Opgaven

De programmeertaal Python

at computing

The Linux/UNIXperts

Nijmegen - Utrecht
www.ATComputing.nl

Versie: 12f

Voorbereiding

Voordat je aan de opgaven begint, moet je eerst de juiste versie van de bestanden klaarzetten. Je kunt daarvoor het volgende commando intypen:

```
selectie P3
```

om de opgaven uit deze cursus met Python 3 te maken.

Je kunt eventueel kiezen om de opgaven uit deze cursus met Python 2 te maken. Dan typ je het commando

```
selectie P2
```

Editors

De opgaven worden gemaakt in een Linux-omgeving. Je hebt diverse tekst-gebaseerde editors ter beschikking, zoals `vim`, `emacs` and `nano` .
Als je deze editors niet kent, kun je een grafische editor gebruiken. Zo'n grafische editor kun je starten via het icoon op je desktop.

Nuttige Linux commando's

`ls`	Toon alle bestandsnamen in de huidige directory.
`ls -l`	Toon alle bestandsnamen met bijbehorende kenmerken in de huidige directory.
`cd` *`dirx`*	Ga naar de directory genaamd *`dirx`* (wordt de huidige directory).
`cd`	Ga naar mijn home directory (wordt de huidige directory).
`mkdir` *`diry`*	Maak een nieuwe directory genaamd *`diry`* .
`cat` *`filex`*	Toon de inhoud van een bestand genaamd *`filex`* (past mogelijk niet op één scherm).
`less` *`filex`*	Toon de inhoud van een bestand genaamd *`filex`* scherm-voor-scherm. Gebruik de spatiebalk om naar een volgend scherm te gaan, de pijltjestoetsen om een regel naar beneden/boven te gaan, typ "`q`" om te eindigen).
`rm` *`filex`*	Verwijder het bestand genaamd *`filex`* .
`rm -r` *`dirx`*	Verwijder de directory genaamd *`dirx`*, inclusief alle bestanden en subdirectories daaronder!

Opgave 1: Strings, expressies

1.a Print je naam

- Maak een klein programma dat je naam, adres en woonplaats afdrukt.
- Verander het programma van hierboven zodanig dat je niet drie `print`-aanroepen nodig hebt, maar slechts één.
  Let op: er zijn meer wegen die naar Rome leiden — kun je er twee vinden?

1.b Een pakezel

- Maak een programma dat onderstaande ASCII-art toont:

```
\n n O
 ====    pakezel
 /\/\
```

- Verander het programma zodanig dat je slechts één `print`-aanroep nodig hebt.

1.c Expressies

Beredeneer wat onderstaande Python expressies opleveren als je ze zou intypen bij de Python shell achter de >>> prompt:

```
a="abcdefghijklmnopqrstuvwxyz"

a[2:4]

a[2:10]

a[2:10:2]

a[10:2:-1]

a="hallo jij daar"

a[-1]

a[3:7]

a[::2]

a[::-1]
```

Opgave 2: Expressies

We zullen in deze opgave zien wat het verschil is tussen de soorten getallen en hoe je ermee kunt rekenen/werken.

2.a Grote getallen

- Maak een programma dat afdrukt wat 2.0 tot de macht 1000 is.
- Maak een programma dat afdrukt wat 2 tot de macht 1000 is.

2.b Een getal als string

Bepaal uit hoeveel cijfers het getal `2 ** 1000` bestaat.

■ Hint:
Je kunt een willekeurig ding in Python converteren naar een string met de functie `str(...)`. □

Opgave 3: Karaktertabel

De ASCII characterset begint bij 0 en loopt tot en met 127. Niet alle characters uit deze set zijn echter “leesbaar”; de leesbare set loopt van 32 (de spatie) tot en met 126 (de tilde).

