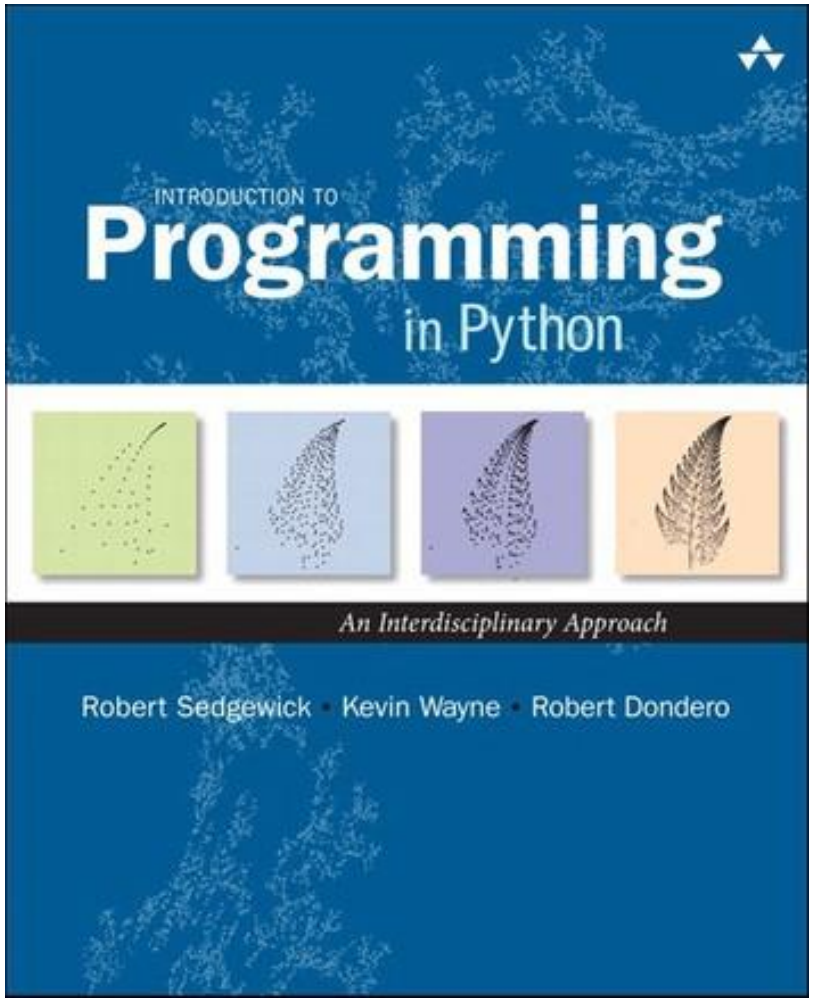
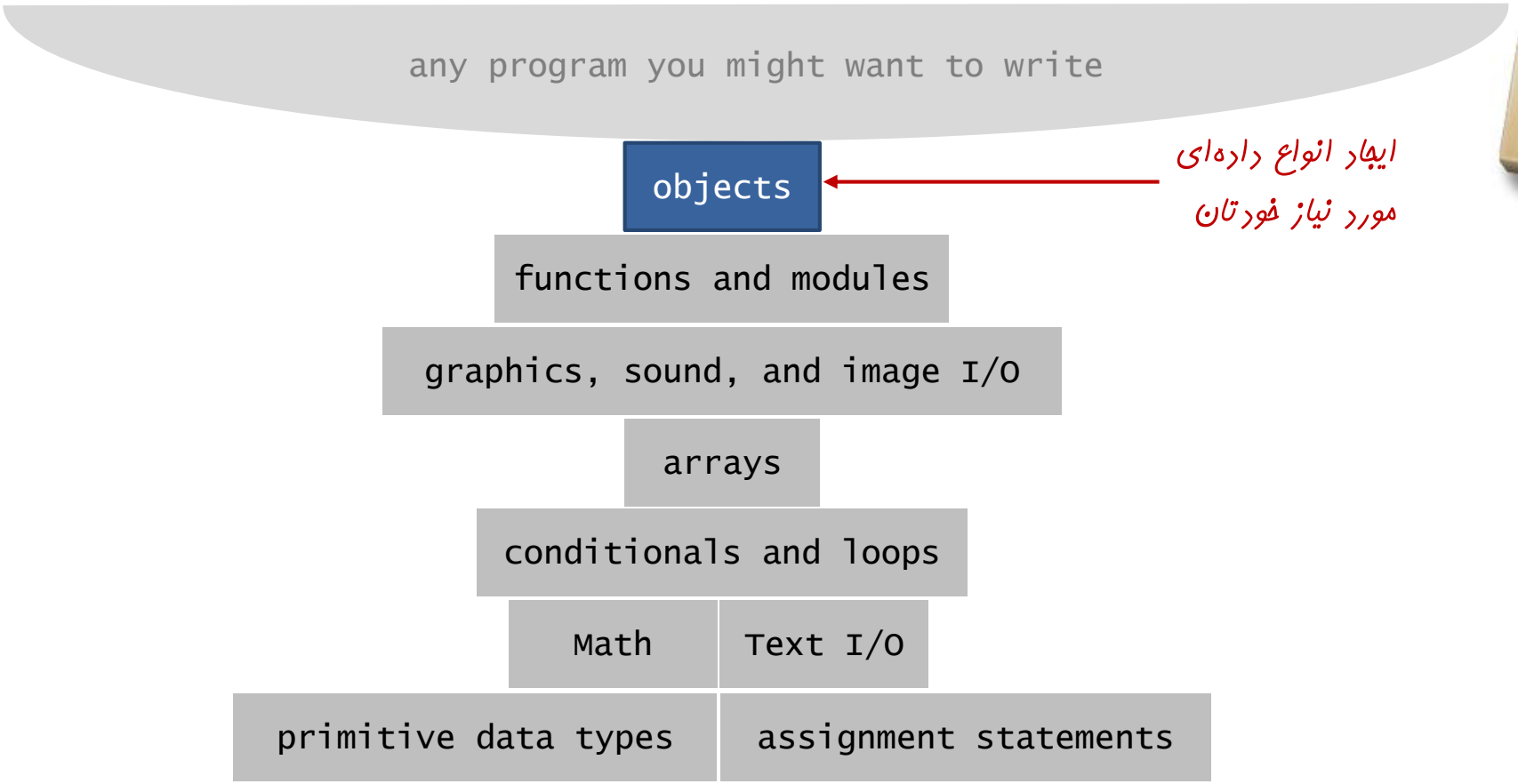
برنامه‌نویسی شی‌گرا:
ایجاد انواع داده‌ای جدید

سید ناصر رضوی **www.snrazavi.ir**
۱۳۹۸

۳-۲ ایجاد انواع داده‌ای



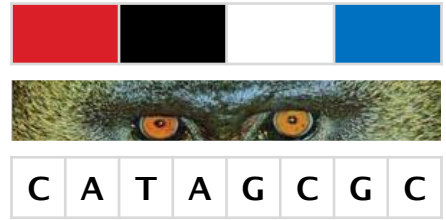
اجزای برنامه‌نویسی



برنامه‌نویسی شی‌گرا

- برنامه‌نویسی شی‌گرا.
  - انواع داده‌ای خودتان را ایجاد کنید.
  - از آنها در برنامه‌های خود استفاده کنید (کار با اشیا) ⟵ *یک شی دربردارنده مقدار یک نوع داده‌ای است*

Data Type	*Values*	*Operations*
Color	three 8-bit integers	get red component, brighten
Picture	2D array of colors	get/set color of pixel
String	sequence of characters	length, substring, compare



- اهمیت. می‌توانیم از اشیا استفاده کنیم بدون آن که از جزییات پیاده‌سازی آنها آگاه باشیم.
  - بخش قبلی: نوشتن برنامه به منظور استفاده از انواع داده‌ای موجود.
  - این بخش: نوشتن برنامه به منظور ایجاد انواع داده‌ای جدید مورد نیاز.

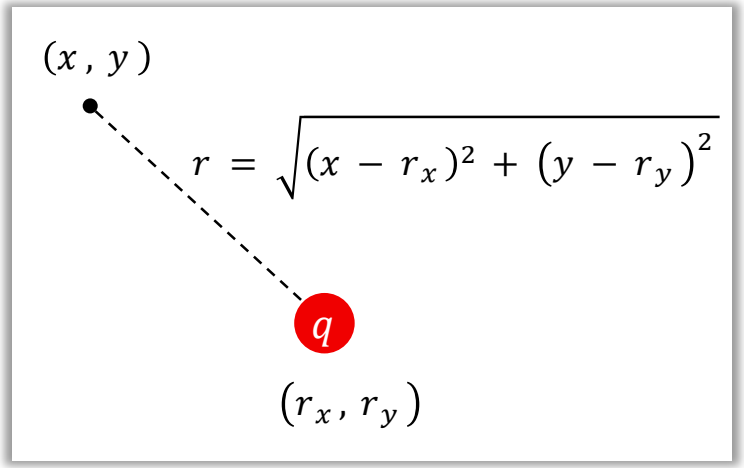
تعریف انواع داده‌ای در پایتون

- برای ایجاد یک نوع داده‌ای باید موارد زیر را مشخص کنیم:
  - مجموعه مقادیر.
  - عملیات قابل انجام بر روی این مجموعه از مقادیر.
  - نحوه ایجاد و مقداردهی اولیه

- کلاس پایتون. تعریف کننده یک نوع داده‌ای با مشخص کردن:
  - متغیرهای نمونه. (مجموعه مقادیر)
  - متدها. (عملیات قابل انجام بر روی این مجموعه از مقادیر)
  - سازنده‌ها. (نحوه ایجاد و مقداردهی اولیه)

مثال: نوع داده‌ای ذرات باردار

- هدف. ایجاد یک نوع داده‌ای به منظور کار کردن با ذرات باردار.
- مجموعه مقادیر. سه مقدار حقیقی. [مکان ذره و مقدار بار الکتریکی]
- عملیات.
  - ایجاد یک ذره باردار جدید در مکان (r_x,r_y) با بار الکتریکی q.
  - تعیین پتانسیل الکتریکی V ناشی از این ذره در مکان (x,y).
  - تبدیل به رشته.

