Voorbeelden van instructies voor docenten

Benny Aalders Vincent Velthuizen

19 juni 2017

1 Doel van dit document

Om te voldoen aan de eisen van het SLO met betrekking tot Digitale Geletterdheid moeten leerlingen bepaalde vaardigheden verwerven. Een deel van die vaardigheden worden in zekere mate al gedekt door het wiskundecurriculum. Dit document beschrijft hoe het leren en verifiëren van die vaardigheden expliciet te maken is. Het is de bedoeling zo min mogelijk af te wijken van de bestaande leerlijnen. Voor de leerlingen bestaat er een apart document. Dit document is opgesteld om gelijk te lopen met de gebruikte wiskunde methode.

In deze versie van het document zijn voor het begin, het midden en het einde van de onderbouw opdrachten opgenomen. Dit geeft een beeld van de verwachtingen die wij hebben van het niveau dat de leerlingen hebben en zullen verwerven gedurende deze periode. Er is momenteel alleen een document voor de leerlingen beschikbaar dat hoort bij de methode Getal & Ruimte.

2 Opbouw van dit document

De hoofdtekst in dit document is bedoeld om de docent in staat te stellen de leerling te helpen bij het zelf verwerven van de vaardigheden. We geven voor ieder probleem een oplossing in pseudocode die deels of geheel aan de leerling kan worden aangeboden om hem of haar op weg te helpen. We geven ook aan waar veelvoorkomende fouten zitten. De opdrachten zelf geven aan welke testdata relevant zijn om de werking van het programma te verifiëren.

Het is aan de docent om te bepalen hoe de code beoordeeld moet worden. Onze suggestie is te streven naar werkende code (niet noodzakelijk mooie, elegante of efficiënte code) en om de beoordeling te beperken tot voldoende/onvoldoende.

In de appendix zijn referentie-implementaties opgenomen voor de verschillende opdrachten. Deze implementaties dienen niet als antwoordmodel maar kunnen door de docent gebruikt worden om snel een goede oplossing op te kunnen zoeken om een leerling verder te helpen.

3 Opdrachten HAVO/VWO

3.1 Jaar 1

De leerlingen zijn bezig geweest om de ggd en het kgv uit te rekenen. Dit doen ze aan de hand van een stappenplan. De bedoeling is om een programma te

schrijven dat hetzelfde stappenplan volgt. In pseudocode ziet dat er uit zoals in Programma 1.

In opdrachten **P1** en **P2** moeten de leerlingen experimenteren met de (aangeleverde) functie isDeler?

In opdracht **P3** moeten de leerlingen een lus gebruiken, dit is misschien nieuw. Merk op dat dit een FOR lus dient te zijn, aangezien het aantal iteraties bekend is. Merk ook op dat elk getal een deler is van zichzelf. Een slimme eindconditie (bijvoorbeeld getal/2) dient daar rekening mee te houden.

De extra moeilijkheid in **P4** is het handig doorlopen van de lijsten die ze maken (die lijsten maken is een kwestie van het kopiëren/plakken van de code uit **P3**). De leerling zou onderaan de lijst kunnen beginnen waardoor juist de kleinste gemene deler wordt gevonden. Ook het bedenken voor welke lijst de iterator verlaagd moet worden, vereist enig inzicht. Mochten veel leerlingen moeite hebben bij deze opdracht dan kan het tekenen van de lijsten op het bord, met waardes en indices een waardevolle tussenstap zijn.

```
1  ggd
2  invoer : A, B
3  uitvoer: GGD
4
5  voor elk GETAL tussen 1 en A
6    als GETAL een deler van A is
7      voeg GETAL toe aan DELERS_A
8
9  voor elk GETAL tussen 1 en B
10   als het GETAL een deler van B is
11     voeg GETAL toe aan DELERS_B
12
13 begin onder aan lijsten DELERS_A en DELERS_B
14   als de waardes gelijk zijn
15     geef GGD
16   als de waarde uit DELERS_A groter dan die uit ···
      ···DELERS_B is
17     ga 1 omhoog in de lijst DELERS_A
18   als de waarde uit DELERS_A kleiner dan die uit ···
      ···DELERS_B is
19     ga 1 omhoog in de lijst DELERS_B
```

Programma 1: *Pseudocode voor* ggd

Bij opdracht **P7** moeten de leerlingen zich realiseren dat deze lijst geen einde heeft. Dit is ook het grootste risico voor fouten in de code (oneindige lus). Mocht een leerling een oneindige lus creëren, laat hem of haar dan in elke iteratie van de lus iets printen. Dit zal ze helpen te beseffen wat er aan de hand is. Help ze doen inzien dat $A * B$ altijd een gemeen veelvoud is, en dat daar voorbij zoeken dus niet nodig is.

