Стандартные Библиотеки для упрощения работы с языком python

Библиотека datetime

- **datetime** – это модуль, упрощающий работу со временем и датами.
- Создание объектов класса date

```
from datetime import datetime, date, time, timedelta
import time
date1 = date(2019, 12, 3)
date2 = date.today()
date3 = date.fromtimestamp(time.time())
# date4 = date.fromisoformat("2020-10-25") Доступно с версии 3.7
```

- Возможности представления строк

```
>>> date1.isoformat()
'2019-12-03'
>>> str(date1)
'2019-12-03'
>>> date1.ctime()
'Tue Dec  3 00:00:00 2019'
```

- Операции сравнения

```
>>> date2 == date3
True
>>> date1>date2
False
```

- Объекты **date** возможно вычитать друг из друга, в этом случае возвращается объект **timedelta**.

```
>>> delta = date2 - date1
>>> delta
datetime.timedelta(381)
>>> delta.days
381
```

- С объектами **timedelta** возможно проводить арифметические операции, в таком случае в качестве значения берутся дни.

```
>>> year = timedelta(days=365)
... ten_years = 10 * year
>>> ten_years
datetime.timedelta(3650)
>>> ten_years.days // 365
10
>>> nine_years = ten_years - year
>>> nine_years
datetime.timedelta(3285)
>>> three_years = nine_years // 3
>>> three_years, three_years.days // 365
(datetime.timedelta(1095), 3)
```

Работа с файловых окружением

- Для работы с файлами и директориями можно использовать библиотеку os и модуль **os.path**. Их возможности отличаются в зависимости от сисетемы, в которой они используются.
- По умолчанию текущая директория находится в пути исполняемого файла. Для ее смены применяется функция **os.chdir(path).**
- Таким образом мы получаем возможность обращения к файлам в директории, без написания полного пути.
- Для просмотра текущей директории **os.getcwd**().
- Для просмотра файлов в директории используется функция **os.listdir**(path=".").
- **os.mkdir**(path) - создаёт директорию.
- **os.makedirs**(path) - создаёт директорию, создавая при этом промежуточные директории.
- **os.remove**(path) – удаляет файл.
- **os.rename**(src, dst) - переименовывает файл или директорию из src в dst.

- **os.path.abspath**(path) - возвращает нормализованный абсолютный путь.

```
>>> import os
>>> os.path.abspath('.')
'C:\\python projects\\anything'
>>> os.path.abspath('..')
'C:\\python projects'
```

Заменяет относительный путь текущей директории на абсолютный
А так же возвращает директорию уровнем выше

- **os.path.join**(path1[, path2[, ...]]) - соединяет пути с учётом особенностей операционной системы.

```
>>> os.path.join('python projects', 'anything', 'file')
'python projects\\anything\\file'
```

```
>>> os.path.join('python projects', 'anything', 'file')
'python projects/anything/file'
```

- **os.path.getsize**(path) - размер файла в байтах.
- **os.path.isabs**(path) - является ли путь абсолютным.
- **os.path.isfile**(path) - является ли путь файлом.
- **os.path.isdir**(path) - является ли путь директорией.
- **os.path.dirname**(path) - возвращает имя директории пути path.
- **os.path.samefile**(path1, path2) - указывают ли path1 и path2 на один и тот же файл или директорию.

- Более расширенным инструментарием обладает библиотека **pathlib**.
- Список всех файлов и директорий в пути

```
>>> from pathlib import Path
>>> p = Path('.')
>>> [x for x in p.iterdir()]
[WindowsPath('.idea'), WindowsPath('1.py'), WindowsPath('2.py'), WindowsPath('3.py'), WindowsPath('venv')]
```

- Список файлов с расширением “.py” в пути и поддиректориях

```
>>> list(p.glob('**/*.py'))
[WindowsPath('1.py'), WindowsPath('2.py'), WindowsPath('3.py'), WindowsPath('venv/Lib/site-packages/easy_
```

- Функционал **pathlib** позволяет аналогично **os.path.join()** складывать пути с помощью оператора /

```
>>> q = p / 'venv' / 'lib'
>>> q
WindowsPath('venv/lib')
>>> q.resolve()
WindowsPath('C:/python projects/anything/venv/Lib')
```

- Преобразование к строковому типу возвращает путь в соответствии с

```
>>> str(q.resolve())
'C:\\python projects\\anything\\venv\\Lib'
```

- Путь можно разбить на части методом **parts**

```
>>> q.resolve().parts
('C:\\', 'python projects', 'anything', 'venv', 'Lib')
```

- Вернуть расширение файла

```
>>> python_file = Path('file.py')
>>> python_file.suffix
'.py'
```

