**КЫРГЫЗСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И.Раззакова**

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХ

СИСТЕМ»

МЕТОДЫ ПИ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Для студентов специализации 710400

«Программная инженерия»

Бишкек - 2018

РАССМОТРЕНО ОДОБРЕНО

На заседании кафедры Методическим советом ФИТ

«Программное обеспечение

компьютерных систем»

Прот.№13 от 30.05.2018г. Прот. №10 от 20.06.2018г.

УДК 004.43

Составитель – Беккулова К.А.

Методы ПИ: методические указания к выполнению лабораторных работ /Кырг. техн. ун-т, Бишкек, 2018, 64 с.

Представлены задания на разработку программ по дисциплине “Методы ПИ”, приведены общие указания к программированию, приведены задания для лабораторных работ.

Предназначено для студентов специальности 710400 «Программная инженерия» (профиль: Технология разработки программного обеспечения) всех форм обучения.

Библиогр. 4 названий.

Рецензент: доц. Макиева З.Дж.

СОДЕРЖАНИЕ

[ЛАБОРАТОРНАЯ РАБОТА №1 5](#_Toc517779956)

[Общие теоретические сведения 5](#_Toc517779957)

[Задания к самостоятельной работе 5](#_Toc517779958)

[Список использованной литературы 6](#_Toc517779961)

[ЛАБОРАТОРНАЯ РАБОТА №2 7](#_Toc517779962)

[Общие теоретические сведения 7](#_Toc517779963)

[Задания к самостоятельной работе 13](#_Toc517779964)

[Список использованной литературы 13](#_Toc517779965)

[ЛАБОРАТОРНАЯ РАБОТА №3 14](#_Toc517779966)

[Задания к самостоятельной работе 22](#_Toc517779967)

[Список использованной литературы 22](#_Toc517779968)

[ЛАБОРАТОРНАЯ РАБОТА №4 22](#_Toc517779969)

[Общие теоретические сведения 23](#_Toc517779970)

[Задания к самостоятельной работе 31](#_Toc517779971)

[Список использованной литературы 32](#_Toc517779972)

[ЛАБОРАТОРНАЯ РАБОТА №5 33](#_Toc517779973)

[Общие теоретические сведения 33](#_Toc517779974)

[Задания к самостоятельной работе 34](#_Toc517779975)

[Список использованной литературы 34](#_Toc517779976)

[ЛАБОРАТОРНАЯ РАБОТА №6 35](#_Toc517779977)

[Общие теоретические сведения 35](#_Toc517779978)

[Задания к самостоятельной работе 42](#_Toc517780030)

[Список использованной литературы 42](#_Toc517780031)

[ЛАБОРАТОРНАЯ РАБОТА №7 43](#_Toc517780032)

[Тема: Работа с файлами 43](#_Toc517780033)

[Общие теоретические сведения 43](#_Toc517780034)

[Задания к самостоятельной работе 54](#_Toc517780188)

[Список использованной литературы 54](#_Toc517780189)

[ЛАБОРАТОРНАЯ РАБОТА №8 55](#_Toc517780190)

[Общие теоретические сведения 55](#_Toc517780191)

[Задания к самостоятельной работе 62](#_Toc517780316)

[Список использованной литературы 62](#_Toc517780317)

# **ЛАБОРАТОРНАЯ РАБОТА №1**

**Тема: Использование Python**

# **Общие теоретические сведения**

Python-программа, установленная по умолчанию, называется интерпретатором. Интепретатор принимает команды и выполняет их после ввода. Очень удобно для тестирования чего-либо.

Чтобы запустить интерпретатор, просто введи python и нажми Enter.

Чтобы узнать, какая версия Python запущена, используй python -V

Когда Python запустится, мы увидим что-то вроде этого (см.Рис1):

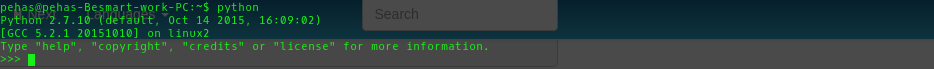


Рис.1

Теперь можем ввести немного Python-кода. Попробуем:

**print**("Hello world")

Нажмем <Enter> и посмотрим, что произошло. После вывода результата Python вернёт нас обратно в интерактивную оболочку, в которой мы можем ввести какую-нибудь другую команду:

**>>> print**("Hello world")  
Hello world  
**>>>** (1 + 4) \* 2  
10

Очень полезна команда help(), которая поможет нам изучить досконально Python, не выходя из интерпретатора. Нажмем q, чтобы закрыть окно со справкой и вернуться в командную строку Python.

Чтобы выйти из интерактивной оболочки, нажмем Ctrl-Z и затем Enter, если Windows, и Ctrl-D, если GNU/Linux или OS X. Этого же можно добиться вводом Python-команды exit().

# **Задания к самостоятельной работе**

### **Задание 1**

Напишите код по следующему словесному алгоритму:

1. Попросить пользователя ввести число от 1 до 9. Полученные данные связать с переменной x.
2. Если пользователь ввел число от 1 до 3 включительно, то …

* попросить пользователя ввести строку. Полученные данные связать с переменной s;
* попросить пользователя ввести число повторов строки. Полученные данные связать с переменной n, предварительно преобразовав их в целочисленный тип;
* выполнить цикл повторения строки n раз;
* вывести результат работы цикла.

1. Если пользователь ввел число от 4 до 6 включительно, то …

* попросить пользователя ввести степень, в которую следует возвести число. Полученные данные связать с переменной m;
* реализовать возведение числа x в степень m;
* вывести полученный результат.

1. Если пользователь ввел число от 7 до 9, то выполнить увеличения числа x на единицу в цикле 10 раз, при этом на экран вывести все 10 чисел.
2. Во всех остальных случаях выводить надпись "Ошибка ввода".

### **Задание 2**

Напишите программу, которая бы выполняла следующие задачи:

1. выводила название программы "Общество в начале XXI века";
2. запрашивала у пользователя его возраст;
3. если пользователь вводит числа от 0 до 7, то программа выводила надпись "Вам в детский сад";
4. от 7 до 18 - "Вам в школу";
5. от 18 до 25 - "Вам в профессиональное учебное заведение";
6. от 25 до 60 - "Вам на работу";
7. от 60 до 120 – "Вам предоставляется выбор";
8. меньше 0 и больше 120 – пятикратный вывод надписи "Ошибка! Это программа для людей!"

В программе желательно использовать все "атрибуты" структурного программирования: ветвление и цикл.

# **Список использованной литературы**

<https://ru.wikibooks.org/wiki/Python/%D0%A3%D1%87%D0%B5%D0%B1%D0%BD%D0%B8%D0%BA_Python_2.6>

* <https://docs.python.org/3.0/whatsnew/3.0.html>
* <http://pythonworld.ru/samouchitel-python>
* <https://www.python.org/downloads/>

Разница между python 2 и python 3

* <http://pythonworld.ru/osnovy/python2-vs-python3-razlichiya-sintaksisa.html>

Pep8,

* <http://pep8online.com/>
* <https://github.com/PyCQA/pep8>
* <http://pep8.ru/doc/pep8/>

tools,

* <http://mashable.com/2007/10/01/python-toolbox/#aQcj.kct1OqA>
* <https://wiki.python.org/moin/DevelopmentTools>

IDE

* <http://pedrokroger.net/choosing-best-python-ide/>
* <https://wiki.python.org/moin/IntegratedDevelopmentEnvironments>

консоль

питон для виндовс

* <http://www.howtogeek.com/197947/how-to-install-python-on-windows/>
* <http://docs.python-guide.org/en/latest/starting/install/win/>
* <https://www.python.org/downloads/windows/>
* <https://docs.python.org/2/using/windows.html>
* <http://xn--80afqpaigicolm.xn--p1ai/python/ustanovka-python-na-windows-7/>
* <http://codeinlife.ru/python/python-nachalo-kak-ustanovit-python-v-windows-i-linux.html>

консолька для питона

* <http://doc.pyschools.com/console>

типы данных

* <http://habrahabr.ru/post/49671/>
* <http://habrahabr.ru/post/30092/>
* <http://pythonworld.ru/tipy-dannyx-v-python>

операторы

* <http://wombat.org.ua/AByteOfPython/operators_and_expressions.html>

синтаксис

* <http://pythonicway.com/python-basic-syntax>
* <http://pythonworld.ru/osnovy/sintaksis-yazyka-python.html>

# **ЛАБОРАТОРНАЯ РАБОТА №2**

**Тема: Работа со строками**

# **Общие теоретические сведения**

**Литералы строк**

Работа со строками в Python очень удобна. Существует несколько литералов строк.

**Строки в апострофах и в кавычках**

S = 'spam"s'  
S = "spam's"

Строки в апострофах и в кавычках - одно и то же. Причина наличия двух вариантов в том, чтобы позволить вставлять в литералы строк символы кавычек или апострофов, не используя экранирование.

**Экранированные последовательности - служебные символы**

Экранированные последовательности позволяют вставить символы, которые сложно ввести с клавиатуры.

|  |  |
| --- | --- |
| **Экранированная последовательность** | **Назначение** |
| \n | Перевод строки |
| \a | Звонок |
| \b | Забой |
| \f | Перевод страницы |
| \r | Возврат каретки |
| \t | Горизонтальная табуляция |
| \v | Вертикальная табуляция |
| \N{id} | Идентификатор ID базы данных Юникода |
| \uhhhh | 16-битовый символ Юникода в 16-ричном представлении |
| \Uhhhh… | 32-битовый символ Юникода в 32-ричном представлении |
| \xhh | 16-ричное значение символа |
| \ooo | 8-ричное значение символа |
| \0 | Символ Null (не является признаком конца строки) |

**"Сырые" строки - подавляют экранирование**

Если перед открывающей кавычкой стоит символ 'r' (в любом регистре), то механизм экранирования отключается.

S = r'C:\newt.txt'

Но, несмотря на назначение, "сырая" строка не может заканчиваться символом обратного слэша. Пути решения:

S = r'\n\n\\'[:-1]  
S = r'\n\n' + '\\'  
S = '\\n\\n'

**Строки в тройных апострофах или кавычках**

Главное достоинство строк в тройных кавычках в том, что их можно использовать для записи многострочных блоков текста. Внутри такой строки возможно присутствие кавычек и апострофов, главное, чтобы не было трех кавычек подряд.

>>> c = '''это очень большая  
... строка, многострочный  
... блок текста'''  
>>> c  
'это очень большая\nстрока, многострочный\nблок текста'  
>>> print(c)  
это очень большая  
строка, многострочный  
блок текста

**Строки (Базовые операции)**

**Конкатенация (сложение)**

>>>

>>> S1 = 'spam'  
>>> S2 = 'eggs'  
>>> print(S1 + S2)  
'spameggs'

**Дублирование строки**

>>>

>>> print('spam' \* 3)  
spamspamspam

**Длина строки (функция len)**

>>>

>>> len('spam')  
4

**Доступ по индексу**

>>>

>>> S = 'spam'  
>>> S[0]  
's'  
>>> S[2]  
'a'  
>>> S[-2]  
'a'

Как видно из примера, в Python возможен и доступ по отрицательному индексу, при этом отсчет идет от конца строки.

**Извлечение среза**

Оператор извлечения среза: [X:Y]. X – начало среза, а Y – окончание;

символ с номером Y в срез не входит. По умолчанию первый индекс равен 0, а второй - длине строки.

