NTNU
Kunnskap for en bedre verden

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgave i TDT4110 Informasjonsteknologi – grunnkurs, med Python

Faglig kontakt under eksamen:

Alf Inge Wang	Mobil: 922 89577
Guttorm Sindre	Mobil: 944 30245
Kristoffer Hagen	Mobil: 907 67307

Eksamensdato: **2015-12-16**

Eksamenstid (fra-til): **09:00 – 13:00**

Hjelpemiddelkode/Tillatte hjelpemidler: **Godkjent kalkulator**

Annen informasjon:

Oppgavesettet inneholder 4 oppgaver. Det er angitt i prosent hvor mye hver oppgave og hver deloppgave teller ved sensur. Les igjennom hele oppgavesettet før du begynner å løse oppgavene. Disponer tiden godt! Gjør rimelige antagelser der du mener oppgaveteksten er ufullstendig, skriv kort hva du antar.

Svar kort og klart, og skriv tydelig. Er svaret uklart eller lenger enn nødvendig trekker dette ned.

Målform/språk: **Bokmål**

Antall sider: **17 (inkl. Forside, svarark og appendiks)**

Innhold:

- Oppgave 1: Flervalgsoppgave (25%)
- Oppgave 2: Kodeforståelse (15%)
- Oppgave 3: Programmering reisetid (20%)
- Oppgave 4: Programmering sensur (40%)
- Appendiks: Nyttige funksjoner
- Svarark til Flervalgsoppgave (2 eksemplarer)

Kontrollert av:

8. des. 2015	Rune Sætre
Dato	Sign

Oppgave 1: Flervalgsoppgave (25%)

Bruk de to vedlagte svarskjemaene for å svare på denne oppgaven (ta vare på det ene selv). Du kan få nytt ark av eksamensvaktene dersom du trenger dette. Kun ett svar er helt riktig. For hvert spørsmål gir korrekt avkryssing 1 poeng. Feil avkryssing eller mer enn ett kryss gir −1/2 poeng. Blankt svar gir 0 poeng. Du får ikke mindre enn 0 poeng totalt på denne oppgaven. Der det er spesielle uttrykk står den engelske oversettelsen i parentes.

1. Hva betyr Random Access Memory?
    a. At det er tilfeldig hvor maskinen lagrer informasjon.
    b. At hukommelsescellene kan aksesseres direkte i tilfeldig rekkefølge.
    c. Hukommelsen er plassert på forskjellige, tilfeldige, plasser på hovedkortet.
    d. At det kan oppstå tilfeldige feil i deler av hukommelsen.
2. Når brukes fotolitografi i produksjonen av datamaskiner?
    a. Når man etser inn navnene på portene bak på maskinen.
    b. Under produksjon av integrerte kretser.
    c. Når man lager bildene som skal inn i brukermanualen.
    d. Når man monterer integrerte kretser på kretskortene.
3. Hva er «pipelining»?
    a. Et uttrykk for det som skjer når man skriver mye data til harddisken samtidig.
    b. En teknikk der man sender data mellom de forskjellige delene i maskinen i «pipes».
    c. En teknikk der en CPU kan utføre flere instruksjoner parallelt.
    d. En teknikk som fungerer som en ”sikker tunnel” mellom din maskin og en tjener.
4. Hva finner alle burst-feil med lengde n bit, gitt en n-bit maske, men er uegnet til kryptografi?
    a. Enkel sjekksum
    b. HASH-funksjoner
    c. Paritet
    d. Syklisk sjekksum (CRC)
5. I TCP/IP-protokollen …
    a. sendes alle pakkene sendes langs den samme ruten til mottakeren.
    b. brukes pakkesvitsjing.
    c. mottas ingen ting før siste IP-pakke er framme.
    d. er det mindre interferens pga. at mindre biter sendes hver for seg.
6. Hva blir det binære tallet `10101010` desimalt?
    a. 170
    b. 180
    c. 190
    d. 200
7. Hvilken av følgende RGB-fargekodinger gir blått?
    a. `f1faf0`
    b. `120012`
    c. `0000ff`
    d. `884311`

