Facit och kommentarer till tentamen 2021-03-12 i DA2004/DA2005

Del A: flervalsfrågor (1p per fråga)

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

Del B: kodfrågor (2p per fråga)

9. Möjliga lösningar:

```
import random

class Dice:

    def roll_die(self):
        return random.randint(1,6)

    def roll_dice(self,n):
        sum = 0
        for i in range(n):
            sum += self.roll_die()
        return sum

d = Dice()
print(d.roll_die())
print(d.roll_dice(2))
print(d.roll_dice(20))
```

10. Möjlig lösning:

```
def roll_two_dice(n_times):
    d = Dice()
    if type(n_times) == int and 0 < n_times:
        return [ d.roll_dice(2) for i in range(n_times) ]
    else:
        raise ValueError("n_times must be a positive integer")

print(roll_two_dice(10))
```

11. Möjlig lösning:

```
def result_of_two_dice_rolls(n_times):
    xs = roll_two_dice(n_times)

    for i in range(2,13):
        print(i,format(xs.count(i)/n_times * 100,'.1f')+"%")

result_of_two_dice_rolls(1000)
```

12. Möjlig lösning:

```
mydict = { 'hej' : 12 , 'hopp' : 10101 , 'a' : 1 }

def print_right_aligned(d):
    max_len_keys = max(map(len,d.keys()))
    max_len_vals = max(map(lambda x: len(str(x)),d.values()))

    for k,v in d.items():
        sv = str(v)
        print((max_len_keys - len(k)) * ' ' + k
             ,(max_len_vals - len(sv)) * ' ' + sv)

print_right_aligned(mydict)
```

13. Möjlig lösning:

```
def best_corner(board):
    n = len(board)
    best_square = ()
    best_score = 0
    for i in range(1, n):
        for j in range(1, n):
            score = 0
            for k1 in range(i-1, i+1):
                for k2 in range(j-1, j+1):
                    score += board[k1][k2]

            if score > best_score:
                best_score = score
                best_square = (i,j)
    return best_square, best_score
```

Lägg märke till att de yttre looparna itererar över de inre hörnen tack vare `range(1, n)`. I den inre två looparna så kommer vi åt de fyra rutor som ligger runt ett hörn.

Lösningen blir nog snyggare och lättare att läsa om man bryter ut score-beräkningen till en hjälpfunktion:

```
def best_corner(board):
    n = len(board)
    best_square = ()
    best_score = 0
    for i in range(1, n):
        for j in range(1, n):
            score = compute_score(board, i, j)
            if score > best_score:
                best_score = score
                best_square = (i, j)
    return best_square, best_score


def compute_score(board, i, j):
    n = len(board)
    score = 0
    for k1 in range(i-1, i+1):
        for k2 in range(j-1, j+1):
            score += board[k1][k2]
    return score
```

På det här viset kanske det är tydligare att i- och j-looparna är över koordinater, och k-looparna i `compute_score` är över rutor?

Eftersom dessa två funktioner hör ihop så tätt föredrar en del att lägga hjälpfunktionen som en *intern* funktion, som inte är synlig utanför `best_corner`:

```
def best_corner(board):
    def compute_score(board, i, j):
        n = len(board)
        score = 0
        for k1 in range(i-1, i+1):
            for k2 in range(j-1, j+1):
                score += board[k1][k2]
```

```
    n = len(board)
    best_square = ()
    best_score = 0
    for i in range(1, n):
        for j in range(1, n):
            score = compute_score(board, i, j)
            if score >= best_score:
                best_score = score
                best_square = (i, j)
    return best_square, best_score
```

Vad tycker du är bäst?

14. Möjlig lösning:

```
class Board:
    def __init__(self, board, occupied_positions=[]):
        self.board = board
        self.board_size = len(board)
        self.occupied_positions = occupied_positions

    def best_valid_corner(self):
        best_square = ()
        best_score = 0

        # Adding 1 to max coordinate because more corners
        # than squares. We are looping over corners here.
        for i in range(self.board_size + 1):
            for j in range(self.board_size + 1):
                if not (i, j) in self.occupied_positions:
                    score = self.compute_score(i, j)
                    if score >= best_score:
                        best_score = score
                        best_square = (i, j)
        return best_square

    def compute_score(self, i, j):
        score = 0
        for k1 in range(max(i-1, 0), min(i+1, self.board_size)):
            for k2 in range(max(j-1, 0), min(j+1, self.board_size)):
                score += self.board[k1][k2]
        return score
```

Lägg märke till att `compute_score` nu hanterar att koordinaterna inte längre är garanterat på de inre hörnen. Med hjälp av min och max garanterar vi att det är giltiga index för rutorna, oavsett vilka hörnkoordinater som ges i anropet.