Eckart Modrow
emodrow@informatik.uni-goettingen.de

Der SQLsnap-Supermarkt

Ein Projekt zur Demonstration der neuen Möglichkeiten für Snap!

Status: 4. März 2014

Eckart Modrow

SQLsnap-Beispiel „Supermarkt“

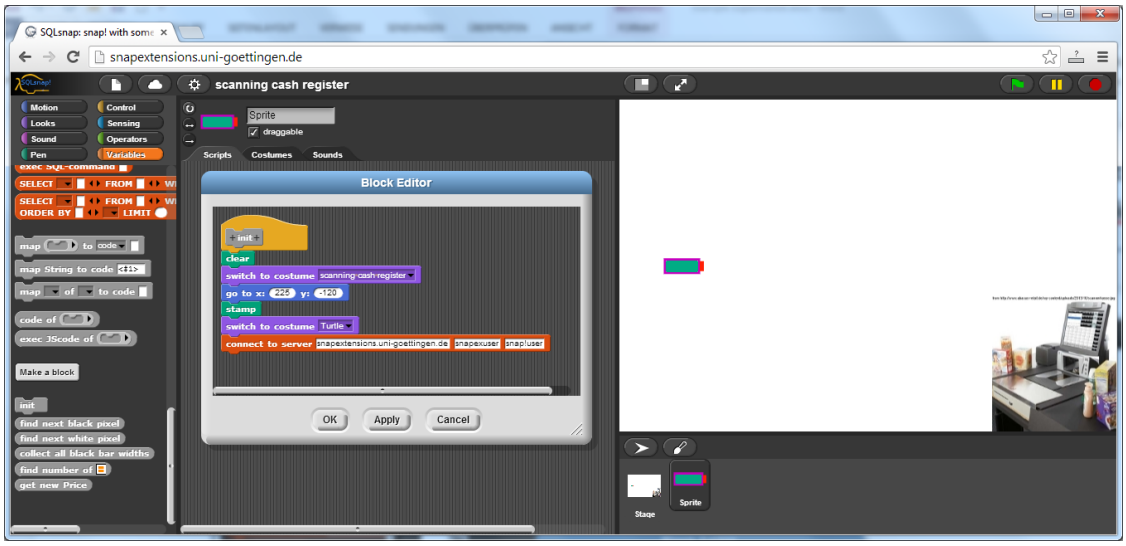
Wir stellen uns einen Supermarkt mit verschiedenen Abteilungen vor:

- einer Scannerkasse (liest Barcodes auf den Produkten, liefert Artikelnummern und Rechnungen)
- eine Lagerverwaltung mit integrierter Datenbank (erhält Artikelnummern, liefert Preise und bestellt, falls nötig, Produkte von den Lieferanten)
- eine “intelligente” Waage für Früchte (erkennt eine Frucht mithilfe einer Kamera, erzeugt Barcodes)
- eine Werbeabteilung (verantwortlich für Payback, Werbung, Sonderangebote, …)
- eine Sicherheitsabteilung (verantwortlich für die Bezahlung der Parkgebühren, Kunden mit Hausverbot, …

Die Implementationen der einzelnen Abteilungen laufen auf verschiedenen Computern und kommunizieren mithilfe von Textdateien auf einem Server. Und wir benutzen keine professionellen Verfahren, sondern nur „naive“ Lösungen, die zu Verbesserungen durch die Lernenden herausfordern.

1. Die Scannerkasse

Der Schönheit halber nehmen wir ein Bild von einer Scannerkasse als neues Kostüm der Turtle, senden diese an die richtige Position und erzeugen einen Abdruck. Dann stellen wir die Verbindung zu einem geeigneten Server her. Die entsprechenden Blöcke fassen wir als Block ***init*** zusammen.

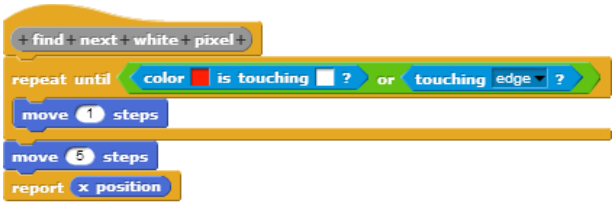


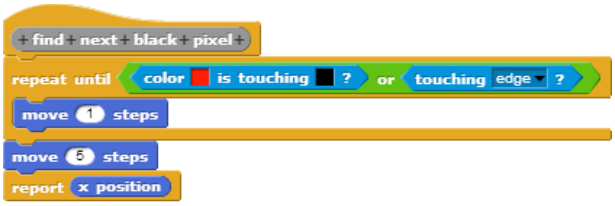
Jetzt brauchen wir Barcodes. Wenn wir keine Ahnung davon haben, erfinden wir welche.

Erster Versuch: Wir zeichnen einige schwarze Balken auf die Bühne und versuchen, deren Breiten zu ermitteln. Dann zeichnen wir ein neues Kostüm für die Turtle, einen „Laserpointer“ mit einem roten Fleck vorne. Wir können nun fragen, ob der rote Fleck schwarze Balken berührt. Zusammen mit der Position des Laserpointers erhalten wir die Breiten der Balken. Die sammeln wir in einer Liste ***widths***.

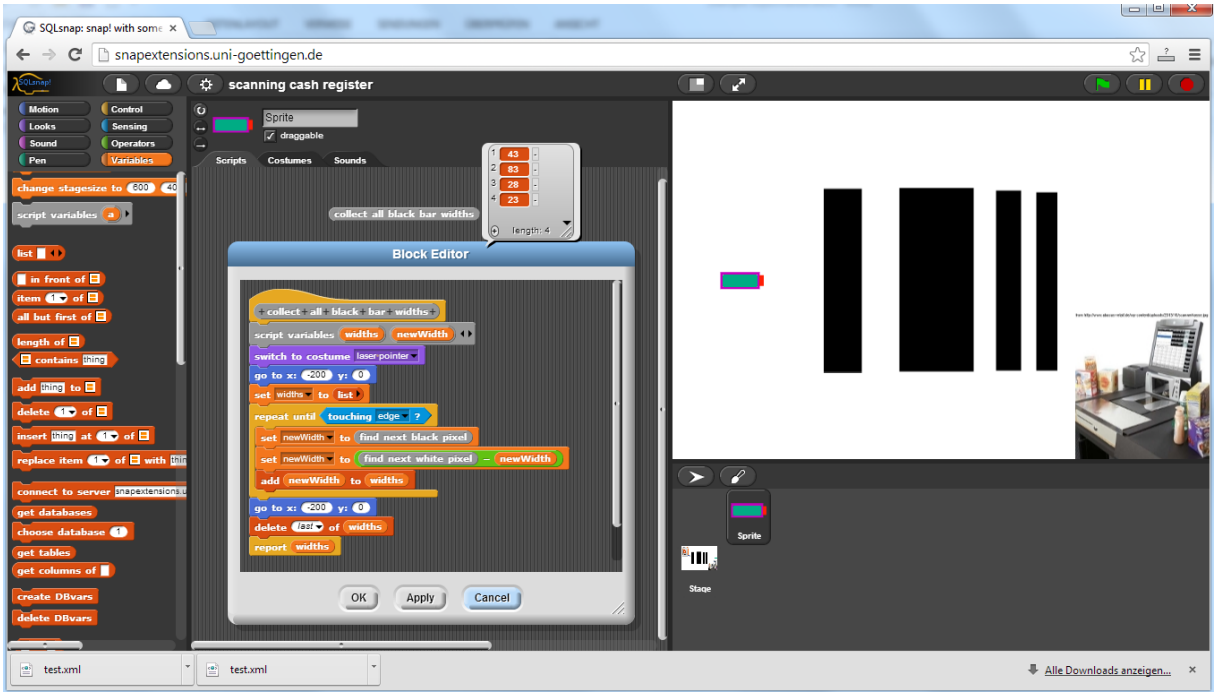
Aber zuerst schreiben wir zwei Methoden ***find next black pixel*** und ***find next white pixel***.

Find next black pixel arbeitet richtig, aber ***find next white pixel*** macht das nicht. Warum? Wenn die Spitze der "roten Nase" des Laserpointers einen schwarzen Balken berührt, hält er an. Aber der Rest der „Nase" berührt immer noch den weißen Hintergrund. Wir müssen also den Laserpointer ein paar Schritte nach rechts bewegen, bevor wir nach weißen Pixeln fragen. Weiterhin soll der Suchprozess am rechten Bildrand stoppen. Wir erhalten:

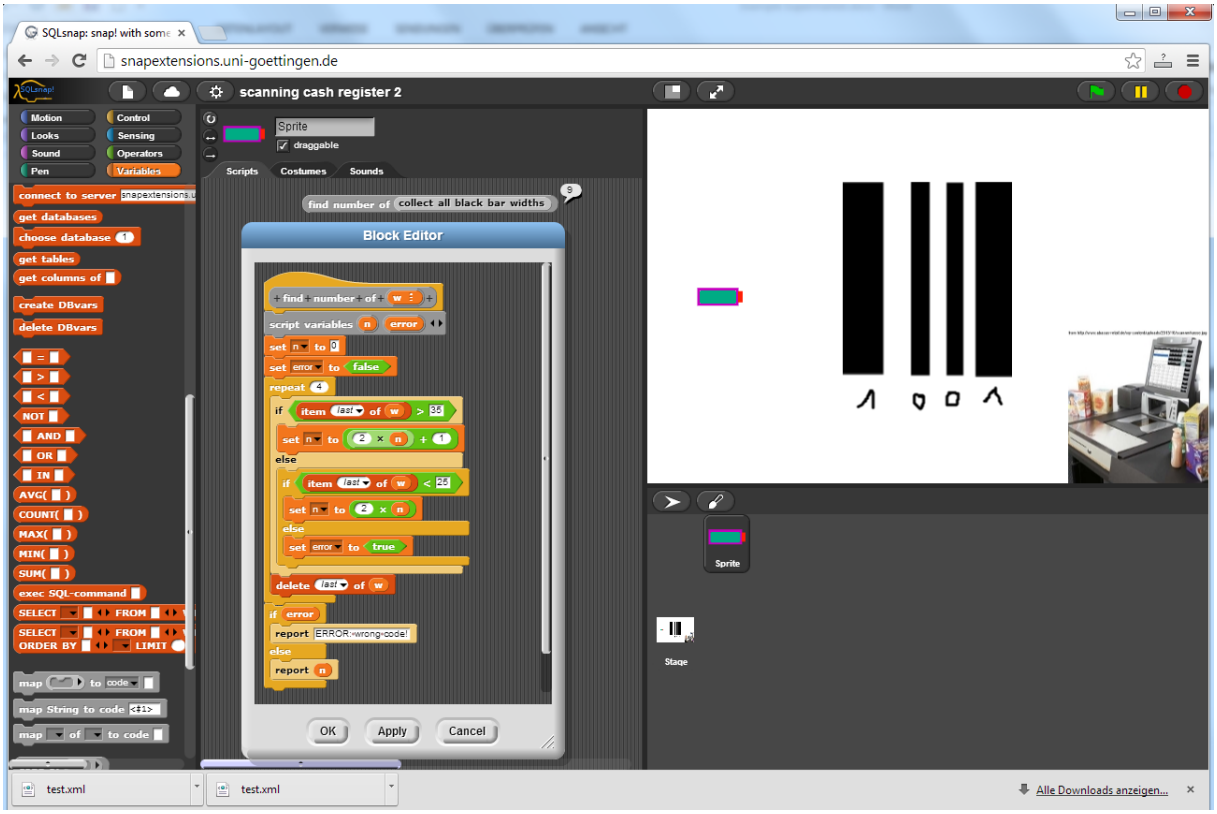




Es ist einfach, die Breiten aller schwarzen Balken zu erhalten: Wir erzeugen zwei Variable (eine Liste ***widths*** und eine Zahl ***newWidth***), messen die x-Positionen des linken und rechten Balkenrandes und packen die Differenz in die Liste. Wegen des Anhaltens am rechten Bildrand ist der letzte Messwert überflüssig und wird gestrichen.



