Unit 3: Loops and Variables

Procedural Abstraction

Learning Objectives

- 4: The student can use programming as a creative tool.
- 20: The student can use abstractions (procedures) to manage complexity in a program.

Readings/Lectures

- Blown to Bits: Chapter 3 (http://www.bitsbook.com/wp-content/uploads/2008/12/chapter3.pdf)
- Lecture Slides 3.01: Functions (/bjc-course/curriculum/03-build-your-own-blocks/readings/01-functions-slides.pdf)
- Lecture Video 3.02: Functions (/bjc-course/curriculum/03-build-your-own-blocks/readings/02-functions-video.mp4)

Labs/Exercises

- Lab 3.01: Build your own blocks (/bjc-course/curriculum/03-build-your-own-blocks/labs/01-build-your-own-blocks)
- Lab 3.02: Project: Brick Wall (/bjc-course/curriculum/03-build-your-own-blocks/labs/02-brick-wall-project)

Who uses steganography today, if anyone? It is very hard to know. *USA Today* reported that terrorists were communicating using steganography in early 2001. A number of software tools are freely available that make steganography easy. Steganographic detectors—what are properly known as steganalysis tools—have also been developed, but their usefulness as yet seems to be limited. Both steganography and steganalysis software is freely available on the World Wide Web (see, for example, `www.cotse.com/tools/stega.htm` and `www.outguess.org/detection.php`).

The use of steganography to transmit secret messages is today easy, cheap, and all but undetectable. A foreign agent who wanted to communicate with parties abroad might well encode a bit string in the tonal values of an MP3 or the color values of pixels in a pornographic image on a web page. So much music and pornography flows between the U.S. and foreign countries that the uploads and downloads would arouse no suspicion!

The Scary Secrets of Old Disks

By now, you may be tempted to delete all the files on your disk drive and throw it away, rather than run the risk that the files contain unknown secrets. That isn't the solution: Even deleted files hold secrets!

A few years ago, two MIT researchers bought 158 used disk drives, mostly from eBay, and recovered what data they could. Most of those who put the disks up for sale had made some effort to scrub the data. They had dragged files into the desktop trash can. Some had gone so far as to use the Microsoft Windows FORMAT command, which warns that it will destroy all data on the disk.

Yet only 12 of the 158 disk drives had truly been sanitized. Using several methods well within the technical capabilities of today's teenagers, the researchers were able to recover user data from most of the others. From 42 of the disks, they retrieved what appeared to be credit card numbers. One of the drives seemed to have come from an Illinois automatic teller machine and contained 2,868 bank account numbers and account balances. Such data from single business computers would be a treasure trove for criminals. But most of the drives from home computers also contained information that the owners would consider extremely sensitive: love letters, pornography, complaints about a child's cancer therapy, and grievances about pay disputes, for example. Many of the disks contained enough data to identify the primary user of the computer, so that the sensitive information could be tied back to an individual whom the researchers could contact.

CLOUD COMPUTING

One way to avoid having problems with deleted disk files and expensive document-processing software is not to keep your files on your disks in the first place! In "cloud computing," the documents stay on the disks of a central service provider and are accessed through a web browser. "Google Docs" is one such service, which boasts very low software costs, but other major software companies are rumored to be exploring the market for cloud computing. If Google holds your documents, they are accessible from anywhere the Internet reaches, and you never have to worry about losing them—Google's backup procedures are better than yours could ever be. But there are potential disadvantages. Google's lawyers would decide whether to resist subpoenas. Federal investigators could inspect bits passing through the U.S., even on a trip between other countries.

The users of the computers had for the most part done what they thought they were supposed to do—they deleted their files or formatted their disks. They probably knew not to release toxic chemicals by dumping their old machines in a landfilll, but they did not realize that by dumping them on eBay, they might be releasing personal information into the digital environment. Anyone in the world could have bought the old disks for a few dollars, and all the data they contained. What is going on here, and is there anything to do about it?

Disks are divided into blocks, which are like the pages of a book—each has an identifying address, like a page number, and is able to hold a few hundred bytes of data, about the same amount as a page of text in a book. If a document is larger than one disk block, however, the document is typically not stored in consecutive disk blocks. Instead, each block includes a piece of the document, and the address of the block where the document is continued. So the entire document may be physically scattered about the disk, although logically it is held together as a chain of references of one block to another. Logically, the structure is that of a magazine, where articles do not necessarily occupy contiguous pages. Part of an article may end with "Continued on page 152," and the part of the article on page 152 may indicate the page on which it is continued from there, and so on.

Because the files on a disk begin at random places on disk, an *index* records which files begin where on the disk. The index is itself another disk file, but one whose location on the disk can be found quickly. A disk index is very much like the index of a book—which always appears at the end, so readers know where to look for it. Having found the index, they can quickly find the page number of any item listed in the index and flip to that page.

Why aren't disks themselves organized like books, with documents laid out on consecutive blocks? Because disks are different from books in two important respects. First, they are dynamic. The information on disks is constantly being altered, augmented, and removed. A disk is less like a book than like a three-ring binder, to which pages are regularly added and removed as information is gathered and discarded. Second, disks are perfectly re-writable. A disk block may contain one string of `0`s and `1`s at one moment, and as a result of a single writing operation, a different string of `0`s and `1`s a moment later. Once a `0` or a `1` has been written in a particular position on the disk, there is no way to tell whether the bit previously in that position was a `0` or a `1`. There is nothing analogous to the faint traces of pencil marks on paper that are left after an erasure. In fact, there is no notion of "erasure" at all on a disk—all that ever happens is replacement of some bits by others.

Because disks are dynamic, there are many advantages to breaking the file into chained, noncontiguous blocks indexed in this way. For example, if the file contains a long text document and a user adds a few words to the middle of the text, only one or two blocks in the middle of the chain are affected. If enough text is added that those blocks must be replaced by five new ones, the new blocks can be logically threaded into the chain without altering any of the other blocks comprising the document. Similarly, if a section of text is deleted, the chain can be altered to "jump over" the blocks containing the deleted text.

Blocks that are no longer part of any file are added to a "pool" of available disk blocks. The computer's software keeps track of all the blocks in the pool. A block can wind up in the pool either because it has never been used or because it has been used but abandoned. A block may be abandoned because the entire file of which it was part has been deleted or because the file has been altered to exclude the block. When a fresh disk block is needed for any purpose—for example, to start a new file or to add to an existing file—a block is drawn from the pool of available blocks.

What Happens to the Data in Deleted Files?

Disk blocks are not re-written when they are abandoned and added to the pool. When the block is withdrawn from the pool and put back to work as part of another file, it is overwritten and the old data is obliterated. But until then, the block retains its old pattern of zeroes and ones. The entire disk file may be intact—except that there is no easy way to find it. A look in the index will reveal nothing. But "deleting" a file in this way merely removes the index entry. The information is still there on the disk somewhere. It has no more been eradicated than the information in a book would be expunged by tearing out the index from the back of the volume. To find something in a book without an index, you just have to go through the book one page at a time looking for it—tedious and time-consuming, but not impossible.

And that is essentially what the MIT researchers did with the disks they bought off eBay—they went through the blocks, one at a time, looking for recognizable bit patterns. A sequence of sixteen ASCII character codes representing decimal digits, for example, looks suspiciously like a credit card number. Even if they were unable to recover an entire file, because some of the blocks comprising it had already been recycled, they could recognize significant short character strings such as account numbers.

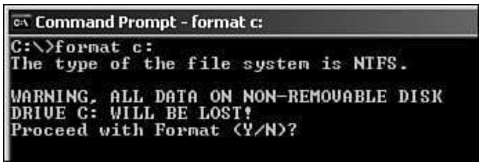
Of course, there would be a simple way to prevent sensitive information from being preserved in fragments of "deleted" files. The computer could be programmed so that, instead of simply putting abandoned blocks into the pool, it actually over-wrote the blocks, perhaps by "zeroing" them—that is, writing a pattern of all `0`s. Historically, computer and software manufacturers have thought the benefits of zeroing blocks far less than the costs. Society has not found "data leakage" to be a critical problem until recently—although that may be changing. And the costs of constantly zeroing disk blocks would be significant. Filling blocks with zeroes might take so much time that the users would complain about how slowly their machines were running if every block were zeroed immediately. With some clever programming the process could be made unnoticeable, but so far neither Microsoft nor Apple has made the necessary software investment.

THE LAW ADJUSTS

Awareness is increasing that deleted data can be recovered from disks. The Federal Trade Commission now requires "the destruction or erasure of electronic media containing consumer information so that the information cannot practicably be read or reconstructed," and a similar provision is in a 2007 Massachusetts Law about security breaches.

And who has not deleted a file and then immediately wished to recover it? Happily for all of us who have mistakenly dragged the wrong file into the trash can, as computers work today, deleted files are not immediately added to the pool—they can be dragged back out. Files can be removed only until you execute an "Empty trash" command, which puts the deleted blocks into the pool, although it does not zero them.

But what about the Windows "FORMAT" command, shown in Figure 3.16? It takes about 20 minutes to complete. Apparently it is destroying all the bits on the disk, as the warning message implies. But that is not what is happening. It is simply looking for faulty spots on the disk. Physical flaws in the magnetic surface can make individual disk blocks unusable, even though mechanically the disk is fine and most of the surface is flawless as well. The FORMAT command attempts to *read* every disk block in order to identify blocks that need to be avoided in the future. Reading every block takes a long time, but rewriting them all would take twice as long. The FORMAT command identifies the bad blocks and re-initializes the index, but leaves most of the data unaltered, ready to be recovered by an academic researcher—or an inventive snooper.



```
C:\>format c:
The type of the file system is NTFS.

WARNING, ALL DATA ON NON-REMOVABLE DISK
DRIVE C: WILL BE LOST!
Proceed with Format (Y/N)?
```

FIGURE 3.16 Warning screen of Microsoft Windows FORMAT command. The statement that all the data will be lost is misleading—in fact, a great deal of it can be recovered.

As if the problems with disks were not troubling enough, exactly the same problems afflict the memory of cell phones. When people get rid of their old phones, they forget the call logs and email messages they contain. And if they do remember to delete them, using the awkward combinations of button-pushes described deep in the phone's documentation, they may not really have accomplished what they hoped. A researcher bought ten cell phones on eBay and recovered bank account numbers and passwords, corporate strategy plans, and an email exchange between a woman and her married boyfriend, whose wife was getting suspicious. Some of this information was recovered from phones whose previous owners had scrupulously followed the manufacturer's instructions for clearing the memory.