>>>

>>> s = 'spameggs'  
>>> s[3:5]  
'me'  
>>> s[2:-2]  
'ameg'  
>>> s[:6]  
'spameg'  
>>> s[1:]  
'pameggs'  
>>> s[:]  
'spameggs'

Кроме того, можно задать шаг, с которым нужно извлекать срез.

>>>

>>> s[::-1]  
'sggemaps'  
>>> s[3:5:-1]  
''  
>>> s[2::2]  
'aeg'

**Другие функции и методы строк**

При вызове методов необходимо помнить, что строки в Python относятся к категории неизменяемых последовательностей, то есть все функции и методы могут лишь создавать новую строку.

>>>

>>> s = 'spam'  
>>> s[1] = 'b'  
Traceback (most recent call last):  
 File "", line 1, in  
 s[1] = 'b'  
TypeError: 'str' object does not support item assignment  
>>> s = s[0] + 'b' + s[2:]  
>>> s  
'sbam'

Поэтому все строковые методы возвращают новую строку, которую потом следует присвоить переменной.

**Форматирование строк**

Форматирование в питоне – мощный инструмент управления строками. Есть несколько подходов – стандартный и с использованием шаблонов. Для форматирования в питоновских строках используется стандартный оператор – символ %. Слева от процента указываем строку, справа – значение или список значений:

>>> s = 'Hello %s' % 'word'  
>>> s  
'Hello word'  
  
>>> s = 'one %s %s' % ('two','three')  
>>> s  
'one two three'

Если нужно преобразование числа в строку, используется числовой спецификатор – %d или %f:

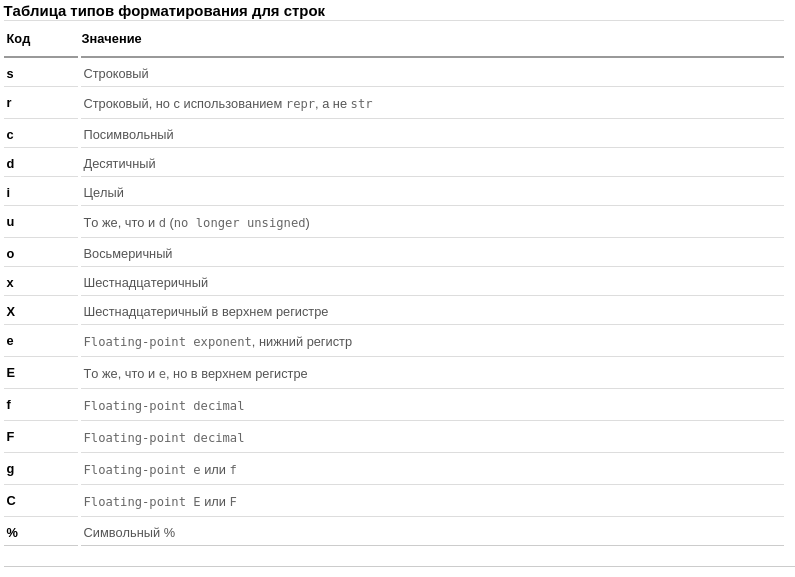
>>> s = 'one %d %f' % (2 , 3.5)  
>>> s  
'one 2 3.500000'

При форматировании можно указать общую ширину строки и точность для чисел, при этом число будет дополнено незначащими нулями. В следующем примере результирующая строка будет иметь длину 10 символов, на дробную часть будет отведено 5 символов:

>>> x = 4/3  
>>> '%10.5f' % x  
' 1.00000'

Пробелы слева можно отформатировать нулями:

>>> from math import pi  
>>> '%015.10f' % pi  
'0003.1415926536'



**Методы работы со строками**

Строки обладают большим набором разнообразных методов. Наиболее популярные из них:

find – находит подстроку в строке – возвращает позицию вхождения строки, либо -1:

>>> s = 'The find method finds a substring'  
>>> s.find('find')  
4  
>>> s.find('finds')  
16  
>>> s.find('findsa')  
-1

join – объединяет через разделитель набор строк:

>>> seq = ['one','two','three']  
>>> sep = ','  
>>> sep.join(seq)  
'one,two,three'

split – это обратная функция для join, разбивает строку на последовательность:

>>> s = '/usr/local/bin'  
>>> s.split('/')  
['', 'usr', 'local', 'bin']

replace – заменяет в строке одну подстроку на другую:

>>> s = 'replace method returns a string'  
>>> s.replace('returns','return')  
'replace method return a string'

strip – удаляет пробелы слева и справа:

>>> ' this is whitespace string '.strip()  
'this is whitespace string'

translate – в отличие от replace, может делать множественную замену. В следующем примере каждый символ '1' в исходной строке будет заменен на символ '3', а символ '2' – на символ '4' соответственно:

>>> from string import maketrans  
>>> table = maketrans('12', '34')  
>>> '1212 5656'.translate(table)  
'3434 5656'

# **Задания к самостоятельной работе**

1. Написать программу поиска самого длинного слова в строке, разделенной пробелами.
2. Написать программу поиска самого длинного слова в строке, разделенной точкой запятой.
3. Написать программу самого короткого слова который выделяется знаком который пользователь вводит в интерактивном режиме
4. Написать программу которое находит введенное слово в строке которое также вводится пользователем в интерактивном режиме
5. Посчитать количество слов в предложении которое вводит пользователь
6. Зарегистрироваться на ресурсе GITHUB
7. Создания гит репозитория для перового и второго задания
8. С клонировать проекты в репозиторий

# **Список использованной литературы**

<http://python-3.ru/page/problemy-s-kodirovkoj-v-python> - если есть проблема с кодировкой

* Работа со строками
  + <http://pythonworld.ru/tipy-dannyx-v-python/stroki-funkcii-i-metody-strok.html>
  + <http://pythonworld.ru/osnovy/formatirovanie-strok-metod-format.html>
  + <http://pythonworld.ru/tipy-dannyx-v-python/stroki-literaly-strok.html>
  + <http://pythonblogg.blogspot.com/2011/03/5.html>
* запуск скрипта
  + <http://pep8.ru/doc/tutorial-2.6/3.html>
* начало работы с гитом
  + <https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git>
  + <https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-Git>
  + <http://habrahabr.ru/post/125799/>
  + <http://eax.me/git-commands/>
  + <https://www.youtube.com/watch?v=Lgs2Ph0Svoo> - видосик по установки и настройки для виндовс
  + <http://habrahabr.ru/post/199144/>
  + <https://raw.githubusercontent.com/github/gitignore/master/Python.gitignore>

# **ЛАБОРАТОРНАЯ РАБОТА №3**

**Тема: Изучение функции, генераторы списков, структура модулей**

**Общие теоретические сведения**

**Функции** (**Именные функции, инструкция def**)

Функция в python - объект, принимающий аргументы и возвращающий значение Обычно функция определяется с помощью инструкции def.

Функции в Пайтоне объявляются не просто, а очень просто. Вот пример самой простой:

def empty\_func():  
 pass

Еще пример, но уже с параметрами

def add(x, y):  
 return x + y

Инструкция **return** говорит, что нужно вернуть значение В нашем случае функция возвращает сумму x и y.

Функция не заканчивающееся инструкцией return вернет значение None:

**Именные функции, инструкция def**

Функция может быть любой сложности и возвращать любые объекты (списки, кортежи, и даже функции!):

>>>

>>> def newfunc(n):  
... def myfunc(x):  
... return x + n  
... return myfunc  
...  
>>> new = newfunc(100) *# new - это функция*  
>>> new(200)  
300

**Аргументы функции**

Функция может принимать произвольное количество аргументов или не принимать их вовсе. Также распространены функции с произвольным числом аргументов, функции с позиционными и именованными аргументами, обязательными и необязательными.

>>> def func(a, b, c=2): *# c - необязательный аргумент*  
... return a + b + c  
...  
>>> func(1, 2) *# a = 1, b = 2, c = 2 (по умолчанию)*  
5  
>>> func(1, 2, 3) *# a = 1, b = 2, c = 3*  
6  
>>> func(a=1, b=3) *# a = 1, b = 3, c = 2*  
6  
>>> func(a=3, c=6) *# a = 3, c = 6, b не определен*  
Traceback (most recent call last):  
 File "", line 1, in  
 func(a=3, c=6)  
TypeError: func() takes at least 2 arguments (2 given)

Функция также может принимать переменное количество позиционных аргументов, тогда перед именем ставится \*:

>>>

>>> def func(\*args):  
... return args  
...  
>>> func(1, 2, 3, 'abc')  
(1, 2, 3, 'abc')  
>>> func()  
()  
>>> func(1)  
(1,)

Как видно из примера, args - это кортеж из всех переданных аргументов функции, и с переменной можно работать также, как и с кортежем.

Функция может принимать и произвольное число именованных аргументов, тогда перед именем ставится \*\*:

>>>

>>> def func(\*\*kwargs):  
... return kwargs  
...  
>>> func(a=1, b=2, c=3)  
{'a': 1, 'c': 3, 'b': 2}  
>>> func()  
{}  
>>> func(a='python')  
{'a': 'python'}

В переменной kwargs у нас хранится словарь, с которым мы, опять-таки, можем делать все, что нам заблагорассудится.

**Анонимные функции, инструкция lambda**

Анонимные функции могут содержать лишь одно выражение, но и выполняются они быстрее. Анонимные функции создаются с помощью инструкции lambda. Кроме этого, их не обязательно присваивать переменной, как делали мы инструкцией def func():

>>>

>>> func = lambda x, y: x + y  
>>> func(1, 2)  
3  
>>> func('a', 'b')  
'ab'  
>>> (lambda x, y: x + y)(1, 2)  
3  
>>> (lambda x, y: x + y)('a', 'b')  
'ab'

lambda функции, в отличие от обычной, не требуется инструкция return, а в остальном, ведет себя точно так же:

>>>

>>> func = lambda \*args: args  
>>> func(1, 2, 3, 4)  
(1, 2, 3, 4)

**Функции со строкой документации**

def safe\_div(x, y):  
 """Do a safe division :-)"""  
 if y != 0:  
 z = x / y  
 print z  
 return z  
 else:  
 print "str"

В этом примере есть несколько нововведений. первое, что бросается в глаза — это строка документации (docstring), идущая сразу после тела функции.

Получить к ней доступ можно и из самой программы, используя свойство \_\_doc\_\_:

print safe\_div.\_\_doc\_\_

**Локальные переменные**

При объявлении переменных внутри определения функции, они никоим образом не связаны с другими переменными с таким же именем за пределами функции – т.е. имена переменных являются *локальными* в функции. Это называется *областью видимости* переменной. Область видимости всех переменных ограничена блоком, в котором они объявлены, начиная с точки объявления имени.

x = 50  
  
def func(x):  
 print('x равен', x)  
 x = 2  
 print('Замена локального x на', x)  
  
func(x)  
print('x по-прежнему', x)

**Зарезервированное слово “global”**

Чтобы присвоить некоторое значение переменной, определённой на высшем уровне программы (т.е. не в какой-либо области видимости, как то функции или классы), необходимо указать Python, что её имя не локально, а *глобально* (*global*). Сделаем это при помощи зарезервированного слова global. Без применения зарезервированного слова global невозможно присвоить значение переменной, определённой за пределами функции.