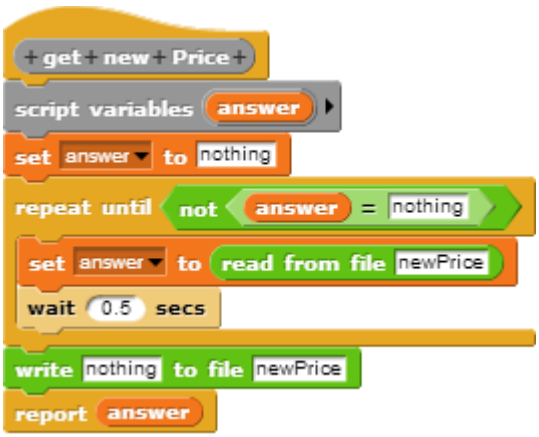
Jetzt brauchen wir einen Code, der anzeigt, was die Balken bedeuten. Wir wählen ein Dualsystem mit vier Stellen, ein breiter Balken (width > 35) bedeutet „1“, ein schmaler (width < 25) bedeutet „0“. Wir zeichnen einige Barcodes dieser Art als Kostüme der Bühne und testen das Skript. Um den numerischen Wert unseres Barcodes zu erhalten, wiederholen wir vier Mal: Lies jeweils das letzte Listenelement und rechne es auf die übliche Art um. Dann lösche dieses Element. Falls es nicht im gewählten Bereich ist, melde einen Fehler.



Wir senden den Wert des Barcodes an die Lagerverwaltung, um den aktuellen Preis zu erhalten. Dies geschieht, indem wir den Wert in eine Textdatei ***newCode*** auf dem Server schreiben.

Der gesamte Prozess kann in einem Block zusammengefasst werden.

Die Lagerverwaltung muss diese Datei immer wieder lesen und bei Bedarf den richtigen Preis und die Bezeichnung des Artikels liefern, in einer Textdatei namens ***newPrice***. Danach setzt sie ***newCode*** wieder auf ***-1***. Die Scannerkasse liest die Datei ***newPrice*** immer wieder und erhält Preis und Bezeichnung, die zum Barcode gehören. Dann schreibt sie ***nothing*** in diese Datei.



Übungen:

1. Echte Barcodes nutzen auch die Lücken zwischen den schwarzen Balken: als weiße Balken. Die Breite der weißen Balken hat die gleiche Bedeutung wie die der schwarzen. Ändern Sie das Skript ***collect all black bar widths*** zu einem ***collect all bar widths***, das die Breiten aller schwarzen und weißen Balken bestimmt. (Der erste und der letzte Balken sind schwarz.) Wenn vier schwarze Balken benutzt werden: welches ist die größte darstellbare Zahl?

2. Zeichnen Sie einige neue Kostüme für ein ***Druckersprite***, das Barcodes auf die Bühne drucken kann. Zuerst soll der Benutzer nach der darzustellenden Zahl gefragt werden.

3. Suchen Sie Informationen über Ihr ***nationales Barcodesystem***. In Europe werden Sie EAN-Codes finden. Verändern Sie das Druckersprite zu einem "nationalen Druckersprite", das diese Codes druckt.

4. Wenn die Lagerverwaltung nicht antwortet, wartet das ***get new price***-Skript bis in alle Ewigkeit. Verändern Sie das Skript zu einer brauchbaren Version.

5. Wenn die Lagerverwaltung richtig arbeitet, erhält die Kasse Antworten der Form

   <Preis>,<Bezeichnung>

   Lassen Sie die Kasse Rechnungen für die Kunden produzieren, die Datum und Zeit sowie alle gekauften Produkte mit Preisen und die Gesamtsumme enthalten. Steuern sollen so wie in Ihrem Land üblich angegeben werden.

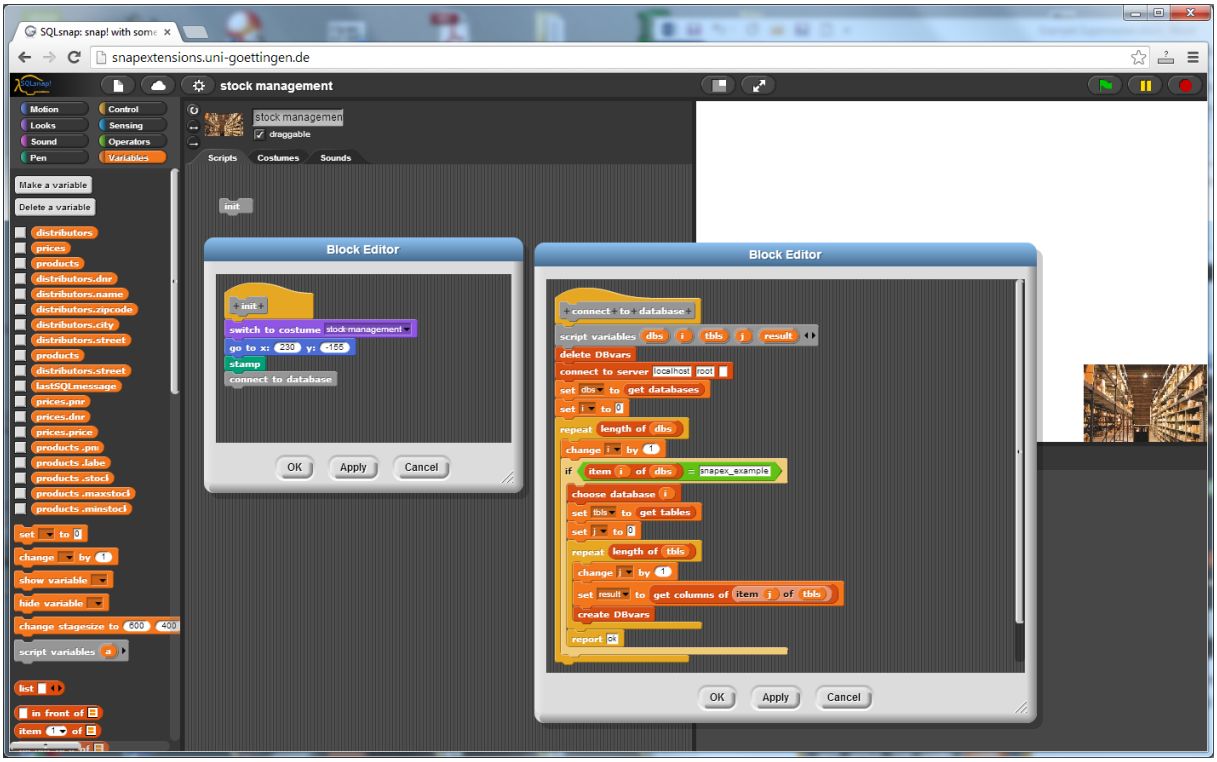
2. Die Lagerverwaltung

Die Lagerverwaltung benutzt eine MySQL-Datenbank auf dem Server. In diesem Fall die ***snapex_example*** Datenbank.

Dort finden wir drei Tabellen:

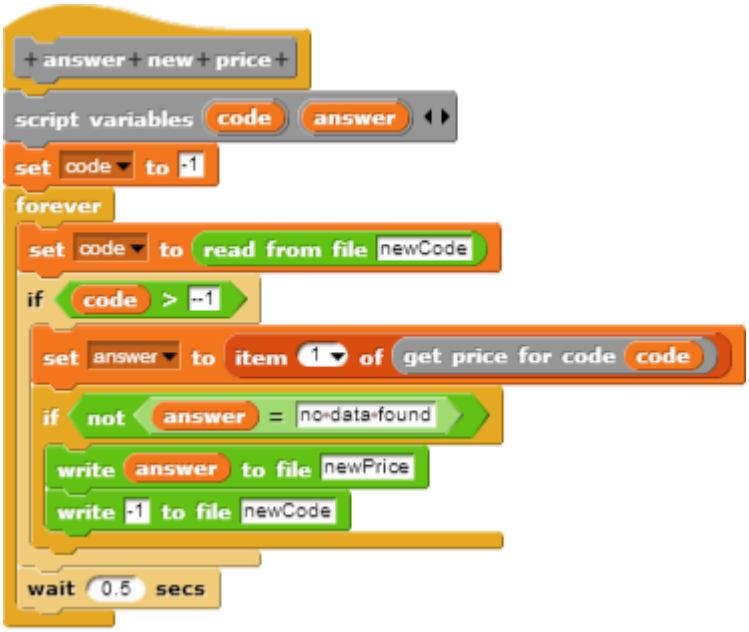
- products(pnr,label,maxstock,minstock,stock)
- distributors(dnr,name,zipcode,city,street)
- prices(pnr,dnr,price)

Wir müssen antworten, wenn die Scannerkasse nach einem Preis fragt. Deshalb müssen wir dauernd nach Änderungen in der Datei ***newCode*** sehen. Aber zuerst drucken wir ein Lagerverwaltungs-Bild auf die Bühne, verbinden uns mit dem Server, wählen die snapex_example Datenbank und erzeugen Variable für alle benutzten Größen: Tabellen und Tabellenspalten.



Eine Anfrage, um den aktuellen Preis und die Produktbezeichnung zu finden, ist einfach. Wir erzeugen den Text der Anfrage, indem wir geeignete Variable im ***SELECT***-Block anordnen. Die Anfrage wird vom ***exec-SQL-command*** Block ausgeführt.

Die unendliche Schleife is tim Block ***answer new price*** gekapselt. Wir sehen dort nach, ob sich die Datei ***newCode*** geändert hat. Falls das geschieht, fragen wir den SQL-Server nach dem Preis und der Bezeichnung und schreiben diese in die Datei ***newPrice*** - falls wir vom Server die richtige Antwort bekommen haben.



