Misconceptions and confusions—Supplemental Course Notes for CSI310, UAlbany, Spring 2011.

(By Prof. S. Chaiken Copyright CC-BY-SA 2011)

These are known to get in the way of the beginner being able to follow explanations and exercise instructions. As a result, they interfere with further learning. Often, when the confused beginner starts writing code based on the misconception, finishing it becomes impossible or becomes an act of desperation so wild guesses get tried or written in exam papers. My examples point out the first appearance of the misconception. The example omits typical wild guesses which follow.

Storing a calculated value versus printing a calculated value

Example: A calculate and store in M[i] the sum of the three elements in array A at the index i and the two preceding indices i-1 and i-2.


The lesson must be learned that data calculated by the computer can be (and most frequently is) stored inside the computer, inside the computer memory, in what are called variables. One of the first things taught is how to get the computer to print or somehow otherwise display something. But just beyond that, programmers must become fluent in making the computer store calculated values inside itself for use in additional calculations to be done in the near future.

Inputting from people versus reading from variables, including array elements

Example: Write code that selects the smallest integer stored in array A by means of the following algorithm. Initialize variable X to A[0], then run a loop that scans the rest of A. Whenever a value smaller than (the value of) X is read from the array, the algorithm replaces the old value of X with the value read.

Correct:

X = A[0];
for( int i = 1; i < A.length; i++ )
{
    if(A[i] < X)
    {
        X = A[i];
    }
}
Incorrect: The beginner confused the class named "Scanner" by the Java utilities class developers with the concept of scanning a series of things in an algorithm.

\[
X = A[0];
\]
Scanner scan = new Scanner(System.in);
for( int i = 1; i < A.length; i++)
{
    int Temp = sc.nextInt();
    if(Temp < X)
    {
        X = Temp;
    }
}
//A is not used except for A[0] and A.length

If you name an integer variable \(i\) and/or use it as a for-loop variable, it will index an array.

Example: Find the sum of the numbers in the array \(A\).

Correct:

\[
\text{double sum} = 0.0
\]
\[
\text{double A[]} = \{1.0, 2.0, 3.0, 7.0, 98.0\};
\]
for( int i = 0; i < A.length; i++)
{
    sum = sum + A[i];
}

Incorrect:

\[
\text{double sum} = 0.0
\]
\[
\text{double A[]} = \{1.0, 2.0, 3.0, 7.0, 98.0\};
\]
for( int i = 0; i < A.length; i++)
{
    sum = sum + A;
}

Confusing an array element's index with an array element's value

Example: Write a loop that computes the sum of the elements in array \(A\) of doubles.

Correct:

\[
\text{double sum} = 0.0;
\]
for( int i = 0; i < A.length; i++)
    sum = sum + A[i];

Incorrect:

\[
\text{double sum} = 0.0
\]
for( int i = 0; i < A.length; i++)
    sum = sum + i;
Another example: Find the index of the largest number in the array.

Correct:

double A[] = {3.0, 9.0, 10.0, 11.0, 2.0, 6.5};
int whereMaxIs = 0;
for( i = 1; i < A.length; i++ )
{
  if(A[whereMaxIs] > A[i])
  {
    whereMaxIs = i;
  }
}

Correct but longer than necessary:

double A[] = {3.0, 9.0, 10.0, 11.0, 2.0, 6.5};
int whereMaxIs = 0;
double maxFoundSoFar = A[whereMaxIs];
for( i = 1; i < A.length; i++ )
{
  if(maxFoundSoFar > A[i])
  {
    whereMaxIs = i;
    maxFoundSoFar = A[i];
  }
}

Incorrect:

double A[] = {3.0, 9.0, 10.0, 11.0, 2.0, 6.5};
double whereMaxIs = 0;
for( i = 1; i < A.length; i++ )
{
  if(whereMaxIs < A[i])
  {
    whereMaxIs = A[i];
  }
}

When A[i]<A[i+1] is found to be true the last time when a loop processes the array, A[i+1] is the maximum entry in the whole array.

This is a misconception. The following program does NOT end up by finding the maximum entry which is 11.0, not 6.5.

