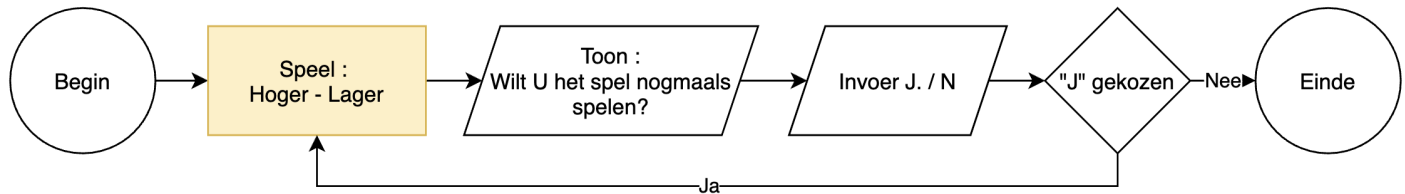


3 Nogmaals spelen

Hieronder staat het stroomdiagram om het spel “Hoger-Lager” nogmaals te spelen. We zien dat het gehele spel dat we in het vorige deel hebben gemaakt hier als een proces staat.



Hoe kunnen we dit nu het mooiste oplossen. We kunnen om de gehele code weer een lus maken. Op die manier wordt de code steeds overzichtelijker. Onoverzichtelijke code leidt tot fouten, fouten leidt tot frustratie en frustratie leidt naar the dark side.

We wille eigenlijk het volgende programma hebben als eindresultaat.

```
24  nogmaals_spelen = True
25
26  while nogmaals_spelen:
27      hogerLager()
28      antwoord = input("Wilt u nogmaals spelen (j/n) ? :")
29      nogmaals_spelen = antwoord == 'j' or antwoord == 'J'
30
31
```

Figuur 2: Code Nogmaals Spelen

Er zijn twee stukjes van de code dat we nu nog niet hebben gehad. Bij het bekijken van het antwoord staat een or. Deze or is een logische operator. Een logische operator is om meerdere vergelijkingen op een logische manier bij elkaar te brengen. Op w3Schools is er het volgende over te vinden:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Figuur 3: Boolean operatoren

Door waarheid tabellen is te zien wat het resultaat is als je twee Booleans met elkaar verbindt. Op deze manier kan je op meerdere dingen kijken of de expressie True of False is. Als

A	And	B	is	Hele statement
True	And	True	is	True
True	And	False	is	False
False	And	True	is	False
False	And	False	is	False

Figuur 4: Waarheid tabel AND

A	Or	B	is	Hele statement
True	Or	True	is	True
True	Or	False	is	True
False	Or	True	is	True
False	Or	False	is	False

Figuur 5: Waarheid tabel OR

not A	is	Hele statement
not True	is	False
not False	is	True

Figuur 6: Waarheid tabel NOT

Net als met operaties met integers worden bij booleans eerst de expressies tussen haakjes geëvalueerd, daarna wordt de expressie van links naar rechts gelezen.

```
>>> True and True and False
False
```

3.1 Onze eigen functie

We hebben al functies in onze code gebruikt die door python beschikbaar zijn gesteld. In ons programma zien we nu de functie: `hogerLager()`.

We zien dat dit een functie is, want achter de naam staan een haakje openen en een sluiten.

We willen van ons “Hoger Lager” spel een functie maken zodat we de code van “Figuur 2: Code Nogmaals Spelen” kunnen uitvoeren. Een functie kunnen we heel simpel maken met het `def`-sleutelwoord. Een functie heeft een naam die moet voldoen aan dezelfde regels als een variabele. Argumenten zijn waarden die we mee kunnen geven aan de functie. Dit hebben we al gezien met de functie `print`, waarbij we een string meegaven aan de functie. Aan het einde van de definitie staat weer een dubbele punt `:` dat altijd betekent dat de volgende regel ingesprongen is en bij de functie hoort.

```
1 def functieNaam(argument1, argument2):
2     # hier staat de code die in de functie wordt uitgevoerd
3     None
4
5 functieNaam(1, 2)
```

Figuur 7: Voorbeeld van een functie

```
1 import random
2
3 naam = input("Hoi! Hoe heet je? ")
4
5 def hogerLager():
6     gekozen_getal = random.randint(1,20)
7     pogingen = 0
8     while pogingen < 5:
9         pogingen = pogingen + 1
10        geraden_getal = int(input("Raad eens. : "))
11
12        if geraden_getal == gekozen_getal:
13            print("Gefeliciteerd, getal geraden in " + str(pogingen) + " pogingen.")
14            break
15        else:
16            if geraden_getal > gekozen_getal:
17                print("Je hebt te hoog geraden.")
18            else:
19                print("Je hebt te laag geraden.")
20    else:
21        print("Verloren, niet binnen " + str(pogingen) + " pogingen.")
22
23
24 nogmaals_spelen = True
25
26 while nogmaals_spelen:
27     hogerLager()
28     antwoord = input("Wilt u nogmaals spelen (j/n) ? :")
29     nogmaals_spelen = antwoord == 'j' or antwoord == 'J'
30
31
```

Figuur 8: Hoger-Lager als een functie

We weten dat een variabele pas bekend is als deze is gedefinieerd, maar met functies werkt dit allemaal net weer wat ingewikkelder. Variabelen die binnen een functie gedefinieerd zijn, zijn ook alleen binnen die functie bekend. We kunnen de code dan niet aanpassen om bij het “Nogmaals spelen” een melding te geven van het aantal pogingen.

```
24 nogmaals_spelen = True
25
26 while nogmaals_spelen:
27     hogerLager()
28     print(str(pogingen) + " nodig gehad.")
29     antwoord = input("Wilt u nogmaals spelen (j/n) ? :")
30     nogmaals_spelen = antwoord == 'j' or antwoord == 'J'
31
```

3.2 Return statement

Als we dit doen krijgen we de foutmelding: **NameError: name 'pogingen' is not defined**.