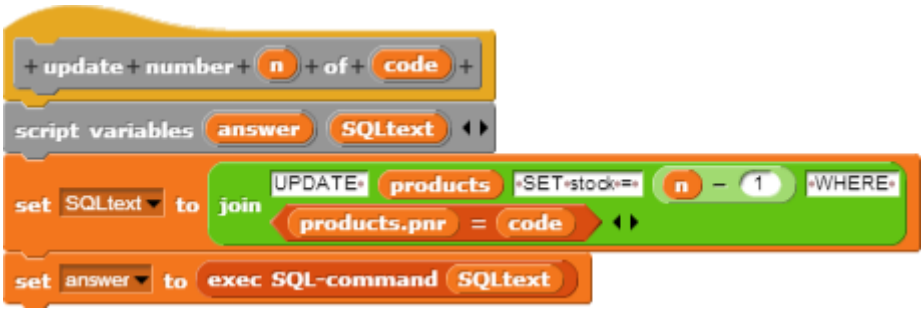
Wenn dieses Skript in der Lagerverwaltungs-Instanz von ***SQLsnap*** läuft, erhält die Scannerkasse die richtige Antwort.

Falls Sie Schreibrechte auf dem Server haben[1], können Sie die Lagernummern auch verwalten. Zuerst müssen Sie die Anzahl der Produkte mit einem bestimmten Code ermitteln.

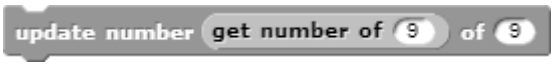
In SQLsnap haben wir

[1] Falls nicht: installieren Sie mySQL auf dem Computer, auf dem die „Lagerverwaltungs-Instanz" von SQLsnap läuft. Verbinden Sie sich mit ***localhost*** als Benutzer ***root***. Sie haben dann Schreibrechte.

derzeit nur Blöcke für ***select***-Kommandos. Deshalb benutzen wir den ***exec sql-command*** Block direkt für ein Update-Kommando.



Wenn der Code ein Produktschlüssel ist, können wir beide Blöcke zu einem Update-Kommando kombinieren:



Übungen:

1. Wenn einige Produkte verkauft worden sind, nimmt der ***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.

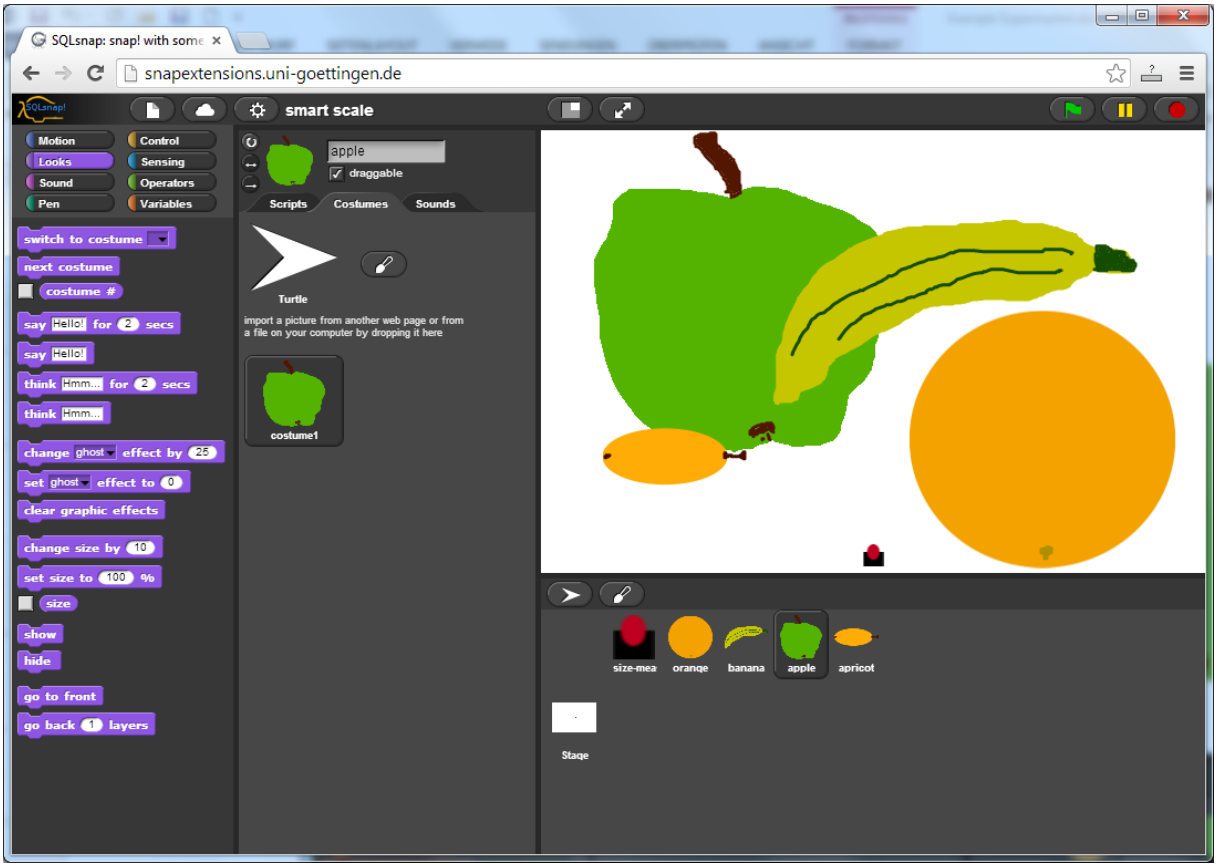
7. Build a block to produce ***update-commands*** for mySQL .
   Syntax: UPDATE <tablename> SET column = value {,column = value} WHERE <condition>;
   Example: UPDATE products SET stock = 99 WHERE pnr = 11;

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:

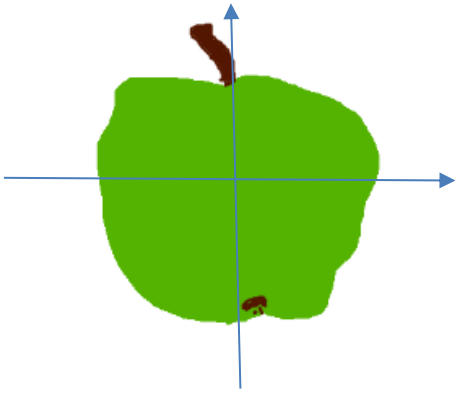
- apple and orange are round, the banana is long
- orange, apricot, and banana are orange-yellow, the apple (here) is green
- apricot is small, the others are bigger

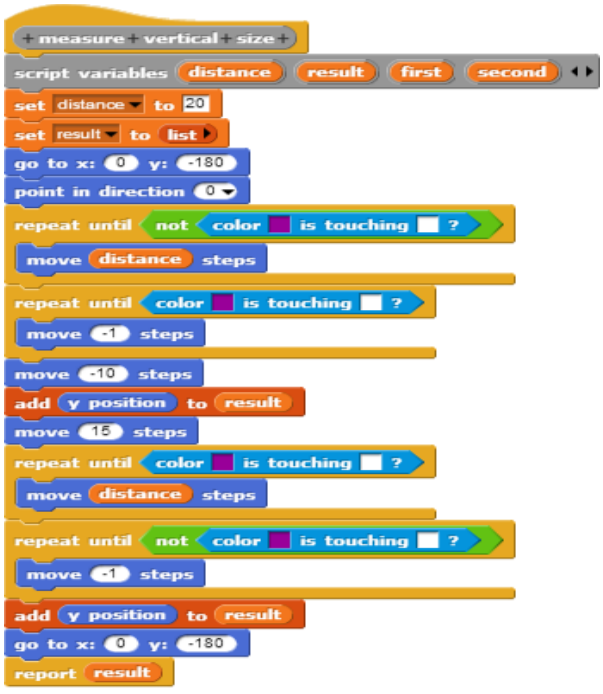
But what means “round”, “long”, “yellow” and “green”, “big”???

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

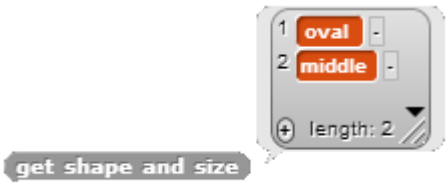
Distinguish *round, oval* **and** *long*

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.



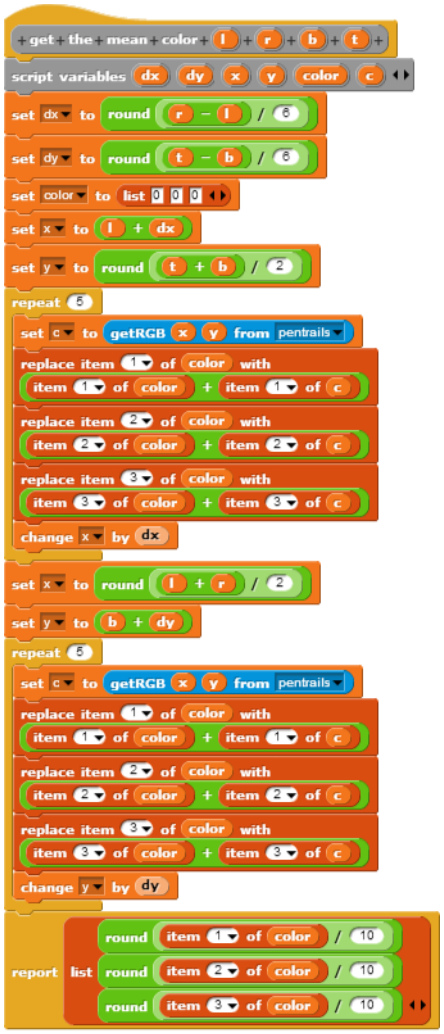
Find out the color of real objects

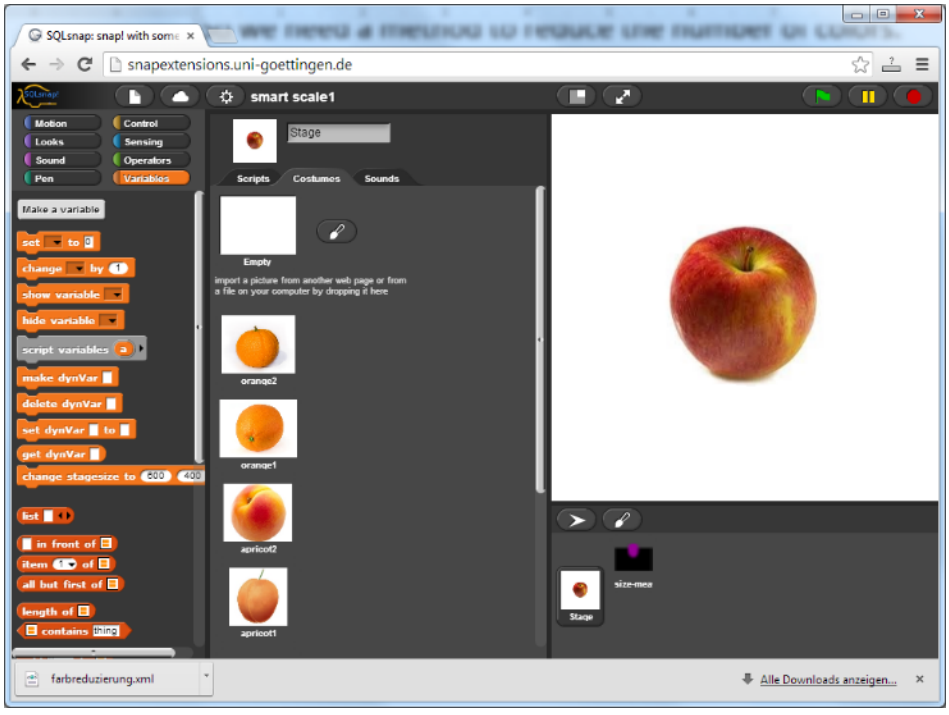
Normal fruits have different colors. So we have to measure the mean color. We do this by measuring ten RGB-values: five on the horizontal slice and five on the vertical. (That was the reason to store the position of the object.) That's easy. But our RGB-value list may describe 256 * 256 * 256 colors.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

