Лекция №6

Возможности стандартной библиотеки

- Стандартная библиотека
- time – примитивная работа со временем
- datetime – работа с датой и временем
- collections –умные последовательности
- random – неслучайные случайности
- sys – взаимодействие с интерпретатором Python
- os – работа с сервисами операционной системы
- shutil – работа с утилитами командной строки
- subprocess – управление процессами
- ZipFile – работа с архивами
- re – регулярные выражения
- pickle – сериализация в набор байтов
- json – сериализация в JSON формат
- Практика

Разминка

1. Реализовать итератор, который бы “читал” заданный текст по параграфам. Символ параграфа задается отдельно.
2. Написать генератор для построчного чтения файла.

Стандартная библиотека Python

Как известно, в Python огромное количество встроенных "батареек". Умение вовремя и уместно их применить - один из главных навыков хорошего Python программиста.

Мы рассмотрим несколько полезных библиотек и некоторые важные методы.

Все остальное можно получить из документации : https://docs.python.org/3/library/index.html

time

Для примитивной работы со временем существует библиотека time.

```
>>> import time
>>> time.time()
1500844054.9932308
>>> time.timezone # текущий часовой пояс
-10800
>>> time.sleep(5)
```

datetime

Модуль datetime предоставляет классы для управления датами и временем. Главный объект - это datetime, который является абстракцией момента времени. Также может пригодиться объект timedelta для работы с разницей.

```
>>> from datetime import datetime
...
... next_day = datetime(2017, 7, 26)
... curr_time = datetime.now()
... next_day > curr_time
True
>>> diff = next_day - curr_time
>>> diff
datetime.timedelta(2, 425, 515255)
>>> diff.total_seconds()
173225.515255
```

collections

Модуль collections предоставляет специализированные типы данных, на основе словарей, кортежей, множеств, списков.
collections.Counter - вид словаря, который позволяет нам считать количество неизменяемых объектов (в большинстве случаев, строк).

```
>>> import collections
>>> c = collections.Counter()
>>> for word in ['spam', 'egg', 'spam', 'counter', 'counter']:
...     c[word] += 1
...
>>> print(c)
Counter({'counter': 3, 'spam': 2, 'egg': 1})
>>> print(c['counter'])
3
>>> print(c['collections'])
0
```

random

Этот модуль генерирует псевдослучайные числа для нескольких различных распределений. Наиболее используемые функции:

random() - генерирует псевдослучайное число из полуоткрытого диапазона [0.0, 1.0).

randint(start, stop) - генерирует псевдослучайное число из полуоткрытого диапазона [start, stop).

choice(s) - выбирает случайный элемент из последовательности s.

shuffle(s) - перемешивает элементы изменяемой последовательности s на месте.

randrange([start,] stop[, step]) - выдает случайное целое число из диапазона range(start, stop, step).

choice(range(start, stop, step)) – то же, что и randrange, но с созданием объекта.

normalvariate(mu, sigma) - выдает число из последовательности нормально распределенных псевдослучайных чисел. Здесь mu - среднее, sigma - среднеквадратическое отклонение (sigma > 0).

random

```
>>> import random
>>> random.randint(0, 100)
76
>>> a = ["Ivan", "Petr", "Jack"]
>>> random.choice(a)
'Petr'
>>> random.shuffle(a)
>>> print(a)
['Ivan', 'Petr', 'Jack']
>>> random.shuffle(a)
>>> print(a)
['Petr', 'Jack', 'Ivan']
>>> random.randrange(1, 24, 2)
23
```

sys

В этом модуле содержатся функции и константы для взаимодействия с интерпретатором Python. В этом модуле, в том числе, содержатся следующие переменные:

- argv — аргументы командной строки;
- byteorder — порядок байтов платформы, 'little' или 'big‘;
- flags — объект, предоставляющий в виде атрибутов информацию о флагах, данных интерпретатору. Например, sys.flags.debug говорит о режиме отладки;
- maxint — максимальное целое;
- platform — идентификатор платформы, например, 'linux-i386‘;
- stdin, stdout, stderr — стандартные потоки ввода, вывода и вывода ошибок;
- version — строка с версией

sys

Одно из самых частых применений sys – чтение параметров переданных в программу при ее запуске.

```
import sys

def my_program(*args):
    print('Program started with arguments: {}'.format(args))

if __name__ == '__main__':
    print('This is main program')
    my_program(*sys.argv)
```

```
$ python test.py first second
This is main program
Program started with arguments: ('test.py', 'first', 'second')
```

