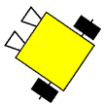


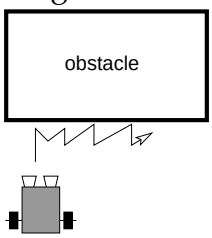
continuing with RoboLab

Name:

---

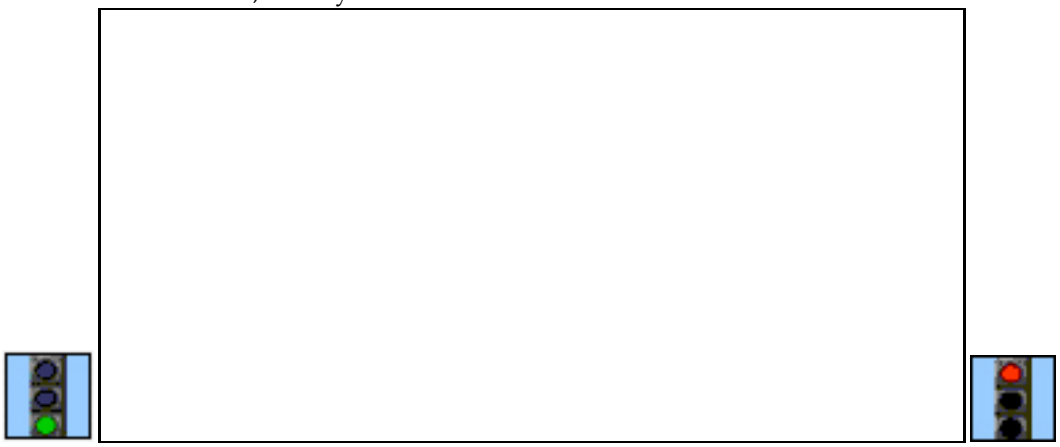
instructions

1. **refresher programming challenge**
   Program the robot to do "wall-following" in order to navigate around an obstacle:



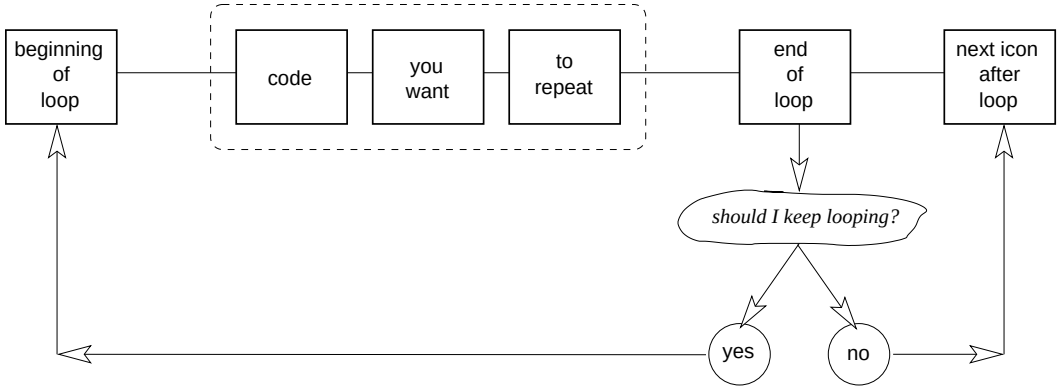
   (a) Design your code before you start programming. Write your *algorithm* below:

   (b) Now in the box below, draw your **code**:



