

Лекция 8

Язык программирования Python.

Хайрулин Сергей Сергеевич

email: s.khairulin@g.nsu.ru, s.khayrulin@gmail.com

Ссылка на [материалы](#)

План

- Лекции/практические занятия
 - Тест
- Дифференцированный зачет в конце семестра
 - Защита задания

Литература

Начальный уровень

- Mark Pilgrim. Dive into Python - <http://www.diveintopython.net/>
- Марк Лутц. Изучаем Python, 4-е издание // Символ-Плюс 2011.
- ...

Стандарт/Документация

- PEP-8 - <https://www.python.org/dev/peps/pep-0008/>
- <https://www.python.org/>
- <https://github.com/python/cpython>

Экспертный уровень

- Лучано Рамальо: Python. К вершинам мастерства
- Mitchell L. Model. Bioinformatics Programming Using Python // O'Reilly 2010.

Версии Python

- Python 2 вышел 2010 году последняя версия 2.7.16 - исправлялись только баги(ошибки) с января 2020 года поддержка прекращена.
- Python 3 появился в 2008, является актуальной версией языка.
Текущая стабильная версия 3.9, в разработке 3.10
 - Python 3 не гарантирует совместимости кода с Python 2

Summary

- Организация кода
 - Модули
 - Пакеты, файл `__init__.py`
 - Зависимости:
 - Ключевое слово `import`
 - Конструкция `from ... import ...`
 - `PYTHONPATH`
 - Точка входа в приложение
- работа с файлами

Организация кода

С ростом сложности приложения возникает необходимость упорядочивания кода. Уровни абстракции кода:

1. Выделение повторяющиеся куски кода в циклы
2. Выделение функций, определенный набор операция реализующий некоторую последовательность вычислений или часть алгоритма, зависящая от входных данных
3. Организация функций в отдельные модули (обычно файлы.)
4. Определение абстрактных структур данных.

Плюсы

Абстрагированный от остального кода модули и пакеты дают возможность писать отделяемый код, который может зависеть только от входящих данных. Кроме того на основе этих пакетов и модулей можно писать переносимые библиотеки, и соответственно, переиспользовать уже имеющуюся кодовую базу.

Организация кода

В python иерархия организации кода выглядит следующим образом

1. Пакет
2. Модуль
3. Функция/глобальная переменная/классы

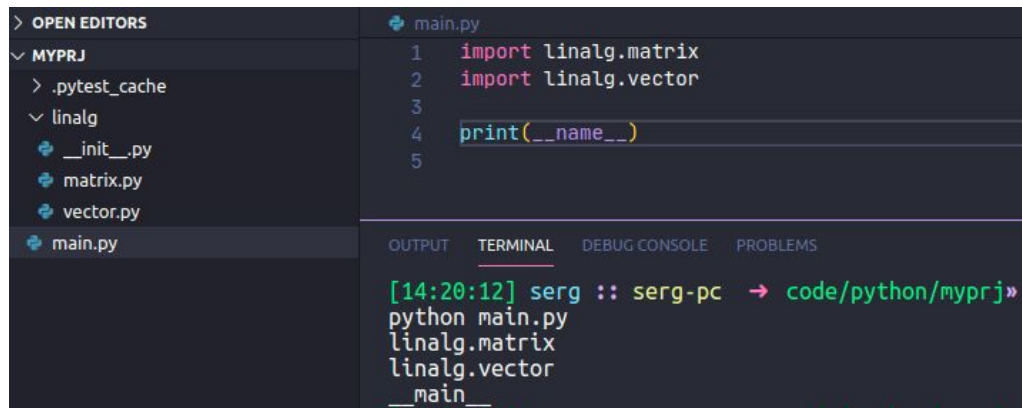
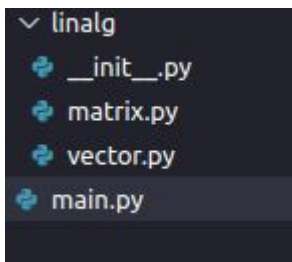
Модуль

Модуль - неделимую сущность объединяющий в себе некоторый набор переменных, функций, классов...

Если говорить более неформально, то модуль - это файл с расширением **.py**. В модуле определяется некоторая часть логики характерная только для этого модуля, это не строгое требование, **НО** было бы очень здорово если бы вы придерживались этого правила.

