Opdrachten week 1 les 2 – Functies in Python

In de vorige les heb je kennis gemaakt met de programmeertaal Python en de IDE Thonny. Je kunt eenvoudige Python programma's schrijven met behulp van variabelen, expressies, statements en ingebouwde functies. Je kunt deze programma's debuggen met behulp van Thonny. In deze les ga je leren hoe je zelf functies kunt schrijven. Je leert hoe je:

- een functie zonder parameters kan definiëren en hoe je zo'n functie kan aanroepen;
- een functie met parameters kan definiëren en hoe je bij de aanroep van deze functie argumenten kan meegeven;
- fouten in functies kunt opsporen met behulp van Thonny.

In de volgende les leer je tekenen met een schildpad (ja, er staat echt "tekenen met een schildpad").

Lees nu paragraaf 3.4 van het boek[1].

1.2.1 Schrijf een Python programma waarin een functie genaamd `line` wordt gedefinieerd die 40 mintekens achter elkaar afdrukt. Sla dit programma op onder de naam `week1_les2.py`. Voer nu dit programma uit door op de afspeelknop in Thonny te klikken, of door op F5 te drukken. Je kunt nu de functie uitvoeren (aanroepen) door in de Python Shell `line()` in te typen. Als het goed is zie je dezelfde uitvoer als in figuur 1.

Figuur 1: De uitvoer van de functie `line()`.

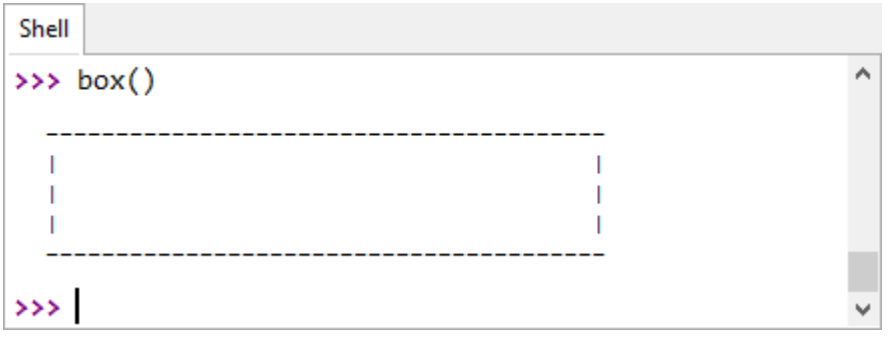
1.2.2 Als je in les 1 van deze week goed hebt opgelet, dan heb je de functie `line` gedefinieerd zonder *zelf* tot 40 te tellen.

[1] Allen B. Downey. *Think Python: How to Think Like a Computer Scientist*. 2de ed. Green Tea Press, 2016. ISBN: 978-1-4919-3936-9. URL: http://greenteapress.com/wp/think-python-2e/.

Kijk nog eens terug naar opdracht 1.1.27 als je wel zelf 40 mintekens hebt ingetypt en verbeter de definitie van de functie `line`[2].

1.2.3 Definieer in het programma `week1_les2.py` ook een functie genaamd `sides` die het teken `|` afdrukt gevolgd door 38 spaties en daarna nogmaals het teken `|`. Als je het slim aanpakt hoef je maar één `print`-statement te gebruiken en niet *zelf* tot 38 te tellen. Je kunt nu de functie uitvoeren (aanroepen) door in de Python Shell `sides()` in te typen. Vergeet niet om eerst het programma uit te voeren door op de afspeelknop in Thonny te klikken, of door op F5 te drukken.

1.2.4 Definieer in het programma `week1_les2.py` ook een functie genaamd `box` die de uitvoer produceert die weergegeven is in figuur 2. Maak daarbij gebruik van de al eerder gedefinieerde functies `line` en `sides`.



Figuur 2: De uitvoer van de functie `box()`.

Lees nu paragraaf 3.5 van het boek.

1.2.5 Zorg er nu voor dat functie `box` vanuit het programma `week1_les2.py` aangeroepen wordt. Als je het programma uitvoert door op de afspeelknop in Thonny te klikken, of door op F5 te drukken, dan moet meteen een rechthoek worden getekend. Tip: je hoeft maar één regel toe te voegen in het programma.

Lees nu paragraaf 3.6 van het boek.

[2] Als je het goed doet, hoef je maar één minteken zelf in te typen.