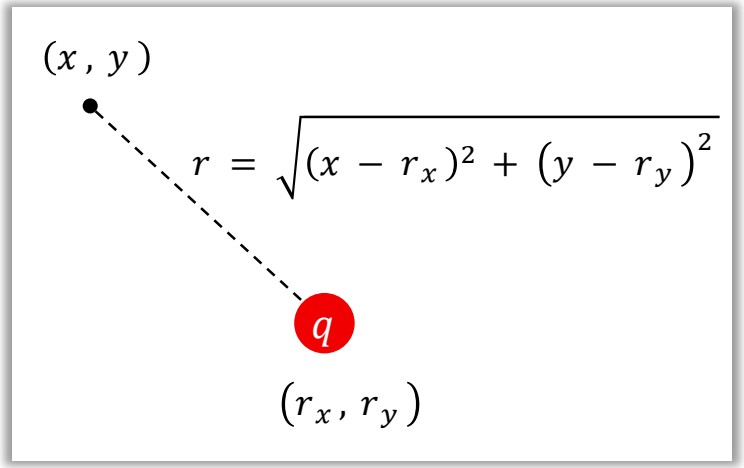


$$V = k\frac{q}{r}$$

$k = 8.99 \times 10^9 \ \mathrm{N.m^2/C^2}$

مثال: نوع داده‌ای ذرات باردار

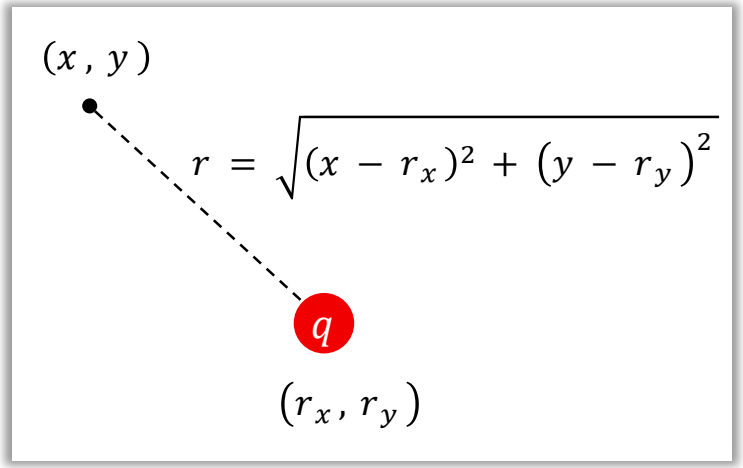
- هدف. ایجاد یک نوع داده‌ای به منظور کار کردن با ذرات باردار.
- مجموعه مقادیر. سه مقدار حقیقی. [مکان ذره و مقدار بار الکتریکی]
- API.



operation	*description*
`Charge(x0, y0, q0)`	*ایجاد یک ذره باردار جدید در مکان* `(x0, y0)` *با بار* `q0`
`c.potentialAt(x, y)`	*پتانسیل الکتریکی ذره باردار* `c` *در نقطه* `(x, y)`
`str(c)`	*نمایش رشته‌ای از ذره باردار* `c` *به صورت* `'q at (rx, ry)'`

مثال: نوع داده‌ای ذرات باردار (پیاده‌سازی)


```
class Charge:
    def __init__(self, x0, y0, q0):
        self._rx = x0
        self._ry = y0
        self._q  = q0

    def potentialAt(self, x, y):
        COULOMB = 8.99e09
        dx = x - self._rx
        dy = y - self._ry
        r = math.sqrt(dx * dx + dy * dy)
        if r == 0.0:  # avoid division by zero
            if self._q >= 0.0: return float('inf')
            else:              return float('-inf')
        return COULOMB * self._q / r

    def __str__(self):
        return '%s at (%s, %s)' %(self._q, self._rx, self._ry)
```



$$V = k\frac{q}{r}$$

$k = 8.99 \times 10^9 \ \mathrm{N.m^2/C^2}$

نوع داده‌ای ذرات باردار: یک برنامه مشتری

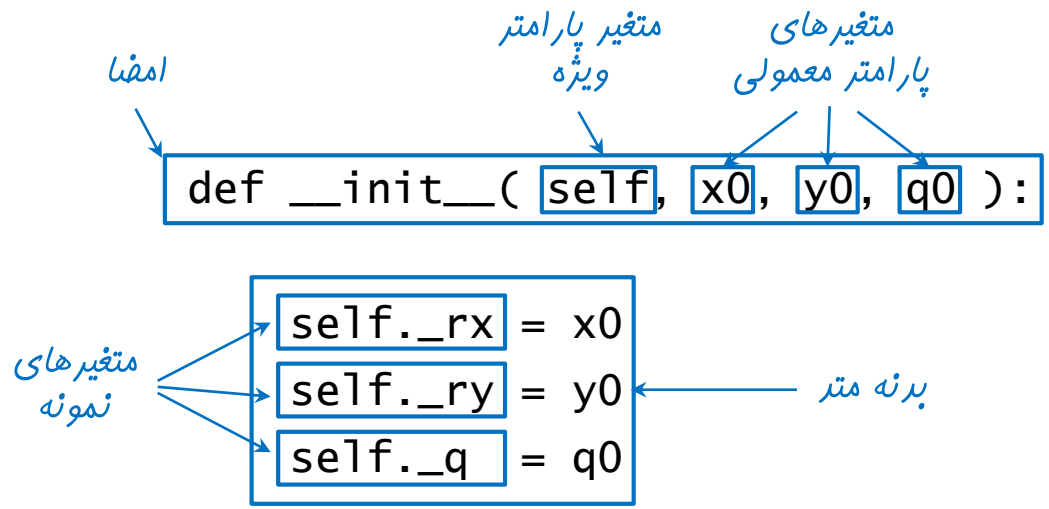
- برنامه مشتری.
  - از عملیات تعریف شده بر روی نوع داده‌ای به منظور محاسبه کمیت مورد نظر استفاده می‌کند.

```
def main():
    x = float(sys.argv[1])
    y = float(sys.argv[2])
    c = Charge(.51, .63, 21.3)   ←—— __init__() فراخوانی متد
    print(c)                     ←—— __str__() فراخوانی متد
    print(c.potentialAt(x, y))

if __name__ == '__main__':
    main()
```

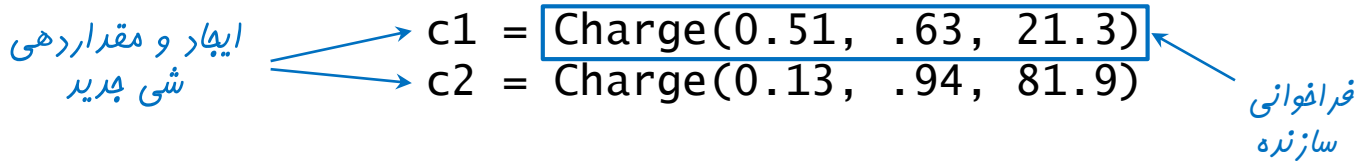
```
% python charge.py .5 .5
21.3 at (0.51, 0.63)
1468638248194.164
```

آناتومی سازنده‌ها

- سازنده. ایجاد و آماده‌سازی اشیای جدید.

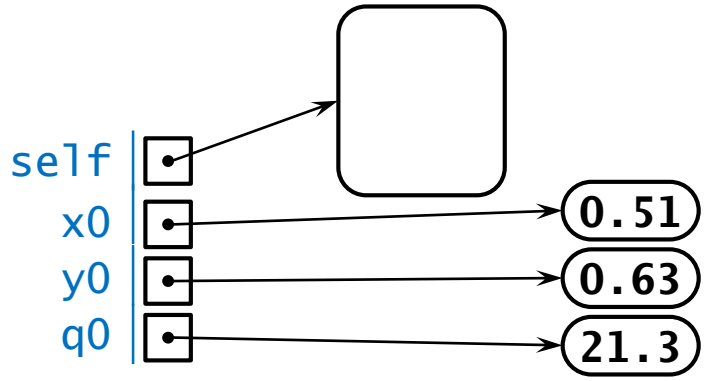


- فراخوانی یک سازنده. برای ایجاد یک شی جدید.



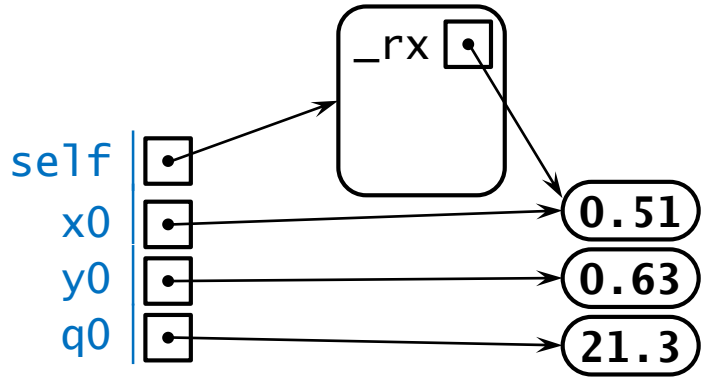
سازنده‌ها: ایجاد و مقداردهی یک شی جدید


```
c1 = Charge(0.51, 0.63, 21.3)
[c1.__init__(self, 0.51, 0.63, 21.3)]
```



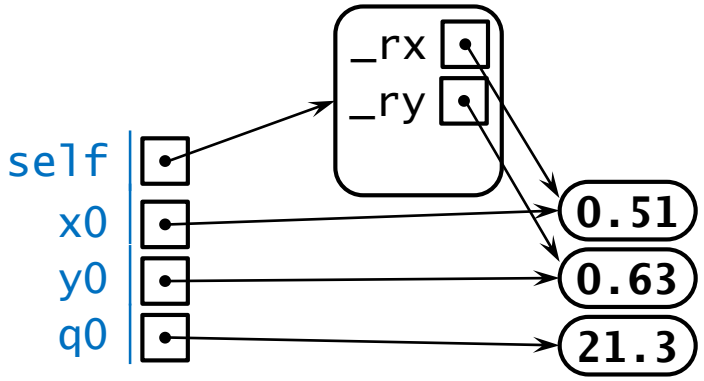
سازنده‌ها: ایجاد و مقداردهی یک شی جدید


```
self._rx = x0
```



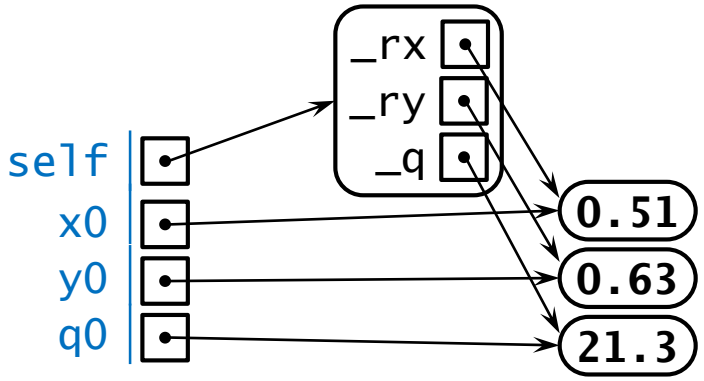
سازنده‌ها: ایجاد و مقداردهی یک شی جدید


```
self._ry = y0
```



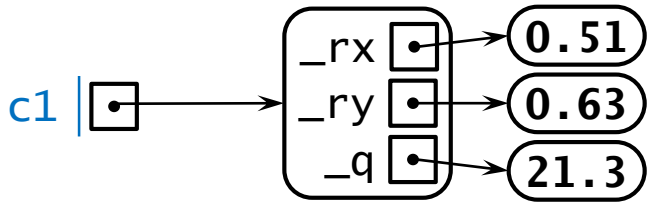
سازنده‌ها: ایجاد و مقداردهی یک شی جدید


```
self._q = q0
```



