Practice with variables and types

1. Types. For each literal or expression, state its type (`String`, `int`, `double`, or `boolean`).

Expression	Type	Expression	Type
`387`	`int`	`"pancakes"`	`String`
`true`	`boolean`	`45.0`	`double`
`"14"`	`String`	`87.98515`	`double`
`"false"`	`String`	`15 >= 71`	`boolean`
`31.6 + 7`	`double`	`(double)(int)93.2`	`double`

2. Declaring and using variables. Only one of the following code snippets is valid (i.e., will compile without errors). Which is it, and what's wrong with each of the others?

Code snippet A:

```
int x = 3;
int y = 17;
int x = x + y;
```

Variable `x` is declared twice.

Code snippet B:

```
int num = 42;
double anotherNum = 81;
num = anotherNum - num;
```

`anotherNum - num` has type `double`, which cannot be stored in `num`, an `int`

Code snippet C:

```
int years = 18;
int months = 7;
double totalAge = years + months/12.0;
```

This one works!

Code snippet D:

```
int p = 5;
int q = 43.7;
p = q;
```

Can't assign a `double` (`q`) to an `int` (`p`).

3. Casting. For each of the following, add a cast to fix the type error.

```
int i = 5;
double j = 21.3;
i = i + (int)j;

int totalLabScore = 84;
int numLabs = 10;
double averageScore = (double)totalLabScore/numLabs;
```

Note: this doesn't cause a compiler error, but it didn't achieve the intended behavior of computing the average score, which should include the decimal places.

4. Using variables. Write a piece of code that asks the user to enter their height (as a number of feet and a number of inches, i.e., 5 7) and tells them their height in meters (i.e., 1.7018). (Note: there are 12 inches in a foot, and there are 3.28 feet in a meter.)

```
System.out.println("Enter the number of feet");
int feet = keyboard.nextInt();
System.out.println("Enter the number of inches");
int inches = keyboard.nextInt();
System.out.println("Your height is: " + (feet + inches/12.0)/3.28);
```

Practice with `if` statements

1. Are they equivalent? Which of the following snippets of code do the same thing? That is, which print the same message(s) on every single input value for `num`?

Code snippet A:

```
int num = keyboard.nextInt();
if(num > 54) {
   if(num > 82) {
      System.out.println("one");
   }
   else {
      System.out.println("two");
   }
}
else {
   System.out.println("three");
}
```

When `num` > 82, prints "one"
When 82 ≥ `num` > 54, prints "two"
When num ≤ 54, prints "three"

Code snippet B:

```
int num = keyboard.nextInt();
if(num > 82) {
   System.out.println("one");
}
else if(num > 54) {
   System.out.println("two");
}
else {
   System.out.println("three");
}
```

When `num` > 82, prints "one"
When 82 ≥ `num` > 54, prints "two"
When num ≤ 54, prints "three"

Code snippet C:

```
int num = keyboard.nextInt();
if(num < 54) {
   System.out.println("three");
}
if(num > 82) {
   System.out.println("one");
}
else {
   System.out.println("two");
}
```

When `num` > 82, prints "one"
When 82 ≥ `num` ≥ 54, prints "two"
When num < 54, prints "three" and "two"

A and B do the same thing.

Code snippet D:

```
int num = keyboard.nextInt();
if(num > 54) {
   if(num < 82) {
      System.out.println("two");
   }
}
else if(num > 82) {
   System.out.println("one");
}
else {
   System.out.println("three");
}
```

When `num` > 82, prints nothing
When 82 > `num` > 54, prints "two"
When num ≤ 54, prints "three"

2. Old enough? Write some code that asks the user for their age and then prints out whether they are old enough to:

1. Vote (age 18)
2. Get a driver's license in MA (age 16)
3. Rent a car (age 25)
4. Drink legally (age 21)

```
System.out.println("How old are you?");
int age = keyboard.nextInt();

if(age >= 16) {
   System.out.println("You can drive");
   if(age >= 18) {
      System.out.println("You can vote");
      if(age >= 21) {
         System.out.println("You can drink");
         if(age >= 25) {
            System.out.println("You can rent a car");
         }
      }
   }
}
```

Other solutions are also possible.

