


Opdracht 2: pannenkoekenbeleg kiezen

Snap! runtime verificatie opdracht - 

Dit is opdracht twee over een pannenkoekenbakker. In deze opdracht mag je kiezen wat je op je pannenkoek wilt. Het probleem is dat de pannenkoekenbakker niet zo goed luistert naar je bestelling. We laten je zien hoe je kan detecteren wanneer een bestelling niet klopt met "pre en postcondities". Daarnaast laten we je zien hoe we pannenkoekenbakker beter op kunnen laten letten.

We gaan ervan uit dat je al weet hoe je Snap! en Scratch moet programmeren. Is dit nieuw voor je? Vraag dan ons om hulp, of kijk eerst naar de Scratch opgaven. Die introduceren de basisconcepten van Snap! en Scratch.

Voordat we beginnen: ontmoet de bakker

Voordat we beginnen leggen we eerst uit hoe de bakker werkt! Dit is de pannenkoekenbakker. Hij verkoopt pannenkoeken voor 3 euro per pannenkoek. De bakker is vergelijkbaar met opdracht 1, behalve dat de bakker je nu een topping laat kiezen. De toppings die de bakker heeft staan onder zijn kraampje. In het plaatje beneden zijn dat "kiwi" en "nutella".



Taak 1: Probeer een paar toppings

Nu je weet hoe de bakker werkt is het tijd om zelf een paar keer een topping te kiezen. Als je dat gedaan hebt, probeer dan ook het volgende:

- Probeer een pannenkoek met kiwi te bestellen. Wat gebeurt er? Is dit logisch?
- Probeer een pannenkoek met stroop te bestellen. Wat gebeurt er? Is dit logisch?

Als je een van de opties kiest die de pannenkoekenbakker voorstelt, zal de bakker de pannenkoek bakken met je topping naar keuze. Maar, als je een topping kiest die de bakker niet heeft, zoals stroop, dan heeft de bakker dat niet door! Hij zegt dat dat hij een pannenkoek voor je heeft, maar er is helemaal niets te zien. Dat is foute boel!

Antwoord

Deel 2: Verkeerde toppings detecteren

Om foute toppings te detecteren voegen we een “postconditie” toe aan **bereid pannenkoeken**.

Een postconditie is een conditie die aan het einde van een blok gecheckt wordt. Anders gezegd, het beschrijft iets wat aan het einde van een blok gebeurd moet zijn. Als de conditie **waar** is gaat Snap! door met de rest van het script. Is de conditie **onwaar** dan laat Snap! een foutmelding zien, en gaat Snap! niet door met de rest van het script.

Er is een belangrijk feit dat **bereid pannenkoeken** moet laten gebeuren: als een pannenkoek bereid is, moet de pannenkoek zichtbaar zijn. We kunnen een blok samenstellen dat precies dit checkt.

Maak een pannenkoek-zichtbaarheids blok

Het blok dat we gaan maken moet de “Pannenkoeken” sprite vragen of het zichtbaar is. Het blok dat dit kan is het **vraag** blok. Je kan het vinden in het “Besturen” paneel. Het is in het engels, en het ziet er zo uit:

ask Sprite1 for

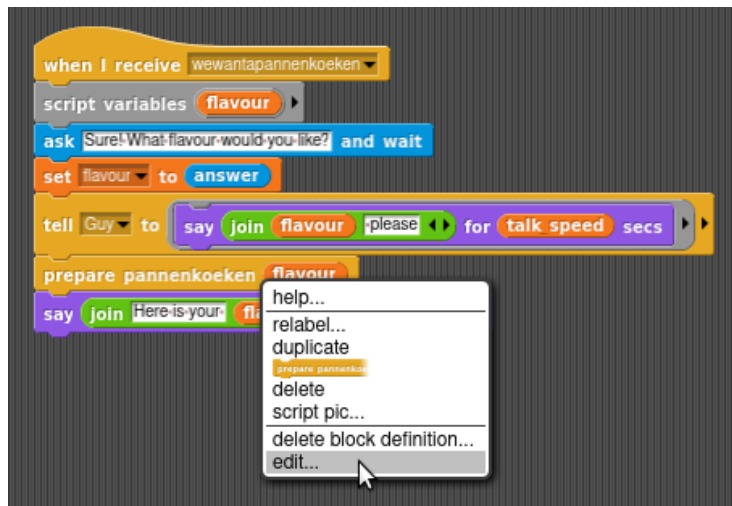
Dit blok stelt een vraag vanuit het perspectief van een sprite die je kan selecteren. Maar wat moeten we vragen? Of de sprite **shown?** is, natuurlijk! Dit is engels voor “zichtbaar”. Je kan dit blok vinden in het “Uiterlijk” paneel. Als je deze twee blokken combineert en de “Pannenkoeken” sprite selecteert, krijg je het volgende blok:

ask Pannenkoeken for shown?

We gaan nu dit blok toevoegen als postconditie!

De postconditie toevoegen

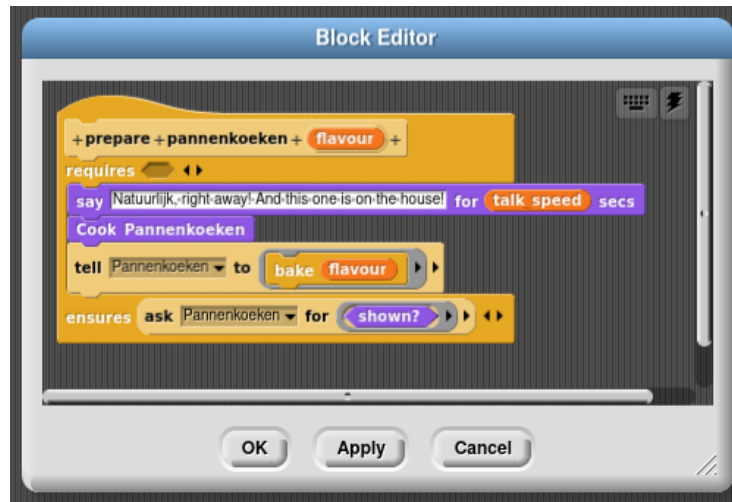
Zoek **bereid pannenkoeken** in de blokken van de bakker, klik met je rechtermuisknop, en klik **edit...**:



Een “Block Editor” verschijnt met het **bereid pannenkoeken** blok:



ensures geeft de postconditie aan. We willen dus dat daar gekeken wordt of de pannenkoek zichtbaar is. Dit doen we door het blok wat we net gemaakt hebben in het **ensures** vak te soppnen. Het resultaat ziet er als volgt uit:



Nadat je de veranderingen hebt toegepast, klik op **OK** om de block editor te sluiten. Voer dan het Snap! programma uit. Wat gebeurt er als je een topping vraagt die niet op de lijst van de bakker staat?

Antwoord
Als je een niet-bestaande topping kiest geeft Snap! een foutmelding dat je postconditie van "bereid pannenkoeken" is geschonden!

Met de toegevoegde postconditie kan het script nog steeds stuk gaan, maar nu weet je tenminste dat het mis gaat, in plaats van dat het stilletjes gebeurt!

Why use a postcondition?

Postconditions are useful for two purposes. First, they are useful as a description or documentation for what the block should accomplish. Secondly, they serve as a guard for when a bug happens, and the block did not manage to accomplish the postcondition.

It is good software engineering practice to write down and check postconditions. Other software engineers can see what you expect of a block, and know they will be alerted by a failing postcondition when this is not actually the case.

Taak 3: Verkeerde soorten beleg voorkomen met precondities

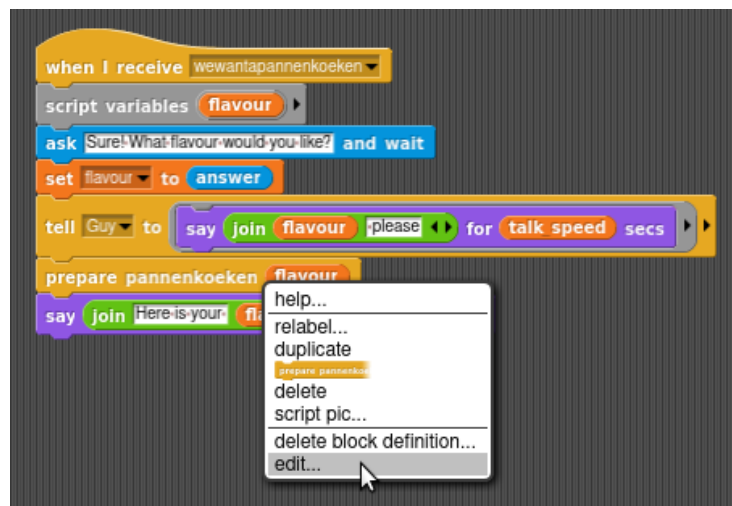
Een ander manier om naar het probleem van de bakker te kijken is dat je een topping kan geven aan **bereid pannenkoeken** die het blok niet kent. In andere woorden, **bereid pannenkoeken** kan alleen veilig uitgevoerd worden als het argument niet “onverwachts” is. In dit geval betekent dat dat het argument ofwel “kiwi” ofwel “nutella” moet zijn.

