Style Guide

A large part of any programmer's job is not to write code, but to read it. It may be code they wrote yesterday, or two years ago, or it might be someone else's code. It is therefore of paramount importance that you make sure your code is *easily readable* and that its *meaning* is easily discerned. Style is a part of this, language design choices is another.

> Write you code for the humans that will read it in the future.

General principles

Some general principles that follow this rule would be:

- Each line of code should try to do "one thing". In particular avoid overly long lines.
- Choose function and variable names with readability in mind. They should reflect as much as possible what the function does or what the variable represents.
- Factor out common patterns into functions.
- Avoid having very long and complicated functions.
- Use variable bindings to give names to intermediate computations when it makes sense to do so.
- Avoid cleverness if it will result in obscure code.
- Write clear comments, and on the things that matter.
- Use a uniform style. The specific choices are less important than is making some choices and being consistent.

Specific style issues

There are numerous conventions one can follow for style. Some are really just good practices, others are more matters of choice.

Whitespace

- Avoid using actual tab characters. You can set your text editor to insert spaces when you press Tab.
- Leave one space around binary operators, and no space after unary operators.
- Always leave a space after commas and semicolons.
- Avoid spaces after an opening bracket/parenthesis and before a closing bracket/parenthesis.
- Use spaces right before an opening bracket/parenthesis and after a closing bracket/parenthesis, if it will not conflict with the previous rule.
- Use empty lines inbetween top-level `let` definitions, but not within those definitions.

Naming

- Use meaningful variable and function names.
- When applicable, use a verb in the function name to indicate intent, e.g. is_positive or has_kings.
- You cannot start normal names with an uppercase letter. That is reserved for constructors (we will see constructors later in the course).
- Use underscores to separate multiple words in variables and functions, e.g. list_of_things.
- Use camel-case for multiple-word constructors, e.g. IntVar.

Language constructs

In most of these constructs we will provide multiple forms.

Top-level let bindings:

```
let x = ...     (* for short assignments *)
let x =
  ...     (* Indent by 2 or 3 spaces typically *)
```

Local let bindings:

```
let x = ... in ...       (* short assignments *)

let x = ... in
...            (* line it up with the let it matches *)
```

My personal favorite:

```
let x = ...
in ...
```

and for multiple bindings:

```
let x = ... in
let y = ... in
let z = ...
in ....
```

Conditionals:

```
if ... then ... else ...      (* for extremely short conditionals *)
if ...
then ...          (* align vertically *)
else ...
```