Facit och kommentarer till prov 2019-03-18 i DA2004

Del A: flervalsfrågor

1. *c*
2. *e* Alt b: ”skapa fil m.py och definiera en klass”, fungerar ju också…
   Godtagbara svar: e, alternativ b och e.
3. *d*
4. *d*
5. *a*
6. *a* Problem! Enligt gammal tenta är svaret (e), ”det beror på vad användaren skriver”
7. *a*, *c*, och *d*
8. *c*
9. *a* och *c*
10. *c*, *d*, *e*

Del B: kodfrågor

11. `=` används för tilldelningar, till exempel `x=7`, och `==` är en jämförelseoperator som avgör om två värden är lika varandra eller inte.

12. Anropet `g('hubba')` returnerar `abbuh`. Basfallet returnerar tomma strängen, `''`, och i varje rekursivt steg skapas en ny sträng som är resultatet av ett *g*-anrop på “svansen” av strängen (`g('ubba')` i första rekursiva anropet) sammansatt med första tecknet i `s` (bokstaven `h` i första rekursiva anropet).

13. Utskriften blir “Hoppsan!” då anropet `int(s)` inte går, dvs innehållet i `s` inte kan konverteras till ett heltal. Det kan också uppstå andra fel, som till exempel att användaren gör tangentkombinationen kontroll-d (C-d, i traditionell notation) som har betydelsen “End Of File”.

    Det är särfallen `ValueError` respektive `EOFError` som uppstår och fångas av `try-except`. Det räcker att svara om problem med typkonvertering.

14. Det man ska lägga märke till är att det är en funktion som beror av två globala variabler. Det är dålig praxis. Funktionskroppen bör, i alla enkla fall som dessa, bara vara beroende av funktionens parametrar.

    Förslag på bättre implementation:

```
from math import log
def logstar(n, base):
    log_levels=0
    while n > 1:
        n = log(n, base)
        log_levels += 1
    return log_levels
```

Den itererade logaritmen är nog inte någonsin meningsfull att implementera. För alla värden på n som kan uppstå av praktiska skäl (säg, antalet atomer i universum) är $\log_2^* n \leq 5$.

15. Förslag:

```
def prefix_sums(nums):
    sums = []
    the_sum = 0
    for num in nums:
        the_sum += num
        sums.append(the_sum)
    return sums
```

16. Förslag:

```
def count_nucleotides(s):
    cnt = {'A': 0, 'C': 0, 'G': 0, 'T': 0}
    for nucleotide in s:
        cnt[nucleotide] += 1
    return cnt
```

17. Det blir fel eftersom variabeln `cs`, som borde ackumulera returvärdet, inte används annat än vid initialisering och funktionsretur. Logiken för att identifiera bokstäver som ska skrivas versalt och enskilda bokstäver görs även versala, men ska notera att de aldrig används för att samla ihop ett svar.

18. c

19. Man skriver lämpligen

```
gene = DNA('ACGTAAAGTC')
```

eller motsvarande.

20. Ett förslag till utökad DNA-klass:

```
class DNA:
    def __init__(self, s):
        self._sequence = s

    def composition(self):
        return count_nucleotides(self._sequence) # Använder
            funktionen från fråga 16
```

Eller:

```
class DNA:
    def __init__(self, s):
        self._sequence = s

    def composition(self):
        cnt = {'A': 0, 'C': 0, 'G': 0, 'T': 0}
        for nucleotide in self._sequence:
            cnt[nucleotide] += 1
        return cnt
```