Inleiding Programmeren (Python)

Inleiding Programmeren (Python)	
Richting	Informatica
Jaar	1BINF

Bespreking

Voor 2011 werd dit vak in de programmeertaal Oberon gegeven.

In 2015-2016 wordt dit vak gedoceerd door Jon Sneyers, hij geeft les op een droge manier. Verder kan er niets over gezegd worden wegens gebrek aan ervaring met hem.

Voordien werd dit vak door de speelse professor Goethals gegeven. Zijn lessen zijn dus heel ludiek en over het algemeen een plezier om te volgen maar hier mag je je niet door laten afleiden van de moeilijkheidsgraad van het vak. Er zijn mensen die al wat konden programmeren en die zal het in het begin iets gemakkelijker afgaan. Echter zelfs zij moeten opletten, want programmeren wordt hier op een andere, formelere manier bekeken als de doorsnee hobbyist dit doet.

Zorg zeker dat je dit vaak slaagt of je geraakt in de problemen voor de rest van je Bachelor aangezien je nog heel wat zult moeten programmeren waarbij ze verwachten dat je dit al perfect kan. Maak er voor jezelf een uitdaging van en je zult merken dat het des te interessanter en leuker wordt.

Theorie

Zoals hierboven gezegd leuk gegeven maar belangrijke theorie. Je examen voor dit deel zal uit een 5-tal vragen bestaan die nakijken of je wel weet hoe een programma ineen zit en wat de filosofie erachter een beetje is. Lees zeker het eerste hoofdstuk van je cursus goed na.

Praktijk

Op het examen zal je al je vaardigheden moeten combineren in één groter programma dat verschillende dingen zal moeten kunnen. Leer vlotjes met functies omgaan en de verschillende datatypes (de subtiliteiten van lijsten etc). Ook is het mogelijk dat je een bestand moet kunnen inlezen dus zorg dat je hiermee geen tijd verspilt en bereid je grondig voor.

Naast je examen zijn er ook de opdrachtjes doorheen het jaar. Deze staan op de helft van je praktijkpunten en zijn je hoofdmethode om het vak in te werken/studeren. Zeker de moeite waard om hiermee serieus om te gaan zodat je des te minder werk hebt tegen het examen.

Puntenverdeling

Theorie: 10/20. Praktijk: 5/20. Taken: 5/20.

