

(Demo) Tentamen Programmeringsteknik I (Python) Datum

Lärare: Johan Öfverstedt

Skrivtid: 08:00 - 13:00

Tänk på följande:

- Skriv läsligt. Använd inte rödpenna.
- Skriv bara på framsidan av varje papper.
- Lägg uppgifterna i ordning. Skriv uppgiftsnummer (gäller B-delen) och din kod överst i högra hörnet på alla papper.
- Fyll i försättsbladet ordentligt.
- Såvida inget annat anges, både får och ska man bygga på lösningar från föregående uppgifter även om dessa inte har lösts.
- På B-delen är det tillåtet att införa hjälpfunktioner/hjälpmetoder och hjälpklasser. Uttrycket skriv en funktion/metod betyder alltså inte att lösningen inte får struktureras med hjälp av flera metoder.
- Du behöver inte skriva import-satser för att använda funktioner och klasser från standardbiblioteken.
- Alla uppgifter gäller programmeringsspråket Python och programkod skall skrivas i korrekt Python. Koden ska vara läslig med lämpliga variabelnamn, korta men beskrivande namn på funktioner, klasser och metoder.

Observera att betyget påverkas negativt av

- icke-privata variabler och onödiga variabler,
- dålig läslighet,
- upprepning av identisk kod,
- underlåtenhet att utnyttja given/egen skriven kod.

Skrivningen består av två delar. Lösningarna till uppgifterna på A-delen ska skrivas in i de tomma rutorna och den delen ska lämnas in. Rutorna är tilltagna i storlek så att de ska rymma svaren. En stor ruta betyder inte att svaret måste vara stort! Lösningarna till uppgifterna på B-delen skrivs på lösa papper. För att bli godkänd (betyg 3) krävs att minst ca 75% av A-delen är i stort sett rätt löst. För betyget 4 krävs dessutom att minst hälften av uppgifterna på B-delen och betyg 5 att alla uppgifterna på B-delen är i stort sett rätt lösta. Vid bedömning av betyg 4 och 5 tas också hänsyn tillkvalitén på lösningarna i A-delen. Observera att B-delen inte rättas om inte A-delen är godkänd.

Lycka till!

Del A

A.1.

| | |
|--|--|
| <p>A)</p> <pre>a = [(3, 'Johan'), (5, 'Erika'), (1, 'Gabriel')] b = sorted(a) c = [v for k, v in b] print(c)</pre> | <p>Vad händer?</p> <ol style="list-style-type: none">1) [Johan, Erika, Gabriel] skrivs ut2) [Gabriel, Erika, Johan] skrivs ut3) [Gabriel, Johan, Erika] skrivs ut4) [1, 3, 5] skrivs ut |
| <p>B)</p> <pre>def increment(x): x = x + 1 return x x = 1 y = increment(x) print(x, y)</pre> | <p>Vad händer?</p> <ol style="list-style-type: none">1) 2 2 skrivs ut2) 2 1 skrivs ut3) 1 2 skrivs ut4) 1 1 skrivs ut5) Det blir ett ValueError |
| <p>C)</p> <pre>for i in range(5, 0, -1): print(i, end=' ')</pre> | <ol style="list-style-type: none">1) 5, 4, 3, 2, 1 skrivs ut2) 0, 1, 2, 3, 4 skrivs ut3) 5, 4, 3, 2, 1, 0 skrivs ut4) 0, 1, 2, 3, 4, 5 skrivs ut |
| <p>D)</p> <pre>def f(x): return x * 2 y = f(2)</pre> <p>Vad har variabeln <code>x</code> för datatyp?</p> | <ol style="list-style-type: none">1) int2) float3) str4) list5) <code>x</code> har ingen bestämd datatyp |
| <p>E)</p> <p>Vad är ett annat namn för <code>__init__</code>-metoden i klasser?</p> | <ol style="list-style-type: none">1) Generator2) Main-metod3) Konstruktör4) Skapare |
| <p>F)</p> <pre>def f(x, y): return x + y</pre> <p>Vad är <code>x</code> och <code>y</code>?</p> | <ol style="list-style-type: none">1) Argument2) Faktorer3) Attribut4) Parametrar |
| <p>G)</p> <pre>print([1, 2] + ([3, 4]*2))</pre> | <p>Vad skrivs ut?</p> <ol style="list-style-type: none">1) [1, 2, 3, 4, 3, 4]2) [7, 10]3) [1, 2, 6, 8]4) Det blir fel |
| <p>H) Vad kan man säga säkert om <code>s</code>?</p> <pre>s = Student('Johan')</pre> | <ol style="list-style-type: none">1) <code>s</code> är en referens till en instans av klassen <code>Student</code>2) Vi vet ej vad <code>s</code> är av för datatyp3) Det blir fel då vi måste anropa <code>Student.__init__('Johan')</code>4) <code>print(s.name)</code> kommer ge utskriften <code>'Johan'</code> |

A.2. Skriv en funktion `pow` som givet två positiva heltal `x`, `y` beräknar x^y , `x` upphöjt till `y`, med en loop och multiplikation [`x*x*... *x` (`y` gånger)].

