САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра астрофизики

Методическое пособие
Введение в Python для студентов-астрономов

составил Марчук Александр

Санкт-Петербург, 2016

О чем это пособие: в этом пособии мы кратко рассмотрим основные конструкции и начало работы с языком программирования Python, сделав упор на его применение в астрономии. Помимо синтаксиса, типов и структур данных будет обсуждаться на примерах работа с вычислениями в numpy, построение графиков в matplotlib, анализ функций в scipy, работа с небесными координатами с помощью pyephem и анализ FITS изображений в astropy.

Чем это пособие не является: эта работа ни в коем случае не является полноценным руководством по Python, здесь *кратко* приведены лишь основные сведения о языке, которых должно хватить для начала работы. Для дальнейшего изучения в конце приведен список дополнительной и полезной литературы.

Почему руководство так выглядит: это руководство составлено в среде Ipython Notebook, которая позволяет писать текст с разметкой и формулами латех, а также исполнять код на питоне и дает удобный способ продемонстрировать читателю результат его выполнения. Данное руководство доступно также на https://github.com/Amarchuk/2FInstability/pyastrotutorial (https://github.com/Amarchuk/2FInstability/pyastrotutorial).

Содержание

О языке

Разработка языка Python была начата в конце 1980-х годов сотрудником голландского института Гвидо ван Россумом. Гвидо начал писать Python на досуге, позаимствовав некоторые наработки из языка ABC. В феврале 1991 года Гвидо опубликовал исходный текст первой версии. Название языка произошло вовсе не от вида змей, автор назвал язык в честь популярного британского комедийного телешоу 1970-х «Летающий цирк Монти Пайтона». Наличие дружелюбного, отзывчивого сообщества пользователей считается одним из факторов успеха Python. Развитие языка происходит согласно чётко регламентированному процессу создания, обсуждения, отбора и реализации предложений по развитию Python (документов PEP). Последняя версия языка на момент написания это 3.5.3, но очень большое количество людей (около половины) по прежнему используют версию 2.7. В Python версий 3.x устранены многие недостатки архитектуры с максимально возможным (*но не полным!*) сохранением совместимости со старыми версиями Python. На сегодняшний день поддерживаются обе ветви развития (Python 3.x и 2.x).

Python это **интерпретируемый** язык, т.е. код на этом языке не компилируется в машинные команды, а исполняется специальной программой-интерпретатором, что делает его более медленным, чем C/C++ или Fortran. Тем не менее множество людей (в 2015 году это самый используемый в астрономии язык - http://arxiv.org/abs/1507.03989 (http://arxiv.org/abs/1507.03989)) предпочитают Python по многим причинам:

- свободный и бесплатный, активно развивающийся
- язык высокого уровня, мультипарадигменный
- простой и ясный синтакс повышает читаемость
- простота и удобство написания программ значительно повышают скорость программирования
- подробные руководства и детальное описание ошибок
- интерактивный режим позволяет легко проводить последовательный анализ шаг за шагом
- наличие большого сообщества и пакетов для решения всех необходимых задач (в т.ч. астрономических)
- умеет легко исполнять C/Fortran код
- вычислительные пакеты для научной работы написаны на C и быстро работают
- портируемость
- многое другое...

У Python есть собственная философия, т.н. *"Дзен Питона"*, следование которому позволяет понять язык и писать более эффективно. Чтобы увидеть список этих правил введите команду

```
import this
```

Приведем здесь перевод этих правил на русский язык:

- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.

- Читаемость важна.
- Особые случаи не настолько особые, чтобы нарушать правила.
- Хотя практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один — и, желательно, только один — очевидный способ сделать это.
- Хотя он поначалу может быть и не очевиден, если вы не голландец.
- Сейчас лучше, чем никогда.
- Хотя никогда зачастую лучше, чем прямо сейчас.
- Если реализацию сложно объяснить, то идея плоха.
- Если реализацию легко объяснить, то идея, возможно, хороша.
- Пространства имён — отличная штука. Давайте делать их больше!

Python и уже написанные на нем программы эффективно решают многие чисто астрономические задачи, такие как перевод единиц измерения, учет даты и времени, переход из одной системы координат в другую, моделирование и приближение функций, работа с FITS файлами, статистика, космологические вычисления, задачи машинного обучения в астрофизике, подготовка изображений к публикации, фотометрическая обработка, анализ спектров и многое другое.

Два способа работы с Python

- Интерактивный режим – взаимодействие через командную строку:

```
$: python
```

- Через программу (скрипт) – создание текстового файла с командами и исполнение его при помощи интерпретатора Python

```
$: python script.py
```

Давайте напишем классическую программу приветствия, создав файл hw.py и написав в нем:

```
#! /usr/bin/env python
print 'Hello world!'
```

Первая строчка указывает путь до интерпретатора. Запуск программы:

```
$: python hw.py
Hello world!
```

То же самое в интерактивном режиме:

```
$: python
>>>print 'Hello world!'
Hello world!
```

Чтобы использовать (например в комментариях к программе) не-ascii символы необходимо явно указать кодировку, добавив в начало файла следующую строчку:

```
# -*- coding: utf-8 -*-
```

Комментарии в программе Python начинают с символа #.

Перейдем непосредственно к языку и его синтаксису.

Типы данных

Логические типы

Логические типы самые простые в языке и сводятся к двум значениям: `True` и `False`, т.е. истина и ложь. Как ложь также интерпретируются нули числовых типов, пустые последовательности и `None`.

In [1]:

```
a = True
b = False
```

Логические операторы "и" `and`, "или" `or` и "не" `not` легко читать и использовать:

In [2]:

```
True and False
```

Out[2]:

```
False
```

In [3]:

```
True or False
```

Out[3]:

```
True
```

In [4]:

```
not True
```

Out[4]:

```
False
```

In [5]:

```
(False and (True or False)) or (not False and True)
```

Out[5]:

```
True
```

Операторы сравнения также имеют в качестве своего значения `True` или `False`:

In [6]:

```
1 == 2
```

Out[6]:

```
False
```

In [7]:

```
2 != 3
```

Out[7]:

```
True
```

In [8]:

```
5 < 8
```

Out[8]:

```
True
```

In [9]:

```
3 >= 3.4
```

Out[9]:

```
False
```

Строки

Строки могут быть представимы в различных стандартах, основные из которых это ascii и unicode. Для обозначения строковой переменной используют одинарные или двойные кавычки:

In [10]:

```
s = 'MNRAS'
print s
```

```
MNRAS
```

In [11]:

```
s = "AJ"
print s
```

```
AJ
```

In [12]:

```
s = '''Nature'''
print s
```

```
Nature
```

Для использования кавычек внутри самих строк и символов переноса используют backslash '\'.

In [13]:

```
s = 'O\'Brien:\n \"Detection\t of\t a\t turbulent\t gas\t component\"'
print s
```

```
O'Brien:
 "Detection	 of	 a	 turbulent	 gas	 component"
```

У строк можно легко обращаться к отдельным буквам и подстрокам, используя отдельные индексы и т.н. срезы (slices):

In [14]:

```
s = 'abcdef'
print s[0], s[1], s[2], s[-3], s[-2], s[-1]
```

```
a b c d e f
```

In [15]:

```
#обращение к несуществующему индексу
print s[100]
```

```
---------------------------------------------------------------------------
----
IndexError                                Traceback (most recent call l
ast)
<ipython-input-15-fc6c7b344ed3> in <module>()
      1 #обращение к несуществующему индексу
----> 2 print s[100]

IndexError: string index out of range
```

Срез задается обычно тремя числами - индекс начала среза, конца и величина шага (в таком порядке). Не обязательно указывать все числа, отрицательные числа обозначают положение с конца строки (-1 - последний элемент, -2 - предпоследний и т.д.). Обратите внимание, что элемент, указанный в качестве конечного, не включается в результат!

In [16]:

```
s = '123456'
print s[0:6:2]
```

```
135
```

In [17]:

```
print s[1:6:2]
```

```
246
```

In [18]:

```
print s[4:5]
```

```
5
```

In [19]:

```
print s[-1:-7:-1]
```

```
654321
```

In [20]:

```
print s[::2]
```

```
135
```

In [21]:

```
print s[1::2]
```

```
246
```

У строковых типов есть много полезных методов:

In [22]:

```
s = 'Zen of Python'
print s.upper()
print s.lower()
```

```
ZEN OF PYTHON
zen of python
```

In [23]:

```
print s.startswith('Z')
```

```
True
```

In [24]:

```
print s.replace('Zen', 'Power')
```

```
Power of Python
```

In [25]:

```
print s.count('n')
```

```
2
```

In [26]:

```
print len(s)
```

```
13
```

Числовые типы

Все типы похожи на те, которые втречаются в других языках:

Целочисленный тип (int)

In [27]:

```
print 2 + 3, 4*5, 6-7, 8/9, 10%3, abs(-3)
```

```
5 20 -1 0 1 3
```

Обратите внимание, что деление также целочисленное. Диапазон значений int от -2147483648 до 2147483647. Если необходимы большие значения, то можно использовать long int:

In [28]:

```
print 1l + 2l, 5*5l, 9**9
```

```
3 25 387420489
```

Операция '**' - это возведение в степень, '%' - остаток от деления. Над целыми числами также можно производить побитовые операции |, ^, &, <<, >>, ~.

Числа с плавающей точкой (float)

Числа с плавающей точкой описывают вещественные числа, однако надо быть аккуратным и помнить об ошибках представления чисел в компьютере. Тип float в питоне не поддерживает длинную арифметику.

In [29]:

```
0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
```

Out[29]:

```
0.9999999999999999
```

In [30]:

```
3.**1000 + 0.1
```

```
---------------------------------------------------------------------
----
OverflowError                             Traceback (most recent call l
ast)
<ipython-input-30-0723d5d45eba> in <module>()
----> 1 3.**1000 + 0.1

OverflowError: (34, 'Result too large')
```

При операциях с данными разного типа действует правило приведения результата к более "старшему" типу:

In [31]:

```
print 1/5, 1./5, 2 + 2l + 3.
print type(1/5), type(1./5), type(2 + 2l + 3.0)
```

```
0 0.2 7.0
<type 'int'> <type 'float'> <type 'float'>
```

Комплексные числа

Мнимая часть у комплексных чисел обозначается буквой `j`.

