Компьютерные технологии в математическом моделировании

Лекция 3.

Ассистент кафедры
математической физики
к.ф.-м.н.
Татьяна Евгеньевна Романенко

Содержание лекции

- ООП
- NumPy
- Matplotlib
- GUI

Объектно-ориентированное программирование

```
class ClassName(BaseClass1,…,ModuleName.BaseClassM):
    <instruction1>
    …
    <instructionN>
```

- Имя базового класса должно быть определено в области видимости производного класса.
- Производные классы могут перегружать методы своих базовых классов (все методы виртуальные(C++))
- Вызов базового класса через BaseClass.Method(self, arguments), super()
- Проверка наследования:
  - isinstance(inst, type)
  - issubclass(type1, type2)
- Рекурсивный порядок нахождения атрибутов (вглубь, слева-направо) (ромбовидная иерархия)

Определение класса

- Объекты-классы поддерживают 2 вида операций:
  - Ссылки на атрибуты
  - Создание экземпляра
- Определение класса:
  - методы: def method_name(self, args)
  - данные: var = value
  - конструктор: def __init()__(self, args)
  - деструктор: def __del()__(self)
  - приватный атрибут: __var = value
  - методы: def method_name(self, args)
  - Метод при выводе объекта: def __str__(self)
- Атрибуты объектов-классов:
  - __name__ - имя класса
  - __module__ - имя модуля
  - __dict__ - словарь атрибутов класса (может изменяться напрямую)
  - __bases__ - кортеж базовых классов в порядке следования
  - __doc__ - строка документации класса
- Ссылки на атрибуты

Атрибуты объектов-классов: пример

```
>>> from person import Person
>>> bob = Person('Bob Smith')

>>> print(bob)                  # Вызов метод __str__ объекта bob
[Person: Bob Smith, 0]

>>> bob.__class__               # Выведет класс объекта bob и его имя
<class 'person.Person'>
>>> bob.__class__.__name__
'Person'

>>> list(bob.__dict__.keys())  # Атрибуты - это действительно ключи словаря
['pay', 'job', 'name']         # Функция list используется для получения
                               # полного списка в версии 3.0
>>> for key in bob.__dict__:
        print(key, '=>', bob.__dict__[key])    # Обращение по индексам

pay => 0
job => None
name => Bob Smith

>>> for key in bob.__dict__:
        print(key, '=>', getattr(bob, key))   # Аналогично выражению obj.attr,
                                              # где attr - переменная
pay => 0
job => None
name => Bob Smith
```

Пример создания и наследования

```
class BaseClass:
    """Base class for master students lesson"""
    _age = 0
    _job = 'base worker'
    _name = ' '
    def __init__(self, age, name):
        self._age = age
        self._name = name

    def work(self, hours):
        while hours > 0:
            print('Doing my base work')
            hours = hours -1

    def print_info(self):
        print('Age: ', self._age, ', name: ', self._name,', job: ',
self._job)
```

```
class JuniorClass(BaseClass):
    """Junior derived class for master students lesson"""
    def __init__(self, age, name):
        self._job = 'junior'
        super().__init__(age, name)

    def work(self, hours):
        while hours > 0:
            print(hours, ' hours till the end of the day')
            hours = hours -1
        print('On my way home!')
```

```
bw = BaseClass(35, 'Alex')
jw = JuniorClass(21, 'Peter')

workers = [bw, jw]

for w in workers:
    w.work(3)

for w in workers:
    w.print_info()
```

```
Doing my base work
Doing my base work
Doing my base work
3  hours till the end of the day
2  hours till the end of the day
1  hours till the end of the day
On my way home!
Age:  35 , name:  Alex , job:  base worker
Age:  21 , name:  Peter , job:  junior
```

Определение класса

- Атрибуты объектов-классов:
  - __getattr____setattr__ - возвращает\присваивает атрибут недоступный обычным способом
  - __delattr__ - удаляет атрибут
  - __getitem__ - получение элемента по индексу\ключу
  - __setitem__ - присваивание элемента по индексу\ключу
  - ...
- Статические методы

```
class CWSM:
 """Just class with static method"""
 def smethod(x):
    print(x)

 @staticmethod
 def smethod2(x):
    print(x, ' too')

 smethod = staticmethod(smethod)

CWSM.smethod(5)
CWSM.smethod2(10)
m = CWSM()
m.smethod(5)
```

```
5
10  too
5
```