SOFTWARE TO SCRUB YOUR DISK

If you really want to get rid of all the data on your disk, a special "Secure empty trash" command is available on Macintosh computers. On Windows machines, DBAN is free software that really will zero your disk, available through `dban.sourceforge.net`, which has lots of other useful free software. Don't use DBAN on your disk until you are sure you don't want anything on it anymore!

In a global sense, bits turn out to be very hard to eradicate. And most of the time, that is exactly the way we want it. If our computer dies, we are glad that Google has copies of our data. When our cell phone dies, we are happy if our contact lists reappear, magically downloaded from our cellular service provider to our replacement phone. There are upsides and downsides to the persistence of bits.

Physical destruction always works as a method of data deletion. One of us uses a hammer; another of us prefers his axe. Alas, these methods, while effective, do not meet contemporary standards for recovery and recycling of potentially toxic materials.

Can Data Be Deleted Permanently?

Rumors arise every now and then that engineers equipped with very sensitive devices can tell the difference between a `0` that was written over a `0` on a disk and a `0` that was written over a `1`. The theory goes that successive writing operations are not perfectly aligned in physical space—a "bit" has width. When a bit is rewritten, its physical edges may slightly overlap or fall short of its previous position, potentially revealing the previous value. If such microscopic misalignments could be detected, it would be possible to see, even on a disk that has been zeroed, what the bits were *before* it was zeroed.

No credible authentication of such an achievement has ever been published, however, and as the density of hard disks continues to rise, the likelihood wanes that such data recovery can be accomplished. On the other hand, the places most likely to be able to achieve this feat are government intelligence agencies, which do not boast of their successes! So all that can be said for certain is that recovering overwritten data is within the capabilities of at most a handful of organizations—and if possible at all, is so difficult and costly that the data would have to be extraordinarily valuable to make the recovery attempt worthwhile.

Copies Make Data Hard to Delete

If your computer has ever been connected to a network, destroying its data will not get rid of copies of the same information that may exist on other machines. Your emails went to and from other people—who may have copies on their machines, and may have shared them with others. If you use Google's Gmail, Google may have copies of your emails even after you have deleted them. If you ordered some merchandise online, destroying the copy of the invoice on your personal computer certainly won't affect the store's records.

How Long Will Data Really Last?

As persistent as digital information seems to be, and as likely to disclose secrets unexpectedly, it also suffers from exactly the opposite problem. Sometimes electronic records become unavailable quite quickly, in spite of best efforts to save them permanently.

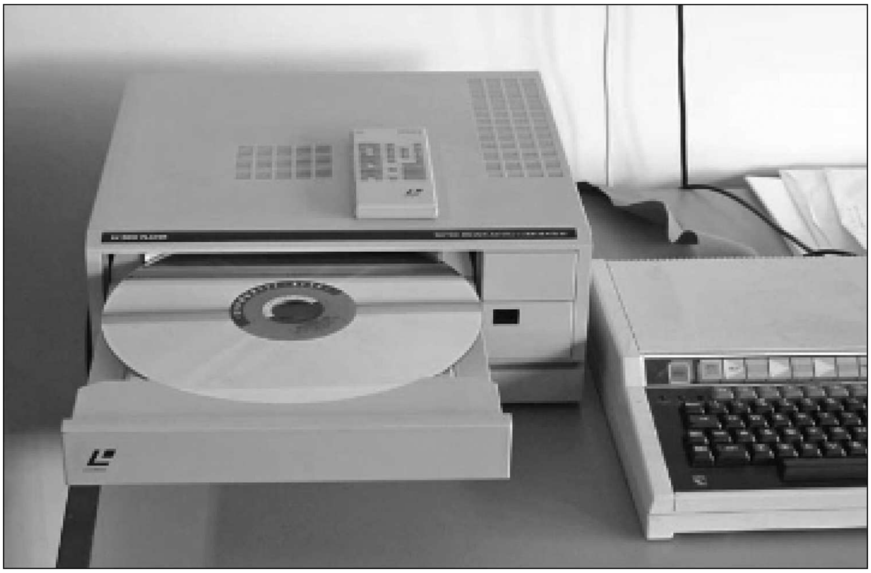
Figure 3.17 shows an early geopolitical and demographic database—the Domesday Book, an inventory of English lands compiled in 1086 by Norman monks at the behest of William the Conqueror. The Domesday Book is one of Britain's national treasures and rests in its archives, as readable today as it was in the eleventh century.

British National Archives.

FIGURE 3.17 The Domesday Book of 1086.

In honor of the 900th anniversary of the Domesday Book, the BBC issued a modern version, including photographs, text, and maps documenting how Britain looked in 1986. Instead of using vellum, or even paper, the material was assembled in digital formats and issued on 12-inch diameter video disks, which could be read only by specially equipped computers (see Figure 3.18). The project was meant to preserve forever a detailed snapshot of late twentieth-century Britain, and to make it available immediately to schools and libraries everywhere.

By 2001, the modern Domesday Book was unreadable. The computers and disk readers it required were obsolete and no longer manufactured. In 15 years, the memory even of how the information was formatted on the disks had been forgotten. Mocking the project's grand ambitions, a British newspaper exclaimed, "Digital Domesday Book lasts 15 years not 1000."



"Domesday Redux," from Ariadne, Issue 56.

FIGURE 3.18 A personal computer of the mid-1980s configured to read the 12-inch videodisks on which the modern "Domesday Book" was published.

Paper and papyrus thousands of years older even than the original Domesday Book are readable today. Electronic records become obsolete in a matter of years. Will the vast amounts of information now available because of the advances in storage and communication technology actually be usable a hundred or a thousand years in the future, or will the shift from paper to digital media mean the loss of history?

The particular story of the modern Domesday Book has a happy ending. The data was recovered, though just barely, thanks to a concerted effort by many technicians. Reconstructing the data formats required detective work on masses of computer codes (see Figure 3.19) and recourse to data structure books of the period—so that programmers in 2001 could imagine how others would have attacked the same data representation problems only 15 years earlier! In the world of computer science, "state of the art" expertise dies very quickly.

The recovered modern Domesday Book is accessible to anyone via the Internet. Even the data files of the original Domesday Book have been transferred to a web site that is accessible via the Internet.

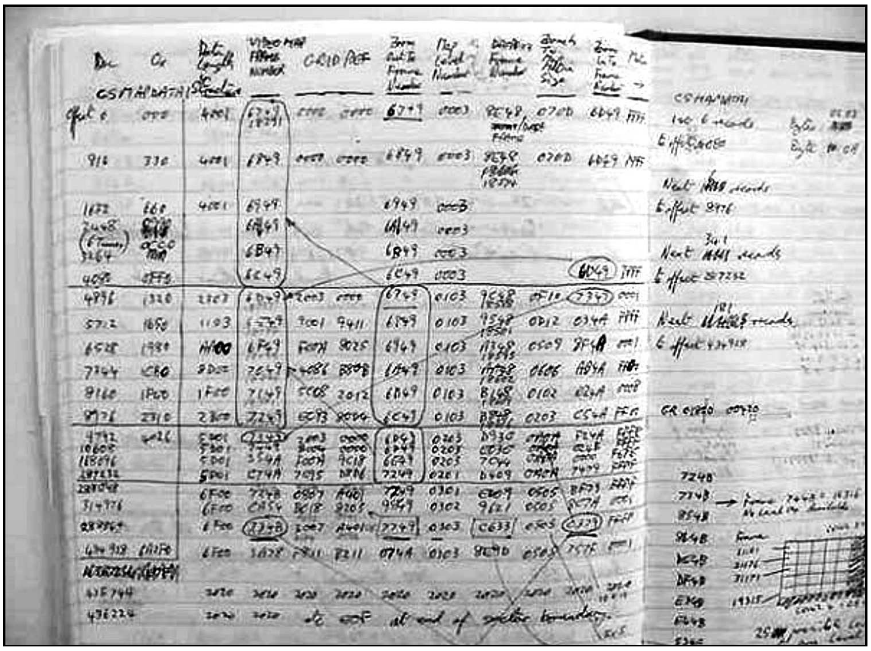


FIGURE 3.19 Efforts to reconstruct, shortly after the year 2000, the forgotten data formats for the modern "Domesday Book," designed less than 20 years earlier.

But there is a large moral for any office or library worker. We cannot assume that the back-ups and saved disks we create today will be useful even ten years from now for retrieving the vast quantities of information they contain. It is an open question whether digital archives—much less the box of disk drives under your bed in place of your grandmother's box of photographs—will be as permanent as the original Domesday Book. An extraordinary effort is underway to archive the entire World Wide Web, taking snapshots of every publicly accessible web page at period intervals. Can the effort succeed, and can the disks on which the archive is held

PRESERVING THE WEB

The Internet Archive (`www.archive.org`) periodically records "snapshots" of publicly accessible web pages and stores them away. Anyone can retrieve a page from the past, even if it no longer exists or has been altered. By installing a "Wayback" button (available from the Internet Archive) on your web browser, you can instantly see how any web page looked in the past—just go to the web page and click the Wayback button; you get a list of the archived copies of the page, and you can click on any of them to view it.

themselves be updated periodically so that the information will be with us forever?

Or would we be wisest to do the apparently Luddite thing: to print everything worth preserving for the long run—electronic journals, for example—so that documents will be preserved in the only form we are *certain* will remain readable for thousands of years?

✷

The digital revolution put the power to document ideas into the hands of ordinary people. The technology shift eliminated many of the intermediaries once needed to produce office memoranda and books. Power over the thoughts in those documents shifted as well. The authority that once accompanied the physical control of written and printed works has passed into the hands of the individuals who write them. The production of information has been democratized—although not always with happy results, as the mishaps discussed in this chapter tellingly illustrate.

We now turn to the other half of the story: how we get the information that others have produced. When power over documents was more centralized, the authorities were those who could print books, those who had the keys to the file cabinets, and those with the most complete collections of documents and publications. Document collections were used both as information choke points and as instruments of public enlightenment. Libraries, for example, have been monuments to imperial power. University libraries have long been the central institutions of advanced learning, and local public libraries have been key democratizing forces in literate nations.

