


Санкт-Петербургский государственный университет

Python

Сергей Шульман
2017



Было в прошлый раз

- Введение
- Встроенные типы данных
 - Числа
 - Ссылочные типы данных
- Основы грамматики
 - Условные операторы и циклы
 - Функции
 - Работа с файлами



Сегодня:

- Классы и объекты
- Исключения
- Модули
- Библиотеки
- Немного про производительность
- Где можно учиться

Классы и объекты

Основные идеи

- `ИмяКласса()` - создание экземпляра класса конструктором без параметров
- `self` - ссылка на данный объект
- методы, принимающие `self` первым параметром могут быть вызваны только у объекта класса
- методы, не принимающие `self`, могут быть вызваны только у класса
- объект имеет доступ к атрибутам класса

Пример класса

```
class MyClass:  
    A = "variable"  
    def info():  
        print("info - статический метод")  
  
    def my_info(self):  
        print("для my_info нужен объект")
```

```
MyClass.info()  
MyClass().my_info()  
print (MyClass.A)  
print (MyClass().A)
```

Изменение полей класса

```
class MyClass:
    A = 1

obj = MyClass()
MyClass.A = 2
obj2 = MyClass()
obj2.A += 1
print (obj.A)           # 2
print (MyClass.A)       # 2
print (obj2.A)          # 3
obj.B = 4
#obj.C += 1 ошибка
print (obj.B)           # 4
```

- Объект может скопировать атрибут своего класса и изменять свою копию
- Объекту можно добавлять новые поля
- Нельзя изменять поля, которых нет у объекта и его класса
- Класс - объект

Конструкторы

```
class MyClass:  
    def __init__(self, a, b):  
        self.a = a  
        self.b = b
```

```
obj = MyClass(1,2)  
print (obj.a)    # 1  
print (obj.b)    # 2  
# obj = MyClass(3)  
# obj = MyClass()
```

- Конструктор вызывается автоматически при создании объекта
- Конструктор может быть только один

Приватные методы и поля

```
class MyClass:
    def __init__(self, a, b):
        self.__a = a
        self.b = b
    def print_a(self):
        print(self.__a)
    def __test_b(self):
        return self.b > 0
    def is_b_positive(self):
        return self.__test_b()
```

```
obj = MyClass(1,2)
# print (obj.__a)
print (obj.b) # 2
obj.print_a() # 1
# print(obj.__test_b())
print (obj.is_b_positive()) # True
```

- начинаются с двух нижних подчёркиваний
- доступны только из методов класса

Наследование

```
class A:
    def a_name():
        print("I'm A")
    def __private(self):
        print("__private")
    def get_private(self):
        self.__private()
```

```
class B:
    def b_name():
        print("I'm B")
```

```
class C(A,B):
    def get_private(self):
        self.__private()
```

```
C.a_name()
C.b_name()
A().get_private()
#C().get_private()
```

- Наследование может быть множественным
- Приватные методы не доступны
- Класс является наследником самого себя
- [isinstance\(\)](#)
- [issubclass\(\)](#)

Рекомендации

- Не меняйте поля классов
- Аккуратно обращайтесь с полями классов ссылочных типов
- Не добавляйте поля объектам и классам
- Не используйте множественное наследование без необходимости
- Делайте методы приватными, если они нужны только внутри класса
- Не переопределяйте функции и классы

Исключения

Мотивация

- Проблемы во взаимодействии с внешней средой
 - пользовательский ввод
 - взаимодействие по сети
 - сбой оборудования
- Ошибки программиста

Блок try - except

```
try:
    a = float(input("Введите число: "))
    print (100 / a)
except ValueError:
    print ("Это не число!")
except ZeroDivisionError:
    print ("На ноль делить нельзя!")
except:
    print ("Неожиданная ошибка.")
else:
    print ("Код выполнен без ошибок")
```

Finally

```
try:
    my_file = open("filename.txt", "r")
    a = float(my_file.readline())
    print (100 / a)
except ValueError:
    print ("Это не число!")
except ZeroDivisionError:
    print ("На ноль делить нельзя!")
except:
    print ("Неожиданная ошибка.")
else:
    print ("Код выполнен без ошибок")
finally:
    my_file.close()
```

