1. Speler

MatchAnalysis Deel 1 — Player

In deze oefening ga je een klasse `Player` maken die later gebruikt wordt in een grafenstructuur voor het analyseren van pasgedrag in een sportteam. We beginnen met alleen de spelerobjecten.

Wat je moet doen (4 punten)

Maak een Python-klasse `Player` met de volgende eigenschappen:

1. Instantievariabelen (0,5 punten)

- `name` *(str)*: de naam van de speler.
- `number` *(int)*: het shirt-nummer van de speler.

2. Constructor (0,5 punten)

- Initialiseert `name` en `number` met waarden die via de parameters worden meegegeven.

3. Methodes (3 punten)

- Een methode `__eq__` met als argument `other`

3. Methodes (3 punten)

- Een methode `__eq__` met als argument `other`
  - Geeft `True` terug als:
    - `other` ook een `Player` is, **en**
    - `name` van beide spelers gelijk is.
  - `number` speelt hierbij geen rol.
- Een methode `__lt__` met als argument `other` (lt staat voor less than). Vergelijkt spelers op basis **shirt-nummer** (laagste nummer eerst)
  - Retourneert True indien huidig Player object lager rugnumer heeft dan 'Other'
  - Als `other` geen `Player` is, retourneer `NotImplemented`.
- Een methode `__str__`
  - Geeft de speler weer als `"Naam (nummer)"`. Bijvoorbeeld:

```
Eden Hazard (10)
Moussa Dembele (19)
Jan Vertonghen (5)
```

Testen (1 punt)

Maak het volgende testscenario:

- Crëeer drie speler objecten en plaats ze in een lijst.
- Print één van de objecten naar de console
- Test de **eq** methode
- Test de **lt** methode door de sorted functie toe te passen op de lijst.
- Print de gesorteerde lijst

2. Pass

MatchAnalysis Deel 2 — Pass

In deze oefening ga je een klasse `Pass` maken die later gebruikt wordt in een grafenstructuur voor het analyseren van pasgedrag in een sportteam.

Wat je moet doen (2,5 punten)

Maak een Python-klasse `Pass` met de volgende eigenschappen:

1. Instantievariabelen (0,5 punten)

- `sender` — het **Player**-object dat de pass geeft.
- `receiver` — het **Player**-object dat de pass ontvangt.
- `nr_of_times` *(int)* — aantal keren dat deze pass in de wedstrijd plaatsvond.

2. Constructor (0,5 punten)

- Ontvangt `sender`, `receiver`, en `nr_of_times` als parameters en slaat deze op in de instantievariabelen.

3. Methodes (1,5 punten)

- De methode `get_weight` retourneert het aantal keren dat

3. Methodes (1,5 punten)

- De methode `get_weight` retourneert het aantal keren dat deze pass werd gegeven (`nr_of_times`).
- De methode `get_start` retourneert de `sender`.
- De methode `get_end` retourneert de `receiver`.
- De methode `__eq__`
  - Retourneert `True` als:
    - `other` ook een `Pass` is, **en**
    - `sender` en `receiver` bij beide passes gelijk zijn.
  - `nr_of_times` speelt hierbij geen rol.
- De methode `__str__` retourneert een string in het formaat:

  ```
  Pass from <sender> to <receiver>
  ```

  Bijvoorbeeld:

  - `Pass from Eden Hazard to Moussa Dembele`
  - `Pass from Jan Vertonghen to Romelu Lukaku`
  - `Pass from Divock Origi to Nacer Chadli`

Testen (0.5 punten)

Maak het volgende testscenario

- aanmaken 3 Player objecten
- aanmaken 3 Pass objecten
- print één Pass object naar de console
- testen **eq** methode
- testen get_weight methode

3. PassGraph

MatchAnalysis Deel 3 — PassGraph

In dit deel bouw je de eigenlijke **graaf** die passes tussen spelers bijhoudt en er eenvoudige analyses op uitvoert.
We gebruiken een **nabijheidslijst**-representatie: per zender (speler) bewaren we een lijst van uitgaande `Pass`-objecten.

Achtergrond

Een graaf bestaat uit **knopen** (nodes) en **verbindingen** (edges).
In onze toepassing is een knoop een **speler** en een verbinding een **pass** tussen twee spelers.