- И аналогичное название без расширения

```
>>> python_file.stem
'file'
```

Модуль collections

- Модуль collections - предоставляет специализированные типы данных, на основе словарей, кортежей, множеств, списков.
- **Counter** - вид словаря, который позволяет нам считать количество неизменяемых объектов.

```
>>> import collections
>>> c = collections.Counter()
>>> for word in ['spam', 'egg', 'spam', 'counter', 'counter', 'counter']:
...     c[word] += 1
...
>>> c
Counter({'counter': 3, 'spam': 2, 'egg': 1})
```

- Метод **elements**() - возвращает список элементов в лексикографическом порядке.

```
>>> list(c.elements())
['spam', 'spam', 'egg', 'counter', 'counter', 'counter']
```

- **deque**(iterable) - создаёт очередь из итерируемого объекта. Очереди очень похожи на списки, за исключением того, что добавлять и удалять элементы можно либо справа, либо слева.

```
>>> d = collections.deque([1, 2, 3, 4])
>>> d.appendleft(0)
>>> d.append(5)
>>> d
deque([0, 1, 2, 3, 4, 5])
```

- Метод **rotate(n)** последовательно переносит n элементов из начала в конец (если n отрицательно, то с конца в начало).

```
>>> d
deque([0, 1, 2, 3, 4, 5])
>>> d.rotate(2)
>>> d
deque([4, 5, 0, 1, 2, 3])
>>> d.rotate(-2)
>>> d
deque([0, 1, 2, 3, 4, 5])
```

- Класс **namedtuple** позволяет создать тип данных, ведущий себя как кортеж, с тем дополнением, что каждому элементу присваивается имя, по которому можно в дальнейшем получать доступ.

```
>>> Point = collections.namedtuple('Point', ['x', 'y'])
>>> p = Point(x=1, y=2)
>>> p.x
1
>>> p[0]
1
```

- **defaultdict** ничем не отличается от обычного словаря за исключением того, что по умолчанию всегда вызывается функция, возвращающая значение.

```
>>> defdict = collections.defaultdict(list)
>>> defdict
defaultdict(<class 'list'>, {})
>>> for i in range(5):
...     defdict[i].append(i)
...
>>> defdict
defaultdict(<class 'list'>, {0: [0], 1: [1], 2: [2], 3: [3], 4: [4]})
```

Регулярные выражения

- В Python для работы с регулярными выражениями есть модуль re.
- Чаще всего регулярные выражения используются для:

  поиска в строке;

  разбиения строки на подстроки;

  замены части строки.
- **re.match(*pattern*, *string*)** – ищет pattern в начале string

```
>>> import re
... result = re.match(r'AV', 'AV Analytics Vidhya AV')
>>> result.group()
'AV'
```

- Для вывода начальной и конечной позиции используем методы **.start()** и **.end()**

```
>>> result.start(), result.end()
(0, 2)
```

- **re.search(pattern, string)** – этот метод похож на match(), но он ищет не только в начале строки.

```
>>> result = re.search(r'Analytics', 'AV Analytics Vidhya AV')
>>> result.group()
'Analytics'
```

- **re.findall(*pattern*, *string*)** - возвращает список всех найденных совпадений.

```
>>> result = re.findall(r'AV', 'AV Analytics Vidhya AV')
>>> result
['AV', 'AV']
```

- **re.split(pattern, string, [maxsplit=0]) - разделяет строку по заданному шаблону. Параметр maxsplit указывает сколько разделений будет произведено, если 0, то все возможные разделения.**

```
>>> result = re.split(r'i', 'Analytics Vidhya')
>>> result
['Analyt', 'cs V', 'dhya']
>>> result = re.split(r'i', 'Analytics Vidhya',maxsplit=1)
>>> result
['Analyt', 'cs Vidhya']
```

- **re.sub(pattern, repl, string)** - ищет шаблон в строке и заменяет его на указанную подстроку. Если шаблон не найден, строка остается неизменной.

```
>>> result = re.sub(r'India', 'the World', 'AV is largest Analytics community of India')
>>> result
'AV is largest Analytics community of the World'
```

- **re.compile(pattern, repl, string)** - мы можем собрать регулярное выражение в отдельный объект, который может быть использован для поиска. Это также избавляет от переписывания одного и того же выражения.

```
>>> pattern = re.compile('AV')
>>> result = pattern.findall('AV Analytics Vidhya AV')
>>> result2 = pattern.findall('AV is largest analytics community of India')
>>> result
['AV', 'AV']
>>> result2
['AV']
```

