Practice with variables and types – Key

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

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

A: Remove the `int` from the last line.

Code snippet B:

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

A: Change the type of `anotherNum` to `int`.

Code snippet C:

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

A: Correct.

Code snippet D:

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

A: Change the value `43.7` to, e.g., `43` (or any other integer)

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

```
int i = 5;
double j = 21.3;
i = i + j; // A: Add a cast (int) to the left of j on the last line
```

```
int totalLabScore = 84;
int numLabs = 10;
double averageScore = totalLabScore / numLabs;
// A: Add a cast (double)  to the left of totalLabScore and/or of numLabs on the
last line.
```

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.)
A:

```
System.out.println("Enter the number of feet: ");
int feet = kb.nextInt();
System.out.println("Enter the number of inches: ");
int inches = kb.nextInt();
System.out.println("Height in meters: " + (feet + inches / 12.0) / 3.28);
```

Practice with `if` statements – Key

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");
}
```

A: This code prints:
•three if num ≤ 54;
•two if $54 \leq$ num ≤ 82;
•one if num > 82;

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");
}
```

A: This code prints:
•three if num ≤ 54;
•two if $54 \leq$ num ≤ 82;
•one if num > 82;

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");
}
```

A: This code prints:
•three two if num < 54;
•two if $54 \leq$ num ≤ 82;
•one if num > 82;

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");
}
```

A: This code prints:
•three if num ≤ 54;
•two if $54 <$ num < 82;
•nothing if num > 82;

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)

A:

```
System.out.print("What is your age? ");
int age = kb.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");
            }
        }
    }
}
```

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);
```

A: The code is correct. It prints 35.

Code snippet B:

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

A: The variable `j` is out of scope when we try to compute the sum on the last line, because `j` is declared inside the `if`-block. We need to declare it outside the `if`-block, initialize it to any integer, and only re-assign it inside the `if`-block. The code will then print 12. Here is the corrected code:

```
int i = 8;
int j = -24532513;
if(i % 2 == 0) {
   j = 4;
}
System.out.println(i + j);
```

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);
```

A: The code is correct. It prints "`-3 -10`".

Code snippet D:

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

A: The variable `num2` is not initialized. We need to initialize it to a valid integer, which is then printed. Here is the corrected code:

```
int num1 = 42;
int num2 = 18;
if(num1 < 10) {
   num2 = 3;
}
System.out.println(num2);
```

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.print("What month is it? ");
int month = kb.nextInt();
if (month == 12 || month <= 2) {
    System.out.println("It's Winter");
} else if (month <= 5) {
    System.out.println("It's Spring");
} else if (month <= 8) {
    System.out.println("It's Summer");
} else {
    System.out.println("It's 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)
```

A: It's true, because it is equivalent to

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

as the `&&` operator has higher precedence than the `||` operator, and indeed `x` (which has value 4) is less-than-or-equal-to 5.

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

A: It's false because the integer division on the left has result 1, which is not *greater* than one.

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

A: It's false. The test to the left of the `||` is equivalent to `((y % 4) % 2 == 0)`, which is true, and therefore so is the boolean expression inside the most external pair of `()`, but then the `!` operator flips the value to 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);
```

A: This code prints "`Tomorrow is 0`", because the integer division is executed first, as the `/` operator has higher precedence than the `+` operator.

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

A: This code prints "`Tomorrow is 2/20`", because the operations are executed left to right as all the operators are the same.

```
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");
```

A: This code prints:

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

because the operations are executed left to right as all the operators are the same.

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");
}
```

A: Change to `if ( ! (x < 17 )) {`

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

A: Change to `if ( x < -1 || x < y) {`

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

A: Change to `if (10 >= x && x > 2) {`

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++;
}
```

A: It prints

```
0 1 2
1 2 3
2 3 4
3 4 5
4 5 6
```

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++;
}
```

A: The corresponding `for` loop is

```
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:

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

A: (different solution than the one given in class, just to see other approaches)

```
int line = 0;
while (line < 6) {
    System.out.print("*");
    int column = 1;
    while (column < 5) {
        if (line == 0 || line == 5) {
            System.out.print("*");
        } else {
            System.out.print(" ");
        }
        column++;
    }
    System.out.println("*");
    line++;
}
```

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);
}
```

Code snippet B:

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

A: The first prints "`1 2 3 4 5 6 7 8 9 10`". The second prints "`0 1 2 3 4 5 6 7 8 9`". The correction changes the print statement in the second `for`-loop to:

```
System.out.println(10 - i + 1);
```

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();
```

A: The same code is repeated over and over again with just different upper limits to the test in the `while`-loop declarations. We can change the whole code to

```
for (int j = 1; j < 5; j++) {
    for (int i = 0; i < j; i++) {
        System.out.print(i + " ");
    }
    System.out.println();
}
```