# OEFENINGEN PYTHON – REEKS 2

## Vraag 1: Functies

In deze oefening zullen we een reeks functies maken waarmee je strings kan maken die een raster voorstellen:

```
#----#----#
|    |    |
|    |    |
|    |    |
#----#----#
|    |    |
|    |    |
|    |    |
#----#----#
```

Als je het raster lijn per lijn bekijkt kan je twee soorten lijnen onderscheiden: de randen van het raster en de middenstukken.

a) Maak een functie *maak_rand* die de string

”#----#----#\n”

als resultaat geeft, en maak een functie *maak_midden* die de string

”|    |    |n”

als resultaat geeft. Probeer hierbij zoveel mogelijk gebruik te maken van de operatoren ‘samenvoegen’ (+) en ‘herhalen’ (*) op strings!

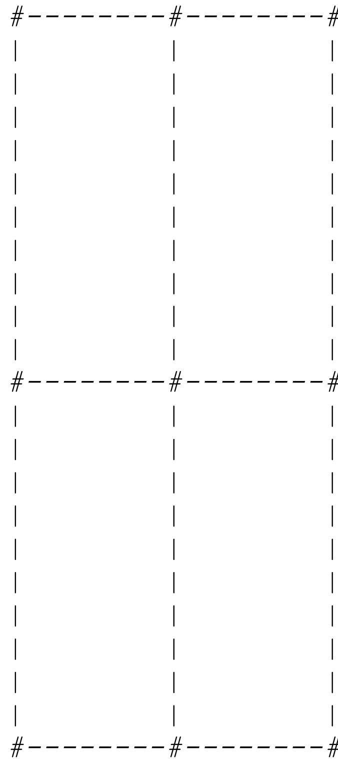
b) Maak nu een functie *maak_raster* die de twee vorige functies gebruikt om het voorbeeldraster te construeren. Maak opnieuw gebruik van de operatoren + en * op strings om zoveel mogelijk typewerk te besparen. Je kan je functie testen door gebruik te maken van de print- functie: *print maak_raster()*

c) Pas de functies *maak_rand* en *maak_midden* aan zodat ze een argument verwachten dat de breedte aangeeft. Deze parameter bepaalt hoe breed elk vierkantje van het raster is en bepaalt dus hoeveel ”-”-tekens en ” ”-tekens je zal moeten genereren. Bijvoorbeeld: *maak_rand(10)* moet volgende string teruggeven:

“#----------#----------#”

d) Zorg vervolgens dat je aan de functie *maak_raster* ook een argument kan meegeven dat de breedte van de vierkantjes uit het raster bepaalt.

e) Voeg een tweede argument toe aan de functie *maak_raster* dat de hoogte van elk vierkantje bepaalt. Dit komt neer op het aanpassen hoeveel “middens” je moet genereren. Uiteindelijk moet bv. *maak_raster(8,10)* het volgende resultaat opleveren:



f) ***Extra Oefening:*** Breid je oefening uit zodat je ook het aantal vakjes kan variëren. Zorg dat code enkel herhaald wordt indien uiterst noodzakelijk!

## Vraag 2: Keuzes maken

Door middel van een ***IF … ELIF … ELSE …*** structuur kan je de uitvoering van je programma laten afhangen van bepaalde voorwaarden die je nagaat. De structuur waarbij *elif* en *else* optioneel zijn is als volgt:

```
if conditie:
        <block>
elif andere_conditie:
        <block>
else:
        <block>
```

a) Maak een functie die een ingegeven jaartal inspecteert en teruggeeft welke type jaartal het is, aan de hand van volgende tabel:

| Type 1 | Geen schrikkeljaar: niet deelbaar door 4 |
|---|---|
| Type 2 | Wel schrikkeljaar: deelbaar door 4 maar niet door 100 |
| Type 3 | Geen schrikkeljaar: deelbaar door 100 maar niet door 400 |
| Type 4 | Wel schrikkeljaar: deelbaar door 400 |

Is het interessanter om *print* of *return* te gebruiken? Welke datatypes zijn mogelijk om terug te geven en welke is de beste keuze om verder te gebruiken in andere code?

