

# Race conditions in Flask

Bente Schopman

Januari 2019

## Abstract

Het doel van het paper is *om bedrijven en organisaties te beschermen en voorkomen van **Flask** voor de **race conditions***. De hoofdvraag is *Wat moet je doen als er race conditions in Flask zijn?*. De eerste deelvraag was *Wat is een race condition*. Het is onderzocht door een van de 50 websites voor Python te gebruiken. Daarna werd RealPython gebruikt als bron voor het antwoord. Het antwoord daarop is race conditions kunnen voorkomen als meerdere threads dezelfde bron gebruiken.[1]. Dit kan gebeuren als twee threads een *sleep* hebben en een gedeelde resource hebben. De tweede deelvraag was *Hoe ontstaan race conditions in Flask?*. Het is onderzocht door RealPython te gebruiken als bron. Daarna is een praktijkonderzoek gedaan door 2 functies te maken in Flask op basis van de RealPython[1] die er uit kwam in Python en veranderd voor de bruikbaarheid van bij het framework Flask. De derde deelvraag was *Hoe kan je race conditions voorkomen in Flask?*. Het is onderzocht door op de InHolland bibliotheek te zoeken op **how to prevent race conditions** in Python. Daarvoor is op Google gezocht op **best websites for python**. Daar kan CodeConquest met 50 websites uit. Realpython kwam daar uit als een van de 50 websites. Daarna een praktijkonderzoek gedaan door 2 functies te maken in Flask op basis van de RealPython[1] die er uit kwam in Python en veranderd voor de bruikbaarheid van bij het framework Flask. De vierde deelvraag was *Hoe kan je jezelf beschermen tegen race conditions in **Flask***. Het is onderzocht door op de InHolland bibliotheek te zoeken op *prevent race conditions* en te kijken of het voor het vakgebied *ICT* was. Het antwoord op deze vraag is [2]:

1. *legacy code* herschrijven
2. *Locks* gebruiken in de code van de applicatie

## 1 Introductie

Het doel van het paper is *om bedrijven en organisaties te beschermen en voorkomen van **Flask** voor de **race conditions***. De hoofdvraag is *Wat moet je doen als er race conditions in Flask zijn?*. De deelvragen zijn:

1. *Wat is een race condition in Flask?*
2. *Hoe ontstaan race conditions in Flask?*
3. *Hoe kan je race conditions voorkomen in Flask?*
4. *Hoe kan je jezelf beschermen tegen race conditions in Flask?*

In dit paper wordt beschreven wat een race condition is, waarom het een probleem is, wat het veroorzaakt en hoe je het kan voorkomen en ook hoe je er tegen kan beschermen. Het doel van dit

onderzoek is om er voor te zorgen dat bedrijven en personen geen last meer hebben van race conditions bij Flask.

## 1.1 Aanpak

```
Testing update. Starting value is 0.
Testing update. Starting value is 0.
Thread 0: starting update
Thread 0: starting update
Thread 0 has this value 1
Thread 0 has this value 1
Thread 0: finishing update
Thread 0: finishing update
Thread 1: starting update
Thread 1: starting update
Thread 1 has this value 2
Thread 1 has this value 2
Thread 1: finishing update
Thread 1: finishing update
Thread 2: starting update
Thread 2: starting update
Thread 2 has this value 3
Thread 2 has this value 3
Thread 2: finishing update
Thread 2: finishing update
Testing update. Ending value is 3.
Testing update. Ending value is 3.
Testing update. Starting value is 0.
Testing update. Starting value is 0.
Thread 0: starting update
Thread 0: starting update
Thread 0 has this value 1
Thread 0 has this value 1
Thread 0: finishing update
Thread 0: finishing update
Thread 1: starting update
Thread 1: starting update
Thread 1 has this value 2
Thread 1 has this value 2
Thread 1: finishing update
Thread 1: finishing update
Thread 2: starting update
Thread 2: starting update
Thread 2 has this value 3
Thread 2 has this value 3
Thread 2: finishing update
Thread 2: finishing update
Testing update. Ending value is 3.
Testing update. Ending value is 3.
Testing update. Starting value is 0.
Testing update. Starting value is 0.
Thread 0: starting update
Thread 0: starting update
```

Figure 1: Resultaat van Race Conditions Voorbeeld 2

Begrip	Definitie
Thread	De threads van een computerprogramma zorgen ervoor dat er meerdere sequentiele acties kunnen worden of meerdere acties tegelijkertijd uitgevoerd kunnen worden. Elke thread in een programma identificeert een proces dat wordt gerund als het programma daarna vraagt
Lock	Een lock is een mechanisme dat er voor zorgt dat verschillende threads gesynchroniseerd worden door verschillende limieten om er voor te zorgen dat er geen ongelimiteerde toegang is voor een bepaalde resource in een computing environment. Het is een methode die bedoeld is om toegang te controleren door simultaneous control policies toe te passen
Multi-threading	Multi-threading is een soort of execution model dat er voor zorgt dat meerdere threads bestaan in de context van een proces die er voor zorgt dat ze onafhankelijk zijn maar wel dezelfde resource gebruiken.

Figure 2: Begrippen met definities

```
Testing update. Starting value is 0.
Testing update. Starting value is 0.
Thread 0: starting update
Thread 0: starting update
Thread 0 has this value 1
Thread 0 has this value 1
Thread 0: finishing update
Thread 0: finishing update
Thread 1: starting update
Thread 1: starting update
Thread 1 has this value 2
Thread 1 has this value 2
Thread 1: finishing update
Thread 1: finishing update
Testing update. Ending value is 2.
Testing update. Ending value is 2.
Testing update. Starting value is 0.
Testing update. Starting value is 0.
Testing update. Starting value is 0.
Testing update. Starting value is 0.
Thread 0: starting update
```

Figure 3: Resultaat van Race Conditions Voorbeeld 1

```
#!/usr/bin/env python3
import threading
import logging
import time

class FakeDatabase:
    def __init__(self, filename):
        self.value = 0
        self.logfile = filename
        self.log = logging.getLogger(filename)

    def update(self, name):
        self.log.info("Thread (%d): starting update", format(name))
        local_copy = self.value
        local_copy += 1
        time.sleep(0.1)
        self.value = local_copy
        self.log.info("Thread (%d) has this value (%d)", format(name, self.value))
        self.log.info("Thread (%d): finishing update", format(name))
```

Figure 4: Fake Database

Het onderzoek is als volgt aangepakt.

1. Wat is een race condition is Flask?

*Aanpak: Gezocht op **race conditions in Python**. De bronnen zijn gekozen op basis van de Google zoekresultaten en of het een betrouwbare bron is. Dat*

betekent dat het een autoriteit moet zijn. Dus dat het vaak wordt gebruikt als referentie. Daarvoor is op Google gezocht op **best websites for python**. Daar kan CodeConquest met 50 websites uit. Die link daarvoor kan je hier vinden: [3]

2. Hoe ontstaan race conditions in Flask?

Aanpak: Gezocht op **race conditions in Python**. De bronnen zijn gekozen op basis van de Google zoekresultaten en of het een betrouwbare bron is. Dat betekent dat het een autoriteit moet zijn. Dus dat het vaak wordt gebruikt als referentie. Daarvoor is op Google gezocht op **best websites for python**. Daar kan CodeConquest met 50 websites uit. Realpython kwam daar uit als een van de 50 websites. Daarna een praktijkonderzoek gedaan door 2 functies te maken in Flask op basis van de RealPython[1] die er uit kwam in Python en veranderd voor de bruikbaarheid van bij het framework Flask. Het antwoord op deze vraag is om een race condition te voorkomen in Flask, kan je het beste een *lock* gebruiken. Een *lock* is een mechanisme die ervoor zorgt verschillende threads gesyn-

choniseerd worden waarbij er limieten zijn om ervoor te zorgen dat er niet ongelimiteerd gebruik van een bepaalde gedeelde *source* in een *computing environment*.

3. Hoe kan je race conditions voorkomen in Flask?

Door op de InHolland bibliotheek te zoeken op **how to prevent race conditions in** Daarvoor is op Google gezocht op **best websites for python**. Daar kan CodeConquest met 50 websites uit. Realpython kwam daar uit als een van de 50 websites. Daarna een praktijkonderzoek gedaan door 2 functies te maken in Flask op basis van de RealPython[1] die er uit kwam in Python en veranderd voor de bruikbaarheid van bij het framework Flask.

4. Hoe kan je beschermen tegen race conditions in Flask?

Door op de Inholland bibliotheek te zoeken op *prevent race conditions* en te kijken of het voor het vakgebied ICT was.

## 2 Begrippen

De begrippen zijn te zien in *figure 2*. De begrippen zijn gemaakt door definities van [4] en [5] en [6].

## 3 Wat is een race condition

Race Conditions kunnen voorkomen als meerdere threads dezelfde bron gebruiken.[1]. Dit kan gebeuren als twee threads een *sleep*time hebben en een gedeelde resource hebben. Dat kan je zien in het volgende voorbeeld van [7]:

```
Thread 123145539575808 started...
Thread 123145539575808 read value 0
Thread 123145544830976 started...
```

```
Thread 123145544830976 read value 0
Thread 123145539575808 slept for 1
Thread 123145539575808 write 1
Thread 123145544830976 slept for 3
Thread 123145544830976 write 1
```

Eigenlijk verwacht je het resultaat 2 maar het resultaat is 1. Dat is dus een goed voorbeeld van een race condition.

## 4 Oorzaak voor race conditions in flask

De oorzaak voor race conditions in Flask is dat meerdere threads de dezelfde source willen gebruiken. De meeste race conditions zijn lastig om te ontdekken. [1] Dit wordt ook onderzocht door de bron [1] te gebruiken om twee voorbeelden van race conditions te laten zien. De code daarvoor is te vinden op [https://github.com/SchopmanBente/race\\_conditions\\_flask](https://github.com/SchopmanBente/race_conditions_flask)[8]. Het eerste voorbeeld gaat over een gedeelde variabele die door twee threads wordt geüpdatet. De FakeDatabase van dit voorbeeld is te zien in *figure 4*. De uitkomst is te zien in *figure 3*.

## 5 Race conditions voorkomen in Flask

Om een race condition te voorkomen in Flask, kan je het beste een *lock* gebruiken. Een *lock* is een mechanisme die ervoor zorgt verschillende threads gesynchroniseerd worden waarbij er limieten zijn om ervoor te zorgen dat er niet ongelimiteerd gebruik van een bepaalde gedeelde *source* in een *computing environment*. Het is toegepast in de applicatie bij het tweede voorbeeld. De tweede functie is een applicatie

met de twee threads daarbij heeft de database.update-functie een log en een lock. Dat is te zien in *figure 1*. De FakeDatabaseTwo is te zien in *figure 5*.

## 6 Beschermen tegen race conditions

Om je te beschermen tegen race conditions kan je volgens Daniel G [2] twee dingen doen:

1. *legacy code* herschrijven
2. *locks* gebruiken in de code van de applicatie

## 7 Conclusie

De hoofdvraag is *Wat moet je doen als er race conditions in Flask zijn?*. De deelvragen zijn:

1. *Wat is een race condition in **Flask**?*
2. *Hoe ontstaan race conditions in **Flask**?*
3. *Hoe kan je race conditions voorkomen in **Flask**?*
4. *Hoe kan je jezelf beschermen tegen race conditions in **Flask**?*

De deelvragen zijn als volgt aangepakt:

1. Wat is een race condition is beantwoord door te zoeken op **race conditions in Python**.

De bronnen zijn gekozen op basis van de Google zoekresultaten en of het een betrouwbare bron is. Dat betekent dat het een autoriteit moet zijn. Dus dat het vaak wordt gebruikt als referentie. Daarvoor is op Google gezocht op **best websites for python**. Daar kan CodeConquest met 50 websites uit. Die link daarvoor kan je hier vinden: [3]

2. Hoe ontstaan race condition in **Flask** is beantwoord door te zoeken op **race conditions in Python**. De bronnen zijn gekozen op basis van de Google zoekresultaten en of het een betrouwbare bron is. Dat betekent dat het een autoriteit moet zijn. Dus dat het vaak wordt gebruikt als referentie. Daarvoor is op Google gezocht op **best websites for python**. Daar kan CodeConquest met 50 websites uit. Realpython kwam daar uit als een van de 50 websites. Daarna een praktijkonderzoek gedaan door 2 functies te maken in Flask op basis van de RealPython[1] die er uit kwam in Python en veranderd voor de bruikbaarheid van bij het framework Flask.
3. Hoe kan je race conditions voorkomen in **Flask** is beantwoord door op de InHolland bibliotheek te zoeken op **how to prevent race conditions in**. Daarvoor is op Google gezocht op **best websites for python**. Daar kan CodeConquest met 50 websites uit. Realpython kwam daar uit als een van de 50 websites. Daarna een praktijkonderzoek gedaan door 2 functies te maken in Flask op basis van de

RealPython[1] die er uit kwam in Python en veranderd voor de bruikbaarheid van bij het framework Flask.

4. Hoe kan je jezelf beschermen tegen race conditions in **Flask** is beantwoord door op de InHolland bibliotheek te zoeken op *prevent race conditions* en te kijken of het voor het vakgebied *ICT* was.

Race Conditions kunnen voorkomen als meerdere threads dezelfde bron gebruiken.[1]. Dit kan gebeuren als twee threads een sleeptime hebben en een gedeelde resource hebben. De oorzaak voor race conditions is dat meerdere threads dezelfde *source* gebruiken. Om race conditions te voorkomen kan je het beste een *lock* gebruiken. Een lock is een mechanisme dat er voor zorgt dat verschillende threads gesynchroniseerd worden waarbij er limieten zijn om er voor te zorgen dat er niet ongelimiteerd gebruik gemaakt kan worden van een bepaalde gedeelde *source* in een *computing environment*. Om je te beschermen tegen race conditions kan je de volgende dingen doen volgens [2]:

1. *legacy code* herschrijven
2. *Locks* gebruiken in de code van de applicatie

## References

- [1] J. Anderson. An intro to threading in python, 2019. Last accessed 5 januari 2020.
- [2] Politics Government Week and Business Insights: Global. "prevention of race conditions in library code through memory page-fault handling mechanisms" in patent application approval process.", 7

- Maart 2013. Laast bezocht op 17 januari 2020.
- [3] Code Conquest. The 50 best websites to learn python, 2016. Last accessed 5 januari 2020.
- [4] TechTerms. Thread, z.d. Laast bezocht op 17 januari 2020.
- [5] Techopedia. Lock, z.d. Laast bezocht op 17 januari 2020.
- [6] Techopedia. Multithreading, z.d. Laast bezocht op 5 februari 2020.
- [7] M. M. Tahrioui. *asyncio Recipes*. Mohamed Mustapha Tahrioui, Darmstadt, Hessen, Germany, 2019.
- [8] SchopmanBente. *race\_conditions\_flask*, z.d. Laast bezocht op 5 februari 2020.

```

class FakeDatabaseTwo:
    def __init__(self, name):
        self.value = 0
        logger = Logger(f"{name} - LOG")
        self.log = logger

    def update(self, name):
        self.log.info("Thread (8): starting update", format(name))
        local_copy = self.value
        local_copy += 1
        time.sleep(0.1)
        self.value = local_copy
        self.log.info("Thread (8) has this value (1)", format(name, self.value))
        self.log.info("Thread (8): finishing update", format(name))

```

Figure 5: Fake Database Two