MATEMATISKA INSTITUTIONEN
STOCKHOLMS UNIVERSITET
Avd. Beräkningsmatematik
Kursledare: Anders Mörtberg
Examinator: Lars Arvestad

Tentamensskrivning i
DA2004 Programmeringsteknik för matematiker
7.5 hp
2019-12-18

- **Skriv tydligt:** svårlästa svar riskerar 0 poäng.
- Skriv bara på en sida av varje papper!
- Del A har flervalsfrågor där *minst* ett svarsalternativ är korrekt. Om man svarar fel eller inte har exakt antal rätta alternativ får man noll poäng på frågan.
- Man måste bli godkänd på del A (5 rätt på 10 frågor) för att del B ska rättas.
- Inga externa bibliotek får användas om de inte nämns i uppgiften, man får dock använda inbyggda funktioner som `len` och `range`.
- **Hjälpmedel:** Ett A4 med så mycket information du vill. Du får skriva på båda sidorna.
- **Betygsgränser:** E: 10, D: 12, C: 14, B: 16, A: 18, av maximala 20.

Del A: flervalsfrågor

*Vänligen samla svaren på del A på **ett** svarspapper.*

1. Vad skrivs ut om vi kör koden till höger?

```
xs = [1, 2, 3]
ys = [4, 5, 6]
zs = []

for i in range(len(xs)):
    x = xs[i]
    y = ys[i]
    zs.append(x + y)

print(zs)
```

a. `[9, 7, 5]`
b. `[5, 7, 9]`
c. `[ [1, 4], [2, 5], [3, 6] ]`
d. `[1, 2, 3, 4, 5, 6]`
e. Inget av alternativen a.-d. då ett särfall (exception) lyfts.

2. Givet `xs = [1, 2, 3]`, vilket eller vilka av följande `print`-uttryck skriver ut listan `[1, 2]`?

a. `print(xs[0:len(xs)-1])`
b. `print(xs[:])`
c. `print(xs[:-1])`
d. `print(xs.pop())`
e. `print(xs[0] + xs[1])`

3. Vad skrivs ut om vi kör koden nedan?

```
p = (2, 4)
p[0] = 3
print(p)
```

a. `(3, 4)`
b. `[3, 4]`
c. `(2, 4)`
d. `p`
e. Inget av alternativen a.-d. då ett särfall lyfts.

4. Vad skrivs ut om vi kör koden till höger?

```
x = 10
y = 20

def f(x):
    y = 5
    return x + y

print(f(2))
```

a. 7
b. 15
c. 22
d. 30
e. Inget av alternativen a.-d. då ett särfall lyfts.

5. Om en fil börjar med `from m import f as foo` hur använder man funktionen `f` från modulen `m`?

a. `f`
b. `m.f`
c. `foo.f`
d. `foo`
e. `m.foo`

6. Vilka påståenden är sanna för Python 3?

a. Listor är muterbara.
b. Listor kan vara nycklar i uppslagstabeller (`dict`).
c. Strängar är listor av bokstäver (`char`).
d. Strängar kan vara nycklar i uppslagstabeller.
e. Listor får vara värden i uppslagstabeller.

7. Vilken eller vilka boolska variabeltilldelningar gör att uttrycket

```
(x == y) and (x or y)
```

evaluerar till `True`?

a. `x = False, y = False`
b. `x = False, y = True`
c. `x = True, y = False`
d. `x = True, y = True`
e. Ingen, det evaluerar alltid till `False`.

8. Vad händer om man kör följande kod och användaren skriver in `42`?

```
while True:

    x = input("Write a number: ")

    if x % 2 == 0:
        print("Even")
    else:
        print ("Odd")
```

a. `Even` skrivs ut och loopen avslutas.
b. `Even` skrivs ut och loopen börjar om.
c. `Odd` skrivs ut och loopen avslutas.
d. `Odd` skrivs ut och loopen börjar om.
e. Inget av alternativen a.-d. då ett särfall lyfts.

9. Vad skrivs ut av följande kod?

```
foo = lambda f, x: f(f(x))

print(foo(lambda x: x + x, "hej"))
```

a. `hej`
b. `hejhejhej`
c. `hejhejhejhejhej`
d. `hejhejhejhej`
e. Inget av alternativen a.-d. då ett särfall lyfts.

10. Vad blir värdet på `xs` efter att koden till höger har körts?

a. `[0, 0]`
b. `[3, 2]`
c. `[[0, 1, 2], [0, 1]]`
d. `[0, 1, 2, 3, 4]`
e. `[0, 1, 2, 0, 1]`

```
xs = []
for x in ['hej', 'du']:
    i = 0
    for y in x:
        xs.append(i)
        i += 1
```

Del B: kodfrågor

Vänligen använd ett papper till varje fråga i del B.

11. Skriv funktionen `repeatString(n,s)` som tar in ett tal `n` och en sträng `s`, och returnerar en sträng där `s` har repeterats `n` gånger.