We moeten informatie uit de functie naar buiten brengen. Natuurlijk zijn hier weer meerdere manieren voor, maar de beste manier is om dit te doe door bij het beëindigen van de functie een waarde terug te geven naar het aanroepende deel. Teruggeven in het Engels is return, dus we kunnen weer p W3Schools kijken of hier wat voor is.

```
def myfunction():
    return 3+3

print(myfunction())
```

Als een functie een return tegenkomt dan wordt gelijk uit de functie gestapt, en wordt de expressie die achter de return staat teruggegeven. Dit betekent dus dan code die na een return staat niet uitgevoerd wordt. Als er geen return in een functie staat wordt standaard de None waarde teruggegeven. Op deze manier kan je in het “Hoger-Lager” spel het aantal beurten teruggeven.

```
1 import random
2
3 naam = input("Hoi! Hoe heet je? ")
4
5 def hogerLager():
6     gekozen_getal = random.randint(1,20)
7     pogingen = 0
8     while pogingen < 5:
9         pogingen = pogingen + 1
10        geraden_getal = int(input("Raad eens. : "))
11
12        if geraden_getal == gekozen_getal:
13            print("Gefeliciteerd, getal geraden in " + str(pogingen) + " pogingen.")
14            break
15        else:
16            if geraden_getal > gekozen_getal:
17                print("Je hebt te hoog geraden.")
18            else:
19                print("Je hebt te laag geraden.")
20        else:
21            print("Verloren, niet binnen " + str(pogingen) + " pogingen.")
22        return pogingen
23
24 nogmaals_spelen = True
25
26 while nogmaals_spelen:
27     pogingen = hogerLager()
28     print(str(pogingen) + " beurten nodig gehad.")
29     antwoord = input("Wilt u nogmaals spelen (j/n) ? :")
30     nogmaals_spelen = antwoord == 'j' or antwoord == 'J'
```

De waarde die teruggegeven wordt kan dan weer in een variabele gestopt worden.

Door gebruik te maken van een functie zijn er opeens weer heel veel extra mogelijkheden. In “Figuur 7: Voorbeeld van een functie” zagen we dat er ook waarden via argumenten naar een functie gestuurd kunnen worden. We kunnen door die argumenten het spel weer verder sturen. Een idee is om de maximale waarde mee te geven als argument. Op regel 27 zetten we tussen de haakjes het maximale getal, en daarachter hoe vaak je mogen raden, gescheiden door een komma <,>.

```
27 pogingen = hogerLager(100, 8)
```

We moeten dan ook de definitie van de functie aanpassen. We zetten twee namen neer die we argumenten noemen en die werken als variabele binnen de functie.

```
5 def hogerLager(max_getal, max_beurten):
```

De gehele code ziet er nu als volgt uit. We hebben regel 6 en 8 aangepast en daar de parameters gebruikt.

LET OP: In een functie mag je alleen de variabelen gebruiken die je in de functie hebt aangemaakt en de parameters die je binnenkrijgt. Technisch is het mogelijk om variabelen te gebruiken van buiten de functie, en deze zelfs aan te passen. Maar dit wordt gezien als slecht programmeren.

```
1 import random
2
3 naam = input("Hoi! Hoe heet je? ")
4
5 def hogerLager(max_getal, max_beurten):
6     gekozen_getal = random.randint(1,max_getal)
7     pogingen = 0
8     while pogingen < max_beurten:
9         pogingen = pogingen + 1
10        geraden_getal = int(input("Raad eens. : "))
11
12        if geraden_getal == gekozen_getal:
13            print("Gefeliciteerd, getal geraden in " + str(pogingen) + " pogingen.")
14            break
15        else:
16            if geraden_getal > gekozen_getal:
17                print("Je hebt te hoog geraden.")
18            else:
19                print("Je hebt te laag geraden.")
20        else:
21            print("Verloren, niet binnen " + str(pogingen) + " pogingen.")
22        return pogingen
23
24 nogmaals_spelen = True
25
26 while nogmaals_spelen:
27     pogingen = hogerLager(100, 8)
28     print(str(pogingen) + " beurten nodig gehad.")
29     antwoord = input("Wilt u nogmaals spelen (j/n) ? :")
30     nogmaals_spelen = antwoord == 'j' or antwoord == 'J'
```

Je kunt in je code zoveel functies maken als je zelf wilt. Je kan ook functies binnen functies aanroepen. Het is dan wel belangrijk dat de functie eerst gedefinieerd is voordat je deze in een andere functie kan aanroepen.

In de programma's die we nu hebben gemaakt zijn we ervan uitgegaan dat de invoer van de gebruiker en het meegeven van parameters overeenkomen met het data type dat we verwachten.

Als we bij de invoer van het getal het woord “twaalf” hadden ingevoerd had het programma met een error gestopt. Dit zijn dingen die we op kunnen lossen, maar we nu nog niet doen.

