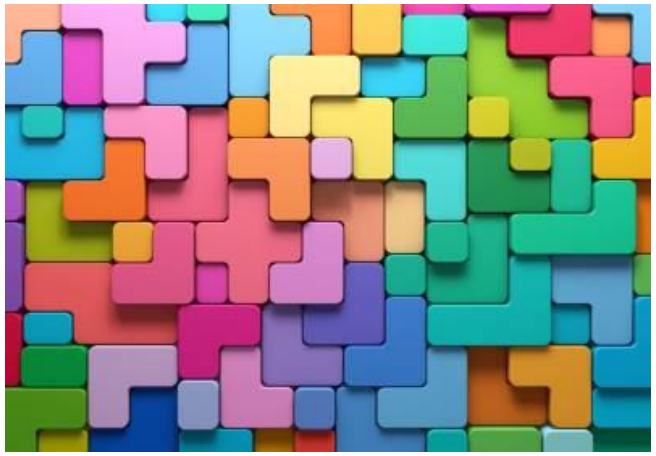
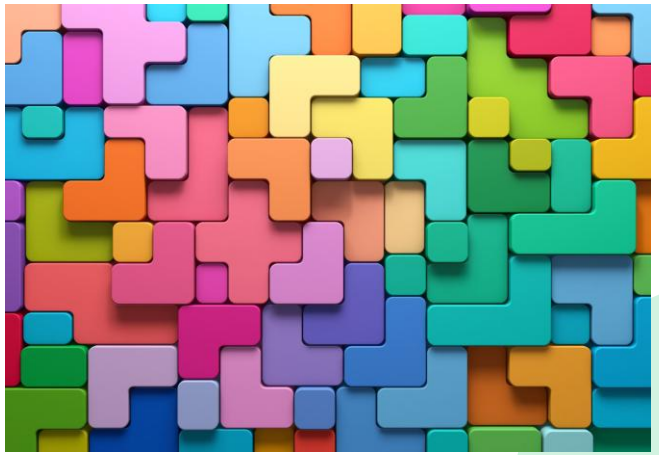
TILFELDIGHET

FORELESNING 17

MANDAG 20/10

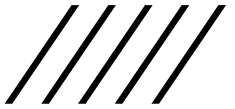




(bilder generert av bing image creator)

Læremål: Tilfeldige tall og simuleringer

- Tilfeldighet brukes i mange vitenskapelige sammenhenger
- Sannsynlighet / statistikk
- Kryptering av data i klartekst
- Likevektstilstander: La partikler bevege seg tilfeldig og se om de havner i bestemte tilstander



- Ikke gjør dette hjemme:

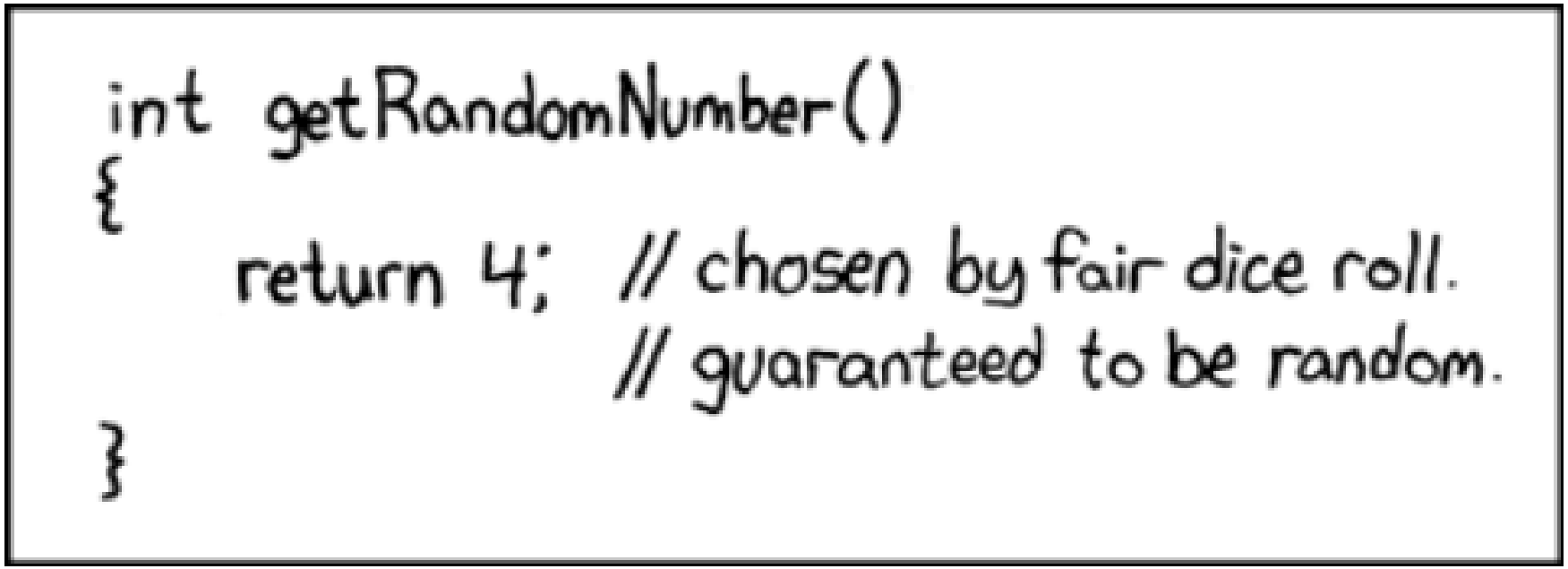
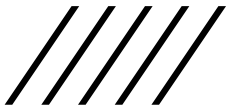