We gebruiken hiervoor een **nabijheidslijst**-representatie:

- Elke speler heeft een lijst van passes die hij heeft gegeven.
- Een pass heeft altijd:
  - een **zender** (*sender* — de speler die de bal geeft)
  - een **ontvanger** (*receiver* — de speler die de bal ontvangt)
  - een **gewicht** (*nr_of_times* — hoe vaak deze pass in

Doel

Implementeer een klasse `PassGraph` waarmee je:

- spelers kunt toevoegen,
- passes kunt registreren en ophogen,
- buren/uitgaande passes kunt opvragen,
- eenvoudige statistieken kunt berekenen, zoals *pasintensiteit* binnen (een deel van) het team.

Gegeven

Je hebt uit Deel 1 en Deel 2:

- `Player(name: str, number: int)`
- `Pass(sender: Player, receiver: Player, nr_of_times: int)` met:
  - `get_weight()` → `nr_of_times`
  - `get_start()` → `sender`
  - `get_end()` → `receiver`
  - `__eq__` vergelijkt enkel op `sender` en `receiver`
  - `__str__` geeft `"Pass from <sender> to <receiver>"`

Wat je moet bouwen (11 punten)

Klasse: `PassGraph`

1) Interne representatie (1 punt)

- `players` — lijst met alle spelers in de graaf.
- `adj` — dict met als key **sender-naam** en als waarden een lijst van `Pass`-objecten die **van die sender** vertrekken.

2) Basisoperaties (6 punten)

- methode `add_player` met `player` als argument
  - Voeg `player` toe als er nog **geen speler** met dezelfde `name` bestaat.
  - Zorg dat er een lijst voor `player` oject bestaat in adj.
- methode `has_player` met als argument `speler` (object of naam).
  - Retourneer `True` als de speler (of naam) in de graaf zit.
  - Ondersteun zowel een `Player`-object als een `str` (naam).
- methode `get_player` met als argument `name`
  - Zoek en retourneer de `Player` met deze naam, of `None` als die niet bestaat.

- methode `add_pass` met als argumenten `sender` (Player object), `receiver` (Player object) en `times` (int - default = 1).
  - Vereist dat **beide spelers** al aan de graaf zijn toegevoegd.
  - Als er al een pass `sender → receiver` bestaat, **verhoog** dan `nr_of_times` met `times`.
  - Anders maak je een nieuwe `Pass(sender, receiver, times)` en voeg je die toe aan `adj` bij de correcte speler.
  - als `times <= 0`, **negeer** of werp `ValueError`
- methode `get_pass` met als argumenten `sender_name` (str) en `receiver_name` (str) .
  - Geef de pass terug als die bestaat; anders `None`.
- methode `neighbors` met als argument `sender_name` (str)
  - Retourneer de **lijst uitgaande passes** van deze zender (lege lijst als onbekend of geen uitgaande passes).

3) Analysefuncties (4 punten)

- methode `total_weight` met als argument `subset` een lijst van **spelernamen**
  - Som van `nr_of_times` over alle passes waarvan

3) Analysefuncties (4 punten)

- methode `total_weight` met als argument `subset` een lijst van **spelernamen**
  - Som van `nr_of_times` over alle passes waarvan **zender én ontvanger** in `subset` zitten.
  - Als `subset` `None` is: neem **alle** spelers.
  - `subset` is een lijst **spelernamen**.
- methode `pass_intensity(self, subset: list[str] | None = None) -> float` met als argument `subset` een lijst van **spelernamen** Deze methode berekent hoe sterk een groep spelers **onderling** met elkaar samenpasst.

  De intensiteit wordt gedefinieerd als:

  $$\text{pass\_intensity} = \frac{\text{totaal aantal passes binnen de subs}}{\text{maximaal mogelijk aantal gerichte passes binn}}$$

  - **Teller (numerator):** de som van alle `nr_of_times` voor passes waarbij zowel de zender als de ontvanger binnen de subset van spelers zitten.
  - **Noemer (denominator):** het maximaal aantal mogelijke **gerichte** passes binnen de subset.
    - Voor (n) spelers in de subset kan elke speler naar (n-1) andere spelers passen.
    - Dus de noemer is (n * (n-1)).

Speciale gevallen

- Als de subset minder dan 2 spelers bevat ((n < 2)), is de intensiteit gedefinieerd als `0.0`.