2. loops

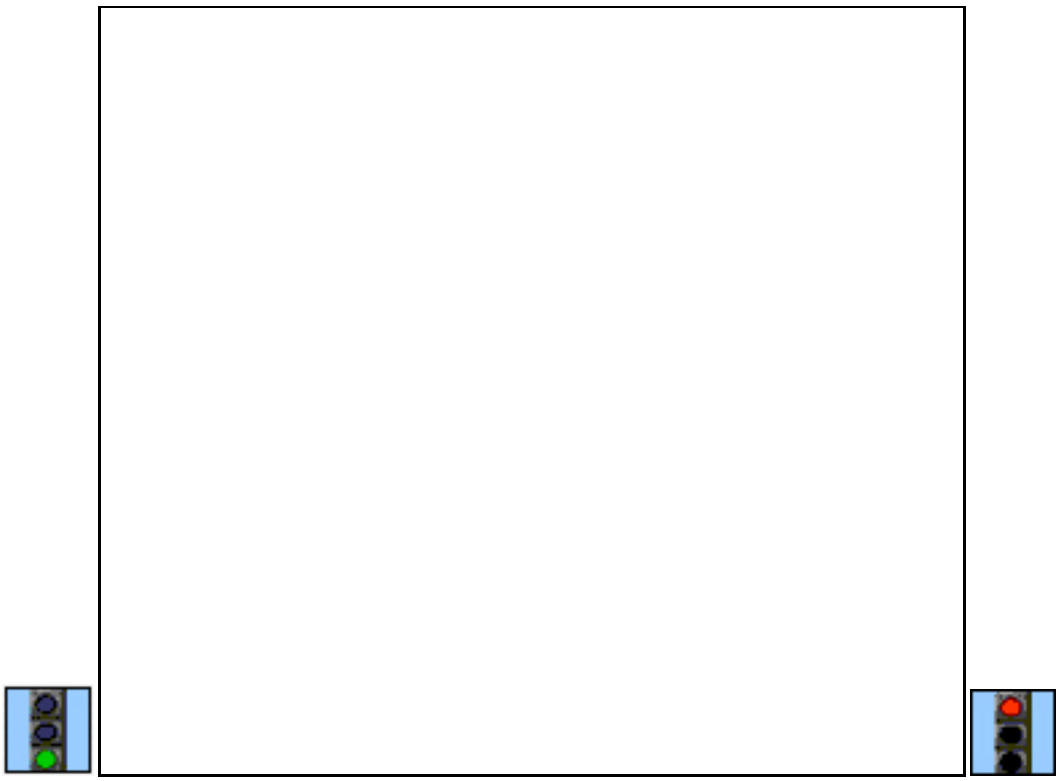
- Loops are structures that allow your program to repeat specific tasks again and again.
- All loops have the same basic structure. This includes a set of commands that you want the program to repeat and something that tells the program when to stop repeating, or "looping".



- Some loops run for a specific number of times. These are called **counter controlled loops**.
- Another type of loop runs forever (i.e., they do not stop unless you force the program to quit in the middle or turn off the robot). These are called **forever loops** or *infinite loops*.
- Other types of loops run until a certain *condition* occurs, for example, go forward until you hit a wall. Those are called **condition controlled loops**. We will talk about these later.

