SGDB Python opgaven

Je leert de basics met de cursus, maar je wordt het pas meester met oefeningen. Kijk in de studiewijzer om te zien welk deel van de Python cursus je eerst moet doen om een bepaalde opgave te maken. Zulke opgaven kun je ook op de praktijktoetsen verwachten.

Je maakt deze opgaven niet met de editor van de cursus, maar met Thonny. Deze vind je in een mapje “in je startmenu, inde submap “Informatica”. Op de informatica site staat een downloadlink voor thuis (ook voor Mac).

1. Wisselkoers

Je gaat op vakantie naar Amerika en wilt graag dollars opnemen. Je schrijft een python script om je te helpen met omrekenen van Euro’s naar Dollars en andersom.

1,00 euro is op dit moment 0,90 dollar waard

Stap 1:

- Maak een variabele met de naam `invoerEuros`
- “Vul” deze variabele met een bedrag (getal) in Euro’s dat je wilt omrekenen (je mag decimalen gebruiken maar zorg wel dat je een punt gebruikt en geen komma (dus 2.45 i.p.v. 2,45)
- Maak en variabele uitvoerDollars aan.
- Reken het bedrag uit `invoerEuros` om naar Dollars en sla het resultaat op in `uitvoerDollars`
- Print de waarde van `uitvoerDollars` op het scherm met het `print()` commando.

Stap 2 (test eerst stap 1):

We gaan ook nog andersom omrekenen. Van Dollars naar Euro’s

- Maak een variabele met de naam `invoerDollars` en “vul” deze variabele met een bedrag (getal) in dollars.
- Reken het bedrag uit `invoerDollars` om naar Euro’s en sla het resultaat op in een variabele `uitvoerEuros`
- Zorg dat je script de waarde van `uitvoerEuros` ook op het scherm afdrukt met `print()`

Stap 3 (Extraatje):

Zorg dat het resultaat wordt afgerond op 2 decimalen. Dit komt pas later in de cursus aan bod, maar hier vind je alvast hoe het moet:
Met het commando `round()` kun je een variabele afronden op het gewenste aantal cijfers. Dus het commando `round(getal, 3)` zal de variabele `getal` afronden op 3 decimalen.
Gebruik dit om de 2 uitkomsten af te ronden op 2 decimalen.

2. Hallo gebruiker!

In de introles van Wt heb je gezien dat je de gebruiker om invoer kunt vragen (dit komt verderop in de cursus ook nog aan bod). Dit gaan we gebruiken in deze opgave. Het vragen van de gebruiker om invoer kan op de volgende manier:

```
kleur = input("Wat is je favoriete kleur? ")
```

Dit zal voor de gebruiker het bericht “Wat is je favoriete kleur?“ afdrukken op het scherm en daarna wacht het programma op invoer. Als de gebruiker wat intypt en op enter drukt, wordt de invoer in de variabele `kleur` opgeslagen.

De opdracht is als volgt:
Vraag de gebruiker om zijn naam en om zijn leeftijd (dat zijn 2 verschillende inputs natuurlijk!) en sla deze op in variabelen met elk een zinnige variabele naam.

Je programma vertelt de gebruiker vervolgens de volgende dingen:

- Groet de gebruiker (Dus zeg zoiets als “Hallo <naam>!”) (met voor <naam> zijn ingevoerde naam natuurlijk)

- Vertel de lengte van zijn/haar naam (het aantal tekens dus). Voor het opvragen van de lengte van dingen kent python de functie `len()`. Bijvoorbeeld:
  `print(len("hallo"))`
  Uitvoer: `5`

- Print het geboortejaar van de gebruiker op het scherm. *(We nemen voor het gemak even huidig_jaartal – leeftijd als berekening. Als de gebruiker dit jaar nog jarig moet worden klopt dit natuurlijk niet, maar dat geeft voor nu even niet...)*

  Hierbij loop je tegen het volgende aan:
  Python maakt verschil tussen tekst (`strings`) en getallen (`integers` zijn gehele getallen, `floats` zijn decimale getallen). Als je het commando `input()` gebruikt om de gebruiker om invoer te vragen, zal het resultaat altijd een `string` (tekst) zijn. Als je dit vervolgens direct wilt printen is dat geen probleem, maar in dit geval willen we een berekening uitvoeren. Als je dus zou proberen het volgende te doen:

  ```
  print(2019-leeftijd)
  ```

  Dan krijg je een foutmelding. Python weet niet hoe hij een tekst (`leeftijd`) van een getal (`2019`) moet aftrekken. Dit is op te lossen door de invoer van de gebruiker om te zetten naar een getal. Bijvoorbeeld:

  ```
  getal_invoer = int(input("Geef een getal"))
  ```

  Hierbij wordt de uitvoer van de functie `input()` meteen de invoer van de functie `int()`. Deze functie zet een `string` om in een geheel getal (`integer`), tenslotte wordt dit integer resultaat in de `getal_invoer` variabele opgeslagen. Als de gebruiker iets geks invult (bijvoorbeeld “appeltaart”), dan zal je programma crashen.

  Gebruik deze nieuwe kennis om het geboortejaar van de gebruiker op het scherm te printen.

- De ingebouwde python functie `pow()` kan machtsverheffen. `pow(x,y)` zal x tot de macht y verheffen. Vraag de gebruiker om zijn geluksgetal (denk aan de conversie naar int!). Verhef het

geluksgetal van de gebruiker tot de macht van zijn leeftijd en print het resultaat op het scherm
NB: Machtsverheffen kun je ook als volgt schrijven in Python: `x ** y` *(is x tot de macht y)*

- *Lollige extra: Je kunt strings ook "vermenigvuldigen". Dat zorgt ervoor dat de string een aantal keer herhaald wordt. Bijvoorbeeld:*

  ```
  print(4 * "hoi")
  ```

  Dit drukt "hoihoihoihoi" af op het scherm.

  Gebruik dit om 100 keer de naam van de gebruiker af te drukken

3. Fahrenheit

Voor je vakantie naar Amerika heb je bij opdracht 1 al een omrekenprogramma van Euro's naar Dollars gemaakt. Je hebt net het weerbericht opgezocht voor je vakantiebestemming, maar die geeft de temperatuur in Fahrenheit. Je doet wat je altijd doet als je een probleem hebt: je pakt Python erbij en schrijft een oplossing 😉

- Vraag de gebruiker om de temperatuur in Fahrenheit op te geven (met een `input()`).
- De gebruiker moet ook decimale waardes in kunnen vullen, dus zorg voor goede conversie van de invoer naar `float`.
- Reken de invoer in Fahrenheit om naar Celcius met de volgende formule:

$$\text{Celsius} = \frac{\text{Fahrenheit} - 32}{1.8}$$

- Spiek even bij opdracht 1 hoe je de uitkomst ook alweer kunt afronden op 1 decimaal.
- Print de uitvoer in deze specifieke vorm op het scherm (let op de apostrof!):

```
86.5 graden Fahrenheit? Da's hetzelfde als 30.3 graden Celcius!
```

Extra:

- *Schijf ook een versie die andersom rekent: van Celcius naar Fahrenheit. De formule daarvoor is:*

$$\text{Fahrenheit} = \text{Celsius} \times 1.8 + 32$$

4. Sommengenerator

Je hebt een broertje op de basisschool en hij heeft moeite met de rekenlessen. Je besluit jouw Python skills in te zetten om een sommen-oefenprogramma voor hem te maken. We beginnen met optelsommen en breiden het daarna verder uit. Eerst moeten we wat weten over het genereren van random getallen.