8. Vi har en liste av navn, à la **liste = ['Jo Å', 'Geir Li', 'Ine By']** men i praksis med vesentlig flere navn enn dette. Lista er ikke sortert og kan inneholde duplikater (dvs. samme navn kan stå flere sted i lista). Vi skal skrive en funksjon **antall (liste, navn)** som skal returnere et heltall som sier hvor mange ganger navnet forekommer i lista. Vi kladder følgende pseudokode:

```
function antall (liste, navn):
        antall ← 0
        la n gå fra og med første til og med siste element i liste:
                hvis n == navn:
                        antall ←antall + 1
        returner antall
```

   Spørsmål: Kjøretidskompleksiteten til pseudokoden over vil være?
   a. $\Theta(n)$
   b. $\Theta(n \log n)$
   c. $\Theta(\log n)$
   d. $\Theta(n^2)$

9. I ei sortert liste kunne vi brukt binærsøk i stedet for løkka **"la n gå… "** i pseudokoden fra spørsmål 8. En alternativ algoritme som først sorterer lista, og deretter bruker binærsøk for å finne navnet, vil ha…
   a. lavere kompleksitet (dvs. være raskere) enn pseudokoden gitt over.
   b. høyere kompleksitet enn pseudokoden over.
   c. samme kompleksitet som pseudokoden over.
   d. høyere kompleksitet hvis navnet fins null eller én gang i lista, lavere hvis det fins flere ganger.

10. Funksjonen **antall** for ei usortert liste, som beskrevet i spørsmål 8, kan i Python implementeres ved den innebygde funksjonen **count**, som gjør at funksjonskroppen kan skrives som en eneste kodelinje. For eksempel

```
def antall (liste, navn):
        return  liste.count(navn)
```

    Spørsmål: Hvilken kjøretidskompleksitet vil denne koden ha?
    a. $\Theta(1)$
    b. $\Theta(\log n)$
    c. $\Theta(n)$
    d. $\Theta(n^2)$

11. Hvorfor ønsker man å bruke en SSD heller enn en vanlig magnetisk harddisk?
    a. En SSD øker minnet på grafikkortet slik at spill og lignende flyter bedre.
    b. I en SSD lagres data i elektroniske kretser. Det er ingen bevegelige deler, og dermed blir disken raskere og mer pålitelig.
    c. Det er lettere å lagre data permanent på en SSD.
    d. En SSD er ikke så utsatt for strømtopper og tåler derfor mer enn en magnetisk disk.

12. Hva er motivasjon til fagfeltet systemutvikling?
    a. Raskere kode.
    b. Utvikle programvare med best mulig kvalitet uavhengig av budsjett og tid.
    c. Legge til rette for at all programvare skal utvikles i spesifiserte faser etter hverandre.
    d. Utvikle programvare med god nok kvalitet innen tid og budsjett.

13. Hva betyr modulering i f.eks. FM og AM?
    a. Det beskriver hvordan man kan sende et signal over en bærebølge.
    b. Det beskriver hvordan man kan få oversikt over hele internett.
    c. Det beskriver hvordan man kan øke strømstyrken slik at flere får tilgang.
    d. Det beskriver hvordan man kan gruppere internett i hensiktsmessige biter.

14. Dersom tekststreng 'Hallo' i ASCII representeres som følgende tall heksadesimalt: `48 61 6c 6c 6f`, hvordan representerer man på samme måte 'Morna'?
    a. `4e 65 69 64 61`
    b. `4e 54 4e 55 21`
    c. `4d 6f 72 6e 61`
    d. `55 66 6g 7h 61`

15. En fordel med vannfallsmodellen kan være:
    a. Enklere å håndtere øyeblikkelige krav fra kunder.
    b. Enklere å følge fremgang i forhold til prosjektplan for prosjektleder.
    c. Systemet reflekterer en gradvis bedre forståelse av brukernes behov.
    d. Gir raskere levering og kortere tid til å ta i bruk fungerende deler av systemet.

16. Hvor mange bytes trengs for å lagre et 24-bits bilde med 1280x1024 piksler uten komprimering?
    a. Ca. 3,8MB
    b. Ca. 1,2MB
    c. Ca. 24MB
    d. Ca. 24GB

17. Hva er den første aktiviteten i ”requirement engineering”-prosessen i følge læreboka?
    a. Gjennomførbarhetsstudie.
    b. Kravinnhenting og analyse.
    c. Kravspesifisering.
    d. Validering av krav.

18. Hva er akseptansetesting?
    a. Teste hvordan ulike deler av systemet fungerer sammen.
    b. Teste at hver enkelt funksjon i systemet fungerer som den skal.
    c. Teste at operativsystemet aksepterer systemet på plattformen.
    d. Teste med kundedata for å sjekke om systemet møter kundens behov.