If everything is just bits and everyone can have as many bits as they want, the problem may not be having the information, but finding it. Having a fact on the disk in your computer, sitting a few inches from your eyes and brain, is irrelevant, if what you want to know is irretrievably mixed with billions of billions of other bits. Having the haystack does you no good if you can't find your precious needle within it. In the next chapter, we ask: Where does the power now go, in the new world where access to information means finding it, as well as having it?

Functions

Computer Science Principles

What is a Function?

- In BYOB/SNAP, as well as other programming languages, we can take code blocks and put them in a new block that we can define.
- These new blocks are called **functions or procedures** because they perform a function for us, such as the ones that you have already been using in BYOB/SNAP.

What is a Function?

- We can create our script (or code block) once ~ and use it as many times as we want *without* having to re-write the code.
- We only need to "call" or "invoke" it.
- This is called *Procedural Abstraction*.

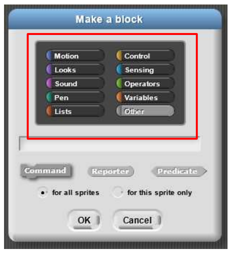
Functions in BYOB/SNAP

- To create a function in BYOB/SNAP, we are going to make our own block.
- Click on **make a block** at the bottom of the **Variables** tab.

- This will open up the **make a block** dialog box.

Make a Block in BYOB/SNAP

- You get to choose which palette the block will be in.
  - Choose the one that "makes sense"
  - Example
    - Our new block will be to draw a square, so we will choose Motion.

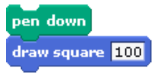


Types of Function Blocks

- There are three (3) types of function blocks you can create in BYOB/SNAP.
  1. Command
     - Creates a block that "makes something happen"
     - The same blocks that looks like puzzle pieces.
  2. Reporter
     - Creates a block that reports or returns a value.
     - These blocks look like some of your operator blocks, such as the () + () block. (rounded ends)
  3. Predicate
     - Creates a block that reports or returns either true or false.
     - These blocks look like some of your operator blocks, such as the () < () block. (pointed ends)

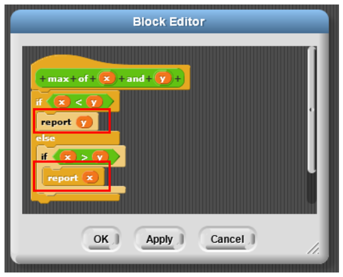
Command Function Blocks

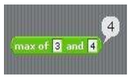
- A command function block is created to have a task carried out without any value being returned.
  - Such as drawing a square.



Reporter Function Blocks

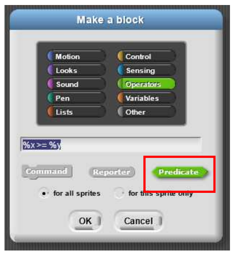
- Reporter blocks can report a value.
- You will need to use the report () block to return or report the value.

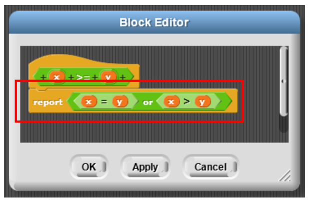




Predicate Function Blocks

- A predicate block returns either true or false.
- You will need to report other blocks that evaluate to either true or false.
- The Block Editor will open with a report block.



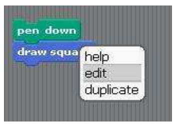


Arguments

- You have created a block that draws a square, but it only draws a square where each side is of length 100 steps.
- It would be great if we could specify how long we wanted each side to be.
- We will edit the block to accept an ***argument*** (or ***input***), which tells it the length of the square it has to draw.

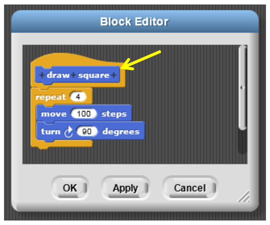
Arguments

- If you are modifying a block to add arguments, right-click on the new block and select **edit** to go back to the block editor.



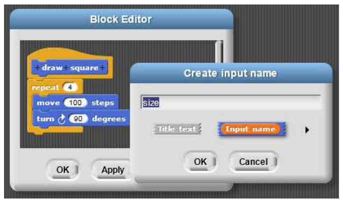
Arguments

- In the Block Editor, notice that when you move the mouse over the top row of the new block, some plus signs (+) show up.
- When you click on these plus signs, you can add more text or arguments.
- When you click on the text between the plus signs, you can delete or modify that text.
- Click on the plus sign at the far right as shown below:



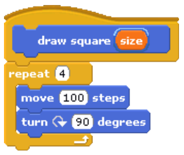
Arguments

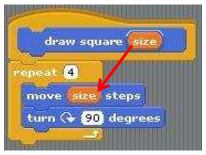
- When you click on the plus sign on the far right, you should get the following dialog box.
- With this dialog box, we can select if we want to add input (orange) or more text (blue).
- We want to add the input size, so we type size, select Input Name and click OK.



Arguments

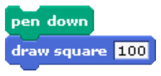
- Now, we have a variable inside our block definition.
- Drag the variable size down into the move block.
- Whenever we need a new copy of a variable, we just grab the copy from that variable in the top row.





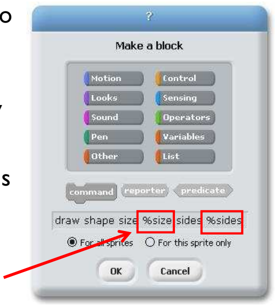
Arguments

- When we click OK, we will see that our draw square block now takes an argument.
- We can put different numbers in the blank and draw squares of different sizes!



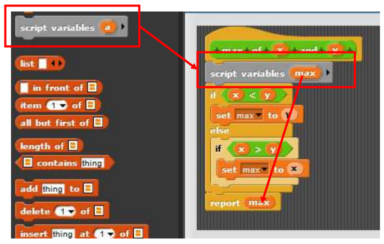
Arguments

- If you are just starting to create your function block, you can create the inputs to this block in exactly the same way as we did in the previous section, by clicking on the plus signs to add input.
- You can also type the names of the input as shown.
  - The percent signs (%) indicate that the word should be an input.



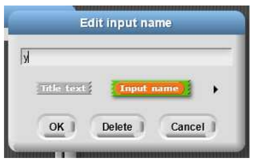
Script Variables

- A script variable is a variable that is only available inside that specific script.
- After you change the name, simply drag it into the Block Editor.
- You can then click and drag the variable's name from the script variables block into spots, such as the report box.



Input Types

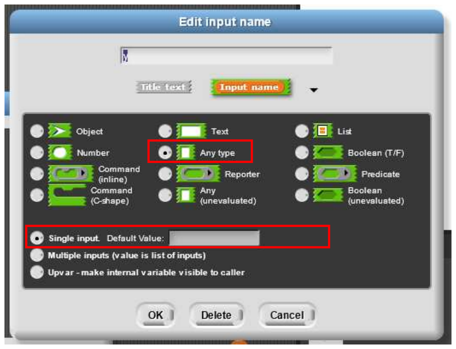
- You can limit the types of values that can be entered into a block's argument.
- Open the Block Editor for the function block and click on the input's name.
  - The following window displays.



Input Types

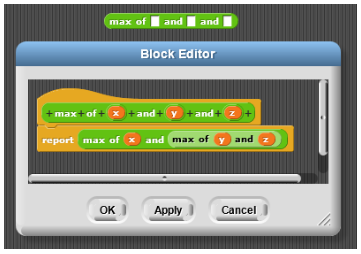
- Then, click on the right arrow in the pop-up box.
- This will open the dialog box shown on the next slide.
  - This allows us to specify the shape of the slot.
- We want a numbers-only slot (as shown selected below).
- We can also specify that we want the variable to have a *default* value.
  - This is similar to blocks like move that always start out with the default value 10.

Input Types



Composition of Functions

- Our custom-made blocks are blocks like any other, and we can use them in other block definitions.

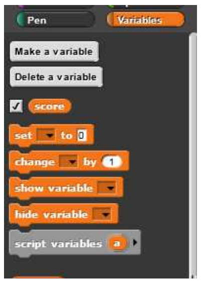

max of and and
Block Editor
+ max + of + x + and + y + and + z +
report max of x and max of y and z
OK
Apply
Cancel

Different Kinds of Variables

- Normal/Global Variables
- Sprite Specific Variables
- Arguments to a function
- Block Variables
- Script Variables

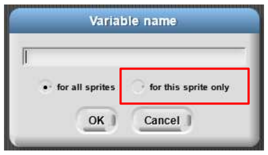
Normal/Global Variables

- These variables are made in the regular menu and can be used ANYWHERE!
- The variable "score" is an example.
- *These can be used by any sprite, in any block or in any script.*



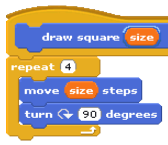
Sprite Specific Variables

- When you create a "normal/global" variable you can select that the variable is "For this sprite only".
- Then these variables will show up as variables listed below the line in the variables tab.
- *We recommend* <u>*not*</u> *using these variables in blocks.*



Arguments to a function

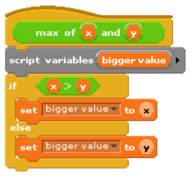
- A variable set by the person calling the function.
- We also refer to this as "**input**".
- *This can ONLY be used within the block editor.*



```
draw square (size)
repeat 4
    move (size) steps
    turn ↻ 90 degrees
```

Script Variables

- The "script variable" block gives us a variable that we can use inside of this script.
- *These can only be used in that particular script.*
- *The script could be a block script (shown below) or a regular script.*



```
max of (x) and (y)
script variables (bigger value)
if <(x) > (y)>
    set bigger value to (x)
else
    set bigger value to (y)
```

Curriculum (/bjc-course/curriculum) / Unit 3 (/bjc-course/curriculum/03-build-your-own-blocks) / Lab 1 (/bjc-course/curriculum/03-build-your-own-blocks/labs/01-build-your-own-blocks) /

Lab: Build Your Own Blocks

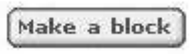
Let’s open up SNAP at http://snap.berkeley.edu/run (http://snap.berkeley.edu/run)

As you have seen, SNAP/BYOB is a powerful language that has a substantial repository of useful blocks for a variety of purposes. However, as you may have noticed, SNAP/BYOB does not have all the blocks that you may need, and often, it would prove useful if we could create new blocks.

