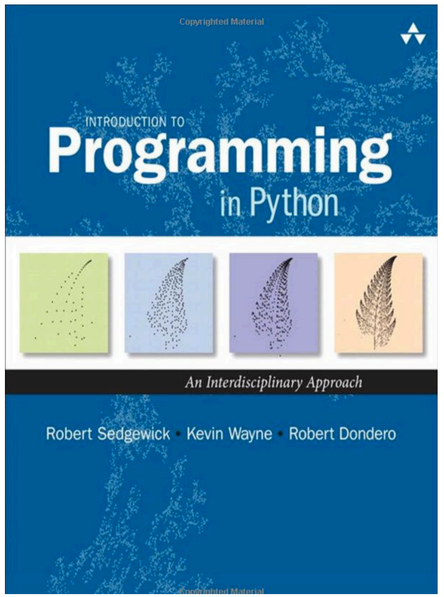
Parte II: Computación científica

Clase 15: Animación y computación numérica

Diego Caro
dcaro@udd.cl



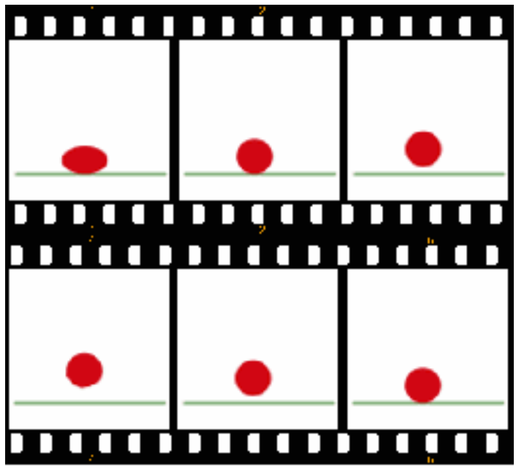
Basada en presentaciones oficiales de libro Introduction to Programming in Python (Sedgewick, Wayne, Dondero).

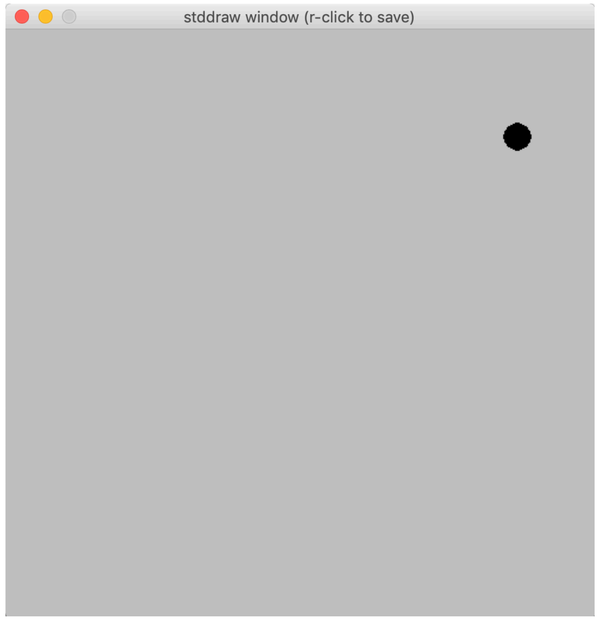
Disponible en https://introcs.cs.princeton.edu/python

Biblioteca multimedia: http://github.com/diegocaro/introcs

Animación

- Movimiento puede simularse intercalando imágenes.
- Ejemplo: programar pelota que rebota en los bordes de la ventana.
- Estrategia: programar un ciclo infinito con
  - Cálculo de posición de pelota
  - Dibujar el fondo
  - Dibujar pelota (con la nueva posición)





Ejemplo: Pelota simple

```
 1 import stddraw
 2
 3 stddraw.setCanvasSize(500, 500)
 4 stddraw.setXscale(-1.0, 1.0)
 5 stddraw.setYscale(-1.0, 1.0)
 6
 7 radius = .05
 8 rx = .080
 9 ry = .060
10 vx = .015
11 vy = .013
12
13 while True:
14     # update position
15     rx = rx + vx
16     ry = ry + vy
17
18     # clear the background
19     stddraw.clear(stddraw.LIGHT_GRAY)
20
21     # draw the ball on the screen
22     stddraw.setPenColor(stddraw.BLACK)
23     stddraw.filledCircle(rx, ry, radius)
24
25     # copy buffer to screen
26     stddraw.show(0)
27     stddraw.pause(20)
```

Configuración de la ventana

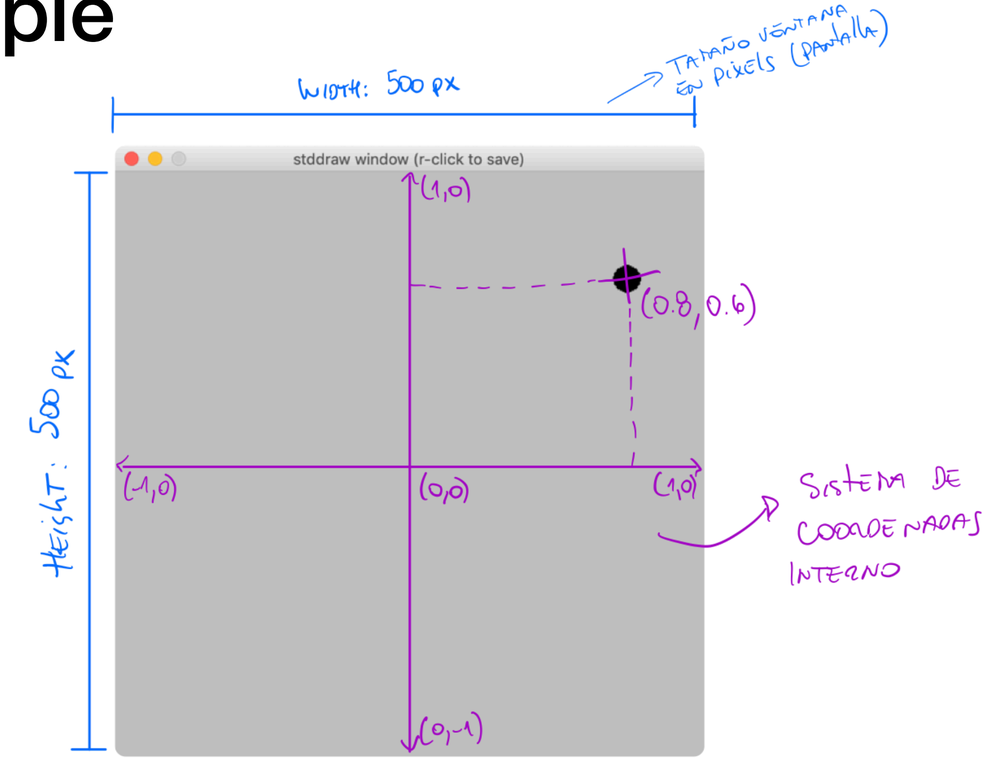
Radio, posición y velocidad

Actualización posición asumiendo velocidad constante (aceleración=0)

