Eckart Modrow

emodrow@informatik.uni-goettingen.de

The SQLsnap-Supermarket

A project illustrating new options for Snap!

Status: 2014-03-04

Eckart Modrow

SQLsnap-example „Supermarket“

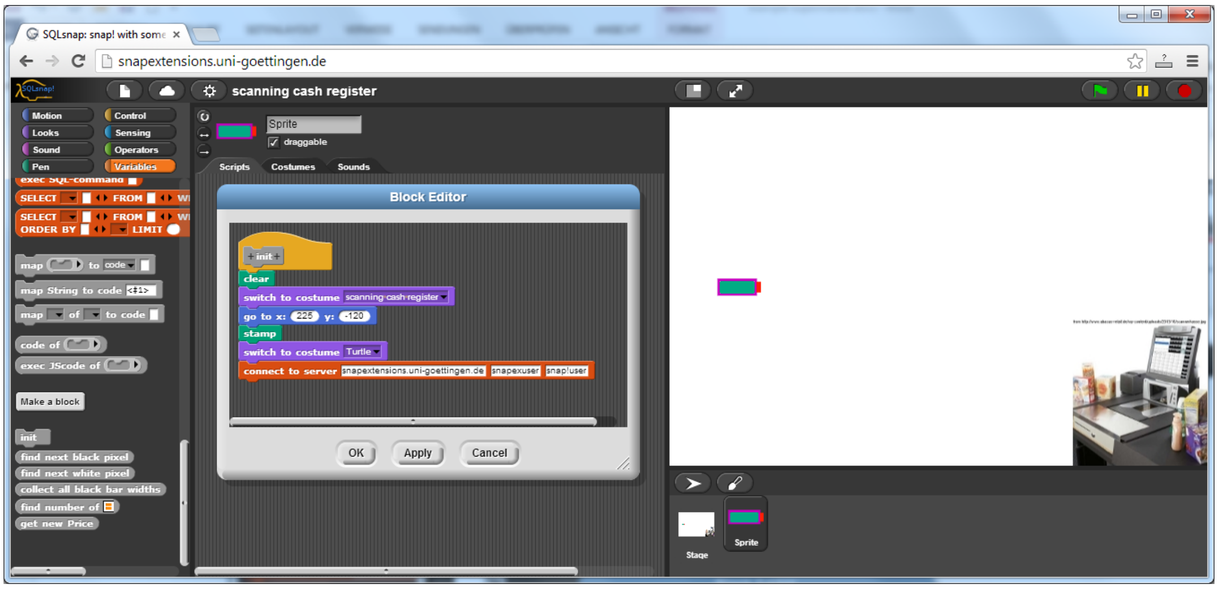
Let’s assume we have a supermarket with different divisions:

- a scanning cash register (reads the barcodes on the products, produces article numbers and bills)
- a stock management with integrated database (gets article numbers, produces prices and orders products from suppliers, if necessary)
- a “smart” scale for fruits (recognizes the fruit with a camera, produces barcodes)
- an advertising department (responsible for payback, promotion, special offers, …)
- a security department (responsible for payment in the parking garage, “banned” customers which doesn’t pay, …

All division-implementations run on different computers and communicate by text files on a server. And we use no “professional” procedures, but only “naïve” solutions asking for enhancement by the students.

1. The scanning cash register

First we need an image of a scanning cash register. We take it as new costume of the turtle, send the turtle to an appropriate position, force a stamp, and connect to an appropriate server. We put all these blocks in a method ***init***.

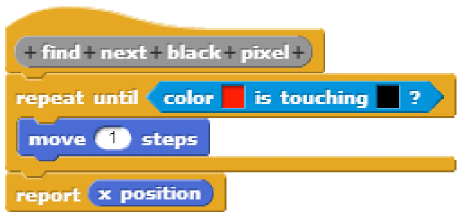


```
+ init +
clear
switch to costume scanning cash register
go to x: 225 y: -120
stamp
switch to costume Turtle
connect to server snapextensions.uni-goettingen.de snapexuser snap!user
```

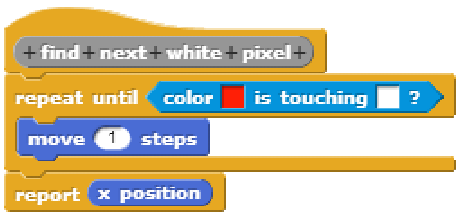
Now we need barcodes. If we don’t know anything about them, we invent new ones.

First attempt: We draw some black bars on the stage and try to find out their width. Then we draw a new costume for the turtle, a “laser pointer” with a red spot at front. We can ask now, whether the red spot touches the black color. Combined with the position of the laser pointer we get the widths of the bars. We collect them in a list ***widths***.

But first we write two methods ***find next black pixel*** and ***find next white pixel***.

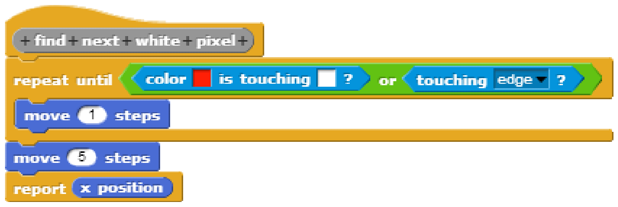


```
+find+next+black+pixel+
repeat until color [red] is touching [black] ?
    move 1 steps
report x position
```

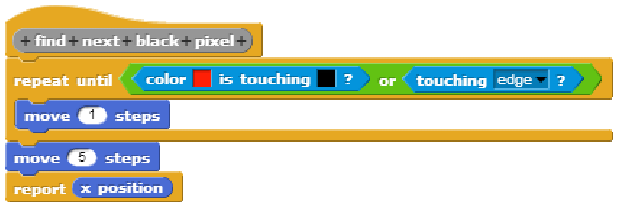


```
+find+next+white+pixel+
repeat until color [red] is touching [white] ?
    move 1 steps
report x position
```

Find next black pixel works well, but ***find next white pixel*** does not. Why? If the front of the red "nose" of the laser pointer touches a black bar, it stops. But the rest of the "nose" still touches the white background. So we have to move the laser pointer some steps right before asking for a white pixel. In addition the search process should stop, if the laser pointer touches the right edge. We get:

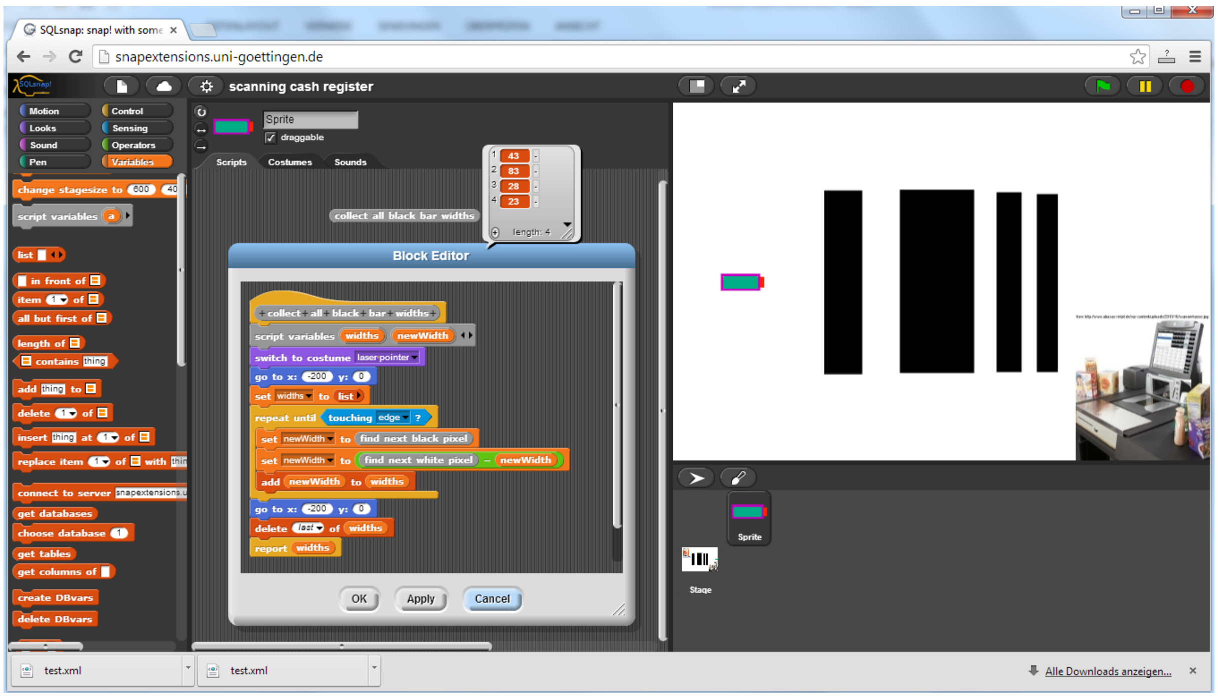


```
+find+next+white+pixel+
repeat until < color [red] is touching [white] ? > or < touching edge ? >
    move 1 steps
move 5 steps
report x position
```



```
+find+next+black+pixel+
repeat until < color [red] is touching [black] ? > or < touching edge ? >
    move 1 steps
move 5 steps
report x position
```

It's easy to get the widths of all black bars: We make two variables (a list ***widths*** and a number ***new-Width***), measure the x-positions of the left and right bar-edges and put the differences in the list. Due to the stopping-procedure the last value is irrelevant.

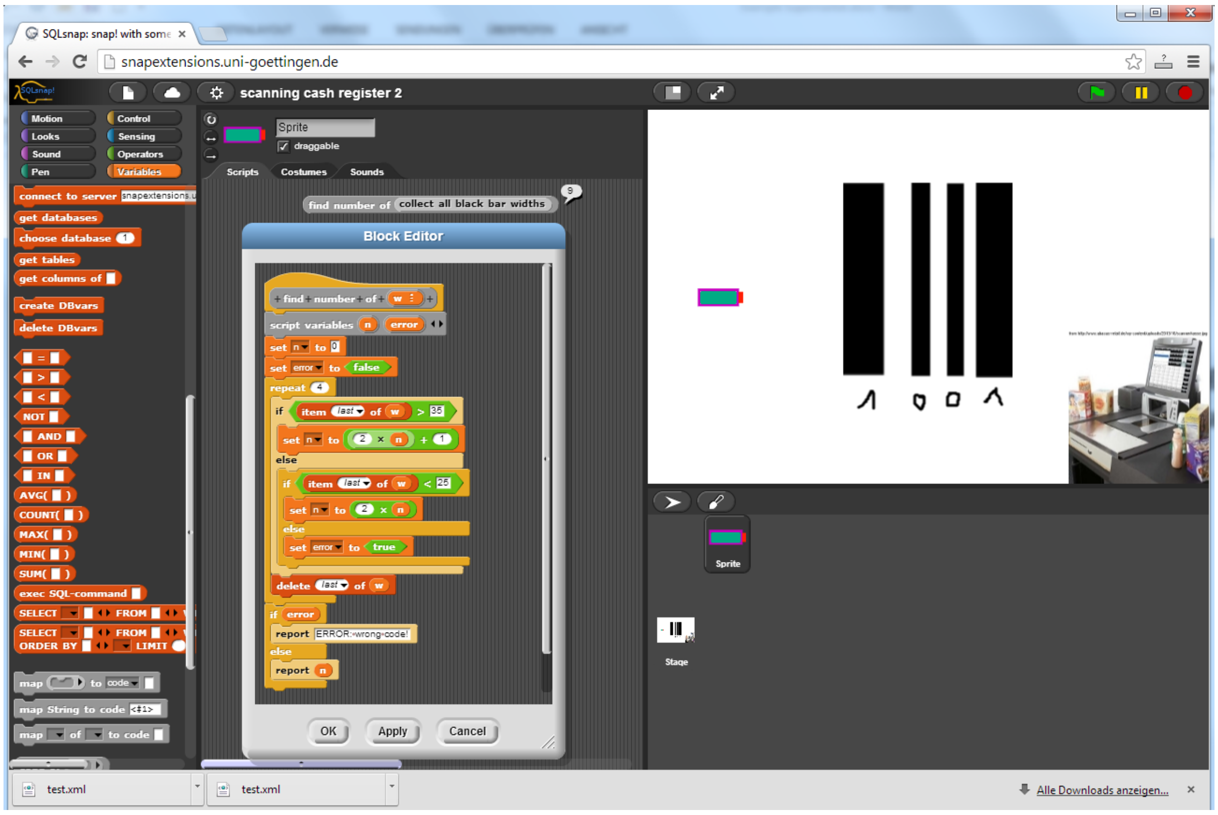


```
collect all black bar widths
```

```
1  43
2  83
3  28
4  23
length: 4
```

```
+collect+all+black+bar+widths+
script variables widths newWidth
switch to costume laser pointer
go to x: -200 y: 0
set widths to list
repeat until touching edge ?
    set newWidth to find next black pixel
    set newWidth to (find next white pixel − newWidth)
    add newWidth to widths
go to x: -200 y: 0
delete last of widths
report widths
```

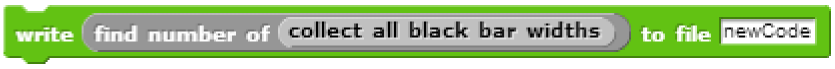
Now we need a code that tells what the bars mean. Let's choose: we take the dual number system with four digits, a "broad bar" (width > 35) means "1" and a small bar (width < 25) means "0". We draw some barcodes of this type as customs of the stage and test the script. To get the numeric value of our barcode, we repeat four times: read the most right list item and change the value of a number ***n*** in the usual way. Then delete the most right list item. If the list item is not in the correct range, we report an error.



```
find number of (collect all black bar widths)    9

Block Editor
+find+number+of+(w ⋮)+
script variables (n) (error)
set n to 0
set error to <false>
repeat 4
    if <(item last of w) > 35>
        set n to ((2 × n) + 1)
    else
        if <(item last of w) < 25>
            set n to (2 × n)
        else
            set error to <true>
    delete last of w
if error
    report ERROR: wrong code!
else
    report n
OK   Apply   Cancel
```

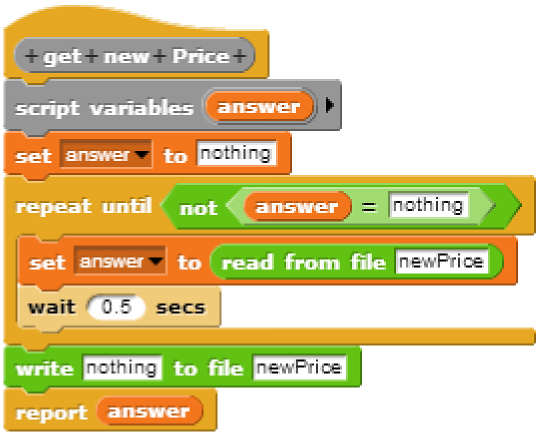
We send the value of the bar code to the stock management to get the actual price. We do this by writing the bar code value to a text file ***newCode*** on the server.

The whole process can be compressed to one block.



```
write (find number of (collect all black bar widths)) to file newCode
```

The stock management has to read this file time by time, and to report the correct price and label to a file ***newPrice***. Afterwards it sets ***newCode*** again to ***-1***. The cash register reads the file ***newPrice*** time by time and gets price and label corresponding to the bar code value. Afterwards it writes ***nothing*** to this file.



```
+get+new+Price+
script variables (answer)
set answer to nothing
repeat until <not <(answer) = nothing>>
    set answer to (read from file newPrice)
    wait 0.5 secs
write nothing to file newPrice
report (answer)
```

Exercises:

1. Real barcodes use the gaps between the black bars in the same way as the black bars: as white bars. The width of the white gaps have the same meaning as the black ones. Change the script ***collect all black bar widths*** to another ***collect all bar widths*** that counts the widths of all black and white bars. (The first and the last bar are black.) If four black bars are used: what is the biggest representable number?

2. Add some costumes to a new ***printer sprite*** that is able to print barcodes on the stage. First the user is asked for a number.

3. Look for information about your ***national bar code system***. In Europe you'll find the EAN codes. Change the printer sprite to a "national bar code printer sprite" that prints these codes.

4. If the stock management doesn't respond, the ***get new price*** script waits till infinity. Change the script to an acceptable version.

5. If the stock management instance works well, the cash register gets answers of the type
   <price>,<label>
   Let the cash register produce bills for the customers including date and time, all purchased products with prices and the sum of all prices. Taxes should be shown according to your national standards.

