(Samen)
H6 controle op grammatica en spelling: https://bjoc-nl.github.io/hoofdstuk6.html?

H8 controle op grammatica en spelling: https://bjoc-nl.github.io/hoofdstuk8.html?

KLAAR
(Sterre) vertalen omlijsting:
woordenlijst:
https://github.com/BJOC-NL/bjoc-nl.github.io/blob/master/shared/glossary%20%26%20decisions.md

https://bjoc-nl.github.io/

wij moeten H8 en 6 op spelling checken: rechtsboven is een knop waarmee je deze aanpassingen voor kan stellen.

Lijst met woorden om door te geven (Weet niet zeker hoe te vertalen):

- recursive recursief
- recursion
- *College Board* AP CS Principles
- computational
- conforming
- CS principles
- slide decks
- visual programming
- control abstraction
- data abstraction
- action scripts
- encryption encryptie
- *Creative commons BY-NC-SA license.*
- *count change*
- *vee*
- *bug*

Vertaling:

De auteurs van het BJC curriculum vinden dat programmeren, van alle activiteiten die men kan doen, een van de meest voldoening gevende activiteiten is. Over het algemeen is het erg leuk om te doen (zolang het niet frustrerend is wanneer je een bug niet kan vinden), maar dat is wat Seymour Papert “Lastig leuk” noemt, geest verbredend en, omdat de autoriteit de computer is en niet de docent, de leerling hoeft zichzelf niet in allerlei bochten te wringen om de stof te leren. Het is een spel waar je vaardigheid in moet krijgen, zoals schaken, maar dan zonder het competitieve aspect en met andere nuttige resultaten naast het leren programmeren zelf. Dit is de beste reden om informatica als vak te leren, en wij willen dat **alle kinderen**, niet alleen diegene die aan het stereotype ‘nerd’ voldoen, de vreugde van programmeren kunnen ervaren.

We vinden ook dat computerprogramma's (en niet alleen de afbeeldingen die zij kunnen creëren) mooi kunnen zijn. Ja, die programma’s kunnen ook lelijk zijn, als ze lange opeenvolgingen of taak verklaringen zonder structuur zijn. Maar goede programmeurs ontwikkelen een gevoel voor programmeer esthetiek. Dit is een van de redenen waarom het belangrijk is om *recursion* in het curriculum mee te nemen: een *recursive* programma kan een complex, gedetailleerd process genereren uit een klein stukje code, wij herinneren deze openbaring als het moment waarop wij de schoonheid van programma’s ontdekte.

Het is niet makkelijk om schoonheid en vreugde te bewaren wanneer lessen vertaalt moeten worden voor een curriculum dat door velen bruikbaar moet zijn. Te veel aanpassingen en het plezier van het project is weg; te weinig aanpassingen en de leerling voelt zich hulpeloos en incompetent. Dat is waarom de oorspronkelijke auteurs van de Universiteit van Californië, Berkeley, partners zijn geworden met de middelbare school curriculum experts van EDC (Education Development Center). Probeer het curriculum uit om er achter te komen wat jij er van vindt.

Informatica *Principles*

Het *College Board* AP CS Principles curriculum raamwerk is georganiseerd rondom zeven “grote ideeën” (dingen om te leren) en zes “*Computational* denkwijze oefeningen” (dingen om te doen):

1. Creativiteit
2. Abstractie
3. Data en informatie
4. Algoritmes
5. Programmeren
6. Het internet
7. Impact op de wereld

1. Verbindend programmeren
2. Het creëren van *computational* artefacten

3. Abstraheren
4. Analyseren van problemen en artefacten
5. Communiceren
6. Samenwerken

Elk *conforming* curriculum moet al deze punten onderwijzen, maar curricula mogen verschillen in de hoeveelheid nadruk die op de punten worden gelegd. (De taartpunten in de grafieken zijn niet daadwerkelijk kwantitatief, het is een suggestie van hoe BJC is opgebouwd.)