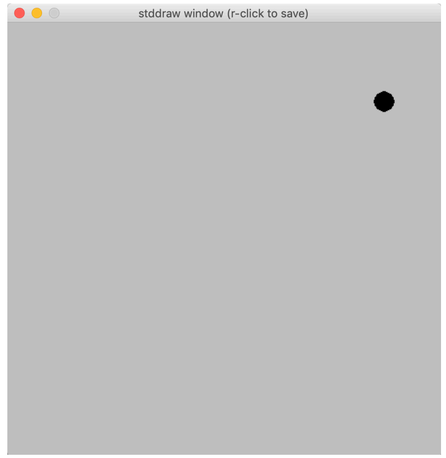
Redibuja el fondo

Dibuja pelota en nueva posición

Espera 20 milisegundos para dibujar siguiente frame

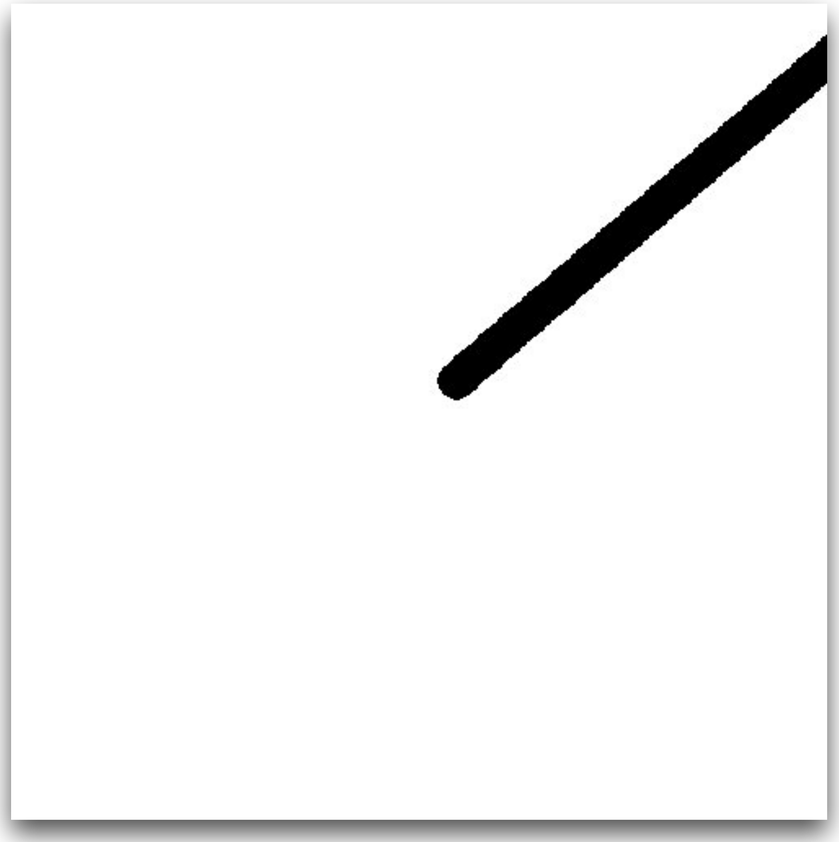


$$x_1 = x_0 + v_0 + \frac{1}{2}a_0 t^2$$



Preguntas

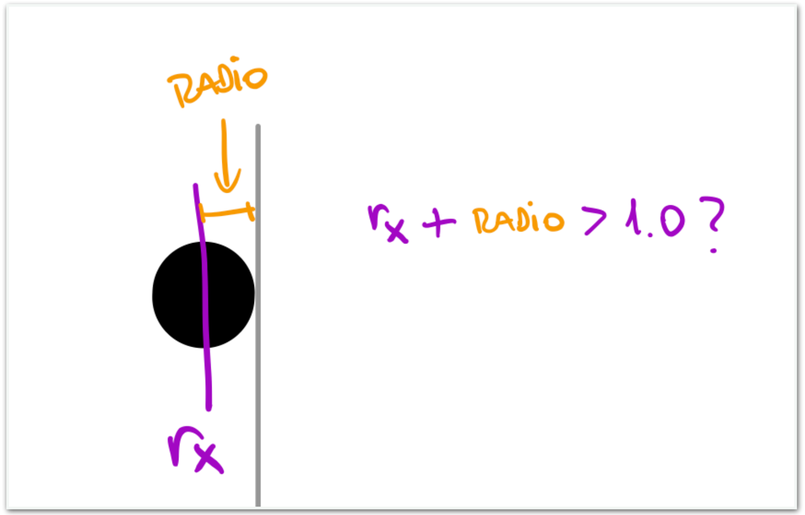
- ¿Qué sucede si no limpiamos el fondo?
  - No se borra lo que dibujamos en el ciclo anterior.



