

- Tentan har flervalsfrågor där minst ett svarsalternativ är korrekt. Om man svarar fel eller inte har exakt antal rätta alternativ får man noll poäng på frågan.
- Man måste bli godkänd på del A (4 rätt på 8 frågor) för att del B ska rättas.
- Del B består av frågor med varierande poäng (totalt 12p).
- Inga `import` (Pythons standardbibliotek eller externa bibliotek) får användas om de inte nämns eller finns med i uppgiften. Man får dock använda inbyggda funktioner som `len`, `range` och `map`.
- All kod avser **Python 3**, dvs *inte* t.ex. Python 2.7
- **Hjälpmedel:** Ett A4 med så mycket information du vill. Du får skriva på båda sidorna.
- **Betygsgränser:** E: 10, D: 12, C: 14, B: 16, A: 18, av maximala 20.

## Del A: flervalsfrågor

1. Vilka av dessa är operatörer för logiska uttryck i Python?

- A. `and`
- B. `not`
- C. `or`
- D. `maybe`
- E. `xor`

2. Vilken lista returneras av funktionen `fcn`?

- A. `[]`
- B. `[1, 2, 3, 4, 5]`
- C. `[2, 4, 6, 8, 10]`
- D. `[1, 4, 9, 16, 25]`
- E. `[2, 2, 2, 2, 2]`

```
def fcn():  
    res = []  
    for i in range(5):  
        print(i**2)  
    return res
```

3. Vad är resultatet av följande uttryck?

```
list(filter(lambda x: x >= 'a' and x <= 'z', 'Älta'))
```

- A. `'Ä'`
- B. `'lta'`
- C. `['l', 't', 'a']`
- D. `[]`
- E. Ett felmeddelande

4. Vad skrivs ut av koden till höger?

- A. 1
- B. 2
- C. 3
- D. 4
- E. 8

```
a = 1  
b = 2  
def some_fcn(c, d):  
    x = a * b + c * d  
    return x  
  
def another_fcn(c, d):  
    return some_fcn(a, b) + some_fcn(c, d)  
  
print(another_fcn(a, b))
```

5. Vad blir resultatet av koden till höger?

- A. 'Amiral'
- B. 'Grönvinge'
- C. 'Påfågelöga'
- D. 'Sorgmantel'
- E. Ett felmeddelande

```
d = {  
    1 : {'a': 'Amiral', 'b': 'Grönvinge'},  
    2 : {'b': 'Påfågelöga', 'a': 'Sorgmantel'}  
}  
print(d[0]['b'])
```

6. Vad är resultatet av anropet `computation(15, 1, 0)`?

- A. 0
- B. 1
- C. 2
- D. 4
- E. 8

```
def computation(x, y, z):  
    if x > y:  
        return computation(x, 2*y, z+1)  
    else:  
        return z
```

7. Vilket av följande alternativ är ett syfte med enhetstestning?

- A. Att utvärdera arbetslag som programmerar.
- B. Att testa kod för filhantering.
- C. Att automatisera testningen av funktioner.
- D. Att experimentera med uppdelning av kod i moduler.
- E. Att skriva kod för konvertering mellan enheter (tex meter och fot, C och F).

8. Betrakta koden till höger. Vad returneras av `thefunction('haparanda')`?

- A. 'a'
- B. 4
- C. 'h'
- D. 9
- E. None

```
def thefunction(lst):  
    d = {}  
    for elem in lst:  
        if elem in d:  
            d[elem] += 1  
        else:  
            d[elem] = 1  
  
    a = 0  
    b = None  
    for k, v in d.items():  
        if v > a:  
            a = v  
            b = k  
    return b
```

## Del B: programmeringsuppgifter

9. (2p) Koderna nedan har skrivits för att reversera den sträng som ges som argument till `rev`, se första exemplet till höger. Det finns dock minst två fel. Förklara vilka.

```
i = 0
def rev(s):
    s_list = []
    while i < len(s):
        s_list.append(s[-(i+1)])
        i = i + 1
    return s_list
```

```
[In: ] rev('hubba')
[Out:] 'abbuh'
[In: ] rev('marsipulami')
[Out:] 'imalupisram'
```

10. (2p) Skriv en korrekt version (dvs fungerar som i de två exemplen) av funktionen `rev` i fråga 9 som använder en `for`-loop (på ett meningsfullt sätt), använder goda programmeringsprinciper, samt ger ett `ValueError` om man skickar in tomma strängen.
11. (2p) En serie är monoton (eng: *monotonic*) om den antingen växer eller avtar: givet en lista  $L = (l_0, l_1, \dots, l_n)$  är antingen  $l_i \leq l_{i+1}$  för  $0 \leq i < n$  eller  $l_i \geq l_{i+1}$  för  $0 \leq i < n$ .