In [32]:

```
(1+1j)*2j
```

Out[32]:

```
(-2+2j)
```

In [33]:

```
abs(3+4j)
```

Out[33]:

```
5.0
```

In [34]:

```
c = 10.5 - 4.3j
print c.real, c.imag
```

```
10.5 -4.3
```

In [35]:

```
(1+2j)/(1+1j)
```

Out[35]:

```
(1.5+0.5j)
```

Структуры данных

Списки

Списки (т.е. по сути массивы) в языке Python имеют два основных вида - собственно тип список `list` и тип `tuple` (его переводят на русский странным словом `'кортеж'`). Разница между ними в том, что список можно изменять (увеличивать, уменьшать), а кортеж - создается один раз и дальше не изменяется. Разница между этими двумя типами в коде программы в различных используемых скобочках:

In [36]:

```
tup = ('alpha', 'beta', 'gamma')
print type(tup)
l = [3.14, 2.41, 1.69]
print type(l)
```

```
<type 'tuple'>
<type 'list'>
```

В одном списке можно хранить объекты разной природы - одновременно строки, числа, функции и т.д.

In [37]:

```
author = ['Alex', 26., max(3, 5), True]
print author
```

```
['Alex', 26.0, 5, True]
```

Элементами списка также могут быть другие списки, и даже разной длины:

In [38]:

```
lls = [[1,2,3], [3,4], [5]]
print lls
```

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

Список можно превратить в кортеж и обратно, к элементам списка можно обращаться по индексу, делать срезы.

In [39]:

```
tl = (0., 45., 90., 180., 360.)
lt = list(tl)
print type(tl), '->', type(lt)
```

```
<type 'tuple'> -> <type 'list'>
```

In [40]:

```
print tl[0], tl[-1], tl[2:4], tl[-1:-7:-1]
```

```
0.0 360.0 (90.0, 180.0) (360.0, 180.0, 90.0, 45.0, 0.0)
```

Также существует множество удобных служебных методов для работы со списками:

In [41]:

```
loc = ['Petergof', 'SAO', 'Pulkovo']
print len(loc) #длина списка
loc.append('Pushino') #добавление нового элемента
print loc
print 'CERN' in loc #проверка на наличие элемента в списке
print sorted(loc)
print loc.count('SAO')
print min([1,2,3]), max([1,2,3]), sum([1,2,3])
print ['42']*10 #список из одинаковых элементов заданной длины
print [1,2,3] + [4,5,6]
```

```
3
['Petergof', 'SAO', 'Pulkovo', 'Pushino']
False
['Petergof', 'Pulkovo', 'Pushino', 'SAO']
1
1 3 6
['42', '42', '42', '42', '42', '42', '42', '42', '42', '42']
[1, 2, 3, 4, 5, 6]
```

Словари

Словарь в Python - это набор пар "ключ-значение" (например созвездие-координаты). Для создания словаря используются фигурные скобки {}.

In [42]:

```
d = {1:2, 3:'four', (1+1j):7}
print d
```

```
{1: 2, 3: 'four', (1+1j): 7}
```

Обратите внимание, что ключи и значения не обязаны быть одного типа. Получение элемента по ключу происходит следующим образом:

In [43]:

```
print d[1]
print d[(1+1j)]
```

```
2
7
```

Схожим образом можно добавить элемент в словарь:

In [44]:

```
d['eight'] = 9.99
print d
```

```
{'eight': 9.99, 1: 2, 3: 'four', (1+1j): 7}
```

Попытка обращения к элементу по несуществующему ключу приведет к ошибке. Можно просмотреть все возможные ключи командой `keys()`, а все возможные значения с помощью `values()`.

In [45]:

```
print d.keys()
print d.values()
print d['not a key']
```

```
['eight', 1, 3, (1+1j)]
[9.99, 2, 'four', 7]

---------------------------------------------------------------------------
----
KeyError                                  Traceback (most recent call l
ast)
<ipython-input-45-cc044142feb4> in <module>()
      1 print d.keys()
      2 print d.values()
----> 3 print d['not a key']

KeyError: 'not a key'
```

Проверить, есть ли данный ключ в списке можно с помощью команды `in` (как ранее для строк и списков):

In [46]:

```
print 'eight' in d
print 'not a key' not in d
```

```
True
True
```

Основные синтаксические конструкции

Как и в других языках, в Python есть основные конструкции условий и циклов.

In [47]:

```
if 10 > 11:
    print 'You broke the math!'
elif -1 > 1:
    print 'You did it again!'
else:
    print 'End'
```

```
End
```

In [48]:

```
i = -4
while i < 0:
    i += 1
    print i
```

```
-3
-2
-1
0
```

Обратите внимание на форматирование строк и на отступы. Это не было упомянуто раньше, но в питоне с самого начала было очень важное правило, которое делало программы читаемыми - "каждая строчка кода во вложенной конструкции должна начинаться с дополнительного отступа". Отступ в данном случае - это табуляция или четыре пробела. Если форматирование в программе не будет соблюдено, то возникнет ошибка `IndentationError`:

In [49]:

```
if i < 5:
print i
```

```
  File "<ipython-input-49-c395e7989343>", line 2
    print i
        ^
IndentationError: expected an indented block
```

Команды `continue` и `break` позволяют продолжить или прервать циклы `for` и `while`:

In [50]:

```
while i < 5:
    print i
    if i == 4:
        break
    else:
        i += 1
        continue
```

```
0
1
2
3
4
```

Цикл `for` немного отличается от общепринятого, в нем необходим "итерриуемый" объект, по всем значениям которого мы будем проходить в цикле. "Итерриуемые" объекты из тех, что мы уже знаем - это `string`, `list`, `tuple` (а также `file` и т.д.).

In [51]:

```
for letter in 'string':
    print letter
```

```
s
t
r
i
n
g
```

In [52]:

```
for number in [0, 9, 8, 7, 6]:
    print number**2
```

```
0
81
64
49
36
```

Функции

Очень важное правило, которое помогает писать программы быстро и эффективно - избегать повторов одного и того же кода. Для этих целей в язык программирования вводятся функции, и Python не исключение. Определение функции очень простое:

```
    def function_name(arguments):
        # code here
        return values
```

По негласным правилам в названиях функций используют прописные буквы. Аргументов может быть несколько:

In [53]:

```
def mult(a, b):
    return a * b

print mult(1, 2)
print mult(3, 4)
```

```
2
12
```

Для упрощения восприятия и читаемости аргументы функции можно делать именнованными - добавлять им обозначения. Именнованные параметры должны иметь значение по умолчанию, поэтому указывать эти аргументы не обязательно (в таком случае будет использовано значение по умолчанию). Важно, что обращение к именнованным аргументам возможно только после неименнованных (легко понять почему):

In [54]:

```
def reduplicator(string, repeat=2):
    return string*repeat

print reduplicator('abs')
print reduplicator('abc', repeat=3)
```

```
absabs
abcabcabc
```

In [55]:

```
print reduplicator(repeat=4, 'abc')
```

```
  File "<ipython-input-55-f7d13b7af15a>", line 1
    print reduplicator(repeat=4, 'abc')
SyntaxError: non-keyword arg after keyword arg
```

Можно возвращать множественные значения:

In [56]:

```
def square_and_cube(a):
    return a**2, a**3

print square_and_cube(3)
a2, a3 = square_and_cube(5) #можно также получить сразу отдельные значения
print a2
print a3
```

```
(9, 27)
25
125
```

Отметим, что функция не обязательно должна возвращать значение:

In [57]:

```
def print_status(b):
    if b:
        print 'status:True'
    else:
        print 'ststus:False'

print_status(True)
```

```
status:True
```

Модули и пакеты

Большинство встроенных функций находятся внутри своих модулей - написанных библиотек, объединенных общим смыслом. Например модуль `math` включает в себя функции acos, acosh, asin, asinh, atan, atan2, atanh, cos, cosh, degrees, e, exp, fabs, factorial, log, log10, log1p, modf, pi, pow, radians, sin, sinh, sqrt, tan, tanh. В Python встроенные модули называются Стандартной Библиотекой (*Standard Library*) и описаны в документации (http://docs.python.org/3/library/index.html). Чтобы использовать функцию из модуля, его нужно **импортировать**, т.е. подключить. Существует три способа сделать это:

In [58]:

```
# способ 1 - подключить модуль целиком и писать его имя перед функцией
import math
print math.exp(3)

# способ 2 - импортировать только функцию
from math import exp
print exp(3)

# способ 3 - импортировать все функции из модуля
from math import *
print exp(3)

#существует также способ сокращения имен модулей
import math as m
print m.exp(3)
```

```
20.0855369232
20.0855369232
20.0855369232
20.0855369232
```

Отметим полезные модули встроенной библиотеки:

- os – функции операционной системы
- string – работа со строками
- cmath – функции для работа с комплексными числами
- random – генерация псевдослучайных чисел
- pickle, csv – хранение объектов
- time – определение и обработка времени
- urllib2 – чтение ресурсов по URL
- gzip, bzip, tarfile – работа с архивами
- и т.д.

Коллекции модулей распространяются как *пакеты* (packages), которых очень и очень много. Для уставновки пакетов можно использовать различные менеджеры пакетов, самый простой из которых `pip`. Чтобы установить, например, пакет для работы с эфемеридами, надо набрать в консоли

```
$ pip install ephem
```

Модули `numpy`, `matplotlib`, `astropy`, о которых речь пойдет ниже, могут быть установлены таким же способом. Отметим, что есть различные сборки пакетов для научной работы, самая известная из которых Anaconda (http://continuum.io/downloads). Эта сборка позволяет за один раз скачать и установить большинство необходимых пакетов.

Полезные замечания

Перед тем, как перейти собственно к примерам работы на питоне, сделаем ряд важных замечаний, не объединенных общим контекстом, которые тем не менее упростят работу.

Для генерации наборов последовательных чисел можно использовать функции `range` и `xrange`:

In [59]:

```
print range(5)
print range(4, 9)
print range(-2, -8, -2)
print xrange(6)
```

```
[0, 1, 2, 3, 4]
[4, 5, 6, 7, 8]
[-2, -4, -6]
xrange(6)
```

Разница между `range` и `xrange` в том, что второе - это т.н. генераторная функция, которая вычисляется не вся сразу (это может быть полезно при работе с большими числами).

