Экспресс введение в Python 3.6

Муромцев Никита Андреевич
Выпускник ВМК МГУ,
аспирант геологического факультета МГУ
2017 год

Занятие 2

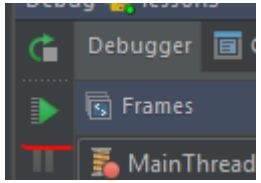
Строки, регулярные выражения, списки, кортежи, словари, множества, собственные модули, работа с файлами

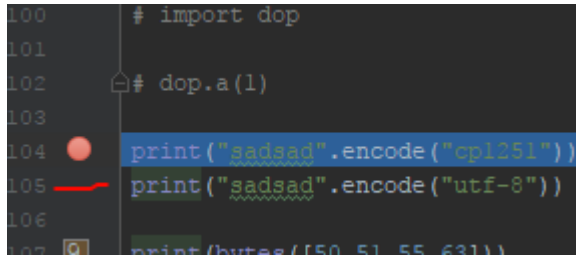
Debug режим (отладка, профилирование)

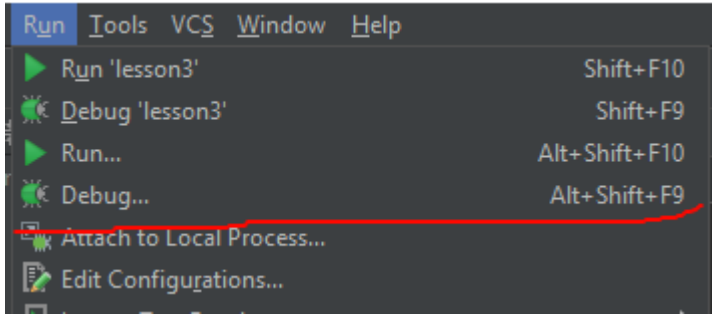
Для использования необходимо сторонуее ПО, к примеру IDE PyCharm

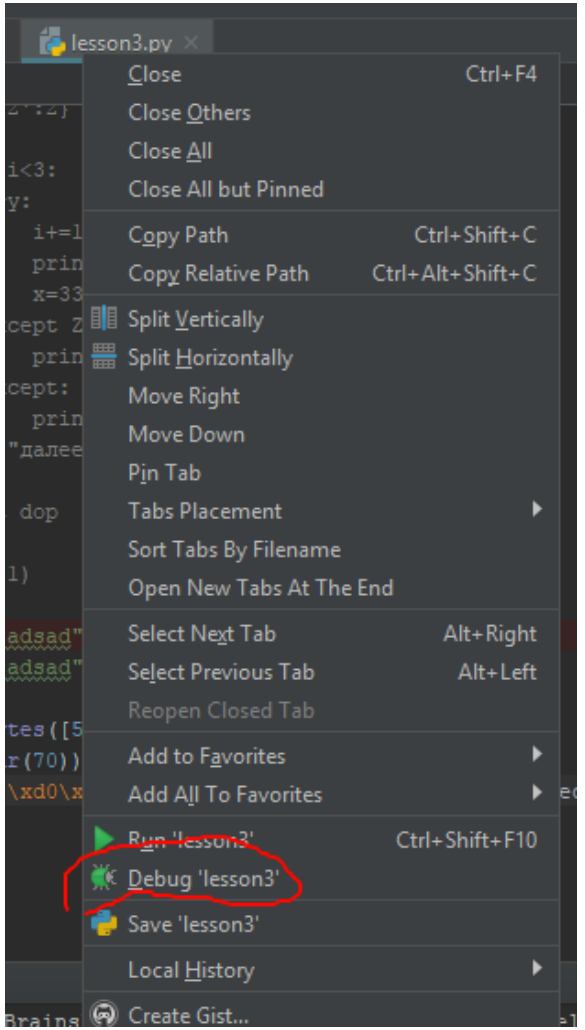
В PyCharm несколько вариантов запуска,
три из них на скриншотах.