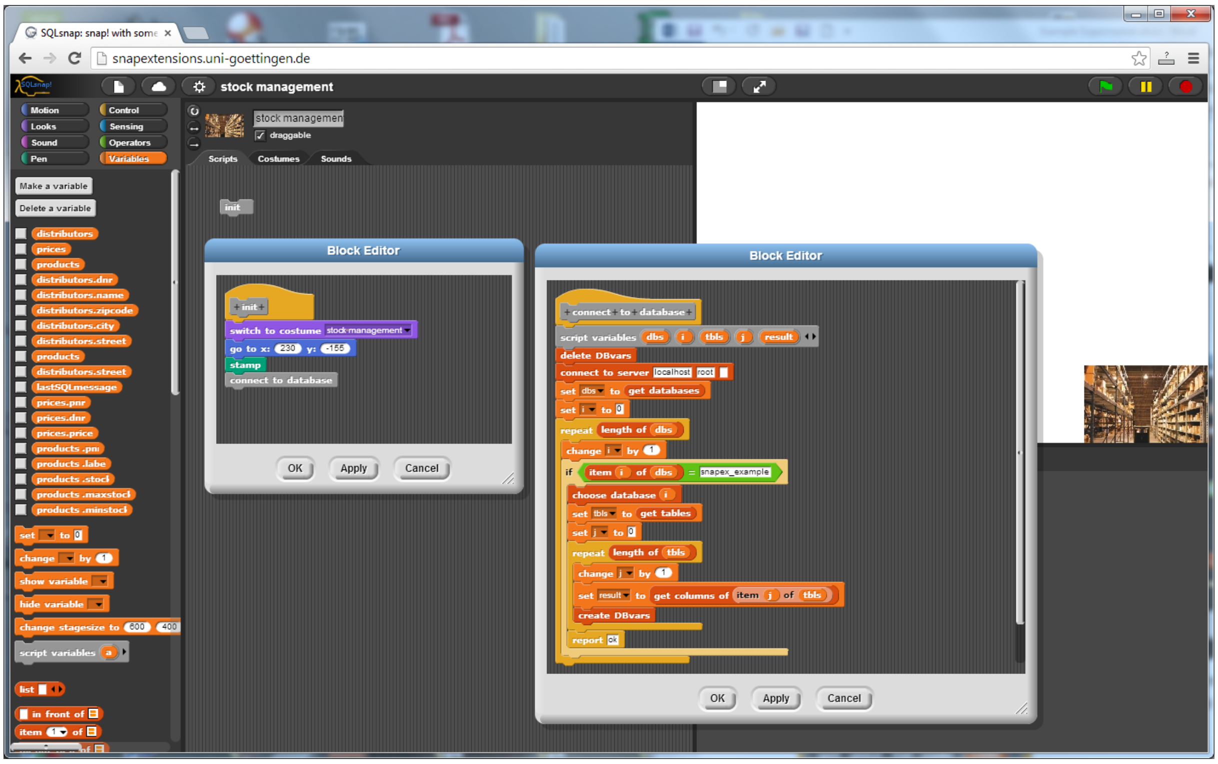
2. The stock management

The stock management uses a mySQL database on the server: here the ***snapex_example*** database.

There are three tables:

- products(<u>pnr</u>,label,maxstock,minstock,stock)
- distributors(<u>dnr</u>,name,zipcode,city,street)
- prices(pnr,dnr,price)

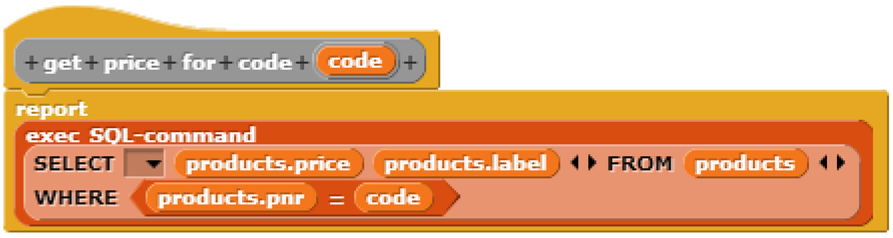
We have to answer if the scanning cash register asks for a price. So we have a permanent look for changes of the ***newCode*** text file. But first we print a stock management image on the stage, connect to the database, choose the snapex_example database and create variables for all used things: tables and table-columns.



```
+ init +
switch to costume stock management
go to x: 230 y: -155
stamp
connect to database
```

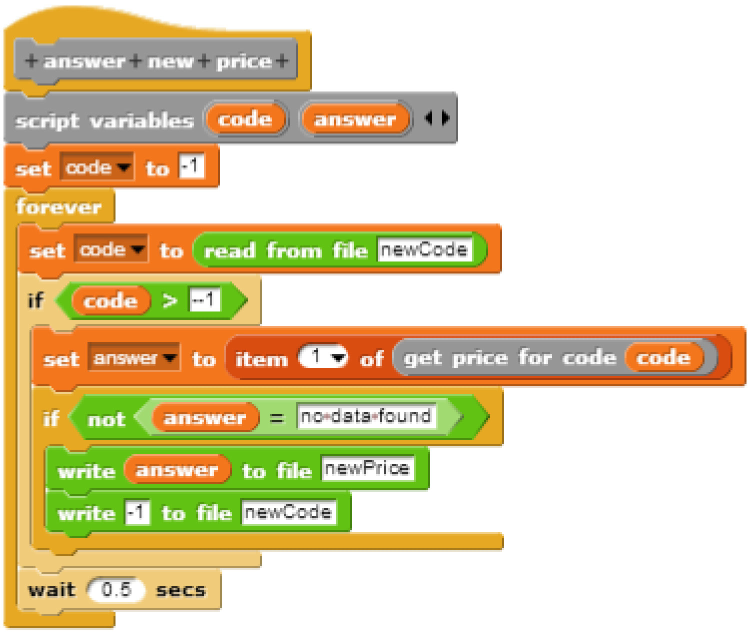
```
+ connect + to + database +
script variables dbs i tbls j result
delete DBvars
connect to server localhost root
set dbs to get databases
set i to 0
repeat length of dbs
    change i by 1
    if item i of dbs = snapex_example
        choose database i
        set tbls to get tables
        set j to 0
        repeat length of tbls
            change j by 1
            set result to get columns of item j of tbls
            create DBvars
        report ok
```

A query to find the actual price and label is simple. We produce the text of a SQL-query by combining appropriate variables in the ***SELECT-block***. The query is executed by the ***exec-SQL-command*** block.



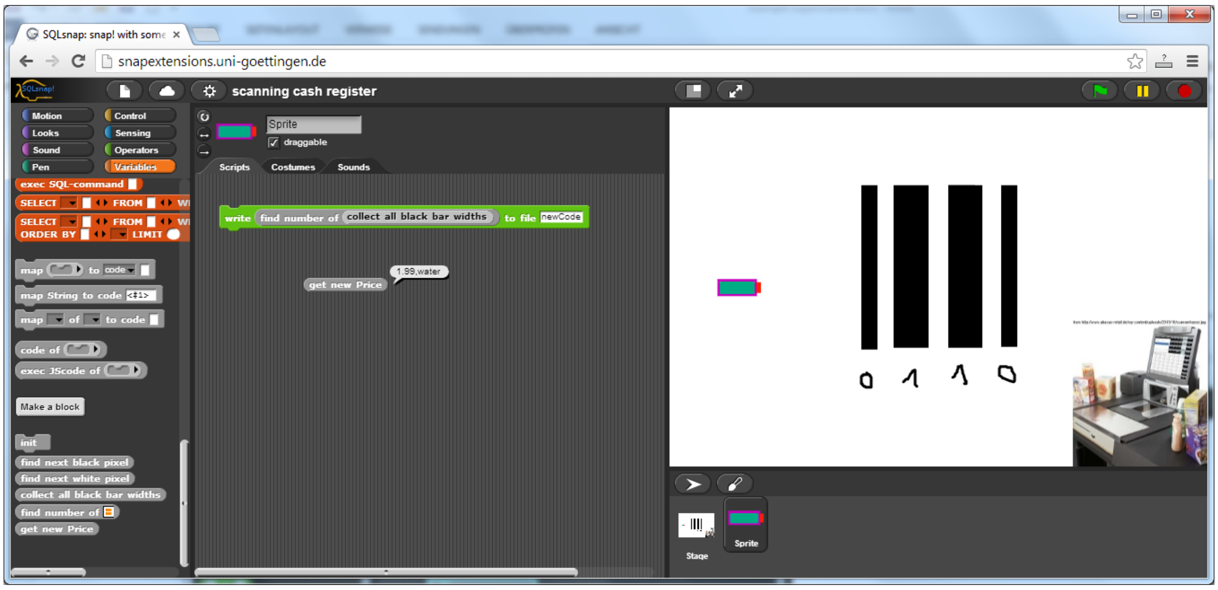
```
+ get + price + for + code + code +
report
  exec SQL-command
    SELECT products.price products.label FROM products
    WHERE products.pnr = code
```

The infinite loop is encapsulated in a block ***answer new price***. We have a look there whether the file ***new-Code*** has changed. If so, we ask the SQL-server for price and label of the product and write these to the file ***newPrice***, if we got a correct an-swer.



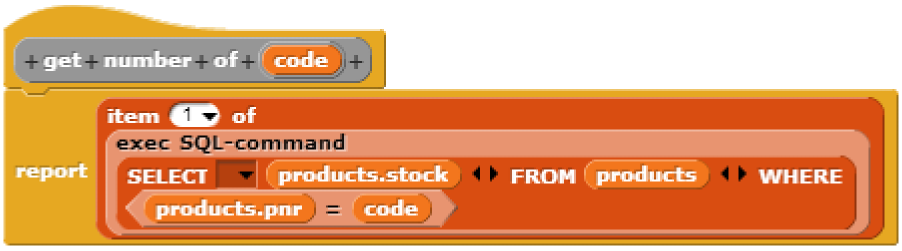
```
+answer+new+price+
script variables (code) (answer) ◂▸
set code ▾ to -1
forever
    set code ▾ to read from file newCode
    if (code) > -1
        set answer ▾ to item (1 ▾) of get price for code (code)
        if not (answer) = no-data-found
            write (answer) to file newPrice
            write -1 to file newCode
    wait 0.5 secs
```

If this script is running on the stock management instance of ***SQLsnap***, the scanning cash register instance gets the correct answer.



```
write find number of collect all black bar widths to file newCode

get new Price  → 1.99,water
```

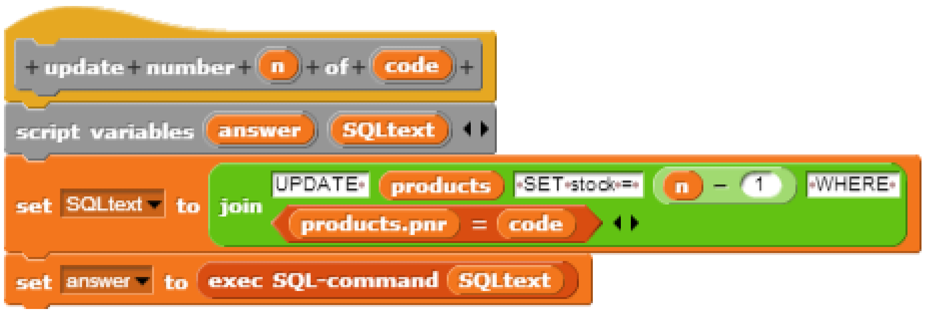
If you have write permissions on the server[1], you can update the stock numbers. First you have to find out the number of products with a certain code in the stock.



```
+get+number+of+(code)+
report item (1 ▾) of
    exec SQL-command
    SELECT ▾ (products.stock) ◂▸ FROM (products) ◂▸ WHERE
    (products.pnr) = (code)
```

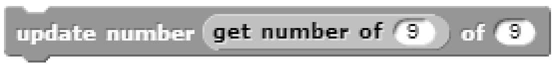
[1] If not: install mySQL on the computer on which the „stock management instance” of SQLsnap runs. Connected to ***localhost*** as user ***root*** you have write permissions.

In SQLsnap actually we have only blocks to construct ***select*** commands. So we use the ***exec sql-command*** block directly for an update command.



```
+ update + number + (n) + of + (code) +
script variables (answer) (SQLtext) ◂▸
set SQLtext ▾ to join UPDATE (products) SET stock = ((n) − (1)) WHERE <(products.pnr) = (code)> ◂▸
set answer ▾ to exec SQL-command (SQLtext)
```

If code is the product key, we can combine both blocks to an update command:



```
update number (get number of (9)) of (9)
```

Exercises:

1. If some products are purchased, the ***stock*** declines below the ***minstock*** value. Order new products, so that the missing product stock reaches the ***maxstock*** value. Find out the distributer with the lowest price for this product.

2. The supermarket wants to become a "bio supermarket". Change the distributers for all possible products and adjust the prices.

3. Insert bio-products with different prices in the product list in addition to the cheep-products, whenever possible.

4. Always on Saturday we need an update run. The prices of the distributers may have changed. So we have to adjust the prices in the products table.

5. The supermarket works well, but he needs more money. Hike all prices up by 10%.

6. The stock management needs statistics about the sales per month and year. Collect the necessary data and show the sales in appropriate diagrams.

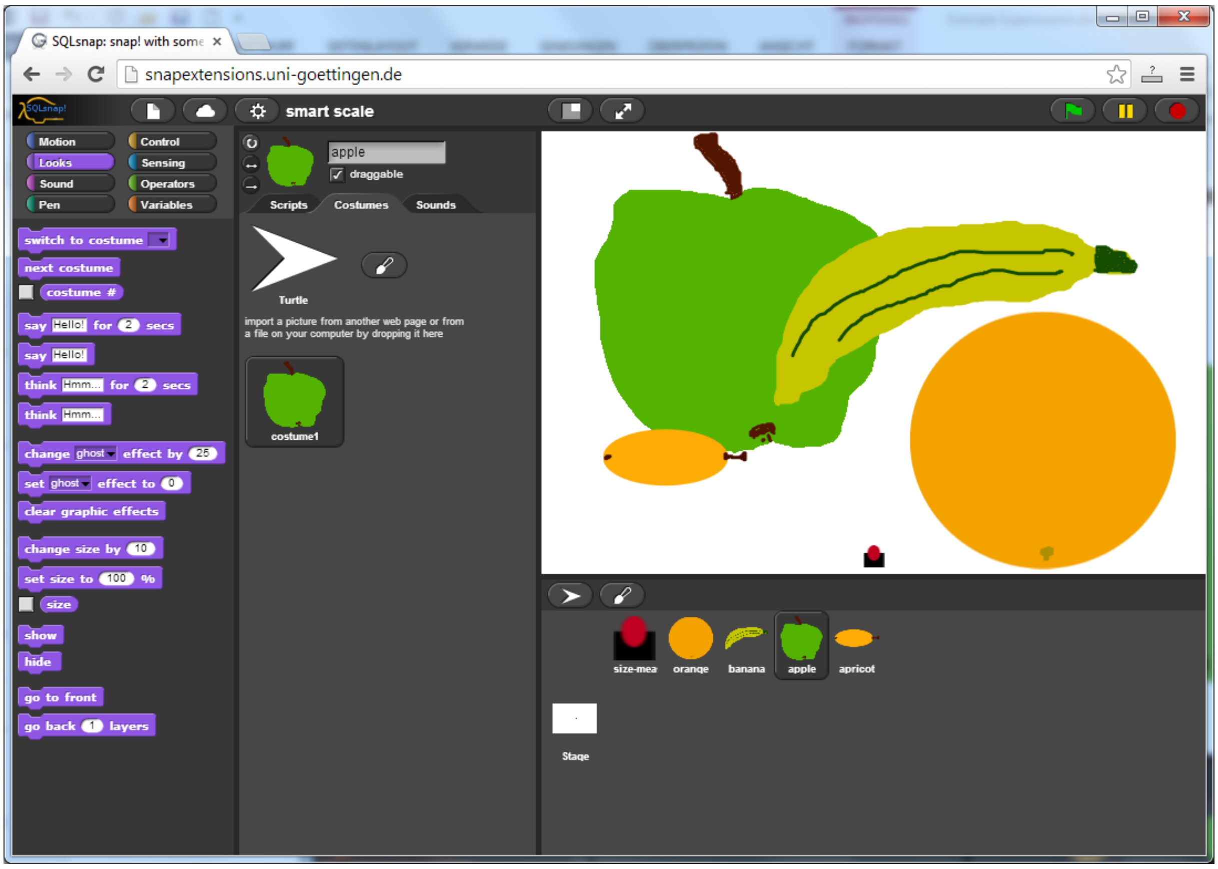
7. Build a block to produce ***update-commands*** for mySQL .
   Syntax: UPDATE <tablename> SET column = value {,column = value} WHERE <condition>;
   Example: UPDATE products SET stock = 99 WHERE pnr = 11;

8. Build a block to produce ***insert-commands*** for mySQL .
   Syntax: INSERT INTO <tablename> (column{,column}) VALUES (value{,value});
   Example: INSERT INTO prices (pnr,dnr,price) VALUES (1,2,3.45);

9. Build a block to produce ***delete-commands*** for mySQL .
   Syntax: DELETE FROM <tablename> WHERE <condition>;
   Example: DELETE FROM distributors WHERE name = 'Miller';

3. The smart scale for fruits

The first attempt is to find some criteria to identify a fruit. We draw an apple, an orange, an apricot and a banana.



The differences are obvious:

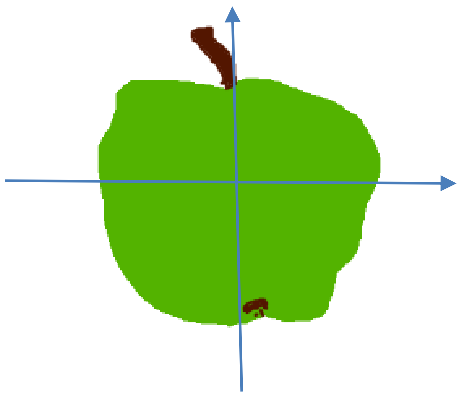
- apple and orange are round, the banana is long
- orange, apricot, and banana are orange-yellow, the apple (here) is green
- apricot is small, the others are bigger

But what means “round”, “long”, “yellow” and “green”, “big”???