Random getallen:

Python heeft allerlei verschillende handige bibliotheken (libraries) met code en commando's, die je kunt gebruiken in je code. Je moet hiervoor dan aan het begin van je code de bibliotheek die je wilt gebruiken importeren (met het `import` commando, gevolgd door de naam van de bibliotheek).

Er zijn bibliotheken voor het manipuleren van afbeeldingen, bibliotheken voor het bedrijven van statistiek, bibliotheken die je helpen games te maken, etc., etc. Voor deze opgave gaan we werken met de `random` bibliotheek. Deze bibliotheek kun je gebruiken om onder andere willekeurige getallen te genereren.

- Importeer de bibliotheek met de naam `random` in je programma door op de eerste regel van je code te zetten:
  ```
  import random
  ```
- Nu kun je in de rest van je code met (bijvoorbeeld) met het commando `random.randint(1,20)` een willekeurig getal tussen de 1 en de 20 genereren.

Laat je programma nu 2 random getallen (a en b) kiezen tussen de 1 en de 100. Print deze getallen als een optelsom op het scherm. Bijvoorbeeld:
44 + 81

Voer je programma een paar keer uit om te zien of hij steeds andere getallen gebruikt.

Automatisch nakijken:

Je zou je broertje een stuk beter helpen als zijn antwoorden ook automatisch nagekeken worden door je programma.

- Vraag de gebruiker om het antwoord op de afgebeelde som in te voeren.
- Reken in je programma het goede antwoord uit door a en b bij elkaar op te tellen
- Vergelijk het goede antwoord met het gegeven antwoord en druk af op het scherm of de gebruiker het goed had of niet. Als hij het fout had, geef dan ook weer wat het antwoord had moeten zijn.
- Zorg dat de uitvoer van je programma wat netter aangekleed wordt. Je uitvoer moet zijn zoals in de volgende 2 voorbeelden:

Vb1:
```
Hoeveel is 21 + 66?
86
Helaas fout. Het goede antwoord was: 87
```

Vb2:
```
Hoeveel is 23 + 16?
39
Goedzo!
```

Extra:
Dit komt eigenlijk pas wat later in de stof, maar het is hier alvast wel handig: het `else`-statement. Als je met `if` een vraag stelt en je wilt ook iets doen als de voorwaarde niet waar is, kun je dat direct doen met een `else` en hoef je geen tweede `if` te gebruiken. Bijvoorbeeld:

```
if leeftijd > 65:
    print('Geniet van uw pensioen!')
else:
    print('Nog even lekker doorwerken loonslaaf!')
```

Herschrijf je code waarbij je gebruik maakt van `else` i.p.v. een tweede `if` .

Kies de moeilijkheidsgraad:

We willen zorgen dat je broertje zelf zijn moeilijkheidsgraad kan kiezen. Aan het begin van ons programma gaan we de gebruiker daarom vragen wat het hoogste getal is dat in de sommen mag voorkomen (tot nu toe was dat 100). Sla dit op in een variabele `moeilijkheid`.
We passen nu het genereren van de random getallen aan, zodat deze rekening houdt met het opgegeven maximum

Verschillende soorten sommen:

Het zou tof zijn als je programma niet alleen optelsommen genereert, maar ook vermenigvuldigings-, delings- en aftreksommen. We laten de gebruiker kiezen wat hij wil. Vraag hem via een input() om het volgende:

```
Kies je som: (o)ptellen, (a)ftrekken, (v)ermenigvuldigen, (d)elen
```

Hierbij gaan we ervan uit dat de gebruiker de eerste letter van zijn keuze invoert. (dus bv. een v voor vermenigvuldigen). Ga daarna aan de slag met zijn keuze. Genereer een som van het juiste type en druk deze af op het scherm. Reken het juiste antwoord uit en vertel de gebruiker of hij het goed heeft gedaan.

Als de gebruiker iets anders invoert dan een `o`, `a`, `v` of `d` druk je een foutmelding op het scherm

Extra:
Ook dit komt eigenlijk pas wat later in de stof, maar het is hier ook wel handig: het `elif`-statement. Als je meer dan 2 opties wil checken met `if`, heb je niet genoeg aan een if en een else. Je kunt met elif de verschillende opties "aan elkaar rijgen" tot een samengestelde vraag waarvan altijd maar 1 optie wordt uitgevoerd. Bijvoorbeeld:

```
if leeftijd > 65:
    print('Geniet van uw pensioen!')
elif leeftijd <= 18:
    print('Lekker lamballen op school')
else:
    print('Nog even lekker doorwerken loonslaaf!')
```

Herschrijf je code van hiervoor tot een if-elif-else constructie.

5. Spelen met je string

Je hebt in de cursus wat nieuwe operaties geleerd die je met strings kunt uitvoeren. Daar gaan we even lekker mee spelen.

We gaan een programma schrijven dat de gebruiker om 2 woorden als invoer vraagt en daar de volgende dingen mee doet:

- De woorden printen met de eerste letters van beide woorden verwisseld
- De woorden printen met de tweede helft van beide woorden verwisseld
- De woorden printen, waarbij de eerste letters van beide woorden in hoofdletters zijn omgezet

Hier zie je een voorbeeld in- en uitvoer van het programma:

```
Geef woord 1: appel
Geef woord 2: taart
Letters verwisseld: tppel aaart
Helften verwisseld: apart tapel
Met hoofdletters: Appel Taart
```

Stap 1: invoer

Deze is eenvoudig. Vraag de gebruiker met behulp van `input()` 2x om een woord en sla deze beide op in een variabele.

Stap 2: Letters verwisselen

Hierbij moet je een manier verzinnen om de eerste letter van een woord te “pakken”. Dit doe je door de juiste index van de string te kiezen. Bijvoorbeeld:

```
print ("appeltaart"[3])
```
heeft als resultaat:
```
e
```

(Let op: de indexen beginnen bij 0 te tellen, dus index 3 is letter nummer 4!).

Daarna moet je een truuk verzinnen om de rest van de letters van het woord te “pakken”. Hiervoor moet je een slice opgeven. Een slice wijst naar een reeks letters uit de string. Bijvoorbeeld:

```
print ("appeltaart"[2:7])
```
heeft als resultaat:
```
pelta
```

Dit print dus letters vanaf index 2 tot (en zonder) index 7.

Als je de eerste index van een slice weglaat, start de slice bij het begin van de string. Als je de 2e index van de slice weglaat, eindige de slice bij het einde van de string:

```
print ("appeltaart"[:7])
```
heeft als resultaat:
```
appelta
```

Onthoud dat je strings “aan elkaar kunt plakken” met +. Dus:

```
x = "appel" + "taart"
print(x)
```
heeft als resultaat:
```
appeltaart
```

Nu zou je het voor elkaar moeten krijgen om de woorden te printen met de 1e letters omgewisseld

Stap 3: helften verwisselen

Om de helften te verwisselen, moet je de woorden “in tweeën hakken”. Dat kan met slices, zoals hierboven is uitgelegd. Dan moet je wel nog even weten wat de index is van het midden van het woord

Met het commando `len()` kun je de lengte van een string opvragen:

```
print(len("appeltaart"))
```
Heeft als resultaat:
```
10
```

Als je de lengte weet, kun je natuurlijk ook uitrekenen waar de middelste letter zit...
Let op, bij een string van oneven lengte ronden we naar beneden af om de middelste letter te vinden.