Можно использовать уже существующие значения переменных, определённых за пределами функции (при условии, что внутри функции не было объявлено переменной с таким же именем). Однако, это не приветствуется, и его следует избегать, поскольку человеку, читающему текст программы, будет непонятно, где находится объявление переменной. Использование зарезервированного слова global достаточно ясно показывает, что переменная объявлена в самом внешнем блоке.

x = 50  
  
def func():  
 global x  
  
 print('x равно', x)  
 x = 2  
 print('Заменяем глобальное значение x на', x)  
  
func()  
print('Значение x составляет', x)

Мы увидели, как получать доступ к переменным в локальной и глобальной области видимости. Есть ещё один тип области видимости, называемый “нелокальной” (nonlocal) областью видимости, который представляет собой нечто среднее между первыми двумя. Нелокальные области видимости встречаются, когда вы определяете функции внутри функций.

Поскольку в Python всё является выполнимым кодом, вы можете определять функции где угодно.

def func\_outer():  
 x = 2  
 print('x равно', x)  
  
 def func\_inner():  
 nonlocal x  
 x = 5  
  
 func\_inner()  
 print('Локальное x сменилось на', x)  
  
func\_outer()

**Генераторы списков**

Генерация списков (list comprehensions) — яркий пример «синтаксического сахара». То есть конструкции, без которой легко можно обойтись, но с ней намного лучше. Генераторы списков, как это не странно, предназначены для удобной обработки списков, к которой можно отнести и создание новых списков, и модификацию существующих.

Допустим, нам необходимо получить список нечетных чисел, не превышающих 25.

>>> res = []  
>>> for x in range(1, 25, 2):  
... res.append(x)  
...  
>>> print res

В общем-то, полученный результат — целиком нас устраивает всем, кроме длинной записи. тут-то на помощь и придет наш «сахарок». В самом простом виде, он обычно

>>> res = [x for x in range(1, 25, 2)]  
>>> print res  
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23]

Рассмотрим такой пример:

>>> a = [2,-2,4,-4,7,5]  
>>> b = [i\*\*2 **for** i **in** a]  
>>> b  
[4, 4, 16, 16, 49, 25]

В данном случае в генераторе списка берется каждый элемент из списка a и возводится в квадрат. Таким образом, 1) что делаем - возводим элемент в квадрат, 2) что берем - элемент, 3) откуда берем - из списка *a*.

В генераторы списков в конце можно добавлять конструкцию if. Например, надо из строки извлечь все цифры:

>>> a = "lsj94ksd231 9"  
>>> b = [int(i) **for** i **in** a **if** '0'<=i<='9']  
>>> b  
[9, 4, 2, 3, 1, 9]

Или заполнить список числами, кратными 30 или 31:

>>> a = [i **for** i **in** range(30,250) **if** i%30 == 0 **or** i%31 == 0]  
>>> a  
[30, 31, 60, 62, 90, 93, 120, 124, 150, 155, 180, 186, 210, 217, 240, 248]

Таким образом, генераторы списков позволяют создавать списки легче и быстрее. Однако заменить ими достаточно сложные конструкции не получится. Например, когда условие проверки должно включать ветку else.

**Как создать модуль на Python**

Мы начнём с создания очень простого модуля. Этот модуль предоставит нам доступ к простейшим арифметическим операциям без какой-либо обработки ошибок. Вот наш первый пример:

*#----------------------------------------------------------------------*  
def add(x, y):  
 """"""  
 return x + y  
   
*#----------------------------------------------------------------------*  
def division(x, y):  
 """"""  
 return x / y  
   
*#----------------------------------------------------------------------*  
def multiply(x, y):  
 """"""  
 return x \* y  
   
*#----------------------------------------------------------------------*  
def subtract(x, y):  
 """"""  
 return x - y

Вы сохранили этот код, Вы получили модуль. Давайте назовём его arithmetic.py. Что теперь с ним можно делать? Вы можете импортировать его и использовать любую из определённых в нём функций. Более того, мы можем даже сделать его исполняемым!

**Как создать пакет для Python**

Основная разница между модулем и пакетом в том, что пакет - это коллекция модулей И он содержит файл \_\_init\_\_.py. В зависимости от сложности пакета, может быть даже не один файл \_\_init\_\_.py. Давайте посмотрим на структуру папки пакета, чтобы это стало чуть более понятно, а затем уже будем писать соответствующий код:

myMath/  
 \_\_init\_\_.py  
 adv/  
 \_\_init\_\_.py  
 sqrt.py  
 fib.py  
 add.py  
 subtract.py  
 multiply.py  
 divide.py

Теперь нам остаётся только воспроизвести эту структуру на практике. Создадим каждый из этих файлов в соответствующей папке с соответствующим именем. Для файлов add, subtract, multiply и divide Вы можете использовать те функции, которые мы уже создали.

Оба файла \_\_init\_\_.py можно оставить пустыми, но в таком случае Вам придётся при импортировании писать что-то вроде **mymath.add.add(x,y),** что, несомненно, неудобно, поэтому мы добавим во внешний \_\_init\_\_.py такой код:

*# outer \_\_init\_\_.py*  
**from** add **import** add  
**from** divide **import** division  
**from** multiply **import** multiply  
**from** subtract **import** subtract  
**from** adv.fib **import** fibonacci  
**from** adv.sqrt **import** squareroot

Теперь, для того, чтобы иметь возможность импортировать этот модуль, мы должны добавить его в наш Python path. Вы можете просто скопировать эту папку в **site-packages**. В Windows она обычно находится по адресу **C:\Python26\Lib\site-packages**. Или же, Вы можете в самом скрипте, где Вы хотите использовать ваш пакет, просто изменить путь поиска модулей:

**import** sys  
   
sys.path.append('C:**\U**sers**\m**driscoll**\D**ocuments')  
   
**import** mymath  
   
**print** mymath.add(4,5)  
**print** mymath.division(4, 2)  
**print** mymath.multiply(10, 5)  
**print** mymath.fibonacci(8)  
**print** mymath.squareroot(48)

Обратите внимание, что саму папку **mymath** я не добавляю в путь. Вам надо добавить папку, содержащую папку с модулем, а не саму папку с модулем.

# **Задания к самостоятельной работе**

1. Переписать первое задание использую генераторы списков
2. Переписать задание из первого урока переведя все проверки в функции
3. Напишите функцию, которая разбивает введённую строку на слова и выводит по очереди само слова тире ее длина.
4. Написать функцию в которую можно передать сколько угодно значений и оно возводит каждое последующее число в степень предыдущего (первое значение возводим в степень один)
5. Переписать первое задание разбив его на модули
6. Собрать все модули в проект

# **Список использованной литературы**

* Функции
  + <http://habrahabr.ru/post/30633/>
  + <http://habrahabr.ru/post/50381/>
  + <http://pythonworld.ru/tipy-dannyx-v-python/vse-o-funkciyax-i-ix-argumentax.html>
  + <http://wombat.org.ua/AByteOfPython/functions.html>
  + <http://younglinux.info/python/function.php>
  + <http://younglinux.info/python/function.php>
* генераторы списков
  + <http://habrahabr.ru/post/30232/>
  + <http://younglinux.info/python/feature/generators>
  + <http://informatics.mccme.ru/mod/book/view.php?id=4655&chapterid=483>
  + <http://pythonworld.ru/tipy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html>
  + <http://pythoner.name/list-generator>
  + <http://pythontutor.ru/lessons/lists/>
* структура модулей
  + <http://python-lab.blogspot.com/2012/07/python-201.html>
  + <http://asvetlov.blogspot.com/2010/05/blog-post.html>
  + <http://pep8.ru/doc/tutorial-3.1/6.html>

# **ЛАБОРАТОРНАЯ РАБОТА №4**

**Тема: Встроенные библиотеки сторонние библиотеки виртуальное окружение**

# **Общие теоретические сведения**

**Встроенные библиотеки**

Большим достоинством Python как инструмента программиста является стандартная библиотека, входящая в дистрибутив. Первое, с чем сталкивается программист - это, конечно, встроенные типы данных, встроенные исключения, и встроенные функции. Список их невелик, и программисты, скажем, перешедшие с Перла, привыкшие, что у них сразу есть функции для манипуляций со строками, с датой и временем, регулярные выражения - спрашивают "да есть у вас в Питоне вообще хоть что-нибудь?" Все есть, только не в списке встроенных функций, а в модулях стандартной библиотеки. Манипуляции со строками делаются с помощью модуля string, дата и время приходят из модуля time, регулярные выражения в модуле re, доступ к функциям операционной системы - модуль os, и так далее.

**Встроенные функции**

**bool(x)** - преобразование к типу bool, использующая стандартную процедуру проверки истинности. Если х является ложным или опущен, возвращает значение False, в противном случае она возвращает True.

**bytearray([источник [, кодировка [ошибки]]])** - преобразование к bytearray. Bytearray - изменяемая последовательность целых чисел в диапазоне 0≤X<256. Вызванная без аргументов, возвращает пустой массив байт.

**bytes([источник [, кодировка [ошибки]]])** - возвращает объект типа bytes, который является неизменяемой последовательностью целых чисел в диапазоне 0≤X<256. Аргументы конструктора интерпретируются как для bytearray().

**complex([real[, imag]])** - преобразование к комплексному числу.

**dict([object])** - преобразование к словарю.

**float([X])** - преобразование к числу с плавающей точкой. Если аргумент не указан, возвращается 0.0.

**frozenset([последовательность])** - возвращает неизменяемое множество.

**int([object], [основание системы счисления])** - преобразование к целому числу.

**list([object])** - создает список.

**memoryview([object])** - создает объект memoryview.

**object()** - возвращает безликий объект, являющийся базовым для всех объектов.

**range([start=0], stop, [step=1])** - арифметическая прогрессия от start до stop с шагом step.

**set([object])** - создает множество.

**slice([start=0], stop, [step=1])** - объект среза от start до stop с шагом step.

**str([object], [кодировка], [ошибки])** - строковое представление объекта. Использует метод \_\_str\_\_.

**tuple(obj)** - преобразование к кортежу.

**abs(x)** - Возвращает абсолютную величину (модуль числа).

**all(последовательность)** - Возвращает True, если все элементы истинные (или, если последовательность пуста).

**any(последовательность)** - Возвращает True, если хотя бы один элемент - истина. Для пустой последовательности возвращает False.

**ascii(object)** - Как repr(), возвращает строку, содержащую представление объекта, но заменяет не-ASCII символы на экранированные последовательности.

**bin(x)** - Преобразование целого числа в двоичную строку.

**callable(x)** - Возвращает True для объекта, поддерживающего вызов (как функции).

**chr(x)** - Возвращает односимвольную строку, код символа которой равен x.

**classmethod(x)** - Представляет указанную функцию методом класса.

**compile(source, filename, mode, flags=0, dont\_inherit=False)** - Компиляция в программный код, который впоследствии может выполниться функцией eval или exec. Строка не должна содержать символов возврата каретки или нулевые байты.

**delattr(object, name)** - Удаляет атрибут с именем 'name'.

**dir([object])** - Список имен объекта, а если объект не указан, список имен в текущей локальной области видимости.

**divmod(a, b)** - Возвращает частное и остаток от деления a на b.

**enumerate(iterable, start=0)** - Возвращает итератор, при каждом проходе предоставляющем кортеж из номера и соответствующего члена последовательности.