Существует ключевое слово `None`, которое можно использовать для инициализации значения (чтобы не выбирать другое значение и не запоминать его). Сравните читаемость кода:

In [60]:

```
l = None
if l:
    l += 1 #работа с l
else:
    l = 1 #инициализация


l = -1 #плохая инициализация, снижает понимаемость кода
if l != -1:
    l += 1 #работа с l
else:
    l = 1 #инициализация
```

Списки можно объединять в пары, используя ключевое слово `zip` ("застежка"):

In [61]:

```
l1 = [1, 2, 3]
l2 = ['one', 'two', 'four']
print zip(l1, l2)
```

```
[(1, 'one'), (2, 'two'), (3, 'four')]
```

Также с помощью этой команды можно произвести обратную операцию:

In [62]:

```
l = zip([1,2,3], range(3))
l1, l2 = zip(*l)
print l1, l2
```

```
(1, 2, 3) (0, 1, 2)
```

Для тех случаев, когда необходимо провести одноразовые вычисления, но заводить функцию не хочется, можно использовать т.н. лямбда-выражения. По сути это те же функции, но с немного другим синтаксисом:

In [63]:

```
foo = lambda l: (l/2.)**2
print foo(1.), foo(2.)
```

```
0.25 1.0
```

Это бывает полезно например для тех случаев, когда необходимо вычислить один раз что-либо для элементов списка. Чтобы применить функцию к каждому элементу списка используют функцию `map` (сначала пишем функцию, затем то, к чему ее надо применить):

In [64]:

```
print map(abs, [-3, 4, -6])
print map(lambda x: x**3-1, range(3))
```

```
[3, 4, 6]
[-1, 0, 7]
```

Тот же результат можно получить в еще более удобной форме, если использовать т.н. "генераторы списков" (list comprehension):

In [65]:

```
print [x**3 - 1 for x in range(3)]
```

```
[-1, 0, 7]
```

In [66]:

```
print [l*0.5 + l**2/3. for l in [3,6,9,12]]
print [l*0.5 + l**2/3. for l in range(3,13,3)]
```

```
[4.5, 15.0, 31.5, 54.0]
[4.5, 15.0, 31.5, 54.0]
```

Этот синаксис можно использовать не только для списков:

In [67]:

```
print {l:l.lower() for l in ['Petergof', 'SAO', 'Pulkovo']}
```

```
{'SAO': 'sao', 'Petergof': 'petergof', 'Pulkovo': 'pulkovo'}
```

Работа с файлами в питоне похожа на другие языки - необходимо открыть файл на чтение(r), запись(w) или запись в конец(a), затем считать информацию и после этого закрыть файл.

In [68]:

```
f = open('data.txt', 'w')
f.write('line 1 \nline 2')
f.close()

f = open('data.txt', 'r')
for line in f.readlines():
    print line
f.close()
```

```
line 1

line 2
```

Для получения справки по функции в ipython необходимо набрать знак вопроса и название функции, например `?open`. В обычном интерпретаторе Python для тех же целей служит команда `help`, например `help(open)`.

Да, в Python конечно же есть возможность писать классы и использовать объектно-ориентированный подход, но здесь мы рассматривать это не будем.

И последнее: отметим, что гораздо удобнее работать в специальной среде, называемой Ipython Notebook (https://ipython.org/documentation.html) (недавно ее переименовали в Jupiter). Она позволяет сильно упростить работу за счет так называемых "магических" команд (например исполнять команды из консоли внутри питона, измерять время работы, использовать сторонние скрипты и многое другое). Я настоятельно рекомендую ознакомиться с этой средой.

Введение в Numpy

Списки в Python являются очень гибкой структурой данных, которые к тому же легко модифицировать и не обязательно приводить к одному типу. Однако гибкость обычно влечет за собой ухудшение скорости работы, и порой значительное. Поэтому для наукоемких вычислений в питоне используют пакет **Numpy** (NUMerical PYthon). Numpy предоставляет возможность работы с т.н. numpy-массивами, которые написаны на C и работают достаточно быстро. Ограничением служит требование единого типа данных. Импортировать модуль принято используя следующее сокращение:

In [69]:

```
import numpy as np
```

Есть несколько способов завести numpy array. Самое простое - создать из списка:

In [70]:

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

Out[70]:

```
array([1, 2, 3, 4])
```

Также существует аналаг функции range, но с вещественным шагом:

In [71]:

```
print np.arange(1., 2., 0.1)
print np.arange(10)
```

```
[ 1.   1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9]
[0 1 2 3 4 5 6 7 8 9]
```

Другой удобный способ создания массива - функции linspace и logspace, которые разбивают промежуток на заданное количество частей равномерно и логарифмически соответственно:

In [72]:

```
print np.linspace(1., 2., 10)
print np.logspace(1., 2., 5, base=2)
```

```
[ 1.          1.11111111  1.22222222  1.33333333  1.44444444  1.5555555
6
  1.66666667  1.77777778  1.88888889  2.        ]
[ 2.          2.37841423  2.82842712  3.36358566  4.        ]
```

Можно создать пустой массив, инициализированный нулями или единицами:

In [73]:

```
print np.ones(10)
print np.zeros(5)
```

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

У каждого массива есть обязательные атрибуты, обозначающие его длину, форму, размерность, тип и т.д.

In [74]:

```
a = np.array([1,2,3,4])
print a.size, a.shape, a.ndim, a.dtype
```

```
4 (4L,) 1 int32
```

Примечание: Numpy arrays на самом деле поддерживают не только два целочисленных и один вещественный типы - они также поддерживают 8-, 16-, 32-, и 64-битные целые и 16-, 32-, и 64-битные вещественные числа, со знаком и без. Это можно использовать для увеличения скорости работы программы.

Массивы поддерживают набор стандартных операций:

In [75]:

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

print a + b
print a / b
print a * b
print a**b
```

```
[4 4 4]
[0 1 3]
[3 4 3]
[1 4 3]
```

Массивы можно также создавать многомерными:

In [76]:

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

Out[76]:

```
array([[ 1.,  2.],
       [ 3.,  4.]])
```

In [77]:

```
print x.size, x.shape, x.ndim, x.dtype
```

```
4 (2L, 2L) 2 float64
```

In [78]:

```
print np.ones([2, 3])
```

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

In [79]:

```
print np.ones([2, 3, 3])
```

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

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

В массивах numpy также можно получать значение по индексу и делать срезы.

In [80]:

```
a = np.arange(6)
print a[0], a[-1], a[2:5], a[::2]
```

```
0 5 [2 3 4] [0 2 4]
```

В случае многомерных масивов индексы можно перечислять в отдельных скобках или через запятую. Также можно выполнять срезы отдельно по каждой из осей:

In [81]:

```
a = np.zeros([6,6,6])
print a[0][1][2]
print a[0, 1, 2]
print a[::2, 1:3, -1:-4:-1]
```

```
0.0
0.0
[[[ 0.  0.  0.]
  [ 0.  0.  0.]]

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

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

Однако есть и более интересные способы работы с индексами. Например, можно передать список индексов:

In [82]:

```
a = np.arange(10)
print a[[0,2,4,6,8]]
a = np.ones([3, 4])
print a[:, [1, 2]]
```

```
[0 2 4 6 8]
[[ 1.  1.]
 [ 1.  1.]
 [ 1.  1.]]
```

Также можно передать массив из `True` и `False` той же длины (такой массив называется **маской**) в качестве индекса, который будет указывать, берем ли мы элемент на соответствующем месте или нет:

In [83]:

```
print np.arange(10)[np.array([True, False, True, False, True, False, True, False, Tru
```

```
[0 2 4 6 8]
```

Это не кажется очень удобным, однако маски можно очень легко получить например сравнением массива:

In [84]:

```
print np.arange(10) >= 5.
```

```
[False False False False False  True  True  True  True  True]
```

Используя такие маски мы получаем очень мощный способ доступа к элементам массива:

In [85]:

```
a = np.arange(10)
print a[a > 5.]
print a[(a > 5.) & (a < 9.)]
```

```
[6 7 8 9]
[6 7 8]
```

Также таким способом можно обновлять значения:

In [86]:

```
a = np.arange(10)
a[a%2 == 0] = -1
print a
```

```
[-1  1 -1  3 -1  5 -1  7 -1  9]
```

Помимо быстро работающих массивов в numpy есть специальные функции работы с ними, которые оптимизированы с помощью методов линейной алгебры и применяются сразу ко всему массиву (т.н. векторизованные функции):

In [87]:

```
incl = np.linspace(0., 2. * np.pi, 10)
print np.sin(incl)
```

```
[  0.00000000e+00   6.42787610e-01   9.84807753e-01   8.66025404e-01
   3.42020143e-01  -3.42020143e-01  -8.66025404e-01  -9.84807753e-01
  -6.42787610e-01  -2.44929360e-16]
```

In [88]:

```
a = np.arange(10)
print np.sum(a)
print np.max(a)
print np.abs(a)
print np.product(a)
```

```
45
9
[0 1 2 3 4 5 6 7 8 9]
0
```

Очень часто используется модуль `random`, позволяющий быстро создавать случайные наборы чисел с заданным распределением:

In [89]:

```
x = np.random.normal(3., 1., 10) #массив из 10 чисел с нормальным распределением со с
print x
```

```
[ 1.85878829  2.34744308  3.21386429  1.67294977  2.89732441  2.6784553
8
  3.23943631  4.38489085  2.33992159  3.9113281 ]
```

Полезными являются также статистические функции:

In [90]:

```
print np.mean(x)
print np.std(x)
print np.median(x)
```

```
2.85444020672
0.817222520154
2.78788989391
```

На этом краткое введение в Numpy завершено. Отметим еще два момента: в этом пакете есть отдельный модуль для работы с матрицами и решения линейных уравнений (модуль `linalg`). Второе и немаловажное - в numpy есть простые способы сохранить результат вычисления в виде

массива на диск и при необходимости не пересчитывать его заново, а просто прочитать (функции `np.save` и `np.load`).

Введение в Matplotlib

Когда начинаешь действительно серьезно заниматься вычислениями в numpy, становится неудобно печатать сотни значений и проверять их. Было бы удобно изображать эти значения на графиках и рисунках. Этому служит пакет matplotlib, являющийся основным пакетом в Python для этих целей.