3.a Simpele tabel

Maak een programma dat de hele ASCII set toont (leesbaar en onleesbaar). Toon iedere getalswaarde decimaal, hexadecimaal, octaal en als character, zoals hier aangegeven:

```
0   0x0   0o0
1   0x1   0o1
....
32   0x20   0o40
33   0x21   0o41   !
34   0x22   0o42   "
....
48   0x30   0o60   0
49   0x31   0o61   1
50   0x32   0o62   2
....
57   0x39   0o71   9
58   0x3a   0o72   :
59   0x3b   0o73   ;
60   0x3c   0o74   <
61   0x3d   0o75   =
62   0x3e   0o76   >
63   0x3f   0o77   ?
64   0x40   0o100   @
65   0x41   0o101   A
66   0x42   0o102   B
....
125   0x7d   0o175   }
126   0x7e   0o176   ~
127   0x7f   0o177
```

Je kunt een integer (stel: variabele *n*) naar een hexadecimale string converteren met de functie `hex(`*n*`)`, naar een octale string converteren met functie `oct(`*n*`)` en naar een leesbaar character converteren met de `chr(`*n*`)` functie.

3.b Aangepaste tabel

Hierboven werd al opgemerkt dat niet alle ASCII characters leesbare tekens zijn, hetgeen je kunt waarnemen in de geproduceerde uitvoer. Pas het programma zodanig aan dat van alle waarden de decimale, hexadecimale en octale representatie wordt getoond, maar dat alleen de leesbare tekens ook als “characters” worden afgedrukt.

Opgave 4: String manipulaties

4.a Frequentie van teken

Bepaal hoe vaak het cijfer 5 voorkomt in het getal 2 tot de macht 1000. Je kunt hiervoor het programma uitbreiden dat je bij opgave 2 hebt gemaakt.

Probeer deze opgave te maken met de method `.count(...)` en met de method `.replace(...)`.

4.b Mootjes hakken en formatteren

- Maak een nieuw programma en declareer daarin de volgende string:

  ```
  uurwaarden = "673,1499,82,119341,13,996308,53,7"
  ```

  Toon de getalswaarden uit deze string stuk-voor-stuk en toon een slotregel met het totaal. De uitvoer zou er zo uit moeten zien (maak nog *geen* gebruik van formattering):

  ```
  Uur  1: 673
  Uur  2: 1499
  Uur  3: 82
  Uur  4: 119341
  ....
  Totaal: 1117976
  ```

- Pas het programma nu zodanig aan, dat de getalswaarden rechts worden uitgelijnd met een kolombreedte van 7. Dus je uitvoer wordt dan:

  ```
  Uur  1:     673
  Uur  2:    1499
  Uur  3:      82
  Uur  4:  119341
  ....
  Totaal: 1117976
  ```

  Genereer deze uitvoer eenmaal met de klassieke formattering en eenmaal met de `.format` method.

- Pas het programma zodanig aan, dat het *dynamisch* bepaalt hoeveel tekens de totaal-teller bevat en dit gebruikt als kolombreedte om de lijst te genereren.

Opgave 5: Lijst uitmiddelen

In het bestaande programma `avglist.py` wordt een lijst aangemaakt.
Breid het programma zodanig uit dat het gemiddelde van de lijst wordt berekend en afgedrukt met decimalen.

```
#!/usr/bin/env python3

l=[ 1, 5, 2, 33, 5, 16, 7 ]

# Voeg hieronder de code toe die het gemiddelde uitrekent.
```

Opgave 6: Begincijfers van getallen

In deze opgave gaan we een curieuze wetmatigheid in getallen bekijken.

6.a Enkele machten

Maak een programma dat de machten van 1 t/m 1000 van het getal twee uitrekent en laat zien.

6.b Frequentie bepalen

Als je nu kijkt naar de uitkomsten, zal het je misschien opvallen dat van ieder getal het begincijfer 1 (één) het meest voor lijkt te komen.

Om deze waarneming wat meer houvast te geven gaan we het programma uitbreiden.
Breid het programma zo uit dat van de afgedrukte getallen bijgehouden wordt hoe vaak ieder begincijfer 1 tot 9 voorkomt.

6.c Andere getallen

Je kunt je nu afvragen of de frequentieverhoudingen veranderen als je met veel meer machten werkt (bijvoorbeeld alle tweemachten tot 10000), of als je niet met tweemachten werkt maar ook met driemachten.

