

Voorbeeld : Yahtzee

In deze notebook gaan we een aantal concepten van het dobbelspel Yahtzee implementeren. Voor een kort overzicht van dit spel, zie [Wikipedia \(https://nl.wikipedia.org/wiki/Yahtzee\)](https://nl.wikipedia.org/wiki/Yahtzee). Om een worp te simuleren maken we gebruik van de functie `randint(1,6)` die een pseudo-random getal genereert van 1 tot 6. Met pseudo-random bedoelen we dat er achterliggend een algoritme gebruikt wordt dat de getallen genereert, maar dat dit algoritme voldoende onvoorspelbaar is voor de meeste toepassingen waarbij toeval een belangrijke rol speelt, zoals spelletjes. Deze functie biedt echter onvoldoende garanties voor toepassingen in domeinen zoals cryptografie.

```
In [1]: from random import randint  
  
        print(randint(1,6))
```

5

Een willekeurige worp met 5 dobbelstenen kan nu voorgesteld worden door een lijst met 5 random elementen van 1 tot en met 6.

```
In [2]: def worp():  
        D=[]  
        for i in range(5):  
            D.append(randint(1,6))  
        return D  
  
        print(worp())
```

[3, 3, 6, 6, 1]

Yahtzee bepalen

Het Walhala van de Yahtzee speler is een worp met 5 maal dezelfde cijferwaarde. Hoe groot is nu de kans op zo'n worp? Hoewel het perfect mogelijk is om door middel van combinatieleer deze kans te bepalen, zullen we hier een simulatie gebruiken omdat het nu eenmaal de bedoeling van deze cursus is om te leren programmeren.

Volgende twee functies bepalen beiden of een worp Yahtzee is. Bekijk ze grondig en probeer te begrijpen hoe en waarom ze werken. Eventueel kan je Python Tutor gebruiken om beide voorbeelden te doorlopen: [Voorbeeld1 \(http://www.pythontutor.com/visualize.html#code=def%20yahtzee%28worp%29%3A%0A%20%20%20%20for%20i%20in%20range%285%29%3A%0A%20%20%20%20D.append\(randint\(1,6\)\)%0A%20%20%20%20return%20D%0A%0A%20%20print\(worp\(\)\)%0A%0A\)](http://www.pythontutor.com/visualize.html#code=def%20yahtzee%28worp%29%3A%0A%20%20%20%20for%20i%20in%20range%285%29%3A%0A%20%20%20%20D.append(randint(1,6))%0A%20%20%20%20return%20D%0A%0A%20%20print(worp())%0A%0A) en [Voorbeeld2 \(http://www.pythontutor.com/visualize.html#code=def%20yahtzee%28worp%29%3A%0A%20%20%20%20teller=0%0A%20%20for%20i%20in%20range%285%29%3A%0A%20%20%20%20teller+=1%20if%20worp\[i\]==worp\[0\]%20%20%20%20return%20teller%0A%0A%20%20print%28yahtzee\(worp\)%0A%0A\)](http://www.pythontutor.com/visualize.html#code=def%20yahtzee%28worp%29%3A%0A%20%20%20%20teller=0%0A%20%20for%20i%20in%20range%285%29%3A%0A%20%20%20%20teller+=1%20if%20worp[i]==worp[0]%20%20%20%20return%20teller%0A%0A%20%20print%28yahtzee(worp)%0A%0A)



```
In [3]: def yahtzee1(worp):
        for i in range(1,5):
            if worp[i]!=worp[0]:
                return False
        return True

def yahtzee2(worp):
    teller=[0,0,0,0,0,0]
    for i in range(5):
        teller[worp[i]-1]+=1
    return 5 in teller
```

Kans op Yahtzee via simulatie bepalen

Om de kans te bepalen op Yahtzee in 1 worp voeren we een groot aantal simulaties uit en tellen de kans op succes.

```
In [4]: nSimulaties=100000
        succes=0
        for i in range(nSimulaties):
            if yahtzee1(worp()):
                succes+=1
        print(succes/nSimulaties)
```

0.00079

Omdat dit slechts een simulatie is, zal de uitkomst niet exact zijn (de exacte uitkomst is $1/6$ tot de vierde macht, of op z'n Pythoniaans geschreven $1/6^{**4}$; dit is 0.0007716049...), maar waarschijnlijk komt het wel aardig in de buurt.

Bij een spelletje Yahtzee hebben we echter meer dan 1 beurt om Yahtzee te gooien; we kunnen bijvoorbeeld volgende strategie volgen:

- kijk welke dobbelsteen waarde het meeste voorkomt
- leg de dobbelstenen met die waarde opzij
- gooi met de resterende dobbelstenen opnieuw
- herhaal dit totdat de worpen op zijn
- Yahtzee? Champagne!

Opnieuw is het cruciaal te beslissen hoe we exact te werk zullen gaan en welke data structuren we gebruiken. Om te bepalen welke dobbelsteenwaarde het vaakst voorkomt, lijkt het interessant om te tellen hoe vaak elke waarde gegooid werd. Ook om te bepalen of de worp Yahtzee is, is dat handig (zie de functie `yahtzee2(worp)` hierboven). We gaan daarom deze functionaliteit, dat is: tellen hoe vaak elke dobbelwaarde in een worp voorkomt, in een aparte functie plaatsen:

```
In [5]: def tel(worp):
        teller=[0,0,0,0,0,0]
        for i in range(len(worp)):
            teller[worp[i]-1]+=1
        return teller
```

```
# Een paar testen:
print(tel([1,2,3,4,5]))
print(tel([2,3,4,5,6]))
print(tel([2,3,3,1,1]))
```

```
[1, 1, 1, 1, 1, 0]
[0, 1, 1, 1, 1, 1]
[2, 1, 2, 0, 0, 0]
```

We kunnen nu bepalen welke cijferwaarde het meeste aantal keren werd gegooit en hoe vaak dat dan was, met behulp van de functie `tel(worp)` .

```
In [6]: def grootste(worp):
        """
        worp: lijst van dobbelwaarden

        output: (d,n) waarbij d de dobbelsteenwaarde is die het meest geworpen werd
        en n het aantal keer is dat die geworpen werd
        """
        C=tel(worp)
        maxAantal=0
        maxCijfer=0
        for i in range(6):
            if C[i]>maxAantal:
                maxAantal=C[i]
                maxCijfer=i+1
        return (maxCijfer,maxAantal)

print(grootste([1,2,3,4,5]))
print(grootste([2,3,4,5,6]))
print(grootste([2,3,3,1,1]))
print(grootste([6, 6, 1, 2, 4]))
```

```
(1, 1)
(2, 1)
(1, 2)
(6, 2)
```

We gaan dus 3 maal werpen, maar niet met alle dobbelstenen; we passen de functie `werp` dus aan:

```
In [7]: def werp(n): # werp n dobbelstenen
        D=[]
        for i in range(n):
            D.append(randint(1,6))
        return D

        def yahtzee(worp):
            return 5 in tel(worp)
```

We gaan 3 maal volgende procedure volgen:

- gooi alle nog te werpen dobbelstenen
- voeg die samen met de dobbelstenen die we opzij legden
- kijk welke cijferwaarde maximaal voorkomt
- leg deze dobbelstenen opzij

In code ziet deze procedure er als volgt uit:

```
In [8]: teWerpen=5 # aantal dobbelstenen die niet opzij gelegd zijn
        opzij=[] # opzij gelegde cijferwaarden
        for i in range(3): # we hebben 3 worpen
            W=werp(teWerpen)
            print("Worp",i+1,':',W)
            (d,n)=grootste(opzij+W)
            opzij=[d]*n
            print("We leggen opzij :",opzij)
            teWerpen=5-n

        if yahtzee(opzij):
            print("Champagne!")
```

```
Worp 1 : [1, 5, 5, 1, 3]
We leggen opzij : [1, 1]
Worp 2 : [4, 5, 2]
We leggen opzij : [1, 1]
Worp 3 : [1, 5, 5]
We leggen opzij : [1, 1, 1]
```

Deze stappen moeten we nu een aantal keer uitvoeren om te simuleren:

```

In [9]: def simulatieStap():
        teWerpen=5 # aantal dobbelstenen die niet opzij gelegd zijn
        opzij=[] # opzij gelegde cijferwaarden
        for i in range(3): # we hebben 3 worpen
            W=werp(teWerpen)
            (d,n)=grootste(opzij+W)
            opzij=[d]*n
            teWerpen=5-n

        if len(opzij)==5:
            return True
        else:
            return False

nSimulaties=10000
succes=0
for i in range(nSimulaties):
    if simulatieStap():
        succes+=1
print(succes,"/",nSimulaties)

```

439 / 10000

Zoals je kan zien is de kans aanzienlijk hoger! Merk wel op dat we ook een dobbelsteen opzij leggen als we 5 verschillende waarden wierpen. In volgende code testen we of het wat uitmaakt :

```
In [10]: def simulatieStap2():
    teWerpen=5 # aantal dobbelstenen die niet opzij gelegd zijn
    opzij=[] # opzij gelegde cijferwaarden
    for i in range(3): # we hebben 3 worpen
        W=werp(teWerpen)
        (d,n)=grootste(opzij+W)
        if n>1:
            opzij=[d]*n
            teWerpen=5-n
        else:
            opzij=[]
            teWerpen=5
    if len(opzij)==5:
        return True
    else:
        return False

nSimulaties=10000
succes=0
succes2=0
for i in range(nSimulaties):
    if simulatieStap2():
        succes2+=1
    if simulatieStap():
        succes+=1
print("strategie 1:",succes,"/",nSimulaties)
print("strategie 2:",succes2,"/",nSimulaties)
```

```
strategie 1: 447 / 10000
strategie 2: 451 / 10000
```

Intuïtief zou de kans licht worden, maar de getallen liggen zo dicht bij elkaar dat het verschil eigenlijk bijna niet te onderscheiden is via simulatie, tenzij je het aantal simulatiestappen erg groot maakt. Toch maar blijven opletten bij wiskunde om te leren hoe je een [exacte oplossing](http://datagenetics.com/blog/january42012/index.html) kan uitrekenen.

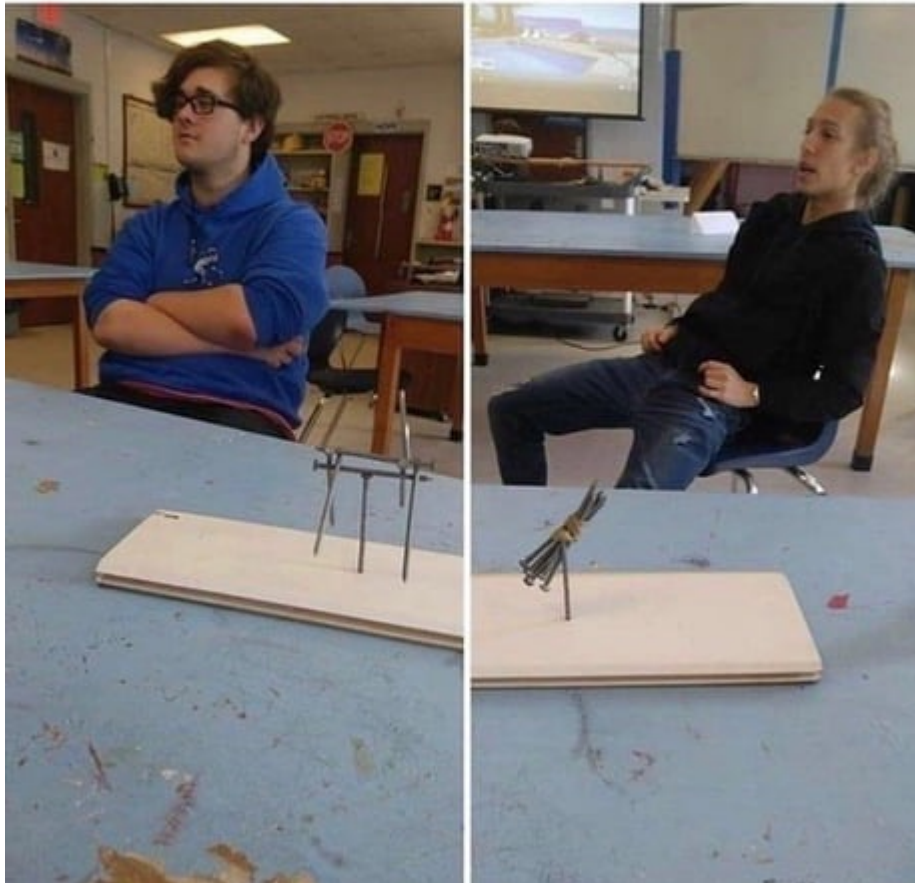
Een uitdaging voor jou: is deze procedure nog verder te optimaliseren? Misschien kan je bij twee keer dezelfde waarde ook beter alles opnieuw werpen? Aan jou om dit uit te zoeken!

In []:

Afsluitende noot

Er zijn honderden manieren (of in ieder geval: heel veel) om te bepalen of een worp yahtzee is. Sommige zijn eleganter, andere sneller, slimmer, of korter. Volg zoveel mogelijk de aanwijzingen die we tijdens de lessen geven, en verder: het belangrijkste is in de eerste plaats dat de code werkt; *it ain't stupid if it works!*

"Balance these 6 nails without letting them touch the wood"



In []: