

Algoritmer og datastrukturer

Kondisjonstest

Python-oppgaver - **LF**

Onsdag 6. oktober 2004

Her er noen repetisjonsoppgaver i Python.

Som alltid er den beste måten å lære å programmere på å sette seg ned og programmere litt. Etter at du er ferdig med denne testen anbefaler vi at du lager noen oppgaver til deg selv og implementerer disse.

Oppgave 1 - Hva skrives ut - The basics

I denne oppgaven skal du se på bruk av alt som ble nevnt i innledningen. Programmet bare skriver ut verdien variable peker til eller verdien av konstanter.

Det vil ofte være slik at det samme blir skrevet ut flere ganger på én linje. Dette er ment for å vise at det finnes flere måter å gjøre ting på og for å få dere til å forstå hvordan ting henger sammen.

Finnt ut hva som skrives ut av de enkelte programmene, og hvorfor det er dette som skrives ut. Hvis du ikke klarer å finne ut *hva* som skrives ut, sjekk siste side og overbevis deg selv om *hvorfor* det som står der skrives ut.

Deloppgave 1a

```
# Denne oppgaven tar for seg:
# True, False, 1, 1.0, 1e0,
# int(), float(), str(), round(), min(), max()
# not, or, and, +, -, //, %, **,
# !=, ==, <, <=, >, >=,
# =, -=, +=,
# print, ", "

# Anta at "print True" skriver ut "True" og at "print 0 == 0" skriver ut "True"
# og tilsvarende for "False".
# Dette er hva som skjer i Python 2.3.
# Python 2.2 ville skrevet hhv. "name 'True' is not defined" og "1"

a = True; b = False
print a, b
print not a, not b
print a or b, a or a, b or b, a and a, b and b
print a != b, not a == b, a == b
print

a = 1; b = 0; c = 1
print a, b, c
print a < b, a < c, b < c
print a > b, a > c, b > c
print a <= b, a <= c, b <= c
print a >= b, a >= c, b >= c
print a == b, a == c, b == c
print str(a) + " + " + str(b) + " + " + str(c) + " = " + str(a+b+c)
print

d, a = 0, a
b = a + c
b -= 1
d, a = a, d
c += b
print a, b, c, d
print 2**2, c**c, 2**(b+d), (b+d)**(c**(c**a))

print float(2) - 1, 2.0 - 1, 2e0 - 1, .2e1 - 1
print 2 - int(1.0), 2 - 1, int(2e0) - 1, int(.2e1 - 1)
print 5 // 2, int(5.0 / 2),
print int(round(5.0 // 2)), int(round(5.0 / 2) - 1), int(round((5 - (5 % 2)) / 2))
print 5.0 / 2, float(5) / 2, 2.5e0, .25e1
print

print min(1, 2), max(1,2), max(min(2, 4), min(1, 3))
```

```
print min(1, min(2, 3)), min(1, 2, 3), min([1, 2, 3]), min([1])
# min(1) og min([1]) er ikke lovlige (og gir heller ingen mening)
print
```

Deloppgave 1b

```
# Denne oppgaven tar for seg:
# ', "', (), [], [:],
# in, not in,
# append(), pop(),

# Anta at "in" og "not in" returnerer "True" eller "False"

a = [1, 2]
b = [3, 4]
c = a + b
d = c
print a, b, c, d
print 1 in a, 3 not in a, not 3 in a

print a, a[0:1] + a[1:2], a[0:2], [a[0], a[1]], a[:]
print a[0] + a[1], a[0:1][0] + a[1:2][0]
c += [5]
c.append(6)
c[0] = 10
print a, b, c, d
c = a + b
print c, d
d = c[:]
print d.pop(0)
d.insert(0, 1)
c[2] = 33
print c, d
print

a = [[1, 2], [3, 4]]
a.append([5, 6])
b = [0] * 6
c = range(0, 6)
for i in c:
    b[i] = a[i / 2][i % 2]
print a, b, c
print

a, b, c = 1, 2, 3
(a, b, c) = (b, c, a)
print a, b, c
(b, c, a) = a, b, c
d = (a, b, c)
print d
a, b = 4, 5
print d
(a, b) = [1, 2], [3, 4]
d = a, b
print d
a[0] = 11; a[1] = 22
b = [33, 44]
print d
print
```

Deloppgave 1c

```
# Denne oppgaven tar for seg:
# {}, []
# {}.items(), {}.values(), {}.keys(), {}.clear()
# in, not in,
```

```
# Anta at "in" og "not in" returnerer "True" eller "False"
```

```
d = {}
d[1] = 'hei'
d['per'] = 13.0
print 1 in d, 'hei' in d
d[1000000] = None
d[1.13] = False
print d.keys() # alle noekler
print d.values() # alle verdier
print d.items() # alle par av verdier og noekler
```

```
print 1000000 in d
d.pop(1000000)
print 1000000 in d
```

```
for k in d:
    print 'key:', k, ' value:', d[k]
```

```
d = {'aoeu': 1, 'snth': 2, 1: 3}
```

```
for k in d:
    print 'key:', k, ' value:', d[k]
```

Deloppgave 1d

```
# Denne oppgaven tar for seg:
# ', ', '"', '[:]', # join(), strip(), split(),
# \ # print x, y + z

a = 'a'; b = "c"; c = 'g'; d = "t"
dna = a + b + c + d
print "a" + "b" + 'c' + 'd',
print a + b + c + d,
print "acgt" + a + b + c + d,
print
b = 2
print "DNA-strenger representeres vanligvis vha. " + str(b**2) + \
    " bokstaver,", 'ac' + c + d
print "DNA-strenger representeres vanligvis vha.", b**2, \
    "bokstaver, " + dna
print 'abc', 'a' + 'b' + 'c'
print "[" + 'abc' + "]"", ['abc']
print

print len(dna)
print dna, dna[0:len(dna)]
print dna[0] + dna[1:], dna[:len(dna)-1] + dna[len(dna) - 1]
dna += dna
print len(dna)
print

dnospace = "a c g t"
dnakomma = "a,c,g,t"
print [dna[0], dna[1], dna[2], dna[3]], dnospace.split(), dnakomma.split(',')
print

a = " a "; b = " c "; c = " g "; d = " t "
print ''.join([a.strip(), b.strip(), c.strip(), d.strip()]), \
    ''.join([a[1], b[1], c[1], d[1]]), \
    a[1:2] + b[1:2] + c[1:2] + d[1:2], \
    ''.join([i[1] for i in a, b, c, d])

letters = a + b + c + d
print [letters, letters.split(), letters.split('c')]
print
```

Oppgave 2 - Beware of Bugs

I hver deloppgave er det gitt et “problem” og flere små programmer. Det er nøyaktig ett riktig program i hver deloppgave. De andre programmene inneholder bugs.

Finn alle bugs (marker relevante deler av programmene). Forstå hvilke feil bugene fører til og hvorfor. Forstå hvordan de riktige programmene fungerer.

Det er meningen at vanskelighetsgraden på deloppgavene skal være økende.

Deloppgave 2a

Skriv ut tallene fra og med 1 til og med 10.

Program 1:

```
for i in range(1, 10):  
    print i
```

Program 2:

```
for i in range(1, 11):  
    print i
```

Program 3:

```
for i in range(0, 10):  
    print i
```

Program 4:

```
for i in range(0, 11):  
    print i
```

Program 5:

```
print 1; print 2; print 3; print 4; print 5; print 7; print 8; print 9; print 10
```

Program 1: Skriver ut tallene fra og med 1 til og med 9.

Program 2: Riktig.

Program 3: Skriver ut tallene fra og med 0 til og med 9.

Program 4: Skriver ut tallene fra og med 0 til og med 10.

Program 5: Skriver ikke ut 6.

Deloppgave 2b

Kopier en liste A

Program 1:

```
B = A[0:len(A)-1]
```

Program 2:

```
B = A[0:len(A)-1]
```

Program 3:

```
for b in B:  
    B[b] = A[b]
```

Program 4:

```
for a in A:  
    B[a] = A[a]
```

Program 5:

```
for i in len(A):
    B[i] = A[i]
```

Program 6:

```
for i in range(0, len(A)):
    B[i] = A[i]
```

Program 7:

```
B = A[:]
```

Program 1: Prøver å trekke en skalar fra en liste.
Program 2: Mangler siste element.
Program 3: Itererer over elementene i B og prøver å bruke disse som indekser.
Program 4: Itererer over elementene i A og prøver å bruke disse som indekser.
Program 5: Kan ikke iterere over en skalar (`len(A)`) og B har ikke nødvendigvis riktig lengde.
Program 6: B har ikke nødvendigvis riktig lengde.
Program 7: Riktig.

