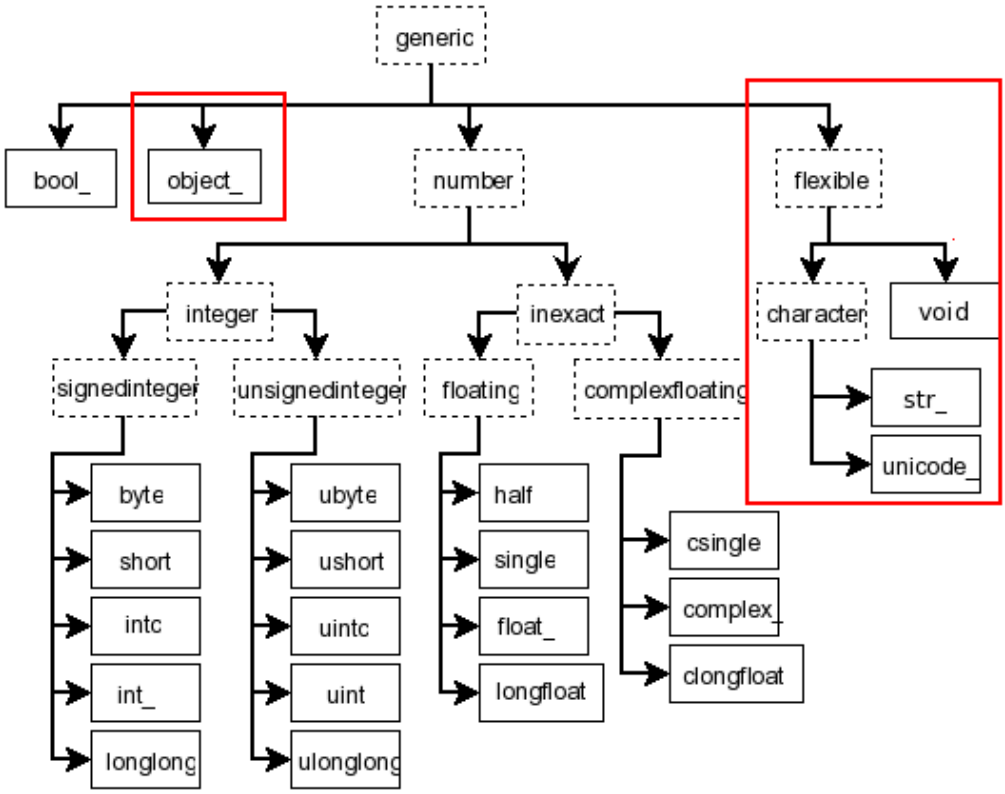
Другие случаи:

- #поставьмнелайк, #делайкакя, #серьгискристалламиростовнадону, ...
- supernaturalflavors.com, babybirthdaygift.com, crosswordmagazines.com, купитьрозы.рф, ...

Работа со строками в numpy

-

- к оглавлению



Иерархия типов объектов, представляющих типы данных массивов Numpy

В numpy для работы со строками предназначены типы:

- `object_`
- наследники абстрактного типа `character` (наследнк `flexible`): `bytes_`, `unicode_`
  - представители типа имеют собственный размер (определяемый при задании максимальной длины строки в массиве)
  - задаются как: `S#` - для `bytes_` (zero-terminated bytes) (аналог bytes, не рекомендован к использованию); `U#` - для `unicode_` (аналок str в Python3), на месте `#` натуральное число.
  - автоматическое определение dtype для массива строк в numpy выполняется по самой длинной строке в массиве.
  - максимальная длина строки `unicode_` не изменяется, попытка присвоить значение длиннее заданного значения приведет к обрезанию строки.

Пример:

In [375]:
```
import numpy as np
from sys import getsizeof
```

In [495]:
```
country = np.array(['USA', 'Japan', 'UK', '', 'India', 'China'])
```

In [377]:
```
country
```

Out[377]:
```
array(['USA', 'Japan', 'UK', '', 'India', 'China'], dtype='<U5')
```

In [378]:
```
country.dtype, country.dtype.itemsize, country.nbytes
```

Out[378]:
```
(dtype('<U5'), 20, 120)
```

In [496]:
```
# Пробуем поменять пустую строку на строку 'New Zealand':
country[country == ''] = 'New Zealand'
print(country) # значение обрезатся по лимиту длины строки:
```

```
['USA' 'Japan' 'UK' 'New Z' 'India' 'China']
```

Способ решения #1: использование типа `objct_`

В случае использования `dtype=object` применяется иной механизм работы с массивом:

- The memory taken by the array now is filled with pointers to Python objects which are being stored elsewhere in memory (much like a Python list is really just a list of pointers to objects, not the objects themselves).
- object arrays behave more like usual Python lists, in the sense that **their contents need not be of the same Python type**.
- The object type is also special because an array containing object_ items does not return an object_ object on item access, but instead **returns the actual object that the array item refers to**.