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

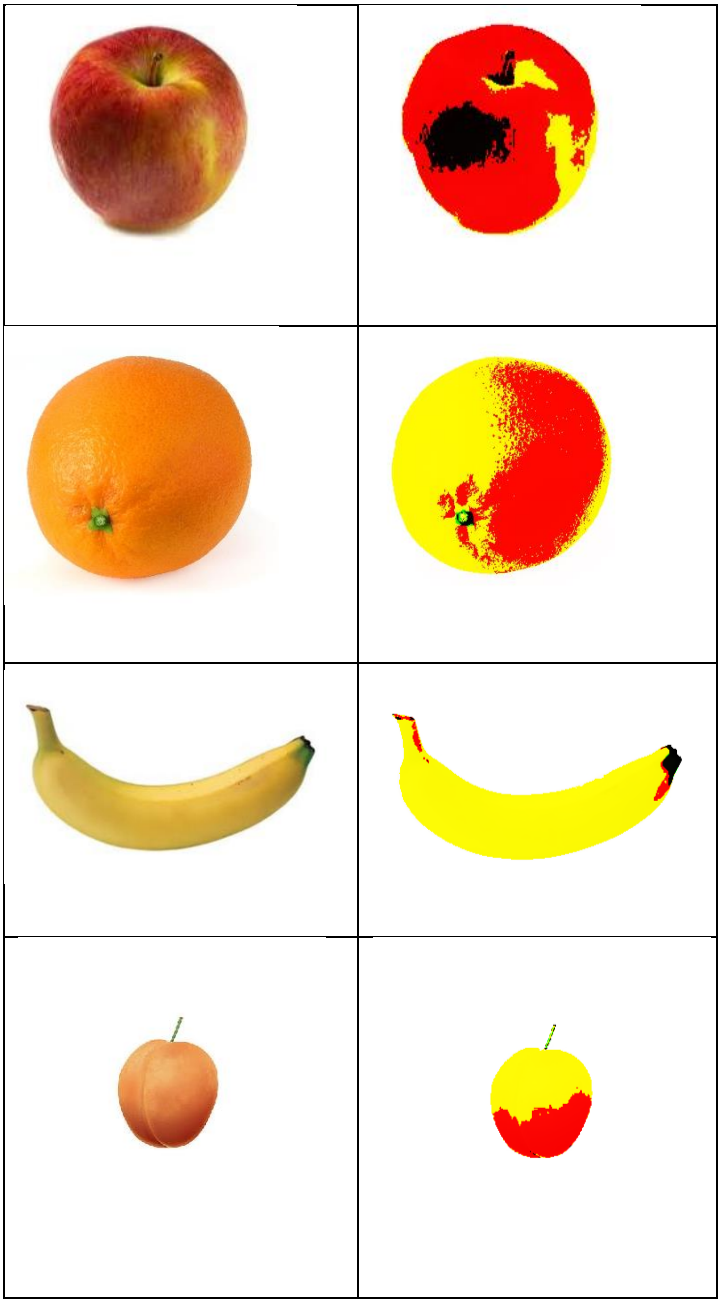
We try it this way: For each RGB-value we decide, whether we have a "high" or a "low" value of this color. If "high" we set the value to 255, if "low we set it to 0. We get two possible values for each color, and a total of 2 * 2 * 2 = 8 possible colors. First we try whether we have enough colors to 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.





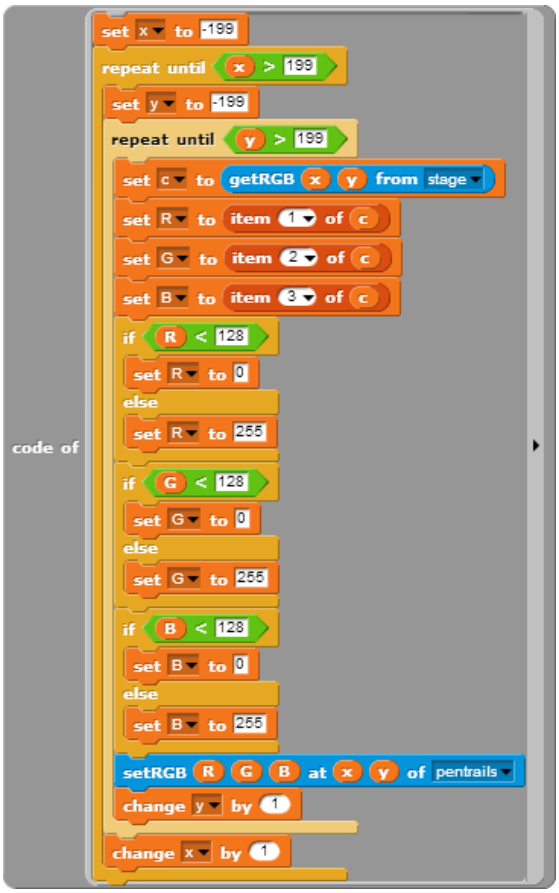
Now we reduce the colors as described …

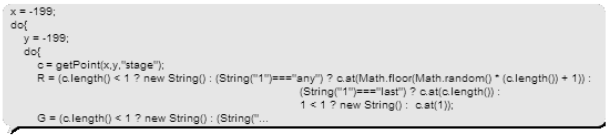


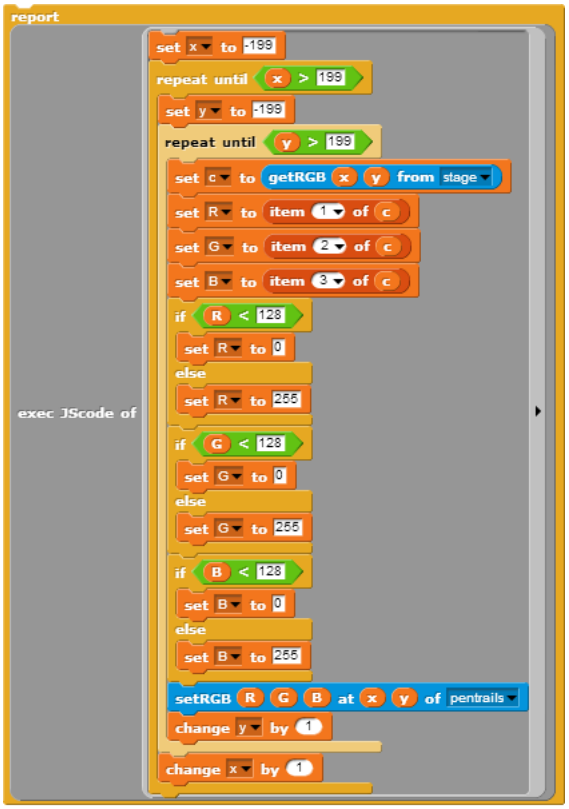
.. and find, that’s ok.

How does it work?

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:







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.

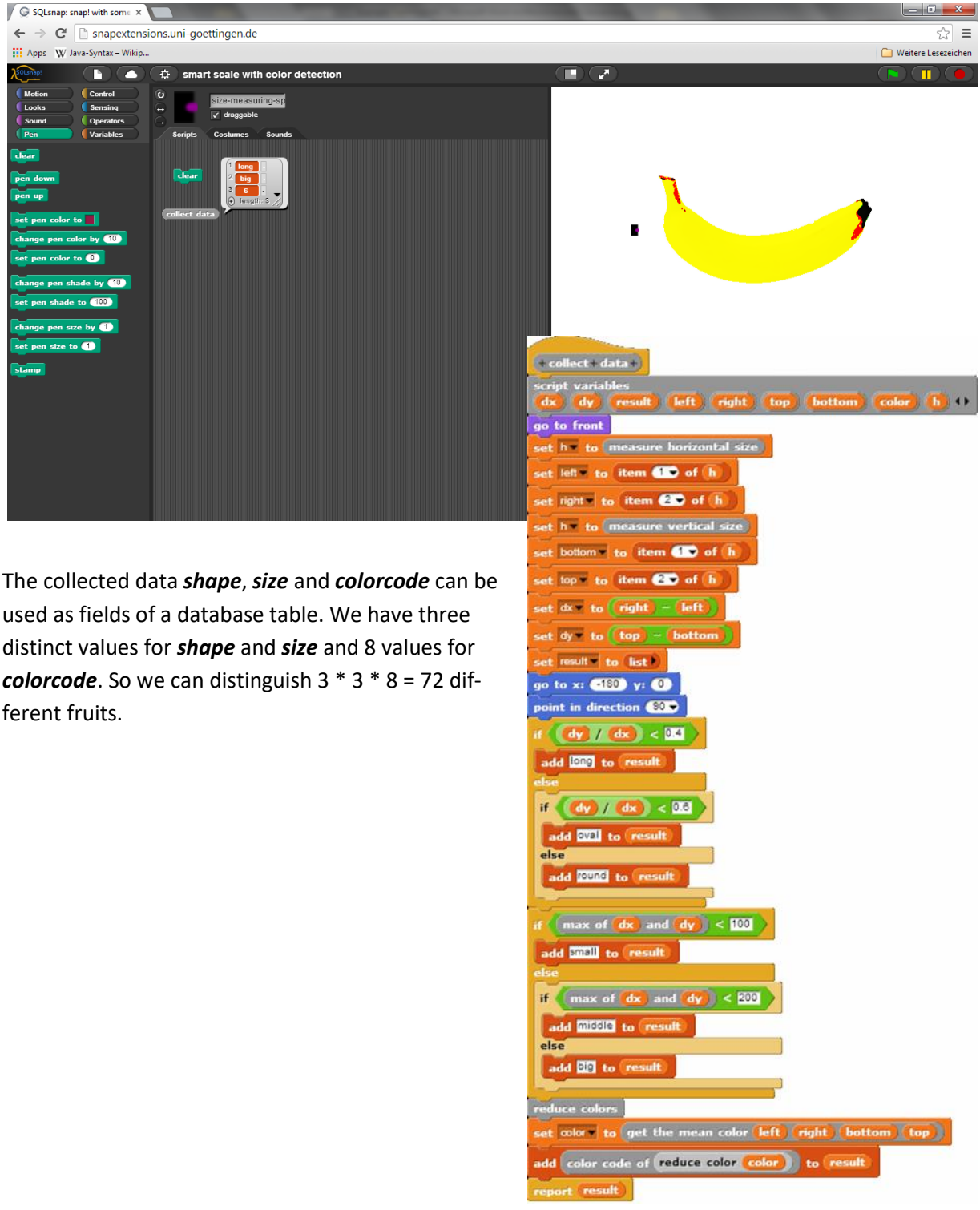
The ***reduce color*** block works as described: it gets a list with RGB-values and returns a list with a reduced color.

