# Julia Cheatsheet

A comprehensive cheat sheet for the Julia programming language, covering syntax, data types, control flow, functions, and more. Designed for quick reference and efficient learning.

## Basics and Syntax

### Basic Syntax

| | |
|---|---|
| **Comments** | `#` Single-line comment<br>`#=` ... `=#` Multi-line comment |
| **Variables** | `x = 10` Assignment<br>`x` Variable name (case-sensitive) |
| **Operators** | `+, -, *, /, ^` Arithmetic operators<br>`==, !=, <, >, <=, >=` Comparison operators<br>`&&, \|\|, !` Logical operators |
| **Line Endings** | Statements can be on the same line if separated by `;` |
| **String literals** | `"Hello, world!"` String literals |
| **String interpolation** | `"The value of x is $(x)"` |

### Data Types

- **Numeric Types:** `Int64`, `Float64`, `ComplexF64`, `BigInt`, `BigFloat`
- **Boolean:** `true`, `false`
- **String:** `String`
- **Character:** `Char`
- **Arrays:** `Array{Type, N}`
- **Tuples:** `Tuple{Type1, Type2, ...}`
- **Dictionaries:** `Dict{KeyType, ValueType}`
- **Symbols:** `:symbol`

## Control Flow

### Conditional Statements

```
if condition
  # code
elseif condition
  # code
else
  # code
end
```

```
condition ? value_if_true : value_if_false
```

### Loops

**For Loop**

```
for i in 1:10
  println(i)
end
```

**While Loop**

```
i = 1
while i <= 10
  println(i)
  i += 1
end
```

**Break/Continue**

`break` - Exit the loop

`continue` - Skip to the next iteration

### Exception Handling

```
try
  # code that might throw an error
catch e
  # handle the error
finally
  # code that always runs
end
```

`throw(ErrorException("message"))` - Throw an exception

## Functions

### Function Definition

```
function my_function(arg1, arg2)
  # function body
  return result
end
```

```
my_function(arg1, arg2) = arg1 + arg2  # Shorthand definition
```

### Arguments

**Positional Arguments**

Arguments passed in the order they are defined.

**Keyword Arguments**

```
function my_function(; kwarg1=default_value, kwarg2)
  # function body
end
```

**Default Argument Values**

```
function my_function(arg1, arg2=default_value)
  # function body
end
```

**Varargs**

```
function my_function(args...)
  # function body
end
```

### Return Values

`return value` - Explicitly return a value

If no return statement, the last evaluated expression is returned.

Multiple return values using tuples:

```
return (value1, value2)
```

### Anonymous Functions

```
x -> x^2  # Anonymous function that squares its argument
```

## Arrays and Data Structures

### Arrays

| | |
|---|---|
| **Creating Arrays** | `a = [1, 2, 3]  # 1D array`<br>`a = [1 2 3; 4 5 6]  # 2D array`<br>`a = Array{Float64}(undef, (3, 3)) # uninitialized 3x3 array` |
| **Indexing** | Arrays are 1-indexed.<br>`a[1]` - First element<br>`a[1:3]` - Slice from 1 to 3<br>`a[:, 1]` - All rows, first column |
| **Array Operations** | `+, -, *, /` Element-wise operations<br>`.*, ./, .^` Broadcasting operations<br>`size(a)` Dimensions of array `a`<br>`length(a)` Length of array `a` |

### Dictionaries

| | |
|---|---|
| **Creating Dictionaries** | `d = Dict("a" => 1, "b" => 2)` |
| **Accessing Values** | `d["a"]` - Access value for key “a” |
| **Adding/Updating Values** | `d["c"] = 3` - Add a new key-value pair<br>`d["a"] = 4` - Update existing value |
| **Checking for Key** | `haskey(d, "a")` - Check if key “a” exists |

### Tuples

```
t = (1, "hello", 3.14) # Creating a tuple
```

`t[1]` - Accessing tuple elements (1-indexed)

Tuples are immutable.