O **Python random** é um módulo que faz parte da [linguagem Python](https://blog.betrybe.com/python/)e é utilizado para gerar números pseudo-aleatórios. Também podemos selecionar os elementos de uma lista de forma aleatória ou exibir o seu resultado embaralhado. Portanto, é um recurso útil para ser utilizado em vários tipos de aplicações, como no [desenvolvimento de jogos](https://blog.betrybe.com/tecnologia/como-criar-um-jogo/), em que precisamos construir alternativas diversificadas.

E não é só isso, esse também é um recurso importante quando queremos, por exemplo, construir algoritmos para realizar cálculos estatísticos, pois ele contém funções que geram resultados baseados em diferentes tipos de distribuições de probabilidades, como a gaussiana, a [distribuição de Pareto](https://blog.betrybe.com/carreira/principio-de-pareto/) e muitas outras. Para demonstrar o potencial desse recurso, preparamos este post que contém os seguintes tópicos:

**Índice**

* **01 |**[Python random: o que é e para que serve esse módulo?](https://blog.betrybe.com/python/python-random/#1)
* **02 |**[Conheça os métodos do módulo Python random](https://blog.betrybe.com/python/python-random/#2)

Continue conosco e boa leitura!

**Python random: o que é e para que serve esse módulo?**

A maioria das [linguagens de programação](https://blog.betrybe.com/linguagem-de-programacao/linguagem-alto-e-baixo-nivel/) contém módulos próprios, que são um conjunto de funções relacionadas a uma determinada característica, como funções para realizar cálculos matemáticos, para a leitura e escrita de arquivos, dentre outros. Os módulos são utilizados conforme a necessidade da [aplicação](https://blog.betrybe.com/desenvolvimento-web/aplicacoes-web/) e mediante a [importação](https://blog.betrybe.com/tecnologia/import-e-export/) desse recurso no código.

O módulo **Python** **random**, portanto, é formado por diversas funções para gerar números pseudo-aleatórios, que são dados gerados a partir de um algoritmo que utiliza um dado original (chamado semente) e produz um resultado aleatório com base nesse valor.

O termo pseudo-aleatório indica que o resultado produzido pelo algoritmo não é “aleatório de verdade”, pois o resultado sempre será igual, se a semente utilizada for a mesma.

Apesar disso, os dados pseudo-aleatórios gerados pelo módulo random são úteis em diversos tipos de aplicações que necessitam de valores aleatórios, como para [desenvolver jogos](https://blog.betrybe.com/carreira/desenvolvimento-de-jogos/), em cálculos estatísticos e em algoritmos de Machine Learning, que utilizam dados diferenciados para treinar o [aprendizado de máquina](https://blog.betrybe.com/tecnologia/aprendizado-de-maquina/).

**Conheça os métodos do módulo Python random**

Para utilizarmos as funções disponíveis no módulo random, devemos adicionar no início do código a importação do recurso por meio do comando **“import random”** e utilizamos a classe **random**, que contém um conjunto de métodos para gerar ou manipular dados aleatoriamente. Confira as principais delas a seguir.

Iniciando o gerador de números aleatórios: seed()

O método **random.seed()** é utilizado para definir o número inicial inteiro que servirá como base para a geração de números flutuantes aleatórios. Portanto, ao definirmos um valor para **seed()**, dizemos ao [compilador](https://blog.betrybe.com/tecnologia/compilacao/) que os números aleatórios serão gerados com base no valor informado. Se não informarmos nenhum valor, o número utilizado como semente será o horário atual do sistema. Veja um exemplo:

import random # necessário para utilizar o módulo random

random.seed(8)

print(random.random()) # resultado: 0.2267058593810488

print(random.random()) # resultado: 0.9622950358343828

random.seed(8)

print(random.random()) # resultado: 0.2267058593810488

print(random.random()) # resultado: 0.9622950358343828

random.seed()          # utilizou o horário do sistema como base

print(random.random()) # resultado: 0.15700150652431577

Perceba que definimos o valor de base como 8 e utilizamos duas vezes a função **random.random()** para exibirmos um número aleatório com base no valor inicial. A seguir, definimos novamente o valor inicial para 8 e **o resultado gerado foi o mesmo que no primeiro**[**conjunto de instruções**](https://blog.betrybe.com/tecnologia/pseudocodigo/)**.**

No bloco de comandos seguinte, geramos um número aleatório que utiliza o horário do sistema como base, pois não definimos o valor da função **seed()**. Portanto, será muito difícil conseguir encontrar esse resultado novamente.

Retorna o estado atual do gerador de números aleatórios: getstate()

O método **random.getstate()** é utilizado para armazenar o estado corrente da sequência de números aleatórios gerados para serem usados em outro momento pelo programa. Por isso, o ideal é armazenar o valor retornado em uma variável, pois ela será acessada pela função **random.setstate()**, de que falaremos próximo tópico.

