Python GGH Workshop 3 (23 April 2021)

Opgave 1: Priemgetallen

Tijdens opgave 1 van de vorige workshop, heb je een code geschreven die controleert of een getal priem is. Dit gaan wij gebruiken om een programma te schrijven dat zoveel priemgetallen vindt als je wil, en in een lijst stopt.

(a) Maak een functie `priem(n)`, die controleert of een getal `n` priem is. Hij geeft `True` of `False` terug als `return` waarde. Gebruik hiervoor je code van Opgave 1, Workshop 2.

(b) Maak een nieuwe functie `priemgetallen_tot(n)`, die een lijst teruggeeft van alle priemgetallen tussen de `1` en `n`. Kun je hiervoor het beste een `for` – loop, of `while` – loop gebruiken?

(c) Maak een nieuwe functie `priemgetallen_maak(n)`, die een lijst teruggeeft van de eerste `n` priemgetallen. Kun je hiervoor het beste een `for` – loop, of `while` – loop gebruiken?

Voorbeelden:

```
priem(7)
True
priem(6)
False
priemgetallen_tot(10)
[2, 3, 5, 7]
priemgetallen_maak(10)
[2, 3, 5, 7, 9, 11, 13, 17, 19, 23]
```

Opgave 2: Recursieve Fibonacci Functie

Mag een functie zichzelf aanroepen? Tuurlijk mag dat! Dit heet een zogenaamde "recursieve functie". Dat is dus een functie, die zichzelf aanroept. Dat kan ontzettend handig zijn, maar het kan ook goed misgaan:

```
def print_lager_dan(n):
     print(n-1)
     print_lager_dan(n-1)
```

De bovenstaande functie print alle getallen kleiner dan `n`, en dat doet hij door zichzelf aan te roepen. Zonder loop dus! Er gaat echter iets mis: de functie zal nooit ophouden, en je programma zal vastlopen.

Daarom moet een recursieve functie altijd een zogeheten "stop-conditie" hebben. Als de functie een bepaalde waarde binnenkrijgt, dan geeft hij een standaard antwoord, en roept hij zichzelf niet weer aan. Zo voorkom je dat je in een eeuwige loop komt. Meestal ziet dat eruit als een `if` statement aan het begin van de functie om de voorwaarde te controleren, gevolgd door een `return` om de functie te eindigen. Nu is de functie aangepast met een stop-conditie, en print hij alle getallen tot en met 1:

```
def print_lager_dan(n):
```

```
    if(n<=1):
        return
    print(n-1)
    print_lager_dan(n-1)
```

De fibonacci-reeks (https://nl.wikipedia.org/wiki/Rij_van_Fibonacci) ziet er als volgt uit:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, etcetera.
```

De eerste twee getallen van de reeks zijn vastgezet als 0 en 1. Elke volgende getal in de reeks, is gelijk aan de som van de twee ervoor.

(a) Maak een recursieve functie fibonacci(n), die het nde fibonacci getal geeft.

HINT 1: Gebruik de formule fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
HINT 2: Welke stop conditie heb je nodig? We weten sowieso al dat `fibonacci(1) == 0` en `fibonacci(2) == 1`.

Voorbeeld:
```
fibonacci(1)
0
fibonacci(3)
1
fibonacci(9)
21
```

(b) Kun je een functie fibonacci_while(n) maken, die in plaats van zichzelf aan te roepen, gewoon een while loop gebruikt?

Maak een flowchart voordat je begint, zo ga je effectief te werk.

Deze opgave is uitdagend, dus vraag Ethan zeker als je vastkomt.

Huiswerk Opgave 1: Sorteren

Schrijf een programma dat eerst een boel (integer) getallen inleest, totdat er geen invoer wordt gegeven. Dan output hij de ingevoerde lijst. Vervolgens sorteert hij deze lijst, en output de gesorteerde lijst.

Voorbeeld:
```
Geef input:
7
6
24
3
8
3
-5

Invoer:
[7, 6, 24, 3, 8, 3, -5]
```

```
Gesorteerde Lijst:
[-5, 3, 3, 6, 7, 8, 24]
```

Maak een flowchart voordat je begint, zo ga je effectief te werk.

HINT: Je kunt hiervoor het eenvoudige "bubblesort" algoritme gebruiken. Deze staat beschreven op: https://nl.wikipedia.org/wiki/Bubblesort#Werking. Pas op, gebruik niet de Python voorbeeld code, dat is valsspelen!

Uitdaging: Het bubblesort algoritme is niet zo efficiënt: om N elementen te sorteren, kan het tot circa N^2 "sorteeracties" gebruiken. Dat schrijft men als O(N^2). Een veel efficiëntere sorteer algoritme is Quicksort. Deze gebruikt slechts O(N log N) sorteeracties. Kun je deze ook implementeren? Hij staat hier beschreven: https://nl.wikipedia.org/wiki/Quicksort . Probeer ook de snelheid van je sorteeralgoritmes te vergelijken op grote testverzamelingen.

Huiswerk Opgave 2: Roulette

Ontwerp een roulettespel, waarin de speler begint met een vast aantal fiches, bijvoorbeeld 10, en vervolgens net zo lang door kan gaan, todat hij besluit terug te trekken.

Gebruik voor het gemak de volgende regels: een roulettewiel heeft 1+36 cijfers: de getallen 0 t/m 36. De even cijfers zijn rood, de oneven cijfers zijn zwart. De speler mag inzetten op een getal (G), tussen 1-36, of op een kleur (K) (R/Z). Als hij inzet op een getal en hij wint, dan krijgt hij 36 keer zijn inzet terug. Als hij inzet op een kleur en wint, dan krijgt hij 2 keer zijn inzet terug.

Voorbeeld ter inspiratie: (Je mag je Roulette zelf maken zoals je wil):

```
Welkom bij Roulette.

Je hebt 10 fiches.
Geef je actie: (G(ETAL), K(LEUR), T(ERUGTREKKEN)): G
Geef het getal: 10
Geef je inzet: 5

Het roulettewiel draait…
De bal kwam op 10, ROOD.
Gefeliciteerd! Je hebt nu 185 fiches.

Je hebt 185 fiches.
Geef je actie: (G(ETAL), K(LEUR), T(ERUGTREKKEN)): K
Geef de kleur: R
Geef je inzet: 184.

Het roulettewiel draait…
De bal kwam op 0, ZWART.
Helaas! Je hebt nu 1 fiches.

Je hebt 1 fiches.
Geef je actie: (G(ETAL), K(LEUR), T(ERUGTREKKEN)): T
```

```
Tot de volgende keer!
```

Probeer het programma netjes en overzichtelijk te schrijven, met zoveel mogelijk functies en comments om de code leesbaar te houden.

HINT: Om een willekeurig getal te nemen, gebruik de random module:
https://docs.python.org/3/library/random.html

```
import random # Noodzakelijk
randgetal = random.randint(1, 10) # Geeft willekeurig getal 1-10.
```

HINT: Om het programma te pauzeren terwijl het roulettewiel draait, gebruik de time module:

```
import time # Noodzakelijk
time.sleep(1) # Programme slaapt voor 1 seconde.
```

Links

Als je meer wil oefenen en leren over deze les, doe dan de lessen 10, 12, 13 op:
https://cscircles.cemc.uwaterloo.ca/nl/