**eval(expression, globals=None, locals=None)** - Выполняет строку программного кода.

**exec(object[, globals[, locals]])** - Выполняет программный код на Python.

**filter(function, iterable)** - Возвращает итератор из тех элементов, для которых function возвращает истину.

**format(value[,format\_spec])** - Форматирование (обычно форматирование строки).

**getattr(object, name ,[default])** - извлекает атрибут объекта или default.

**globals()** - Словарь глобальных имен.

**hasattr(object, name)** - Имеет ли объект атрибут с именем 'name'.

**hash(x)** - Возвращает хеш указанного объекта.

**help([object])** - Вызов встроенной справочной системы.

**hex(х) -** Преобразование целого числа в шестнадцатеричную строку.

**id(object)** - Возвращает "адрес" объекта. Это целое число, которое гарантированно будет уникальным и постоянным для данного объекта в течение срока его существования.

**input([prompt])** - Возвращает введенную пользователем строку. Prompt - подсказка пользователю.

**isinstance(object, ClassInfo)** - Истина, если объект является экземпляром ClassInfo или его подклассом. Если объект не является объектом данного типа, функция всегда возвращает ложь.

**issubclass(класс, ClassInfo)** - Истина, если класс является подклассом ClassInfo. Класс считается подклассом себя.

**iter(x)** - Возвращает объект итератора.

**len(x)** - Возвращает число элементов в указанном объекте.

**locals()** - Словарь локальных имен.

**map(function, iterator)** - Итератор, получившийся после применения к каждому элементу последовательности функции function.

**max(iter, [args ...] \* [, key])** - Максимальный элемент последовательности.

**min(iter, [args ...] \* [, key])** - Минимальный элемент последовательности.

**next(x)** - Возвращает следующий элемент итератора.

**oct(х)** - Преобразование целого числа в восьмеричную строку.

**open(file, mode='r', buffering=None, encoding=None, errors=None, newline=None, closefd=True)** - Открывает файл и возвращает соответствующий поток.

**ord(с)** - Код символа.

**pow(x, y[, r])** - ( x \*\* y ) % r.

**reversed(object)** - Итератор из развернутого объекта.

**repr(obj)** - Представление объекта.

**print([object, ...], \*, sep=" ", end='\n', file=sys.stdout)** - Печать.

**property(fget=None, fset=None, fdel=None, doc=None)**

**round(X [, N])** - Округление до N знаков после запятой.

**setattr(объект, имя, значение)** - Устанавливает атрибут объекта.

**sorted(iterable[, key][, reverse])** - Отсортированный список.

**staticmethod(function)** - Статический метод для функции.

**sum(iter, start=0)** - Сумма членов последовательности.

**super([тип [, объект или тип]])** - Доступ к родительскому классу.

**type(object)** - Возвращает тип объекта.

**type(name, bases, dict) -** Возвращает новый экземпляр класса name.

**vars([object])** - Словарь из атрибутов объекта. По умолчанию - словарь локальных имен.

**zip(\*iters)** - Итератор, возвращающий кортежи, состоящие из соответствующих элементов аргументов-последовательностей.

**Взаимодействие с операционной системой**

Модуль os предоставляет десятки функций для взаимодействия с операционной системой.

>>> import os  
>>> os.getcwd() # возвращает путь к текущему каталогу  
'C:\\Python27'  
>>> os.system('dir \*.txt') # выполнить указанную команду ОС  
...список текстовых файлов, выведенных командой...  
0  
>>> os.chdir('/server/accesslogs') # сменить текущий каталог  
Лучше всего применять import os вместо from os import \*. Это предохранит встроенную функцию open() от замещения функцией os.open(), имеющей несколько иное назначение.

В интерактивном режиме встроенные функции dir() и help() помогут разобраться с большими модулями вроде os:

>>> import os  
>>> dir(os)  
...список всех атрибутов модуля...  
>>> help(os)  
...страница руководства по модулю на основе строк документации...

Для управления файлами и каталогами удобный высокоуровневый интерфейс предоставляет модуль shutil:

>>> import shutil  
>>> shutil.copyfile('data.db', 'archive.db') # копировать файл  
>>> shutil.move('/build/executables', 'installdir') # переместить каталог

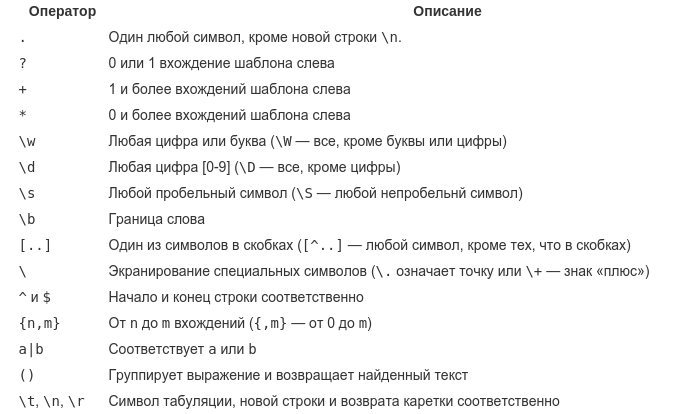
**Сравнение строк по шаблонам**

Модуль re предоставляет инструментарий для работы с регулярными выражениями для нетривиальной обработки текста. С помощью регулярных выражений можно создавать выразительные и оптимизированные решения для поиска и обработки строк.

>>> import re  
>>> re.findall(r'\bf[a-z]\*', 'which foot or hand fell fastest')  
['foot', 'fell', 'fastest']  
>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')  
'cat in the hat'

Когда требуются более простые возможности, методы строковых объектов предпочтительнее, так как их легче читать и отлаживать:

>>> 'чай для двоих'.replace('для', 'на')  
'чай на двоих'



**Математические функции**

Модуль math предоставляет доступ к библиотеке языка C для функций над числами с плавающей запятой:

>>> import math  
>>> math.cos(math.pi / 4)  
0.70710678118654757  
>>> math.log(1024, 2)  
10.0

С помощью модуля random можно делать случайный выбор:

>>> import random  
>>> random.choice(['яблоко', 'груша', 'банан'])  
'яблоко'  
>>> random.sample(range(100), 10) # выборка без повторений  
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]  
>>> random.random() # случайное число с плавающей запятой  
0.17970987693706186  
>>> random.randrange(6) # случайное целое из диапазона range(6)  
4  
>>> random.sample([1, 2, 3, 4, 5], 3) # случайные три элемента из списка  
[4, 1, 5]

Проект SciPy <[http://scipy.org](http://scipy.org/)> имеет много других модулей для численных расчётов.

**Протоколы интернет**

В стандартной библиотеке имеется целый набор модулей для различных сервисов и протоколов интернет. Наиболее употребимыми можно считать urllib.request для получения данных по заданному адресу (URL) и smtplib для отправки сообщений электронной почты:

>>> from urllib.request import urlopen  
>>> for line in urlopen('http://tycho.usno.navy.mil/cgi-bin/timer.pl'):  
... if 'EST' in line or 'EDT' in line: # временные зоны  
... print(line)  
   
<BR>Nov. 25, 09:43:32 PM EST  
   
>>> import smtplib  
>>> server = smtplib.SMTP('localhost')  
>>> server.sendmail('soothsayer@example.org', 'jcaesar@example.org',  
... """To: jcaesar@example.org  
... From: soothsayer@example.org  
...  
... Beware the Ides of March.  
... """)  
>>> server.quit()

**Дата и время**

Модуль datetime предлагает классы для работы с датами и временем как в простых, так и сложных случаях. Кроме поддержки календарной арифметики, реализация обращает особенное внимание на эффективность вычисления составных частей для вывода и дальнейших манипуляций. Модуль также предлагает объекты с поддержкой временных зон.

# даты можно легко составлять и выводить в требуемом формате  
>>> from datetime import date  
>>> сейчас = date.today()  
>>> сейчас  
datetime.date(2009, 2, 3)  
>>> сейчас.strftime("%d.%m.%Y")  
'03.02.2009'  
   
# даты поддерживают календарную арифметику  
>>> день\_рождения = date(1964, 7, 31)  
>>> возраст = сейчас - день\_рождения  
>>> возраст.days  
16258

**Сжатие данных и архивы**

Наиболее распространённые форматы сжатия и архивации напрямую поддерживаются модулями стандартной библиотеки:zlib, gzip, bz2, zipfile и tarfile.

>>> import zlib  
>>> s = "Закрой замок на замок, чтобы замок не замок"  
>>> len(bytes(s, "utf-8"))  
78  
>>> t = zlib.compress(s)  
>>> len(t)  
54  
>>> print(str(zlib.decompress(t), "utf-8"))  
Закрой замок на замок, чтобы замок не замок

**Виртуальное окружение**

Virtualenv - предназначен для создания виртуального окружения для проекта. К примеру, вашему проекту нужен отличный от установленного python или какие-либо библиотеки отличных версий, вы можете создать общее окружение для нескольких проектов или по одному собственному для каждого проекта. Тем самым вы сможете устанавливать, изменять и удалять пакеты, и это не повлияет на другие ваши проекты или системное окружение.

**Использование**

**--no-site-packages**

Запретить использование системного site-packages (для полной изоляции вашего окружения от системы). Например у вас в системе установлена "Django 1.3", если вы будете использовать эту опцию, то в созданном окружении эта "Django" не будет доступна.

**В текущих версиях virtualenv эта опция используется по умолчанию, можете её не указывать.**

**--system-site-packages**

Эта опция противоположна предыдущей, то есть заставляет окружение использовать установленные в системе пакеты, если не нашлись онные в окружении.

**-p PYTHON\_EXE, --python=PYTHON\_EXE**

Вы можете указать нужную вам версию интерпретатора python, при этом он должен быть установлен в системе. Если вы опустили эту опцию, то будет использоваться умолчательный (Выполните which python чтобы узнать какой он у вас, но скорее всего это будет /usr/bin/python).

**--extra-search-dir=SEARCH\_DIRS**

Указанные директории будут использовать для поиска в них пакетов при установке через setuptools/distribute/pip. Это подходит в случае когда вы не хотите их устаналивать из PyPI.

**--never-download**

Не использовать сеть для получения пакетов. Используется совместно с--extra-search-dir=SEARCH\_DIRS.

**--relocatable**

Сделать существующее окружение относительным, то есть после этого вы сможете перемещать каталог с окружением и оно не будет зависеть от своего места дислокации, но опция в данный момент экспериментальная.

**--prompt=PROMPT**

Выводимое имя окружения (подсказка), например вы можете создать с названием(MY\_VENV):

$ virtualenv --prompt="(MY\_VENV)" **<venv\_name>**  
source venv/bin/activate  
(MY\_VENV)$

**--clear**

Используется для очистки существующего окружения от пакетов и прочих изменений.

**Создание окружения**

Создаем общее окружение (для нескольких проектов) без использования системногоsite-packages:

$ mkdir ~/venv && cd ~/venv  
$ virtualenv --no-site-packages <venv\_name>

Часто я создаю окружение непосредственно в каталоге проекта:

$ cd ~/work/**<project\_name>**  
$ mkdir venv && echo "Virtualenv directory" > venv/README  
$ git add venv && echo "/venv/" >> .gitignore && git add -f .gitignore  
$ virtualenv --no-site-packages --prompt="(**<project\_name>**)" **<venv\_name>**

**Структура виртуального окружения**

После того как вы установили новое окружение, у вас будет следующая структура:

~/venv/**<venv\_name>**  
├── bin  
│ ├── activate  
│ ├── easy\_install  
│ ├── pip  
│ ├── python  
│ └── ...  
├── include  
│ └── python2.7 -> /usr/include/python2.7  
├── lib  
│ └── python2.7  
│ ├── distutils  
│ │ └── ...  
│ ├── site-packages  
│ │ └── ...  
│ └── ...  
└── local  
 ├── bin -> /home/username/venv/**<venv\_name>**/bin  
 ├── include -> /home/username/venv/**<venv\_name>**/include  
 └── lib -> /home/username/venv/**<venv\_name>**/lib

Таким образом вам сразу же доступны изолированные python, pip и easy\_install.