Gebruik deze kennis om de woorden te printen waarbij de tweede helft van beide woorden verwisseld zijn.

Stap 4: Hoofdletters maken

We zullen eerst moeten checken of de eerste letter van het woord een hoofdletter is. Met het commando `ord()` kun je de ascii code van een letter opvragen.

```
print(ord("q"))
```
Heeft als resultaat:
```
113
```

Op de ascii tabel kun je zien dat alle kleine letters zitten op posities 97 (a) t/m 122 (z).

Door de eerste letter van het woord zo om te zetten naar zijn ascii code, kun je checken of dit een kleine letter is. Je kunt in dat geval de eerste letter vervangen door de bijbehorende hoofdletter door de gevonden ascii positie om te rekenen naar de positie van de bijbehorende hoofdletter.

Met het commando `chr()` kun je een ascii code weer omzetten in de bijbehorende letter:

```
print(chr(99))
```
Heet als resultaat:
```
c
```

Als de eerste letter geen kleine letter is, hoef je niets aan te passen en moet je script gewoon het originele woord afdrukken

6. Schrikkeljaar

Schrikkeljaren zijn uitgevonden, omdat een jaar eigenlijk ietsje langer is dan 365 dagen. Elke 4 jaar is het jaar daarom 366 dagen, zodat de kalender gelijk blijft lopen met de werkelijkheid.

Stiekem is het nog wat ingewikkelder dan dat. Op de lange termijn zou de kalender zo toch nog “scheef” gaan lopen, omdat de afwijking niet precies een kwart dag per jaar is. Daarom zijn de precieze regels rondom schrikkeljaren als volgt:

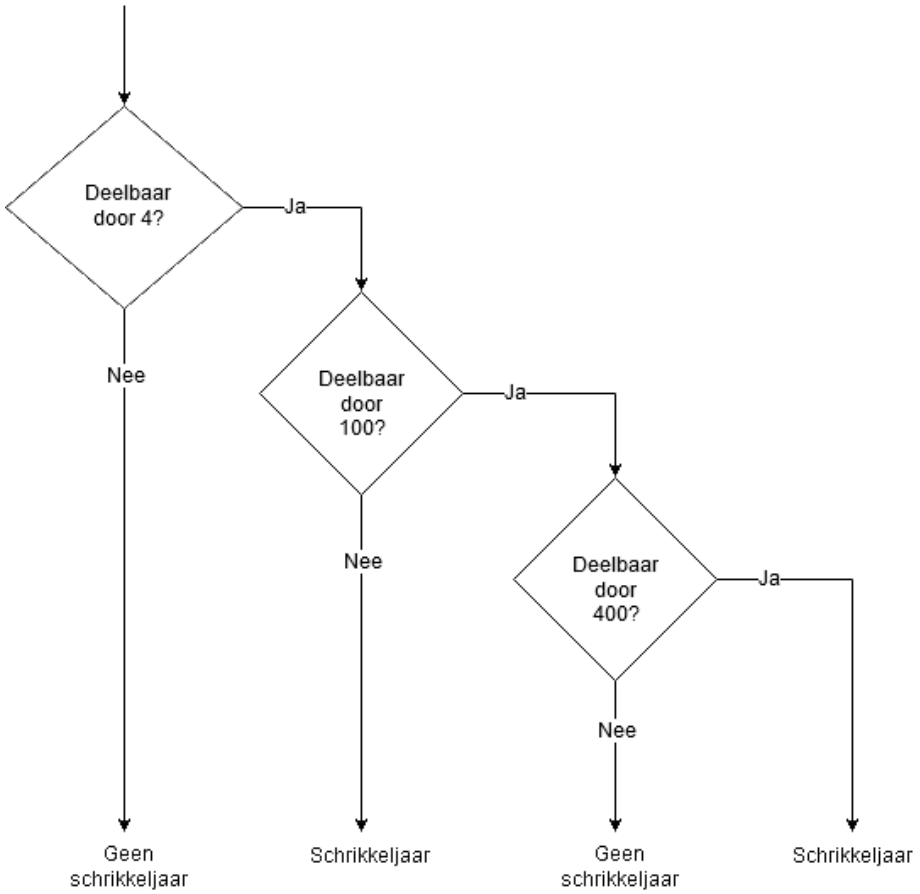
- Als het jaartal deelbaar is door 4 is het een schrikkeljaar, tenzij:
- Als het jaartal ook deelbaar is door 100, is het geen schrikkeljaar, tenzij:
- Als het jaartal ook deelbaar is door 400, is het toch wel een schrikkeljaar

Enkele voorbeelden:

- 2015 is geen schrikkeljaar (niet deelbaar door 4)
- 2016 is wel een schrikkeljaar (wel deelbaar door 4, maar niet door 100
- 2100 is geen schrikkeljaar (deelbaar door 4, maar ook deelbaar door 100. Niet deelbaar door 400)
- 2400 is wel een schrikkeljaar (deelbaar door 4, ook door 100, maar ook weer door 400)

Schrijf een Python script dat de gebruiker om een jaartal vraagt. Het script vertelt je vervolgens of het gegeven jaar een schrikkeljaar is of niet.

Je kunt voor dit soort problemen ook een flowchart/beslisboom maken als je dat handig vindt. Als je die hebt, is het makkelijker je programma correct te schrijven. Voor dit probleem zou dat er zo uit zien:



Elk ruitje zal hierbij overeenkomen met een if-blok in je python script.

7. Tafels

Schrijf een Pythonscript dat om een getal vraagt. Vervolgens drukt je script de tafel van dat getal op het scherm af. Bijvoorbeeld:

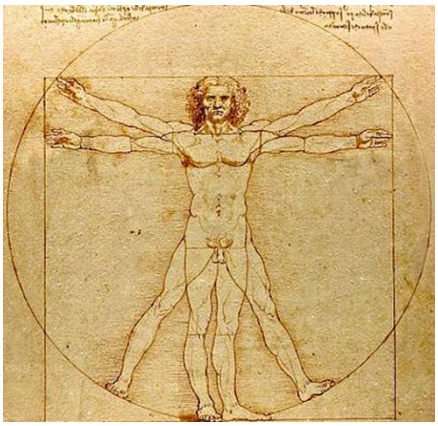
Welke tafel wil je maken?
8

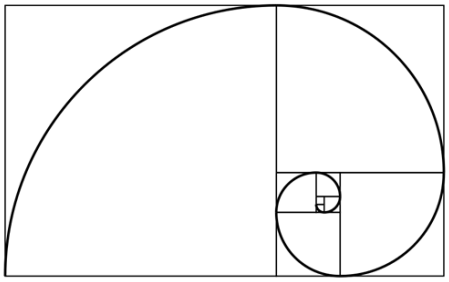
1 x 8 = 8
2 x 8 = 16
3 x 8 = 24
4 x 8 = 32
5 x 8 = 40
6 x 8 = 48
7 x 8 = 56
8 x 8 = 64
9 x 8 = 72
10 x 8 = 80

