|  |  |
| --- | --- |
| Biostatica  Door Alistair Vardy en Mark Schrauwen | Matlab Wk1.2 |

Inhoudsopgave

[Versiebeheer 2](#_Toc492975571)

[1 Inleiding 3](#_Toc492975572)

[2 Netjes programmeren 4](#_Toc492975573)

[2.1 Opdrachten en vragen 7](#_Toc492975574)

[3 Debuggen en gebruik maken van breakpoints 8](#_Toc492975575)

[3.1 Bugs 8](#_Toc492975576)

[3.2 Syntax fouten 8](#_Toc492975577)

[3.3 Programmeerfouten 8](#_Toc492975578)

[3.3.1 Voorbeeld van een programmeerfout 9](#_Toc492975579)

[3.4 Debuggen 9](#_Toc492975580)

[3.4.1 Breakpoints 9](#_Toc492975581)

[3.4.2 Voorbeeld 9](#_Toc492975582)

[3.5 Opdrachten en vragen 12](#_Toc492975583)

[3.6 Antwoorden 12](#_Toc492975584)

[4 Standaard Matlab functies 13](#_Toc492975585)

[4.1 Wat is een functie? 13](#_Toc492975586)

[4.2 Het aanroepen van een functie 13](#_Toc492975587)

[4.2.1 Functies zonder input 14](#_Toc492975588)

[4.2.2 Andere voorbeelden 14](#_Toc492975589)

[4.3 Het vinden van standaardfuncties 16](#_Toc492975590)

[4.4 Vragen en opdrachten 17](#_Toc492975591)

[4.5 Antwoorden 17](#_Toc492975592)

[5 Zelf een eenvoudige functie schrijven 18](#_Toc492975593)

[5.1 Inputs en outputs 18](#_Toc492975594)

[5.1.1 Hoe zoek je hulp bij het gebruik van functies 18](#_Toc492975595)

[5.2 Hoe gebruik je een functie? 19](#_Toc492975596)

[5.3 Zelf een functie schrijven 19](#_Toc492975597)

[5.3.1 Opzet van een functie en beschrijving van de werking daarvan 20](#_Toc492975598)

[5.3.2 Gebruik van commentaar 23](#_Toc492975599)

[5.3.3 Help tekst van de functie 23](#_Toc492975600)

[5.4 Nog een voorbeeld: de oppervlakte van een cirkel 24](#_Toc492975601)

[5.5 Vragen en opdrachten 28](#_Toc492975602)

[5.6 Antwoorden 29](#_Toc492975603)

# Versiebeheer

|  |  |  |  |
| --- | --- | --- | --- |
| Versie | Datum | Beschrijving | Door |
| 0.0 | 11-07-2017 | Eerste versie | Alistair Vardy |
| 0.1 | 28-08-2017 | Toevoegen subscripts en kleine verbeteringen. | Mark Schrauwen |
| 0.2 | 29-08-2017 | Opmerkingen van Denice Vis en Timothy Roos verwerkt. | Mark Schrauwen |
| 0.3 | 11-09-2017 | Aanvullingen: Inleiding. Zoeken van functies. Voorbeelden. Meer uniformiteit met andere readers. | Mark Schrauwen |
| 0.4 |  |  |  |
|  |  |  |  |

# Inleiding

Begin deze week heb je voor het eerste kennisgemaakt met Matlab. Je hebt geleerd wat variabelen zijn en je hebt geleerd wat operatoren zijn. Zoals je hebt gemerkt bevat deze cursus van maar 5 lesweken heel veel informatie. Je zult in deze beperkte tijd dus veel kennis moeten eigen maken. Zorg dat je niet gaat achterlopen en dat je begrijpt wat er in deze readers is uitgelegd.

In deze reader gaan we in op de volgende vragen:

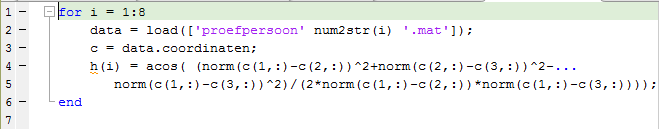
1. Hoe programmeer je netjes?
2. Wat zijn functies?
3. Waarom zijn functies handig?
4. Hoe gebruik je functies?
5. Hoe maak je functies aan?

Aan het einde van deze week worden er weekeindopdrachten uitgedeeld. Deze weekopdrachten staan op blackboard of krijg je via e-mail. Via blackboard moet je de gemaakte opdrachten weer opsturen. Dat kan tot en met de woensdag daarop tot 23:59u. Daarna is inleveren niet meer mogelijk en krijg je een onvoldoende voor de weekopdracht.

**Zie je een fout? Of heb je een aanbeveling dan horen we dat graag! Stuur dan een e-mail naar** [**mjschrau@hhs.nl**](mailto:mjschrau@hhs.nl) **en wij passen het dan z.s.m. aan.**

# Netjes programmeren

We bespreken twee programma’s. Ze doen hetzelfde; het berekenen van de kniehoek voor een gegeven boven- en onderbeenlengte voor een aantal verschillende proefpersonen, en er zijn duidelijke verschillen tussen deze lengtes (zie Figuur 1). Laat je niet afleiden door de complexiteit van de code, dit zal gaandeweg steeds duidelijker worden.



Figuur 1: een for-loop

We kunnen aan deze code nauwelijks zien wat het programma doet. Stel je voor dat hier ergens een fout in zou zitten. Het is dan buitengewoon lastig om te achterhalen wat er mis is gegaan. Als je deze code met ander persoon deelt dan is de kans groot dat hij/zij er niets van begrijpt.

Er zijn twee manieren van programmeren:

1. netjes
2. ‘quick and dirty’[[1]](#footnote-1).

Dat laatste is niet aan te raden.

Waarom niet?

Quick and dirty programmeren is soms onvermijdelijk. Deadlines moeten worden gehaald, of je hebt slechts iets gepland. Maar quick and dirty is veel te vaak de modus operandi. Het nadeel van deze manier van programmeren is dat je niet rekening houdt met jezelf en met een ander. Stel je hebt programma in Figuur 1 geschreven. Op het moment van schrijven begrijp je het helemaal.

