Queen Mary
University of London

Box Variables

Age group:	10 – adult
Abilities assumed:	Very simple programming, limited exposure to assignment and variables
Time:	15-20 minutes, or a full lesson with extension exercises
Size of group:	3 upwards

Focus

Variables
Assignment
Sequencing
Programming

Syllabus Links

This activity is appropriate for any syllabus aim about learning to program that requires an understanding of variables and assignment.

Summary

You dry run simple programs that involve variables and assignment by running them on a computer made of students. Students with boxes act as variables as values are copied between them following the instructions of the program. You physically demonstrate the creation of variables, how accessing a variable involves taking a copy of its value, and how storing values in a variable destroys any previous value stored.

Technical Terms

Assignment, Variable, Value, Declaration, Initialisation, Sequencing.

Materials

- 3 A4 Boxes
    - (e.g. lids of photocopy paper boxes)
- Variable Name Labels
    - Print off the sheets, laminate hole punch, tie with string or ribbon
- Value cards
    - Print on to paper the colour of the text.
- The program
    - Program slides for relevant language, write programs up on board, or give out as a handout

What to do

The Grab:
Explain that understanding variables and assignment is critical to being able to program but is an easy thing to be confused about. Get over this hurdle and programming will be far easier.

A good way to think of variables is that they are like special boxes with their own private shredder and photocopier built in - boxes that can store but also create and destroy.

The activity:
Display the fist program. It swaps the values stored in the variables. Explain that you are going to demonstrate how the program works, and in particular what and how variables and assignment work.

It is helpful if the class to have seen a simple use of variables prior to this activity – e.g., just to store a value input that is immediately printed. The following is a general introduction if not.

One of the most important things that computers do is process data. They store values – numbers, text, images, sounds – manipulate them, do calculations with them, make changes to them and store the results until they are needed. For example, music editing software might be used to record samples of sounds, edit them, combine them with others, store the results and then play the final version. A texting or tweeting program stores the words you type, allowing you to edit them until you are ready to send the message.

Programming languages therefore need a way for programmers to write commands that tell the computer to store data and to move it around. The first thing needed are a way to refer to places that store data. Programmers call these **variables**. As a program is likely to have lots of variables you need a way of knowing which one is which so variables are given **names.** The names are used to indicate the particular one the program should use at any point.

Before variables can be used you need to create them. In some languages (eg Python) they are created automatically the first time they are mentioned. In others like Scratch you have to tell the system to create them. In yet others (like Java) you have to explicitly create them using special instructions in the language itself. These instructions are called declarations. *[Explain just the situation in the language you are teaching. We will use Python syntax in the following explanation: substitute the appropriate language].*

When you create a variable, you are first of all picking a place in memory to store data. Point to the first instruction in the program.

```
colour1 = “red”
```

This creates a new variable called colour1 and stores the string “red” in it. Emphasise that it is best to pronounce the equals symbol as ‘gets the value” as in ‘colour1 gets the value “red”’.

Let’s demonstrate what that means in practice…

Pick a student from the audience and give them a box to hold. Point out that you have just created a new variable – a storage space. A variable is just like a box that can hold a single thing. As your program will be creating more variables, you need a way to tell which one is which. You need to give them a name. Take the name label for **colour1** and get them to put it round their neck. Point out that any previous name (eg the one their mother gave them!) is now forgotten. From now on they will be called **colour1**. Explain that in writing the program you could have made up any name for them as long as you use that name consistently through the program to refer to that variable, that storage box.

The rest of the command is an **assignment**. It just stores a value in a variable. It puts something in the box. First (on the left hand side of an = sign) you give the name of the variable i.e., the place to store the value. On the right hand side you give the value to be stored there. Here the value to be stored is the string "red".

```
colour1 = "red"
```

So this assignment puts value "red" into the variable called colour1. Get one of the value cards with "red" written on it and put it in the box called colour1 following the instruction.

Move to the next command:

```
colour2 = "green"
```

This is very similar. As one doesn't already exist, it creates a new variable called colour2. It then stores value "green" in it. Get another volunteer, give them a box and name label, and put a value card with "green" on it in their box.

Storing something in a variable (putting something in a box) for the first time is called **initialising** the variable. In the above the assignments **initialised** colour1 with the value "red" and colour2 with the value "green".

Emphasise the difference between **values** and **names of variables**. Values are the things that can be stored in boxes – the cards. Names are just labels (hung round the necks of volunteers). In the programming language string values have quotes round them so you know they are values not names of variables. If there are no quotes then it will either be a keyword or the name of a variable.

We have initialised our two variables with starting values. Now we can do something with the data we have stored there. In our simple little program, we will just swap the two values over, so that at the end, colour1 holds "green" and colour2 holds "red".

The next instruction is

```
temp = colour1
```

We haven't seen temp before so we need to declare the variable. Pull out a third volunteer and give them a box and the temp label. This assignment is a little different to the previous two though. The right hand side is the name of a variable (no quotes round it) not a value. That means first get a ***copy*** of what is in that variable, i.e., what is in colour1, and put the copy in the variable called temp.

Go to the person representing colour1, making a show of checking their name. Ask them what is in their box. Create a copy of it, either by getting a blank card and writing the value out or getting a ready-made card. Put that copy in to the box labelled temp. Emphasise that colour1 has not changed at all – it still has its original value which was not touched. The variable temp now just holds a copy of it. Emphasise also that the assignment was 'temp gets a copy of the value in colour1' not the other way round i.e., the movement is in the direction

temp ⟵ colour1

The next assignment is similar.

colour1 = colour2

We have finished creating variables though – the variables referred to here are both ones we previously created. This assignment just makes a copy of the value in colour2 and places it in colour1.