Opdracht **P8** is vergelijkbaar met **P4**. Het kan de moeite waard zijn om leerlingen te wijzen op de overeenkomsten mochten ze er niet uitkomen. De leerlingen die er wel uitkomen wordt gevraagd hierover na te denken in vraag **P11**.

Opdracht **P12** is vooral bedoeld om de snelle leerlingen iets te doen te geven. Het maken van een goede gebruikersinterface is belangrijk maar daar dient deze les niet voor.

```
1  kgv
2  invoer : A, B
3  uitvoer: KGV
4
5  voor elk GETAL tussen 1 en B
6    voeg GETAL * A toe aan VEELVOUDEN_A
7
8  voor elk GETAL tussen 1 en A
9    voeg GETAL * B toe aan VEELVOUDEN_B
10
11 begin boven aan lijsten VEELVOUDEN_A en VEELVOUDEN_B
12   als de waardes gelijk zijn
13     geef KGV
14   als de waarde uit DELERS_A kleiner dan die uit ···
      ···DELERS_B is
15     ga 1 omlaag in de lijst DELERS_A
16   als de waarde uit DELERS_A kleiner dan die uit ···
      ···DELERS_B is
17     ga 1 omlaag in de lijst DELERS_B
```

Programma 2: *Pseudocode voor* kgv

3.2 Jaar 2

Gemiddelde, Mediaan en Modus

De leerlingen moeten een lijst kunnen laden om deze opdrachten op uit te voeren (**P13a**). Het kan zijn dat de leerlingen wat hulp nodig hebben om in te zien dat een lijst bestaande uit meerdere elementen toch één parameter kan zijn. Het is belangrijk dat de leerlingen het itereren over een lijst goed onder de knie krijgen dat gaan ze dit hoofdstuk uitgebreid nodig hebben en oefenen.

```
1 gemiddelde
2 invoer : LIJST
3 uitvoer: de gemiddelde waarde van de lijst
4
5 voor elke waarde in LIJST
6   tel op bij SOM
7
8 GEMIDDELDE is SOM gedeeld door het aantal elementen in ···
    ···de lijst
9 geef GEMIDDELDE
```

Programma 3: *Pseudocode voor gemiddelde*

De leerlingen moeten eerst een aantal hulpfuncties implementeren (**P14** , **P15** , **P16(a,b,c)** . Het schrijven en vervolgens gebruiken van de hulpfuncties zou ze moeten stimuleren in toekomstige opdrachten te kijken of er nuttige bestaande functies zijn die ze kunnen (her)gebruiken.

In opdracht **P16(d, e, f)** schrijven ze vervolgens de code voor het vinden van de mediaan, en het testen van dat programma.

```
1 max
2 invoer : LIJST
3 uitvoer: MAX
4
5 MAX := eerste waarde uit de lijst
6 voor elk GETAL in LIJST (behalve de eerste)
7   is GETAL groter dan MAX?
8     MAX := GETAL
9 geef MAX
```

Programma 4: *Pseudocode voor max*

```
1 mindex
2 invoer : LIJST
3 uitvoer: GETAL (van het kleinste getal uit LIJST)
4
5 MIN := eerste waarde uit LIJST
6 INDEX := INDEX van de eerste waarde uit LIJST (vaak 0 ···
    ···of 1)
7 voor elk GETAL in LIJST (behalve de eerste)
8   is GETAL kleiner dan MIN?
```

```
9       MIN := GETAL
10      INDEX := huidige index in de lijst
11  geef INDEX
```

Programma 5: *Pseudocode voor mindex*

```
1  sort
2  invoer : LIJST
3  uitvoer: GESORTEERDE_LIJST
4
5  zolang LIJST niet leeg is
6    voeg de kleinste waarde van LIJST toe aan ···
       ···GESORTEERDE_LIJST
7    verwijder die waarde uit LIJST
8
9  geef GESORTEERDE_LIJST
```

Programma 6: *Pseudocode voor sort*

```
1  mediaan
2  invoer : LIJST
3  uitvoer: MEDIAAN
4
5  sorteer LIJST
6  is het aantal elementen in de lijst even?
7    tel de middelste 2 bij elkaar en deel door twee; dit ···
       ···is de mediaan
8    geef MEDIAAN
9  anders
10   de middelste waarde is de mediaan
11   geef MEDIAAN
```

Programma 7: *Pseudocode voor mediaan*

Een algoritme voor het vinden van de modus wordt in het boek niet gegeven. We willen de leerlingen uitdagen zelf over een algoritme na te denken maar verwachten dat dit nog een flinke uitdaging kan zijn. In opdracht P17 dagen we ze uit hierover na te denken. We verwachten dat dit onderwerp zich goed leent om klassikaal te behandelen. Hierbij kan het "denken" van de computer op het bord uitgeschreven worden en kan de klas collectief nadenken over de te zetten stappen.

