Tentamen Programmeringsteknik I (Python) 2022-01-11

Lärare: Simon Sticko
Skrivtid: 14:00 - 19:00

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 långt! Lösningarna till uppgifterna på B-delen skrivs på lösa papper.

Varje uppgift A1-A8 och B1-B3 blir antingen godkänd eller underkänd. En lösning måste inte nödvändigtvis vara helt korrekt för att uppgiften ska bli godkänd. Mindre fel kan tillåtas.

Betygskriterier

- För betyg 3 krävs minst 6 av 8 godkända uppgifter på A-delen.
- För betyg 4 krävs betyg 3 samt minst 2 av 3 godkända uppgifter på B-delen.
- För betyg 5 krävs betyg 3 samt 3 av 3 godkända uppgifter på B-delen.
- Observera att B-delen inte rättas om inte A-delen är godkänd.

Tänk på följande

- Skriv läsligt. Använd inte röd penna.
- Skriv bara på framsidan av varje papper.
- Lägg uppgifterna i ordning. Skriv uppgiftsnummer (gäller B-delen) och din anmälningskod överst på alla papper.
- Fyll i försättsbladet ordentligt.
- 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 eller funktioner.
- 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 och korta men beskrivande namn på funktioner, klasser och metoder. Glöm inte indentering! Betyget påverkas negativt av icke-lokala variabler, onödiga variabler, dålig läslighet och upprepning av identisk kod.

Lycka till!

Tentakod:	

Del A

A1

Denna uppgift består av 5 st flervalsfrågor. Kryssa i rutan bredvid rätt alternativ. För godkänt resultat på uppgiften krävs rätt svar på minst 4 av 5 flervalsfrågor.

a) Vilken datatyp kommer `x` ha?

```
first = ["a", 1.0]
second = [(2, 4), 3]
third = first + second
x = third[-2]
```

- ☐ `str`
- ✓ `tuple`
- ☐ `int`
- ☐ `list`

b) Vad händer när man kör följande kod?

```
list_1 = [1, 2]
list_2 = list_1
list_2[0] = 3
print(list_1)
```

- ☐ Man får ett "run-time error". Listor är oföränderliga.
- ☐ Man får utskriften `[1, 2]`
- ✓ Man får utskriften `[3, 2]`

c) Vad händer när man kör följande kod?

```
tpl_1 = (1, 2)
tpl_2 = tpl_1
tpl_2[0] = 3
print(tpl_1)
```

- ✓ Man får ett "run-time error". Tupler är oföränderliga.
- ☐ Man får utskriften `(1, 2)`
- ☐ Man får utskriften `(3, 2)`

d) Vad är det som är speciellt med `__str__`-metoden?

- ☐ Den måste returnera `None`.
- ✓ Den anropas automatiskt när man skickar objektet till `print`-funktionen.
- ☐ Man måste implementera den för att kunna köra sitt program.
- ☐ Den anropas alltid av klassens konstruktor.

Tentakod:	

e) Vad är `csv` i följande kod?

```
import csv

with open('people.csv', 'r') as csv_file:
    reader = csv.reader(csv_file)
    for row in reader:
        print(row)
```

- ☐ En funktion.
- ✓ En modul.
- ☐ En klass.
- ☐ Ett objekt.

Tentakod:	

A2

Skriv en funktion `turn_backwards` som tar in en text-sträng, vänder den baklänges och returnerar den nya strängen. Följande kod:

```
sentence = "Kul nästan jämt."
backwards = turn_backwards(sentence)
print(backwards)
```

Ska alltså ge följande utskrift:

```
.tmäj natsän luK
```

```python
def turn_backwards(sentence):
    backwards = ""
    for character in sentence:
        backwards = character+backwards
    return backwards


# Alternative solution
def turn_backwards_alternative(sentence):
    return sentence[::-1]
```

Tentakod:	

A3

Skriv en funktion `odd_removed`. Funktionen ska ta en lista med heltal som inparameter och returnera en ny lista där bara de jämna talen finns med. Följande kod:

```
numbers = [1, 12, 5, 8, 7]
even_numbers = odd_removed(numbers)
print(even_numbers)
```

ska alltså ge utskriften:

```
[12, 8]
```

Tips: Använd modulus-operatorn `%` för att testa om ett tal är udda eller jämt. Denna returnerar resten vid heltalsdivision. Till exempel: `3%2` returnerar `1` och `4%2` returnerar `0`.

```python
# Solution with a comprehension
def odd_removed(numbers):
    return [number for number in numbers if number % 2 == 0]


# Solution without a comprehension
def odd_removed_alternative(numbers):
    only_even = []
    for number in numbers:
        if number % 2 == 0:
            only_even.append(number)
    return only_even
```

Tentakod:	

A4

Skriv en funktion `convert_to_numbers` som tar en lista med strängar som representerar flyttal (decimaltal) som inparameter och returnerar en lista med motsvarande flyttal. Följande kod:

```
numbers_as_strings = ["2.1", "3.5", "4.6"]
numbers = convert_to_numbers(numbers_as_strings)
print(numbers)
```

ska alltså ge utskriften:

```
[2.1, 3.5, 4.6]
```

```
def convert_to_numbers(numbers_as_strings):
    return [float(number) for number in numbers_as_strings]
```

Tentakod:	

A5

Skriv en funktion `frequency_to_words`. Funktionen ska ta en lista bestående av tupler, som representerar ords förekomster i en text, till exempel:

```
word_frequency = [('jag', 1), ('längtar', 1), ('hem', 2),
                  ('till', 1), ('min', 1), ('planet', 1)]
```

Funktionen ska sedan returnera ett lexikon som mappar antalet förekomster till en lista med ord. Följande anrop

```
frequency2words = frequency_to_words(word_frequency)
print(frequency2words)
```

ska alltså ge utskriften

```
{1: ['jag', 'längtar', 'till', 'min', 'planet'], 2: ['hem']}
```

Tips: Använd `in` för att kolla om en nyckel finns i ett lexikon. Exempel:

```
price_per_kg = {"apples": 12, "pears": 16}
if "apples" in price_per_kg:
    print("Apples cost", price_per_kg["apples"], "kr/kg")
```

```
def frequency_to_words(word_frequency):
    frequency2words = {}
    for word, frequency in word_frequency:
        if frequency in frequency2words:
            frequency2words[frequency].append(word)
        else:
            frequency2words[frequency] = [word]
    return frequency2words
```

Tentakod:	

A6

Skriv en funktion `rolls_to_reach`. Funktionen ska ta en inparameter `target_sum` av typen `int` och returnera hur många gånger man behöver kasta en tärning (med 6 sidor) för att summan av alla tärningskast ska komma över `target_sum`. Resultatet av funktionen är slumpmässigt men följande kod:

```
target_sum = 100
rolls = rolls_to_reach(target_sum)
print(f"Needed {rolls} rolls to reach higher than {target_sum}.")
```

skulle *till exempel* kunna ge utskriften:

```
Needed 28 rolls to reach higher than 100.
```

Tips: `random.randint(1, 6)` returnerar ett slumpmässigt heltal mellan 1 och 6.

```
import random


def rolls_to_reach(target_sum):
    dice_min = 1
    dice_max = 6
    n_rolls = 0
    sum = 0
    while sum < target_sum:
        roll = random.randint(dice_min, dice_max)
        sum += roll
        n_rolls += 1
    return n_rolls
```

Tentakod:	

A7

Skriv en klass `Rectangle`. Klassen ska ha

- 2 instansvariabler: `width` och `height`.
- En konstruktor som tar både `width` och `height` som inparametrar.

```
class Rectangle:

    def __init__(self, width, height):
        self.width = width
        self.height = height
```

A8

Skriv en metod, `area`, i klassen `Rectangle` i föregående uppgift, som returnerar rektangelns area. Följande kod

```
width = 3
height = 4
rectangle = Rectangle(width, height)
print("The rectangle has area =", rectangle.area())
```

ska alltså ge utskriften:

```
The rectangle has area = 12
```

```
class Rectangle:

    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width*self.height
```