Go to the person acting as colour2 and find out what value they are storing. Make a copy. Again something new is happening. We are not just doing an initialisation. We are storing a value in a variable that already has a value. It is important to emphasise that a variable can only store one value. When you put a new value in a box, you destroy the old one. As you put the new value in colour1, make a show of taking out the old one and destroying it – screw it up, tear it up, eat it, jump up and down on it – something that theatrically destroys it. Point out that that value is now gone forever and the new value stored is the one just copied from the other variable.

The final assignment is similar, this time copying the value from temp into colour2 whose value is destroyed: colour2 gets the value in temp.

colour2 = temp

Act this out in a similar way to the previous one. Point out that the value in temp was the one saved from colour1 in the earlier step. That means even though the value in colour1 was destroyed, because a copy was stored safely first, we end up with the same value in colour2.

Finally look at the results. What has happened? The values in colour1 and colour2 have been swapped over – using temp as extra storage space. Originally colour1 was "red" and colour2 "green. Now colour1 is "green" and colour2 "red".

Finish by re-emphasising that variables are like special storage boxes and in particular:

- variables are given names so they can be referred to again,
- variables hold values: the actual data that is being stored,
- it is important not to confuse the names of variables with their values,
- a variable can only store one value at a time,
- when you get a value from a variable you are making a copy, that variable's value is untouched, and
- when you store a new value in a variable you destroy anything that was previously there.

Refer back to the boxes as you cover these points. Give the class a chance to ask questions about what happened while the volunteers are still out the front, and use the props to explain any misunderstanding.

Variations and Extensions

Faulty swap

Repeat the dry run process using a faulty swap program (see below) that doesn't use an extra temporary variable.

```
colour1 = "red"
colour2 = "green"
colour1 = colour2
colour2 = colour1
```

See if the class can predict in advance what will happen. Get them to call out what should be done at each step and at the end get them to try to explain what has happened and why.

Assignment Dry Run

We strongly recommend doing the Assignment Dry run exercises soon after this demonstration to reinforce the understanding and also quickly correct any mistaken mental models students might hold.

Other programs

Demonstrate what is happening using boxes as variables in a series of other short programs including answers to simple programming exercises the students have attempted.

Types

Demonstrate types in a similar way, using, for example, small boxes for integers and bigger boxes for strings. This is best kept as a separate follow-on activity rather than explaining it at the same time as variables and assignment. The key thing students struggle with a correct mental model of, and need to understand first, is the way variables and assignment work in general.

Arrays

Arrays can be demonstrated as boxes with compartments with the whole box named and each compartment numbered. Staple a series of boxes together to make one.

Further Reading

Computing without computers

A free booklet by Paul Curzon on programming, data structures and algorithms explained using links to everyday concepts. Available from http://teachinglondoncomputing.org/resources/

Links to other activities

Assignment Dry Run

Dry run on paper a series of short fragments of code involving assignment.
This is an important activity to do after explaining variables and assignment. It reinforces understanding and helps identify faulty mental models so they can be fixed. It is a pencil and paper exercise executing code. Being able to do this kind of dry run for any new construct is an important prerequisite to being able to actually write code.

The swap puzzle

Solve a puzzle, coming up with an algorithm that your team can follow faster than anyone else.
This gives a way to introduce the idea of the solution to a problem being a set of instructions that allow others to 'solve' it with no understanding. It also explores how different algorithms can solve the same problem but may not be equally good – some may be faster.

The intelligent piece of paper

Take part in a test of intelligence against an intelligent piece of paper!
This is a good introduction to what an algorithms is and how a computer program is just an algorithm. It can also be used to start a discussion on what it would mean for a computer to be intelligent. It can lead on to an unplugged programming activity creating winning instructions.

The Invisible Palming Trick

Teach a trick where the magician invisibly moves a card between 2 piles.
This is a fun way to introduce the idea of an algorithm, showing how algorithms are a series of steps that if followed precisely lead to something (in this case magical) being guaranteed to happen – even if the person (or computer) following the algorithm doesn't know what they are doing.

Live demonstration of this activity

Teaching London Computing give live sessions for teachers demonstrating this and our other activities. See http://teachinglondoncomputing.org/ for details. Videos of some activities are also available or in preparation.

Name card

colour1

Name card

colour2

Name card

temp

Value card

“Red”

Value card

“Red”

Value card

“Red”

Value card

“Red”

Queen Mary
University of London

Value card

“Green”

Teaching London Computing / cs4fn: Box Variables
www.teachinglondoncomputing.org
www.cs4fn.org

Value card

“Green”

Value card

“Green”

Value card

“Green”

Scratch:
What does this program fragment do?

```
SET colour1 TO “red”
SET colour2 TO “green”

SET temp TO colour1
SET colour1 TO colour2
SET colour2 TO temp
```

Scratch:
What does this program fragment do?

```
SET colour1 TO “red”
SET colour2 TO “green”

SET colour1 TO colour2
SET colour2 TO colour1
```

Python:
What does this program fragment do?

```
colour1 = “red”
colour2 = “green”

temp = colour1
colour1 = colour2
colour2 = temp
```

Python:
What does this program fragment do?

```
colour1 = “red”
colour2 = “green”

colour1 = colour2
colour2 = colour1
```

Java:
What does this program fragment do?

```
String colour1 = “red”;
String colour2 = “green”;

temp = colour1;
colour1 = colour2;
colour2 = temp;
```

Java:
What does this program fragment do?

```
String colour1 = “red”;
String colour2 = “green”;

colour1 = colour2;
colour2 = colour1;
```