Repaso Incendio Forestal

ExactasPrograma

Facultad de Ciencias Exactas y Naturales, UBA

Verano 2023

```
def suceso_aleatorio(probabilidad):
    numero = random.randint()
    if numero<probabilidad:
        return True
    else:
        return False
```

O, más corto:

```
def suceso_aleatorio(probabilidad):
    numero = random.ranint()
    return numero<probabilidad:
```

O más corto:

```
def suceso_aleatorio(probabilidad):
    return random.randint()<probabilidad
```

Crear un bosque vacío:

```
def crear_bosque(n):
    bosque_vacio = [0]*n
    return bosque_vacio
```

Brotes aleatorios: ¿Está bien este código? ¿Qué es lo que hace?

```
def brotes(bosque,p):
    bosque_vacio = [0]*n
    for i in range(len(bosque_vacio)):
        tiro_un_numero = random.randint()
        if tiro_un_numero < p:
            bosque_vacio[i] = 1
        else:
            bosque_vacio[i] = 0
    return bosque_vacio
```

¿Cómo lo podemos arreglar?

Utilizando el *parámetro* bosque y aprovechando la función `suceso_aleatorio`:

```
def brotes(bosque,p):
    for i in range(len(bosque)):
        if suceso_aleatorio(p):
            bosque[i]=1
    return bosque
```

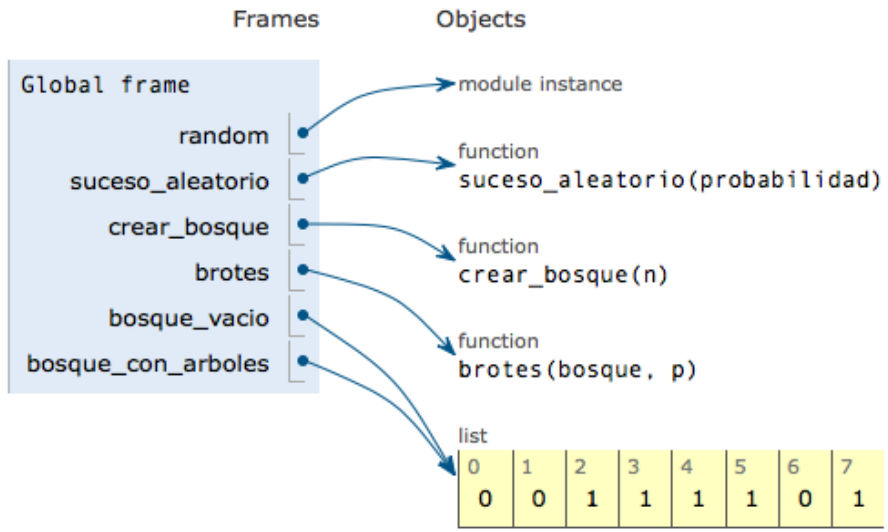
¿Hace falta el return?

Veamoslo en PythonTutor:

Python 3.6

```
1  import random
2
3  def suceso_aleatorio(probabilidad):
4      return random.random()<probabilidad
5
6  def crear_bosque(n):
7      bosque_vacio = [0]*n
8      return bosque_vacio
9
10 def brotes(bosque,p):
11     for i in range(len(bosque)):
12         if suceso_aleatorio(p):
13             bosque[i]=1
14     return bosque
15
16 bosque_vacio = crear_bosque(8)
17 bosque_con_arboles = brotes(bosque_vacio,0.6)
```

Edit this code



Un ciclo anual de la forma:

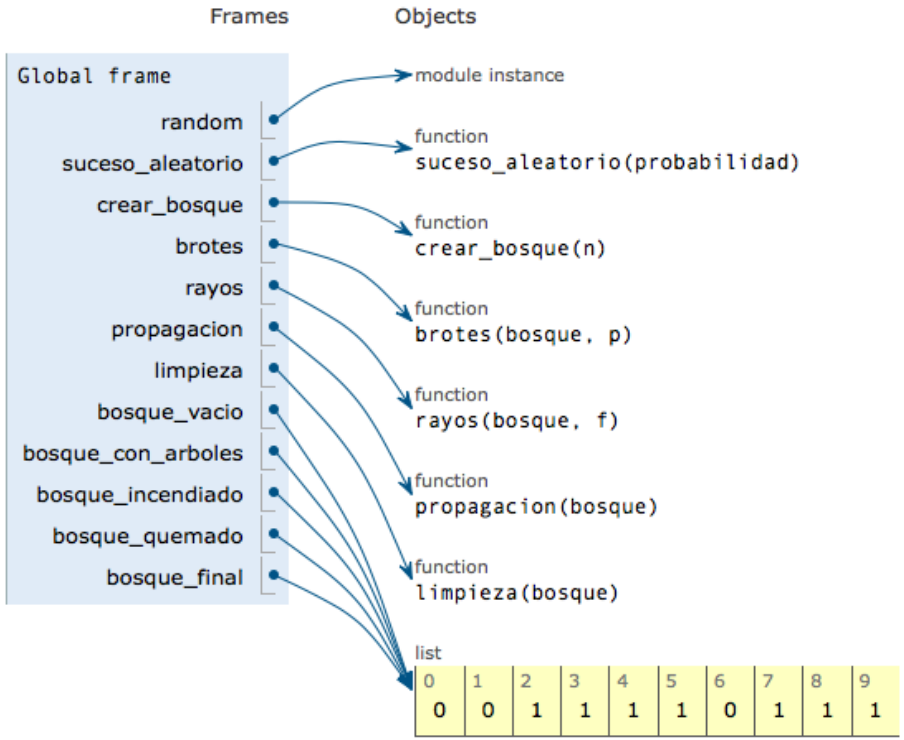
```
bosque_vacio = crear_bosque(10)
bosque_con_arboles = brotes(bosque_vacio,0.6)
bosque_incendiado = rayos(bosque_con_arboles,0.2)
bosque_quemado = propagar(bosque_incendiado)
bosque_final = limpieza(bosque_quemado)
```

... da como resultado:

Python 3.6

```
39 bosque_vacio = crear_bosque(10)
40 bosque_con_arboles = brotes(bosque_vacio,0.6)
41 bosque_incendiado = rayos(bosque_con_arboles,0.2)
42 bosque_quemado = propagacion(bosque_incendiado)
43 bosque_final = limpieza(bosque_quemado)
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
```

Edit this code



Es más conveniente usar funciones que no devuelvan nada (como `random.shuffle(lista)`):

```
def brotes(bosque,p):
    for i in range(len(bosque)):
        if suceso_aleatorio(p):
            bosque[i]=1


def ciclo_anual(bosque,p,f):
        brotes(bosque,p)
        rayos(bosque,f)
        propagar(bosque)
        limpieza(bosque)
```

La evolución completa puede implementarse así:

```
def evolucion(n,p,f,nrep):
    sobreviven = []
        bosque = crear_bosque(n)
        for i in range(nrep):
                ciclo_anual(bosque,p,f)
                arboles = contar(bosque,1)
                sobreviven.append(arboles)
        return np.mean(sobreviven)
```

Por último un recordatorio de cómo graficar:

```
import numpy as np
import matplotlib.pyplot as plt
p = 0
probs = []
cantidad = []
paso = 1
while p<=100
        probs.append(p)
        sobrevivientes = evolucion(100,p,2,1000)
        cantidad.append(sobrevivientes)
        p = p + paso
plt.plot(probs,cantidad,'.')
plt.title('Supervivencia en funci\'on de la probabilidad de brote')
plt.xlabel('p')
plt.ylabel('arboles')
plt.show()
```