

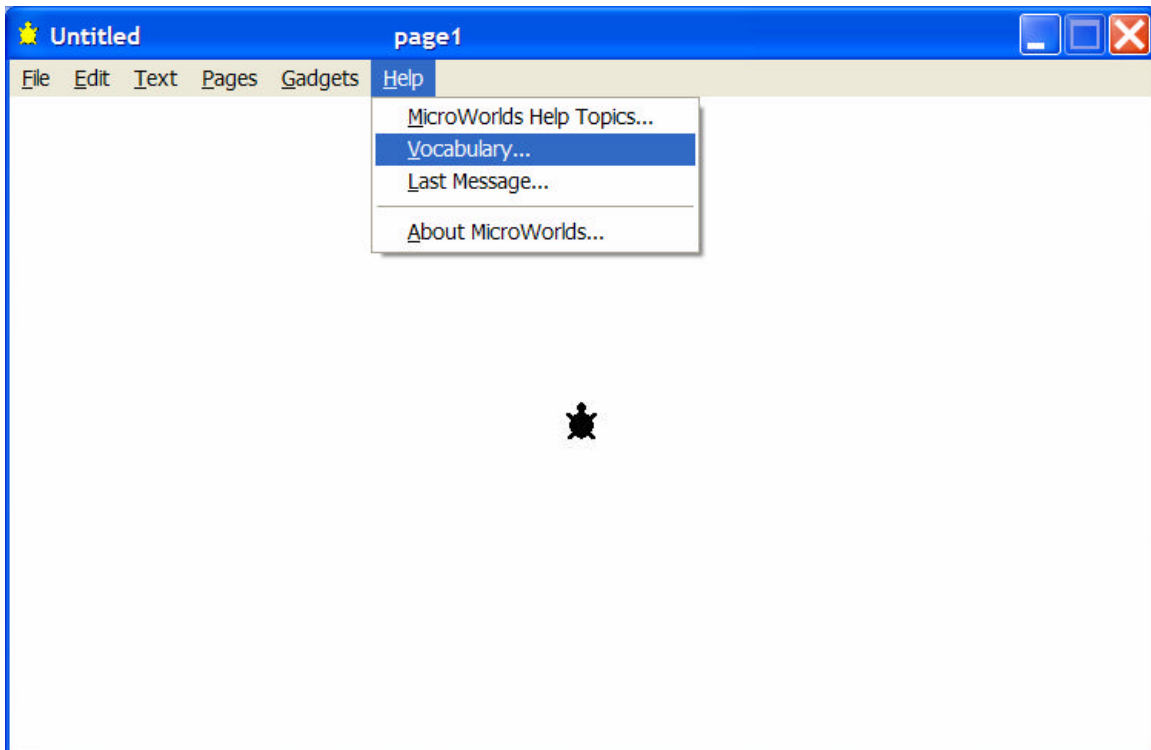
A Guide to the MicroWorlds 2.0 Vocabulary

The MicroWorlds 2.0 Vocabulary describes the Logo programming language. This language can be used in the command center or inside programmable objects like turtles and colors.