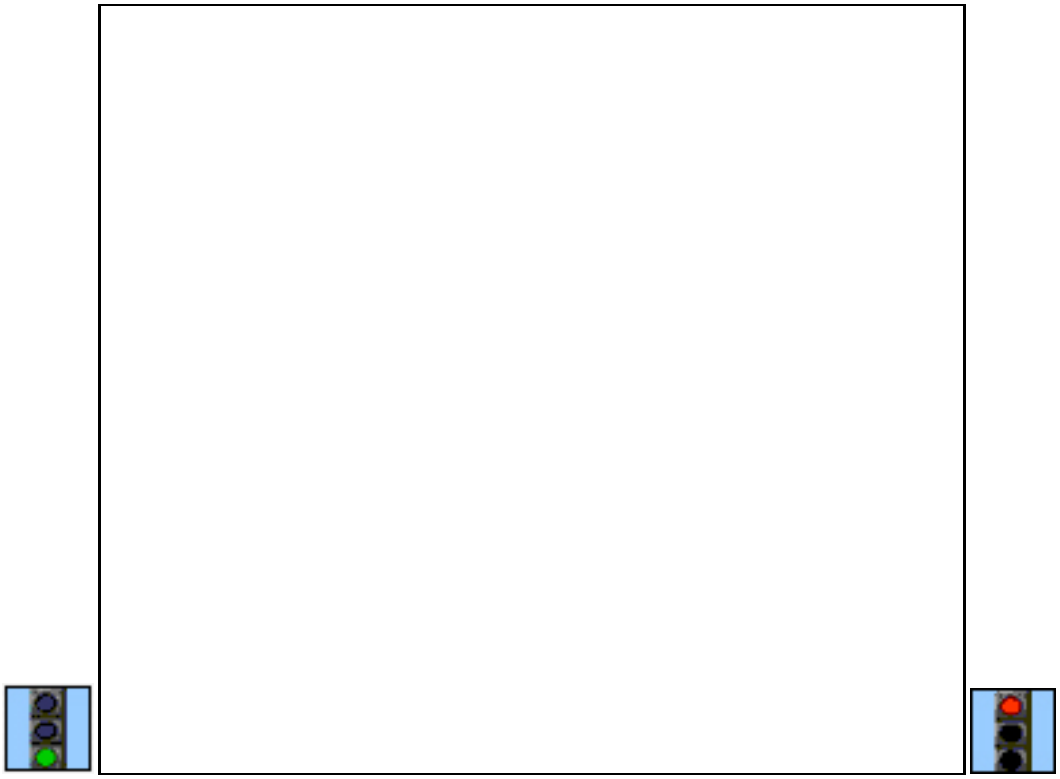
3. **"forever loops" using "jumps"**

- In RoboLab, forever loops are implemented using "jumps".
- Jumps begin with a **land** icon  and end with a **jump** icon  .
- In between the **land** and the **jump** is the section of your program that you want to repeat forever.
- **programming challenge**
  Modify the program you wrote earlier in which your robot moves in a square: change the program so that it uses a jump and, instead of copy/pasting the same bits of code four times, repeats the "go straight and turn" forever.
  Draw the code in the box below.



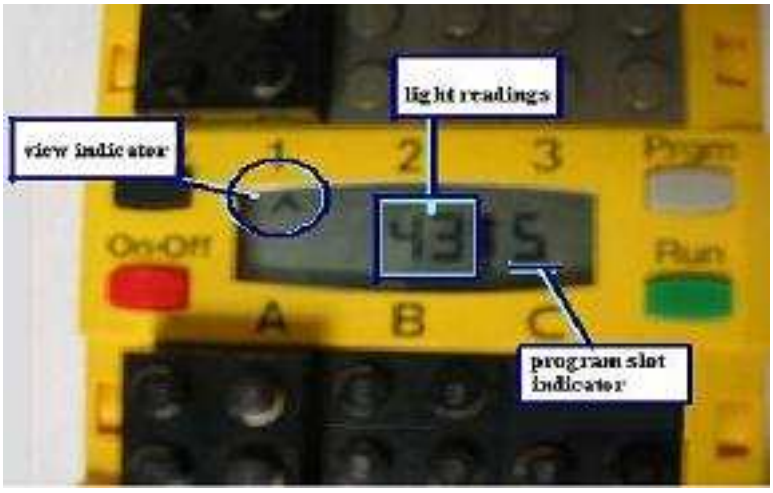
4. counter controlled loops

- In RoboLab, counter controlled loops begin with a **start of loop** icon  and end with an **end of loop** icon  .
- There is also a **loop counter** which tells the loop how many times to repeat. This is specified using the **numeric constant** icon  .
- Note that if you do not assign a loop counter, the default is to loop (i.e., repeat the set of commands inside the loop) only twice.
- Inside the loop is the section of your program that you want to repeat.
- **programming challenge**
  Modify the program you wrote earlier in which your robot moves in a square: change the program so that it uses a loop instead of repeating the same bits of code four times.
  Draw the code in the box below.



