# CoffeeScript Quick Reference

□

coffeescript.org

□

□

## General[1]

- Whitespace is significant
- Ending a line will terminate expressions - no need to use semicolons
- Semicolons can be used to fit multiple expressions onto a single line
- Use indentation instead of curly braces `{ }` to surround blocks of code in functions, *`if`* statements, *`switch`*, and *`try`*/*`catch`*
- Comments starts with # and run to the end of the line

## Functions

- Functions are defined by an optional list of parameters in parentheses, an arrow, and an optional function body. The empty function looks like: `->`
- Mostly no need to use parentheses to invoke a function if it is passed arguments. The implicit call wraps forward to the end of the line or block expression.
- Functions may have default values for arguments. Override the default value by passing a non-null argument.

[1] E. Hoigaard © 2554/2011 Rev. α

## Objects and arrays

- Objects and arrays are similar to JavaScript
- When each property is listed on its own line, the commas are optional
- Objects may be created using indentation instead of explicit braces, similar to YAML
- Reserved words, like *`class`*, can be used as properties of an object without quoting them as strings

## Lexical Scoping and Variable Safety

- Variables are declared implicitly when used (no `var` keyword).
- The compiler ensures that variables are declared within lexical scope. An outer variable is not redeclared within an inner function when it is in scope
- Using an inner variable can not shadow an outer variable, only refer to it. So avoid reusing the name of an external variable in a deeply nested function
- CoffeeScript output is wrapped in an anonymous function, making it difficult to accidentally pollute the global namespace
- To create top-level variables for other scripts, attach them as properties on `window`, or to `exports` in CommonJS. Use: `exports ? this`

## Splats

- Splats `...` can be used instead of the variable number of `arguments` object and are available for both function definition and invocation

## Loops and Comprehensions

- Comprehensions *`for`* `...` *`in`* work over arrays, objects, and ranges
- Comprehensions replace for loops, with optional *`when`* guard clauses and the value of the current array index: `for value, index in array`
- Array comprehensions are expressions, and can be returned and assigned
- Comprehensions may replace *`each`*/*`forEach`*, *`map`* or *`select`*/*`filter`*
- Use a range when the start and end of a loop is known (integer steps)
- Use *`by`* to step in fixed-size increments
- When assigning the value of a comprehension to a variable, CoffeeScript collects the result of each iteration into an array
- Return *`null`*, *`undefined`* or *`true`* if a loop is only for side-effects
- To iterate over the key and value properties in an object, use *`of`*
- Use: `for own key, value of object` to iterate over the keys that are directly defined on an object
- The only low-level loop is the *`while`* loop. It can be used as an expression, returning an array containing the result of each iteration through the loop
- *`until`* is equivalent to *`while not`*
- *`loop`* is equivalent to *`while true`*
- The *`do`* keyword inserts a closure wrapper, forwards any arguments and invokes a passed function

## Try/Catch/Finally

- *`try`*/*`catch`* statements are as in JavaScript (although expressions)

## If, Else, Unless, and Conditional Assignment

- *`if`*/*`else`* can be written without parentheses and curly braces
- Multi-line conditionals are delimited by indentation
- *`if`* and *`unless`* can be used in postfix form i.e. at the end of the statement
- *`if`* statements can be used as expressions. No need for `?:`

## Chained Comparisons

- Use a chained comparison to test if a value is within a range: `minimum < value < maximum`

## Array Slicing and Splicing with Ranges

- Ranges can be used to extract slices of arrays
- With two dots `[3..6]`, the range is inclusive (3, 4, 5, 6)
- With three dots `[3...6]`, the range excludes the end (3, 4, 5)
- The same syntax can be used with assignment to replace a segment of an array with new values, splicing it
- Strings are immutable and can not be spliced

## Embedded JavaScript

- Use backquotes `` ` ` `` to embed JavaScript code within CoffeeScript

## Everything is an Expression

- Functions return their final value
- The return value is fetched from each branch of execution
- Return early from a function body by using an explicit `return`
- Variable declarations are at the top of the scope, so assignment can be used within expressions, even for variables that have not been seen before
- Statements, when used as part of an expression, are converted into expressions with a closure wrapper. This allows assignment of the result of a comprehension to a variable
- The following are not expressions: `break`, `continue`, and `return`