Je kunt de volgorde van uitvoering (het boek noemt dit 'the flow of execution') goed volgen met behulp van de in Thonny ingebouwde debugger.

1.2.6 Vergelijk het programma `week1_les2_box.py`[3] met het programma `week1_les2.py` dat je bij opdracht 1.2.5 hebt ontwikkeld. Als het goed is zijn ze gelijk.

1.2.7 Voer het programma `week1_les2_box.py` stap voor stap uit door eerst op het de debug-knop te klikken of door op Ctrl + F5 te drukken en daarna steeds op de step-into-knop te klikken of op F7 te drukken. Elke keer als er een functie aangeroepen wordt, opent Thonny een nieuw window waarin je de uitvoering van die functie kunt volgen. Je kunt zo'n window verplaatsen onafhankelijk van de overige Thonny windows.

1.2.8 Herhaal opdracht 1.2.7 maar gebruik nu de step-over-knop of F6 om door het programma te stappen. Begrijp je wat er gebeurt?

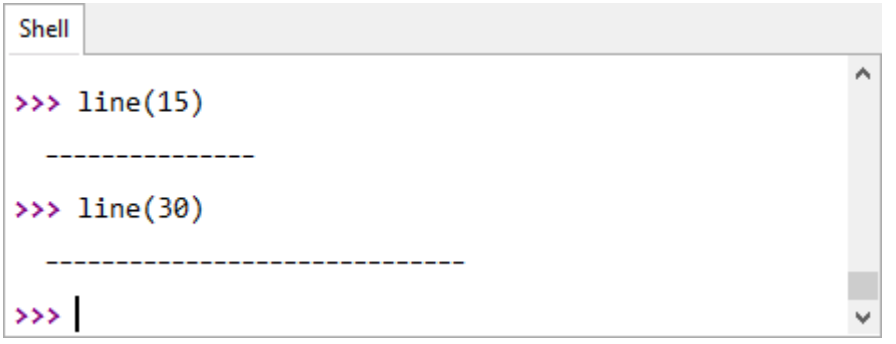
1.2.9 Herhaal opdracht 1.2.7 maar gebruik nu eerst drie maal de step-over-knop of F6, dan zes maal de step-into-knop of F7 en daarna weer steeds de step-over-knop of F6, om door het programma te stappen. Beschrijf het verschil tussen step-into en step-over in je eigen woorden.

1.2.10 Herhaal opdracht 1.2.7 maar gebruik nu eerst negen maal de step-into-knop of F7 en daarna steeds de step-out-knop of F8, om door het programma te stappen. Beschrijf de werking van de step-out-knop in je eigen woorden.

Lees nu paragraaf 3.7 van het boek.

1.2.11 Schrijf een nieuw Python programma `week1_les2_par.py` waarin een functie genaamd `line` wordt gedefinieerd die een parameter genaamd `width` heeft. Deze functie moet `width` mintekens achter elkaar afdrukken. Voer het programma uit en roep de functie `line` aan vanuit de Python Shell met verschillende argumenten, bijvoorbeeld 15 en 30, zie figuur 3.

[3] Zie: https://bitbucket.org/HR_ELEKTRO/ems10/raw/master/Opdrachten/progs/week1Les2Box.py.



Figuur 3: De functie `line` moet aangeroepen kunnen worden met verschillende argumenten.