NumPy

- Многомерные массивы
- Создание
- Базовые операции
- Индексы, итерации, срезы
- Изменение формы
- Копирование массивов
- NumPy для пользователей MATLAB

NumPy

- Основной объект: однородный многомерный массив numpy.ndarray:
  - **ndarray.ndim** – число измерений
  - **ndarray.shape** – форма массива
  - **ndarray.size** – число элементов массива
  - **ndarray.dtype** – объект, описывающий тип элементов массива (numpy.int32, numpy.int16, numpy.complex32, numpy.float64)
  - **ndarray.itemsize** – размер каждого элемента массива в байтах
  - **ndarray.data** – буфер, содержащий данные
- Индексация кортежем положительных чисел
- Axes and rank
  - [1,2,3] : 1 axis (length 3)
- ndarray vs array.array

NumPy: создание массивов

- Из списка или кортежа, используя **numpy.array**:
  - Последовательности -> 1D
  - Последовательности последовательностей -> 2D
  - Последовательности последовательностей последовательностей -> 3D
- С возможностью явного задания типа массива при создании
  - Аргумент dtype = …
- Создание инициализированных массивов (тип по умолчанию – float64)
  - zeros, ones, empty, full, eye, ***_like
- Аналог range для создания массивов-сеток:
  - arange(start, end, step), linspace(start, end, number)
- Создание массивов:
  - fromfile, fromfunction, random.rand, random.randn, load, save, savez,savetxt, loadtxt
- Создание массивов созданного структурированного типа
  - dtype

NumPy: печать массивов

- Последняя координата (axis) слева направо
- Предпоследняя координата (axis) сверху вниз
- Остальные: сверху вниз с разделением пустой строкой

NumPy: создание массивов

```
a1 = np.array([1, 2, 3, 4, 5])
a2 = np.array([[1, 2, 3], [4, 5, 6]])
a3 = np.array([1.0, 2, 3,  4, 5],
        dtype=np.complex)
a4 = np.zeros((3,3))
a5 = np.ones((3,3))
a6 = np.eye((3))
a7 = np.empty((3,4))
a8 = np.full((3,4), 7.40)
```

```
[1 2 3 4 5]
[[1 2 3]
 [4 5 6]]
[ 1.+0.j  2.+0.j  3.+0.j  4.+0.j  5.+0.j]
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
```

```
[[ 7.4  7.4  7.4  7.4]
 [ 7.4  7.4  7.4  7.4]
 [ 7.4  7.4  7.4  7.4]]
```

NumPy: создание массивов

```
a9 = np.arange(5, 15, 5)
a10 = np.arange(0, 1.0, 0.3)
a11 = np.linspace(0, 1.0, 11)

x = np.linspace(0, np.pi, 10)
y = np.sin(x)

z = np.fromfunction(
lambda i, j: i + j, (3,2),
dtype = int)
```

```
[ 5 10]
[ 0.   0.3  0.6  0.9]
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8
  0.9  1. ]
[ 0.         0.34906585  0.6981317
  1.04719755  1.3962634   1.74532925
  2.0943951   2.44346095  2.7925268
  3.14159265]
[  0.00000000e+00   3.42020143e-01
   6.42787610e-01   8.66025404e-01
   9.84807753e-01   9.84807753e-01
   8.66025404e-01   6.42787610e-01
   3.42020143e-01   1.22464680e-16]
[[0 1]
 [1 2]
 [2 3]]
```

NumPy: создание массивов

```
custom_type = np.dtype([('time', [('min', int), ('sec', int)]),
                        ('value', float)])

x = np.zeros((1,), dtype=custom_type)
x['time']['min'] = 5
x['time']['sec'] = 11
x['value'] = 113.84

fname = 'testfile'
fname2 = 'testfile2'
x.tofile(fname)

y = np.fromfile(fname, dtype=custom_type)
np.save(fname2, x)
z = np.load(fname2 + '.npy‘)
```

```
[((5, 11), 113.84)]
[((5, 11), 113.84)]
[((5, 11), 113.84)]
```

NumPy: создание массивов

```
fname = 'testfile'
x = np.linspace(0, np.pi, 3)
y = np.sin(x)
print(x)
print(y)
```

```
[ 0.         1.57079633  3.14159265]
[  0.00000000e+00   1.00000000e+00   1.22464680e-16]
[ 0.         1.57079633  3.14159265]
[  0.00000000e+00   1.00000000e+00   1.22464680e-16]
```

```
np.savez(fname, x=x, y=y)
res = np.load(fname + '.npz')

print(res['x'])
print(res['y'])
```

