Язык программирования Питон (Python)

Начало работы

PEP8 (стилистические рекомендации по оформлению кода)

- отступ – 4 пробела
- длина строки < 80 символов
- переменные: `var_recommended`
- константы: `CONST_RECOMMENDED`

Каждый модуль пограммы на языке Python представляет собой текстовый файл в ***кодировке***, совместимой с 7-битной кодировкой ASCII.

Для кодировок, использующих старший бит, необходимо явно указывать название кодировки. Например, модуль, комментарии или строковые литералы которого записаны в кодировке KOI8-R, должен иметь в первой или второй строке следующую спецификацию:

```
# -*- coding: koi8-r -*-
```

Начало работы в *Spyder* - the Scientific PYthon Development EnviRonment

1. Создадим в рабочей папке программный файл с именем `hello.py`

```
print('\n','Привет, Мир!')      # print -- это функция
```

– *регистр букв в командах различается*
– *если перед первым символом в строке вставить пробел – возникнет ошибка*
– *текст программы говорит о том, КАК, а комментарии должны объяснять, ПОЧЕМУ*

2. Использование метода **`format`** (файл `str_format.py`)

Пример:

```
age = 26
name = 'Ivan'
rost = 1.76

print('')
print('Возраст {0} лет -- Имя {1} -- Рост {2} м.' .format(age,
                                                    name, rost))
# Иначе
print('Возраст ' + str(age) + ' лет -- Имя ' + name +
      ' -- Рост ' + str(rost) + ' м.')
```

3. Организации ***ввода данных*** с клавиатуры. Функция `input()`

– в качестве ***аргумента*** рекомендуется использовать ***строку-подсказку*** в двойных или в одиночных кавычках;

– интерпретатор останавливает программу и после строки-подсказки требуется ввести требуемое значение переменной и нажать <ENTER>;

– данные ***возвращаются в виде строки***, даже если было введено число;

– если требуется ***получить число***, то результат выполнения функции `input()` изменяют с помощью функций `int()` или `float()`

Пример:

```
a = input('Введите длину основания треугольника (мм): ')
h = input('Введите значение высоты (мм): ')
a = float(a)
h = float(h)
s = a*h/2
print('\n\nПлощадь треугольника S = ', str(s), 'мм.кв')
```

4. ***Чтение*** из файла и ***запись*** в файл

(самый простой вариант – работа с ***текстовыми файлами в текущем каталоге без указания формата данных***)

> *Для работы с файлом прежде всего нужно создать специальный объект – «**дескриптор файла**», а потом использовать методы этого дескриптора для чтения и записи данных.*

При создании дескриптора нужно указать:

– строку с именем файла;

– строку с описанием варианта доступа: `'r'` – чтение;
`'w'` – запись;
`'a'` – дополнение.

Чтение *возможно только из существующего файла.*

*Если при открытии на **запись** или **дополнение** указано имя несуществующего файла, он будет создан.*

Пример:

```
str1 = """ В мире не происходит ничего, в чём не был бы
           виден смысл какого-нибудь максимума или минимума. """

str2 = '\n\t\t\t\t\t\t   Л. Эйлер'

fd = open('Test_data.dat', 'w')
fd.write(str1)                              # Запись в файл
fd.close()
```

```
fd = open('Test_data.dat', 'r')
strk = fd.read()                              # Чтение из файла
fd.close()
print(strk)

fd = open('Test_data.dat', 'a')
fd.write(str2)                                # Добавление в файл
fd.close()

fd = open('Test_data.dat', 'r')
strk = fd.read()                              # Чтение из файла
fd.close()
print('\n', strk)

fd = open('Test_data.dat', 'r')
strk = fd.readline()                          # Чтение из файл построчно
print('\n', 'строка 1:', strk)
strk = fd.readline()
print('\n', 'строка 2:', strk)
strk = fd.readline()
print('\n', 'строка 3:', strk)
fd.close()

fd = open('Test_data.dat', 'r')
lst = fd.readlines()                          # Формирование списка строк
print('\n', 'Количество строк:', len(lst))
fd.close()
```

Логическая строка – это то, что Python видит как единое предложение.
Python неявно предполагает, что каждой физической строке соответствует логическая строка.