Skriv en funktion `monotonic(lst)` som returnerar `True` om `lst` är monoton och annars `False`. Vi utgår ifrån att det finns minst två element i listan.

### Exempelanvändning:

```
[In: ] monotonic([1, 2, 3, 4])
[Out:] True
[In: ] monotonic([4, 3, 2, 1])
[Out:] True
[In: ] monotonic([1, 9, 2, 7])
[Out:] False
[In: ] monotonic([0, 0, 0, 0, 0])
[Out:] True
```

12. (2p) Skriv en funktion `loan_analysis(initial_loan, monthly_repayment, yearly_interest)` som visar hur ett lån betalas av över tid, där `initial_loan` är den ursprungliga lånesumman (som exempel: 10000), `monthly_repayment` är den summa som amorteras varje månad (rak amortering, tex 1200), och `yearly_interest` är räntesatsen (tex 5%). Funktionen ska skriva ut en enkel tabell med månadsnummer, aktuell skuld, och månadens räntebetalning. Se nedan för två exempel.

- Din funktion ska iterera tills skulden är mindre än den månatliga amorteringen.
- Om skulden en viss månad är  $S$  och räntesatsen är  $r$  är månadens ränta  $S \times r/12$ , och ska läggas till skulden.
- Varje månad ska den månatliga amorteringen dras ifrån skulden.

### Exempelkörningar:

```
[In: ] loan_analysis(10000, 1200, 0.05)
[Out:]
1 8841.67 41.67
2 7678.51 36.84
3 6510.50 31.99
4 5337.63 27.13
5 4159.87 22.24
6 2977.20 17.33
7 1789.60 12.40
8 597.06 7.46

[In: ] loan_analysis(10000, 3000, 0.15)
[Out:]
1 7125.0 125.0
2 4214.06 89.06
3 1266.74 52.68
```

Vänstra kolumnen är månaden, mittkolumnen är skulden vid månadens slut, och högra kolumnen är ränta. I exemplen har jag avrundat till två decimaler och justerat kolumnerna för att utdata ska bli lättläst. De behöver du inte göra. Du behöver inte heller förhålla dig till hur banker avrundar vid beräkningar.

13. (4p) Skriv en klass `Table` som fungerar likt en uppslagstabell (`dict`), men inte gör skillnad på nyckel (*key*) och värde (*value*). Syftet med `Table` är att lagra ordnade par, dvs vi gör skillnad på  $(a, b)$  och  $(b, a)$ . Man ska kunna slå upp en ett värde med en nyckel, och slå upp en nyckel med ett värde, om man använder terminologin från uppslagstabeller.

Din lösning ska fungera som i exemplet nedan, och specifikt ska du implementera följande:

- Konstruktorn ska inte ta några argument, men initierar viktiga attribut.
- Metoden `add` lägger till ett par av värden.
- Metoden `items` ska fungera som metoden med samma namn i en uppslagstabell och returnera en generator som går igenom alla par av värden.
- Metoden `get` ska, givet att argument  $x$  som finns i tabellen, returnera värdet som  $x$  är parat med. Om ett värde finns både som vänster och höger värde i tabellen (se värdet 2 i exemplet), så ska det vänstra värdet ha prioritet vid uppslagning. Om  $x$  inte finns i tabellen ska det bli ett `KeyError`.

*Tips:* Instanser av klassen kan innehålla två uppslagstabeller där man, givet ett par  $(a, b)$  låter  $a$  vara nyckel i ena tabellen och  $b$  vara nyckel i andra tabellen.

#### Exempelanvändning:

```
[In: ] t = Table()
[In: ] t.add('a', 1)
[In: ] t.add('b', 2)
[In: ] t.add(2, 'd')
[In: ] print('left', '\t', 'right')
[In: ] for left, right in t.items():
[In: ]     print(left, '\t', right)
[Out:] left    right
[Out:] a      1
[Out:] b      2
[Out:] 2      d
[In: ] print(t.get('a'))
[Out:] 1
[In: ] print(t.get(2))
[Out:] d
[In: ] print(t.get('d'))
[Out:] 2
[In: ] print(t.get('c'))
Traceback (most recent call last):
  [COMMENT: Details removed by examiner]
KeyError: "'c' is not found in the table"
```