Breid het programma zo uit dat ook voor machten van 3 tot en met 20 dezelfde frequentieverhoudingen worden afgedrukt.

De wet van Benford

De wetmatigheid die je ontdekt hebt heet officieel “de wet van Benford”. Het blijkt dat alle getallen die iets met elkaar te maken hebben (bijvoorbeeld alle file-groottes op een systeem, alle inwoneraantallen van steden, alle geldbedragen die op de bank staan, etc.) zich aan deze wetmatigheid houden.
Voor meer informatie, zie:

```
http://en.wikipedia.org/wiki/Benford
```

Opgave 7: Regels, woorden en tekens tellen

Het UNIX-programma `wc` (wordcount) telt het aantal tekens, woorden en regels in een file. In deze opgave gaan we dit programma nabouwen. Als invoer zullen we steeds de file `verhaal.txt` gebruiken.

- Lees de file `verhaal.txt` regel-voor-regel. Tel het aantal regels.
- Voeg code toe die het aantal woorden telt.
- Voeg code toe die het aantal tekens telt.

Toon aan het eind het aantal gevonden regels, woorden en tekens.

De getallen in de uitvoer van je programma moeten overeenkomen met de getallen die het commando "`wc verhaal.txt`" toont:

```
$ wc verhaal.txt
 16 166 943 verhaal.txt
```

Opgave 8: Bytes tellen

Maak de volgende veranderingen in het programma dat je voor opgave 7 hebt geschreven: wijzig in de aanroep van `open` de bestandsnaam `verhaal.txt` naar de bestandsnaam `limerick.txt` en voeg de parameter "`encoding='utf-8'`" toe[1].

- Hoeveel characters bevat dit bestand volgens jouw programma?
- Het UNIX-programma `wc` toont de volgende uitvoer:

```
$ wc limerick.txt
  5  22 156 limerick.txt
```

  Komt het aantal characters overeen met de uitkomst van jouw programma?

Het commando `wc` toont by default het aantal *bytes* als derde getal in de uitvoer. In dit UTF-8 bestand vind je echter één teken dat uit twee bytes bestaat: de é in "privé". Het aantal *tekens* kun je opvragen met de optie `-m` van het `wc` commando:

```
$ wc -m limerick.txt
155 limerick.txt
```

- Wijzig je programma zo, dat zowel het aantal bytes alsook het aantal tekens wordt geteld[2] (uiteraard lees je het bestand maar eenmaal).
  De uitvoer van je programma toont dan het aantal regels, woorden, tekens en bytes (vier waarden).

1. Als je jouw practicumopgaven maakt in Python 2, voeg de parameter `encoding=...` dan niet toe!
2. Als je jouw practicumopgaven maakt in Python 2, lees dan eerst het laatste deel van hoofdstuk 5 over "*Python 2 files en encoding*".

Opgave 9: Het weer de wereld rond

In de file met de naam `weer.txt` vind je een tabel met steden en bijbehorende temperaturen. We gaan met deze gegevens een beetje rekenen en sorteren. Om je een idee te geven van de layout van de regels in deze file:

```
# temperaturen in Europa op 6 mei 2005 08:00
Athene 19
Barcelona 15
Berlijn 7
Las-Palmas 17
... enzovoort ...
# temperaturen in wereldsteden buiten Europa op 6 mei 2005 08:00
Bangkok 37
Buenos-Aires 9
... enzovoort ...
```

Aan het einde van deze opgave heb je een programma geschreven dat van alle steden aangeeft of ze een onder- of bovengemiddelde temperatuur hadden.

9.a File inlezen en commentaar skippen

Open de file met de naam `weer.txt` en lees deze regel-voor-regel in. Druk alle regels die *niet* met een hekje beginnen op het scherm af.

9.b Gemiddelde uitrekenen

Reken het gemiddelde uit van de temperaturen van de steden.

9.c Opslaan in dictionary

Vul tijdens het inlezen van de regels een dictionary `temperatuur` waarin je voor iedere stad de gemeten temperatuur opslaat.