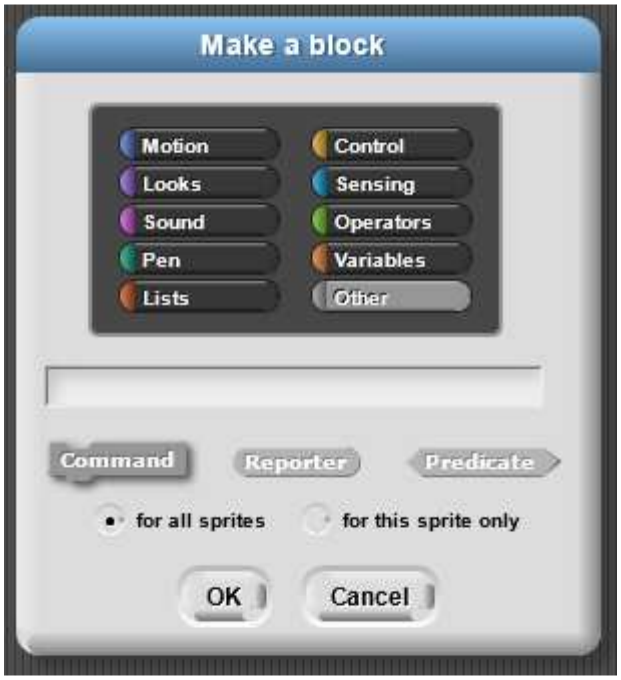
Make Your Own Block: A Tutorial

We are going to teach the computer how to draw a square using a block named `draw square`. Please follow the steps below:

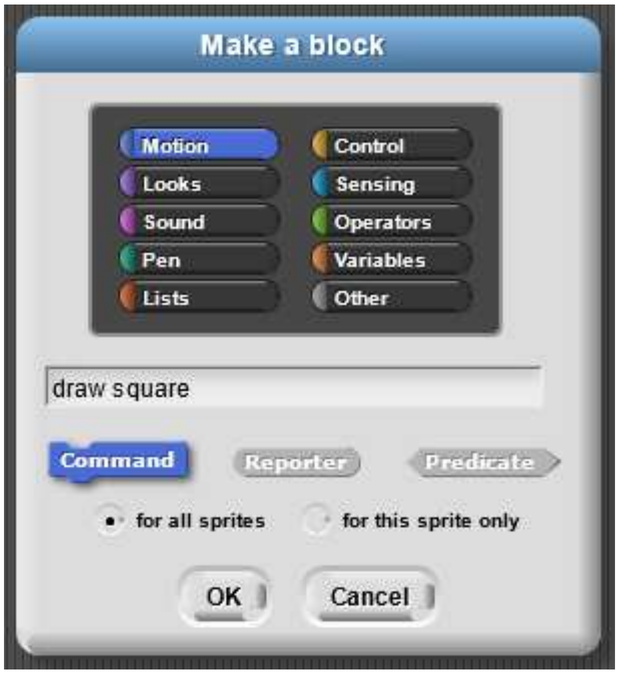
- Click on make a block at the bottom of the variables tab.



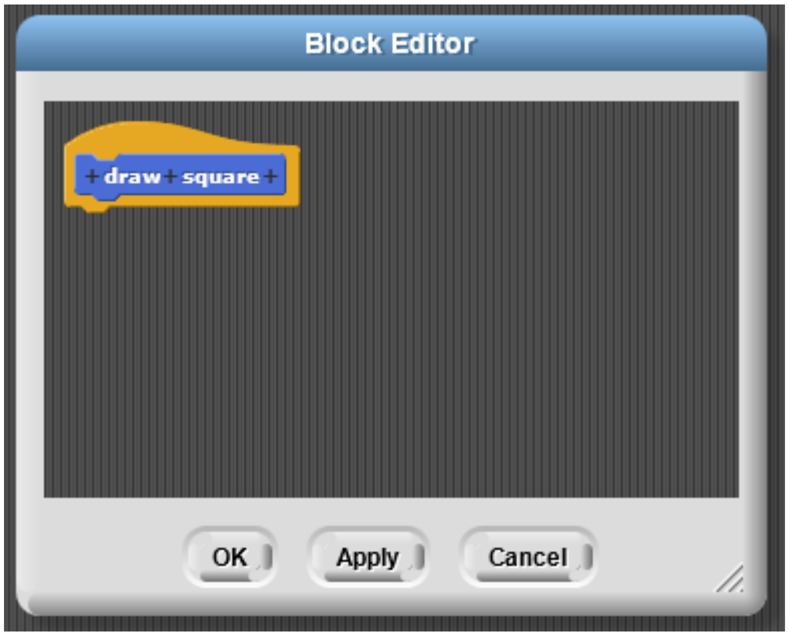
- This will open up the `make a block` dialog box. Now, you get to choose which tab the block should go into. Our block is going to draw a square, so let us choose `Motion`.



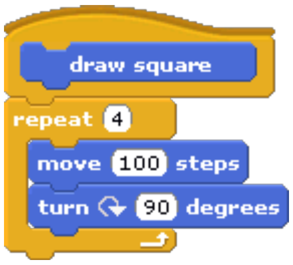
- When we selected `Motion`, the block became blue. We now have the option of making blocks of different shapes. Right now, however, we are just going to make a (regular) command block.



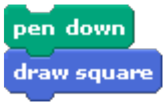
- When we click `OK`, we should see the block editor below.



- Use the blocks from the regular menus to create a script that draws a square, as shown below.


```
draw square
repeat 4
    move 100 steps
    turn ↻ 90 degrees
```

- When you click `OK`, you should be able to use this block as if it were a regular block. Since you created the block as a `Motion` block, it will end up at the bottom of the `Motion` tab.

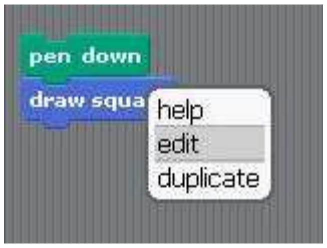

```
pen down
draw square
```

Congratulations! You have just created your own block!

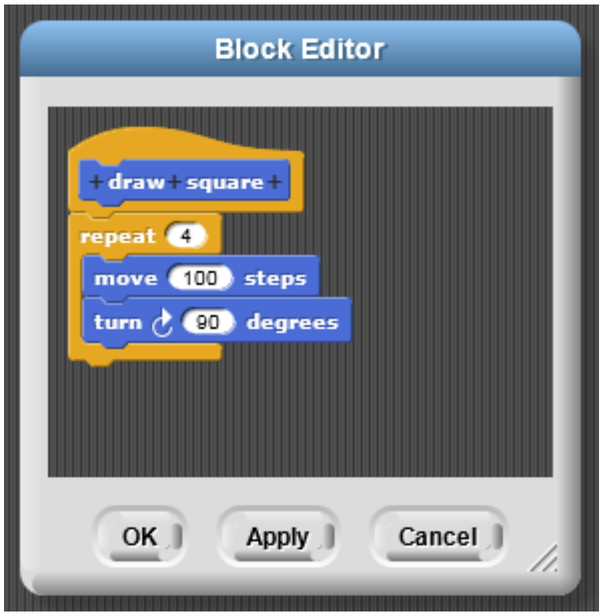
Improving the draw square block

You have created a block that draws a square, but it only draws a square where each side is of length `100` steps. It would be great if we could specify how long we wanted each side to be. We will edit the block to accept an *argument* (or *input*), which tells it the length of the square it has to draw.

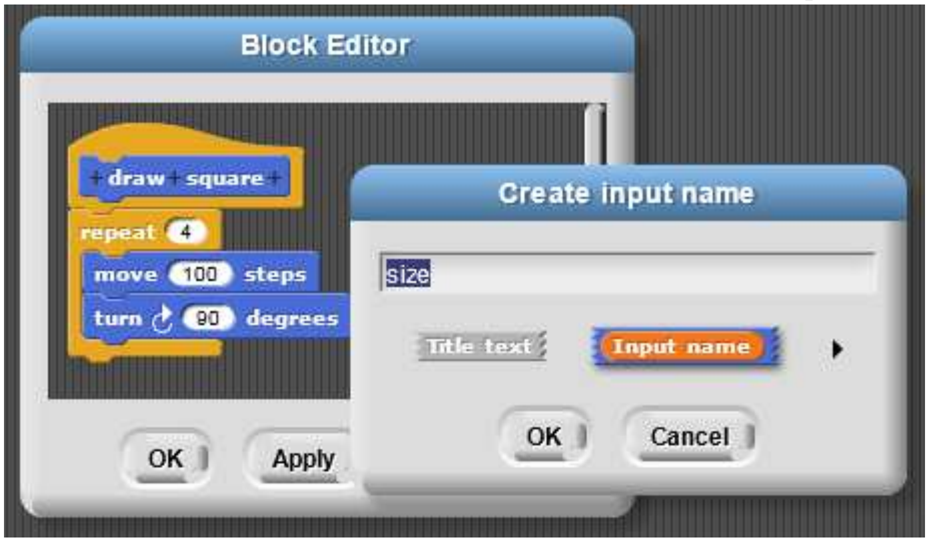
- We are going to go back and edit the block. Right-click on the new block and select edit to go back to the block editor.



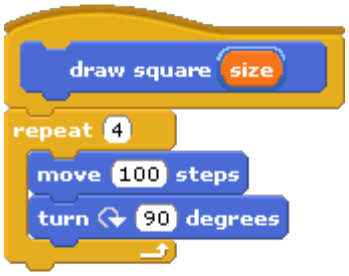
- In the `Block Editor`, notice that when you move the mouse over the top row of the new block, some plus signs (+) show up. When you click on these plus signs, you can add more text or arguments. When you click on the text between the plus signs, you can delete or modify that text. Click on the plus sign at the far right as shown below:



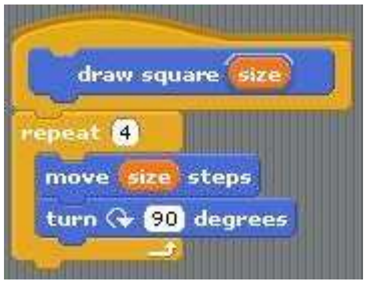
- When you click on the plus sign on the far right, you should get the following dialog box. With this dialog box, we can select if we want to add input (orange) or more text (blue). We want to add the input `size`, so we type `size`, select `Input Name` and click `OK`.