Extra: Breid je script uit dat je ook vraagt tot hoever de tafel moet doorgaan (i.p.v. standaard tot 10) en druk dan de tafel af

8. Fibonacci

Fibonacci is een beroemde wiskundige uit de 13e eeuw. Hij is het meest bekend om zijn zogenaamde “Fibonacci-reeks”. Deze reeks houdt verband met de zogenaamde “Gulden snede”. Die beschrijft een ideale verhouding tussen opeenvolgende afmetingen. Dit is een fenomeen dat zich in de natuur veel voordoet en ook in de kunstwereld veel wordt toegepast. (De beroemde Vitruviaanse Man, van Leonardo Da Vinci zit er vol mee bijvoorbeeld.) Meer info op Wikipedia





De Fibonacci-reeks is een oneindige reeks getallen. De reeks begint met twee keer een 1. Alle volgende getallen zijn steeds de som van de twee getallen ervoor. De reeks ziet er dus als volgt uit:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Schrijf een Python script dat de Fibonacci reeks kan uitschrijven. Het script vraagt de gebruiker om een getal om aan te geven hoeveel Fibonaccigetallen er moeten worden uitgerekend. Vervolgens drukt je script precies het gevraagde aantal stappen van de Fibonacci reeks af op het scherm.

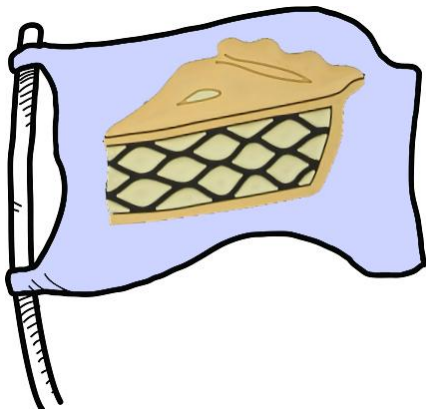
Het gulden getal

De verhouding tussen 2 opeenvolgende getallen in de gulden snede wordt aangeduid met de Griekse letter ϕ (phi) en is ongeveer 1,61803. Met de Fibonacci-reeks kun je het getal phi benaderen door steeds 2 opeenvolgende Fibonacci getallen door elkaar te delen. Hoe verder je in de reeks komt, des te nauwkeuriger is de benadering.

Pas je Fibonacci script aan dat hij niet alleen de reeks met getallen afdrukt (en druk ze nu onder elkaar af), maar steeds ook de verhouding van dat getal met het vorige getal (afgerond op 10 decimalen). De uitvoer zou er zo uit kunnen zien:

```
1
1
2 - phi: 2.0
3 - phi: 1.5
5 - phi: 1.6666666667
8 - phi: 1.6
13 - phi: 1.625
21 - phi: 1.6153846154
…
…
```

9. Versla de draak



In het mythische koninkrijk Ahpeltaertia houdt een gevaarlijke draak de bevolking in zijn greep. Gelukkig staan er helden op die proberen de draak te verslaan en de prinses (of prins natuurlijk, laten we wel gender-inclusief blijven) te bevrijden. Nu is de ene held de andere niet, dus het hangt nogal van de vaardigheden van de held af of hij/zij in staat is de draak te verslaan.

- Vraag de gebruiker om de held een naam te geven
- Gebruik de `random` library om de volgende eigenschappen te genereren voor de held.
    - Kracht: tussen 2 en 10
    - Geluk: tussen 0 en 5
    - Snelheid: tussen 0 en 20
    - Charisma: tussen 1 en 6
- Print een overzicht van de held met zijn naam en eigenschappen op het scherm
- Laat de held op enter drukken (met `input()`) om de draak te lijf te gaan
- Check daarna of de held in staat is om de draak te verslaan
- Als aan minstens een van de volgende voorwaarden is voldaan, wordt de draak verslagen. Anders wordt de held geroosterd door de draak:
    - Kracht is minstens 8 en de snelheid is minstens 10
    - Geluk is minstens 4
    - Kracht is onder de 10, snelheid is onder de 6, maar charisma is minstens 5
- Er is een uitzondering: Als geluk 1 of lager is, verliest de held altijd

Voorbeeld in- en uitvoer 1:

```
Het koninkrijk Ahpeltaertia wordt bedreigd door een gemene draak! Welke held staat
op om hem te verslaan?
Wat is uw naam, o held? Wt de Verschrikkelijke

Aanschouw de dappere held:

------------------
Wt de Verschrikkelijke
------
Kracht: 2
Geluk: 1
Snelheid: 3
Charisma: 2
------------------

Druk op enter als u klaar bent de draak te lijf te gaan machtige Wt de
Verschrikkelijke

Met een luide strijdkreet stormt Wt de Verschrikkelijke op de draak af!
*stofwolken en strijdgeluiden*

Als het stof neerdwarrelt is van de held slechts een smeulend hoopje as over.
O wanhoop! Is er dan geen held die de draak de baas kan?
```

Voorbeeld in- en uitvoer 2:

```
Wat is uw naam, o held? Wilhelm

Aanschouw de dappere held:

------------------
Wilhelm
------
Kracht: 8
Geluk: 4
Snelheid: 14
Charisma: 1
------------------

Druk op enter als u klaar bent de draak te lijf te gaan machtige Wilhelm

Met een luide strijdkreet stormt Wilhelm op de draak af!
*stofwolken en strijdgeluiden*

De stofwolk klaart op en daar staat onze held. De draak ligt in mootjes gehakt op
de grond!
Hoera! Het koninkrijk viert feest. Appeltaarten voor iedereen!
```

Meerdere helden achter elkaar

Maak met `while` een loop die net zo lang nieuwe helden genereert en op de draak loslaat totdat de draak verslagen is. De gebruiker moet steeds een nieuwe naam invullen voor elke nieuwe held.

Extra: Kleed het programma aan met meer tekst/verhaal, extra stats en voorwaarde en eventueel ascii-art.

Extra 2: Spiek vast vooruit naar Hoofdstuk 13: lijsten. Als je weet hoe je met lijsten kunt werken, kun je bijhouden welke helden allemaal gesneuveld zijn voordat de draak uiteindelijk werd verslagen. Gebruik deze kennis om op het einde af te drukken welke helden aan de zegevierende held voorafgingen als een soort In Memoriam.

10. Schrikkeljaar functie

Bij vraag 6 heb je een schrikkeljaarprogramma gemaakt. Dit is typisch een stukje code dat in een functie thuishoort. Immers als je er een functie van maakt zou je in een later, uitgebreider programma gebruik kunnen maken van de functie en hoef je je nooit meer druk te maken om schrikkeljaren.
Maak een nieuwe versie van de schrikkeljaar-bepaler uit vraag 4, maar nu in de vorm van een functie.

Noem de functie `is_schrikkeljaar()`. De functie moet een jaartal als argument nemen en de waarde `True` teruggeven (returnen) als het een schrikkeljaar is en de waarde `False` als het geen schrikkeljaar is.

