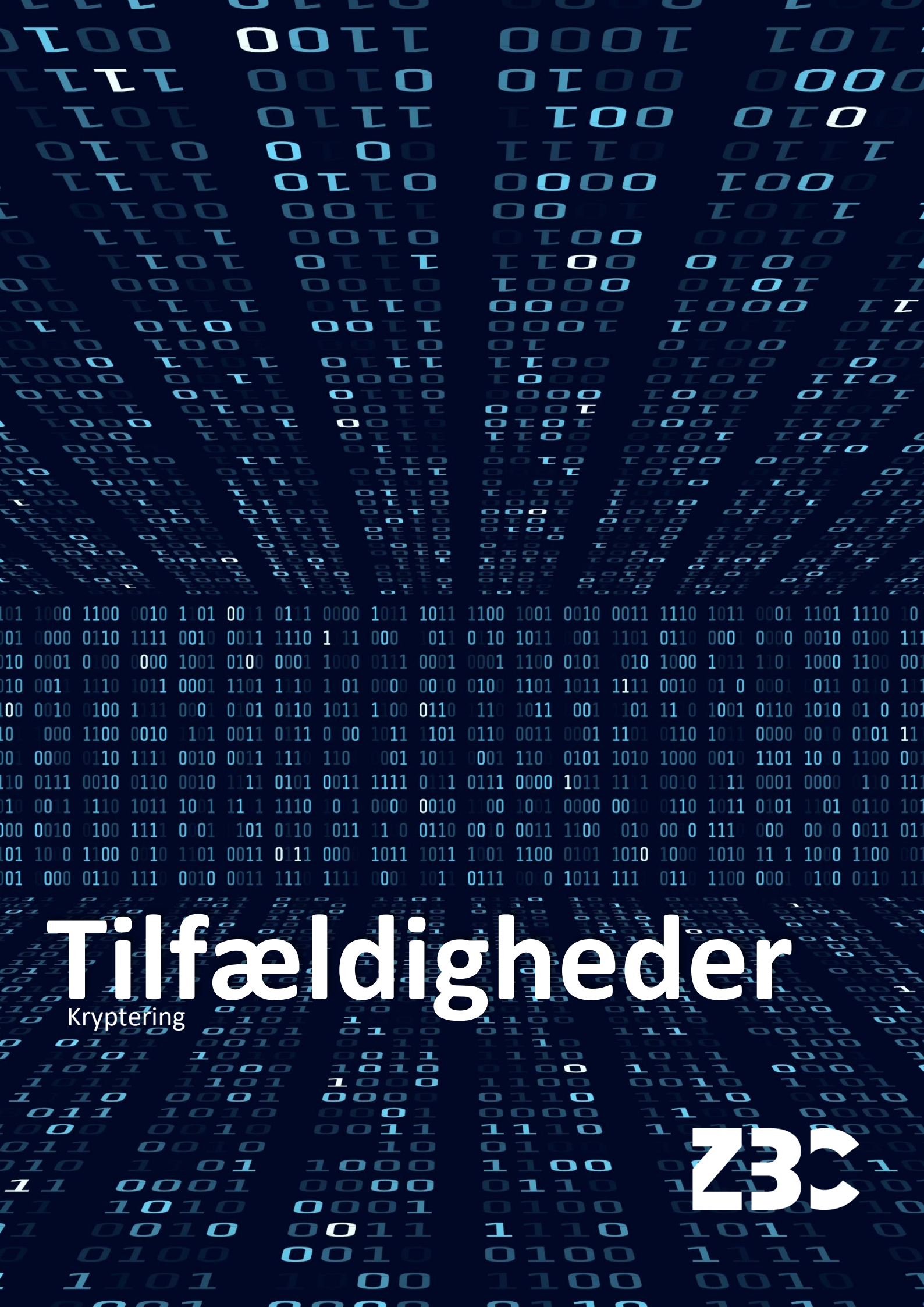

Tilfældigheder

Kryptering

Introduktion til Tilfældighed og Benchmarking

Tilfældighed er en vigtig egenskab inden for datavidenskab og softwareudvikling. Det refererer til evnen til at generere og arbejde med tilfældige eller tilsyneladende uforudsigelige værdier. Tilfældighed spiller en afgørende rolle i mange applikationer, herunder spil, kryptografi, simuleringer, dataanalyse og meget mere.