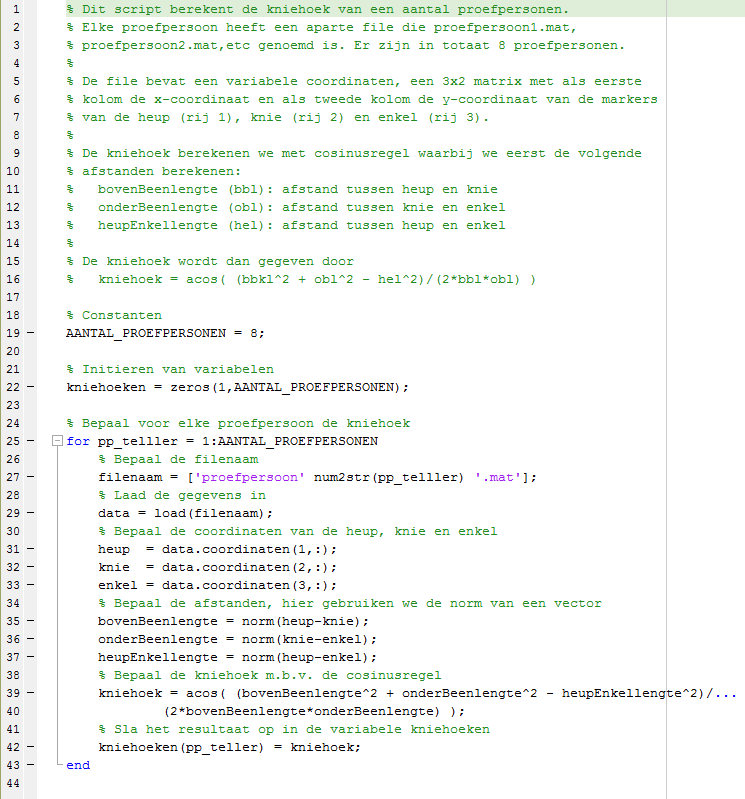
…

Bedenk je nu eens dat je twee weken later naar dat programma gaat kijken. Je hebt er twee weken niet aan gedacht. Denk je dat je het dan nog steeds scherp hebt?

Wat heb jij twee weken geleden op dit tijdstip gedaan?

Wat was je aan het doen?

Net programmeren houdt in dat je rekening houdt met de toekomstige lezer van de code. Dat kan jij zelf zijn of een ander persoon. **Bekijk nu het volgende programma:**



Figuur 2: een nette versie van hetzelfde programma

Dit programma is goed leesbaar en het is duidelijk wat de bedoeling is. Wat maakt dit programma nou zo veel beter? Hieronder een lijst met goede programmeereigenschappen:

1. **Maak gebruik van commentaar**

Hiermee geef je de lezer informatie over het programma en hoe verschillende tussenstappen uitgevoerd worden. Dit is essentieel als je met meerdere mensen samenwerkt of als jouw code hergebruikt zal worden. Bedenk ook dat je soms na een aantal weken je eigen code moet lezen en dan is het handig als je het nog snapt op basis van jouw eerder geschreven commentaar.

1. **Geef variabelen intuïtieve namen**

Vooral als je programma groter wordt, is het netjes benoemen van variabelen belangrijk. Hoewel het lijkt alsof het programmeren hiermee langer duurt, zal je meer tijd kwijt zijn met het oplossen van problemen (debuggen) als je hierop beknibbelt. Daarnaast wordt het programma ook veel leesbaarder, iets dat in punt 1 ook is aangestipt. Er zijn meerdere systemen voor naamgeving mogelijk. Welke je gebruikt maakt niet uit maar wees consequent. Hieronder volgt een voorbeeldsysteem waarbij in commentaar de eenheden wordt vermeld.

* Constanten (zoals de zwaartekrachtversnelling) worden vaak met hoofdletters aangegeven:
  + ZWAARTEKRACHTVERSNELLING = 9.81; % m/s^2
* Variabelen. Begin met een kleine letter en geef de eerste letter van een eventueel volgend woord een hoofdletter (dit wordt CamelCasing genoemd):
  + onderBeenlengte = 0.6; % m
  + bovenBeenlengte = 0.5; % m
  + kniehoekSnelheid = 0.1; % rad/sec

1. **Maak gebruik van lege regels**

Dit geeft meer overzicht, niets meer, niets minder

Geen van de bovenstaande programmeereigenschappen maakt het programma trager of slechter. De voordelen zijn dat het programma beter leesbaar wordt en er minder kans op programmeerfouten ontstaan. Bovendien kun je het programma na twee weken/maanden veel sneller begrijpen omdat je het sneller kunt lezen. Ook anderen zullen er blij mee zijn. Het is dus winst voor de programmeur en de collega’s van die programmeur.

## Opdrachten en vragen

Voor dit onderdeel zijn er maar twee opdrachten:

1. Zorg dat je bovenstaand hoofdstuk heb gelezen en begrepen.
2. Pas het netjes programmeren altijd toe!

# Debuggen en gebruik maken van breakpoints

## Bugs

Als we programmeren maken we altijd fouten. Dit is helaas onvermijdelijk. Om fouten op te sporen heeft Matlab een aantal hulpmiddelen. Voordat we beginnen is het belangrijk je te realiseren dat er twee soorten fouten zijn:

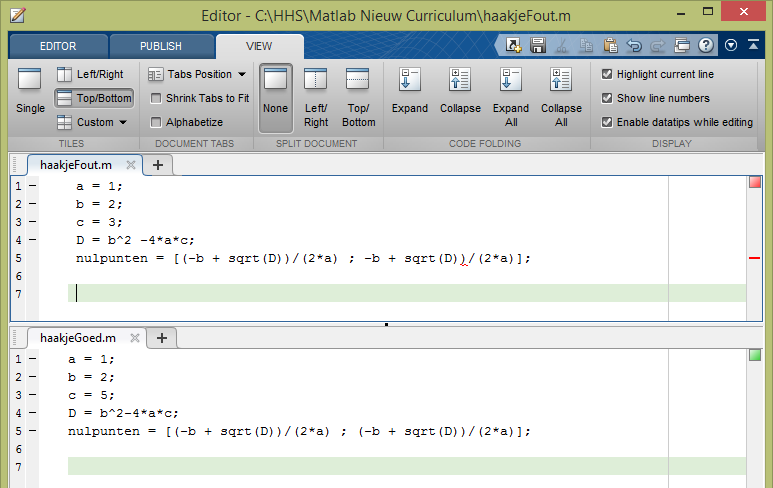
1. Syntax fouten
2. Programmeerfouten

## Syntax fouten

Syntax fouten zijn fouten in de manier waarop je een commando invoert. Matlab zal je vertellen dat er iets mis is. Bijvoorbeeld:

* Het vergeten van een haakje
* De naam van een variabelen verkeerd spellen
* Een niet bestaande functie aanroepen

Als je het programma probeert te runnen, zal Matlab ook een foutmelding geven. De Matlab editor helpt je hier ook bij (zie Figuur 3). Als je ergens een haakje bent vergeten of een andere fout hebt gemaakt geeft Matlab dit aan met rode en oranje kleurtjes aan de rechterkant van de code:

**

Figuur 3: een voorbeeld met haakjes. Aan de rechterkant van het bovenste script zie je rode blokjes. Matlab geeft hier aan dat er iets fout gaat. Dit helpt de programmeur snel fouten te herkennen.

