Exercise 1: Getting Acquainted with the System

1. Login.
2. Start PVS
3. Make a new pvs file inside pvs-emacs. `C-x C-f ___.pvs`
4. Create a new theory. The structure of a theory file is:

   ex1: THEORY

   BEGIN

   END ex1

   You can either type this in or issue command `Esc 1 M-x nt`
5. Create a function after BEGIN. Example: `f(x) = x+1`
6. Type check your theory: `M-x tc` (This should give an error.)
7. The type of the function must be declared. Example: `f(x):nat = x+1`
8. Typecheck your theory: `M-x tc`
9. More type information needed. The type of the variable `x` must be declared. Either declare `x` as a variable before the function or within the function:

   ```
   x : VAR nat
   f(x):nat = x+1
   ```

   or

   ```
   f(x:nat):nat = x+1
   ```

10. The theory should type check without errors now.
11. Create a theorem based on the previous function.
    Example: `trivial : THEOREM (FORALL (x:nat): f(x) > x)`
12. A PVS interactive session to prove "trivial" can be started by placing the cursor over the declaration of "trivial" and typing `M-x pr`
13. Prove the theorem by giving the command grind at the prompt: `(grind)`
14. Let's prove the theorem again the "old fashion way". Again place the cursor over the declaration of "trivial" and typing `M-x pr`
15. try again?: yes
16. Rerun existing proof?: no
17. Eliminate the universal quantifier: `(skosimp*)`
18. Expand the function `f` with its definition: `(expand "f")`

19. Use arithmetic simplification to prove the formula: `(assert)`
20. Compare the Run Time and the Real Time for the two methods of proof. This will be an issue when formulas become complex.
21. Let's prove it a third time. Issue `M-x pr` again and answer all the questions. Once the sequent appears in the `*pvs*` buffer, type `TAB *`. Then put the cursor on the `f` in the sequent and type `TAB e`. Then type `TAB a`.
22. Now type `M-x spt`.
23. Place cursor on the `THEOREM` keyword, and type M-x edit-proof. Then type `C-x o` followed by `C-x 1`.
24. End of Exercise 1.

If you finish early add

```
a: VAR nat
another:  LEMMA f(a-f(a)) = 0
```

to your theory. Issue `M-x tc`. Next issue `M-x show-tccs`. Notice that a window pops up with the following in it:

```
another_TCC1: OBLIGATION FORALL (a: nat): a - f(a) >= 0;
```

The message tells you what line generated this obligation. Why do you think this obligation was created? Is it provable? Prove lemma `another` using `M-x pr (grind)`. Issue `M-x spt` and notice that `trivial` is now `unchecked`. Whenever you change something in a theory the proofs have to be rerun. Issue command `M-x prt` to reprove everything in the theory. Notice that `another_TCC1` is not proved and the status of `another` is `proved - incomplete`. This tells you that this lemma depends upon something that is unproven. Your job is not done until you see `proved - complete` everywhere.