Voorbeeld

- Bij een subset van 3 spelers zijn er maximaal (3 * 2 = 6) mogelijke gerichte passes.
- Stel dat er in werkelijkheid 9 passes tussen deze 3 spelers gebeurden, dan is de intensiteit (9 / 6 = 1.5).
  (De waarde kan dus groter zijn dan 1, aangezien passes meerdere keren kunnen voorkomen.)

methode `top_pairs` met argument k (int, default = 5)

- Retourneer de **top k** passes als lijst met de hoogste `nr_of_times` (globaal).

methode `distribution_from` met als argument sender_name (str)

- Retourneer voor een zender een lijst `(receiver_name, count)` gesorteerd **dalend op count**.
- Handig om te zien **naar wie** een speler het meest passt.

Opmerkingen

- Methoden die namen verwachten moeten robuust omgaan met **onbekende namen**:
  - Geef `None` of `[]` terug (zoals gespecificeerd hierboven), maar **gooi geen exceptie** in normale query-methodes.
- Ga ervan uit dat **spelernamen uniek** zijn binnen dezelfde `PassGraph`.

Testen (1 punt)

Maak het volgende testscenario

- aanmaken PassGraph object
- aanmaken 4 spelers
- voeg spelers toe aan PassGraph object
- voeg 6 passen toe
- test de verschillende analyse methodes

4. Opslaan en inlezen

MatchAnalysis Deel 4 — Opslaan en inlezen

In dit deel voeg je functies toe om de **PassGraph** op te slaan naar een **.txt**-bestand en om opnieuw in te lezen.

Bestandsformaat

Twee secties: `[PLAYERS]` en `[PASSES]`.

```
[PLAYERS]
<name>;<number>
<name>;<number>
[PASSES]
<sender_name> -> <receiver_name> : <nr_of_times>
```

Regels

- Commentregels beginnen met `#` en worden genegeerd.
- Lege regels worden genegeerd.
- Dubbele passes → tel gewichten op.
- Ongeldige input → `ValueError`.

Voorbeeldbestand

```
Team: Rode Duivels
[PLAYERS]
Eden Hazard;10
```

```
Eden Hazard;10
Moussa Dembele;19
Jan Vertonghen;5
Romelu Lukaku;9
[PASSES]
Eden Hazard -> Moussa Dembele : 7
Eden Hazard -> Romelu Lukaku : 2
Jan Vertonghen -> Romelu Lukaku : 1
```

Wat je moet doen (5 punten)

Breid je `PassGraph`-klasse uit met:

1) Basisoperaties (1 punt)

- methode `players`
  - Geef een **kopie** van de spelerslijst terug.
- methode `passes(self)`
  - Geef **alle** passes in de graaf terug.

2) Constructor met pad-parameter (2 punten)

Signatuur De constructor heeft als argument een path_naam(Str).

Gedrag

Gedrag

- Zonder path maakt de constructor een lege graaf.
- Met path leest de constructor onmiddellijk het tekstbestand in en vult de graaf.

Inlees-eisen

- Ondersteun commentregels (# ...) en lege regels.
- Herken secties [PLAYERS] en [PASSES]; elke andere sectie → ValueError met korte boodschap.
- In [PLAYERS]: elke regel is name;number. Spaties rond velden weghalen met strip(). number moet naar int geconverteerd kunnen worden. Voeg spelers éérst toe met add_player.
- In [PASSES]: elke regel is sender_name -> receiver_name : nr. Spaties rond velden weghalen. nr moet een positief geheel getal zijn. Voeg passes toe met add_pass. Als dezelfde pass meerdere keren voorkomt, moet het totaal kloppen.
- Als een pass verwijst naar een speler die niet in [PLAYERS] staat → ValueError.

3) save_to_txt(self, path) (1 punt)

Slaat de volledige graaf op in exact het hierboven beschreven formaat.

Eisen

- Overschrijf het doelbestand indien het bestaat.
- Schrijf eerst [PLAYERS] (één speler per regel in de vorm name;number).
- Schrijf daarna [PASSES] (één pass per regel in de vorm sender -> receiver : nr).
- Volgorde mag je kiezen (bijv. spelers alfabetisch, passes per zender), maar wees consistent.

Testen (1 punt)

Schrijf een testscript dat:

- Een PassGraph opbouwt (minstens 4 spelers en 4 passes, waarvan 1 pass meerdere keren wordt toegevoegd).
- save_to_txt("team.txt") aanroept.
- Met de constructor een nieuwe graaf inleest: g2 = PassGraph("team.txt")