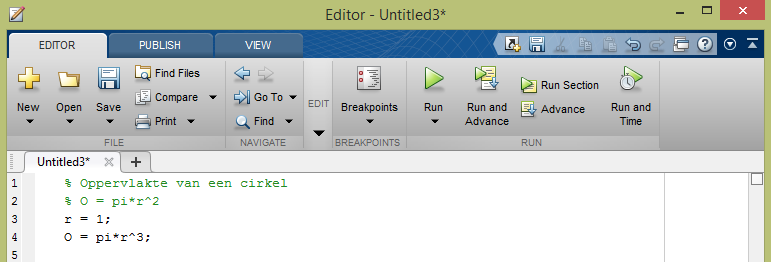
## Programmeerfouten

Programmeerfouten zorgen dat het programma iets anders doet dan je zou willen. Hierbij kan het zelfs gebeuren dat er geen foutmelding komt. Dat betekent dat jij, de programmeur, denkt dat jouw programma werkt, terwijl in werkelijkheid klopt er niets van. Dat is hetzelfde als denken dat je naar huis reist, maar aan het einde van je reis aan de andere kant van het land bent. Je bent dan van A naar B gereist, maar had bij C moeten uitkomen.

Misschien heb je het nu nog niet door, maar dit type fouten zijn de ergste fouten die je kunt krijgen als programmeur en ze zijn onvermijdelijk! Sterker nog je zult straks voorbeeldprogramma’s zien waar jij de fouten uit moet gaan halen.

### Voorbeeld van een programmeerfout

Hieronder is een voorbeeld gegeven waarbij de code op zich werkt, maar er is een fout in de gebruikte formule voor de oppervlakte van een cirkel.



Figuur 4: een programmeerfout

Deze laatste situatie is bijzonder vervelend. Gelukkig biedt Matlab een aantal oplossingen om dit soort problemen (en de voorgaande problemen) doeltreffend aan te pakken.

## Debuggen

Het oplossen van fouten wordt ook wel ‘debuggen’ genoemd. De term bug komt oorspronkelijk komt deze term uit de tijd van de enorme kasten van computers die soms kortsluiting veroorzaakte doordat er een insect (bug) geplet tussen de printplaten (rekeneenheden) zat. Deze term wordt nu gebruik voor een foutje in een stuk code.

### Breakpoints

Als we willen weten waar een fout zit, is het handig om het programma even op een bepaalde plek stil te zetten en dan stapje voor stapje verder te gaan. Matlab heeft hiervoor ‘breakpoints’. Hoewel het mogelijk is om vanuit het Command Window line te debuggen, beschrijven we in deze reader alleen het debuggen vanuit de editor. Het plaatsen van een breakpoint kun je in de editor doen door naast het regelnummer te klikken:

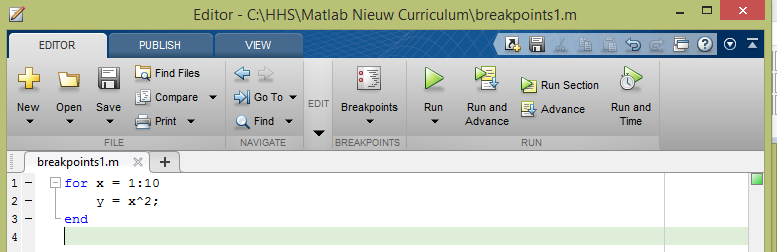
### Voorbeeld

Je hebt de functie . Als je een overzicht wilt hebben van het gedrag van deze functie, kun je bijvoorbeeld x laten lopen van 1 t/m 10 en voor iedere x een y-waarde berekenen. In Matlab programmeer je dat met behulp van een lus (of loop) zo:

for x = 1:10

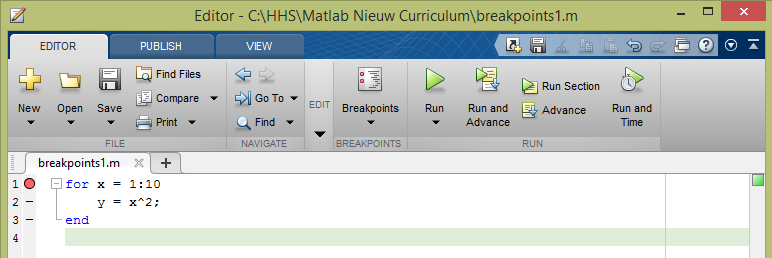
y = x^2;

end

****

Figuur 5: breakpoint voorbeeld, er is nog geen breakpoint geplaatst.

We kunnen een zogenaamd break point zetten waar we willen dat het programma stopt tijdens het uitvoeren. Dit doen we door op het streepje rechts van het regelnummer te klikken. Als het breakpunt wil verwijderen kun je nog een keer klikken:

****

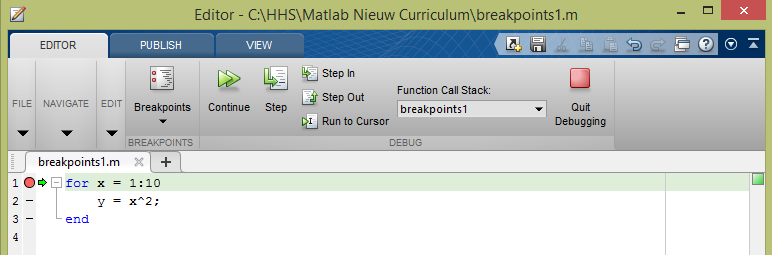
Figuur 6: nu is er een breakpoint geplaatst

Schrijf de code uit Figuur 5 over (het maakt niet uit dat je het nu nog niet begrijpt).

Ze een breakpoint (het zou er nu uit moeten zien als in Figuur 6)

Run het programma

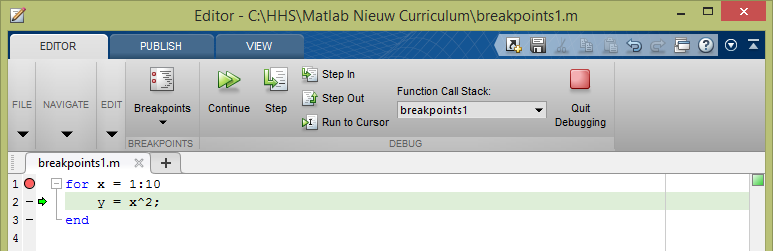
Als we het programma runnen dan zien we een groene pijl die aangeeft waar het programma zich nu bevindt. Het programma is nu gepauzeerd. De editor ziet er nu zo uit:



Figuur 7: runnen van een programma met een breakpoint

De groene pijl geeft aan dat het programma bij regel 1 is gepauzeerd. Rechtsboven zien we ook de rode knop “Quit debugging”[[2]](#footnote-2) waarmee je het debug proces kan beëindigen.