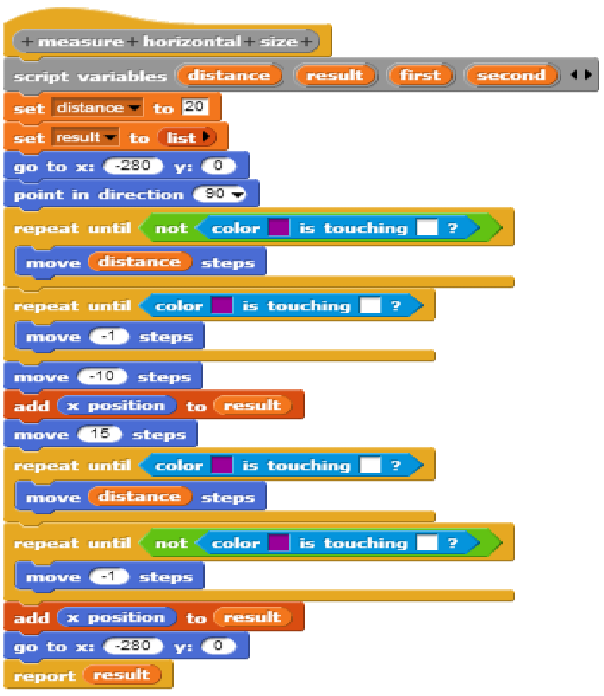
We know, but the computer not. We have to teach him.

Distinguish *round, oval* and *long*

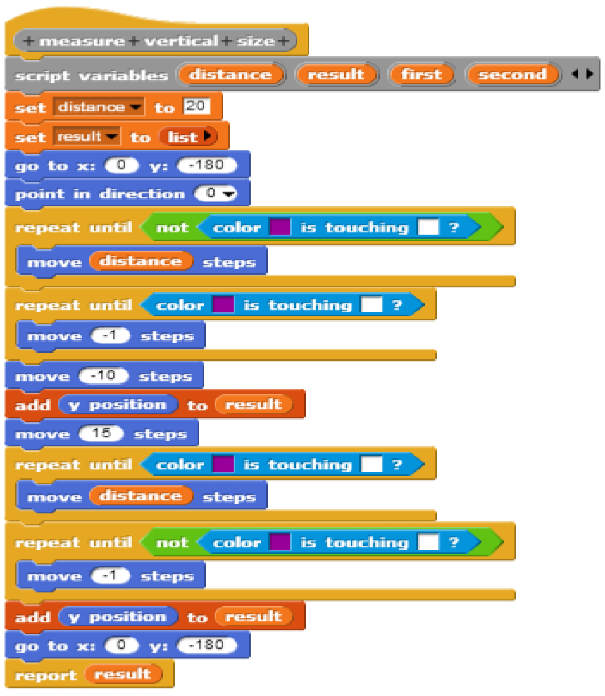
We put the object in the middle of the stage and send a ***size-measuring-sprite*** from left to right and from bottom to top. We measure the size of the object on these sections and calculate the ratio. “Round” object should have a ratio near 1, “long” objects another. For “oval” objects we should use more directions, but for now “oval” means “not long and not round”.



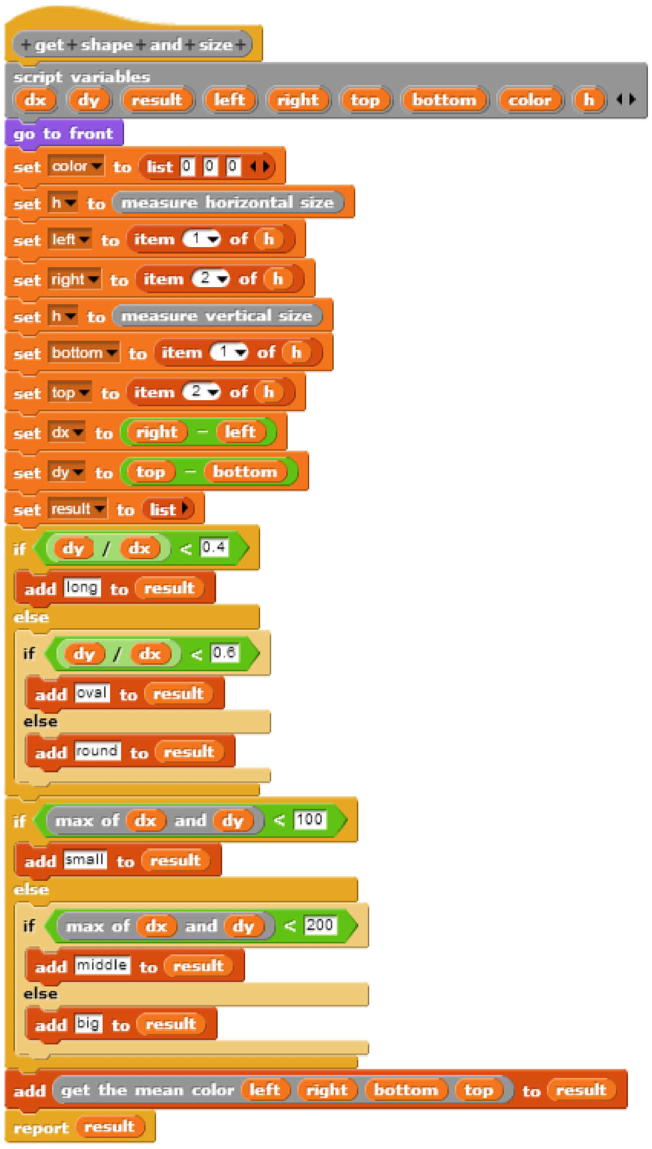
The ***measure horizontal size*** block delivers a list of two values: left and right border, as well as ***measure vertical size*** block delivers bottom and top of the object. With these results we can decide whether an object is long, round or oval, and we have the total size.



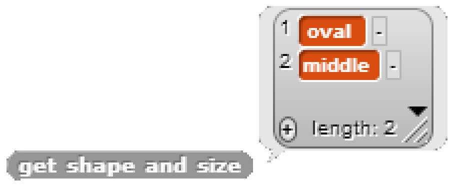
```
+measure+horizontal+size+
script variables distance result first second
set distance to 20
set result to list
go to x: -280 y: 0
point in direction 90
repeat until not color 🟪 is touching ⬜ ?
    move distance steps
repeat until color 🟪 is touching ⬜ ?
    move -1 steps
move -10 steps
add x position to result
move 15 steps
repeat until color 🟪 is touching ⬜ ?
    move distance steps
repeat until not color 🟪 is touching ⬜ ?
    move -1 steps
add x position to result
go to x: -280 y: 0
report result
```



```
+measure+vertical+size+
script variables distance result first second
set distance to 20
set result to list
go to x: 0 y: -180
point in direction 0
repeat until not color 🟪 is touching ⬜ ?
    move distance steps
repeat until color 🟪 is touching ⬜ ?
    move -1 steps
move -10 steps
add y position to result
move 15 steps
repeat until color 🟪 is touching ⬜ ?
    move distance steps
repeat until not color 🟪 is touching ⬜ ?
    move -1 steps
add y position to result
go to x: 0 y: -180
report result
```



```
+get+shape+and+size+
script variables
dx dy result left right top bottom color h
go to front
set color to list 0 0 0
set h to measure horizontal size
set left to item 1 of h
set right to item 2 of h
set h to measure vertical size
set bottom to item 1 of h
set top to item 2 of h
set dx to right - left
set dy to top - bottom
set result to list
if dy / dx < 0.4
    add long to result
else
    if dy / dx < 0.6
        add oval to result
    else
        add round to result
if max of dx and dy < 100
    add small to result
else
    if max of dx and dy < 200
        add middle to result
    else
        add big to result
add get the mean color left right bottom top to result
report result
```



```
1 oval
2 middle
length: 2
get shape and size
```

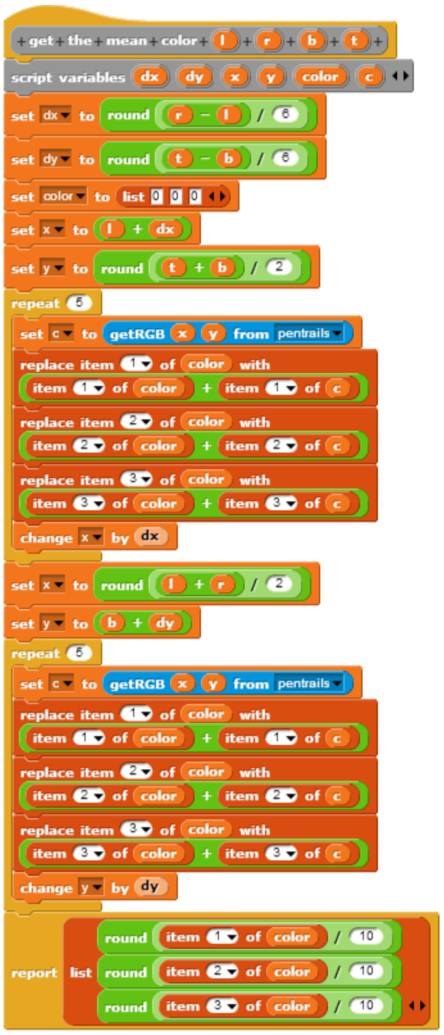
Find out the color of real objects

Normal fruits have different colors. So we have to measure the mean color. We do this by measuring ten RGB-values: five on the horizontal slice and five on the vertical. (That was the reason to store the position of the object.) That's easy. But our RGB-value list may describe 256 * 256 * 256 colors. That are 16.777.218 colors. A bit too much.

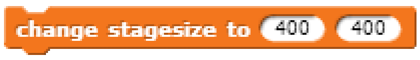
So we need a method to reduce the number of colors.

We try it this way: For each RGB-value we decide, whether we have a "high" or a "low" value of this color. If "high" we set the value to 255, if "low we set it to 0. We get two possible values for each color, and a total of 2 * 2 * 2 = 8 possible colors. First we try whether we have enough colors to see anything relevant on a picture – or not.

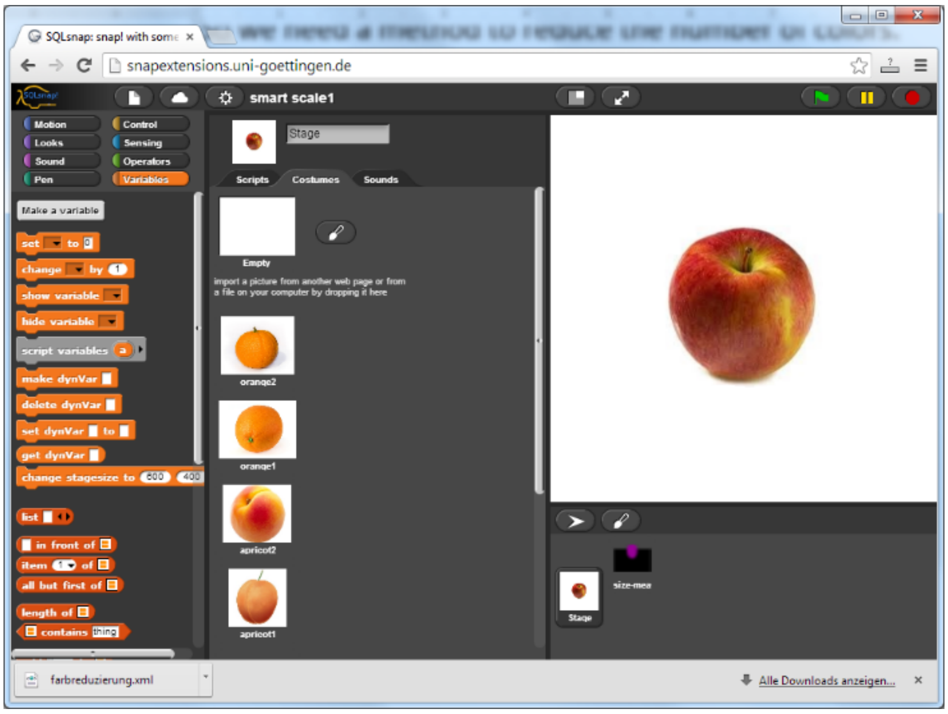
We change the stagesize to 400 x 400 and load some pictures of fruit as backgrounds of the stage.



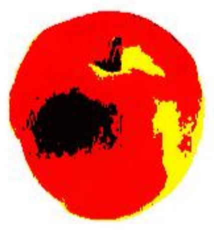
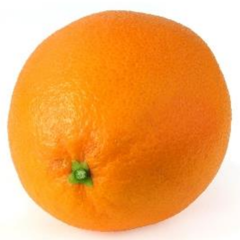
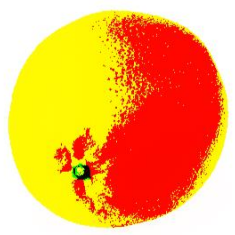
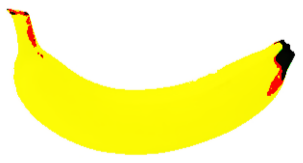
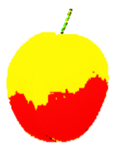
```
+get+the+mean+color+(l)+(r)+(b)+(t)+
script variables (dx) (dy) (x) (y) (color) (c)
set dx to round ((r − l) / 6)
set dy to round ((t − b) / 6)
set color to list 0 0 0
set x to (l + dx)
set y to round ((t + b) / 2)
repeat 5
    set c to getRGB (x) (y) from pentrails
    replace item 1 of color with
        (item 1 of color) + (item 1 of c)
    replace item 2 of color with
        (item 2 of color) + (item 2 of c)
    replace item 3 of color with
        (item 3 of color) + (item 3 of c)
    change x by dx
set x to round ((l + r) / 2)
set y to (b + dy)
repeat 5
    set c to getRGB (x) (y) from pentrails
    replace item 1 of color with
        (item 1 of color) + (item 1 of c)
    replace item 2 of color with
        (item 2 of color) + (item 2 of c)
    replace item 3 of color with
        (item 3 of color) + (item 3 of c)
    change y by dy
report list (round ((item 1 of color) / 10))
            (round ((item 2 of color) / 10))
            (round ((item 3 of color) / 10))
```

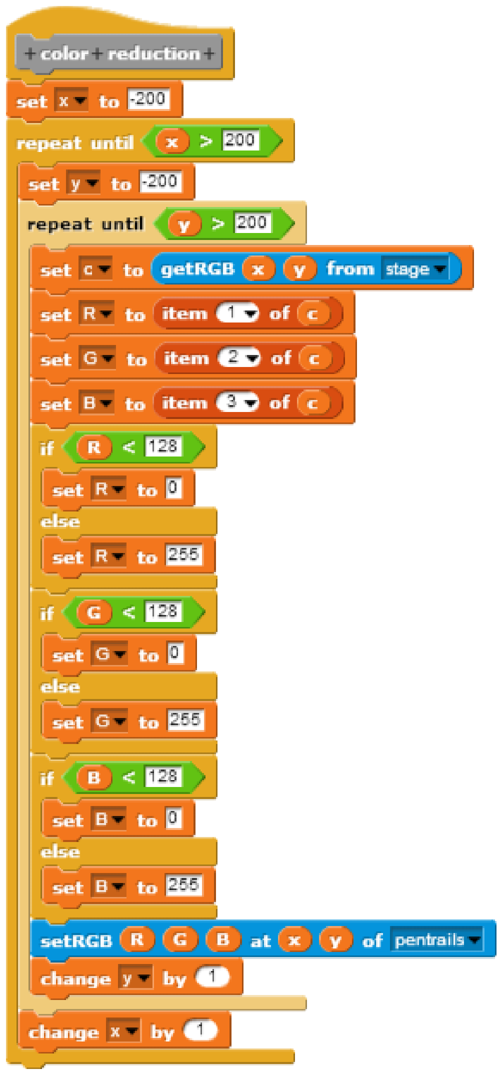


```
change stagesize to 400 400
```



Now we reduce the colors as described …

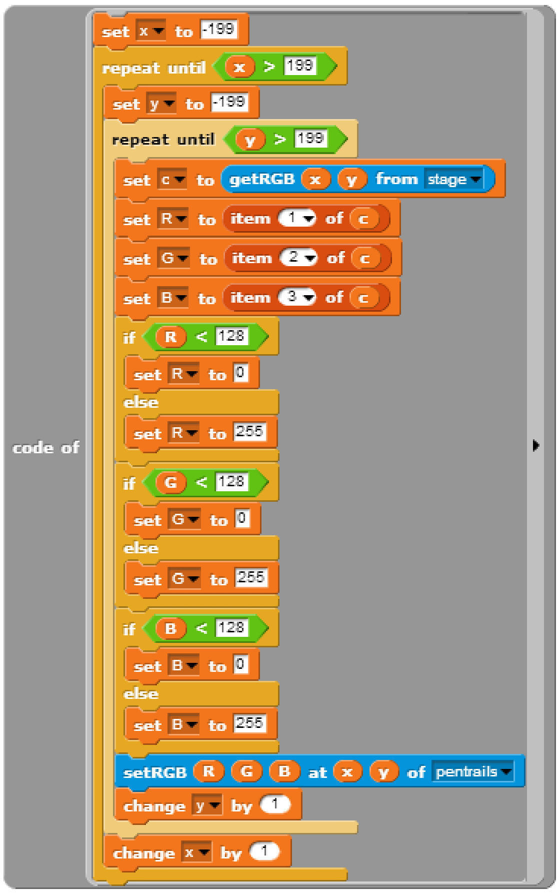


```
+ color + reduction +
set x to -200
repeat until (x > 200)
    set y to -200
    repeat until (y > 200)
        set c to getRGB x y from stage
        set R to item 1 of c
        set G to item 2 of c
        set B to item 3 of c
        if (R < 128)
            set R to 0
        else
            set R to 255
        if (G < 128)
            set G to 0
        else
            set G to 255
        if (B < 128)
            set B to 0
        else
            set B to 255
        setRGB R G B at x y of pentrails
        change y by 1
    change x by 1
```

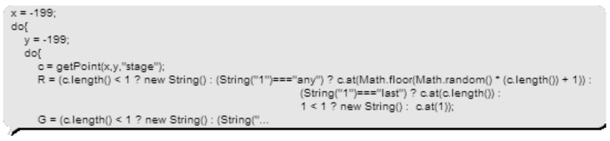
.. and find, that’s ok.