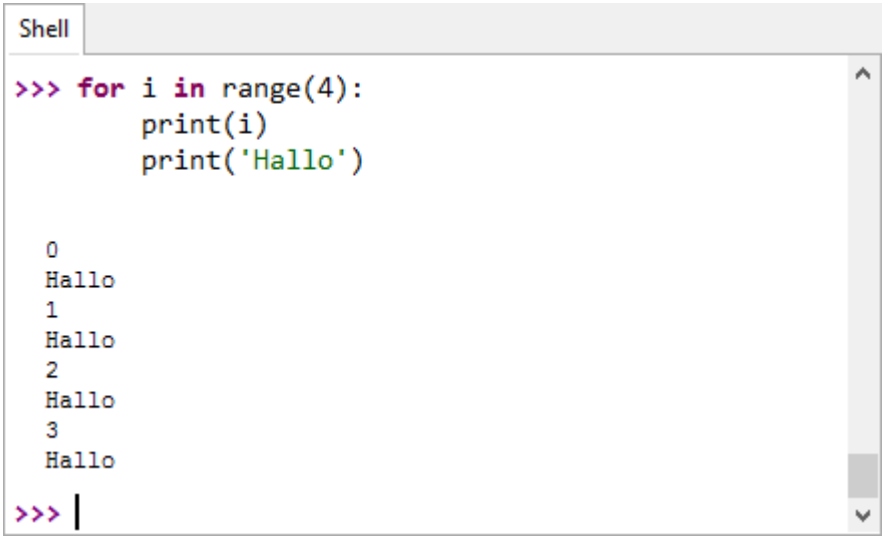
1.2.12 Definieer in het programma `week1_les2_par.py` ook een functie genaamd `sides` die een parameter genaamd `width` heeft. Deze functie moet het teken `|` afdrukken gevolgd door `width - 2` spaties en daarna nogmaals het teken `|`. Voer het programma uit en roep de functie `sides` aan vanuit de Python Shell met verschillende argumenten, bijvoorbeeld 15 en 30, zie figuur 4.

Figuur 4: De functie `sides` moet aangeroepen kunnen worden met verschillende argumenten.

1.2.13 Wat gebeurt er als je de functie `sides` aanroept met `2` als argument? Wat gebeurt er als je `0` als argument gebruikt? Wat gebeurt er als je een negatief getal als argument gebruikt? Wat gebeurt er als je een string, bijvoorbeeld `'Hallo'`, als argument gebruikt? Begrijp je wat er allemaal gebeurt?

Als je bepaalde Python statements meerdere malen uit wilt laten voeren dan kan dat door gebruik te maken van een zogenoemd **`for`**-statement. In figuur 5 zie je hoe je dit in de Python Shell kunt gebruiken. Let er daarbij op dat de statements die herhaald zullen worden

automatisch ingesprongen worden. In de Shell moet het `for`-statement afgesloten worden met een lege regel. Als je een `for`-statement in een programma gebruikt, dan is het niet nodig om het `for`-statement af te sluiten met een lege regel.

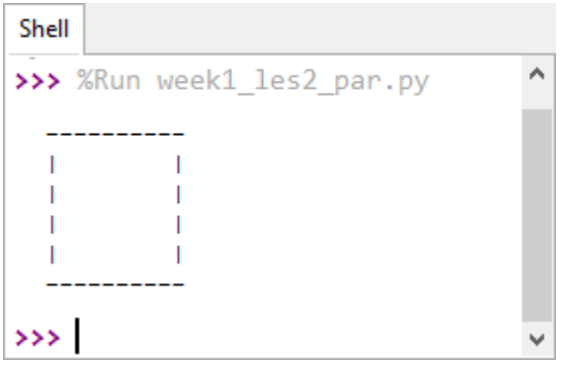


Figuur 5: Met een `for`-statement kan een stukje Python code meerdere keren uitgevoerd worden.

1.2.14 Zet de code uit figuur 5 in een programma genaamd `for.py` zodat de uitvoer die gegeven is in figuur 5 verschijnt als je op de afspeelknop klikt, of op F5 drukt. Wat gebeurt er als je het laatste `print`-statement niet inspringt? Begrijp je waarom dit gebeurt?

1.2.15 Definieer in het programma `week1_les2_par.py` ook een functie genaamd `box` die twee parameters heeft genaamd `width` en `height`. Deze functie moet een rechthoek afdrukken met een breedte van `width` karakters en een hoogte van `height` regels. Maak daarbij gebruik van de al eerder gedefinieerde functies `line` en `sides`. Roep de functie als volgt aan in het programma: `box(10, 6)`. Als het goed is, komt de uitvoer overeen met figuur 6.

Lees nu paragraaf 3.8 en 3.9 van het boek.



Figuur 6: De gewenste uitvoer van het programma `week1_les2_par.py`.