Nu je er een functie van hebt gemaakt, kun je gemakkelijk de code herhaald uitvoeren.

Maak nu een for-loop, die loopt van jaartal 0 t/m jaartal 2000. Van elk jaar checkt hij met de schrikkeljaar-functie of het jaar een schrikkeljaar is of niet. Je drukt dit voor elk jaartal af op je scherm. Je uitvoer ziet er dan dus als volgt uit:

```
Het jaar 0 is geen schrikkeljaar
Het jaar 1 is geen schrikkeljaar
Het jaar 2 is geen schrikkeljaar
Het jaar 3 is geen schrikkeljaar
Het jaar 4 is wel een schrikkeljaar
Het jaar 5 is geen schrikkeljaar
...
...
```

11. Stringfuncties combineren

Maak een nieuw script met daarin 3 functies (`defs`)

```
def omdraaien(tekst)
def wisselen(tekst)
def knippen(tekst)
```

Elk van deze functies heeft een string "tekst" als argument en returnt ook een string als uitkomst
De functies gedragen zich als volgt

- `omdraaien` draait de letters van de string om en returnt deze omgedraaide versie
- `wisselen` wisselt de eerste en laatste letter van de string om en returnt het resultaat
- `knippen`, gooit de laatste 2 letters van de invoer weg en returnt het resultaat

Nadat je deze 3 functies hebt gecodeerd, vraag ja in het hoofdprogramma de gebruiker om een string als invoer met behulp van `input()`
Print op het scherm achtereenvolgens de uitvoer van deze 3 functies

Voorbeelden van uitvoer:

```
omdraaien("appeltaart")      -->   "traatleppa"
wisselen("appeltaart")       -->   "tppeltaara"
knippen("appeltaart")        -->   "appeltaa"

omdraaien("informatica")     -->   "acitamrofni"
wisselen("informatica")      -->   "anformatici"
knippen("informatica")       -->   "informati"
```

Tenslotte print je ook het resultaat van het "aaneen rijgen" van deze functies in verschillende volgorden.
1. omdraaien --> 2. wisselen --> 3. knippen
1. knippen --> 2. wisselen --> 3. omdraaien
1. wisselen --> 2. knippen --> 3. omdraaien

Voorbeelden (merk op dat de functie die als eerste moet worden gedaan, binnenin de haakjes staat):

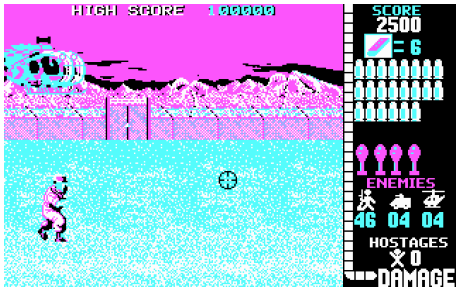
```
knippen(wisselen(omdraaien("appeltaart")))   --> "araatlep"
omdraaien(wisselen(knippen("appeltaart")))   --> "aatleppa"
omdraaien(knippen(wisselen ("appeltaart")))  --> " aatleppt"

knippen(wisselen(omdraaien("informatica")))  --> " icitamrof"
omdraaien(wisselen(knippen("informatica")))  --> " itamrofni"
omdraaien(knippen(wisselen ("informatica"))) --> " itamrofna"
```

12. Een python op je bord

Werken met de console heeft als nadeel dat de meeste uitvoer er een beetje saai uit ziet. (Na de toetsweek gaan we met een PO aan de slag waarin je wel echte graphics kunt gebruiken, dus houd vol!).

Computers zijn pas een jaar of 40 krachtig genoeg om graphics te tekenen. Lange tijd was dat ook nog niet heel best. CGA graphics (begin jaren '80) konden blauw, roze, wit en zwart weergeven. Met prachtige *kuch* graphics als gevolg:



Operation Wolf in CGA (er was ook een VGA versie met meer kleuren, maar de meeste mensen hadden geen computer die dat aan kon…)

In de tijd voor graphics werd er met ASCII-art gewerkt: plaatjes die zijn opgebouwd met alleen maar ASCII tekens. (Dit is een hele kunstvorm geworden overigens en er bestaan ook ASCII-art generators die een plaatje omzetten in ASCII art).

ASCII art bord

In deze opgave ga je een functie schrijven `bord()`. Deze neemt een string als parameter en print met ASCII art een uithangbord met de gegeven tekst erop:

Invoer: Appeltaart

Uitvoer:

```
*-------------*
*             *
* Appeltaart *
*             *
*-------------*
       ||
       ||
```

Invoer: Frikandel

Uitvoer:

```
*-----------*
*           *
* Frikandel *
*           *
*-----------*
      ||
      ||
```

De lengte van het woord is hier heel belangrijk. Denk goed na hoe je de lengte van het woord kunt gebruiken om te bepalen hoeveel streepjes, spaties e.d. je op elke regel moet afdrukken. Let op: als het woord van oneven lengte is (zoals "Frikandel" hierboven), past het paaltje er niet precies onder, maar staat het 1 teken rechts van het midden.

Verdeel het probleem in stukjes. Er zijn een aantal verschillende soorten regels in de output. Sommigen ervan komen 2x voor:

- De bovenrand en onderrand van het bordje zijn gelijk (Hoeveel streepjes heb je nodig?)
- De regel onder en boven het woord zijn gelijk (hoeveel spaties heb je nodig?)
- De regel met het woord erop
- De twee regels met het paaltje (hoeveel spaties staan er voor?)

Marge

Het bord ziet er al sjiek uit, maar het zou tof zijn als je wat opties hebt om het uiterlijk te kiezen. We beginnen met de marge. Naast de string parameter die bepaalt welk woord er op het bord moet verschijnen, voegen we een tweede parameter toe: een integer die aangeeft hoe groot de marge moet zijn. Pas je functie aan zodat je bord de opgegeven marge overneemt. Zie de volgende voorbeelden:

Woord: Appel
Marge: 0

Uitvoer:

```
*-----*
*     *
*Appel*
*     *
*-----*
   ||
   ||
```

Woord: Peer
Marge: 2

Uitvoer:

```
*---------*
*         *
*  Peer   *
*         *
*---------*
     ||
     ||
```

Woord: Citroen
Marge: 5

Uitvoer:

```
*-----------------*
*                 *
*     Citroen     *
*                 *
*-----------------*
          ||
          ||
```

Verschillende versies van het bord

Voeg nog een integer parameter aan je functie toe die het mogelijk maakt te kiezen uit verschillende versies van je bord:

Woord: Peer
Marge: 2
Versie: 1

Uitvoer:

```
*---------*
*         *
*  Peer   *
*         *
*---------*
     ||
     ||
```

Woord: Peer
Marge: 2
Versie: 2

Uitvoer:

```
|\
| \
|  \
|   \
*---------*
*         *
*  Peer   *
*         *
*---------*
```

Woord: Peer
Marge: 2
Versie: 3

Uitvoer:

```
    /\
   /  \
*---------*
*         *
*  Peer   *
*         *
*---------*
```

Meerdere regels

Het zou extra sjiek zijn als je ook meerdere regels op je bord kon printen (en dat het bord netjes langer wordt om alle regels te laten passen). Hiervoor heb je echter kennis van Python “lijsten” nodig. Dat is het volgende hoofdstuk uit de cursus, dus hier komen we in een toekomstige opgave op terug.