3.3 Opdrachten

3.3.1 Blokken verplaatsen

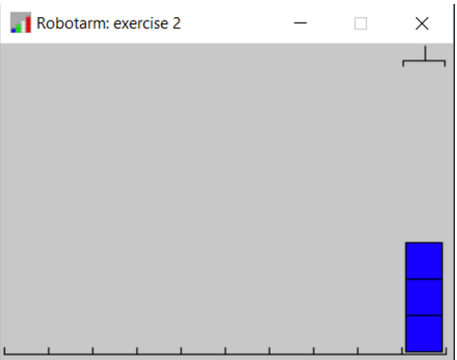
Stapel alle blokken op aan de rechterkant. Maak een functie aan waaraan je met een parameter meegeeft hoeveel stappen de robotarm naar rechts moet.

```
from RobotArm import RobotArm

robotArm = RobotArm('exercise 2')

# Jouw python instructies zet je vanaf hier:

# Na jouw code wachten tot het sluiten van de window:
robotArm.wait()
```



3.3.2 Aftellen

Maak een functie die twee parameters binnenkrijgt. Een met een getal waarvan afgeteld moet worden, de andere met de delay tussen de verschillende getallen in seconden. Dus als er (100, 3) meegegeven wordt aan de functie wordt van 100 tot 0 geteld, met 3 seconden tussen iedere tel. Toon iedere keer als het getal lager wordt deze waarde op het scherm getoond.

3.3.3 Schrikkeljaar

Maak een functie die als parameter een jaartal krijgt. Als dat jaartal een schrikkeljaar is wordt **True** teruggegeven. Indien geen schrikkeljaar dan **False**. We gaan eerst kijken wat een schrikkeljaar is.

Schrikkeljaar

Een **schrikkeljaar** is een **kalenderjaar** met 366 **dagen** in plaats van 365. Deze extra dag, een **schrikkel****dag**, wordt ingevoerd omdat een kalenderjaar van 365 dagen ongeveer 6 uur korter duurt dan het **tropisch jaar**. Om te voorkomen dat de seizoenen, die vast aan het tropisch jaar verbonden zijn, te veel in een jaar verschuiven, wordt om de circa 4 jaar een correctie toegepast.

De schrikkeldag valt in de **Gregoriaanse kalender** op **29 februari** en komt voor als het jaartal restloos deelbaar is door 4, maar niet door 100 – tenzij het jaartal restloos deelbaar door 400 is. 2020 is een schrikkeljaar. Daarvoor waren 2004, 2008, 2012 en 2016 (allemaal deelbaar door 4, maar niet door 100) schrikkeljaren. Ook 1600 (deelbaar door 400) was een schrikkeljaar. 1700, 1800 en 1900 waren dat niet (deelbaar door 100, maar niet door 400) en 2000 weer wel.

We zien het woord restloos deelbaar in de tekst. We weten al dat we dit kunnen doen met de modulo **<%>** operator.

In het jaar -48 is de Juliaanse kalender ingevoerd. Voor die datum was er helemaal geen schrikkeljaar. Daarna was er een schrikkeljaar om de 4 jaar.

Bij het invoeren van de Gregoriaanse kalender in 1582 kwam nog de regel dat de eeuwwisselingen geen schrikkeljaar was, behalve als de eeuw door 400 restloos deelbaar is.

3.3.4 Alleen integer invoer

Als we een integer getal willen, hebben we afgesproken dat we netjes dat in de input invoeren.

We hebben nu alleen de techniek om een functie te maken die kijkt of de input een integer getal is. Als dat niet het geval is dan wordt wederom gevraagd om de invoer opnieuw te doen.

Hieronder staat de code. Maak van deze code een functie.

```

1 correct = False
2 while not correct:
3     invoer = input("Geef een getal : ")
4     if getal.isnumeric():
5         invoer = int(invoer)
6         correct = True
7
8 print(invoer, type(invoer))
9

```

3.3.5 Abc-Formule (Als je deze formule niet ken, kan deze opdracht moeilijk zijn)

De Abc-formule wordt gebruikt om de snijpunten van een 2^e-graads vergelijking met de x-as te berekenen. Uit die vergelijking komen 0, 1 of 2 antwoorden.

Hieronder zie je een voorbeeld hoe je twee resultaten kan retourneren.

```

1 def tweeResultaten():
2     return 1, 2
3
4 a, b = tweeResultaten()
5 print(a,b)

```

Shell x

```

>>> %Run -c $EDITOR_CONTENT
1 2

```

We spreken voor deze functie af, dat we altijd 2 resultaten retourneren. Als er maar 1 antwoord is, is alleen de eerste ingevuld, en de tweede is NONE. Is er geen resultaat, dan wordt tweemaal None geretourneerd.

We bepalen eerst de discriminant: $D=b^2-4ac$

Als de discriminant kleiner is dan 0, is er geen oplossing (None, None)

Als de discriminant gelijk is aan 0, is er 1 oplossing (oplossing, None)

Is de discriminant groter dan 0, zijn er 2 oplossingen.

$$x = \frac{-b - \sqrt{D}}{2a} \vee x = \frac{-b + \sqrt{D}}{2a}$$

Maak de abc-formule in een functie. De waarden voor A, B en C worden als parameters meegegeven.

Aan het einde van de functie wordt teruggeven hoeveel antwoorden er zijn. De antwoorden worden met print in de functie getoond.

Voor deze Abc-formule heb je de wortel van een getal nodig.


```

1 import math
2 print( math.sqrt(2) )
3

```

Let op dat de functie niet mag crashen. Als de oplossing niet mogelijk is, dan toon je de tekst “Geen oplossing”.