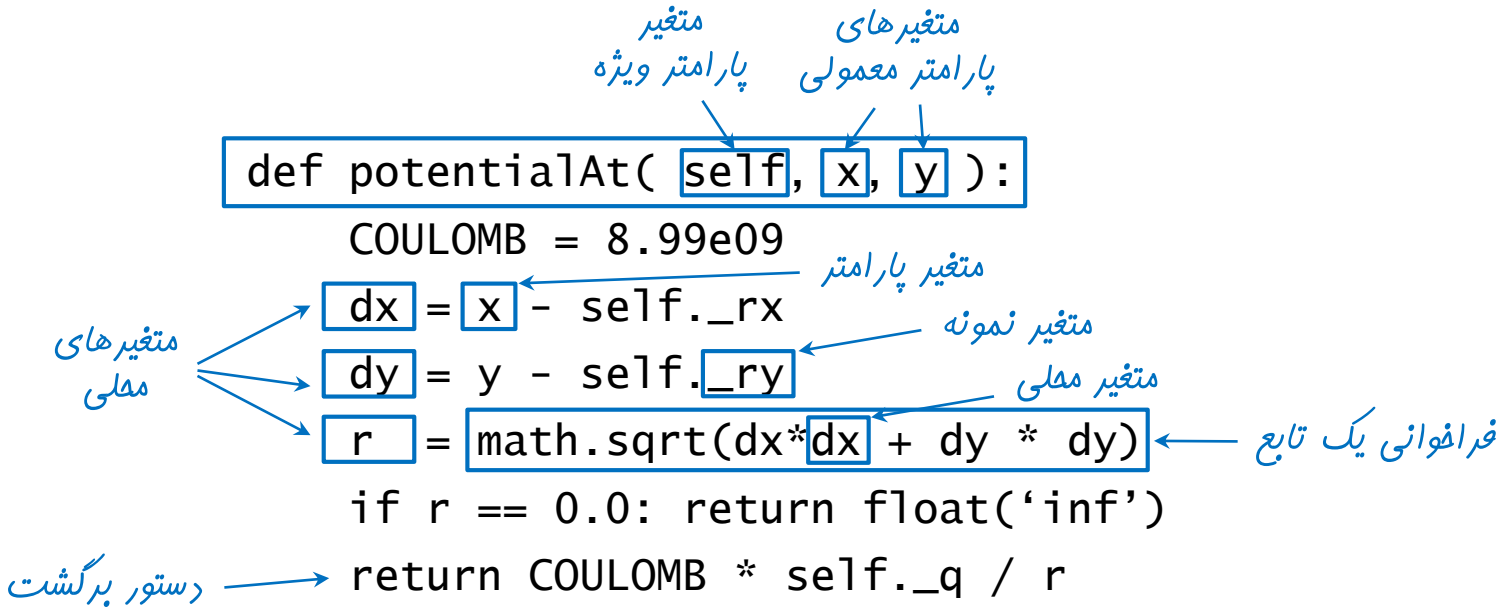
سازنده‌ها: ایجاد و مقداردهی یک شی جدید


```
[return self]
c1 = Charge(0.51, 0.63, 21.3)
```

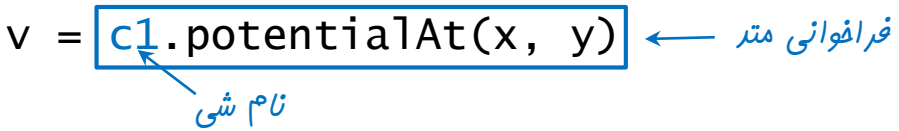


آناتومی متدهای نمونه

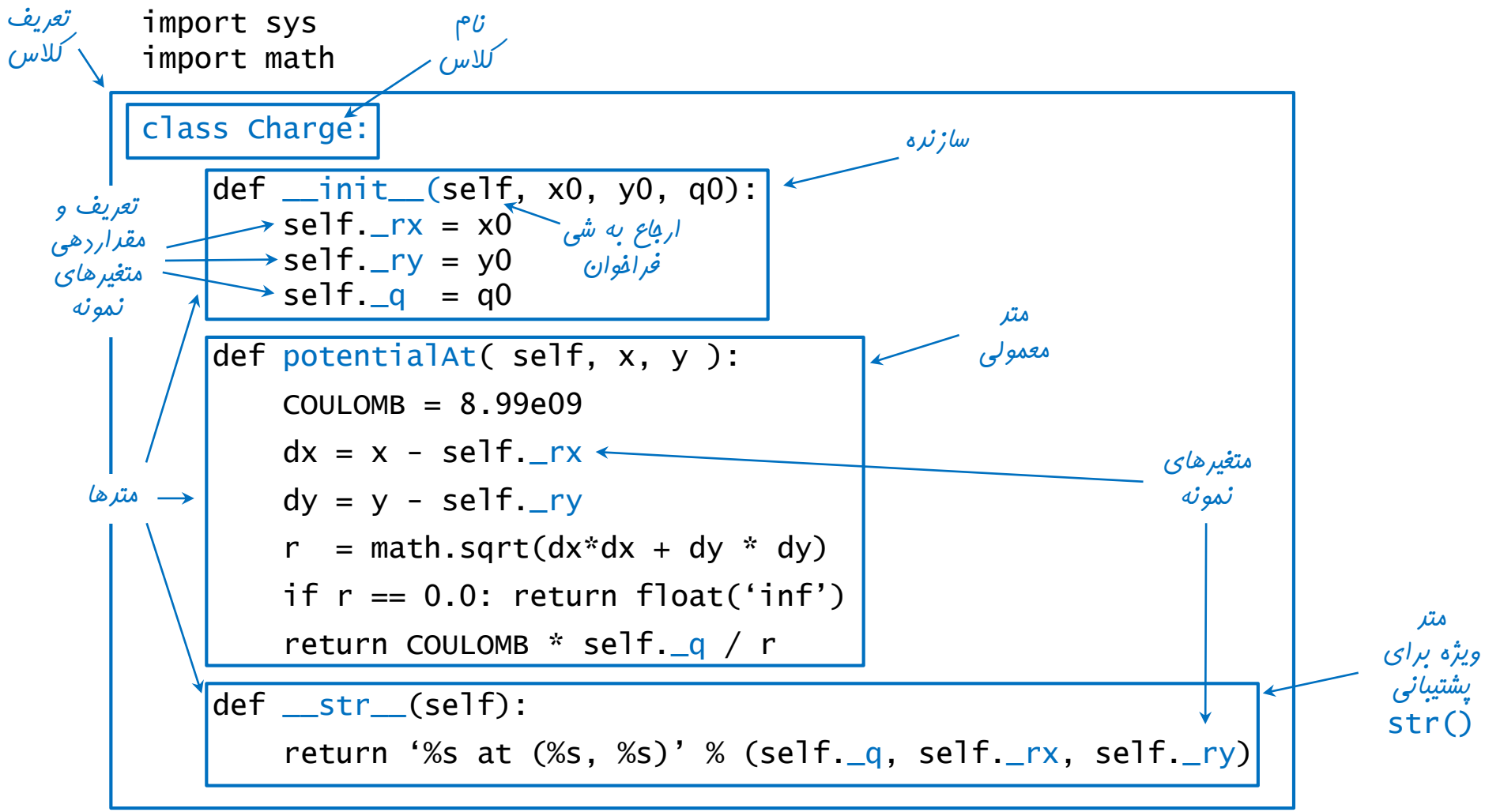
- متد. تعریف کننده عملیات قابل انجام بر روی متغیرهای نمونه.



- فراخوانی یک متد. از عملگر نقطه برای فراخوانی یک متد استفاده کنید.



آناتومی یک کلاس

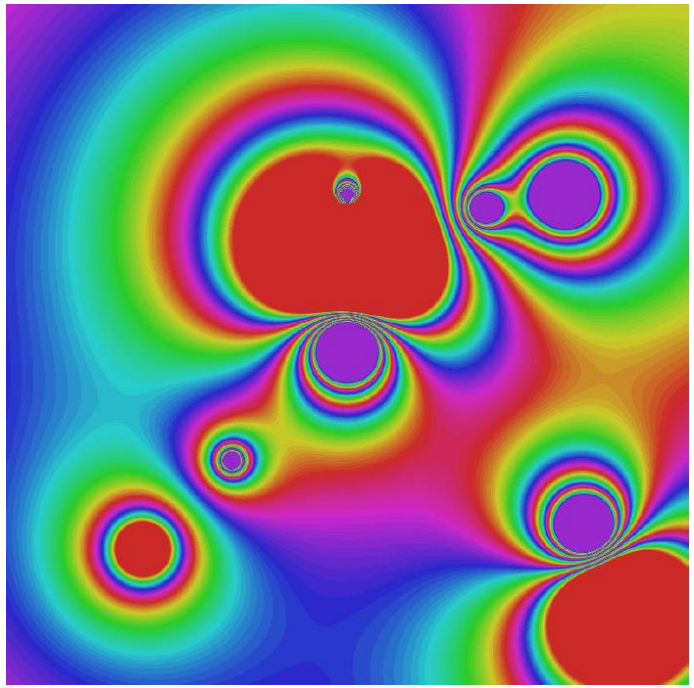


به تصویر کشیدن پتانسیل

- به تصویر کشیدن پتانسیل. از ورودی استاندارد مشخصات n بار الکتریکی را بخوانید؛ سپس پتانسیل کل هر نقطه را در یک مربع واحد محاسبه کنید.

```
% more charges9.txt
9
.51 .63 -100
.50 .50 40
.50 .72 10
.33 .33 5
.20 .20 -10
.70 .70 10
.82 .72 20
.85 .23 30
.90 .12 -50
```

```
% python potential.py < charges9.txt
```



به تصویر کشیدن پتانسیل

- متد کمکی. خواندن ذرات از جریان ورودی (فایل) و ذخیره آنها در یک آرایه.

```
def readCharges():
    n = stdio.readInt()
    a = [None] * n
    for i in range(n):
        x0 = stdio.readFloat()
        y0 = stdio.readFloat()
        q0 = stdio.readFloat()
        a[i] = Charge(x0, y0, q0)
    return a
```

```
% more charges9.txt
9
.51 .63 -100
.50 .50 40
.50 .72 10
.33 .33 5
.20 .20 -10
.70 .70 10
.82 .72 20
.85 .23 30
.90 .12 -50
```

به تصویر کشیدن پتانسیل

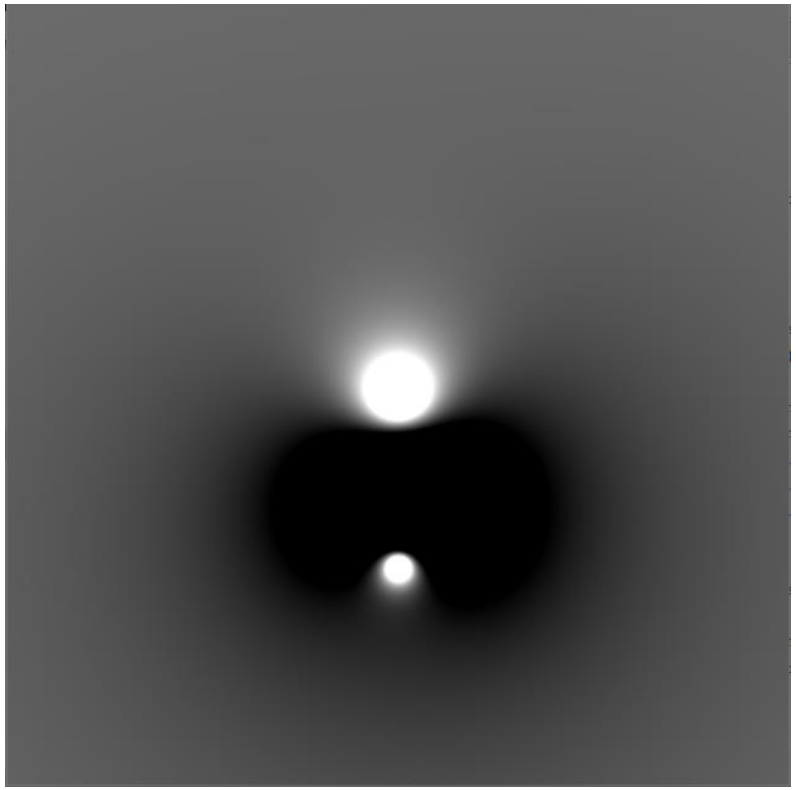
- متد کمکی. تبدیل پتانسیل الکتریکی به یک رنگ خاکستری.

```
def toColor(v):
    v = 128 + v / 2.0e10
    if   v < 0:   t = 0
    elif v > 255: t = 255
    else:         t = int(v)
    return Color(t, t, t)
```