Fig. 54 Source: XKCD #221

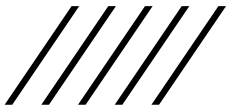
Hvor tilfeldige er tilfeldige tall?

- Egentlig ikke tilfeldige i det hele tatt – datamaskiner er deterministiske
- Hvis man gir en RNG (*random number generator*) samme start-tilstand (*seed*) vil den alltid gjenskape de samme tallene
- Derfor viktig å ikke starte med et fast seed, da vil programmet bare får de samme tallene hver gang det kjøres!
- En mulighet er å bruke antall sekunder siden midnatt på 1. januar 1970 (**time.time()** i Python) som seed, siden dette vil være forskjellig hver gang.



Pseudorandom numbers

- Selv om tallene ikke egentlig er tilfeldige, men er fast bestemt av start-tilstand (seed), så ønsker vi at de skal komme ut i en rekkefølge som er statistisk fordelt slik ekte tilfeldige tall ville vært
- Dette er faktisk nok for våre formål (vitenskapelige simuleringer)
- pRNG = pseudoRandomNumberGenerator



OBS: Viktig å unngå at fordelingen er annerledes enn ekte tilfeldige tall

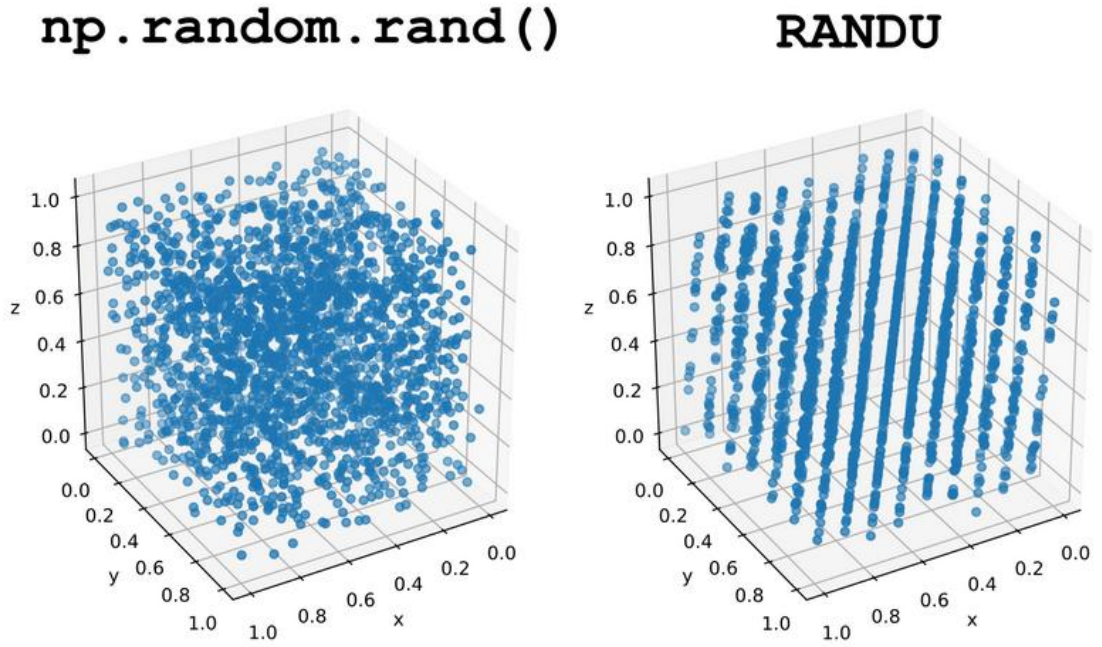
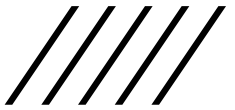