Стандартные исключения

BaseException

- **Exception**

- **ArithmeticError**
 - **FloatingPointError**
 - **OverflowError**
 - **ZeroDivisionError**
- **EOFError**
- **NotImplementedError**
- **TypeError**
- **ValueError**

- **LookupError**
 - **IndexError**
 - **KeyError**
- **MemoryError**
- **OSError**
 - **FileNotFoundError**
 - **PermissionError**
-

Собственное исключение

```
class MyException(Exception):  
    pass
```

```
try:  
    raise MyException('Hi')  
except MyException as e:  
    print(e)  
    raise
```

- Обязательно нужно унаследовать своё исключение от одного из базовых классов
- Можно выбрасывать исключения и пробрасывать пойманные
- Сначала ловят потомков

Traceback

Traceback (most recent call last):

File "sergei_shulman_05.py", line 309, in <module>

 p = Pipeline().fit(df_learn)

File "sergei_shulman_05.py", line 232, in fit

 self.wrapper = ForwardSelection(30, 0.2).fit(X_learn, X_test, y_learn, y_test)

File "sergei_shulman_05.py", line 142, in fit

 y_pred = LinearRegression().fit(X_l, y_learn).predict(X_t)

File "sergei_shulman_05.py", line 106, in fit

 XX = np.matmul(XT, X)

TypeError: Object arrays are not currently supported

Продвинутые возможности

Итераторы

```
class SimpleFilter:
    def __init__(self, iterable, func):
        self.it = iterable
        self.pos = 0
        self.fu = func

    def __iter__(self):
        return self

    def __next__(self):
        while True:
            if self.pos >= len(self.it):
                raise StopIteration
            self.pos += 1
            if self.fu(self.it[self.pos-1]):
                return self.it[self.pos-1]
```

Итераторы

```
def gt2(x):  
    return x > 2  
  
lst = [1,2,3,4,5]  
iterator = SimpleFilter(lst, gt2)  
for i in iterator:  
    print (i, end=' ')  
print()
```

Результат: 3 4 5

Итераторы используются в цикле for.

Лямбда функции

```
def f(x):  
    return x*2  
print(f(3)) # 6
```

```
g = lambda x: x*2  
print(g(3)) # 6
```

```
print((lambda x: x*2)(3)) # 6
```

- Функция с одним стейтментом
- Аргументов может быть много
- Может быть присвоена переменной

Генераторы

```
import itertools

def Fib_generator():
    F0 = 0
    F1 = 1
    while True:
        yield F1
        tmp = F1
        F1 = F0 + F1
        F0 = tmp

print(list(itertools.takewhile(
    lambda x : x <= 100, Fib_generator()))))

# [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

REGEX

```
import re

li = ['9999999999', '999999-999', '99999x9999', '67890', '8122128506']

for val in li:
    if re.match(r'[8-9]{1}[0-9]{9}', val) and len(val) == 10:
        print('yes')
    else:
        print('no')
```

Результат: yes no no no yes

- Очень эффективный способ обработки текста
- Не всегда работает быстро
- Конечные автоматы

Модули

Импорт

```
import re  
from math import tan, sin as s  
from random import *
```

- Можно импортировать целиком или частично
- Возможно переименование
- Вариант `from <module> import` добавляет имена в текущее пространство имён

```
import math as m  
print(m.sin(1))
```

```
from math import *  
print(sin(1))
```

Категории стандартных модулей

Text Processing Services

Binary Data Services

Data Types

Numeric and Mathematical Modules

Functional Programming Modules

File and Directory Access

Data Persistence

Data Compression and Archiving

File Formats

Cryptographic Services

Generic Operating System Services

Internet Protocols and Support

Собственные модули

Любой файл .py может быть подключён как модуль.