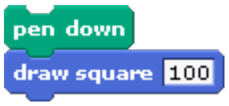
- Now, we have a variable inside our block definition.



- Drag the variable `size` down into the move block. Whenever we need a new copy of a variable, we just grab the copy from that variable in the top row.



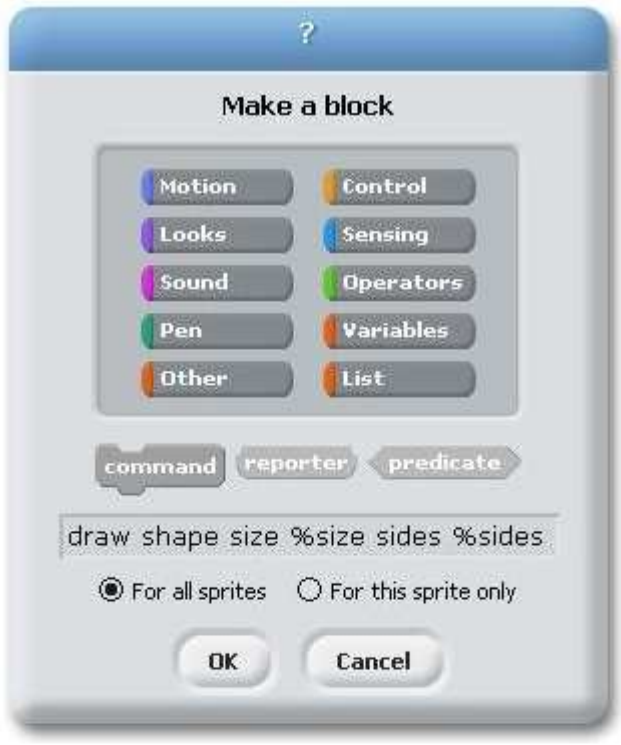
- When we click OK, we wll see that our draw square block now takes an argument. We can put different numbers in the blank and draw squares of different sizes!



Make a draw shape Block

Now, you are going to make a block that takes two inputs. We want to create a `draw shape` block that takes a number of sides and a number of pixels for the length of each side. We will call these input arguments `n` and `pixel`. This exercise should be done with a partner, so introduce yourself to your neighbor and get started on the exercise together!

By the way, you can create the inputs to this block in exactly the same way as we did in the previous section, by clicking on the plus signs to add input; however, you can also type the names of the input as shown below.

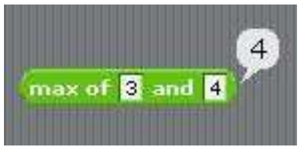


draw shape size %size sides %sides

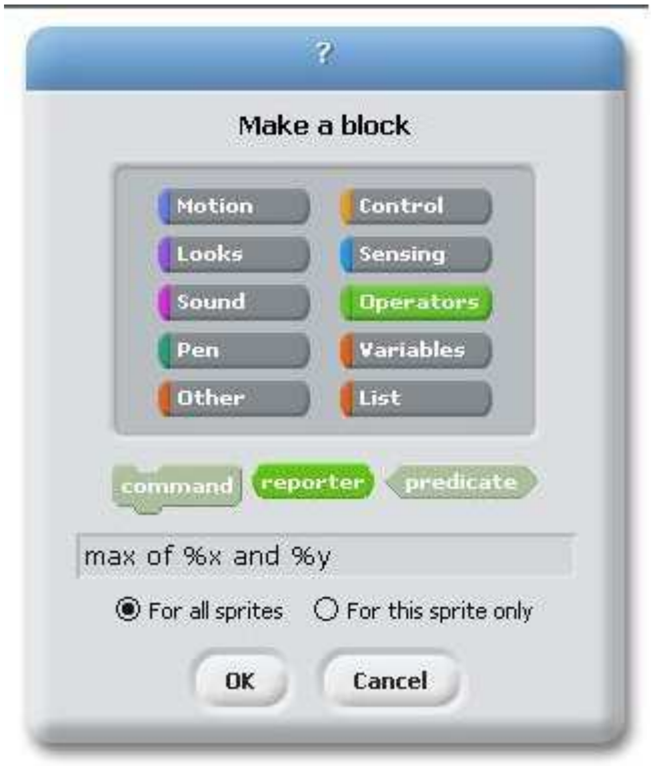
The percent signs (%) indicate that the word should be an input. We want you to feel comfortable with both entry methods.

The Max block

We will now make a different kind of block – a *reporter* block. To demonstrate this, we will make a block called `max` that takes two numbers as input and reports the bigger value (the maximum).

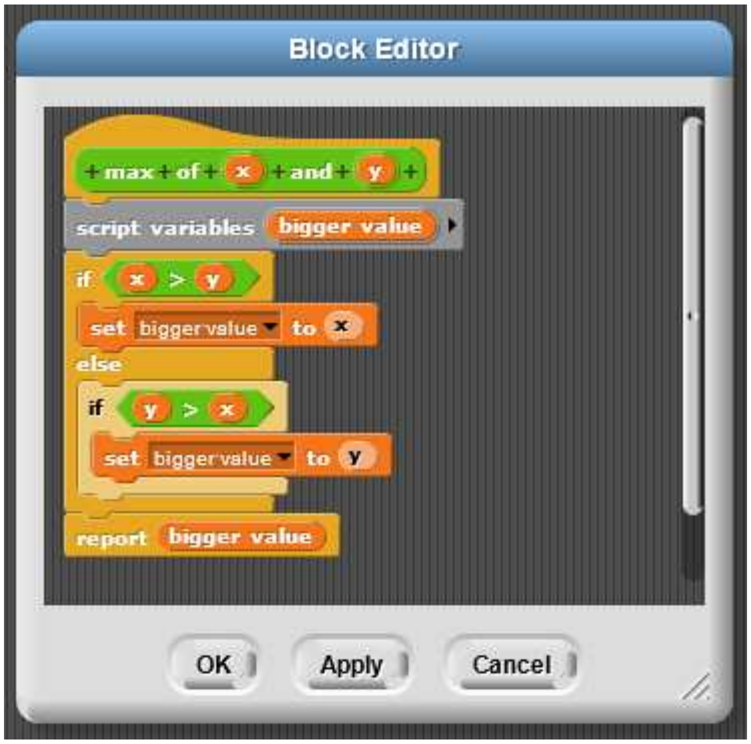


- Click `Make a block` and select the `Operators` tab. We want a reporter block. This will give the block its round shape as shown above. As the name implies, reporter blocks can report a value. In the image below, you can see that we used the % shortcut for making input variables.

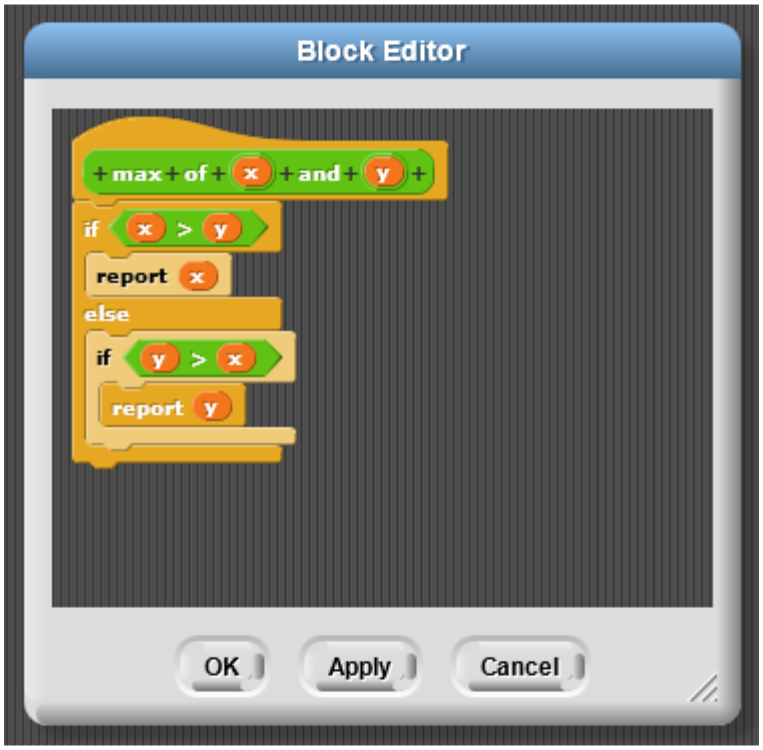


max of %x and %y

- This should give you a blank Block editor. We need to figure out what should be reported. To keep track of the value to be reported, we are going to make another variable. There are two ways to do this: Use a `Script Variable` block. You can click on the name of the variable and change it to bigger value. Alternatively, you can just report which of the two is larger.



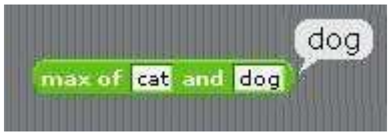
```
+max+of+(x)+and+(y)+
script variables (bigger value)
if <(x) > (y)>
    set biggervalue to (x)
else
    if <(y) > (x)>
        set biggervalue to (y)
report (bigger value)
```


Block Editor

```
+ max + of + (x) + and + (y) +
if <(x) > (y)>
    report (x)
else
    if <(y) > (x)>
        report (y)
```

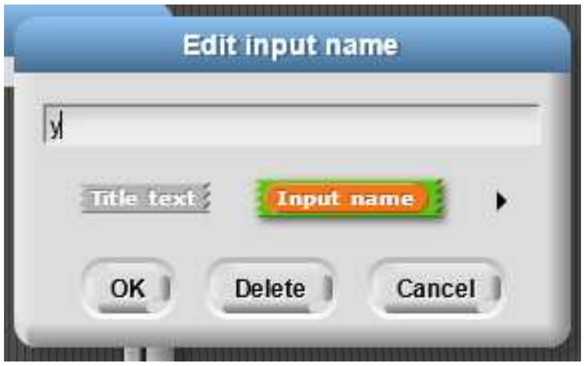
Input Types

Unfortunately, we have a bug with our `max` block! We wanted the `max` block to work only for numbers. Yet, you can type text in!