# **Задания к самостоятельной работе**

1. Переделать первое домашние задание:
   1. Вынесением каждого условия в отдельную функцию
   2. Затем вынести каждую функцию по модулям
   3. Собрать модули в пакет с подключением пакета в программу
2. Написать программу, которая в интерактивном режиме просит ввести путь к папкам. Затем просит ввести название папки/файла. Затем если по заданному пути находится файл/папка, то выводить: ее полный путь, размер, тип объекта (файл или директория), дату создания изменения объекта.
3. Написать программу шар предсказания. Создать массив, в котором содержится до девяти вариантов ответов. Затем при запусках программы пользователя просят ввести вопрос, а программа выдает случайный ответ из с генерированного диапазона ответов. Ответ выдавать: вопрос тире ответ.
4. Написать программу, которая заполняет случайным текстом строку.
   1. Попросить ввести символы пользователя
   2. Заменить регулярным выражением это слово на ее ревертированное значение если это слово в начале или конце это предложения
5. Создать виртуальное окружение для каждой программы совой

# **Список использованной литературы**

* Встроенные библиотеки
  + <https://docs.python.org/2.7/>
  + <http://pep8.ru/doc/tutorial-3.1/10.html>
  + <http://pythonworld.ru/osnovy/vstroennye-funkcii.html>
  + <https://www.opennet.ru/docs/RUS/python/builts_in.html>
  + <http://pythoner.name/taxonomy/built-in>
  + <https://ru.wikibooks.org/wiki/Python/%D0%A1%D0%BF%D1%80%D0%B0%D0%B2%D0%BE%D1%87%D0%BD%D0%B8%D0%BA_%D0%BF%D0%BE_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B5_Python_2.6>
* сторонние библиотеки
  + <https://pypi.python.org/pypi>
  + <http://www.lfd.uci.edu/~gohlke/pythonlibs/>
  + <http://kpavlovsky.pro/2011/10/15/python-pip-django-workplace-setup/>
  + <http://pythontips.com/2013/07/30/20-python-libraries-you-cant-live-without/>
  + <http://www.catswhocode.com/blog/python-50-modules-for-all-needs>
* виртуальное окружение
  + <http://www.razrabotka.org/django/venv/>
  + <http://itman.in/python-windows-virtualenv/>
  + <http://habrahabr.ru/post/157287/>
  + <http://proft.me/2010/04/3/python-i-okruzhenie-virtualenv/>
  + <http://swasher.pp.ua/python-i-virtualnoe-okruzhenie.html>
* <https://www.youtube.com/watch?v=ca-W83Y397A>

# **ЛАБОРАТОРНАЯ РАБОТА №5**

**Основы ООП**

# **Общие теоретические сведения**

Python соответствует принципам объектно-ориентированного программирования. В python всё является объектами - и строки, и списки, и словари, и всё остальное.

Но возможности ООП в python этим не ограничены. Программист может написать свой тип данных (класс), определить в нём свои методы.

Это не является обязательным - мы можем пользоваться только встроенными объектами. Однако ООП полезно при долгосрочной разработке программы несколькими людьми, так как упрощает понимание кода.

**Первый класс**

Приступим теперь собственно к написанию своих классов на python. Попробуем определить собственный класс:

>>>

>>> *# Пример самого простейшего класса*  
... class A:  
... pass

Теперь мы можем создать несколько экземпляров этого класса:

>>>

>>> a = A()  
>>> b = A()  
>>> a.arg = 1 *# у экземпляра a появился атрибут arg, равный 1*  
>>> b.arg = 2 *# а у экземпляра b - атрибут arg, равный 2*  
>>> print(a.arg)  
1

**Метод в классе**

Классу возможно задать собственные методы:

>>>

>>> class A:  
... def g(self): *# self - обязательный аргумент, содержащий в себе экземпляр*  
... *# класса, передающийся при вызове метода,*  
... *# поэтому этот аргумент должен присутствовать*  
... *# во всех методах класса.*  
... return 'hello world'  
...  
>>> a = A()  
>>> a.g()  
'hello world'

**Пример**

Еще один пример:

>>>

>>> class B:  
... arg = 'Python' *# Все экземпляры этого класса будут иметь атрибут arg,*  
... *# равный "Python"*  
... *# Но впоследствии мы его можем изменить*  
... def g(self):  
... return self.arg  
...  
>>> b = B()  
>>> b.g()  
'Python'  
>>> B.g(b)  
'Python'  
>>> b.arg = 'spam'  
>>> b.g()  
'spam'

# **Задания к самостоятельной работе**

1. Переделать первое домашнее задание:
   1. Создать класс для первого задани
   2. Классу добавить атрибуты
   3. Вынести все функции функциями классами(методом класса)
   4. Создать конструктор который будет в соответствии с параметром атрибута выполнять функцию класса
2. Переделать второе задание первого домашнего задания:
   1. Создать класс для второго задания
   2. Классу добавить атрибуты
   3. Вынести все функции функциями классами
   4. Создать конструктор который будет в соответствии с параметром атрибута выполнять функцию класса

# **Список использованной литературы**

* <http://sheremetov.com/oop/classes-object/>
* <http://habrahabr.ru/post/148015/>
* <http://megamozg.ru/post/6908/>
* <http://habrahabr.ru/post/147927/>
* <http://www.remoteshaman.com/programming/common/oop-for-dummy>
* <http://younglinux.info/oopython/oop.php>
* Синтаксис
  + <https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_Python>
  + <http://pythonworld.ru/osnovy/obektno-orientirovannoe-programmirovanie-obshhee-predstavlenie.html>

# **ЛАБОРАТОРНАЯ РАБОТА №6**

**Тема: Классы и ООП в Python**

# **Общие теоретические сведения**

**Инкапсуляция** — ограничение доступа к составляющим объект компонентам (методам и переменным). Инкапсуляция делает некоторые из компонент доступными только внутри класса.

Инкапсуляция в Python работает лишь на уровне соглашения между программистами о том, какие атрибуты являются общедоступными, а какие — внутренними.

**Инкапсуляция**

Одиночное подчеркивание в начале имени атрибута говорит о том, что переменная или метод не предназначен для использования вне методов класса, однако атрибут доступен по этому имени.

class A:  
 def \_private(self):  
 print("Это приватный метод!")  
  
>>> a = A()  
>>> a.\_private()  
Это приватный метод!

**Инкапсуляция**

Двойное подчеркивание в начале имени атрибута даёт большую защиту: атрибут становится недоступным по этому имени.

>>> class B:  
... def \_\_private(self):  
... print("Это приватный метод!")  
...  
>>> b = B()  
>>> b.\_\_private()  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
AttributeError: 'B' object has no attribute '\_\_private'

Однако полностью это не защищает, так как атрибут всё равно остаётся доступным под именем \_ИмяКласса\_\_ИмяАтрибута:

>>> b.\_B\_\_private()  
Это приватный метод!

**Наследование**

**Наследование** подразумевает то, что дочерний класс содержит все атрибуты родительского класса, при этом некоторые из них могут быть переопределены или добавлены в дочернем. Например, мы можем создать свой класс, похожий на словарь:

## >>> class Mydict(dict): ... def get(self, key, default = 0): ... return dict.get(self, key, default) ... >>> a = dict(a=1, b=2) >>> b = Mydict(a=1, b=2)

# **Множественное наследование**

## При описании предметной области классы могут образовывать иерархию, в корне которой стоит базовый класс, а нижележащие классы (подклассы) наследуют свои атрибуты, уточняя и расширяя поведение вышележащего класса (надкласса).

## Python поддерживает как одиночное наследование, так и множественное, позволяющее классу быть производным от любого количества базовых классов.

## >>> **class** **Par1**(object): *# наследуем один базовый класс - object* **def** name1(self): **return** 'Par1' >>> **class** **Par2**(object): **def** name2(self): **return** 'Par2' >>> **class** **Child**(Par1, Par2): *# создадим класс, наследующий Par1, Par2 (и, опосредованно, object)* **pass** >>> x = Child() >>> x.name1(), x.name2() *# экземпляру Child доступны методы из Par1 и Par2* 'Par1','Par2'

# **Наследование**

## Класс Mydict ведёт себя точно так же, как и словарь, за исключением того, что метод get по умолчанию возвращает не None, а 0.

## >>> b['c'] = 4 >>> print(b) {'a': 1, 'c': 4, 'b': 2} >>> print(a.get('v')) None >>> print(b.get('v')) 0

# **Полиморфизм**

## **Полиморфизм** - разное поведение одного и того же метода в разных классах. Например, мы можем сложить два числа, и можем сложить две строки. При этом получим разный результат, так как числа и строки являются разными классами.

## >>>

## >>> 1 + 1 2 >>> "1" + "1" '11'

# Метод super() в Python

## В Python есть метод super(), который обычно применяется к объектам.

## Его главная задача это возможность использования в классе потомке, методов класса-родителя.

## # Родительский класс

## class A(object):

## def \_\_init\_\_(self):

## print(u'конструктор класса A')

## # Потомок класса А

## class B(A):

## def \_\_init\_\_(self):

## print(u'конструктор класса B')

## super(B,self).\_\_init\_\_()

# **Переопределение методов**

## Использование полиморфизма при наследовании классов позволяет переопределять методы суперклассов их подклассами. Например, может возникнуть ситуация, когда все подклассы реализуют определенный метод из суперкласса, и лишь один подкласс должен иметь его другую реализацию. В таком случае метод переопределяется в подклассе. Пример:

## **class** Base: **def** \_\_init\_\_(self,n): self.numb = n **def** out(self): **print** (self.numb) **class** One(Base): **def** multi(self,m): self.numb \*= m **class** Two(Base): **def** inlist(self): self.inlist = list(str(self.numb)) **def** out(self): i = 0 **while** i < len(self.inlist): **print** (self.inlist[i]) i += 1 obj1 = One(45) obj2 = Two('abc') obj1.multi(2) obj1.out() *# Вывод числа 90* obj2.inlist() obj2.out() *# Вывод в столбик букв a, b, c*

## В данном случае объект obj1 использует метод out из cуперкласса Base, а obj2 – из своего класса Two. Атрибуты ищутся «снизу вверх»: сначала в классах, затем суперклассах. Поскольку для obj2 атрибут out уже был найден в классе Two, то из класса Base он не используется. Другими словами, класс Two переопределят атрибут суперкласса Base.

# **Расширения методов**

## При ООП может возникнуть ситуация, когда метод суперкласса в принципе подходит для реализации того или иного действия с объектами класса, однако требует небольших изменений. В таком случае можно использовать так называемое расширение метода, когда из тела метода класса вызывается метод суперкласса и дописываются дополнительные инструкции. В примере ниже в методе класса Subclass вызывается метод другого класса (в данном случае его суперкласса; однако может вызываться метод, не принадлежащий собственному суперклассу):

## **class** Base: **def** \_\_init\_\_(self,N): self.numb = N **def** out(self): self.numb /= 2 **print** (self.numb) **class** Subclass(Base): **def** out(self): **print** ("**\n**----") Base.out(self) **print** ("----**\n**") i = 0 **while** i < 10: **if** 4 < i < 7: obj = Subclass(i) **else**: obj = Base(i) i += 1 obj.out()

# **Абстрактные классы**

## Начиная с версии языка 2.6 в стандартную библиотеку включается модуль abc, добавляющий в язык абстрактные базовые классы (далее АБК).