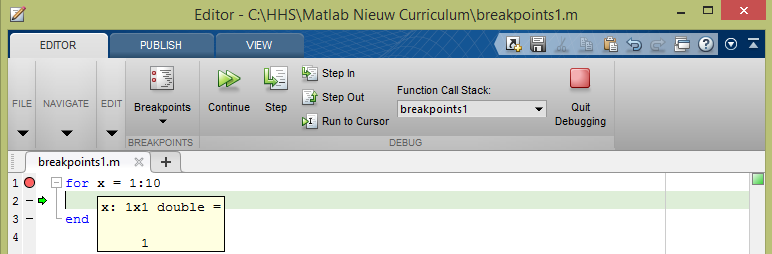
Door op de F10 toets te drukken kunnen we regel voor regel door het programma. Het volgende figuur geeft aan wat er dan gebeurt:



Figuur 8: de breakpoint staat nu op de volgende regel.

We kunnen nu ook informatie krijgen over de inhoud van de variabelen. Dit kan op twee manieren:

1. Als we de muis boven een variabele houden, dan zien we de inhoud daarvan:



Figuur 9: extra informatie als je met de muis op de regel gaat staan.

Let wel, je kunt alleen informatie zien die in de vorige stappen is uitgevoerd. Het programma is gepauzeerd op de regel waar het groene pijltje staat en deze regel is nog niet uitgevoerd.

1. In de Command Window zie je niet de normale Matlab prompt (het >> teken waar je commando’s achter typt) maar een aangepaste prompt: K>>. Hier kun je ook de inhoud van variabelen opvragen.



Figuur 10: de inhoud van een variabele terwijl je aan het debuggen bent.

Dit proces van stap voor stap door het programma lopen, wordt **debuggen** genoemd. Je kunt door het inspecteren van de waarden erachter komen of er programmeerfouten in staan.

## Opdrachten en vragen

1. Zie Tabel 1. Kopieer deze code in Matlab. Op welke regel kun je het beste een breakpoint geplaatst worden als we willen weten wat er in de variabele *vector* staat?
2. Bekijk wat er in de variabele *vector* staat.

|  |
| --- |
| 1. clear all; 2. close all; 3. clc; 4. vector = randn(1,11); 5. vecKwad = vector .^ vector; |

1. We nemen even aan dat je nog niet weet wat de code in regel 1 van Tabel 1 doet. Op welke

regel kun je nu het beste een breakpoint zetten?

Tabel 1: code voorbeeld vraag 1

1. Zie de code in Tabel 2. Het is niet de bedoeling dat je elke regel code gaat begrijpen. Dat is voor nu te ingewikkeld. Wij verwachten dat je in 99% van de gevallen code begrijpt en nu hoeft dat een keer niet. Op welke regel(s) moet je een breakpoint zetten om te zien wat de functie round() doet?
2. Zoek m.b.v. een breakpoint uit wat er in regel 10 in de variabele *trunced* staat.
3. Wat is het verschil tussen de waarde van *output* en *trunced*?
4. Zet een breakpoint op regel 4 en verander de waarde van de variabele naar de waarde 5. Onderzoek door gebruik te maken van breakpoints wat het programma doet.

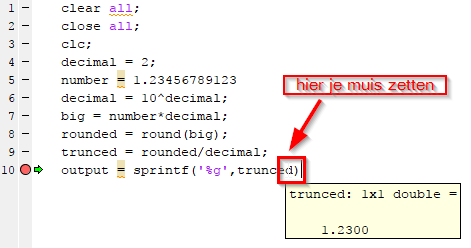
Tabel 2: code die een bewerking op decimalen uitvoert.

|  |
| --- |
| 1. clear all; 2. close all; 3. clc; 4. decimal = 2; 5. number = 1.23456789123 6. decimal = 10^decimal; 7. big = number\*decimal; 8. rounded = round(big); 9. trunced = rounded/decimal; 10. output = sprintf('%g',trunced) |

1. Wat is handig aan het gebruiken van breakpoints?

## Antwoorden

1. Op regel 5. Waarom? Omdat dan de code in regel 4 al is uitgevoerd en het resultaat van de fucntie randn(1,11) in de variabele *vector* is opgeslagen. Je wilde weten wat er in de variabele *vector* is opgeslagen.
2. Dit is een opdracht er is dus geen antwoord.
3. Op regel 1. Let op! Nu is deze regel nog niet uitgevoerd. Dat uitvoeren gebeurd pas als je op F10 hebt gedrukt, dan spring je naar de volgende regel. Tijdens die actie kun je zien wat deze regel exact doet. Als het goed is, is je opgevallen dat alle variabelen in de Workspace worden verwijderd.
4. Regel 8 of 9. Regel 9 is het meest handig als je alleen het resultaat van de functie round() wil bestuderen. Regel 8 is het meest handig als je wil bekijken wat er exact gebeurt voordat je de functie round() wil uitvoeren. In de praktijk zal je allebei de scenario’s even vaak zien voorkomen.
5. Je hebt een breakpoint op regel 10 gezet en je gaat vervolgens met de muis op de variabele *trunced* staan:



Figuur 11: Matlab geeft tijdens het debuggen aan wat de waarde is van een variabele als je met muis op de variabele staat.

1. Het verschil is te zien in de Workspace. Daar staan immers de variabele die je hebt gebruikt tijdens het uitvoeren van het programma. Het verschil is dat: output een string is en trunced een getal. Een ander verschil is dat trunced de waarde 1.2300 bevat en output de waarde ‘1.23’
2. Je hebt gezien dat je met deze code de aantal decimalen van een variabele kunt beperken. In dit geval bevat de variabele *output* de waarde ‘1.23457’ dat zijn 5 decimalen achter de komma.
3. Het handige is dat je krijgt te zien wat er in elke regel exact gebeurd. Als je jezelf afvraagt: ‘wat doet deze code exact’? Dan helpen breakpoints je om dat heel erg gemakkelijk te bepalen. Er bestaat onzes inziens geen krachtiger hulpmiddel om te leren wat een stuk code doet. Vind je het antwoord van deze vraag onvolledig? Of heb je een beter antwoord? Stuur mij een e-mail: [mjschrau@hhs.nl](mailto:mjschrau@hhs.nl). Als je een verbetering heb, zal ik dat antwoord hier plaatsen. Je mag ook andere opmerkingen over de readers/cursus sturen, die zal ik altijd z.s.m. verwerken.