Een “preconditie” doet precies dit. In het algemeen is een preconditie een conditie die gecontroleerd wordt voordat Snap! begint met het uitvoeren van een blok. Als de conditie **waar** is voert Snap! het blok uit. Is de conditie **onwaar**, dan geeft Snap! een foutmelding en begint het niet aan het blok, en gaat het ook niet verder met de rest van het script.

Om te voorkomen dat niet-bestaande toppings in **bereid pannenkoeken** terecht komen voegen we een “preconditie” toe dat stelt dat het argument ofwel “kiwi” ofwel “nutella” moet zijn.

Een preconditie toevoegen

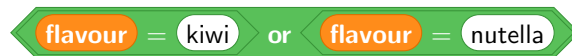
Vind wederom het **bereid pannenkoeken** blok, klik met de rechtermuisknop, en klik op **edit...**:



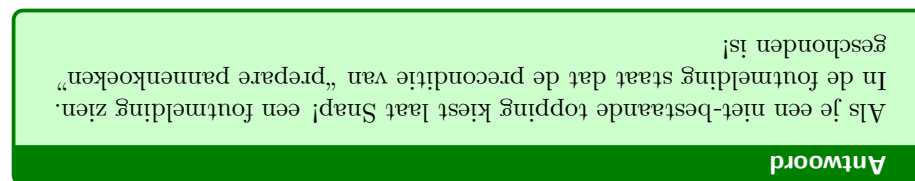
Een “**Block Editor**” zal verschijnen met het **bereid pannenkoeken** blok zoals je het achtergelaten hebt:



We stoppen een blok in het **requires** vak dat controleert of de topping “kiwi” of “nutella” is. Dat blok ziet er zo uit:



Klik “OK”, en voer het script uit. Wat gebeurt er als je een topping kiest die niet op de lijst staat?



De pre- en postcondities die we in Taak 2 en 3 hebben toegevoegd detecteren hetzelfde probleem. Maar, ze worden gecontroleerd op verschillende momenten, en werken op verschillende manieren. De preconditionie wordt gecontroleerd voordat het blok wordt uitgevoerd, de postconditie pas na het blok. De preconditionie controleert of het argument wel logisch is, dat wil zeggen, of het argument een mogelijke topping is. De postconditie controleert of wat we willen wat gebeurt daadwerkelijk gebeurd is, dat wil zeggen, of er een pannenkoek zichtbaar is.

Een conditie op meerdere plekken controleren is handig omdat het de kans verhoogt dat je een probleem zo vroeg mogelijk opmerkt. Bovendien, een conditie op meerdere manieren verwoorden verhoogt ook de kans dat je de conditie juist formuleert. Dit is gebruikelijk voor software verificatie: door een eigenschap op meerdere manieren te formuleren kun je ze vergelijken en fouten opsporen.

Taak 4: Leer de bakker om op te letten

In de laatste taak doe je een programmeer uitdaging. We willen graag dat de bakker ons vertelt dat hij een pannenkoek niet kan bakken als hij de topping niet heeft. Je kan dit implementeren in de “Bakker” sprite, alle variabelen die je nodig hebt zijn daar. Als je een hint nodig hebt, kijk dan naar het blauwe blok beneden. Succes!

Hint

Misschien kun je nog een check maken, met een **if** blok, om te kijken of de bestelling logisch is. Het **stop** all blok is hier ook handig: daarmee kun je het gehele script stoppen.