Формат .npy

- Стандартный модуль pickle:
  - Дублирование данных в памяти
  - Данные массива недоступны напрямую
- tofile & fromfile:
  - Нет информации о размере и типе данных
  - Нет возможности записывать массивы объектов
- NPY use-case:
  - Небольшие задачи
  - Хранение промежуточной информации для внутренней обработки
  - Многопоточная запись в общий файл-массив

Базовые операции

- Арифметические операции действуют **поэлементно** и возвращают **новый** массив
- Оператор * действует поэлементно, для скалярного произведения **numpy.dot**
- *= , += и т.п. **меняют** массив
- При работе с массивами разных типов все приводится к более общему\«точному» типу
- Унарные операции (max, min, sum,…) – методы **ndarray**
  - **Как операции относительно 1D массива**
  - **Как операции по выбранному измерению**
- «Универсальные функции»:
  - transpose, vdot, cross, trace, argmin, conj, floor, inv,…

Базовые операции

```
a = np.linspace(0, np.pi, 6)
b = np.arange( 6 )
c = a-b
c *= 2
d = b**3
e = 2*np.cos(a)
f = e > 0
```

```
[ 0.          0.62831853  1.25663706  1.88495559
  2.51327412  3.14159265]
[0 1 2 3 4 5]
[ 0.         -0.37168147 -0.74336294 -1.11504441 -
  1.48672588 -1.85840735]
[ 0.         -0.74336294 -1.48672588 -2.23008882 -
  2.97345175 -3.71681469]
[  0   1   8  27  64 125]
[ 2.          1.61803399  0.61803399 -0.61803399 -
  1.61803399 -2.        ]
[ True  True  True False False False]
```

```
A = np.array([[1,1], [1,2]])
B = np.array([[1,0], [0,2]])
```

```
[[1 1]
 [1 2]]
[[1 0]
 [0 4]]
```

```
C = A * B
D = np.dot(A, B)
```

```
[[1 2]
 [1 4]]
```

```
A.max()
A.max(axis = 1)
```

```
2       [1 2]
```

Копирование: три «типа»

- Присваивание не копирует массивы
- Новый объект, указывающий на те же данные (**view \ поверхностная копия**)
- Глубокое копирование (**copy**)

```
a = np.arange(6)
b = a
print(b is a)
b.shape = 3,2
print(a.shape)
c = a.view()
print(c is a)
print(c.base is a)
c.shape = 1,6
c[0,0] = 999
d = a.copy()
d[0,0] = -1
```

```
True
(3, 2)
[[0 1]
 [2 3]
 [4 5]]
False
True
[[999   1   2   3   4   5]]
[[999   1]
 [  2   3]
 [  4   5]]
[[999   1]
 [  2   3]
 [  4   5]]
[[-1  1]
 [ 2  3]
 [ 4  5]]
```

Индексация, срезы и итерирование

- 1D допускают те же операции, что и списки, и другие последовательности
- Индексация многомерных массивов: через кортежи
- Итерация многомерных массивов: по 1-му измерению
- **.flat** для итерации по всем элементам
- Изменение формы через **.shape**
- Соединение в один массив с помощью **column_stack**, **vstack**, **hstack**
- Разделение с помощью **hsplit**, **vsplit**, **array_split**

Индексация, срезы и итерирование

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

for i in a:
    print(i**2)
```

```
[0 1 2 3 4 5]
[0 1 0 3 0 5]
[5 0 3 0 1 0]
0
1
0
9
0
25
```

Индексация, срезы и итерирование

```
b = np.fromfunction(lambda i, j: i**2 + j**2, (5,4), dtype=int)
print(b[0:5, 1])
print(b[ : ,1])
print(b[1:3, : ])
print(b[-1])
print(b[-1,:])
print(b[-1,...])

for row in b:
    print(row)

for element in b.flat:
    print(element)

b.shape = (2,10)
```

```
[ 1  2  5 10 17]
[ 1  2  5 10 17]
[[ 1  2  5 10]
 [ 4  5  8 13]]
[16 17 20 25]
[16 17 20 25]
[16 17 20 25]
[0 1 4 9]
[ 1  2  5 10]
[ 4  5  8 13]
[ 9 10 13 18]
[16 17 20 25]
0
1
4
9
1
2
5
10
13
18
16
17
20
25
```

Индексация, срезы и итерирование