I programmering er der forskellige metoder til at generere tilfældige tal. To almindelige tilgange er **Random** og **RandomNumberGenerator.Create()**. **Random** er en pseudotilfældig talgenerator, der bruger en seed-værdi til at beregne en række tilfældige tal, som i virkeligheden ikke er rigtig tilfældige, men ser tilfældige ud. **RandomNumberGenerator.Create()**, som er en del af **System.Security.Cryptography**, er en mere sikker tilfældig talgenerator, der bruger ægte tilfældighed fra systemet, hvilket gør den mere velegnet til sikkerhedsfølsomme applikationer som kryptografi.

Benchmarking er en metode til at evaluere ydeevnen af forskellige metoder eller funktioner i programmering. Ved at køre flere gentagelser af en given opgave og måle den tid, det tager, kan vi identificere, hvilken metode der er mest effektiv, hurtig eller pålidelig i en given kontekst. Benchmarking hjælper udviklere med at optimere deres kode og træffe informerede beslutninger om valg af algoritmer, datastrukturer og metoder i deres applikationer.

Sammenfattende er tilfældighed og benchmarking to vigtige begreber inden for programmering. Forståelse af, hvordan man genererer tilfældige tal og hvordan man evaluerer ydeevnen af forskellige kodestykker, er nøglen til at skabe pålidelige, sikre og effektive softwareløsninger.

RandomNumberGenerator.Create()

På "The Moodle" er der vedhæftet et program der er navngivet Randomness lige under denne øvelse**.** I Randomness er der defineret to metoder: **TestRandomnessWithRNGCryptoServiceProvider** og **TestRandomnessWithRandom**. Begge metoder udfører tilfældighedstests ved at generere et tilfældigt array af bytes og konvertere det til et heltal ved hjælp af **BitConverter.ToInt32**. Den første metode bruger **RandomNumberGenerator.Create()**, mens den anden metode bruger **Random**.

Bemærk, at denne Randomness koden blot demonstrerer tilfældighedstests og ikke ydeevnetests. For at udføre ydeevnetests, skal du implementere en benchmarking-metode, der kører de tilsvarende metoder mange gange med store datamængder og måler den tid, det tager at udføre operationerne. Resultaterne af både tilfældighedstest og ydeevnetest bør opsamles og analyseres i et værktøj som Excel.

Øvelse: Tilfældighedstest med Random og RandomNumberGenerator
I denne øvelse vil du generere en række tilfældige tal ved hjælp af både **Random** og **RandomNumberGenerator.Create()** og udføre en tilfældighedstest for at se, hvor godt de generere tilfældige tal.

1. Opret en konsolapplikation i C#.
2. Implementer en metode, der bruger **Random** til at generere en liste med 100 tilfældige tal mellem 0 og 999
3. Implementer en metode, der bruger **RandomNumberGenerator.Create()** til at generere den samme liste med 100 tilfældige tal mellem 0 og 999
4. Afprøv begge metoder fra Main

5. Undersøg de genererede tal i outputtet og overvej følgende spørgsmål:
    - Ser tallene tilfældige ud?
    - Er der forskelle mellem tallene genereret af Random og RandomNumberGenerator?
    - Er der nogen gentagelser i tallene?
    - Diskuter dine observationer og refleksioner omkring forskellen mellem Random og RandomNumberGenerator.Create() med dine gruppe.

Denne øvelse blot er en simpel demonstration og ikke en formel tilfældighedstest. For mere avancerede tilfældighedstests, skal du bruge statistiske tests som Chi-square test, Kolmogorov-Smirnov test eller andre specialiserede tilgange.

Øvelse: Benchmarking af Random og RandomNumberGenerator

I denne øvelse vil du generere en række tilfældige tal ved hjælp af både **Random** og **RandomNumberGenerator.Create()** og udføre en benchmarking for at måle ydeevnen af de to metoder.