Restaura o estado atual do gerador de números aleatórios: setstate()

A função **random.setstate()** restaura o estado dos números aleatórios gerados e armazenado em uma determinada variável pela função **random.getstate()**. Veja um [código de exemplo](https://blog.betrybe.com/desenvolvimento-web/hello-world-ola-mundo/):

import random # necessário para utilizar o módulo random

random.seed(8)

numeros = random.getstate()

print(random.sample(range(10), k=5))

#resultado: [3, 5, 6, 1, 7]

random.seed(10)

print(random.sample(range(10), k=5))

#resultado : [9, 0, 6, 3, 4]

random.setstate(numeros)

print(random.sample(range(10), k=5))

#resultado: [3, 5, 6, 1, 7]

No código acima, utilizamos a função **random.getstate()** para armazenar o estado dos números gerados randomicamente na variável números com base na função **seed()** igual a 8. A seguir, usamos a função **random.sample()**, de que falaremos mais ao final, para exibirmos 5 números gerados e que estão salvos pela função **random.getstate()**.

No próximo [bloco de códigos](https://blog.betrybe.com/tecnologia/codigo-fonte/), criamos outra sequência aleatória com a semente igual a 10 apenas para demonstrarmos como o método será capaz de recuperar o estado anterior que armazenamos na variável. Perceba que os resultados retornados na tela nesse momento correspondem a [9, 0, 6, 3, 4].

Por fim, restauramos o estado original por meio do comando **random.setstate(numeros).**Ao imprimirmos o resultado, perceba que os valores retornados são iguais aos exibidos no primeiro **print()** do código que correspondem a: [3, 5, 6, 1, 7].

Gerando um número aleatório a partir de um número de bits: getrandbits()

A função **random.getrandbits()** retorna um número aleatório inteiro de acordo com o tamanho especificado em [bits](https://blog.betrybe.com/tecnologia/computadores-quanticos/). Vale ressaltar que os valores retornados serão diferentes a cada vez que a função for executada. Além disso, não é possível utilizar o número zero, caso contrário a função retornará um [erro de execução](https://blog.betrybe.com/tecnologia/o-que-e-bug/). Veja um exemplo:

import random

print(random.getrandbits(1))

#resultado: 0

print(random.getrandbits(2))

#resultado: 2

print(random.getrandbits(8))

#resultado: 30

print(random.getrandbits(16))

#resultado: 56789

print(random.getrandbits(20))

#resultado: 337675

print(random.getrandbits(32))

#resultado: 972043261

Gerando um número decimal aleatório entre um intervalo: randrange()

O método **random.randrange()** retorna um número aleatório existente entre um determinado intervalo. A sintaxe do comando é:

random.randrange(start, stop, step)

Em que:

* **start**: indica o valor inicial, que pode ser incluído no resultado aleatório;
* **stop**: indica o ponto de parada e o valor não faz parte do resultado;
* **step**: representa um valor inteiro que será somado ao valor inicial para determinar o resultado. Se nenhum número for indicado, o valor correspondente será 1.

Veja um exemplo:

import random

print(random.randrange(10, 100, 5))

#resultado: 85

print(random.randrange(10, 100, 5))

#resultado: 20

print(random.randrange(10, 100, 5))

#resultado: 60

Perceba que os valores gerados correspondem a um múltiplo de 5, que é o número definido no parâmetro **step**. Além disso, os resultados são diferentes a cada vez que o comando é executado.

Gerando um número inteiro aleatório entre um intervalo: randint()

A função **random.randint()** retorna um número inteiro aleatório entre um determinado intervalo. A sintaxe do comando é:

random.randint(start, stop)

Em que:

* **start**: indica o valor inicial, que faz parte dos resultados possíveis;
* **stop**: representa o ponto de parada, no qual o valor indicado também pode ser selecionado como retorno.

Veja um exemplo:

import random

print(random.randint(2, 10))

#resultado: 7

print(random.randint(2, 10))

#resultado: 2

print(random.randint(2, 10))

#resultado: 10

print(random.randint(2, 10))

#resultado: 7

Perceba que os números determinados para o range (2 e 10) também foram apresentados como resultado da função **random.randint()**.

