INFO132 Grunnkurs i programmering

Obligatorisk innlevering 4

Innleveringsfrist: 25. september, 14:00

I denne oppgaven skal du besvare noen spørsmål i et eget dokument. Du skal også skrive tre program-filer. Du skal levere *ett* tekstdokument og *tre* py-filer.

Oppgave 1 – Teori

I tekstdokumentet skal du besvare følgende spørsmål (`pdf`/`doc`/`docx`/`txt`/spør gruppelederen din):

1. Hva skiller tomme funksjoner ("*void functions*") fra ikke-tomme funksjoner ("*fruitful functions*")?
2. Hva skiller deterministiske funksjoner fra ikke-deterministiske funksjoner?
3. Forklar *kort* hva vi mener med *argument* og *parameter*.
4. Redegjør *kort* for hva nøkkelordene `def` og `return` betyr i funksjonsdefinisjoner.
5. For hver av funksjonene `print`, `input`, `len` og `randrange` (fra `random`-modulen), angi hvorvidt det er en tom eller ikke-tom funksjon, og hvorvidt den er en deterministisk eller ikke-deterministisk funksjon.
6. Vi har sett to måter å *importere* eksternt definerte funksjoner. For eksempel:
   - `from random import randrange`, eller bare
   - `import random`

   Begge disse gir deg tilgang til funksjonen `randrange`, men du må skrive ulike ting for å kalle funksjonen.

   (a) Hvordan kan du kalle funksjonen i de to tilfellene?

   (b) Hvilken av disse importerer en *modul* (Kap. 4.5)?
7. I oppgave 2 listes det opp vilkår for tre ulike returverdier. Finnes det tilfeller som *ikke* er dekket var listen? Altså, finnes det *tekstverdier* vi kan gi som argument som *ikke* er beskrevet i noen av de tre punktene? (Det kan være lurt å fullføre oppgave 2 før eller samtidig som du svarer på denne.)
8. I oppgave 3 skal du skrive et program som bruker kode du har skrevet et annet sted (oppgave 2). Forklar hvordan vi kan bruke en slik teknikk for å *abstrahere* bort fra kompleks kode (forelesning 2).
9. I oppgave 4 skal du definere en *rekursiv* funksjon. Forklar *kort* hva vi mener med en rekursiv funksjon?

Oppgave 2 – Én bit av hangman

Opprett en ny fil og kall den `hangman.py`. I denne filen skal du definere en funksjon: `correct_guess`. Denne funksjonen skal ta *tre* argumenter:

1. `solution`,
2. `attempts`, og
3. `proposal`.

Vi antar at alle er tekstverdier. Funksjonen din skal ha *tre* mulige utfall. Den skal returnere:

`None` dersom `proposal` har lengde 0, eller mer enn 2 (altså hvis det ikke er ett enkelt symbol), eller dersom `proposal` *er i* tekstverdien `attempts` (altså dersom har blitt gjettet tidligere).

`True` dersom forrige punkt ikke gjelder *og* `proposal` *er i* tekstverdien `solution`.

`False` dersom første punkt ikke gjelder *og* `proposal` *ikke er i* tekstverdien `solution`.

Oppgave 3 – Et ufullstendig spill

I denne oppgaven skal du skrive en ufullstendig spill som *importerer* og bruker funksjonen du skrev i oppgave 2. Opprett en fil du kaller `spill.py` og plasser den i *samme* mappe som `hangman.py` fra forrige oppgave. Du skal ha tre variabler definert i programmet ditt: `løsning`, `forsøk` og `neste`. De første to skal ha verdier hhv. `"hangman"` og `"efgh"`. Den siste skal du lese inn fra brukeren.

Hensikten med disse variablene er at vi skal skrive et program som interagerer med brukeren, og at vi nå skal simulere en tilstand hvor brukeren spillet et hangman spill, løsningen (som vi tenker er ukjent for brukeren) er "hangman", og brukeren har foreløpig tippet bokstavene 'e', 'f', 'g', og 'h'.

Skriv et program med konvensjonell struktur (Appendiks: Organisering av kode) som består av en `import`-setning som gir deg tilgang til løsningen i oppgave 2, *én* funksjon som du kaller `main`, og det betingede funksjonskallet til hovedfunksjonen på slutten. I hovedfunksjonen, skal programmet spørre brukeren om et tegn som tilordnes til variabelen `neste`. Så skal du kalle `correct_guess` funksjonen med argumenter: `løsning`, `forsøk`, og `neste`. Presenter resultatet av funksjonskallet på en fornuftig måte.

Oppgave 4 – Rekursjon

Skriv en python fil med to funksjonsdefinisjoner: `fibonacci` og `main`. Den første funksjonen skal være en *rekursiv* funksjon, som har ett parameter, `n` som vi antar er et ikke-negativt heltatt, og returnerer det `n`-te Fibonacci tallet.

Hovedfunksjonen, `main`, skal be brukeren om et tall, konvertere svaret til et heltall dersom det er mulig, og skrive ut det tallet i Fibonacci-sekvensen brukeren ba om. Dersom det ikke er mulig å konvertere svaret til et heltall, skal programmet skrive ut en feilmelding som du har forfattet (ikke Python sin standard feilmelding).

Fibonacci-sekvensen definerer vi som: 1, 1, 2, 3, 5, 8, osv. Der neste tall er summen av de to forrige. Vi begynner å telle fra 0, så hvis brukeren skriver inn teksten `"4"`, skal programmet altså skrive ut 5 som svar.

Appendiks: Organisering av kode

Det kan være veldig lurt å organisere programkoden vår i ulike funksjoner. En vanlig måte å skrive en programfil på er å begynne med eventuelle `import`-setninger, etterfulgt av funksjoner som utgjør programmet vårt. En av disse (ofte den siste, men det er ikke kritisk), er *hovedfunksjonen*. Denne kalles gjerne `main` og beskriver oppførselen av hele programmet.

Helt til slutt kommer to linjer som *ikke* er skrevet i en funksjon, men ligger "ytterst" i programkoden. De to linjene utgjør et betinget kall til hovedfunksjonen. Betingelsen for å kalle hovedfunksjonen er at variabelen `__name__` er lik tekstverdien `"__main__"`[a]. Merk at både i variabelnavnet, `__name__`, og i literalen vi tester likhet med, `"__main__"`, er det *to* understreker, _, både før og etter.

Dette er den konvensjonelle måten å organisere et Python-program på. Oppgave 3 og 4 skal organiseres på denne måten.

[a]Hensikten med denne betingelsen er å sørge for at hovedfunksjonen kun kalles når programfilen din *kjøres* som et script eller program. https://docs.python.org/3.7/library/__main__.html

```
hjelpe_funksjon_1

hjelpe_funksjon_2

main

if __name__ == "__main__":
  main()
```

Konvensjonell struktur

Appendiks: Delstrenger

Vi sier at en tekstverdi, `x1`, er en *delstreng* av en annen tekstverdi, `x2`, hvis og bare hvis du kan omforme `x1` til `x2` ved å legge til tegn på begynnelsen og slutten. Eller med andre ord, hvis du kan finne `x1` et sted i `x2`. Tekstverdien "er" er en *delstreng* av tekstverdien "Bergen".

Vi kan utforme et sannhetsuttrykk som beskriver at tekstverdien `x1` er en delstreng av tekstverdien `x2` som følger: `x1 in x2`. Vi kan uttrykke en *delstreng-test* på samme måte som vi for eksempel uttrykker en likhetstest, som illustrert under.