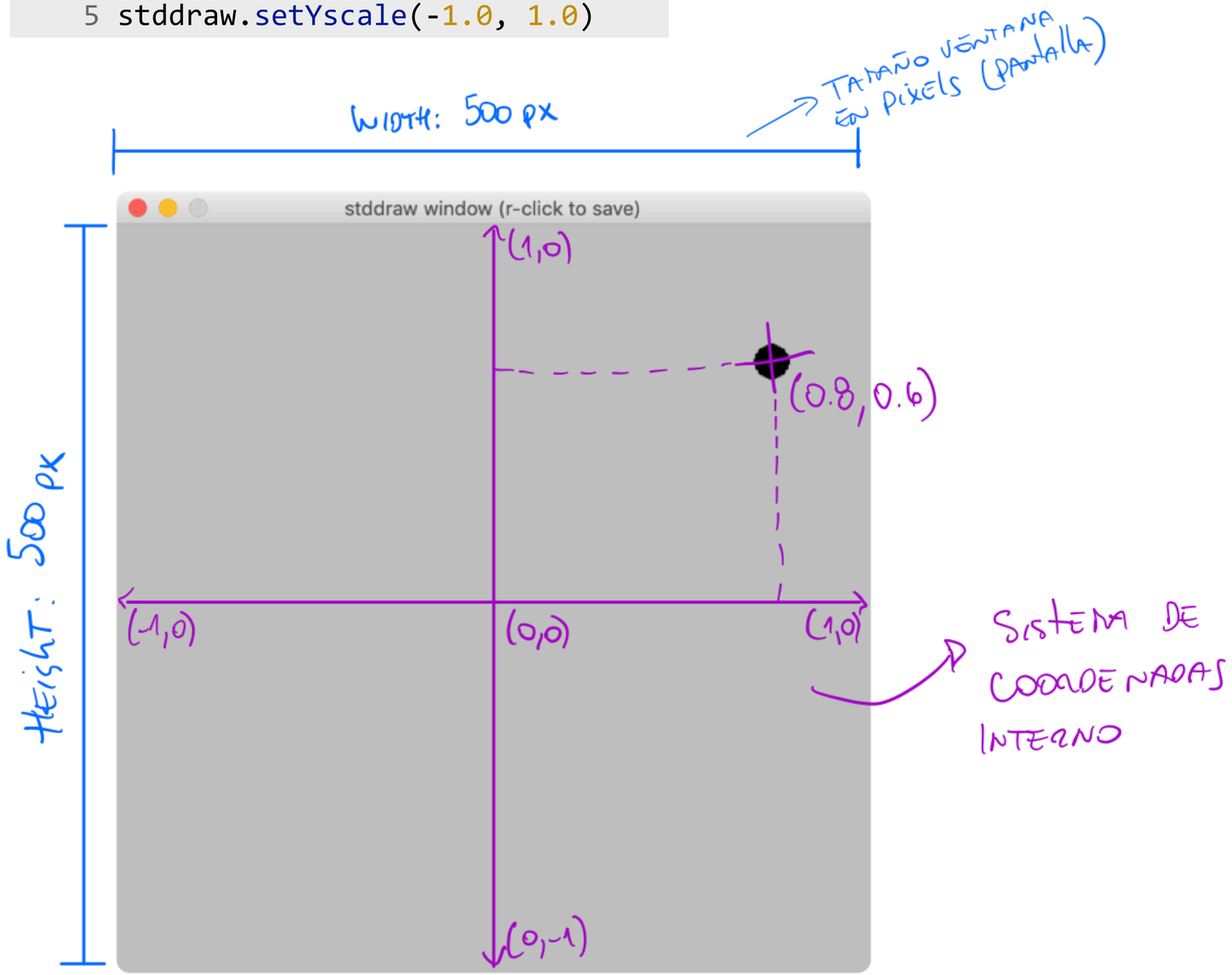
Preguntas

```
3 stddraw.setCanvasSize(500, 500)
4 stddraw.setXscale(-1.0, 1.0)
5 stddraw.setYscale(-1.0, 1.0)
```

- ¿Cómo podemos detectar que la pelota sale de la ventana?

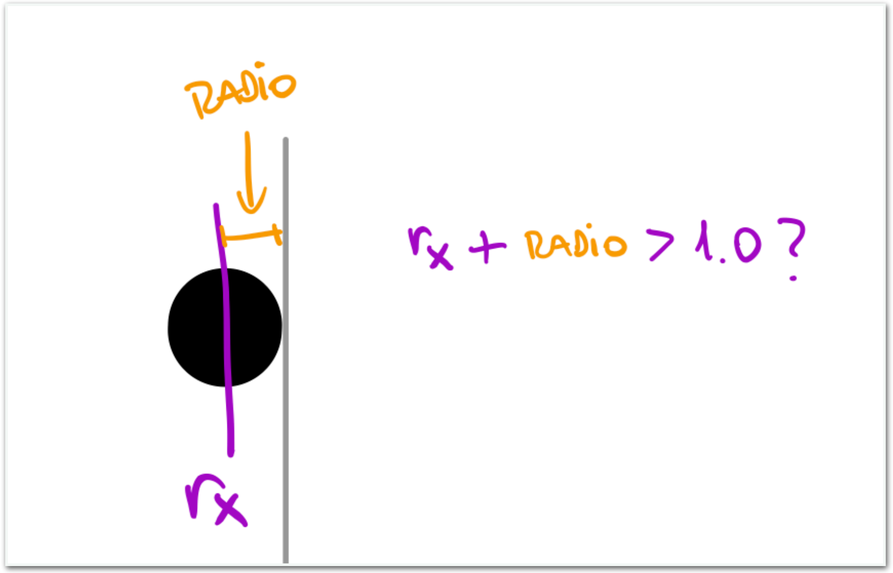



```
if abs(rx + vx) + radius > 1.0:
    print('choque con borde!')
if abs(ry + vy) + radius > 1.0:
    print('choque con borde!')
```



Preguntas

- ¿Cómo podemos hacer que la pelota rebote siguiendo las leyes de colisión elástica?
  - Asume que el borde de la ventana es de masa infinita y no se mueve.



Conservación del momento lineal:

$$m_1 v_1 + m_2 v_2 = m_1 u_1 + m_2 u_2$$

Conservación de la Energía (cinética):

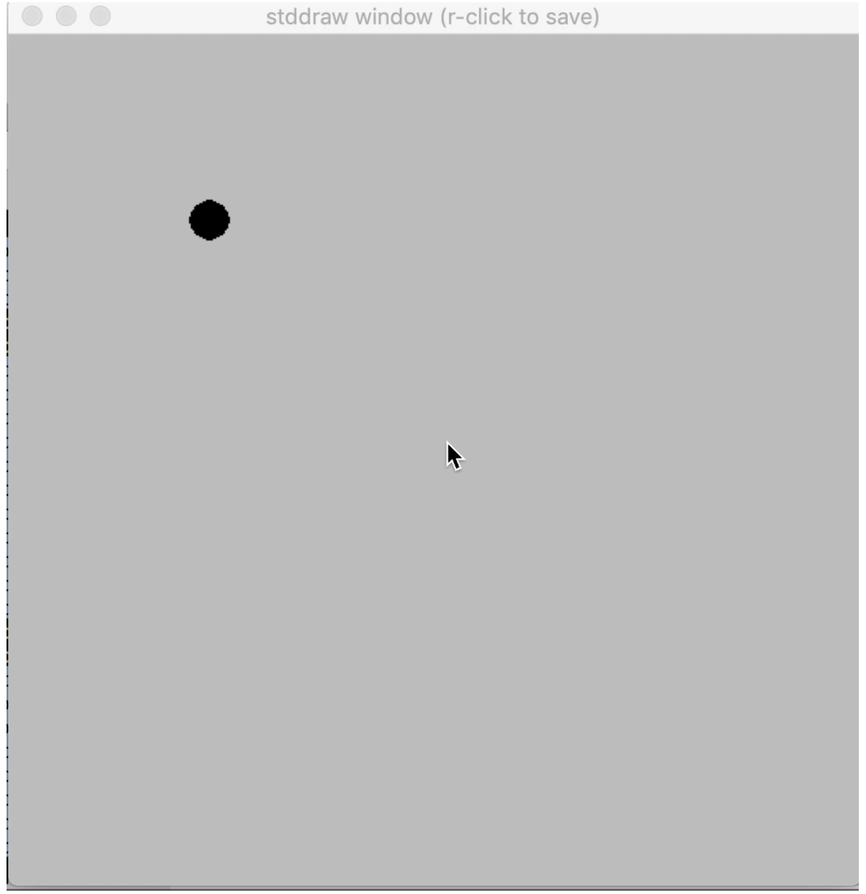
$$\frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2 = \frac{1}{2} m_1 u_1^2 + \frac{1}{2} m_2 u_2^2$$