```
if __name__ == "__main__":  
    pass
```

Основной код программы должен быть сделан невыполняемым, если мы подключаем программу в качестве модуля.

Получение информации о модулях

- `help("modules")` - список всех установленных модулей
- `sys.path` - переменная, хранящая список всех директорий, в которых ищется модуль
- `dir(modulename)` - список всех имён модуля

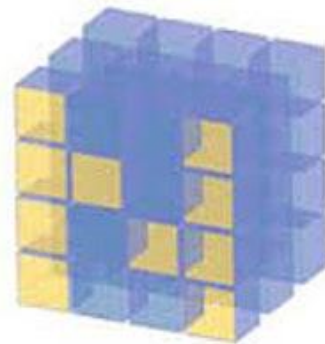
Пакет модулей:

```
from my_package.inside_file import foo
```

в `my_package` должны быть файлы `__init__.py` и `inside_file.py`

Библиотеки

NumPy



NumPy - фундаментальный пакет для научных вычислений на Python. Среди прочего включает:

- многомерные массивы
- сложные функции
- инструменты для интеграции кода на C, C++ и Fortran
- линейную алгебру, преобразование Фурье etc

Ndarray

Многомерный массив фиксированного типа
Этот тип может быть object

```
import numpy as np
```

```
a = np.array([[1.1, 2.2],[3.3, 4.4]])  
print (a)  
print ("a.ndim: ", a.ndim)  
print ("a.shape:", a.shape)  
print ("a.size: ", a.size)  
print ("a.dtype:", a.dtype)
```

```
$ python3 array.py  
[[ 1.1  2.2]  
 [ 3.3  4.4]]  
a.ndim: 2  
a.shape: (2, 2)  
a.size: 4  
a.dtype: float64
```


Ndarray: создание

Есть много способов создания массива:

```
import numpy as np
```

```
a = np.array([1,2,3,4,5])
b = np.array([1,2,3], dtype='float')
c = np.zeros((2,3))
d = np.ones((3,2))
e = np.empty((3,1))
f = np.identity(4)
g = np.arange(0, 1, 0.2)
h = np.linspace(0, 1, 5)
print(a,b,c,d,e,f,g,h, sep='\n\n')
```

```
[1 2 3 4 5]
```

```
[ 1.  2.  3.]
```

```
[[ 0.  0.  0.]
 [ 0.  0.  0.]]
```

```
[[ 1.  1.]
 [ 1.  1.]
 [ 1.  1.]]
```

```
[[ 6.90546448e-310]
 [ 1.88341381e-316]
 [ 1.58101007e-322]]
```

```
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

```
[ 0.  0.2  0.4  0.6  0.8]
```

```
[ 0.  0.25  0.5  0.75  1. ]
```

Ndarray: индексирование и срезы

```
import numpy as np
```

```
a = np.array([[1,2,3],[4,5,6]])  
print(a[0])  
print(a[0,2])  
print(a[0,0:1])  
print(a[a>2])  
print(a[(a%2==0)&(a>2)])  
a[a>2] = 2  
print(a)
```

```
$ python3 array.py  
[1 2 3]  
3  
[1]  
[3 4 5 6]  
[4 6]  
[[1 2 2]  
 [2 2 2]]
```

Сложные операции могут менять форму массива!

Ndarray: операции

```
import numpy as np
```

```
a = np.array([1,2,3])
```

```
b = np.array([4,5,6])
```

```
print(a+b)
```

```
print(a*b)
```

```
print(a**b)
```

```
print(np.sin(a))
```

```
print(np.sum(a))
```

```
print(np.product(a))
```

```
print(np.min(a))
```

```
print(np.max(a))
```

```
print(np.abs(a))
```

```
$ python3 array.py
```

```
[5 7 9]
```

```
[ 4 10 18]
```

```
[  1 32 729]
```

```
[ 0.84147098  0.90929743  0.14112001]
```

```
6
```

```
6
```

```
1
```

```
3
```

```
[1 2 3]
```

Numpy.linalg

```
import numpy as np
```

```
a = np.array([[1,2,3],[1,3,2],[4,5,4]])
```

```
ainv = np.linalg.inv(a.astype(float))
```

```
print(ainv)
```

```
print(np.dot(a, ainv))
```

```
[[ -0.18181818 -0.63636364  0.45454545]
```

```
 [ -0.36363636  0.72727273 -0.09090909]
```

```
 [ 0.63636364 -0.27272727 -0.09090909]]
```

```
[[ 1.00000000e+00  0.00000000e+00  5.55111512e-17]
```