Color codes:

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).

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.



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.

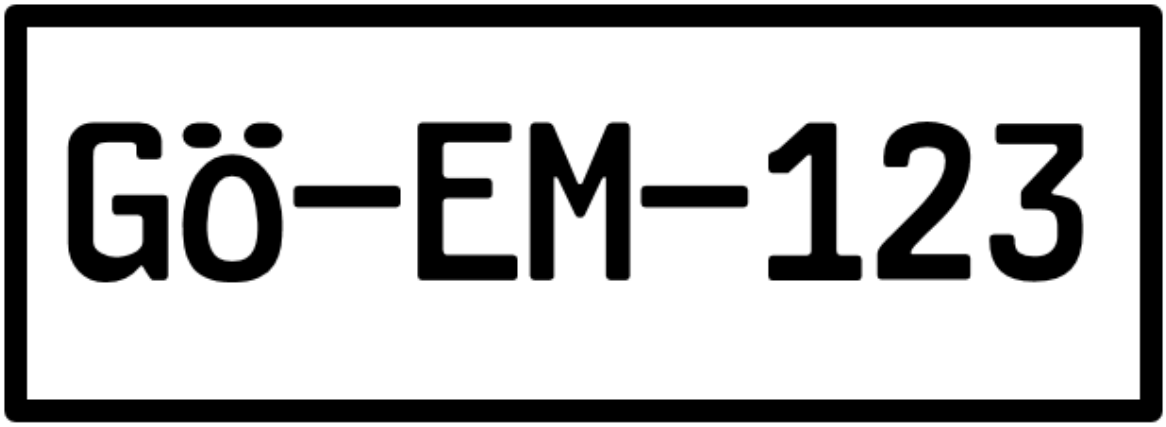
5. Our fruit recognition process only works well if we center the object in the middle and align it horizontal. If we take a sprite with an appropriate costume we can center and align it automatically before we leave a stamp. Do this.

6. If we use a color more detailed 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?

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.



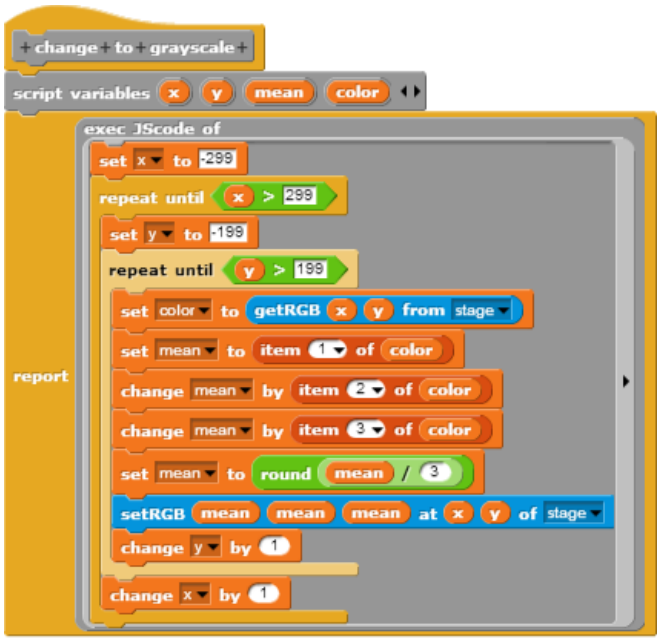
Car number plates have special fonts which are fine for pattern recognition. In Europe they have a black border – good for uns. We try to identify the numbers on the plate. (To recognize the other chars is your term.) Fortunately we have developed almost all tools to do this. We only have to ask the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

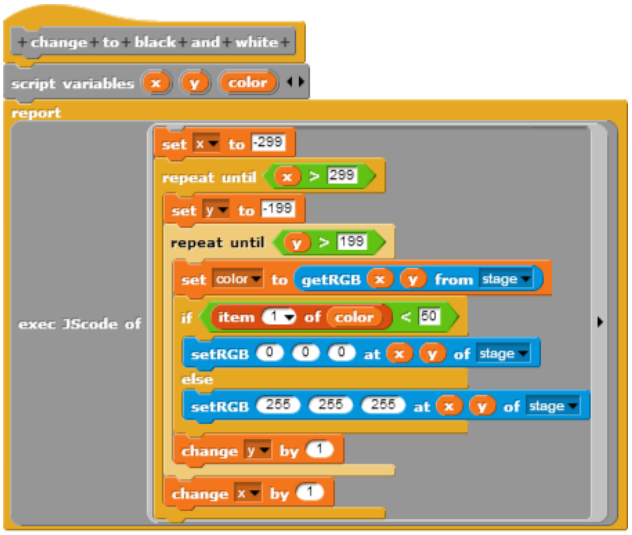
Dirty car plates

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?

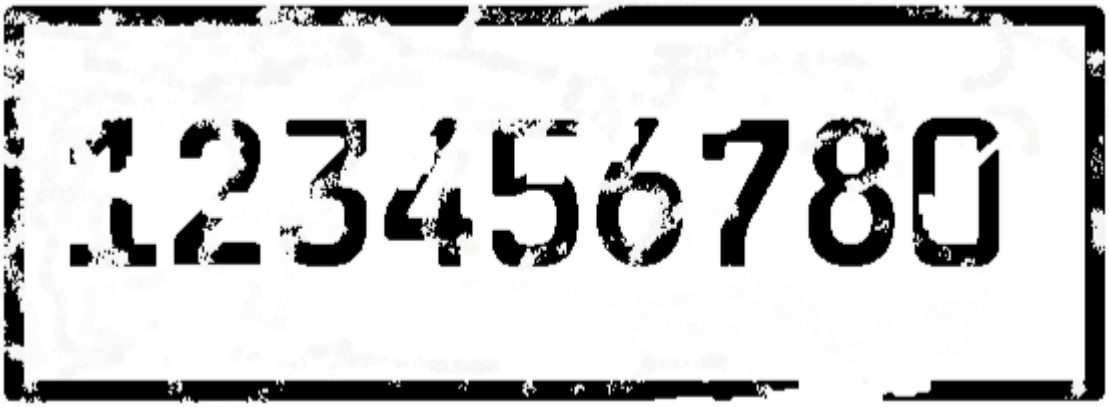
First we need a grayscale picture: we calculate the mean value of the red-, green- and blue-value of each pixel and take this as value for all three colors. The pixel will be gray. We store the result in the stage image. To speed up we put the script in the ***exec JScode of*** block and have the transformed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



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.

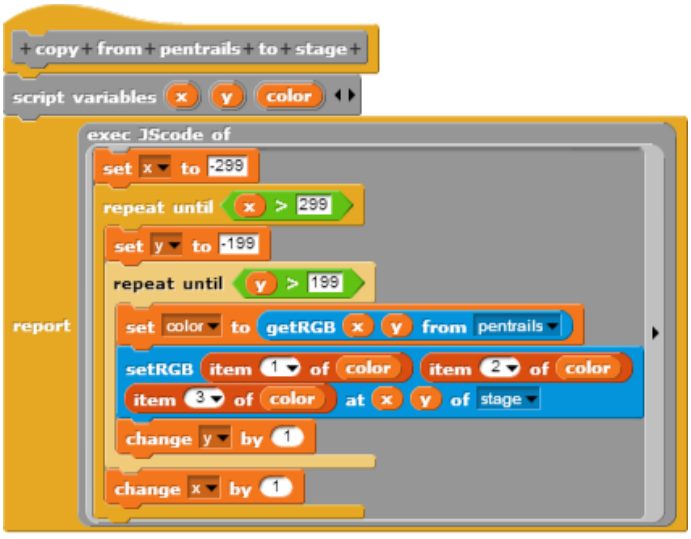


The result is:

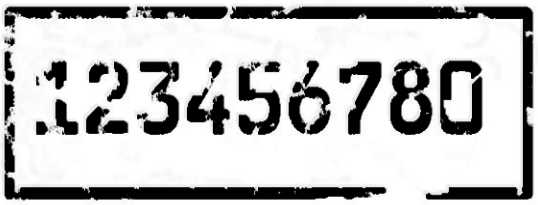


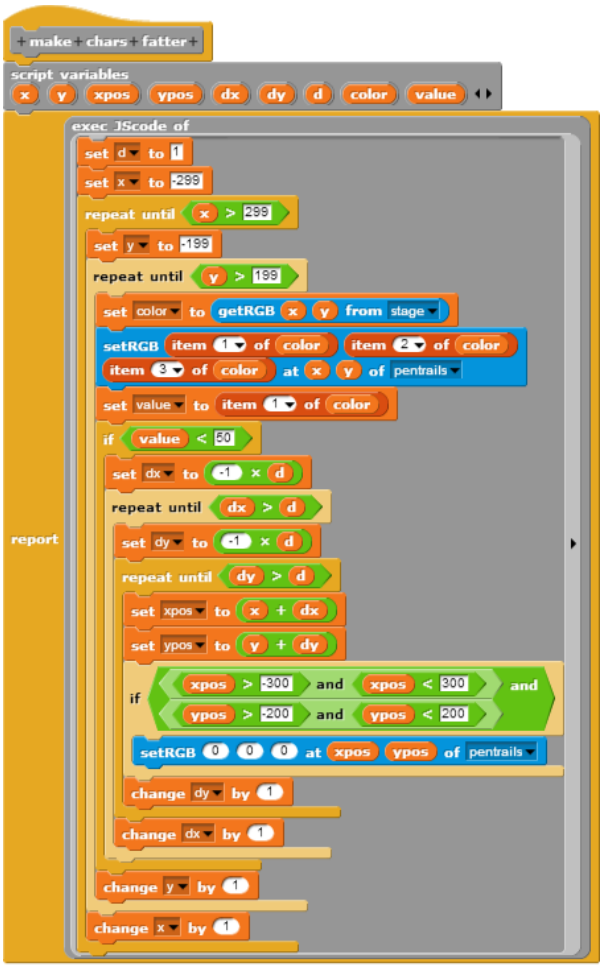
Closing the gaps

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.