# Standaard Matlab functies

We behandelen in dit hoofdstuk de volgende onderwerpen:

* Wat is een functie?
* Inputs en outputs
* Hoe gebruik je een functie?
* Hoe zoek je hulp bij het gebruik van functies
* Zelf een functie schrijven

## Wat is een functie?

Een functie in Matlab is een programma dat een bepaalde opdracht uitvoert. Dit kan bijvoorbeeld het berekenen van het gemiddelde van een rij getallen zijn. Zo zijn er een hele hoop (wiskundige) berekeningen die we kunnen doen op basis van **standaardfuncties** in Matlab. Een aantal van deze standaardfuncties staan hieronder beschreven en komen dus standaard met Matlab bijgeleverd. Dit is een greep:

* Goniometrische functies
  + sin, cos, tan – de sinus, cosinus en tangens van een hoek (in radialen)
  + asin, acos, atan – de inverse sinus, cosinus en tangens van een hoek (in radialen)
  + sind, cosd, tand – de sinus, cosinus en tangens van een hoek (in graden)
  + asind, acosd, atand – de sinus, cosinus en tangens van een hoek (in graden)
* De logaritme en exponentiele functie
  + log – de natuurlijke logaritme
  + log2, log10 – de logaritme met grondtal 2 en 10
  + exp – de e-macht
* Veelgebruikte bewerkingen
  + mean – het gemiddelde van een vector of een matrix (rijen of kolommen)
  + std – de standaarddeviatie van een vector of een matrix (rijen of kolommen)
  + sum – de som van een van een vector of een matrix (rijen of kolommen)
  + cumsum – de cumulatieve som (handig voor numeriek integreren)
  + trapz – de trapeziumregel voor integreren
  + cumtrapz – de cumulatieve trapeziumregel voor integreren
  + gradient – de numerieke afgeleide van een rij getallen of een matrix

*Waarom is Matlab zo handig voor een Bewegingstechnoloog?*

Dat was één van de eerste vragen gesteld in deze cursus. Een antwoord op deze vraag is: omdat Matlab zoveel standaardfuncties bevat. Deze standaardfuncties besparen programmeurs en bewegingstechnologen veel tijd in het ontwikkelen van hun algoritmes. Dat is dus erg prettig.

## Het aanroepen van een functie

De hierboven beschreven standaardfuncties kun je gebruiken door ze aan te roepen. Dat aanroepen van een functie is heel erg gemakkelijk.

Typ in het Command Window: cos(pi)

Het antwoord is natuurlijk -1.

De functie cos() wordt standaard meegeleverd en kun je één argument meegeven. Dat argument is in dit geval ‘pi’[[3]](#footnote-3).

Doorgaans kun je een functie herkennen aan haakjes:

**cos ( )**

Tussen de haakjes geef je iets op aan de functie, dat was in dit geval ‘pi’. Dit noemen we een *argument* of *input* van de functie.

### Functies zonder input

Het is echter niet zo dat elke functie een input nodig heeft. Je hebt ook functies die werken zonder het geven van input. Een voorbeeld van zo’n functie is randn().

Typ in het Command Window: randn()

Herhaal dit een aantal keer

Wat valt op?

Als je de functie randn() een aantal keer uitvoert, zie je telkens een ander getal in het Command Window. Deze functie genereert random getallen voor de gebruiken en vereist dus geen input.

Je ziet dat het aanroepen van een functie heel erg gemakkelijk is, typ de naam van de functie en geef de functie een argument of waarde.

### Andere voorbeelden

Je kunt in Matlab ook de sinus berekenen van een hoek uitgedrukt in graden:

Typ in het Command Window: sind(90)

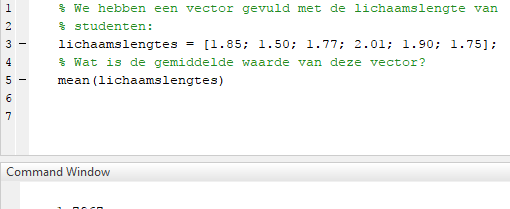
Is het resultaat wat je verwacht?

Sind() is een functie die de sinus van een getal in graden evalueert. De ‘d’ van sind() staat voor: ‘degree’. We weten uit de Wiskunde dat

Typ in het Command Window: sin(pi/2)

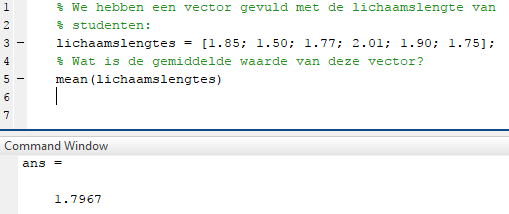
Daar komt gelukkig hetzelfde antwoord uit. In het ene geval geven we de functie het argument in graden mee. In het tweede geval geven we de functie het argument in radialen mee.

Stel, we willen de gemiddelde lichaamslengte bereken van een aantal studenten. We maken dan een vector met lichaamslengtes (zie Figuur 11):



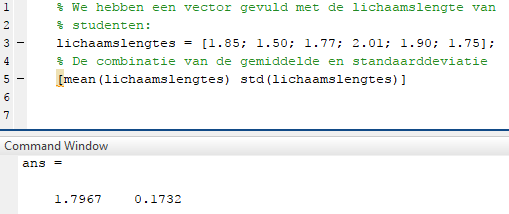
Figuur 12: een voorbeeld van een aantal lengtes in een vector.

Het berekenen van de gemiddelde lengte kan dan eenvoudig met de fucntie mean(). De mean van een vector is hetzelfde als *het gemiddelde* van een vector. In Figuur 13 is de toepassing van deze functie te zien en de uitkomst.



Figuur 13: een voorbeeld van de functie mean().

Dit voorbeeld kan worden uitgebreid met de standaard deviatie. We willen hier ook weten wat de standaarddeviatie is van de lengtes van het groepje studenten:



Figuur 14: het combineren van twee maten (gemiddelde en standaarddeviatie) in één vector.

**Typ de code over en vul de lichaamslengtes aan met jouw lichaamslengte.**

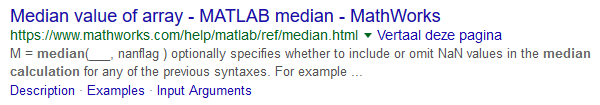
Het aanroepen en gebruiken van functies is, zoals je hebt gezien erg handig en zal ons veel tijd gaan besparen in de toekomst.