os

Модуль предоставляет функции переносимого интерфейса к основным сервисам операционной системы, определяет некоторые переменные (например, environ - для доступа к переменным окружения).
Модуль os.path служит для манипуляций с путями к файлам в независимом от платформы виде.

```
>>> import os.path
>>> os.path.join("/tmp/1", "temp.file") # конкатенация путей
'/tmp/1/temp.file'
>>> os.path.dirname("/tmp/1/temp.file") # имя каталога по заданному полному пути
'/tmp/1'
>>> os.path.basename("/tmp/1/temp.file") # имя файла по заданному полному пути
'temp.file'
>>> os.path.normpath("/tmp//2/../1/temp.file") # нормализация пути
'/tmp/1/temp.file'
>>> os.path.exists("/tmp/1/temp.file") # существует ли путь?
False
```

shutil

Модуль shutil содержит набор функций высокого уровня для обработки файлов, групп файлов, и папок. В частности, доступные здесь функции позволяют копировать, перемещать и удалять файлы и папки. Часто используется вместе с модулем os.

```
>>> import os
>>> import shutil as sh
>>> os.makedirs('/tmp/test')
>>> test_dir = '/tmp/test'
>>> os.makedirs('/tmp/test', exist_ok=True)
>>> with open(os.path.join(test_dir, 'text.txt'), 'w') as f:
...     f.write('test text')
...
>>> sh.copytree(test_dir, os.path.join(test_dir, 'inner_dir'))
'/tmp/test/inner_dir`

>>> with open(`myfile.txt', 'w') as f:
...     f.write('test text')
...
>>> sh.copy('myfile.txt', 'myfile2.txt')
'myfile2.txt'
```

subprocess

Модуль subprocess отвечает за выполнение следующих действий: порождение новых процессов, соединение с потоками стандартного ввода, стандартного вывода, стандартного вывода сообщений об ошибках и получение кодов возврата от этих процессов.

```
from subprocess import Popen, PIPE

proc = Popen(['ls', '-l', '/tmp'], stdout=PIPE, stderr=PIPE)
proc.wait() # дождаться выполнения
res = proc.communicate() # получить tuple('stdout', 'stderr')
if proc.returncode:
    print(res[1])
print('result:', res[0])
```

ZipFile

В следующем примере в файле archive.zip будет заархивирован файл file.txt, содержащий текст «text in the file». Это ещё один пример использования менеджера контекста.

```
from zipfile import ZipFile

with ZipFile('archive.zip', 'w') as ziparc:
    ziparc.writestr('file.txt', 'text in the file')
```

Чтение архива происходит аналогично. В следующем примере будут напечатаны имена файлов, содержащиеся в архиве:

```
from zipfile import ZipFile

with ZipFile('archive.zip', 'r') as ziparc:
    for fileinfo in ziparc.filelist:
        print(fileinfo.filename
```

re

Регулярное выражение – это специальная последовательность символов, предназначенная для сопоставления и поиска строк или наборов строк, представляющая собой шаблон, задаваемый в соответствии с определенным синтаксисом (более подробно можно прочитать здесь: https://habr.com/post/349860/).

Модуль re обеспечивает полную поддержку Perl-подобных регулярных выражений в Python. Модуль выбрасывает исключение re.error если ошибка происходит при компиляции или использовании регулярных выражений.

Пример задачи, где нужно использовать модуль re. Есть лог-файл, в котором хранятся записи об http запросах к различным серверам, прошедших через данный узел. Для каждого сервера надо вывести количество посещений и самую свежую дату и время. Содержимое лог-файла выглядит так:

```
netops.microsoft.com - - [01/Jul/1995:07:43:07 -0400] "GET /history/gemini/gemini.html HTTP/1.0" 200 2522
mcdiala09.it.luc.edu - - [01/Jul/1995:07:43:08 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
pm2_9.digital.net - - [01/Jul/1995:07:43:08 -0400] "GET /shuttle/sts-71/sts-71-patch-small.gif HTTP/1.0" 200 12054
p1107.pip.dknet.dk - - [01/Jul/1995:07:43:08 -0400] "GET /cgi-bin/imagemap/countdown?333,188 HTTP/1.0" 302 97
netops.microsoft.com - - [01/Jul/1995:07:43:09 -0400] "GET /images/gemini-logo.gif HTTP/1.0" 200 4452
p1107.pip.dknet.dk - - [01/Jul/1995:07:43:10 -0400] "GET /shuttle/countdown/lps/fr.html HTTP/1.0" 200 1879
```

re

re.match(pattern, string, flags=0) – сравнение по шаблону.
Функция re.match возвращает match-объект в случае успеха и None в противном случае. Мы можем использовать функции group(num) или groups() match-объекта, чтоб получить совпавшее выражение.