## АБК позволяют определить класс, указав при этом, какие методы или свойства обязательно переопределить в классах-наследниках:

## from abc import ABCMeta, abstractmethod, abstractproperty class Movable(): \_\_metaclass\_\_=ABCMeta @abstractmethod def move(): """Переместить объект""" @abstractproperty def speed(): """Скорость объекта"""

## Таким образом, если мы хотим использовать в коде объект, обладающий возможностью перемещения и определенной скоростью, то следует использовать класс Movable в качестве одного из базовых классов.

## Наличие необходимых методов и атрибутов объекта теперь гарантируется наличием АБК среди предков класса:

## class Car(Movable): def \_\_init\_\_: self.speed = 10 self.x = 0 def move(self): self.c += self.speed def speed(self): return self.speed assert issubclass(Car, Movable) assert ininstance(Car(), Movable)

## Видно, что понятие АБК хорошо вписывается в иерархию наследования классов, использовать их легко, а реализация, если заглянуть в исходный код модуля abc, очень проста. Абстрактные классы используются в стандартных модулях collections и number, задавая необходимые для определения методы пользовательских

## классов-наследников.

# **Статически методы**

## Внутри класса можно создать метод, который будет доступен без создания экземпляра класса. Для этого перед определением метода внутри класса следует указать декоратор @staticmethod. Вызов статического метода без создания экземпляра класса осуществляется следующим образом:

## <Название класса>.<Название метода>(<Параметры>)

## Кроме того, можно вызвать статический метод через экземпляр класса:

## <Экземпляр класса>.<Название метода>(<Параметры>)

## Пример использования статических методов приведен ниже.

## class Class1(object): @staticmethod def sum1(x, y): # Статический метод return x + y def sum2(self, x, y): # Обычный метод в классе return x + y def sum3(self, x, y): return Class1.sum1(x, y) # Вызов из метода класса print Class1.sum1(10, 20) # Вызываем статический метод c1 = Class1() print c1.sum2(15, 6) # Вызываем метод класса print c1.sum1(50, 12) # Вызываем статический метод # через экземпляр класса print c1.sum3(23, 5) # Вызываем статический метод # внутри класса

## Обратите внимание на то, что в определении статического метода нет параметра self. Это означает, что внутри статического метода нет доступа к атрибутам и методам экземпляра класса. Методы класса создаются с помощью декоратора @classmethod. В качестве первого параметра в метод класса передается ссылка на класс, а не на экземпляр класса. Вызов метода класса осуществляется следующим образом.

## <Название класса>.<название метода>(<Параметры>)

## Кроме того, можно вызывать метод класса через экземпляр класса

## <Экземпляр класса>.<Название метода>(<Параметры>)

# **Задания к самостоятельной работе**

1. Переписать первое домашнее задание
2. Написать абстрактный класс в котором будет только один атрибут который и есть введённое нами число по умолчанию в конструкторе задается значение ноль
3. написать Класс в котором будут описанны все функции с первого домашнего задания
4. создаем конечный класс в котором наследуем два предыдущих класса в нем же перегружаем конструктор

# **Список использованной литературы**

* <https://www.python.org/dev/peps/pep-3119/>
* <http://habrahabr.ru/post/72757/>
* <https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_Python>
* <http://smuncertainty.blogspot.com/2013/05/python-1.html>
* <http://younglinux.info/book/export/html/36>
* <http://gos-it.wikia.com/wiki/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5_%D0%BF%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF%D1%8B_%D0%9E%D0%9E%D0%9F:_%D0%B8%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F,_%D0%BD%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5,_%D0%BF%D0%BE%D0%BB%D0%B8%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D0%BC>
* статические методы
  + <http://python-3.ru/page/staticheskie-metody-i-metody-klassa#cut>
* Наследование
  + <http://python-3.ru/page/nasledovanie-oop-python>
* super
  + <http://fkn.ktu10.com/?q=node/4087>

# **ЛАБОРАТОРНАЯ РАБОТА №7**

# **Тема: Работа с файлами**

# **Общие теоретические сведения**

# **Открытие файла**

## Прежде, чем работать с файлом, его надо открыть. С этим замечательно справится встроенная функция open:

## f = open('text.txt', 'r')

|  |  |
| --- | --- |
| Режим | Обозначение |
| 'r' | открытие на чтение (является значением по умолчанию). |
| 'w' | открытие на запись, содержимое файла удаляется, если файла не существует, создается новый. |
| 'x' | открытие на запись, если файла не существует, иначе исключение. |
| 'a' | открытие на дозапись, информация добавляется в конец файла. |
| 'b' | открытие в двоичном режиме. |
| 't' | открытие в текстовом режиме (является значением по умолчанию). |
| '+' | открытие на чтение и запись |

## У функции open много параметров, нам пока важны 3 аргумента: первый, это имя файла. Путь к файлу может быть **относительным** или **абсолютным**. Второй аргумент, это режим, в котором мы будем открывать файл.

# **Чтение из файла**

## Открыли мы файл, а теперь мы хотим прочитать из него информацию. Для этого есть несколько способов, но большого интереса заслуживают лишь два из них.

## Первый - метод read, читающий весь файл целиком, если был вызван без аргументов, и n символов, если был вызван с аргументом (целым числом n).

## >>> f = open('text.txt') >>> f.read(1) 'H' >>> f.read() 'ello world!\nThe end.\n\n'

## Ещё один способ сделать это - прочитать файл построчно, воспользовавшись циклом for:

## >>> f = open('text.txt') >>> for line in f: ... line ... 'Hello world!\n' '\n' 'The end.\n' '\n'

# **Методы объектов файлов**

## После того как вы вызвали метод read() на файловом объекте, если вы повторно вызовете read(), то увидите лишь пустую строку. Это происходит потому, что после первого прочтения указатель находится вконце файла. Для того чтобы узнать позицию указателя можно использовать метод tell().

## my\_file = open("some.txt")

## my\_file.read(10)

## print ("Я на позиции:", my\_file.tell())

## my\_file.close()

## Говоря проще, метод tell() сообщает в скольки байтах от начала файла мы сейчас находимся.

## Чтобы перейти на нужную нам позицию, следует использовать другой метод - seek().

## my\_file.seek(offset, [from])

## Аргумент offset указывет на сколько байт перейти. опциональный аргумент *from* означает позицию, с которой начинается движение. 0 - означает начало файла, 1 нынешняя позиция, 2 - конец файла.

## my\_file = open("some.txt", "r")

## print(my\_file.read(10))

## print("Мы находимся на позиции: ", my\_file.tell())

## # Возвращаемся в начало

## my\_file.seek(0)

## print(my\_file.read(10))

## my\_file.close()

# **Запись в файл**

## Теперь рассмотрим запись в файл. Попробуем записать в файл вот такой вот список:

## >>> l = [str(i)+str(i-1) for i in range(20)] >>> l ['0-1', '10', '21', '32', '43', '54', '65', '76', '87', '98', '109', '1110', '1211', '1312', '1413', '1514', '1615', '1716', '1817', '1918']

## Откроем файл на запись:

## >>> f = open('text.txt', 'w')

## Запись в файл осуществляется с помощью метода write:

## >>> for index in l: ... f.write(index + '\n') ... 4 3 3 3 3

## Для тех, кто не понял, что это за цифры, поясню: метод write возвращает число записанных символов.

## После окончания работы с файлом его обязательно нужно закрыть с помощью метода close:

## >>> f.close()

# **Запись в файл**

## Теперь попробуем воссоздать этот список из получившегося файла. Откроем файл на чтение, и прочитаем строки.

## >>> f = open('text.txt', 'r') >>> l = [line.strip() for line in f] >>> l ['0-1', '10', '21', '32', '43', '54', '65', '76', '87', '98', '109', '1110', '1211', '1312', '1413', '1514', '1615', '1716', '1817', '1918'] >>> f.close()

## Мы получили тот же список, что и был.

# Постраничная работа с файлом

## Обычно мы имеем дело с текстовыми файлами. Прочитать одну строку:

## **file.readline()**

## Функция readline() без параметра читает всю строку, наличие параметра указывает функции максимальное число символов строки, которое будет прочитано. Прочитать все строки и вернуть список строк:

## **file.readlines()**

## Записать строки в файл:

## **file.writelines()**

## Пример. Прочитать файл и записать его содержимое в другой файл:

## **f = open(r'my\_file') lines = f.readlines() f.close() lines[0] = "This is a my\_file2 \n" # изменяем 1-ю строку f = open(r'my\_file2','w') f.writelines(lines) f.close()**

# **Закрытие файла**

## Для закрытия файла есть метод close(). Обычно файл закрывается сам после того, как вы выходите из программы, но файлы нужно закрывать вручную по нескольким причинам.

## Питон может буферизировать запись в файл ваших данных, что может привести к неожиданным эффектам и возникновению ошибок.

## У операционной системы есть ограничение на число одновременно открытых файлов.

## При доступе к файлу из разных мест одновременно и на чтение, и на запись необходимо синхронизировать файловые операции. Буферизация записи может привести к тому, что запись уже произошла, а данных в файле еще нет.

## Для полной уверенности в закрытии файла можно использовать блок try/finally:

## try: # Тут идет запись в файл finally: file.close()

## Если вы все же не хотите закрывать файл, то синхронизировать многопользовательский доступ к файлу на чтение/запись можно с помощью функции flush(), которая актуализирует все операции записи на диск. При этом возможна блокировка файла на чтение.

## **Работа с файловой системой**

## Стандартный модуль os имеет интерфейс работы с файловой системой. Каждая программа имеет текущий каталог. Функция os.getcwd возвращает текущий каталог:

## import os cwd = os.getcwd() print cwd

## Проверить наличие файла в текущем каталоге:

## os.path.exists('my\_file')

## Вывести список файлов и подкаталогов для данного каталога:

## os.listdir(path)

## Следующий пример **рекурсивно** выводит список всех файлов и подкаталогов для данного каталога:

## import os def walk(dir): for name in os.listdir(dir): path = os.path.join(dir, name) if os.path.isfile(path): print path else: walk(path) walk(path)

## В следующем примере мы получим статистическую информацию о текущем каталоге: общий размер каталога в байтах, число файлов, число подкаталогов. Стандартная функция os.path.walk имеет три параметра: каталог, пользовательская функция, список для подсчета:

## import os, sys def getlocaldata(sms,dr,flst): for f in flst: fullf = os.path.join(dr,f) if os.path.islink(fullf): continue # don't count linked files if os.path.isfile(fullf): sms[0] += os.path.getsize(fullf) sms[1] += 1 else: sms[2] += 1 def dtstat(dtroot): sums = [0,0,1] # 0 bytes, 0 files, 1 directory so far os.path.walk(dtroot,getlocaldata,sums) return sums report = dtstat('.') print report

## В следующем примере сделана интерпретация системной утилиты grep. В текущем каталоге будут найдены файлы с питоновским расширением, в которых будет найдена поисковая строка 'import os':

## import os, sys, fnmatch mask = '\*.py' pattern = 'import os' def walk(arg,dir,files): for file in files: if fnmatch.fnmatch(file,mask): name = os.path.join(dir,file) try: data = open(name,'rb').read() if data.find(pattern) != -1: print name except: pass os.path.walk('.',walk,[])

## **CSV** (от англ. *Comma-Separated Values* — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Каждая строка файла — это одна строка таблицы. Значения отдельных колонок разделяются разделительным символом (delimiter) — запятой (,). Однако, большинство программ вольно трактует стандарт CSV и допускают использование иных символов в качестве разделителя.