A.3. Skriv en funktion `kth_element(lst, k)` som givet en lista returnerar det `k = [0, len(lst))` minsta elementet. Exempel:

```
print(kth_element([1, 2, 5, 4, 3], 2))  
print(kth_element([1, 2, 5, 4, 3], 1))
```

Ger utskrift:

```
3  
2
```

A.4. Skriv en funktion `less(lst, a)` som returnerar en lista med index för alla element i `lst` som är mindre än `a`. Exempel:

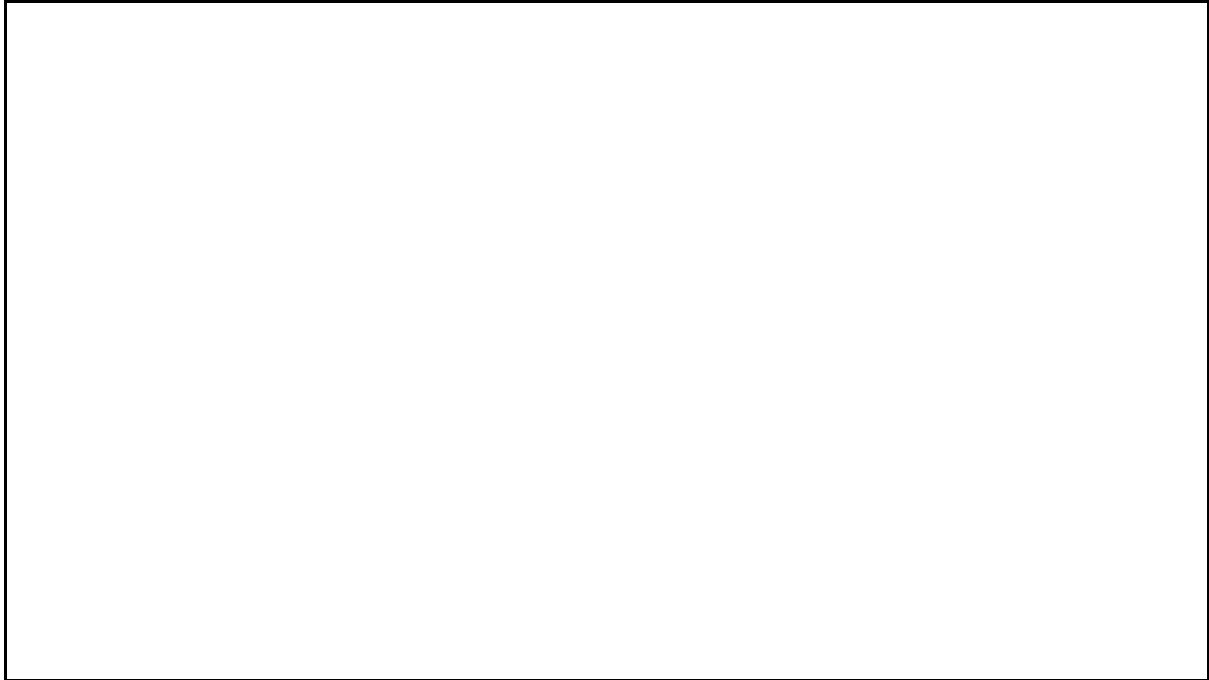
```
print(less([1, 2, 6, 4, 0, 7, 9, 3, 2], 3))
```

Ger utskrift:

```
[0, 1, 4, 8]
```

A.5. Skriv en klass `Person` som representerar ett namn och e-postadress. Klassen ska innehålla en metod `print_person` (utöver konstruktorn) som skriver ut namnet och e-postadressen på en rad.

Skriv även kod för att skapa ett person-objekt med namnet 'Jan Karlsson' och e-postadress 'jan.karlsson@gmail.com'.



A.6. Skriv en funktion `word_freq(s)` som skapar och returnerar ett lexikon med ordfrekvenser för strängen `s`. Exempel:

```
print(word_freq('hej hej monica hej på dig monica'))
```

ger utskriften:

```
{'hej': 3, 'på': 1, 'dig': 1, 'monica': 2}
```

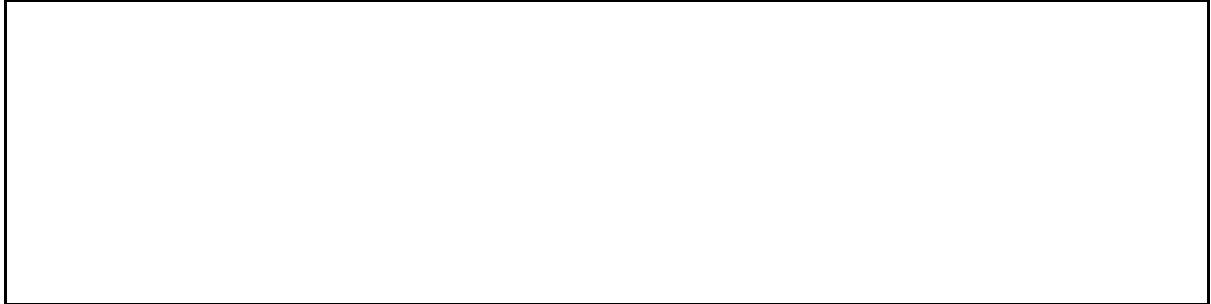
A.7. Skriv en funktion `random_value(lst)` som givet en lista med element returnerar ett slumpmässigt element från listan och kod som anropar och skriver ut returvärdet från metoden för listan `['a', 'b', 'c']` (`random.random()` ger ett slumpmässigt flyttal mellan 0.0 och 1.0).

