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

Del A: flervalsfrågor (1p per fråga)

1. *A,B,C,D*
2. *E*
3. *E*
4. *A*
5. *C*
6. *E*
7. *D*. Funktionen beräknar *n*:te Fibonaccitalet.
8. *C*

Del B: kodfrågor (2p per fråga)

9.
```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f'<Point ({self.x}, {self.y})>'

    def translate(self, dx, dy):
        self.x += dx
        self.y += dy
```

   För att instantiera ett Point-objekt och translatera det $(1, 1)$ skriver vi:

```
p = Point(0, 0)
p.translate(1, 1)
```

10. Notera att klassen ska ärva från `Point`. Ett enkelt sätt att få koordinaterna på rätt sätt är att anropa superklassens konstruktor med rätt transformation.

    Att få `translate` att fungera följer av arvet.

```
import math
class PolarPoint(Point):
    def __init__(self, r, theta):
        super().__init__(r * math.cos(theta), r * math.sin(theta))
```

11. Lösningsförslag:

```
def punctuation(filename):
    txt = ''
    with open(filename) as h:
        for line in h:
            for character in line:
                if character in '.,:;!?':
                    txt += character
    return txt
```

    Notera att man ska läsa från fil för full poäng.

12. Lösningsförslag:

```
def swe_to_ascii(s):
    res = ''
    for character in s:
        if character == 'å':
            res += 'aa'
        elif character == 'ä':
            res += 'ae'
        elif character == 'ö':
            res += 'oe'
        else:
            res += character
    return res
```

13. I den här uppgiften ska man kunna översätta den algoritm som är föreslagen, inte hitta på sin egen algoritm. Det innebär dels att man ska ta föremål efter föremål och lägga i första bästa tillgängliga låda, dels att man ska använda särfall då det inte går att packa föremålen.

Lösningsförslag:

```
def pack(boxes, objects):
    assert boxes != []
    packing = [list() for box in boxes] # Initialise packing
    for obj, obj_size in objects.items():
        for i, box_space in enumerate(boxes):
            if obj_size <= box_space:
                packing[i].append(obj)
                boxes[i] -= obj_size
                break
        else:                   # We end up here if we have looped through all boxes
            without breaking
            raise Exception('Packing did not work')
    return packing
```

Själva beräkningsproblemet, att packa ”föremål i lådor”, kallas ”kappsäcksproblemet” (eng: *knapsack problem*). Det finns ingen effektiv algoritm som ger en optimal packning för alla indata, så man får nöja sig med suboptimala lösningar av det slag vi provar här.

14. Lösningsförslag:

```
import random
def pack_testdata(boxes):
    objects = dict()
    for i, box_size in enumerate(random.sample(boxes, len(boxes))):
        obj_name = f'obj{i}'
        objects[obj_name] = box_size
    return objects
```

Funktionen `enumerate` är alltid praktisk när man vill iterera över en datastruktur och samtidigt hålla reda på index. Här kombinerar vi det med `random.sample`, som är bättre att använda än t.ex. `random.randint` om man vill slumpa fram element ur en lista.