Модуль (глобальная переменная `__name__`)

A module is a file containing Python definitions and statements. The file name is the module name with the suffix `.py` appended. Within a module, the module's name (as a string) is available as the value of the global variable `__name__`. (<https://docs.python.org/3/tutorial/modules.html>)



Функция dir

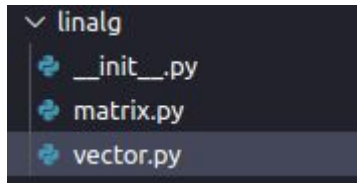
[dir\(\[objects\]\)](#) - встроенная функция, которая позволяет узнать все доступные внутри объекта (в том числе и в модуле) функции, классы, атрибуты, методы и так далее... Если в качестве аргумента передать объект, то функция выводит список всех доступных подсущностей объектов

```
>>> import struct
>>> dir() # show the names in the module namespace
['__builtin__', '__name__', 'struct']
>>> dir(struct) # show the names in the struct module
['Struct', '__all__', '__builtin__', '__cached__', '__doc__', '__file__',
 '__initializing__', '__loader__', '__name__', '__package__',
 '__clearcache__', 'calcsize', 'error', 'pack', 'pack_into',
 'unpack', 'unpack_from']
>>> class Shape:
...     def __dir__(self):
...         return ['area', 'perimeter', 'location']
>>> s = Shape()
>>> dir(s)
['area', 'location', 'perimeter']
```

Пакеты

Множество всех модулей в пределах одной папки называется пакетом

Packages are a way of structuring Python's module namespace by using “dotted module names”. For example, the module name `A.B` designates a submodule named `B` in a package named `A`. Just like the use of modules saves the authors of different modules from having to worry about each other's global variable names, the use of dotted module names saves the authors of multi-module packages like NumPy or Pillow from having to worry about each other's module names. (<https://docs.python.org/3/tutorial/modules.html#packages>)



Пакеты (__init__.py)

```
sound/                                Top-level package
  __init__.py                         Initialize the sound package
  formats/                           Subpackage for file format conversions
    __init__.py
    wavread.py
    wavwrite.py
    aiffread.py
    aiffwrite.py
    auread.py
    auwrite.py
    ...
  effects/                           Subpackage for sound effects
    __init__.py
    echo.py
    surround.py
    reverse.py
    ...
  filters/                           Subpackage for filters
    __init__.py
    equalizer.py
    vocoder.py
    karaoke.py
    ...
```

Файл `__init__.py`

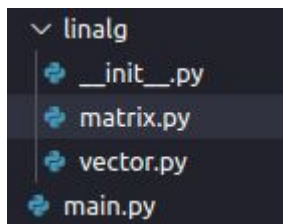
Для того чтобы дать понять интерпретатору, что папка является пакетом, нужно добавить в папку файл `__init__.py`. Обычно этот файл не содержит никакой бизнес логики приложения и является пустым.

Зависимости: import

Для того чтобы воспользоваться функциями определенными в некотором пакете можно импортировать их в модуль где вы собираетесь их использовать.

После импортирования модуля или пакета. У вас появляется возможность пользоваться всеми объектам представленными в модуле. Доступ предоставляется через операто “.”

Зависимости: import



```
main.py  __init__.py  vector.py  X
linalg > vector.py > v_sum
1 def v_sum(v1, v2):
2     if len(v1) != len(v2):
3         return None
4     result = []
5
6     for i in range(len(v1)):
7         result.append(v1[i] + v2[i])
8
9     return result
10
```

```
main.py > ...
1 import linalg.matrix
2 import linalg.vector
3
4 v1 = [1, 2, 3]
5 v2 = [4, 5, 6]
6 print(linalg.vector.v_sum(v1, v2))
7
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS
[16:10:54] serg :: serg-pc → code/py
python main.py
[5, 7, 9]
```

Зависимости: `from ... import ...`

Запись импорта `from package_name....module_name import functions, class, var,....`
Эта запись импортирует непосредственные функции, классы, модули и так далее \Rightarrow у вас отпадает необходимость обращаться к импортируемой сущности через операцию “.”. Обратите внимание, что в таком случае у вас могут появиться коллизии - ситуации, когда в двух разных пакетах, определяются сущности с одинаковым названием, например функции