Deloppgave 2c

Finn den minste verdien i en liste A.

Program 1:

```
minste = float(1E1000) # uendelig
for i in range(len(A)):
    if a < minste:
        minste = i
```

Program 2:

```
minste = float(1E1000)
for i in range(len(A)):
    if i < minste:
        minste = a
```

Program 3:

```
minste = float(1E1000)
for a in A:
    if a < minste:
        minste = a
```

Program 4:

```
for a in A:
    minste = float(1E1000)
    if a < minste:
        minste = a
```

Program 5:

```
minste = float(1E1000)
for a in A:
    if a < minste:
        a = minste
```

Program 1: Setter `minste` lik `i`, som er en indeks, og ikke en verdi.
Program 2: Sjekker størrelsen på `i`, som er en indeks, og ikke en verdi.
Program 3: Riktig.
Program 4: Setter startverdi for `minste` for hver runde i løkken.
Program 5: Setter `a` i stedet for `minste`.

Deloppgave 2d

Finn gjennomsnittet av tallene i en liste A.

Program 1:

```
for a in A:
    sum = a
avg = sum / (len(A) - 1)
```

Program 2:

```
if len(A) == 0:
    raise Exception("ops")
else:
    sum = 0.0
    for a in A:
        sum += a
    avg = sum / len(A)
```

Program 3:

```
sum = 0.0
for a in A:
    sum += a
avg = sum / len(A) - 1
```

Program 4:

```
sum = 0.0
for a in A:
    sum += a
avg = sum / len(A) - 1
```

Program 5:

```
sum = 0.0
for a in A:
    sum += a
avg = sum / (len(A) - 1)
```

Program 6:

```
sum = 0.0
for a in A:
    sum = a
avg = sum / (len(A) - 1)
```

Program 1: Deler på $\text{len}(A) - 1$ i stedet for $\text{len}(A)$. Initialiserer ikke `sum` til 0.
Program 2: Riktig. Gjennomsnittet av ingen tall er ikke definert.
Program 3: Trekker 1 fra gjennomsnittet.
Program 4: Likt program 3.
Program 5: Deler på $\text{len}(A) - 1$ i stedet for $\text{len}(A)$.
Program 6: Deler på $\text{len}(A) - 1$ i stedet for $\text{len}(A)$. `sum` blir lik siste tallet i A.

Oppgave 3 - Hva skjer her?

I denne oppgaven skal du prøve å finne ut hva forskjellige programmer gjør.

Deloppgave 3a

Hva skriver programmet ut? Hva er det funksjonen `f` gjør?

```
def f(t):
    for i in range(2, t):
        if t % i == 0:
            return 0
    return 1

print f(23), f(24), f(25)
```

f sjekker om et tall t er et primtall.
Programmet skriver ut: 1 0 0

Deloppgave 3b

Hva skriver programmet ut? Hva er det funksjonen g gjør?

```
def g(M, N):
    im = len(M)
    jm = len(N[0])
    km = len(M[0])
    if km != len(N):
        raise Exception()
    K = [None] * im
    for i in range(im):
        K[i] = [0] * jm
    for i in range(im):
        for j in range(jm):
            s = 0
            for k in range(km):
                s += M[i][k] * N[k][j]
            K[i][j] = s
    return K

A = [[2, 3, 4], [5, 6, 7]]
B = [[1, 2], [3, 4], [5, 6]]
print g(A, B)
```

g finner matrise-matrise-produktet av de to innparametrene sine.
Programmet skriver ut $[[31, 40], [58, 76]]$.