## Het vinden van standaardfuncties

Een veel voorkomende en terechte vraag van studenten is: ‘hoe vindt je een bepaalde Matlab functie’? Het antwoord is vervelend genoeg: door veel te lezen!

|  |
| --- |
| **<Een zoekmachine> is your friend.** |

Stel je wil de *mediaan* berekenen van de lijst van lichaamslengtes. Je kunt in een willekeurige zoekmachine een aantal keywords typen zoals: *Matlab calculate median*. Je krijgt dan dit resultaat:

[](https://www.google.nl/search?client=firefox-b-ab&dcr=0&q=Matlab+calculate+median&oq=Matlab+calculate+median&gs_l=psy-ab.3..0i19k1j0i8i30i19k1l3.13988.13988.0.14213.1.1.0.0.0.0.39.39.1.1.0....0...1.1.64.psy-ab..0.1.38.TNPsdLHDhsk)

Figuur 15: het zoeken van een functie.

Bekijk Figuur 14

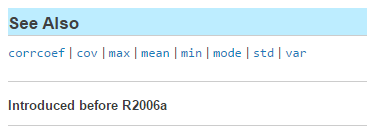
Zie je het voorbeeld van de mediaan functie?

Het voorbeeld laat zien dat de mediaan functie in Matlab: median() heet. Dat is valt te verwachten natuurlijk. Een andere manier om meer functies te vinden is om de Matlab documentatie te lezen. Verreweg de gemakkelijkste manier is om de *See Also* kopjes te lezen.

Typ in het Command Window: doc median

Er wordt nu een venster geopend met Matlab documentatie.

**Klik op *See Also*.** Dit is het resultaat:



Figuur 16: See Also informatie van de functie mean().

Je ziet in Figuur 15 dat Matlab een aantal suggesties doet voor andere functies. In dit geval zie je de mean() functie terugkomen, maar ook een max() functie en een min() functie. Heel handig zijn dit interne links naar andere documentatie pagina’s.

Klik op de link ‘max’

Je merkt dat je nu uitkomt op de max documentatie pagina van Matlab. Als je dit met regelmaat doet, kom je vanzelf bij handige functies uit voor de toekomst. Later in deze cursus gaan we je laten zien hoe je de documentatie het beste kunt benaderen en gebruiken. Je kunt hier natuurlijk nu al mee beginnen.

## Vragen en opdrachten

1. Je ziet aan het begin van de hoofdstuk een aantal standaard Matlab functies. Je moet in deze en de volgende oefeningen gebruik maken van deze functies. Maak een nieuw Matlab script aan en vul het met deze variabele: lichaamslengtes = [1.85; 1.50; 1.77; 2.01; 1.90; 1.75]; Tel m.b.v. een functie alle waardes bij elkaar op.

## Antwoorden

# Zelf een eenvoudige functie schrijven

## Inputs en outputs

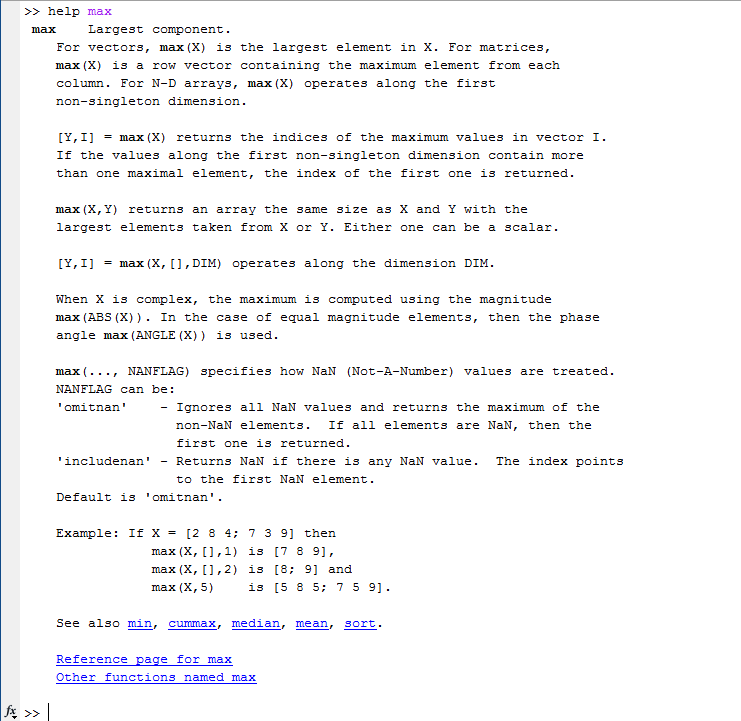
Een functie heeft daarbij ook nog een belangrijke eigenschap. In plaats van een set commando’s uit te voeren, heeft een functie een aantal *inputs* en *outputs*. We geven hieronder een schematische weergave.



Figuur 17: een schematische weergave van een functie

### Hoe zoek je hulp bij het gebruik van functies

Als we een onbekende functie willen gebruikten, dan kunnen we de help-tekst daarvan lezen door bijvoorbeeld het commando help max in de command line in te typen.



Figuur 18: de output van de Matlab help

Als we nog meer informatie willen dan kunnen we ook de documentatie raadplegen. Dit kun je doen door de link te volgen of het commando doc max in de command line in te voeren. Het command doc brengt je naar de algemene help pagina van Matlab.

##### Kijk nu naar de documentatie van de functie mean

Bepaal het commando waarmee je de rijgemiddelden van de volgende matrix berekent



## Hoe gebruik je een functie?

## Zelf een functie schrijven

Matlab heeft een enorme collectie ingebouwde functies. Uiteraard kun je zelf ook functies schrijven die precies doen wat jij wilt. We geven een voorbeeld.

Stel, je hebt een tweedegraadsvergelijking waarvan je de nulpunten wilt berekenen:



Dit kunnen we oplossen m.b.v. de *abc*-formule:

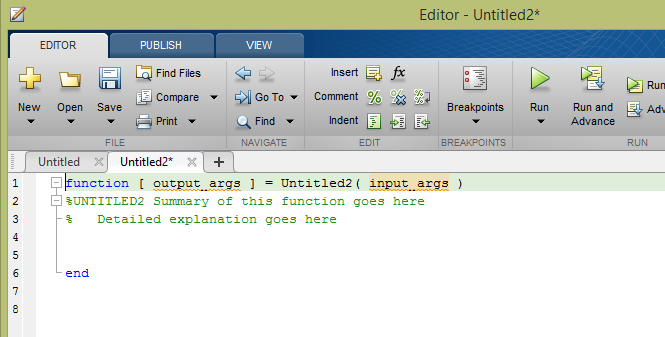


Om dit op te kunnen lossen hebben we dus alleen de waarden van , en  nodig. Dit zullen dan ook onze *inputs* zijn. Het resultaat van de berekening levert twee nulpunten, dat zijn onze *outputs*. We weten overigens dat niet voor elke combinatie van , en er twee oplossingen zijn. Deze situatie ondervangen we in de functie.

