Practicum vrijdag

Collapse context

Your answer passed the tests! Your score is 100.0%. [Submission #6712872f3baf09455ab8630b (**2024-10-18 16:05:03**)]

Question 1: Splits

✓ Perfect

Schrijf een functie `splits` die een lijst in twee splitst. De functie geeft een tupel terug met als eerste element de eerste helft en als tweede element de tweede helft. Als de lijst een oneven aantal elementen bevat, zal het middelste element niet meer voorkomen in de resulterende lijsten. Je neemt dan alles ervoor en alles erna, zonder het element zelf.

```
1  def splits(lijst):
2      return (lijst[:len(lijst)//2], lijst[len(lijst)//2:] if not len(lijst) % 2 else
   lijst[(len(lijst)//2) + 1:])
```

Submit

Question 2: Debuggen maar!

✓ Perfect

Volgende code bevat een paar bugs. Probeer die te vinden met de debugger.

```
1  def is_prime(n):
2      """
3      check whether n is a prime number
4      :param n: the number being tested
5      :return: True iff the number is prime
6
7      >>> is_prime(2)
8      True
9      >>> is_prime(4)
10     False
11     """
12     if n <= 0:
13         return None
14
15     for i in range(n - 1, 1, -1):
16         if n % i == 0:
17             return False
18
19     return True
```

Submit

Question 3: Catering bedrijf (1)

✓ Perfect

Voor de volgende oefeningen moet je code schrijven om een catering bedrijf te helpen met het organiseren en voorbereiden van de bestellingen van de klanten. Je krijgt per klant een lijst met de gewenste items en je moet een programma schrijven dat het aantal items dat klaargemaakt moet worden telt. De mogelijke items zijn: salade, hamburger en water.

Schrijf een functie `item_order` dat een lijst krijgt als input. Deze lijst bevat telkens de eerste letter van elk item dat besteld wordt (`s` voor salade, `h` voor hamburger en `w` voor water). Deze functie geeft een lijst met drie elementen terug. Het eerste element is het aantal salades, het tweede het aantal hamburgers en het laatste het aantal waters. Als een bestelling een element niet bevat dan is de hoeveelheid gelijk aan 0. We noemen het resultaat vanaf nu de bestelling. We krijgen dus een lijst als input en zetten die om naar een bestelling (een geordende lijst van telkens 3 waarden). Bijvoorbeeld:

```
>>> item_order(["s", "w", "h", "s", "h"])
[2, 2, 1]
>>> item_order(["h", "w", "h"])
[0, 2, 1]
```

```
1  def item_order(lijst):
2      return [lijst.count('s'), lijst.count('h'), lijst.count('w')]
```

Submit

Question 4: Catering bedrijf (2)

✓ Perfect

Als klanten opnieuw willen bestellen, is het handig dat ze hun vorige bestelling terug kunnen opvragen zodat ze die gewoon opnieuw kunnen bestellen. De functie `item_order` uit de vorige vraag geeft een lijst terug. Die lijst willen we nu bijhouden in een dictionary geïndexeerd op het e-mailadres van de klant. Die dictionary is eigenlijk een databank van de voorbije bestellingen. Je kan die dan makkelijk als volgt opvragen (als de dictionary als naam `orders` heeft), bijvoorbeeld

```
>>> orders["jefke@gmail.com"]
[2, 2, 1]
```

Om vanaf nu een bestelling te doen, moet de databank (de dictionary) gebruikt worden, het e-mailadres en de bestelling die binnenkomt (net zoals in de vorige vraag, bijvoorbeeld ["s", "w", "s", "h"]). Schrijf daarvoor een nieuwe functie `place_order(db,email,order)` die als volgt werkt: we starten met een lege databank en na de oproep is de bestelling (de geordende lijst van 3 items) toegevoegd aan de databank. Als er al een bestelling staat, wordt die overschreven. We houden dus enkel de laatste bestelling bij. De functie geeft niets terug.

```
>>> db
{}
>>> place_order(db,"jefke@gmail.com",["s", "w", "h", "s", "h"])
>>> db
{'jefke@gmail.com': [2, 2, 1]}
```

