CoffeeScript Cheatsheet

A concise reference for CoffeeScript, covering syntax, features, and best practices for this elegant JavaScript dialect.

Core Syntax

Basic Structure

Function Definition

```
square = (x) -> x * x
```

Equivalent to:

```
var square = function(x) {
  return x * x;
};
```

Implicit Return

CoffeeScript functions implicitly return the value of the last expression.

```
calculate = (a, b) ->
  sum = a + b
  sum * 2  # Implicitly returned
```

Object Literals

```
person = {
  name: 'Alice'
  age: 30
}
```

Equivalent to:

```
var person = {
  name: 'Alice',
  age: 30
};
```

Array Literals

```
numbers = [1, 2, 3, 4, 5]
```

Equivalent to:

```
var numbers = [1, 2, 3, 4, 5];
```

String Interpolation

```
name = 'Bob'
greeting = "Hello, #{name}!"
```

Results in:

```
var name = 'Bob';
var greeting = "Hello, " + name + "!";
```

Multiline Strings

```
longString = """
  This is a very
  long string.
"""
```

Equivalent to:

```
var longString = "This is a very\nlong string.";
```

Operators and Keywords

`is`, `isnt`

Equality checks. `is` is `===` and `isnt` is `!==`.

```
if age is 18
  console.log 'You are 18'
```

`not`

Logical NOT. `not true` is equivalent to `!true`.

```
if not authenticated
  console.log 'Access denied'
```

`and`, `or`

Logical AND and OR.

```
if sunny and temp > 25
  console.log 'Enjoy the weather'
```

`unless`

The opposite of `if`. Executes the block if the condition is false.

```
unless raining
  console.log 'Lets go outside'
```

`@`

Shorthand for `this`. Useful in class methods.

```
class Person
  constructor: (@name)
  greet: -> console.log "Hello, @name!"
```

`?`

Existential operator. Returns `true` if a variable is not `null` or `undefined`.

```
console.log name?  # Checks if name exists
```

Control Flow & Loops

Conditional Statements

`if / else`

```
if age >= 18
  console.log 'Adult'
else
  console.log 'Minor'
```

`unless`

```
unless hungry
  console.log 'Not hungry'
```

`else if`

```
if score > 90
  grade = 'A'
else if score > 80
  grade = 'B'
else
  grade = 'C'
```

Loops

`for...in`

Iterates over the keys of an object.

```
obj = {a: 1, b: 2, c: 3}
for key, value of obj
  console.log "#{key}: #{value}"
```

`for...of`

Iterates over the elements of an array.

```
numbers = [10, 20, 30]
for num in numbers
  console.log num
```

`for...from...to`

Creates a range-based loop.

```
for i in [1..5]
  console.log i
```

`while`

```
i = 0
while i < 5
  console.log i
  i++
```

`until`

The opposite of `while`.

```
i = 0
until i >= 5
  console.log i
  i++
```

List Comprehensions

CoffeeScript's list comprehensions provide a concise way to generate arrays.

```
squares = (x * x for x in [1..5])
# squares is now [1, 4, 9, 16, 25]
```

With conditions:

```
evenSquares = (x * x for x in [1..10] when x % 2 is 0)
# evenSquares is now [4, 16, 36, 64, 100]
```

Classes & Objects

Class Definition

Basic Class

```
class Animal
  constructor: (@name)
  move: ->
    console.log "#{@name} moved"

animal = new Animal('Lion')
animal.move()
```

Inheritance

```
class Dog extends Animal
  bark: ->
    console.log 'Woof!'

dog = new Dog('Buddy')
dog.move()
dog.bark()
```

Class Variables

```
class MathUtils
  @PI: 3.14159
  @square: (x) -> x * x

console.log MathUtils.PI
console.log MathUtils.square(5)
```

Object Creation

Creating instances of classes is straightforward:

```
class Point
  constructor: (@x, @y)

point = new Point(10, 20)
console.log point.x, point.y
```

Prototypes

CoffeeScript classes automatically manage prototypes, making inheritance and method sharing simple.

```
class Vehicle
  start: -> console.log 'Engine started'

class Car extends Vehicle
  drive: -> console.log 'Driving'

car = new Car()
car.start()
car.drive()
```

Functions

Function Definition

Basic Function	`add = (a, b) -> a + b`
	`console.log add(5, 3)`

Functions with no arguments	`sayHello = ->`
	`console.log 'Hello!'`
	`sayHello()`

Multiline Functions	`calculate = (x, y) ->`
	`  sum = x + y`
	`  sum * 2`
	
	`console.log`
	`calculate(2, 3)`

Arguments

Default Arguments	`greet = (name = 'Guest') ->`
	`console.log "Hello, #`
	`{name}!"`
	
	`greet()`
	`# Output: Hello, Guest!`
	`greet('Alice')`
	`# Output: Hello, Alice!`

Splats (Variable Arguments)	`sum = (numbers...) ->`
	`  total = 0`
	`  total += num for num in`
	`numbers`
	`  total`
	
	`console.log sum(1, 2, 3, 4)`

Bound Functions

Use `=>` instead of `->` to bind the function to the current `this` context. This is particularly useful in event handlers and callbacks.

```
class Button
  constructor: (@element)
    @element.addEventListener 'click', (event)
=> @handleClick(event)

  handleClick: (event) ->
    console.log 'Button clicked', @element
```