Exercise 2: choosing pancake toppings

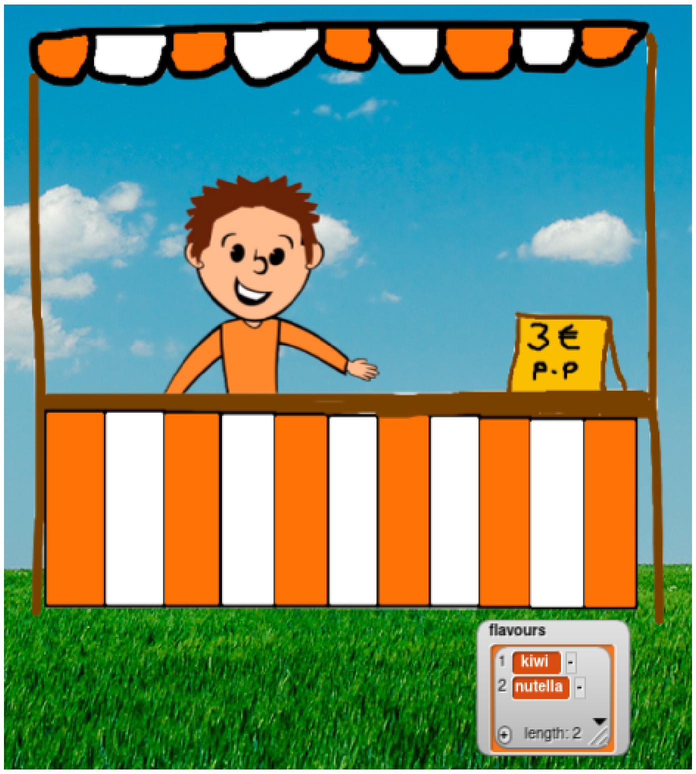
Snap! runtime verification assignment -

This is assignment two about a pancake baker selling pancakes to you. In this assignment, you will get to choose a topping for the pancake you ordered. The problem is that the pancake baker does not listen very well to your order. We will show you how to detect when the order does not make sense by using “pre and postconditions”. In the advanced final exercise you will show the baker how to attention to the order.

This exercise assumes that you know Snap! or Scratch already. If this is not the case, ask us for help, or have a look at the Scratch exercises. They introduce you to the basics of Snap! and Scratch.

Before we begin: meet the baker

However, before we start, we will first briefly explain to you how the baker works! Below is the pancake baker. He sells pancakes for 3 euros per pancake. The baker is similar to exercise 1, except that this time the baker has some toppings for you. The toppings the baker has available are shown below the stand. In the picture below, they are “kiwi” and “nutella”.



Task 1: Try choosing some toppings

Now that you know how the baker works, it's time for you to choose a topping for your pancakes. Once you had a few tries, also try the following:

- Try ordering a pancake with kiwi topping. What happens? Is this what you expect?
- Try ordering a pancake with stroop topping. What happens? Does this make sense?

Answer

When you pick one of the options, the baker will bake the pancake with that topping. However, if you choose a topping that the baker doesn't have, like stroop, the baker doesn't notice! He then says he made you a pancake, but there is no pancake at all. That's bad!

Part 2: Detect bad toppings with postconditions

To detect bad toppings, we will add a "postcondition" to **prepare pannenkoeken** ◯.

A postcondition is a condition that is checked at the end of the block. Alternatively, it describes what should have happened by the end of the block. If the condition is **true** ●, Snap! continues with the script. If the condition is ● **false**, Snap! will show an error, and not continue with the script.

There is one important fact that **prepare pannenkoeken** ◯ should accomplish: when a pancake is prepared, a pancake should be **visible**. We can make a block that checks exactly this.

Creating a pancake visibility checking block

The block we want to make should ask the "Pannenkoeken" sprite whether or not it is visible. The block that allows us to ask questions is the **ask** block. It can be found in the "Control" pane, and looks like this:

ask Sprite1 ▾ **for** ⬡

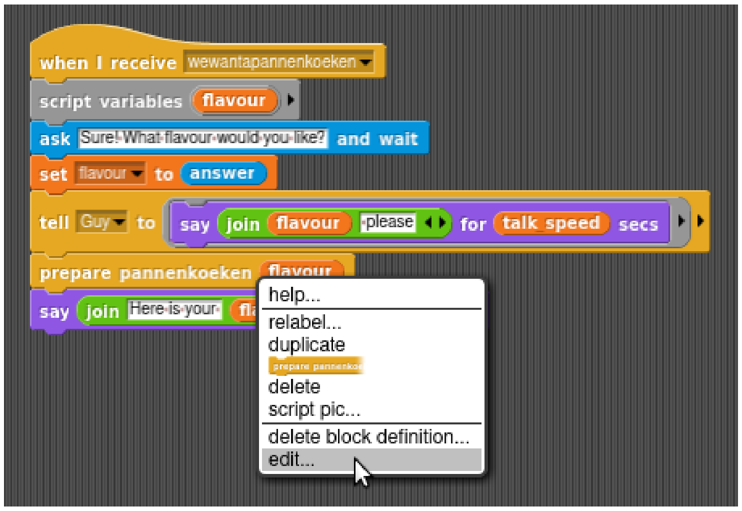
This block asks a question from the perspective of the sprite that you select. But what should we ask? Whether or not the sprite is **shown?**, of course! You can find this block in the "Looks" pane. If you combine these two blocks and select the "Pannenkoeken" sprite, you get the following block:

ask Pannenkoeken ▾ **for** **shown?**

Now let's add this block as as a postcondition!