Fig. 55 (Left) For a "good" pRNG, like the one used by `np.random`, we can see no discernible patterns in the random samples, i.e., they are uncorrelated. (Right) RANDU however, fails this test, as we can see the points line up into 15 parallel planes.

At the time of RANDU's creation, not enough diligence was put into verifying its properties. And because of its simplicity and speed, it was quite popular in the 60s and 70s. Consequently, many scientific results that rely on stochastic simulations from that time are seen with skepticism.

Hva med repetisjoner?

- Hvis hvert nye tall som genereres kun er avhengig av forrige tall i rekken, vil enhver repetisjon av ett tall repetere alle tallene som kom etter det tallet
- Men i en tilfeldig tallrekke kan repetisjoner forekomme
- Enkel løsning: Hvis hvert nye tall i stedet avhenger av de siste 2 eller 3 tallene, kan vi repetere ett enkelt tall uten at det blir problemer
- (Men repeterer vi lenge nok kommer vi til utgangspunktet)



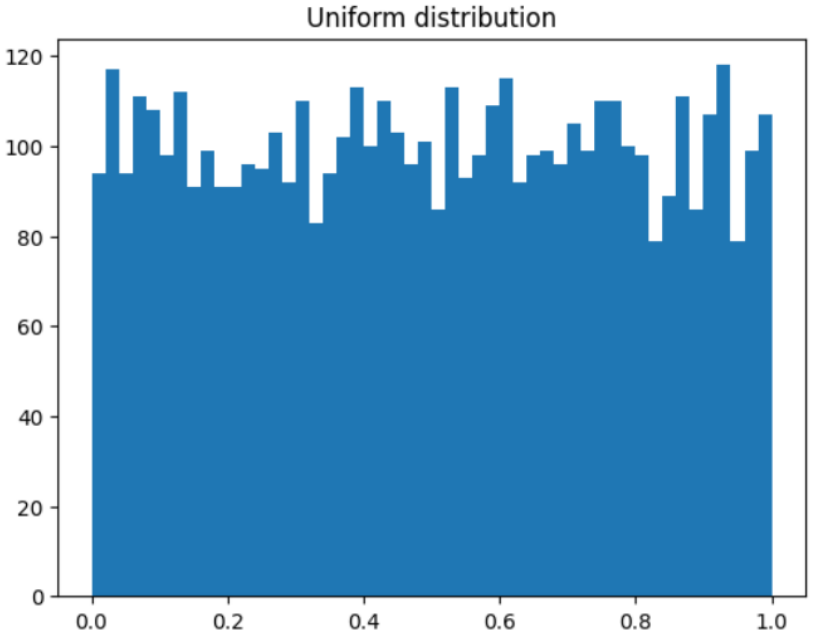
Perioden til en pRNG

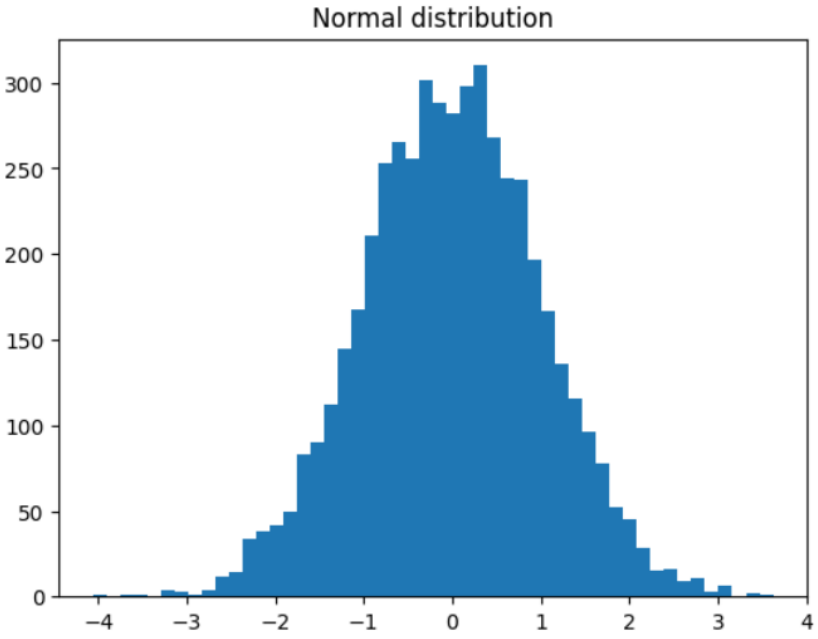
- Viktig at algoritmen kan gå veldig lenge før den kommer tilbake til start-tilstanden (hvor den vil begynne på nytt)
- Mersenne Twister-algoritmen (som brukes av **random**-biblioteket i Python) har kan produsere $2^{19937} - 1$ unike tall før den kommer tilbake til utgangspunktet (periode: $2^{19937} - 1$)
- Disse tallene danner da basisen for tallrekker som er (statistisk sett) tilfeldig fordelt
- $2^{19937} - 1$ er et stort tall:

```
>>> print(2**19937 - 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: Exceeds the limit (4300 digits) for integer string conversion
```

Sannsynlighet