Er is veel overlapping tussen de Ideeën en toepassingen. creativiteit is een idee, maar creëren is een toepassing. Je moet de eerste taartdiagram zien als de texten gegeven in het lesmateriaal van het curriculum, en de tweede taartdiagram moet je zien als hoe er verwacht wordt dat de leerlingen hun tijd gaan besteden. De cirkels in het midden van de taartdiagrammen wijzen op het idee dat programmeren en creëren het centrum van BJC vormen, en overlappen met alle andere punten.

Als eerste, de Grote Ideeën. BJC legt een zware nadruk op het punt **Programmeren**. Wij geloven dat Snap!, de programmeertaal die we gebruiken, ons in staat stelt om een diverse samenstelling van beginners te bereiken, die in de eerste instantie ook niet geïnteresseerd in programmeren hoeven te zijn, omdat het het gemak van *visual* programmeren combineert met de expressieve kracht die vooraf alleen gevonden kon worden in geavanceerde op tekst gebaseerde talen. (meer hierover onderaan de pagina.) We gaan ver voorbij de CS *Principles* vereisten, met de geavanceerde technieken van *recursion* en functies van hogere ordes.

We zien **Abstractie** als het centrale idee van informatica, en we leggen de nadruk op het gebruik van abstractie in context van programmeren. Dit omvat zowel het *control abstraction* om programmeer patronen te generaliseren, als *data abstraction* om de implementatie van een abstract data type te isoleren van zijn gebruik.

Onze tweede grootste nadruk ligt op **de impact op de wereld**, de sociale betrekkingen van programmeren. We gebruiken lezingen en classroom discussies om verschillende aspecten van dit onderwerp te ontdekken in ieder hoofdstuk van de lessen. Details over de onderwerpen en leerdoelen staan beneden.

Op twee van de vier onderwerpen die overblijven wordt minder nadruk op gelegd dan de anderen. Leerlingen passen **Creativiteit** toe in hun programmeer projecten, maar we bespreken het punt niet zoveel als de andere onderwerpen. Op dezelfde manier ontwikkelen leerlingen **Algoritmes** tijdens het programmeren, maar er wordt niet vaak over algoritmes gesproken als een apart onderwerp, behalve dan als het analyseren van algoritmes en asymptotische volgordes van groei wordt onderwezen.

De laatste categorie met onderwerpen, **Data** en **Het internet**, zijn belangrijk en worden uitvoerig behandeld, maar waar mogelijk neemt deze behandeling de vorm aan van

activiteiten met programmeren in plaats van, bijvoorbeeld, het gebruik van commerciële database software.

Nu, over te toepassingen: De zin “*Computational* Artefacten” is bedoeld om op video's, *slide decks,* blogs, programma’s, muziek, spreadsheets -- en wat je nog meer met de computer kan creëren te wijzen. Het Creëren van, vinden wij, is een van de meest belangrijke toepassingen, maar de meest belangrijke artefacten zijn de computerprogramma's! Om preciezer te zijn, als een leerling een programma schrijft voor een spel, bijvoorbeeld, dan word creatie op twee verschillende manieren beoefend. De leerling is geïnteresseerd in het maken van een spel, en past zijn creativiteit toe in het design van de spel. Maar zij maakt de spel door een programma te schrijven, en ook hierbij wordt van haar creativiteit vereist in de structuur van het programma dat ze schijft. Het laatste is het echte informatica.

Net als dat het idee van abstractie centraal staat in, en onafscheidelijk is van, het idee van programmeren, staat het beoefenen van abstractie centraal bij programmeren. We moedigen leerlingen constant aan om de lagen van abstractie toe te passen in de structuur van een programmeer project.

“Verbindend programmeren” kan betekenen dat je het aan hobby's of aan industrie verbindt, of misschien zelfs wetenschap. Maar nog belangrijker, voor ons, leerlingen verbinden programmeren aan zijn sociale toepassingen, onze tweede grootste focus.