19. Hvilken av følgende teknikker er en tapsløs komprimering?
    a. Run-length encoding.
    b. Analog-to-digital conversion.
    c. JPEG-encoding.
    d. Check-sum generation.

20. Hva er det Boehm’s spiralmodell inneholder, som man ikke finner i vannfallsmodellen eller inkrementell utvikling?
    a. Risikoanalyse.
    b. Testing/Validering.
    c. Kravspesifisering.
    d. Vedlikehold.

Oppgave 2 Kodeforståelse (15%)

Oppgave 2a (5%)
Hva blir skrevet ut til skjerm når du kjører koden som vist under? (3 %)
Forklar med en setning hva funksjonen **`mystery`** gjør (2 %)

```
def mystery(x,y):
  z=""
  for i in range(0,len(x)):
    if (i%2==1):
      z+=x[i]
    else:
      z+=y[i]
  return z

A="SUNEAILSUN"
B="JALTNCSAES"
D=mystery(A,B)
print(D)
```

Oppgave 2b (5%)
Hva blir skrevet ut til skjerm når du kjører koden som vist under? (3 %)
Forklar med en setning hva funksjonen **`compute`** gjør (2 %)

```
def compute(x):
  if x<10:
    return x*compute(x*2)
  else:
    return 1

print(compute(1))
```

Oppgave 2c (5%)
Hva blir skrevet ut til skjerm når du kjører koden som vist under? (3 %)
Forklar med en setning hva funksjonen **`a`** gjør (2 %)

```
def a(c):
  for b in range(len(c)):
    d = b
    for e in range(b+1,len(c)):
      if c[e]>c[d]:
        d = e
    f = c[b]
    c[b] = c[d]
    c[d] = f
  return c

print(a([2,5,3,8,6,1,7]))
```

Oppgave 3 Programmering reisetid (20%)

Skriv funksjonene slik at du kan gjenbruke dem. Hvis du ikke klarer å løse en oppgave kan du likevel gjenbruke funksjoner fra den oppgaven i en senere oppgave.

Oppgave 3a (5%)

Lag funksjonen **readTime** (ingen input-parametere) som spør brukeren om å skrive inn et klokkeslett med time, minutt og sekund angitt separat som angitt i eksemplet under. Funksjonen skal sørge for at brukeren skriver inn et korrekt klokkeslett (man kan anta at bruker skriver inn et tall og ikke bokstaver). Den skal gi feilmelding ved feil brukerinput og spørre brukeren å oppgi time, minutt eller sekund på nytt hvis tallet ikke er korrekt. Funksjonen skal returnere en liste med tre elementer [hour, minute, sec].

Eksempel på kjøring og på hva som skal skrives ut til skjerm (brukerinput er vist med understrekning og fet skrift):

```
>>> time = readTime()
Enter hour: 24
- ERROR: Hour must be between 0 and 23!
Enter hour: 12
Enter minute: -5
- ERROR: Minute must be between 0 and 59!
Enter minute: 34
Enter second: 65
- ERROR: Second must be between 0 and 59!
Enter second: 20
>>> print(time)
[12, 34, 20]
>>>
```

Oppgave 3b (5%)

Lag funksjonen **convertTime** som tar inn to parametere **time** og **mode**. Parameteren **mode** kan ha to verdier: ’time’ eller ’sec’. Hvis mode har verdien ’time’, skal funksjonen konvertere tid angitt i sekunder til tid angitt som en liste på formatet [hour, min, sec] og returnere dette. Hvis mode har verdien ’sec’, skal funksjonen returnere antall sekunder en liste på formatet [hour, min, sec] tilsvarer. Parameteren **time** kan altså både være et heltall og en liste med heltall avhengig av verdien av **mode**.

Eksempel på kjøring:

```
>>> time = convertTime(3857,'time')
>>> print(time)
[1, 4, 17]
>>> time = convertTime([1,4,17],'sec')
>>> print(time)
3857
>>>
```

Oppgave 3c (5%)
Lag funksjonen **travelTime** som hverken har input parametere eller returnerer noe. Funksjonen skal først spørre brukeren om et starttidspunkt og deretter et sluttidspunkt for en reise. Funksjonen skal sørge for at det blir lagt inn gyldige tidspunkt. Hvis brukeren prøver å legge inn et sluttidspunkt som er tidligere enn starttidspunktet, skal funksjonen skrive følgende feilmelding: *"- ERROR: Arrival time must be later than Departure time"*, samt spørre brukeren om et nytt sluttidspunkt. Funksjonen trenger ikke å ta hensyn til reiser som varer over midnatt. Funksjonen skal ikke returnere noe, men skrive ut reisetid angitt i timer, minutter og sekunder som vist under.