به تصویر کشیدن پتانسیل


```
def main():
    SIZE = 512
    c = readCharges()
    pic = Picture(SIZE, SIZE)
    for col in range(SIZE):
        for row in range(SIZE):
            x = 1.0 * col / SIZE
            y = 1.0 * row / SIZE
            v = 0.0
            for c in a:
                v += c.potentialAt(x, y)
            pic.set(col, SIZE-1-row, toColor(v))
    stddraw.setCanvasSize(SIZE, SIZE)
    stddraw.picture(pic)
    stddraw.show()
```

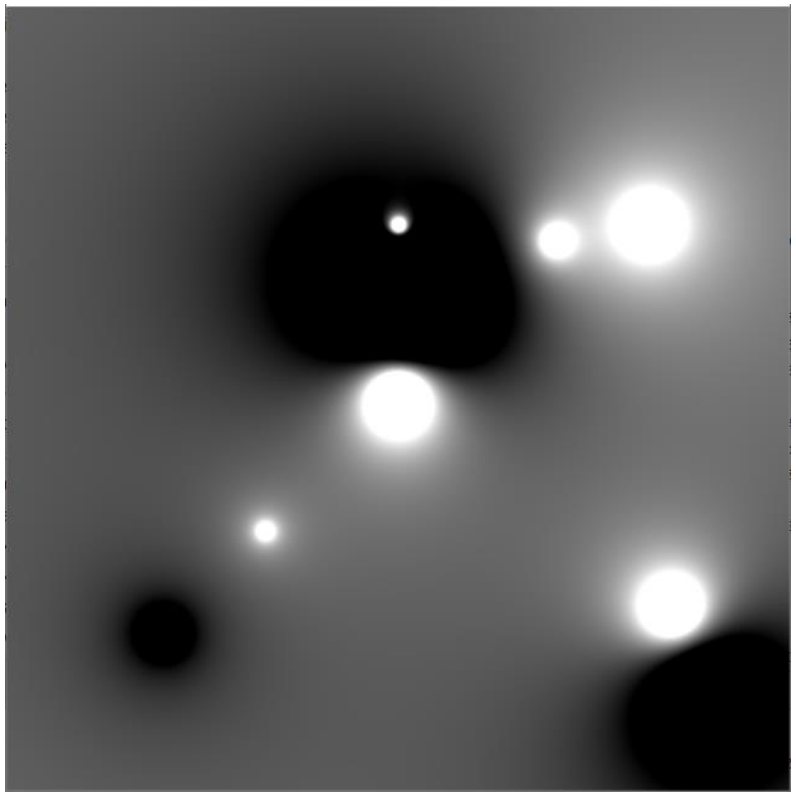
```
% python potential.py < charges3.txt
```



به تصویر کشیدن پتانسیل

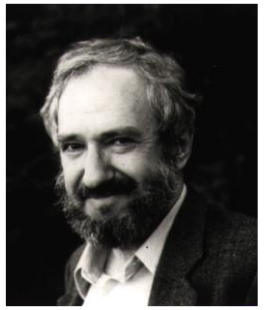

```
def main():
    SIZE = 512
    c = readCharges()
    pic = Picture(SIZE, SIZE)
    for col in range(SIZE):
        for row in range(SIZE):
            x = 1.0 * col / SIZE
            y = 1.0 * row / SIZE
            v = 0.0
            for c in a:
                v += c.potentialAt(x, y)
            pic.set(col, SIZE-1-row, toColor(v))
    stddraw.setCanvasSize(SIZE, SIZE)
    stddraw.picture(pic)
    stddraw.show()
```

```
% python potential.py < charges9.txt
```



گرافیک لاک‌پشتی

- ایده. یک لاک‌پشت یک مدل ایده‌آل از یک دستگاه ترسیم است.



سیمور پاپرت
۱۹۲۸ – ۲۰۱۶

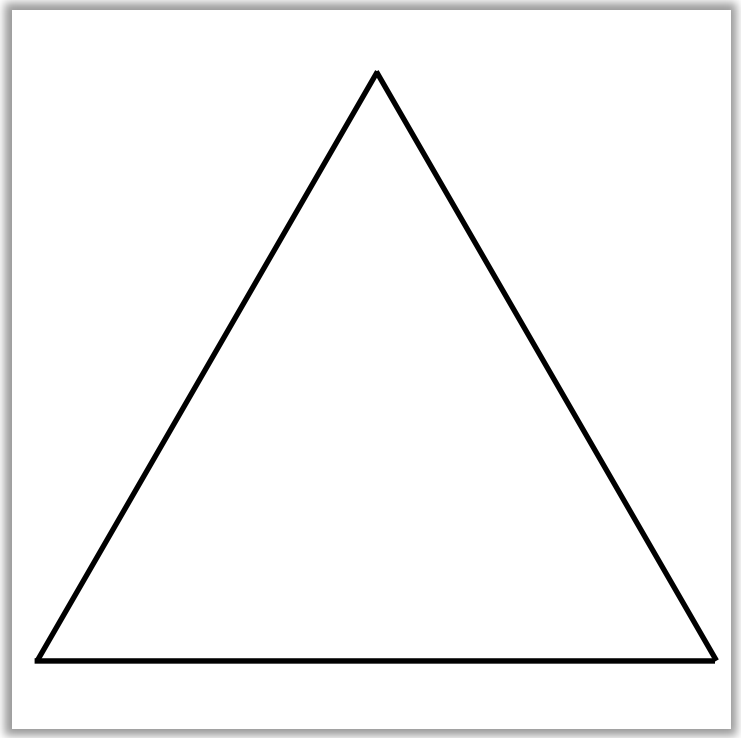
position (*x*, *y*)	(.5, .5)	(.25, .75)	(.22, .12)
orientation	90°	135°	10°

operation	*description*
`Turtle(x0, y0, a0)`	*ایجاد یک لاک‌پشت جدید در مکان* (x0, y0) *با زاویه* a0 *درجه با محور* x
`t.turnLeft(delta)`	*به اندازه دلتا درجه به چپ گردش کن (در خلاف جهت عقربه‌های ساعت)*
`t.goForward(step)`	*به اندازه* step *به جلو برو و یک خط ترسیم کن*

گرافیک لاک‌پشتی: ترسیم مثلث

- یک توصیه. با پیاده‌سازی یک برنامه آزمایشی ساده شروع کنید.

```
def main():
    turtle = Turtle(0.0, 0.0, 0.0)
    turtle.goForward(1.0)
    turtle.turnLeft(120.0)
    turtle.goForward(1.0)
    turtle.turnLeft(120.0)
    turtle.goForward(1.0)
    turtle.turnLeft(120.0)
    stddraw.show()
```

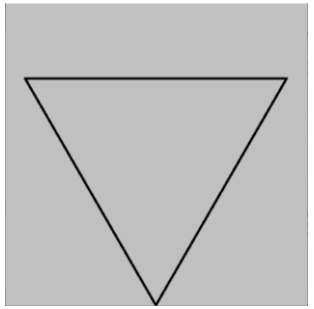


گرافیک لاک‌پشتی: ترسیم چندضلعی

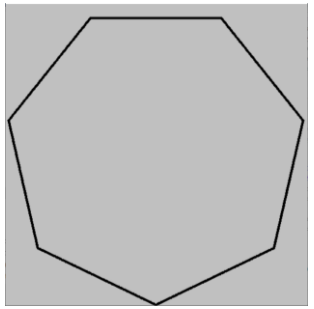
- یک توصیه. با پیاده‌سازی یک برنامه آزمایشی ساده شروع کنید.

```
def main():
    n = int(sys.argv[1])
    turtle = Turtle(0.5, 0.0, 180.0 / n)
    step = math.sin(math.radians(180.0 / n))
    stddraw.clear(stddraw.LIGHT_GRAY)

    for i in range(n):
        turtle.goForward(step)
        turtle.turnLeft(360.0 / n)

    stddraw.show()
```

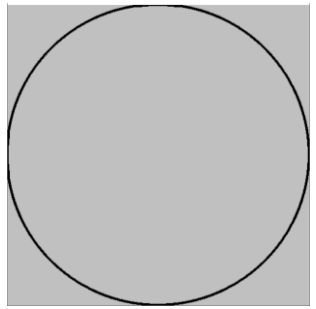
```
% python turtle.py n
```



n = 3



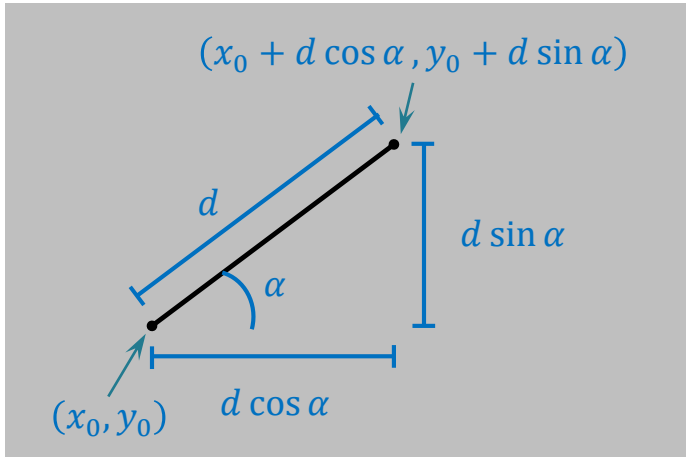
n = 7



n = 1440

پیاده‌سازی گرافیک لاک‌پشتی

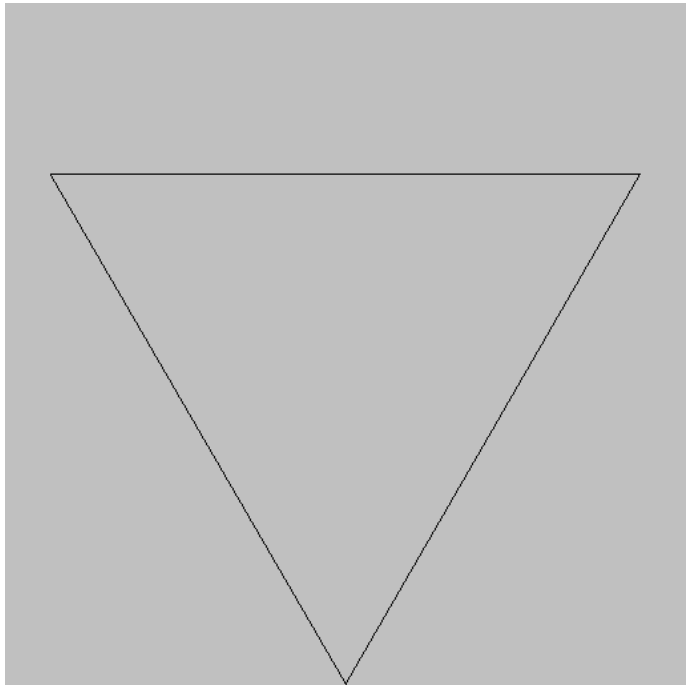

```
class Turtle:
    def __init__(self, x, y, angle):
        self._x = x
        self._y = y
        self._angle = angle

    def turnLeft(self, delta):
        self._angle += delta

    def goForward(self, step):
        old_x = self._x
        old_y = self._y
        self._x += step * math.cos(math.radians(self._angle))
        self._y += step * math.sin(math.radians(self._angle))
        stddraw.line(old_x, old_y, self._x, self._y)
```



