Oppgave 2:

```
def a (x):
    x = x + 1
    y = 1 + x * 2

    return y


def b (n):
    y = 0
    if (n<40):
        y = 2 * n
    elif (n<10):
        y = n
    else:
        y = 1

    return y


def c (w):
    z = b(a(w))
    if (z<10):
        z = z + w
    return z
```

a) (2 %) Hva er verdien til x etter at vi har kjørt x = 2; x = a(x);?
b) (2 %) Hva er verdien til x etter at vi har kjørt x = 2; x = b(x);?
c) (2 %) Hva er verdien til x etter at vi har kjørt x = 2; x = c(x);?
d) (2 %) Hva er verdien til x etter at vi har kjørt x = 20; x = b(x);?
e) (2 %) Hva er verdien til x etter at vi har kjørt x = 50; x = b(x) + c(x);?

Løsning:
a) 7, b) 4, c) 14, d) 40, e) 52

Oppgave 3

a) (5 %) Poeng for hopplengde beregnes med følgende formel:

distance_points = 60 + (jump_distance – kpoint) * meter_value

Tallene kpoint og meter_value er fastsatt for hver hoppbakke.

Hoppbakken i Granåsen har kpoint 124 og meter_value 1,8. En skihopper som hopper 140 meter i denne bakken vil få 60 + (140 – 124) ! 1.8 = 88.8 lengdepoeng (distance points).

Skriv en metode distance_points som tar inn parametrene distance (hopplengde), kpoint (k-punkt) og meter_value (meterverdi), og returnerer lengdepoeng.

Løsning:

```
def distance_points(distance, kpoint, meter_value):
    return 60.0 + (distance-kpoint)*meter_value
```

b) (10 %) Et skihopp belønnes med 0–60 stilpoeng. Fem dommere gir 0–20 poeng hver.

Den laveste og den høyeste poengsummen strykes, og de tre resterende poengsummene legges sammen og utgjør hoppets stilpoeng.

Hvis et skihopp får poengsummene 17, 17.5, 17.5, 18, 19, strykes 17 og 19, og stilpoengene blir 17.5 + 17.5 + 18 = 53.

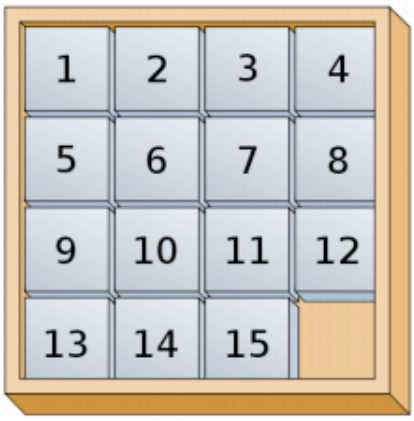
Skriv en metode style_points som tar inn en usortert liste points med de fem dommerpoengsummene, og returnerer hoppets stilpoeng.

Løsning:

```
def style_points(points):
    max = max(points)
    min = min(points)
    sum = 0
    for point in points:
        if (point != max and point != min):
            sum=sum+point;
    return sum
```

Oppgave 4

I denne oppgaven skal du programmere metoder til "15-spillet" som er illustrert i figur 1. Spillet består av et brett med 4x4 ruter med 15 brikker med tallene 1 til 15. En rute er tom, og den kan brukes til å flytte om på brikkene innbyrdes for å endre på rekkefølgen. Spillet starter ved at brikkene står i tilfeldig rekkefølge. Målet med spillet er å få brikkene i riktig rekkefølge fra 1 til 15 med den siste ruten fri slik som vist i figur 1. Spillebrettet skal i koden representeres som en 4x4-tabell med heltall som vist i figur 2.



Figur 1. Spillebrett til "15-spillet".

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	0

Figur 2. Spillebrettet representert som tabell.

I denne oppgaven er det hensiktsmessig å gjenbruke metoder du lager. Du kan bruke metoder fra andre deloppgaver selv om du ikke har klart å løse deloppgaven hvor du skal lage metoden.

a) (5 %) Skriv metoden number_in_list som tar inn et heltall, number, og en liste (endimensjonal tabell) med heltall, list. Hvis tallet number finnes i listen skal metoden returnere true, ellers false.

Løsning:

```
def number_in_list(number, list):
    if list.count(number) > 0:
        return True
    return False
```

b) (5 %) Skriv metoden random_list som tar inn et heltall, number, og returnerer en liste (endimensjonal tabell) med heltallene fra og med 1 til og med tallet number i tilfeldig rekkefølge. Bruk metoden number_in_list fra oppgave a) i løsningen av denne oppgaven.

Gjør man metodekallet random_list(15) kan for eksempel følgende tabell returneres:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	2	15	13	12	9	1	11	5	14	7	6	4	8	10

Figur 3. Tabell med 15 tall i tilfeldig rekkefølge.

Løsning:

```
def random_list(number):
    table = range(1, number+1)
    random.shuffle(table)
    return table
```

c) (5 %) Skriv metoden new_level som tar inn en liste (endimensjonal tabell), list, bestående av 15 heltall i tilfeldig rekkefølge som vist i figur 3. Metoden skal returnere en 4x4-tabell der tallene i list er satt inn fortløpende, radvis, fra rad 0, kolonne 0, til rad 3, kolonne 2. Elementet med indeks 3,3 skal ha verdien 0.

Kalles metoden new_level med listen vist i figur 3, skal den returnere tabellen vist i figur 4.

	0	1	2	3
0	3	2	15	13
1	12	9	1	11
2	5	14	7	6
3	4	8	10	0

Figur 4. Et tilfeldig spillebrett i 4x4-tabell.

Løsning:

```
def new_level(liste):
    a = []; b = []; c = []; d = []
    for x in range(4):
        a.append(liste[x])
        b.append(liste[x+4])
        c.append(liste[x+8])
        if x+12 < 15:
            d.append(liste[x+12])
        else:
            d.append(0)
    level = [a, b, c, d]
    return level
```

d) (15 %) Skriv metoden move_tile som tar inn et spillebrett, level, og en tekststreng, direction, som angir retning der "l" er venstre (left), "r" er høyre (right), "d" er ned (down), og "u" er opp (up). Metoden skal returnere et spillebrett der ruten med tallet 0 har byttet plass med tallet som befinner seg i naboruten i angitt retning hvis et slikt bytte er mulig. Hvis byttet ikke er mulig skal metoden returnere et uendret spillebrett.

I eksemplet vist i figur 4, kan ruten med tallet 0 bytte plass opp (med tallet 6) og til venstre (med tallet 10).

Merk at metoden skal være generell og fungere uansett hvor ruten med tallet 0 er plassert i tabellen. Hvis metoden move_tile kalles med tabellen vist i figur 4 og retning "u" (opp) som parametre, så vil tallet 6 bytte plass med tallet 0 i tabellen.

Løsning:

```
def move_tile(level, direction):
    x = -1
    y = -1

    #  Finn tom plass i brett (verdi 0)
    for row in level:
        if (row.count(0) > 0):
            x = row.index(0)
            y = level.index(row)
            break

    # sjekk retning og om den kan flyttes i den retningen
    if (direction == 'l' and x>0):
        level[y][x]=level[y][x-1]
        level[y][x-1]=0
    elif (direction == 'r' and x<3):
        level[y][x]=level[y][x+1]
        level[y][x+1]=0
    elif (direction == 'u' and y>0):
        level[y][x]=level[y-1][x]
        level[y-1][x]=0
    elif (direction == 'd' and y<3):
        level[y][x]=level[y+1][x]
        level[y+1][x]=0

    return level
```

e) (5 %) Skriv metoden correct_place som tar inn et spillebrett, level, og returnerer antall tall som er riktig plassert på spillebrettet. Korrekt plassering av tallene er som vist i figur 2. Kalles metoden correct_place med spillebrettet fra

figur 4, vil metoden gi svaret 2, ettersom tallene 0 og 2 er korrekt plassert på spillebrettet.

Løsning:

```
def correct_place(level):
    correct = 0
    counter = 1

    # Sjekk om tallene står på riktig plass i tabellen
    for y in range(len(level)):
        for x in range(len(level[y])):
            if (level[y][x] == counter):
                correct = correct + 1
                counter = counter + 1

    # Sjekk om 0 står på riktig plass
    if (level[3][3]==0):
        correct = correct + 1

    return correct
```

f) (5 %) Skriv metoden level_to_html som tar inn et spillebrett, level, og returnerer en tekststreng der spillebrettet er formatert som en tabell i HTML. HTML-koden skal formateres slik at tabellen vil se ut som på figur 5 hvis den skrives ut til en nettleser. Avstand fra tallene til strekene i tabellen skal være på 20 piksler.

14	9	7	8
1	10	5	4
2	11	13	3
12	15	6	0

Figur 5. Spillebrett formatert som HTML-tabell.

g) (10 %) Lag koden for å utføre følgende i et JSP-skript:

1. Opprett nødvendige variabler.
2. Fyll et spillebrett med tallene 1 til 15 i tilfeldig rekkefølge. Vi ønsker at spillebrettet skal ha minst 10 tall på riktig sted, så skriptet må fylle ut nye brett helt til et slikt spillebrett er funnet.
3. Skriv ut spillebrettet med minst 10 riktig plasserte tall til nettleseren ved hjelp av metoden level_to_html.