Deloppgave 3c

Hva skriver programmet ut? Hva er det funksjonen h gjør?

```
def h(T, P):
    n = len(T)
    m = len(P)
    if m > n:
        return -1
    for i in range(n - m):
        ok = 1
        for j in range(m):
            if T[i + j] != P[j]:
                ok = 0
                break
        if ok:
            return i
    return -1

print h('aoeuvoffaoeu', 'voff'),
print h([1, 1, 2, 3, 5, 8, 13], [5, 8]),
print h('nsatoeuhsnaoteuhsnaoteuh', 'mjau')
```

h sjekker om en sekvens er en sub-sekvens av en annen. Hvis den er det, returnerer h posisjonen sub-sekvensen starter i. Hvis ikke, returnerer h ut -1 .
Programmet skriver ut 4 4 -1 .

Deloppgave 3d

Hva skriver programmet ut? Hva er det funksjonen a gjør?

```
def a(NL, x, y):
    n = len(NL)
    queue = [x]
    onpath = [False] * n
    onpath[x] = True
    while queue:
        curr = queue.pop(0)
        for next in NL[curr]:
            if not onpath[next]:
                queue.append(next)
                onpath[next] = True
    entries = [0] * n
    for u in range(n):
        if onpath[u]:
            for v in NL[u]:
                if onpath[v]:
                    entries[v] += 1
    ways = [0] * n
    ways[x] = 1
    queue = [x]
    while queue:
        curr = queue.pop(0)
        for next in NL[curr]:
            if onpath[next]:
                ways[next] += ways[curr]
                entries[next] -= 1
                if entries[next] == 0:
                    queue.append(next)
    return ways[y]

NL = [[1, 2], [3], [3], [4, 5], [6], [6], []]

print b(NL, 0, 6)
```

a finner antall stier fra noden x til noden y i en DAG (Directed Acyclic Graph / rettet asyklisk graf) definert av nabolistene NL.
Programmet skriver ut 4.

Oppgave 4 - Finn bugs!

I denne oppgaven skal du finne feilene i litt større programmer. Du får oppgitt en kort beskrivelse av hvordan programmet skal fungere og ett program, per deloppgave. Endre programmet så det fungerer som det skal.

Deloppgave 4a

Dette programmet skal finne det største tallet i en liste. Hvert enkelt element i listen kan være en ny liste eller ett tall. (Programmet skal søke rekursivt i listene for å finne det største tallet.)

```

def recmax(X):
    if X.__class__ == [].__class__:
        return X
    else:
        m = 99999
        for x in X:
            r = recmax(x)
            if x > m:
                return x
        return m

print recmax([1,2,[2,[4]]],1)

```

- Casene i if-blokkene skulle vært byttet om. Prøver nå å se på elementene i et allerede atomisk element.
- Når vi starter er m positiv i stedet for negativ.
- Hvis vi bytter fortegn på m, er det fremdeles et alt for høyt tall. Programmet blir feil hvis alle tallene våre er lavere enn -99999 .
- Vi sammenligner m med x i stedet for r. Men x kan være en liste.
- Vi returnerer med en gang vi finner noe som er større enn m, i stedet for å se igjennom resten av listen.
- Funksjonen kalles med to argumenter, men tar bare ett.

```

def recmax(X):
    if X.__class__ == [].__class__:
        m = float('-Inf')
        for x in X:
            r = recmax(x)
            if r > m:
                m = r
        return m
    else:
        return X

print recmax([1,2,[2,[4]])

```

Deloppgave 4b

Dette programmet inneholder en klasse for rettede grafer og en for urettede.

```

class Graf:

    NM = None

    def __init__(self, n):
        self.NM = [[None] * n] * n

    def sett_kant(self, u, v, c):
        self.NM[u][v] = c

    def antall_noder(self):
        return len(self.NM)

```

```

def antall_kanter(self):
    kanter = 0
    for u in self.antall_noder():
        for v in self.antall_noder():
            if self.NM[u][v] != None:
                kanter += 1
    return kanter

def kant_kost(self, u, v):
    return self.NM[u][v]

class UrettetGraf:

    def legg_til_kant(self, u, v):
        self.sett_kant(u, v, 1)

    def fjern_kant(self, u, v):
        self.sett_kant(u, v, 0)

    def finnes_kant(self, u, v):
        return Graf.kant_kost(self, u, v) != None

    def kant_kost(self, u, v):
        raise Exception("Ikke definert")