Преимущества и недостатки:

- эффективно использует память в случае строк с существенно различающейся длинной (не резервирует лишнюю память).
- эффективно использует память в случае большого количества повторяющихся коротких строк (используется ссылка на один и тот же объкт)
- для каждой строки создается новая ссылка, это существенные накладные расходы для коротких строк
- отсутствие поддержки векторизованных строковых операций (vectorized string operations)
- (НЕ изменяемость)??

-

array with dtype=object is different. The memory taken by the array now is filled with pointers to Python objects which are being stored elsewhere in memory (much like a Python list is really just a list of pointers to objects, not the objects themselves).

Arithmetic operators such as * don't work with arrays such as ar1

object arrays behave more like usual Python lists, in the sense that their contents need not be of the same Python type.

The object type is also special because an array containing object_ items does not return an object_ object on item access, but instead returns the actual object that the array item refers to.

While creating the array assign the ‘object’ dtype to it. This lets you have all the behaviors of the python string.

In [497]:
```
country2 = np.array(['USA', 'Japan', 'UK', '', 'India', 'China'], dtype = 'object')
country2
```

Out[497]:
```
array(['USA', 'Japan', 'UK', '', 'India', 'China'], dtype=object)
```

In [498]:
```
country2.dtype, country2.dtype.itemsize, country2.nbytes
```

Out[498]:
```
(dtype('O'), 8, 48)
```

In [499]:
```
country2[country2 == ''] = 'New Zealand'
country2
```

Out[499]:
```
array(['USA', 'Japan', 'UK', 'New Zealand', 'India', 'China'],
      dtype=object)
```

In [382]:
```
country2[3], getsizeof(country2[3])
```

Out[382]:
```
('New Zealand', 60)
```

Способ решения #2:

- Изменяем dtype имеющегося массива с помощью функции `numpy.astype()` так, чтобы новая строка помещалась в заданный максимальный размер строки.

In [500]:
```
# копируем данные с преобразование типа:
country3 = country.astype('U256')
country3
```

Out[500]:
```
array(['USA', 'Japan', 'UK', 'New Z', 'India', 'China'], dtype='<U256')
```

In [501]:
```
country3.dtype, country3.dtype.itemsize, country3.nbytes
```

Out[501]:
```
(dtype('<U256'), 1024, 6144)
```

In [502]:
```
country3[country3 == ''] = 'New Zealand'
country3
```

Out[502]:
```
array(['USA', 'Japan', 'UK', 'New Z', 'India', 'China'], dtype='<U256')
```

Способ решения #3:

Сразу создаем стрку нужного размера.

In [230]:
```
country4 = np.array(['USA', 'Japan', 'UK', '', 'India', 'China'], np.dtype(('U', 256)))
country4
```

Out[230]:
```
array(['USA', 'Japan', 'UK', '', 'India', 'China'], dtype='<U256')
```

In [231]:
```
country4.dtype, country4.dtype.itemsize, country4.nbytes
```

Out[231]:
```
(dtype('<U256'), 1024, 6144)
```

In [232]:
```
country4[country4 == ''] = 'New Zealand'
country4
```

Out[232]:
```
array(['USA', 'Japan', 'UK', 'New Zealand', 'India', 'China'],
      dtype='<U256')
```

Сравниваем решения на датасетах

In [8]:
```
import csv
import pandas as pd
```

In [9]:
```
ds1 = 'COVID.csv' # COVID Tweets
ds2 = 'distinct_users_from_search_table_real_map.csv' # Tweeter Usernames
```

In [10]:
```
df1 = pd.read_csv(ds1)
df1[:3]
```

Out[10]:

	Tweet Id	Tweet URL	Tweet Posted Time (UTC)	Tweet Content	Tweet Type	Client	Re Re
0	"1233417783175778304"	https://twitter.com/Giussi92/status/1233417783...	28 Feb 2020 15:44:49	Also the entire Swiss Football League is on ho...	Tweet	Twitter for iPhone	
1	"1233417742520332290"	https://twitter.com/LAMofficial/status/1233417...	28 Feb 2020 15:44:40	World Health Org Official: Trump's press confe...	Tweet	Twitter Web App	
2	"1233417741027225602"	https://twitter.com/mitchellvii/status/1233417...	28 Feb 2020 15:44:39	I mean, Liberals are cheer-leading this #Coron...	Tweet	Twitter Web App	