Чтобы импортировать нужный модуль наберите

In [91]:

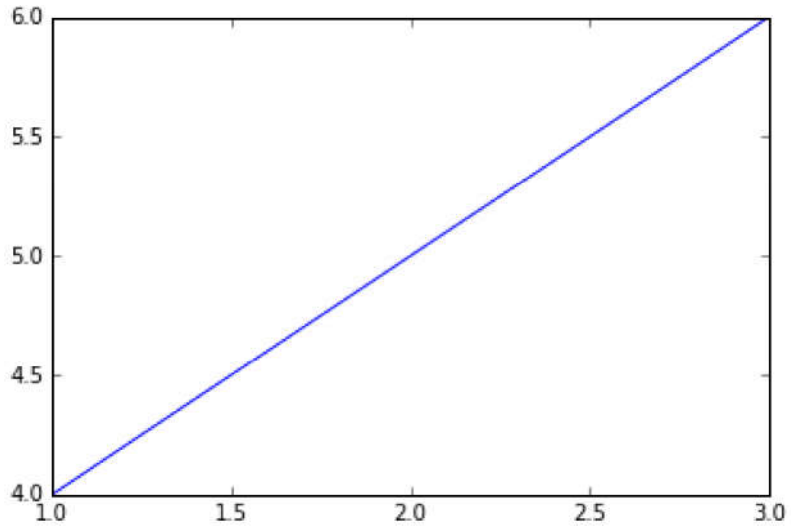
```
import matplotlib.pyplot as plt
```

Если вы используете IPython Notebook, то с помощью магической команды `%matplotlib inline` графики будут встраиваться внутрь среды. В противном случае они будут появляться в отдельном окне и исчезать после закрытия.

Главная функция для построения графика называется `plot`:

In [92]:

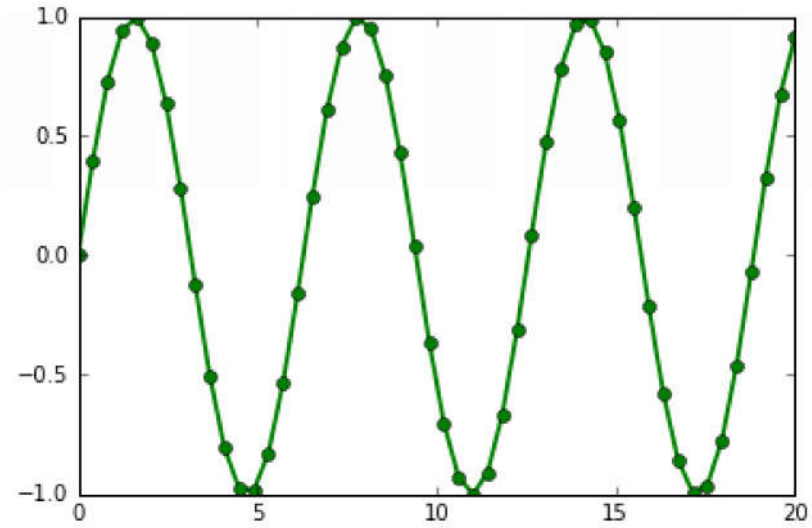
```
%matplotlib inline
plt.plot([1,2,3], [4,5,6])
plt.show() #эта команда выводит график на экран
```



Matplotlib также может принимать на вход numpy массивы:

In [93]:

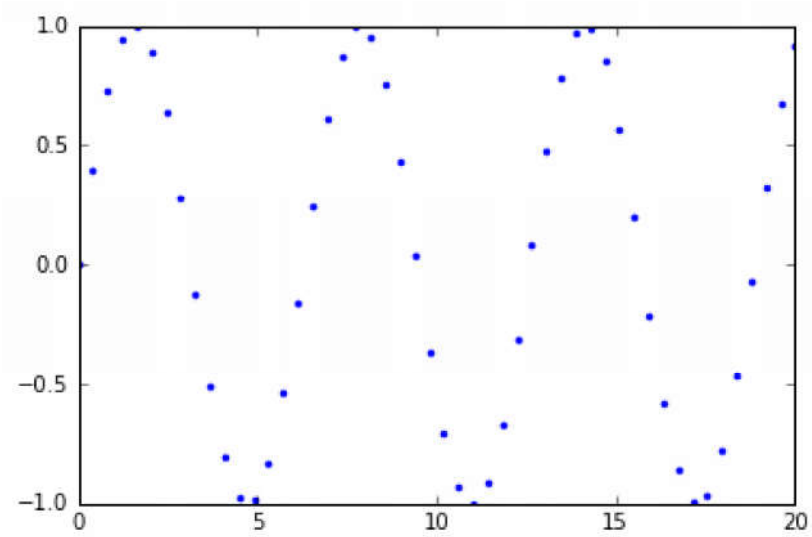
```
x = np.linspace(0., 20., 50)
y = np.sin(x)
plt.plot(x, y, marker='o', color='green', linewidth=2)
plt.show()
```



Обратите внимание, что мы также добавили дополнительные параметры - тип точки, цвет и ширину линии. Таких параметров достаточно много, например, если мы хотим оставить только точки:

In [94]:

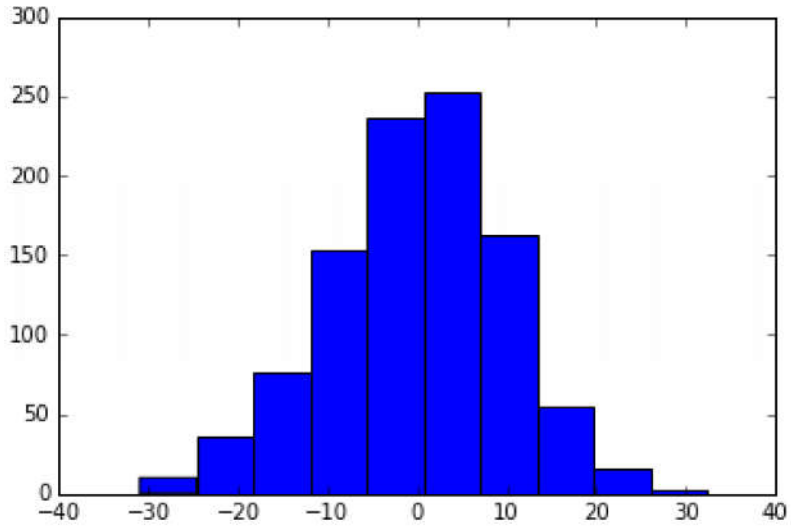
```
plt.plot(x, y, '.')
plt.show()
```



Считается, что графики только из точек лучше строить командой `plt.scatter()`, но это не принципиально. Из других востребованный методов, которые реализованы в matplotlib, хотелось бы отметить построение гистрограмм и изображений:

In [95]:

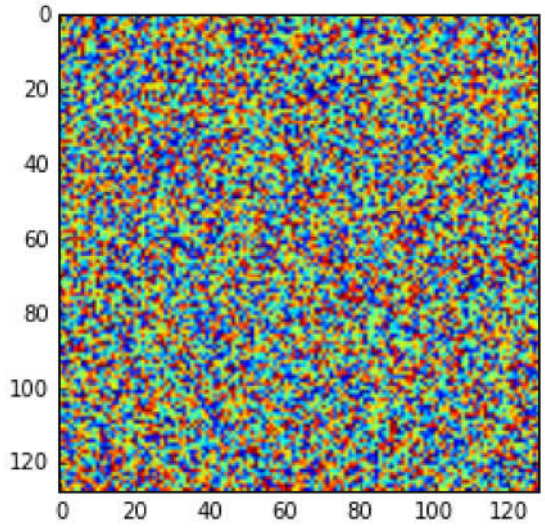
```
v = np.random.normal(0., 10., 1000)
plt.hist(v, bins=10) #10 столбцов или "бинов"
plt.show()
```



Для отображения изображений их сперва надо считать в двумерный numpy array и затем отобразить с помощью команды `imshow()`. Обычный двумерный массив тоже можно изобразить таким образом, например случайный шум:

In [96]:

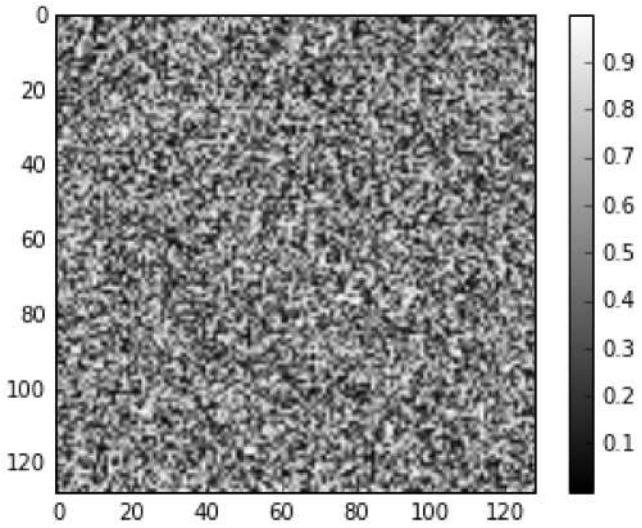
```
array = np.random.random((128, 128))
plt.imshow(array)
plt.show()
```



А вот так можно поменять цветовую гамму и добавить диапазон возможных значений отдельной шкалой:

In [97]:

