Programação em Python

Geração de números pseudo-aleatórios

2023

Departamento de Ciência de Computadores

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Conteúdo

Geração de números pseudo-aleatórios

geração de números pseudo-aleatórios

Por vezes necessitamos de simular acontecimentos aleatórios no computador:

- simular o lançamento de um dado;
- baralhar uma pilha de cartas;
- escolher uma posição no ecrã para uma nave espacial;
- mais genericamente: simular eventos imprevistos.

Podemos fazer tudo isto usando geradores de números pseudo-aleatórios.

Módulo random

Para usar geradores pseudo-aleatórios devemos importar o módulo `random`.

```
>>> import random
```

Módulo random (cont.)

A função `random()` gera um número pseudo-aleatório no intervalo $[0, 1)$.

```
>>> random.random()
0.3904112430918001
>>> random.random()
0.4508590778674775
>>> random.random()
0.6126532811362886
```

NB: o limite superior nunca é atingido!

Módulo random (cont.)

Podemos multiplicar o resultado para obter números noutro intervalo.

Por exemplo: gerar um inteiro entre 1 e 1000.

```
>>> 1 + int(1000*random.random())
731
>>> 1 + int(1000*random.random())
171
>>> 1 + int(1000*random.random())
909
```

O módulo `random` inclui funções específicas para simplificar este processo.

Módulo random (cont.)

A função `randint(`*a*`,` *b*`)` gera um inteiro pseudo-aleatório entre *a* e *b* inclusivé.

```
>>> random.randint(1,1000)
970
>>> random.randint(1,1000)
83
>>> random.randint(1,1000)
897
```

Exemplo

Simular dez lançamentos de um dado de 6 faces.

```
import random
for i in range(10):
    d = random.randint(1,6)
    print(d)
```

Exemplo de execução:

```
5
2
5
2
3
```

```
2
1
5
3
5
```

Colecionar estatísticas

Vamos simular 1000 lançamentos de um dado e colecionar algumas estatísticas:

- valor máximo;
- valor mínimo;
- valor médio.

Valor máximo

- Guardamos o *maior valor* obtido numa variável `dmax`
- Efetuamos um lançamento fora do ciclo para inicializar a variável `dmax`

```
import random
d = random.randint(1,6)
dmax = d
for i in range(1000):
    d = random.randint(1,6)
    dmax = max(dmax, d)
print ("Valor máximo= ", dmax)
```

O resultado dá sempre 6.

Valor mínimo

Análogo ao anterior: guardamos o *menor valor* obtido.

```
import random
d = random.randint(1,6)
dmin = d
for i in range(1000):
    d = random.randint(1,6)
    dmin = min(dmin, d)
print ("Valor mínimo= ", dmin)
```

O resultado dá sempre 1.

Valor médio

- Acumulamos a soma de todos os valores obtidos
- Dividimos pelo número de lançamentos

```
import random
s = 0
for i in range(1000):
    d = random.randint(1,6)
    s = s + d
print ("Valor médio= ", s/1000)
```

O resultado dá aproximadamente 3.5. Porquê?

Distribuição uniforme

Random e *randint* geram números pseudo-aleatórios com distribuição uniforme.

Com n lançamentos cada face sai aproximadamente $n/6$ vezes; logo:

$$\begin{aligned} s &\approx \frac{n}{6} \times 1 + \frac{n}{6} \times 2 + \frac{n}{6} \times 3 + \frac{n}{6} \times 4 + \frac{n}{6} \times 5 + \frac{n}{6} \times 6 \\ &= \frac{n}{6} \times (1 + 2 + 3 + 4 + 5 + 6) \\ &= \frac{n}{6} \times 21 \\ &= \frac{7n}{2} \end{aligned}$$

Logo $s/n \approx 7/2 = 3.5$.

Repetibilidade

Os geradores pseudo-aleatórios são algoritmos determinísticos—obtemos a mesma sequência se começarmos com a mesma "semente".

Podemos inicializar a "semente" usando a função `seed(s)`.

Isto permite repetibilidade das computações com números pseudo-aleatórios.

Repetibilidade (cont.)