3 rows × 22 columns

In [506]:
```
len(df1) # количество строк
```

Out[506]:
```
60160
```

In [507]:
```
dict(df1.dtypes) # типы столбцов
```

Out[507]:
```
{'Tweet Id': dtype('O'),
 'Tweet URL': dtype('O'),
 'Tweet Posted Time (UTC)': dtype('O'),
 'Tweet Content': dtype('O'),
 'Tweet Type': dtype('O'),
 'Client': dtype('O'),
 'Retweets Received': dtype('int64'),
 'Likes Received': dtype('int64'),
 'Tweet Location': dtype('O'),
 'Lat': dtype('float64'),
 'Long': dtype('float64'),
 'Tweet Language': dtype('O'),
 'User Id': dtype('O'),
 'Name': dtype('O'),
 'Screen Name': dtype('O'),
 'User Bio': dtype('O'),
 'Verified or Non-Verified': dtype('O'),
 'Profile URL': dtype('O'),
 'Protected or Non-protected': dtype('O'),
 'User Followers': dtype('int64'),
 'User Following': dtype('int64'),
 'User Account Creation Date': dtype('O')}
```

In [508]:
```
df1['Tweet Content'][:3]
```

Out[508]:
```
0    Also the entire Swiss Football League is on ho...
1    World Health Org Official: Trump’s press confe...
2    I mean, Liberals are cheer-leading this #Coron...
Name: Tweet Content, dtype: object
```

In [509]:
```
for t in df1['Tweet Content'][:3]:
    print(t)
```

```
Also the entire Swiss Football League is on hold. Postponing games from the profession
al and amateur level... #coronavirus https://t.co/UShMuqnAVC (https://t.co/UShMuqnAVC)
World Health Org Official: Trump’s press conference on #coronavirus ‘incoherent’
World Health Organization Special Adviser to the Director Dr. Ezekiel Emanuel says "I
 found most of what [Trump] said incoherent."
https://t.co/v4WIBW9Fld (https://t.co/v4WIBW9Fld)
I mean, Liberals are cheer-leading this #Coronavirus like it's their high school footb
all team.

#TDS
```

In [510]:
```
tweets_txt_obj_arr = np.array(df1['Tweet Content'])
tweets_txt_obj_arr[:3]
```

Out[510]:
```
array(['Also the entire Swiss Football League is on hold. Postponing games from the pr
ofessional and amateur level... #coronavirus https://t.co/UShMuqnAVC', (https://t.co/U
ShMuqnAVC',)
       'World Health Org Official: Trump’s press conference on #coronavirus ‘incoheren
t’\nWorld Health Organization Special Adviser to the Director Dr. Ezekiel Emanuel says
"I found most of what [Trump] said incoherent."\nhttps://t.co/v4WIBW9Fld',
       "I mean, Liberals are cheer-leading this #Coronavirus like it's their high scho
ol football team.\n\n#TDS"],
      dtype=object)
```

In [511]:
```
len(tweets_txt_obj_arr), tweets_txt_obj_arr.dtype # dtype: 'object'
```

Out[511]:
```
(60160, dtype('O'))
```

In [512]:
```
getsizeof(tweets_txt_obj_arr[0]), getsizeof(tweets_txt_obj_arr[1])
```

Out[512]:
```
(195, 544)
```

In [513]:
```
sum(getsizeof(tweets_txt) for tweets_txt in set(tweets_txt_obj_arr)), getsizeof(tweets_t
```

Out[513]:
```
(9572846, 481376)
```

In [514]:
```
max_tweet_len = max(len(s) for s in tweets_txt_obj_arr)
max_tweet_len
```

Out[514]:
```
922
```

In [515]:
```
tweets_txt_u_arr = tweets_txt_obj_arr.astype(np.dtype(('U', max_tweet_len)))
len(tweets_txt_u_arr), tweets_txt_u_arr.dtype, getsizeof(tweets_txt_u_arr)
```