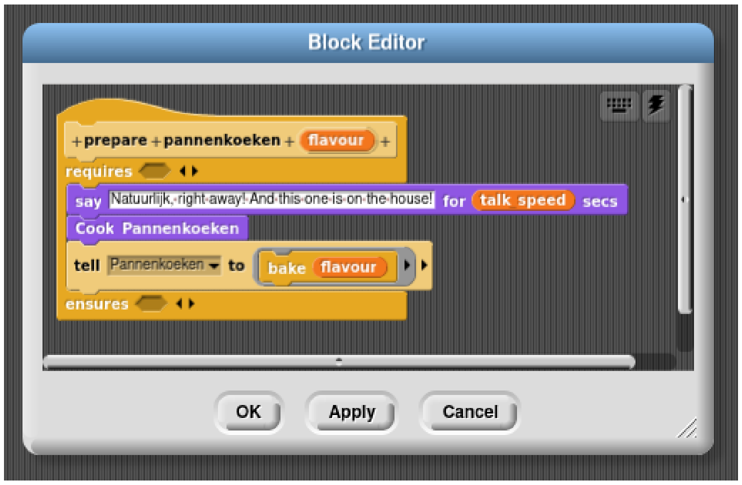
Adding the postcondition

Find prepare pannenkoeken in the baker, right click on it, and click `edit...`:



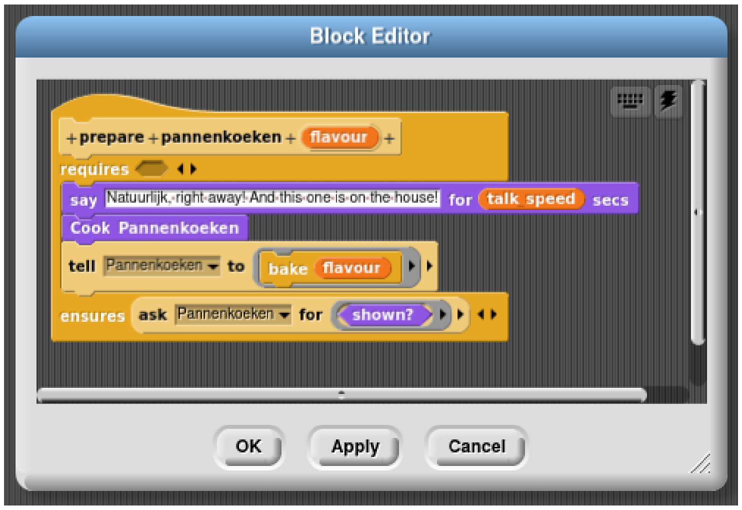
```
when I receive wewantapannenkoeken
script variables flavour
ask Sure! What flavour would you like? and wait
set flavour to answer
tell Guy to (say (join flavour please) for talk speed secs)
prepare pannenkoeken flavour
say join Here is your fl
```

A "**Block Editor**" will appear with the prepare pannenkoeken block:



```
+prepare +pannenkoeken + flavour +
requires
say Natuurlijk, right away! And this one is on the house! for talk speed secs
Cook Pannenkoeken
tell Pannenkoeken to (bake flavour)
ensures
```

ensures indicates the postcondition, where we will check if the pancake is visible. We will do this by adding the block we just made to the to the **ensures** slot. The final result should look something like this:



```
+prepare+pannenkoeken+ flavour +
requires
say Natuurlijk, right away! And this one is on the house! for talk speed secs
Cook Pannenkoeken
tell Pannenkoeken to bake flavour
ensures ask Pannenkoeken for shown?
```

After making the changes, click **OK** to close the block editor. Then run the Snap! program again. What happens if you choose a topping that is not on the list of the baker now?

Answer

If you pick a non-existent topping, Snap! will show you an error saying the postcondition of „prepare pannenkoeken" is violated!

With the added postcondition, the script can still go wrong, but at least we will get a clear error, instead of the error quietly hiding.

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.

Task 3: Prevent bad toppings with preconditions

Another way of looking at the problem with this Snap! program is that you can give a topping to **prepare pannenkoeken** that it doesn't know about.