Examenvragen

Academiejaar 2019 - 2020 - examen oktober

Voor het examen kreeg je deze pdf en de Python code hieronder. Bestand:Python-oktober-2019.pdf

```
# TERUGGEWERKT VAN MIJN EIGEN OPLOSSING NAAR DE ORIGNELE FILE VAN HET EXAMEN

from datetime import datetime
import time


# Objecten van klasse message stellen boodschappen voor
# - sender: object van type account ; de verzender van de boodschap
# - receiver: object van type account ; de ontvanger van de boodschap
# - text: de inhoud van de boodschap
class message:

    # 1.1.2 CODE UITBREIDEN

    # Initializer ; de nieuwe message krijgt sender, receiver en text
    def __init__(self, sender, receiver, text):
        self.sender=sender
        self.receiver=receiver
        self.text=text

    # Print de inhoud van de boodschap af;
    # dat is: de verzender en de boodschap.
    # de ontvanger wordt niet afgedrukt
    def display(self):
        print(self.sender.name+" : "+self.text)


# Objecten van deze klasse stellen de account van een gebruiker voor
# De account bevat volgende member variabelen:
# - name: de naam van de accounthouder
# - inbox: lijst van ontvangen berichten in volgorde van ontvangst
class account:
    # Initializer; nieuwe account met naam name
    def __init__(self,name):
        self.name=name
        self.inbox=[]   # We beginnen met een lege inbox

    # voeg een boodschap m toe aan de inbox
    def addMessage(self,m):
        self.inbox.append(m)


    # print alle boodschappen af in de inbox van deze account
    def displayMessages(self):
        print("Messages of "+self.name)
        for m in self.inbox:
            m.display()

    def removeMessagesFrom(self,name):
        for m in self.inbox:
            if m.sender == name:
                self.inbox.delete(m)


# Een object van deze klasse stelt een berichtenservice voor.
# De berichtenservice is verantwoordelijk voor het bewaren van alle accounts,
```

```
# de versturing van nieuwe boodschappen (ervoor zorgen dat ze in de juiste
# inbox terechtkomen), opvragen van boodschappen, etc.
# Members:
# - accounts: dictionary die gebruikersnamen koppelt aan hun account.
#       bijvoorbeeld: accounts["Jan"] is het account object dat werd aangemaakt
voor gebruiker "Jan"
class messageService:
    # Initialiseer de berichtenservice
    def __init__(self):
        self.accounts=dict()

    # maak een nieuwe account aan voor gebruiker "name"
    def createAccount(self,name):
        a=account(name)
        self.accounts[name]=a
        return a

    # Geef de account van gebruiker name
    def getAccount(self,name):
        return self.accounts[name]

    # voeg een nieuwe boodschap toe van sender naar receiver met text als
boodschap
    def newMessage(self,sender,receiver,text):
        s=self.accounts[sender]
        r=self.accounts[receiver]
        m=message(s,r,text)
        r.addMessage(m)
        return m   # Geef de message terug die aangemaakt werd


    # Geef een lijst met alle boodschappen aan gebruiker name terug
    def getMessages(self,name):
        return self.accounts[name].inbox

    # Druk alle boodschappen van gebruiker name af
    def displayMessages(self,name):
        self.accounts[name].displayMessages()


M=messageService()
M.createAccount("Toon")
M.createAccount("Stephen")
M.createAccount("Tom")

M.newMessage("Toon","Tom","Will you be present at the exam?")
M.newMessage("Tom","Toon","Sorry, can't be; too busy.")
M.newMessage("Toon","Stephen","Can you be present at the exam? Tom can't.")
M.newMessage("Stephen","Toon","Sure!")
M.newMessage("Stephen","Tom","Could you help grading?")
M.newMessage("Tom","Stephen","OK, np")


M.displayMessages("Toon")
messages_of_toon=M.getMessages("Toon")
```

```
print(datetime.now())
```

Academiejaar 2015 - 2016 - examen november

Theorie

Elke vraag stond op 5 punten

1. Wat is *polymorfisme* en *duck typing*?
2. Wat is het verschil tussen *'deep copy* en *'shallow copy'*? Leg dit uit aan de hand van een voorbeeld.
3. Wat is *inheritance*? Geef enkele voordelen hiervan.
4. Geef het verschil tussen een *'break'*, *'return'* en *'continue'* statements.

Praktijk

DOELSTELLING
bedoeling van deze opdracht is dat je een herbruikbare object-georiënteerde oplossing met een recursief algoritme schrijft dat alle anagrammen berekent van een datastructuur.

OPDRACHT
Implementeer een programma voor het inlezen, het recursief aanmaken en het weergeven van de anagrammen. Lees eerst alle opdrachten aandachtig voor je begint te implementeren. De opdracht staat op een totaalscore van 50.

1. [2 punten] Maak een klasse *Woord* aan met een attribuut *woord*, dat in de constructor geïnitialiseerd wordt.
2. [4 punten] Maak de volgende methodes aan in *Woord*
    - een methode die de lengte teruggeeft van het *Woord*
    - een methode die het *Woord* en een ander gegeven *Woord* achter elkaar toevoegt en het resultaat (ook een *Woord*) teruggeeft
    - een methode die het een element of subsequentie (slice) van het *Woord* teruggeeft, gegeven een begin-index en een optionele eind-index.
3. [4 punten] Gebruik voor de drie bovenstaande methodes operator overloading.
    - De methode die de lengte teruggeeft moet len heten zodat ze opgeroepen wordt bij het uitvoeren van de len-functie met een Woord als parameter.
    - Zorg ervoor dat het achter elkaar toevoegen wordt ge ¨ımplementeert als de +-operator.
    - De methode die een subsequentie teruggeeft moet getitem heten zodat ze opgeroepen wordt bij het toepassen van de index-operator (vierkante haakjes) voor het opvragen van een element of slice. Voor een instantie Woord-w moet het mogelijk zijn om de volgende expressies uit te voeren: w[1], w[:2], w[1:3], w[1:], etc. getitem heeft een parameter index, dat ofwel van het type int ofwel van het built-in type slice (als : gebruikt wordt om te slicen) is. Je kan de ondergrens van je slice terugkrijgen via zijn attribuut start en de bovengrens via stop.