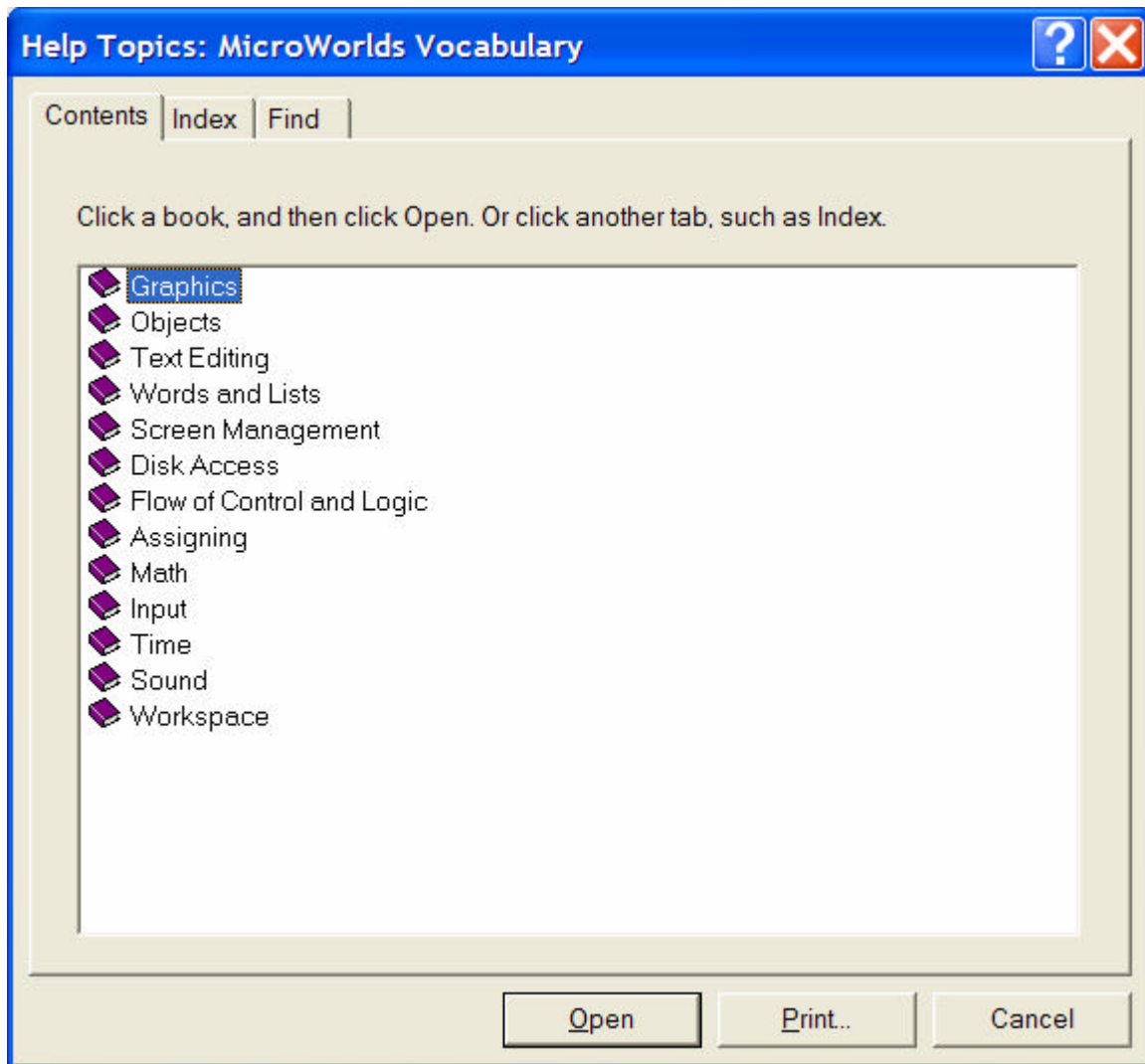
The Logo language is made up of instructions together with information that explain how the instructions are carried out.

Some of these instructions come with MicroWorlds when you buy the software. These instructions are called “primitives”. You can also create your own custom instructions called “procedures”.

To see what primitives are available in the MicroWorlds vocabulary, select “Vocabulary...” from the “Help” menu.