How does it work in acceptable time?

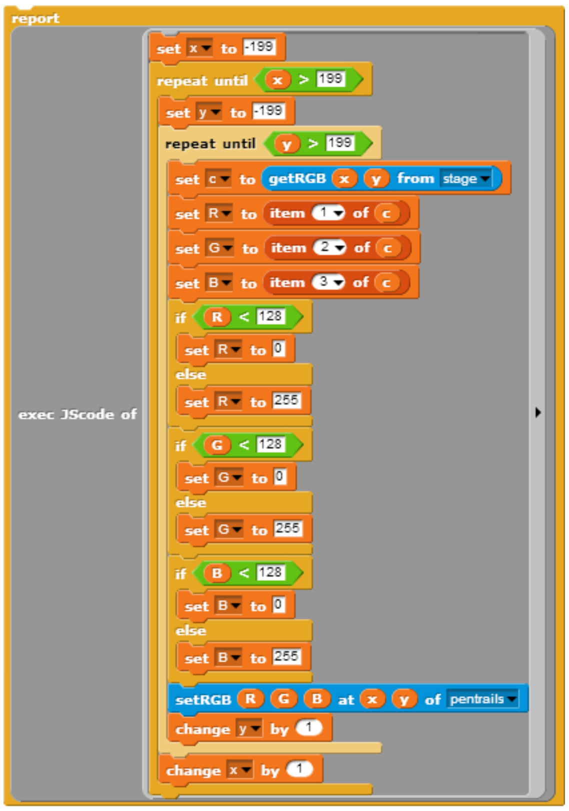
If we run the script ***color reduction***, it works, but it lasts very very long. So we use the ***code mapping*** ability of snap. SQLsnap fits JavaScript code to some blocks. We can show the code if we put a script in the ***code of*** block:



```
code of
  set x to -199
  repeat until (x > 199)
    set y to -199
    repeat until (y > 199)
      set c to getRGB x y from stage
      set R to item 1 of c
      set G to item 2 of c
      set B to item 3 of c
      if (R < 128)
        set R to 0
      else
        set R to 255
      if (G < 128)
        set G to 0
      else
        set G to 255
      if (B < 128)
        set B to 0
      else
        set B to 255
      setRGB R G B at x y of pentrails
      change y by 1
    change x by 1
```



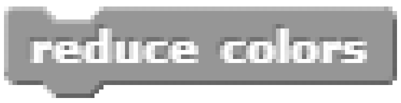
```
x = -199;
do{
   y = -199;
   do{
      c = getPoint(x,y,"stage");
      R = (c.length() < 1 ? new String() : (String("1")==="any") ? c.at(Math.floor(Math.random() * (c.length()) + 1)) :
                                           (String("1")==="last") ? c.at(c.length()) :
                                           1 < 1 ? new String() :  c.at(1));
      G = (c.length() < 1 ? new String() : (String("...
```



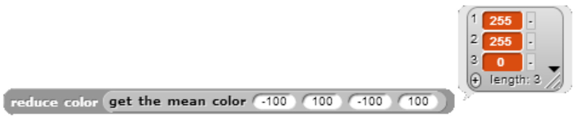
```
report
  exec JScode of
    set x to -199
    repeat until (x > 199)
      set y to -199
      repeat until (y > 199)
        set c to getRGB x y from stage
        set R to item 1 of c
        set G to item 2 of c
        set B to item 3 of c
        if (R < 128)
          set R to 0
        else
          set R to 255
        if (G < 128)
          set G to 0
        else
          set G to 255
        if (B < 128)
          set B to 0
        else
          set B to 255
        setRGB R G B at x y of pentrails
        change y by 1
      change x by 1
```

To run the code directly in JavaScript we use the appropriate ***exec JScode of*** block. If we execute this block, we get the changed image within some seconds.

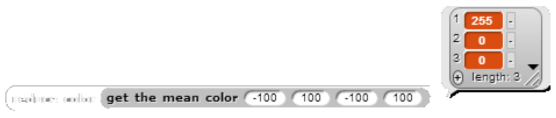
We encapsulate the ***exec JScode of*** with the embedded script in a new ***reduce colors*** block and have a very fast image processing block now.



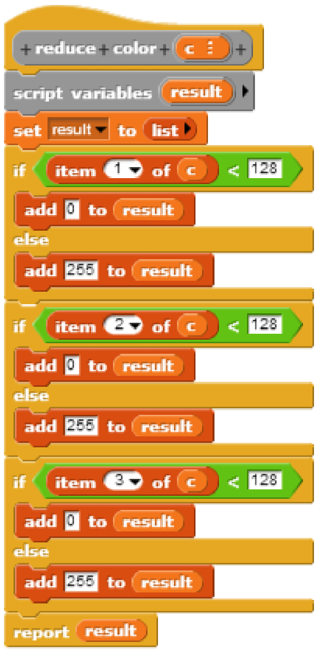
Now we calculate the mean color of the fruit and reduce the values again. If we do this with an orange, we get a nice yellow.



And an apple is red – of course.

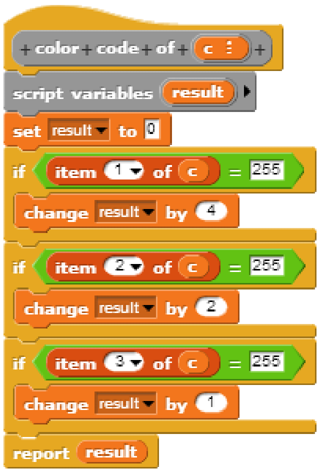


The ***reduce color*** block works as described: it gets a list with RGB-values and returns a list with a reduced color.



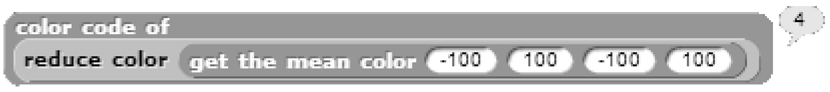
```
+reduce+color+(c :)+
script variables (result)
set result to list
if item 1 of c < 128
    add 0 to result
else
    add 255 to result
if item 2 of c < 128
    add 0 to result
else
    add 255 to result
if item 3 of c < 128
    add 0 to result
else
    add 255 to result
report result
```

Color codes:



```
+color+code+of+(c :)+
script variables (result)
set result to 0
if item 1 of c = 255
    change result by 4
if item 2 of c = 255
    change result by 2
if item 3 of c = 255
    change result by 1
report result
```

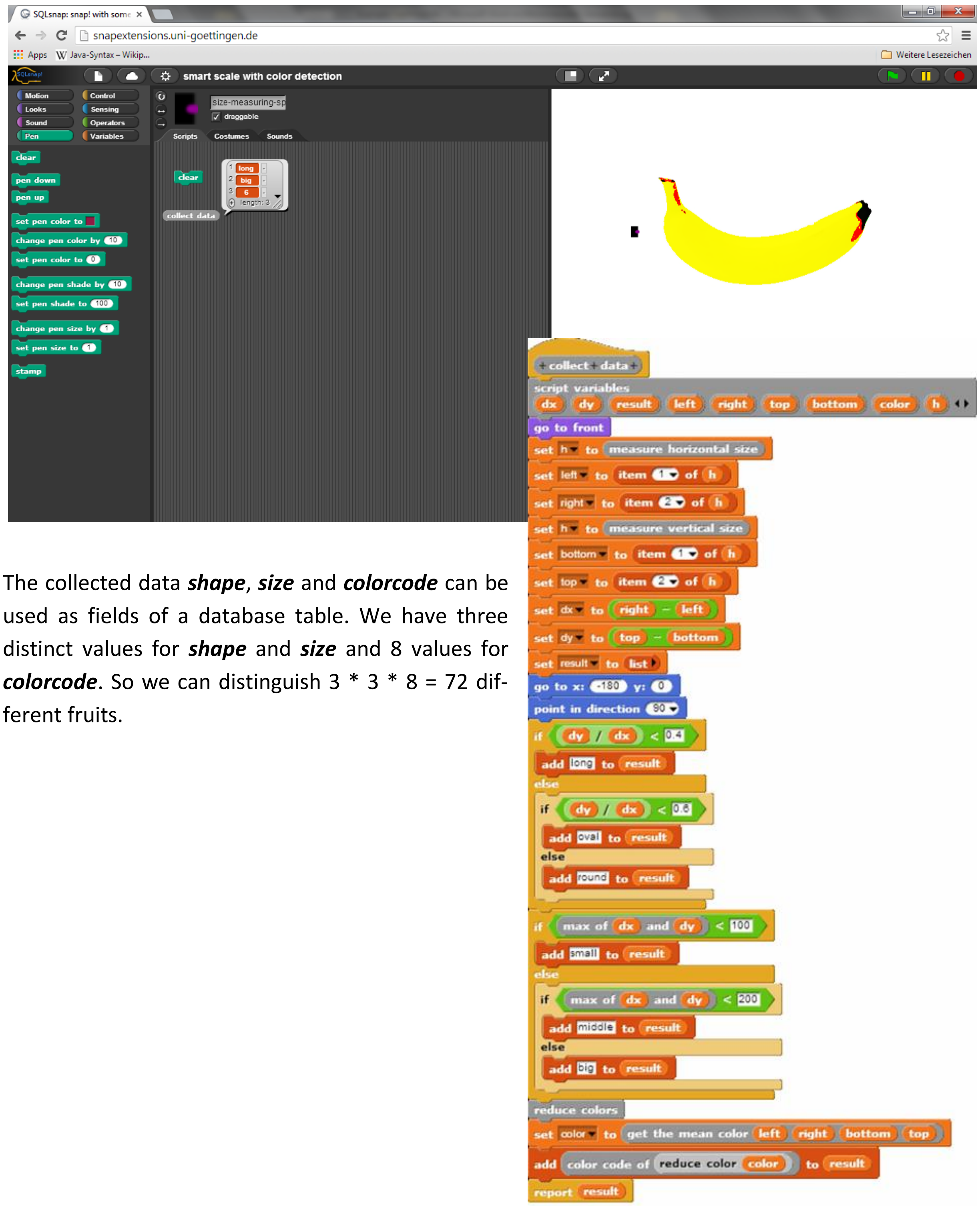
We can derive ***color codes*** from the reduced colors: if we interpret 255 as “1” and 0 as “0” we get a dual code: yellow is “110” (6) and red “100” (4).



```
color code of (reduce color (get the mean color -100 100 -100 100))
```

Now we have the full toolbox for fruit recognition:

1. Take an image of a fruit as background of the stage or "stamp" it to the pen trails. You can do this with help of a laptop camera or a smartphone. The background should be white.
2. Reduce the colors of the image.
3. Measure shape and size of the fruit.
4. Measure the mean color of the fruit and reduce it again.
5. Calculate the color code for the fruit.



```
collect data
script variables dx dy result left right top bottom color h
go to front
set h to measure horizontal size
set left to item 1 of h
set right to item 2 of h
set h to measure vertical size
set bottom to item 1 of h
set top to item 2 of h
set dx to (right - left)
set dy to (top - bottom)
set result to list
go to x: -180 y: 0
point in direction 90
if ((dy / dx) < 0.4)
    add long to result
else
    if ((dy / dx) < 0.8)
        add oval to result
    else
        add round to result
if ((max of dx and dy) < 100)
    add small to result
else
    if ((max of dx and dy) < 200)
        add middle to result
    else
        add big to result
reduce colors
set color to get the mean color left right bottom top
add color code of (reduce color color) to result
report result
```

The collected data ***shape***, ***size*** and ***colorcode*** can be used as fields of a database table. We have three distinct values for ***shape*** and ***size*** and 8 values for ***colorcode***. So we can distinguish 3 * 3 * 8 = 72 different fruits.

Exercises:

1. Build a table of the following type for fruits:

PNr	label	shape	size	color code
123	red apple	round	big	4
223	cherry	round	small	4
456	banana	long	big	6
…	…	…	…	…

2. Add a table ***fruits*** to your database.

3. Change the ***collect data*** procedure to report label and price of the scanned object. Use database commands to do this.

4. The color reduction process used is very rough. Invent a better method.

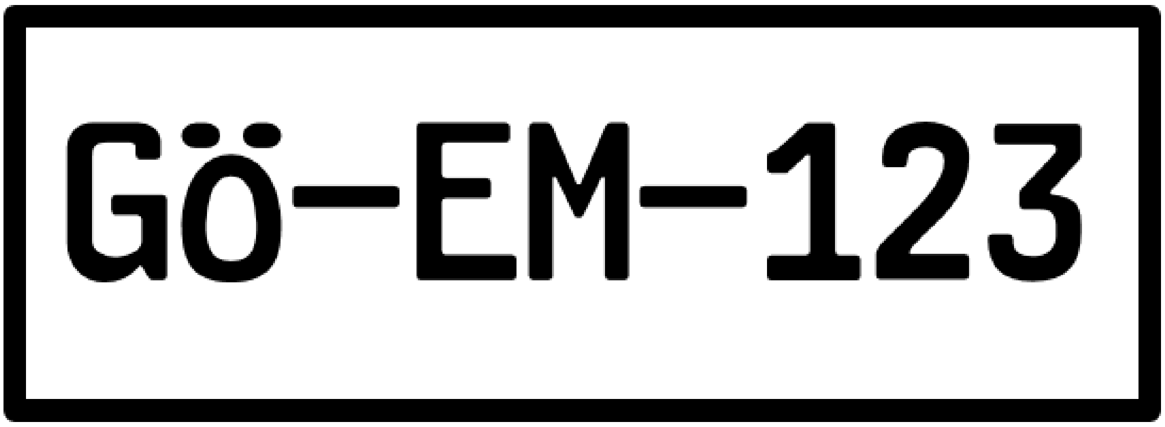
5. Our fruit recognition process only works well if we center the object in the middle and align it horizontal. If we take a sprite with an appropriate costume we can center and align it automatically before we leave a stamp. Do this.

6. If we use a more detailed color code we can describe more fruits. Would this be an advantage in all situations?

7. Maybe the background of the fruit image is not white. Can you help?

4. The security department

Our security department (among other things) is responsible for the parking garage. To simplify the payment procedure the department implements an automatic recognition of car number plates. Registered customers don’t need to stop in front of the barrier at the gate of the garage – that’s the hope.

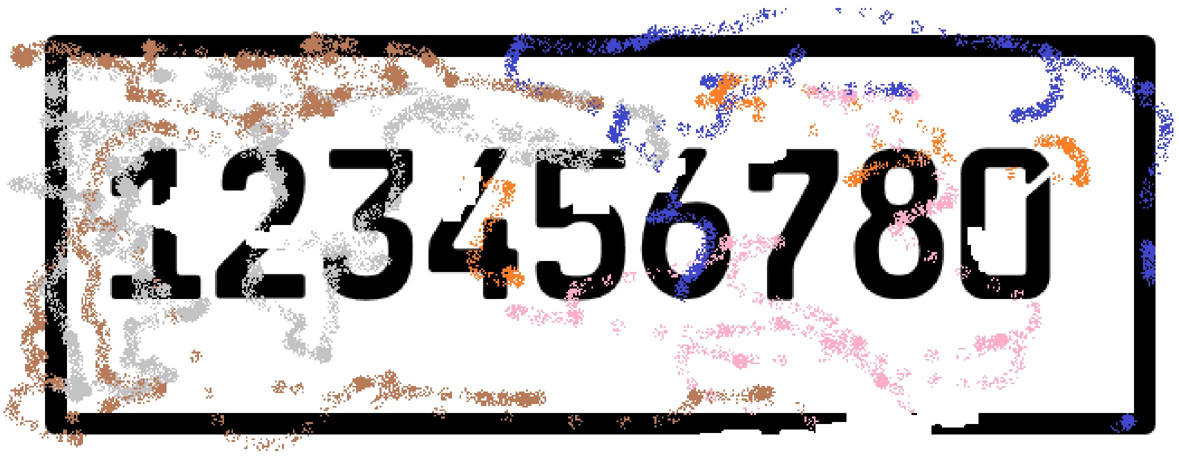


Car number plates have special fonts which are fine for pattern recognition. In Europe they have a black border – good for uns. We try to identify the numbers on the plate. (To recognize the other chars is your term.) Fortunately we have developed almost all tools to do this. You only have to ask the people at the smart scale.

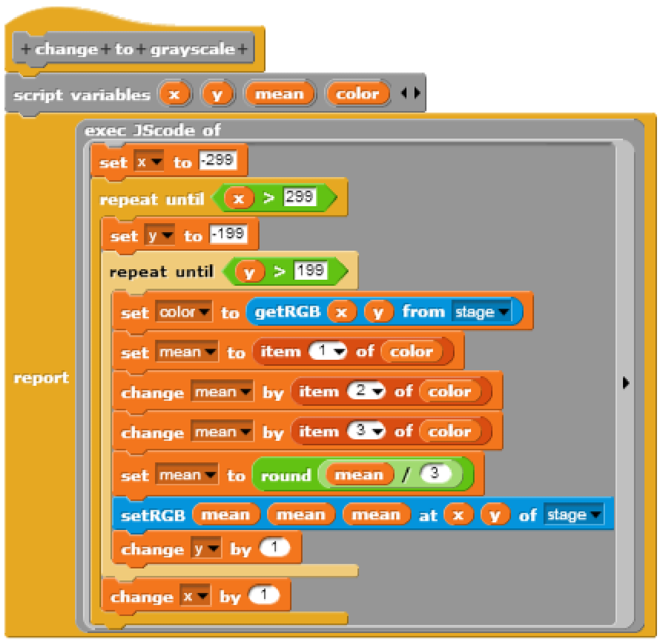
We try to find an extremely simple method to recognize chars on a car plate. The result will be very sensitive to changes in size and position of the plate. But these disadvantages are easily correctable by using more detailed measurements. Look on the exercises.

Dirty car plates

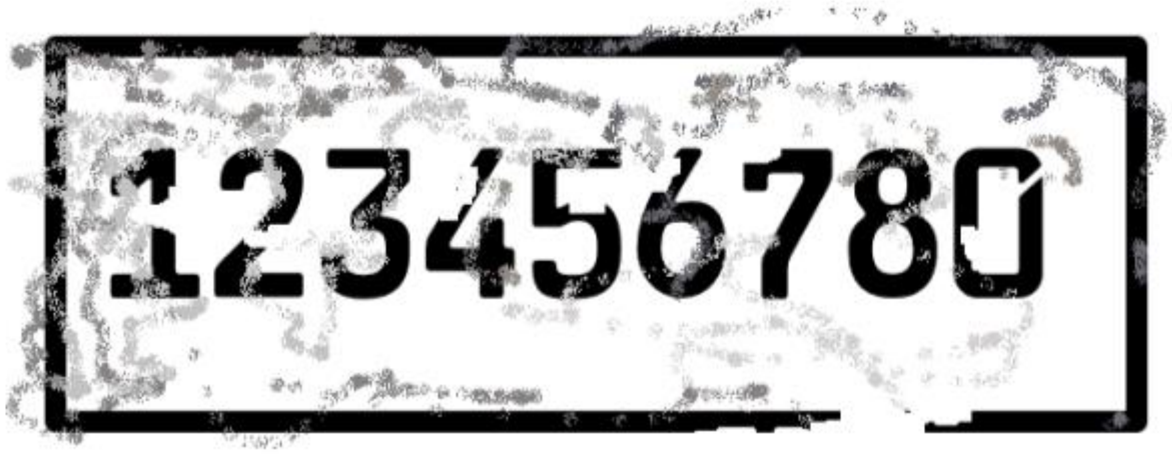
Normally car plates are not clean. They have some mud on it, and sometimes the black color has partly vanished in the car wash. What to do?



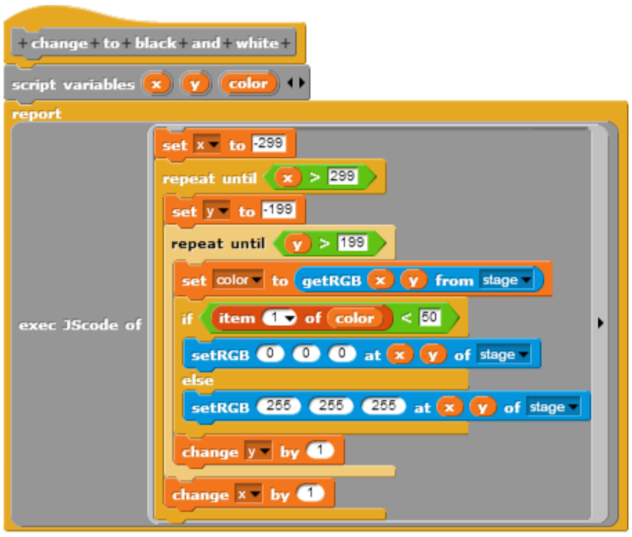
First we need a grayscale picture: we calculate the mean value of the red-, green- and blue-value of each pixel and take this as value for all three colors. The pixel will be gray. We store the result in the stage image. To speed up we put the script in the ***exec JScode of*** block and have the transformed picture after some seconds. We put this block in a new command block named ***change to grayscale***.



```
+ change + to + grayscale +
script variables x y mean color
report exec JScode of
    set x to -299
    repeat until x > 299
        set y to -199
        repeat until y > 199
            set color to getRGB x y from stage
            set mean to item 1 of color
            change mean by item 2 of color
            change mean by item 3 of color
            set mean to round mean / 3
            setRGB mean mean mean at x y of stage
            change y by 1
        change x by 1
```

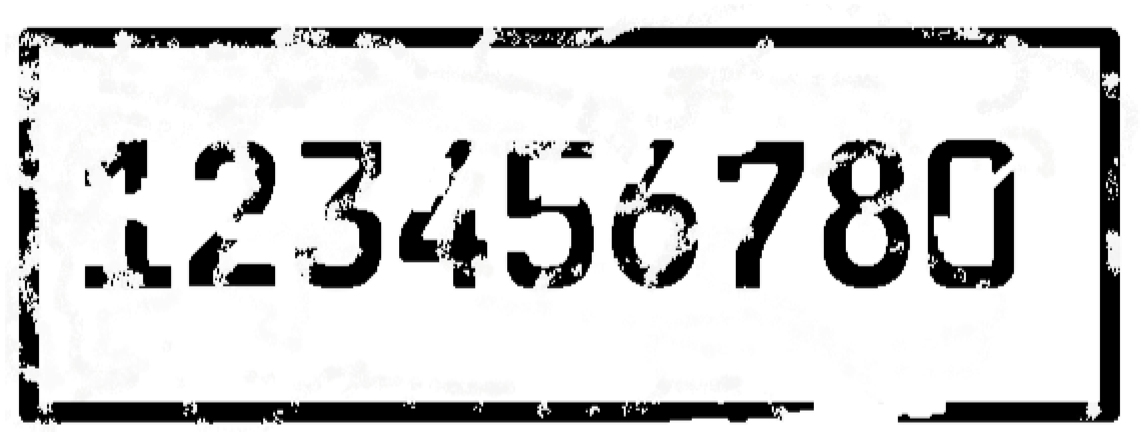


The next step is to transform the picture to a black-and-white one. We choose a threshold value (in this case ***50***) to split white and black pixels. (It would be clever to do this together with the grayscale transformation!) We build a ***change to black and white*** block.



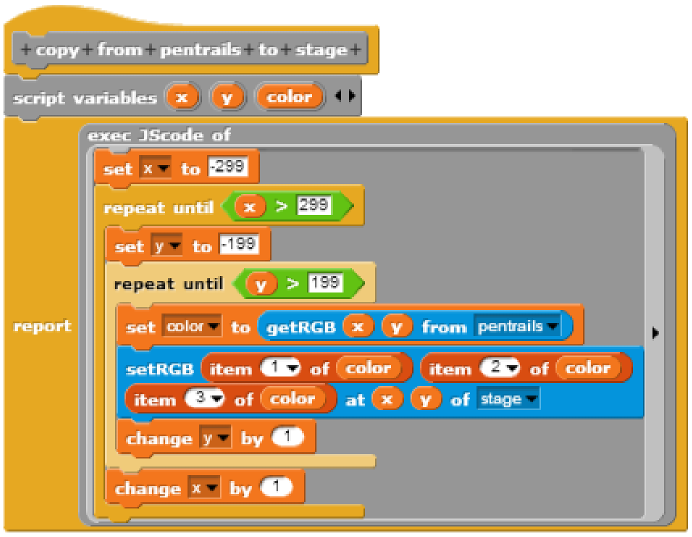
```
+change+to+black+and+white+
script variables x y color
report
  exec JScode of
    set x to -299
    repeat until x > 299
      set y to -199
      repeat until y > 199
        set color to getRGB x y from stage
        if item 1 of color < 50
          setRGB 0 0 0 at x y of stage
        else
          setRGB 255 255 255 at x y of stage
        change y by 1
      change x by 1
```

The result is:



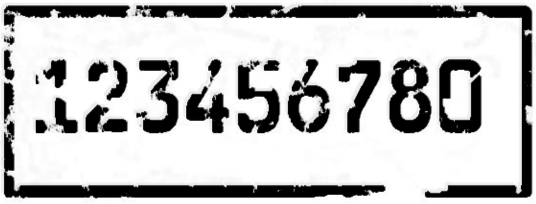
Closing the gaps

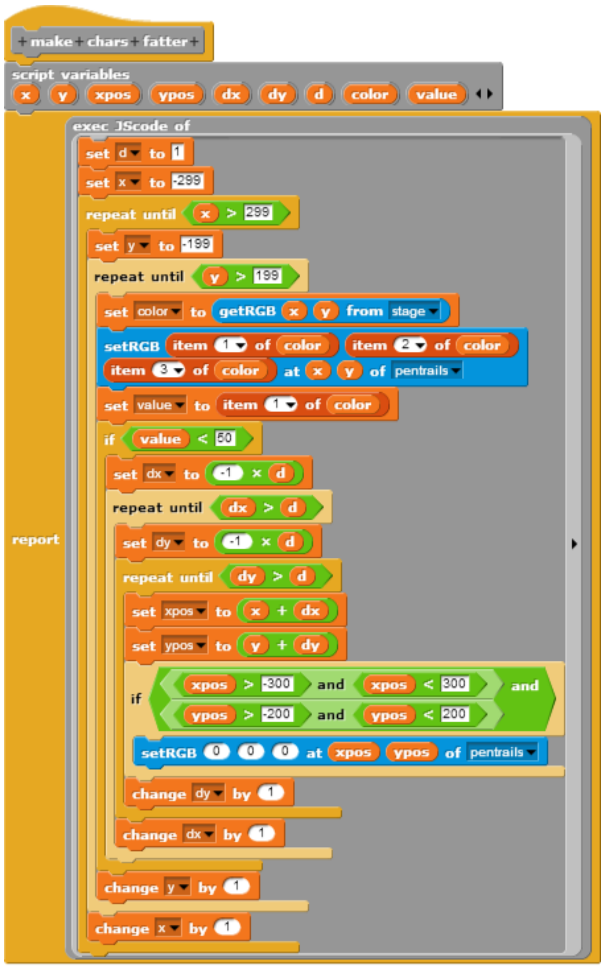
Now we have a nice black and white image of the car plate, but the chars are fragmented where had been mud. We try to close these gaps by filling them with black color. So we thicken the black parts of the image by attaching a black border to all black parts. We choose the thickness of this border to 1. But there is a problem. If we write changed pixels into the same picture we read from, the changes will be read same steps later. So we need a picture for reading and another for writing – and we have. We read from the stage and write to the pen trails. If we subsequently copy the changed image from pen trails to stage (***copy from pentrails to stage*** block) the process can be run again, and there are no collisions between original and changed pixels.



```
+copy+from+pentrails+to+stage+
script variables x y color
report
  exec JScode of
    set x to -299
    repeat until x > 299
      set y to -199
      repeat until y > 199
        set color to getRGB x y from pentrails
        setRGB item 1 of color item 2 of color item 3 of color at x y of stage
        change y by 1
      change x by 1
```

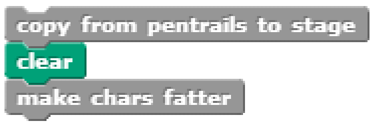
Result after one call of make chars fatter:





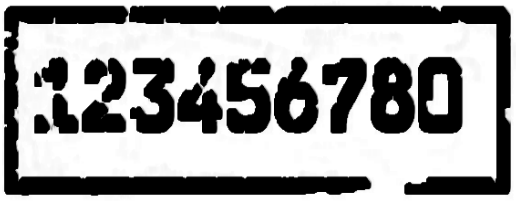
```
make chars fatter
script variables x y xpos ypos dx dy d color value
report exec JScode of
  set d to 1
  set x to -299
  repeat until x > 299
    set y to -199
    repeat until y > 199
      set color to getRGB x y from stage
      setRGB item 1 of color item 2 of color
        item 3 of color at x y of pentrails
      set value to item 1 of color
      if value < 50
        set dx to -1 × d
        repeat until dx > d
          set dy to -1 × d
          repeat until dy > d
            set xpos to x + dx
            set ypos to y + dy
            if xpos > -300 and xpos < 300 and
               ypos > -200 and ypos < 200
              setRGB 0 0 0 at xpos ypos of pentrails
            change dy by 1
          change dx by 1
      change y by 1
    change x by 1
```

After some more calls of the sequence

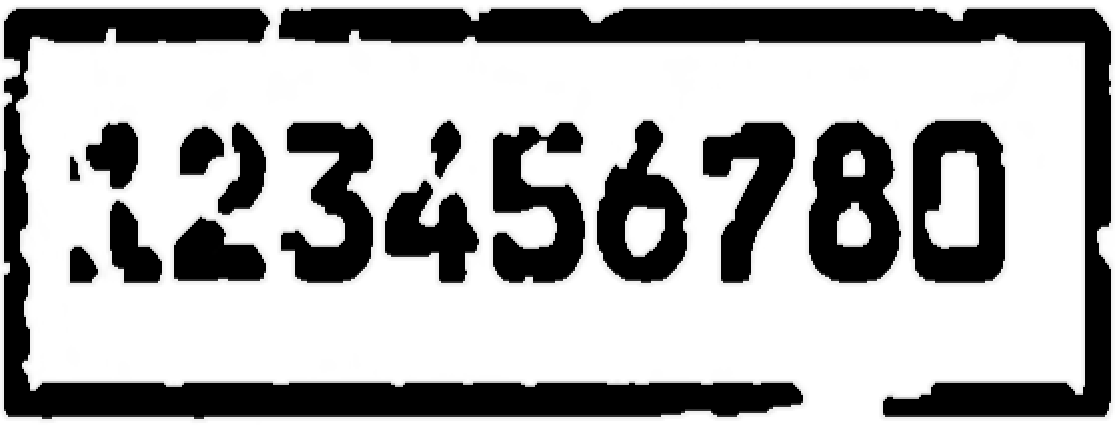


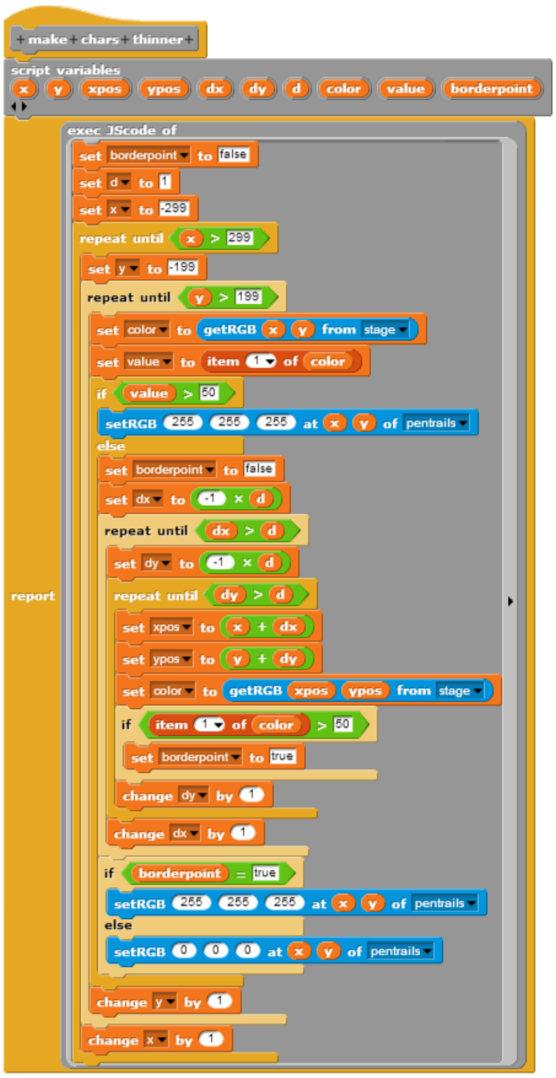
```
copy from pentrails to stage
clear
make chars fatter
```

we get:



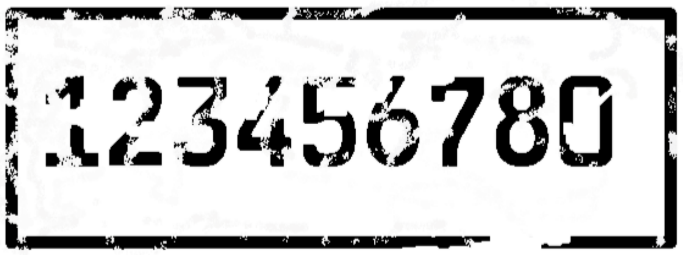
We need a method ***make chars thinner*** to scrape a layer of the black border – with the same problems as above – and the same solution (next page). The result of one call is:

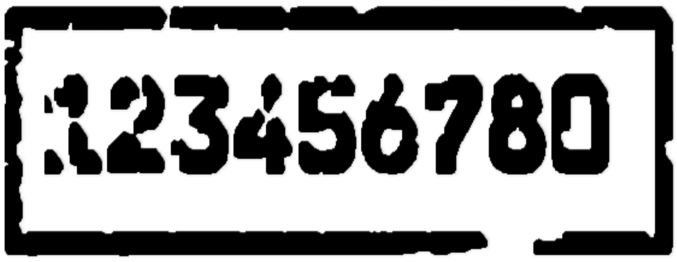




```
make chars thinner
script variables x y xpos ypos dx dy d color value borderpoint
report
  exec JScode of
    set borderpoint to false
    set d to 1
    set x to -299
    repeat until x > 299
      set y to -199
      repeat until y > 199
        set color to getRGB x y from stage
        set value to item 1 of color
        if value > 50
          setRGB 255 255 255 at x y of pentrails
        else
          set borderpoint to false
          set dx to -1 × d
          repeat until dx > d
            set dy to -1 × d
            repeat until dy > d
              set xpos to x + dx
              set ypos to y + dy
              set color to getRGB xpos ypos from stage
              if item 1 of color > 50
                set borderpoint to true
              change dy by 1
            change dx by 1
          if borderpoint = true
            setRGB 255 255 255 at x y of pentrails
          else
            setRGB 0 0 0 at x y of pentrails
        change y by 1
      change x by 1
```

We can switch between attaching black layers to existing black regions and scraping the layers again. It's a bit craftsmanship to find out the best combination of growing and melting phases to get readable letters. But the result is much more compact than in the beginning.