Optionele uitbreidingen:

Je kunt deze functie zou gek en uitgebreid maken als je maar wilt natuurlijk. Misschien kun je extra parameters aan je functie toevoegen die nog meer opties mogelijk maken. Bijvoorbeeld: de keuze tussen enkele en dubbele lijntjes, extra versieringen rondom het bord, etc.

13. Staafdiagram

Even een basic oefening met functies, lists en de for-loop. Schrijf een functie `staafdiagram()` die een lijst met gehele getallen als argument neemt en een “staafdiagram” van sterretjes afdrukt. Bijvoorbeeld:

`staafdiagram([4, 8, 5])` zou de volgende uitvoer moeten produceren:

```
****
********
*****
```

Hint: je kunt strings “vermenigvuldigen” weet je nog? Het commando "hoi" * 3 zal als uitvoer hebben: hoihoihoi

14. Yahtzee

Bij het dobbelspelletje yahtzee wordt er gegooid met 5 dobbelstenen. Sommige combinaties leveren punten op. We gaan een yahtzee worp simuleren en kijken of er scorende combinaties gegooid zijn.

Stap 1:
Om 5 dobbelstenen te "gooien", genereert je script 5 willekeurige getallen tussen de 1 en de 6. Sla deze getallen op in een list `worp`. Druk de worp ook af op het scherm.

Stap 2:
Schrijf een functie `check_yahtzee(worp)`. Deze functie neemt als argument een list met 5 getallen (je worp) en returnt True als er "Yahtzee" is gegooid en False als dat niet zo is. "Yahtzee" wil zeggen dat je 5 dezelfde getallen hebt gegooid. Dus bijvoorbeeld:

- `check_yahtzee([5,5,5,5,5])` levert `True` op
- `check_yahtzee([3,5,5,5,4])` levert `False` op.

Stap 3:
Schrijf een functie `straat(worp)`. Deze functie neemt als argument een list met 5 getallen (je worp) en returnt True als er een straat is gegooid en False als dat niet zo is. Een straat wil zeggen dat er 4 opeenvolgende getallen in de worp voorkomen. Bv 1 t/m 4 of 3 t/m 6. De volgorde van de dobbelstenen maakt hierbij niet uit. Dus bijvoorbeeld:

- `straat([5,5,5,5,5])` levert `False` op
- `straat([3,5,2,5,4])` levert `True` op (2 t/m 5 zitten in de worp)
- `straat([5,4,2,1,3])` levert `True` op (1 t/m 4 en 2 t/m 5 zitten in de worp)

Stap 4:
Schrijf een functie `full_house(worp)`. Deze functie neemt als argument een list met 5 getallen (je worp) en returnt True als er "Full house" is gegooid en False als dat niet zo is. "Full house" wil zeggen dat je een combinatie van 2 en 3 dezelfde getallen hebt gegooid. Dus bijvoorbeeld:

- `full_house([5,5,5,5,5])` levert `False` op
- `full_house([3,5,5,5,4])` levert `False` op
- `full_house([3,5,5,5,3])` levert `True` op
- `full_house([1,2,1,2,2])` levert `True` op

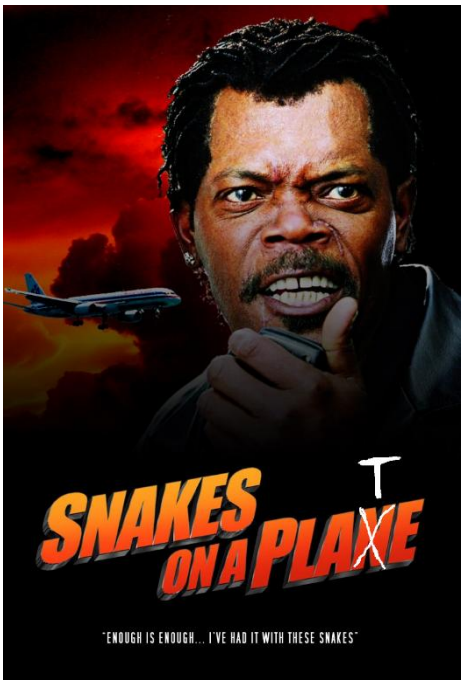
Stap 5:
Laat de functies uit stap 2 en 3 los op je worp uit stap 1 en print je resultaat. Voorbeeld uitvoer:

Worp: [4,2,3,1,4] Geen yahtzee Straat! Geen full house	Worp: [3,3,4,3,4] Geen yahtzee Geen straat Full house!	Worp: [2,2,2,2,2] Yahtzee! Geen straat Geen full house	Worp: [4,1,5,3,6] Geen yahtzee Straat! Geen full house	Worp: [4,4,5,3,4] Geen yahtzee Geen straat Geen full house

NB: als je moeite hebt met functies, mag je ze eventueel ook weglaten en je programma zonder functies schrijven. Is wel minder sjiek en uitbreidbaar daardoor natuurlijk…

NB 2: Leuk project voor de echte doorzetter: maak een volledig werkende yahtzee met alle mogelijke worpen, het bijhouden van de echte spelscores, meerdere spelers, etc, etc.

15. Nog meer pythons op je bord



In opgave 12 hebben we een functie gemaakt die een ASCII art-bordje afdrukt met een gegeven tekst erop. Nu je weet wat lijsten zijn, kunnen we deze functie uitbreiden, zodat ook meerdere regels tekst op 1 bord kunnen worden gezet.

"Enters" in een string

Een string in python kan naast "gewone" karakters ook speciale karakters bevatten. Een daarvan is het "newline" teken. Dit is een teken dat aangeeft dat er naar een nieuwe regel moet worden gesprongen. Dit speciale teken wordt geschreven als \n

Test het volgende maar eens:

```
print("Appeltaart\nis lekker")
```

Je ziet dat de tekst nu over 2 regels heen verspreid wordt en dat de \n dus als een soort "enter" fungeert.

Test bovenstaande string eens op je functie uit opgave 12. Dit zal geen mooi bordje opleveren, de code uit opgave 12 houdt geen rekening met invoer die over meerder regels verspreid is en daarom zal de tweede regel "van het bord afvallen". Dat gaan we fixen!

De Python split() methode

Python heeft een heleboel handige functies ingebouwd. Een daarvan is de methode `split()` die je op een string kunt loslaten. Deze splitst een string in een lijst met meerdere substrings. Je kunt zelf aangeven op welk teken de splitsing moet plaatsvinden. Een voorbeeld:

```
boodschappen = "appels,peren,druiven"
boodschappenlijst = boodschappen.split(",") #splits op komma's
print boodschappen #originele string
print boodschappenlijst #gesplitse lijst
print boodschappenlijst[2] #3e item uit de gesplitste lijst
```

Uitvoer:

```
"appels,peren,druiven"
["appels", "peren", "druiven"]
"druiven"
```

Deze functie kunnen we mooi gebruiken om ons bordje te laten werken met strings van meerdere regels:

1. We gebruiken `.split()` om de invoer parameter te splitsen in een lijst van afzonderlijke regels
2. We kijken welk van deze regels het langst is, om te bepalen hoe breed het bordje moet worden
3. We printen het begin en einde van het bordje net zoals eerst, maar we gebruiken een for-loop om alle regels tekst van het bordje te printen

Ad 1: Splitsen in afzonderlijke regels:

Bekijk het bovenstaande voorbeeld van `.split()` en herschrijf de `bord()` functie zodanig dat de invoer wordt gesplitst in een lijst met regels. Sla deze lijst Dat kan simpelweg met `.split("\n")`
Sla deze lijst op in een aparte variabele in je functie.