گرافیک لاک‌پشتی: ترسیم مارپیچ


```
n = int(sys.argv[1])
wraps = int(sys.argv[2])
decay = float(sys.argv[3])

angle = 360.0 / n
step = math.sin(math.radians(angle / 2.0))
turtle = Turtle(0.5, 0.0, angle / 2.0)

stddraw.setPenRadius(0.0)
stddraw.clear(stddraw.LIGHT_GRAY)

for i in range(wraps * n):
    step /= decay
    turtle.goForward(step)
    turtle.turnLeft(angle)

stddraw.show()
```

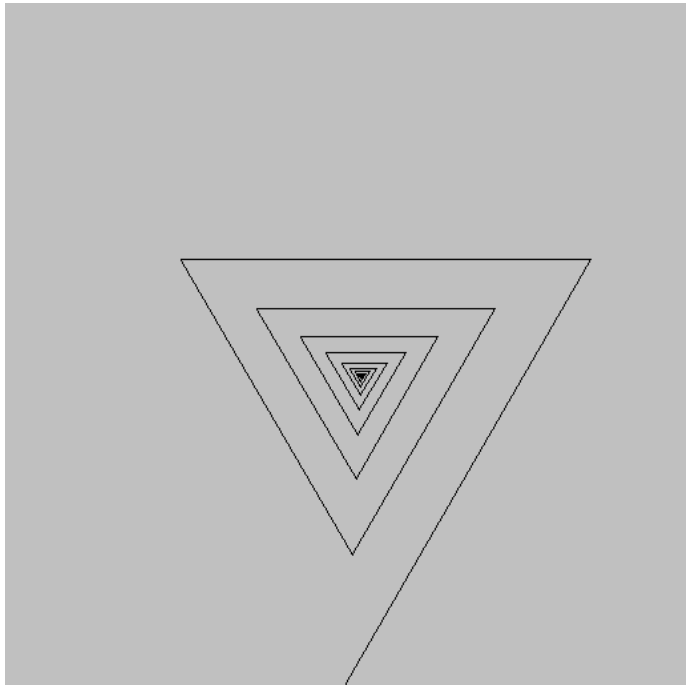
```
% python spiral.py 3 1 1.0
```



گرافیک لاک‌پشتی: ترسیم مارپیچ


```
n = int(sys.argv[1])
wraps = int(sys.argv[2])
decay = float(sys.argv[3])

angle = 360.0 / n
step = math.sin(math.radians(angle/ 2.0))
turtle = Turtle(0.5, 0.0, angle/ 2.0)

stddraw.setPenRadius(0.0)
stddraw.clear(stddraw.LIGHT_GRAY)

for i in range(wraps * n):
    step /= decay
    turtle.goForward(step)
    turtle.turnLeft(angle)

stddraw.show()
```

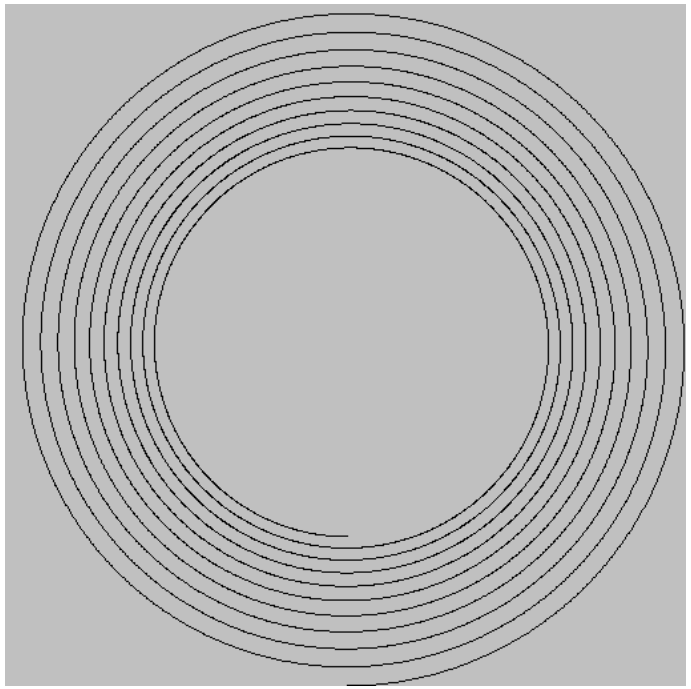
```
% python spiral.py 3 10 1.2
```



گرافیک لاک‌پشتی: ترسیم مارپیچ


```
n = int(sys.argv[1])
wraps = int(sys.argv[2])
decay = float(sys.argv[3])

angle = 360.0 / n
step = math.sin(math.radians(angle/ 2.0))
turtle = Turtle(0.5, 0.0, angle/ 2.0)

stddraw.setPenRadius(0.0)
stddraw.clear(stddraw.LIGHT_GRAY)

for i in range(wraps * n):
    step /= decay
    turtle.goForward(step)
    turtle.turnLeft(angle)

stddraw.show()
```

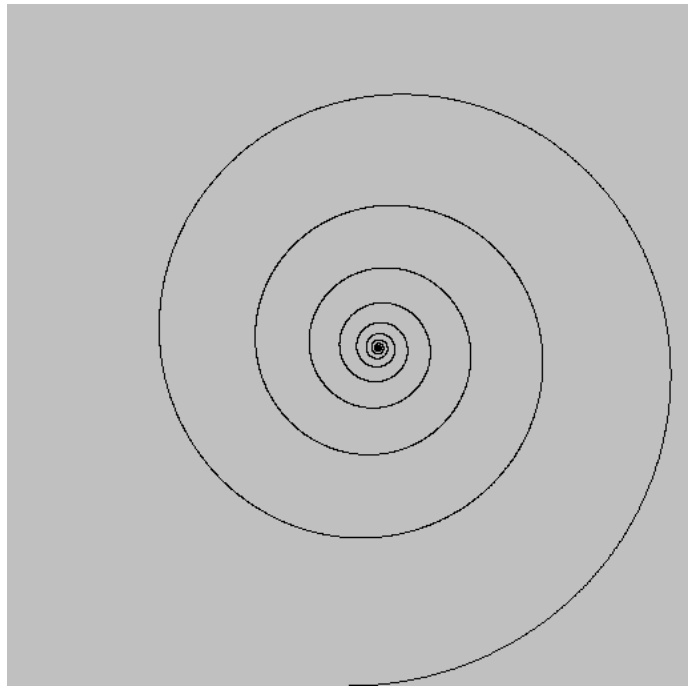
```
% python spiral.py 1440 10 1.00004
```



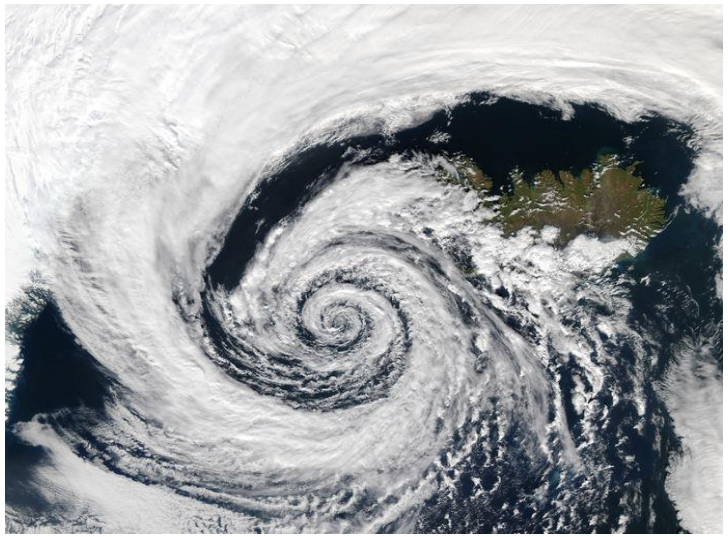
گرافیک لاک‌پشتی: ترسیم مارپیچ


```
n = int(sys.argv[1])
wraps = int(sys.argv[2])
decay = float(sys.argv[3])

angle = 360.0 / n
step = math.sin(math.radians(angle/ 2.0))
turtle = Turtle(0.5, 0.0, angle/ 2.0)

stddraw.setPenRadius(0.0)
stddraw.clear(stddraw.LIGHT_GRAY)

for i in range(wraps * n):
    step /= decay
    turtle.goForward(step)
    turtle.turnLeft(angle)

stddraw.show()
```

```
% python spiral.py 1440 10 1.0004
```



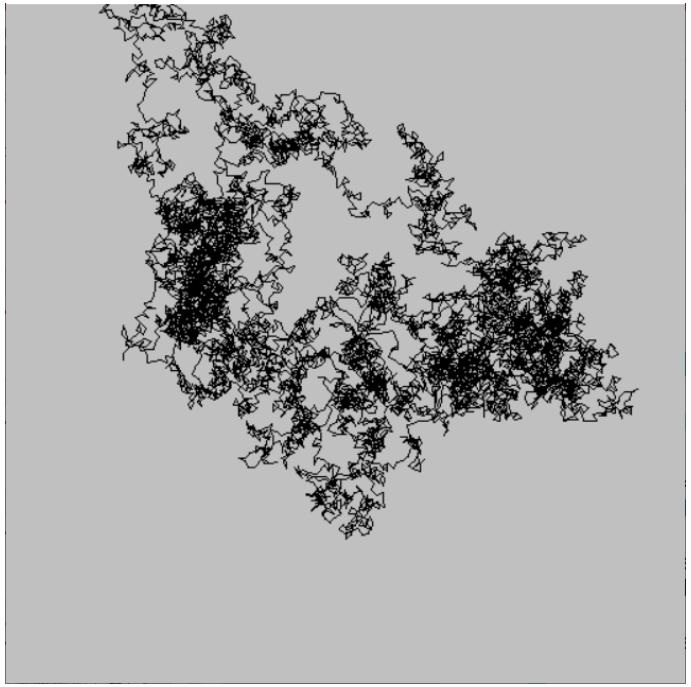
شکل‌های مارپیچ در طبیعت



حرکت تصادفی (براونی)


```
stepCount = int(sys.argv[1])
stepSize = float(sys.argv[2])

stddraw.setPenRadius(0.0)
stddraw.clear(stddraw.LIGHT_GRAY)

t = Turtle(0.5, 0.5, 0.0)


for i in range(stepCount):
    t.turnLeft(stdrandom.uniformFloat(0.0, 360.0))
    t.goForward(stepSize)
    stddraw.show(0.0)

stddraw.show()
```

```
% python drunk.py 10000 0.01
```



اعداد مختلط

- هدف. ایجاد یک نوع داده‌ای به منظور کار کردن با اعداد مختلط.
- مجموعه مقادیر. دو عدد حقیقی: قسمت حقیقی و موهومی.

operation	*special method*	*description*
`Complex(x, y)`	`__init__(self, x, y)`	*ایجاد یک عدد مختلط جدید به شکل* `x + yi`
`a.re()`		*قسمت حقیقی* `a`
`a.im()`		*قسمت موهومی* `a`
`a + b`	`__add__(self, other)`	*مجموع* `a` *و* `b`
`a * b`	`__mul__(self, other)`	*حاصل‌ضرب* `a` *و* `b`
`abs(a)`	`__abs__(self)`	*اندازه* `a`
`str(a)`	`__str__(self)`	*بازنمایی رشته‌ای* `a` *به صورت* `'x + yi'`

