Suggested solutions and comments for the 2021-04-21 exam in DA2004/DA2005

Part A: multiple choice questions (1p per question)

1. *C*
2. *A, D*
3. *D* — This double list comprehension returns a list of three elements (due to `for i in range(3)`). For each value assigned to i, the inner list comprehension extracts a list containing element i from the two lists `[1,2,3]` and `[4,5,6]`.
4. *B*
5. *A* — The global variables a and b are shadowed by the function parameter a and local variable b. The call $f()$ therefore computes $a + b + c = 1 + 5 + 4 = 10$.
6. *C*
7. *A* — There are two cases: if the random number is less than 0.5, then x is returned immediately; otherwise x is compared with param and maybe incremented, but with $x = 0$ the comparison always evaluates to False causing a call to `break`. Therefore, 0 is always returned.
8. *D* — The function computes the depth of nested lists, by recursively examining elements. Everytime a list is found, 1 is added to the result of applying the depth function (using map) to each element.

Part B: coding problems (2p per problem)

9. Possible solution

```
import random

class Dice:

    def __init__(self,n=6):
        if n in [4,6,8,12,20]:
            self.sides = n
        else:
            raise ValueError("number of sides must be 4, 6, 8, 12, or 20")

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

    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))

t20 = Dice(20)
print(t20.roll_die())
print(t20.roll_dice(2))

# Raises a ValueError
# t19 = Dice(19)
```

10. Possible solution:

```
class ArbDice:

    def __init__(self,n=6,xs=[]):
        if n in [4,6,8,12,20] and (xs == [] or len(xs) == n):
            self.number_of_sides = n
```

```python
            if xs == []:
                self.sides = { i : i for i in range(1,n+1) }
            else:
                self.sides = { i : xs[i-1] for i in range(1,n+1) }
        else:
            raise ValueError("number of sides must be 4, 6, 8, 12, or 20")

    def roll_die(self):
        return self.sides[random.randint(1,self.number_of_sides)]

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

d4 = ArbDice(4,[10,20,30,40])
print(d4.roll_die())
print(d4.roll_dice(2))

d6 = ArbDice()
print(d6.roll_die())
print(d6.roll_dice(2))
```

11. Possible solution:

```python
def mask(s1,s2):
    out = ''
    for c in s1:
        if s2 == '':
            out += c
        elif c.lower() == s2[0].lower():
            out += '*'
            s2 = s2[1:]
        else:
            out += c

    return out

print(mask('Hello, how are you Anders?', 'ehoaar'))
print(mask('Hello, how are you Anders?', 'H,HAAR?A'))
```

12. Possible solution:

```python
def matrix_encrypt(s,cols):
    split_s = []
    for i in range(0,len(s),cols):
        split_s += [s[i:i+cols]]

    out = ''
    for i in range(cols):
        for x in split_s:
            if i < len(x):
                out += x[i]

    return out

print(matrix_encrypt('Secret text',3))
print(matrix_encrypt('Secret text',2))
print(matrix_encrypt('Secrettext',5))
```

13. Possible solution:

```python
def matrix_decrypt(s,cols):
    # calculate how many rows we should have
    rows = len(s) // cols
    if len(s) % cols != 0:
        rows += 1
    # now we can do the same as matrix_encrypt, but using rows instead of cols
    split_s = []
    for i in range(0,len(s),rows):
        split_s += [s[i:i+rows]]

    out = ''
    for i in range(rows):
        for x in split_s:
            if i < len(x):
                out += x[i]

    return out

print(matrix_decrypt('Sr xeettcte',3))
print(matrix_decrypt('Sce eterttx',2))
print(matrix_decrypt('Stetcerxet',5))
```

14. Possible solution:

```
def encrypt_file(f,enc):
    with open(f + ".txt",'r') as h_in, open(f + "_encrypted.txt",'w') as h_out:
        enclines = map(enc,h_in.read().splitlines())
        h_out.writelines('\n'.join(enclines))

encrypt_file('myfile',lambda x: matrix_encrypt(x,3))
```