# Haskell Cheat Sheet

This cheat sheet lays out the fundamental elements of the Haskell language: syntax, keywords and other elements. It is presented as both an executable Haskell file and a printable document. Load the source into your favorite interpreter to play with code samples shown.

## Basic Syntax

### Comments

A single line comment starts with '`--`' and extends to the end of the line. Multi-line comments start with '`{-`' and extend to '`-}`'. Comments can be nested.

Comments above function definitions should start with '`{- |`' and those next to parameter types with '`-- ^`' for compatibility with Haddock, a system for documenting Haskell code.

### Reserved Words

The following words are reserved in Haskell. It is a syntax error to give a variable or a function one of these names.

- `case`
- `class`
- `data`
- `deriving`
- `do`
- `else`
- `if`
- `import`
- `in`
- `infix`
- `infixl`
- `infixr`
- `instance`
- `let`
- `of`
- `module`
- `newtype`
- `then`
- `type`
- `where`

### Strings

- `"abc"` – Unicode string, sugar for `['a','b','c']`.
- `'a'` – Single character.

**Multi-line Strings** Normally, it is a syntax error if a string has any newline characters. That is, this is a syntax error:

```
string1 = "My long
string."
```

Backslashes ('\') can "escape" a newline:

```
string1 = "My long \
\string."
```

The area between the backslashes is ignored. Newlines *in* the string must be represented explicitly:

```
string2 = "My long \n\
\string."
```

That is, `string1` evaluates to:

```
My long string.
```

While `string2` evaluates to:

```
My long
string.
```

**Escape Codes** The following escape codes can be used in characters or strings:

- `\n`, `\r`, `\f`, etc. – The standard codes for newline, carriage return, form feed, etc. are supported.
- `\72`, `\x48`, `\o110` – A character with the value 72 in decimal, hex and octal, respectively.
- `\&` – A "null" escape character which allows numeric escape codes next to numeric literals. For example, `\x2C4` is ∧ (in Unicode) while `\x2C\&4` is `,4`. This sequence cannot be used in character literals.

### Numbers

- `1` – Integer or floating point value.
- `1.0, 1e10` – Floating point value.
- `0o1, 0O1` – Octal value.
- `0x1, 0X1` – Hexadecimal value.
- `-1` – Negative number; the minus sign ("`-`") cannot be separated from the number.

### Enumerations

- `[1..10]` – List of numbers – `1, 2, ..., 10`.
- `[100..]` – Infinite list of numbers – `100, 101, 102, ...` .
- `[110..100]` – Empty list, but `[110, 109 .. 100]` will give a list from 110 to 100.
- `[0, -1 ..]` – Negative integers.
- `[-110..-100]` – Syntax error; need `[-110.. -100]` for negatives.
- `[1,3..99]`, `[-1,3..99]` – List from 1 to 99 by 2, -1 to 99 by 4.

In fact, any value which is in the `Enum` class can be used:

- `['a' .. 'z']` – List of characters – `a, b, ..., z`.
- `['z', 'y' .. 'a']` – `z, y, x, ..., a`.
- `[1.0, 1.5 .. 2]` – `[1.0,1.5,2.0]`.
- `[UppercaseLetter ..]` – List of `GeneralCategory` values (from `Data.Char`).



jgbailey@codeslower.com