اعداد مختلط: آزمایش

- یک توصیه. با پیاده‌سازی یک برنامه آزمایشی ساده شروع کنید.

```
def main():
    a = Complex( 3.0, 4.0)
    b = Complex(-2.0, 3.0)
    print('a = ' % a)
    print('b = ' % b)
    print('a + b = ' % a + b)
    print('a * b = ' % a * b)
```

```
% python complex.py
a = 3.0 + 4.0i
b = -2.0 + 3.0i
a + b = 1.0 + 7.0i
a * b = -18.0 + 1.0i
```

اعداد مختلط: پیاده‌سازی


```
class Complex:
    def __init__(self, re, im):
        self._re = re
        self._im = im

    def re(self):
        return self._re

    def im(self):
        return self._im

    def __add__(self, other):
        re = self._re + other._re
        im = self._im + other._im
        return Complex(re, im)
```

اعداد مختلط: پیاده‌سازی


```
def __mul__(self, other):
    re = self._re * other._re - self._im * other._im
    im = self._re * other._im + self._im * other._re
    return Complex(re, im)
```

```
def __eq__(self, other):
    return (self._re == other._re) and (self._im == other._im)
```

```
def __ne__(self, other):
    return not self.__eq__(other)
```

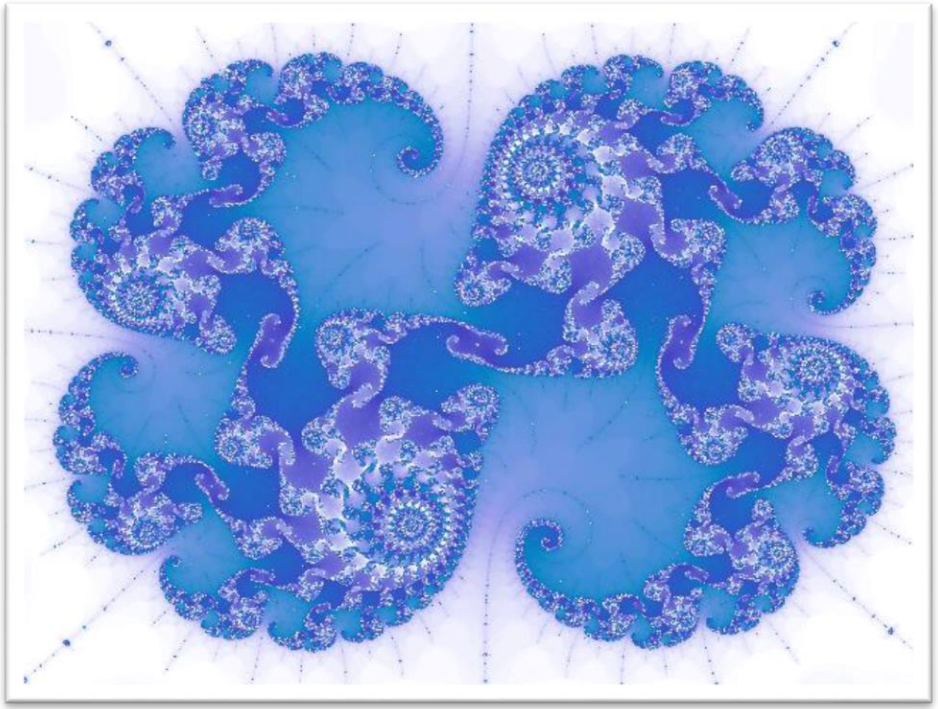
```
def __abs__(self, other):
    return math.sqrt(self._re * self._re + self._im * self._im)
```

```
def __str__(self, other):
    return str(self._re) + ' + ' + str(self._im) + 'i'
```

کاربردهای اعداد مختلط

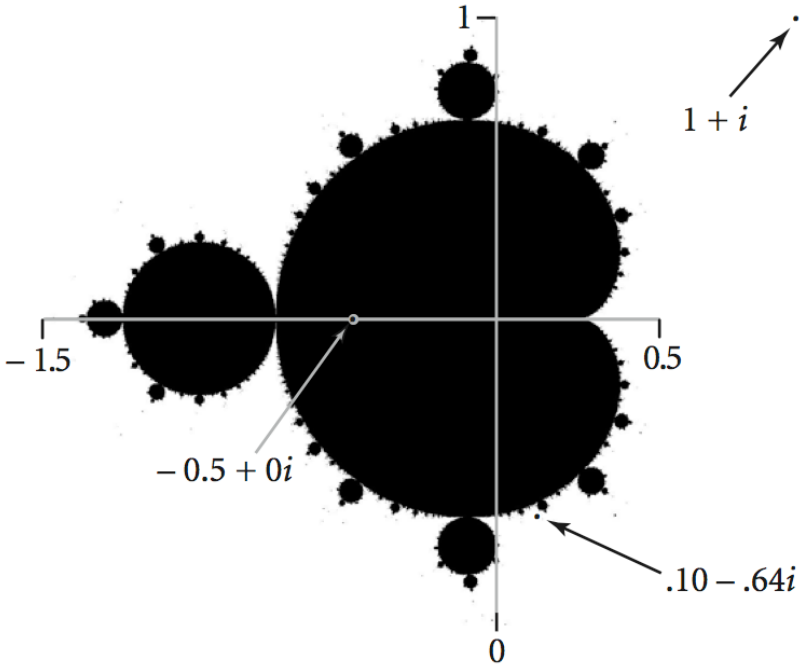
- اهمیت. یک مدل ریاضی فراگیر.

- کاربردها.
  - برخال‌ها (فرکتال‌ها).
  - مقاومت ظاهری در مدارهای RLC.
  - پردازش سیگنال و تحلیل فوریه.
  - نظریه کنترل و تبدیلات لاپلاس.
  - مکانیک کوانتومی و فضاهای هیلبرت.
  - ...



مجموعه مندلبرو

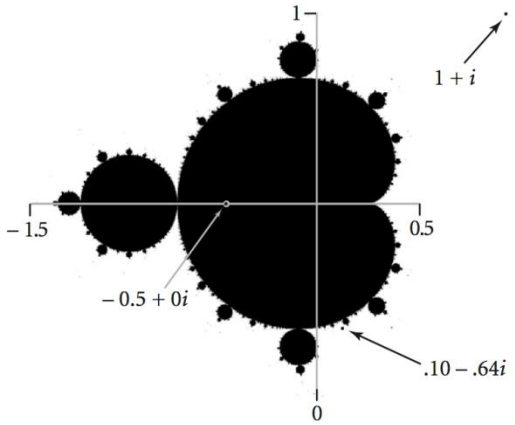
- مجموعه مندلبرو. یک مجموعه از اعداد مختلط.
- ترسیم. اگر `z = x + yi` به مجموعه تعلق دارد، نقطه `(x, y)` را به رنگ سیاه ترسیم کن.



- هیچ فرمول ساده‌ای برای تعیین این که کدام اعداد به مجموعه تعلق دارند، وجود ندارد.
- اما یک توصیف الگوریتمی وجود دارد!

مجموعه مندلبرو

- مجموعه مندلبرو. آیا عدد مختلط z_0 به مجموعه تعلق دارد؟
  - تکرار: $z_{t+1} = (z_t)^2 + z_0$
  - اگر $|z_t|$ واگرا شود، آنگاه z_0 به مجموعه تعلق ندارد؛
  - در غیر این صورت، z_0 به مجموعه تعلق دارد.



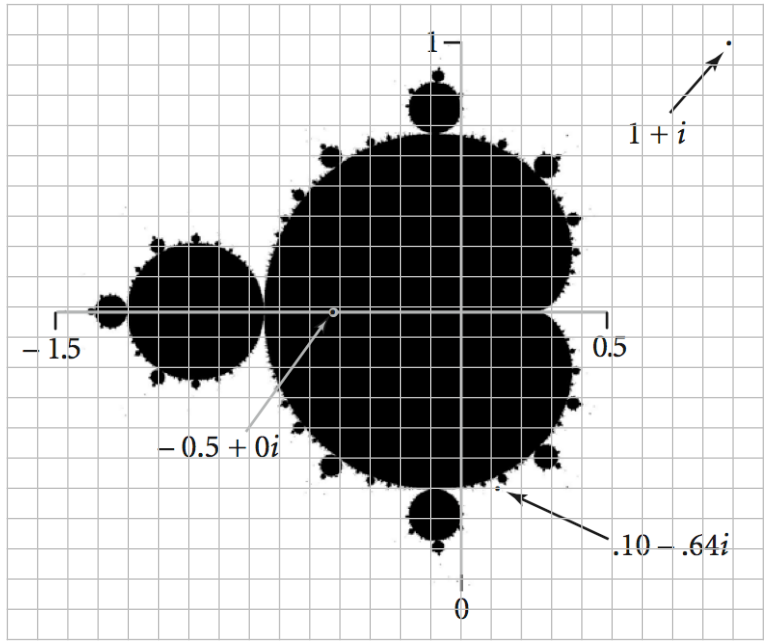
عدد 1 + i در مجموعه مندلبرو قرار ندارد

t	z_t
0	1 + i
1	1 + 3i
2	-7 + 7i
3	1 - 97i
4	-9407 - 193i
5	88454401 + 3631103i

عدد 1/2 در مجموعه مندلبرو قرار دارد

t	z_t
0	-1/2 + 0i
1	-1/4 + 0i
2	-7/16 + 0i
3	-79/256 + 0i
4	-26527/65536 + 0i
5	-1443801919/4294967296 + 0i

ترسیم مجموعه مندلبرو



- ملاحظات عملی.
  - نمی‌توان بی‌نهایت نقطه را ترسیم نمود.
  - نمی‌توان یک حلقه را بی‌نهایت بار تکرار کرد.

- یک راه‌حل تقریبی.
  - از یک شبکه n در n از نقاط صفحه نمونه‌برداری کن.
  - حقیقت: اگر $|z_t| > 2$ به ازای هر t، آنگاه z در مجموعه مندلبرو قرار ندارد.
  - شبه‌حقیقت: اگر $|z_{255}| \leq 2$، آنگاه احتمالاً z در مجموعه مندلبرو قرار دارد.

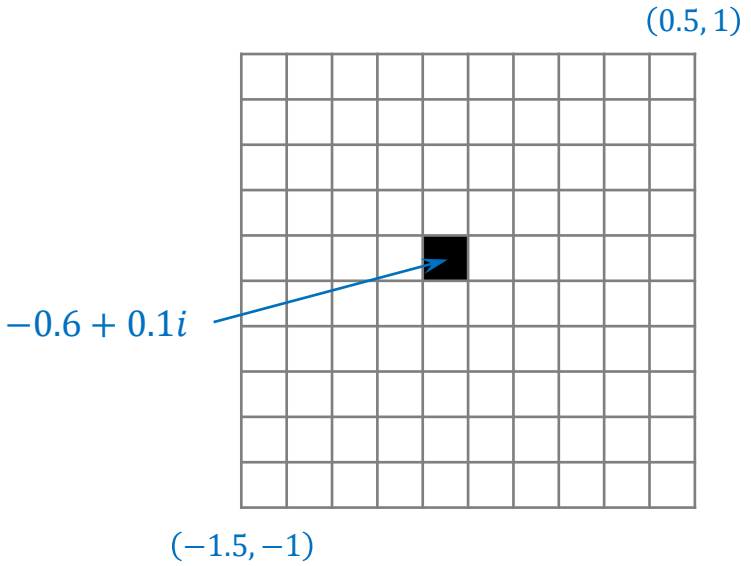
ترسیم مجموعه مندلبرو

- ملاحظات عملی.
  - نمی‌توان بی‌نهایت نقطه را ترسیم نمود.
  - نمی‌توان یک حلقه را بی‌نهایت بار تکرار کرد.

