Algemeen

Gebruik van `self`

Als je werkt met klassen, dan is het belangrijk dat je onderscheid maakt tussen twee soorten variabelen. Er zijn objecteigenschappen die kunnen aangesproken worden in alle klassemethoden en er zijn lokale variabelen die enkel toegankelijk zijn binnen de methode waarin ze gedefinieerd worden. Enkele de namen van de objecteigenschappen moeten voorafgegaan worden door `self`. De lokale variabelen van een methode moeten niet voorafgegaan worden door `self`, en het getuigt van een zeer slechte programmeerstijl als je dat toch doet.

Objecteigenschappen initialiseren in de intialisatiemethode

Voor je begint aan de implementatie van een klasse, moet je eerste bepalen welke eigenschappen de objecten van die klasse moeten hebben. Elk van die eigenschappen komt dan overeen met een variabele die voorafgegaan wordt door de prefix `self.`. Deze variabelen beschrijven de interne toestand van de individuele objecten en kunnen in alle methoden van de klasse gebruikt worden. Het is altijd een goed idee om deze objecteigenschappen te definiëren in de `__init__` methode en er daar een initiële waarde aan toe te kennen.

Panini

Algemene info

Gegevenstype controleren met `isinstance`

Om na te gaan of een gegeven object een bepaald gegevenstype heeft, kan je natuurlijk gebruik maken van de ingebouwde functie `type(o)` die het gegevenstype van het object `o` teruggeeft. Het is echter beter om hiervoor de ingebouwde functie `isinstance(o, t)` te gebruiken. Deze functie geeft een Booleaanse waarde terug die aangeeft of het object `o` al dan niet behoort tot het type `t`.

```
>>> type(3) == int
True
>>> isinstance(3.14, int)
False
>>> isinstance(3.14, float)
True
>>> isinstance([1, 2, 3], list)
True
```

De vijfde kaart

Algemene info

Vergelijkingsoperatoren

Om twee objecten van een zelf gedefinieerd type te kunnen vergelijken, moeten de vergelijkingsoperatoren geïmplementeerd worden. Dit kan door volgende methodes te overschrijven.

Methode	Operator
___lt___	$<$
___le___	$\leq$
___gt___	$>$
___ge___	$\geq$
___eq___	$=$
___ne___	$\neq$

Let er op dat in veel gevallen de meeste operatoren geïmplementeerd kunnen worden met behulp van een andere operator.

Huffman codering

Specifieke info

Naast recursieve functies (functies die een oproep naar zichzelf bevatten) bestaan er ook recursieve datastructuren. Een voorbeeld daarvan is een gelinkte lijst. Dit is een lijst waaruit je van ieder element alleen zijn buur kunt opvragen. Het toevoegen en verwijderen van elementen in een gelinkte lijst is zeer efficiënt, maar het nadeel is dat voor het opvragen van het i-de element vaak alle elementen op positie $0, \ldots, i-1$ overlopen moeten worden. Hieronder zie je een mogelijke implementatie van zo'n gelinkte lijst in python. Hierbij bevat ieder element een verwijzing naar zijn buur, als die niet bestaat (het is het laatste element uit de lijst) is deze None.

```
class Element:

    def __init__(self, inhoud, volgende = None):
        self.inhoud = inhoud
        self.volgende = volgende

    def getInhoud(self):
        return self.inhoud

    def voegToe(self, volgende):
        self.volgende = volgende
        return self.volgende

    def gaVerder(self):
        return self.volgende
```

Dan kan er als volgt met deze datastructuur gewerkt worden.

```
# We maken twee nieuwe element aan om in onze lijst te stoppen
>>> eerste = Element(1)
>>> tweede = Element(2)

# We linken het tweede aan het eerste
>>> eerste.voegToe(tweede)
<__main__.Element object at 0x10298a7f0>
>>> eerste.getInhoud()
1
```

```
>>> eerste.gaVerder().getInhoud()
2
>>> eerste.gaVerder() == tweede
True
>>> eerste.gaVerder().gaVerder()
None

# We voegen nog twee element toe aan overlopen vervolgens de volledige lijst
>>> derde = Element(3)
>>> vierde = Element(4)
>>> tweede.voegToe(derde).voegToe(vierde)
<__main__.Element object at 0x10298a7e0>
>>> huidig = eerste
>>> while huidig is not None:
...     print(huidig.getInhoud())
...     huidig = huidig.gaVerder()
...
1
2
3
4
```

Mad Libs

Algemene info

Opmerkingen

Stringmethoden `islower()` en `isupper()`

De stringmethode `islower` kijkt of *alle* tekens van een string kleine letters zijn. De stringmethode `isupper` controleert of *alle* letters van een string hoofdletters zijn.

```
>>> 'ABCD'.lower()
False
>>> 'ABCD'.upper()
True
>>> 'abcd'.lower()
True
>>> 'abcd'.upper()
False
>>> 'AbCd'.lower()
False
>>> 'AbCd'.upper()
False
```

Stringmethode `capitalize()`

De stringmethode `capitalize` kan gebruikt worden om het eerste karakter van een string om te zetten naar een hoofdletter (als het een letter is), en alle overige karakters naar kleine letters (als het letters zijn).

```
>>> 'aBcD'.capitalize()
'Abcd'
```