https://es.wikipedia.org/wiki/Choque_el%C3%A1stico

```
radius = .05
rx = .480
ry = .860
vx = .015
vy = .023

while True:
    if abs(rx + vx) + radius > 1.0:
        vx = -vx
    if abs(ry + vy) + radius > 1.0:
        vy = -vy

    rx = rx + vx
    ry = ry + vy

    stddraw.clear(stddraw.LIGHT_GRAY)

    stddraw.setPenColor(stddraw.BLACK)
    stddraw.filledCircle(rx, ry, radius)

    stddraw.show(0)
    stddraw.pause(20)
```

Colisión elástica con la pared

Actualización posición asumiendo velocidad constante (aceleración=0)



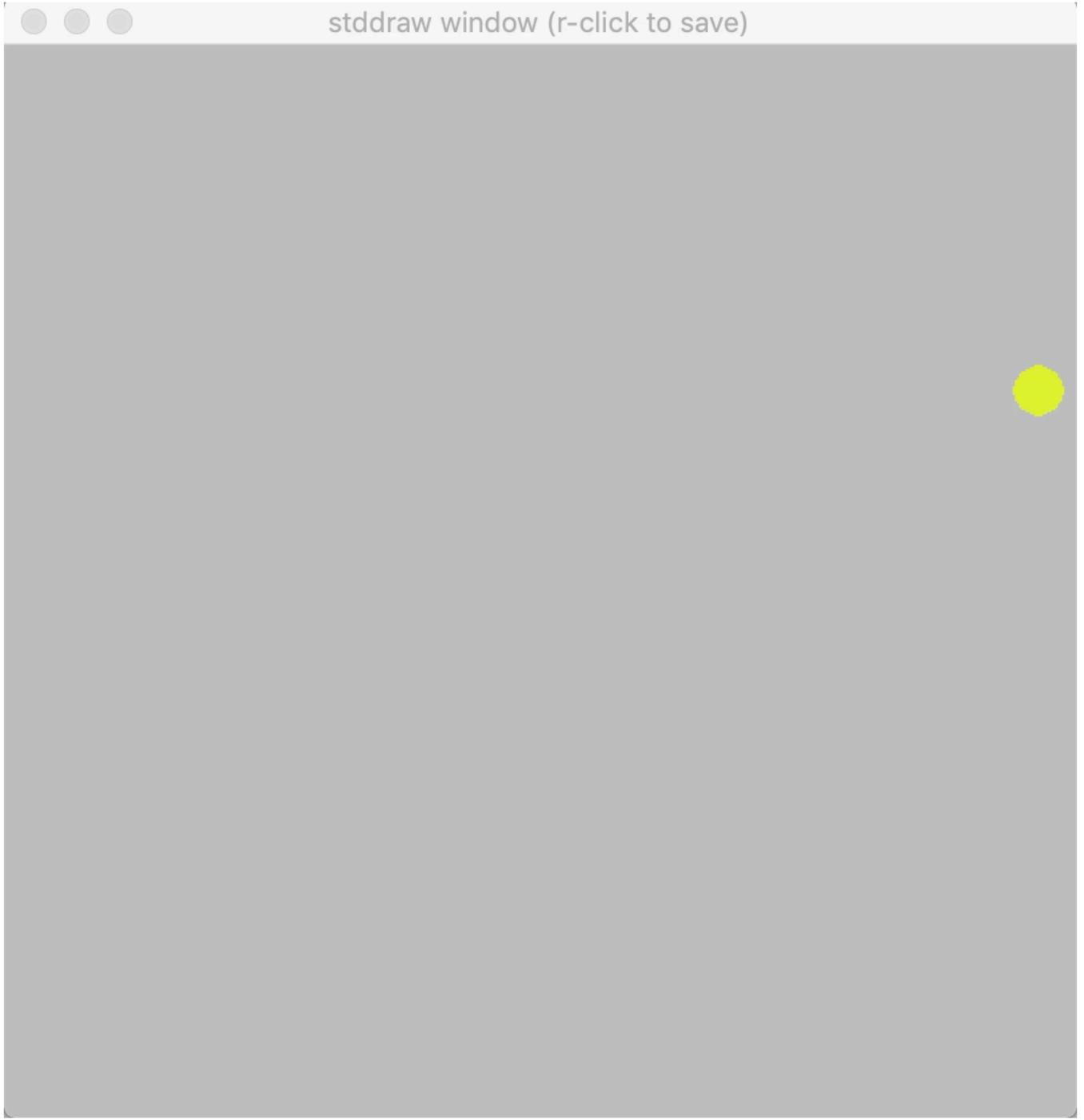
DEMO TIME

$ python3 bouncingball.py

¿Cómo hacer una pelota multicolor?