Gerando um elemento aleatório a partir de uma sequência: choice()

A função **Python random choice** retorna um elemento aleatório que pertença a uma sequência, que pode ser uma variável do tipo string, uma lista, uma [tupla](https://blog.betrybe.com/tecnologia/tuplas-em-python/) ou uma sequência numérica. Veja alguns exemplos com diferentes tipos de variáveis:

import random

lista\_nomes=['Maria', 'João', 'Pedro', 'Cláudia']

print(random.choice(lista\_nomes))

#resultado: Maria

nome = 'Ivani'

print(random.choice(nome))

#resultado: v

tupla\_nomes=('Maria', 'João', 'Pedro', 'Cláudia')

print(random.choice(tupla\_nomes))

#resultado: Cláudia

lista\_dicionario\_nomes= [{"nome": "Maria", "idade": 20}, {"nome": "Cláudia", "idade": 20}]

print(random.choice(lista\_dicionario\_nomes))

#resultado: {'nome': 'Maria', 'idade': 20}

print(random.choice(range(2,20)))

#resultado: 16

Gerando uma lista aleatória a partir de uma sequência: choices()

A função **random.choices()** permite o retorno de uma quantidade determinada de elementos. Também podemos definir a probabilidade para a repetição de cada elemento da sequência que servirá de base para a produção do resultado aleatório. A sintaxe do comando é:

random.choices(sequence, weights=None, cum\_weights=None, k=1)

No qual:

* **sequence**: é obrigatório e representa a sequência original de elementos, que pode ser uma lista, uma tupla ou um conjunto de números;
* **weights**: é opcional e indica a probabilidade relativa de repetição para cada elemento;
* **cum\_weights**: é opcional e indica a probabilidade acumulada de repetição de cada elemento;
* **k**: é opcional e indica o número de elementos retornado pela função, sendo o valor padrão igual a 1.

Veja um exemplo:

import random

lista\_nomes=['Maria', 'João', 'Pedro', 'Cláudia']

print(random.choices(lista\_nomes,weights=[2, 5, 1, 8], k=10))

#resultado: ['Maria', 'Pedro', 'João', 'João', 'Cláudia', 'Cláudia', 'Cláudia', 'João', 'João', 'Cláudia']

print(random.choices(lista\_nomes,cum\_weights=[2, 5, 1, 8], k=10))

#resultado: ['Cláudia', 'Cláudia', 'Cláudia', 'Cláudia', 'Cláudia', 'Maria', 'João', 'Cláudia', 'João', 'Cláudia']

No exemplo acima, utilizamos uma lista de nomes e exibimos o resultado aleatório com 10 elementos, tanto para a opção **weights**, que considera a probabilidade relativa, quanto para a alternativa **cum\_weights**, que faz o cálculo acumulado da probabilidade.

Randomizando uma sequência: shuffle()

A função **random.shuffle()** é utilizada para exibir o resultado de uma lista ou de uma tupla de forma embaralhada. Sua sintaxe é:

random.shuffle(sequence, function)

Em que:

* **sequence**: representa uma sequência, que pode ser uma lista ou uma tupla;
* **function**: é opcional e deve retornar um valor entre 0 e 1, que será utilizado para determinar o embaralhamento da sequência.

É importante dizer que, se o parâmetro **function** não for definido, será utilizada a função **random()** para embaralhar a lista. Já se utilizarmos uma função que retorna um valor fixo, o resultado da sequência será sempre o mesmo. Veja um exemplo:

import random

def resultado():

return 0.8

lista\_nomes=['Maria', 'João', 'Pedro', 'Cláudia']

random.shuffle(lista\_nomes, resultado)

print(lista\_nomes)

# resultado: ['Maria', 'João', 'Pedro', 'Cláudia'] (resultado fixo)

random.shuffle(lista\_nomes)

print(lista\_nomes)

# resultado: ['João', 'Cláudia', 'Maria', 'Pedro'] (resultado aleatório)

Retorna uma determinada parte de uma sequência: sample()

A função **random.sample()** retorna um determinado trecho de uma sequência. Sua sintaxe é:

random.sample(sequence, k)

No qual:

* **sequence**: representa uma sequência que pode ser uma lista, uma tupla, um range etc.;
* **k**: indica a quantidade de itens retornados.

Veja um exemplo:

import random

lista\_nomes=['Maria', 'João', 'Pedro', 'Cláudia']

print(random.sample(lista\_nomes, 2))

#resultado: ['Pedro', 'João']

print(random.sample(lista\_nomes, 2))

#resultado: ['João', 'Maria']

Retorna um número decimal entre 0 e 1: random()

A função random() retorna um número decimal aleatório entre 0 e 1.

Veja um exemplo:

import random

print(random.random())

#resultado: 0.41597137457321787

print(random.random())

#resultado: 0.7462409720122745

Gerando um número decimal aleatório entre um intervalo: uniform()

A função **random.uniform()** retorna um número decimal aleatório de acordo com os valores iniciais e finais definidos na função. Veja a sintaxe:

random.uniform(a, b)

No qual:

* **a**: é obrigatório e indica o valor inicial;
* **b**: é obrigatório e representa o valor final.