Eksempel på kjøring (bruker-input er vist med understrekning og fet font):

```
>>> travelTime()
Give departure time in hour, minute and second:
Enter hour: 26
- ERROR: Hour must be between 0 and 23!
Enter hour: 15
Enter minute: 20
Enter second: 20
Give arrival time in hour, minute and second:
Enter hour: 13
Enter minute: 15
Enter second: 39
- ERROR: Arrival time must be later than Departure time
Give arrival time in hour, minute and second:
Enter hour: 18
Enter minute: 59
Enter second: 59
Traveltime: 3 hours, 39 min, 39 sec
>>>
```

Oppgave 3d (5%)
Lag funksjonen **analyzeBusRoutes** som tar inn en parameter **BusTables** som er en todimensjonal tabell av heltall der hver rekke inneholder følgende: nummer på bussrute, starttidspunkt (angitt med time, minutt) og sluttidspunkt (angitt med time, minutt). Funksjonen skal ikke returnere noe, men skal skrive ut til skjerm nummer og reisetid på bussruten som tar lengst tid, og nummer og reisetid på bussruten som tar kortest tid som vist i eksempel på kjøring under. Hvis det finnes flere bussruter med samme kjøretid, skal funksjonen skrive ut rutenummer på den første den finner.

Hvis man kjører funksjonen med følgende opplysninger:

- Buss nr. 1, Starttid 15:00, stopptid: 15:19
- Buss nr. 3, Starttid 15:32, stopptid: 16:45
- Buss nr. 4, starttid 15:45, stopptid: 16:23
- Buss nr. 5, starttid 15:55, stopptid: 16:11

vil eksempel på kjøring bli som følger:

```
>>> Busses=[[1,15,0,15,19],
       [3,15,32,16,45],
       [4,15,45,16,23],
       [5,15,55,16,11]]
>>> analyzeBusRoutes(Busses)
The slowest bus route is bus nr. 3 and it takes 1 hour,  13 min.
The fastest bus route is bus nr. 5 and it takes 0 hour,  16 min.
>>>
```

Oppgave 4 Programmering Sensur (40%)

Hvert år tar ca. 2000 studenter ITGK-eksamen, og mye tid går med til å sensurere og sette karakterer. Derfor er det bestemt at det skal programmeres et system som skal hjelpe til med dette. For å sikre at andre universiteter/fag også kan få nytte av programmet bør det gjøres en del generaliseringer.

Du skal lage et system for registrere sensur av eksamen. Eksamen består vanligvis av fire oppgaver som hver teller 25%, 15%, 20% og 40% av karakteren. Den andre oppgaven har tre underoppgaver, den tredje har fire og den fjerde har åtte underoppgaver. Underoppgavene skal vektes slik at hver teller et bestemt antall prosent av hundre.

Det er en fordel å gjenbruke tidligere del-oppgaver der det er mulig. Du kan anta at alle funksjonene mottar gyldige argumenter (inn-verdier), hvis ikke annet er oppgitt.

Oppgave 4 a) (5%)

Lag starten på hovedprogrammet (ikke funksjon) som lagres i fila **grades.py**. Starten skal bare inneholde de konstantene som programmet trenger og ikke skrive ut noe til skjerm.

Eksempel på kjøring av Python-programmet **grades** og hvilke verdier NTNU_scores, NTNU_letters, TASKS og WEIGHTS da vil ha etter kjøring:

```
>>>
>>> NTNU_scores
(89, 77, 65, 53, 41, 0)
>>> NTNU_letters
('A', 'B', 'C', 'D', 'E', 'F')
>>> TASKS
('1', '2a', '2b', '2c', '3a', '3b', '3c', '3d', '4a', '4b', '4c', '4d', '4e',
'4f', '4g', '4h')
>>> WEIGHTS
(25, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5)
>>>
```

Oppgave 4 b) (5%)

Lag funksjonen **`makeArray`** som har to inn-parametere (**`Numbers`** og **`Texts`**), der **Numbers** er en liste av tall, og **Texts** er en liste av tegn og/eller bokstaver (begge listene har samme lengde). Funksjonen skal returnere en to-dimensjonal tabell (lister i liste) som inneholder all informasjonen fra de to inn-parameterne som vist i eksemplet på kjøring under.