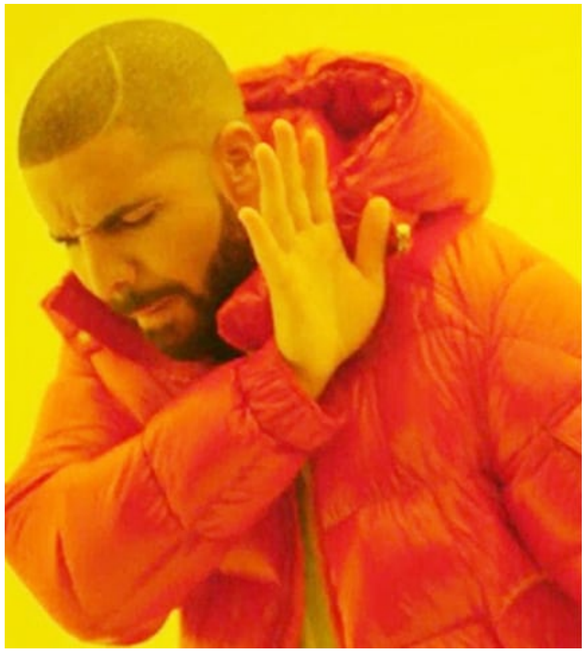
```
while True:
    if abs(rx + vx) + radius > 1.0:
        vx = -vx
    if abs(ry + vy) + radius > 1.0:
        vy = -vy

    rx = rx + vx
    ry = ry + vy

    stddraw.clear(stddraw.LIGHT_GRAY)

    stddraw.setPenColor(stddraw.BLACK)
    stddraw.filledCircle(rx, ry, radius)

    stddraw.show(0)
    stddraw.pause(20)
```

```
while True:
   if abs(rx + vx) + radius > 1.0:
        vx = -vx
    if abs(ry + vy) + radius > 1.0:
        vy = -vy

    rx = rx + vx
    ry = ry + vy

    stddraw.clear(stddraw.LIGHT_GRAY)

    r = randrange(256)
    g = randrange(256)
    b = randrange(256)
    c = Color(r, g, b)

    stddraw.setPenColor(c)
    stddraw.filledCircle(rx, ry, radius)

    stddraw.show(0)
    stddraw.pause(20)
```

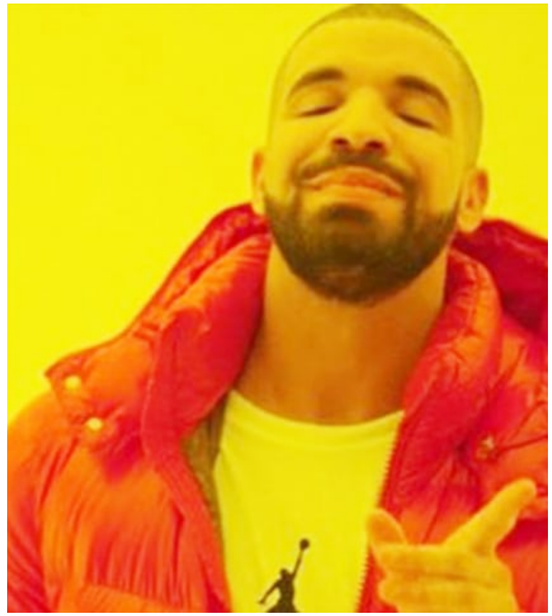

stddraw window (r-click to save)

¿Cómo mostrar 3 pelotas?

```
import
stddraw.setCanvasSize(500, 500)
stddraw.setXscale(0, 1.0)
stddraw.setYscale(0, 1.0)

radius = .05
rx = .480
ry = .860
vx = .015
vy = .023

radius2 = .05
rx2 = .480
ry2 = .860
vx2 = .030
vy2 = .063

radius3 = .05
rx3 = .480
ry3 = .860
vx3 = .01
vy3 = .013

while True:
    # bounce of wall according to elastic collision
    # update velocity
    if abs(rx + vx) + radius > 1.0:
        vx = -vx
    if abs(ry + vy) + radius > 1.0:
        vy = -vy
    if abs(rx2 + vx2) + radius2 > 1.0:
        vx2 = -vx2
    if abs(ry2 + vy2) + radius2 > 1.0:
        vy2 = -vy2
    if abs(rx3 + vx3) + radius3 > 1.0:
        vx3 = -vx3
    if abs(ry3 + vy3) + radius3 > 1.0:
        vy3 = -vy3
    # update position
    rx = rx + vx
    ry = ry + vy
    rx2 = rx2 + vx2
    ry2 = ry2 + vy2
    rx3 = rx3 + vx3
    ry3 = ry3 + vy3
    # clear the background
    stddraw.clear(stddraw.LIGHT_GRAY)
    # draw the ball on the screen
    stddraw.setPenColor(stddraw.BLACK)
    stddraw.filledCircle(rx, ry, radius)
    stddraw.filledCircle(rx2, ry2, radius2)
    stddraw.filledCircle(rx3, ry3, radius3)
    # display and pause for 20 ms
    stddraw.show(0)
    stddraw.pause(20)
```



Estrategia: crear una clase



```
class Ball:
    def __init__(self, rx, ry, vx, vy, radius, color):
        self.rx = rx
        self.ry = ry
        self.vx = vx
        self.vy = vy
        self.radius = radius
        self.color = color

    def update(self):
        if abs(self.rx + self.vx) + self.radius > 1.0:
            self.vx = -self.vx
        if abs(self.ry + self.vy) + self.radius > 1.0:
            self.vy = -self.vy

        self.rx = self.rx + self.vx
        self.ry = self.ry + self.vy

    def draw(self):
        stddraw.setPenColor(self.color)
        stddraw.filledCircle(self.rx, self.ry, self.radius)
```

Colisión elástica con la pared

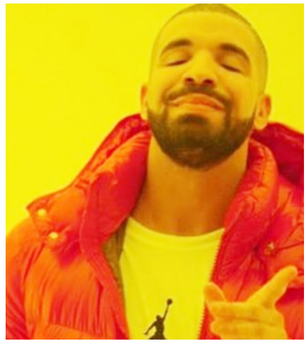
Actualización posición

Dibujar!

