



Basics & Syntax

Basic Types

<code>Int32</code>	32-bit signed integer (default <code>Int</code>)
<code>Int64</code>	64-bit signed integer
<code>Float3</code>	32-bit floating point number
<code>2</code>	
<code>Float6</code>	64-bit floating point number (default <code>Float</code>)
<code>4</code>	
<code>Bool</code>	<code>true</code> or <code>false</code>
<code>String</code>	UTF-8 encoded character sequence
<code>g</code>	
<code>Char</code>	Unicode code point

Variable Declaration

```
var name = value - Mutable variable with type inference.  

name = new_value - Assigning a new value.  

name : Type = value - Mutable variable with explicit type.  

CONST_NAME = value - Constant (immutable).
```

Operators

Arithmetic	<code>+, -, *, /, %, **</code> (exponentiation)
Comparison	<code>==, !=, >, <, >=, <=</code>
Logical	<code>&&, , !</code>
Bitwise	<code>&, , ^, ~, <<, >></code>
Assignment	<code>=, +=, -=, *=, /=, %=, **=</code>

Control Flow

Conditional Statements

```
`if condition
```

code

```
end`
```

```
`if condition
```

code

```
else
```

code

```
end`
```

```
`if condition
```

code

```
elsif condition
```

code

```
else
```

code

```
end`
```

```
`unless condition
```

code

```
end` (opposite of if)
```

Looping

```
`while condition
```

code

```
end`
```

```
`until condition
```

code

```
end` (opposite of while`)
```

```
`loop do
```

code

```
break if condition
```

```
end`
```

```
(0..5).each do |i| puts i end
```

Case Statement

```
`case value
```

```
when condition1
```

code

```
when condition2
```

code

```
else
```

code

```
end`
```

Methods & Blocks

Method Definition

```
`def method_name(arg1 : Type, arg2 : Type)
```

code

```
return value  
end`
```

```
`def method_name(arg1 : Type, arg2 : Type) : ReturnType
```

code

```
return value  
end`
```

`def method_name = value` (shorthand for simple methods)

Methods can have default argument values: `def method_name(arg1 : Type = default_value)`

Blocks & Procs

Blocks are anonymous functions passed to methods.

```
`method do |arg1, arg2|
```

code

```
end`
```

Procs are first-class blocks.

```
`my_proc = proc do |arg1, arg2|
```

code

```
end my_proc.call(value1, value2)`
```

Lambdas are similar to procs but enforce argument arity.

```
`my_lambda = ->(arg1, arg2) {
```

code

```
} my_lambda.(value1, value2)`
```

Classes & Modules

Class Definition

```
`class ClassName
```

code

```
end`
```

```
`class ClassName < SuperClass
```

code

```
end` (inheritance)
```

`initialize` method is the constructor.

```
Instance variables: @variable
```

Module Definition

```
`module ModuleName
```

code

```
end`
```

Modules can be used for namespacing and mixins.

`include ModuleName` - includes module methods as instance methods.

`extend ModuleName` - includes module methods as class methods.

Structs

```
struct Point property x : Int32 property y : Int32 end
```

Structs are value types (passed by value).