### Opzet van een functie en beschrijving van de werking daarvan

We beginnen met het aanmaken van een nieuwe functie waarbij we beschrijven wat de functie doet. Dit kunnen we in de editor doen door New - > Function te kiezen

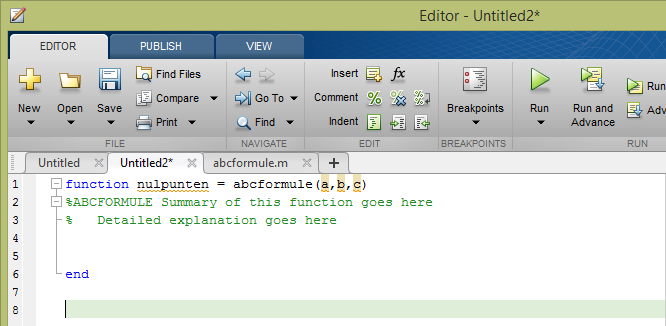
We krijgen dan de volgende code



Figuur 19: de opbouw van een functie.

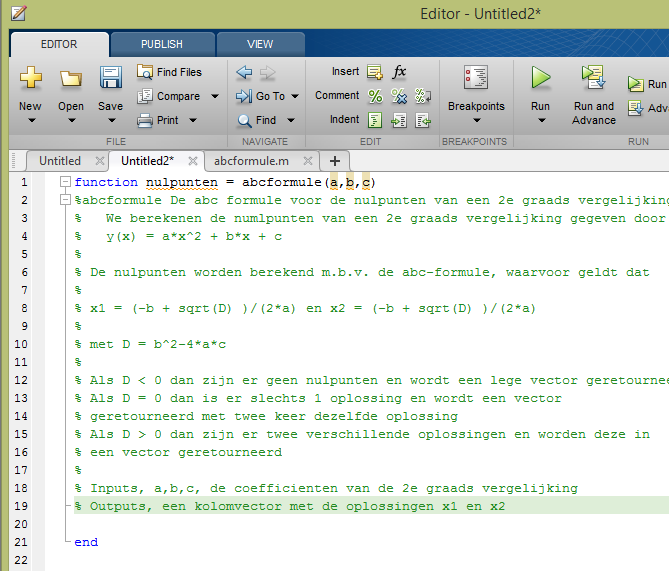
|  |
| --- |
| Merk op dat als je een bestand wijzigt en deze wijzigingen zijn nog niet opgeslagen dan staat er een \* boven de bestandsnaam/tab (zie Figuur 18). |

Matlab geeft ons alvast het recept om een functie te bouwen. De naam is nu Untitled. Vervang dit door de naam van de functie en vervang ook de inputs en outputs:



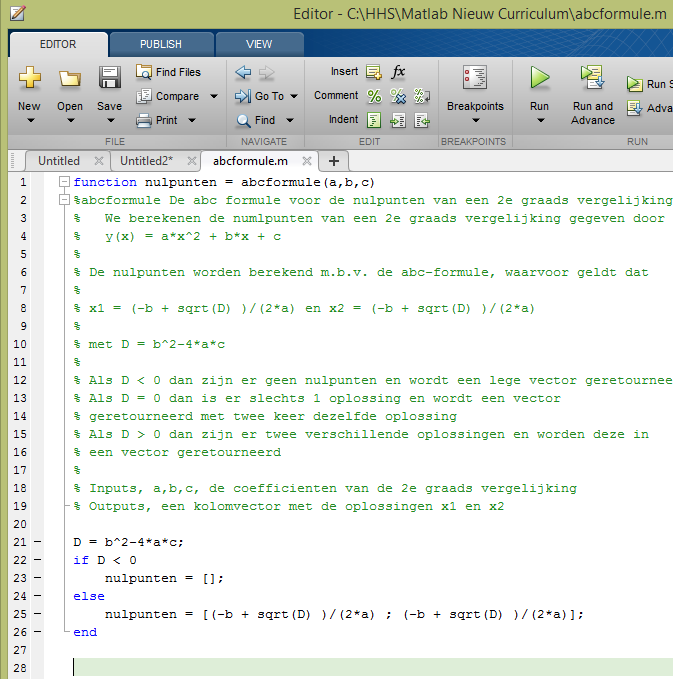
Figuur 20: de functie verder aangevuld.

Voeg een beschrijving van de functie toe en geef ook aan wat de inputs en outputs aan met commentaar zodat de gebruiker precies weet wat hij/zij in moet voeren



Figuur 21: de functie aangevuld met commentaar

We voeren de formules in (als je nog niet alles begrijpt dan is dat in dit geval nog niet erg):



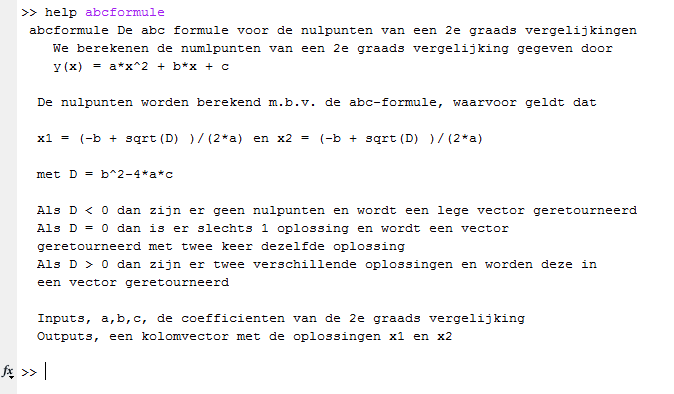
Figuur 22: de functie verder afgemaakt

### Gebruik van commentaar

We zien dat er verschillende mogelijkheden zijn, 1, 2 of geen oplossingen. Om de gebruiker te helpen voegen we commentaar toe waarbij we uitleggen wat we doen. Dit helpt enorm bij de leesbaarheid van een programma.

### Help tekst van de functie

Omdat we een beschrijving hebben gegeven van wat het programma doet kan een gebruiker nu ook help abcformule intoetsen en deze tekst lezen:

**

Figuur 23: de help output van de door ons gemaakte functie

De functie wordt opgeslagen met dezelfde naam als de functie; in ons geval dus abcformule.m. Als je dit vergeet dan geeft Matlab een foutmelding. Dus de naam van de m-file, in dit geval een function file genaamd omdat het enkel een functie beschrijft, moet gelijk zijn aan de naam van de functie.