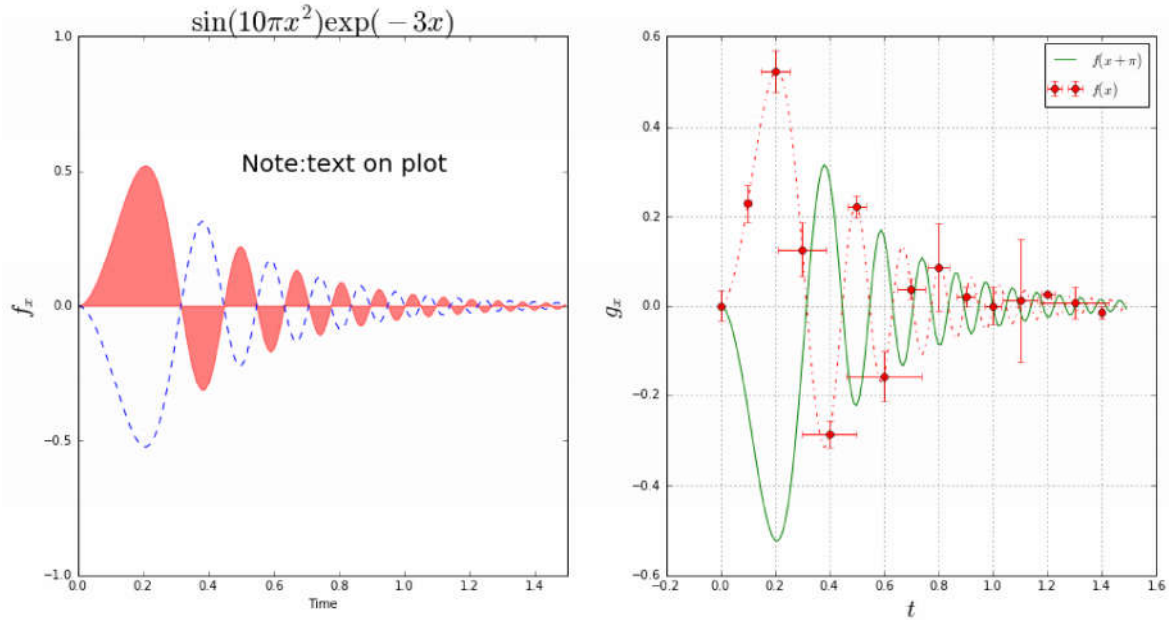
```
plt.imshow(array, cmap=plt.cm.gray)
plt.colorbar()
plt.show()
```



На следующем рисунке мы приведем набор полезных команд для настройки внешнего вида и некоторых параметров графика:

In [98]:

```
#создание картинки определенного размера с двумя отдельными изображениями (они лежат
fig, axes = plt.subplots(figsize=[16, 8], ncols = 2, nrows = 1)

t = np.arange(0.0, 1.5, 0.01)
s = np.sin(10*np.pi*t**2)*np.exp(-3*t)
s1 = np.sin(10*np.pi*t**2 + np.pi)*np.exp(-3*t)

axes[0].plot(t, s1, '--', color='b') #график прерывистой синей линией; обратите внима
axes[0].fill_between(t, s, color='red', alpha=0.5) #заливка красным цветом области ме

axes[0].set_xlim(0, 1.5) #ограничения на ось x; если картинка одна - то это plt.xlim(
axes[0].set_ylim(-1, 1) #ограничения на ось y
axes[0].set_xlabel('Time') #подписываем оси; если картинка одна - то это plt.xlabel(.
axes[0].set_ylabel('$f_x$', fontsize=20) #проверяем работу latex и изменяем размер на
axes[0].set_title('$\sin (10\pi x^2)\exp(-3x)$', fontsize=24) #заголовок изображения
axes[0].text(0.5, 0.5, 'Note:text on plot', fontsize=20) #текст внутри изображения

axes[1].plot(t, s1, '-', color='g', label='$f(x+\pi)$') #рисуем на втором изображении
xerror = 0.05*abs(np.random.randn(len(s))) #создаем массив ошибок
yerror = 0.05*abs(np.random.randn(len(s)))
#рисуем десятую часть всех точек с показанными ошибками, формат точек - красные круги
axes[1].errorbar(t[::10], s[::10], xerr=xerror[::10], yerr=yerror[::10], fmt='ro', la
axes[1].plot(t, s, '-.', color='r') #красная линия формата точка-тире
axes[1].grid() #сетка
axes[1].set_xlabel('$t$', fontsize=20) #подписываем оси
axes[1].set_ylabel('$g_x$', fontsize=20)
axes[1].legend() #добавляем легенду, подписи берет из значений label

plt.savefig('my_plot.png') #сохраняем полученное изображение
plt.show()
```



В основном это все, что нужно знать для начала работы с matplotlib. Отметим только еще возможность построения трехмерных графиков и контурных карт (изолиний). Примеры работ, выполненых в matplotlib можно посмотреть на сайте (http://matplotlib.org/gallery.html). Особый

интерес также представляет Matplotlib Gallery For Astronomy (ссылка (https://leejjoon.github.io/matplotlib_astronomy_gallery/)).

Обзор SciPy

Scientific Python (scipy) - это пакет с набором функций для интегрирования, приближения, интерполирования, анализа и преобразования функций. Включает в себя следующие модули:

- Специальные функции (scipy.special)
- Интегрирование (scipy.integrate)
- Оптимизация (scipy.optimize)
- Интерполяция (scipy.interpolate)
- Преобразование Фурье (scipy.fftpack)
- Обработка сигналов (scipy.signal)
- Линейная алгебра (scipy.linalg)
- Многомерные структуры и алгоритмы (scipy.spatial)
- Статистика (scipy.stats)
- Обработка изображений (scipy.ndimage)
- Работа с потоками входа-выхода (scipy.io)
- другое

Приведем некоторые примеры работы с функциями этого пакета:

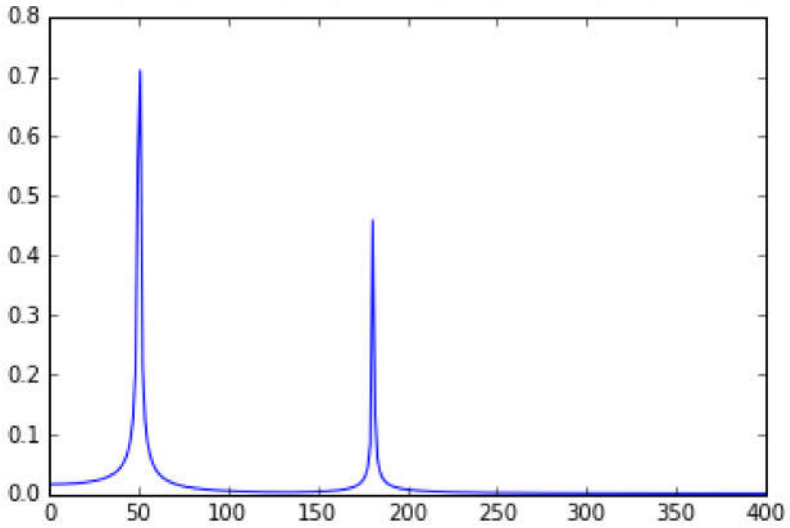
In [99]:

```
#численное интегрирование
from scipy import integrate
def f(x):
    return np.sin(x)

print integrate.quad(f, 0, np.pi)

a = np.arange(0, np.pi, 0.1)
print integrate.simps(f(a), a)
```

```
(2.0, 2.220446049250313e-14)
1.99913036628
```

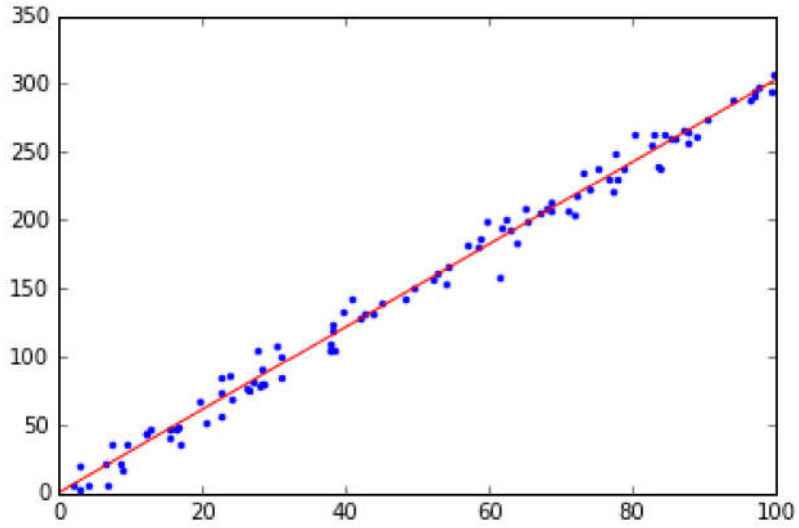
In [100]:

```
#быстрое преобразование Фурье
from scipy import fft
N = 600 #количество точек
T = 1.0 / 800.0 #расстояние между точками
x = np.linspace(0.0, N*T, N)
y = np.sin(50.0 * 2.0*np.pi*x) + 0.5*np.sin(180.0 * 2.0*np.pi*x)
yf = fft(y)
xf = np.linspace(0.0, 1.0/(2.0*T), N/2)
plt.plot(xf, 2.0/N * np.abs(yf[0:N/2]))
plt.show()
```



In [101]:

```
#приближение функций
from scipy import optimize
def f(x, a, b):
    return a * x + b

x = np.random.uniform(0., 100., 100)
y = 3.*x + 2. + np.random.normal(0., 10., 100)

popt, pcov = optimize.curve_fit(f, x, y)

plt.plot(x, y, '.')
xdata = np.linspace(0., 100., 100)
plt.plot(xdata, f(xdata, popt[0], popt[1]), 'r-')
plt.show()
```



Это самые простые примеры из того, для чего можно использовать SciPy. За всеми подробностями применения пакета отсылаем к документации (https://docs.scipy.org/doc/scipy/reference/).

Отдельные полезные пакеты

Как уже было упомянуто, я не ставил своей целью в этом методичесом руководстве привести примеры использования всех полезных пакетов. Поэтому я дам список других (помимо рассмотренных выше) часто используемых в научной работе пакетов, после чего мы перейдем к обсуждению астрономических задач.

- statsmodels (http://statsmodels.sourceforge.net/) - полезный пакет для продвинутой статистики
- pandas (http://pandas.pydata.org/) - пакет для анализа данных в форме таблиц
- sympy (http://www.sympy.org/en/index.html) - пакет сивольных вычислений, например для упрощения формул
- seaborn (https://web.stanford.edu/~mwaskom/software/seaborn/) - надстройка над matplotlib для более красивых графиков
- scikit-learn (http://scikit-learn.org/stable/) - набор методов для машинного обучения

Python в астрономии

Это последняя часть данного руководства, здесь мы кратко коснемся чисто астрономических задач, которые можно решать с помощью python.