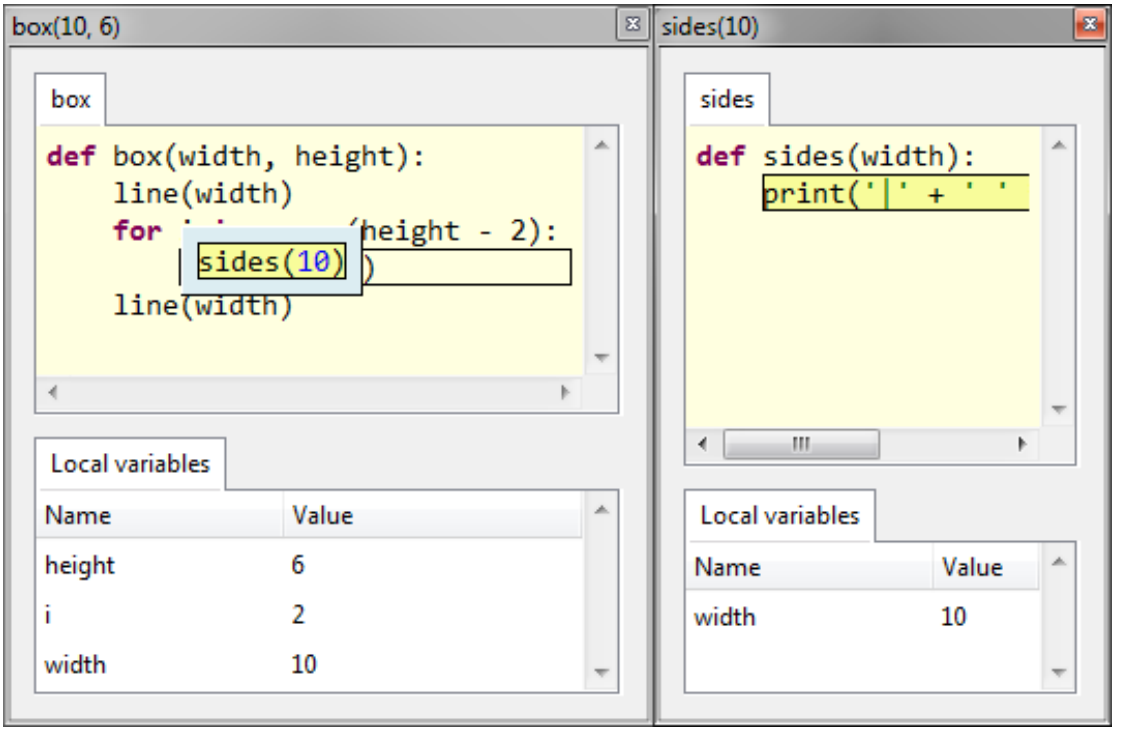
1.2.16 Voer het programma `week1_les2_par.py` stap voor stap uit door eerst op het de debug-knop te klikken of door op Ctrl + F5 te drukken en daarna steeds op de step-into-knop te klikken of op F7 te drukken. Elke keer als er een functie aangeroepen wordt, opent Thonny een nieuw window waarin je de uitvoering van die functie kunt volgen. Ook worden de waarden van de parameters en variabelen van de betreffende functie getoond. Je kunt zo'n window verplaatsen onafhankelijk van de overige Thonny windows, zie figuur 7.

Lees nu paragraaf 3.10 van het boek.

In Python kun je functies maken *met* en *zonder* return-waarde. Het boek noemt dit respectievelijk 'fruitful function' en 'void functions'. In de lessen van volgende week gaan we hier dieper op in.

1.2.17 Noem 2 voorbeelden van functies met een return-waarde en twee voorbeelden van functies zonder return-waarde anders dan de al in paragraaf 3.10 van het boek genoemde functies.

Lees nu paragraaf 3.11 van het boek.



Figuur 7: Bij het stap voor stap uitvoeren van functies worden ook de waarden van de parameters en variabelen getoond.

1.2.18 Er worden in het boek vier redenen genoemd om functies te gebruiken. Vertel in je eigen woorden welke redenen dit zijn. Wat is volgens jou de *belangrijkste* reden om functies te gebruiken. Verklaar je antwoord[4].

Lees nu paragraaf 3.12 van het boek.