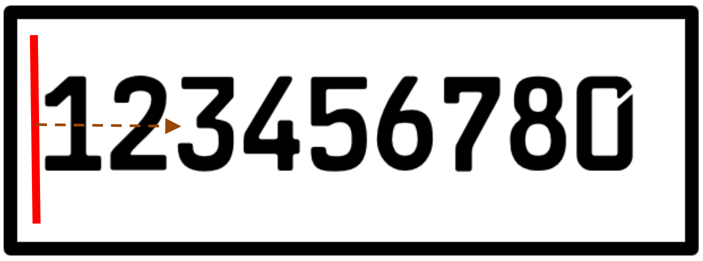


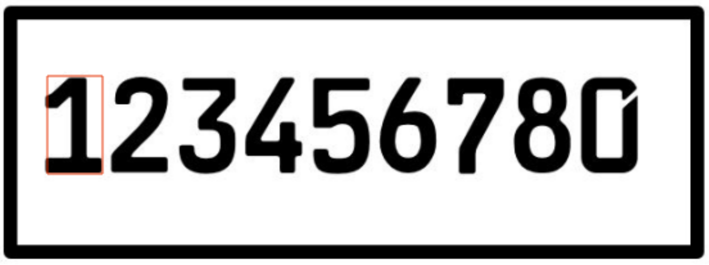


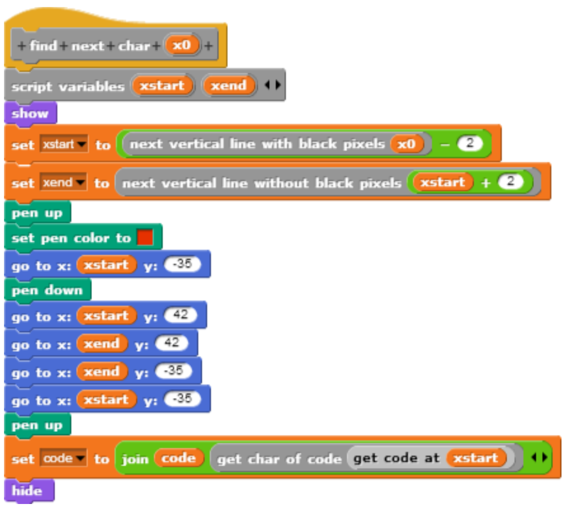
Character recognition

OCR (Optical Character Recognition) uses complex operations, often with neural networks, to recognize chars. We invent a simpler method, known from the smart scale. Because all characters have the same width, we can find them easily if we've found the border of the car plate. Learn from the smart scale scripts how to do this!

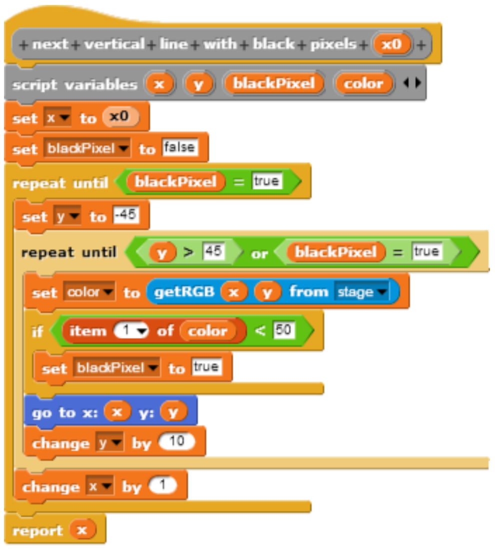
To simplify matters we use “clean plates”. We begin to search vertical lines on the plate with black pixels on it from left to right at the position x. If we’ve found the first one, we got the beginning of the first character. Now we look for the next vertical line without a black pixel. The x-position gives the end of the first character. We have a “window” with the first character inside. The next line with black pixels gives the width of the gap between chars. ***xpos*** is a global variable to store the beginning of the actual char.





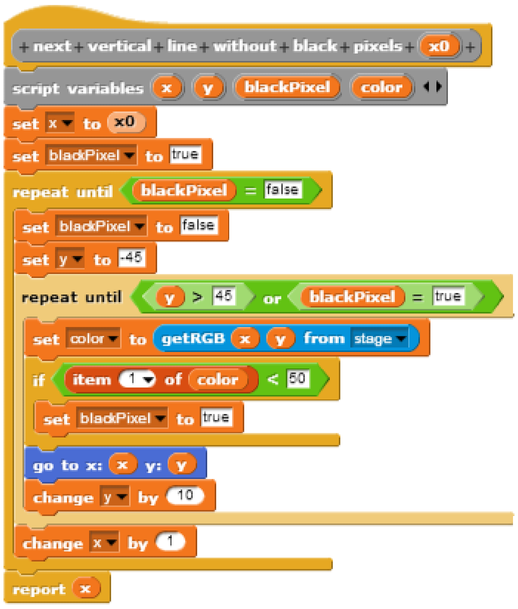


```
find next char (x0)
script variables (xstart) (xend)
show
set xstart to ((next vertical line with black pixels (x0)) − (2))
set xend to (next vertical line without black pixels ((xstart) + (2)))
pen up
set pen color to 🟥
go to x: (xstart) y: (-35)
pen down
go to x: (xstart) y: (42)
go to x: (xend) y: (42)
go to x: (xend) y: (-35)
go to x: (xstart) y: (-35)
pen up
set code to (join (code) (get char of code (get code at (xstart))))
hide
```



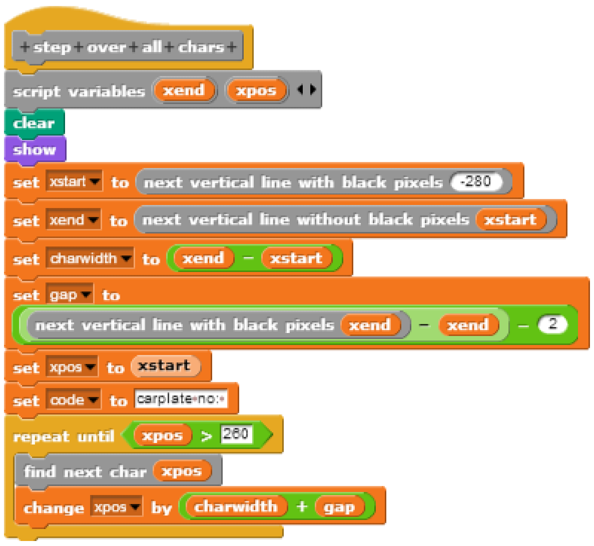
```
next vertical line with black pixels (x0)
script variables (x) (y) (blackPixel) (color)
set x to (x0)
set blackPixel to false
repeat until <(blackPixel) = true>
    set y to -45
    repeat until <<(y) > 45> or <(blackPixel) = true>>
        set color to (getRGB (x) (y) from stage)
        if <(item 1 of (color)) < 50>
            set blackPixel to true
        go to x: (x) y: (y)
        change y by 10
    change x by 1
report (x)
```

How to find the char code is described later.

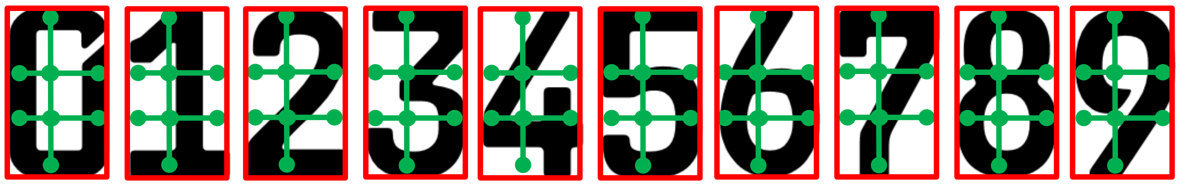


```
next vertical line without black pixels (x0)
script variables (x) (y) (blackPixel) (color)
set x to (x0)
set blackPixel to true
repeat until <(blackPixel) = false>
    set blackPixel to false
    set y to -45
    repeat until <<(y) > 45> or <(blackPixel) = true>>
        set color to (getRGB (x) (y) from stage)
        if <(item 1 of (color)) < 50>
            set blackPixel to true
        go to x: (x) y: (y)
        change y by 10
    change x by 1
report (x)
```

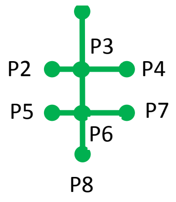
We can step over all characters with the red rectangle by calculating width of characters and gap between them first. The next script shows how to do this.


```
step over all chars
script variables xend xpos
clear
show
set xstart to next vertical line with black pixels -280
set xend to next vertical line without black pixels xstart
set charwidth to xend − xstart
set gap to
    ((next vertical line with black pixels xend) − xend) − 2
set xpos to xstart
set code to carplate-no:
repeat until xpos > 260
    find next char xpos
    change xpos by charwidth + gap
```

Now we try something like OCR. The starting point is that we have characters and rectangles around them.



We imagine a “sensor-field” of three crossing green lines for each character and measure the colors at the round points. We number the round points as shown. Let’s have a look on the results (gray: difficult to say).

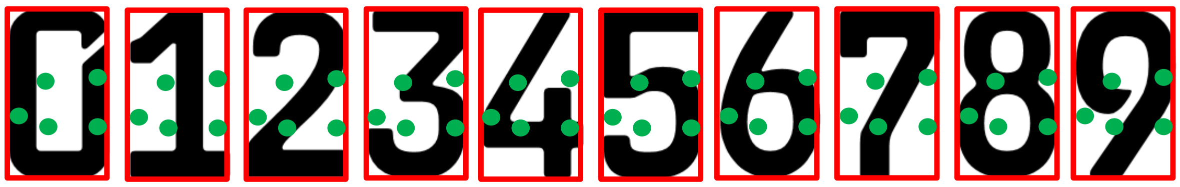


char	P1	P2	P3	P4	P5	P6	P7	P8	Code(s)
0	⬛	⬛	⬜	⬛	⬛	⬜	⬛	⬛	00100100
1	⬛	⬜	⬜	⬜	⬜	⬜	⬜	⬛	01111110
2	⬛	⬜	⬜	⬛	⬜	⬛	⬜	⬛	01101010
3	⬛	⬜	▨	⬜	⬜	⬜	⬛	⬛	01011100 01111100
4	⬜	⬜	⬛	⬜	⬛	⬛	⬛	⬜	11010001
5	⬛	⬛	⬛	⬛	⬜	⬜	⬛	⬛	00001100
6	⬛	⬜	⬛	⬛	⬛	⬜	⬛	⬛	0100100
7	⬛	⬜	⬜	⬜	⬜	⬛	⬜	⬛	01111010
8	⬛	▨	⬛	⬜	⬛	⬜	⬛	⬛	00010100 01010100
9	⬛	⬛	⬜	⬛	⬜	⬜	▨	⬛	00101100 00101110

At characters “3”, “8”, and “9” may occur errors if points P2, P3 and P7 aren’t well adjusted. But that doesn’t matter too much. If we shift the sensors P3, P4 and P7 a bit so that they deliver distinct values, we can abstain from sensors P1, P2 and P8 – for example – and have already a usable code.

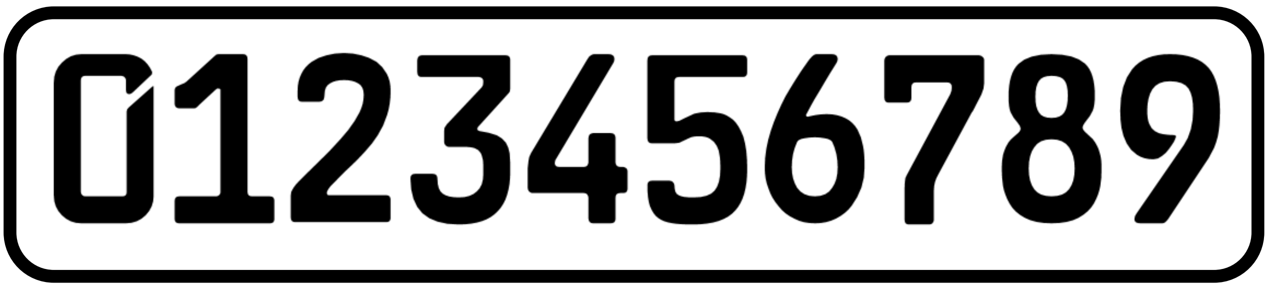
char	P1	P2	P3	P4	P5	P6	P7	P8	code(s)	value(s)
0	■	■	□	■	■	□	■	■	10010	18
1	■	□	□	□	□	□	□	■	11111	31
2	■	□	□	■	□	■	□	■	10101	21
3	■	□	□	□	□	□	■	■	11110	30
4	□	□	■	□	■	■	■	□	01000	8
5	■	■	■	■	□	□	■	■	00110	6
6	■	□	■	■	■	□	■	■	00010	2
7	■	□	□	□	□	■	□	■	11101	29
8	■	▨	■	□	■	□	■	■	01010	10
9	■	■	□	■	□	□	□	■	10111	23

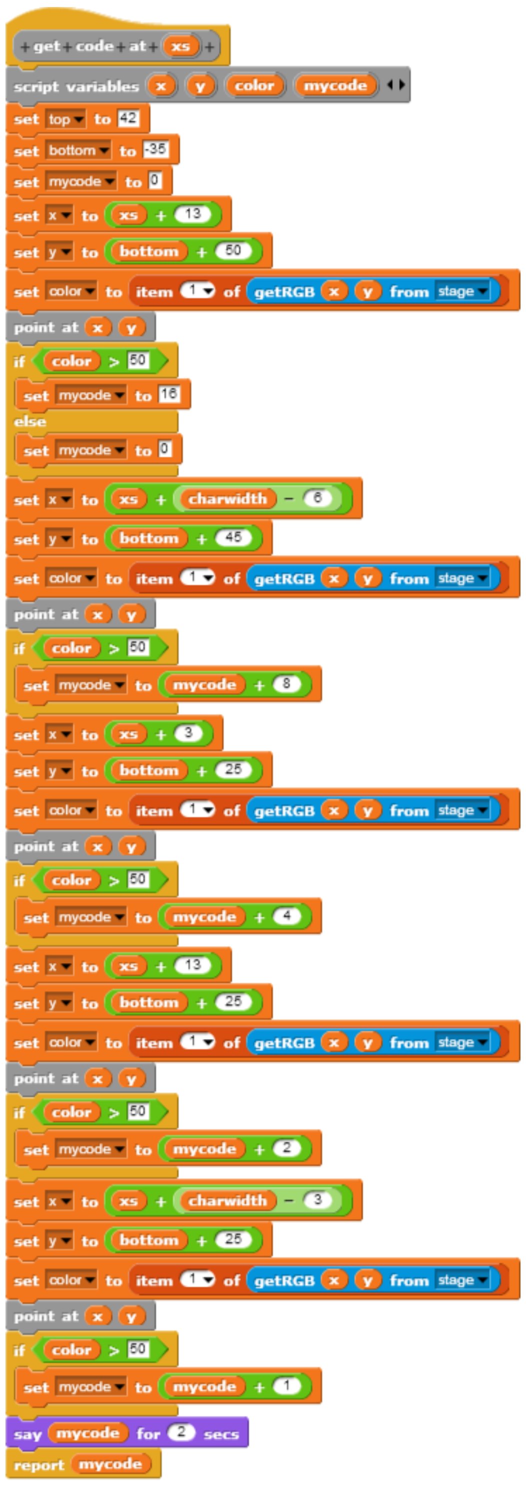
A practicable layout for the sensor-points could be:



Let’s go!

We take a car plate with all numbers on it as image for the stage.