Out[515]:
```
(60160, dtype('<U922'), 221870176)
```

In [516]:
```
# Второй датасет
df2 = pd.read_csv(ds2)
df2[:3]
```

Out[516]:

	user_id	user_screen_name	indegree	outdegree	bad_user_id
0	147240385	Barra_Fake	540	34	119586070
1	97515585	Thamiris1996	229	294	86129642
2	164770319	ReehMuruci	3821	2827	132235657

In [517]:
```
len(df2), dict(df2.dtypes)
```

Out[517]:
```
(736930,
 {'user_id': dtype('int64'),
  'user_screen_name': dtype('O'),
  'indegree': dtype('int64'),
  'outdegree': dtype('int64'),
  'bad_user_id': dtype('int64')})
```

In [518]:
```
tweets_nickname_obj_arr = np.array(df2['user_screen_name'])
tweets_nickname_obj_arr[:3]
```

Out[518]:
```
array(['Barra_Fake', 'Thamiris1996', 'ReehMuruci'], dtype=object)
```

In [519]:
```
len(tweets_nickname_obj_arr), tweets_nickname_obj_arr.dtype
```

Out[519]:
```
(736930, dtype('O'))
```

In [520]:
```
max_nickname_len = max(len(s) for s in tweets_nickname_obj_arr)
max_nickname_len
```

Out[520]:
```
15
```

In [521]:
```
tweets_nickname_u_arr = tweets_nickname_obj_arr.astype(np.dtype(('U', max_nickname_len)))
len(tweets_nickname_u_arr), tweets_nickname_u_arr.dtype
```

Out[521]:
```
(736930, dtype('<U15'))
```

- Модуль char для работы со строками: https://numpy.org/doc/stable/reference/routines.char.html (https://numpy.org/doc/stable/reference/routines.char.html)
  - функции np.char являются тонкими обертками над методами Python string.

- Методы String operations возвращают новый массив.

In [522]:
```
import numba
from numba import jit, njit
```

Датасет 1:

In [523]:
```
%%timeit
np.average(np.char.str_len(tweets_txt_u_arr))
```

```
222 ms ± 1.38 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [524]:
```
print(np.average(np.char.str_len(tweets_txt_u_arr)))
```

```
217.61427859042553
```

In [525]:
```
%%timeit
np.sum(np.char.str_len(tweets_txt_u_arr))/len(tweets_txt_u_arr)
```

```
221 ms ± 1.66 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [526]:
```
print(np.sum(np.char.str_len(tweets_txt_u_arr))/len(tweets_txt_u_arr))
```

```
217.61427859042553
```

In [527]:
```
%%timeit
sum(len(s) for s in tweets_txt_obj_arr) / len(tweets_txt_obj_arr)
```

```
9.61 ms ± 149 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [528]:
```
sum(len(s) for s in tweets_txt_obj_arr) / len(tweets_txt_obj_arr)
```

Out[528]:
```
217.61427859042553
```

In [529]:
```
# работа со строками в питру плохо поддерживается в numba:
@jit
def avg_len(s_arr):
    la = np.char.str_len(tweets_txt_u_arr)
    return np.average(la)
```

Датасет 2:

In [530]:
```
%%timeit
np.average(np.char.str_len(tweets_nickname_u_arr))
```

```
336 ms ± 3.23 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [531]:
```
np.average(np.char.str_len(tweets_nickname_u_arr))
```

Out[531]:
```
10.16212123268153
```

In [532]:
```
%%timeit
sum(len(s) for s in tweets_nickname_obj_arr) / len(tweets_nickname_obj_arr)
```

```
98.2 ms ± 203 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [533]:
```
sum(len(s) for s in tweets_nickname_obj_arr) / len(tweets_nickname_obj_arr)
```

Out[533]:
```
10.16212123268153
```

Эксперимент с более сложным методом:

In [534]:
```
%%timeit
(np.char.find(tweets_txt_u_arr, 'for') > -1).sum()
```

```
280 ms ± 1.43 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [535]:
```
(np.char.find(tweets_txt_u_arr, 'for') > -1).sum()
```

Out[535]:
```
10488
```

In [536]:
```
%%timeit
sum(1 for s in tweets_txt_obj_arr if s.find('for')>-1)
```

```
32.1 ms ± 383 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [537]:
```
sum(1 for s in tweets_txt_obj_arr if s.find('for')>-1)
```

Out[537]:
```
10488
```

In [538]:
```
sum(1 for c in 'abcd' if c > 'b')
```

Out[538]:
```
2
```

In [539]:
```
print(sum(len(s) for s in tweets_nickname_obj_arr) / len(tweets_nickname_obj_arr))
```