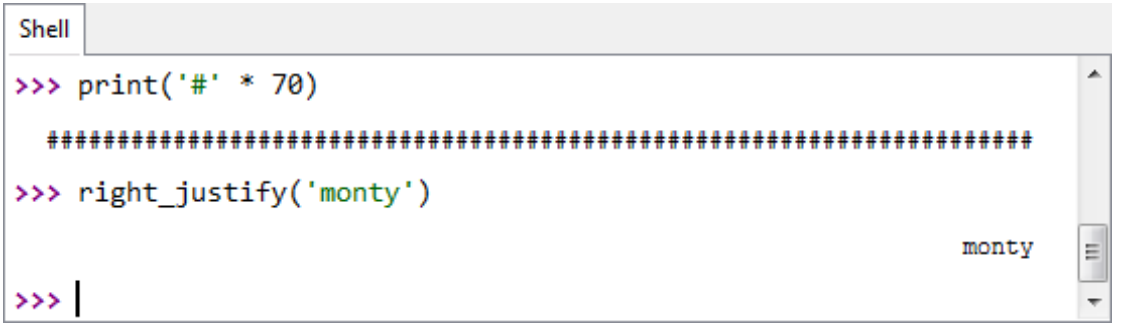
Bij het debuggen kun je, zoals je waarschijnlijk al hebt ervaren, handig gebruik maken van de debug-knop en verschillende step-knoppen, en van Thonny.

Paragraaf 3.13 van het boek definieert de verschillende (vak)termen die in hoofdstuk 3 gebruikt worden. Het kan handig zijn om deze paragraaf te raadplegen als je niet meer weet wat met een bepaalde (vak)term bedoeld wordt.

[4] Het gaat bij deze vraag om de logische onderbouwing van je antwoord. Er zijn vele correcte antwoorden mogelijk.

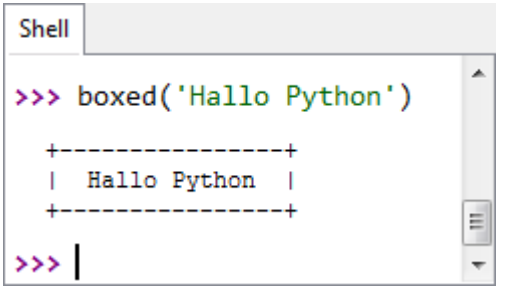
Paragraaf 2.10 van het boek bevat enkele oefeningen. De oefeningen die de moeite waard zijn, zijn hieronder in het Nederlands vertaald.

1.2.19 Schrijf een functie genaamd `right_justify` die een parameter heeft genaamd `s`. Deze functie moet de als argument meegegeven string afdrukken voorafgaand door genoeg spaties om er voor te zorgen dat de laatste letter van de string in kolom 70 van de uitvoer staat. In figuur 8 is de uitvoer van een correct werkende functie `right_justify` te zien. Tip: Maak gebruik van de string-operatoren '+' (concatenation) en '*' (repetition). Maak ook gebruik van de ingebouwde Python-functie `len` die de lengte van (het aantal karakters in) een string teruggeeft. Bijvoorbeeld: de waarde van `len('monty')` is 5.



Figuur 8: De uitvoer van een correct werkende functie `right_justify`.

1.2.20 Schrijf een functie genaamd boxed met een parameter genaamd `s`. Als deze functie wordt aangeroepen met een string als argument dan moet deze string met een rechthoek eromheen worden afgedrukt, zie figuur 9.



Figuur 9: De uitvoer van de functie-aanroep `boxed('Hallo Python')`.

1.2.21 Er zijn bij een programmeeropdracht altijd meerdere oplossingen mogelijk. Bestudeer de oplossingen die gegeven zijn op https://bitbucket.org/HR_ELEKTRO/ems10/wiki/boxed.md en vergelijk die met je eigen oplossing.

1.2.22 Over het ontstaan van het schaakspel doen vele legendes de ronde. Een van die legendes[5] vertelt over een wijze uit India die ongeveer 1500 jaar geleden het schaakspel uitvond. Hij leerde het spel aan zijn koning en die was hem zo dankbaar dat hij de wijze zelf zijn beloning liet uitkiezen. De wijze vroeg de koning om 1 graankorrel op het eerste veld van het schaakbord, 2 korrels op het tweede veld, 4 korrels op het derde, 8 korrels op het vierde enzovoorts. Dus totdat alle velden gevuld waren, op ieder veld steeds het dubbele aantal graankorrels van het vorige veld. De koning vond dit in eerste instantie maar een bescheiden wens, maar al snel bleek dat er op de hele wereld niet genoeg graankorrels beschikbaar waren om de wijze te belonen.

Schrijf een programma in Python dat voor elk veld van het schaakbord (een schaakbord heeft 64 velden) het nummer van het veld en het aantal graankorrels op dat veld afdrukt. Druk aan het eind van het programma ook het totaal aantal graankorrels af. Gebruik in het programma alleen de rekenkundige operaties optellen en vermenigvuldigen.