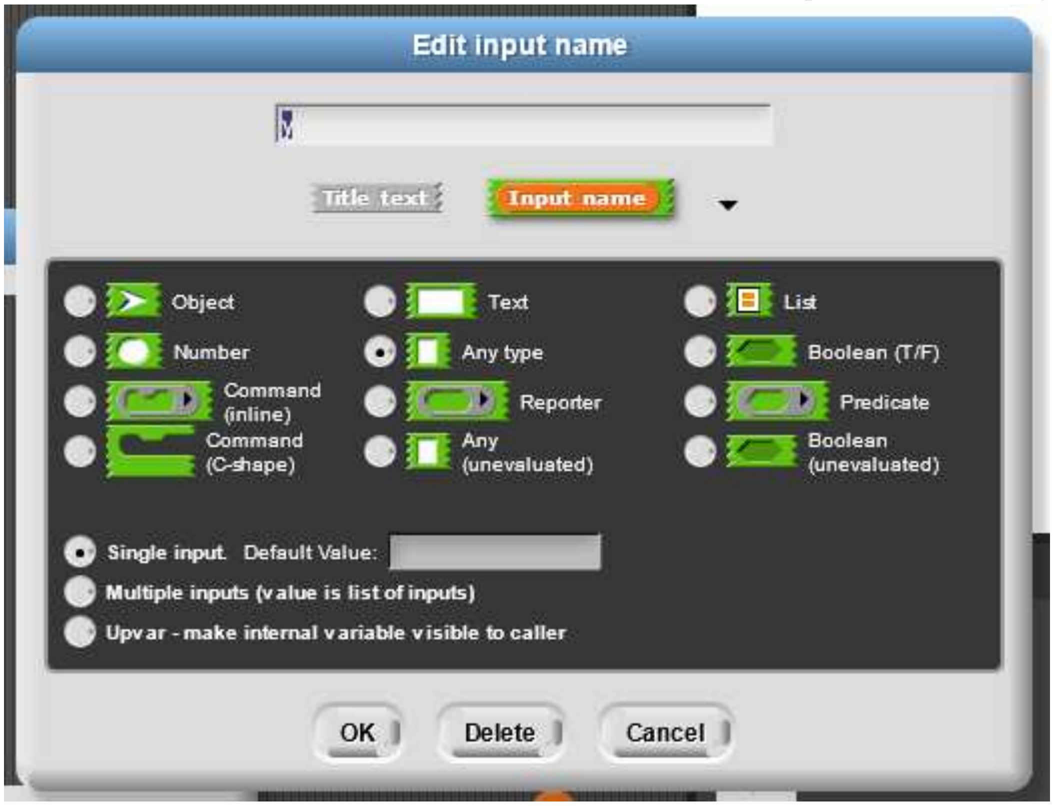


We are going to limit the `max` block to accept only numbers as arguments.

- Open the Block Editor for the `max` block and click on the input `x`

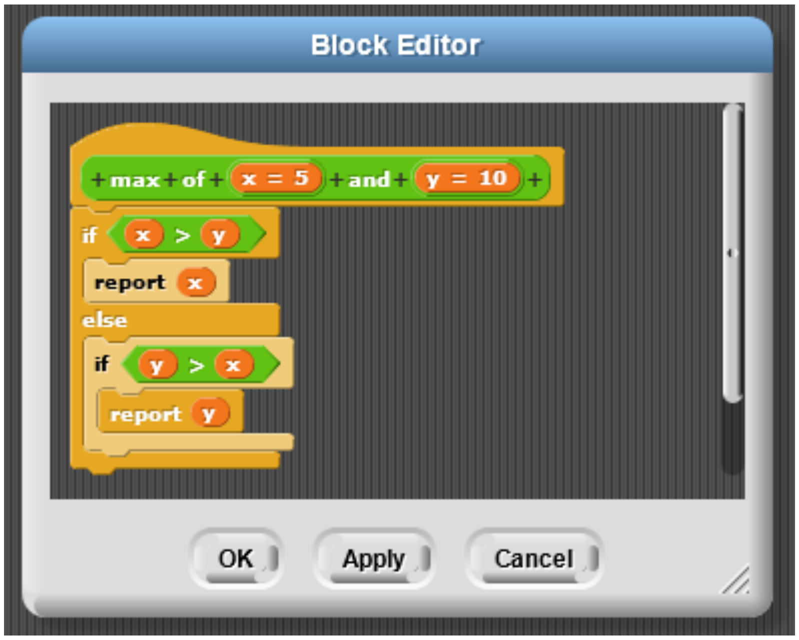


- Then, click on the right arrow in the pop-up box shown above. This will open the dialog box shown below. This allows us to specify the shape of the slot. We want a numbers-only slot (as shown selected below). We can also specify that we want the variable to have a *default* value; this is similar to blocks like move that always start out with the default value 10.



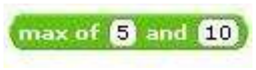
Single input. Default Value:
Multiple inputs (value is list of inputs)
Upvar - make internal variable visible to caller

- Modify both the `x` and `y` variables to take only numbers and to have the default values of `5` and `10`, respectively. Your `Block Editor` should then look like this, with the default values shown in the header.



```
+max+of+ (x = 5) +and+ (y = 10) +
if (x > y)
  report x
else
  if (y > x)
    report y
```

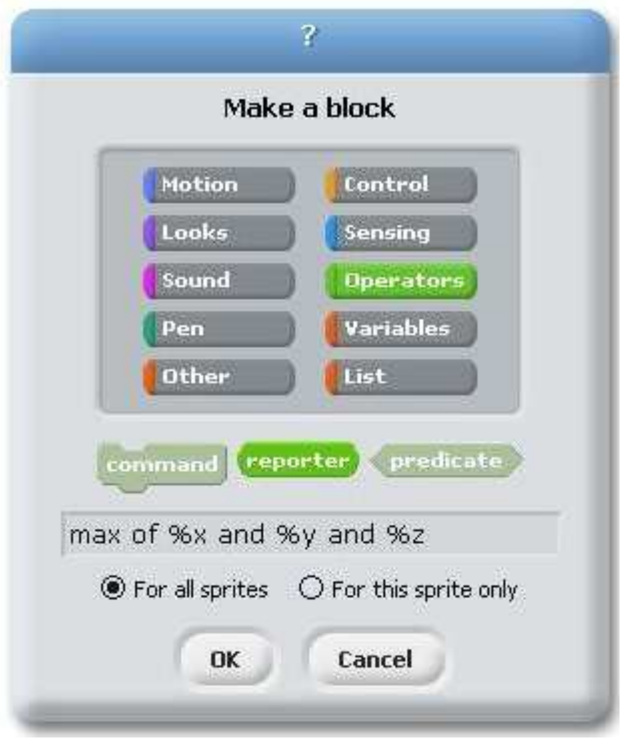
- When you click OK, you should be able to see your block in the Operators tab, with the default values filled in. Also, note that you will no longer be able to enter text.



Note: Maybe we *did* want the `max` block to work with words! However, for the `draw square` and `draw shape` blocks, we definitely only wanted numbers. Modify those blocks to only take in numbers.

Composition of Functions

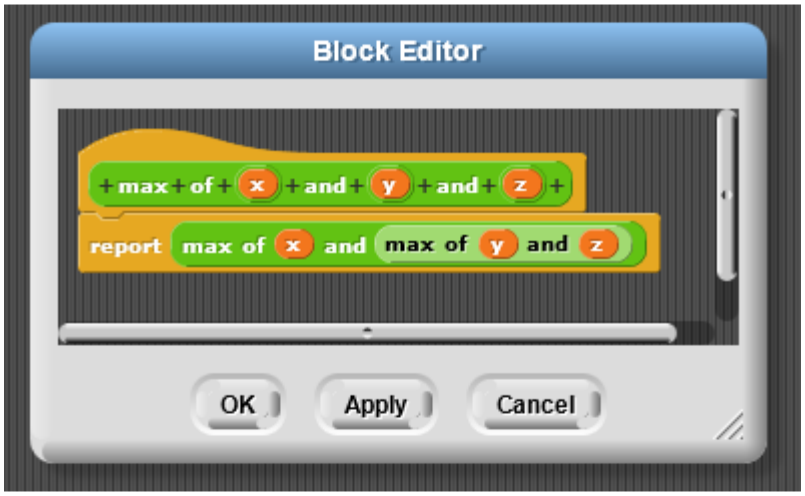
Our custom-made blocks are blocks like any other, and we can use them in other block definitions. To demonstrate this, we are going to make a block that computes the maximum of three values



max of %x and %y and %z

[✓ For all sprites] For this sprite only

Repeat the steps from the last tutorial to make this version of the `max` block also only take numbers. Then, in the script for the block, we can use two copies of our `max` block.



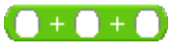
+max+of+ x +and+ y +and+ z +

report max of x and max of y and z

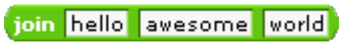
Try It! AddNumbers and JoinText

Your challenge is to create the following two blocks. You can use the same project file.

- A three-argument addition operator that only accepts numbers.



- A three-argument `join` operator that has the default values shown below and only accepts text.

