Dienst Curriculum & vorming
Team secundair onderwijs
Guimardstraat 1 - 1040 BRUSSEL
www.katholiekonderwijs.vlaanderen

Computationeel denken in leerplannen wiskunde D-finaliteit

2021-06-24

1 Inleiding

In de verschillende leerplannen wiskunde B, B+C', VB, VB+C' en C voor de 2de graad D-finaliteit is een leerplandoel opgenomen over computationeel denken. De concrete nummering van het doel hangt af van het leerplan.

LPD 1 De leerlingen ontwerpen algoritmes om problemen digitaal op te lossen.

★ Concepten van computationeel denken: decompositie, patroonherkenning, abstractie, algoritme

Organisatie, modellering, simulatie en digitale representatie van informatie

Debuggen

Principes van programmeren: opeenvolging, herhalingsstructuur, keuzestructuur

Ingebouwde functies

Elementen van programmeertalen: variabelen, datatypes, operatoren, parameters, condities, procedures of functies

Dit document heeft als bedoeling om meer inhoudelijke duiding te geven bij het doel, samen met inspirerende voorbeelden in de programmeertaal Python. Bij eventuele vragen over het document kan je terecht bij Filip.Cools@katholiekonderwijs.vlaanderen .

2 Verklaring van enkele begrippen in de afbakening

Hieronder worden enkele begrippen in de afbakening van het leerplandoel uitgelegd. Het is niet de bedoeling dat leerlingen deze begrippen kunnen definiëren. Belangrijk is dat ze inzicht hebben in de concepten en dat ze de concepten kunnen hanteren tijdens het computationeel denken.

2.1 Concepten van computationeel denken

- *Decompositie*: opdelen van een complex probleem in eenvoudigere deelproblemen.
- *Patroonherkenning*: door gelijkenissen te zien in verschillende problemen kunnen dezelfde oplossingsmethodes worden gebruikt. Door gelijkenissen te zien in algoritmes kunnen stukjes code zo worden geschreven dat ze verschillende keren kunnen worden gebruikt.
- *Abstractie*: tot de essentie of kern van de zaak komen door overtolligheden weg te laten om zo het probleem eenvoudiger te maken.
- *Algoritme*: een reeks instructies/stappen die leiden naar een beoogd doel vanuit een gegeven begintoestand.

2.2 Debuggen

- *Debuggen* (of foutopsporing): fouten in algoritmen kunnen eerst worden opgespoord (bv. door het algoritme te testen) en nadien worden verbeterd. Er kan ook gebruik gemaakt worden van de mogelijkheden van de programmeeromgeving zelf.

2.3 Principes van programmeren

- *Opeenvolging* (of sequentie): een algoritme bestaat uit een reeks van opeenvolgende instructies, die ook in die volgorde worden uitgevoerd.
- *Herhalingsstructuur* (of iteratie): voor situaties waarbij een aantal instructies een aantal keer moeten worden uitgevoerd, zijn er zijn structuren voorhanden die ervoor zorgen dat de instructies zelf niet herhaald of gekopieerd moeten worden in het programma. Denk hierbij aan een “for”-lus (voor situaties waarbij het aantal herhalingen op voorhand vast ligt) of een “while”-lus (voor situaties waarbij het aantal herhalingen afhankelijk is van een bepaalde voorwaarde).
- *Keuzestructuur* (of selectie): een algoritme kan een reeks instructies uitvoeren als aan bepaalde voorwaarden voldaan is. Denk hierbij aan een “als … dan …”-structuur.

2.4 Elementen van programmeertalen:

- *Variabelen*: informatie wordt opgeslagen onder de vorm van een variabele. Variabelen hebben een naam, zodat via die naam de informatie kan worden toegekend, opgeroepen of veranderd.
- *Datatypes*: een variabele is van een bepaald type. Dat type bepaalt welk soort informatie in de variabele kan worden opgeslagen en welke bewerkingen/operaties er kunnen worden mee uitgevoerd.
- *Functies*: een structuur die toelaat om op een gestructureerde manier operaties uit te voeren (vaak op een input) en een output terug te geven. *Ingebouwde functies* zijn ingebouwd in de programmeertaal en moet je dus zelf niet meer schrijven. Je kan ook zelf functies schrijven.
- *Procedures*: vergelijkbaar met functies, maar i.t.t. functies geven ze geen waarde of output terug.
- *Parameters*: in de definiërende code van een functie of procedure met een input wordt een speciale soort van variabelen gebruikt om de input voor te stellen. Deze plaatsvervangers zijn de parameters. Bij het oproepen van de functie of procedure worden dan concrete waardes voor de parameters meegegeven.
  Voorbeeld: bij een gegeven functie $f(x)$ is x de parameter.
- *Operatoren*: vergelijkbaar met functies, maar ze verschillen qua syntaxis. Operatoren worden ook niet echt opgeroepen, maar maken deel uit van de programmeertaal zelf.
  Voorbeelden: bewerkingen met twee getallen (bv. a + b) of logische operatoren (bv. a or b).
- *Condities*: in een programma kunnen bepaalde operaties worden uitgevoerd als aan een voorwaarde voldaan is. Die voorwaarde is de conditie.