3.3.6 Groot naar klein

Op de plek waar het blauwe blokje staat wil ik de rode stapel, dan de witte, groene en ten slotte de blauwe. Maak een functie die dit makkelijk kan maken. Denk aan een functie die alle blokjes in een kolom verplaatst naar een andere kolom. Je kan meegeven hoeveel blokjes er al zitten in een kolom. Je kan ook misgrijpen, en dan weet je dat je ze allemaal hebt verplaatst.

example.py

```

1 from RobotArm import RobotArm
2
3 robotArm = RobotArm('exercise 9')
4
5 # Jouw python instructie
6 robotArm.grab()
7 print(robotArm.scan())
8 # Na jouw code wachten t
9 robotArm.wait()

```

Robotarm: exercise 9

Shell

```

>>> %Run example.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.11)
Hello from the pygame community. <a href="https://www.pygame.org/contribute.html">https://www.pygame.org/contribute.html</a>
blue

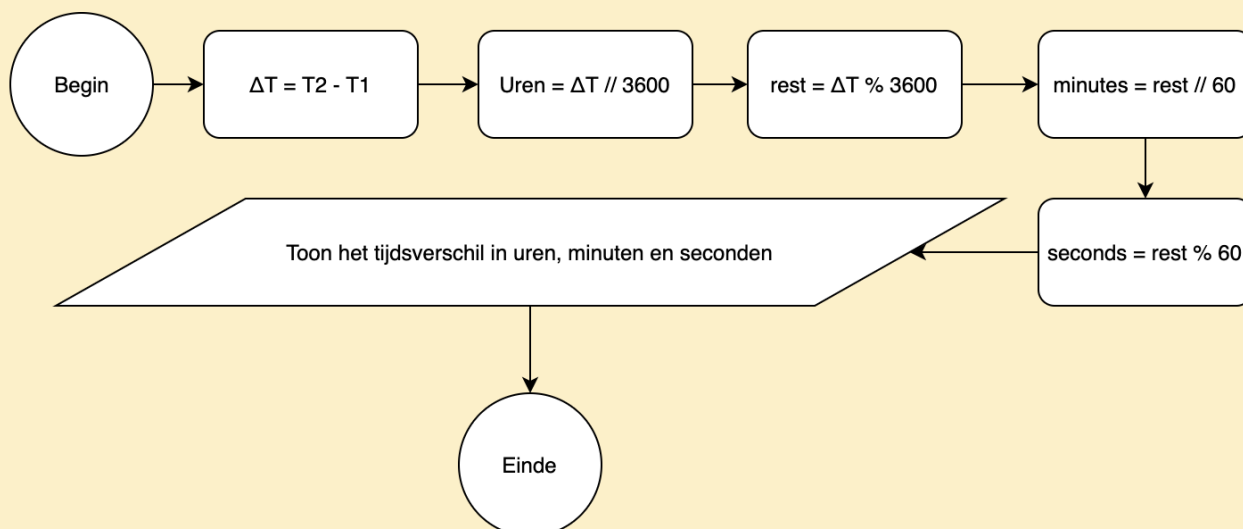
```


3.3.7 Tijdverschil

We hebben het volgende programma in Python. Maak het ontbrekende stukje code aan de hand van het stroomdiagram.

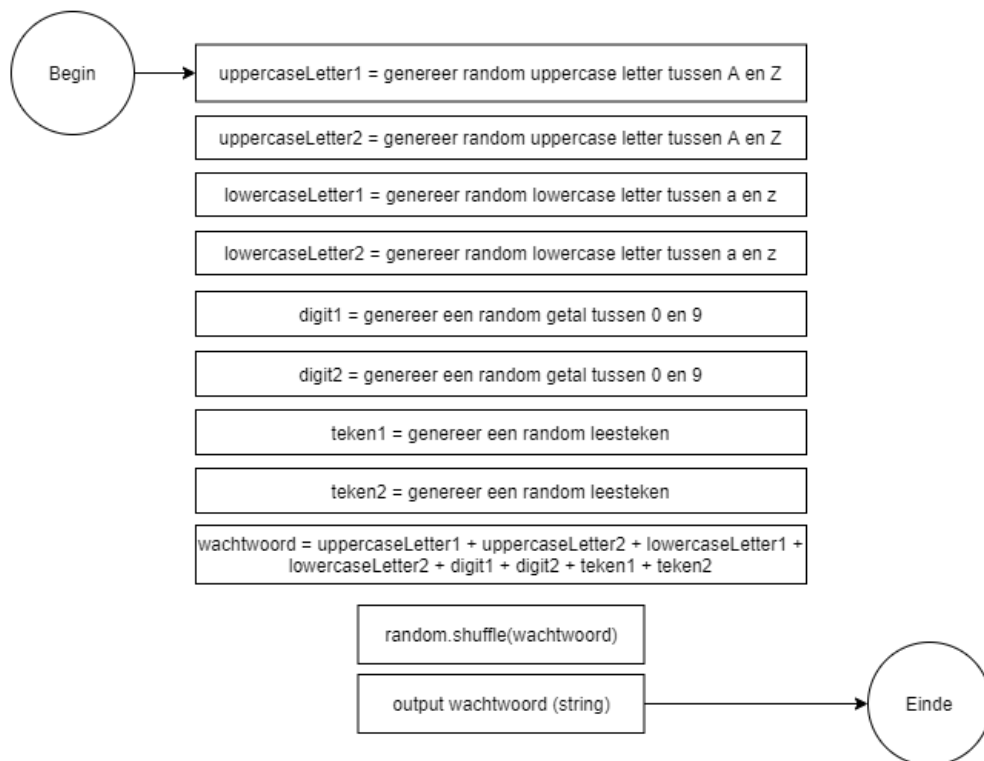
```
1  import time
2
3  def printTijdverschil(time1, time2):
4      verschil = int(time2 - time1)
5      print(time2, "-", time1, "=", verschil)
6
7
8
9
10     None
11
12 time1 = time.time()          # tijd bv. 1593625582.5557299
13 input("Wacht even en Druk op <return>")
14 time2 = time.time()          # tijd bv. 1593625587.5025039
15 printTijdverschil(time1, time2)
16
```

def printTijdverschil(T1, T2)



3.3.8 Wachtwoord generator

We hebben nu kennis opgedaan van strings, ASCII en random. Deze kennis kunnen we nu gebruiken om zelfstandig een programma te maken. Aan de hand van het onderstaande stroomdiagram gaan we een wachtwoord generator maken.