Código cliente para una pelota

```
ball = Ball(.480, .860, .015, .023, .05, stddraw.BLACK)

while True:
    # update velocity
    ball.update()
    # clear the background
    stddraw.clear(stddraw.LIGHT_GRAY)

    # draw the ball on the screen
    ball.draw()

    # copy buffer to screen
    stddraw.show(0)
    stddraw.pause(20)
```

Solución: crear una lista de objetos Ball



```
1  import stddraw
2  from ball import Ball
3
4  stddraw.setCanvasSize(500, 500)
5
6  stddraw.setXscale(-1.0, 1.0)
7  stddraw.setYscale(-1.0, 1.0)
8
9  balls = [
10     Ball(.480, .860, .015, .023, .05, stddraw.BLACK),
11     Ball(.480, .860, .030, .046, .05, stddraw.BLUE),
12     Ball(.180, .260, .040, .026, .05, stddraw.GREEN)
13 ]
14
15 while True:
16     # update velocity
17     for b in balls:
18         b.update()
19
20     # clear the background
21     stddraw.clear(stddraw.LIGHT_GRAY)
22
23     # draw the ball on the screen
24     for b in balls:
25         b.draw()
26
27     # copy buffer to screen
28     stddraw.show(0)
29     stddraw.pause(20)
```

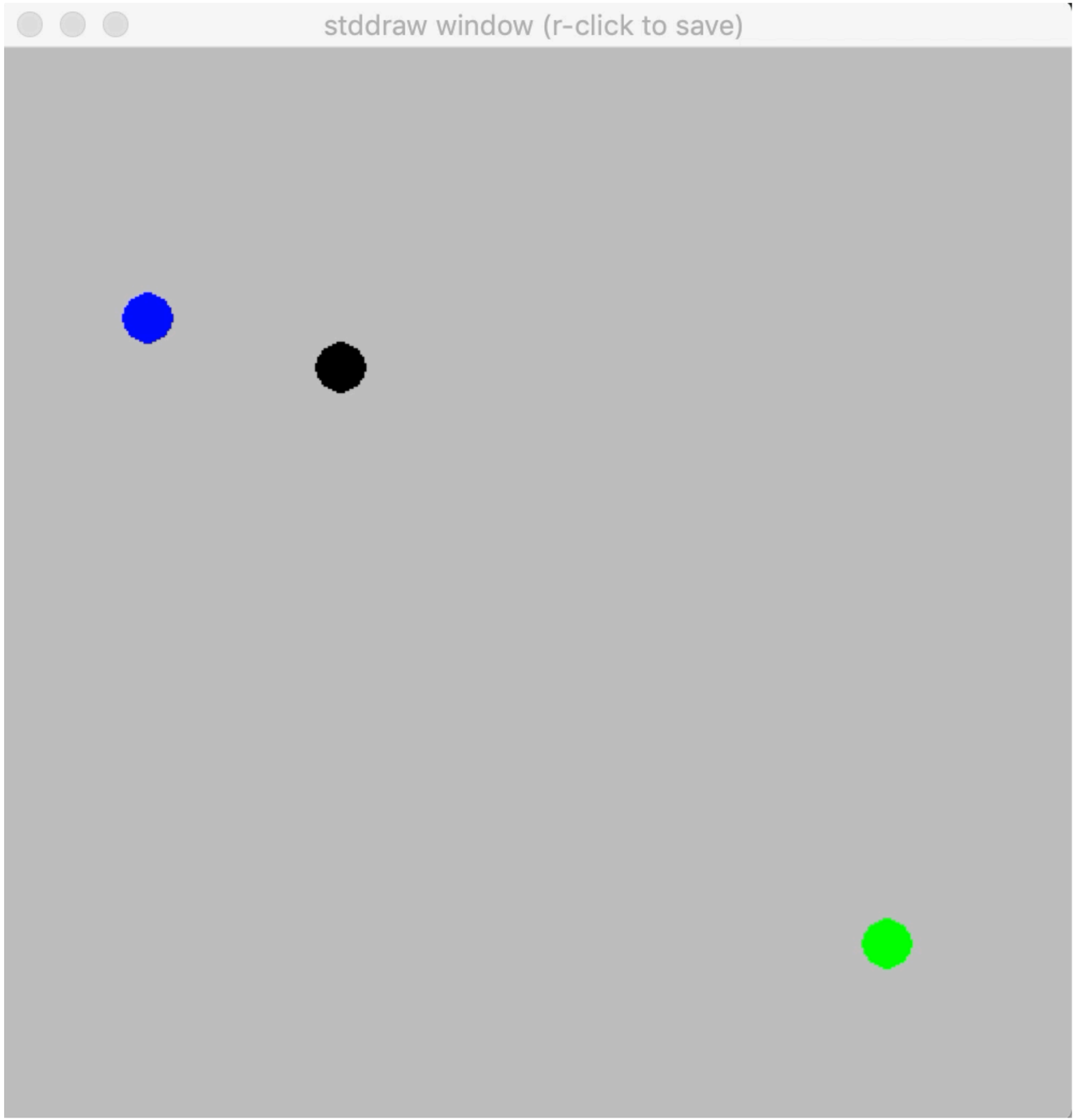
Módulo ball.py

```
1  import stddraw
2
3  class Ball:
4      def __init__(self, rx, ry, vx, vy, radius, color):
5          self.rx = rx
6          self.ry = ry
7          self.vx = vx
8          self.vy = vy
9          self.radius = radius
10         self.color = color
11
12     def update(self):
13         """
14         Bounce of wall according to elastic collition and
15         update velocity.
16         """
17         if abs(self.rx + self.vx) + self.radius > 1.0:
18             self.vx = -self.vx
19         if abs(self.ry + self.vy) + self.radius > 1.0:
20             self.vy = -self.vy
21
22         self.rx = self.rx + self.vx
23         self.ry = self.ry + self.vy
24
25     def draw(self):
26         stddraw.setPenColor(self.color)
27         stddraw.filledCircle(self.rx, self.ry, self.radius)
```

Colisión

Actualización posición

Dibujar!


stddraw window (r-click to save)

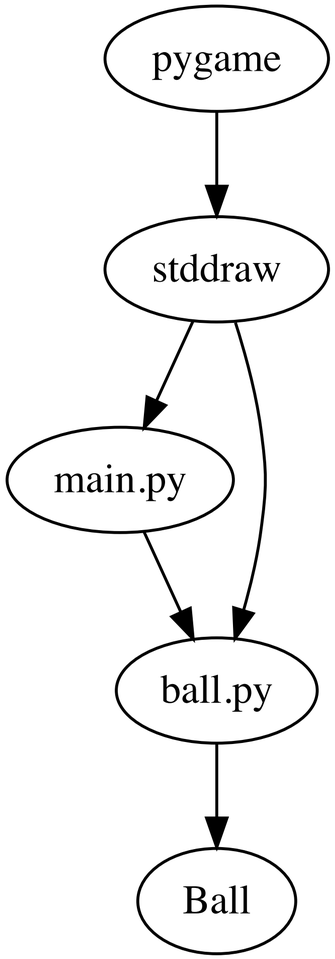
Usando el teclado

```
balls = [
    Ball(.480, .860, .015, .023, .05, stddraw.BLACK),
    Ball(.480, .860, .030, .046, .05, stddraw.BLUE),
    Ball(.180, .260, .040, .026, .05, stddraw.GREEN)
]

while True:
    # get keystrokes
    if stddraw.hasNextKeyTyped():
        k = stddraw.nextKeyTyped()
        if k == stddraw.K_UP:
            for b in balls: b.increase_speed(0.1, 0.1)
        elif k == stddraw.K_DOWN:
            for b in balls: b.increase_speed(-0.1, -0.1)

    # update velocity
    for b in balls: b.update()

    # clear the background
    stddraw.clear(stddraw.LIGHT_GRAY)

    # draw the ball on the screen
    for b in balls: b.draw()

    # copy buffer to screen
    stddraw.show(0)
    stddraw.pause(20)
```