Астрономические пакеты

Начнем с неполного списка астрономических пакетов (более полный можно найти например на сайте http://www.astropython.org/packages/ (http://www.astropython.org/packages/)):

- Astropy (http://www.astropy.org/) - наиболее важный и полный пакет астрономических утилит, попытка сообщества объединить все самое нужное
- Pyfits (http://www.stsci.edu/institute/software_hardware/pyfits) - работа с изображениями и таблицами FITS (сейчас перенесен в astropy)
- Astrolib modules (https://www.stsci.edu/trac/ssb/astrolib) - астрономические пакеты для работы с системами координат и ascii-таблицами
- pyephem (http://rhodesmill.org/pyephem/) - работа с небесными координатами и эфемеридами различных небесных тел
- PyRAF (http://www.stsci.edu/institute/software_hardware/pyraf) - командный язык для работы с IRAF из Python
- APLpy (http://aplpy.sourceforge.net/) - работа с астрономическими изображениями
- ATpy (http://atpy.sourceforge.net/) - работа с астрономическими таблицами (Fits, VOtable, IPAC, SQL, ...)
- Asciitable (http://cxc.harvard.edu/contrib/asciitable/) - чтение ASCII таблиц
- Kapteyn Package (http://www.astro.rug.nl/software/kapteyn/) - большая и хорошо документированная коллекция библиотек от Kapteyn Astronomical Institute. Включает в себя работу с координатами, таблицами и изображениями
- astLib (http://astlib.sourceforge.net/) - астрономический пакет с модулями для работы со статистикой, изображениями, таблицами и т.д.
- pywcsgrid2 (http://leejjoon.github.com/pywcsgrid2/) - надстройка над matplotlib с астрономическим уклоном
- SAMPy (http://cosmos.iasf-milano.inaf.it/pandora/sampy.html) - реализация протокола SAMP в Python (для общения астрономического софта между собой)
- pyregion (http://leejjoon.github.com/pyregion/) - парсер ds9.
- CosmoPy (http://roban.github.com/CosmoloPy/) - космологические вычисления
- idlsave (http://idlsave.sourceforge.net/) - чтение данных IDL в numpy
- работа с ds9 из-под питона (http://hea-www.harvard.edu/saord/ds9/pyds9/)
- python-sao (http://code.google.com/p/python-sao/) - еще один интерфейс к ds9
- Astropysics (http://packages.python.org/Astropysics/) - набор астрофизических утилит на питоне
- cosmics.py (http://obswww.unige.ch/~tewes/cosmics_dot_py/) - детектирование космических лучей (L.A.Cosmic algorithm)
- Alipy (http://obswww.unige.ch/~tewes/alipy/) - пакет для работы с FITS (использует pyraf и sextractor)
- python-montage (http://python-montage.sourceforge.net/) - оболочка для построения мозаичных изображений

- Chantipy (http://chiantipy.sourceforge.net/) - интерфес на питоне для доступа к CHIANTI (http://www.chiantidatabase.org/) базе данных астрофизической спектроскопии
- pyTMP (http://phn.github.com/pytpm/) - интерфейс к TPM (Telescope Pointing Machine)
- SunPy (http://www.sunpy.org/): солнечная физика
- AsPyLib (http://www.aspylib.com/): обработка FITS изображений (включая фотометрию и астрометрию)
- astroquery (http://www.astropy.org/astroquery/) - онлайн доступ к астрономическим данным (NED, VizieR, SIMBAD, ADS и т.д.)
- pySpecKit (http://pyspeckit.bitbucket.org): астрономический анализ спектров
- PEP (http://vivaldi.ll.iac.es/sieinvens/siepedia/pmwiki.php?n=HOWTOs.PythonAndCLibraryForSourceExtractionAndPhotometry?action=edit) - фотометрия и извлечение изображений
- Моделирование оптики в пакете poppy (https://github.com/mperrin/poppy)
- https://github.com/andrewpaulreeves/pyAOS (http://vivaldi.ll.iac.es/sieinvens/siepedia/pmwiki.php?n=HOWTOs.PyAOS?action=edit) - моделирование адаптивной оптики методами Монте-Карло
- padova (https://pypi.python.org/pypi/padova) - анализ эволюционных звездных треков
- yt (http://yt-project.org/doc/intro/index.html) - анализ звездных гало, SFR, создание синтетических наблюдений

Работа с координатами (ephem)

Пакет ephem (pyephem) предназначен для упрощения работы с координатами и эфемеридами, их переводу из одной системы в другую. Приведем некоторые примеры работы с ним:

In [102]:

```
import ephem
s = ephem.Saturn()
s.compute('2015/10/3')
print s.ra, s.dec
```

```
15:57:42.38 -18:40:51.9
```

In [103]:

```
#экваториальные координаты
eq = ephem.Equatorial('12:22:12.5','-59:15:59.4')
print eq.ra, eq.dec, eq.epoch
```

```
12:22:12.50 -59:15:59.4 2000/1/1 00:00:00
```

In [104]:

```
#найти созвездие по координатам
ephem.constellation((eq.ra,eq.dec))
```

Out[104]:

```
('Cru', 'Crux')
```

In [105]:

```
#галактические координаты
ga = ephem.Galactic('2:34:25.5','+79:39:59.9')
print ga.lon, ga.lat
```

```
2:34:25.5 79:39:59.9
```

Перевод координат:

In [106]:

```
#экваториальных между различными эпохами
eq1 = ephem.Equatorial('12:22:12.5','-59:15:59.4')
eq2 = ephem.Equatorial(eq1,epoch=ephem.B1950)
print eq1.get()
print eq2.get()
```

```
(3.2384947881015616, -1.0343955611332194)
(3.2265773908139925, -1.0295565421598851)
```

In [107]:

```
#из галактических в экваториальные
ga1 = ephem.Galactic('0','90',epoch=ephem.J2000)
eq1 = ephem.Equatorial(ga1)
print eq1.ra, eq1.dec
```

```
12:51:26.28 27:07:41.7
```

In [108]:

```
#вычисление юлианской даты
ephem.julian_date('2001/1/1')
```

Out[108]:

```
2451910.5
```

Astropy: все в одном

Astropy - это попытка астрономического сообщества собрать все необходимые астрономические утилиты в одном пакете. Пакет большой и содержит в себе много функций для анализа, моделирования, работы с координатами, FITS изображениями и физическими величинами. Последние три задачи мы и рассмотрим.

Анализ координат в astropy выполняется с помощью функции SkyCoord и поддерживает следующие системы : ICRS, FK4, FK5, Galactic и AltAz.

In [109]:

```
from astropy.coordinates import SkyCoord
```

In [110]:

```
SkyCoord('00h42m44.3s +41d16m9s')
```

Out[110]:

```
<SkyCoord (ICRS): (ra, dec) in deg
    (10.68458333, 41.26916667)>
```

In [111]:

```
#Координаты можно получить по имени объекта
SkyCoord.from_name("M18")
```

Out[111]:

```
<SkyCoord (ICRS): (ra, dec) in deg
    (274.9917, -17.1017)>
```

Чтобы получить координаты объекта по имени, astropy отправляет строковый запрос в Sesame (http://cds.u-strasbg.fr/cgi-bin/Sesame).

In [112]:

```
SkyCoord.from_name('NGC 338')
```

Out[112]:

```
<SkyCoord (ICRS): (ra, dec) in deg
    (15.1517083, 30.6689194)>
```

In [113]:

```
#в галактических координатах
SkyCoord.from_name("M51", frame='galactic')
```

Out[113]:

```
<SkyCoord (Galactic): (l, b) in deg
    (104.85158472, 68.56070181)>
```

In [114]:

```
c = SkyCoord.from_name("M32", frame='icrs')

print c
#различные способы извлечения координат
print c.ra.hms, c.dec.hms
print c.ra.degree, c.dec.degree
print c.ra.radian, c.dec.radian
print c.ra.hour
```

```
<SkyCoord (ICRS): (ra, dec) in deg
    (10.6742708, 40.8651694)>
hms_tuple(h=0.0, m=42.0, s=41.82499200000052) hms_tuple(h=2.0, m=43.0,
s=27.640656000002366)
10.6742708 40.8651694
0.186301170709 0.713231755415
0.711618053333
```

In [115]:

```
#если использовать Ipython Notebook, то координаты выглядат достаточно красиво
c.ra
```

Out[115]:

$10^\circ 40' 27.3749''$

In [116]:

```
#переход в другую систему координат
c.transform_to('galactic')
```

Out[116]:

```
<SkyCoord (Galactic): (l, b) in deg
    (121.1500176, -21.97633425)>
```

Перейдем теперь к работе с FITS файлами. Ее условно можно разделить на три части: работа с непосредственно изображением, его заголовками и с таблицами. Здесь мы рассмотрим только работу с изображением и заголовками.

In [117]:

```
from astropy.utils.data import download_file

#загружаем изображение NGC 3245 с SDSS DR9 в полосе U
n3245 = download_file( 'http://dr9.mirror.sdss3.org./sas/dr12/boss/photoObj/frames/30
```

```
Downloading http://dr9.mirror.sdss3.org./sas/dr12/boss/photoObj/frames/
301/5079/2/frame-u-005079-2-0045.fits.bz2 (http://dr9.mirror.sdss3.or
g./sas/dr12/boss/photoObj/frames/301/5079/2/frame-u-005079-2-0045.fits.
bz2) [Done]
```

In [118]:

```
from astropy.io import fits
hdulist = fits.open(n3245) #открытие FITS файла
```

In [119]:

```
hdulist.info()
```

```
Filename: c:\users\root\appdata\local\temp\tmpwkj4gh
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU      96   (2048, 1489)   float32
1                ImageHDU         6   (2048,)      float32
2                BinTableHDU     27   1R x 3C      [49152E, 2048E, 1489
E]
3                BinTableHDU     79   1R x 31C     [J, 3A, J, A, D, D,
2J, J, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D, D,
E, E]
```

Как мы видим, это файл содержит несколько кубов данных. Нас интересует первый куб, давайте посмотрим на его заголовки:

In [120]:

```
hdu = hdulist[0]
hdr = hdu.header
hdr[0:10] #первые 10 заголовков
```

Out[120]:

```
SIMPLE  =                    T /
BITPIX  =                  -32 / 32 bit floating point
NAXIS   =                    2
NAXIS1  =                 2048
NAXIS2  =                 1489
EXTEND  =                    T /Extensions may be present
BZERO   =              0.00000 /Set by MRD_SCALE
BSCALE  =              1.00000 /Set by MRD_SCALE
TAI     =        4610251581.14 / 1st row - Number of seconds since Nov
17 1858
RA      =            156.58050 / 1st row - Right ascension of telescope
boresigh
```

In [121]:

```
#мы можем получить отдельные заголовки, обратившись по их имени
hdr['INCL']
```