9.d Dictionary afdrukken

Druk nu twee lijsten af:

- Een lijst met namen van steden (en temperatuur) met een lagere dan gemiddelde temperatuur.
- Een lijst met namen van steden (en temperatuur) met minstens de gemiddelde temperatuur.

Opgave 10: Omkeringen

10.a Hele file op zijn kop

Maak een programma dat de file `verhaal.txt` inleest, en op zijn kop weer afdrukt.

■ *Hint:*
Met de method `readlines()` kan in één klap een file in een lijst worden ingelezen. □

10.b Rijmen

Woorden rijmen vaak als ze eindigen op dezelfde tekens. Wij gaan (gegeven een lijst van woorden) een rijmwoordenboek maken.

In de file `woorden.txt` tref je een lijst van Nederlandse woorden aan. Maak een programma dat — gegeven deze lijst — een rijmwoordenboek afdrukt.

■ *Hint*:
Als je de woorden omkeert en dan de lijst van omgekeerde woorden sorteert en daarna de woorden weer omkeert, zijn de woorden gesorteerd *van achter af*. □

Opgave 11: Woorden, dictionaries

11.a Woordfrequenties

- Maak een programma dat van de file `verhaal.txt` telt hoe vaak ieder woord voorkomt. Een stukje voorbeelduitvoer:

  ```
  dan: 1
  de: 6
  den: 4
  die: 8
  ```

  Dit betekent dat in de file `verhaal.txt` het woord “dan” eenmaal voorkwam, “de” zesmaal, “den” viermaal en “die” achtmaal.
- Worden de woorden gesorteerd afgedrukt? Zo niet, is dat lastig te maken?
- Breid het programma nu zo uit dat identieke woorden die eindigen (of beginnen) met een leesteken als “,” of “.” op één hoop geveegd worden.

11.b Cross-reference maken

Breid het programma van zojuist nu zodanig uit dat niet het aantal malen dat een woord voorkwam wordt getoond, maar op welke regelnummers het woord voorkwam. Ter verduidelijking een deel van de uitvoer:

```
tweedehandsch: [11]
uitleende: [13]
uitvreter: [2, 3, 4, 8, 16]
van: [1, 5, 6, 7, 8, 10, 10, 12, 12, 13]
```

Dit betekent dat het woord “uitleende” op regel 13 voorkwam, terwijl het woord “van” op de regels 1, 5, 6, ... voorkwam.

Opgave 12: Lijst uitmiddelen — opnieuw

In opgave 5 heb je code gemaakt die het gemiddelde van een lijst uitrekent. In de file `avglistfun.py` hebben wij de uitwerking van deze opgave alvast voor je klaargezet:

```
#!/usr/bin/env python3

l = [ 1, 5, 2, 33, 5, 16, 7 ]

som = sum(l)

print("het gemiddelde van de", len(l),
      "items in", l, "is:", float(som)/len(l))
```

12.a Uitmiddelen in functie

Pas de code zodanig aan dat het uitrekenen van het gemiddelde nu in een functie gedaan wordt. Deze functie krijgt de lijst met getallen mee als parameter en geeft het gemiddelde terug.

12.b Meerdere return-waarden

Pas de code nu zodanig aan dat de functie de lengte van de lijst en het gemiddelde teruggeeft.

Opgave 13: Functie-aanroepen

13.a Resultaat teruggeven

Maak een functie `dubbelen` die gegeven een lijst een nieuwe lijst maakt waarin de indexen staan van duplicaat-elementen in de invoerlijst. De functie `dubbelen()` mag ervan uit gaan dat de lijst gesorteerd is. Dus gegeven de invoerlijst

```
[ 'els', 'els', 'els', 'els', 'henk', 'henk',
  'jan', 'jan', 'john' 'piet', 'piet' ]
```

kan de resultaatlijst zijn:

```
[ 1, 2, 3, 5, 7, 10 ]
```

omdat de elementen met indexen 1, 2, 3, 5, 7 en 10 duplicaat-elementen zijn in de invoerlijst.