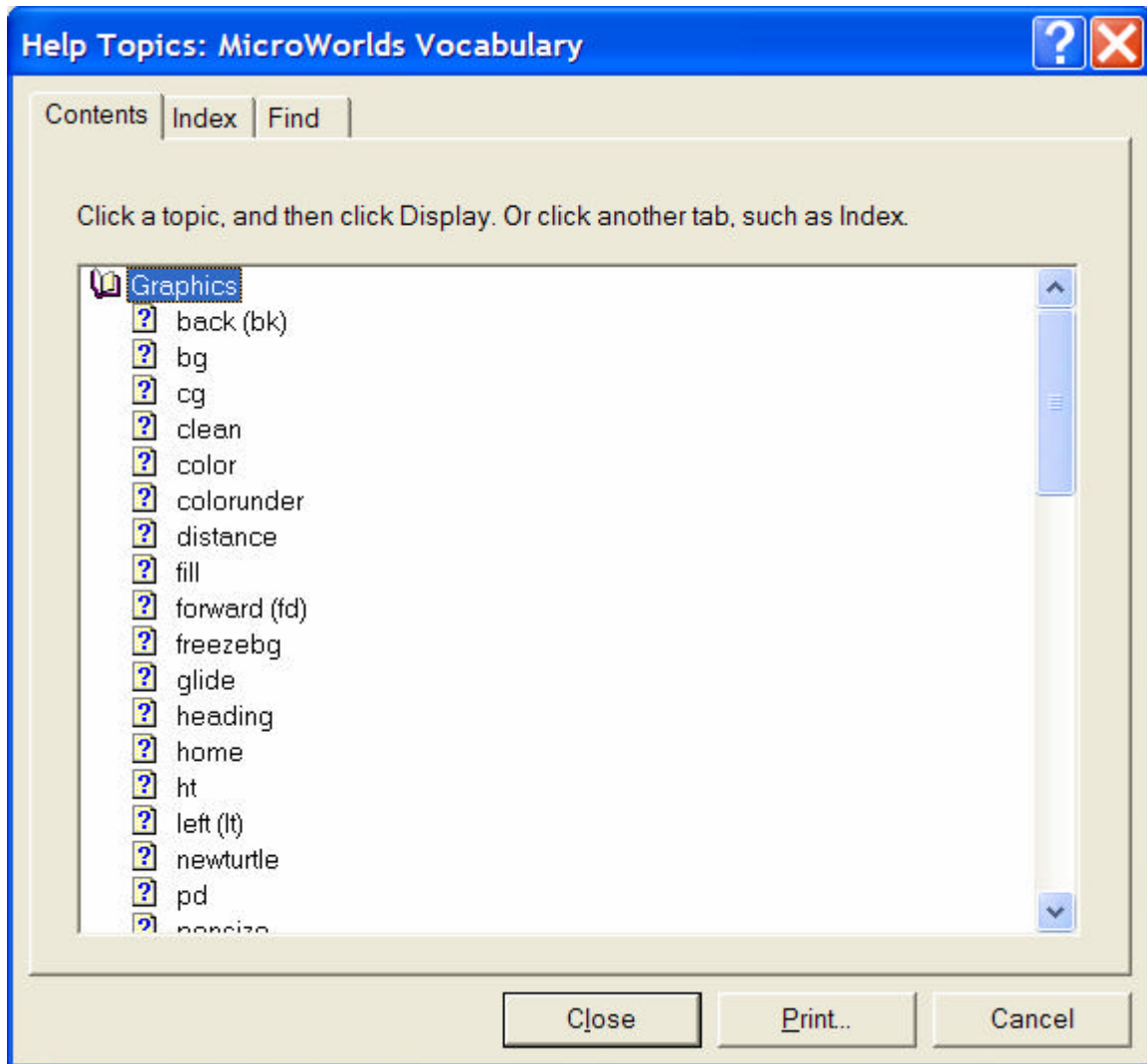


This will bring up a new window that looks like this:

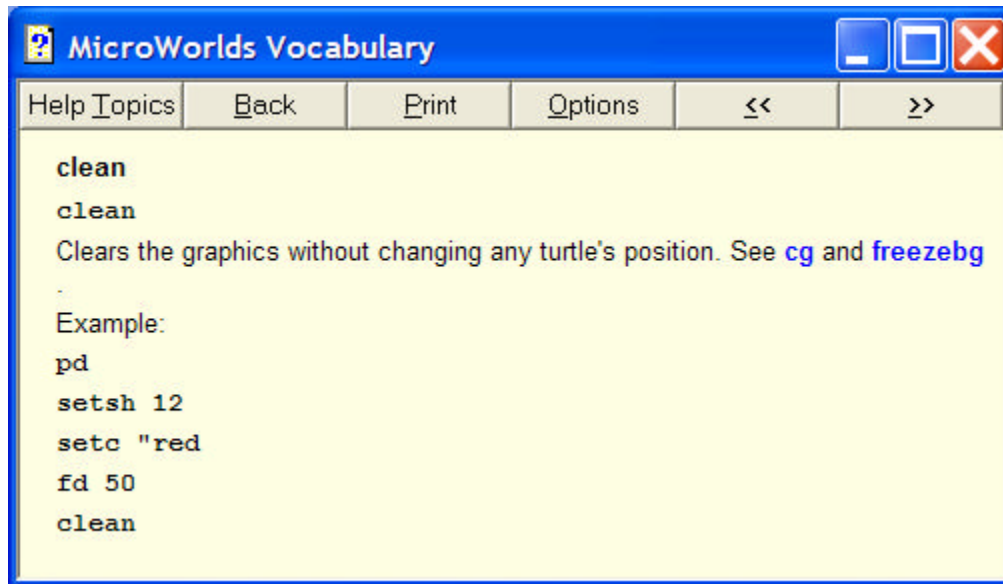


From this window, we can see that the MicroWorlds vocabulary is divided into different sections.

Let's take a look at the Graphics section. Double-click on the word "Graphics". The book next to the word "Graphics" should open up and show a bunch of "pages" like this:



You can click on any of the pages you like to see what's in them. For now, we're going to take a closer look at the page labeled "clean":



At the top of the page are buttons to help you navigate the help pages. The “Help Topics” button will take you back to the vocabulary list.

In the content area, the first thing we see is the word **clean**, which is the name of the primitive we’re looking at.

Next, we see

```
clean
```

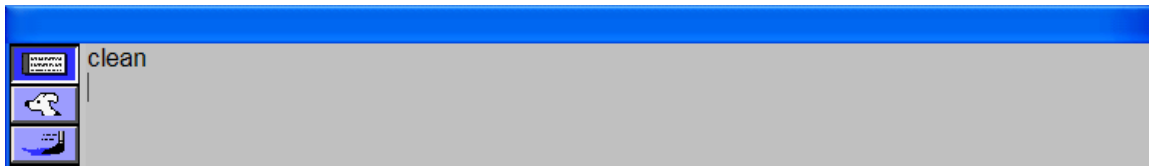
This line describes how the primitive is used. The next line explains what “clean” does. The Example is a little complicated, so, for now, we’ll try out the instruction another way.

Bring up the Drawing Center by clicking on the brush to the left of the Centers area:



Now draw something on the current page of your project. Try out the different drawing tools to see what they do.

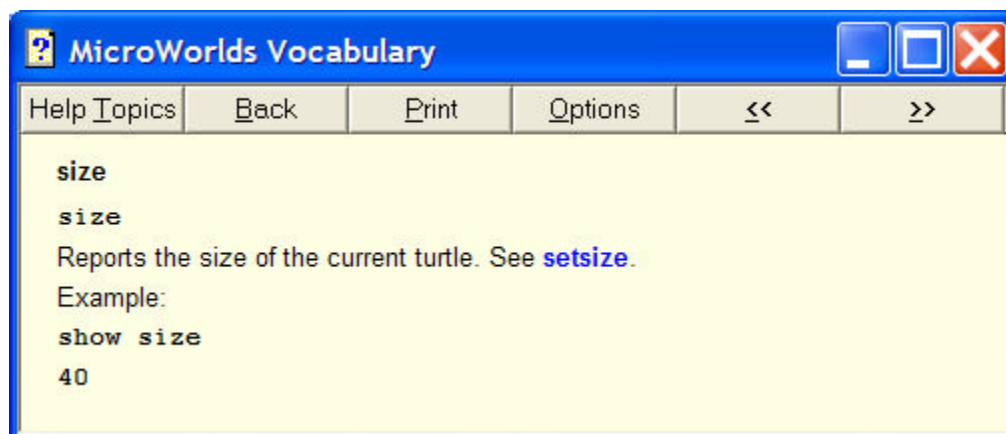
When you are finished, bring the Command Center back up by clicking on the Command Center button. Type the word “clean” in the command center and press Enter. What happens?



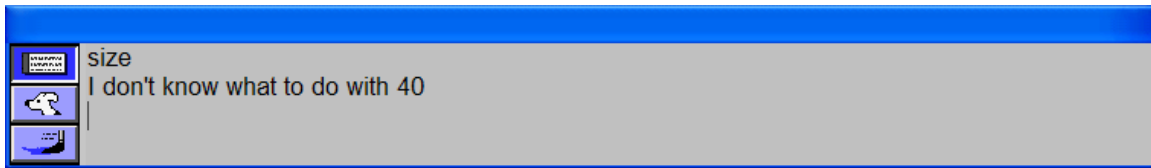
All instructions either make changes on the screen or give us information. If an instruction changes something on the screen, we call it a *command*. If an instruction reports some piece of information, we call it a *reporter*.

The instruction “clean” is a command. It changes something on the screen (What does it change?).

An example of a reporter is **size**:



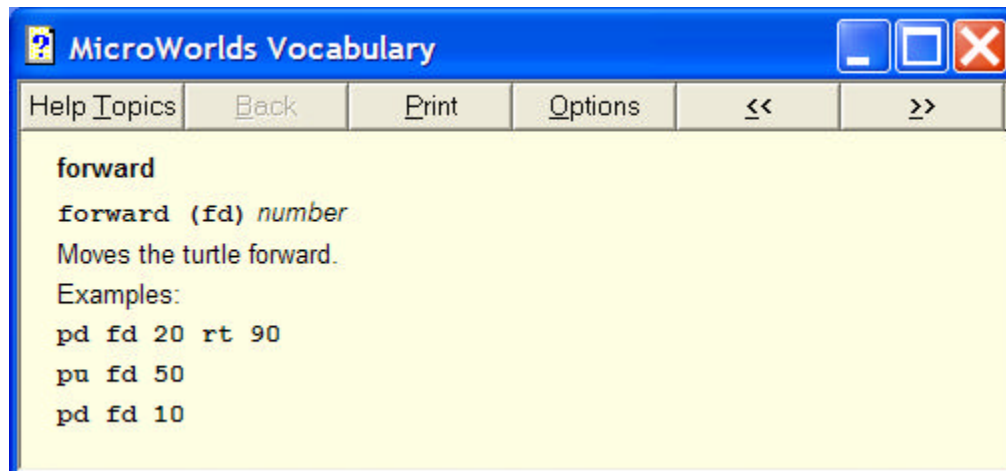
Try typing `size` into the Command Center and pressing Enter. What happens?



The size instruction reported the size of the turtle, 40. But the number 40 by itself does not tell the Command Center how to change the screen.

But why does it say “I don’t know what to do with 40”? Because there are some commands that can use the number 40 (or other kinds of information) to make screen changes.

For example, let’s look at the vocabulary page labeled “forward (fd)”. In this section, we will see how the forward command uses information to tell it how far to go.



In the content area, the first thing we see is the word **forward**, which is the name of the primitive we’re looking at.