Uit de ASCII-tabel kan je zien dat de letter [s] het getal 115 heeft. Hierdoor kan je vanuit het getal 115 dan ook de letter [s] krijgen door `chr(115)`. De verschillende letters, tekens en cijfers worden bij elkaar gezet in een string. Deze string zetten we om in een lijst. Hoe dat moet zie je in onderstaand voorbeeld. De lijst kunnen we vervolgens schudden (`shuffle`) zodat alle elementen door elkaar zitten. De geschudde lijst zetten we met de `join` methode weer om naar een string.

```
1 import random
2
3 #zet een willekeurig getal tussen 65 en 90
4 #om naar een karakter (vanuit ASCII tabel)
5 print(chr(random.randint(65, 90)))
6
7 #zet een string om in een lijst
8 woordLijst = list("wachtwoord")
9 print(woordLijst)
10 random.shuffle(woordLijst)
11 geschud = "".join(woordLijst)
12 print(geschud)
```

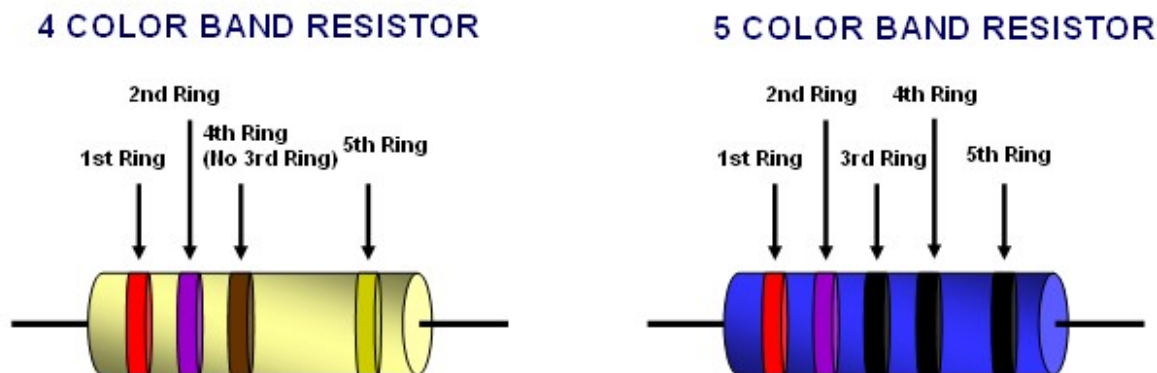
Shell ×

```
>>> %Run test.py
```

```
I
['w', 'a', 'c', 'h', 't', 'w', 'o', 'o', 'r', 'd']
twdchrawoo
```

3.3.9 Weerstandskleuren

In deze opdracht gaan we aan de hand van de kleuren op een weerstand teruggeven wat de weerstandswaarde is. Er zijn twee soorten weerstanden, de ene met 4 gekleurde ringen, en de andere met de 5 kleuren ring.



Color	1st Ring	2nd Ring	3rd Ring	4th Ring (Multiplier)	5th Ring (Tolerance)
Black	0	0	0	1	1 %
Brown	1	1	1	10	
Red	2	2	2	100	
Orange	3	3	3	1000	
Yellow	4	4	4	10000	
Green	5	5	5	100000	
Blue	6	6	6	1000000	
Violet	7	7	7	10000000	
Gray	8	8	8	100000000	
White	9	9	9	1000000000	
Gold	-	-	-	-	5 %
Silver	-	-	-	-	10 %
No Color					20%

Ten eerste ga je een stroomdiagram maken hoe je dit ga aanpakken. Het is goed om in te zien dat alle kleuren op de ringen dezelfde getallen vertegenwoordigen. Je kan dus 1 functie maken die een kleur als parameter krijgt (of afkorting van die kleur) en het getal 0 tot 9 teruggeeft. De functie voor de multiplier ring kan die functie ook aanroepen, en dan het resultaat berekenen door $10^{**\{waarde\}}$. Dit levert een resultaat van 10 tot de macht $\{waarde\}$. Het resultaat van de functies zijn integers.

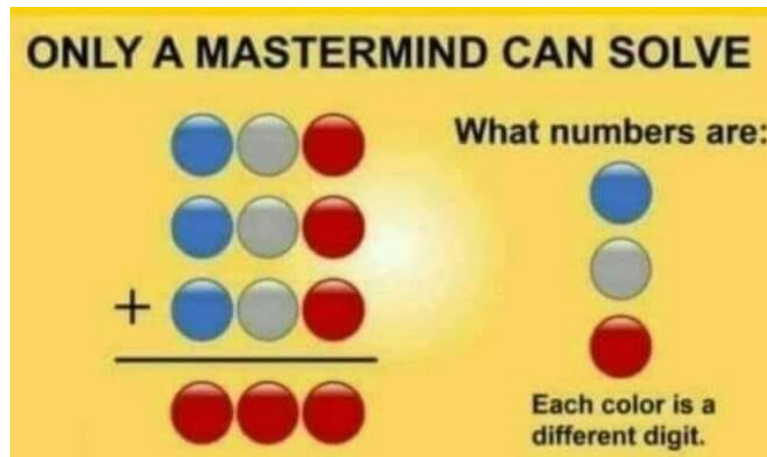
Vervolgens maag je twee functies, de ene voor 3 ringen, en dus ook met 3 parameters, en een functie voor 4 ringen, en dus ook met 4 parameters. Deze functies geeft de gehele weerstandswaarde als integer terug.