1. Implementer en metode, der bruger Random til at generere en liste med 1.000.000 tilfældige tal mellem 0 og 999
2. Implementer en metode, der bruger RandomNumberGenerator.Create() til at generere den samme liste med 1.000.000 tilfældige tal mellem 0 og 999
3. Implementer en metode til benchmarking, der måler den tid, det tager for hver metode at generere 1.000.000 tilfældige tal:

```
private static void RunBenchmark(Action benchmarkMethod, string methodName)
{
    long startTime = DateTime.Now.Ticks;
    benchmarkMethod();
    long endTime = DateTime.Now.Ticks;
    long elapsedTime = endTime - startTime;
    Console.WriteLine($"{methodName} tid (ticks): {elapsedTime}");
}
```

4. Afprøv benchmarking i Main-metoden
5. Undersøg resultaterne af benchmarking og overvej følgende spørgsmål:
    - Hvilken metode er hurtigst til at generere de tilfældige tal?
    - Hvad er forskellen i ydeevne mellem de to metoder?
    - Diskuter dine observationer og refleksioner omkring ydeevnen af Random og RandomNumberGenerator.Create() med din gruppe.

Bemærk, at denne øvelse giver en simpel benchmarking, men for mere præcise resultater og sammenligninger mellem forskellige datamængder og algoritmer, skal du overveje at bruge specialiserede benchmarking-værktøjer som BenchmarkDotNet.

Substitution kryptering

En ombytningskode — også kaldet en substitutionskode — udskifter bogstaver og tegn i en klartekst med andre tegn. Deraf navnet ombytning eller substitution. En ombytningskode kaldes ofte for et chiffer. Der findes rigtig mange forskellige sådanne koder, og de kan fungere på vidt forskellige måder.

Cæsar-koden, også kendt som forskydningskoden, er en kendt krypteringsteknik. Det er en ombytningskode, hvor hvert bogstav i "klarteksten" udskiftes af med et bogstav et fast antal pladser længere nede ad alfabetet. Med en forskydning på tre plaser bliver A udskiftet med D, B med E, C med F og så videre. Koden er opkaldt efter Julius Cæsar, som selv brugte den i sin korrespondance.

A-K kode

Til at kryptere og dekryptere Cærsarkoder bruger man en oversættelsestabel, hvor alfabetet står øverst og det forskudte alfabet står nedenunder. For en A-K kode ser tabellen sådan her ud:

Normal A-K skrive-tabel

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	Æ	Ø	Å
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	Æ	Ø	Å	A	B	C	D	E	F	G	H	I	J

Man krypterer sin besked ved at finde bogstavet i det øverste alfabet, og erstatte det med bogstavet nedenunder. Eksempelvis kan vi oversætte:

```
Klartekst: Vi kom, Vi så, Vi sejrede.
Kode:      Cs uyw, Cs åj, Cs åotøono.
```

Når man skal dekryptere den kodede besked, bruges tabellen omvendt. Man finder altså hvert bogstav i den nederste tabel og erstatter det med bogstavet ovenover.

W-problemet

Et problem man ofte møder, når man forsøger bryde en simpel Cæsar- eller A-K kode er, at bestemme hvilket alfabet, der er brugt. Det kan synes mærkeligt, da der jo kun findes et dansk alfabet, men der er alligevel plads til visse misforståelser.

F.eks. sker det ofte at man udelader "W" og "Q" fra alfabetet, da de sjældent bruges. Derfor skal man tit forsøge sig med flere alfabeter, hvis man prøver at gætte en løsning på koden.

A-K skrive-tabel uden W

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	X	Y	Z	Æ	Ø	Å

K	L	M	N	O	P	Q	R	S	T	U	V	X	Y	Z	Æ	Ø	Å	A	B	C	D	E	F	G	H	I	J

A-K skrive-tabel uden W og Q

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	U	V	X	Y	Z	Æ	Ø	Å
K	L	M	N	O	P	R	S	T	U	V	X	Y	Z	Æ	Ø	Å	A	B	C	D	E	F	G	H	I	J

Øvelse Encrypter

Opret en klasse "Encrypter" for at kryptere og dekryptere tekst efter reglerne i substitution kryptering.

Klassen skal have en Encrypt-metode, der modtager en streng og returnerer en anden streng. Det må gerne være en statisk metode, så du ikke behøver at oprette et objekt af typen "Encrypter".

Der skal også være en Decrypt-metode.

Din krypteringsmetode er en meget enkel metode: for at kryptere tilføjer vi 1 til hvert tegn, så "Hello" bliver "Ifmmb", og for at dekryptere trækkes 1 fra hvert tegn.