## Operators and Aliases

- CoffeeScript compiles `==` into `===`, and `!=` into `!==`. There is no equivalent to the JavaScript `==` operator
- The alias `is` is equivalent to `===`, and `isnt` corresponds to `!==`
- Logical operator aliases: `and` is `&&`, `or` is `||` and `not` is an alias for `!`
- In `while`, `if`/`else` and `switch`/`when` statements the `then` keyword can be used to keep the body on the same line
- Alias for boolean `true` is `on` and `yes` (as in YAML)
- Alias for boolean `false` is `off` and `no`
- For single-line statements, `unless` can be used as the inverse of `if`
- Use `@property` or `@method` instead of `this.something`
- Use `in` to test for array presence
- Use `of` to test for object-key presence

## Existential Operator

- Use the existential operator `?` to check if a variable exists. `?` returns `true` unless a variable is `null` or `undefined`
- Use `?=` for safer conditional assignment than `||=` with numbers or strings
- The accessor variant of the existential operator `?.` can be used to soak up null references in a chain of properties
- Use `?.` instead of the dot accessor `.` in cases where the base value may be `null` or `undefined`. If all of the properties exist then the expected result is returned, if the chain is broken, then `undefined` is returned instead

## Classes, Inheritance, and Super

- Object orientation as in most other object oriented languages
- The `class` structure allows to name the class, set the superclass with `extends`, assign prototypal properties, and define a `constructor`, in a single assignable expression
- Constructor functions are named as the `class` name, to support reflection
- Lower level operators: The `extends` operator helps with proper prototype setup. `::` gives access to an object's prototype. `super()` calls the immediate ancestor's method of the same name
- A class definition is a block of executable code, which may be used for meta programming.
- In the context of a class definition, `this` is the class object itself (the `constructor` function), so static properties can be assigned by using `@property: value`, and functions defined in parent classes can be called with: `@inheritedMethodName()`

## Destructuring Assignment

- To make extracting values from complex arrays and objects convenient, CoffeeScript implements destructuring assignment
- When assigning an array or object literal to a value, CoffeeScript breaks up and matches both sides against each other, assigning the values on the right to the variables on the left
- The simplest case is parallel assignment `[a,b] = [b,a]`
- It can be used with functions that return multiple values
- It can be used with any depth of array and object nesting to get deeply nested properties and can be combined with splats

## Function binding

- The fat arrow `=>` can be used to define a function and bind it to the current value of `this`
- This is helpful when using callback-based libraries, for creating iterator functions to pass to `each` or event-handler functions to use with `bind`
- Functions created with `=>` are able to access properties of the `this` where they are defined

## Switch/When/Else

- The `switch` statement do not need a `break` after every case
- A `switch` is a returnable, assignable expression
- The format is: `switch` condition, `when` clauses, `else` the default case
- Multiple values, comma separated, can be given for each `when` clause. If any of the values match, the clause runs

## String Interpolation, Heredocs, and Block Comments

- Single-quoted strings are literal. Use backslash for escape characters
- Double-quoted strings allow for interpolated values, using `#{ ... }`
- Multiline strings are allowed
- A heredoc `'''` can be used for formatted or indentation-sensitive text (or to avoid escaping quotes and apostrophes)
- The indentation level that begins a heredoc is maintained throughout, so the text can be aligned with the body of the code
- Double-quoted heredocs `"""` allow for interpolation
- Block comments `###` are similar to heredocs, and are preserved in the generated code

## Extended Regular Expressions

- Extended regular expressions are delimited by `///` and are similar to heredocs and block comments.
- They ignore internal whitespace and can contain comments

## Aliases

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| `and` | : | `&&` | `or` | : | `\|\|` | `not` | : | `!` |
| `is` | : | `==` | `isnt` | : | `!=` | | | |
| `yes` | : | `true` | `no` | : | `false` | | | |
| `on` | : | `true` | `off` | : | `false` | | | |

## Miscellaneous

- Twitter comments to @autotelicum

http://autotelicum.github.com/Smooth-CoffeeScript/