Geef invulling aan de functie `dubbelen()` in het raamwerk-programma `dubbelen.py`:

```
#!/usr/bin/env python3

def dubbelen(lijst):
    """
    Maakt een lijst aan met indexnummers van de
    dubbelen in lijst.  De lijst moet gesorteerd zijn.
    Parameter:
    lijst  - de te onderzoeken lijst
    """
    pass


namen = [ 'jan', 'piet', 'henk', 'els', 'piet',
          'els', 'john', 'els', 'jan', 'els', 'henk']

namen.sort()

print("De lijst was:", namen)

dub= dubbelen(namen)

print("De indexen van de dubbelen zijn:", dub)
```

13.b Call by reference

Maak met behulp van de hulpfunctie `dubbelen()` een nieuwe functie `listuniq()` die uit een lijst alle duplicaat-elementen verwijdert. Je mag er van uitgaan dat de

lijst gesorteerd is.

Let op: de list die `listuniq()` meekrijgt is mutable en wordt *call by reference* meegegeven; het is de bedoeling dat de functie `listuniq()` geen returnwaarde geeft: de lijst wordt *in place* gewijzigd. Dus de code:

```
namen = [ 'jan', 'piet', 'henk', 'els', 'piet',
          'els', 'john', 'els', 'jan', 'els', 'henk']

namen.sort()
print("De lijst was:", namen)

listuniq(namen)

print("De lijst is: ", namen)
```

heeft als uitvoer:

```
De lijst was: ['jan', 'jan', 'piet', 'piet', 'els', 'els', 'els',
         'els', 'henk', 'henk', 'john']
De lijst is:  ['els', 'henk', 'jan', 'john', 'piet']
```

13.c Zonder sorteren

Maak een nieuwe versie van functie `dubbelen()` die niet uitgaat van een gesorteerde lijst.

■ *Hint:* als je een set gebruikt in `dubbelen()` die bijhoudt of je een bepaalde waarde al gezien hebt, hoeft de lijst niet meer gesorteerd te zijn. □

Bij een invoerlijst van

```
[ 'jan', 'piet', 'henk', 'els', 'piet',
  'els', 'john', 'els', 'jan', 'els', 'henk']
```

moet de resultaatlijst dus zijn:

```
de lijst is: [4, 5, 7, 8, 9, 10]
```

Opgave 14: Functies als objecten

We gaan in deze opgave functies maken die een string als parameter meekrijgen en een string als returnwaarde teruggeven. De functie doet "iets" met behulp van de meegegeven string en geeft het resultaat terug. Door deze functies in een bepaalde volgorde aan te roepen, kun je flexibel data-manipulaties uitvoeren.

In het raamwerk-programma `gnirts.py` staan al enkele functies gedefinieerd, zoals de functie `omkeer()` die van een string de omgekeerde teruggeeft en de functie `makeupper()` die van een string de uppercase-variant teruggeeft:

```
#!/usr/bin/env python3

def omkeer(s):
    """Keert een string om
    parameter:
    s - de om te keren string (VALUE)
    """
    pass

def makeupper(s):
    """Maakt van een string een uppercase string
    parameter:
    s - de naar hoofdletters te converteren string (VALUE)
    """
    pass

def doalle(l, s):
    """Laat een lijst van functies los op een string
    parameters:
    l - de lijst van functies
    s - de string
    """
    pass


l=[ omkeer, makeupper ]

print( doalle(l, "lower case string") )
# uitvoer: GNIRTS ESAC REWOL
```

- Vul de functies `omkeer()` en `makeupper()` verder in.
- Maak de functie `doalle()` die alle functies aanroept die in een lijst gespecificeerd zijn, zodat je programma voor de string `lower case string` de uitvoer `GNIRTS ESAC REWOL` geeft.

Opgave 15: Fibonacci generator

De wiskundige reeks “Fibonacci sequence” bestaat uit een reeks getallen beginnend met `0` en `1`, waarbij ieder volgend getal de optelling is van de twee voorgaande getallen:

```
0  1  1  2  3  5  8  13  21  34  55  89  144 ...
```

Maak een generator-functie genaamd `fibo` die een Fibonacci-reeks genereert als-ie wordt aangeroepen in een `for`-iteratie. Deze functie moet worden aangeroepen met één argument dat de *bovengrens* aangeeft van de te generen reeks.