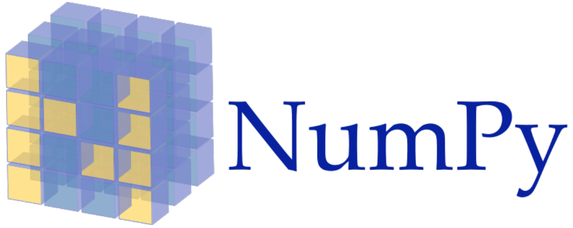
Códigos para teclas en https://github.com/josiest/pygtails/blob/master/docs/pygstants.rst

Keycode Name	Ascii	Description
K_BACKSPACE	\b	backspace
K_TAB	\t	tab
K_CLEAR		clear
K_RETURN	\r	return
K_PAUSE		pause
K_ESCAPE	^[	escape
K_SPACE		space
K_UP		up arrow
K_DOWN		down arrow
K_RIGHT		right arrow
K_LEFT		left arrow

Dependencia entre módulos



Nota: módulo pygame se escapa del ámbito de este curso. Usaremos la biblioteca introcs disponible en https://github.com/diegocaro/introcs



Intro a módulo Numpy

http://www.numpy.org/

- Python no incluye un tipo de dato para matrices, por lo tanto solo queda la alternativa de implementarla con listas....
- Pero existe numpy, un módulo para representar vectores, matrices y tensores.
  - Además incluye muchísimas operaciones para trabajar con matrices!
- Puede representar bool, int, float y complex.

```
1  import numpy as np
2  # integer array:
3  intarr = np.array([1, 4, 2, 5, 3])
4  print(intarr)
5
6  mixedarr = np.array([3.14, 4, 2, 3])
7  print(mixedarr)
8
9  print(np.zeros(10, dtype=int))
10
11 print(np.ones((3, 5), dtype=float))
```

Se pueden mezclar de datos numéricos

Vector de ceros

matriz de 3 filas y 5 cols lleno con 1s

```
[1 4 2 5 3]
[3.14 4.   2.   3.  ]
[0 0 0 0 0 0 0 0 0 0]
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

Operaciones básicas

```python
import numpy as np
np.random.seed(0)  # seed for reproducibility

# One-dimensional array (vector)
x1 = np.random.randint(10, size=6)

# Two-dimensional array (matrix)
x2 = np.random.randint(10, size=(3, 4))

print(x1)
print(x2)

print("x2 ndim: ", x2.ndim)
print("x2 shape:", x2.shape)
print("x2 size: ", x2.size)
print("dtype:", x2.dtype)

print('x1[0]:', x1[0])
print('x1[4]:', x1[4])
print('x1[-1]:', x1[-1])

print('x2[0, 0]:',x2[0, 0])
print('x2[2, -1]:',x2[2, -1])

x2[0, 0] = 99
print('x2[0, 0]:',x2[0, 0])
```

Se puede acceder con índices

En matrices se debe indicar col, row

También se pueden actualizar valores

```
[5 0 3 3 7 9]
[[3 5 2 4]
 [7 6 8 8]
 [1 6 7 7]]
x2 ndim:  2
x2 shape: (3, 4)
x2 size:  12
dtype: int64
x1[0]: 5
x1[4]: 7
x1[-1]: 9
x2[0, 0]: 3
x2[2, -1]: 7
x2[0, 0]: 99
```

https://docs.scipy.org/doc/numpy-1.15.4/reference/generated/numpy.random.randint.html

https://github.com/jakevdp/PythonDataScienceHandbook/blob/be23269c7eb119e093a6d5ce91e464f5e686d9ab/notebooks/02.02-The-Basics-Of-NumPy-Arrays.ipynb

Subvector / submatrices y acceso

```
5 x2 = np.random.randint(10, size=(3, 4))
6 print('x2[:2, :3]:', x2[:2, :3])  # two rows, three columns
7
8 print('x2[:, 0]:', x2[:, 0])  # first column of x2
9
10 print('x2[0, :]:', x2[0, :])  # first row of x2
11
12 x2_sub = x2[:2, :2]
13 print('x2[:2, :2]:',x2_sub)
14
15 # update the array
16 x2_sub[0, 0] = 99
17 print('x2_sub:', x2_sub)
18 print('x2:',x2)
19
20 x2_sub_copy = x2[:2, :2].copy()
21 print('x2_sub_copy:', x2_sub_copy)
22
23 x2_sub_copy[0, 0] = 42
24 print('x2_sub_copy:', x2_sub_copy)
25 print('x2', x2)
```

Alerta de alias!

```
x2[:2, :3]: [[5 0 3]
 [7 9 3]]
x2[:, 0]: [5 7 2]
x2[0, :]: [5 0 3 3]
x2[:2, :2]: [[5 0]
 [7 9]]
x2_sub: [[99  0]
 [ 7  9]]
x2: [[99  0  3  3]
 [ 7  9  3  5]
 [ 2  4  7  6]]
x2_sub_copy: [[99  0]
 [ 7  9]]
x2_sub_copy: [[42  0]
 [ 7  9]]
x2 [[99  0  3  3]
 [ 7  9  3  5]
 [ 2  4  7  6]]
```

Concatenación

```
 1 import numpy as np
 2 x = np.array([1, 2, 3])
 3 y = np.array([3, 2, 1])
 4 c = np.concatenate([x, y])
 5 print('c:', c)
 6
 7 grid = np.array([[1, 2, 3],
 8                  [4, 5, 6]])
 9
10 # concatenate along the first axis
11 print('axis=0:',np.concatenate([grid, grid]))
12
13 # concatenate along the second axis (zero-indexed)
14 print('axis=1:',np.concatenate([grid, grid], axis=1))
15
```

Axis = 0 significa concatenar por filas
Axis = 1 significa concatenar por columnas