4. [8 punten] Implementeer een methode anagram die alle anagrammen van het Woord teruggeeft (dit zijn ook Woord-instanties). Je mag in deze methode nooit rechtstreeks gebruik maken van het veld woord: je mag ´enkel gebruik maken van je zelf ge ¨ımplementeerde methodes om zo de datastructuur te manipuleren.
5. [8 punten] Implementeer het algoritme om alle anagrammen te zoeken op een recursieve manier. Je mag een hulpmethode aanmaken als je dat nodig vindt.
6. [1 punt] Maak een file main.py aan. Gebruik de klasse Woord in deze file door een instantie aan te maken met het woord “bal” als woord, en de methode anagram op te roepen. Print de resulterende anagrammen af.
7. [12 punten] Maak een klasse Interview aan die een subklasse van Woord is, die alle anagrammen in een interview kan berekenen. Een Interview heeft een veld zinnen dat een lijst is van strings, waarbij elke vraag en elk antwoord een element is. Het Interview hierboven bestaat dus uit zes elementen (drie vragen en drie antwoorden). Zorg ervoor dat in de klasse Interview de methodes voor de index-operator, optelling en lengte zo worden gedefinieerd dat het oproepen van anagram het gewenste gedrag heeft waarbij een interview bestaande uit zes strings de juiste 6 anagrammen berekent.
8. [5 punten] Lees de zinnen van het interview in uit een bestand dat je zelf aanmaakt, waarin elke zin van het interview op een nieuwe lijn begint.
9. [1 punt] Gebruik de klasse Interview door een interview in te lezen, en de methode anagrams op te roepen. Doe dit weer in de file main.py. Print de resulterende anagrammen af.
10. [5 punten] documenteer je code!

VOORBEELD

```
input_word = Woord(”bal”)
print(input_word.anagram())
```

Bovenstaande code geeft onderstaande output:

```
[bal, bla, abl, alb, lba, lab]
```

Academiejaar 2011 - 2012 - 2de zittijd

Theorie

1. Waarvoor wordt import gebruikt in een class? Leg uit en als je 1 van de volgende woorden gebruikt, leg dan ook uit: *function*, *module*, *package*.
2. Kan je elk recursief programma omzetten naar een equivalent programma zonder recursies?
3. Leg volgende programmacode uit:

```
def function(v):
print(v)
v[0] = 42
print(v)
print(v[:-1])

answer = [0, 1, 2, 3]
function(answer)
print(answer)
```

1. Definieer en leg uit 'self' in een class, gebruik maximaal 10 lijnen.
2. Duid in volgende programmacode de fouten aan en verbeter:

```
def isPaly(x):

n = length(x)

while i in range(n):
if x[i] !== x[-i]
return false
else:
return True
```

Academiejaar 2011 - 2012 - 1ste zittijd

Praktijk

Opgave: Bier Sam, Ruben en Bart hebben hun ware roeping gevonden en willen graag samen een café openen in de studentenbuurt van Wilrijk. Omdat ze niet over één nacht ijs gaan, hebben ze eerst een marktonderzoek laten doen. Aan een hoop Wilrijkse studenten werd gevraagd welke bieren ze graag drinken en welke café's ze soms bezoeken. Er werd ook nagegaan welke bieren elk café verkoopt. Ze hebben van het onderzoeksbureau van de UA een bestand gekregen met daarin de resultaten van het onderzoek. Het is aan jou om daaruit een aantal interessante zaken te berekenen. Het input-bestand voor je programma ziet eruit als volgt:

```
Seppe likes Leffe
Tim likes Leffe
...
Tim visits DenEngel
Tim visits Kelderke
Noah visits Plantsoen
...
DenEngel serves Duvel
Plantsoen serves Duvel
Plantsoen serves Westmalle
...
```

Je kan hier drie entiteiten (mensen, bieren en café's) en drie relaties (likes, visits en serves) onderscheiden. Elke entiteit wordt uniek geïdentificeerd door een string (zonder spaties). Merk op dat niet elk café noodzakelijkerwijs bezocht wordt, dat niet elke persoon per se café's bezoekt, dat niet elk bier in een café verkocht wordt, ... Je maakt een programma dat in staat is om een bestand in bovenstaand formaat in te lezen. Je programma bestaat uit twee modules: *beer* en *testbeer*.

