**AP CS: Modular Arithmetic**

Imagine we decide to all arithmetic in base 5. Doing arithmetic in different number bases is not always easy; for example, you don’t want to memorize a multiplication table for base 16. (B\*C=84?!) So just to make it easier on ourselves, we will consider only the last digits. All numbers which have the same last digit in base 5 will be considered equal:

**25 = 125 = 225 = 325 =…**

In base 10, this looks like

**2=7=12=17=…**

The usual way to show that we are using this sytem is to replace the = with a , and also append the suffix (mod 5). We thus write, for example, 12 7 (mod 5). We say 12 is congruent to 7 mod 5.

Another way to look at mods is that 2,7,12,17, etc. all have the same remainder (2) when dividing by 5. This method of viewing modular arithmetic makes actual computation much easier. It is often useful to find the smallest nonnegative integer which is congruent to x mod y. For example, the smallest integer congruent to 12 mod 5 is 2. When we perform this task, we say that we ‘mod out’ the 12. To mod out 7631 in mod 7, we can either find 7631 in base 7 and look at the last digit, or we can divide by 7 and look at the remainder.

In this discussion of remainders, we have used mods to denote the amount more than a multiple of 5 a given number is. For example, 2, 7, 12, 17, etc. are all exactly 2 more than a multiple of 5. In the same way, we can define negative mods as the amount less than a multiple of 5 a s number is. Since 2, 7, 12, 17, etc. are all 3 less than the nearest multiple of 5, they are all congruent to -3 mod 5. Extending this reasoning, we can write in mod 5:

…

Note that each term is five away from the one before it.

You may recall that in grade school, you learned division like this: 13/5= 2 with a remainder of 3. And it follows that 5\*2+3=13. In Java, the remainder operator, %, returns the remainder when used with two integers.

Type and run the following.

class RemainderExample

{

public static void main ( String[] args )

{

int quotient, remainder;

quotient = 17 / 3;

remainder = 17 % 3;

System.out.println("The quotient : " + quotient );

System.out.println("The remainder: " + remainder );

System.out.println("The original : " + (quotient\*3 + remainder) );

}

}

In many calculations, it is convenient to do everything with integers, so both **/** and **%** are needed.

**You Try**

**Evaluate the following.**

1. **10%3**
2. **7%5**
3. **129%100**
4. **17%2**
5. **7%5**
6. **10%6**
7. **1999%100**
8. **18%2**
9. **If X is odd, what is X%2?**
10. **If X is even, what is X%2?**
11. **If X is divisible by 6, what is X%3?**
12. **If X is divisible by 4, what is X%4?**
13. **If you exchange 372 pennies for dollar bills, how many bills do you get? How many pennies are left over? Write two lines Java code to answer this.**

**Answers:**

1. **10%3 >1**
2. **7%5 >2**
3. **129%100 >29**
4. **17%2 >1**
5. **7%5 >2**
6. **10%6 >4**
7. **1999%100 >99**
8. **18%2 >0**
9. **If X is odd, what is X%2? >1**
10. **If X is even, what is X%2? >0**
11. **If X is divisible by 6, what is X%3? >0**
12. **If X is divisible by 4, what is X%4? >0 or 4**
13. **If you exchange 372 pennies for dollar bills, how many bills do you get? How many pennies are left over? Write two lines Java code to answer this.**

**372%100= 72 pennies**

**372/100=3 dollars**