PO Informatica Galgje (met Python)

In deze opdracht gaan we het galgje-spel bouwen: een persoon speelt tegen een computer. Deel de opdracht op in stukjes. Probeer het steeds zo te programmeren dat je tussendoor zo veel mogelijk stukjes kunt testen zonder dat het spel helemaal af hoeft te zijn. Zo weet je zeker dat je altijd een werkend product hebt. Hoe verder je komt, hoe hoger je cijfer. Het is de bedoeling dat je zo veel mogelijk **zelfstandig** aan de slag gaan, zonder te veel hulp van de docent. Help elkaar als je vastloopt. Laat zien wat je geleerd hebt, dus ook hoe je fouten uit je eigen code kunt halen.

Toelichting opdracht

Samenwerking: Het wordt **aanbevolen** om in tweetallen te werken (drietallen zijn niet toegestaan).

Inleveren op: zie planner (Te laat inleveren betekend een half punt aftrek van je cijfer, met daarna per 24 uur weer een half punt aftrek)

Inleveren in Teams > Opdrachten:

1. Verslag:
    - (een foto van) je stroomdiagram
    - Korte uitleg van wat wel/niet werkt, en welke uitbreidingen je hebt toegevoegd.
2. Je python code.

Je mag elkaar helpen, maar iedereen moet zijn eigen code schrijven en inleveren. Kopieer niet van iemand anders! Plagiaat wordt niet getolereerd. Let op: Je (geheime) woord moet op deze manier in een lijst worden opgeslagen: [“v”,”a”,”a”,”s”], en hier werkt je programma mee verder (zie Deel B). Let op: zonder vooraf toestemming van Renske wordt elk ander aanpak **niet** geaccepteert.

Beoordeling:

Je **proces** wordt beoordeeld op:	
Voortgang, Verslag & Ontwerp	Je programma en verslag zijn compleet en op tijd en op de juiste wijze ingeleverd (zie toelichting opdracht). Je werkt zelfstandig en lost zelf (of met andere leerlingen) fouten in de code op. Een stroomdiagram is vooraf gemaakt. Je ontwerp maakt duidelijk uit welke eenheden je programma is opgebouwd (zie **Deel A**). Je ontwerp wordt gedurende het gehele programmeertraject gebruikt als leidraad. Er is een duidelijke overeenstemming tussen je ontwerp en je code.
Je **product** wordt beoordeeld op:	
Correctheid, Volledigheid & Originaliteit	De basis van het programma is af volgens de omschrijving in **Deel B**. Hoe verder je komt qua functionaliteit (**Deel C**: uitbreidingen, originele invulling), hoe hoger je cijfer. Je uitbreidingen staan kort beschreven in je verslag. Je code voldoet aan de verwachting volgens omschrijving **Deel B**. Je programma is robuust en geeft duidelijke foutmeldingen bij onvoorziene omstandigheden (bv. ongeldige invoer). In je verslag staat wat wel/(nog)niet goed werkt.
Documentatie & Presentatie	Code is makkelijk te lezen en begrijpen. **Commentaar** is aanwezig. Bij blokken code (bv. wat bij een loop of een functie hoort) wordt kort samengevat wat het doel van dat blok is. Zowel keuzes als problemen in je code worden met commentaar toegelicht. Nieuwe constructies die je jezelf hebt eigen gemaakt worden middels commentaar toegelicht. **Namen** beschrijven de bedoeling nauwkeurig en zijn compleet, onderscheidend, beknopt, correct gespeld en hebben consistent gebruik van conventies (camelUpperCase of met_streepjes) en constanten met hoofdletters. Functies zijn middels werkwoorden omschreven, variabelen middels zelfstandige naamwoorden. De **opmaak** is consistent en logisch gestructureerd. Vergelijkbare delen van code duidelijk herkenbaar en staan bij elkaar: globale variabelen, definities en hoofdprogramma.
Constructie	De kwaliteit van de code: het is duidelijk, efficiënt, elegant, logisch en goed gestructureerd. De **flow** is eenvoudig. Het meest gebruikelijke pad door de code is duidelijk. De hoofdprogramma leest als een inhoudsopgave en geeft op hoofdlijnen een duidelijk overzicht van wat het programma doet. Er wordt goed gebruik gemaakt van loops en condities. Coderegels en condities zijn kort, eenvoudig en makkelijk te begrijpen. Er komen geen lange of diep geneste brokken voor brokken code voor. Er komt geen code dubbel voor. Er komt geen code voor dat nooit uitgevoerd wordt. Gebruik van 'harde' waardes (echte getallen) worden zo veel mogelijk vermeden. Er wordt geen gebruik gemaakt van break(), quit of while(true). **Functies** hebben één (duidelijke) doel, en zijn kort-en-krachtig. Er wordt gebruik gemaakt van parameters en retourwaarden (om het gebruik van globale variabelen te beperken). Returns worden op logische, eenduidige en overzichtelijke wijze gebruikt. Variabelen worden niet hergebruikt voor verschillende doeleinden.

Goed programmeren gaat dus **niet alleen** om een werkende **oplossing**, maar juist om een **degelijke aanpak en oplossing. Laat zien wat je geleerd hebt!** Ga gestructureerd te werk, bedenk uit welke losse onderdelen je programma bestaat en pak ze los van elkaar aan.

DEEL A: ONTWERP

Voordat we gaan beginnen met het spel ga je eerst een stroomdiagram tekenen waarin je het spel in logische delen opbreekt. Gebruik de onderstaand uitleg van deel B om je ontwerp te maken. Hieruit blijkt welke onderdelen achter elkaar uitgevoerd worden, en waar herhaling zit en wat de voorwaarden voor herhaling zijn. Ook blijkt daar uit welke dingen je moet onthouden, oftewel, welke variabelen of constanten je hebt. Maak een lijstje van de gegevens die je moet opslaan/onthouden en wat hun typen zijn (bv. String gebruikersgok). Mogelijk zal je gaandeweg aanpassingen maken en tot nieuwe inzichten komen. Pas je stroomdiagram daaropaan.

Je kan je ontwerp prima op papier schetsen en daar een foto van maken. Voor het (digitaal) tekenen van een stroomdiagram kun je gebruik maken van:
http://course.cs.ru.nl/pythonVO/FlowchartTool/Stream.html

DEEL B: BASISPROGRAMMA

Begin simpel! Zorg eerst dat de basis van je programma werkt, en breid het daarna uit.

Het simpelste programma zal er voor de gebruiker als volgt uit zien:

```
Hoi. Welkom bij Galgje.
Ik heb een woord in gedachten.

Tot nu toe geraden woord: _ _ _ _ _
Je mag nog 10 keer fout gokken.
Tot nu toe geraden letters:

Wil je een letter gokken? e
Goed geraden! De letter ‘e’ zit in mijn geheime woord.
Tot nu toe geraden woord: _ _ e _ _
Je mag nog 10 keer fout gokken.
Tot nu toe geraden letters: e

Wil je een letter gokken? r
Helaas, de letter ‘r’ zit niet in mijn geheime woord.
Tot nu toe geraden woord: _ _ e _ _
Je mag nog 9 keer fout gokken.
Tot nu toe geraden letters: e, r
…
```

Hieronder staat uitgelegd hoe je de programmeeromgeving in moet stellen. Daaronder staat waar het programma aan moet voldoen en tips om deze te maken.

> **Opzetten programmeeromgeving:** Volg de stappen in de Tutorial voor Visual Studio Code:
>
> http://course.cs.ru.nl/pythonVO/docs/PO/VisualStudio_installeren.pdf. Hier wordt uitgelegd welke programmeeromgeving je het beste kunt gebruiken en hoe je dat de eerste keer moet instellen.

Waar het programma aan moet voldoen en extra tips:

a) *Geheim woord:* Omdat we wel hebben geleerd om met lijsten te werken, maar nog niet met strings, gaan we het geheimwoord in een lijst onthouden. Maak een lijst en zet daarin de letters van het woord, dus bijvoorbeeld: [“v”,”a”,”a”,”s”].
b) *Geraden woord:* Maak een ‘geraden woord’ met een juiste hoeveelheid puntjes.
c) *Controle:* Controleer of een letter in het geheime woord voorkomt. Zo ja, print dit. Vervang ook op de juiste plek in het geraden woord het puntje door de letter. Zorg ervoor dat alle voorkomens van het letter vervangen wordt. Tip: begin maak hier een functie (def) van zodat je code overzichtelijker blijft.
d) *Gok:* Laat de gebruiker een gokletter in voeren.
e) *Foute gok:* geef de gebruiker ook terugkoppeling als de gok niet in het woord voorkomt.
f) *Beurten:* Het spel zelf komt eigenlijk neer op een grote loop. Bij elke iteratie van de loop is de ´gebruiker een keer aan de beurt´. Aan welke condities moet zijn voldaan voordat de gebruiker nog een keer aan de beurt mag komen? Hoe bepaal je of ‘ie niet al verloren heeft? En hoe bepaal je of ‘ie gewonnen heeft?
g) *Spelstand:* Toon de huidige spelstand (aantal pogingen, geraden woord tot nu toe) aan de gebruiker, en vraag om een nieuwe letter. Je hoeft (nog) geen rekening te houden met foutieve input (getallen, hele zinnen, of letters die al geraden zijn), dit doe je later.
h) *Einde spel:* Toon een bericht aan het einde wanneer de gebruiker heeft gewonnen of verloren. Waar kan je aan zien of de gebruiker won of verloor? Onderscheid deze twee gevallen. Laat ook nog even zien wat het te raden woord was.
i) *Bereken woord lengte:* Indien je dit nog niet gedaan hebt, pas je code zo aan dat het werkt voor een woord van elke willekeurige lengte. Gebruik dus geen getal voor de waarde, maar bereken deze.
j) *Willekeurige woorden*: Voeg meer geheime woorden toe (in een lijst) en laat het programma er eentje willekeurig kiezen. Tip1: Zoek op internet naar de Python functie `choice()`. Tip2: Tijdens het testen kan het handig zijn om deze af te drukken.
k) *Omgaan met foutieve invoer*: Zorg voor een goede afhandeling van foutieve invoer. Controleer wat de gebruiker invoert, geef een duidelijke foutmelding en geef de gebruiker de mogelijkheid om de fout te herstellen. Bijvoorbeeld, bij een foutieve invoer (getallen, hele zinnen of helemaal geen invoer).

*Je hebt al een aantal **vergelijkbare opdrachten** gemaakt. Het kan handig zijn om terug te kijken naar hoofdstuk 9 en hoofdstuk 10, met name opgaven 9.4.1, 10.4.1 en afsluitende opgave 9.4.*

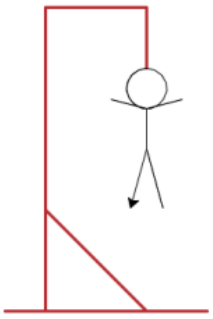
DEEL C: UITBREIDINGEN

- Wanneer de gebruiker een bepaalde letter al geraden heeft, is het een beetje zonde als het nog een gok kost wanneer hij of zij die letter per ongeluk herhaalt. Hoe kan je handig bijhouden welke letters al geraden zijn? Bedenk wanneer je een letter toe moet voegen, en voorkom dat een letter twee keer fout gerekend wordt.
- Je kunt ook de gebruiker vragen om een woord in te vullen. Zo kan je het spel met twee mensen spelen. Zorg er wel voor dat je het scherm even leeg maakt zodat de ander niet ziet wat je hebt ingevuld. Dat kan natuurlijk met een aantal lege regels te printen, maar dat kan ook netter. Met de juiste zoektermen in Google moet je het juiste commando wel kunnen vinden.

- Geef de gebruiker telkens willekeurig verschillende zinnen als die een fout of juist een goede gok heeft. Zo kan een speler telkens weer iets anders te zien krijgen.
- Je kunt ook een speler meerdere potjes laten spelen en de score bijhouden en telkens laten zien nadat een potje voorbij is. Je kunt ook een 'high-score' bij lagen houden, namelijk het minst aantal gokken dat nodig zijn geweest om een potje te winnen.
- Teken een galg door een combinatie van streepjes, sterretjes, nulletjes, etc. af te drukken.

```
 _________
|         |
|         0
|        /|\
|        / \
|
|
```

- Om je code voor het tekenen van de galg overzichtelijk te houden is het goed om hier een aparte functie van te maken. Welke informatie heeft deze functie allemaal nodig? Hoe kan je deze functie op een nette manier opschrijven?
- Combineer wat je geleerd hebt bij TurtleGraphics met het spel wat je gemaakt hebt om een galg te tekenen. Ga na waar je precies een stukje moet tekenen. Om het venster open te houden moet je aan het **einde van je code** turtle.mainloop() aanroepen (maak geen gebruik van turtle.done()).



Figuur 1: Een voorbeeld van een leuke galg

- Voeg zelf iets origineels of iets slims toe waardoor je programma meer kan of waardoor je programma leuker of je code eleganter of efficiënter wordt.

Een voorbeeld opzet van je code zou het volgende kunnen zijn:

```
# DIT IS HET GALGJE SPEL
# GESCHREVEN DOOR RENSKE
# Doel: het programma heeft een geheim woord,
# de gebruiker mag letter voor letter raden of die letter in het woord voorkomt
# de spel is afgelopen als de gebruiker het juiste woord geraden heeft, of teveel fouten gemaakt heeft

#LET OP: dit programma is nog lang niet af.

####################
#GLOBALE VARIABELEN
####################
geheim_woord = ['h','a','l','l','o']  #dit is wat de gebruiker moet gaan raden
tot_nu_toe_geraden_woord = []         #dit is wat de gebruiker tot nu toe goed heeft geraden
resterende_pogingen = 8               #maximaal pogingen

###############
# FUNCTIE DEFINITIES
###############
def welkomstWoord():
  print("Welkom bij Galgje!")
  print("Dit zijn de spelregels ...")

# Deze functie vult de geraden woord aan met even veel streepjes als dat het geheim woord lang is
def maakWoordMetStreepjes():
  streepjes_woord = []
  for i in range(len(geheim_woord)):        #voor elk letter in geheim woord
    streepjes_woord.append("_")   #per letter in het te raden woord aanvullen met streepjes
  print("tot_nu_toe_geraden_woord", streepjes_woord)  #even printen om te testen
  return streepjes_woord
```

```
# Deze functie verlaagt aantal pogingen bij een foute gok,
# anders aantal pogingen niet aanpassen
def werkAantalPogingenBijAlsGokFout(gok, resterende_pogingen):
  if not gok in geheim_woord: #gok was fout
    resterende_pogingen -= 1
  print("aantal resterende pogingen:", resterende_pogingen) #even printen
  return resterende_pogingen

#################
# HOOFDPROGRAMMA
#################
welkomstWoord()
print("geheim_woord", geheim_woord) #even printen om te testen

tot_nu_toe_geraden_woord = maakWoordMetStreepjes() #leeg woord aanvullen met juist aantal streepjes

#zolang het spel nog niet is afgelopen:
gok = input ("Geef een letter: ")     #vraag om gokletter
resterende_pogingen = werkAantalPogingenBijAlsGokFout(gok, resterende_pogingen)  #als fout geraden, dan aantal
pogingen bijwerken
# ...
```