Voorbeeld:

```
def fibo(maxval):
    ....

for val in fibo(250):
    print(val)
```

geeft als uitvoer:

```
0
1
1
2
3
5
8
13
21
34
55
89
144
233
```

Hint:
Gebruik een `while`- in plaats van een `for`-loop in de generator-functie.

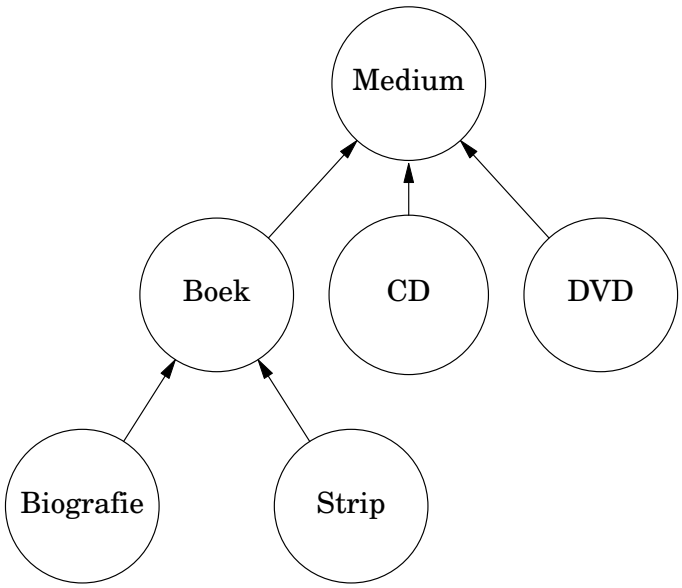
Opgave 16: Modules

- Maak van de uitwerking van opgave 14 een module.
- Maak een programma dat van de module gebruik maakt.
- Zorg ervoor dat als de module geïmporteerd wordt, alleen de functies gedefinieerd worden. Als de module als los programma wordt gestart, moet een test worden gedraaid (`GNIRTS ESAC REWOL`).

  Wat valt je op als je met het commando "`ls -l`" in de directory kijkt nadat je de module gebruikt hebt?

Opgave 17: Inheritance

In deze opgave is het de bedoeling een eenvoudige class hiërarchie te maken die media-dragers in hun verschillende vormen weergeeft. Zo'n hiërarchie zou bijvoorbeeld in een bibliotheek gebruikt kunnen worden.
Een mogelijke media-hiërarchie staat hieronder:



- Bedenk — voordat je elke mogelijke media-variant apart uitwerkt — dat de verschillende media-objecten een aantal zaken gemeenschappelijk hebben. Elk object uit de media-hiërarchie heeft bijvoorbeeld een titel. Een `Strip` echter heeft eigenschappen die uniek zijn voor een `Strip`: een tekenaar.
  Probeer te achterhalen welke elementen uniek zijn en welke niet. Bekijk hiervoor ook de parameters in de aanroepen van `Boek()`, `Strip()` en `Dvd()` in de onderstaande code.
- Maak een paar classes voor de media-hiërarchie, maar minstens `Medium` en `Boek` zodat de code werkt die in het raamwerk-programma `media.py` staat (als je de classes `Strip` en/of `Dvd` ook maakt, kun je de commentaartekens voor de betreffende aanroepen verwijderen):

```
#!/usr/bin/env python3

class Medium(object):
    """Een bibliotheek-medium klasse
    bedoeld als superclass van echte media
    """
    pass
```

```
class Boek(Medium):
    """Een bibliotheek-boek klasse
    """
    pass

#          titel             prijs   auteur                      paginas
lp = Boek("Learning Python", 35.50, "Mark Lutz & David Ascher", 595)

#            titel                  prijs  auteur    paginas tekenaar
#strx=Strip("Asterix en Cleopatra", 3.95, "Goscinny", 32, "Uderzo")

#          titel                   prijs  regisseur         min leeftijd
#film=Dvd("2001, a space odyssey", 17.50, "Stanley Kubrik", 12)

lp.pr()    # druk lp af

#strx.pr()
#film.pr()
```

Opgave 18: Gebruik van __str__ in media

In opgave 17 heb je de media-hiërarchie gemaakt. Eigenlijk willen we in die klassen geen `pr()` method hebben, maar zouden we een medium-object willen kunnen afdrukken met `print`.

