# LEARNING loops

#loops

# Table of Contents

**About** 1

**Chapter 1: Getting started with loops** 2

Remarks 2

Examples 2

Types of loops 2

**Condition loops** 2

For loops 2

While loops 2

Variants 2

**Collection loops** 2

Foreach loops 3

**Goto loops** 3

**Recursive loops** 3

**Chapter 2: For loops** 4

Syntax 4

Examples 4

General for loop 4

**Credits** 5

# About

# Chapter 1: Getting started with loops

## Remarks

This section provides an overview of what loops is, and why a developer might want to use it.

It should also mention any large subjects within loops, and link out to the related topics. Since the Documentation for loops is new, you may need to create initial versions of those related topics.

## Examples

### Types of loops

A loop is a control flow structure to definitely or indefinitely run a set of statement written only once in code, until a certain condition is met or the process is terminated.

## Condition loops

These loops are repeated based on the state of their conditions.

### For loops

For loops are usually run upon a variable as the subject of iteration. For example, for loops can be run upon an integer to limit the number of times the loop should be run, or upon an array to iterate over it.

### While loops

While loops is the most basic type of condition loop that keeps running until its condition is changed (or until a `break` statement is executed).

### Variants

A variant of `while` loops is the `do... while` loop. It is the same as `while` loops, except that the content of the loop is run once before checking the condition.

Another variant is the `until`/`do... until` loops, which does the same as their counterparts in `while` except that they check the condition in the opposite way -- `while` loops run until the condition is false, and `until` loops run until the condition is true.

## Collection loops

These loops are repeated by iterating upon collections, such as arrays or iterables.

### Foreach loops

A `forEach` loop runs on a collection by executing the code once per item in collection, giving the value and/or the key of the item as parameter.

## Goto loops

`goto` loops are a set of statement between a label and a goto statement.

## Recursive loops

In functional programming, recursive loops can be used to run a function recursively until a condition is met. This is a common cause for stack overflow errors.

Read Getting started with loops online: https://riptutorial.com/loops/topic/5080/getting-started-with-loops

# Chapter 2: For loops

## Syntax

- for(init; condition; increment){ content_code(); } // general syntax
- for(int i = 0; i < numberRuns; ++i){ actions_with(i); } // run an action for a numberRuns times
- for(int i = 0; i < sizeof(array); ++i){ actions_with(array[i]); } // iteration over an array

## Examples

### General for loop

Most programming languages support the `for`-loop control structure.

It is generally implemented in this way:

```
for(init; condition; increment){
    content_code();
}
```

The above pseudocode is identical with the pseudocode below:

```
init;
start_loop:
if(condition){
    content_code();
    increment;
    goto start_loop;
}
```

This shows that:

- `init` is run before the loop, used to initialize things for running that loop
  - In some programming languages like Java, variables can be *declared* in `init`, and the scope of the declared variables will be limited to that loop.
- `condition` is a condition to determine when the loop can be run. If this evaluates to `false`, the loop will stop executing.
- `increment` is usually a statement used to manipulate parameters used in `condition`, so when `increment` is run a certain number of times, `condition` becomes `false` and the loop breaks.
- `content_code()` is the core code to be run within the loop.

Read For loops online: https://riptutorial.com/loops/topic/6135/for-loops

# Credits

| S. No | Chapters | Contributors |
|---|---|---|
| 1 | Getting started with loops | Community, SOFe |
| 2 | For loops | SOFe |