## Для чтения и записи файла в формате CSV предназначен модуль csv. Этот модуль предназначен для работы с различными диалектами: разделитель запятая, разделитель точка с запятой, разделитель табуляция (Excel). Простейший пример:

## **import** **csv** reader = csv.reader(open("some.csv", "rb")) **for** row **in** reader: **print** row

## Модуль позволяет настроить формат читаемых и записываемых файлов. Например можно выставить разделитель полей :, разделитель строк — |, символ цитирования — ` (вместо " по умолчанию).

## **import** **csv** csv\_fh = open("some.csv", "wb"): writer = csv.writer(csv\_fh, delimiter=':', quoting=csv.QUOTE\_MINIMAL, quotechar='`', lineterminator='|') writer.writerows([ [1997, "Ford", "E350", "ac, abs, moon", "3000.00"], [1999, "Chevy", "Venture `Extended Edition`", "", "4900.00"], [1996, "Jeep", "Grand Cherokee", "air, moon roof, loaded MUST SELL!", "4799.00"] ])

## **JSON** (англ. *JavaScript Object Notation*) — текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком. Как и многие другие текстовые форматы, JSON легко читается людьми.

## Несмотря на происхождение от JavaScript , формат считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

## **json.dump(obj, fp, skipkeys=False, ensure\_ascii=True, check\_circular=True, allow\_nan=True, cls=None, indent=None, separators=None, default=None, sort\_keys=False, \*\*kw)** - сериализует obj как форматированный JSON поток в fp.

## Если **skipkeys = True**, то ключи словаря не базового типа (str, unicode, int, long, float, bool, None) будут проигнорированы, вместо того, чтобы вызывать исключение TypeError.

## Если **ensure\_ascii = True**, все не-ASCII символы в выводе будут экранированы последовательностями \uXXXX, и результатом будет строка, содержащая только ASCII символы. Если ensure\_ascii = False, строки запишутся как есть.

## Если **check\_circular = False**, то проверка циклических ссылок будет пропущена, а такие ссылки будут вызывать OverflowError.

## Если **allow\_nan = False**, при попытке сериализовать значение с запятой, выходящее за допустимые пределы, будет вызываться ValueError (nan, inf, -inf) в строгом соответствии со спецификацией JSON, вместо того, чтобы использовать эквиваленты из JavaScript (NaN, Infinity, -Infinity).

## Если indent является неотрицательным числом, то массивы и объекты в JSON будут выводиться с этим уровнем отступа. Если уровень отступа 0, отрицательный или "", то вместо этого будут просто использоваться новые строки. Значение по умолчанию None отражает наиболее компактное представление. Если indent - строка, то она и будет использоваться в качестве отступа.

## Если sort\_keys = True, то ключи выводимого словаря будут отсортированы.

## **json.dumps(obj, skipkeys=False, ensure\_ascii=True, check\_circular=True, allow\_nan=True, cls=None, indent=None, separators=None, default=None, sort\_keys=False, \*\*kw)** - сериализует obj в строку JSON-формата.

## Аргументы имеют то же значение, что и для dump().

## Ключи в парах ключ/значение в JSON всегда являются строками. Когда словарь конвертируется в JSON, все ключи словаря преобразовываются в строки. В результате этого, если словарь сначала преобразовать в JSON, а потом обратно в словарь, то можно не получить словарь, идентичный исходному. Другими словами, loads(dumps(x)) != x, если x имеет нестроковые ключи.

## **json.load(fp, cls=None, object\_hook=None, parse\_float=None, parse\_int=None, parse\_constant=None, object\_pairs\_hook=None, \*\*kw)** - десериализует JSON из fp.

## **object\_hook** - опциональная функция, которая применяется к результату декодирования объекта (dict). Использоваться будет значение, возвращаемое этой функцией, а не полученный словарь.

## **object\_pairs\_hook** - опциональная функция, которая применяется к результату декодирования объекта с определённой последовательностью пар ключ/значение. Будет использован результат, возвращаемый функцией, вместо исходного словаря. Если задан так же object\_hook, то приоритет отдаётся object\_pairs\_hook.

## **parse\_float**, если определён, будет вызван для каждого значения JSON с плавающей точкой. По умолчанию, это эквивалентно float(num\_str).

## **parse\_int**, если определён, будет вызван для строки JSON с числовым значением. По умолчанию эквивалентно int(num\_str).

## parse\_constant, если определён, будет вызван для следующих строк: "-Infinity", "Infinity", "NaN". Может быть использовано для возбуждения исключений при обнаружении ошибочных чисел JSON.

## **json.loads(s, encoding=None, cls=None, object\_hook=None, parse\_float=None, parse\_int=None, parse\_constant=None, object\_pairs\_hook=None, \*\*kw)** - десериализует s (экземпляр str, содержащий документ JSON) в объект Python.

## Остальные аргументы аналогичны аргументам в load().

# **JSON (Пример)**

## Кодирование основных объектов Python:

## >>> import json >>> json.dumps(['foo', {'bar': ('baz', None, 1.0, 2)}]) '["foo", {"bar": ["baz", null, 1.0, 2]}]' >>> print(json.dumps("\"foo\bar")) "\"foo\bar" >>> print(json.dumps('\u1234')) "\u1234" >>> print(json.dumps('\\')) "\\" >>> print(json.dumps({"c": 0, "b": 0, "a": 0}, sort\_keys=True)) {"a": 0, "b": 0, "c": 0}

## Компактное кодирование:

## >>> import json >>> json.dumps([1,2,3,{'4': 5, '6': 7}], separators=(',', ':')) '[1,2,3,{"4":5,"6":7}]'

# **JSON (Пример)**

## Красивый вывод:

## >>>

## >>> import json >>> print(json.dumps({'4': 5, '6': 7}, sort\_keys=True, indent=4)) { "4": 5, "6": 7 }

## **Декодирование (парсинг) JSON:**

## >>>

## >>> import json >>> json.loads('["foo", {"bar":["baz", null, 1.0, 2]}]') ['foo', {'bar': ['baz', None, 1.0, 2]}] >>> json.loads('"\\"foo\\bar"') '"foo\x08ar'

# **Кодировщик декодировщик**

## **Класс json.JSONDecoder(**

## **object\_hook=None, parse\_float=None,**

## **parse\_int=None, parse\_constant=None, strict=True,**

## **object\_pairs\_hook=None)** - простой декодер JSON.

## Выполняет следующие преобразования при декодировании:

|  |  |
| --- | --- |
| **JSON** | **Python** |
| object | dict |
| array | list |
| string | str |
| number (int) | int |
| number (real) | float |
| true | True |
| false | False |
| null | None |

## Он также понимает NaN, Infinity, и -Infinity как соответствующие значения float, которые находятся за пределами спецификации JSON.

## **Класс json.JSONEncoder(skipkeys=False, ensure\_ascii=True, check\_circular=True, allow\_nan=True, sort\_keys=False, indent=None, separators=None, default=None)**

## Расширяемый кодировщик

## JSON для структур данных Python.

## Поддерживает следующие объекты и типы данных по умолчанию:

|  |  |
| --- | --- |
| **Python** | **JSON** |
| dict | object |
| list, tuple | array |
| str | string |
| int, float | number |
| True | true |
| False | false |
| None | null |

# **Задания к самостоятельной работе**

1. Дан файл с прайс листом одежды. Помимо названия товара указывается количество штук и количество пар. В одной строке указывается только один товар. Посчитать количество товаров которые считаются штуками и количество товаров считаемых парами. Результаты дописать в конец файла
2. Выполнить первое задание считав все значения из csv
3. Выполнить первое задание считывая значения из json файла результат выдать в другом json файле
4. Сохранить в json структуру любой папки у Вас на компьютере

# **Список использованной литературы**

* Работа с файлами
  + <http://pythonworld.ru/tipy-dannyx-v-python/fajly-rabota-s-fajlami.html>
  + <https://www.ibm.com/developerworks/ru/library/l-python_part_8/>
  + <http://pythonicway.com/python-fileio>
  + <http://slusar.su/izuchaem-python-11-rabota-s-fajlami/>
  + <http://python-3.ru/page/rabota-s-fajlami-v-python>
  + <http://gorodovets.blogspot.com/2013/07/python_27.html>
  + <http://santysoft.narod.ru/articles/art_py_use_file.html>
  + <http://master-develop.com/rabota-s-fajlami-v-python/>
  + CSV
    - <https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_Python#CSV>
    - <http://www.py-my.ru/post/4c0f5e791d41c80f34000003>
  + JSON
    - <http://www.programru.com/blog/MYDMzADMwIT5.html>
    - <http://www.py-my.ru/post/4bfb3c6a1d41c846bc0000b5>
    - <http://pythonworld.ru/moduli/modul-json.html>
    - <http://python-lab.ru/documentation/27/stdlib/json.html>
    - <http://pythonicway.com/education/basics/17-python-json-parse>

# **ЛАБОРАТОРНАЯ РАБОТА №8**

**Тема:** **Работа с датой и временем в Python**

# **Общие теоретические сведения**

В Python есть несколько стандартных модулей для работы с датой и временем:

datetime;

time;

calendar — модуль для работы с календарем; вывод календаря в виде похожем на вывод **cal**.

Все три модуля предназначены для работы с датой и временем, но однако отличаются своими возможностями.

**Datatime**

Модуль datetime предоставляет классы для обработки времени и даты разными способами. Поддерживается и стандартный способ представления времени, однако больший упор сделан на простоту манипулирования датой, временем и их частями.

Классы, предоставляемые модулем datetime

Класс datetime.date(year, month, day) - стандартная дата. Атрибуты: year, month, day. Неизменяемый объект.

## Класс datetime.time(hour=0, minute=0, second=0, microsecond=0, tzinfo=None) - стандартное время, не зависит от даты. Атрибуты: hour, minute, second, microsecond, tzinfo.

## Класс datetime.timedelta - разница между двумя моментами времени, с точностью до микросекунд.

## Класс datetime.tzinfo - абстрактный базовый класс для информации о временной зоне (например, для учета часового пояса и / или летнего времени).

## Класс datetime.datetime(year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=None) - комбинация даты и времени.

## Обязательные аргументы:

## datetime.MINYEAR (1) ≤ year ≤ datetime.MAXYEAR (9999)

## 1 ≤ month ≤ 12

## 1 ≤ day ≤ количество дней в данном месяце и году

## Необязательные:

## 0 ≤ minute < 60

## 0 ≤ second < 60

## 0 ≤ microsecond < 1000000

# **Методы класса datetime**

## **datetime.today()** - текущая дата, время равно 0.

## **datetime.fromtimestamp(timestamp)** - дата из стандартного представления времени.

## **datetime.fromordinal(ordinal)** - дата из числа, представляющего собой количество дней, прошедших с 01.01.1970.

## **datetime.now(tz=None)** - объект datetime из текущей даты и времени.

## **datetime.combine(date, time)** - объект datetime из комбинации объектов date и time.

## **datetime.strptime(date\_string, format)** - преобразует строку в datetime

## **datetime.strftime(format)** - см. функцию strftime из модуля time.

## **datetime.date()** - объект даты (с отсечением времени).

## **datetime.time()** - объект времени (с отсечением даты).

## **datetime.replace([year[, month[, day[, hour[, minute[, second[, microsecond[, tzinfo]]]]]]]])** - возвращает новый объект datetime с изменёнными атрибутами.