```
c: [1 2 3 3 2 1]
axis=0: [[1 2 3]
 [4 5 6]
 [1 2 3]
 [4 5 6]]
axis=1: [[1 2 3 1 2 3]
 [4 5 6 4 5 6]]
```

Operaciones básicas vectores

- Aritmética con vectores, vector & escalar, vector & vector
  - + - * / %

Versión numpy

```
 1 import numpy as np
 2
 3 a = np.array([5, 0, 3, 3, 7, 9])
 4 b = np.array([3, 5, 2, 4, 7, 6])
 5
 6 c = 2 + a # scalar + vector
 7 print('c:', c)
 8
 9 c = a + b # vector + vector (also -,/,*,%)
10 print('c:', c)
```

```
c: [ 7  2  5  5  9 11]
c: [ 8  5  5  7 14 15]
```

Versión listas

```
 1 a = [5, 0, 3, 3, 7, 9]
 2 b = [3, 5, 2, 4, 7, 6]
 3 c = [0]*len(a)
 4
 5 # scalar and vector
 6 for i in range(len(a)): c[i] = 2 + a[i]
 7 print('c:', c)
 8
 9 # vector and vector
10 for i in range(len(a)): c[i] = a[i] + b[i]
11 print('c:', c)
```

```
c: [7, 2, 5, 5, 9, 11]
c: [8, 5, 5, 7, 14, 15]
```

Operaciones básicas matrices

- Aritmética con matrices + - * / %

```python
import numpy as np

a = np.array([[5, 0, 3, 3, 7, 9],[1, 1, 1, 1 ,1, 1]])
b = np.array([3, 5, 2, 4, 7, 6])

c = 2 + a # scalar + matrix
print('c:', c)

c = b + a # vector + matrix
print('c:', c)

c = a + a # matrix + matrix
print('c:', c)
```

```
c: [[ 7  2  5  5  9 11]
 [ 3  3  3  3  3  3]]
c: [[ 8  5  5  7 14 15]
 [ 4  6  3  5  8  7]]
c: [[10  0  6  6 14 18]
 [ 2  2  2  2  2  2]]
```

Más operaciones

```python
import numpy as np
m = np.array([[1, -2, 3],
              [4, 5, -6]])
# matrix transpuesta
print('m.T:', m.T)

# dot product
print('m.dot(m.T):', m.dot(m.T))

# inverse matrix
a = np.array([[1., 2.], [3., 4.]])
ainv = np.linalg.inv(a)
print('ainv:', ainv)
print('a * ainv:', a.dot(ainv))
```

```
m.T: [[ 1  4]
 [-2  5]
 [ 3 -6]]
m.dot(m.T): [[ 14 -24]
 [-24  77]]
ainv: [[-2.   1. ]
 [ 1.5 -0.5]]
a * ainv: [[1.0000000e+00 0.0000000e+00]
 [8.8817842e-16 1.0000000e+00]]
```

https://docs.scipy.org/doc/numpy-1.15.1/reference/routines.array-manipulation.html

Aplicaciones: resolver sistemas de ecuaciones

De acuerdo con la primera ley de Kirchhoff (ley de los nodos), tenemos:

$$i_1 - i_2 - i_3 = 0$$

La segunda ley de Kirchhoff (ley de las mallas), aplicada a la malla según el circuito cerrado s_1, nos hace obtener:

$$-R_2 i_2 + \epsilon_1 - R_1 i_1 = 0$$

La segunda ley de Kirchhoff (ley de las mallas), aplicada a la malla según el circuito cerrado s_2, por su parte:

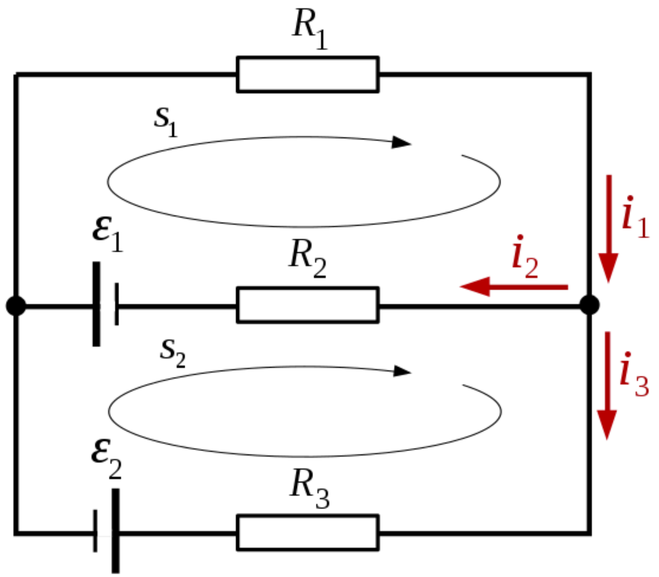
$$-R_3 i_3 - \epsilon_2 - \epsilon_1 + R_2 i_2 = 0$$

Debido a lo anterior, se nos plantea un sistema de ecuaciones con las incógnitas i_1, i_2, i_3:

$$\begin{cases} i_1 - i_2 - i_3 & = 0 \\ -R_2 i_2 + \epsilon_1 - R_1 i_1 & = 0 \\ -R_3 i_3 - \epsilon_2 - \epsilon_1 + R_2 i_2 & = 0 \end{cases}$$

Dadas las magnitudes:

$$R_1 = 100,\ R_2 = 200,\ R_3 = 300,\ \epsilon_1 = 3,\ \epsilon_2 = 4\,,$$



$$Ax = b$$

$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & -R_2 & 0 \\ o & R_2 & -R_3 \end{bmatrix} \times \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} 0 \\ -e_1 \\ e_2 + e_2 \end{bmatrix}$$

https://es.wikipedia.org/wiki/Leyes_de_Kirchhoff

Aplicaciones: resolver sistemas de ecuaciones

```python
import numpy as np

A = np.array([[1, -1, -1],[-100, -200, 0], [0, 200, -300]])
b = np.array([0, -3, 3 + 4])
x = np.linalg.solve(A, b)
print(x)
```

```
[ 0.00090909  0.01454545 -0.01363636]
```

Dadas las magnitudes:

$$R_1 = 100,\ R_2 = 200,\ R_3 = 300,\ \epsilon_1 = 3,\ \epsilon_2 = 4,$$

la solución definitiva sería:

$$\begin{cases} i_1 = \frac{1}{1100} \\ i_2 = \frac{4}{275} \\ i_3 = -\frac{3}{220} \end{cases}$$

```
>>> 1/1100
0.0009090909090909091
>>> 4/275
0.014545454545454545
>>> -3/220
-0.013636363636363636
```