Veja um exemplo:

import random

print(random.uniform(5, 10))

#resultado: 9.289345141311909

Gerando um número decimal aleatório entre um intervalo: triangular()

A função **random.triangular()** é utilizada para retornar um número decimal aleatório entre um intervalo específico. Entretanto, ela aceita um terceiro valor, que serve para direcionar o resultado para mais próximo de um dos dois valores. Veja a sintaxe:

random.triangular(low, high, mode)

Em que:

* **low**: é opcional e indica o valor inicial;
* **high**: é opcional e representa o valor final;
* **mode**: é opcional e é utilizado para indicar o lado com maior peso no resultado.

Veja um exemplo:

import random

print(random.triangular(5, 10, 6))

#resultado: 6.053220062760362

print(random.triangular(5, 10, 6))

#resultado: 5.335742986788887

print(random.triangular(5, 10, 6))

#resultado: 6.06840605871783

Gerando um número decimal aleatório entre 0 e 1: betavariate()

A função **random.betavariate()** é utilizada em [algoritmos de cálculos estatísticos](https://blog.betrybe.com/carreira/data-science/) para obter um valor decimal aleatório entre 0 e 1 a partir dos valores **alpha** e **beta**, que devem ser maiores que zero. Sua sintaxe é:

random.betavariate|(alpha, beta)

Gerando um número decimal aleatório exponencialmente: expovariate()

A função **random.expovariate()** também é utilizada em cálculos estatísticos e retorna um número decimal de [distribuição exponencial](https://pt.wikipedia.org/wiki/Distribui%C3%A7%C3%A3o_exponencial) aleatório. Os resultados variam de 0 a infinito positivo, se o parâmetro informado for positivo. Já se o parâmetro for negativo, o resultado varia de 0 a infinito negativo. A sintaxe do comando é:

random.expovariate(lambd)

O parâmetro **lambd** representa o número 1 dividido pela média desejada, sendo que o valor não pode ser igual a zero.

Gammavariate (alpha, beta)

A função **random.gammavariate()** também é utilizada em cálculos estatísticos. Ela retorna um número decimal aleatório com base na [distribuição gamma](https://pt.wikipedia.org/wiki/Distribui%C3%A7%C3%A3o_gama). Os parâmetros **alpha** e **beta** devem ser maiores que zero.

Gauss (mu, sigma)

A função **random.gauss()**é aplicada em [algoritmos](https://blog.betrybe.com/tecnologia/algoritmo/) de teorias de probabilidade e retorna um número decimal aleatório baseado na [distribuição gaussiana](https://pt.wikipedia.org/wiki/Distribui%C3%A7%C3%A3o_normal).

Lognormvariate (mu, sigma)

A **random.lognormvariate()** é outra função do módulo random utilizado em algoritmos de teorias de probabilidade. Ela retorna um valor decimal aleatório baseado na [distribuição log-normal](https://pt.wikipedia.org/wiki/Distribui%C3%A7%C3%A3o_log-normal).

Gerando um número decimal aleatório: normalvariate()

A função **random.normalvariate()** é utilizada em algoritmos de teorias de probabilidade e retorna um número decimal aleatório baseado na distribuição normal, que também é chamada de distribuição gaussiana.

Vonmisesvariate (mu, kappa)

A função **random.vonmisesvariate()** é utilizada em algoritmos de estatística direcional e retorna um número decimal aleatório baseado na [distribuição de von Mises](https://en.wikipedia.org/wiki/Von_Mises_distribution).

Paretovariate (alpha)

Outra função utilizada em algoritmos de teorias de probabilidade é o **random.paretovariate()**, que retorna um número decimal aleatório baseado na [distribuição de Pareto](https://pt.wikipedia.org/wiki/Princ%C3%ADpio_de_Pareto).

Weibullvariate (alpha, beta)

A função **random.weibullvariate()** é utilizada em cálculos estatísticos e retorna um valor decimal aleatório baseado na [distribuição de Weibull](https://pt.wikipedia.org/wiki/Distribui%C3%A7%C3%A3o_de_Weibull).

O **módulo Python random** oferece uma série de métodos que permitem a geração de números aleatórios. Trata-se de um recurso muito útil, pois podemos [desenvolver](https://blog.betrybe.com/carreira/desenvolvedor-de-software/) desde códigos simples para retornar um número qualquer, até aplicações complexas com algoritmos que realizam cálculos estatísticos e de teorias da probabilidade.

Gostou do nosso conteúdo sobre o método Python random? Então, confira nosso post sobre [como programar, com o passo a passo imperdível para iniciantes](https://blog.betrybe.com/carreira/como-programar/)!