Python les-materialen

# Functie opdrachten

Problemen zijn gerangschikt in oplopende moeilijkheidsgraad: \* Opwarmen - deze kunnen worden opgelost met behulp van basisvergelijkingen en methoden \* Niveau 1 - dit kan betrekking hebben op als/dan voorwaardelijke uitspraken en eenvoudige methoden \* Niveau 2 - hiervoor kan het nodig zijn om reeksen te herhalen, meestal met lus \* Uitdagend - deze zullen wat creativiteit vergen om op te lossen

## OPWARMINGSSECTIE

#### LESS OF TWO EVENS: Schrijf een functie die het kleinste van twee gegeven getallen retourneert *als* beide getallen even zijn, maar de grotere retourneert als een of beide getallen oneven zijn

lesser\_of\_two\_evens(2,4) --> 2  
lesser\_of\_two\_evens(2,5) --> 5

def lesser\_of\_two\_evens(a,b):  
 pass

# Controleren  
lesser\_of\_two\_evens(2,4)

# Controleren  
lesser\_of\_two\_evens(2,5)

#### ANIMAL CRACKERS: Schrijf een functie neemt een string van twee woorden en retourneert True als beide woorden met dezelfde letter beginnen

animal\_crackers('Levelheaded Llama') --> True  
animal\_crackers('Crazy Kangaroo') --> False

def animal\_crackers(text):  
 pass

# Controleren  
animal\_crackers('Levelheaded Llama')

# Controleren  
animal\_crackers('Crazy Kangaroo')

#### MAKES TWENTY: Gegeven twee gehele getallen, retourneer True als de som van de gehele getallen 20 is *of* als een van de gehele getallen 20 is. Zo niet, retourneer False

makes\_twenty(20,10) --> True  
makes\_twenty(12,8) --> True  
makes\_twenty(2,3) --> False

def makes\_twenty(n1,n2):  
 pass

# Controleren  
makes\_twenty(20,10)

# Controleren  
makes\_twenty(2,3)

# NIVEAU 1 PROBLEMEN

#### OLD MACDONALD: Schrijf een functie die de eerste en vierde letter van een naam als hoofdletter gebruikt

old\_macdonald('macdonald') --> MacDonald

Note: 'macdonald'.capitalize() retourneert 'Macdonald'

def old\_macdonald(name):  
 pass

# Controleren  
old\_macdonald('macdonald')

#### MASTER YODA: Geef een zin terug met de woorden omgekeerd (reversed)

master\_yoda('I am home') --> 'home am I'  
master\_yoda('We are ready') --> 'ready are We'

Opmerking: de methode .join() kan hier handig zijn. Met de methode .join() kun je strings samenvoegen in een lijst met een connectorstring. Sommige toepassingen van de methode .join() zijn bijvoorbeeld: >>> “–”.join([‘a’,‘b’,‘c’]) >>> ‘a–b–c’

Dit betekent dat als je een lijst met woorden had die je weer in een zin wilde veranderen, je ze gewoon kon samenvoegen met een enkele spatie:

>>> " ".join(['Hello','world'])  
>>> "Hello world"

def master\_yoda(text):  
 pass

# Controleren  
master\_yoda('I am home')

# Controleren  
master\_yoda('We are ready')

#### BIJNA ER: Gegeven een geheel getal n, retourneer True als n binnen 10 van 100 of 200 ligt

almost\_there(90) --> True  
almost\_there(104) --> True  
almost\_there(150) --> False  
almost\_there(209) --> True

OPMERKING: abs(num) geeft de absolute waarde van een getal

def almost\_there(n):  
 pass

# Controleren  
almost\_there(104)

# Controleren  
almost\_there(150)

# Controleren  
almost\_there(209)

# NIVEAU 2 PROBLEMEN

#### VIND 33:

Gegeven een lijst met gehele getallen, retourneer True als de array ergens een 3 naast een 3 bevat.

has\_33([1, 3, 3]) → True  
has\_33([1, 3, 1, 3]) → False  
has\_33([3, 1, 3]) → False

def has\_33(nums):  
 pass

# Controleren  
has\_33([1, 3, 3])

# Controleren  
has\_33([1, 3, 1, 3])

# Controleren  
has\_33([3, 1, 3])

#### PAPER DOLL: Geef een tekenreeks, retourneer een tekenreeks waarbij voor elk teken in het origineel drie tekens zijn

paper\_doll('Hello') --> 'HHHeeellllllooo'  
paper\_doll('Mississippi') --> 'MMMiiissssssiiippppppiii'

def paper\_doll(text):  
 pass

# Controleren  
paper\_doll('Hello')

# Controleren  
paper\_doll('Mississippi')

#### BLACKJACK: Gegeven drie gehele getallen tussen 1 en 11, als hun som kleiner is dan of gelijk is aan 21, retourneer dan hun som. Als hun som hoger is dan 21 *en* er een elf is, verlaag dan de totale som met 10. Ten slotte, als de som (zelfs na aanpassing) groter is dan 21, retourneert u ‘BUST’

blackjack(5,6,7) --> 18  
blackjack(9,9,9) --> 'BUST'  
blackjack(9,9,11) --> 19

def blackjack(a,b,c):  
 pass

# Controleren  
blackjack(5,6,7)

# Controleren  
blackjack(9,9,9)

# Controleren  
blackjack(9,9,11)

#### SUMMER OF ’69: Retourneer de som van de getallen in de array, behalve negeer secties van getallen die beginnen met een 6 en doorlopen tot de volgende 9 (elke 6 wordt gevolgd door ten minste één 9). Retourneer 0 voor geen getallen.

summer\_69([1, 3, 5]) --> 9  
summer\_69([4, 5, 6, 7, 8, 9]) --> 9  
summer\_69([2, 1, 6, 9, 11]) --> 14

def summer\_69(arr):  
 pass

# Check  
summer\_69([1, 3, 5])

# Check  
summer\_69([4, 5, 6, 7, 8, 9])

# Check  
summer\_69([2, 1, 6, 9, 11])

# UITDAGENDE PROBLEMEN

#### SPY GAME: Schrijf een functie die een lijst met gehele getallen inneemt en True retourneert als deze 007 in de juiste volgorde bevat.

spy\_game([1,2,4,0,0,7,5]) --> True  
 spy\_game([1,0,2,4,0,5,7]) --> True  
 spy\_game([1,7,2,0,4,5,0]) --> False

def spy\_game(nums):  
 pass

# Controleren  
spy\_game([1,2,4,0,0,7,5])

# Controleren  
spy\_game([1,0,2,4,0,5,7])

# Controleren  
spy\_game([1,7,2,0,4,5,0])

#### COUNT PRIMES: Schrijf een functie die het *aantal* teruggeeft van priemgetallen die bestaan tot en met een bepaald getal.

count\_primes(100) --> 25

Volgens conventies zijn 0 en 1 geen priemgetallen.

def count\_primes(num):  
 pass

# Controleren  
count\_primes(100)

### Voor de lol:

#### PRINT BIG: Schrijf een functie die een enkele letter nodig heeft, en retourneert een 5x5 representatie van die letter

print\_big('a')  
  
out: \*   
 \* \*  
 \*\*\*\*\*  
 \* \*  
 \* \*  
 \* \*

HINT: Overweeg om een dictionary te maken van mogelijke patronen en het alfabet toe te wijzen aan specifieke 5-regelige combinaties van patronen. Voor deze oefening is het oké als je woordenboek stopt bij “E”.

def print\_big(letter):  
 pass

print\_big('a')

## Veel succes!