```
"""
Utils para Montecarlo
(c) Carlos Martinez, abril 2022
"""

import random
import time
import math
from scipy.stats import norm

class timeit:
    """
    Medir tiempos
    """
    def __init__(self):
        self._t0 = time.perf_counter()
        self._laps = []

    def lap(self):
        t1 = time.perf_counter()
        _lap = t1 - self._t0
        self._t0 = t1
        self._laps.append(_lap)
        return _lap

    def reset(self):
        self._t0 = time.perf_counter()
# end class timeit

def sortearPuntoRN(dim=2):
    """
    Seortea un punto en R^N dentro del hiper-cubo [0,1]^N
    """
    punto = []
    for n in range(0, dim):
        punto.append(random.uniform(0.0, 1.0))
    # end for
```

```
    return punto
# end fun sortearPuntoRN

def sortearPuntoRNRangos(dim, rangos):
    """
    Seortea un punto en R^N dentro del hiper-cubo [0,1]^N pero contemplando rangos
    por dimensión.

    Parámetros:
    dim: dimensión del vector
    rangos: vector de pares, [ (x0, x1), ... ]
    """
    punto = []
    for n in range(0, dim):
        punto.append(random.uniform(rangos[n][0], rangos[n][1]))
    # end for

    return punto
# end fun sortearPuntoRN

## Estimar numero de replicaciones para obtener ciertos errores y varianzas
def npuntoN(delta_, epsilon_, estimV_, n_):
    """
    npuntoN: devuelve el numero de muestra estimado para una cierta variana y cierto
    error
    """
    x = (norm.ppf(1-delta_/2)**2)*estimV_*n_/(epsilon_**2)
    return math.ceil(x)
## end def


random.seed()

if __name__ == "__main__":
    print("Es una biblioteca, no es para correr directamente")
```