Een punt van aandacht is ook de definitie van modus. Volgens het boek zijn er 0 of 1 modi (als er meerdere zijn dan heeft de lijst volgens het boek geen modus). Het is, voor het door ons voorgestelde algoritme, beter als de leerlingen accepteren dat een lijst wel meerdere modi kan hebben.

```
1  modus
2  invoer : LIJST
3  uitvoer: MODI
4
5  sorteer LIJST
```

```
6  houd bij:
7  welk getal tot nu toe het meest voor kwam, hoe vaak ···
       ···dat was
8  met welk getal je bezig bent, hoe vaak je die hebt gezien
9
10 voor elk GETAL in LIJST
11   is dit het getal waar je nu mee bezig bent?
12     aantal voorkomens van dit getal + 1
13     is het aantal voorkomens nu groter dan het vorige ···
       ···maximum?
14       dit getal is het nieuwe maximum, dit aantal ···
       ···voorkomens is het nieuwe maximum aantal voorkomens
15     is dit aantal voorkomens gelijk aan het vorige ···
       ···maximum?
16       dit getal is ook een modus voeg het toe aan de ···
       ···lijst van modi, het maximum aantal voorkomens ···
       ···blijft gelijk
17   anders
18     het getal waar je nu mee bezig bent is het getal ···
       ···uit de lijst
19     het aantal voorkomens is 1
20     is dit aantal voorkomens gelijk aan het vorige ···
       ···maximum?
21       dit getal is ook een modus voeg het toe aan de ···
       ···lijst van modi, het maximum aantal voorkomens ···
       ···blijft gelijk
22
23 geef MODI
```

Programma 8: *Pseudocode voor modus*

3.3 Jaar 3

De code voor de *abc*-formule is redelijk recht-toe-recht-aan. We verwachten dat de leerlingen (met inmiddels 3 jaar programmeer ervaring) hier wel uit zouden moeten komen.

abc-**Formule**

```
1  abc
2  invoer :
3  uitvoer:
4  vraag de gebruiker om A, B, C in de formule ax^2+bx+c
5
6  DISCRIMINANT is B*B-4*A*C
7
8  als DISCRIMINANT kleiner dan 0 is
9    zeg: er zijn geen oplossingen
10 anders, als de discriminant 0 is
11   zeg: er is 1 oplossing
12   OPLOSSING is (-B)/(2*A)
13   geef OPLOSSING
14 anders
15   zeg: er zijn 2 oplossingen
16   EERSTE_OPLOSSING is (-B - wortel(discriminant))/(2*A)
17   TWEEDE_OPLOSSING is (-B + wortel(discriminant))/(2*A)
18   zeg: Xl=EERSTE_OPLOSSING, Xr=TWEEDE_OPLOSSING
```

Programma 9: *Pseudocode voor abc-formule*

Appendix

A De ggd en het kgv

```
1  EXPORT ISDELER(A, B)
2  BEGIN
3
4    IF FLOOR(A) ≠ ABS(A) THEN
5      RETURN FALSE;
6    END;
7    IF FLOOR(B) ≠ ABS(B) THEN
8      RETURN FALSE;
9    END;
10   RETURN B MOD A == 0;
11 END;
```

Programma 10: *Script van het programma isDeler, geschreven in HPPL.*

```
1  EXPORT GGD(VAR1,VAR2)
2  BEGIN
3    LOCAL iterator1, iterator2, deler, delers1 := {}, ···
       ···delers2 := {};
4    FOR deler FROM 1 TO VAR1 DO
5      IF ISDELER(deler, VAR1) THEN
6        delers1 := CONCAT(delers1,{deler});
7      END;
8    END;
9
10   FOR deler FROM 1 TO VAR2 DO
11     IF ISDELER(deler, VAR2) THEN
12       delers2 := CONCAT(delers2,{deler});
13     END;
14   END;
15
16   iterator1 := SIZE(delers1);
17   iterator2 := SIZE(delers2);
18
19   WHILE iterator1 >= 0 AND iterator2 >= 0 DO
20     IF delers1(iterator1) == delers2(iterator2) THEN
```

```
21        RETURN delers1(iterator1);
22      END;
23      IF delers1(iterator1) > delers2(iterator2) THEN
24        iterator1 := iterator1 - 1;
25      ELSE
26        iterator2 := iterator2 - 1;
27      END;
28    END;
29  END;
```