```
a = np.floor(10*np.random.random((3,2)))
b = np.floor(10*np.random.random((3,2)))
c = np.vstack((a,b))
d = np.hstack((a,b))
e = np.column_stack((a,b))
print(a)
print(b)
print(c)
print(d)
print(e)
```

```
[array([[ 5.,  4.,  7.,  9.],
        [ 9.,  4.,  4.,  6.]]), array([[ 3.,  1.,  6.,  8.],
        [ 5.,  5.,  9.,  0.]]), array([[ 3.,  6.,  2.,  4.],
        [ 7.,  0.,  8.,  2.]])]
[array([[ 5.,  4.,  7.],
        [ 9.,  4.,  4.]]), array([[ 9.],
        [ 6.]]), array([[ 3.,  1.,  6.,  8.,  3.,  6.,  2.,  4.],
        [ 5.,  5.,  9.,  0.,  7.,  0.,  8.,  2.]])]
```

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

Индексация, срезы и итерирование

```
a = np.floor(10*np.random.random((2,12)))
print(a)
b = np.hsplit(a,3)
c = np.hsplit(a,(3,4))
print(b)
print(c)
```

```
[[ 4.  9.  6.  2.  8.  5.  9.  9.  6.  5.  7.  9.]
 [ 3.  2.  5.  9.  2.  8.  9.  4.  2.  6.  3.  6.]]
[array([[ 4.,  9.,  6.,  2.],
       [ 3.,  2.,  5.,  9.]]), array([[ 8.,  5.,  9.,  9.],
       [ 2.,  8.,  9.,  4.]]), array([[ 6.,  5.,  7.,  9.],
       [ 2.,  6.,  3.,  6.]])]
[array([[ 4.,  9.,  6.],
       [ 3.,  2.,  5.]]), array([[ 2.],
       [ 9.]]), array([[ 8.,  5.,  9.,  9.,  6.,  5.,  7.,  9.],
       [ 2.,  8.,  9.,  4.,  2.,  6.,  3.,  6.]])]
```

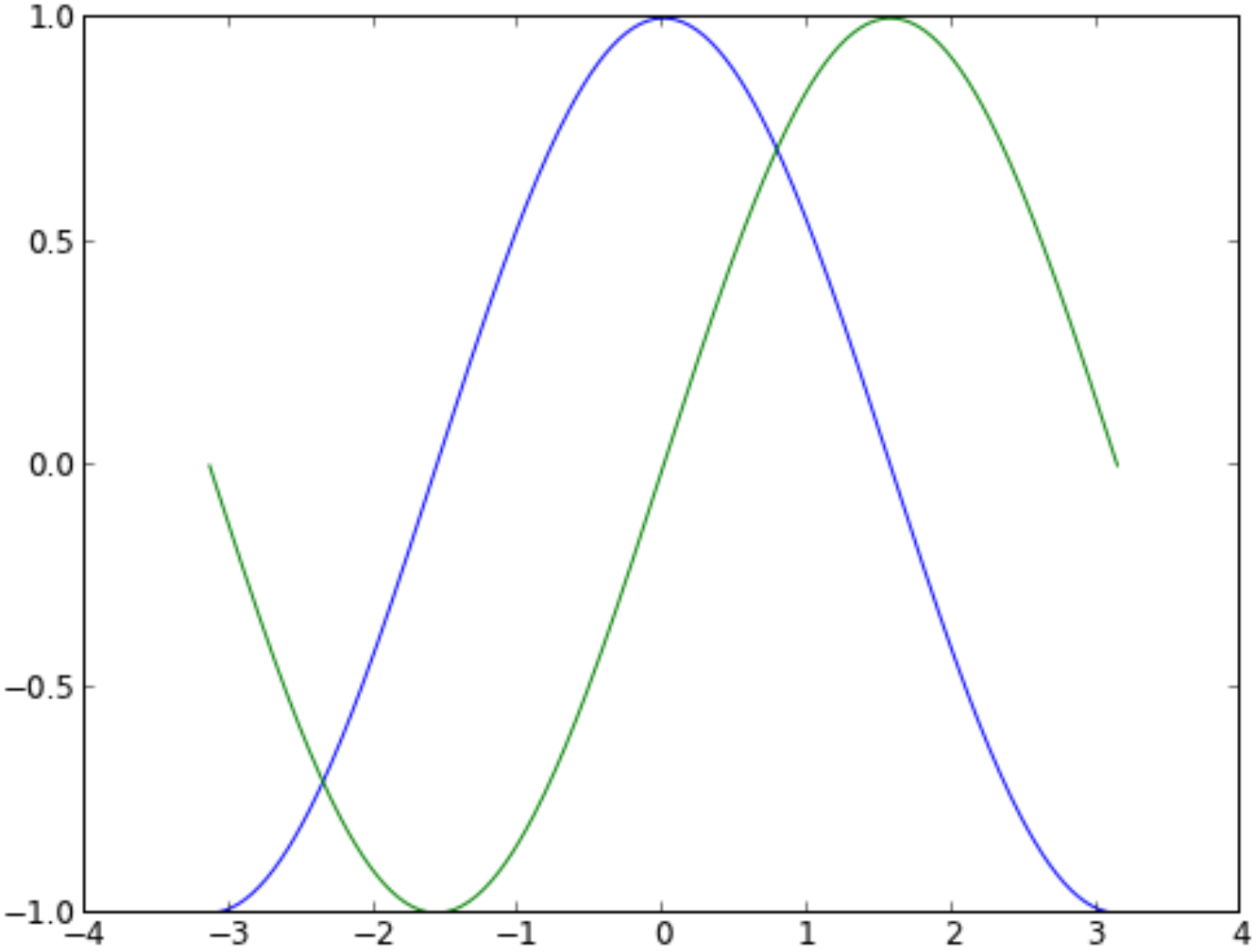
NumPy для пользователей MATLAB

Python	MATLAB
Индексация с 0	Индексация с 1
Передача по ссылке	Передача по значению с «ленивым» копированием
Стандартные операции над 2D массивами поэлементны. Есть отдельный тип matrix.	Стандартные операции над 2D массивами трактуются как матричные

Matplotlib

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

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C,S = np.cos(X), np.sin(X)

plt.plot(X,C)
plt.plot(X,S)

plt.show()
```