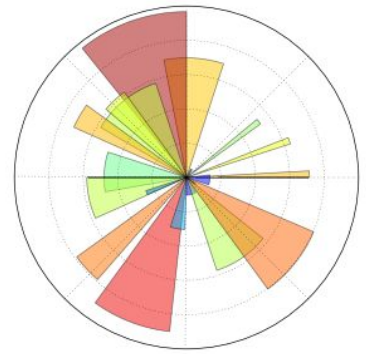
```
 [ 2.22044605e-16  1.00000000e+00  5.55111512e-17]
```

```
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

Numpy

- Используется очень часто
- Написан преимущественно на C
- Работает гораздо быстрее, чем чистый Python
- Может параллелить вычисления
- Использует нестандартный random seed:
`np.random.seed`
- Параллельные потоки стартуют с одинакового seed
- Хорошо налажена связь с другими пакетами

Matplotlib



[Matplotlib](#) - библиотек для 2D и 3D графики

Изначально - подражание MATLAB

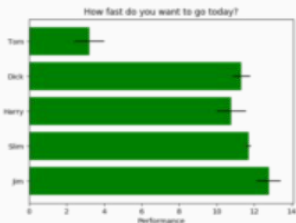
Имеет процедурный интерфейс

Требует NumPy

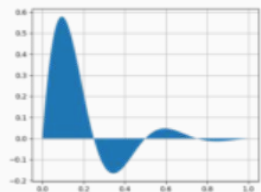
Matplotlib: виды графиков

- Графики (line plot)
- Диаграммы разброса (scatter plot)
- Столбчатые диаграммы (bar chart) и гистограммы (histogram)
- Круговые диаграммы (pie chart)
- Ствол-лист диаграммы (stem plot)