Programma 11: *Script voor het berekenen van de* ggd *geschreven in HPPL.*

Merk op dat er in het KGV programma (Programma 12) nog "hulp" uitvoer staat (regels 8, 13 en 16) om de werking van het programma te zien. Het kan de leerlingen helpen inzicht te krijgen in hoe het programma werkt (en de computer "denkt") om zulke uitvoer te kunnen bestuderen.

```
1   EXPORT KGV(VAR1, VAR2)
2   BEGIN
3     LOCAL iterator1 := 1, iterator2 := 1, veelvoud, ···
        ···veelvouden1 := {}, veelvouden2 := {};
4     FOR veelvoud FROM 1 TO VAR2 DO
5       veelvouden1 := CONCAT(veelvouden1,{VAR1*veelvoud});
6     END;
7     PRINT();
8     PRINT(veelvouden1);
9
10    FOR veelvoud FROM 1 TO VAR1 DO
11      veelvouden2 := CONCAT(veelvouden2,{VAR2*veelvoud});
12    END;
13    PRINT(veelvouden2);
14
15    WHILE iterator1 <= VAR2 AND iterator2 <= VAR1 DO
16      PRINT("vergelijk " + veelvouden1(iterator1) + " en ···
        ···" + veelvouden2(iterator2));
17      IF veelvouden1(iterator1) == ···
        ···veelvouden2(iterator2) THEN
18        RETURN veelvouden1(iterator1);
19      END;
20      IF veelvouden1(iterator1) < veelvouden2(iterator2) ···
        ···THEN
21        iterator1 := iterator1 + 1;
22      ELSE
23        iterator2 := iterator2 + 1;
24      END;
25    END;
26  END;
```

Programma 12: *Script voor het berekenen van het* kgv*, geschreven in HPPL.*

Programma 13 en Programma 14 laten efficiënte varianten van de programma's zien. Ze gebruiken het algoritme van Euclides. Het bewijs hiervan zal waarschijnlijk nog niet in het bereik van deze leerlingen liggen.

```
1  EXPORT GGD_EUCLID(A, B)
2  BEGIN
3    IF a MOD b ≠ 0 THEN
4      RETURN GGD_EUCLID(B, A mod B);
5    ELSE
6      RETURN B;
7    END;
8  END;
```

Programma 13: *Efficiënt script voor het berekenen van de* ggd*, geschreven in HPPL.*

```
1  EXPORT KGV_EUCLID(A, B)
2  BEGIN
3    LOCAL KGV := A;
4    WHILE (KGV MOD B) ≠ 0 DO
5      KGV := KGV + A;
6    END;
7    RETURN KGV;
8  END;
```

Programma 14: *Efficiënt script voor het berekenen van het* kgv*, geschreven in HPPL.*

B Centrummaten

```
1  EXPORT MAX(list)
2  BEGIN
3    LOCAL max, iterator;
4    max := list(0);
5    FOR iterator FROM 1 TO SIZE(list) DO
6      IF list(iterator) > max THEN
7        max := list(iterator);
8      END;
9    END;
10   RETURN max;
11 END;
```

Programma 15: *Script voor het vinden van de maximale waarde in een lijst, geschreven in HPPL.*

```
1  EXPORT MINDEX(list)
2  BEGIN
3  LOCAL min, mindex, iterator;
4    min := list(1);
```

```
5     mindex := 1;
6     FOR iterator FROM 2 TO SIZE(list) DO
7       IF list(iterator) < min THEN
8         min := list(iterator);
9         mindex := iterator;
10      END;
11    END;
12    RETURN mindex;
13  END;
```

Programma 16: *Script voor het vinden van de index van de minimale waarde in een lijst, geschreven in HPPL.*

```
1   EXPORT SORT(list)
2   BEGIN
3     LOCAL sorted_list := {}, index;
4
5     WHILE( SIZE(list) > 0 ) DO
6       index := MINDEX(list);
7       sorted_list := CONCAT(sorted_list,{list(index)});
8       list := ···
        ···CONCAT(SUB(list,1,index),SUB(list,index,SIZE(list)));
9     END;
10    RETURN sorted_list;
11  END;
```