- En lang pseudorandom tallrekke kan brukes til å produsere uniformt fordelte desimaltall mellom 0 og 1
- Hvis vi f.eks. vil ha tall mellom 5 og 10 kan vi ta $y = 5x + 5$ der $x \in [0, 1)$
- Kan også konvertere til andre sannsynlighetsfordelinger

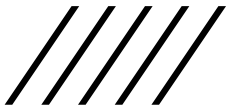




If we want to shift the standard deviation of the normal distribution or its mean, we can simply multiply each sample by σ, and if we want to move the mean, we simply add μ to each sample.

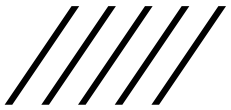
OBS: rand() i C++ har lager problemer!

- Lager samme tallrekke hver gang hvis vi bruker **rand()**
- Må gi den et seed – med for eksempel **srand(time(nullptr))** - for å få en ny tallrekke hver gang
- Bare garantert på ha en periode på minst 32 000 tall – altfor lite for en vitenskapelig applikasjon



Bedre løsning:

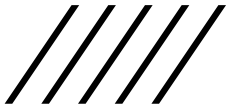
```cpp
#include <random>
#include <ctime>

int main()
{
    // use current time as seed
    int seed = std::time(nullptr);

    // set up the random engine that produces randomness
    std::mt19937 engine = std::mt19937(seed);

    // object that converts randomness to the distribution we want
    std::uniform_real_distribution<float> uniform =
        std::uniform_real_distribution<float>(0, 1);

    // random number between 0 and 1 (uniform distribution)
    float random_number = uniform(engine);
}
```




LIVEKODING:
TILFELDIGHET
I PYTHON

Velg riktig bilbiotek!

- Skal du lage noe som har med kryptering og sikkerhet å gjøre? **import secrets**
- Trenger du mange tilfeldige tall på en gang? **import numpy**
- Ingen av de to over? **import random**



https://www.menti.com/

Kode: 8774 1587

Etter forelesningen

- Devilry-innlevering for prosjekt 2
- Lykke til med innspurten!