Pas al je medium-klassen zo aan dat ze afgedrukt kunnen worden met `print`. Je moet daarvoor een `__str__()` method maken die een object representeert in een string. Als er zo'n `__str__()` is, dan zal `print` deze gebruiken om een medium naar een string te converteren en deze string af te drukken.

Opgave 19: Een Geld class

19.a Eerste poging

- Maak een eenvoudige "Geld" class.
  Een "Geld-object" is een koppel (euros, centen) om bedragen in op te slaan. De munteenheid is in Euro's.
- Denkvraag: Welke operatoren zijn zinvol voor `Geld`? Met andere woorden, welke bewerkingen op `Geld` hebben betekenis?
- Je mag er in het programma vanuit gaan dat negatieve bedragen niet voor kunnen komen. Je hoeft alleen het stukje voorbeeldcode hieronder aan de praat te krijgen.
  Maak de operatoren `+`, `*` en `/` zodat het raamwerk-programma `geld.py` werkt. Je kunt in de onderstaande code steeds meer commentaartekens weghalen, al naargelang je meer van de klasse `Geld` hebt geïmplementeerd.

> Je hoeft nog geen rekening te houden met "overflows" in het centen deel (`EUR` 2,59 + `EUR` 1,55 wordt 3 Euro en 114 centen), dat komt verderop in de opgave pas aan bod.

```
#!/usr/bin/env python3

class Geld(object):
    """een Geld klasse
    """
    pass

# g1=Geld(2,17)
# print(g1)
#
# g2=Geld(2, 88)
# print(g2)
#
# g3=g1+g2
# print(g3)
#
# g3=g1*7
# print(g3)
#
# g3=7*g1
# print(g3)
#
# g4=g3/3
# print(g4)
```

19.b En nu correct

Bij het maken van `Geld`-berekeningen moet je natuurlijk zorgen dat het resultaat een geldig bedrag blijft. Als je bijvoorbeeld twee `Geld`-bedragen bij elkaar optelt, dan worden de euros en de centen bij elkaar opgeteld. Maar als het aantal centen groter wordt dan 99, dan moet je de euros met 1 ophogen en van de centen 100 aftrekken. Als je een algemene method maakt die een `Geld`-object “normaliseert”, kun je in alle methods die de waarde zouden kunnen veranderen gebruik maken van deze normalisatie-functie.

Opgave 20: Een gesorteerde dictionary

Het komt nogal eens voor dat we de keys van een dictionary door willen lopen, maar dan in gesorteerde volgorde. Wat dan vaak geschreven wordt, is:

```
l = mydict.keys()
l.sort()
for key in l:
    .... l .... mydict[l] .....
```

Het zou erg praktisch zijn als er een type `SortedDict` bestond waarbij de `keys` method meteen een gesorteerde lijst oplevert.

- Maak in het raamwerk-programma `dictsort.py` een class `SortedDict` die zich precies zo gedraagt als een `dict`, maar waarvan de `keys` method de sleutels in gesorteerde volgorde oplevert.

  ■ *Hint:* je hoeft maar heel weinig zelf te maken als je erft van `dict`. □

- Zorg er nu ook voor dat bij het afdrukken van een `SortedDict` een gesorteerde volgorde gebruikt wordt.

  ■ *Hint:* hiervoor maak je een eigen `__str__` method. □

Opgave 21: Een exceptionele Geld class

We gaan de `Geld` class van opgave 19 aanpassen. Als je de uitwerking niet hebt, kun je gebruik maken van de uitwerking die klaarstaat, `geldexcept.py` genaamd.