Параметр	Описание
pattern	Регулярное выражение для сравнения.
string	Строка, в которой осуществляется поиск шаблона с начала строки.
flags	Можно указать различные флаги, используя побитовое ИЛИ (|). Это т.н. модификаторы.

Методы match-объекта	Описание
group(num=0)	Возвращает совпавшее выражение полностью (либо его часть с индексом num).
groups()	Возвращает все кортеж из всех совпавших частей (пустой кортеж, если совпадений не найдено).

re

Шаблон	Описание
^	Начало строки.
$	Конец строки.
.	Любой единичный символ кроме символа перевода строки. Флаг m позволяет включить также и символ новой строки.
[...]	Любой единичный символ в скобках.
[^...]	Любой единичный символ НЕ из указанных в скобках.
re*	0 или больше включений предшествующего выражения.
re+	1 или больше включений предшествующего выражения.
re?	0 или 1 включение предшествующего выражения.
re{ n}	Ровно n включений предшествующего выражения.
re{ n,}	n или больше включений предшествующего выражения.
re{ n, m}	От n до m включений предшествующего выражения.
a| b	Либо a, либо b.
(re)	Группирует регулярные выражения и запоминает найденный текст.
(?imx)	Включает i, m, или x опции для конкретного регулярного выражения. Скобки, если есть, определяют группу, на которую это действует.
(?-imx)	Отключает i, m, или x опции для конкретного регулярного выражения. Скобки, если есть, определяют группу, на которую это действует.
(?: re)	Группирует регулярные выражения, не запоминая найденный текст.
(?imx: re)	Включает i, m, или x опции для конкретного регулярного выражения в скобках.
(?-imx: re)	Отключает i, m, или x опции для конкретного регулярного выражения в скобках.
(?#...)	Комментарий.
(?= re)	Задает позицию, используя шаблон. Не имеет диапазона.

re

Шаблон	Описание
(?! re)	Задает позицию, используя отрицание шаблона. Не имеет диапазона.
(?> re)	Независимый шаблон без предыстории.
\w	Буквенные символы.
\W	Небуквенные символы.
\s	Пробелы. Эквивалентно [\t\n\r\f].
\S	НЕ пробелы.
\d	Цифры. Эквивалентно [0-9].
\D	НЕ цифры.
\A	Начало строки.
\Z	Конец строки, сам символ конца строки не включается.
\z	Конец строки.
\G	Точка, где закончился предыдущий поиск.
\b	Начало или конец слова (слева пусто или не-буква, справа буква и наоборот).
\B	Не граница слова: либо и слева, и справа буквы, либо и слева, и справа НЕ буквы
\n, \t, etc.	Символ перевода строки, возврата каретки, табуляции, и т.д..

re

В функции для работы с регулярными выражениями можно передавать модификаторы для управления аспектами сравнения.
Модификаторы указываются как опциональный флаг, можно указать несколько модификаторов используя знак оператора ИЛИ (|).

Модификатор	Описание
re.I	Осуществляет сравнение без учета регистра.
re.L	Интерпретирует слова в соответствии с текущей локализацией. Такая интерпретация влияет на определение алфавитных групп (\w и \W), а также на определение границ слов (\b и \B).
re.M	Делает символ $ обозначающим конец строки (а не просто конец текста) и символ ^ обозначающим начало строки (а не просто начало текста).
re.S	Делает период (точку) соответствующим любому символу, включая перенос строки.
re.U	Интерпретирует буквы как символы Unicode. Этот флаг влияет на интерпретацию \w, \W, \b, \B.
re.X	Разрешает более изящный синтаксис регулярных выражений. Игнорирует пробелы (если только они не указаны внутри [] или после обратного слеша) и рассматривает # без предваряющего слеша как маркер комментария.

re

```
import re

line = "Cats are smarter than dogs"
matchObj = re.match(r'(.*) are (.*?) .*', line, re.M|re.I)

if matchObj:
    print("matchObj.group(): {}".format(matchObj.group()))
    print("matchObj.group(1): {}".format(matchObj.group(1)))
    print("matchObj.group(2): {}".format(matchObj.group(2)))
else:
    print("No match!")
```

```
matchObj.group(): Cats are smarter than dogs
matchObj.group(1): Cats
matchObj.group(2): smarter
```

re

re.search(pattern, string, flags=0) – поиск по шаблону.