Bijna net zo belangrijk als het analyseren van programma’s: *debugging*, het voorspellen van het gedrag van iemands code, en in acht nemen van de efficiëntie. Maar we benadrukken dat deze vaardigheid niet het uiteindelijke doel is; uiteindelijk is het de bedoeling dat men op zijn of haar werk programma’s kan maken. We zijn lang niet zo geïnteresseerd in het analyseren van “artefacten” als we zijn in het analyseren van programma’s.

De twee laatste toepassingen zijn belangrijk, maar bij BJC ligt er minder de nadruk op. Zoals de meeste CSP curricula, gebruiken wij *pair programming*, zodat leerlingen constant overleggen met hun partners en het werk verdelen. Aan het begin van de les onderwijzen we het process van *pair programming*, en in de discussie van sociale toepassingen van programmeren moeten leerlingen hun ideeën overleggen, maar communiceren en samenwerken zijn niet exclusieve leerpunten voor het vak informatica.

Visual **Programmeren**

Snap!, de programmeertaal ontworpen om deze les te ondersteunen, start met het Scratch design van sleep-en-laat-los blokken die primitieve mogelijkheden representeren. Scratch wordt geroutineerd geleerd door achtjarigen die het helemaal zelf doen, het is dus niet zo intimiderend en het is niet moeilijk om mee te beginnen. De vormen en kleuren van de blokken herinneren de gebruikers eraan in wat voor categorieën de blokken zitten.

De hapblok is een lus, en omsluit ook visueel de code die herhaald moet worden. De groene blokken gaan over tekeningen en de blauwe blokken over bewegen.

Maar omdat Scratch ontworpen is om gebruikt te worden door achtjarigen, hebben de ontwerpers van het programma cruciale onderdelen voor het leren van informatica weggelaten. Bij Snap! zijn deze ontbrekende onderdelen toegevoegd, zonder de zorgvuldig ontworpen visuele metaforen te verstoren die het makkelijker maken om het programma te begrijpen.

Voor *control abstraction* kunnen de gebruikers van Snap! hun eigen blokken bouwen, hierbij zijn functies en *action scripts* inbegrepen.

deze functie is essentieel voor het grote idee van abstractie, maar zorgt er ook voor dat we recursion kunnen onderwijzen, waardoor een klein programma een complex resultaat kan opleveren.

Omdat blokken van Snap! andere blokken of scripts als invoer kunnen gebruiken, en omdat de notatie voor anonieme functies erg simpel is, kunnen we een krachtigere vorm van *control abstraction* van functies van hogere ordes onderwijzen.

Deze functie, eersteklas procedures, maakt Snap! veel vermogend en expressiever van de meeste op tekst gebaseerde programmeertalen. De visuele representatie maakt de data van de procedure concreter voor leerlingen, zoals in deze *lijst van procedures:*

Als laatste, omdat lijsten de eerste klas data zijn in Snap!, kunnen we abstracte data types bouwen en deze gebruiken in grotere data structuren, zoals deze driehoek:

Voor leerlingen die erop staan om te programmeren in tekst, Snap! voorziet toegang tot een Javascript omgeving waarin dit mogelijk is:

(Op de afbeelding is maar 1 regel te zien, maar er zit geen limiet aan de lengte of complexiteit van de Javascript functie die gedefinieerd wordt door het blok.) Met deze functie krijg je het beste van beide werelden, met respect voor het (zinloze, vinden wij) argument over blokken talen tegenover tekst talen. We leren geen Javascript bij BJC, maar het is een mogelijkheid voor extra activiteit, mocht dat nodig zijn.