- Pas de `Geld` class zodanig aan dat ook een muntsoort wordt onthouden. Bij het afdrukken van een `Geld` object, moet de muntsoort ook afgedrukt worden. De default muntsoort is `"EUR"`.
- Voeg een controle toe aan de implementatie van de optelling zodat geen bedragen van verschillende muntsoorten kunnen worden opgeteld. Genereer in dat geval een exception van het type `ValutaError` (dit type kun je zelf maken).

Het onderstaande stukje source-code moet werken en bij het laatste statement een exception veroorzaken:

```
class Geld(object):
    """Een Geld klasse
    """
    pass

g1=Geld(2, 17)
print(g1)

g2=Geld(2, 88)
print(g2)

g3=g1+g2
print(g3)

g3=g1*7
print(g3)

g3=7*g1
print(g3)

g3=g1*7
print(g3/3)

g4=Geld(1, 32, "DKR")
g5=Geld(7, 99, "DKR")

print(g4+g5)
print(g4+g3)      # dit moet een exception geven...
```

- Vang de exception af bij de laatste regel van dit programma, zodat het programma niet ongecontroleerd afbreekt.

Opgave 22: Benford en file-groottes

Om te kijken of de wet van Benford (zie opgave 6) ook opgaat op andere plaatsen, gaan we nu een cijferanalyse doen op file-groottes.

22.a File-groottes bepalen

Maak een programma dat van een directory bepaalt hoe groot iedere file is die in én onder deze directory staat. De te "tellen" directory moet op de commandoregel meegegeven kunnen worden.
Houd er rekening mee dat de functie `getsize()` een exception kan geven als de betreffende file niet benaderd kan worden.

22.b Benford-wetmatigheid op files

Pas het programma uit het vorige onderdeel zo aan dat — op dezelfde manier als in opgave 6 — geteld wordt hoe vaak alle begincijfers voorkomen in de in onderdeel 22.a bepaalde file-groottes.

Druk deze aantallen af (eventueel als percentage). We zijn niet geïnteresseerd in files met grootte 0.

Opgave 23: URLs en reguliere expressies

23.a IMG tags tellen

Maak een programma dat telt naar hoeveel plaatjes verwezen wordt in een bepaalde web-pagina. Een verwijzing naar een plaatje kun je vinden door te zoeken naar een `<IMG ...` tag in de HTML code.

Zorg dat het programma voldoet aan de volgende eisen:

1. De URL voor de webpagina geef je op als commandoregel-argument.
   Let op dat `urllib.request` eist dat web-URL's beginnen met `http://`, dus als dit niet volledig op de commandoregel is opgegeven, moet je programma dat voor de URL plaatsen.
2. Open de URL en vang de exception af die kan ontstaan als je bijvoorbeeld een niet-bestaande URL meegeeft.
3. Lees de volledige web-pagina (merk op dat dit een `bytes` object wordt en geen string) en decodeer die naar een string. Ga er voorlopig vanuit dat de web-pagina `UTF-8` geëncodeerd is.
4. Bepaal het aantal keren dat de tag "`<IMG `" voorkomt in de web-pagina. Houd daarbij rekening dat deze tag *niet* case-sensitive is (mag — deels — in kleine letters).

De website `http://python.zaal.atcomputing.nl` bevat 11 plaatjes.

23.b Correcte encodering

Elke web-pagina kan volgens een andere standaard geëncodeerd zijn. De werkelijke encodering zou in de HTTP header `Content-Type` kunnen staan, bijvoorbeeld:

```
Content-Type: text/html; charset=utf-8
```

Dit zou je kunnen onderzoeken met de volgende aanroep (`web` is hier de returnwaarde van `urlopen`):

```
encoding = web.info().get_content_charset()
```

Deze `charset=` definitie is echter niet verplicht in de HTTP header; als deze ontbreekt, geeft bovenstaande aanroep de waarde `None` terug.
In dat geval zou je nog in de webpagina-inhoud op zoek kunnen gaan naar een meta-tag waarin de `Content-Type` wordt gedefinieerd, bijvoorbeeld:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Om het voor deze opgave niet te moeilijk te maken, zou je (voorlopig) alleen kunnen zoeken op "`charset=`" in de pagina-inhoud.