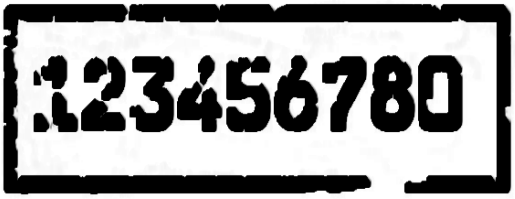
Result after one call of make chars fatter:



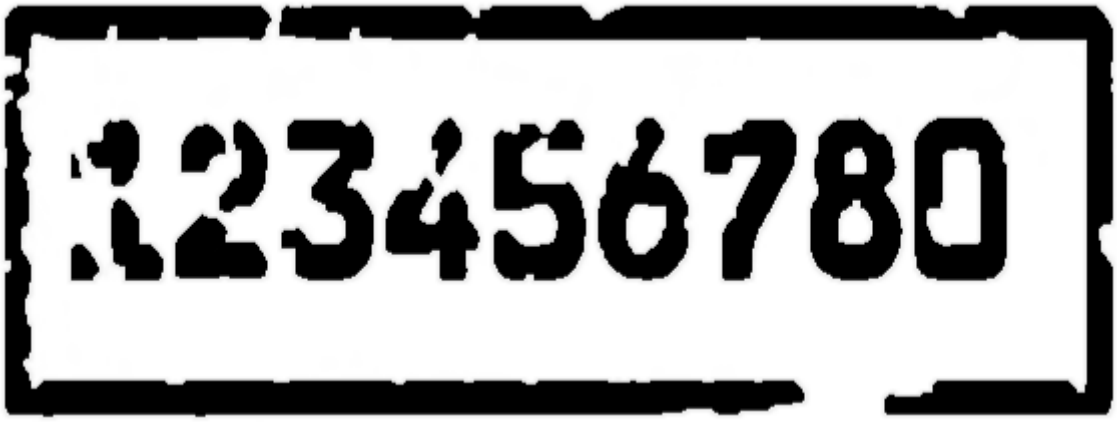


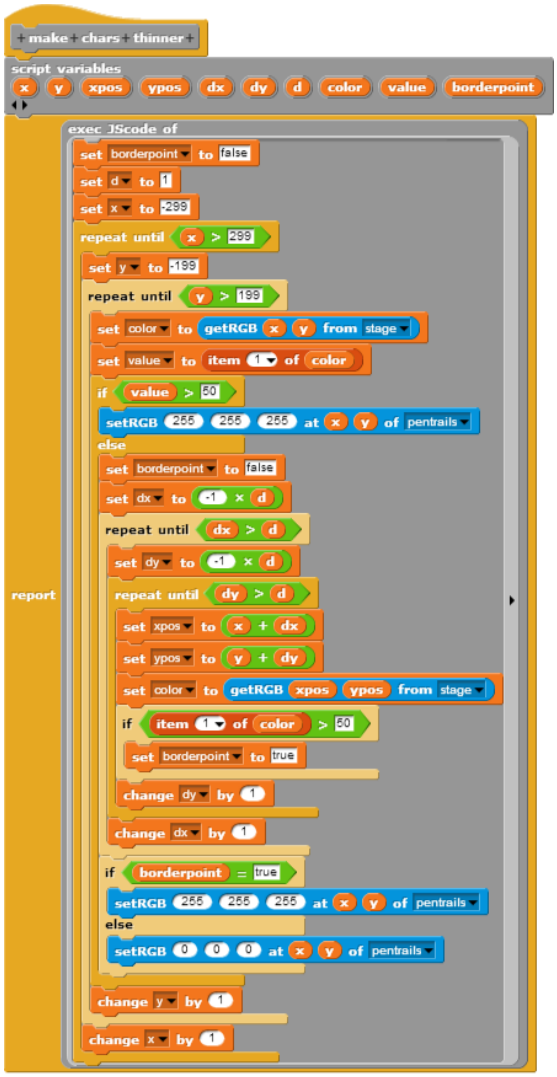
After some more calls of the sequence

we get:

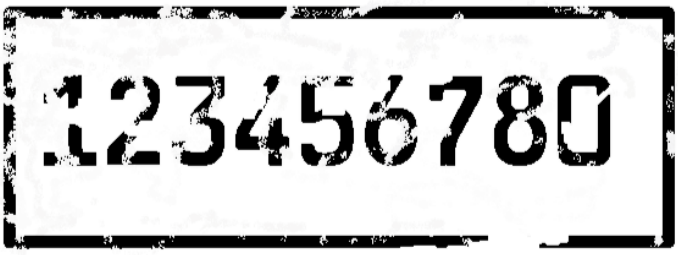


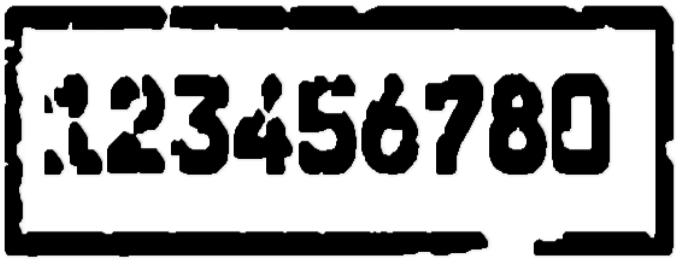
We need a method ***make chars thinner*** to scrape a layer of the black border – with the same problems as above – and the same solution (next page). The result of one call is:





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.

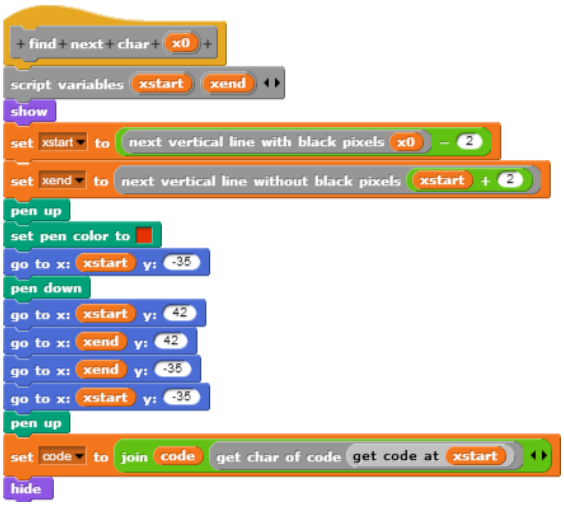


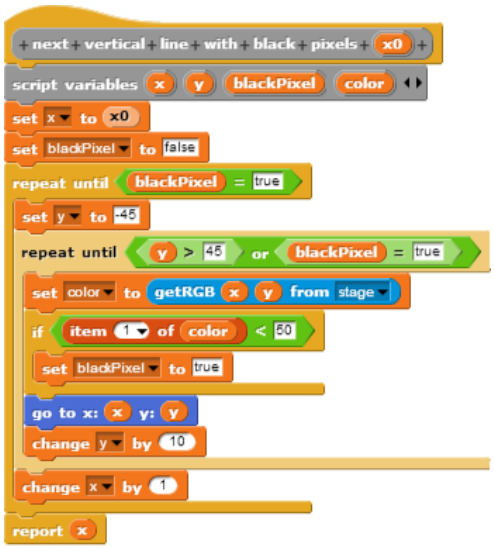


Character recognition

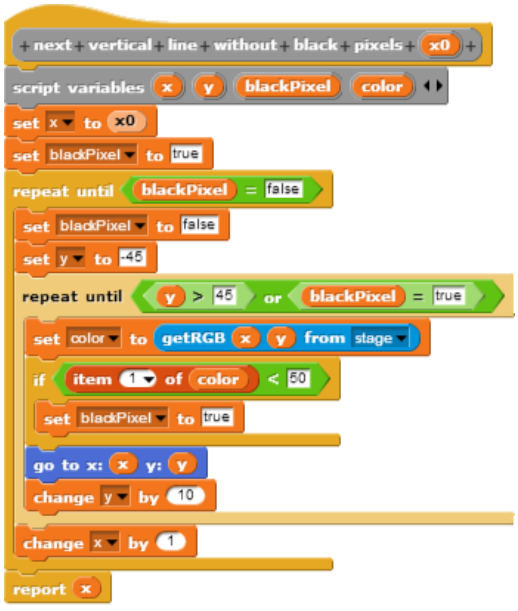
OCR (Optical Character Recognition) uses complex operations, often with neural networks, to recognize chars. We invent a simpler method, known from the smart scale. Because all characters have the same width, we can find them easily if we've found the border of the car plate. Learn from the smart scale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

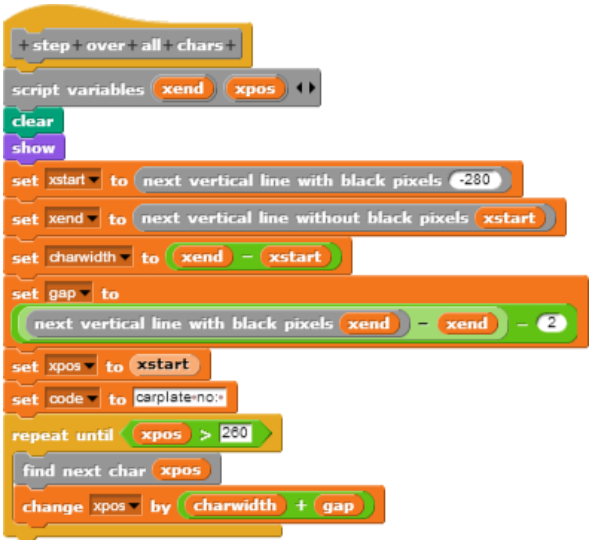




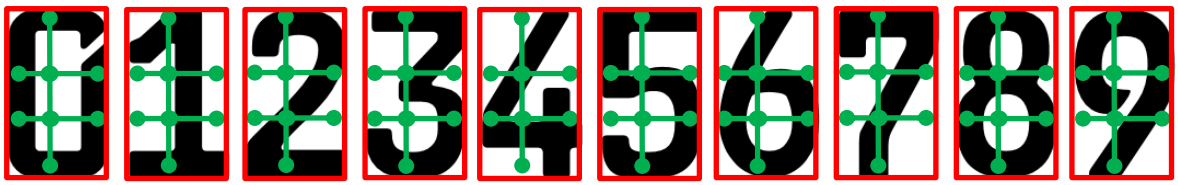
How to find the char code is described later.



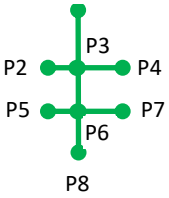
We can step over all characters with the red rectangle by calculating width of characters and gap between them first. The next script shows how to do this.



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).