3 Voorbeelden in de programmeertaal Python

3.1 Enkele veel voorkomende commando's in Python

Verschillende datatypes

	Betekenis van het datatype	Toekenning aan een variabele	Omzetting vanuit ander datatype (indien mogelijk)
Integer	geheel getal	bv. a = 5 en a = -10	int(.)
Float	decimaal getal	bv. a = 3.14	float(.)
Boolean	waarheidswaarde	bv. a = True en a = False	bool(.)
String	reeks van tekens na mekaar	bv. a = "Hello, world!"	str(.)

Algemene functies:

print(a)	tonen/afprinten van de waarde van een variabele
print(a,…,b)	Tonen/afprinten van verschillende zaken na mekaar (met spatie tussen)
input(<<vraag>>)	waarde voor variabele vragen met als resultaat String bv. maand = input("In welke maand ben je geboren?")

Vergelijken van twee variabelen van hetzelfde datatype:

a == b	gelijk aan (resultaat TRUE/FALSE van type Boolean)
a != b	niet gelijk aan (resultaat TRUE/FALSE van type Boolean)

Operatoren en ingebouwde functies voor specifieke datatypes:

- bij numerieke gegevens (datatype Integer of Float):

a + b	optelling of som
a - b	aftrekking of verschil
a * b	vermenigvuldiging of product
a / b	deling of quotiënt
a ** b	macht
a % b	rest bij deling
a // b	quotiënt bij deling met rest
a > b	groter dan (resultaat TRUE/FALSE van type Boolean)
a < b	kleiner dan (resultaat TRUE/FALSE van type Boolean)
a >= b	groter dan of gelijk aan (resultaat TRUE/FALSE van type Boolean)
a <= b	kleiner dan of gelijk aan (resultaat TRUE/FALSE van type Boolean)
a += b	variabele a met b laten toenemen; alternatief voor toekenning a = a+b
round(a,n)	afronden tot op n decimalen (met default-waarde 0 voor n)

- bij booleaanse uitdrukkingen (datatype Boolean):

a and b	logische 'en'
a or b	logische 'of'
not a	logische 'niet'

- bij strings (datatype String):

a + b	aan mekaar plakken / concatenatie
a < b (e.d.)	alfabetisch vergelijken (resultaat TRUE/FALSE van type Boolean)

Keuzestructuur:

if <<booleaanse uitdrukking>> : <<taken 1>> elif << booleaanse uitdrukking>> : <<taken>> else : <<taken>>	als-dan-anders-structuur (mag ingekort worden tot enkel if-gedeelte of if-else-gedeelte)

Herhalingsstructuren:

while <<booleaanse uitdrukking>>: <<taken>>	herhaling tot zolang de voorwaarde is voldaan
for <<parameter>> in <<uitdrukking>>> : <<taken>> *Voorbeelden:* for a in range(n) : … for a in range(n,m) : … for a in <<type string>> : …	herhaling over een parameter a gaat van 0 t.e.m. n-1 a gaat van n t.e.m. m-1 in stappen van 1 a gaat de tekens van een string af

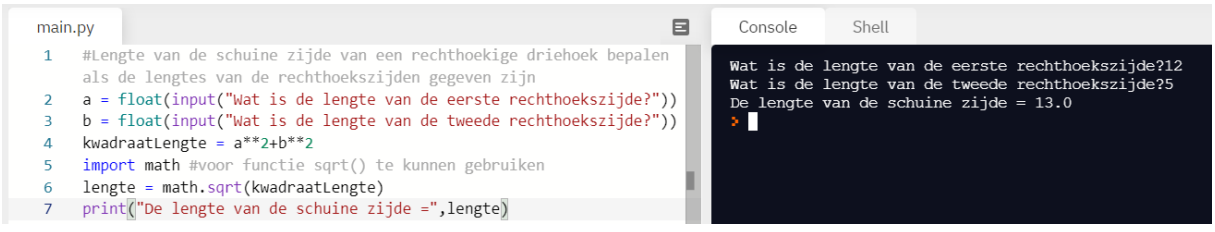
Definiëren en oproepen van functies:

def <<naam functie>>(<<invoer>>): <<taken>> (return <<uitvoer>>)	functie definiëren met bepaalde invoer en uitvoer
<<naam functie>>(<<invoer>>)	uitvoeren van een functie met bepaalde invoer