Incorrect:

double A[] = {3.0, 9.0, 10.0, 11.0, 2.0, 6.5};
int whereMaxIs = 0;
for( i = 0; i < A.length-1; i++ )
{
  if(A[i] < A[i+1])
  {

```
Yes, there was a time when whereMaxIs was set to 3, where A[3] == 11.0 is the maximum. But later on, when i was 4, the computer compared A[4] == 2.0 to A[5] == 6.5 and changed whereMaxIs to 5, which is wrong. What counts is whether the answer is correct when the computation is finished, not whether it had been correct sometime in the duration.

A single change fixes this example:

Correct:

```java
double A[] = {3.0, 9.0, 10.0, 11.0, 2.0, 6.5};
int whereMaxIs = 0;
for( i = 0; i < A.length-1; i++ )
{
    if(A[whereMaxIs] < A[i+1])
    {
        whereMaxIs = i+1;
    }
}
```

Modifying the value of a variable requires some kind of put or modify method or operation, not just a simple assignment statement.

This is a misconception.

Example: See the example to find the smallest integer in array A. Each time a new smaller element is found, modify the count stored in minCount by adding 1 to it.

Correct:

```java
X = A[0];
int minCount = 1;
for( int i = 1; i < A.length; i++ )
{
    if(A[i] < X)
    {
        X = A[i];
        minCount = minCount + 1;
    }
}
```

Incorrect:

```java
X = A[0];
int minCount(1);  //This is not Java.
for( int i = 1; i < A.length; i++ )
{
    if(A[i] < X)
    {
        X = A[i];
        minCount.modify(1);
    }
}
This misconception is apparently caused by trying to teach people about accessors and mutators before they really understand variables! Even though class definitions often define objects with fields that are manipulated through accessor or mutator methods only, local variables are important to use within methods. Furthermore, the assignment operation must be used to implement a mutator. In other words, the body of a mutator method contains an assignment operation. So you have to really know that the basic way to change the value of a variable is to use an assignment operation.

Example:

class Color
{
    private final int MAXVAL = 255;
    private int redVal, greenVal, blueVal;
    //The mutator methods ensure that these color intensities are valid.
    //Alternative class designs would be to report an error by throwing
    //an exception, or by refusing to change the value, if the new value
    //would have been invalid.
    public void setRed(int x)
    {
        redVal = x;   //Assignment operator used!
        if(redVal > MAXVAL) redVal = MAXVAL; //Assignment operator used again!
        if(redVal < 0) redVal = 0;  //Assignment operator used again!
    }
    public void addToRed(int x)
    {
        redVal = redVal + x; //Assignment operator used!
        if(redVal > MAXVAL) redVal = MAXVAL; //Assignment operator used again!
        if(redVal < 0) redVal = 0;  //Assignment operator used again!
    }
    // ... there would be more
}

Exercise:

(1) Refactor the above code so the checking and correcting operations are coded in only one place.

(2) Implement the design alternative that the intensity is not changed if the new value would have been invalid.

(3) Implement the design alternative that an exception is thrown if the new value would have been invalid.

A test or homework problem to implement some task can be correctly solved by writing code that calls some method with the task's name.

Example: Show how to code a loop that computes the sum of array of doubles A and then print the result.

Correct:

double sum = 0.0
double A[] = {1.0, 2.0, 3.0, 7.0, 98.0};
for(int i = 0; i < A.length; i++)
{
    sum = sum + A[i];
}
Incorrect:

double A[] = {1.0, 2.0, 3.0, 7.0, 98.0};
System.out.println(A.sum()); // Ha! I got away without writing the loop!

In some languages, arrays will come with methods for sums, etc. But that is not Java. Also, knowing how to code loops to implement such things is important and is often required in tests, homework, and professional practice.

Print a result means print stuff from each intermediate step.

Example: Show how to code a loop that computes the sum of array of doubles A and then print the result.

Correct: See above.

Incorrect:

double sum = 0.0
double A[] = {1.0, 2.0, 3.0, 7.0, 98.0};
for(int i = 0; i < A.length; i++)
{
    sum = sum + A[i];
    System.out.println(sum);
}
System.out.println(sum);

What if A.length == 1000000?

Confusing a method that prints something with a method that returns something

Example: Write a (static) method that given double (double precision floating point number), prints its square root.

Only correct solutions will be given. It is incorrect to mix them up.

Correct:

static void printSquareRoot(double x)
{
    System.out.println(Math.sqrt(x));
}

Example: Write a (static) method that given a double (double precision floating point number), returns its square root.

Correct:

static double returnSquareRoot(double x)
```java
{ 
    return Math.sqrt(x);
}

Why it matters: Suppose the task is to use one of the above solutions to help write code for a method that prints the 1.0 plus the square root of x.

Incorrect:

```java
static void printWhatsWanted(double x) 
{ 
    println(1.0 + printSquareRoot(x));
}
```