- В данном режиме можно пошагово выполнить все инструкции программы с разной степенью детализации.
- На всех шагах можно посмотреть текущее значение переменных.
- Можно устанавливать стоп-точки, чтобы быстро выполнить инструкции до точки останова (без пошагового прохода).









Строки

Строки в апострофах и кавычках - одно и то же

Строка в python 3 – список символов.

'spam"s' и "spam's" - верно
"spam"s" - ошибка

Экранированные последовательности:

Есть служебные комбинации, которые преобразовываются в другие символы:

- \n – перевод строки
- \t – горизонтальная табуляция
- \v – вертикальная табуляция
- Есть и другие символы (используются сильно реже)

Для отключения экранирования перед строкой пишем r, к примеру: r‘text'

Можно писать большие блоки текста в тройных апострофах

```
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 - операции

Различных функций очень много, вот наиболее популярные

Сложение (объединение)	'test' + 'quest' = 'testquest'
Дублирование	'test' * 3 = 'testtesttest'
Длина	len('test‘) = 4
Индексация и срез	'test'[2] = 's'; 'test'[0:1:-1] = 'et'
Привести операнд в строковый тип	str(x)
Поиск подстроки	str.find(temp, st, end); str.index(temp, st, end)
Разделение строки (результат – список)	str.slpit('разделитель')
Объединить элементы списка в строку	'символ для промежутков'.join(список)
Удаление незначащих пробелов	str.strip()

Строки - форматирование

Часто требуется вывести большое количество данных в читаемом виде

```
# Вставить символ (по порядку)
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'

# Вставить символ в личном порядке
>>> '{0}, {1}, {0}'.format('a', 'b', 'c')
'a, b, a'

# Для передачи списка в качестве аргумента - '*'
>>> c = [1,2,3]
>>> '{2}, {1}'.format(*c)
'3 2'
# Для передачи словаря использовать '**'

# Возможности адрессации
>>> '{second[1]}, {first}'.format(first = 'a', second = ['b','c'])
'c, a'


# Выравнивание - символы '>', '<', '^', '='
# Символ заполнителя до символа выравнивания (в примере '*')
>>> '{:<30}'.format('left aligned')
'left aligned                  '
>>> '{:>30}'.format('right aligned')
'                 right aligned'
>>> '{:*^30}'.format('centered')
'***********centered***********'
```

```
# Использование знака:
# '+' - все числа
# '-' - только у отрицательных
# ' ' - минусы и пробелы для положительных
>>> '{:+f}; {:+f}'.format(1, -1)
'+1; -'
>>> '{:+f}; {:+f}'.format(1, -1)
' 1; -1'
>>> '{:+f}; {:+f}'.format(1, -1)
'1; -1'

# Вывод в разных форматах, в том числе с префиксом (через '#')
>>> "int: {0:d};  hex: {0:x}; hex: {0:#x};  oct: {0:#o};  bin: {0:b}".format(42)
'int: 42;  hex: 2a; hex: 0x2a;  oct: 0o52;  bin: 101010'
# Точность
>>> '{0:.3f}'.format(75.765367)
'75.765'
# Отступ
>>> 'aaa{0:4}aaa {1:5}'.format(3,"kkk")
'aaa   4aaa kkk  '
# Есть и другие

# Проценты
>>> '{:.2%}'.format(34/67)
'50.75%'

#Возможны комбинации, пример
>>> '{:*>+#30.5}'.format(34.34)
'**********************-34.340'
```

Для комбинаций используется следующая грамматика (в [] – элементы, которые могут указываться 0-1 раз):
"{[имя поля] [!преобразование (r или s)] [:[[символ заполнения]символ выравнивания (>, <, =, ^)][знаковость (пробел, +, -)][#][0][ширина поля (знаков)][,][.точность (обратите внимание на точку, указывается число символов на числе конкретно)][тип (b, c, d, e, E, f, F, g, G, n, o, s, x, X, %)]]}"

Регулярные выражения

Зная структуру строки, её проще преобразовывать подобными выражениями

import re # не забыть