De functie uit de vorige vraag moet je niet opnieuw toevoegen.

```
1  def place_order(db,email,order):
2      db[email] = item_order(order)
```

Submit

Question 5: Catering bedrijf (3)

✓ Perfect

Als een klant een vorige bestelling terug wil opvragen, gebruikt die een functie `print_previous_order(db,email)`. De functie drukt de laatste bestelling af op het scherm als er een is en anders "Je hebt nog geen bestellingen geplaatst.".

```
>>> db
{}
>>> print_previous_order(db,"jefke@gmail.com")
Je hebt nog geen bestellingen geplaatst.
>>> place_order(db,"jefke@gmail.com",["s", "w", "h", "s", "h"])
>>> print_previous_order(db,"jefke@gmail.com")
Je bestelde 2 salade(s).
Je bestelde 2 hamburger(s).
Je bestelde 1 water(s).
```

```
1  def print_previous_order(db,email):
2      if len(db.keys()) == 0: print('Je hebt nog geen bestellingen geplaatst.')
3      else:
4          orders = db[email]
5          for i in range(len(orders)):
6              print("Je bestelde", orders[i], end=" ")
7              if i == 0: print('salade(s).')
8              elif i == 1: print('hamburger(s).')
9              else: print('water(s).')
```

Submit

Question 6: Craps (1)

✓ Perfect

Craps is een dobbelspel met twee dobbelstenen. In totaal zijn er 36 mogelijke worpen ((1,1), (1,2), ..., (2,1), ... (6,6)), en elke worp kan toegekend worden aan 1 set. In totaal zijn er 11 verschillende sets, namelijk voor elke mogelijk som van dobbelstenen eentje (2 t.e.m. 12).

In de volgende oefeningen bouwen we stap voor stap alle sets met alle mogelijke worpen op. Begin eerst met een functie `create_dice`, die een lijst van 13 lege sets aanmaakt (noem deze `dice`) en teruggeeft. De reden dat we in onze lijst 13 sets bijhouden (i.p.v. 11) is zodat we in de volgende oefeningen gebruik kunnen maken van het feit dat de som van de worpen gelijk is aan de index in deze lijst.

Heel belangrijk: doe dit niet: dice=[set()]*13, maar gebruik een loop om die te initialiseren.

```
1  def create_dice():
2      dice = list()
3      for i in range(13):
4          dice.append(set())
5      return dice
```

Submit

Question 7: Craps (2)

✓ Perfect

Pas je vorige functie aan en gebruik nu een loop om alle 36 combinaties van dobbelstenen te vormen. Elke combinatie wordt weergegeven door een 2-tuple. De 36 2-tuples beginnen met `(1,1)` en eindigen met `(6,6)`. De som van beide elementen is een index binnen de lijst `dice`. We willen elk 2-tuple nu toevoegen aan de correcte set binnen de lijst `dice`. Merk op dat de sets op index 0 en 1 leeg blijven. Je krijgt dan deze lijst:

```
[{},
{},
{(1,1)},
{(2, 1), (1, 2)},
{(3, 1), (1, 3), (2, 2)},
....]
```

Als we willen weten op welke manieren je 7 kan gooien, dan doe je dat als volgt:

```
>>> create_dice()[7]
{(6, 1), (5, 2), (1, 6), (4, 3), (2, 5), (3, 4)}
```

```
1  def create_dice():
2      result = [set(), set()]
3      for i in range(2, 13):
4          result.append(set())
5          for j in range(1, i if i <= 7 else 7):
6              if i - j <= 6:
7                  result[i].add((j, i - j))
8
9      return result
```

Submit

Question 8: Craps (3)

✓ Perfect

Schrijf de volgende functies, die telkens een set teruggeven.