Matplotlib



Matplotlib

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6), dpi=80)
plt.subplot(111)
X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
C,S = np.cos(X), np.sin(X)
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")
plt.plot(X, S, color="green", linewidth=1.0, linestyle="-")
plt.xlim(-4.0,4.0)
plt.xticks(np.linspace(-4,4,9,endpoint=True))
plt.ylim(-1.0,1.0)
plt.yticks(np.linspace(-1,1,5,endpoint=True))
plt.show()
```

Matplotlib. SubPlot.

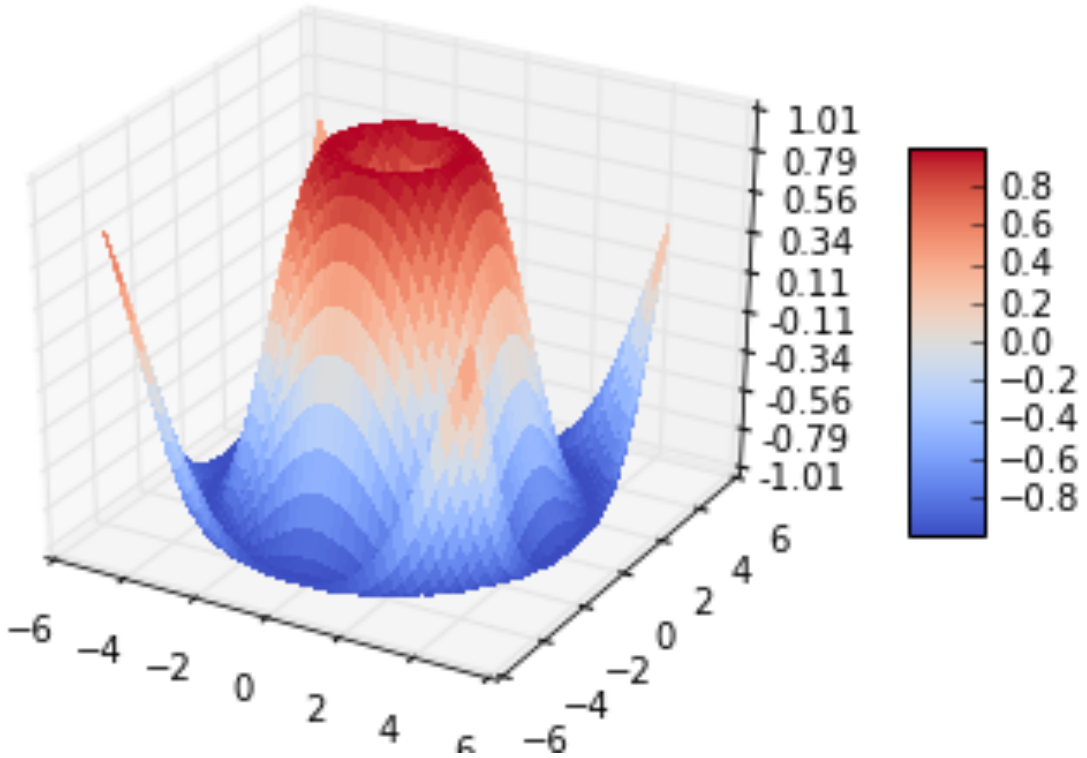
```
from pylab import *
subplot(2,2,1)
xticks([]), yticks([])
text(0.5,0.5, 'subplot(2,2,1)',ha='center',va='center',size=20,alpha=.5)
subplot(2,2,2)
xticks([]), yticks([])
text(0.5,0.5, 'subplot(2,2,2)',ha='center',va='center',size=20,alpha=.5)
subplot(2,2,3)
xticks([]), yticks([])
text(0.5,0.5, 'subplot(2,2,3)',ha='center',va='center',size=20,alpha=.5)
subplot(2,2,4)