Зависимости: from ... import ...

```
linalg > vector.py > ...
1  def v_sum(v1, v2):
2      if len(v1) != len(v2):
3          return None
4      result = []
5
6      for i in range(len(v1)):
7          result.append(v1[i] + v2[i])
8
9      return result
10
11
12  def scalar_mult(scalar, v):
13      result = []
14      for i in range(len(v)):
15          result.append(scalar * v[i])
16
17      return result
```

```
main.py > ...
1  import linalg.matrix
2
3  from linalg.vector import v_sum, scalar_mult
4
5  v1 = [1, 2, 3]
6  v2 = [4, 5, 6]
7  print(v_sum(v1, v2))
8  print(scalar_mult(2, v1))
9
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS
[16:14:48] serg :: serg-pc → code/python/myprj»
python main.py
[5, 7, 9]
[2, 4, 6]
```

Глобальная переменная `__all__`


`__all__` - встроенная глобальная переменная (относительно модуля), которая позволяет ограничивать список сущностей экспортируемых с помощью конструкции

```
from package_name import *
```

Что дословно означает импортировать все что есть в модуле `package_name`

Поиск зависимостей

При запуске программы Python начинает интерпретировать код сверху вниз (это вы уже знаете). И соответственно чем раньше вы проимпортируете те зависимости, которые вам нужны, тем раньше они попадут в контекст программы и, соответственно, вы сможете ими пользоваться. После объявления `import *` → интерпретатор ищет модуль обход начинается

1. С текущего каталога - поиск во всех дочерних подкаталогах
2. Если не успех - поиск среди установленных библиотек (обычно они располагаются рядом с интерпретатором)
3. Поиск среди в стандартной библиотеке
4. Поиск среди папок перечисленных в переменной окружения PYTHONPATH
5. 

PyPi

Самый большой репозиторий библиотек для python - прежде чем пытаться изобрести свой велосипед, нужно проверить не изобрел ли уже его кто-нибудь еще. Обычно для установке библиотеки нужно воспользоваться менеджером пакетов `pip` - который идет вместе с установщиком Python

Точка входа в приложение

C/C++

```
1 int main(int argc, char** argv){  
2     ...  
3 }
```

Java

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         System.out.println("Hello World!");  
4     }  
5 }
```

Go

```
1 func main(){  
2     ...  
3 }
```

Точка входа в приложение

Python выполняет код последовательно.

```
vector.py > ...  
  
def v_sum(v1, v2):  
    if len(v1) != len(v2):  
        return None  
    result = []  
  
    for i in range(len(v1)):  
        result.append(v1[i] + v2[i])  
  
    return result  
  
def scalar_mult(scalar, v):  
    result = []  
    for i in range(len(v)):  
        result.append(scalar * v[i])  
  
    return result  
  
v1 = [1, 2, 3]  
v2 = [4, 5, 6]  
print(v_sum(v1, v2))  
print(scalar_mult(2, v1))
```

```
main.py x  __init__.py  vector.py  matrix.py  func main(){ Untitled-1 ●  
  
main.py > ...  
1  import linalg.matrix  
2  
3  from linalg.vector import v_sum, scalar_mult  
4  
5  v = list(map(float, input("provide vector separate with coma: ")))  
6  scalar = input("provide scalar value (should be number): ")  
7  print(scalar_mult(scalar, v))  
8  
  
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS  
  
[16:53:50] serg :: serg-pc → code/python/myprj»  
python main.py  
[5, 7, 9]  
[2, 4, 6]  
provide vector separate with coma: 1,2,3  
provide scalar value (should be number): 22  
[22.0, 44.0, 66.0]
```


Точка входа в приложение

```
1 def v_sum(v1, v2):
2     if len(v1) != len(v2):
3         return None
4     result = []
5
6     for i in range(len(v1)):
7         result.append(v1[i] + v2[i])
8
9     return result
10
11
12 def scalar_mult(scalar, v):
13     result = []
14     for i in range(len(v)):
15         result.append(scalar * v[i])
16
17     return result
18
19
20 if __name__ == '__main__':
21     v1 = [1, 2, 3]
22     v2 = [4, 5, 6]
23     print(v_sum(v1, v2))
24     print(scalar_mult(2, v1))
```

main.py > ...