Exemplo: o programa

```
import random
random.seed(0)
for i in range(10):
    print(random.randint(1,6))
```

produz sempre os seguintes números:

```
4
4
1
3
5
```

```
4
4
3
4
3
```

Estimar probabilidades

Podemos usar o módulo `random` para estimar probabilidades calculando frequência de ocorrências.

$$\text{Frequência} = \frac{\text{Número de casos favoráveis}}{\text{Número de experiências}}$$

Exemplo
Estimar a probabilidade do lançamento de dois dados dar soma 7.

Estimar probabilidades (cont.)

```
# Programa dados.py
# simular o lançamento de dois dados
# e contar quantas vezes somou 7
import random

numexpr = 10000 # número de experiências a simular
soma7 = 0 # contador de acontecimentos favoráveis

for i in range(numexpr):
    # lançar dois dados
    d1 = random.randint(1,6)
    d2 = random.randint(1,6)
    if d1+d2==7:
        soma7 = soma7 + 1
print("Frequência = ", soma7/numexpr)
```

Estimar probabilidades (cont.)

Obtemos valores ligeiramente diferentes em sucessivas execuções:

```
$ python3 dados.py
Frequência =  0.1646
$ python3 dados.py
Frequência =  0.166
$ python3 dados.py
Frequência =  0.1721
```

Conclusão: entre 16–17% dos lançamentos somaram 7.

Jogo Hi-Lo

Jogo *Hi-Lo*

- O computador escolhe um inteiro aleatório entre 1 e 1000
- O jogador humano tenta adivinhar
- Para cada tentativa, o computador diz se é maior, menor ou se acertou
- A pontuaçao do jogador é o número de tentativas efetuadas

Jogo *Hi-Lo* (cont.)

```
import random
number = random.randint(1,1000)
guess = 0
tentativas = 0

while guess != number:
    tentativas = tentativas + 1
    guess = int(input("Escreva um número 1-1000: "))
    if guess > number:
        print(guess, " é demasiado alto.")
    elif guess < number:
        print(guess, " é demasiado baixo.")

print(guess, " é correto.")
print(tentativas, " tentativas")
```

Jogo *Hi-Lo* (cont.)

Algumas questões:

1. Qual é a melhor estratégia para o jogador humano?
2. Qual é o menor número de tentativas no pior caso?

Passeio aleatório

Passeio aleatório

Vamos usar os módulos *turtle* e *random* para simular um passeio aleatório da tartaruga:

1. avançar uma distância fixa (passo);
2. rodar um ângulo aleatório;
3. repetir um número fixo de passos.

Primeira versão

```
import turtle
import random

def passeio(n,a):
    '''Desenhar um passeio aleatório.
     n : número de passos; a : angulo máximo.'''
    step = 10     # medida de cada passo
    for i in range(n):
        turtle.forward(step)
        angle = random.randint(-a,a)
        turtle.left(angle)
```

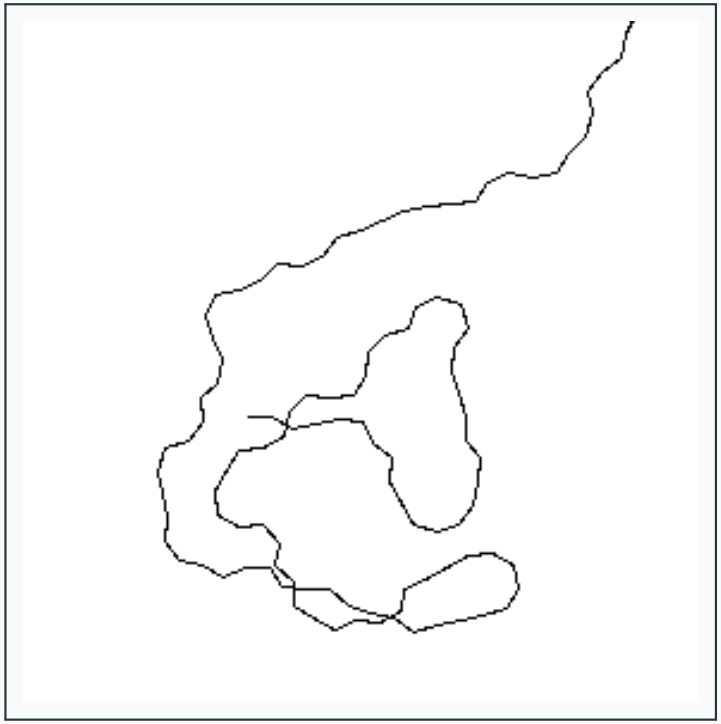
Primeira versão (cont.)

```
>>> passeio(100,30)

>>> passeio(200,60)

>>> passeio(200,120)
```

Muito frequentemente a tartaruga sai fora da janela!