xticks([]), yticks([])
text(0.5,0.5, 'subplot(2,2,4)',ha='center',va='center',size=20,alpha=.5)
show()
```

Matplotlib. SubPlot.

Matplotlib. Surface.

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.gca(projection='3d')
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```

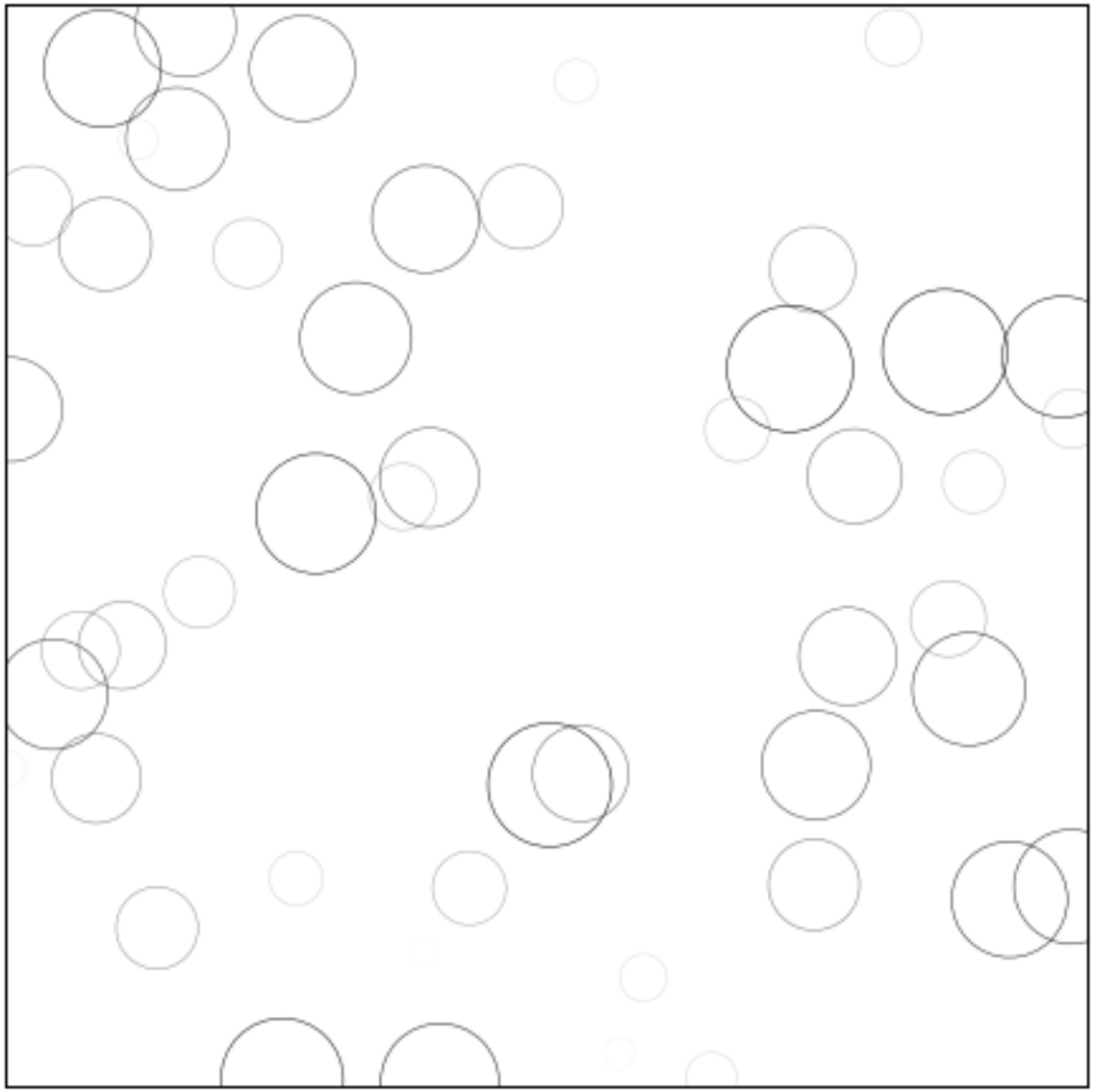
Matplotlib. Surface.



Matplotlib. Scatter.

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(6,6), facecolor='white')
ax = fig.add_axes([0.005,0.005,.99,.99], frameon=True, aspect=1)
n = 50
size_min = 50
size_max = 50*50
P = np.random.uniform(0,1,(n,2))
C = np.ones((n,4)) * (0,0,0,1)
C[:,3] = np.linspace(0,1,n)
S = np.linspace(size_min, size_max, n)
scat = ax.scatter(P[:,0], P[:,1], s=S, lw = 0.5,
                  edgecolors = C, facecolors='None')
ax.set_xlim(0,1), ax.set_xticks([])
ax.set_ylim(0,1), ax.set_yticks([])

plt.show()
```