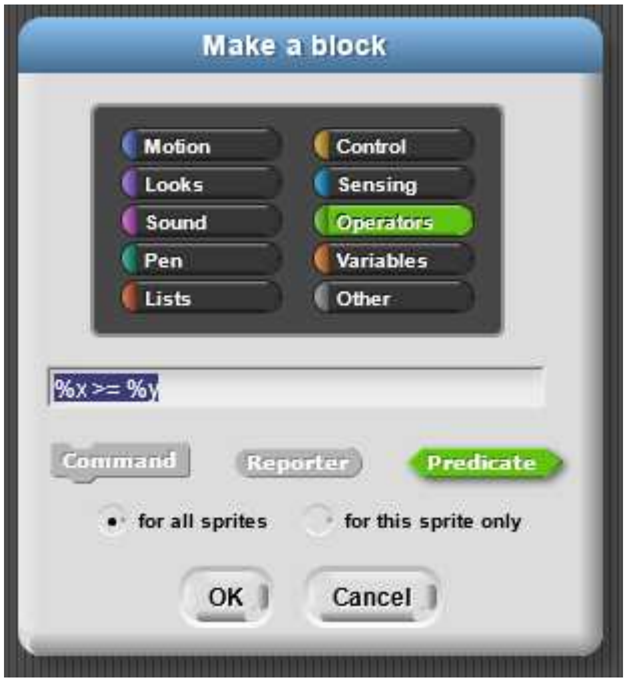


Predicates

We want to make our own predicate, a kind of block that reports either `true` or `false`. We have a “greater than” operator (), an “equal” operator (=), and a “less than” operator (<), but we want a new “greater than or equal to” (=)

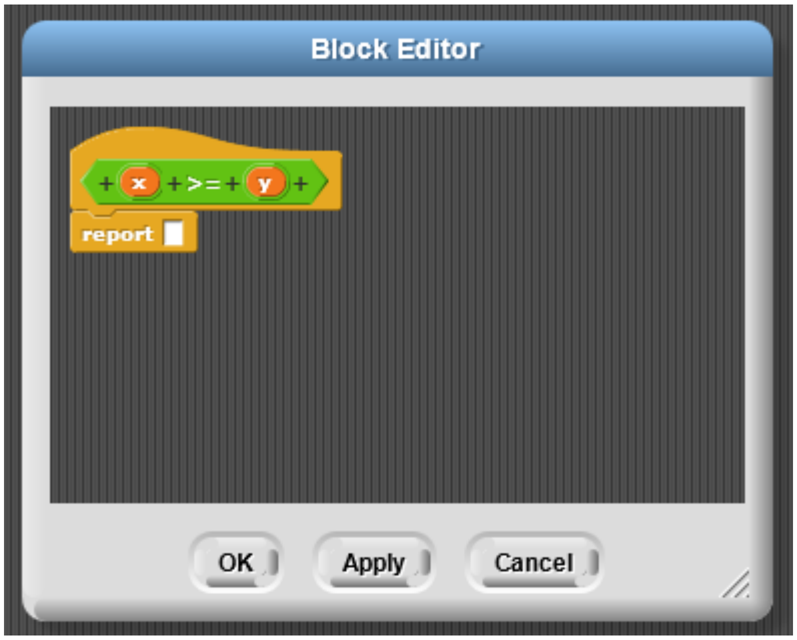
operator.

- We will create the new block and select the `predicate` shape.

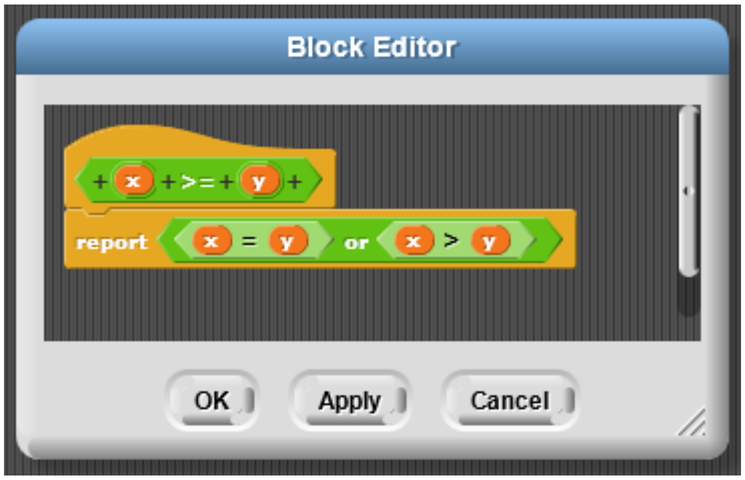


%x >= %y

- This gives a `Block Editor` that has a predicate-shaped blank at the bottom. There, we need to place the predicate that we want to report.

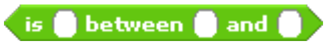


- We can fill that in with a composition of a “greater than”, “or”, and “equal” operators. Make this with your partner and then try it out. Notice that this predicate block reports either `true` or `false`.



Try It! Predicates: Make a between block

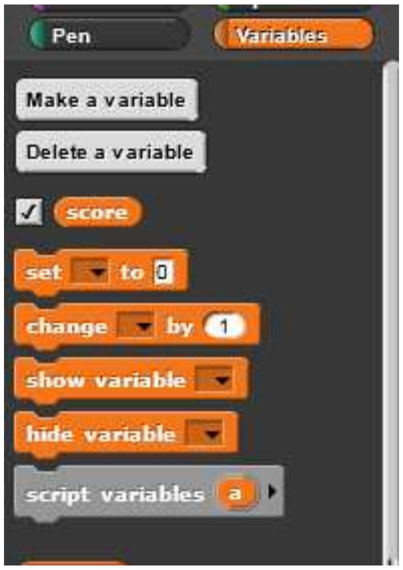
Create a new predicate block that determines if a number is between two other numbers. The block should return `true` if the first number is between the two numbers or if it is equal to either of the numbers.



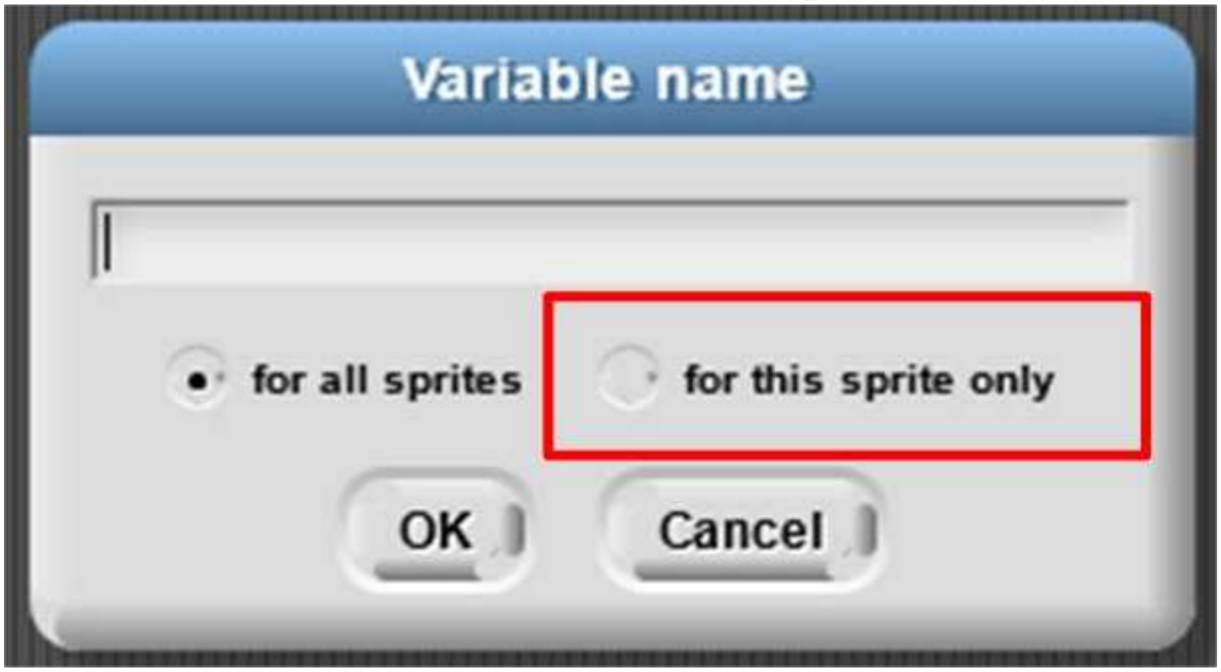
Different Kinds of Variables

We've seen a lot of different types of variables.

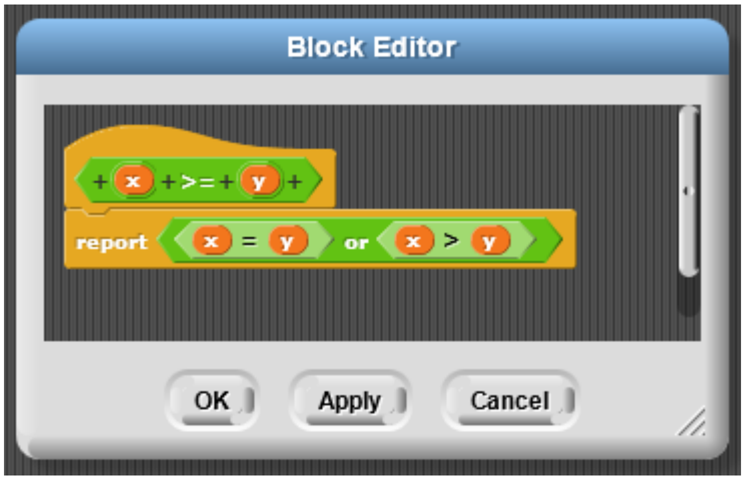
- **Normal/Global variables**: These variables are made in the regular menu and can be used ANYWHERE! The variable "score" below is an example. *These can be used by any sprite, in any block or in any script*.