```
1 import linalg.matrix
2
3 from linalg.vector import v_sum, scalar_mult
4
5 if __name__ == '__main__':
6     v = list(map(float, input("provide vector separate with coma: ")))
7     scalar = input("provide scalar value (should be number): ")
8     print(scalar_mult(scalar, v))
9
```

Работа с файлами

```
lg > vector.py > ...
1 def v_sum(v1, v2):
2     if len(v1) != len(v2):
3         return None
4     result = []
5
6     for i in range(len(v1)):
7         result.append(v1[i] + v2[i])
8
9     return result
10
11 def scalar_mult(scalar, v):
12     result = []
13     for i in range(len(v)):
14         result.append(scalar * v[i])
15
16     return result
17
18 def load_vector_from_file(f_name):
19     v = []
20     with open(f_name, "r") as f:
21         for l in f:
22             v.append(float(l))
23     return v
```

```
main.py > ...
1 import linalg.matrix
2
3 from linalg.vector import v_sum, scalar_mult, load_vector_from_file
4
5
6 if __name__ == '__main__':
7     f_list = input("Provide file name list: ").split(",")
8     v1 = load_vector_from_file(f_list[0])
9     v2 = load_vector_from_file(f_list[1])
10    print(v_sum(v1, v2))
11
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[17:18:01] serg :: serg-pc → code/python/myprj»
python3 main.py
Provide file name list: 1.vec,2.vec
[5.0, 7.0, 9.0]
[17:19:05] serg :: serg-pc → code/python/myprj»
```

Работа с файлами

Функция [open\(path_to_file, mode\)](#) - открытие файла

`path_to_file` - путь до файла подходит как полный, так и относительный

`mode` - может быть:

- **w** - на запись в этом случае содержимое файла обнулиться,
- **r** - на чтение,
- **r+** - открывает файл и для чтения и записи,
- **a** - на запись в конец файла

Пути

- Относительный - то есть относительно текущей директории
 - . - текущий каталог
 - .. - родительский каталог
- Полные /a/b/c/d/file_name.file_ext

Чтение

Для того чтобы прочитать из файла данные нужно открыть файловый дескриптор на чтение. Считать можно с помощью функции

1. `read` - считываем все что есть в файле в память.
2. `readLines` - считываем строку за строкой данные

Чтение

```

1 def main(file_name):
2     f = open(file_name, "r")
3     # possible bad idea case it will read all data to memory
4     data = f.read()
5     print(f"data is {data}")
6     f.close()
7
8 if __name__ == '__main__':
9     main(file_name=input("input file name: "))
10

```

[TERMINAL](#)
[SQL CONSOLE](#)
[PROBLEMS](#)
[OUTPUT](#)
[DEBUG CONSOLE](#)

```
> zsh
```

[illegible]

Чтение

```

8      data = f.read()
9      print(f"data is {data}")
10     f.close()
11
12     def read_line_by_line(file_name):
13         f = open(file_name, "r")
14         data = []
15         for line in f.readlines():
16             data.append(line)
17         print(f"data is {data}")
18         f.close()
19
20     if __name__ == '__main__':
21         main(file_name=input("input file name: "))
22

```

[TERMINAL](#)
[SQL CONSOLE](#)
[PROBLEMS](#)
[OUTPUT](#)
[DEBUG CONSOLE](#)

```
> zsh
```

```
serg@matrix ~/git/github/python_course/code/src/8 <master*>
```

```
python read_data.py
```

```
input file name: data.dat
```

```
data = data + ['ACGTAAGAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAA\n', 'ACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAA']
```

Запись

Для того чтобы прочитать из файла данные нужно открыть файловый дескриптор запись. Читать можно с помощью функций:

1. `write` - записываем все что в память в файл.
2. `writeln` - записываем одну строку

Запись

```
1 def main(file_name):
2     write_all_data(file_name)
3     #write_line_by_line(file_name)
4
5 def write_all_data(file_name):
6     f = open(file_name, "w")
7     data = ['Some data', 'Some other data']
8     f.write(str(data))
9     f.close()
10
11 def write_line_by_line(file_name):
12     f = open(file_name, "w")
13     data = ['Some data', 'Some other data']
14
15     f.writelines(data)
16     f.close()
17
18 if __name__ == '__main__':
19     main(file_name=input("input file name: "))
20
```

```
TERMINAL  SQL CONSOLE  PROBLEMS  OUTPUT  DEBUG CONSOLE
serg@matrix ~/git/github/python_course/code/src/8 <master*>
python write_data.py
input file name: out.out
serg@matrix ~/git/github/python_course/code/src/8 <master*>
cat out.out
['Some data', 'Some other data']%
serg@matrix ~/git/github/python_course/code/src/8 <master*>
```

Запись