Exempelanvändning:

```
>>> print(repeatString(3,'hej'))
hejhejhej
>>> print(repeatString(0,'hej'))

>>> print(repeatString(5,'du'))
dudududududu
```

12. Skriv funktionen `printRightAligned(xs)` som tar in en lista med strängar och skriver ut strängarna ställda till höger istället för vänster:

Exempelanvändning:

```
>>> printRightAligned(['Lycka','till','på','tentan!'])
  Lycka
   till
     på
tentan!
```

Tips: beräkna längden på den längsta strängen och använd det talet för att beräkna hur många mellanslag som ska sättas in framför alla kortare ord. Ni får använda `repeatString` från uppgiften ovan även om ni inte löst den uppgiften.

Obs: Det längsta ordet ska inte vara indraget alls.

13. Skriv `divisors(n)` som tar in ett heltal och returnerar en lista med alla delare mellan `1` och `n - 1`.

```
>>> print(divisors(6))
[1, 2, 3]
>>> print(divisors(12))
[1, 2, 3, 4, 6]
```

14. Ett *perfekt* tal är ett tal som är lika med summan av dess delare (bortsett från sig själv). Använd en listomfattning och `divisors` från förra uppgiften för att beräkna alla perfekta tal mellan 1 och 1000.

Exempel: talet `6` är perfekt då dess delare är `1`, `2` och `3` samt `1 + 2 + 3 = 6`. Talet `12` är inte perfekt då dess delare är `1`, `2`, `3`, `4` och `6` men `1 + 2 + 3 + 4 + 6 = 15`.

Obs: ni får endast använda en listomfattning, så varken `for`, `while` och rekursion är tillåtet. Ni behöver inte skriva en funktion utan det räcker med koden för listomfattningen. Ni får dock använda den inbyggda funtionen `sum` för att beräkna summan av elementen i en lista.

15. Betrakta koden nedan.

```
def function(n):
    counter = 0

    while (n > 0):
        n //= 10
        counter += 1

    return counter
```

Beskriv i ord vad funktionen gör.

Tips: Vad returnerar `function(0)`, `function(1)`, `function(523)`, `function(123456)`?

16. Skriv en klass `Poly` för polynom som representeras på samma sätt som i labbarna, dvs som listor av koefficienter utan nollor i slutet. Denna klass ska ha följande metoder:

- En konstruktormetod `__init__` som tar in en lista med tal och använder `drop_zeroes` från labben för att ta bort nollor i slutet.
- En metod `getList` som returnerar listan av koefficienter från ett polynom.

Exempelanvändning:

```
>>> p0 = Poly([2, 0, 1, 0])
>>> q0 = Poly([0, 0, 0])
>>> print(p0.getList())
[2, 0, 1]
>>> print(q0.getList())
[]
```

Obs: Ni behöver *inte* skriva `drop_zeroes` igen på tentan utan kan bara använda funktionen från labben.

17. Lägg till en metod `degree` till `Poly` för att beräkna graden på polynom. Då graden av nollpolynomet (representerad med tomma listan) är odefinierad ska ett särfall lyftas om användaren försöker beräkna dess grad.

Exempelanvändning med `p0` och `q0` som ovan:

```
>>> print(p0.degree())
2
>>> print(q0.degree())
Traceback (most recent call last):
...
ValueError: Degree of zero polynomial is undefined
```

Så för `q0.degree()` lyfts särfallet `ValueError`.

18. Skriv en högre ordnings funktion `zipwith(f,xs,ys)` som tar in en funktion `f` och två listor `xs` och `ys`, och sedan applicerar `f` på alla par av värden från `xs` och `ys`.

Exempelanvändning:

```
>>> print(zipwith(lambda x, y: (x, y), [1, 2, 3], [4, 5, 6]))
[(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)]
>>> print(zipwith(lambda x, y: x + y, [1, 2, 3], [4, 5, 6]))
[5, 6, 7, 6, 7, 8, 7, 8, 9]
>>> print(zipwith(lambda x, y: x - y, [1, 2, 3], [4, 5, 6]))
[-3, -4, -5, -2, -3, -4, -1, -2, -3]
```

Obs: Ni får inte använda någon av `zip` funktionerna ni skrev i labb 6. Lösningen på denna uppgift bör dock vara lik vad ni gjorde i labben.

19. Skriv en *rekursiv* funktion `take(n,xs)` som tar ut de `n` första elementen ur listan `xs`. Om `n` är större än längden på `xs` så ska hela `xs` returneras.

Exempelanvändning:

```
>>> print(take(3, [4, 2, 9, 8, 9]))
[4, 2, 9]
>>> print(take(3, [4, 2]))
[4, 2]
>>> print(take(0, [4, 2]))
[]
```

Obs: er lösning måste vara rekursiv och varken `for`, `while` eller listomfattningar får användas.

20. Sista uppgiften på den här tentan är att skriva *FizzBuzz*.[1] Det är ett litet program som skriver ut alla tal från `1` till `100`, men för multipler av `3` skriver det istället ut `Fizz`, för multipler av `5` skriver det istället ut `Buzz`, och för multipler av både `3` och `5` skriver det istället ut `FizzBuzz`.

De första femton utskrifterna ska alltså vara:

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
...
```

[1] Att skriva *FizzBuzz* är en vanlig uppgift i den praktiska delen av arbetsintervjuer för programmerare.