A.8.

Givet två geometriska punkter A och B som representeras av tupler/par av två flyttal, skriv en funktion `euclidean_distance(a, b)` som beräknar det Euklidiska avståndet mellan dem, vilket ges av formeln:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Kvadratroten kan beräknas med `math.sqrt(...)`.



Del B

I denna del ska alla svar skrivas på ett löst papper. Kom ihåg att skriva tentamenskod på varje blad.

Följande program (i filen `game.py`) är givet, vilket implementerar ett litet Tic-Tac-Toe-spel.

```
import board as brd

def main():
    sz = 3
    b = brd.Board(sz)

    player = 0
    symbols = ['X', 'O']
    while b.count() < sz*sz:
        print('-----')
        print(b)
        print(f'Player {symbols[player]}\''s turn')
        x = int(input('x: '))
        y = int(input('y: '))
        try:
            b.place(x, y, symbols[player])
            w = b.check_for_winner()
            if w is not None:
                print(b)
                print(f'The winner is player {w}.')
                return
            player = 1 - player
        except ValueError as ve:
            print(ve)
    print(b)
    print('It is a tie. No winner.')

if __name__ == '__main__':
    main()
```

Testutskrift vid en provkörning:

| | |
|----------------------------|----------------------------|
| ----- | ----- XOX .O. .X. |
|----------------------------|----------------------------|

| | |
|---|--|
| <pre> Player X's turn x: 0 y: 0 ----- X.. Player O's turn x: 1 y: 1 ----- X.. .O. ... Player X's turn x: 2 y: 0 ----- X.X .O. ... Player O's turn x: 1 y: 0 ----- XOX .O. ... Player X's turn x: 1 y: 2 </pre> | <pre> Player O's turn x: 0 y: 1 ----- XOX OO. .X. Player X's turn x: 2 y: 1 ----- XOX OOX .X. Player O's turn x: 2 y: 2 ----- XOX OOX .XO Player X's turn x: 0 y: 2 XOX OOX XXO It is a tie. No winner. </pre> |
|---|--|

B.1. Definiera klassen `Board` och skriv en konstruktör som givet en spelplansstorlek `n` (heltal) skapar en nästlad lista `n x n` (bestående av en lista med rader, som i sin tur består av kolumner med pjäser) som enda attribut, initialt fylld med tomma element (antingen `None` eller `'.'`).

B.2. Skriv en metod `count(self)` som räknar hur många platser har fyllts med en pjäs.

B.3. Skriv metoden `place(self, x, y, symbol)` i klassen `Board` som placerar den givna symbolen på plats `x` och `y`. Om `x` eller `y` är utanför spelplanen, eller om symbolen är en annan än `'X'` eller `'O'` ska ett fel signaleras (assert fungerar bra för detta ändamål). Om platsen redan är upptagen ska ett `ValueError` ges med ett lämpligt felmeddelande på formen:

```
Illegal move at ({x}, {y}).
```

B.4. Skriv metoden `__str__(self)` i klassen `Board` som returnerar en sträng som representerar spelplanens status i ett rutnät, exempelvis:

.X.
OX.
XO.

Tips: '\n' är representationen för radbrytning.

B.5. Antag att du har en metod (i klassen `Board`), `get_configurations(self)`, som returnerar alla tänkbara *n-konfigurationer* där *n* är spelplanens storlek, i form av en lista med strängar, som kan innebära att spelet har vunnits av någon av spelarna.

Exempelvis för spelplanen i B.4:

```
[ '.X.', 'OX.', 'XO.', '.OX', 'XXO', '...', '.X.', '.XX' ]
```

Skriv en metod `check_for_winner(self)` som testat om någon av spelarna har vunnit spelet (om vi har lika många i rad som storleken på spelplanen, utan att antaga att den är 3 stor).

Om spelare 'X' har vunnit, returnera 'X', om spelare 'O' har vunnit, returnera 'O', och om ingen har vunnit, returnera `None`.

Referensblad:

Listor

`len(lst)` - Ger listans längd.

Strängar

`s.join(lst)` - sammanfogar en lista till en sträng med `s` mellan varje element.

Exempel `'-'.join('1', '2', '3')` ger `'1-2-3'`.

`s.count(subs)` - räknar antalet förekomster av strängen `subs` i strängen `s`.

Exempel `'abbabba'.count('bb')` ger 2, `'abbabba'.count('b')` ger 4.

Sekvenser

`range(start, stop, step)` - skapar en sekvens med heltal från start till men inte med stop med steglängd `step`.

Exempel:

`list(range(7, 2, -1))` ger `[7, 6, 5, 4, 3]`.