Del B

Du vill skicka mail och gratulera alla som fick en 5:a på tentan. Den data som krävs för att göra detta är dock lite utspridd. För det första har du en lista som parar ihop den 11-siffriga tenta-koden med listor med de resultat varje student fick på A och B-delen:

```
exam_results = [
    ('P2YC-APIM-T8X', ['F', 'P', 'P', 'P', 'P', 'F', 'P', 'P'], ['F', 'P', 'P']),
    ('CC5T-E8FJ-YXJ', ['P', 'P', 'P', 'P', 'P', 'F', 'P', 'P'], ['P', 'P', 'P'])
    ... # More rows with results
]
```

Listan innehåller alltså tupler som är tripletter:

`(Exam Code, Result Part A, Result Part B),`

där `"P"` och `"F"` beskriver att en given fråga på A eller B-delen blev godkänd (Pass) eller underkänd (Fail). Du har även en lista, `students`, som innehåller alla studenter på kursen. Listan innehåller objekt av typen `Student` som har namn och email-address:

```
class Student:

   def __init__(self, name, email, id):
       self.name = name
       self.email = email
       self.id = id
```

Eftersom olika studenter kan ha samma namn, har varje student även ett unikt id, som är en sträng med 15 tecken. Om man skulle skriva ut alla studenter:

```
for student in students:
    print(f"{student.id}, {student.name}, {student.email}")
```

skulle de första raderna i utskriften vara:

```
C14AN35RKC75UEP, Kai Siegbahn, kai@siegbahn.se
XC0IWY0SQ3LTXI3, Evert Taube, evert@taube.se
...
```

Eftersom tentan skrivs anonymt, finns det även en lista med tupler som parar ihop varje tenta-kod med ett student-id:

```
code_id_pairs = [
    ('P2YC-APIM-T8X', 'C14AN35RKC75UEP'),
    ('CC5T-E8FJ-YXJ', 'XC0IWY0SQ3LTXI3')
    ...
    ]
```

Alla studenter anmälde sig inte till tentan och alla som anmälde sig till tentan dök inte upp. Listorna `exam_results`, `students`, och `code_id_pairs` är därmed inte lika långa, men innehåller alla i storleksordningen 100 element.

B1

Skriv en funktion `compute_grade` som tar in de två listorna med resultat för A och B-delen och returnerar det betyg studenten fick som en sträng. Funktionen ska alltså returnera `"F"`, `"3"`, `"4"` eller `"5"`. Följande betygskriterier ska användas:

- För betyg 3 krävs minst 6 av 8 godkända uppgifter på A-delen.
- För betyg 4 krävs betyg 3 samt minst 2 av 3 godkända uppgifter på B-delen.
- För betyg 5 krävs betyg 3 samt 3 av 3 godkända uppgifter på B-delen.

Tips: Använd metoden `count` för att räkna hur många gånger ett element förekommer i en lista. Exempel:

```
characters = ["A", "B", "A", "C"]
print(characters.count("A"))
```

Ger utskriften 2.

```
def compute_grade(part_A, part_B):
    n_correct_on_A = part_A.count("P")
    n_correct_on_B = part_B.count("P")

    required_on_A_to_pass = 6
    required_on_B_for_grade_4 = 2
    required_on_B_for_grade_5 = 3

    if n_correct_on_A >= required_on_A_to_pass:
        if n_correct_on_B >= required_on_B_for_grade_5:
            return "5"
        elif n_correct_on_B >= required_on_B_for_grade_4:
            return "4"
        return "3"
    return "F"
```

B2

Konstruera ett lexikon som har studenters unika id:n som nycklar och betyget på tentan som värden. Du kan använda funktionen `compute_grade` från uppgiften B1 (även om du inte löst B1).

```
    # First construct a dict that maps student-id to a course code.
    code2id = {code: id for code, id in code_id_pairs}

    # Then use the function from B1 and code2id to create
    # the dict we actually want:
    id2grade = {code2id[code]: compute_grade(part_A, part_B)
                for code, part_A, part_B in exam_results}
```

B3

Skriv kod som skickar ut "Congratulations!" till alla studenter som fick en 5:a på kursen. Du kan anta att det finns en funktion för att skicka mail: `send_email(email_adress, message)`, som tar 2 strängar som inparametrar. Du kan använda lexikonet från uppgift B2 (även om du inte löst B2).

```
for student in students:
    if student.id in id2grade:
        grade = id2grade[student.id]
        if grade == "5":
            send_email(student.email, "Congratulations to grade 5!")
```