(Opmerking: als er invoer wordt gegeven zonder een \n erin, werkt alles nog steeds. De `.split()` functie levert dan een lijst op met maar 1 element erin, namelijk de hele invoerstring.)

Ad 2: De langste regel bepalen

Nu dankzij `.split()` alle afzonderlijke regels als apart element in een lijst zitten, kunnen we bepalen wat de langste zin is uit deze lijst. De langste zin is immers bepalend voor hoe breed het bordje moet worden.

Er is een slim algoritme om de langste string in een lijst te bepalen:

- Maak een variabele `langste` en zet deze op 0
- Met een for-loop loop je langs alle zinnen uit de lijst:
    - Kijk of de lengte van de huidige zin groter is dan `langste`
    - Als dat zo is, maak `langste` gelijk aan de lengte van de huidige zin

Als je alle zinnen bekeken hebt, bevat `langste` de lengte van de langste zin uit de invoer.

Ad 3: Printen van het bordje

Het printen van de boven- en onderkant van het bordje gaat hetzelfde als voorheen. Het printen van de tekstregels is nu wat ingewikkelder:

Loop met een for-loop langs alle zinnen uit de lijst:

- Bereken hoeveel spaties er nodig zijn voor de huidige regel (hiervoor moet je kijken wat het verschil in lengte is tussen de huidige regel en `langste`)
- Print de huidige regel, voorzien van het juiste aantal spaties op het scherm

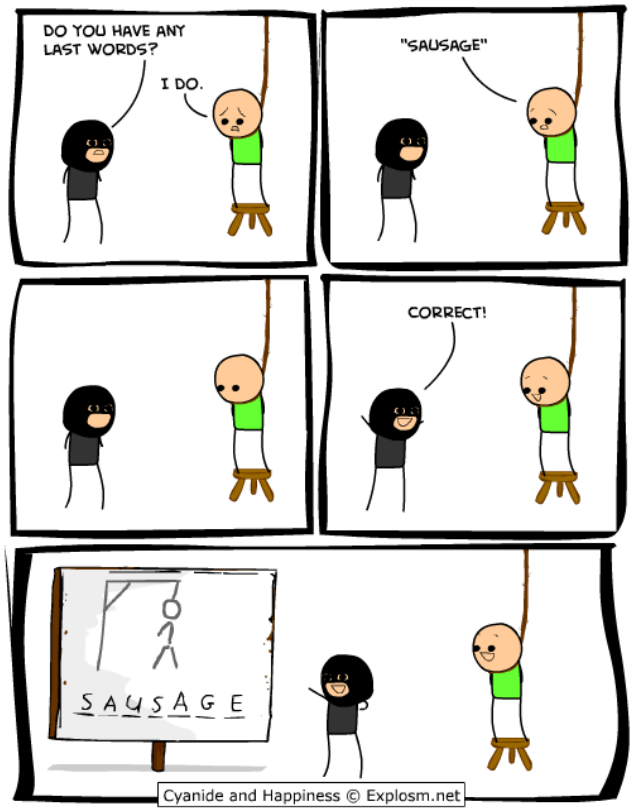
Invoer: Appeltaart\nis\nlekker
Marge: 2
Versie: 3

Uitvoer:

```
       /\
      /  \
*--------------*
*              *
*  Appeltaart  *
*      is      *
*    lekker    *
*              *
*--------------*
```

Tip: Deze functie is erg geschikt om het drakenprogramma van opgave 9 op te leuken… 😉

Vanaf hier: Bonusopdrachten (extra uitdaging, dit is geen toetsstof)



16. Bonusopdracht: Galgje

Nu je de meester bent van Python lists, kunnen we een spelletjes-klassieker maken: galgje

Woordenlijst

Om te beginnen hebben we een woordenlijst nodig. Dit is de lijst met woorden waaruit aan het begin van een nieuw spel een willekeurig woord gekozen wordt en door de speler moet worden geraden.

Deze lijst mag zo lang zijn als je wilt. Hoe meer woorden, hoe meer variatie natuurlijk. Maak een variabele `woordenlijst` en vul deze met een stelletje strings. Bv:

```
woordenlijst = ["appeltaart",
"worstenbroodjesverkoper",
"informaticamonster"]
```

Galg afdrukken

Voor het afdrukken van de galg, gaan we een functie schrijven. Dit is handig, want dan kunnen we op elk moment de juiste galg op het scherm afdrukken dor deze functie aan te roepen:

```
def print_galg(fase)
```

Bij de aanroep print_galg(3), zou hij bijvoorbeeld dit moeten printen (de galg voor 3 fouten):

```
  _______
 |/      |
 |
 |
 |
 |
 |
_|___
```

Met een groep if-elif-else commando's print deze functie de juiste versie van de galg af, afhankelijk van de waarde van fase (1 is 1 fout, 2 is 2 fouten, etc.). Bedenk zelf in hoeveel stappen je je galg wilt verdelen (dus hoeveel fouten de speler mag maken voor hij verloren heeft). Je kunt onderstaande ASCII art gebruiken of zelf iets "tekenen" natuurlijk. Test je functie goed voor je verder gaat.

```
  _______
 |/      |
 |      (_)
 |      \|/
 |       |
 |      / \
 |
_|___
```

Woord weergeven

Een belangrijk onderdeel van galgje is dat het woord wordt geprint voor zover dat al geraden is, zodat de speler weet hoe lang het woord is en welke letters hij al heeft:

```
******aa**
*pp***aa**
*pp**taa*t
```
etc.

Hiervoor schrijven we ook een functie. Deze functie heeft 2 parameters nodig: welk woord het is en een lijstje met letters die al geraden zijn:

```
def print_woord(woord, geraden_letters)
```

Bv. bij de aanroep van `print_woord("appeltaart", ["a","s","e","t","n"])`
zou het volgende geprint moeten worden:

```
a**e*taa*t
```

Om dit voor elkaar te krijgen doe je het volgende:

- Maak een lege string `uitvoer`
- Maak een for-loop, die langs alle letters van `woord` loopt:
    - Check of de huidige letter voorkomt in de lijst `geraden_letters` (gebruik hiervoor het `in` commando uit hoofdstuk 14)
    - Zo ja: voeg de huidige letter toe aan `uitvoer`
    - Zo nee: voeg een * toe aan `uitvoer`
- Als je klaar bent, print je `uitvoer` naar het scherm

Test je functie goed voor je verder gaat.

