Del 1

Uppgift 1 – Abstraktion

Du ska implementera fyra funktioner enligt beskrivningen nedan så att det givna programmet i **uppgift1.py** fungerar som förväntat. Given kod i **uppgift1.py** får inte ändras.

create_menu – Denna funktion ska skapa en tom meny. En meny representeras av en dictionary och består av namnet (*string*) på maträtter som associeras med ett pris (*int*).

add_item - Denna funktion ska lägga till en maträtt i menyn. Funktionen behöver ta emot namnet på rätten samt dess pris i kronor. Om maträtten redan finns i menyn ska priset uppdateras.

take_order - Denna funktion ska kunna ta emot en beställning från ett sällskap. En beställning representeras av en dictionary där nycklarna är namnen på rätterna, och värdena är antalet av respektive rätt som beställs

print_receipt - Denna funktion skriver ut ett kvitto (se funktionella krav nedan). Formatteringen av utskriften är valfri, men informationen ska stämma.

Funktionella krav (användarinmatning i fetstil):

```
Pad Thai      2         220 kr
Panäng Curry  1         150 kr
Vårrullar     3         90 kr
```

Uppgift 2 – Algoritmisk problemlösning

Du ska implementera en variant av algoritmen "Fizz Buzz". Algoritmen beskrivs nedan:

- Börja med ett heltal som vi kallar för: **`n`**.
- För varje tal mellan **1** till och med **`n`**:
  - Om talet är jämnt delbart med **3**, skriv ut "**Fizz**".
  - Om talet är jämnt delbart med **5**, skriv ut "**Buzz**".
  - Om talet är jämnt delbart med både **3** och **5**, skriv ut "**FizzBuzz**".
  - Annars, skriv ut talet självt.

Formatteringen av utskriften är valfri, men informationen ska stämma.

Funktionella krav (användarinmatning i fetstil):

```
1    2    Fizz  4    Buzz  Fizz  7    8    Fizz  Buzz  11   Fizz  13
14   FizzBuzz
```

Uppgift 3 – Behållare

Skriv ett program där användaren får mata in en sträng. Programmet ska beräkna hur många gånger varje ord förekommer i strängen. Du kan förutsätta att användaren endast matar in små bokstäver (a-z), mellanslag och avslutar med return (enter). Beräkningen ska sparas i en dictionary som sedan skrivs ut, se funktionella krav nedan.

Funktionella krav (användarinmatning i fetstil):

```
Ange din sträng: hej hej hej pa dig dig

{'hej': 3, 'pa': 1, 'dig': 2}
```

Del 2

Uppgift 4 – Referenssemantik

I den givna filen **uppgift4.py** finns ett givet huvudprogram med en given datastruktur och funktionsanrop. Dessa variabler representerar små sensorer i en fabrik som mäter fyra olika kritiska värden för fabrikens produktion: temperatur (”c”, celsius), tryck (”kPa”, kilopascal) samt luftfuktighet (”hum”, luftfuktigheten i procent). Samtliga värden sparas som heltal. Du ska funktionerna som krävs för att det givna huvudprogrammet fungerar enligt de funktionella kraven. Följande ickefunktionella krav ska också uppfyllas:

Funktionen **create_sensor_array** tar emot en dictionary av sensorer och namnen på två sensorer som ska länkas samman till vad vi kallar en sensoruppsättning. Funktionen ska länka ihop två sensorer så att de delar data, alltså ska båda sensorerna ha en referens till en och samma dictionary. När två sensorer länkas samman ska resultatet normaliseras genom att lagra medelvärdet av de enskilda mätvärdena i den delade datastrukturen. Du kan förutsätta att sensorerna som länkas samman inte är länkade med någon annan sensor.

Funktionen **sensor_read** tar emote en dictionary av sensorer, namnet på en sensor, den nyckel (temperatur, tryck eller luftfuktighet) som ska ändras, och hur mycket värdet ändras (ett heltal, både positiva och negativa tal ska funka). Om sensorn vars namn som skickas in har delad data skall alltså bägge sensorer som är en del av sensoruppsättningen få det nya resultatet. Notera att värdet för både sensor A och C uppdateras i exmplet nedan då de har delad data.

Funktionen **unlink_sensor_array** tar emote en dictionary av sensorer och namnen på två sensorer. Funktionen ska brya länken mellan de två sensorerna så att de inte längre delar data. När sensoruppsättningen bryts ska de individuella sensorerna inte längre ha en delad referens till samma dictionary, utan ha varsin dictionary. Båda sensorerna behåller värdena de hade innan delningen. Du kan förutsätta att sensorerna som kopplas isär sedan tidigare är kopplade med varandra.

Funktionella krav:

```
Original sensor values:
{'sensor_a': {'c': 103, 'kPa': 922, 'hum': 51},
 'sensor_b': {'c': 64, 'kPa': 405, 'hum': 56},
 'sensor_c': {'c': 110, 'kPa': 894, 'hum': 37},
 'sensor_d': {'c': 103, 'kPa': 349, 'hum': 31}}

After linking sensor A and C:
{'sensor_a': {'c': 106, 'kPa': 908, 'hum': 44},
 'sensor_b': {'c': 64, 'kPa': 405, 'hum': 56},
 'sensor_c': {'c': 106, 'kPa': 908, 'hum': 44},
 'sensor_d': {'c': 103, 'kPa': 349, 'hum': 31}}

After changing readings:
{'sensor_a': {'c': 113, 'kPa': 980, 'hum': 44},
 'sensor_b': {'c': 60, 'kPa': 405, 'hum': 56},
 'sensor_c': {'c': 113, 'kPa': 1012, 'hum': 44},
 'sensor_d': {'c': 103, 'kPa': 349, 'hum': 31}}
```

Uppgift 5 – Skapa algoritm

Minimum edit distance, eller redigeringsavstånd på svenska (även kallat Levenshtein-avstånd) är ett mått på hur många operationer som krävs för att omvandla en sträng till en annan. Givet två stycken strängar kan vi räkna antalet operationer som krävs för att dessa strängar ska bli identiska. Den riktiga algoritmen är aningen komplex, så din uppgift är att implementera en enklare variant enligt beskrivningen nedan i form av en funktion: **edit_distance**.

Algoritmen har tre operationer: lägga till ett tecken i slutet, ta bort ett tecken i slutet, och byta (substituera) ett tecken mot ett annat. Redigeringsavståndet är det minsta antalet operationer som krävs för att en sträng ska få samma innehåll som en annan sträng. Förhållande är kommutativt (avståndet från sträng a till sträng b är alltid samma som från sträng b till sträng a, se funktionella krav nedan). Ett **tecken** i denna kontext kan vara en bokstav, en siffra, ett mellanslag eller ett skiljetecken.

Varianten du ska implementera kontrollerar förs hur många tecken som behöver läggas till i eller tas bort från slutet av strängen. Sedan kontrolleras antalet substitutioner som krävs.

Till exempel ger strängarna ”hund” och ”hinkar” redigeringsavståndet 4 med denna algoritm, då två bokstäver behöver läggas till på ”hund”, sedan behöver två bokstäver bytas ut.

Steg 1: ”hinkar” – ”hund**ar**” – Två bokstäver läggs till i slutet (vi lägger till rätt bokstäver)

Steg 2: ”hinkar” – ”h**u**n**d**ar” – Två bokstäver byts ut.

Strängarna ”kaffekopp” och ”kaffeplanta” ger redigeringsavståndet 6 då vi plockar bort två tecken från ”kaffeplanta” och sedan byter ut fyra tecken.

Ickefunktionellt krav: Din funktion ska beräkna avståndet, du får inte modifiera parametrarna eller skapa nya strängar (eller något som liknar strängar ex. en lista av tecken).

Krav: Minst två funktioner med tydligt avgränsade syften som följer principerna för procedurell abstraktion skall användas för att lösa problemet.

Krav: Fall som inte kräver några operationer ska returnera resultatet omedelbart.

OBS: Hur utskriften ser ut och formatteras spelar ingen roll. Se bara till att vi kan se att ni testar med exempelströngarna nedan. Exempelströngarna finns i den givna filen **uppgift5.py**.

Funktionella krav:

```
"flaggstång" – "flakmoppe" -> 7
"kaffekopp" – "kaffeplanta" -> 6
"laptop" – "laptop" -> 0
"katt" – "katterna" -> 4
"katterna" – "katt" -> 4
"hyperneuroakustiska diafragmakontravibrationer" – "hicka" -> 45
"hicka" – "hyperneuroakustiska diafragmakontravibrationer" -> 45
"2024-10-14" – "1988-10-14" -> 4
"2024-10-14" – "1988-10-14" -> 4
```

Uppgift 6 – Abstrakta datatyper

I den givna filen **tournament.py** finner du ett huvudprogram som använder en ADT som importeras från den givna filen **uppgift6.py**. Din uppgift är att implementera den ADT som definierats i **uppgift6.py** så att det går att köra **tournament.py** med förväntat resultat. Det är rekommenderat att du kontrollerar hur **tournament.py** fungerar innan du börjar skriva egen kod.

Den ADT du ska designa ska fungera för ett enklare ligahanteringssystem, där ett godtyckligt antal spelare ska kunna läggas till i en turneringsliga. I denna liga ska alla spelare köra exakt en match mot varje annan deltagare. Detta innebär att du kommer att behöva tänka på hur du ska representera både spelarna samt den faktiska turneringen. Funktionerna i ADT:n har följande funktionella krav:

new_league() -- Denna funktion ska returnera en associativ behållare (dict) som representerar en ny turnering. Du får själv besluta vilken data denna behållare behöver innehålla.

add_player(league, name) – Lägger till en spelare med namnet **name** i ADT-behållaren **league**. Du får designa ADT-representationen för en spelare på valfritt sätt, men en spelare ska ha koll på sitt eget namn samt vilka andra spelare de har vunnit eller förlorat matcher mot.

is_league_ongoing(league) – Returnerar **true** så länge som det finns matcher kvar att spela. Returnerar **false** när alla matcher har körts. **Tips:** Kolla hur denna används i huvudprogrammet!

generate_matchups(league) – Denna funktion ska förbereda ligan för start genom att generera en lista med alla matcher som behöver spelas i turneringen. Tänk på att varje spelare ska spela exakt en match mot alla andra deltagare. Hur detta ska lagras i din ADT bestämmer du själv.

play_matchup(league) – Här ska exakt en match mellan två spelare från den förberedda listan som skapade i **generate_matchups** spelas. För denna uppgift kan du slumpa fram vem som vinner en match med `random.randint(x, y)` som ger dig ett slumpat tal mellan x och y. Denna funktion ska skriva ut vilka som spelar samt vem som vann, och registrera både vinsten och förlusten i din spelar-ADT. Se hur utskriften från denna funktion ska se ut i körexemplet.

print_statistics(league) – Denna funktion ska skriva ut alla spelares namn, hur många matcher de har vunnit samt hur många de förlorat. Din implementation ska formateras på samma sätt som exemplet nedan.

Funktionella krav (ett förenklat exempel med tre spelare)**:**

```
Love vs Hillevi
Round won by Hillevi!

Love vs Pontus
Round won by Pontus!

Hillevi vs Pontus
Round won by Hillevi!

Name            Wins            Losses
-----------------------------------------
Love            0               2
Pontus          1               1
Hillevi         2               0
```