```

- `__init__` lager ikke en liste av lister, men en liste som inneholder flere referanser til samme listen. Prøv for eksempel koden `A = [[0] * 4] * 4; A[1][1] = 13; print A`
- I `antall_kanter` er det to for-løkker som prøver å iterere over et tall.
- Det er ikke spesifisert at `UrettetGraf` skal arve fra `Graf`.
- `legg_til_kant` og `fjern_kant` setter `NM[u][v]`, men ikke `NM[v][u]`. Resten av `UrettetGraf` går ut fra at `NM` er symmetrisk.
- `fjern_kant` setter kantkost til 0 i stedet for `None`, som brukes til å representere ingen kant i resten av koden.
- `antall_kanter` definert i `Graf` er ikke riktig. Antall kanter i `UrettetGraf` er halvparten av antall elementer forskjellig fra `None` i `NM[v][u]`.

```
class Graf:
    NM = None

    def __init__(self, n):
        self.NM = [None] * n
        for i in range(n):
            self.NM[i] = [None] * n

    def sett_kant(self, u, v, c):
        self.NM[u][v] = c

    def antall_noder(self):
        return len(self.NM)

    def antall_kanter(self):
        kanter = 0
        for u in range(self.antall_noder()):
            for v in range(self.antall_noder()):
                if self.NM[u][v] != None:
                    kanter += 1
        return kanter

    def kant_kost(self, u, v):
        return self.NM[u][v]

class UrettetGraf(Graf):

    def legg_til_kant(self, u, v):
        self.sett_kant(u, v, 1)
        self.sett_kant(v, u, 1)

    def fjern_kant(self, u, v):
        self.sett_kant(u, v, None)
        self.sett_kant(v, u, None)

    def antall_kanter(self):
        return Graf.antall_kanter(self) // 2

    def finnes_kant(self, u, v):
        return Graf.kant_kost(self, u, v) != None

    def kant_kost(self, u, v):
        raise Exception("Ikke definert")
```

Deloppgave 4c

Dette programmet skal sortere tallene i en liste.

```
def sort(L, s, e):
    if s != e:
        p = L[e]
        i = s
        for j in range(s, e):
            if L[j] < p:
                L[i], L[j] = L[j], L[i]
                i = i + 1
        L[i], L[e] = L[i], L[e]
        sort(L, s, i-1)
        sort(L, i+1, e)
    else:
        return L

A = [0,8,6,5,54,7,8,9,9,7,6,54,4,6,8,9,8,7,65,4,4,7,8]
sort(A, 0, len(A))
print A
```

- `e` er en inklusiv øvre grense, men `sort` får inn `e == len(L)`. Kallet skulle vært `sort(A, 0, len(A) - 1)`.
- `s` kan bli større enn `e`, og `s != e` treffer fremdeles på. Skulle vært `s < e`.
- `return L` gir ingen mening, da returverdien ikke sendes videre oppover.
- `L[i], L[e] = L[i], L[e]` gjør ingenting, skulle vært `L[i], L[e] = L[e], L[i]`.

```
def sort(L, s, e):
    if s < e:
        p = L[e]
        i = s
        for j in range(s, e):
            if L[j] < p:
                L[i], L[j] = L[j], L[i]
                i = i + 1
        L[i], L[e] = L[e], L[i]
        sort(L, s, i-1)
        sort(L, i+1, e)

A = [0,8,6,5,54,7,8,9,9,7,6,54,4,6,8,9,8,7,65,4,4,7,8]
sort(A, 0, len(A) - 1)
print A
```

Oppgave 5 - Skriv programmer

I denne oppgaven skal du skrive programmer selv.

For hver oppgave, skriv også kode som tester programmet.

Deloppgave 5a

Skriv en funksjon som finner min, maks, sum og snitt av en liste, og returnerer en tuppel som inneholder disse verdiene.