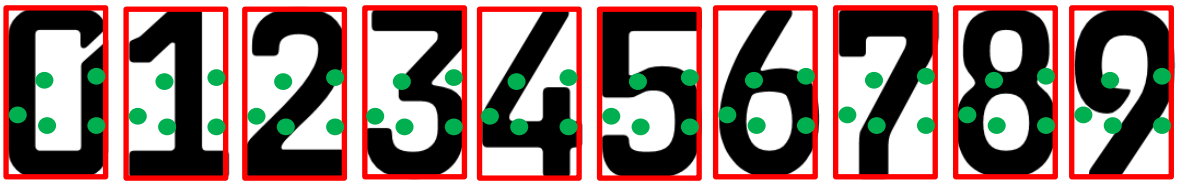


char	P1	P2	P3	P4	P5	P6	P7	P8	Code(s)
0	■	■		■	■		■	■	00100100
1	■							■	01111110
2	■			■		■		■	01101010
3	■		▒				■	■	01011100 01111100
4			■		■	■	■		11010001
5	■	■	■	■			■	■	00001100
6	■		■	■	■		■	■	0100100
7	■					■		■	01111010
8	■	▒	■		■		■	■	00010100 01010100
9	■	■		■			▒	■	00101100 00101110

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:



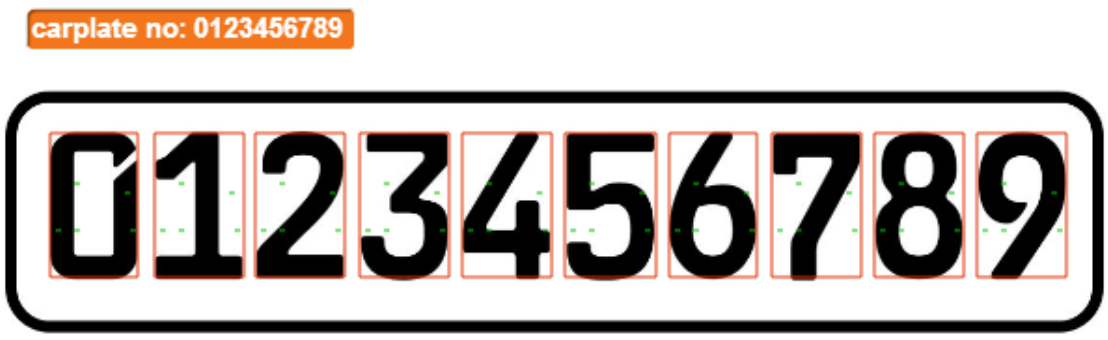
Let’s go!

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

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:

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 sensor-positions 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.

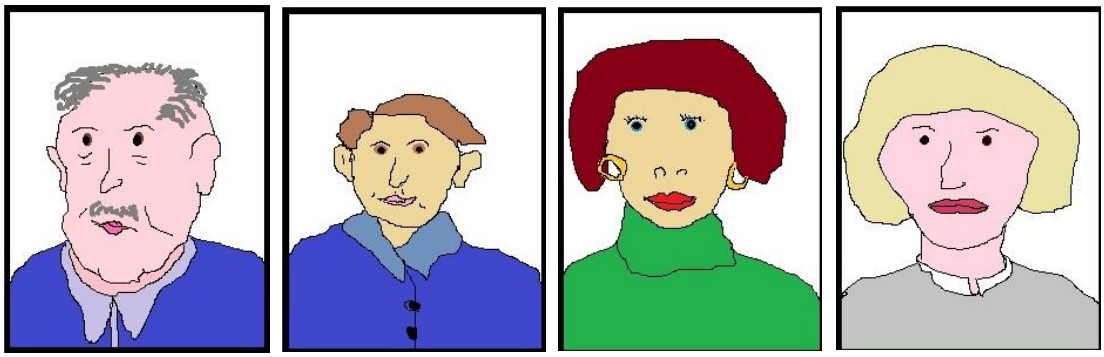
11. In German highways the toll for trucks is determined by toll-collect-barriers which read the car-plates crossing. They read ALL car-plates and delete the identification numbers of passenger cars. Why is this appropriate? Discuss the consequences of storing all numbers and positions.

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.

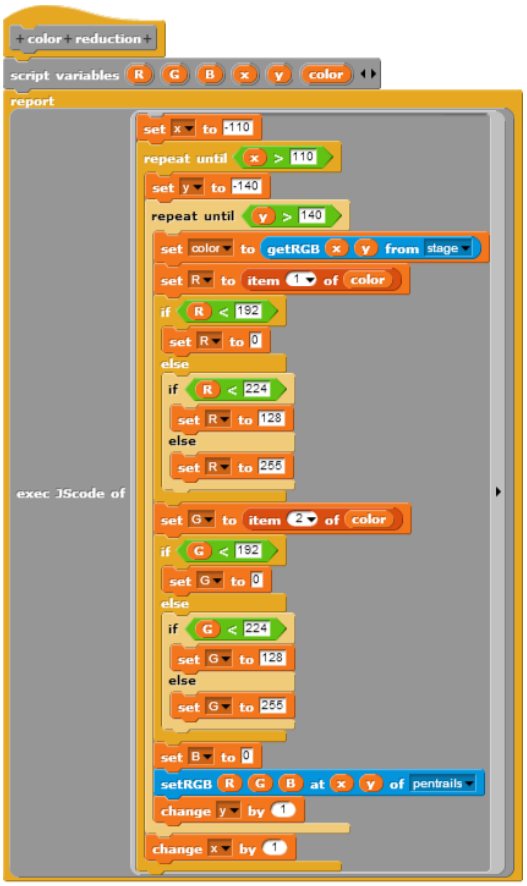
Face recognition

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.

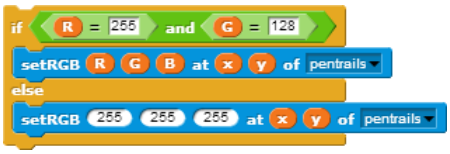


Paul Peter Mary Hannah

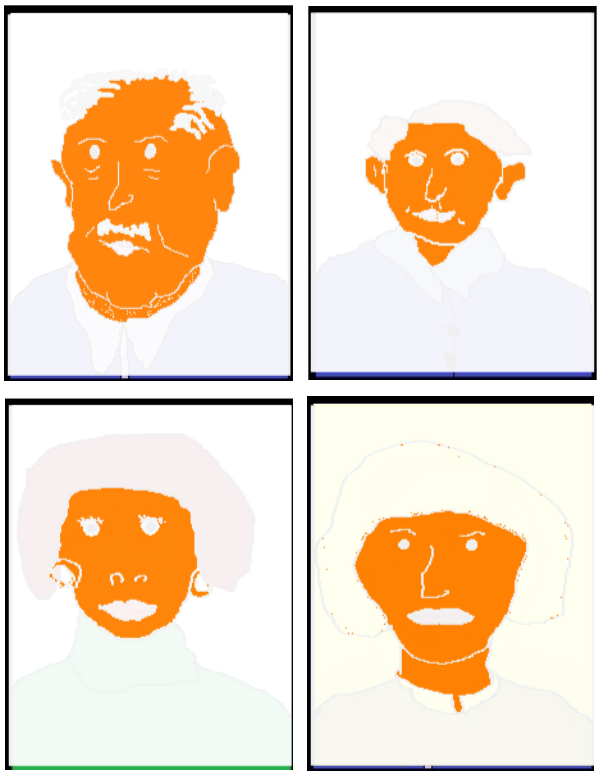
Let's look for faces. We do a color reduction, but a bit more accurate as with the fruits. We ignore the blue-chanel of the pixels and reduce the red- and green-values to three numbers: 0, 128 and 255. The result of this simple transformation is quite good: the face-color is now always the same.



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

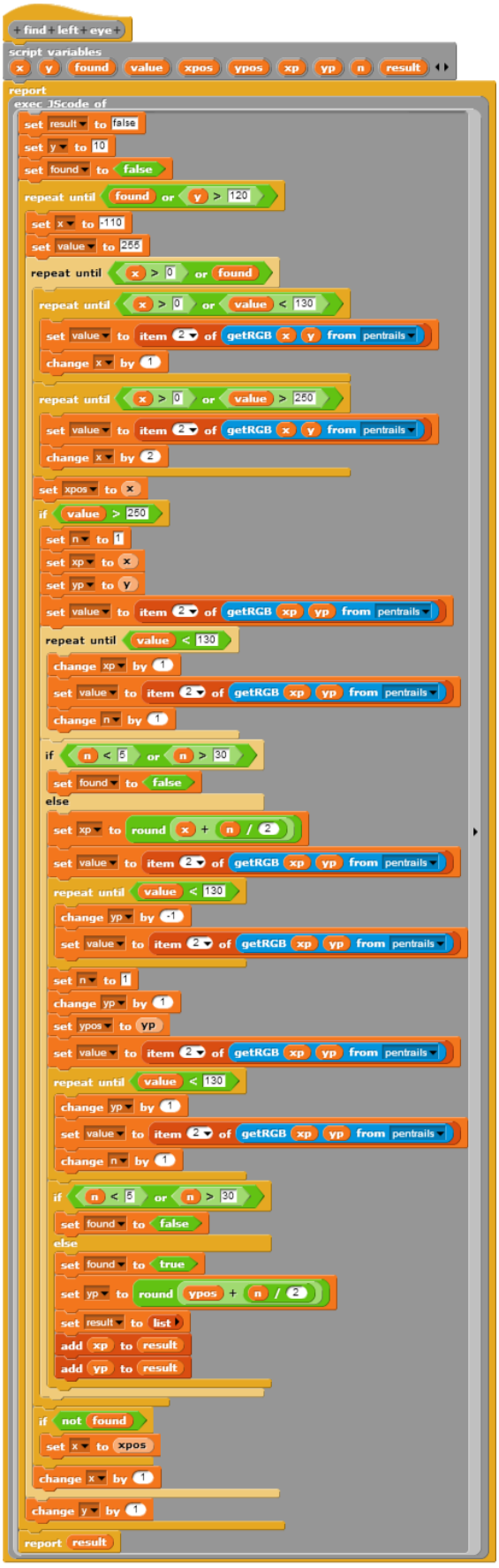


… and get …

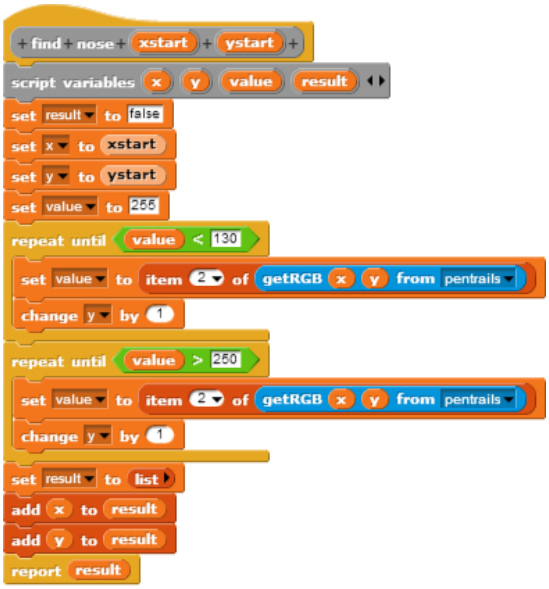


We've got the faces, but we need parameters to describe them! We use size ratios similar to the fruit. So we measure the distance between the eyes and the distance to the mouse, the length of the nose compared with the width of the face, …