- یک راه‌حل تقریبی.
  - از یک شبکه n در n از نقاط صفحه نمونه‌برداری کن.
  - حقیقت: اگر $|z_t| > 2$ به ازای هر t، آنگاه z در مجموعه مندلبرو قرار ندارد.
  - شبه حقیقت: اگر $|z_{255}| \leq 2$، آنگاه احتمالاً z در مجموعه مندلبرو قرار دارد.

ترسیم مجموعه مندلبرو

- یک راه‌حل تقریبی.
  - از یک شبکه n در n از نقاط صفحه نمونه‌برداری کن.
  - حقیقت: اگر $|z_t| > 2$ به ازای هر t، آنگاه z در مجموعه مندلبرو قرار ندارد.
  - شبه حقیقت: اگر $|z_{255}| \leq 2$، آنگاه احتمالاً z در مجموعه مندلبرو قرار دارد.



نوع داده‌ای عدد مختلط: یک برنامه مشتری دیگر

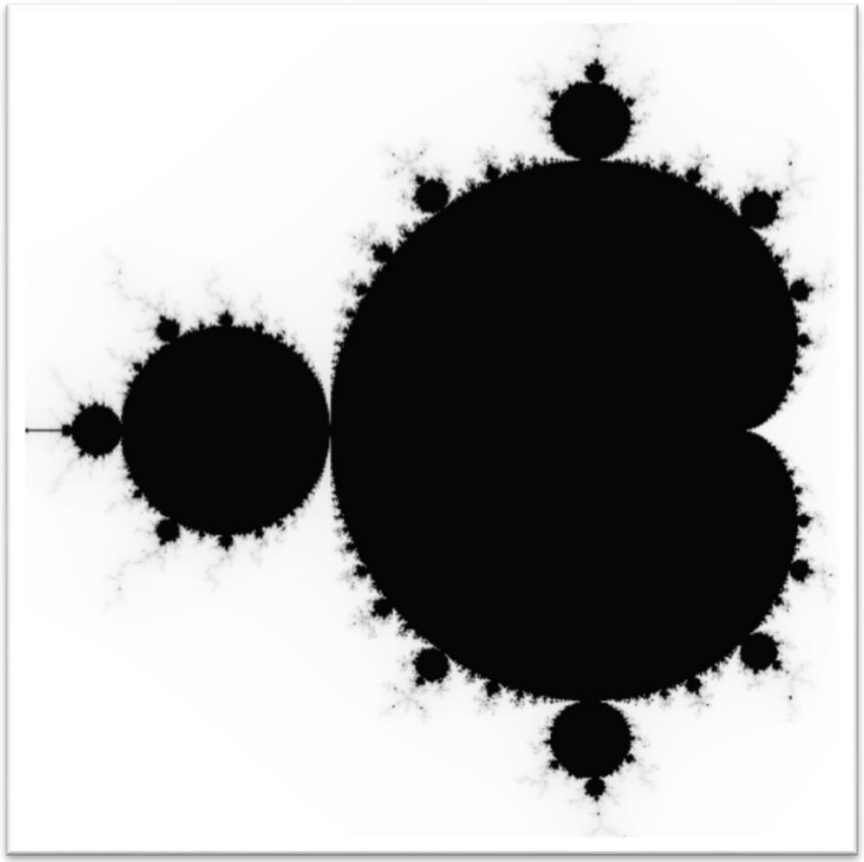
- تابع مندلبرو با اعداد مختلط.
  - آیا z_0 در مجموعه مندلبرو قرار دارد؟
  - خروجی:
    - سفید (قطعاً خیر)
    - سیاه (احتمالاً بله)

```
def mandel(z0, limit=255):
    z = z0
    for t in range(limit):
        if abs(z) > 2.0: return t
        z = z * z + z0
    return limit
```

- تصاویر جذاب‌تر. به جای رنگ سفید از سطوح خاکستری یا رنگی استفاده کن.

نوع داده‌ای عدد مختلط: یک برنامه مشتری دیگر

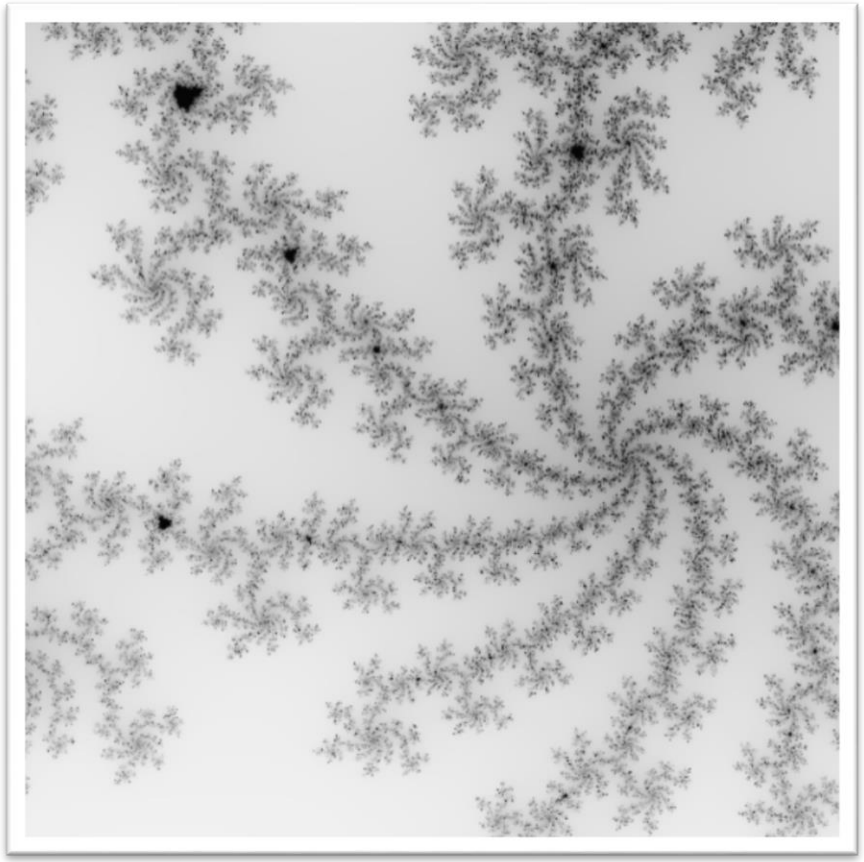

```
MAX = 255

n = int(sys.argv[1])
xc = float(sys.argv[2])
yc = float(sys.argv[3])
size = float(sys.argv[4])

pic = Picture(n, n)
for col in range(n):
    for row in range(n):
        x0 = xc - (size / 2) + (size * col / n)
        y0 = yc - (size / 2) + (size * row / n)
        z0 = Complex(x0, y0)
        gray = MAX - mandel(z0, MAX)
        color = Color(gray, gray, gray)
        pic.set(col, n-1-row, color)

stddraw.setCanvasSize(n, n)
stddraw.picture(pic)
stddraw.show()
```



```
% python mandelbrot.py 512 -.5. 0 2
```

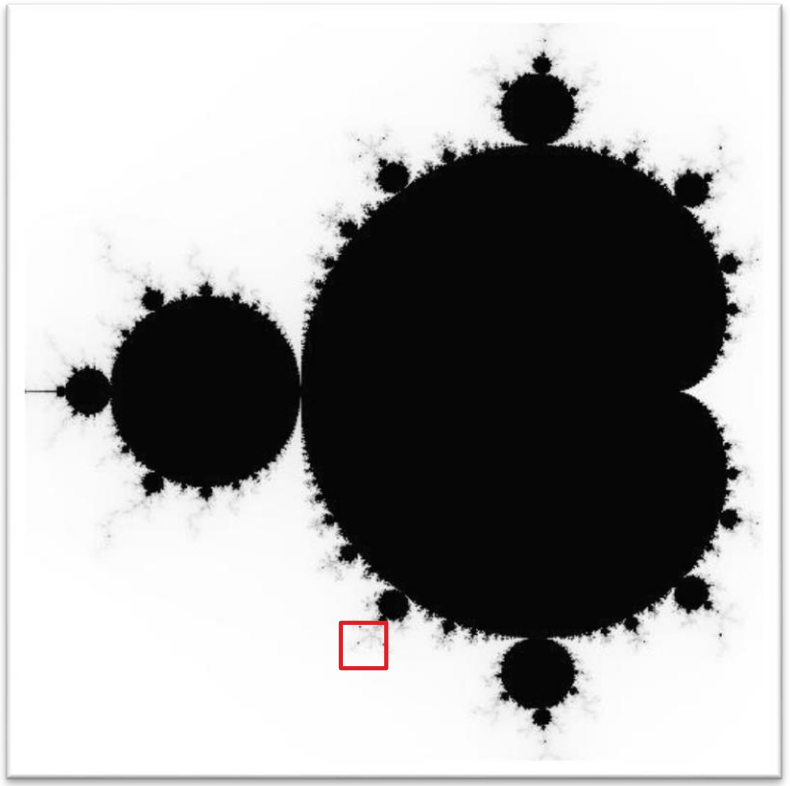
نوع داده‌ای عدد مختلط: یک برنامه مشتری دیگر


```
MAX = 255

n = int(sys.argv[1])
xc = float(sys.argv[2])
yc = float(sys.argv[3])
size = float(sys.argv[4])

pic = Picture(n, n)
for col in range(n):
    for row in range(n):
        x0 = xc - (size / 2) + (size * col / n)
        y0 = yc - (size / 2) + (size * row / n)
        z0 = Complex(x0, y0)
        gray = MAX - mandel(z0, MAX)
        color = Color(gray, gray, gray)
        pic.set(col, n-1-row, color)

stddraw.setCanvasSize(n, n)
stddraw.picture(pic)
stddraw.show()
```

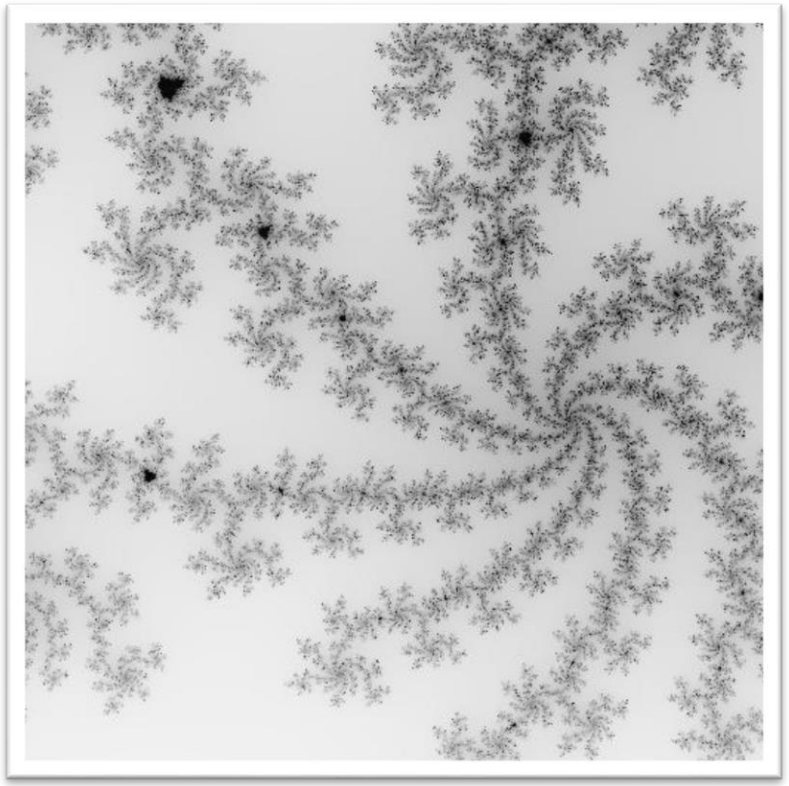


```
% python mandelbrot.py 512 .1045 -.637 .01
```