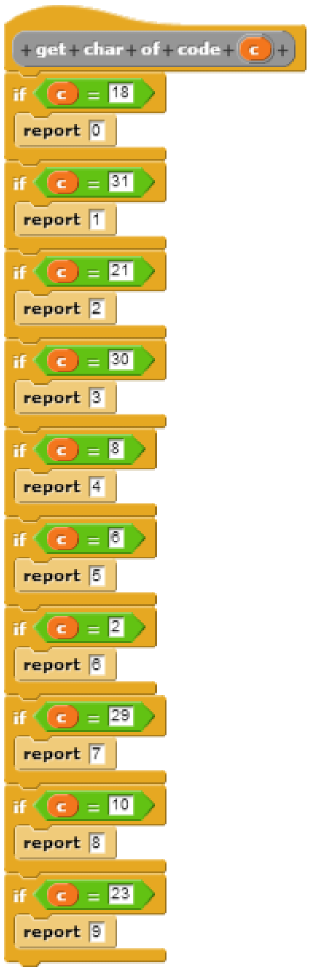




```
get code at (xs)
script variables (x) (y) (color) (mycode)
set top to 42
set bottom to -35
set mycode to 0
set x to (xs + 13)
set y to (bottom + 50)
set color to item 1 of getRGB x y from stage
point at x y
if color > 50
    set mycode to 16
else
    set mycode to 0
set x to (xs + (charwidth - 6))
set y to (bottom + 45)
set color to item 1 of getRGB x y from stage
point at x y
if color > 50
    set mycode to (mycode + 8)
set x to (xs + 3)
set y to (bottom + 25)
set color to item 1 of getRGB x y from stage
point at x y
if color > 50
    set mycode to (mycode + 4)
set x to (xs + 13)
set y to (bottom + 25)
set color to item 1 of getRGB x y from stage
point at x y
if color > 50
    set mycode to (mycode + 2)
set x to (xs + (charwidth - 3))
set y to (bottom + 25)
set color to item 1 of getRGB x y from stage
point at x y
if color > 50
    set mycode to (mycode + 1)
say mycode for 2 secs
report mycode
```

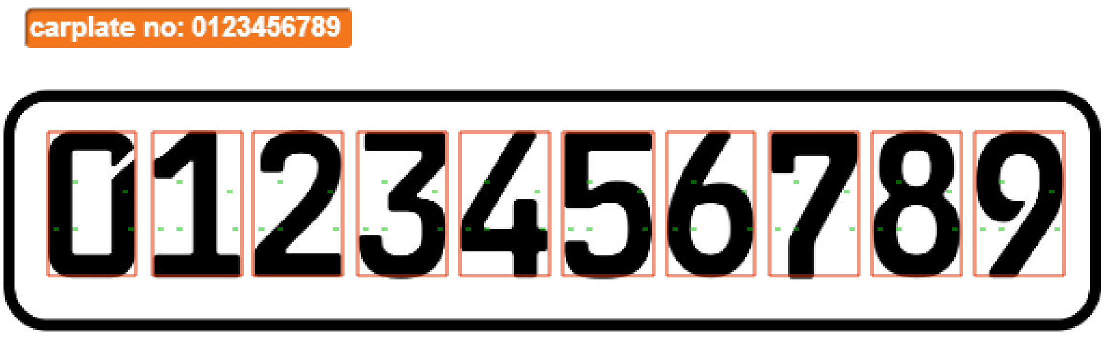
We have to place the “sensors” at appropriate places (here: (13|50), …), to read the color from the stage and to build a code number from the colors interpreted as dual numbers.

If we have done this, we transform the code number into a plate number with:



```
get char of code (c)
if c = 18
    report 0
if c = 31
    report 1
if c = 21
    report 2
if c = 30
    report 3
if c = 8
    report 4
if c = 6
    report 5
if c = 2
    report 6
if c = 29
    report 7
if c = 10
    report 8
if c = 23
    report 9
```

We painted green dots at the sensor positions and a red frame round the characters – and the result is what we wanted.



Exercises:

1. In the examples only the horizontal alignment of the car-plate is measured. Find out the vertical position as well. In the examples the sensorpositions in the char-rectangle are given absolutely ("13 pixel right from border", …). Address them relative to the size of the char-rectangle.

2. The pattern recognition in the examples is very simple, but very sensitive against changes in position and size of the car-plate. Use more sensors on better positions to identify the numbers on the plate.

3. If you have dirty car-plates the chars have no sharp borders. In consequence some sensors will produce errors. Better the results by finding the "nearest correct code" of a wrong code.

4. Recognition programs can learn. If the script finds a not identifiable pattern, it should show its result and ask for the correct character. But learned pattern and chars in a database-table and use queries to identify new unknown patterns.

5. Use a sensor-field with more sensors. Try to identify more characters of your national font.

6. The security department needs a database with license plates and car-owners and their status (customer, member of the staff, persona non grata, extern, …). Can you help?

7. Car-plate recognition is a big success of the security department. All members are very proud and the staff admires the "sheriffs". The advertising department wants to use the data of the car-plate-table to honor frequent customers as VIP-customer. They get special parking areas near the elevator. Write a query to identify VIP-customers.

8. After a while the VIP-customer area is occupied by the cars of pensioners and jobless persons. So the advertising department extends the criteria for VIP-customers to a minimum volume of sales. Because almost all customers use credit-cards that's no problem. Better your VIP-customer-query.

9. The advertising department considers that it would be useful to know not only, how much sales volume a customer produces, but what he has bought. If they know the interests of the customers they can supply them with specific promotion and special prices. Identify necessary new tables and columns in the database to do this. Write an appropriate query.

10. The advertising department wants to know whether their promotion campaigns are successful. Did they reach the customers? Try to answer the question based on the stored data.

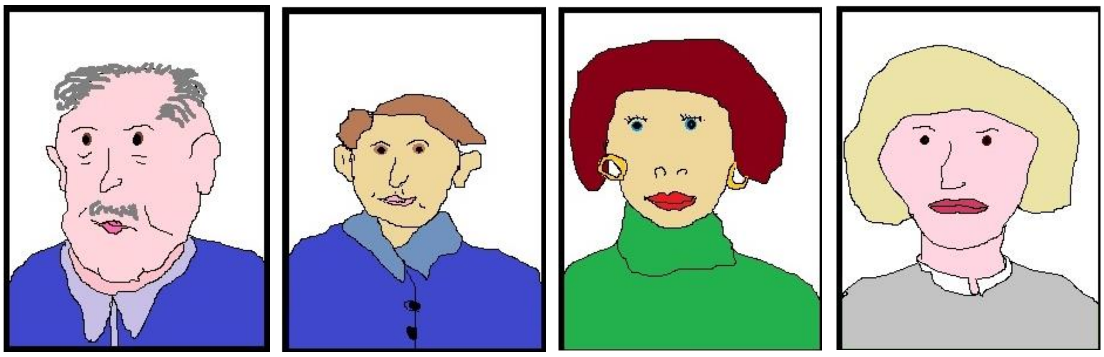
11. In German highways the toll for trucks is determined by toll-collect-barriers which read the car-plates crossing. They read ALL car-plates and delete the identification numbers of passenger cars. Why is this appropriate? Discuss the consequences of storing all numbers and positions.

5. The advertising department

The advertising department is emphasized about the scope of character recognition and wants to expand this area: they try to find out who entered the supermarket. With a face recognition program the customers shall be identified.

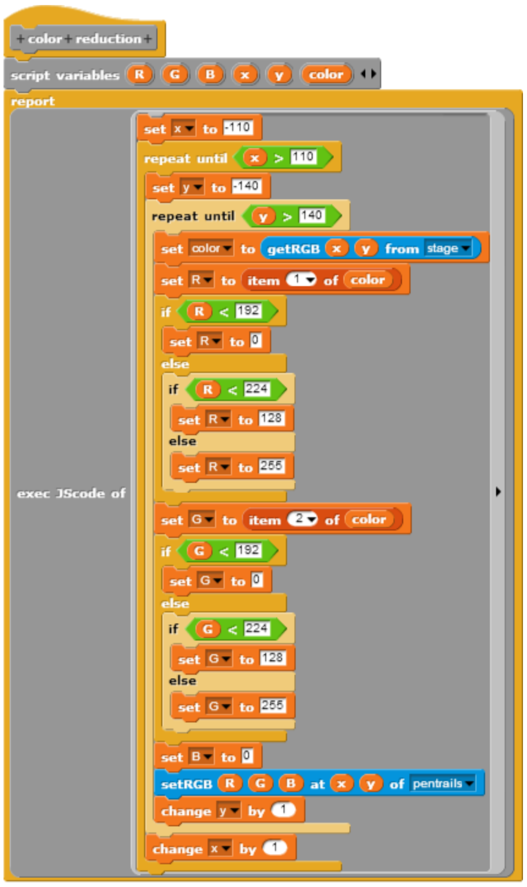
Face recognition

We try this on a similar way as fruit recognition. At first we draw some faces (similar to passport photos) and try to identify them.

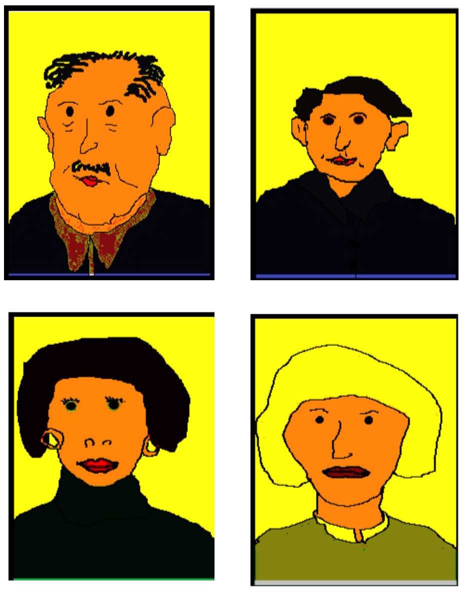


Paul Peter Mary Hannah

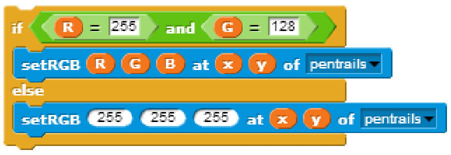
Let's look for faces. We do a color reduction, but a bit more accurate as with the fruits. We ignore the blue-channel of the pixels and reduce the red- and green-values to three numbers: 0, 128 and 255. The result of this simple transformation is quite good: the face-color is now always the same.



```
+ color + reduction +
script variables (R) (G) (B) (x) (y) (color)
report
  exec JScode of
    set x to -110
    repeat until (x > 110)
      set y to -140
      repeat until (y > 140)
        set color to getRGB (x) (y) from stage
        set R to item 1 of (color)
        if (R < 192)
          set R to 0
        else
          if (R < 224)
            set R to 128
          else
            set R to 255
        set G to item 2 of (color)
        if (G < 192)
          set G to 0
        else
          if (G < 224)
            set G to 128
          else
            set G to 255
        set B to 0
        setRGB (R) (G) (B) at (x) (y) of pentrails
        change y by 1
      change x by 1
```

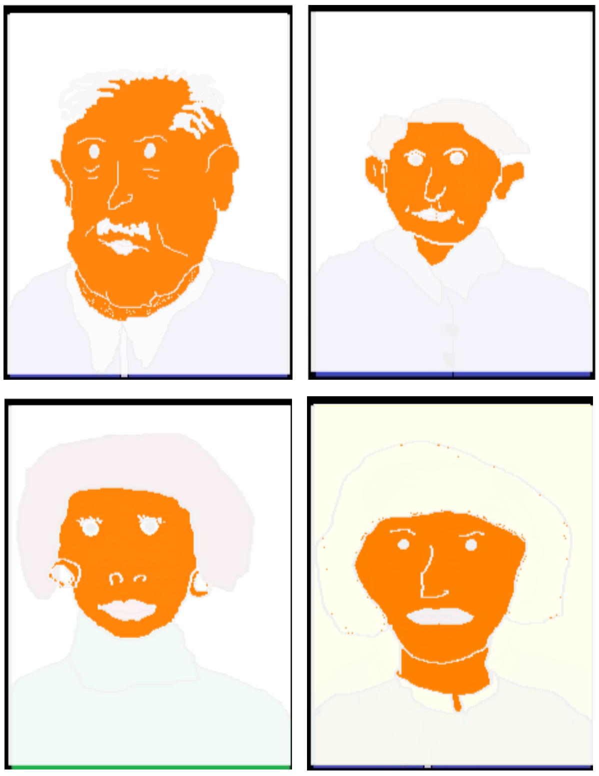


The next step is to ignore everything except the faces: all different colors than orange are set to white. We add …



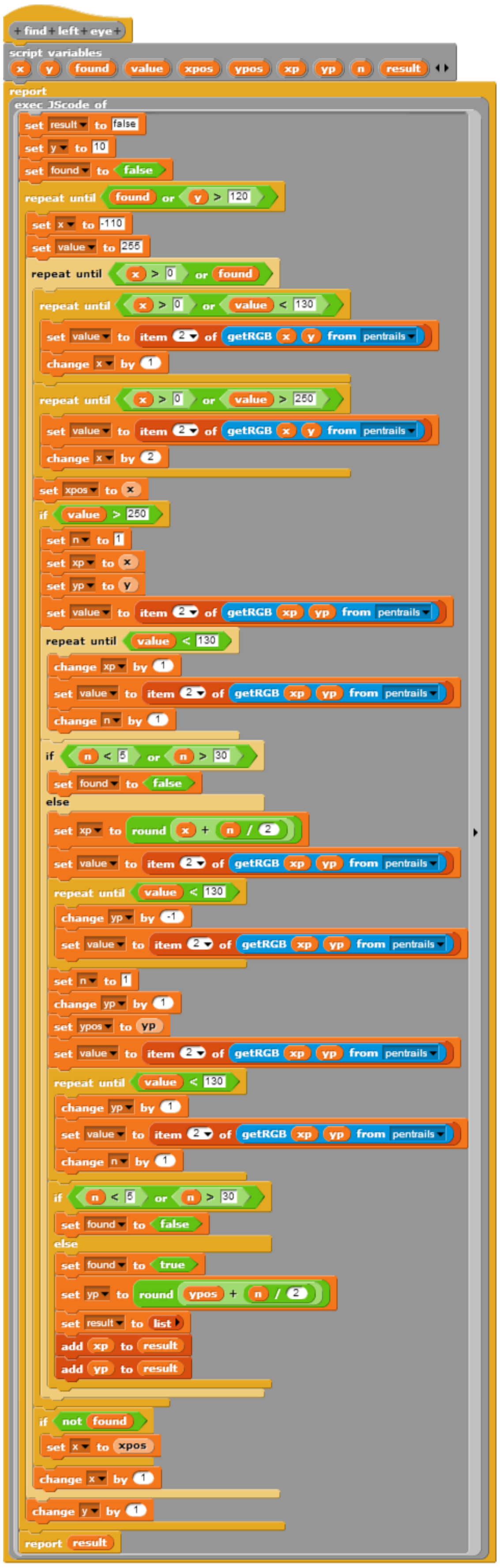
```
if <<(R) = 255> and <(G) = 128>>
    setRGB (R) (G) (B) at (x) (y) of pentrails
else
    setRGB 255 255 255 at (x) (y) of pentrails
```

… and get …



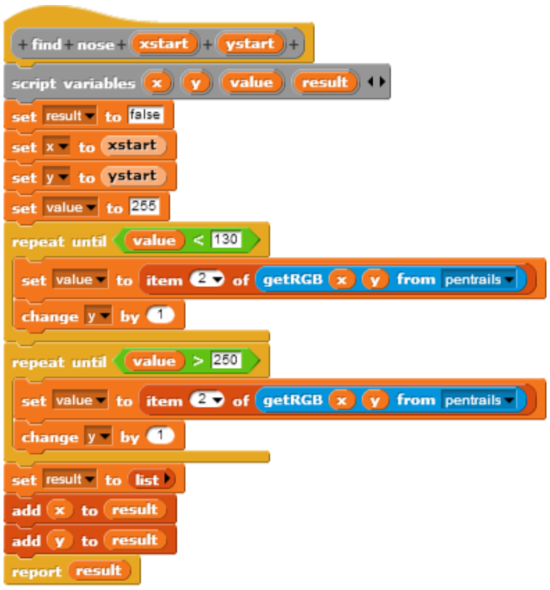
We've got the faces, but we need parameters to describe them! We use size ratios similar to the fruit. So we measure the distance between the eyes and the distance to the mouth, the length of the nose compared with the width of the face, …

"Eyes" and "mouth" are holes in the orange face. At first we are looking for the eyes. The left one should be above the middle and left of them on a passport photo. We should find there white pixels with some layers of white pixels around them. (Remember ***make chars fatter***!)