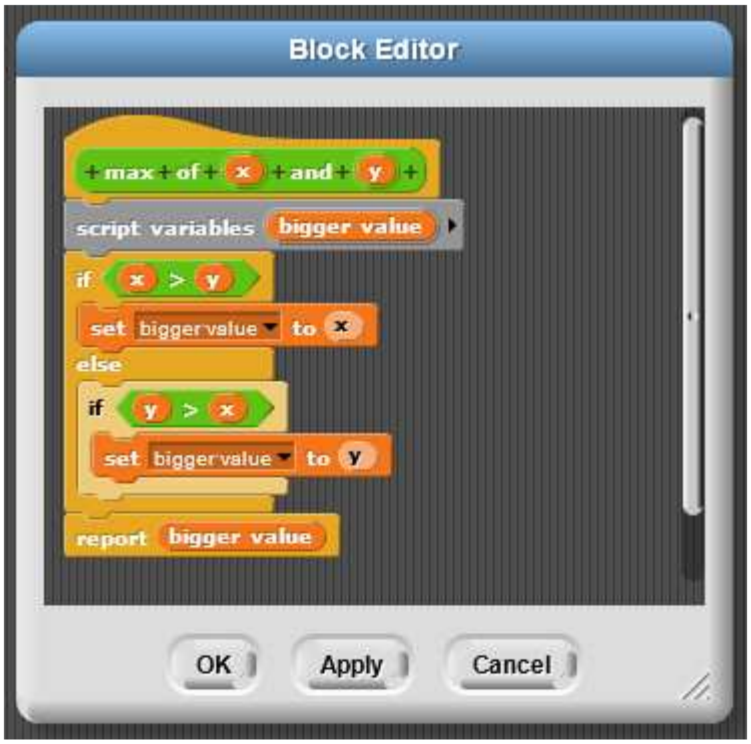
- **Sprite Specific Variables**: When you create a "normal/global" variable you can select that the variable is "For this sprite only". Then these variables will show up as variables listed below the line in the variables tab. *We recommend not using these variables in blocks.*



- **Arguments to a function**: A variable set by the person calling the function. We also refer to this as “input”. *This can ONLY be used within the block editor.*



- **Script Variables**: The “script variable” block gives us a variable that we can use inside of this script. *These can only be used in that particular script. The script could be a block script (shown below) or a regular script.*

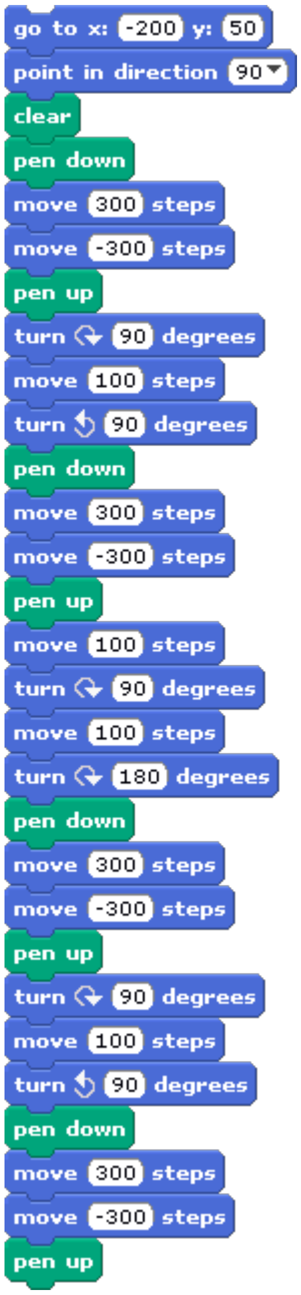


```
+ max + of + (x) + and + (y) +
script variables (bigger value)
if <(x) > (y)>
  set biggervalue to (x)
else
  if <(y) > (x)>
    set biggervalue to (y)
report (bigger value)
```

Try It!: Simplifying a tic-tac-toe board drawer using functions

In the previous lab, we used a `repeat` block to avoid duplicating code. Similarly, we can use a function to avoid duplicating code. Below is some code to draw a tic-tac-toe board. Your goal is to create functions that make this code simpler. One important thing to keep in mind is to give your new functions really intuitive names, so that it is easy to read the code and understand what it does.

Using the blocks below, create two new function blocks, draw line and next line, to draw the tic-tac-toe board. You will still have some of the code from the original script. You can create additional blocks for those functions.

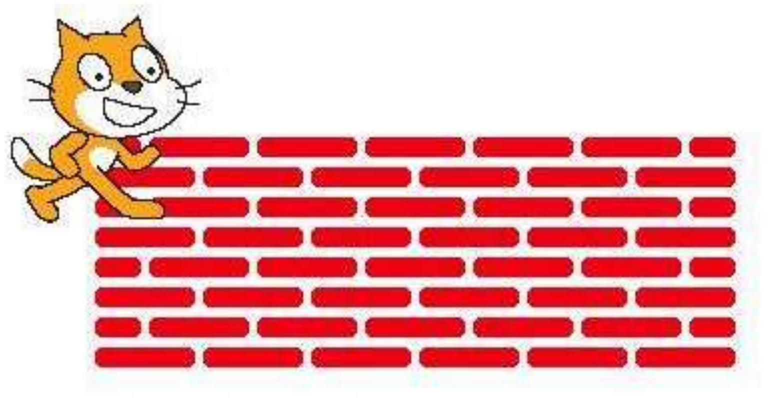


```
go to x: -200 y: 50
point in direction 90
clear
pen down
move 300 steps
move -300 steps
pen up
turn ↻ 90 degrees
move 100 steps
turn ↺ 90 degrees
pen down
move 300 steps
move -300 steps
pen up
move 100 steps
turn ↻ 90 degrees
move 100 steps
turn ↻ 180 degrees
pen down
move 300 steps
move -300 steps
pen up
turn ↻ 90 degrees
move 100 steps
turn ↺ 90 degrees
pen down
move 300 steps
move -300 steps
pen up
```

Project: Brick Wall

Let's open up SNAP at http://snap.berkeley.edu/run (http://snap.berkeley.edu/run)

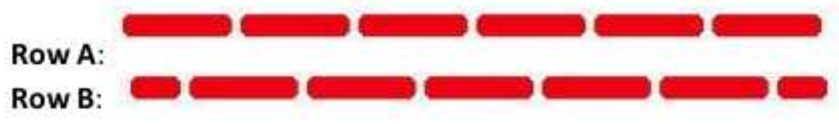
Sometimes, when we write programs and scripts, it feels like we have hit a brick wall! (This is a good sign - it is supposed to be hard!) We are going to draw this brick wall.



This project is not just about drawing; it is also about practicing abstraction. You will draw the following brick wall by implementing the blocks listed below.

Note: You must implement these blocks and adhere to the abstraction described below.

There are two kinds of rows in this brick wall:



The big idea is that there are three levels of abstraction. Start creating your new blocks at Level 3, then create for Level 2 (using your new blocks from Level 1), and then create Level 1 using your Level 2 blocks.

Note: a "brick" is just a thick line.

At the lowest level of abstraction (Level 3):

- You need to figure out how to draw individual bricks, small bricks and spaces. The bricks are simply thick lines.
- This level of abstraction contains the following blocks:
- The `Draw Brick` block, which draws a single brick.
- The `Draw Small Brick` block, which draws the small brick for the edges of row B. Note that this brick will not be exactly half as long as the full brick. Part of this assignment is figuring out how long the "small brick" should be.
- The `Draw Space` block, which draws a space between each brick or small brick.

At the middle level of abstraction (Level 2):

- You can use the functionality provided by the bottom level of abstraction to make entire rows of bricks.
- The rows referred to as "Row A" and "Row B" should look like the rows shown above.

- This level of abstraction contains the following blocks:
- The `Initialize Pen` block, which should initialize the pen color and size.
- The `Initialize Character Position` and `Direction` block, which should initialize the position and direction of the character.
- The `Draw Row A` block, which should draw a single copy of Row A.
- The `Draw Row B` block, which should draw a single copy of Row B.
- The `Transition between Row A and B with __ space` block, which should transition between the end of Row A and the beginning of Row B, leaving a space as wide as the number of pixels specified by the input argument.
- The `Transition between Row B and A with __ space` block, which should transition between the end of Row B and the beginning of Row A, leaving a space as wide as the number of pixels specified by the input argument.

HINT: You will need to determine if there is an even number of rows or an odd number of rows to be drawn as well as if the row being drawn is an even number or odd number one. You can use the `() mod ()` block to help.

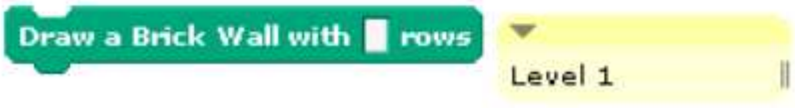
Example: if TotalNumberOfRows mod 2 = 0 then the total number of rows is even (any even number divided by 2 leaves a remainder of 0). You can use the same logic to determine if the row to be drawn is even or odd (if RowNumber mod 2 = 0).

At the highest level of abstraction (level 1)

- You will put together the full brick wall using only the functionality provided by the middle level of abstraction.
- This level of abstraction contains only the `Draw a Brick Wall with __ rows` block, which draws a brick wall with the specified number of rows.

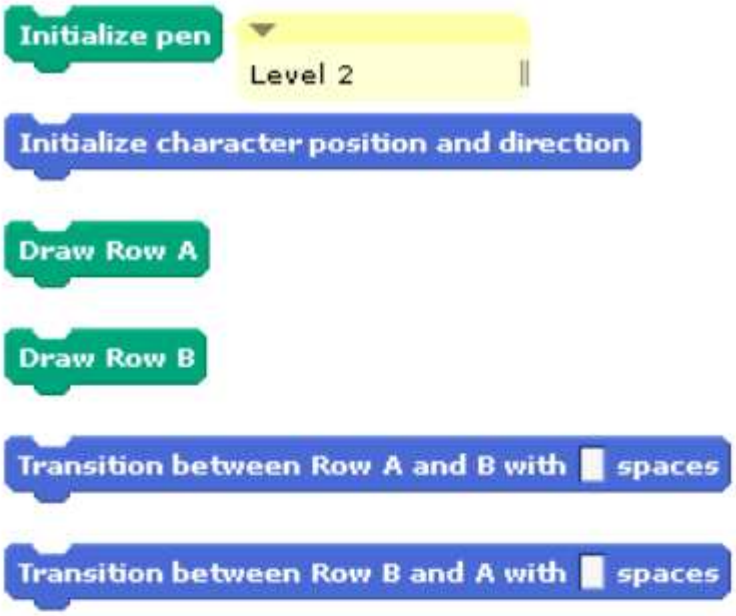
Note: Whenever you need to refer to a number in the program, use a variable. This is generally considered good style, because you can use the same variable in multiple places in your program, and you only need to change the value of the variable to change it in multiple places at once.

In summary, you should implement the following blocks:



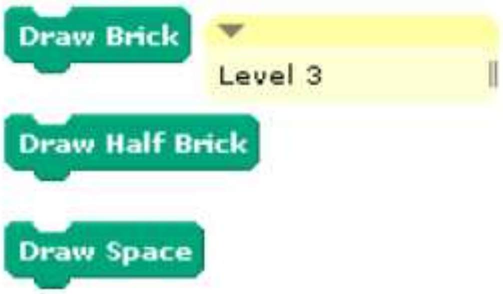
```
Draw a Brick Wall with [ ] rows
```

Level 1



```
Initialize pen
Initialize character position and direction
Draw Row A
Draw Row B
Transition between Row A and B with [ ] spaces
Transition between Row B and A with [ ] spaces
```

Level 2



```
Draw Brick
Draw Half Brick
Draw Space
```

Level 3