3. Scope. Determine whether each of the following code snippets will compile successfully. If not, correct the error. Then determine what prints.

Code snippet A:

```
int i = 5;
if(i > 2) {
    i = i * 7;
}
System.out.println(i);
```

Compiles and prints 35

Code snippet B:

```
int i = 8;
int j = 0;
if(i % 2 == 0) {
    j = 4;
}
System.out.println(i + j);
```

The scope of `j` is only inside the `if` statement. Can fix this by declaring and initializing `j` before the `if` statement. With the above correction, 12 will print.

Code snippet C:

```
int x = -3;
int y = -2;
if(x * y > 0) {
    int z = x + y;
    y = z * 2;
}
System.out.println(x + " " + y);
```

Compiles and prints "-3 -10"

Code snippet D:

```
int num1 = 42;
int num2 = 0;
if(num1 < 10) {
    num2 = 3;
}
System.out.println(num2);
```

The compiler doesn't know whether `num2` will be initialized before the print statement. We can fix this by initializing `num2` before the `if` statement. This prints 0.

4. Seasons. Write some code that asks the user to enter the current month (as an `int`, 1=January and 12=December) and then prints the season (Winter for Dec-Feb, Spring for Mar-May, Summer for June-Aug, Fall for Sep-Nov).

```
System.out.println("What month is it?");
int month = keyboard.nextInt();

if(month == 12 || month <= 2) {
   System.out.println("Winter");
}
else if(month <= 5) {
   System.out.println("Spring");
}
else if(month <= 8) {
   System.out.println("Summer");
}
else {
   System.out.println("Fall");
}
```

Practice with boolean expressions and order of operations

1. `true` or `false`? Evaluate each of the following boolean expressions when `int x = 4` and `int y = 6`.

```
x <= 5 || y + x > 12 && !(x % 3 == 1)
```

`true`. We have `x <= 5` is `true`, `y + x > 12` is `false`, and `x % 3 == 1` is `true`, so `!(x % 3 == 1)` is `false`. So our expression becomes `true|| false && false`, and the AND gets evaluated first, so `true || false`, which is `true`.

```
y/x > 1 && x != 17
```

`false`. Since `x` and `y` are both `int`s, the quotient `y/x` is also an `int`, so it has value 1. We then have `false && true`, which is `false`.

```
!(y % 4 % 2 == 0 || !((x + y / 3) >= y))
```

`false`. Simplifying this one step at a time, we have:

```
!(6 % 4 % 2 == 0 || !((4 + 6 / 3) <= 6))
!(2 % 2 == 0 || !(4 + 2 <= 6))
!(true || !true)
!(true || false)
!true
false
```

2. What prints? What prints when each of the following pieces of code runs?

```
int month = 2;
int day = 20;
System.out.println("Tomorrow is " + month/day);
```

Tomorrow is 0

```
int month = 2;
int day = 20;
System.out.println("Tomorrow is " + month + "/" + day);
```

Tomorrow is 2/20

```
int age = 19;
System.out.println("In three years your age will be: " + age + 3);
System.out.println("Your age in three years is: " + (age + 3));
System.out.println(age + 3 + " is your age in three years");
```

In three years your age will be 193
Your age in three years is: 22
22 is your age in three years

3. Broken code. Assume that the declaration and initialization `int x = 7;` appears somewhere earlier in the code. None of the following pieces of code will compile without error. Make a small change to fix the error without changing the intended meaning of the code.

```
if(!x < 17) {
   System.out.println("yes");
}
```

```
if(!(x < 17)) {
   System.out.println("yes");
}
```

Needs parenthesis around `x < 17` because `!` has higher precedence than <.

```
int y = 4;
if(x < -1 || < y) {
   System.out.println("yes");
}
```

```
int y = 4;
if(x < -1 || x < y) {
   System.out.println("yes");
}
```

Needs a complete boolean expression on either side of the OR.

```
if(10 >= x > 2) {
   System.out.println("yes");
}
```

```
if(10 >= x && x > 2) {
   System.out.println("yes");
}
```

Can't chain inequalities.