- Контурные графики (contour plot)
- Поля градиентов (quiver)
- Спектральные диаграммы (spectrogram)

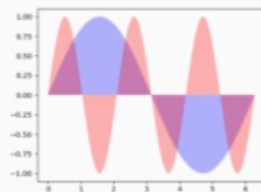
Matplotlib: примеры



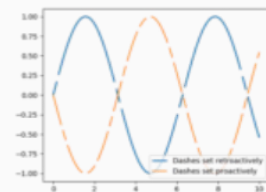
barh_demo



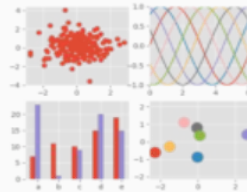
fill_demo



fill_demo_features



line_demo_dash_control



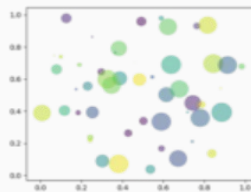
plot_ggplot



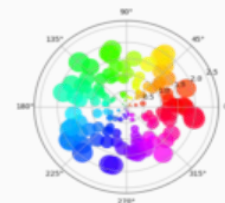
artist_reference



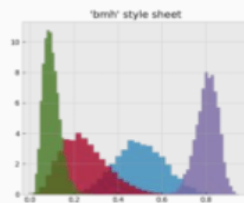
path_patch_demo



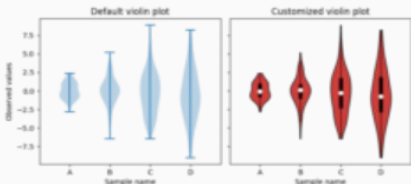
scatter_demo



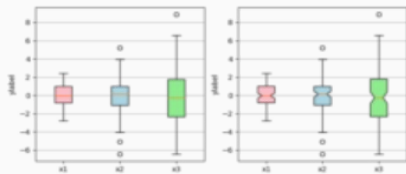
polar_scatter_demo



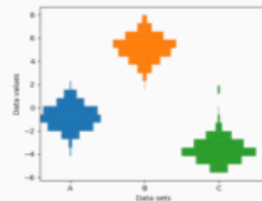
plot_bmh



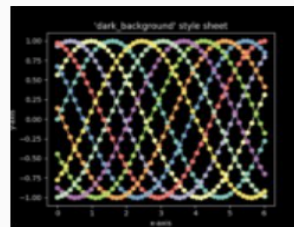
customized_violin_demo



boxplot_color_demo



multiple_histograms_side_by_side



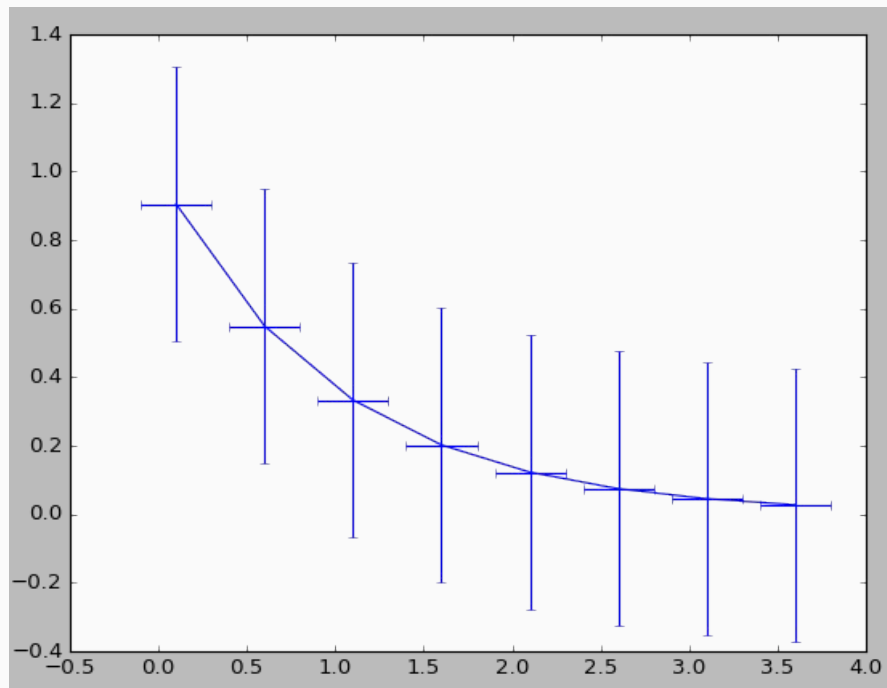
plot_dark_background

Matplotlib: код