Функция search ищет первое вхождение паттерна RE внутри строки и возвращает match-объект в случае успеха и None в противном случае. Мы можем использовать функции group(num) или groups() match-объекта, чтоб получить совпавшее выражение.

```
import re

line = "Cats are smarter than dogs"
matchObj = re.search(r'(.*) are (.*?) .*', line, re.M|re.I)

if matchObj:
    print("matchObj.group(): {}".format(matchObj.group()))
    print("matchObj.group(1): {}".format(matchObj.group(1)))
    print("matchObj.group(2): {}".format(matchObj.group(2)))
else:
    print("Nothing found!")
```

```
matchObj.group(): Cats are smarter than dogs
matchObj.group(1): Cats
matchObj.group(2): smarter
```

re

В чем разница между re.match и re.search?
re.match проверяет совпадение только от начала строки, тогда как re.search выполняет поиск совпадений по всей строке (как раз то, что Perl делает по умолчанию).

```
import re

line = "Cats are smarter than dogs"

matchObj = re.match(r'dogs', line, re.M|re.I)
if matchObj:
    print("match --> matchObj.group(): }".format(matchObj.group()))
else:
    print("No match!")

searchObj = re.search(r'dogs', line, re.M|re.I)
if searchObj:
    print("search --> searchObj.group(): {}".format(searchObj.group()))
else:
    print("Nothing found!")
```

```
No match!
search --> searchObj.group(): dogs
```

re

re.sub(pattern, repl, string, max=0) – поиск и замена по шаблону.
Одна из самых важных функций re - это sub. Этот метод заменяет либо все включения шаблона RE в строке string строкой, либо не больше max первых включений. Функция возвращает новую строку.

```
import re

phone = "2004-959-559 # This is Phone Number"

# Удаление комментариев
num = re.sub(r'#.*$', '', phone)
print("Phone number: {}".format(num))

# Удаление всех символов кроме цифр
num = re.sub(r'\D', '', phone)
print("Phone number: {}".format(num))
```

```
Phone number: 2004-959-559
Phone number: 2004959559
```

re

Решение для задачи с анализом лог-файла:

```
import os
import datetime
import re

if __name__ == "__main__":
    filepath = "logfile"
    if os.path.isfile(filepath):
        with open(filepath, "r") as f:
            addrs = dict()
            for line in f:
                result = re.search(r'(?P<addr>(.)*) - - \[(?P<time>(.)*)\]', line)
                if result:
                    grdict = result.groupdict()
                    dt = datetime.datetime.strptime(grdict['time'], "%d/%b/%Y:%H:%M:%S -%f")
                    if grdict['addr'] in addrs.keys():
                        addrs[grdict['addr']][0] += 1
                        addrs[grdict['addr']][1] = dt
                    else:
                        addrs[grdict['addr']] = [0, dt]

            for key, value in addrs.iteritems():
                print key, "-", value[0], "-", value[1].strftime("%d/%b/%Y:%H:%M:%S")
```

pickle

Модуль pickle реализует базовый, но эффективный алгоритм для сериализации и десериализации объектов Python. “Pickling” (консервирование) – это процесс конвертирования иерархии объекта в поток байтов, тогда как “unpickling” – это обратная операция – получение из потока байтов иерархии объекта. Pickling (и unpickling) также известны как “сериализация”, “маршаллинг” или “флаттеринг”.

В Python2 существует также аналог модуля pickle – cPickle, написанный на C и потому в 1000 раз более быстрый, чем pickle.

В Python3 модуль pickle уже сделан на основе cPickle.

pickle

```
import pickle

class A:
    def __init__(self, arg):
        self.a = arg
    def __repr__(self):
        return '<A(a={}) at 0x{:x}>'.format(self.a, id(self))

a = A('onetwothree')
print('a = {}'.format(a))

# сериализация
p0 = pickle.dumps(a, protocol=0)
print('serialized p0:\n{}\n'.format(p0))
p1 = pickle.dumps(a, protocol=1)
print('serialized p1:\n{}\n'.format(p1))
p2 = pickle.dumps(a, protocol=2)
print('serialized p2:\n{}\n'.format(p2))
p3 = pickle.dumps(a)
print('serialized def:\n{}\n'.format(p3))
p4 = pickle.dumps(a, protocol=pickle.HIGHEST_PROTOCOL)
print('serialized high:\n{}\n'.format(p4))
# десериализация
print('deserialized def: {}\n'.format(pickle.loads(p3)))
print('deserialized high: {}\n'.format(pickle.loads(p4)))
```

