SwiftLang Language Reference Manual

This document summarizes the core syntax of SwiftLang, including its keywords, operators, and main statement forms. It is meant to be a quick reference rather than a full tutorial.

---

1. Lexical Structure

Identifiers

- Names for variables.
- Start with a letter or underscore, followed by letters, digits, or underscores.
- Examples: x, total_sum, _temp1.

Literals

- **Integer**: sequence of digits
    - Examples: 0, 10, 42
- **Float**: digits with a decimal point
    - Examples: 2.5, 3.14
- **String**: text enclosed in double quotes
    - Example: "hello"
- **Boolean**:
    - true, false
- **Null**:
    - null

Comments

- Single-line:
- // this is a comment
- Multi-line:
- /* comment spanning
- multiple lines */

Comments are ignored by the tokenizer and do not affect execution.

---

2. Keywords

The following words are reserved and cannot be used as identifiers:

- let – variable declaration
- if – start of a conditional
- else – alternate branch of a conditional
- while – looping construct
- print – output a value
- read – simple input into a variable
- true – boolean literal
- false – boolean literal
- null – null literal

Additional reserved words recognized by the tokenizer but **not all fully implemented yet** in the parser or interpreter:

- fun, return, for, switch, case, default, break, continue, try, catch, throw, thread

You should avoid using these words as variable names, even if they are not currently used in the language's grammar.

---

3. Operators

3.1 Arithmetic Operators

Used with numeric types (integers and floats):

- + : addition
- - : subtraction
- * : multiplication
- / : division
- % : remainder

Example:

```
let x = 10;

let y = 3;

let sum = x + y;

let ratio = x / y;
```

3.2 Comparison Operators

Produce a boolean result (true or false):

- == : equal
- != : not equal
- < : less than
- > : greater than
- <= : less than or equal
- >= : greater than or equal

Example:

```
let a = 5;

let b = 8;

let isSmaller = a < b;

let isEqual = a == b;
```

3.3 Boolean Operators

Used with boolean values:

- and
- or
- not

Example:

```
let done = false;

let hasTime = true;
```

let shouldContinue = hasTime and not done;

3.4 Grouping

Use parentheses to control precedence:

let result = (a + b) * c;

let check = (x > 10) and (y < 5);

---

4. Expressions

General expression syntax combines identifiers, literals, operators, and parentheses:

a + b * 2

(a + b) * c

x > 10 and not done

"hello " + name

- Arithmetic operators work on integers and floats.
- Comparison operators compare numeric or boolean values and produce true or false.
- Boolean operators combine true and false values.
- Strings can be printed and may be combined with other expressions depending on implementation details (the interpreter may convert values to strings when printing).

---

5. Statements

A SwiftLang program is a sequence of statements, each ending with a semicolon ; (except blocks, which use { ... }).

5.1 Variable Declaration

let identifier = expression;

Examples:

let x = 10;

let name = "Alice";

let flag = true;

- Declares a new variable and initializes it with the value of expression.
- The initial value determines the variable's type at that point.

5.2 Assignment

identifier = expression;

Examples:

x = x + 1;

flag = false;

name = "Bob";

- Updates an existing variable's value.
- The semantic analyzer tracks the current type of each variable.
- Types may change as new values are assigned (the language supports dynamic typing at the semantic level).

5.3 Print Statement

print(expression);

Examples:

print(x);

print("Result:");

print(x + y);

- Evaluates the expression and prints the result to standard output followed by a newline.

5.4 Read Statement

read(identifier);

Example:

let x = 0;

read(x);

- Reads a value from standard input and stores it in the given variable.

- (Runtime behavior can depend on the environment where the interpreter is run.)

5.5 If / Else Statement

Full form:

```
if (condition) {
   statement_list
} else {
   statement_list
}
```

Example:

```
if (x > 0) {
   print("positive");
} else {
   print("non-positive");
}
```

- condition must evaluate to a boolean value.
- If condition is true, the first block runs.
- If condition is false and an else block exists, the else block runs instead.

else is optional:

```
if (x == 0) {
   print("zero");
}
```

5.6 While Statement

```
while (condition) {
   statement_list
}
```

Example:

```
let i = 0;
while (i < 3) {
  print(i);
  i = i + 1;
}
```

- Evaluates condition before each iteration.
- If condition is true, executes the block and then checks condition again.
- Loop ends when condition becomes false.

---

6. Blocks and Statement Lists

A block of code is a sequence of statements enclosed in braces { and }:

```
{
  let x = 10;
  print(x);
}
```

Inside if, else, and while, you can write either:

- A **single statement**, or
- A **block** containing multiple statements.

Examples:

Single statement:

```
if (flag)
  print("single statement");
```

Block:

```
if (flag) {
  let x = 1;
  print(x);
```

}

---

7. Operator Precedence (Informal)

From **highest** to **lowest** precedence (operations at the top bind more tightly):

1. Parentheses: (...)
2. Unary: not, unary -
3. Multiplicative: *, /, %
4. Additive: +, -
5. Comparison: <, >, <=, >=
6. Equality: ==, !=
7. Boolean and
8. Boolean or

When in doubt, add parentheses to make the intended order explicit.

---

8. Example Putting It All Together

```
let x = 5;

let y = 2;


if (x > y and x < 10) {
   print(x + y * 2);
} else {
   print(0);
}
```

- Declares two integers x and y.
- Uses a combined condition with and and comparison operators.
- If the condition is true, it prints x + y * 2.

- Otherwise, it prints 0.