## **datetime.timetuple()** - возвращает struct\_time из datetime.

## **datetime.toordinal()** - количество дней, прошедших с 01.01.1970.

## **datetime.timestamp()** - возвращает время в секундах с начала эпохи.

## **datetime.weekday()** - день недели в виде числа, понедельник - 0, воскресенье - 6.

## **datetime.isoweekday()** - день недели в виде числа, понедельник - 1, воскресенье - 7.

## **datetime.isocalendar()** - кортеж (год в формате ISO, ISO номер недели, ISO день недели).

## **datetime.isoformat(sep='T')** - красивая строка вида "YYYY-MM-DDTHH:MM:SS.mmmmmm" или, если microsecond == 0, "YYYY-MM-DDTHH:MM:SS"

# **Пример**

## >>> from datetime import datetime, date, time >>> *# Using datetime.combine()* >>> d = date(2005, 7, 14) >>> t = time(12, 30) >>> datetime.combine(d, t) datetime.datetime(2005, 7, 14, 12, 30) >>> *# Using datetime.now() or datetime.utcnow()* >>> datetime.now() datetime.datetime(2007, 12, 6, 16, 29, 43, 79043) # GMT +1 >>> datetime.utcnow() datetime.datetime(2007, 12, 6, 15, 29, 43, 79060) >>> *# Using datetime.strptime()* >>> dt = datetime.strptime("21/11/06 16:30", "*%d*/%m/%y %H:%M") >>> dt datetime.datetime(2006, 11, 21, 16, 30)

# **Пример**

## >>> *# Using datetime.timetuple() to get tuple of all attributes* >>> tt = dt.timetuple() >>> for it in tt: ... print(it) ... 2006 # year 11 # month 21 # day 16 # hour 30 # minute 0 # second 1 # weekday (0 = Monday) 325 # number of days since 1st January -1 # dst - method tzinfo.dst() returned None >>> *# Date in ISO format* >>> ic = dt.isocalendar() >>> for it in ic: ... print(it) ... 2006 # ISO year 47 # ISO week 2 # ISO weekday >>> *# Formatting datetime* >>> dt.strftime("%A, *%d*. %B %Y %I:%M%p") 'Tuesday, 21. November 2006 04:30PM'

# **Еще пример**

## now\_date = datetime.date.today() # Текущая дата (без времени)

## now\_time = datetime.datetime.now() # Текущая дата со временем

## 

## cur\_year = now\_date.year # Год текущий

## cur\_month = now\_date.month # Месяц текущий

## cur\_day = now\_date.day # День текущий

## cur\_hour = now\_time.hour # Час текущий

## cur\_minute = now\_time.minute # Минута текущая

## cur\_second = now\_time.second # Секунда текущие

## num\_week = now\_date.isoweekday() # узнаем номер недели (от 1 до 7)

## 

## now\_date = now\_date.replace(2011,6,30) # меняем полностью дату на 30.06.2011

## now\_date = now\_date.replace(day=cur\_day) # меняем только день

## now\_date = now\_date.replace(month=cur\_month) # меняем только месяц

## now\_date = now\_date.replace(year=cur\_year) # меняем только год

## 

## ny\_2011 = datetime.date(2011,2,1) # создали дату: 1 февраля 2011 года

## delta = ny\_2011 - now\_date # разница (дельта) в между 2-мя датами

## 

## delta = datetime.timedelta(days=2) # дельта в 2 дня

## now\_date = now\_date + delta # Узнаем какое число будет через 2 дня

## now\_date = now\_date - delta # или какое число было 2 дня назад

## 

## print(now\_time.strftime("%d.%m.%Y %I:%M %p")) # форматируем дату

## Наиболее интересные директивы используемые для форматирования времени.

## Расположены не в алфавитном порядке, а в логическом)

## %S — секунды. От 0 до 61

## %M — минуты. От 00 до 59

## %H — час. От 00 до 23

## %I — час. От 1 до 12

## %p -После перед полуднем или после (AM или PM)

## %d — день. От 1 до 31

## %j — день как номер года. От 001 до 366

## %m — месяц. От 01 до 12

## %y — год в виде 2-х последних чисел. От 00 до 99

## %Y — год в виде полного числа

# **Модуль time**

## **Time** - модуль для работы со временем в Python.

## **time.altzone** - смещение DST часового пояса в секундах к западу от нулевого меридиана. Если часовой пояс находится восточнее, смещение отрицательно.

## **time.asctime([t])** - преобразовывает кортеж или struct\_time в строку вида "Thu Sep 27 16:42:37 2012". Если аргумент не указан, используется текущее время.

## **time.clock()** - в Unix, возвращает текущее время. В Windows, возвращает время, прошедшее с момента первого вызова данной функции.

## **time.ctime([сек])** - преобразует время, выраженное в секундах с начала эпохи в строку вида "Thu Sep 27 16:42:37 2012".

## **time.daylight** - не 0, если определено, зимнее время или летнее (DST).

## **time.gmtime([сек])** - преобразует время, выраженное в секундах с начала эпохи в struct\_time, где DST флаг всегда равен нулю.

## **time.localtime([сек])** - как gmtime, но с DST флагом.

## **time.mktime(t)** - преобразует кортеж или struct\_time в число секунд с начала эпохи. Обратна функции time.localtime.

## **time.sleep(сек)** - приостановить выполнение программы на заданное количество секунд.

## **time.strptime(строка [, формат])** - разбор строки, представляющей время в соответствии с форматом. Возвращаемое значение struct\_time. Формат по умолчанию: "%a %b %d %H:%M:%S %Y".

## Класс **time.struct\_time** - тип последовательности значения времени. Имеет интерфейс кортежа. Можно обращаться по индексу или по имени.

## tm\_year

## tm\_mon

## tm\_mday

## tm\_hour

## tm\_min

## tm\_sec

## tm\_wday

## tm\_yday

## tm\_isdst

## **time.time()** - время, выраженное в секундах с начала эпохи.

## **time.timezone** - смещение местного часового пояса в секундах к западу от нулевого меридиана. Если часовой пояс находится восточнее, смещение отрицательно.

## **time.tzname** - кортеж из двух строк: первая - имя DST часового пояса, второй - имя местного часового пояса.

## **Модуль calendar**

## Модуль calendar позволяет напечатать себе календарик (а также содержит некоторые другие полезные функции для работы с календарями).

## **calendar.Calendar(firstweekday=0)** - класс календаря. firstweekday - первый день недели (0 - понедельник, 6 - воскресенье).

## *Методы*:

## **iterweekdays()** - итератор дней недели, начиная с firstweekday.

## **itermonthdates(year, month)** - итератор для месяца month года year. Возвращает все дни этого месяца (как объекты datetime.date), а также дни до и после этого месяца до полной недели.

## **itermonthdays2(year, month)** - как itermonthdates, только дни возвращаются не как datetime.date объекты, а кортежи (номер дня, номер дня недели).

## **itermonthdays(year, month)** - как itermonthdates, только дни возвращаются не как datetime.date объекты, а номера дней.

## **monthdatescalendar(year, month)** - список недель в месяце. Неделя - список из 7 объектов datetime.date.

## **monthdays2calendar(year, month)** - как monthdatescalendar, но объекты - кортежи (номер дня, номер дня недели).

## **monthdayscalendar(year, month)** - как monthdatescalendar, но объекты - номера дней.

## **calendar.TextCalendar(firstweekday=0)** - класс для генерации текстового календаря.

## *Методы:*

## **formatmonth(theyear, themonth, w=0, l=0)** - возвращает календарь на месяц в виде строки, с шириной колонки w и высотой l.

## **prmonth(theyear, themonth, w=0, l=0)** - печатает календарь на месяц.

## **formatyear(theyear, w=2, l=1, c=6, m=3)** - возвращает календарь на год; из m колонок, шириной даты w, высотой недели l и количеством пробелов между месяцами c.

## **pryear(theyear, w=2, l=1, c=6, m=3)** - печатает календарь на год.

## **calendar.HTMLCalendar(firstweekday=0)** - класс для генерации HTML календаря.

## *Методы*:

## **formatmonth(theyear, themonth, withyear=True)** - календарь на месяц в виде HTML таблицы. Если withyear True, номер года будет включен в заголовок.

## **formatyear(theyear, width=3)** - календарь на год в виде HTML таблицы. width - количество месяцев в ряду.

## **formatyearpage(theyear, width=3, css="calendar.css", encoding=None)** - календарь на год в виде полноценной HTML страницы, с подключением файла css (который вы можете создать сами), и в кодировке encoding.

## **calendar.LocaleTextCalendar(firstweekday=0, locale=None)** - позволяет создать текстовый календарь с названиями на родном языке.

## calendar.LocaleHTMLCalendar(firstweekday=0, locale=None) - позволяет создать HTML календарь с названиями на родном языке.

## Например, вот такой календарик получился у меня:

## import calendar a = calendar.LocaleHTMLCalendar(locale='Russian\_Russia') with open('calendar.html', 'w') as g: print(a.formatyear(2014, width=4), file=g)

## Также модуль calendar предоставляет несколько полезных функций:

## **calendar.setfirstweekday(weekday)** - устанавливает первый день недели (0 - понедельник, 6 - воскресенье). Также предоставлены значения calendar.MONDAY, calendar.TUESDAY, calendar.WEDNESDAY, calendar.THURSDAY, calendar.FRIDAY, calendar.SATURDAY и calendar.SUNDAY.

## **calendar.firstweekday()** - возвращает первый день недели.

## **calendar.isleap(year)** - является ли год високосным.

## **calendar.leapdays(y1, y2)** - количество високосных лет в последовательности от y1 до y2.

## **calendar.weekday(year, month, day)** - день недели для этой даты.

## **calendar.monthrange(year, month)** - день недели первого дня месяца и количество дней в этом месяце.

# **Задания к самостоятельной работе**

1. Создаём переменную формата даты из числовой, такой как 19901204
2. Создаём переменную формата даты из 3 числовых переменных, содержащих день, месяц и год
3. Переводим строчную переменную в формат даты
4. Отнять от даты на текущей даты месяц сохранить в переменную и потом к этой переменной добавить один год и определить какой это по счету день
5. По заданному числу n от 1 до 365 определите, на какое число какого месяца приходится день с номером n.
6. Решите обратную предыдущей задачу: по дате текущего дня определите номер дня в году.
7. Дата задана в формате dd.mm.yyyy. Выведите ее в формате "Month d, y", где Month - английское название месяца, d - номер дня в месяце, без лидирующих нулей, y - номер года без лидирующих нулей

# **Список использованной литературы**

* <https://ru.wikipedia.org/wiki/%D0%A7%D0%B0%D1%81%D0%BE%D0%B2%D0%BE%D0%B9_%D0%BF%D0%BE%D1%8F%D1%81>
* работа с датой
  + <http://websiteprojector.blogspot.com/2010/11/python.html>
  + <https://borisnote.wordpress.com/2010/11/16/python-date-tim/>
  + <http://ligh.te.ua/forum/?id=1189>
* datetime
  + <http://pythonworld.ru/moduli/modul-datetime.html>
  + <http://admin-ltd.ru/2015/01/31/235/>
* time
  + <http://pythonworld.ru/moduli/modul-time.html>
* calendar
  + <http://pythonworld.ru/moduli/modul-calendar.html>