Practice with loops

1. What prints? Consider the following `while` loop. What is the output?

```
int i = 0;
while(i < 5) {
   int j = 0;
   while(j < 3) {
      System.out.print(i + j);
      j++;
   }
   System.out.println();
   i++;
}
```

```
012
123
234
345
456
```

2. `while` and `for`. Translate the following `while` loop into a `for` loop that does the same thing.

```
int i = 0;
while(i < 100) {
   System.out.println(i * 7);
   i++;
}
```

```
for(int i = 0; i < 100; i++) {
   System.out.println(i * 7);
}
```

3. Pretty patterns. Write some nested `while` loops that print the following pattern:

```
******
*    *
*    *
*    *
*    *
******
```

```
int i = 0;
while(i < 6) {
   if(i == 0 || i == 5) {
      System.out.println("******");
   }
   else {
      System.out.println("*    *");
   }
   i++;
}
```

Another solution that prints an `n` by `n` border for any `int n`:

```
int i = 0;
while(i < n) {
   if(i == 0 || i == n - 1) {
      int j = 0;
      while(j < n) {
         System.out.print("*");
         j++;
      }
      System.out.println();
   }
   else {
      System.out.print("*");
      int j = 0;
      while(j < n - 2) {
         System.out.print(" ");
         j++;
      }
      System.out.println("*");
   }
   i++;
}
```

And yet another solution:

```
for(int row = 0; row < n; row++) {
   for(int col = 0; col < n; col++) {
      if(row == 0 || row == n-1 || col == 0 || col == n-1) {
         System.out.print("*");
      }
      else {
         System.out.print(" ");
      }
   }
```

```
    System.out.println();
}
```

Other solutions are also possible.

4. Comparing code. Do the following two pieces of code do the same thing? If so, what do they both do? If not, change the second in some small way so that they do the same thing.

Code snippet A:

```
for(int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```

Prints `1 2 3 4 5 6 7 8 9 10`

Code snippet B:

```
for(int i = 10; i > 0; i--) {
    System.out.println(10 - i);
}
```

Prints `0 1 2 3 4 5 6 7 8 9`

We can get them to do the same thing by making the second loop counter `i` start at 9.

5. Improving code that already works. What is stylistically not so great about the following piece of code? Fix it to improve the code style without changing what it does.

```
int i = 0;
while(i < 1) {
    System.out.print(i + " ");
    i++;
}
System.out.println();
i = 0;
while(i < 2) {
    System.out.print(i + " ");
    i++;
}
System.out.println();
i = 0;
while(i < 3) {
    System.out.print(i + " ");
    i++;
}
System.out.println();
i = 0;
```

```
while(i < 4) {
   System.out.print(i + " ");
   i++;
}
System.out.println();
```

Since we're repeating what is essentially the same code four times in a row, it would be better to put it in some sort of loop to make the code more concise. Here's one way to do it:

```
for(int j = 1; j <= 4; j++) {
   int i = 0;
   while(i < j) {
      System.out.print(i + " ");
      i++;
   }
   System.out.println();
}
```

6. Finding factors. Write some code to print out all of the factors of all numbers from 1 to 100. For example, the first few lines of your code's output should be:

```
1: 1
2: 1 2
3: 1 3
4: 1 2 4
5: 1 5
```

Do this twice, once with `while` loops and once with `for` loops.

Here's a solution with `while` loops:

```
int i = 1;
while(i <= 100) {
   System.out.print(i + ": ");
   int j = 1;
   while(j <=i) {
      if(i % j == 0) {
         System.out.print(j + " ");
      }
      j++;
   }
   System.out.println();
   i++;
}
```

Here's a solution with `for` loops:

```
for(int i = 1; i <= 100; i++) {
   System.out.print(i + ": ");
   for(int j = 1; j <= i; j++) {
      if(i % j == 0) {
         System.out.print(j + " ");
      }
   }
   System.out.println();
}
```

Practice with methods

1. Method headers. Write the *header* for each of the following methods:

- A method that, given a length and a width, prints out a rectangle of $\star$s of the specified size.
  public static void printRect(int length, ind width)
- A method that, given an integer n, computes and returns $n!$ ($n!$, read as "n factorial," is the product of all integers from 1 to n).
  public static int factorial(int n)
- A method that, given an integer x, returns whether or not x is prime.
  public static boolean isPrime(int x)
- A method that, given an array of doubles, finds and returns the largest number stored in that array.
  public static double findLargest(double[] theArray)
- A method that, given a string and a character (stored in a variable of type `char`, which we haven't talked about yet—a `char` is a single letter, number, or other symbol), returns a count of the number of times that character appears in the string.
  public static int countChar(String str, char c)

2. Method bodies. Now write the *body* of the first three methods listed above. (The fourth method, which uses arrays, is something that you should be able to write at this point but is more involved than what I expect you to know about arrays on this exam. The fifth method, which involves manipulating Strings, is not something that I expect you to know how to write at this point.)

I've interpreted the "length" of the rectangle to mean the number of rows, and the "width" to mean the number of columns.

```
public static void printRect(int length, ind width) {
   for(int row = 0; row < length; row++) {
      for(int col = 0; col < width; col++) {
         System.out.print("*");
      }
      System.out.println();
   }
}

public static int factorial(int n) {
   int result = 1;
   for(int i = n; i > 0; i--) {
      result = result * i;
   }
   return result;
}
```

```
public static boolean isPrime(int x) {
   for(int y = 2; y < x; y++) {
      if(x%y == 0) {
         return false;
      }
   }
   return true;
}
```

3. Programs that use methods. Write a program that asks the user to enter two prime numbers, then prints a rectangle of the dimensions specified by the user. Your program should ensure that the user behaves well (i.e., ask the user to try again until they've actually entered positive prime numbers). You can assume that the user will only enter integers. Your program should call the methods you wrote above, and you can feel free to write new methods if you'd like.

```
public static void main(String [] args) {
   int firstNum = getAPrime();
   int secondNum = getAPrime();
   printRect(firstNum, secondNum);
}

public static int getAPrime() {
   System.out.println("Enter a prime number.");
   int x = keyboard.nextInt();
   while(!isPrime(x)) {
      System.out.println("That wasn't prime. Please try again.");
      x = keyboard.nextInt();
   }
   return x;
}
```

This program calls the isPrime and printRect methods written above.

Practice with arrays

1. Creating arrays. Write a piece of code to declare and allocate an array of size 10.

```
int[] myArray = new int[10];
```

If we had wanted an array that stores values of a different type, we could have written this differently, e.g.:

```
double[] anotherArray = new double[10];
boolean[] yetAnother = new boolean[10];
```

and so forth.

2. Storing values in arrays. Write a piece of code to store multiples of 2 in an array.

We'll use the array `myArray` declared above. I'm assuming that we want our multiples of 2 to start at 2.

```
for(int i = 0; i < myArray.length; i++) {
   myArray[i] = 2 * (i + 1);
}
```

3. Printing arrays. Write a method that prints the contents of an array of ints.

We've seen this one a few times at this point!

```
public static void printArray(int[] toPrint) {
   for(int i = 0; i < toPrint.length; i++) {
      System.out.print(toPrint[i] + " ");
   }
   System.out.println();
}
```