Spelverloop

De belangrijkste functies zijn nu gebouwd. Tijd om de hoofdcode van het spel zelf te gaan maken. Het globale spelverloop is als volgt:

- Zet een variabele `foutenteller` op 0
- Zet een variabele `geraden_letters` op de lege lijst (hierin gaan we de geraden letters bijhouden)
- Kies een random woord uit de woordenlijst en sla deze op in een variabele `woord`
- Maak een variabele `woord_geraden` en zet deze op `False`
- Zolang de speler nog niet dood is en het woord nog niet geraden is (gebruik een while):
    - Print de huidige versie van het woord met behulp van de print_woord() functie.
    - Vraag de speler om een letter in te voeren
    - Voeg deze letter toe aan de lijst met geraden letters
    - Check of de letter in het woord voorkomt, zo nee: verhoog de foutenteller met 1
    - Print de juiste versie van de galg (mbv de functie en de foutenteller)
    - Check of alle letters van het woord geraden zijn, zo ja ze `woord_geraden` op `True`
        - Hiervoor zet je `woord_geraden` eerst op True
        - Dan loop je alle letters van `woord` langs
        - Als je een letter vindt die niet in `geraden_letters` zit, zet je `woord_geraden` weer op `False`
        - Als de loop dan klaar is, zal `woord_geraden False` zijn als er nog te raden letters zijn en anders `True`
- Als de while loop eindigt, kunnen er 2 dingen aan de hand zijn:
    - De speler heeft gewonnen, druk dan een felicitatie af

- De speler heeft verloren. Druk een melding af en vertel wat het woord had moeten zijn.

Extra:

- Zorg dat het niet uitmaakt of een speler hoofdletters of kleine letters invoert. Gebruik de .lower() functie om alle invoer naar kleine letters om te zetten
- De speler moet eigenlijk ook gestraft worden voor het raden van een letter die hij al eerder geraden heeft. Het is dan wel sjiek om tussendoor steeds het lijstje met al geraden letters af te drukken op het scherm.

17. Bonusopdracht: Zeeslag met 2D lists

Lijsten bieden nog meer mogelijkheden als je ze gaat combineren met… nog meer lijsten. Je krijgt dan 2-dimensionale lijsten. Hier kun je allerlei handige dingen mee, zoals het opslaan van de toestand van het veld in verschillende 2D spelletjes (tetris, snake, 4-op-een-rij, etc, etc).

In deze opgave ga je een eenvoudige versie van het spelletje Zeeslag maken om eens met 2D lijsten te stoeien.

Wt heeft een opzet voor je gemaakt met een deel van de code, voorzien van uitleg. Neem de code over en maak hem af. Je vind de code hier:

https://repl.it/@wt2/Opg15zeeslag#main.py

In de basis opzet van Wt is er al een 2D lijst gemaakt en helemaal gevuld met het symbool . (een . in een vakje, betekent dat daar nog niet geschoten is)
Het spel kiest een random plek voor het te zinken schip en de speler mag steeds schieten op een coördinaat. Aan jou om te zorgen dat deze input verwerkt wordt en er gekeken wordt wat voor effect het schot had. Hierbij werk je steeds de toestand van het veld bij.

Uitbreidingen:

Je kunt hier helemaal loos gaan op uitbreidingen als je wil. Denk aan:

- Detecteren wanneer de speler bijna raak schiet om hem een hint te geven dat hij in de buurt zit
- Het vergroten van het veld
- Meerdere schepen
- Grotere schepen (denk eraan dat je dan bij het random plaatsen van de schepen wel rekening houdt dat ze niet over de rand van het scherm gaan)
- Een superwapen dat de speler 1x mag gebruiken om 3x3 vakjes tegelijk te raken
- …

18. Bonusopdracht: Roulette (met functies)

Bij roulette zijn er verschillende manieren van geld inzetten. Onder andere:

- Inzetten op een specifiek getal
- Inzetten op een specifieke kleur
- Inzetten op even of oneven

We geen een aantal functies schrijven om zo een eenvoudig roulettespel te maken.

Het roulettewiel

We moeten natuurlijk aan het roulettewiel kunnen draaien. We hebben vooralsnog geen graphics tot onze beschikking, maar we kunnen de boel wel een beetje opleuken door een heleboel getallen onder elkaar te printen. Eerst snel, dan steeds langzamer om zo een draaiend roulettewiel te simuleren dat uiteindelijk op 1 getal uitkomt.

Schrijf een functie `draai_wiel()` die dit voor elkaar krijgt. De functie moet 30 getallen op het scherm printen, eerst heel snel en elk volgend getal een stukje. Het uiteindelijke getal waar het wiel op "blijft staan" moet als return-waarde door de functie worden terug gegeven. Roulettegetallen zitten tussen de 0 en de 36. JE gebruikt natuurlijk `random.randint()` om de getallen te genereren.

Om de vertraging te maken, kun je `time.sleep()` gebruiken. Zie het voorbeeld

```
import time
print("voor de pauze")
time.sleep(0.5)
print("na de pauze")
```

importeer bovenaan je code de `time` library. Nu kun je met `time.sleep(x)` een pauze van x seconden inbouwen. Het bovenstaande voorbeeld print beide zinnen, maar wacht een halve seconde tussen de 2 prints.

Pluis nu zelf uit hoe je met een loop de 30 getallen kunt uitprinten waarbij je steeds langere sleep lengtes gebruikt. Vergeet niet dat het laatste getal als return waarde wordt teruggegeven door de functie (die hebben we zo meteen nog nodig namelijk

Inzetten op even en oneven

Schrijf een functie `gok_even_oneven(inzet, gok)`. Deze functie heeft 2 argumenten:

- Inzet: hoeveel geld zet je in?
- Gok: 0 is inzetten op even, 1 is inzetten op oneven

Deze functie roept op zijn beurt de `draai_wiel()` functie aan om aan het wiel te draaien en te weten op welk getal het roulettewiel is gevallen.

Check of het gedraaide getal overeenkomt met de gok van de gebruiker. Als dat zo is, krijgt de speler 1,8 keer zijn inzet uitbetaald. Als het niet zo is, krijgt de speler niets.

In beide gevallen print je wat tekst op het scherm om de speler de uitkomst te vertellen. En in beide gevallen is de return-waarde van `gok_even_oneven(inzet, gok)` gelijk aan het uitgekeerde bedrag (dus dat is óf 0 óf 1,8 keer de inzet)

Het hoofdspel

We gaan nu wat algemene code schrijven om het spel te spelen. Deze code maakt gebruik van de eerder geschreven functies

- Print een algemene introductie tekst
- Maak een variabele aan waarin je opslaat hoeveel geld de speler nog heeft
- Maak een loop, waarin je steeds vraagt of de speler wil stoppen of wil doorgaan
- Als de speler wil stoppen, print je zijn eindscore (geld) op het scherm
- Als hij wil doorgaan. Vraag of hij even of oneven wil gokken en wat zijn inzet is.
- Roep met deze invoer de `gok_even_oneven()` functie aan.
- Update het geld van de speler op basis van de uitkomst

Uitbreiden

De speler kan nu alleen maar gokken op even en oneven. Voeg zelf nieuwe functies toe om te gokken op een los getal, of op rood/zwart, of op 1tm/18 of 19tm/36, of …

Bij alle functies maak je natuurlijk gebruik van de `draai_wiel()` functie om aan het wil te draaien. Geef in het hoofdprogramma de gebruiker de keuze welke soort gok hij wil doen.