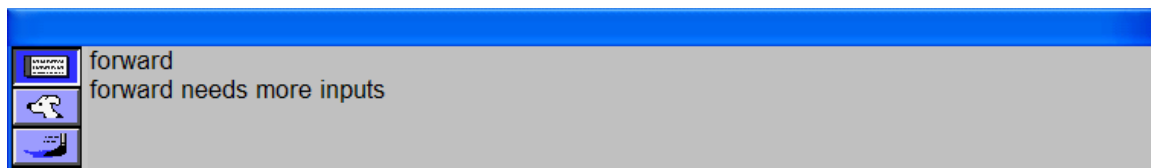
Next, we see

```
forward (fd) number
```

The word in parentheses (fd) is a synonym of the command. Synonyms are common in English, for example “fast” is a synonym for “rapid” or “quick”. In Logo, synonyms are often used to give abbreviations for an instruction.

The description also has word *number* in italics. We’ll talk about this in a minute. First, go to the command center and try typing in the word “forward” or “fd” and see what happens.

If you spelled either word correctly, you probably got something like this:



What’s an input?

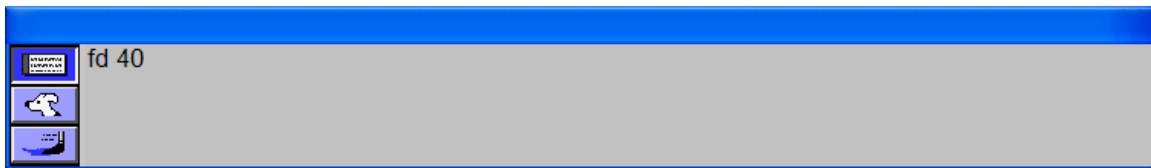
We said earlier that some commands can use information provided by reporters to make screen changes. The information given to commands is called input. Let’s take another look at the vocabulary description of forward.

```
forward (fd) number
```

The *number* in italics describes the input for the forward command.

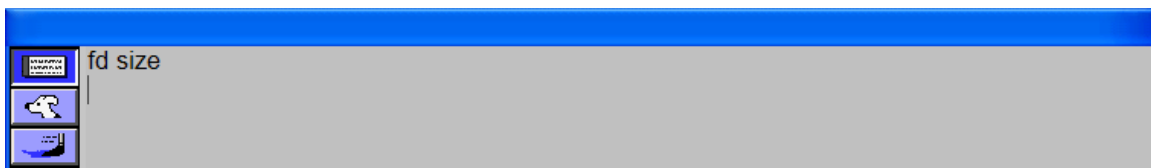
There are different ways to deliver inputs to commands. One way is to pass them directly.

For example, we can type the following into the Command Center (try it):



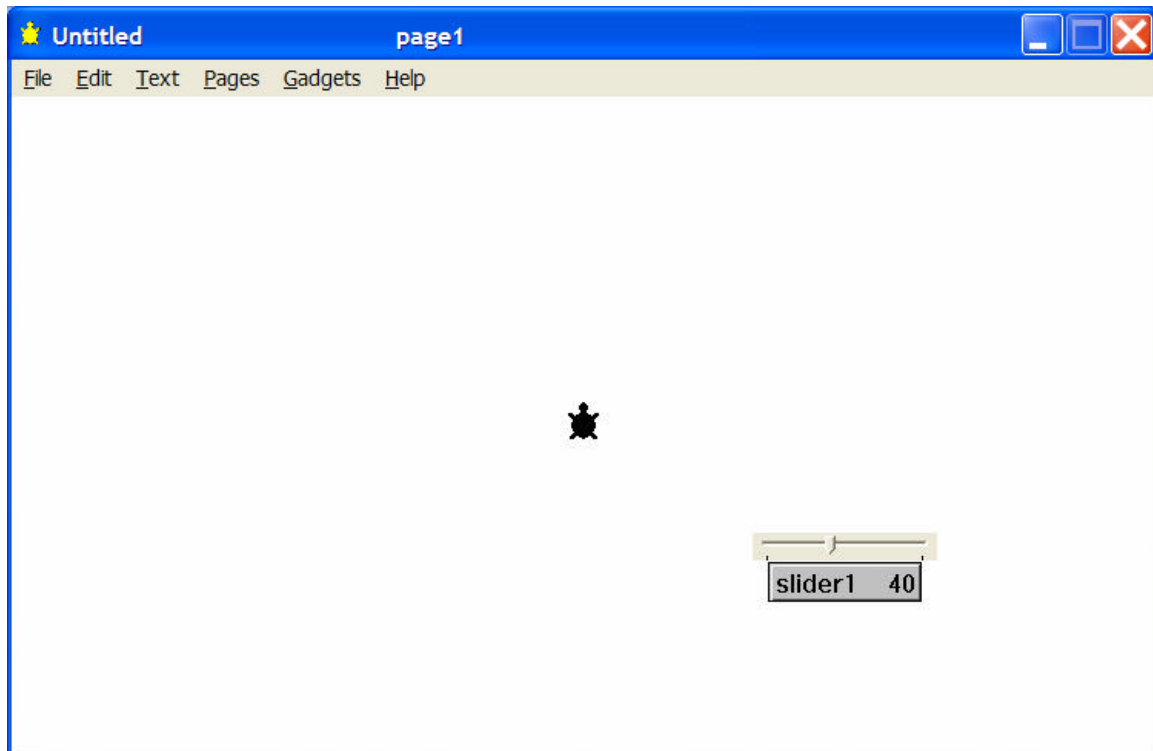
What happens?

