Toepassingsopdracht

Tom Hofkens

1 Algemene informatie

1.1 Toepassingsopdracht: verdeling

De toepassingsopdracht bestaat uit twee grote delen: een individueel deel en een groepsdeel, elk op 5 punten.

1.1.1 Individueel deel

Dit deel bestaat enerzijds uit de implementatie van alle basisstructuren (1 punt) en 1 geavanceerde structuur (4 punten). Meer informatie volgt in de latere opdrachten. De basisstructuren zijn een stack, een queue, een ketting en een binaire zoekboom.

1.1.2 Groepsdeel

Het groepsgedeelte bestaat uit de implementatie van een reservatiesysteem zoals in de probleemstelling beschreven staat. De nadruk ligt daar op goede documentatie, goede contracten en flexibiliteit. Uiteraard ga je daar de structuren uit het individuele deel gebruiken.

1.1.3 Deadlines

We werken met soft en hard deadlines. Soft deadlines zijn niet verplicht. Ze zorgen ervoor dat het werk gespreid wordt over de verschillende weken. Je doet er goed aan om minstens een deel van de oplossing in te dienen tegen de soft deadline, maar je mag er achteraf nog aan verder werken. Hard deadlines liggen vast, zijn verplicht en moeten gevolgd worden.

1.1.4 Al doende leren

Het is geen probleem dat je fouten maakt bij TOg. In software ontwikkeling wordt er meestal met iteraties gewerkt. In elke iteratie komt er functionaliteit bij en wordt bestaande code herschreven. Dat werkt veel beter dan alles van de eerste keer goed te doen. Dat is ook volledig in lijn met het onderwijs paradigma 'al doende leren'. Je zal dus opdrachten uitvoeren die achteraf foutief blijken te zijn. Dat is absoluut geen probleem. Integendeel, net door die fouten te maken, ga je die in de toekomst niet meer maken. En dat is de kern van de zaak bij 'al doende leren'.

Terwijl je werkt kan je individuele feedback verwachten via INGinious. Je dient daar een opdracht in en krijgt dan automatische feedback. Van zodra we starten met het groepswerk, ga je feedback geven op het werk van anderen (en ontvang je dus ook feedback) op basis van een feedbackformulier dat informatie zal bevatten van dingen waar je misschien al wel of misschien nog niet aan gedacht hebt. Je krijgt hierdoor ook een idee wat voor soort vragen je kan verwachten op het project-examen.

2 Opdracht 1: De eerste ADT's

We stellen een ADT op voor de stack en queue.

1. Maak een nieuw project aan in PyCharm.
2. Maak een python bestand per ADT. Start de naam met een hoofdletter en zet er My voor. Je krijgt dus MyStack.py en MyQueue.py. Dit is heel belangrijk omdat Python zelf een eigen implementatie heeft. Anders ga je conflicten krijgen.
3. Zet bovenaan in commentaar waarvoor je een ADT opstelt.
4. Welke data bevat jouw ADT? Maak variabelen met goede namen aan en initialiseer die op goede waarden.
5. Welke functionaliteit gaat jouw ADT aanbieden? Voeg een contract toe aan elk ADT. Een contract bestaat uit verschillende functies. Een contract bestaat uit volgende zaken:
   - de naam van de functie,
   - de input parameter(s) met uitleg,
   - de beschrijving van de functie (in een docblock),
   - de output parameter(s) (je geeft die terug via een return van een tuple),
   - de pre- en postcondities. Precondities zijn voorwaarden waar de oproeper van je methode zich aan moet houden (bv n moet een strikt positief geheel getal zijn). Een postconditie is een belofte die je maakt die waar is nadat de methode werd opgeroepen (bv er zit nu 1 element meer in de lijst).

   Je doet dit door een Python functie te definiëren met als body 'pass'. Dat is een functie die op dit moment nog niets doet, maar dan krijg je geen errors. Je kan je baseren op de contracten in de cursus.

2.1 Een voorbeeld

Merk op dat dit slechts een voorbeeld is dat zeker niet volledig is, maar in dit voorbeeld komt alles aan bod wat nodig is telkens toegepast op dat voorbeeld. Vergeet ook niet dat het hier maar over 1 functie gaat en je er vermoedelijk meerdere zal nodig hebben per ADT.

```
# ADT Stack
## data
items = []

## functionaliteit
def push(item):
  """
  voeg een item toe aan de top van de stack

  preconditie: er is nog plaats op de stack
  postconditie: de stack is 1 item groter en de top bevat het toegevoegde
    item

  :return: geeft True terug asa het toevoegen gelukt is
  """
  pass
```

3 Opdracht 2: Van ADT naar klasse

Ondertussen heb je bij Inleiding Programmeren het concept klasse gezien. Een klasse is een implementatie van een ADT aangezien het ook zowel data als functionaliteit combineert.

Voer nu volgende taken uit.

1. Zet elk ADT om in een klasse en elke functie in een methode door de bestanden MyStack.py en MyQueue.py aan te passen. Voorlopig moeten die nog niets doen.

2. In de cursus wordt er een onderscheid gemaakt tussen in en out parameters. in parameters zijn de gewone parameters van een functie. De out parameters geeft je altijd terug via een return. Als dat er meerdere zijn, gebruik je een tupel.

3. Zorg dat je de benamingen uit de cursus gebruikt en de juiste parameters in juiste volgorde (incl. de out parameters). De create mag je vervangen door de constructor (initializer) en de destroy mag je laten vallen.

4. Implementeer nu de methodes van de queue en de stack. Merk op:

   - dequeue: Bij de queue implementeer je de dequeue uit de cursus die het item ook teruggeeft en dus niet de versie die die enkel verwijdert.
   - pop: Bij de stack implementeer je de pop uit de cursus die het item ook teruggeeft en dus niet de versie die die enkel verwijdert.
   - Bij de stack en de queue mag je kiezen uit een linkbased of een arraybased implementatie (maar geen lijstimplementatie; je mag dus niet een Lijst implementeren en die gebruiken als stack). Als je kiest voor een arrayimplementatie, mag je geen operaties van de Python list gebruiken (dus geen append, + operator, ...). Je constructor krijgt een max_size parameter mee en je maakt een lege list van vaste lengte als volgt: `a = [None] * max_size`.

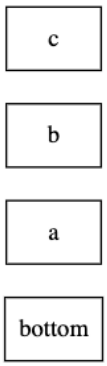
5. Implementeer een save methode die de stack en queue voorstelt als een list in Python (zie verder voor de correcte output).

6. Implementeer een load methode die een leeg object aanmaakt en vult met een list. De list is hetzelfde als de output van een save. Je doet dat voor beide datastructuren.

7. Dien alles in via INGinious.

3.1 Een stack

De volgende stack ...



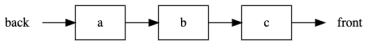
... wordt omgezet naar volgende list via de methode save.

```
[a,b,c]
```

De top van de stack staat dus achteraan.

3.2 Een queue

Volgende queue ...



... wordt omgezet naar volgende list via de methode save.

```
[a,b,c]
```

Er wordt dus vooraan toegevoegd en achteraan verwijderd. Als we aan dit voorbeeld een item d toevoegen via enqueue, dan komt er dit.

```
[d,a,b,c]
```