Wat is de relatie tussen de *IF*-structuur, en de booleaanse operatoren *OR* en *AND*?

## Vraag 3: Iteraties

Een ***FOR*** lus (counted loop) laat je toe een bepaald stuk code in je programma een op voorhand bepaald aantal keren uit te voeren. De FOR lus gaat stap voor stap elk element in een sequentie af (lijst, tupel, string), doet een assignment van dat element in de iteratie-variabele, en voert de geïtereerde code uit. De structuur is:

```
for iteratie-variabele in sequentie:
        <block geïtereerd>
```

De sequentie kan een reeds bestaande lijst, tupel, string zijn. Er kan gebruik gemaakt worden van de **range()** functie om snel een lijst aan te maken. Zo is er *range(eind)*, *range(begin, eind)* en *range(begin,eind,stap)*. Let hierbij op dat het laatste element niet meer tot de lijst behoort.

a) Maak een functie waaraan je kan meegeven hoeveel keer het woordje 'bla' moet worden geprint. Werk deze keer NIET met de operatoren * of + bewerkingen op strings, maar gebruik een *FOR* lus in combinatie met de *range* functie.

b) Maak een functie die de volgende pijl op het scherm print:

```
*
**
***
****
***
**
*
```

Zorg ervoor dat je de grootte van de pijl (in het voorbeeld is dit 4) aan de functie kunt meegeven.

Soms weet je niet op voorhand hoeveel iteraties je juist nodig hebt. In dat geval kan je gebruik maken van een ***WHILE*** lus (conditional loop). De lus loopt zolang er aan een bepaalde voorwaarde is voldaan. Merk op dat het hierdoor is het mogelijk om oneindige lussen te maken. De structuur van de while lus is:

```
while conditie:
        <block geïtereerd>
```

c) Maak oefening a en b opnieuw maar deze keer met een WHILE lus in plaats van een FOR lus.

d) Maak een functie *mijn_deling* waaraan je een deeltal en een deler kunt meegeven. De functie zal dan het quotiënt en de rest van de gehele deling weergeven. Gebruik enkel de operatoren *+* en - voor je berekeningen!

## Vraag 4: Iteraties over lijsten

Indien je met lijsten werkt zal je zeer vaak een bewerking moeten doen waarbij je alle (of een deel van de) elementen uit de lijst moet afgaan. Dit kan je in Python gemakkelijk uitdrukken door een *FOR* lus te schrijven die itereert over die bepaalde lijst.

a) Wiskundige bewerkingen met lijsten
    1. Maak een functie die een lijst aanmaakt waarbij de elementen de opeenvolgende machten van -2 uitdrukken. Gebruik hiervoor NIET de '**' operator. Aan de functie kan je meegeven hoeveel elementen je wenst te krijgen.
    2. Maak een functie die het gemiddelde van een lijst berekent en teruggeeft. Gebruik deze functies om het gemiddelde van de eerste 15 machten van -2 te berekenen.

b) Maak een functie die een gegeven lijst omkeert (het laatste element wordt het eerste etc.). Implementeer deze functie door de waarden van de inputlijst te kopiëren in een nieuwe lijst. Maak GEEN gebruik van de *list.reverse()* methode!

c) Maak een functie waaraan je een string meegeeft. De functie zal dan teruggeven of deze string al dan niet een palindroom is (een woord dat je kunt omkeren en toch hetzelfde blijft, zoals 'lepel' of 'koortsmeetsysteemstrook').

## Vraag 5: Fouten en debuggen

a) In het volgende stukje code staat een veel voorkomende fout:

```
temp = 0
while temp<100:
  temp = temp/2-1
```

Schrijf het stukje code in je bestand, en evalueer het. Wat gebeurt er en waarom? Nu gaan we debuggen…

Plaats een breakpoint door in de marge te klikken bij een lijn code. Start de debugger door te klikken op de insect (bug) knop (F5). We zien nu de huidige waarden van alle reeds geassigneerde variabelen en extra additionele variabelen in de *stack data* tab. De code kan stap per stap uitgevoerd worden door de knoppen met pijltjes (F6 – F8) waarbij *step into* (F7) meestal de interessantste optie is.