```
import numpy as np
import matplotlib.pyplot as plt

# example data
x = np.arange(0.1, 4, 0.5)
y = np.exp(-x)

fig, ax = plt.subplots()
ax.errorbar(x, y, xerr=0.2, yerr=0.4)
plt.show()
```



SciPy



SciPy - библиотека для научных и инженерных расчётов.

Основная структура данных - `numpy.ndarray`

Для визуализации часто применяется `matplotlib`

Может выполнять анализ данных на R

SciPy: список модулей

constants - физические константы и коэффициенты
fftpack - дискретные алгоритмы преобразования Фурье
integrate - интегрирование
interpolate - интерполяция
io - ввод-вывод данных
lib - работа со сторонними библиотеками.
linalg - линейная алгебра
optimize - оптимизация
sparse - поддержка разреженных матриц
special - специальные функции
stats - Статистические функции

SciPy: численное интегрирование

$$\int_0^{\pi} \sin(x) dx$$

```
from scipy import integrate
import math
```

```
def f(x):
    return math.sin(x)
```

```
print(integrate.quad(f, 0, math.pi))
# (2.0, 2.220446049250313e-14)
```

SciPy: интерполяция

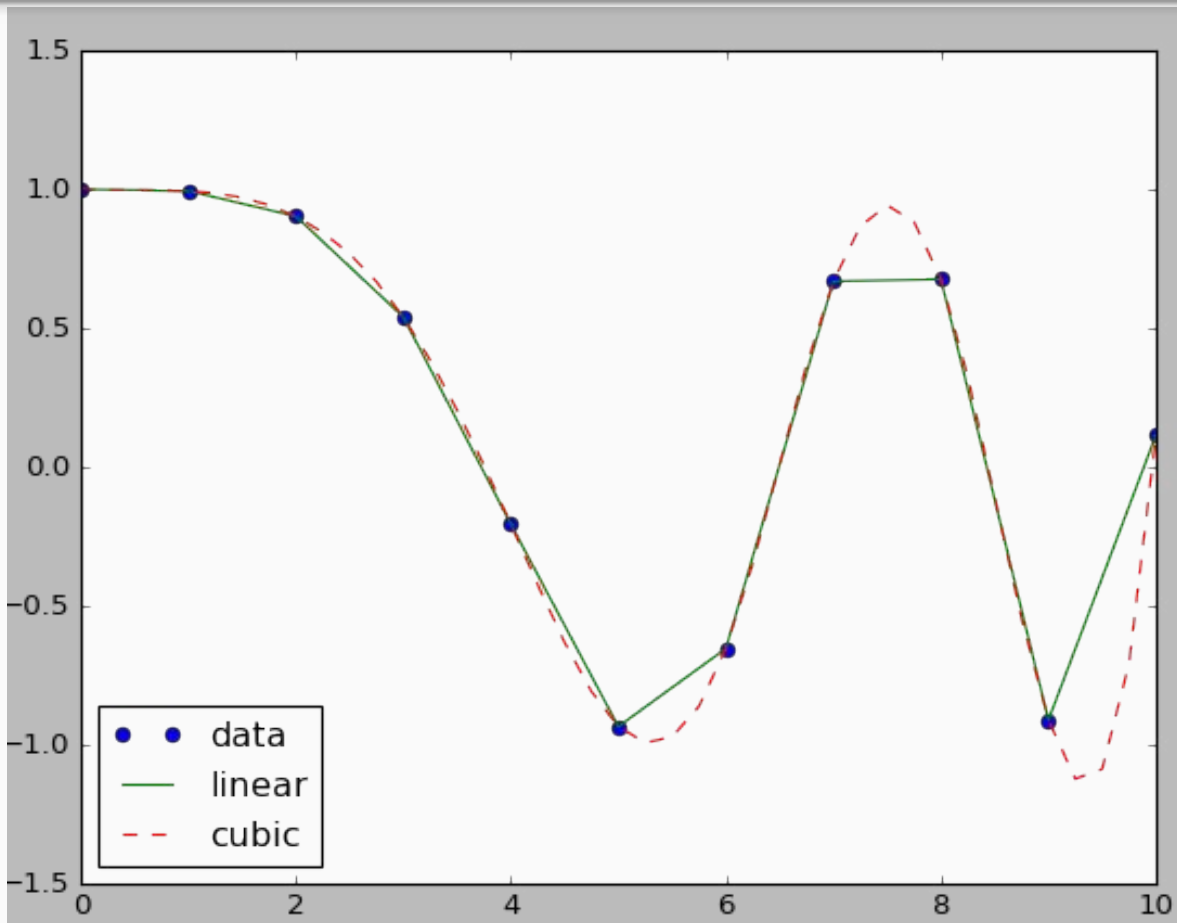
```
from scipy.interpolate import interp1d  
import numpy as np
```

```
x = np.linspace(0, 10, num=11, endpoint=True)  
y = np.cos(-x**2/9.0)  
f = interp1d(x, y)  
f2 = interp1d(x, y, kind='cubic')
```