Eksempel på kjøring av funksjonen og hva den skal returnere (med input fra Oppgave 4a)):

```
>>> limitLetters = makeArray(NTNU_scores, NTNU_letters)
>>> print(limitLetters)
[[89, 'A'], [77, 'B'], [65, 'C'], [53, 'D'], [41, 'E'], [0, 'F']]
>>>
```

```
>>> weightTasks = makeArray(WEIGHTS,TASKS)
>>> print(weightTasks)
[[25, '1'], [5, '2a'], [5, '2b'], [5, '2c'], [5, '3a'], [5, '3b'], [5, '3c'],
[5, '3d'], [5, '4a'], [5, '4b'], [5, '4c'], [5, '4d'], [5, '4e'], [5, '4f'],
[5, '4g'], [5, '4h']]
>>>
```

Oppgave 4 c) (5%)

Sensor retter eksamen og setter poeng på hver oppgave med en poengsum som går fra og med 0 til og med 10, der 0 er dårligst (0% score) mens 10 er best (100% score). Sensur av en eksamen består av en liste med ett tall for hver deloppgave, i samme rekkefølge som de kommer (dvs. `1`,`2a`,`2b`, osv.)

Lag funksjonen **computeScore** med inn-parameteren **Points** som er en liste med poengsummer for deloppgavene, og konstanten **WEIGHTS** fra Oppgave 4 a). Funksjonen skal regne ut en totalscore i prosent for en eksamen, basert på vektingen av oppgavene.
Eksempel på kjøring av funksjonen og hva den returnerer:

```
>>> computeScore([10,10,10,10,10,10,10,10,10,10,10,10,10,10,10],WEIGHTS)
100.0
>>> computeScore([10,0,0,0,10,10,10,10,0,0,0,0,0,0,0,0],WEIGHTS)
45.0
>>> computeScore([5,0,0,0,10,10,10,10,0,0,0,0,0,0,0,0],WEIGHTS)
32.5
>>> computeScore([4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4],WEIGHTS)
40.0
>>>
```

Oppgave 4 d) (5%)

Skriv en funksjon **score2Letter** med to inn-parametere **scoreSum** (prosentvise total-poengsummen for en kandidat fra Oppgave 4 c)) og **limitLetters**-tabellen fra Oppgave 4b).
Funksjonen skal returnere den høyeste bokstavkarakteren som kandidaten har nok poeng til (uten avrunding). For eksempel vil 40.9 poeng gi F, mens 41.0 poeng gir E.
Eksempel på kjøring av funksjonen og hva den returnerer:

```
>>> score2Letter(88.9, limitLetters)
'B'
>>>
```

Oppgave 4 e) (5%)

Skriv en funksjon **addCandidate** som tar inn tre parametere: **candidateNumber**, **Scores** (en liste), og konstanten **WEIGHTS** fra Oppgave 4a). Funksjonen skal legge til følgende (adskilt med tabulator) som en linje på slutten av en fil med navn '**eksamen.txt**':

candidateNumber kommer først på linjen, så alle del-poengsummene fra **Scores**, og til slutt den prosentvise scoren med én desimals nøyaktighet og et linjeskift. Dersom fila ikke finnes, skal den opprettes. Dersom den ikke kan opprettes skal funksjonen avsluttes med Python's systemgenererte (default) feilmelding .

Eksempel på kjøring av funksjonen og hva den returnerer:

```
>>> addCandidate(12392,[10,0,0,0,10,10,10,10,10,0,0,0,0,0,0,0],WEIGHTS)
>>> addCandidate(33322,[0,10,10,10,0,0,0,0,0,10,10,10,10,10,10,10],WEIGHTS)

# Etter at minnepinnen som inneholder fila er fjernet…
>>> addCandidate(12492,[0,10,10,10,0,0,0,0,0,10,10,10,10,10,10,10],WEIGHTS)
[Errno 2] No such file or directory: 'eksamen.txt'

# Innholdet i filen eksamen.txt på minnepenna er nå:
12392 10    0     0     0     10    10    10    10    10    0     …     0     50.0
33322 0     10    10    10    0     0     0     0     0     10    …     10    50.0
```