We can also pass input to a command through a reporter. Try this:

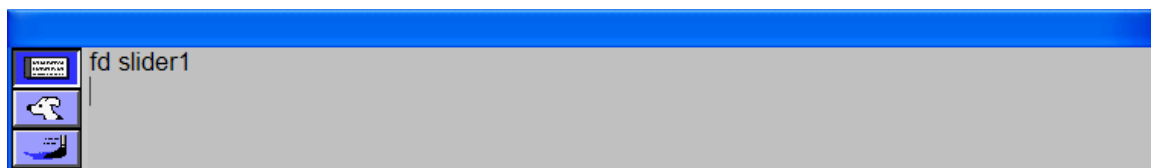


Notice (assuming the size of your turtle is 40) that we moved the turtle forward by the same amount each time. The Command Center doesn't care how the input gets reported. It just knows that it needs input.

Here's another way to deliver input to the **forward** command. Create a slider like the one below:



Now type the following into the Command Center (Don't forget to press Enter):



If your slider was set to 40, then your turtle once again moved forward 40.

Here are some more examples of instructions with different inputs from the MicroWorlds Vocabulary:

```
back (bk) number
towards turtle-name
setshape (setsh) name-or-number
setpos [x y]
ask who instruction-list
```

All inputs to commands and reporters are WORDs or LISTs

Examples of WORDs are: "false 7 "x "hello "t1

Examples of LISTs are: [-40 72] [You win!] [fd 30 rt 90]

Words and lists can be delivered directly to commands as CONSTANTs, generated by REPORTERs, or extracted from VARIABLEs.

To deliver a word or list as a CONSTANT, just type in the word or list directly.

```
"goodbye [50 50] [!t 90] "true [Nice job!] 18 "t3 "s
```

To extract a word or list from a VARIABLE, type a colon (:) followed by the name of the VARIABLE.

```
:keyTyped :startpos :foo :person
```

To generate a word or list with REPORTER, type the REPORTER plus inputs needed by the REPORTER (if any).

```
touching? "t1 "t2  
colorunder = 15  
equal? :answer 15  
word? "hello  
pos  
heading
```

There are two important things to notice here:

1. The equal sign (=) is a special kind of reporter because it appears between its inputs instead of in front of them. It reports “true if its inputs are equal and “false if its inputs are not equal. Compare this reporter with the **equal?** reporter. There are other special reporters like the equal sign, including +, -, *, /, < and >.
2. Inputs to a reporter can be constants, variables, or other reporters.

