Facit och kommentarer till tentamen 2021-08-23 i DA2005/DA2004

Del A: flervalsfrågor (1p per fråga)

1. *B*
2. *C*
3. *C*
4. *E*
5. *E*
6. *B*
7. *B, C, D*
8. *A*

Del B: kodfrågor (2p per fråga)

9. Möjlig lösning:

```
def f(d):
    for k, v in d.items():
        print(k, len(v))
```

   eller

```
def f(d):
  for k in d:
      print(k, len(d[k]))
```

10. Möjlig lösning:

```
def spell_check_helper(word, bigrams):
    if len(word) < 2:
        return False
    for i in range(len(word) - 1): # Always 2 chars at end!
        bigram = word[i : i+2]
        if bigram not in bigrams:
            return False
    return True
```

    Notera att vi utformar loopen så att det alltid är bokstavspar som plockas ut. Om man hittar ett bokstavspar som inte finns i uppslagstabellen kan man returnera False direkt.

11. Möjlig lösning:

```
class Bug:
    def __init__(self):
        self.step_size = 0.1
        self.position_x = 0
        self.position_y = 0
        self.n_steps = 0

    def step(self):
        direction = random.choice(['left', 'right', 'up', 'down'])
        if direction == 'left':
            self.position_x -= self.step_size
        elif direction == 'right':
            self.position_x += self.step_size
        elif direction == 'up':
            self.position_y += self.step_size
        else:
            self.position_y -= self.step_size
        self.n_steps += 1
```

```
    def walk(self, n_steps):
        for i in range(n_steps):
            self.step()

    def distance_from_origin(self):
        return math.sqrt(self.position_x ** 2 + self.position_y ** 2)
```

Man behöver inte ha ett attribut `step_size`, men det är bra när man vet att man kommer att upprepa ett värde (här: 0.1) på flera ställen. Även när man inte upprepar en konstant så blir koden extra tydlig med en variabel för konstanten.

12. Möjlig lösning:

```
class Harkrank(Bug):
    def __init__(self):
        super().__init__()
        self.step_size = 1

    def walk(self, n_steps):
        if n_steps > 50:
            n_steps = 50
        super().walk(n_steps)
```

Lägg märke till att superklassens konstruktor anropas och att vi på så sätt undviker att upprepa kod. Därefter anpassas attributet för den aktuella klassen (`self.step_size = 1`).

Sen omdefinieras metoden `walk` för att hantera begränsningen på antalet steg, men vi fortsätter att utnyttja metoden i superklassen för att undvika att implementera samma funktionalitet två gånger.

13. Möjlig lösning:

```
import random
def minesweeper_map(size, p):
    '''
    Construct a square map for a minesweeper game.
    Params
      size: the number or rows and columns in the created map.
      p:    the probability of putting a mine in a position. Constraint: 0 <= p <= 1.

    Returns
      A list of length "size" with strings of length "size". Each character is
      either a space or the character "o".
    '''
    res_map = list()
    for i in range(size):
        s = ''
        for j in range(size):
            if random.random() < p:
                s += 'o'
            else:
                s += ' '
        res_map.append(s)
    return res_map
```

14. Felet är att `increment_neighborhood` inte tar hänsyn till matrisens gränser. Om man har en mina på kartans ”kant” så kommer det bli fel; med ett särfall om man är på rad eller kolumn n, men även på rad/kolumn 0, eftersom man får en ”wrap-around” effect som vi inte har tänkt oss. Ser du hur?

Möjlig rättning:

```
def increment_neighborhood(res, i, j):
    n = len(res) # Assumed matrix size
    for delta_i in [-1, 0, 1]:
        for delta_j in [-1, 0, 1]:
            mi = i + delta_i
            if mi >= 0 and mi <= n-1:
                mj = j + delta_j
                if mj >= 0 and mj <= n-1:
                    res[mi][mj] += 1
```

Här ser vi till att vi bara inkrementerar för element inom matrisens index.