☐ Tilfeldige tall

LAST NED PDF

Introduksjon

Det er helt nødvendig å ha tilfeldige tall for å få god kryptering. Vi skal se mer av det i en senere leksjon, men inntil videre kan du tenke tilbake til Cæsar-chifferet som du har jobbet med tidligere. I kryptografien er det et grunnleggende prinsipp at motstanderen skal få vite alt om systemet, bortsett fra den hemmelige nøkkelen. Hvis man bestemmer at man alltid skal justere alfabetet tre plasser, så er det altså essensielt sett ingen sikkerhet, siden det ikke *er* noen hemmelig nøkkel. Du kan få det litt bedre ved å velge et tilfeldig tall som du vil justere alfabetet med.

I denne leksjonen skal vi lære hvordan vi kan trekke tilfeldige tall på forskjellige måter, og vi skal også lære å bruke det som kanskje er programmererens aller viktigste verktøy: Leverandørens dokumentasjon.

Steg 1: Trekke et tilfeldig tall

Python gjør det enkelt å trekke tilfeldige tall, det er et helt bibliotek som heter `random` og som inneholder en rekke nyttige funksjoner. Hele listen over funksjoner kan du lese på python.org, i denne omgangen bryr vi oss kun om funksjonene som jobber med heltall.

Sjekkliste

- ☐ Gå inn på https://docs.python.org/3/library/random.html#functions-for-integers, og les om `randrange`og `randint`.
- ☐ Åpne en ny fil i IDLE med den følgende koden

  ```
  from random import randint

  for i in range(100):
      print(randint(0, 1000))
  ```

  Kjør programmet.
- ☐ Modifiser koden slik at den i stedet skriver ut tilfeldige verdier mellom 100 og 800. Sjekk dokumentasjonen for å se om også tallene 100 og 800 er mulige output fra koden din.
- ☐ Modifiser koden slik at den nå bare skriver ut *partall* mellom 100 og 800. (*Hint:* Se på den andre varianten av `random.randrange` i dokumentasjonen.)

Legge dataene inn i et histogram

I matematikktimene har du sett hvordan man kan bruke et *histogram* for å se hvordan verdier er fordelt. Det kan du gjøre her også for å se at dataene blir jevnt fordelt, og at det ikke blir for kraftig samling på visse steder. Kopier alle de tilfeldige tallene fra koden i punkt 2 over (du må antageligvis rulle oppover for å få med deg alle), og gå inn på https://www.wessa.net/rwasp_histogram.wasp. Lim inn tallene der det står "Data". Sett "Sample Range" til henholdsvis 0 og 1000 (eller andre grenser dersom du har gjort endringene i koden over). Sett "Number of bins" til 4 eller 5. Trykk til slutt "Compute", og du vil få opp en graf som viser hvordan tallene er fordelt. Vi har bare 100 tall, så variasjonen er antageligvis ganske høy, men om du øker antallet tilfeldige tall, vil du se at det nærmerer seg like store søyler. Det er det første viktige kravet vi må stille til våre tilfeldige tall: Alle må være like sannsynlige.

Steg 2: Trekke det samme tilfeldige tallet flere ganger

Datamaskiner er egentlig veldig dårlige på å lage tilfeldige tall, og det er derfor tittelen på kapittel 9.6 i dokumentasjonen er *pseudo-random numbers*. Man kan sammenligne det med å kaste en terning: Dersom man bygger en maskin som kaster terning, men som alltid bruker nøyaktig den samme kraften, og sender den ut med den samme vinkelen, så vil terningkastet også alltid bli det samme -- så lenge terningen settes likt inn i maskinen.

Poenget med dette er altså at datamaskiner trenger en *startinnstilling* for å trekke tilfeldige tall, og det kalles en *seed*. Så lenge man bruker samme seed, vil man alltid få ut de samme tallene.

Sjekkliste

- [ ] Endre filen fra steg 1 slik at den nå ser slik ut

```
from random import randint

seed('Kodeklubben')

for i in range(100):
    print(randint(0, 1000))
```

- [ ] Kjør koden flere ganger. Ser du at tallene blir de samme hver gang?

Det vi kan lære av dette eksempelet er at de tilfeldige tallene vi lager aldri blir bedre enn den verdien vi legger inn i starten. Python bruker som standard det nøyaktige tidspunktet for når programmet startet som inndata. Dersom en kryptolog gjør som Python, og bruker tidspunktet som oppstart for sine tilfeldige tall, holder det å gjette på tidspunktet (og det er ofte ikke så veldig vanskelig å vite) for å knekke hele koden. Vi kommer tilbake til hvordan vi kan løse dette problemet.

Steg 3: Tilfeldige tall for hemmelige koder

I steg 1 så vi at det er viktig at de tilfeldige tallene blir fordelt *uniformt*, altså at alle er like sannsynlige, og i steg 2 så vi at de tilfeldige tallene fullt og helt avhenger av hva vi legger inn i starten. Det er også et tredje problem vi må tenke på: Dersom man får se på tidligere verdier, kan man da forutse hva de neste tallene skal bli? Under innledningen i Python-dokumentasjonen av `random` kan man lese følgende advarsel i en rød boks:

> **Warning:** The pseudo-random generators of this module should not be used for security purposes. For security or cryptographic uses, see the `secrets` module.

Slike advarsler er grunnen til at man alltid bør lese dokumentasjonen, iallfall når man skal gjøre noe viktig. Årsaken til at man ikke bør bruke `random`-biblioteket til hemmelige koder er nettopp at det kanskje kan være forutsigbart.

Vi skal derfor nå se nærmere på biblioteket `secrets`.

Sjekkliste

- [ ] Les Python-dokumentasjonen for tilfeldige tall i `secrets`-biblioteket, avsnitt 15.3.1.
- [ ] Gjør punktene fra steg 1 på nytt, men denne gangen ved å bruke passende funksjoner fra `secrets` i stedet for `random`.

Hint

I stedet for `randint` må du nå bruke `randbelow`. Legg merke til at den ikke vil trekke den den øvre grensen du legger inn, så du må gjøre den øvre grensen én større. Hvis du vil velge fra et intervall, kan du trekke ved hjelp av `randbelow`, og så legge til den nedre grensen din.