Here are some more examples of words and lists that are used as inputs to commands and reporters:

word

Any word.

list

Any list.

word-or-list

word-or-list1

word-or-list2

Some commands and reporters can take any word or list as input. Examples are the command **show** (which takes a single word or list) and the reporter **equal?** (which takes a word or list followed by another word or list).

number
number1 number2

These keywords stand for numbers like 1, 2, etc. Numbers are words. They are a special case of words because they don't take a quote (") in front when you write them as constants.

true-or-false
true-or-false1
true-or-false2

A *true-or-false* is a word that can have one of two values, either "true or "false

char

A *char* is a word that has exactly one character. So, for example, "x is a char, but "xy is not a char.

who
turtle-name

A word that names a turtle. So, for example, if you have a turtle named t1, then "t1 would work. Notice that there is also a reporter named **who** that reports the current turtle.

instruction-list
list-to-run

A list of one or more instructions. For example, here is an instruction list:

[fd 50 rt 90 fd 50]

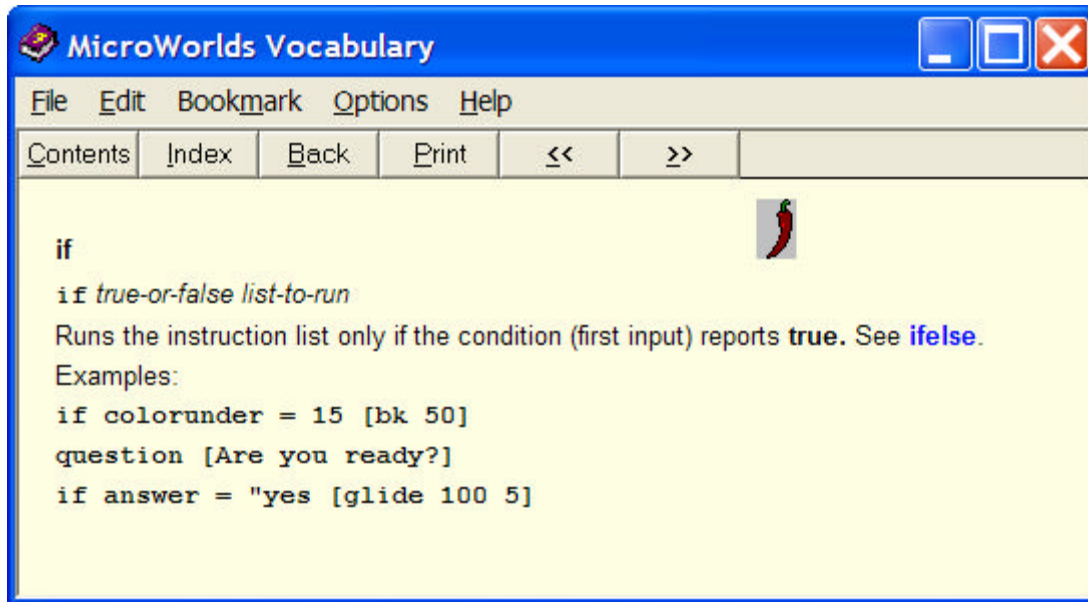
path

A word that names a directory, like "C:\MicroWorlds.

[x y]

A list of two coordinates.

Now let's take a look at **if**:



Remember, instructions don't care if their inputs are **CONSTANTS**, **VARIABLES**, or **REPORTERS**

So all of the statements below are legal. The *true-or-false* inputs have been circled. The *list-to-run* inputs have been underlined.

```
if equal? colorunder 5 [setpos [40 0]]
if greater? xcor 100 [setshape 20]
if less? size 160 [setsize size + 10]
if "true [rt 90]
make "instruction-list [fd 20 rt 90]
if touching? "t1 "t2 :instruction-list
if "true :instruction-list
if list? :instruction-list :instruction-list
make "test "true
if :test [rt 90]
if :test :instruction-list
```