Finne ut om en løsning er helt riktig og korrigere ved behov

Finurlige feil og debugging av kode

IN1000, uke5
Henrik H. Løvold

Oppgave
(Lett modifisert fra eksamen 2014)

Skriv en funksjon

```
def pris(gratis, alder):
```

Dersom parameteren **gratis** har verdien **True**, skal funksjonen *alltid* returnere 0. Dersom parameteren **gratis** har verdien **False** og verdien av **alder** er mindre enn 18, skal funksjonen returnere 100, ellers 200. Altså skal f.eks. kallet **pris (true, 10)** returnere 0, kallet **pris(false,10)** returnere 100 og kallet **pris(false, 50)** returnere 200.

En mulig løsning

```
def pris(gratis, alder):
    if gratis:
        svar = 0
    elif alder < 18:
        svar = 100
    else:
        svar = 200

    return svar
```

En annen mulig løsning
*(returnere inni **if**)*

```
def pris(gratis, alder):
    if gratis:
        return 0
    elif alder < 18:
        return 100
    else:
        return 200
```

En annen mulig løsning
*(returnere inni **if**)*

```
def pris(gratis, alder):
if gratis:
    return 0
if alder < 18:
    return 100
return 200
```

En tredje mulig løsning
*(bare rene **if** - ingen else..)*

```
def pris(gratis, alder):
    if gratis:
        svar = 0
if ((not gratis) and alder < 18):
    svar = 100
if ((not gratis) and alder >= 18):
    svar = 200

return svar
```

En uriktig løsning

```
def pris(gratis, alder):
    if gratis != False:
        if alder < 18:
            svar = 100
elif alder>= 18:
    svar = 200
else:
    svar = 0
return svar
```

Finne feil og utforske hvordan kode helt presist kjører

- Hva gjør man dersom et program ikke gjør akkurat det man ønsker?
  - Man kan kikke direkte på koden og se om man skjønner hvor i programmet feilen ligger og hva som går galt
  - Man kan legge inn *print* ulike steder i koden for å se i hvilken rekkefølge ulike linjer kjører og hvilke verdier ulike variable har
  - Man kan legge inn *assert* ulike steder i koden
  - Man kan skrive/kopiere bestemte linjer til tolkeren for å se om linjen evaluerer slik man tror
  - Man kan se hva metoder returnerer når de kalles med ulike argumenter
  - Man kan kjøre koden med en **avluser (debugger)**

Kikke på koden

```
def pris(gratis, alder):
    if gratis != False:
        if alder < 18:
            svar = 100
    elif alder>= 18:
        svar = 200
    else:
        svar = 0
    return svar


pris(False, 15)
```

- Se hvilken **if** hver **elif**/**else** hører sammen med
- Se på uttrykk og tenke hva det evaluerer til med ulike variabelverdier
- Følge kjøring med blyant og papir (eller i hodet)

Legge inn *print* i koden

```
def pris(gratis, alder):
    if gratis != False:
        if alder < 18:
            print("Linje4"+str(gratis))
            svar = 100
    elif alder>= 18:
        print("Linje7"+str(gratis))
        svar = 200
    else:
        print("Linje10"+str(gratis)+str(alder))
        svar = 0
    return svar


pris(False, 15)
```

Legge inn *assert* i koden

```
def pris(gratis, alder):
    if gratis != False:
        if alder < 18:
            assert not gratis
            svar = 100
    elif alder>= 18:
        assert not gratis
        svar = 200
    else:
        assert gratis
        svar = 0
    return svar


pris(False, 15)
```

Se på bestemte linjer i tolkeren (interpreter)

```
gratis=False
gratis != False
```

```
gratis=True
gratis != False
```

- I stedet for å kjøre hele kodefiler kan man kjøre én og én linje i python-tolkeren:
- Fra kommandolinjen skriver man "python3" for å starte tolkeren
- Man kan da skrive og utføre en enkelt linje av gangen
- Nyttig for å utforske hvordan ulike varianter av uttrykk helt presist evaluerer

Sjekke hva som returneres med ulike argumenter

```
def pris(gratis, alder):
    if gratis != False:
        if alder < 18:
            svar = 100
    elif alder>= 18:
        svar = 200
    else:
        svar = 0
    return svar


print( pris(False, 15) )
print( pris(False, 20) )
print( pris(True, 15) )
print( pris(True, 20) )
print( pris(False, 18) )
```

- Legg inn funksjonskall med ulike argumenter som dekker de ulike måtene å påvirke kjøring av funksjonskoden

Avluser (debugger)

```
def pris(gratis, alder):
    if gratis != False:
        if alder < 18:
            svar = 100
    elif alder>= 18:
        svar = 200
    else:
        svar = 0
    return svar


pris(False, 15)
```

- En avluser lar oss kjøre hele programmer og samtidig ha kontroll på enkeltlinjer
- Svært nyttig for å finne feil i store programsystemer, og potensielt nyttig når man lærer å programmere
  - Men sørg i tilfelle for at det støtter læring av programmering, ikke stjeler fokus fra det..
- Enkel variant å eventuelt prøve ut: pythontutor.com

Oppgave

```
def alle_er_innenfor(tallene):
  for tall in tallene:
      if tall>10 and tall<20:
          innenfor = True
      else:
          innenfor = False
  return innenfor
```

- Hvorfor er ikke koden over riktig?
  (i henhold til intensjonen - sjekke om alle tall i listen er mellom 10 og 20)
- Hvordan går man best frem for å finne ut om en slik løsning er riktig eller ikke?
  (og man ikke har kompilator tilgjengelig - som i eksamens-situasjonen)
- På hvilke måter kan man endre koden til å bli riktig?
  (forsøk gjerne å finne flere ulike korrekte løsninger)
- Hva kan man lære fra denne oppgaven?
  (viser den noen generelle poeng som også vil gjelde andre oppgaver)

Mulige fremgangsmåter

- Hvorfor koden ikke er riktig
  - Kjører **`alle_er_innenfor([5,15])`** i Pythontutor
- Hvordan finne ut at en slik løsning ikke er riktig
  - Den greieste måten kan være å tenke på mulige kall..
- Endre koden til å bli riktig
  - a) Sette **`innenfor=True`** før løkka, ikke inni(!)
  - b) Returnere ved første `False`
- Hva kan man lære fra denne oppgaven?
  - Åpen for forslag!
  - Étt poeng: løsninger med løkker krever ofte asymmetri