4.3. The `random` module

We often want to use **random numbers** in programs. Here are a few typical uses:

- To play a game of chance where the computer needs to throw some dice, pick a number, or flip a coin,
- To shuffle a deck of playing cards randomly,
- To randomly allow a new enemy spaceship to appear and shoot at you,
- To simulate possible rainfall when we make a computerized model for estimating the environmental impact of building a dam,
- For encrypting your banking session on the Internet.

Python provides a module `random` that helps with tasks like this. You can take a look at it in the documentation. Here are the key things we can do with it.

Save & Run 4/17/2021, 4:11:49 PM - 2 of 2 Show in CodeLens

```
1 import random
2
3 prob = random.random()
4 print(prob)
5
6 diceThrow = random.randrange(1,7)        # return an int, one of 1,2,3,4,5,6
7 print(diceThrow)
8
```

```
0.584165529301
4
```

Activity: 1 -- ActiveCode (ac13_2_1)

Press the run button a number of times. Note that the values change each time. These are random numbers.

The `randrange` function generates an integer between its lower and upper argument where the lower bound is included, but the upper bound is excluded. So, `randrange(1,7)` will include numbers from 1-6. If you omit the first parameter it is assumed to be 0 so `randrange(10)` will give you numbers from 0-9. All the values have an equal probability of occurring (i.e. the results are *uniformly* distributed).

The `random()` function returns a floating point number in the range [0.0, 1.0) — the square bracket means "closed interval on the left" and the round parenthesis means "open interval on the right". In other words, 0.0 is possible, but all returned numbers will be strictly less than 1.0. It is usual to *scale* the results after calling this method, to get them into a range suitable for your application.

In the case shown below, we've converted the result of the method call to a number in the range [0.0, 5.0). Once more, these are uniformly distributed numbers — numbers close to 0 are just as likely to occur as numbers close to 3, or numbers close to 5. If you continue to press the run button you will see random values between 0.0 and up to but not including 5.0.

Save & Run 4/17/2021, 4:12:03 PM - 2 of 2 Show in CodeLens

```
1 import random
2
3 prob = random.random()
4 result = prob * 5
5 print(result)
6
```

```
0.134830145922
```

Activity: 2 -- ActiveCode (ac13_2_2)

It is important to note that random number generators are based on a **deterministic** algorithm — repeatable and predictable. So they're called **pseudo-random** generators — they are not genuinely random. They start with a *seed* value. Each time you ask for another random number, you'll get one based on the current seed attribute, and the state of the seed (which is one of the attributes of the generator) will be updated. The good news is that each time you run your program, the seed value is likely to be different meaning that even though the random numbers are being created algorithmically, you will likely get random behavior each time you execute.

Check your understanding

modules-2-1: The correct code to generate a random number between 1 and 100 (inclusive) is:

A. prob = random.randrange(1, 101)

B. prob = random.randrange(1, 100)

C. prob = random.randrange(0, 101)

D. prob = random.randrange(0, 100)

Check me | Compare me

✔ This will generate a number between 1 and 101, but does not include 101.

Activity: 3 -- Multiple Choice (question13_2_1)

modules-2-2: One reason that lotteries don't use computers to generate random numbers is:

A. There is no computer on the stage for the drawing.

B. Because computers don't really generate random numbers, they generate pseudo-random numbers.

C. They would just generate the same numbers over and over again.

D. The computer can't tell what values were already selected, so it might generate all 5's instead of 5 unique numbers.

Check me | Compare me

✔ Computers generate random numbers using a deterministic algorithm. This means that if anyone ever found out the algorithm they could accurately predict the next value to be generated and would always win the lottery.

Activity: 4 -- Multiple Choice (question13_2_2)

You have attempted 5 of 4 activities on this page

✔ Completed. Well Done!

4.2. Modules">
dules">
4.4. Glossary">

| Back to top