In de vorige opgave heb je (als het goed is[6]) een variabele gebruikt om het totaal aantal graankorrels bij te houden. Als je bijvoorbeeld specifieke rijen met getallen wilt printen, dan is het ook vaak handig om een of meer variabelen te gebruiken.

We bespreken eerst een geval waarbij dat nog niet per se nodig is. Stel dat je gevraagd wordt om een Python-functie te schrijven die de eerste n getallen uit de volgende rij afdrukt: $1, 4, 7, 10, 13, 16, \ldots$[7]. Het valt je vast op dat elk getal gelijk is aan zijn voorganger plus drie. Je kunt dus eenvoudig het volgende getal berekenen als je het huidige getal in een variabele bijhoudt. De gevraagde functie ziet er dan als volgt uit:

```
def druk_rij_af(n):
    huidige = 1
    for i in range(n):
        print(huidige)
```

[5] H.J. Raverty. “History of Chess and Backgammon”. In: *Journal of the Royal Asiatic Society of Bengal* 71.1 (1902), p. 47.

[6] Vergelijk je oplossing met de oplossingen die gegeven zijn op https://bitbucket.org/HR_ELEKTRO/ems10/wiki/schaken_met_graan.md.

[7] Dit is een rekenkundige rij met een beginterm van 1 en een verschil van 3.

```
        huidige = huidige + 3
```

In dit geval kan het ook zonder een variabele te gebruiken. De waarde van term i is namelijk gelijk aan $1 + i \times 3$. Dus je kunt de functie ook als volgt programmeren:

```
def druk_rij_af(n):
    for i in range(n):
        print(1 + i * 3)
```

Stel nu dat je gevraagd wordt om een Python-functie te schrijven die de eerste n getallen uit de volgende rij afdrukt: $1, 2, 4, 7, 11, 16, 22, 29, \ldots$. Het valt je vast op dat elk getal gelijk is aan zijn voorganger plus een verschil en dat dit verschil bij elke stap met één toeneemt. Je kunt dus eenvoudig het volgende getal berekenen als je het huidige getal en het verschil in een variabele bijhoudt. De gevraagde functie ziet er dan als volgt uit:

```
def druk_rij_af(n):
    huidige = 1
    verschil = 1
    for i in range(n):
        print(huidige)
        huidige = huidige + verschil
        verschil = verschil + 1
```

Merk op dat de volgorde van de statements in de `for`-loop heel belangrijk is voor de juiste werking van de functie. Als de statements in een andere volgorde worden geplaatst, werkt het programma niet meer correct.

Misschien heb je al gezien dat de waarde van de variabele `verschil` steeds gelijk is aan de waarde van de variabele `i` plus één. Dus je kunt de functie ook als volgt programmeren:

```
def druk_rij_af(n):
    huidige = 1
    for i in range(n):
        print(huidige)
        huidige = huidige + i + 1
```

Als laatste voorbeeld bekijken we een ingewikkelde opdracht: Schrijf een functie die de eerste n getallen uit de volgende rij afdrukt: $1, 2, 4, 8, 15, 26, 42, 64, \ldots$. Omdat dit geen intelligentietest is zullen we de systematiek uitleggen. Als je de verschillen tussen de getallen op een rijtje zet, dan krijg je: $1, 2, 4, 7, 11, 16, 22, \ldots$. Komt dat je al bekend voor? Als je de verschillen tussen de verschillen op een rijtje zet, dan krijg je: $1, 2, 3, 4, 5, 6, \ldots$. Nu je de systematiek (het patroon) van de rij begrijpt kun je de functie proberen te schrijven.

Probeer het eerst zelf!

Een mogelijk oplossing is gegeven in listing 1.

```
def druk_rij_af(n):
    huidige = 1
    verschil = 1
    for i in range(n):
        print(huidige)
        huidige = huidige + verschil
        verschil = verschil + i + 1
```

Listing 1: De functie druk_rij_af. Zie druk_rij_af.py.

Bij de volgende opdracht heb je ook meerdere variabelen nodig om dingen te onthouden.