Oppgave 4 f) (5%)

Skriv en funksjon **readResultFile** som har en inn-parameter **filename**. Funksjonen skal lese innholdet i fila med navn **filename** som er formatert som beskrevet i Oppgave 4 e) og legge innholdet i en to-dimensjonal tabell der hver rekke inneholder kandidatnummer, alle del-poengsummene og den prosentvise scoren. Kandidatnummer og del-poengsummene skal være av typen heltall, mens den prosentvise scoren skal være av typen flyttall. Funksjonen *trenger ikke* å håndtere feil som at filen ikke finnes eller filen ikke kan åpnes.

Eksempel på kjøring med fila "eksamen.txt" med innhold som i Oppgave 4e):

```
>>> Table=readResultFile('eksamen.txt')
>>> print(Table)
[[12392, 10, 0, 0, 0, 10, 10, 10, 10, 10, 0, 0, 0, 0, 0, 0, 0, 50.0],
 [33322, 0, 10, 10, 10, 0, 0, 0, 0, 0, 10, 10, 10, 10, 10, 10, 10, 50.0]]
>>>
```

Oppgave 4 g) (5%)

Skriv en funksjon **checkResultOK** som tar inn to parametere **filename** (navnet på fila med eksamensresultater som skal sjekkes) og **WEIGHTS** (fra Oppgave 4a)). Funksjonen skal lese inn fila og returnerer **True** hvis følgende er oppfylt:
- ingen kandidat er listet mer enn en gang i samme fil
- ingen har fått mindre enn 0 poeng eller mer enn 10 poeng på noen oppgave
- den prosentvise poengsummen for alle kandidatene er korrekt utregnet som i Oppgave 4c) over.

Funksjonen skal skrive ut feilmeldinger hvis den finner feil.

Eksempel på kjøring av funksjonen som viser hva den skriver ut og hva den returnerer (fil uten feil):

```
>>> checkResultOK('eksamen.txt',WEIGHTS)
True
>>>
```

```
# Filen eksamen2.txt har følgende innhold (inneholder feil):
12300   0   10  10  10  0   0   0   0   0   10  10  10  10  10  10  10 50.0
44400   4   4   11  0   0   0   0   0   0   0   0   0   0   0   0   0  19.0
12300   9   0   0   0   10  10  10  10  10  0   0   0   0   0   0   0  47.5
```

```
>>> checkResultOK('eksamen2.txt',WEIGHTS)
ERROR: Candidate 44400 scores are not between 0-10!
ERROR: Candidate 44400 has wrong total score!
ERROR: Candidate 12300 appears more than once!
False
>>>
```

Oppgave 4 h) (5%)

Skriv en funksjon **listAll** med inn-parametere **filename** og **limitLetters**. Funksjonen skal lese inn resultatene fra **filename** med samme formatet som i Oppgave 4e), og skrive ut en liste der hver linje inneholder femsifret kandidatnummer, prosentvis poengsum med ett siffers nøyaktighet og tilhørende bokstavkarakter etter reglene i Oppgave 4d). Kolonnene skal være justert vertikalt som vist her.

```
10000   75.4 C
10004    7.4 F
10008   88.1 B
10017   94.9 A
10019 100.0 A
10098   87.4 B
```

Utskriften skal være sortert etter kandidatnummer i stigende rekkefølge. Funksjonens returverdi skal være antallet kandidater som ble skrevet ut.

Eksempel på kjøring av funksjonen, hva den skriver ut, og hva den returnerer:

```
>>> listAll('eksamen2.txt',limitLetters)
12300  50.0 E
12300  47.5 E
44400  19.0 F
3
>>>
```

Denne siden er med hensikt blank!

Built-in:

%

Remainder (modulo operator): Divides one number by another and gives the remainder.

except ExceptionName as variable:

Catches an exception as default error message.

except ExceptionName:

Typical ExceptionNames are ValueError, ZeroDivisionError, IOError and Exception.

format(numeric_value, format_specifier)

Formats a numeric value into a string according to the format specifier, which is a string that contains special characters specifying how the numeric value should be formatted. Examples of various formatting characters are "f=floating-point, e=scientific notation, %=percentage, d=integer". A number before the formatting character will specify the field width. A number after the character "." will format the number of decimals.

int(x)

Convert a string or number to a plain integer.

float(x)

Convert a string or a number to floating point number.

len(s)

Return the length (the number of items) of a string, tuple, list, dictionary or other data structure.

round(number[, ndigits])

Return the floating point value number rounded to ndigits digits after the decimal point. If ndigits is omitted, it returns the nearest integer to its input.

range(start, stop[, step])

Rather than being a function, range is actually an immutable sequence type, as documented in Ranges and Sequence Types — list, tuple, range.

str([object])

Return a string containing a nicely printable representation of an object.

String methods:

s.isspace()

Returns true if the string contains only whitespace characters, and is at least one character in length. Returns false otherwise. (Whitespace characters are spaces, newlines (\n), and tabs (\t)).

s.ljust(width)

Return the string left justified in a string of length width.

s.rjust(width)

Return the string right justified in a string of length width.

s.strip()

Returns a copy of the string with all leading and trailing white space characters removed.

s.strip(char)

Returns a copy of the string with all instances of *char* that appear at the beginning and the end of the string removed.

s.split(str)

Returns a list of all the words in the string, using str as the separator (splits on all whitespace if left unspecified).

s.endswith(substring)

The substring argument is a string. The method returns true if the string ends with substring.

s.startswith(substring)

The substring argument is a string. The method returns true if the string starts with substring.

s.find(substring)

The substring argument is a string. The method returns the lowest index in the string where substring is found. If substring is not found the method returns -1.

s.replace(old, new)

The old and new arguments are both strings. The method returns a copy of the string with all instances of old replaced by new.

List operations:

s[i:j:k]
: Return slice starting at position i extending to position j in k steps. Can also be used for strings.

item in s
: Determine whether a specified item is contained in a list.

min(list)
: Returns the item that has the lowest value in the sequence.

max(list)
: Returns the item that has the highest value in the sequence.

s.append(x)
: Append new element x to end of s.

s.insert(index,item)
: Insert an item into a list at a specified position given by an index.

s.index(*item*)
: Return the index of the first element in the list containing the specified item.

s.pop()
: Return last element and remove it from the list.

s.pop(i)
: Return element i and remove it from the list.

s.remove(item)
: Removes the first element containing the item.

s.reverse()
: Reverses the order of the items in a list.

s.sort()
: Rearranges the elements of a list so they appear in ascending order.

Dictionary operations:

d.clear()
: Clears the contents of a dictionary

d.get(key, default)
: Gets the value associated with a specific key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.

d.items()
: Returns all the keys in a dictionary and their associated values as a sequence of tuples.

d.keys()
: Returns all the keys in a dictionary as a sequence of tuples.

d.pop(key, default)
: Returns the value associated with a specific key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.

d.popitem()
: Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary.

d.values()
: Returns all the values in dictionary as a sequence of tuples.

Files

open()
: Returns a file object, and is most commonly used with two arguments: open(filename, mode). Mode can be ‘r’ (read only), ‘w’ (writing only), ‘a’ (appending), ‘r+’ (both reading and writing).

f.read(size)
: Reads data from file and returns it as a string. Size is an optional and if left out the whole file will be read.

f.readline()
: Reads a single line from the file (reads until newline character (\n) is found), and returns it as a string.

f.readlines()
: Reads data from the file and returns it as a list of strings.

f.write(string)
: Writes the contents of string to file.

f.close()
: Close the file and free up any system resources taken up by the open file.

Svarskjema flervalgsoppgave

Kandidatnummer: _____ Program: _____

Fagkode: _____ Dato: _____

Antall sider: _____ Side: _____

Oppgavenr	***A***	***B***	***C***	***D***
1.1				
1.2				
1.3				
1.4				
1.5				
1.6				
1.7				
1.8				
1.9				
1.10				
1.11				
1.12				
1.13				
1.14				
1.15				
1.16				
1.17				
1.18				
1.19				
1.20				

Denne siden er med hensikt blank!

Svarskjema flervalgsoppgave

Kandidatnummer: _____ Program: _____

Fagkode: _____ Dato: _____

Antall sider: _____ Side: _____

Oppgavenr	*A*	*B*	*C*	*D*
1.1				
1.2				
1.3				
1.4				
1.5				
1.6				
1.7				
1.8				
1.9				
1.10				
1.11				
1.12				
1.13				
1.14				
1.15				
1.16				
1.17				
1.18				
1.19				
1.20				