```
1 def main(file_name):
2     #write_all_data(file_name)
3     write_line_by_line(file_name)
4
5 def write_all_data(file_name):
6     f = open(file_name, "w")
7     data = ['Some data', 'Some other data']
8     f.write(str(data))
9     f.close()
10
11 def write_line_by_line(file_name):
12     f = open(file_name, "w")
13     data = ['Some data', 'Some other data']
14
15     f.writelines(data)
16     f.close()
17
18 if __name__ == '__main__':
19     main(file_name=input("input file name: "))
20
```

TERMINAL

SQL CONSOLE

PROBLEMS

OUTPUT

DEBUG CONSOLE

```
serg@matrix ~/git/github/python_course/code/src/8 <master*>
python write_data.py
input file name: data.out
serg@matrix ~/git/github/python_course/code/src/8 <master*>
cat data.out
Some dataSome other data%
serg@matrix ~/git/github/python_course/code/src/8 <master*>
```

Дописывание данных

Обратите внимание, при открытии файла на запись если файл существует, то его содержимое **будет стерто!** Если файла не существует, то он создастся с пустым содержимым сначала. Для того чтобы не потерять уже сохраненные данные, нужно пользоваться режимом “a” - append. При этом запись будет проводиться в конец файла.

Навигация

Файл, представляется как непрерывный поток линейных данных. Соответственно перемещаться в этом потоке можно только в двух направлениях вперед/назад. Перемещение осуществляется с помощью функции [seek\(offset\[, whence\]\)](#)

Форматы

Обычно при чтении файл открывается в текстовом режиме и считывает данные символ за символом в зависимости от кодировки по умолчанию это utf-8. Но файл также можно считать в бинарном виде при этом файл считывает байт за байтом. Для этого при открытии файла нужно задать тип файла, пример

```
open("file_name", "wb")
```

json

Текстовый формат структурированных данных - является одним из самых распространенных форматов межсервисного общения. Представляется как словарь с полями, где поля могут быть также словарем или массивом, или одним из интегральных типов данных `string|int|float|bool|nil`. Python предоставляет возможность работы с json из под капота, для этого вам нужно проимпортировать библиотеку в ваш модуль.

json

```
1 import json
2
3 def read_json(file_name):
4     f = open(file_name, "r")
5     obj = json.load(f)
6     print(obj)
7
8 def write_json(file_name, obj):
9     f = open(file_name, "w")
10    json.dump(obj, f)
11
12 def main():
13     write_json(input("push file name here: "),
14               {
15                   'propInt': 1,
16                   'propFloat': 3.14,
17                   'propBool': True,
18                   'propString': "Hello JSON",
19                   'propDict': {
20                       'key1': 1,
21                       'key1': 2,
22                   },
23                   'propList': [1,2,3]
24               }
25     )
26     read_json(input("push file name here: "))
27
28 if __name__ == '__main__':
29     main()
```

TERMINAL SQL CONSOLE PROBLEMS OUTPUT DEBUG CONSOLE

```
serg@matrix ~/git/github/python_course/code/src/8 <master*>
python json_worker.py
push file name here: js.json
push file name here: js.json
{'propInt': 1, 'propFloat': 3.14, 'propBool': True, 'propString': 'Hello JSON', 'propDict': {'key1': 2}, 'propList': [1, 2, 3]}
```

Менеджер контекста

Заккрытие файла является необходимым атрибутом работы с файлами, так как открытый файловый дескриптор отнимает ресурс у системы. Так как закрытие файла происходит после работы с ним, часто можно забыть вызвать эту функцию.

```
with open("file_name", "file_mode") as f: # f - объект файла  
    #do_smth
```


Менеджер контекста

```
context_manager.py > ...
1  def main(file_name):
2      with open(file_name, 'r') as f:
3          print(f.read())
4
5  if __name__ == '__main__':
6      main(input("Put name of file: "))
```

TERMINAL SQL CONSOLE PROBLEMS OUTPUT DEBUG CONSOLE

```
serg@matrix ~/git/github/python_course/code/src/8 <master*>
python context_manager.py
Put name of file: data.dat
ACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAA
ACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAAACGTAAAA
```

Благодаря менеджеру контекста мы можем быть уверены, что все ресурсы будут освобождены даже при возникновении аварии.

Задачи

1. Напишите скрипт, в котором есть несколько математических операций, определенных в отдельных функциях: `sum`, `div`, `mult`, `subtraction with imputed number`
2. При запуске вашего скрипта он должен спрашивать ваше имя, затем в бесконечном цикле вы должны подавать на ввод вашему скрипту сначала имя функции, которую вы хотите вызвать, затем аргументы функции. Например