- **lose** : je verliest indien je 2, 3 of 12 gooit. Deze functie geeft enkel de worpen terug die horen bij een 2, 3 of 12. Maak gebruik van set operaties om dat met heel weinig code te doen.
- **win** : je wint als je 7 of 11 gooit. Deze functie geeft enkel de worpen terug die horen bij een 7 of 11. Maak gebruik van set operaties om dat met heel weinig code te doen.

```
1  def lose():
2      return create_dice()[2] | create_dice()[3] | create_dice()[12]
3
4  def win():
5      return create_dice()[7] | create_dice()[11]
```

Submit

Question 9: Craps (4)

✓ Perfect

Schrijf een functie `roll_dice` die een worp simuleert. Deze functie maakt een 2-tuple aan op basis van 2 random getallen. (Hint: gebruik hiervoor `randint` uit de module `random`).

```
1  from random import randint
2
3  def roll_dice():
4      return (randint(1, 6), randint(1, 6))
```

Submit

Question 10: Craps (5)

✓ Perfect

Schrijf een (heel korte) functie `game_ends` die als input een worp meekrijgt en `True` teruggeeft indien deze worp ervoor zorgt dat het spel eindigt (indien de worp ervoor zorgt dat de speler wint of verliest).

```
1  def game_ends(worp):
2      return worp in win() | lose()
```

Submit

Question 11: Kleinste element

✓ Perfect

Schrijf een functie `minimum_elements` die een lijst van integers als input krijgt en het kleinste element teruggeeft, samen met het aantal keer dat dit element voorkomt in de lijst (geef dit terug in een tuple). Maak geen gebruik van de ingebouwde functies `min` en `sort` en schrijf ook zelf geen sorteeralgoritme. Zoek gewoon het kleinste element en tel hoeveel keer dat voorkomt.

```
1  def minimum_elements(l):
2      ke = l[0]
3      for e in l:
4          if e < ke:
5              ke = e
6
7      return ke, l.count(ke)
```

Submit

Question 12: Matrices optellen

✓ Perfect

Schrijf een functie `add_matrices` die twee matrices als input krijgt en als output de elementsgewijze som van beide matrices teruggeeft. We gebruiken de voorstelling van een matrix uit de notebooks, namelijk: een matrix is een lijst van rijen, elke rij is opnieuw een lijst van floats. Bijvoorbeeld: de 3x3 eenheidsmatrix wordt voorgesteld als: [[1,0,0],[0,1,0],[0,0,1]] Kijk ook na of de dimensies van de matrices correct zijn, geef `None` terug indien dit niet het geval is.

```
def add_matrices(m1, m2):
    for i in range(len(m1)):
        if len(m1[i]) != len(m2[i]):
            return None

    result = []
    for i in range(len(m1)):
        result.append(list())
        for j in range(len(m1[0])):
            result[i].append(m1[i][j] + m2[i][j])

    return result
```

Submit

Question 13: Matrices vermenigvuldigen

✓ Perfect

Schrijf een functie `multiply_matrices` die twee matrices als input krijgt en als output het matrix-product van beide matrices teruggeeft. We gebruiken de voorstelling van een matrix uit de notebooks, namelijk: een matrix is een lijst van rijen, elke rij is opnieuw een lijst van floats. Bijvoorbeeld: de 3x3 eenheidsmatrix wordt voorgesteld als: [[1,0,0],[0,1,0],[0,0,1]]. Kijk ook na of de dimensies van de matrices correct zijn, geef `None` terug indien dit niet het geval is.

```
def multiply_matrices(m1, m2):
    if len(m1[0]) != len(m2):
        return None

    result = []
    for i in range(len(m1)):
        result.append(list())
        for j in range(len(m1)):
            result[i].append(0)
            for z in range(len(m2)):
                result[i][j] += m1[i][z] * m2[z][j]

    return result

```

Submit