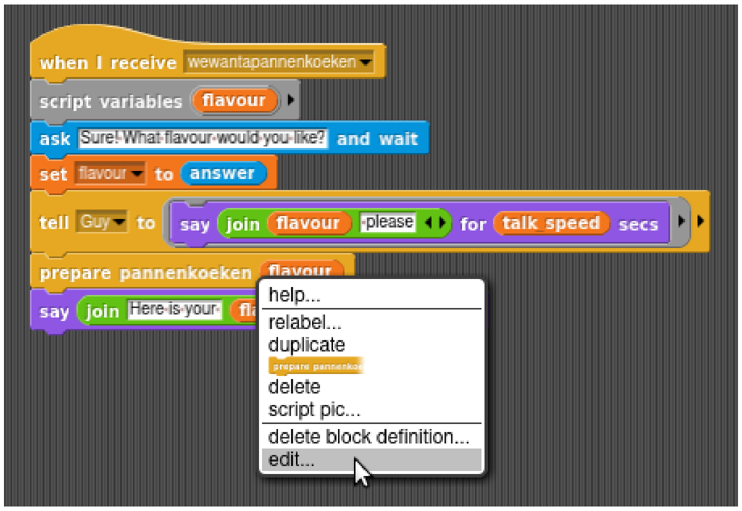
Specifically, **prepare pannenkoeken** can only safely be executed if the argument is not "unexpected". In this case, it means the argument must be either "kiwi" or "nutella".

A "precondition" does exactly this. Generally, a precondition is a condition that Snap! checks before it continues with a block. If the condition is **true**, Snap! continues with the block. If the condition is **false**, Snap! will show an error, and not continue with the block. It will also not continue with the rest of the script.

To prevent bad toppings from being given to **prepare pannenkoeken**, we will add a "precondition" to it that says that the argument must be either "kiwi" or "nutella".

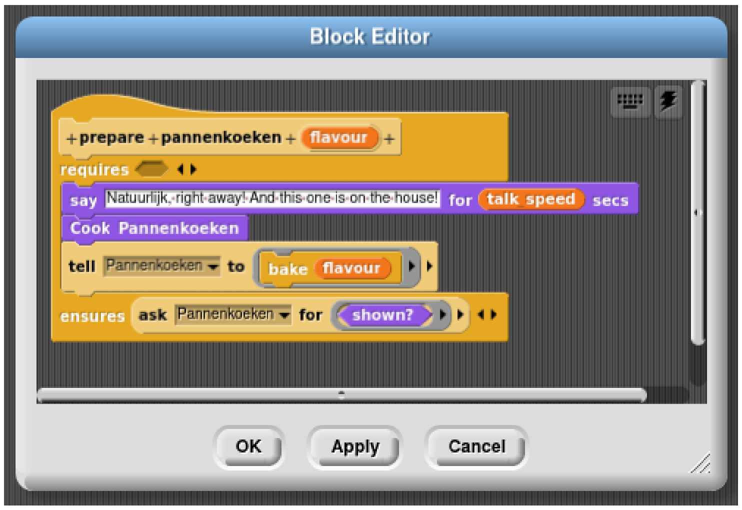
Adding a precondition

Again, find the **prepare pannenkoeken** block, right click on it, and click `edit...`:



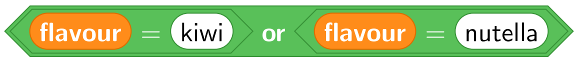
```
when I receive wewantapannenkoeken
script variables flavour
ask Sure! What flavour would you like? and wait
set flavour to answer
tell Guy to say join flavour please for talk speed secs
prepare pannenkoeken flavour
say join Here is your fl
```

A "**Block Editor**" will appear with the **prepare pannenkoeken** block as you left it:



```
+prepare+pannenkoeken+ (flavour) +
requires
say Natuurlijk, right away! And this one is on the house! for (talk speed) secs
Cook Pannenkoeken
tell Pannenkoeken to (bake (flavour))
ensures ask Pannenkoeken for (shown?)
```

We will put a block in the **requires** slot that checks if the input is either “kiwi” or “nutella”. Such a block looks like this:



```
(flavour) = kiwi or (flavour) = nutella
```

Click “**OK**”, and run the script. What happens if you choose a topping that is not on the list of the baker now?

Answer

If you pick a non-existent topping, Snap! will show you an error saying the precondition of „prepare pannenkoeken“ is violated!

The pre and postconditions that we added in Task 2 and Task 3 detect the same bug. However, they work at different times, and in different ways. The precondition is checked before the block, the postcondition after the block. The precondition checks whether the input of the block makes sense, meaning, whether the flavour chosen is one that is allowed. The postcondition checks if what we want to happen actually happened, meaning, whether a pancake is visible on screen.

Checking a condition in multiple places is useful as it increases the chance you catch a problem as early as possible. In addition, expressing the same property in different ways also increases the chance that you get it right. This is common practice in software verification: by expressing one property in multiple ways, you can compare them, and catch mistakes.

Task 4: Teaching the baker how to pay attention

In the last task you will finish with a programming challenge. We would like the baker to tell us that he cannot make a pancake with a topping he does not know about. You can implement this in the “Baker” sprite, all the variables you need are available there. If you need a hint, look at the blue block below. Good luck!

Hint

Maybe you can implement another check, using an **if** block, to see if the order makes sense. The **stop** all ▾ block is also useful here: it allows you to terminate the entire script.