```
xnew = np.linspace(0, 10, num=41, endpoint=True)
```

```
import matplotlib.pyplot as plt  
plt.plot(x, y, 'o', xnew, f(xnew), '-', xnew, f2(xnew), '--')  
plt.legend(['data', 'linear', 'cubic'], loc='best')  
plt.show()
```

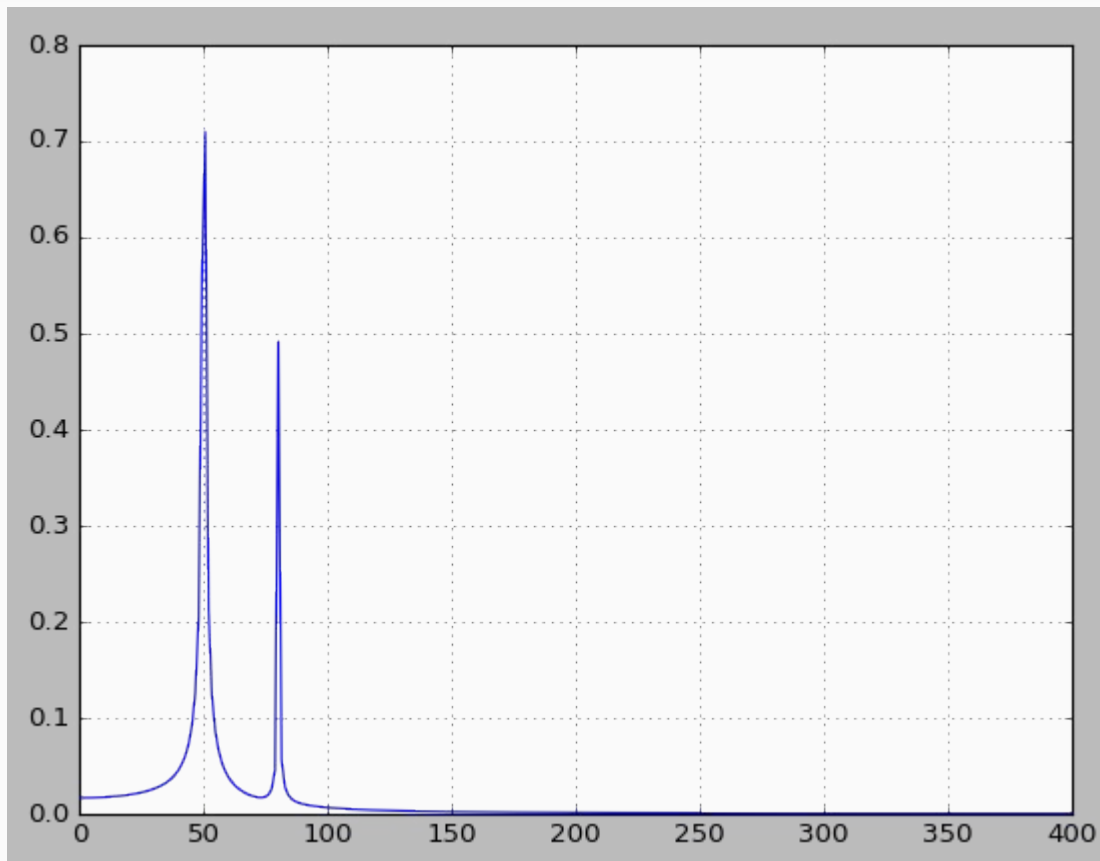
SciPy: интерполяция. Результат



SciPy: FFT

```
from scipy.fftpack import fft
import numpy as np
N = 600
T = 1.0 / 800.0
x = np.linspace(0.0, N*T, N)
y = np.sin(50.0 * 2.0*np.pi*x) + 0.5*np.sin(80.0 * 2.0*np.pi*x)
yf = fft(y)
xf = np.linspace(0.0, 1.0/(2.0*T), N//2)
import matplotlib.pyplot as plt
plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]))
plt.grid()
plt.show()
```

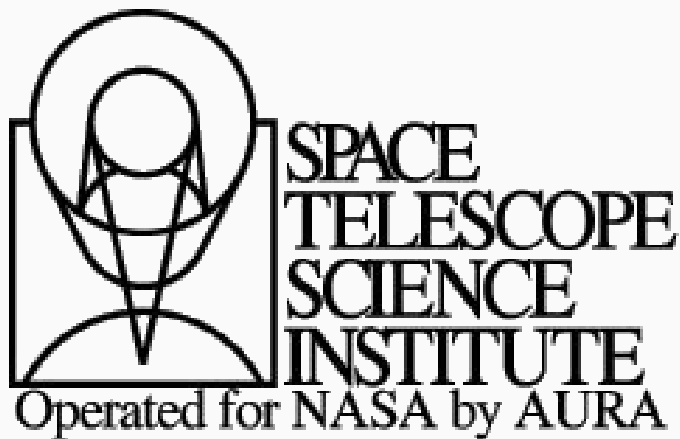
SciPy: FFT. Результат



AstroPy



[AstroPy](#) - библиотека для выполнения астрономических вычислений.



AstroPy

- Конвертация физических величин и константы
- Небесные координаты и преобразования времени
- World coordinate system (WCS) support
- FITS
- Virtual Observatory (VO) tables
- Common ASCII table formats, e.g. for online catalogues or data supplements of scientific publications
- [Hierarchical Data Format](#) (HDF5) files
- Космологические преобразования
- Инструменты статистического анализа

PyEphem



[PyEphem](#) - астрономический пакет для вычисления эфемерид

PyEphem: пример