5. the LEGO light sensor

- The LEGO light sensor has a transmitter and a receiver. It transmits infrared light (also called "IR"), which bounces off objects and then returns in the direction of the receiver. The receiver records a value indicating how much light it read, which basically tells you about the brightness of the object that the light sensor is pointing at. The light sensor produces a value between 0 and 100, where 100 means very bright and 0 means very dark.
- Did you know that your TV remote control uses IR to talk to your television?
- **Here is how you can see what the light sensor's value is:**
  This is done by turning on the RCX and pressing the **VIEW** button until the arrow at the top of the screen is under the port the light sensor is plugged into. For example, my light sensor is connected to port 2. I would turn on the RCX and press the VIEW button twice. Next to the image of the robot man, there is a number between 0 and 100—this is the value being read by the light sensor.



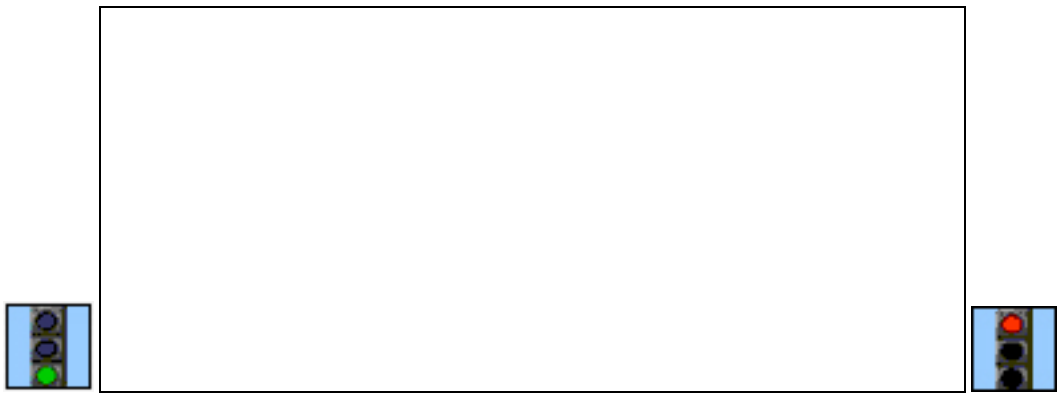
- In order to use the light sensor, you need to tell the RCX that the light sensor is plugged into port 2 (or whichever port you choose—we'll use #2 because we're already using #1 and #3 for the touch sensors inside the bumper). The easiest way to do this is to write and download a short program that uses the light sensor plugged into port #2. Try this example:

- Put your robot on at least four different locations: (a) white foamcore or paper, (b) black tape, (c) green tape, and (d) silver or gold tape. As shown in the previous step, **VIEW** the different values for the light sensor and record the values in the table below:

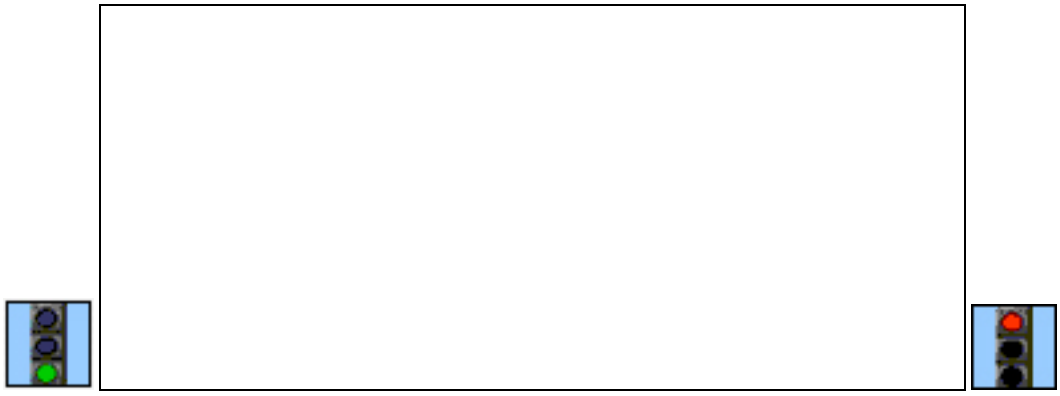
	black	white	green	silver/gold
light sensor value				

- **programming challenges**

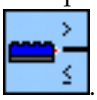
  (a) Program the robot to go forward until it sees something black, and then stops for one second, then goes backwards for two secconds. **Edit** the program in RoboLab, **download** it again and **test** it.

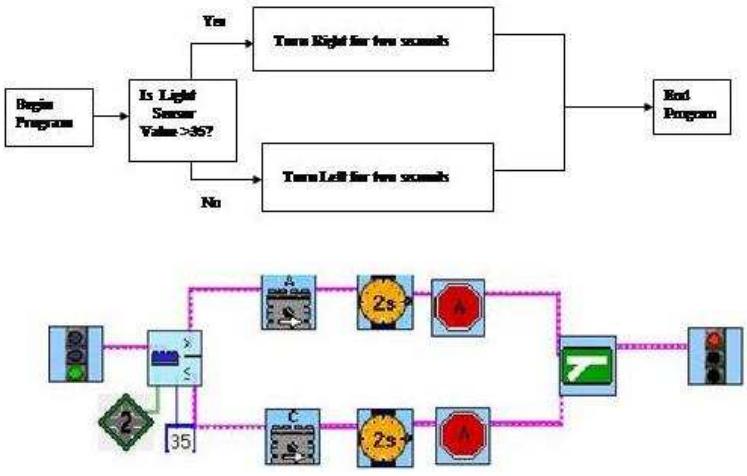


  (b) Program the robot to go forward until it sees something silver or gold, and then stops for one second, then turns to the left and goes forward for two seconds.



6. decision-making

- Up to now, your programs have been sequential and linear. In other words, your programs have been executing each command, starting from the **green light** and going until the **red light**, without skipping any commands. However there are situations where we might like to program the robot to do one action (i.e., go forward) when some **condition** is true (i.e., light sensor sees black) or else do some other action (i.e., go backward) when the condition is not true (i.e., light sensor sees white). In order to do this, we need a **decision-making** mechanism so that our robots can react to their environment **autonomously** (i.e., without a human touching it). This decision making ability is based on its sensory inputs (e.g., the value of the light sensor) and will make the robot a little more intelligent! Decision making is achieved by **conditional execution** in the programming environment. In RoboLab, by using **fork** structures, we can allow programs to behave differently based on different values of sensor inputs.
- **How does a fork work?**
  - When a fork is reached in a program, one of two **branches** will be taken based on the value read from one of the robot's sensors.
  - All of the fork commands on the forks palette use the greater-than ($>$) or less-than-or-equal ($\leq$) condition to determine which branch to execute. These conditions are based on a **threshold value**  . If the value read by the fork's sensor is greater than the threshold value, then the program follows the top branch. Otherwise, if the value read by the fork's sensor is less than or equal to the threshold value, then the program follows the bottom branch. For example, RoboLab can let a program make a decision depending on the value it gets from the **light sensor** when you use the **light sensor fork**  .
  - You must attach the correct port number modifier  to the fork where your actual sensor is conncected.
  - A **fork merge**  is required at some point in the program with any fork command. It completes your conditional structure and brings the two branches of the program back together.
- **Example:**



- Create this program in RoboLab and use the light sensor value returned when the robot sees black tape as the threshold value. **REMEMBER:**
  - The light sensor fork requires you to specify a threshold value. For instance, the above example uses a light sensor fork with a threshold value of 35—but you need to **calibrate** your robot to determine which value it reads when it sees black tape, and then use that value instead of 35.
  - Don't forget to attach the correct port number to the fork. For example, our light sensor is attached to **port 2**.

- **programming challenge**

  Program the robot to go forward.
  If it sees something black, it should stop for one second, then go backwards for two seconds, then go forward again.
  If it sees something silver or gold, it should stop for one second, then turn to the left and go forward again.
  It should do this behavior forever.

  First, draw a **flowchart**, like the one above, to describe the way you want your robot to behave.

  Then, write the code in RoboLab and when it works, copy it below.