Daarnaast, omdat Snap! op elke moderne browser beschikbaar is, kunnen leerlingen mobiele apps maken voor IOS of Android door een snelkoppeling te maken naar de URL van het project op het bureaublad van het betreffende apparaat. Vanaf een browser kan geen gebruik gemaakt worden van de specifieke informatie van de telefoons, zoals contactlijsten en GPS locaties (versies voor computers die geen deel uitmaken van een netwerk worden op dit moment gemaakt), maar dingen zoals videoSpel projecten werken prima.

Snap! kan ook verbonden worden met bepaalde robots en sensoren (Finch, Hummingbird, Sphero, Lego NXT, Wiimote, LEAP Motion, Arduino, etc.) door kleine applicaties te downloaden op de computer in gebruik.

Maatschappelijke invloed van programmeren

Hier zijn de onderwerpen die worden behandeld in de hoofdstukken van BJC:

- Privacy en zoekmachines
- VideoSpellen en geweld
- Bezitten van ideeën (Copyrights en patenten)
- *Encryption*
- Innovatie
- Computers en Gemeenschap
    - Sociale netwerken
    - online pesten
- Computers en oorlog
- Computers en werk

In elk van deze onderwerpen is ons doel om alle vormen van vooroordelen te ontwijken; We kijken naar alternatieve perspectieven op het onderwerp. Bijvoorbeeld, we nemen niet aan dat het kapen van gecopyright werk slecht is; omdat veel leerlingen dit ook doen, we proberen een andere kant van het verhaal te bekijken waarom het wel oke zou zijn, en dan overwegen we andere manieren om artiesten en schrijvers te ondersteunen. Zowel de stof voor de leerlingen en de gids van de docent wijzen op uitdagende lezingen over ieder onderwerp.

Over het algemeen hopen wij dat leerlingen optimistisch kijken naar de voordelen van technologie, maar daarnaast ook kritisch nadenken over *specifieke* technologie. We herinneren leerlingen er ook aan dat beslissingen over hoe een nieuw soort technologie gebruikt wordt worden gemaakt door mensen, zijzelf daarbij inbegrepen als ze besluiten van informatica hun carrière te maken, dus ze zouden zich niet hulpeloos moeten voelen bij confrontatie met een verondersteld technische noodzaak.

We gebruiken het geweldige Blown to Bits als tekstboek voor dit onderdeel van de cursus. (Er is geen tekstboek voor de technische onderdelen van de cursus, naast de online materialen die wij verzorgen.) Het boek is bedoeld voor volwassen lezers, en zijn voornamelijk lastig voor ESL leerlingen, dus gebruiken we korte fragmenten en discussiëren we over alternatieve presentaties in de Gids van de docent. Net als de rest van het lesmateriaal is ook dit boek gratis online beschikbaar, met een *Creative commons BY-NC-SA license.*

Het boek is een aantal jaar oud nu, dus we vullen het aan met recentere lezingen, zoekend naar onderwerpen die relevant zijn voor de leerlingen. Leerlingen kiezen hun eigen onderwerpen uit waarmee ze verder de diepte in willen; een van de manieren waarop de cursus aantrekkelijk is voor een breed publiek. We starten lessen met “Programmeren in het nieuws,” waarbij artikelen van de krant van die ochtend worden behandeld, een met een positieve ondertoon en een over een probleem. Na wat door de leraar gekozen voorbeelden om het idee te illustreren, moedigen we de leraren aan om de leerlingen afwisselend artikelen mee te laten nemen.

Over Snap!

Snap! is een visuele programmeertaal gebaseerd op Scratch (MIT Media Lab), maar uitgebreid om geavanceerde computer ideeën te ondersteunen, voornamelijk *recursion* en op functie gebaseerd programmeren.

- Snap! Hoofdpagina
- Snap! 4.0 Handleiding
- Run Snap! Live
- Snap! Projecten:
    - *Vee*
    - Boom Demo
    - Live Boom
    - Palindroom oefening
    - *count change*
- Meld een *bug* in Snap!