Module beer

In de module *beer* maak je een klasse *Beer* die onderstaande methoden implementeert. Opgelet: geef de methoden exact dezelfde namen en parameters als in de opgave gespecifieerd. Als een methode die een lijst teruggeeft geen resultaat heeft, dient een lege lijst teruggegeven te worden.

1. `__init__(file)` Initialiseert de klasse met een bestandsnaam, en slaat de gegevens uit het bestand op in een interne datastructuur.
2. `(*)get_bars()` Geeft een gesorteerde lijst terug met de namen van de beschikbare café's, zonder dubbels.
3. `(*)get_people()` Geeft een gesorteerde lijst terug met de namen van de ondervraagde studenten, zonder dubbels.
4. `(*)get_beers()` Geeft een gesorteerde lijst terug met de namen van de geserveerde of gedronken biersoorten, zonder dubbels.
5. `(*)get_beers_liked_by(person)` Geeft een gesorteerde lijst terug met de namen van de bieren, zonder dubbels, die de meegegeven persoon drinkt.
6. `(*)get_bars_serving(beer)` Geeft een gesorteerde lijst terug met de namen van de café's, zonder dubbels, die het gegeven bier serveren.
7. `(*)get_beers_served_at(bar)` Geeft een gesorteerde lijst terug met de namen van de bieren, zonder dubbels, die in het gegeven café geserveerd worden.
8. `(*)get_bars_visited_by(person)` Geeft een gesorteerde lijst terug met de namen van de café's, zonder dubbels, die de meegegeven persoon bezoekt.
9. `(*)get_people_visiting(bar)` Geeft een gesorteerde lijst terug met de namen van de studenten, zonder dubbels, die het gegeven café bezoeken.
10. `(*)get_beers_served_at_bars_visited_by(person)` Geeft een gesorteerde lijst terug met de namen van alle bieren, zonder dubbels, die geserveerd worden in café's die bezocht worden door de meegegeven persoon.
11. `(**)get_barmates_of(person)` Geeft een gesorteerde lijst terug met de namen van alle personen, zonder dubbels, die een café bezoeken dat ook door de meegegeven persoon bezocht wordt. Een persoon is uiteraard geen *barmate* van zichzelf.
12. `(**)get_bars_with_the_most_beers()`
13. `(**)get_average_number_of_beers_served()`
14. `(**)get_beers_served_nowhere()` Geeft een gesorteerde lijst terug van bieren, zonder dubbels, die door minstens één student gedronken worden maar die in geen enkel café verkrijgbaar zijn.

15. `(***)get_pals_of(person)` Geeft een gesorteerde lijst terug met de namen van alle personen, zonder dubbels, die alle café's bezoeken die ook door de meegegeven persoon bezocht wordt. Een persoon is uiteraard geen vriend van zichzelf.
16. `(***)get_interesting_bars_for(person)` Geeft een gesorteerde lijst terug met de namen van de café's die minstens één bier serveren dat de meegegeven persoon drinkt en die nog niet bezocht worden door de meegegeven persoon.
17. `(***)get_lucky_people()`Geeft een gesorteerde lijst terug met de namen van alle personen die minstens één bier drinken en die elk bier dat ze lekker vinden kunnen drinken in een café dat ze bezoeken.

Module testbeer

De module *testbeer* test alle geïmplementeerde functionaliteit van de Beer-klasse. Je maakt een Beer-object aan met het meegeleverde bestand *fbeer.txt*, en print voor verschillende input-waarden de terugkeerwaarde van alle methoden af. Werk incrementeel: schrijf onmiddellijk een test voor elke nieuwe methode die je implementeert in je Beer-klasse. Geef ook veel output bij het testen! Merk op dat je de methoden *get-bars()*, *get_people()* en *get_beers()* kan gebruiken om inputwaarden voor andere methoden te genereren.