1.2.23 De rij van Fibonacci[8] is een bekende rij getallen waarbij elk getal uit de rij gelijk is aan de som van zijn twee voorgangers. De eerste tien getallen uit de rij van Fibonacci zijn: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34. Schrijf een functie genaamd `print_rij_van_Fibonacci` met een parameter genaamd `n` die de eerste `n` getallen uit de rij van Fibonacci afdrukt. Je mag er daarbij van uitgaan dat $n \geq 2$ is.

Schrijf ook een testprogramma waarin de functie aangeroepen wordt met verschillende waarden als argument (bijvoorbeeld 2, 10 en 100).[9]

Hier eindigen de verplichte opdrachten van week 1 les 2. Er volgen nu nog twee gedeelten:

- oefening, in dit gedeelte vind je extra oefeningen die je helpen om de leerstof van deze les beter te onthouden;
- verdieping, in dit gedeelte vind je een uitdagende, verdiepende opdracht.

[8] Zie: https://nl.wikipedia.org/wiki/Rij_van_Fibonacci.

[9] De meeste beginnende programmeurs vinden dit een lastige opdracht. Als je er niet uitkomt, dan kun je een oplossing vinden op https://bitbucket.org/HR_ELEKTRO/ems10/wiki/print_rij_van_Fibonacci.md. Maar probeer het eerst zelf!

Oefening

Alle onderstaande oefeningen (met de bijbehorende *uitwerking*) zijn ook beschikbaar op Anki flashcards zodat je ze regelmatig kunt herhalen. Deze stok kun je hier downloaden: EMS10 Week 1 Les 2.apkg.

1.2.24 Geef de Pythoncode voor een functie genaamd `print_hallo` die de volgende tekst afdrukt:

```
Hallo!
```

1.2.25 Gegeven de onderstaande functiedefinitie in Python:

```
def print_hallo():
    print('Hallo!')
```

Met welke Pythoncode kun je deze functie aanroepen?

1.2.26 Gegeven de onderstaande functiedefinitie in Python:

```
def print_hallo():
    print('Hallo!')
```

Geef de Pythoncode voor een functie genaamd `print_3x_hallo` die het volgende afdrukt:

```
Hallo!
Hallo!
Hallo!
```

Maak daarbij gebruik van de functie `print_hallo`.

1.2.27 Geef de Pythoncode voor een **`for`**-loop die de volgende tekst afdrukt:

```
Hallo 0
Hallo 1
Hallo 2
Hallo 3
```

1.2.28 Geef de Pythoncode voor een **for**-loop die de volgende tekst afdrukt:

```
0 x 0 = 0
1 x 1 = 1
2 x 2 = 4
3 x 3 = 9
4 x 4 = 16
5 x 5 = 25
6 x 6 = 36
7 x 7 = 49
8 x 8 = 64
9 x 9 = 81
```

1.2.29 Geef de Pythoncode voor een **for**-loop die de volgende tekst afdrukt:

```
0 x 7 = 0
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
```

1.2.30 Geef de Pythoncode voor een `for`-loop die de volgende tekst afdrukt:

```
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
```

1.2.31 Geef de Pythoncode voor een `for`-loop die de volgende tekst afdrukt:

```
1 x 1 = 1
2 x 2 = 4
3 x 3 = 9
4 x 4 = 16
5 x 5 = 25
6 x 6 = 36
7 x 7 = 49
8 x 8 = 64
9 x 9 = 81
10 x 10 = 100
```

1.2.32 Geef de Pythoncode voor een `for`-loop die de volgende tekst afdrukt:

```
Hallo 0
Hallo 2
Hallo 4
Hallo 6
Hallo 8
```

1.2.33 Geef de Pythoncode voor een functie genaamd `druk_even_getallen_af` die de eerste n even getallen afdrukt. Bijvoorbeeld de aanroep `druk_even_getallen_af(5)` moet de volgende uitvoer opleveren:

```
0
2
4
6
8
```

1.2.34 Geef de Pythoncode voor een functie genaamd `druk_machten_van_2_af` die de eerste n machten van 2 afdrukt. Bijvoorbeeld de aanroep `druk_machten_van_2_af(6)` moet de volgende uitvoer opleveren:

```
2 tot de macht 0 is 1
2 tot de macht 1 is 2
2 tot de macht 2 is 4
2 tot de macht 3 is 8
2 tot de macht 4 is 16
2 tot de macht 5 is 32
```