```
import ephem
```

```
mars = ephem.Mars()  
mars.compute()  
print (mars.ra, mars.dec)  
# 13:32:48.66 -8:33:31.9
```

```
print(ephem.constellation(mars))  
#('Vir', 'Virgo')
```

```
boston = ephem.Observer()  
boston.lat = '42.37'  
boston.lon = '-71.03'  
mars.compute(boston)  
print (mars.az, mars.alt)  
# 265:53:58.4 -8:14:14.4
```

```
boston.next_rising(mars)  
print (mars.az)  
# 101:15:51.1
```

```
boston.next_transit(mars)  
print (mars.alt)  
# 38:55:03.1
```

Другие пакеты

PyFits - работа с fits (отдельно от AstroPy)

Pandas - обработка и анализ данных (поверх NumPy)

Scikit-learn - машинное обучение

Scikit-image - обработка изображений

Statsmodels - статистические модели

SymPy - символьные вычисления

....

Способы повышения производительности

- Работа с кодом на Python
- Часть кода на C

Код на чистом Python

```
from random import randint, random
import timeit
import itertools
from functools import reduce
```

```
list_of_list = [[random() for _ in range(randint(1, 10))] for _ in range(10**5)]
```

```
a = timeit.default_timer()
f_list = []
for sub_list in list_of_list:
    for item in sub_list:
        f_list.append(item)
print(timeit.default_timer()-a) # 0.06736829000146827
```

```
a = timeit.default_timer()
f_list= []
for sub_list in list_of_list:
    f_list += sub_list
print(timeit.default_timer()-a) # 0.013693414999579545
```

```
a = timeit.default_timer()
f_list= itertools.chain.from_iterable(list_of_list)
print(timeit.default_timer()-a) # 0.0017860860025393777
```

```
a = timeit.default_timer()
f_list= reduce(lambda y, x: y + x, list_of_list, [])
print(timeit.default_timer()-a) # 207.3064950869957
```

ctypes

```
unsigned long fib(unsigned long x)
{
    if(x < 2) return x;
    return fib(x-1) + fib(x-2);
}
```

```
gcc -shared -Wl,-soname,myfib.so -o
myfib.so -fPIC -O3 fib.c
```

```
102334155
0.46477239899832057
102334155
35.483732111999416
```

```
import ctypes
import timeit
```

```
a = timeit.default_timer()
fib = ctypes.CDLL('./myfib.so').fib
fib.restype = ctypes.c_long
fib.argtypes = (ctypes.c_ulong,)
```

```
a = timeit.default_timer()
print(fib(40))
print(timeit.default_timer()-a)
```

```
a = timeit.default_timer()
def pyfib(x):
    if x < 2: return x
    return pyfib(x-1) + pyfib(x-2)
print(pyfib(40))
print(timeit.default_timer()-a)
```

<http://www.py-my.ru/post/50997be0bbddb2f44000002>

Где можно учиться



[Программирование на Python](#)

[Python: основы и применения](#)

[Adaptive Python](#)

The background features three large, overlapping diagonal stripes in teal, light green, and yellow. The PyCharm Edu logo is centered in the upper half, consisting of a black square with 'PE' and a horizontal line, followed by the text 'PyCharm Edu' in a bold, black, sans-serif font.

PyCharm Edu

Easy and Professional Tool
to Learn & Teach Programming with Python

DOWNLOAD FREE

Что читать?

[Python 3.6.3 documentation](#)

[Python Enhancement Proposals 8](#)

СПИСОК КНИГ:

<https://wiki.python.org/moin/PythonBooks>

<https://wiki.python.org/moin/IntroductoryBooks>

Learning Python, 5th Edition

(Mark Lutz, O'Reilly Media, June 2013)

The Python standard library by example

(D. Hellmann, Addison-Wesley, 2011)

Python Cookbook (D.Beazley, B. Jones, O'Reilly, 2013)

Вопросы?

Спасибо за внимание!