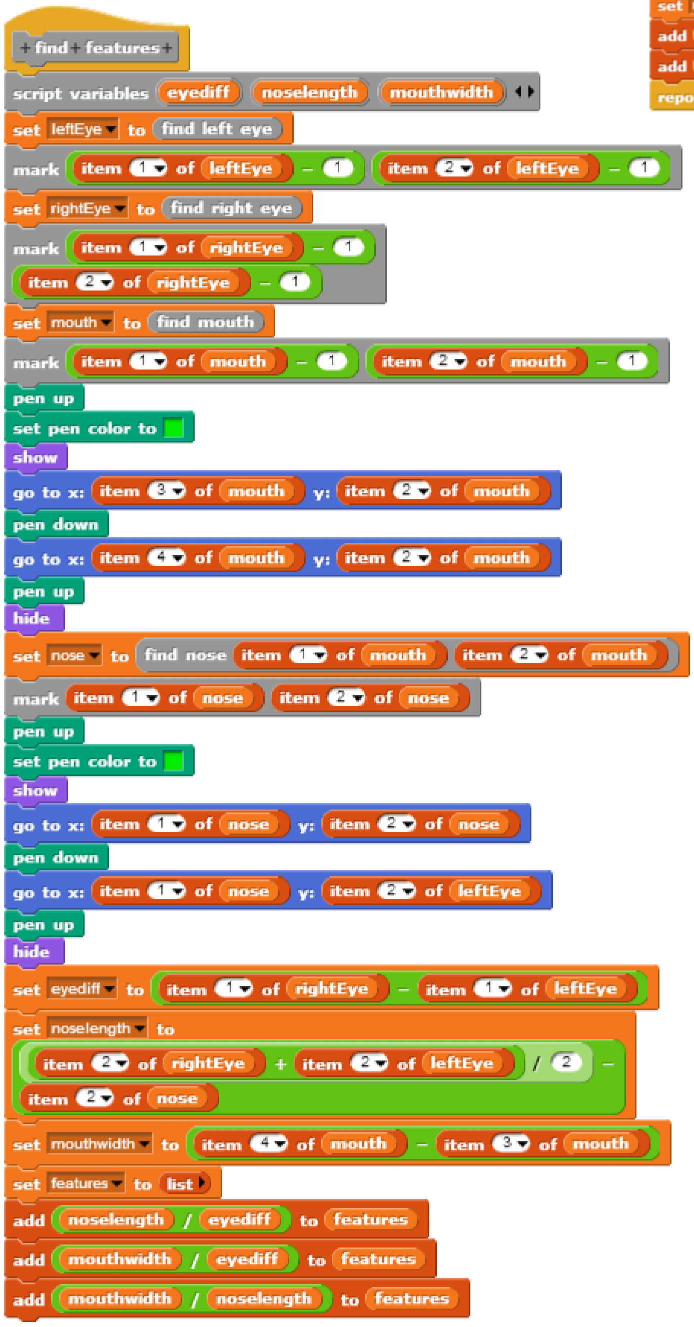
```
+ find + left + eye +
script variables (x) (y) (found) (value) (xpos) (ypos) (xp) (yp) (n) (result)
report
  exec JScode of
    set result to false
    set y to 10
    set found to <false>
    repeat until <(found) or <(y) > 120>>
        set x to -110
        set value to 255
        repeat until <<(x) > 0> or (found)>
            repeat until <<(x) > 0> or <(value) < 130>>
                set value to (item 2 of (getRGB (x) (y) from pentrails))
                change x by 1
            repeat until <<(x) > 0> or <(value) > 250>>
                set value to (item 2 of (getRGB (x) (y) from pentrails))
                change x by 2
            set xpos to (x)
            if <(value) > 250>
                set n to 1
                set xp to (x)
                set yp to (y)
                set value to (item 2 of (getRGB (xp) (yp) from pentrails))
                repeat until <(value) < 130>
                    change xp by 1
                    set value to (item 2 of (getRGB (xp) (yp) from pentrails))
                    change n by 1
                if <<(n) < 5> or <(n) > 30>>
                    set found to <false>
                else
                    set xp to (round ((x) + ((n) / 2)))
                    set value to (item 2 of (getRGB (xp) (yp) from pentrails))
                    repeat until <(value) < 130>
                        change yp by -1
                        set value to (item 2 of (getRGB (xp) (yp) from pentrails))
                    set n to 1
                    change yp by 1
                    set ypos to (yp)
                    set value to (item 2 of (getRGB (xp) (yp) from pentrails))
                    repeat until <(value) < 130>
                        change yp by 1
                        set value to (item 2 of (getRGB (xp) (yp) from pentrails))
                        change n by 1
                    if <<(n) < 5> or <(n) > 30>>
                        set found to <false>
                    else
                        set found to <true>
                        set yp to (round ((ypos) + ((n) / 2)))
                        set result to (list)
                        add (xp) to (result)
                        add (yp) to (result)
            if <not (found)>
                set x to (xpos)
            change x by 1
        change y by 1
    report (result)
```

The right eye could be found on the same way as the left one. Only the start-coordinates are different. The same is for the mouth, but the mout should be bigger th an an eye. An the nose is marked by the first white pixel above the mouth.



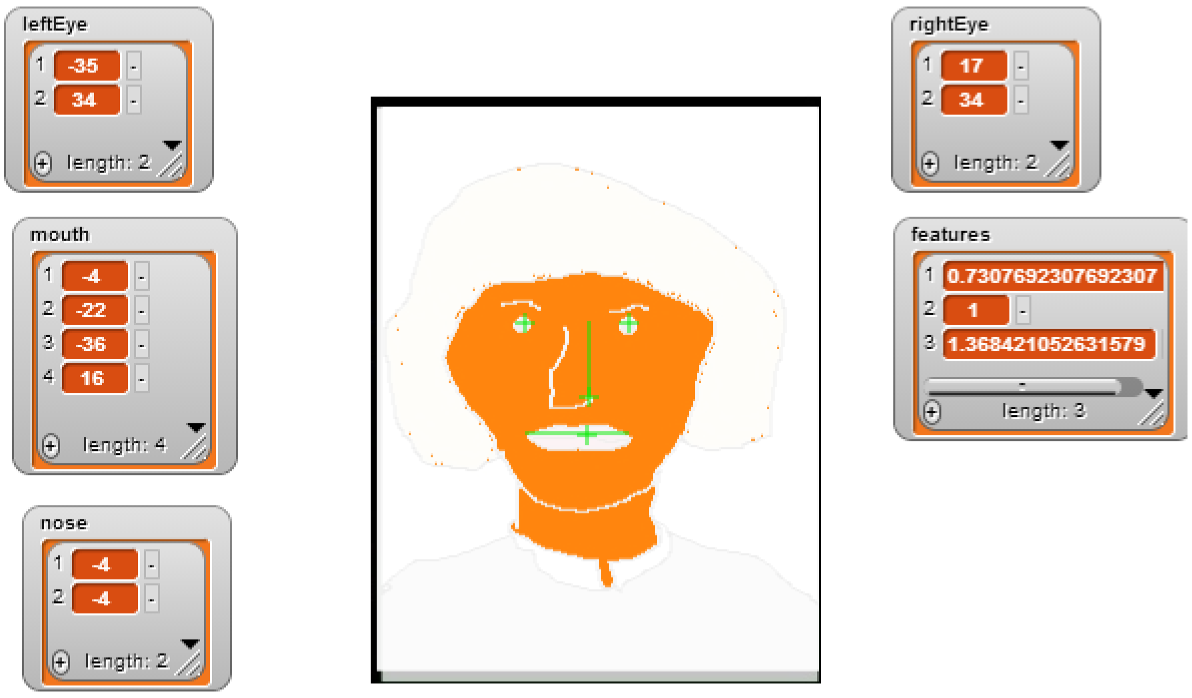
```
+find+nose+ xstart + ystart +
script variables x y value result
set result to false
set x to xstart
set y to ystart
set value to 255
repeat until value < 130
    set value to item 2 of getRGB x y from pentrails
    change y by 1
repeat until value > 250
    set value to item 2 of getRGB x y from pentrails
    change y by 1
set result to list
add x to result
add y to result
report result
```

So we can measue the positions of the eyes, nose and mouth and can calculate their distances and the ratios between these values. In ***find features*** these results are marked green.

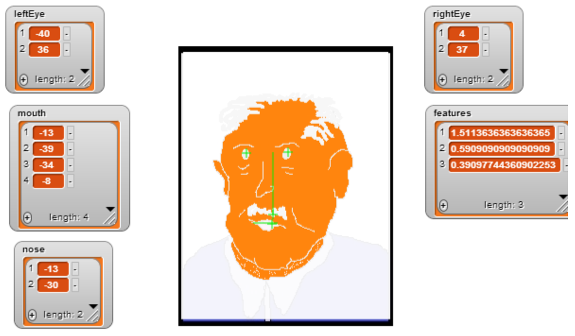


```
+find+features+
script variables eyediff noselength mouthwidth
set leftEye to find left eye
mark (item 1 of leftEye) - 1 (item 2 of leftEye) - 1
set rightEye to find right eye
mark (item 1 of rightEye) - 1
     (item 2 of rightEye) - 1
set mouth to find mouth
mark (item 1 of mouth) - 1 (item 2 of mouth) - 1
pen up
set pen color to 🟩
show
go to x: item 3 of mouth y: item 2 of mouth
pen down
go to x: item 4 of mouth y: item 2 of mouth
pen up
hide
set nose to find nose item 1 of mouth item 2 of mouth
mark item 1 of nose item 2 of nose
pen up
set pen color to 🟩
show
go to x: item 1 of nose y: item 2 of nose
pen down
go to x: item 1 of nose y: item 2 of leftEye
pen up
hide
set eyediff to (item 1 of rightEye) - (item 1 of leftEye)
set noselength to
    ((item 2 of rightEye + item 2 of leftEye) / 2) -
    item 2 of nose
set mouthwidth to (item 4 of mouth) - (item 3 of mouth)
set features to list
add noselength / eyediff to features
add mouthwidth / eyediff to features
add mouthwidth / noselength to features
```

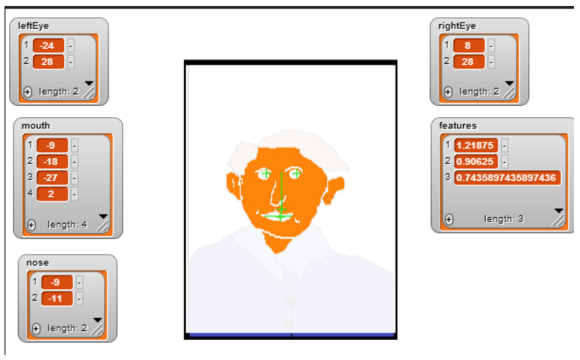
The results are:



```
leftEye
1 -35
2 34
length: 2

rightEye
1 17
2 34
length: 2

mouth
1 -4
2 -22
3 -36
4 16
length: 4

features
1 0.7307692307692307
2 1
3 1.368421052631579
length: 3

nose
1 -4
2 -4
length: 2
```

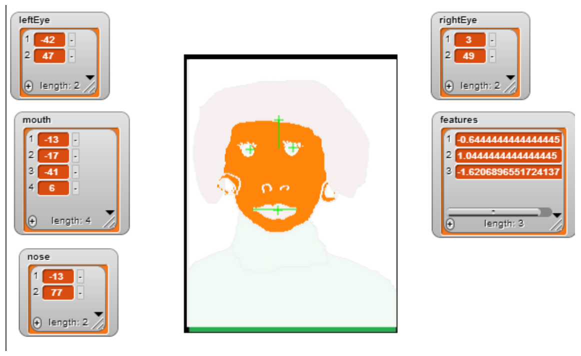


```
leftEye
1 -40
2 36
length: 2

rightEye
1 4
2 37
length: 2

mouth
1 -13
2 -39
3 -34
4 -8
length: 4

features
1 1.5113636363636365
2 0.5909090909090909
3 0.39097744360902253
length: 3

nose
1 -13
2 -30
length: 2
```



```
leftEye
1 -24
2 28
length: 2

rightEye
1 8
2 28
length: 2

mouth
1 -9
2 -18
3 -27
4 2
length: 4

features
1 1.21875
2 0.90625
3 0.7435897435897436
length: 3

nose
1 -9
2 -11
length: 2
```

If there is no real nose on the picture, the result is consequently wrong.

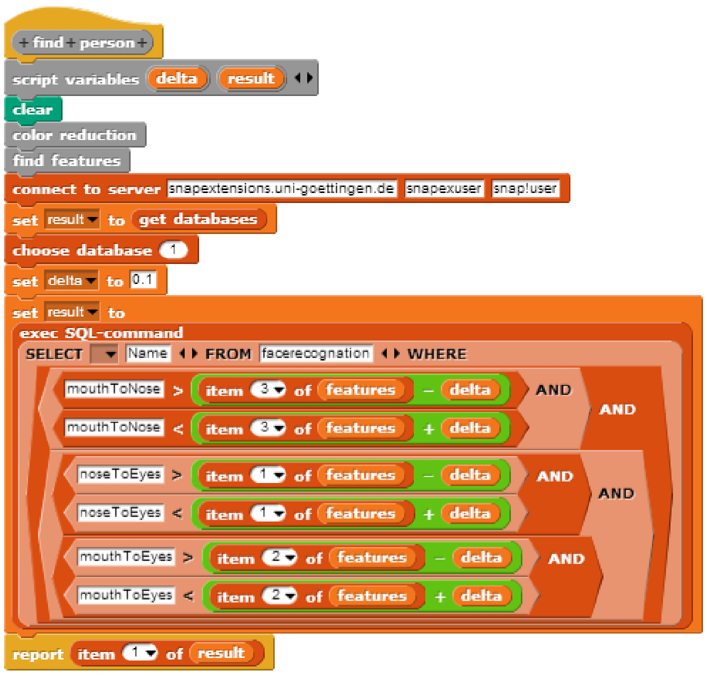


```
leftEye
1 -42
2 47
length: 2

rightEye
1 3
2 49
length: 2

mouth
1 -13
2 -17
3 -41
4 6
length: 4

features
1 -0.6444444444444445
2 1.0444444444444445
3 -1.6206896551724137
length: 3

nose
1 -13
2 77
length: 2
```

Now we can put the results in a database-table.

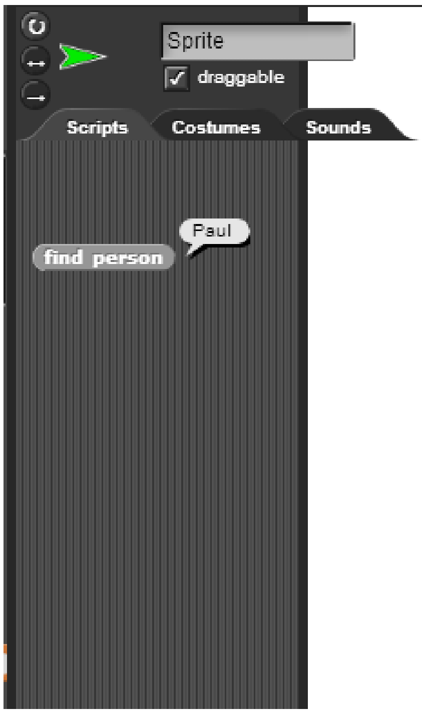
Name	noseToEyes	mouthToEyes	mouthToNose
Peter	1.22	0.91	0.74
Paul	1.59	0.59	0.39
Mary	0	1.04	0
Hannah	0.73	1.00	1.37

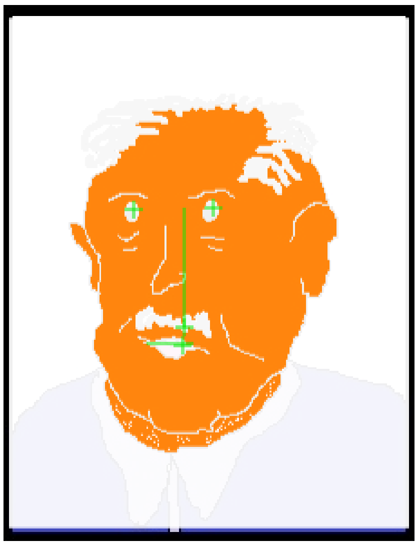
If we ask the database for the name of a person, …



```
+ find + person +
script variables (delta) (result) ◂▸
clear
color reduction
find features
connect to server [snapextensions.uni-goettingen.de] [snapexuser] [snap!user]
set [result ▾] to (get databases)
choose database (1)
set [delta ▾] to [0.1]
set [result ▾] to
  exec SQL-command
    SELECT [ ▾] [Name] ◂▸ FROM [facerecognation] ◂▸ WHERE
      < < < [mouthToNose] > ((item (3 ▾) of (features)) - (delta)) > AND
            < [mouthToNose] < ((item (3 ▾) of (features)) + (delta)) > > AND
        < < < [noseToEyes] > ((item (1 ▾) of (features)) - (delta)) > AND
            < [noseToEyes] < ((item (1 ▾) of (features)) + (delta)) > > AND
          < < [mouthToEyes] > ((item (2 ▾) of (features)) - (delta)) > AND
            < [mouthToEyes] < ((item (2 ▾) of (features)) + (delta)) > > > >
report (item (1 ▾) of (result))
```

… we get an answer – if the person's data are stored.





Exercises:

1. The four images of the example are highly simplified. Do some experiments with real pictures. Try to prepare them with an image manipulation tool so that they are ready to be analyzed by our scripts or reduce the number of colors used.

2. Find some additional parameters to describe faces.

3. The security department of our supermarket has to reject unwanted people (thieves, tramps, …). If the face recognition identifies an “unwanted person” there will be an alarm in the security sitting room and some security members will intervene. Sometimes this process produces loud anger, so the security department decides to reject these persons a bit more sophisticated: the barrier will not open, if these persons are identified in a car, the elevator doesn’t work, doors doesn’t open, … Discuss the consequences of this decision.

4. The advertising department also has fine ideas. There are a lot of people staying in the supermarket but doesn’t buy anything or only few products. Others are buying, but only special offers or cheap products. These are also “unwanted persons” because they occupy space which better could be used by VIP-customers. Discuss the consequences of this decision.

5. “Unwanted persons” have to stay for a while in the supermarket before they can be identified. So the security department together with the advertising department creates “profiles” for these persons, so that they could be identified BEFORE they enter the supermarket the first time. Describe such “profiles” and discuss the consequences of this decision.

6. The advertising department knows from the cash register what customers buy. But many people have a look on products, but don’t buy them. So the walk of customers through the supermarket shall be tracked. This can be done by “car plates” or RFID-chips on the shopping card or with face recognition. Now the advertising department knows in which products customers are interested, they know their unfilled desires. Personalized special offers direct on their smartphone can be produced. Or the customer-data could be sold to specialized shops. Discuss the consequences of this decision.

7. The supermarket team wants to focus on VIP-customers. They identify premium-customers by creating profiles accordingly (type of car, place of residence, personal criteria derived by face recognition, buying behavior in the past, …). To avoid anger the “not-VIP-customers” are allowed to enter the supermarket indeed, but there occur some difficulties (elevator doesn’t work, … see exercise 3). Discuss the consequences of this decision.

8. Face recognition is available everywhere a camera is available, on smartphones, “smart glasses”, laptops, … Because internet also is available everywhere, the images can be compared with reachable databases like social networks, … Reachable by the user of the camera or reachable by anyone who reads the image data! So everywhere the persons on a picture can be identified in real time. Discuss the consequences of this development in different contexts.