Contents

Random

- Random Number Generation
  - Produce a number within some bounds following some statistical rules.
  - A true random number is a number that is **nondeterministically** selected from a set of numbers wherein each possible selection has an equal probability of occurrence.
  - Usually in computer science we contend with **pseudo-random** numbers. These are not truly nondeterministic, but an approximation of random selection based on some algorithm.
  - Since pseudo-random selections are "determined" by an algorithm, or set of rules, they are technically **deterministic**.
- Random Class in C#
  - Instantiate a random number generator and use to select numbers:

    ```
    Random rand = new Random();
    Random randB = new Random(seed_int);
    ```

  - Notice that we can create a generator with or without an argument. The argument is called a **seed** for the generator.
  - A seed tells the generator where to start its sequence. Using the same seed will always reproduce the same sequence of numbers.
  - The default constructor still has a seed value, but it is a hidden value pulled from the clock time during instantiation.
  - Time-based seeds only reset approximately every 15 milliseconds.
  - The random class is not "random enough" for cryptography.
  - For cryptographic randomness, use the RNGCryptoServiceProvider[1] class or System.Security.Cryptography.RandomNumberGenerator[2].
- Using Random

[1] https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rngcryptoserviceprovider?view=net-5.0

[2] https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.randomnumbergenerator?view=net-5.0

- Next() method returns a pseudo-random number between 0 and 2,147,483,647 (max signed `int`), inclusive.
- By default, the number is always non-negative and within that range.

```
int randomInt = rand.Next();
```

- What if we wanted to create a random number between 0 and 100?
- We could use rand.Next() and then use modulo to cut down the answer range!
- Alternatively, we could give the Next() method an `int` argument to set a ceiling.

```
int randomUpto100 = rand.Next(101);
```

- The ceiling value is **exclusive**, so remember to use one number higher than what you want to be your max number.
- We can also pass two arguments in order to set a range for the values.

```
int random50to100 = rand.Next(50,101);
```

- The ceiling value is still exclusive, but the floor is **inclusive**.
- NextDouble() returns a **normalized** value (value between 0.0 and 1.0 inclusive).
- What if we want a different range? Adjust with math!

```
double randNeg2to3 = (rand.NextDouble()*5)-2;
```

- NextBytes() method takes a `byte` array as an argument and generates a random `byte` value for each index.
- Remember, a `byte` has an unsigned value between 0 and 255 inclusive.

```
byte[] byteArray = new byte[10];
rand.NextBytes(byteArray);
```

- Creating Random Strings
  - What if we want to construct random strings made of a, b, c, and d?
  - Other techniques are available, but we can use a loop and switch!

```
Random rand = new Random();
string answer = "";
int selection = 0;
```

```
for(int i = 0; i < 10; i++)
{
    selection = rand.Next(4);
    switch(selection){
    case(0):
        answer+="a";
        break;
    case(1):
        answer+="b";
        break;
    case(2):
        answer+="c";
        break;
    default:
        answer+="d";
        break;
    }
}
```