– *чтобы записать более одной логической строки на одной физической – используют точки с запятой:*

```
i = 5; print(i)
```

– *для написания одной логической строки, занимающей несколько физических строк:*

1. *явное объединение строк (набирать в консоли):*

```
s = 'Это строка. \
Это строка продолжается.'
print(s)
```

2. *неявное объединение строк – когда в логической строке есть открывающаяся круглая, квадратная или фигурная скобка, но нет закрывающейся.*

```
print(" \n Это - очень длинная строка, которая не помещается в ",
       str(80), 'знакоместах')
```

Отступы

Пробелы в начале логической строки используются для группировки предложений. Предложения, идущие вместе, должны иметь одинаковый отступ. Каждый такой набор предложений называется ***блоком***.

Не смешивайте пробелы и символы табуляции в отступах, поскольку не на всех платформах это работает корректно!!!

Получение помощи

help(print) – покажет справку по функции **print**

help('return') – если интересует информация об операторах, их необходимо указывать в кавычках

print? – быстрый доступ к строке документации

– *Создадим функцию* **square(a)**

```
def square(a):
    """Return the square of a."""
    return a ** 2
```

help(square) – вызов справочной строки

```
Help on function square in module __main__:
square(a)
    Return the square of a.
```

square?

```
Signature: square(a)
Docstring: Return the square of a.
File:      d:\___tekrabota\_python\python_work\untitled0.py
Type:      function
```

`square??` **– обращение к исходному коду**

(если объект реализован не на языке Python, а на C или каком-либо другом транслируемом языке, добавление ?? приводит к такому же результату, что и добавление ?)

```
Signature: square(a)
Source:
def square(a):
    """Return the square of a."""
    return a ** 2
File:      d:\___tekrabota\_python\python_work\untitled0.py
Type:      function
```

– *Создадим список:*

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

Списки – упорядоченные изменяемые коллекции объектов произвольных типов (почти как массив, но типы элементов могут отличаться)

Python рассматривает всё, что есть в программе, как ***объекты***, которые имеют свои ***атрибуты*** и ***методы***.

`dir(L)` – возвращает список **атрибутов и методов**

```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__',
'__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
'__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count',
'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

L?

```
Type:        list
String form: [1, 2, 3]
Length:      3
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items
```

L.insert?

```
Docstring: L.insert(index, object) -- insert object before index
Type:      builtin_function_or_method
```

L.<TAB> – интерфейс **Tab-автодополнения**

L.append L.copy L.extend L.insert L.remove L.sort
L.clear L.count L.index L.pop L.reverse

L.c<TAB>

L.clear L.copy L.count

from itertools import co<TAB>

combinations *compress*
combinations_with_replacement *count*

***Warning?** – вывод списка всех имён объектов, заканчивающихся словом Warning

str.*find*?

```
str.find
str.rfind
```

Объекты `In` и `Out` оболочки

Вводы и выводы отображаются в командной оболочке с метками **`In/Out`**.
При этом создаются переменные языка Python с именами **`In`** и **`Out`**, автоматически обновляемые так, что они отражают историю:

```
In [1]: import math
In [2]: math.sin(2)
Out[2]: 0.9092974268256817
In [3]: math.cos(2)
Out[3]: -0.4161468365471424

In [4]: print(In)
['', 'import math', 'math.sin(2)', 'math.cos(2)', 'print(In)']

In [5]: Out
Out[5]: {2: 0.9092974268256817, 3: -0.4161468365471424}

In [8]: Out[2] ** 2 + Out[3] ** 2
Out[8]: 1.0
```

`print(_)` – значение переменной _ (одиночн.симв.подч.) соотв. **предыдущ.выводу**

`math.sin(2);` – способ **подавления вывода** команды — точка с запятой в конце строки

Запуск python-программ из командной строки Windows

При подключении библиотек (numpy и др.) возникает ошибка ImportError.
Можно использовать Anaconda Prompt

1. Открыть командное окно Windows:

– Пуск (прав.кн.мыши) / Выполнить / **cmd**

2. Запуск программы **str_format.py** , когда не установлена должным образом системная переменная PATH

```
c:\ProgramData\Anaconda3\python  d:\___TekRabota\_PYTHON\python_work\str_format.py
```

3. Запуск программы, если путь к python.exe добавлен в системные переменные Windows

```
python  d:\___TekRabota\_PYTHON\python_work\str_format.py
```

4. Запуск программы, если рабочая директория является текущей

```
python  str_format.py
```

– *Вызов справки*

```
C:\Users\Сергей>help
C:\Users\Сергей>help cd
```

– *Смена текущей директории:*

```
C:\Users\Сергей>cd /d D:\___TekRabota\_PYTHON\python_work
D:\___TekRabota\_PYTHON\python_work>
```

Для принудительного завершения работы программы нажмите ***Ctrl-Z****, а затем клавишу* ***Enter***