1.2.35 Geef de Pythoncode voor een functie genaamd `druk_omtrek_cirkel_af` die de omtrek van een cirkel met een straal s afdrukt. Bijvoorbeeld de aanroep `druk_omtrek_cirkel_af(1.5)` moet de volgende uitvoer opleveren:

```
9.42477796076938
```

1.2.36 Geef de Pythoncode voor een functie genaamd `druk_inhoud_bol_af` die de inhoud van een bol met een straal s afdrukt. De inhoud van een bol met straal s is gelijk aan:

$$\frac{3}{4} \cdot \pi \cdot s^3$$

Bijvoorbeeld de aanroep `druk_inhoud_bol_af(1.5)` moet de volgende uitvoer opleveren:

```
7.9521564043991635
```

1.2.37 Gegeven de volgende Pythonfunctie:

```
def druk_even_getallen_af(n):
    for i in range(n):
        print(i * 2)
```

Geef de Pythoncode waarmee deze functie aangeroepen kan worden om alle even getallen tot en met 10 af te drukken:

```
0
2
4
6
8
10
```

1.2.38 Geef de Pythoncode voor een functie genaamd `druk_omtrek_rechthoek_af` die de omtrek van een rechthoek met een lengte len en een breedte br afdrukt. Bijvoorbeeld de aanroep `druk_omtrek_rechthoek_af(1.5, 2.5)` moet de volgende uitvoer opleveren:

```
8.0
```

1.2.39 Gegeven de volgende Pythonfunctie:

```
def druk_omtrek_rechthoek_af(lengte, breedte):
    print(2 * (lengte + breedte))
```

Geef de Pythoncode waarmee deze functie aangeroepen kan worden om de omtrek van een rechthoek met een lengte van 1,6 en een breedte van 2,7 af te drukken.

1.2.40 Geef de Pythoncode voor een functie genaamd `druk_machten_van_getal_af` die de eerste n machten van een geheel getal afdrukt. Bijvoorbeeld de aanroep `druk_machten_van_getal_af(3, 5)` moet de volgende uitvoer opleveren:

```
3 tot de macht 0 is 1
3 tot de macht 1 is 3
3 tot de macht 2 is 9
3 tot de macht 3 is 27
3 tot de macht 4 is 81
```

1.2.41 Geef de Pythoncode voor een functie genaamd `print_rij` die het volgende afdrukt (merk op dat het verschil steeds met 1 toeneemt):

```
0
1
3
6
10
15
```

Verdieping

Een verdiepende oefening uit paragraaf 2.10 van het boek is hieronder in het Nederlands vertaald.

1.2.42 Een functie-object is een waarde die je kunt toekennen aan een variabele of door kunt geven als argument. Bijvoorbeeld de in listing 2 gegeven functie `do_twice` heeft een parameter `f`. Als je aan deze functie een functie-object als argument meegegeeft dan roept `do_twice` deze functie twee maal aan. In listing 2 wordt een voorbeeld gegeven waarbij `do_twice` gebruikt wordt om een functie genaamd `print_spam` twee keer aan te roepen. Let goed op hoe je het functie-object `print_spam` als argument meegeeft aan de functie `do-twice`.

A Type het programma gegeven in listing 2 over en test het.

B Wat gebeurt er als de laatste regel van dit programma gewijzigd wordt in: `do_twice(print_spam())`? Begrijp je waarom dit niet werkt?

C Pas de functie `do_twice` aan zodat de functie aangeroepen kan worden met twee argumenten, een functie-object `f` en een waarde `v`. De functie `do_twice` moet nu twee maal het functie-object `f` aanroepen met de waarde `v` als argument.

D Kopieer de functie `line` die je in opdracht 1.2.11 hebt gemaakt in dit programma.

E Gebruik de aangepaste versie van `do_twice` om de functie `line` twee keer aan te roepen met de waarde 40 als argument.

F Definieer een nieuwe functie genaamd `do_four` waaraan je een functie-object `f` en een waarde `v` als parameters kunt doorgeven. De functie `do_four` moet nu viermaal het functie-object `f` aanroepen. Maak bij de implementatie van `do_four` gebruik van de al eerder geschreven functie `do_twice`.

G Gebruik de functie `do_four` om de functie `line` vier keer aan te roepen met de waarde 10 als argument.

```
def do_twice(f):
    f()
    f()

def print_spam():
    print('spam')

do_twice(print_spam)
```

Listing 2: De functie do_twice. Zie do_twice.py.