Programma 17: *Naïef script voor het sorteren van een lijst, geschreven in HPPL.*

```
1   EXPORT MEAN(list)
2   BEGIN
3     LOCAL sum := 0, iterator;
4     FOR iterator FROM 1 TO SIZE(list) DO
5       sum := sum + list(iterator);
6     END;
7     RETURN sum/SIZE(list);
8   END;
```

Programma 18: *Script voor het berekenen van het gemiddelde van een lijst, geschreven in HPPL.*

```
1   EXPORT MEDIAN(list)
2   BEGIN
3     LOCAL middle;
4     list := SORT(list);
5     IF SIZE(list) mod 2 == 0 THEN
6       // Even aantal elementen in de lijst
7       middle := SIZE(list)/2;
8       RETURN (list(middle)+list(middle+1))/2;
9     ELSE
```

```
10      // Oneven aantal elementen in de lijst
11      middle := (SIZE(list)+1)/2;
12      RETURN list(middle);
13    END;
14  END;
```

Programma 19: *Script voor het berekenen van de mediaan van een lijst, geschreven in HPPL.*

```
1   EXPORT MODE(list)
2   BEGIN
3     LOCAL maxOcc, currentOcc, maxVal, currentVal, index;
4     list := SORT(list);
5
6     maxOcc := 1;
7     maxVal := {list(1)};
8     currentOcc := 1;
9     currentVal := list(1);
10    FOR index FROM 2 TO SIZE(list) DO
11      IF ( list(index) == currentVal ) THEN
12        // Zelfde getal nog een keer aantal voorkomens + 1
13        currentOcc := currentOcc + 1;
14        IF currentOcc > maxOcc THEN
15        // Is dit het nieuwe maximum?
16          maxOcc := currentOcc;
17          maxVal := {currentVal};
18        ELSE
19          IF currentOcc == maxOcc THEN
20            // Is deze gelijk gekomen met een oud ···
      ···maximum, dan hebben we mogelijk meer modi, voeg ···
      ···deze toe
21            maxVal := CONCAT(maxVal, {currentVal});
22          END;
23        END;
24      ELSE
25        // Nieuwe waarde gevonden, reset de teller en de ···
      ···"huidige waarde"
26        currentVal := list(index);
27        currentOcc := 1;
28        IF currentOcc == maxOcc THEN
29          // Is deze gelijk gekomen met een oud maximum, ···
      ···dan hebben we mogelijk meer modi, voeg deze toe
30          maxVal := CONCAT(maxVal, {currentVal});
31        END;
32      END;
33    END;
34
35    RETURN maxVal;
36
```

```
37 END;
```

Programma 20: *Script voor het berekenen van de modus (of modi) van een lijst, geschreven in HPPL.*

```
1  EXPORT STATS(list)
2  BEGIN
3    LOCAL modi, index;
4    PRINT();
5    PRINT("Gemiddelde: " + MEAN(list));
6    list := SORT(list);
7    PRINT("Mediaan: " + MEDIAN(list));
8    modi := MODE(list);
9    FOR index FROM 1 TO SIZE(modi) DO
10     PRINT("Modus " + index + ": " + modi(index));
11   END;
12 END;
```

Programma 21: *Script voor het geven van een aantal statistieken van een lijst, geschreven in HPPL.*

C De *abc*-formule

```
1  EXPORT ABCD()
2  BEGIN
3    LOCAL A,B,C,D,X0,X1;
4    MSGBOX("Schrijf de formule om naar\n ax^2+bx+c\n en ···
       ···geef a, b en c.");
5
6    INPUT(A);
7    INPUT(B);
8    INPUT(C);
9
10   D := B*B-4*A*C;
11   PRINT();
12   PRINT("De discriminant is: " + D);
13
14   IF ( D < 0 ) THEN
15     PRINT("Er zijn geen oplossingen");
16   ELSE
17     IF ( D == 0 ) THEN
18       PRINT("Er is een oplossing");
19       X0 := (-B)/(2*A);
20       PRINT("x = " + X0);
21     ELSE
22       PRINT("Er zijn twee oplossingen");
23       X0 := (-B - √(D))/(2*A);
24       X1 := (-B + √(D))/(2*A);
```

```
25            PRINT("x0 = " + X0);
26            PRINT("x1 = " + X1);
27        END;
28    END;
29 END;
```

Programma 22: *Script voor het oplossen van* $ax^2+bx+c=0$ *(over* $\mathbb{R}$*), geschreven in HPPL.*