3.2 Enkele voorbeelden van algoritmes

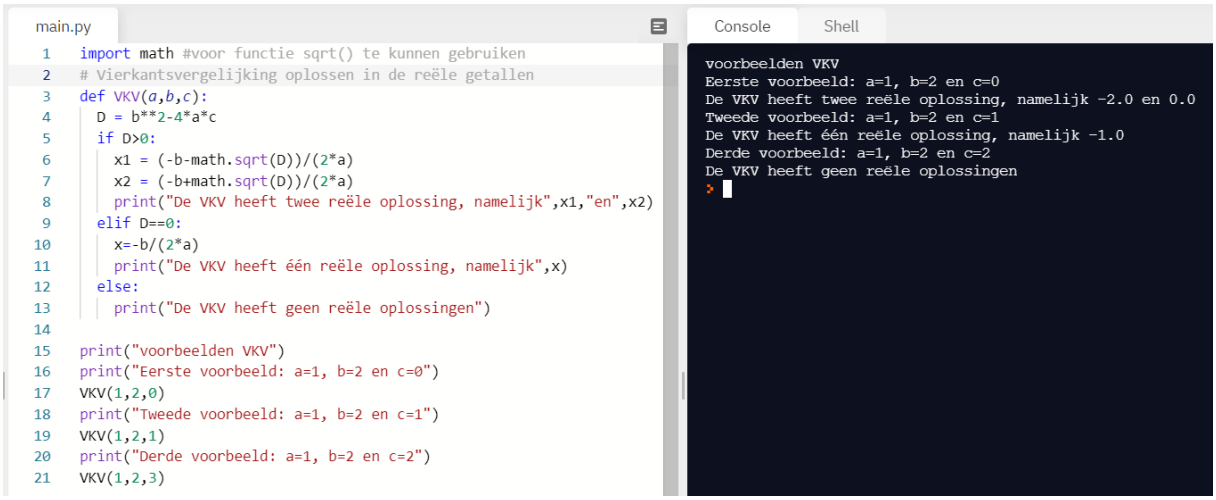
In de eerste twee voorbeelden gebruiken we de input-functie om de waarden van variabelen op te vragen en werken we daarna direct verder met die opgeslagen waarden. In de andere voorbeelden definiëren we eerst een functie om deze daarna toe te passen. Beide opties zijn steeds mogelijk.

3.2.1 Stelling van Pythagoras



3.2.2 Som der cijfers

3.2.3 *Vierkantsvergelijking oplossen*



3.2.4 *Grootste gemene deler via het algoritme van Euclides*

3.2.5 *Drie getallen sorteren van klein naar groot*

3.2.6 *Tekentabel van een eerstegraadsfunctie*

main.py

```
# tekentabel of -schema bepalen van een eerstegraadsfunctie met voorschrift f(x)=ax+b
def tekentabelEerstegraadsfunctie(a,b):
    nulpunt = round(-b/a,2)
    print("  x |  ",nulpunt,"  ")
    print("----------------")
    if a>0:
        print("f(x)| -   0   + ")
    else:
        print("f(x)| +   0   - ")

print("Eerste voorbeeld: f(x)=3x+2")
tekentabelEerstegraadsfunctie(3,2)
print("")
print("Tweede voorbeeld: f(x)=-x+1")
tekentabelEerstegraadsfunctie(-1,1)

```

Console Shell

```
Eerste voorbeeld: f(x)=3x+2
  x |   -0.67
----------------
f(x)| -   0   +

Tweede voorbeeld: f(x)=-x+1
  x |   1.0
----------------
f(x)| +   0   -
>
```

3.2.7 *Gepast betalen in euromunten*

main.py

```
# Functie voor een bedrag gepast te betalen met zo weinig mogelijk euromunten en -biljetten
def GepastBetalen(bedrag):
    for munt in 200, 100, 50, 20, 10, 5, 2, 1:
        aantal = 0
        while bedrag >= munt :
            aantal = aantal + 1
            bedrag = bedrag - munt
        if aantal > 0 :
            print(aantal,'x',munt)
# Een voorbeeld
GepastBetalen(146)
```

Console Shell

```
1 x 100
2 x 20
1 x 5
1 x 1
>
```