Gjøre-mest-mulig-selv:

```
def funksjon(Liste):
    n = len(Liste)
    if n == 0:
        return (None, None, 0, None)

    min = Liste.pop(0)
    maks = min
    sum = min

    for element in Liste:
        if element < min:
            min = element
        elif element > maks:
            maks = element
        sum += element

    return (min, maks, sum, float(sum) / n)

print funksjon([1, 2, 3, 4])
print funksjon([4, 3, 2, 1])
print funksjon([-10, 10])
print funksjon([])
```

Gjøre-minst-mulig-selv:

```
def funksjon(Liste):
    n = len(Liste)
    if n == 0:
        return (None, None, 0, None)

    _min = min(Liste)
    maks = max(Liste)
    _sum = sum(Liste)
    return (_min, maks, _sum, float(_sum) / n)

print funksjon([1, 2, 3, 4])
print funksjon([4, 3, 2, 1])
print funksjon([-10, 10])
print funksjon([])
```

Deloppgave 5b

Skriv et program som finner prikk-produktet av to vektorer.

```

def prikkprodukt(V1, V2):
    if len(V1) != len(V2):
        return None

    pp = 0
    for i in range(len(V1)):
        pp += V1[i] * V2[i]

    return pp

print prikkprodukt([1, 2], [1])
print prikkprodukt([1, 2], [2, 1])
print prikkprodukt([0, 3], [3, 3])

```

Deloppgave 5c

Skriv et program som sjekker om et ord er et anagram (feks “agnes i senga”).

Gjøre-mest-mulig-selv:

```

def anagram(Ord):
    if len(Ord) == 0:
        return True

    i = 0
    j = len(Ord) - 1
    while i <= j:
        if Ord[i] != Ord[j]:
            return False
        i += 1
        j -= 1

    return True

print anagram("agnes i senga")
print anagram("aba")
print anagram("abba")
print anagram("")
print anagram("abca")
print anagram("accb")
print anagram("ab")

```

Gjøre-minst-mulig-selv:

```

def anagram(Ord):
    Ord1 = [c for c in Ord]
    Ord2 = Ord1[:]
    Ord2.reverse()
    return Ord1 == Ord2

print anagram("agnes i senga")
print anagram("aba")
print anagram("abba")
print anagram("")
print anagram("abca")
print anagram("accb")
print anagram("ab")

```

Deloppgave 5d

Skriv et program som finner ut hvor mange ganger hvert enkelt ord i en tekst er repetert. Bruk dictionaries i Python.

```
def repetisjoner(Tekst):
    teller = {}
    for Ord in Tekst.split():
        if Ord in teller:
            teller[Ord] += 1
        else:
            teller[Ord] = 1

    stat = teller.items()
    stat.sort()
    return stat

print repetisjoner("tre en to to tre tre")
print repetisjoner("");
```

Deloppgave 5e

Skriv et program som finner alle nodene i en graf som kan nås fra alle de andre.

Floyds algoritme. Nabomatrise fra standard inn (stdin):

```
from sys import stdin
Inf = float('Inf')

# forventer input paa formen:
# 0 1 0
# 0 0 1
# 1 0 0
# som tolkes som at vi har folgende kanter (0-indeksert):
# 0 -> 1
# 1 -> 2
# 2 -> 0

# innlesning
linjer = stdin.readlines()
n = len(linjer)
D = [None] * n
for i in range(0, n):
    D[i] = [Inf] * n
    for j in range(0, n):
        temp = int(linjer[i][j*2])
        if temp > 0:
            D[i][j] = temp
        elif i == j:
            D[i][j] = 0 # kan naa seg selv

# Floyd
for k in range(0, n):
    for i in range(0, n):
        for j in range(0, n):
            D[i][j] = min(D[i][j], D[i][k] + D[k][j])

# Losning
print "Naas av alle:",
for j in range(0, n):
    naabar = True
    for i in range(0, n):
        if D[i][j] == Inf:
            naabar = False
            break
    if naabar:
        print j,
print
```

Deloppgave 5f

Skriv et program som finner alle nodene i en graf som kan nå alle de andre.