Out[121]:

```
32.5
```

In [122]:

```
#точно так же можно изменить заголовки, давайте запишем правильное имя объекта
hdr['OBJECT'] = "NGC3245"
print hdr['OBJECT']
```

```
NGC3245
```

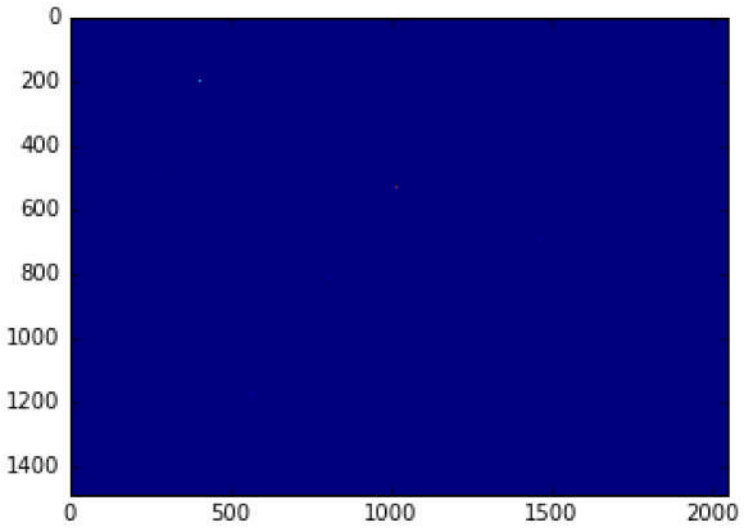
In [123]:

```
#обращение к данным происходит через поле data; данные - это массив
image_data = hdulist[0].data
print(type(image_data))
print(image_data.shape)
```

```
<type 'numpy.ndarray'>
(1489L, 2048L)
```

In [124]:

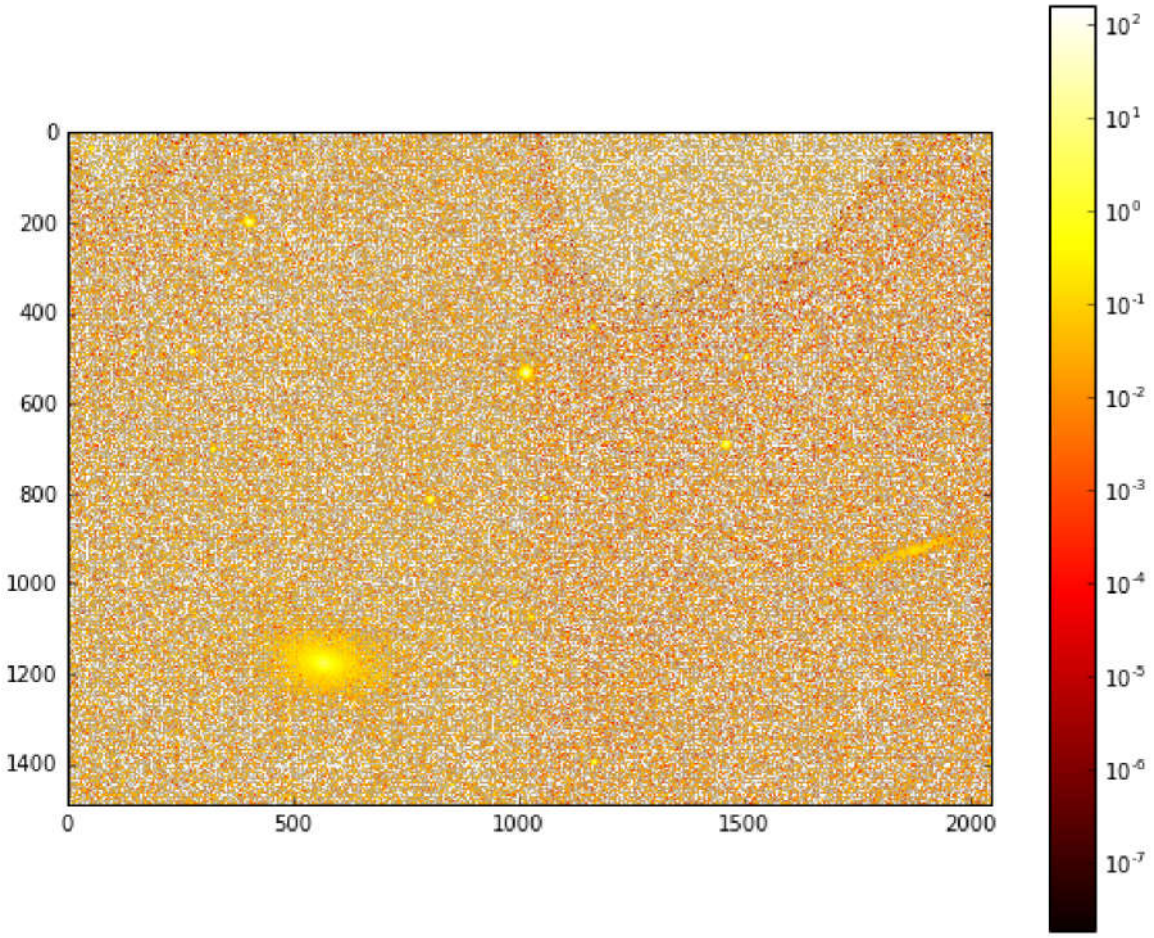
```
#как мы уже знаем, для изображения массива используется функция imshow
plt.imshow(image_data)
plt.show()
```



Как и ожидалось - ничего не видно, попробуем нарисовать в логарифмическом масштабе (обратите внимание, что числа по осям - это номер пикселя изображения):

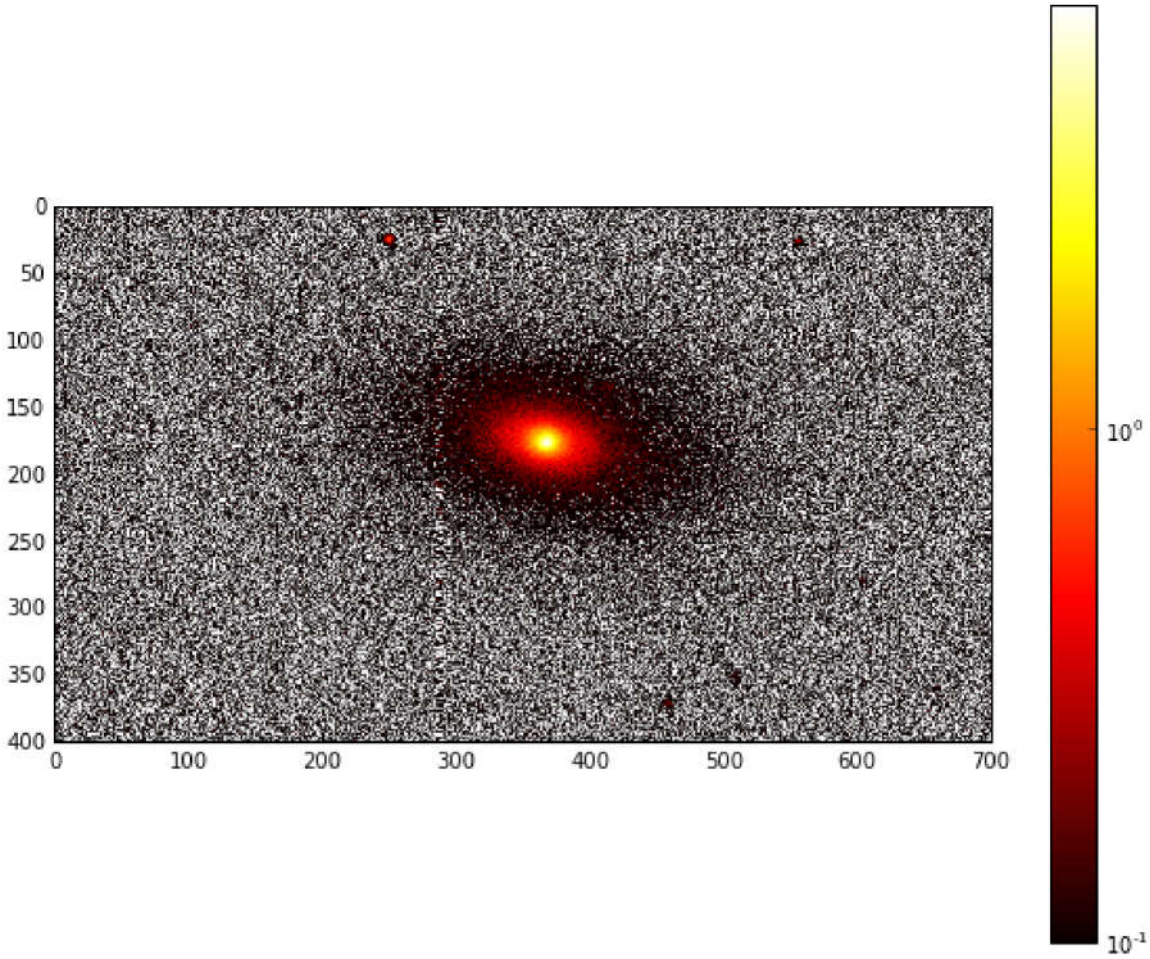
In [125]:

```
from matplotlib.colors import LogNorm

fig = plt.figure(figsize=[10,8])
plt.imshow(image_data, cmap='hot', norm=LogNorm())
cbar = plt.colorbar()
plt.show()
```



In [126]:

```
#нарисуем только левый нижний угол и зададим границы отображения
fig = plt.figure(figsize=[10,8])
plt.imshow(image_data[1000:1400, 200:900], cmap='hot', norm=LogNorm(), vmin=0.1)
cbar = plt.colorbar()
plt.show()
```



In [127]:

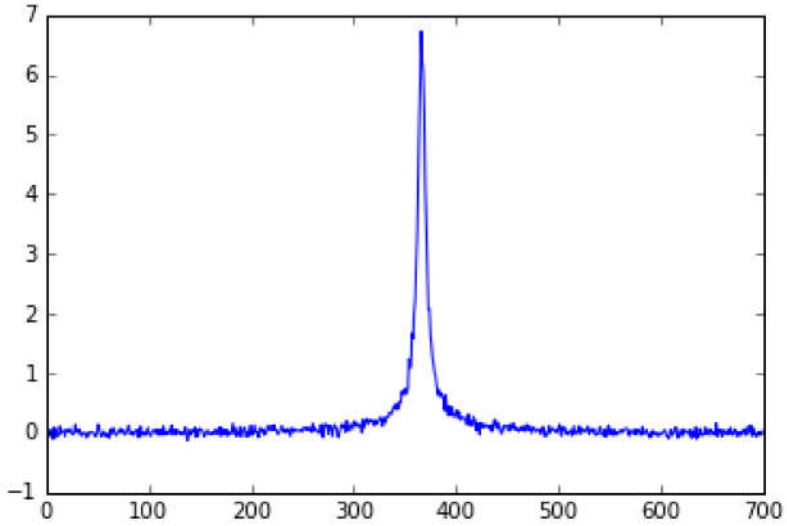
```
#перезапишем изображение; работа с ним ничем не отличается от работы с numpy array
image_data = image_data[1000:1400, 200:900]

#базовые параметры изображения
print 'Min:', np.min(image_data)
print 'Max:', np.max(image_data)
print 'Mean:', np.mean(image_data)
print 'Std:', np.std(image_data)
```

```
Min: -0.239014
Max: 6.72656
Mean: 0.0212397
Std: 0.116012
```

Построим разрез вдоль оси X изображения:

In [128]:

```
#позиция максимума
maxarg = np.unravel_index(image_data.argmax(), image_data.shape)

plt.plot(range(len(image_data[0])), image_data[maxarg[0], :])
plt.show()
```



In [131]:

```
#неконец, чтобы записать файл ипользуется функция writeto
#чтобы перезаписать существующий файл необходимо использовать параметр clobber=True
fits.writeto('output_file.fits', image_data, hdr, clobber=True)
```

Последняя часть рассмотрения пакета astropy будет посвящена работе с физическими величинами. Благодаря модулю `astropy.units` числам и целым массивам чисел можно давать физическую размерность, что упрощает работу с ними и позволяет реже ошибаться.

In [132]:

```
from astropy import units as u
u.pc #размерность - парсек
```

Out[132]:

pc

In [133]:

```
u.N, u.N == u.Newton
```

Out[133]:

(Unit("N"), True)

In [134]:

```
u.m / u.kg / u.s**2
```

Out[134]:

$$\frac{\mathrm{m}}{\mathrm{kg\ s^2}}$$

Наиболее полезно использовать физические величины вместе с числами, создавая объект `Quantity`:

In [135]:

```
3 * u.kg
```

Out[135]:

$$3\ \mathrm{kg}$$

In [136]:

```
np.array([1.2, 2.2, 1.7]) * u.pc / u.year
```

Out[136]:

$$[1.2,\ 2.2,\ 1.7]\ \frac{\mathrm{pc}}{\mathrm{yr}}$$

In [137]:

```
q1 = 10. * u.m
q2 = 22. * u.cm / u.s / u.g**2
q1 * q2
```

Out[137]:

$$220\ \frac{\mathrm{cm\ m}}{\mathrm{s\ g^2}}$$

In [138]:

```
#перевод в другие единицы
(q1 * q2).to(u.m**2 / u.kg**2 / u.s)
```

Out[138]:

$$2200000\ \frac{\mathrm{m^2}}{\mathrm{s\ kg^2}}$$

In [139]:

```
#упрощение производится методом decompose
(q1 * q2).decompose()
```

Out[139]:

$$2200000\ \frac{\mathrm{m^2}}{\mathrm{s\ kg^2}}$$

In [140]:

```
#по умолчанию используется перевод в СИ, но можно в другие системы, например СГС
(q1 * q2).decompose(u.cgs.bases)
```

Out[140]:

$22000 \ \frac{\mathrm{cm}^2}{\mathrm{s}\ \mathrm{g}^2}$

In [141]:

```
#можно передавать в функции Numpy
q = np.sin(30 * u.degree)
print type(q), q
```

```
<class 'astropy.units.quantity.Quantity'> 0.5
```

In [142]:

```
#получение значения и размерности
q.value, q.unit
```

Out[142]:

```
(0.49999999999999994, Unit(dimensionless))
```

Еще один очень полезный модуль - `astropy.constants`, в котором, как нетрудно догадаться, содержатся физические константы. Давайте вычислим с помощью этого модуля притяжение Земли Солнцем:

In [143]:

```
from astropy.constants import G

F = (G * 1. * u.M_sun * 1. * u.M_earth) / (1. * u.au)**2
F
```

Out[143]:

$6.67384 \times 10^{-11} \ \frac{\mathrm{m}^3\ \mathrm{M}_\oplus\ \mathrm{M}_\odot}{\mathrm{kg}\ \mathrm{AU}^2\ \mathrm{s}^2}$

In [144]:

```
#переведем в Ньютоны
F.to(u.N)
```

Out[144]:

$3.5437358 \times 10^{22} \ \mathrm{N}$

Отметим также, что иногда величины не переводятся напрямую друг в друга и необходимо использовать дополнительные условия:

In [145]:

```
(1234. * u.nm).to(u.GHz)
```

```
---------------------------------------------------------------------
----
UnitConversionError                       Traceback (most recent call l
ast)
<ipython-input-145-d89e2d67b3df> in <module>()
----> 1 (1234. * u.nm).to(u.GHz)

C:\Anaconda\lib\site-packages\astropy\units\quantity.pyc in to(self, un
it, equivalencies)
    632         unit = Unit(unit)
    633         new_val = self.unit.to(unit, self.view(np.ndarray),
--> 634                                equivalencies=equivalencies)
    635         return self._new_view(new_val, unit)
    636

C:\Anaconda\lib\site-packages\astropy\units\core.pyc in to(self, other,
value, equivalencies)
    966             If units are inconsistent
    967         """
--> 968         return self._get_converter(other, equivalencies=equival
encies)(value)
    969
    970     def in_units(self, other, value=1.0, equivalencies=[]):

C:\Anaconda\lib\site-packages\astropy\units\core.pyc in _get_converter
(self, other, equivalencies)
    867         except UnitsError:
    868             return self._apply_equivalencies(
--> 869                 self, other, self._normalize_equivalencies(equi
valencies))
    870         return lambda val: scale * _condition_arg(val)
    871

C:\Anaconda\lib\site-packages\astropy\units\core.pyc in _apply_equivale
ncies(self, unit, other, equivalencies)
    858         raise UnitConversionError(
    859             "{0} and {1} are not convertible".format(
--> 860                 unit_str, other_str))
    861
    862     def _get_converter(self, other, equivalencies=[]):

UnitConversionError: 'nm' (length) and 'GHz' (frequency) are not conver
tible
```

In [146]:

```
(1234. * u.nm).to(u.GHz, equivalencies=u.spectral())
```

Out[146]:

242943.65 GHz

In [147]:

```
(13.7 * u.eV).to(u.nm, equivalencies=u.spectral())
```

Out[147]:

90.499411 nm

Вместо заключения

Я очень надеюсь, что данное обзорное руководство сподвигнет вас на дальнейшее изучение языка Python. Невозможно описать, какое облегчение я испытал, когда вместо программ на C и Fortran с графиками в Gnuplot я начал работать с этим языком. Скорость работы и проверки гипотез выросла во много раз, а болезненных ошибок стало меньше. Нельзя не отметить и элегантность языка, его лаконичность и эстетику. Астрономическое сообщество (и не только оно) не даром отдает в большинстве своем предпочтение именно детищу Гвидо ван Россума. Попробуйте его и вы.

Использованные источники и полезная литература

- Лучший сайт о питоне для астрономов: http://www.astropython.org/ (http://www.astropython.org/)
- http://www.astrobetter.com/ (http://www.astrobetter.com/) 'Tips and Tricks for Professional Astronomers' - полезный блог
- Книга: Numerical Methods in Engineering with Python (Jaan Kiusalaas, Cambridge University Press, август 2013)
- Книга: Learning Python, 5th Edition (Mark Lutz, O'Reilly Media, июнь 2013)
- Книга: Python in a Nutshell, 2nd Edition (Alex Martelli, O'Reilly Media, 2006)
- Книга: Practical programming (J. Elkner, A.Downey, et. al, 2011)
- Книга: The Python standard library by example (D. Hellmann, Addison-Wesley, 2011)
- Книга: Python Cookbook (D.Beazley, B. Jones, O'Reilly, 2013)
- Книга: Python for data analysis (W. McKinney, O'Reilly, 2012)
- Официальная документация: Python 2 http://docs.python.org/2/ (http://docs.python.org/2/) и Python 3 http://docs.python.org/3/ (http://docs.python.org/3/)
- Больше книг: https://wiki.python.org/moin/PythonBooks (https://wiki.python.org/moin/PythonBooks) и https://wiki.python.org/moin/IntroductoryBooks (https://wiki.python.org/moin/IntroductoryBooks)
- Материалы STFC summer school in astronomy (https://sites.cardiff.ac.uk/astronomy-summer-school/), Cardiff University, август 2015
- Practical Python for Astronomers с примерами и заданиями: https://python4astronomers.github.io/ (https://python4astronomers.github.io/)
- Работа со звездными изохронами: http://nbviewer.jupyter.org/github/jonathansick/padova/blob/master/notebooks/demo.ipynb (http://nbviewer.jupyter.org/github/jonathansick/padova/blob/master/notebooks/demo.ipynb)
- Работа с астрономическими координатами: http://balbuceosastropy.blogspot.ru/2013/09/working-with-astronomical-coordinate.html

(http://balbuceosastropy.blogspot.ru/2013/09/working-with-astronomical-coordinate.html)

- Всеобъемлющий список полезных пакетов: http://www.iac.es/sieinvens/siepedia/pmwiki.php?n=HOWTOs.EmpezandoPython (http://www.iac.es/sieinvens/siepedia/pmwiki.php?n=HOWTOs.EmpezandoPython)

Информация о версиях пакетов, использовавшихся в этом руководстве:

In [148]:

```
%load_ext version_information
%version_information numpy, matplotlib, scipy, ephem, astropy
```

Out[148]:

Software	Version
Python	2.7.11 64bit [MSC v.1500 64 bit (AMD64)]
IPython	4.0.1
OS	Windows 7 6.1.7601 SP1
numpy	1.10.4
matplotlib	1.5.1
scipy	0.17.0
ephem	3.7.5.3
astropy	1.1.2
Mon Apr 18 10:12:52 2016 RTZ 2 (зима)	