Theoretical Computer Science

1. Knights and Knaves

You wake up. Wait, where are you? You groggily come to your senses. The last thing you remember was falling asleep during a discrete math final. You look around, and spot a helpful sign:

> THE ISLAND OF KNIGHTS AND KNAVES
>
> Welcome to our cozy isle! All our inhabitants are very friendly, but be warned: half of them are knaves, who always lie. The other half, happily, are knights, who always tell the truth. Have a nice time!

Maybe if you wander around a bit, you can figure out how to get back home…

Please justify all your answers for this section.

1.1 The first encounter

You meet two inhabitants, Rémi and Miguel. Rémi says something to you, but he mumbles and you're unable to make it out. Miguel notices that you didn't catch it, and says "Rémi said that he is a knave." Is Rémi a knight or a knave? Is Miguel a knight or a knave?

1.2 The second encounter

You meet another inhabitant, who says "I am about to tell you the truth about whether I'm a knight or a knave." He makes another statement, and true to his word, you then know with 100% certainty whether he's a knight or a knave. What was his second statement, and is he a knight or a knave?

1.3 A conundrum

Is it possible for any inhabitant to say "You can't deduce if I'm a knight or a knave from this statement"?

2. SKI Combinator Calculus

In this system, there are 3 functions: S, K, and I.

I is the identity function: $Ix = x$.

K is the constant function. It takes two arguments and always returns the first: $Kxy = x$.

S follows the rule $Sxyz = xz(yz)$.

Function application is denoted by juxtaposition and is left-associative (eg. fxy means $(f(x))(y)$). If a function does not have enough arguments, it will simply not reduce further (eg. Sxy evaluates to Sxy).

For example, let's suppose we have a function *eq* that returns true if its arguments are equal (like *eq* SS), and false if they aren't (like *eq* SK). Let's also suppose we have a function *rev* that reverses a string. We could check if a string was a palindrome or not like so: *S eq rev*. If this was applied to a string: *S eq rev str*, the result would be *eq str (rev str)* by S's reduction rule, checking if *str* is a palindrome. However, we actually don't have strings or booleans or *eq* or *rev* (all we have are the functions S, K, and I); this section is merely hypothetical to help explain S.

As another example, consider the function $Mx = xx$. It takes an argument and applies the argument to itself. One way to define it using S, K, and I is like this: $M = SII$. Then, $Mx = SIIx = Ix(Ix) = I(x)(I(x))$. Now, we know that $Ix = x$, so this reduces to $x(x)$, or simply xx, as intended.

2.1 False

One way to represent a boolean false is with a function that discards its first argument: $Fxy = y$. Define this in terms of S, K, and I.

2.2 Idiot

It turns out that you don't actually need I! Define it in terms of S and K. (I is sometimes known as the *idiot bird*.)

2.3 Compose

Define the B combinator, which follows the rule $Bxyz = x(yz)$. In other words, it composes x and y.