```
10.16212123268153
```

Хеширование строк

- к оглавлению

Хеш-функция - выполняет преобразование массива входных данных произвольной длины (ключа, сообщения) в (выходную) битовую строку установленной длины (хеш, хеш-код, хеш-сумму).

Хеш-функции применяются в следующих задачах:

- построение ассоциативных массивов;
- **поиске дубликатов в сериях наборов данных**;
- **построение уникальных идентификаторов для наборов данных**;
- вычислении контрольных сумм от данных (сигнала) для последующего обнаружения в них ошибок (возникших случайно или внесённых намеренно), возникающих при хранении и/или передаче данных;
- сохранении паролей в системах защиты в виде хеш-кода (для восстановления пароля по хеш-коду требуется функция, являющаяся обратной по отношению к использованной хеш-функции);
- выработке электронной подписи (на практике часто подписывается не само сообщение, а его «хеш-образ»);
- ...и многих других.

Для решения различных задач требования к хеш-функциям могут очень существенно отличаться.

"Хорошая" хеш-функция должна удовлетворять двум свойствам:

- быстрое вычисление;
- минимальное количество коллизий.

Кллизией называется ситуация, когда два ключа могут быть хешированы одну и ту же ячейку.

При построении хеш-функции хорошим подходом является подбор функции таким образом, чтобы она никак **не коррелировала с закономерностями**, которым могут подчиняться существующие данные. Например, мы можем потребовать, чтобы **"близкие" в некотором смысле ключи давали далекие хеш-значения** (например, хеш функция для подряд идущих целых чисел давала далекие хеш-значения). В некоторых приложениях хеш-функций требуется противоположное свойство - непрерывность (близкие ключи длолжны порождать близкие хеш-значения).

Обычно от хеш-функций ожидается, что значения хеш-функции находиятся в диапазоне от 0 до $m - 1$. Причём, часто удобно, есчли $m = 2^n$. Таким образом значение хеш-функции может, например, без преобразований хранится в машинном слове.

Использование хэш-функций "в быту"

Если строки используются в качестве уникальных идентификаторов, то они могут быть заменены на хэш-значения. Минус: риск возникновения коллизии.

Проблема **Birthday attack**:

- Парадокс дней рождения: в группе, состоящей из 23 или более человек, вероятность совпадения дней рождения (число и месяц) хотя бы у двух людей превышает 50 %.

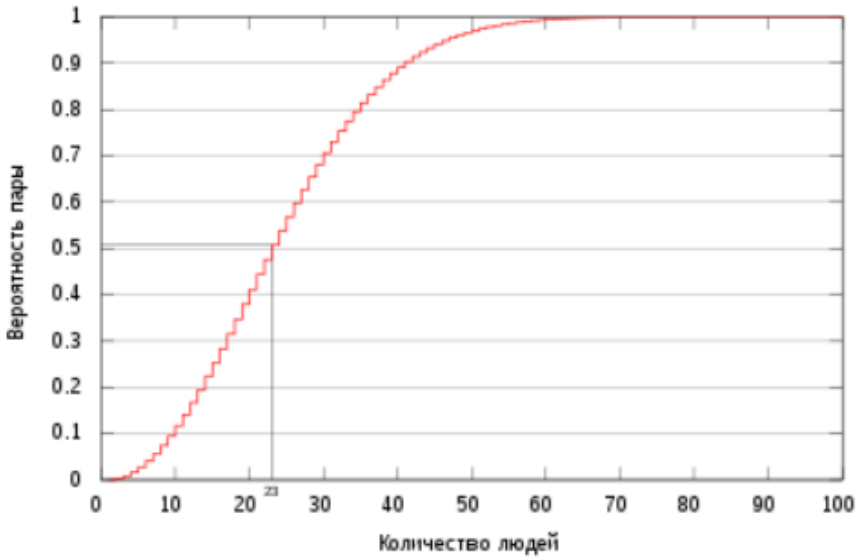


График зависимости вероятности совпадения дней рождения хотя бы у двух человек от количества людей

Биты	Возможные выходы (N)	Желаемая вероятность случайной коллизии (P)									
		10^{-18}	10^{-15}	10^{-12}	10^{-9}	10^{-6}	0,1 %	1 %	25 %	50 %	75 %
16	2^{16} (~6,5 × 10^{3})	<2	<2	<2	<2	<2	11	36	190	300	430
32	2^{32} (~4,3 × 10^{9})	<2	<2	<2	3	93	2900	9300	50 000	77 000	110 000
64	2^{64} (~1.8 × 10^{19})	6	190	6100	190 000	6 100 000	1,9 × 10^{8}	6,1 × 10^{8}	3,3 × 10^{9}	5,1 × 10^{9}	7,2 × 10^{9}