Bredde-først-søk. Objekter (noder) med referanser (kanter) = nabolister:

```
class Node(str):
    def __init__(self, id):
        self.naboer = []
        str.__init__(id)

class Graf:
    def __init__(self):
        self.nodedict = {}

    def legg_til_node(self, id):
        self.nodedict[id] = Node(id)

    def legg_til_kant(self, fra_id, til_id):
        self.nodedict[fra_id].naboer.append(self.nodedict[til_id])

    def __str__(self):
        ret = " "
        for til_node in self.noder():
            ret += til_node + " "
        ret += "\n"
        for fra_node in self.noder():
            ret += fra_node + " "
            for til_node in self.noder():
                if til_node in fra_node.naboer:
                    ret += "1 "
                else:
                    ret += "0 "
            ret += "\n"

        return ret

    def noder(self):
        temp = self.nodedict.values()
        temp.sort()
        return temp

def losning(noder):
    print "Naar alle:",
    for node in noder:
        for temp in noder:
            temp.besokt = False
        if tellNaabare(node) == len(noder):
            print node,
    print

def tellNaabare(node):
    node.besokt = True
    queue = [node]
    naabare_teller = 1
    while len(queue) > 0:
        node = queue.pop(0)
        for nabo in node.naboer:
            if not nabo.besokt:
                nabo.besokt = True
                queue.append(nabo)
                naabare_teller += 1
    return naabare_teller
```

Graf-klassen er ikke nødvendig, men gjør det kanskje litt lettere å lese testkoden.

Testkode:

```
g = Graf()
g.legg_til_node('a')
g.legg_til_node('b')
g.legg_til_node('c')
g.legg_til_node('d')
g.legg_til_kant('a', 'b');
g.legg_til_kant('a', 'c');
g.legg_til_kant('b', 'd');
g.legg_til_kant('c', 'a');
losning(g.noder())
print g

g2 = Graf()
g2.noder = g.noder # samme node-objekter som g
for node in g2.noder():
    node.naboer = g2.noder()
losning(g2.noder())
print g2

g3 = Graf()
g3.noder = g.noder # samme node-objekter som g og g2
for node in g3.noder():
    node.naboer = []
losning(g3.noder())
print g3
```

Oppgave 1 - Hva skrives ut

Her lister vi utskriftene fra Python-programmene i oppgaven med samme tittel. Ikke noe tjuvtitting :)

Deloppgave 1a

```
True False
False True
True True False True False
True True False
```

```
1 0 1
False False True
True False False
False True True
True True False
False True False
1 + 0 + 1 = 2
```

```
0 1 2 1
4 4 4 4
1.0 1.0 1.0 1.0
1 1 1 1
2 2 2 2
2.5 2.5 2.5 2.5
```

```
1 2 2
1 1 1 1
```

Deloppgave 1b

```
[1, 2] [3, 4] [1, 2, 3, 4] [1, 2, 3, 4]
True True True
[1, 2] [1, 2] [1, 2] [1, 2] [1, 2]
3 3
[1, 2] [3, 4] [10, 2, 3, 4, 5, 6] [10, 2, 3, 4, 5, 6]
[1, 2, 3, 4] [10, 2, 3, 4, 5, 6]
1
[1, 2, 33, 4] [1, 2, 3, 4]

[[1, 2], [3, 4], [5, 6]] [1, 2, 3, 4, 5, 6] [0, 1, 2, 3, 4, 5]

2 3 1
(1, 2, 3)
(1, 2, 3)
([1, 2], [3, 4])
([11, 22], [3, 4])
```

Deloppgave 1c

```
True False
[1000000, 1, 1.1299999999999999, 'per']
[None, 'hei', False, 13.0]
[(1000000, None), (1, 'hei'), (1.1299999999999999, False), ('per', 13.0)]
True
False
key: 1 value: hei
key: 1.13 value: False
key: per value: 13.0
key: 1 value: 3
key: aoeu value: 1
key: snth value: 2
```

Deloppgave 1d

```
abcd acgt acgtacgt
DNA-strenger representeres vanligvis vha. 4 bokstaver, acgt
DNA-strenger representeres vanligvis vha. 4 bokstaver, acgt
abc abc
['abc'] ['abc']

4
acgt acgt
acgt acgt
8

['a', 'c', 'g', 't'] ['a', 'c', 'g', 't'] ['a', 'c', 'g', 't']

acgt acgt acgt acgt
['a c g t ', ['a', 'c', 'g', 't'], [' a ', ' g t ']]
```