Limites da janela

Queremos garantir que a tartaruga fica dentro da janela; para isso vamos usar algumas funções extra do módulo *turtle*:

- **`xcor()`** obter a coordenada horizontal da tartaruga;
- **`ycor()`** obter a coordenada vertical da tartaruga;

`window_width()` obter a largura da janela;

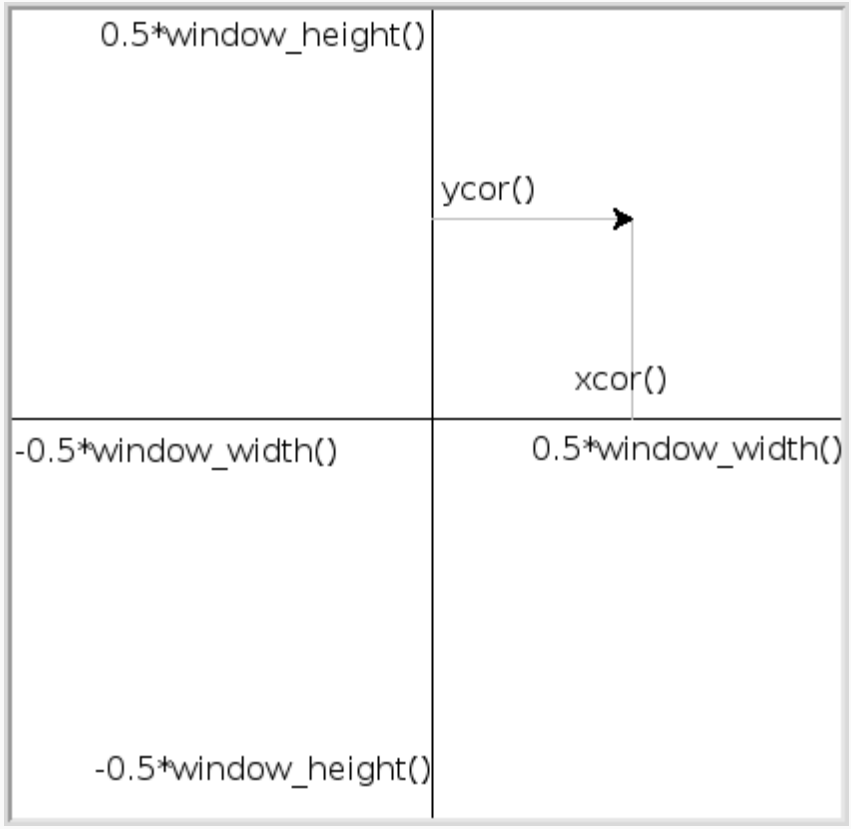
`window_height()` obter a altura da janela;

`setheading(`α**`)`** definir a orientação da tartaruga;

`towards(`*x*`,`*y*`)`** calcular a orientação para um ponto.

Verificar limites

Testar se a tartaruga está visível (dentro da janela):



Verificar limites (cont.)

```
def visível():
    "Verificar se a tartaruga está dentro da janela."
    w = 0.5*turtle.window_width()    # dimensões da janela
    h = 0.5*turtle.window_height()
    x = turtle.xcor()                # posição atual
    y = turtle.ycor()
    return (x<=w and x>=-w and y<=h and y>=-h)
```

Segunda versão

```
import turtle
import random

def passeio(n,a):
    '''Desenhar um passeio aleatório.
     n : número de passos; a : angulo máximo.'''
    step = 10     # medida de cada passo
    for i in range(n):
        turtle.forward(step)
        if visível():  # rodar aleatoriamente
            angle = random.randint(-a,a)
            turtle.left(angle)
        else:          # orientar para a origem
            angle = turtle.towards(0,0)
            turtle.setheading(angle)
```