128	2^{128} (~3.4 × 10^{38})	2,6 × 10^{10}	8,2 × 10^{11}	2,6 × 10^{13}	8,2 × 10^{14}	2,6 × 10^{16}	8,3 × 10^{17}	2,6 × 10^{18}	1,4 × 10^{19}	2,2 × 10^{19}	3,1 × 10^{19}
256	2^{256} (~1,2 × 10^{77})	4,8 × 10^{29}	1,5 × 10^{31}	4,8 × 10^{32}	1,5 × 10^{34}	4,8 × 10^{35}	1,5 × 10^{37}	4,8 × 10^{37}	2,6 × 10^{38}	4,0 × 10^{38}	5,7 × 10^{38}
384	2^{384} (~3,9 × 10^{115})	8,9 × 10^{48}	2,8 × 10^{50}	8,9 × 10^{51}	2,8 × 10^{53}	8,9 × 10^{54}	2,8 × 10^{56}	8,9 × 10^{56}	4,8 × 10^{57}	7,4 × 10^{57}	1,0 × 10^{58}
512	2^{512} (~1,3 × 10^{154})	1,6 × 10^{68}	5,2 × 10^{69}	1,6 × 10^{71}	5,2 × 10^{72}	1,6 × 10^{74}	5,2 × 10^{75}	1,6 × 10^{76}	8,8 × 10^{76}	1,4 × 10^{77}	1,9 × 10^{77}

Количество хешей n(P) необходимых для достижения заданной вероятности успеха - коллизии (в предположении, что все хеши одинаково вероятны)

- Для сравнения, от 10^{-18} до 10^{-15} — некорректируемый коэффициент ошибок на бит типичного жесткого диска.

- Теоретически, MD5 хеши или UUID, составляющий 128 бит, должны оставаться в пределах этого диапазона до примерно 820 миллиардов документов, даже если его возможные результаты намного больше.
- максимальное значение int 32 (знакового): 2 147 483 647 (включительно).
- максимальное значение int 64 (знакового): 9 223 372 036' 854 775 807 (включительно).

In [1]:
```
# использовать для контрольной суммы функцию hash() не рекомендуется
# т.к. при повторном запуске скрипта hash() вернет другое значение
hash('Hello world!')
```

Out[1]:
```
6746642367865715353
```

In [2]:
```
import hashlib
```

In [542]:
```
hashlib.algorithms_available
```

Out[542]:
```
{'blake2b',
 'blake2b512',
 'blake2s',
 'blake2s256',
 'md4',
 'md5',
 'md5-sha1',
 'mdc2',
 'ripemd160',
 'sha1',
 'sha224',
 'sha256',
 'sha3-224',
 'sha3-256',
 'sha3-384',
 'sha3-512',
 'sha384',
 'sha3_224',
 'sha3_256',
 'sha3_384',
 'sha3_512',
 'sha512',
 'sha512-224',
 'sha512-256',
 'shake128',
 'shake256',
 'shake_128',
 'shake_256',
 'sm3',
 'whirlpool'}
```

In [543]:
```
hash('Hello world!')
```

Out[543]:
```
6268589131895341199
```

In [3]:
```
hash_object = hashlib.md5(b'Hello World') # строки unicode (str) без перевода в кодировк
hd = hash_object.hexdigest()
hd, type(hd)
```

Out[3]:
```
('b10a8db164e0754105b7a99be72e3fe5', str)
```

In [5]:
```
bd = hash_object.digest()
bd, len(bd), type(bd)
```

Out[5]:
```
(b'\xb1\n\x8d\xb1d\xe0uA\x05\xb7\xa9\x9b\xe7.?\xe5', 16, bytes)
```

In [6]:
```
16*8
```

Out[6]:
```
128
```

xxHash

- xxHash на Python: https://github.com/ifduyue/python-xxhash (https://github.com/ifduyue/python-xxhash)
- xxHash базовая библиотека: https://github.com/Cyan4973/xxHash (https://github.com/Cyan4973/xxHash)
- анализ коллизий: https://github.com/Cyan4973/xxHash/wiki/Collision-ratio-comparison (https://github.com/Cyan4973/xxHash/wiki/Collision-ratio-comparison)

In [2]:
```
%pip install xxhash
```