Lijst met aangebrachte veranderingen:

H6:

- **homepage**
  - bij softwaredomein een spatie ertussen --> 'Het Software Domein'
- **Les 1**
  - dichter bij → staan dichterbij
  - Verschillende mensen kunnen dit diagram iets anders tekenen → Dit diagram is op verschillende manieren te tekenen
  - en gebruikt, zijn software → geen komma
  - ultralage → ultra lage
  - (map , keep en combine →) op het eind
  - Andere veel voorkomende taken, weten bijvoorbeeld waar de muisaanwijzer zich bevindt, rekenen op grote aantallen of geluid afspelen, hebben ook complexe onderdelen. → Andere veelvoorkomende taken, zoals weten waar de muisaanwijzer zich bevindt, rekenen met grote aantallen of geluid afspelen, hebben ook complexe onderdelen.
  - om al die code → om dezelfde code
  - bibliotheken → bibliotheken (libraries)
  - dat dat blok gebruiken → die dat blok gebruikt
  - ook wel Operating System (OS) in het Engels → (ook wel Operating System (OS) in het Engels)
  - Everyone talks about (...) of the digital domain. → dit moet worden verwijderd, het is de engelse versie die er na de vertaling nog een keer in staat.
  - Babbage was mainly (...) time was clockwork—gears. → verwijderen, engels.
  - stop zetten → moest stopzetten
  - werkten→ werkte
  - The abstraction of software (a program stored in the computer's memory) is what makes a computer usable for more than one purpose. → De abstractie van software (een programma ingebouwd in het geheugen van een computer) is wat een computer bruikbaar maakt voor meer dan een doel.
  - , niet Java, C of Snap!, of iets anders. → geen Java, C, Snap! of iets dergelijks.
  - apparat → apparaat
  - Een processor is een IC maar niet alle processors zijn ICs;
  - electronice → elektronische
  - (...) berekeningen uit te voeren zijn interessanter. → interessanter dan de circuits voor het geheugen.
  - drie mogelijke uitvoerwaarden heeft, omdat 1 + 1 = 2, wat noch 0 noch 1.
  - or → of
  - , ongewenste veranderingen in spanningen, → (ongewenste veranderingen in spanningen)
  - vanwege vele reden → door vele redenen;
  - Dit is de reden waarom computers nullen en enen gebruiken: → daarom gebruiken computers enen en nullen
- **Les 2**

- bist → bits
- 2017 → 2019
- programmere → programmeren
- regels → regel
- ziet- Maar waarom niet?
- ziet- Maar waarom niet? → ziet, maar waarom is dat het geval?
- of iets anders zelfs misschien..? → of misschien zelfs nog iets anders?
- werken → werken.
- je de processor van je computer. → de processor van je computer.
- gebruikten → gebruiken
- tegenwoord → tegenwoordig
- vrij onintuïtieve → niet intuïtieve
- geen → een
- 37de → 37ste
- de → zich te
- daarentegen is moeilijk mo → daarentegen, zijn moeilijk om
- for → voor
- het variabele → de variabele
- VInd → vind
- op door de door de → op door de
- totdat → Totdat
- schrijve → schrijf
- Now, read the number off --> Lees nu dit nummer
- avn die die → die
- eht → het
- hexadecimal → hexadecimaal
- 128,0,255 → 0, 0, 0
- red 0, green 149, and blue 235
  red 128, green 90, and blue 0
  red 163, green 0, and blue 84
  → red = rood, green = groen, blue = blauw and = en
- een → Een

- **Les 3**
  - limit → limiet
  - wt → wat
  - innovaties → zorgen voor innovaties
  - echt → echte

H8:

- **homepage:**
    - Je zal een aantal van de meest belangrijke ideeën leren in de informatica
        - Je zal een aantal van de meest belangrijke ideeën van informatica leren.
    - daarnaast zal je creatief moeten zijn
        - daarnaast zal je creatief moeten nadenken
- **les 1:**

    pagina 1:
    - Hoeveel lijnen zitten er in een boom op ieder niveau
        - Hoeveel lijnen zitten er op ieder niveau in de boom?
    - *Hoeveel lijnen zitten er op een niveau meer dan op het vorige niveau*
        - *Hoeveel lijnen meer zitten er op ieder volgend niveau?*
    - *die van die voor de*
        - *de code voor de*
    - *voor*
        - *voor de*
    - *riepen*
        - *riepen de*
    - *weer*
        - *weer de*
    - *in*
        - *in de*

    Pagina 2:
    - *gebruikt*
        - *gebruik*

    Pagina 3:
    - *het product van de getallen van 1 tot n*
        - *een product met een variabele waarde n*
    - *boom*
        - *de boom*
    - *AllendiezichinNederlandbevinden*
        - *-*
    - *map*
        - *de map*
    - *oplossen.*
        - *oplossen.)*

- ***Les 2:***

    Pagina 1:
    - *Dit zouden dan ook getallen kunnen zijn of nog iets totaal anders dan*
        - *Dit zouden dan bijvoorbeeld ook getallen kunnen zijn, of iets anders dat*
    - *getalen*
        - *getallen*
    - *dat*

    - *zodat*
- *het*
    - *dit*
- *rapporteert een nieuwe lijst*
    - *die een nieuwe lijst rapporteert*

  Pagina 2:
- *Selection Sort*
    - *Selectief sorteren*
- *heb je code nodig voor een basisgeval*
    - *heb je een code nodig voor het basisgeval.*
- *Write code die de index van het voorste item vindt*
    - *Schrijf een code die de index van het voorste item vind.*
- *BOuw*
    - *Bouw*
- *bestand met de populariteit van namen in 2014*
    - *bestand met de populairste namen van 2014*
- *die start met een*
    - *gebruik alleen de namen die starten met de*

  Pagina 2:
- *Partition Sort*
    - *Verdeeld sorteren / het verdeeld sorteren*
- *code nodig voor een basisgeval en code die partition sort volgt.*
    - *de code nodig voor het basisgeval en de code die het verdeeld sorteren uitvoert.*
- *selection sort*
    - *selectief sorteren*
- *een algoritme dat over het algemeen langzaam is*
    - *de langzaamste algoritmes*
- *kan*
    - *kunnen*
- *Je kan ook een lange lijst van namen hier gebruiken*
    - *Je kan hier ook een lange lijst van namen gebruiken.*

- ***Les 3:***

  Pagina 1:
- *,*
    - *-*
- *, rij en kolom en*
    - *; rij en kolom, en daarmee*

  Pagina 2:
- *memoisatie*
    - *memorisatie*

  Pagina 3:
- *die het getal weergeeft in binair*
    - *het getal binair weergegeven*

  Pagina 4:

- *getalteken*
    - *getalketen*
- *recursive case, the rightmost binary digit is the remainder when dividing the number by 2. The other digits come from a recursive call on the quotient when dividing by 2. The combiner is*
    - -

Pagina 5:

- .
    - *te gebruiken.*
- *kommen met 3 bolletjes*
    - *mogelijke combinaties van kommen met 3 bolletjes*
- *met een extra invoer voor het aantal bolletjes*
    - *de extra invoer n staat voor het aantal bolletjes.*

Pagina 6:

- *Gegeven een set dingen, dan bevat een subset nul of meer van de elementen van die set, zonder dubbele elementen. Net zoals bij de kommen met ijs maakt de volgorde niet uit.*
    - *Een gegeven set 'dingen', waarbij iets nul, een of meerdere van de elementen uit die set kan bevatten, zonder een van de dingen meerdere keren voorkomt, heet een subset.*
- *naar de Subset Hints page gaan voor wat help*
    - *‘Hints voor subsets’ bekijken voor wat hulp.*

- ***Les 4:***

Pagina 1:

- *willen*
    - *willen maken*

Pagina 3:

- *zoals gewoonlijk*
    - -