## Nog een voorbeeld: de oppervlakte van een cirkel

We gaan nu een heel eenvoudige functie in Matlab maken. De functie heeft als input de straal van een cirkel en als output de oppervlakte van een cirkel met die straal.

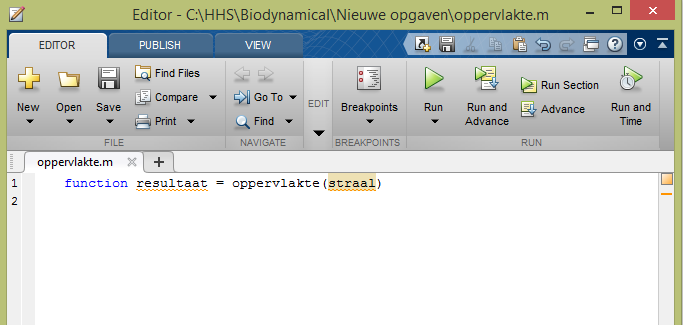
Het aanmaken van een nieuwe functie

Een Matlab functie begint door een nieuwe file te openen en te beginnen met een regel die het volgende aangeeft:

1. Dat het hier een Matlab functie en geen script betreft
2. Wat de inputs zijn
3. Wat de outputs zijn

Voor onze oppervlakte berekening beginnen we met

function resultaat = oppervlakte(straal)



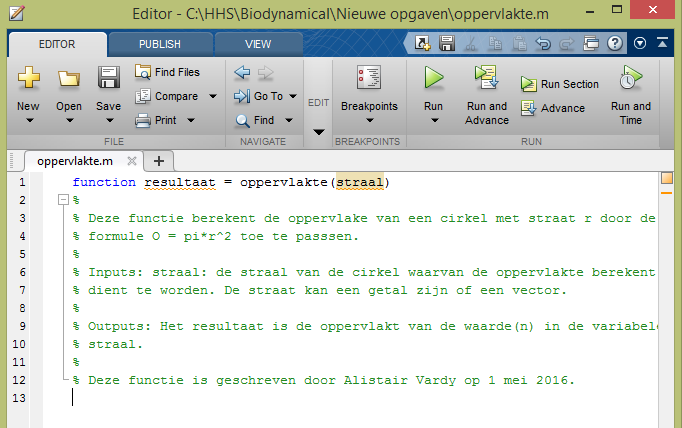
Figuur 24: een nieuwe functie voor het bereken van de oppervlakte van een cirkel.

We slaan vervolgens de file op. Matlab zal een suggestie doen, namelijk *oppervlakte.m.* Het is belangrijk dat de functienaam en de naam van de file hetzelfde zijn.

We kunnen nu regels code toevoegen om de bewerking compleet te maken. Echter, voordat we dat doen, is het altijd raadzaam op extra informatie toe te voegen.

Als je bijvoorbeeld in het Command Window “help mean” intypt, dan krijg je informatie te zien over hoe je de functie moet gebruiken. Voeg altijd deze hulp tekst toe zodat anderen eenvoudig kunnen zien hoe ze de functie moeten gebruiken. Ook voor jezelf is het goed omdat het opschrijven meteen een kader biedt voor de code die je later gaat toevoegen.

Hulp-tekst kun je als commentaar meteen achter de eerste regel toevoegen. Commentaar is tekst dat voorafgegaan wordt door een %-teken.



Figuur 25: verder aangevuld.

##### Voeg nu de overige code toe

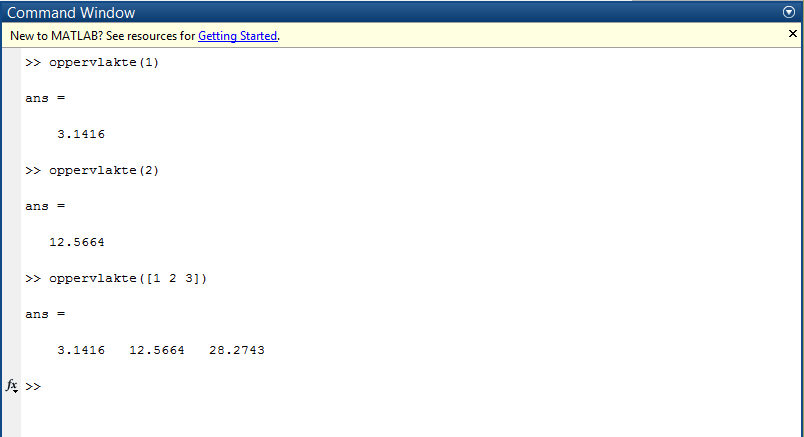
##### Test de functie in Matlab door de volgende commando’s uit te voeren:

oppervlakte(1);

oppervlakte(10);

oppervlakte([1 10 100]);

##### Hieronder volgen een aantal andere voorbeelden. Komen jouw antwoorden overeen?



Figuur 26: de output in het Command Window

##### Check de help tekst door help oppervlakte in te toetsen.

**Je kunt, zoals in het derde voorbeeld hierboven, ook een vector van stralen invoeren. Dit werkt echter alleen als de elementen bij het kwadrateren puntsgewijs vermenigvuldigd worden. Is dit bij jouw functie het geval?**

## Vragen en opdrachten

1. Hoe kun je in Matlab opzoeken hoe je de functie **round** moet gebruiken?
2. Hoe kan je ervoor zorgen dat een vermenigvuldiging of machtsverheffing tussen twee vectoren of matrices puntsgewijs wordt uitgevoerd?
3. Als je een functie maakt met de volgende regel

function resultaat = test1(a)

Hoe kun je dan het beste de filenaam noemen als je deze functie op wil slaan?

1. Hoe bereken je het gemiddelde van de vector v = [1 2 3 4 5];?
2. Bekijk de hulp tekst van de functie round en geef zonder het uit te voeren aan wat het resultaat is van het commando round(3.14159,3);
3. Welke functie wordt gebruikt voor het berekenen van de standaarddeviatie?

## Antwoorden

1. M.b.v. het commando help round
2. M.b.v. een “.” ; bijvoorbeeld straal.^2
3. test1.m
4. M.b.v. het commando mean(v)
5. 3.1220
6. Std()

1. Quick and dirty is uiteindelijk slow and cumbersome [↑](#footnote-ref-1)
2. Je kunt ook op shift+F5 drukken. [↑](#footnote-ref-2)
3. De variabele ‘pi’ is een standaard Matlab variabele. [↑](#footnote-ref-3)