- Специальные символы для работы с re.
- . Один любой символ, кроме новой строки \n.
- ? 0 или 1 вхождение шаблона слева
- + 1 и более вхождений шаблона слева
- * 0 и более вхождений шаблона слева
- \w Любая цифра или буква (\W — все, кроме буквы или цифры)
- \d Любая цифра [0-9] (\D — все, кроме цифры)
- \s Любой пробельный символ (\S — любой непробельный символ)
- \b Граница слова
- [..] Один из символов в скобках ([^..] — любой символ, кроме тех, что в скобках)
- \ Экранирование специальных символов (\. означает точку или \+ — знак «плюс»)
- ^ и $ Начало и конец строки соответственно
- {n,m} От n до m вхождений ({,m} — от 0 до m)
- a|b Соответствует a или b
- () Группирует выражение и возвращает найденный текст
- \t, \n, \r Символ табуляции, новой строки и возврата каретки соответственно

- Разбиение строки на символы (кроме “\n”)

```
>>> result = re.findall(r'.', 'AV is largest Analytics community of India\n')
>>> result
['A', 'V', ' ', 'i', 's', ' ', 'l', 'a', 'r', 'g', 'e', 's', 't', ' ', 'A', 'n', 'a', 'l', 'y', 't', 'i',
```

- Только цифры или буквы

```
>>> result = re.findall(r'\w', 'AV is largest Analytics community of India')
>>> result
['A', 'V', 'i', 's', 'l', 'a', 'r', 'g', 'e', 's', 't', 'A', 'n', 'a', 'l', 'y', 't', 'i', 'c', 's', 'c',
```

- Разбиение по словам

```
>>> result = re.findall(r'\w+', 'AV is largest Analytics community of India')
>>> result
['AV', 'is', 'largest', 'Analytics', 'community', 'of', 'India']
```

- Используя символы ^ и $ получим первое и последнее слово соответственно

```
>>> result = re.findall(r'^\w+', 'AV is largest Analytics community of India')
>>> result
['AV']
>>> result = re.findall(r'\w+$', 'AV is largest Analytics community of India')
>>> result
['India']
```

- Вернем все почтовые домены, для этого напишем выражение

```
>>> result = re.findall(r'@\w+', 'abc.test@gmail.com, xyz@test.in, test.first@analyticsvidhya.com, first.
>>> result
['@gmail', '@test', '@analyticsvidhya', '@rest']
```

- Для того чтобы получить окончания немного преобразуем паттерн

```
>>> result = re.findall(r'@\w+.\w+', 'abc.test@gmail.com, xyz@test.in, test.first@analyticsvidhya.com, fi
>>> result
['@gmail.com', '@test.in', '@analyticsvidhya.com', '@rest.biz']
```

- Извлечем дату, указав количество вхождений цифр

```
>>> result = re.findall(r'\d{2}-\d{2}-\d{4}', 'Amit 34-3456 12-05-2007, XYZ 56-4532 11-11-2011, ABC 67-89
>>> result
['12-05-2007', '11-11-2011', '12-01-2009']
```

- Для вывода только определенного значения используем скобки

```
>>> result = re.findall(r'\d{2}-\d{2}-(\d{4})', 'Amit 34-3456 12-05-2007, XYZ 56-4532 11-11-2011, ABC 67-
>>> result
['2007', '2011', '2009']
```

Модуль itertools

- **Itertools -** модуль предоставляющий функции-итераторы. Он позволяет эффективнее обрабатывать последовательности и облегчает работу с подстановками.
- **itertools.count(start=0, step=1)** - бесконечная арифметическая прогрессия с первым членом start и шагом step.
- **itertools.cycle(iterable)** - возвращает по одному значению из последовательности, повторенной бесконечное число раз. Похоже на очередь deque, но более проста в использовании.
- **itertools.repeat(elem, n=Inf)** - повторяет elem n раз. Похоже на counter, но суть и функционал отличаются.
- **itertools.chain**(*iterables) - возвращает по одному элементу из первого итератора, потом из второго, до тех пор, пока итераторы не кончатся.
- **itertools.filterfalse**(func, iterable) - все элементы, для которых func возвращает ложь.

- **product(iterable) -** декартово произведение

  product('ABCD', repeat=2)

  Результат: AA AB AC AD BA BB BC BD CA CB CC CD DA DB DC DD

- **permutations(iterable)** – все возможные перестановки без повторения

  permutations('ABCD', 2)

  Результат: AB AC AD BA BC BD CA CB CD DA DB DC

- **combinations() -** комбинации без повторяющихся элементов в отсортированном порядке

  combinations('ABCD', 2)

  Результат: AB AC AD BC BD CD

- combinations_with_replacement() – комбинации с повторяющимися элементами

  combinations_with_replacement('ABCD', 2)

  Результат: AA AB AC AD BB BC BD CC CD DD