```
Collecting xxhash
  Downloading xxhash-3.0.0-cp39-cp39-win_amd64.whl (29 kB)
Installing collected packages: xxhash
Successfully installed xxhash-3.0.0
Note: you may need to restart the kernel to use updated packages.
```

In [7]:
```
import xxhash
```

In [4]:
```
h_str = xxhash.xxh64('Hello World').hexdigest()
h_str, type(h_str)
```

Out[4]:
```
('6334d20719245bc2', str)
```

In [8]:
```
h_int = xxhash.xxh64('Hello World').intdigest()
h_int, type(h_int)
```

Out[8]:
```
(7148569436472236994, int)
```

In [9]:
```
# h_int в шестнадцатиричной записи:
print(f'{h_int:x}')
```

```
6334d20719245bc2
```

In [10]:
```
h_byt = xxhash.xxh64('Hello World').digest()
h_byt, len(h_byt), type(h_byt)
```

Out[10]:
```
(b'c4\xd2\x07\x19$[\xc2', 8, bytes)
```

In [433]:
```
xxhash.xxh32('Hello World').intdigest()
```

Out[433]:
```
2986153710
```

In [11]:
```
# для предсказуемого изменения значения хэша можно использовать seed:
xxhash.xxh32('Hello World', seed=1234).intdigest()
```

Out[11]:
```
3249965130
```

In [12]:
```
#128-битный хэш:
xxhash.xxh128('Hello World').intdigest()
```

Out[12]:
```
85338068650070623445356165842316821138
```

In [13]:
```
xxhash.xxh128('Hello World').digest()
```

Out[13]:
```
b'@3\x83\xa1\\\x99\xbe\xee\x9a\xc3\xaf!&\xa0\x02\x92'
```

Выполняем хэширование для больших массивов строк:

In [28]:
```
import csv
import numpy as np
import pandas as pd
from sys import getsizeof
```

In [15]:
```
ds1 = 'COVID.csv' # COVID Tweets
ds2 = 'distinct_users_from_search_table_real_map.csv' # Tweeter Usernames
```

In [16]:
```
df1 = pd.read_csv(ds1)
```

In [17]:
```
df1.head()
```

Out[17]:

	Tweet Id	Tweet URL	Tweet Posted Time (UTC)	Tweet Content	Tweet Type	Cl
0	"1233417783175778304"	https://twitter.com/Giussi92/status/1233417783...	28 Feb 2020 15:44:49	Also the entire Swiss Football League is on ho...	Tweet	Tw iPh
1	"1233417742520332290"	https://twitter.com/LAMofficial/status/1233417...	28 Feb 2020 15:44:40	World Health Org Official: Trump's press confe...	Tweet	Tw
2	"1233417741027225602"	https://twitter.com/mitchellvii/status/1233417...	28 Feb 2020 15:44:39	I mean, Liberals are cheer-leading this #Coron...	Tweet	Tw
3	"1233417699264356357"	https://twitter.com/HelenKennedy/status/123341...	28 Feb 2020 15:44:29	Under repeated questioning, Pompeo refuses to ...	Tweet	Tw iPh
4	"1233417674274807808"	https://twitter.com/W7VOA/status/1233417674274...	28 Feb 2020 15:44:23	#coronavirus comments now from @larry_kudlow h...	Tweet	Tw iPh

5 rows × 22 columns

In [18]:
```
len(df1)
```

Out[18]:
```
60160
```

In [19]:
```
tweets_txt_obj_arr = np.array(df1['Tweet Content'])
```

In [20]:
```
tweets_txt_obj_arr
# tweets_txt_arr
```

Out[20]:
```
array(['Also the entire Swiss Football League is on hold. Postponing games from the pr
ofessional and amateur level... #coronavirus https://t.co/UShMuqnAVC', (https://t.co/U
ShMuqnAVC',)
       'World Health Org Official: Trump’s press conference on #coronavirus ‘incoheren
t’\nWorld Health Organization Special Adviser to the Director Dr. Ezekiel Emanuel says
"I found most of what [Trump] said incoherent."\nhttps://t.co/v4WIBW9Fld',
       "I mean, Liberals are cheer-leading this #Coronavirus like it's their high scho
ol football team.\n\n#TDS",
       ...,
       "It's my party, you're invited!\n\nPS, this is my life philosophy\n\n#Q #DevilS
ticks #TimAndEricDotCom #Matthew #ChinaVirus #WeveBeenHacked https://t.co/KQpLqorNau",
(https://t.co/KQpLqorNau",)
       'Amy’s a survivor! #bariclab #pnnl #movingon #coronavirus #bsl3 #science #grown
ups #professorlife https://t.co/sND6q0r52I', (https://t.co/sND6q0r52I',)
       'A review of asymptomatic and sub-clinical Middle East Respiratory Syndrome #Co
ronavirus Infections https://t.co/aQsUvaBVBp'], (https://t.co/aQsUvaBVBp'],)
      dtype=object)
```

In [21]:
```
uniq_txt = set(tweets_txt_obj_arr)
len(uniq_txt)
```

Out[21]:
```
24943
```

In [29]:
```
len(tweets_txt_obj_arr), sum(getsizeof(tweets_txt) for tweets_txt in uniq_txt), getsizeo
```

Out[29]:
```
(60160, 12234695, 481384)
```

In [22]:
```
v_xxh64 = np.vectorize(lambda s: xxhash.xxh64(s).intdigest())
```

In [23]:
```
tweets_hash_arr = v_xxh64(tweets_txt_obj_arr)
tweets_hash_arr
```

Out[23]:
```
array([17722182809062452388,  17955711301766647O4,  76398654752565154O7,
       ...,  6240978887485535503, 13703536851283103952,
        9858620071029291323], dtype=uint64)
```

In [ ]:
```
get
```

In [26]:
```
# сравниваем хэш для первого значения:
tweets_txt_obj_arr[0], xxhash.xxh64(tweets_txt_obj_arr[0]).intdigest()
```

Out[26]:
```
('Also the entire Swiss Football League is on hold. Postponing games from the professi
onal and amateur level... #coronavirus https://t.co/UShMuqnAVC', (https://t.co/UShMuqn
AVC',)
 17722182809062452388)
```

In [30]:
```
len(tweets_hash_arr), getsizeof(tweets_hash_arr)
```

Out[30]:
```
(60160, 481384)
```

In [31]:
```
len(set(tweets_hash_arr)) # количество уникальных значений совпадает
```

Out[31]:
```
24943
```

Поиск расположения самых частых твитов:

In [32]:
```
%%timeit
unique_t, counts_t= np.unique(tweets_txt_obj_arr, return_counts=True)
```