Er moet ook nog een functie *getTolerantie(...)* gemaakt worden die als parameter de namen *gold* of *silver* krijgt en dan de tolerantie als een integer teruggeeft. Als iets anders ingevoerd wordt (of niets) geeft deze functie de waarde 20 terug.

In het hoofdprogramma wordt gevraagd hoeveel ringen er zijn. Aan de hand van dat antwoord wordt een keuze gemaakt, of er 4 of 5 kleuren gevraagd worden. Deze kleuren worden dus aan de bovenstaande functies gegeven. Het resultaat wordt op het scherm getoond.

De code is pas af als er ook goede testen bij zijn, en goed commentaar.

3.3.10 Mastermind



Bovenstaand probleem kan je uit je hoofd oplossen, maar je kunt dit ook python laten oplossen. Ga alle getallen na van 0 t/m 999 en kijk welke cijfers er in de rode, witte en blauwe bolletjes moeten. Je begint dus met 3 variabelen met 0. Dat zou een oplossing zijn, maar er is nog 1 regel, de kleuren zijn een ander getal. Je gaat de achterste 1 omhoog....
Nog steeds niet toegestaan zo ga je door dat je 0, 1, 2 hebt. $012 + 012 + 012 = 222$?
Dat klopt niet, dus naar de volgende. Tot je het antwoord hebt gevonden.

3.3.11 Romeinse getallen

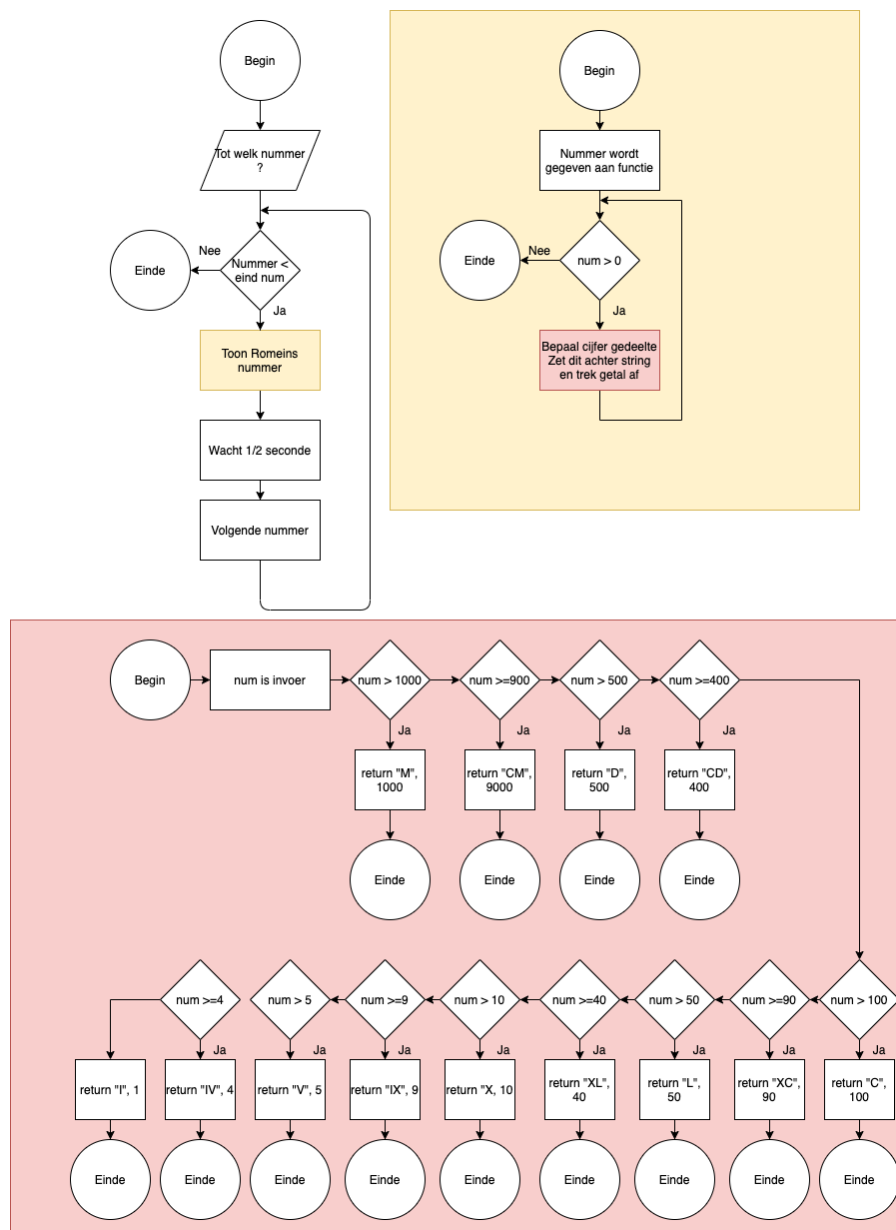
In een vorige opdracht hebben we een functie gemaakt die vraagt vanaf welk getal je terug wilt tellen, en met hoeveel seconden daar tussenin zit.