pickle

```
a = <A(a=onetwothree) at 0x216c33ee240>

serialized p0:
b'ccopy_reg\n_reconstructor\np0\n(c__main__\nA\np1\nc__builtin__\nobject\np2\nNtp3\nRp4\n(d
p5\nVa\np6\nVonetwothree\np7\nsb.'

serialized p1:
b'ccopy_reg\n_reconstructor\nq\x00(c__main__\nA\nq\x01c__builtin__\nobject\nq\x02Ntq\x03Rq\
x04}q\x05X\x01\x00\x00\x00aq\x06X\x0b\x00\x00\x00onetwothreeq\x07sb.'

serialized p2:
b'\x80\x02c__main__\nA\nq\x00)\x81q\x01}q\x02X\x01\x00\x00\x00aq\x03X\x0b\x00\x00\x00onetwo
threeq\x04sb.'

serialized def:
b'\x80\x03c__main__\nA\nq\x00)\x81q\x01}q\x02X\x01\x00\x00\x00aq\x03X\x0b\x00\x00\x00onetwo
threeq\x04sb.'

serialized high:
b'\x80\x04\x95+\x00\x00\x00\x00\x00\x00\x00\x8c\x08__main__\x94\x8c\x01A\x94\x93\x94)\x81\x
94}\x94\x8c\x01a\x94\x8c\x0bonetwothree\x94sb.'

deserialized def: <A(a=onetwothree) at 0x216c33ee358>

deserialized high: <A(a=onetwothree) at 0x216c33ee358
```

json

JSON (JavaScript Object Notation) - простой формат обмена данными, основанный на подмножестве синтаксиса JavaScript. Модуль json позволяет кодировать и декодировать данные в удобном формате.

```
import json

my_list = ['foo', {'bar': ('baz', None, 1.0, 2)}]
my_json_str = json.dumps(my_list)
print('my_json_str: '.format(my_json_str))

my_pretty_json_str = json.dumps(my_list, sort_keys=True, indent=4)
print('my_pretty_json_str:\n{}'.format(my_pretty_json_str))

list_from_json = json.loads(my_pretty_json_str)
print('list_from_json: {}'.format(list_from_json))

json.dump(my_list, open('temp.json','w'), sort_keys=True, indent=4)
list_from_json_file = json.load(open('temp.json','r'))
print('list_from_json_file: {}'.format(list_from_json_file))
```

json

```
my_json_str:
my_pretty_json_str:
[
    "foo",
    {
        "bar": [
            "baz",
            null,
            1.0,
            2
        ]
    }
]
list_from_json: ['foo', {'bar': ['baz', None, 1.0, 2]}]
list_from_json_file: ['foo', {'bar': ['baz', None, 1.0, 2]}]
```

НИИТ

Практика

1. Написать функцию для подсчета количества рабочих дней между двумя датами (даты передаются в качестве параметров).
2. С помощью библиотеки subprocess прочитать содержимое произвольного файла с использованием утилиты cat в Linux (имя файла должно передаваться как параметр в вашу функцию).
3. Создать класс Human с 5-10 атрибутами (имя, фамилия, возраст, мето жительства и т.д.). Написать функцию, которая создавала бы указанное количество экземпляров, сериализовывала их и сохраняла в файл human.data, и другую функцию, которая бы читала файл human.data, десериализовывала его содержимое и выводила результат на печать. Примечание: чтоб у экземпляров Human были разные значения атрибутов, можно воспользоваться функциями random.randint() и random.choice().

Практика*

Написать программу, которая уничтожает файлы и папки по истечении заданного времени. Вы указываете при запуске программы путь до директории, за которой нашему скрипту необходимо следить. После запуска программа не должна прекращать работать, пока вы не остановите ее работу с помощью Ctrl+C (подсказка: для постоянной работы программы необходим вечный цикл, например, "while True:", при нажатии Ctrl+C автоматически остановится любая программа). Программа следит за объектами внутри указанной при запуске папки и удаляет их тогда, когда время их существования становится больше одной минуты для файлов и больше двух минуты для папок (то есть дата создания отличается от текущего момента времени больше чем на одну/две минуты). Ваш скрипт должен смотреть вглубь указанной папки. Например, если пользователь создаст внутри нее папку, внутри нее еще одну, а внутри этой какой-то файл, то этот файл должен удалиться первым (так как файлу положено жить только одну минуту, а папкам две). Вам понадобятся библиотеки os и shutil. Внимательно перечитайте задание и учтите возможные ошибки.