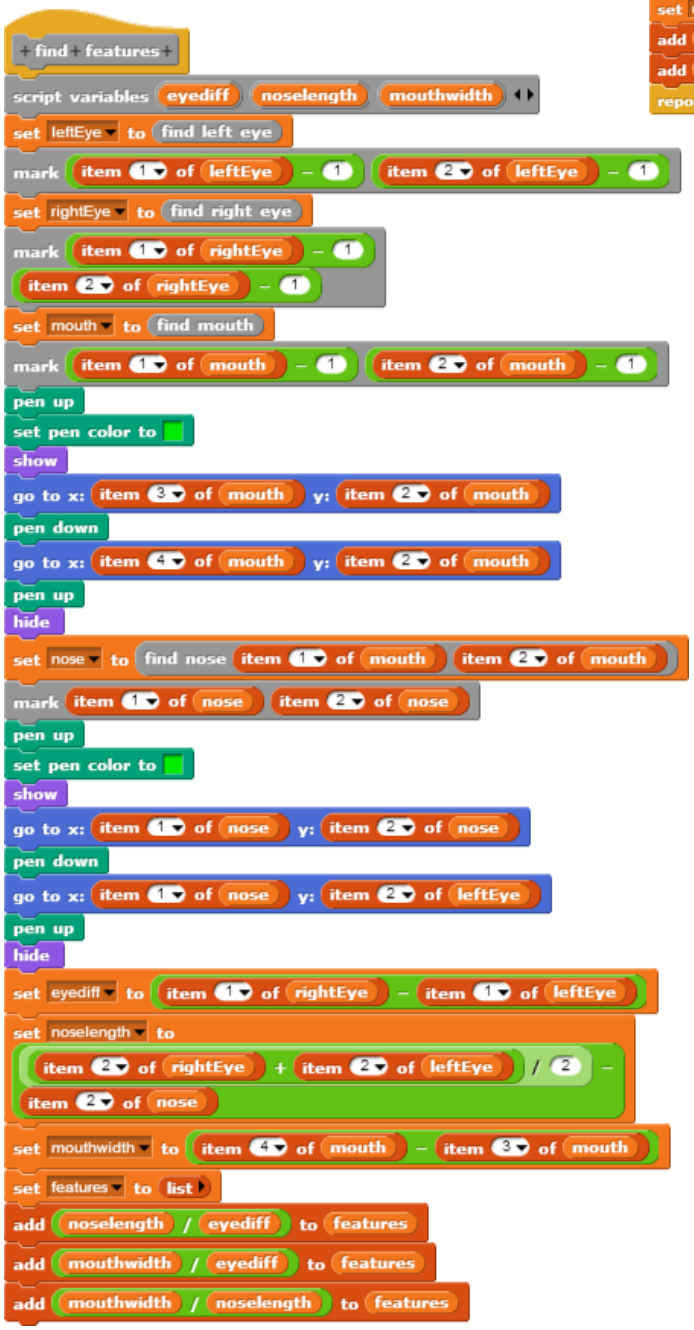
"Eyes" and "mouth" are holes in the orange face. At first we are looking for the eyes. The left one should be above the middle and left of them on a passport photo. We should find there white pixels with some layers of white pixels around them. (Remember ***make chars fatter***!)



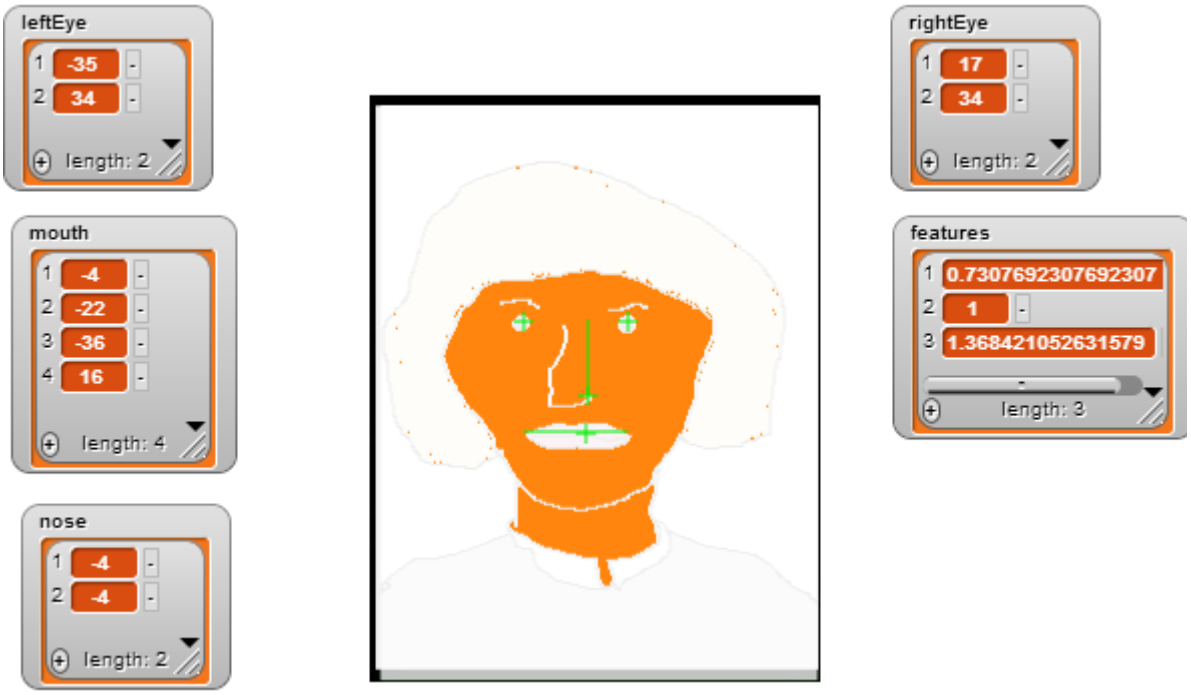
The right eye could be found on the same way as the left one. Only the start-coordinates are different. The same is for the mouth, but the mout should be bigger th an an eye. An the nose is marked by the first white pixel above the mouth.

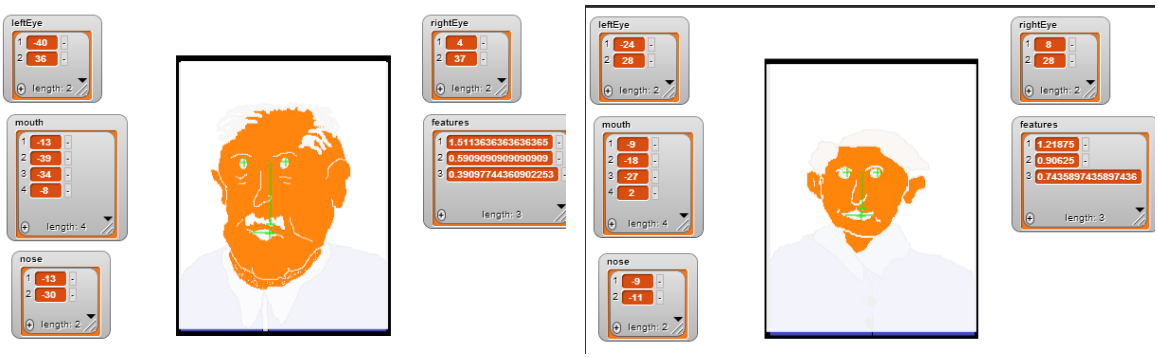


So we can measue the positions of the eyes, nose and mouth and can calculate their distances and the ratios between these values. In ***find features*** these results are marked green.

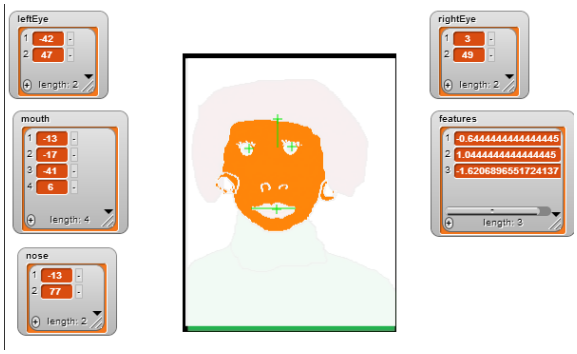


The results are:





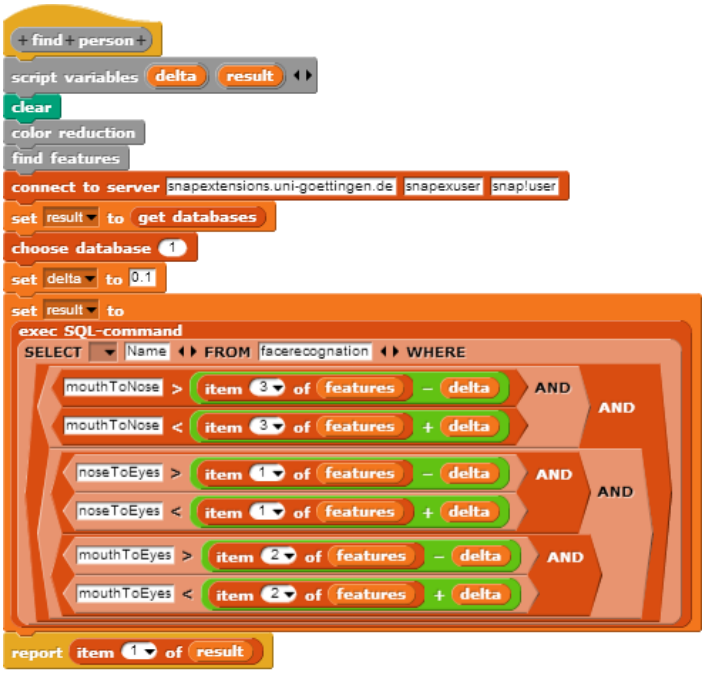
If there is no nose on the picture, the result is consequently wrong.



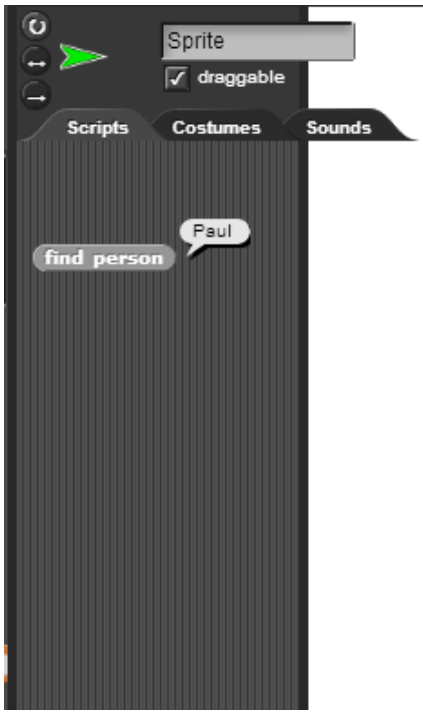
Now we can put the results in a database-table.

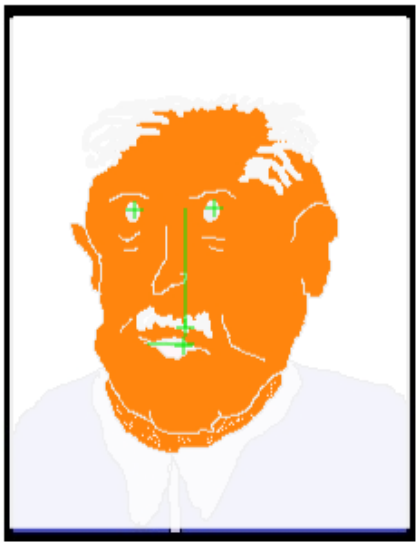
Name	noseToEyes	mouthToEyes	mouthToNose
Peter	1.22	0.91	0.74
Paul	1.59	0.59	0.39
Mary	0	1.04	0
Hannah	0.73	1.00	1.37

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



… 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.