We willen deze functie weer gebruiken, maar nu willen we dit terugtellen doen door de romeinse getallen te tonen.

Als we beginnen dan maken we een functie aan die Arabische getallen (zoals onze getallen heten) omzetten naar Romeinse getallen. Deze functie is aan het begin leeg, alleen het getal dat in de functie binnen komt wordt weer teruggegeven.

Zo maken we een lus die de getallen onder elkaar neerzet. Als dit werkt gaan we verder met het invullen van de functie.

Als we op de site: <https://www.romeinsecijfer.nl/lijs.html> kijken zien we dat Romeinse getallen eigenlijk een 5-talig stelsel is. We volgen het stroomdiagram, en maken een lus die steeds wat van het nummer aftrekt en een gedeelte van de string erbij zet. Er is geen symbool voor het getal 0 (dat pas door de Arabieren in de Middeleeuwen werd bedacht).



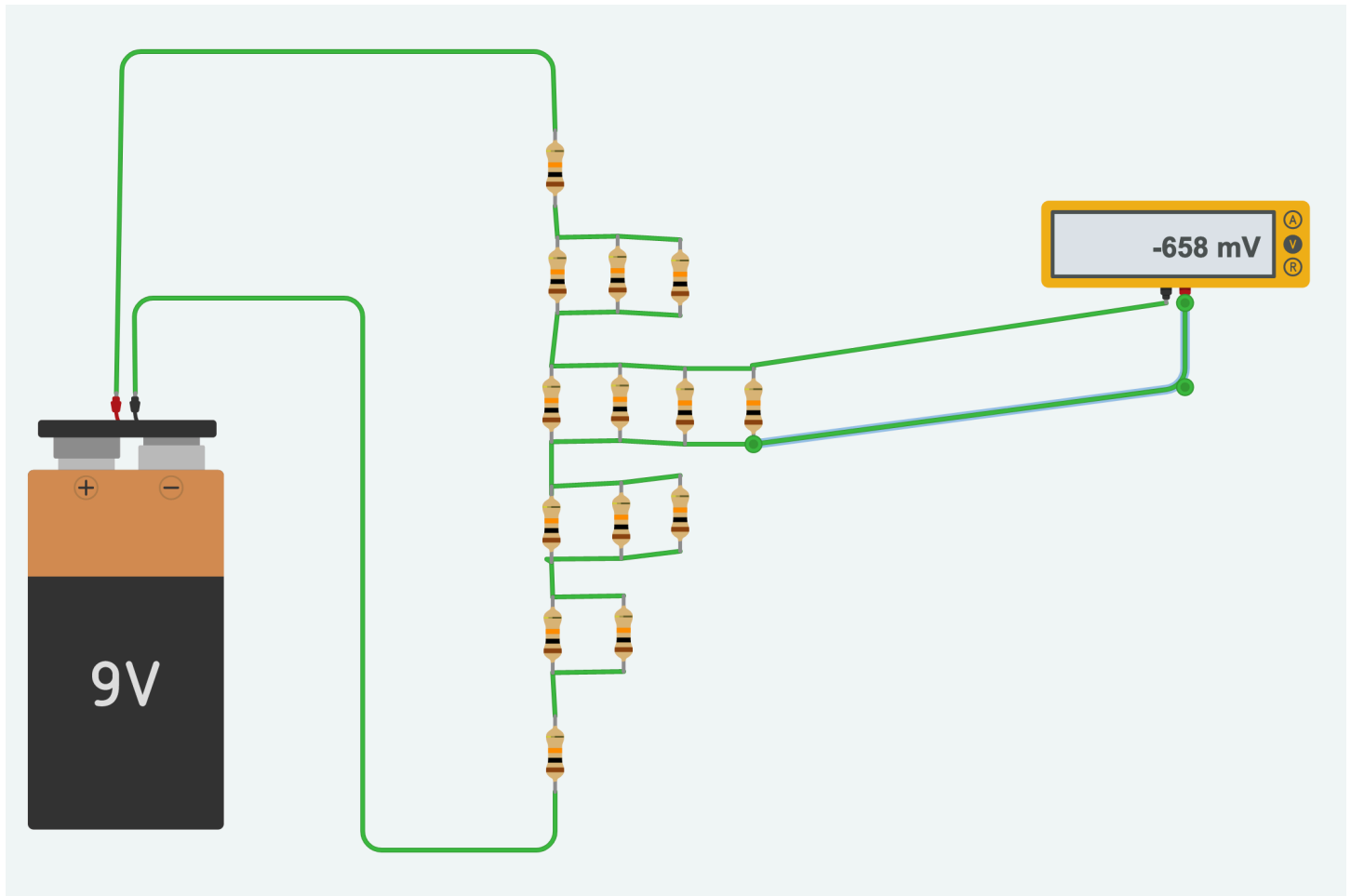
3.3.12 Parallel en serie weerstanden

Maak een programma dat onderstaande schakeling bereken of dit klopt.

Begin met het schrijven van 2 functies. Een voor de vervangingsweerstand te berekenen van 2 weerstanden in serie, en een om de vervangingsweerstand te berekenen van 2 weerstanden parallel.

Je kan dan in meerdere stappen de totale vervangingsweerstand berekenen.

Voorbeeld: als er 3 weerstanden van 100 Ohm parallel staan. Bereken je eerst 2 weerstanden parallel, daarna staat de vervangingsweerstand parallel aan de andere weerstand. Dus je moet 2x dezelfde functie aanroepen.



3.3.13 Verliesweerstand

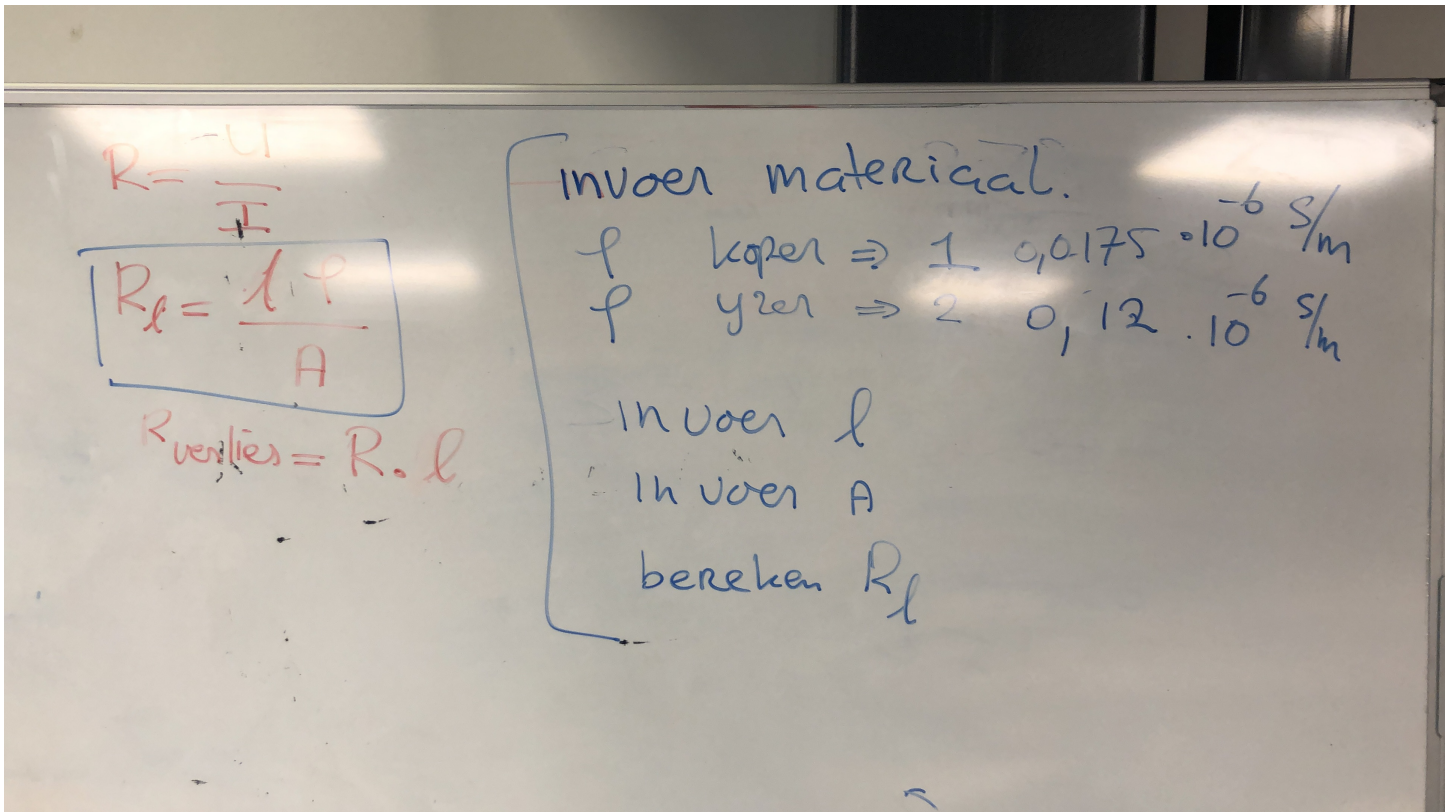
Bij het rondlopen door het gebouw zag ik dat een leraar het onderstaande op het bord had geschreven. Let op dat deze leraar ook een blunder heeft gemaakt. Zie je hem?

Deze leraar had heel veel sommetjes om de verliesweerstand te bereken.

Wij kunnen dat natuurlijk veel slimmer, we gaan een python programma maken dat l, a en een keuze van materiaal invoert, en de verliesweerstand berekend.

Maak een menu waar je minstens de 4 bekendste materialen inzet met een getal ervoor. Dat met dat getal kies je de waarde.

De functie die je maakt heeft GEEN print of input, deze berekend alleen de verlies weerstand.



Bron: Wikipedia.

Wet van Pouillet [\[bewerken | brontekst bewerken \]](#)

De **weerstand** van een **elektrische component** kan aan de hand van de soortelijke weerstand berekend worden met behulp van de **wet van Pouillet**.

$$R = \frac{\rho l}{A}$$

waarbij:

R = de **weerstand** van de draad in **ohm**

ρ = de soortelijke weerstand van het **materiaal** in **ohm-meter**

l = de **lengte** van de geleider in **meter**

A = de **dwarsdoorsnede** in **vierkante meter**.

Vaak wordt voor de eenheid van de soortelijke weerstand van het materiaal niet als 'ohm-meter²/meter', maar als 'ohm-meter' geschreven, omdat de termen boven en onder de deelstreep tegen elkaar wegvallen. Dit is echter discutabel, want het betreffende oppervlak en de afstand staan haaks op elkaar.

Omgekeerd kan de soortelijke weerstand berekend worden met:

$$\rho = \frac{RA}{l}$$