مجموعه مندلبرو



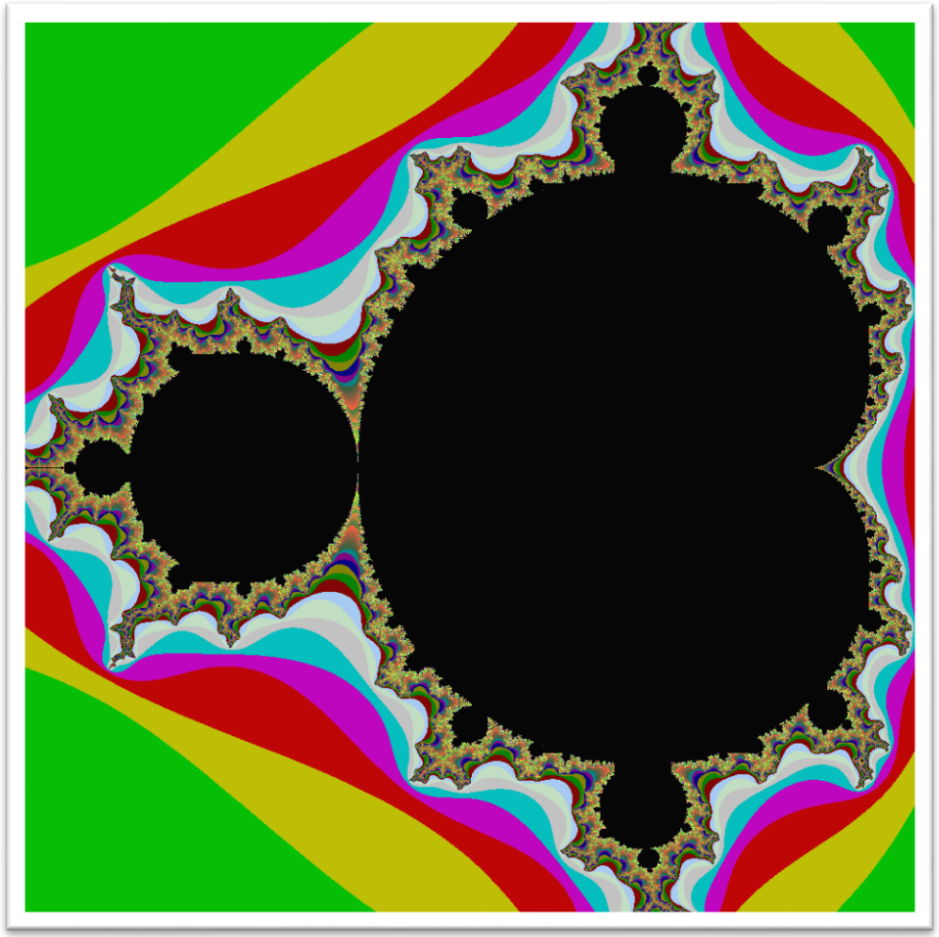
```
% python mandelbrot.py -.5 0 2
```



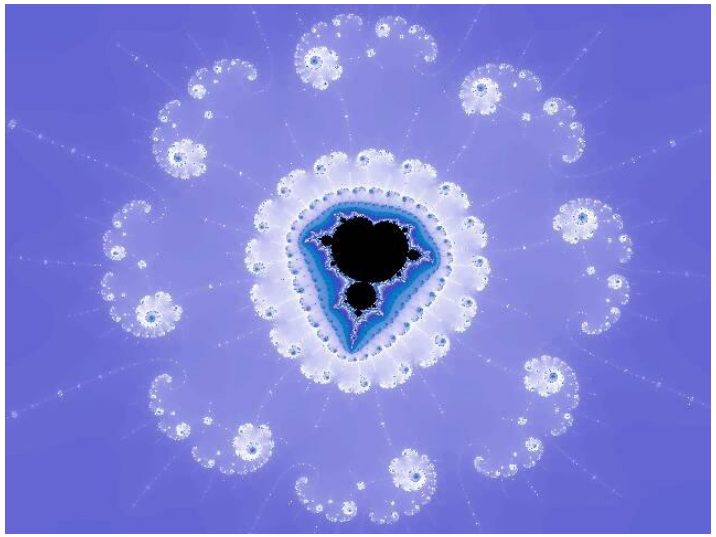
```
% python mandelbrot.py .1045 -.637 .01
```

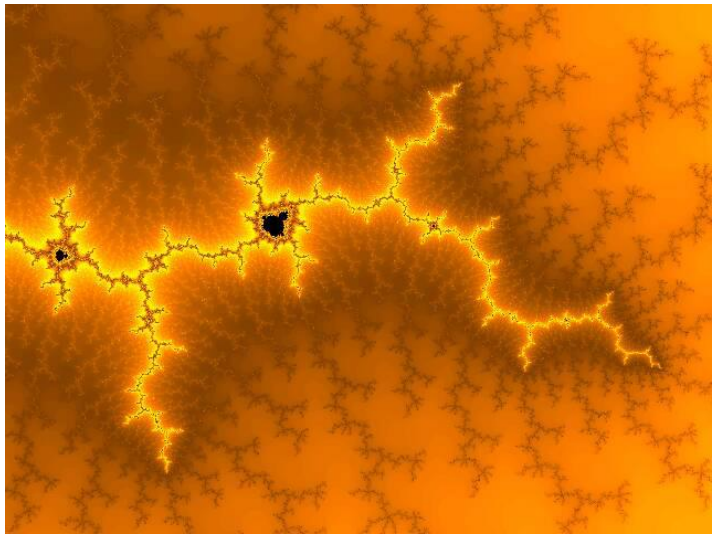
مجموعه مندلبرو


```
% python Colormandelbrot.py -.5 0 2 < mandel.txt
```

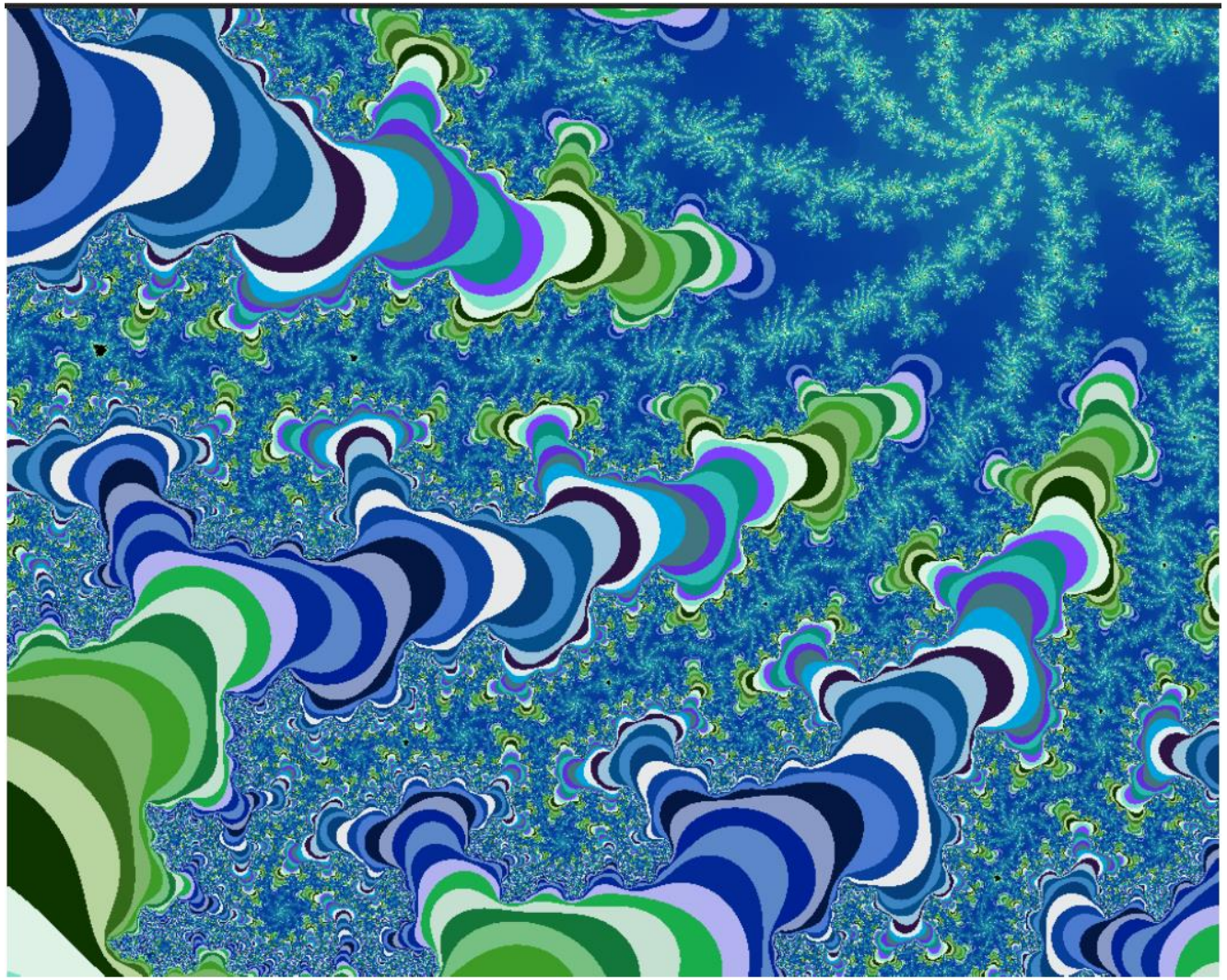


مجموعه مندلبرو





مجموعه مندلبرو



انواع داده‌ای: کاربردها

- نوع داده‌ای. یک مجموعه از مقادیر به همراه عملیات قابل انجام بر روی آن مجموعه.

- شبیه‌سازی دنیای فیزیکی.
  - اشیای پایتون مدل کننده اشیای دنیای واقعی هستند.
  - ایجاد مدل‌هایی که بیانگر واقعیت هستند، همیشه ساده نیست.
  - مثال: ذره باردار، مولکول، دانشجوی کلاس پایتون، ...

- گسترش دادن زبان پایتون.
  - در پایتون همه اشیای مورد نیاز در کاربردهای گوناگون پیش‌بینی نشده است.
  - انواع داده‌ای به ما امکان می‌دهند انواع مورد نیازمان را خودمان ایجاد کنیم.
  - مثال: عدد مختلط، بردار، چندجمله‌ای، ماتریس، ...