```
220 ms ± 87.3 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [34]:
```
unique_t, counts_t= np.unique(tweets_txt_obj_arr, return_counts=True)
```

In [35]:
```
tweet_am_idx = np.argmax(counts_t)
unique_t[tweet_am_idx] #, counts[tweet_am_idx]
```

Out[35]:
```
'RT @SaludPublicaEs: El Centro Nacional de Microbiología, del Instituto de Salud Carlo
s III, confirma un caso de #coronavirus detectado en España.\nSe trata del análisis a
una de las muestras enviadas desde La Gomera, Canarias. El paciente está ingresado y a
islado en un centro hospitalario de la isla. https://t.co/Jc8ignBZNO' (https://t.co/Jc
8ignBZNO')
```

In [36]:
```
tweets_txt_obj_arr[1]
```

Out[36]:
```
'World Health Org Official: Trump’s press conference on #coronavirus ‘incoherent’\nWor
ld Health Organization Special Adviser to the Director Dr. Ezekiel Emanuel says "I fou
nd most of what [Trump] said incoherent."\nhttps://t.co/v4WIBW9Fld'
```

In [41]:
```
%%time
# позиции самого частого твита:
tweets_txt_obj_arr == unique_t[tweet_am_idx]
```

```
CPU times: total: 15.6 ms
Wall time: 3.99 ms
```

Out[41]:
```
array([False, False, False, ..., False, False, False])
```

Решение аналогичной задачи с помощью массива хэшей:

In [37]:
```
%%timeit
unique, counts = np.unique(tweets_hash_arr, return_counts=True)
```

```
14.2 ms ± 5.22 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [39]:
```
unique, counts = np.unique(tweets_hash_arr, return_counts=True)
```

In [40]:
```
tweet_am_idx = np.argmax(counts)
unique[tweet_am_idx], counts[tweet_am_idx]
```

Out[40]:
```
(3917492860000152660, 1943)
```

In [42]:
```
%%time
# позиции самого частого твита:
tweets_hash_arr == unique[tweet_am_idx]
```

```
CPU times: total: 0 ns
Wall time: 693 µs
```

Out[42]:
```
array([False, False, False, ..., False, False, False])
```

In [ ]: