**Генерация случайных чисел**

**(модуль random)**

*«Генерация случайных чисел слишком важна, чтобы оставлять её на волю случая»*

—  Роберт Кавью

Python порождает случайные числа на основе формулы, так что они не на самом деле случайные, а, как говорят, псевдослучайные [[1]](https://ps.readthedocs.io/ru/latest/random.html#id3). Этот способ удобен для большинства приложений (кроме онлайновых казино) [[2]](https://ps.readthedocs.io/ru/latest/random.html#id5).

В этом уроке будут рассмотрены функции random(), randrange() и randint() из модуля random. Обратите внимание, что модуль random содержит одноименную функцию random(). Так бывает.

Чтобы обращаться к функциям, надо импортировать модуль random:

>>> import random

Или импортировать отдельные функции из него:

>>> from random import random, randrange, randint

Функции randint() и randrange() генерируют псевдослучайные целые числа. Первая из них наиболее простая и всегда принимает только два аргумента – пределы целочисленного диапазона, из которого выбирается любое число:

>>> random.randint(0, 10)

6

или (если импортировались отдельные функции):

>>> randint(100, 200)

110

В случае randint() обе границы включаются в диапазон,

Числа могут быть отрицательными:

>>> random.randint(-100, 10)

-83

Но первое число всегда должно быть меньше или, по крайней мере, равно второму.

Функция randrange() сложнее. Она может принимать один аргумент, два или даже три. Если указан только один, то она возвращает случайное число от 0 до указанного аргумента. Причем сам аргумент в диапазон не входит.

>>> random.randrange(10)

4

Если в randrange() передается два аргумента, то она работает аналогично randint() за одним исключением. Верхняя граница не входит в диапазон, т. е. [a; b).

>>> random.randrange(1, 2)

1

Здесь результатом второго вызова всегда будет число 1.

Если в randrange() передается три аргумента, то первые два – это границы диапазона, как в случае с двумя аргументами, а третий – так называемый шаг. Если, например, функция вызывается как randrange(10, 20, 3), то "случайное" число будет выбираться из чисел 10, 13, 16, 19:

>>> random.randrange(10, 20, 3)

13

## Функция random() – "случайные" вещественные числа

Чтобы получить случайное вещественное число, или, как говорят, число с плавающей точкой, следует использовать функцию random() из одноименного модуля **random** языка Python. Она не принимает никаких аргументов и возвращает число от 0 до 1, не включая 1:

>>> random.random()

0.17855729241927576

Результат содержит много знаков после запятой. Чтобы его округлить, можно воспользоваться встроенной в Python функцией round():

0.84

>>> round(random.random(), 3)

0.629

Чтобы получать случайные вещественные числа в иных пределах, отличных от [0; 1), прибегают к математическим приемам. Так если умножить полученное из random() число на любое целое, то получится вещественное в диапазоне от 0 до этого целого, не включая его:

>>> random.random() \* 10

2.510618091637596

>>> random.random() \* 10

6.977540211221759

## Выбрать один элемент случайным образом из списка, строки, кортежа или range().

#### **Синтаксис:**

**import** **random**

elem = random.choice(seq)

#### **Описание:**

Функция [random.choice()](https://docs-python.ru/standart-library/modul-random-python/funktsija-random-choice/" \o "Функция choice() модуля  в Python, выбирает случайный элемент.) модуля [random](https://docs-python.ru/standart-library/modul-random-python/" \o "Модуль random, случайные числа в Python.) возвращает один случайный элемент из непустой последовательности seq

Последовательность seq может быть неизменяемой ([кортеж](https://docs-python.ru/tutorial/osnovnye-vstroennye-tipy-python/tip-dannyh-tuple-kortezh/), [строка](https://docs-python.ru/tutorial/osnovnye-vstroennye-tipy-python/tip-dannyh-str-tekstovye-stroki/) и т.д.) или изменяемая ([список](https://docs-python.ru/tutorial/osnovnye-vstroennye-tipy-python/tip-dannyh-list-spisok/)). Функция random.choice() поддерживает диапазоны [range()](https://docs-python.ru/tutorial/vstroennye-funktsii-interpretatora-python/klass-range/" \o "Класс range() в Python, генерирует арифметические последовательности.).

Если последовательность seq пуста, то поднимается [исключение IndexError](https://docs-python.ru/tutorial/vstroennye-iskljuchenija-interpretator-python/vstroennye-iskljuchenija/).

Неупорядоченные последовательности [set() и frozenset()](https://docs-python.ru/tutorial/osnovnye-vstroennye-tipy-python/tip-dannyh-set-frozenset-mnozhestva/" \o "Множество set и frozenset в Python.) не поддерживаются. Множество set()/frozenset() сначала должно быть преобразовано в список или кортеж.

## Практическая работа

1. Используя функцию randrange() получите псевдослучайное четное число в пределах от 6 до 12. Также получите число кратное пяти в пределах от 5 до 100.
2. Напишите программу, которая запрашивает у пользователя границы диапазона и какое (целое или вещественное) число он хочет получить. Выводит на экран подходящее случайное число.
3. Хороший пароль должен быть произвольным и состоять минимум из 6 символов, в нём должны быть цифры, строчные и прописные буквы. Приготовить такой пароль можно по следующему рецепту:

**Функция map**

Теперь, когда мы с вами рассмотрели базовые типы данных языка Python, отметим несколько весьма полезных и часто используемых функций. И начнем с функции **map**. Она позволяет преобразовывать элементы итерируемого объекта в соответствии с некоторой указанной функцией:

map(func, \*iterables)

Давайте, я поясню работу этой функции на таком простом примере. Предположим, что у нас есть упорядоченный список и мы хотим получить список из квадратов этих чисел. Для этого создадим функцию возведения числа в квадрат: и вызовем функцию map:

lst = [1,-2,3,-4,-5]

**def** sq(x):

**return** x\*\*2

b = map(sq, lst)

a = list(b)

**print**(a)

Обратите внимание, мы передаем ссылку на функцию, то есть, записываем ее имя без круглых скобок. В результате переменная b будет ссылаться на итератор map и мы из предыдущего занятия знаем, что для перебора всех значений можно воспользоваться функцией next:

**print**(next(b))

**print**(next(b))

**print**(next(b))

Или же сформировать новый список с помощью функции list:

То есть, функция map к каждому элементу списка lst применяет функцию sq и возвращает объект-генератор для последовательного выбора новых, преобразованных значений.

Причем, во всех случаях функция sq или какая-либо другая должна принимать только один аргумент. А вот возвращать она может несколько аргументов,

**def** sq(x):

**return** x, x\*\*2

В результате:

lst = [1,-2,3,-4,-5]

b = map(sq, lst)

a = list(b)

**print**(a)

получим список кортежей:

[(1, 1), (-2, 4), (3, 9), (-4, 16), (-5, 25)]

Подобные преобразования можно выполнять с любыми типами данных, например, строками:

lst = ["Москва", "Рязань", "Смоленск", "Тверь", "Томск"]

b = map(len, lst)

a = list(b)

**print**(a)

На выходе получим список с длинами соответствующих строк:

[6, 6, 8, 5, 5]

Если нужно применить встроенные строковые методы, то это делается так:

b = map(str.upper, lst)

И в результате получаем следующий список:

['МОСКВА', 'РЯЗАНЬ', 'СМОЛЕНСК', 'ТВЕРЬ', 'ТОМСК']

Довольно часто первым аргументом функции map указывают анонимные (лямбда) функции, например, так:

b = map(**lambda** x: x[::-1], lst)

Получим строки, записанные наоборот:

['авксоМ', 'ьназяР', 'кснеломС', 'ьревТ', 'ксмоТ']

Далее, так как функция map вторым аргументом принимает любой итерируемый объект, то мы можем результат работы первой функции map: использовать во второй функции map:

b = map(**lambda** x: x.replace("а", "А"), lst)

c = map(sorted, b)

res1 = list(c)

**print**(res1)

Получим результат:

[['А', 'М', 'в', 'к', 'о', 'с'], ['А', 'Р', 'з', 'н', 'ь', 'я'], ['С', 'е', 'к', 'л', 'м', 'н', 'о', 'с'], ['Т', 'в', 'е', 'р', 'ь'], ['Т', 'к', 'м', 'о', 'с']]

То есть, строки коллекции b: ['МосквА', 'РязАнь', 'Смоленск', 'Тверь', 'Томск']

были преобразованы в список символов, отсортированных по возрастанию и на выходе мы получили общий список с этими вложенными списками.

Очень часто в программах на Python функцию map используют для ввода с клавиатуры нескольких чисел через какой-либо разделитель, например, пробел. Если мы все запишем вот в таком виде:

a = int(input())

и будем вводить целые числа через пробел, то при выполнении функции int возникнет ошибка, т.к. пробел – это не цифровой символ.

И здесь нам на помощь приходит функция map. И мы получим генератор для получения введенных чисел. Но это все удобнее преобразовать сразу к списку:

a = list(map(int, input().split()))

И теперь, при вводе любого количества чисел через пробел, будем получать упорядоченный список, состоящий из чисел. Это бывает очень удобно.

**Функция filter**

Следующая аналогичная функция – это filter. Само ее название говорит, что она возвращает элементы, для которых, переданная ей функция возвращает True:

Предположим, у нас есть список из которого нужно выбрать все нечетные значения. Для этого определим функцию: И далее, вызов функции filter:

a=[1,2,3,4,5,6,7,8,9,10]

**def** odd(x):

**return** x%2

b = list(filter(odd, a))

**print**(b)

На выходе получаем итератор. Или же преобразовать итератор в список:

Конечно, в качестве функции здесь можно указывать лямбда-функцию и в нашем случае ее можно записать так:

b = list(filter(**lambda** x: x%2, a))

И это бывает гораздо удобнее, чем объявлять новую функцию.

Функцию filter можно применять с любыми типами данных, например, строками. Пусть у нас имеется вот такой кортеж:

lst = ("Москва", "Рязань1", "Смоленск", "Тверь2", "Томск")

b = filter(str.isalpha, lst)

**for** x **in** b:

**print**(x, end=" ")

и мы вызываем метод строк isalpha, который возвращает True, если в строке только буквенные символы. В результате в консоли увидим:

Москва Смоленск Тверь Томск

**Функция reduce**

И map, и filter работали с отдельными элементами независимо. Но ведь встречаются и циклы, которые агрегируют результат — формируют результирующее значение (одно), комбинируя элементы между собой с использованием аргумента-аккумулятора.

Типичным примером агрегации может быть сумма всех элементов списка. Или, скажем, произведение. Представим, что нам нужно сложить элементы списка [1, 2, 3, 4, 5]. С точки зрения математики сумма 1 + 2 + 3 + 4 + 5 может быть выражена как (((((0 + 1) + 2) + 3) + 4) + 5).

Ноль здесь — тот самый аккумулятор (его начальное состояние). Он не добавляет к сумме ничего, но может служить отправной точкой. А ещё — будет результатом, если входной список пуст.

С помощью цикла мы бы суммировали так:

acc **=** 0

**for** item **in** items:

acc **=** acc **+** item

А умножали бы так:

acc **=** 1

**for** item **in** items:

acc **=** acc **\*** item

Циклы отличаются только начальным значением аккумулятора (0 и 1) и операцией, которая комбинирует элемент и аккумулятор (+ и \*).

В в математике такая функция называется левая свёртка (left fold). И имя "свёртка" вполне себе говорящее — применяя эту функцию, мы сворачиваем список в одно значение!

А левая наша свёртка потому, что "схлопывать" элементы с аккумулятором мы начинаем слева. Существует ещё и правая свёртка (right fold), но в виде встроенной функции в Python она не представлена. Правая свёртка для суммы выглядит так: (1 + (2 + (3 + (4 + (5 + 0)))))

В большинстве случаев обе свёртки дают одинаковый результат, если применяемая операция ассоциативна (т.е. позволяет расставлять скобки произвольно — как в случае наших сумм). Но через цикл итерации элементов проще реализовать именно левую, поэтому она и используется чаще.

**Функция zip**

Следующая весьма полезная функция позволяет объединять между собой соответствующие элементы упорядоченных коллекций. Например, у нас имеется два списка: И вызывая для них функцию zip: Получим итератор, который возвращает следующую коллекцию:

a = [1,2,3,4]

b = [5,6,7,8]

it = zip(a, b)

**print**(it)

**print**( list(it ) )

и мы увидим:

[(1, 5), (2, 6), (3, 7), (4, 8)]

То есть, у нас были объединены в кортеж соответствующие элементы этих двух списков.

Давайте теперь добавим еще один итерируемый объект – строку:

c = "abracadabra"

И вызовем функцию zip для всех этих трех объектов:

it = zip(a, b, c)

**print**( list(it ) )

В результате получим коллекцию:

[(1, 5, 'a'), (2, 6, 'b'), (3, 7, 'r'), (4, 8, 'a')]

Смотрите, мы здесь имеем всего четыре кортежа, в каждом из которых по три элемента. То есть, все оставшиеся символы строки "abracadabra" были просто отброшены. Получается, что функция zip формирует выходной список, длина которого равна длине наименьшей из указанных коллекций.

Вот в этом и заключается удобство этой функции: она позволяет автоматически объединить несколько списков в наборы кортежей из соответствующих значений.

## Практическая работа

На вход поступает строка из целых чисел, записанных через пробел. С помощью функции map преобразовать эту строку в список целых чисел, взятых по модулю. Сформируйте именно список lst из таких чисел. Отобразите его на экране в виде набора чисел, идущих через пробел.

**Sample Input:** -5 6 8 11 -10 0

**Sample Output:** 5 6 8 11 10 0

#---------------------------------------------------

Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию map, которая бы возвращала названия городов только длиной более 5 символов. Вместо остальных названий - строку с дефисом ("-"). Сформировать список из полученных значений и отобразить его на экране в одну строчку через пробел.

**Sample Input:**

Москва Уфа Вологда Тула Владивосток Хабаровск

**Sample Output:**

Москва - Вологда - Владивосток Хабаровск

#---------------------------------------------------

Вводится список целых чисел в одну строчку через пробел. Необходимо оставить в нем только двузначные числа. Реализовать программу с использованием функции filter. Результат отобразить на экране в виде последовательности оставшихся чисел в одну строчку через пробел.

**Sample Input:** 8 11 0 -23 140 1

**Sample Output:** 11 -23

#---------------------------------------------------

Вводятся два списка целых чисел. Необходимо попарно перебрать их элементы и перемножить между собой При реализации программы используйте функции zip и map.

**Sample Input:**

-7 8 11 -1 3

1 2 3 4 5 6 7 8 9 10

**Sample Output:**

-7 16 33