Matplotlib. Scatter.



Matplotlib. Scatter. Animation

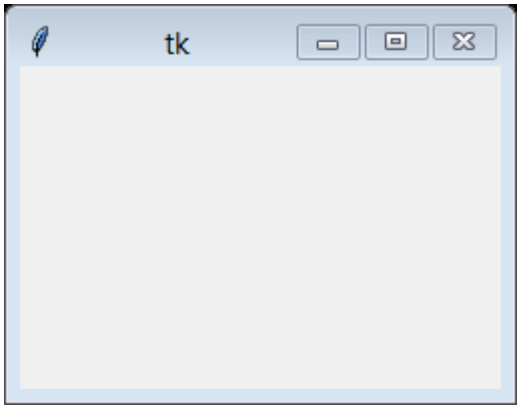
```
def update(frame):
    global P, C, S
    C[:,3] = np.maximum(0, C[:,3] - 1.0/n)
     S += (size_max - size_min) / n
     i = frame % 50
    P[i] = np.random.uniform(0,1,2)
    S[i] = size_min
    C[i,3] = 1

    scat.set_edgecolors(C)
    scat.set_sizes(S)
    scat.set_offsets(P)
    return scat,

animation = FuncAnimation(fig, update, interval=10)
plt.show()
```

Python GUI

- PyQt
- Tkinter
- wxPython
- pyGTK
- pyFLTK

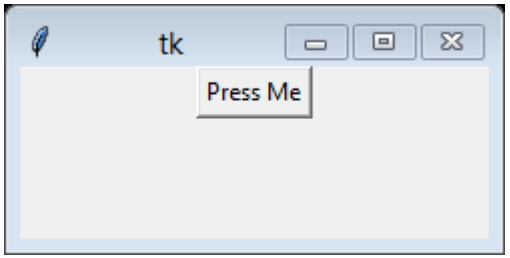
Tkinter

```
import tkinter

root  = tkinter.Tk()

root.mainloop()
```



Tkinter

```
from tkinter import Button

def act():

        print ("I-M-pressed")

foo = Button(None,text="Press Me",command=act)

foo.pack()

foo.mainloop()
```



Tkinter

```
from tkinter import *
root = Tk(className ="My first GUI")
svalue = StringVar()
w = Entry(root,textvariable=svalue)
w.pack()
def act():
    print ("you entered")
    print ('%s' % svalue.get())
foo = Button(root,text="Press Me", command=act)
foo.pack()
root.mainloop()
```