```
# Поиск совпадений по шаблону с начала строки, возвращает элемент класса, содержимое
выводим через group
result = re.match(r'AV', 'AV Analytics Vidhya AV')
result.group(0) # = AV
result.start() # = 0
result.end() # = 2
# Если будем искать 'Analytics', то не будет ничего найдено
# Поиск '.*Analytics' даст результат
re.match(r'.*Analytics', 'AV Analytics Vidhya AV').group(0)
# = AV Analytics

# Есть метод search, который ищет не с начала строки, он найдёт 'Analytics'

# Поиск всех вхождений
result = re.findall(r'AV', 'AV Analytics Vidhya AV')
result # = ['AV', 'AV']

# re.split(шаблон, строка, [максимальное кол-во разделений=0]) - разделяет по шаблону

# re.sub(шаблон, на что заменять,строка) - делает замены шаблона
re.sub(r'India', 'the World', 'AV is largest Analytics community of India')
# = 'AV is largest Analytics community of the World'
```

Примеры использования:

```
re.findall(r'\w', 'AV is largest')
# ['A', 'V', 'i', 's', 'l', 'a', 'r', 'g', 'e', 's', 't']

result = re.findall(r'\w*', 'AV is largest')
# ['AV', '', 'is', '', 'largest']

result = re.findall(r'\w+', 'AV is largest') # ['AV', 'is', 'largest']
result = re.findall(r'^\w+', 'AV is largest') # ['AV']
re.findall(r'\w+$', 'AV is largest') # ['largest']
re.findall(r'\w\w', 'AV is largest') # ['AV', 'is', 'la', 'rg', 'es']
re.findall(r'\b\w.', 'AV is largest') # ['AV', 'is', 'la']
```

Шпаргалка для составления шаблонов:

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

Списки

Самый популярный элемент

list(элемент) – преобразовать в список

s = [] – создать пустой список

l = [‘s’, [‘a’, 1, 2], ’o’, 5] – список со списком

l[2] # = [‘a’, 1, 2]

l[2][1] # = 1

Генераторы списков:

```
[c * 3 for c in 'list'] # = ['lll', 'iii', 'sss', 'ttt']
[c + d for c in 'list' if c != 'i' for d in 'spam' if d != 'a']
# ['ls', 'lp', 'lm', 'ss', 'sp', 'sm', 'ts', 'tp', 'tm']
[‘a’]*3 # [‘a’, ‘a’, ‘a’]
```

Важно помнить, что операции производятся над самими списками и не возвращают новый элемент:

```
a = [1, 2 ,3]; b = a ; b[1] = 5
# поменяет и значение в a, т.к. переменные хранят лишь указатель
```

Основные методы:

list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Добавить в конец все элементы списка L
list.insert(i, x)	Вставляет на i-ую позицию x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует
list.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
list.count(x)	Возвращает количество элементов со значением x
list.sort([key=функция])	Сортирует список, может быть использована стороння функция возвращающая значения, которые использовать для сортировки
list.reverse()	Разворачивает список
list.copy()	Поверхностная копия списка
list.clear()	Очищает список

Словарь

Удобно для структурированной работы с большим объёмом данных

Создать словарь:

```
d = {'dict': 1, 'dictionary': 2}
d = dict(dict=1, dictionary=2)
d = dict([('dict', 1), ('dictionary', 2)])
d = dict.fromkeys(['a', 'b'], 100)
# {'a': 100, 'b': 100}
d = {a: a ** 2 for a in range(7)}
# {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}

d = {1: 2, 2: 4, 3: 9}
d[1] # = 2, так обращаемся по ключу
d[4] = 4 ** 2
```

так можно поменять элемент, если его нет, создастся новый

Методы словарей

dict.clear()	очищает словарь.
dict.copy()	возвращает копию словаря.
dict.get(key [, default])	возвращает значение ключа, если его нет default (по умолчанию None).
dict.items()	возвращает пары (ключ, значение).
dict.keys()	возвращает ключи в словаре.
dict.pop(key [, default])	удаляет ключ и возвращает значение. Если ключа нет, возвращает указанный default / исключение.
dict.popitem()	удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. !!! Словари неупорядочены.
dict.setdefault(key[, default])	возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ с значением default (по умолчанию None).
dict.update([other])	добавляет пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).
dict.values()	возвращает значения в словаре.

Кортежи (tuple)

Неизменяемые списки занимают меньше места в памяти и могут быть использованы в качестве ключей словаря

Создать кортеж:

```
t = ()
t = tuple()
t = ('s') # получится строка
t = ('s', ) # получится кортеж
t = 's', # тоже кортеж
```

Операции над списками, не изменяющие список применимы и к кортежам

Благодаря кортежам возможна операция:

```
a, b = b, a
```

Множества

Список из неповторяющихся элементов

Создать множество:

```
s = set()
s = {1, 2, 3, 4}
s = {i ** 2 for i in range(10)}
# {0, 1, 4, 81, 64, 9, 16, 49, 25, 36}
s = frozenset() # неизменяемое множество
```

Операции, меняющие множество

set.update(other, ...); set |= other | ...	объединение
set.intersection_update(other, ...); set &= other & ...	пересечение
set.difference_update(other, ...); set -= other | ...	вычитание
set.add(elem)	добавляет элемент в множество
set.remove(elem)	удаляет элемент из множества
set.discard(elem)	удаляет элемент, если он находится в множестве
set.pop()	удаляет и возвращает некоторый элемент множества
set.clear()	очистить

Операции, возвращающие результат

len(s)	размер множества
x in s	принадлежность к множеству
set.isdisjoint(other)	истина, если set и other не имеют общих элементов
set == other	множества идентичны
set.issubset(other) или set <= other	все элементы set принадлежат other
set.issuperset(other) или set >= other	аналогично
set.union(other, ...) или set | other | ...	объединение нескольких множеств
set.intersection(other, ...) или set & other & ...	пересечение
set.difference(other, ...) или set - other - ...	множество из всех элементов set, не принадлежащие ни одному из other
set.symmetric_differenc e(other); set ^ other	множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих
set.copy()	копия множества
set.symmetric_differenc e_update(other); set ^= other	множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих

Файлы

Открыть файл (связать с переменной):
f = open(путь до файла, режим)
Путь до файла: 'test.txt', 'folder//test.txt', 'C://folder//test.txt'

Режимы:

'r'	чтение (по умолчанию).
'w'	запись, перезапись
'x'	запись, ошибка, если нет
'a'	запись, дополнение
'+'	запись и чтение

Режимы работы:

'b'	двоичный режим
't'	текстовый (по умолчанию).

В текстовый файл можно писать только текст
В двоичный файл можно писать структуры

После работы с файлом его следует закрыть (особенно при записи):
f.close()

Читаем с помощью read()
Пишем с помощью write(строка/текст)

```
>>> f = open('text.txt')
>>> f.read(1)
'H'

>>> f.read()
'ello world!\nThe end.\n\n'
>>> f.close()
>>> f = open('text.txt')
>>> for line in f:
...     line

>>> for i in [1,2,3,4,5]:
...     f.write(str(i) + '\n')
```

Задачи для практики по занятию 2

Список задач опубликован по ссылке:
https://docs.google.com/spreadsheets/d/1d5UjabePXB9QpW2oTY6QtUkjn7NDMgBMvQzpkW_ccZ0/edit#gid=258125635

На соседней есть результаты работы нашей группы:
https://docs.google.com/spreadsheets/d/1d5UjabePXB9QpW2oTY6QtUkjn7NDMgBMvQzpkW_ccZ0/edit#gid=0

Времени на решение всех задач 1, 2 и 3 блока + задачи по моделированию у вас до конца семестра.

Для допуска к сдачи задачи по моделированию необходимо решить 45 основных задач и набрать не менее 60 баллов.