

## 4. Loops

---

### Introduction

Let's make some pancakes!



So far we only looked at programs that have a fixed number of steps. For example look the *algorithm* to make a pancake:

- put 1/4 cup of batter in the frying pan
- cook for about 2 minutes
- flip and cook for another 2 minutes
- remove the pancake

How would the algorithm to make 10 pancakes would look? Would it be much different?

- ```
10 times do:  
  - put 1/4 cup of batter in the frying pan  
  - cook for about 2 minutes  
  - flip and cook for another 2 minutes
```

```
- remove the pancake
```

Loops let you describe repetitive processes. They could have a fixed amount of steps like the example above. Or they could have an unknown number of steps, for example a more realistic algorithm for making pancakes:

```
while you have pancake batter do:
  - put 1/4 cup of batter in the frying pan
  - cook for about 2 minutes
  - flip and cook for another 2 minutes
  - remove the pancake
```

## while

A `while` loop performs a set of statements until a condition becomes `false`.

```
while condition {
  statements
}
```

For example in order to `print` all the numbers from 1 to 10. We need to create a variable with the initial value of 1. Print the value and increase it by one and until it becomes bigger than 10.

```
var i = 1
while i <= 10 {
  print(i)
  i = i + 1
}
```

## repeat

`repeat` loops while a condition is met. The difference between a `while` and a `repeat` loop is that the `repeat` loop evaluates the condition after executing the statements from the loop.

```
repeat {
  statements
} while condition
```

```
var i = 1
repeat {
  print(i)
  i = i + 1
} while i < 10
```

Both `while` and `repeat` are best used in loops where the numbers of steps is unknown. Take for example the algorithm of converting a number to binary: divide the number by two until it becomes 0. Write the remainders from right to left to get the binary form of the number.

```
var number = 123

var binary = 0
var digit = 1

while number > 0 {
    let remainder = number % 2

    // add the new digit to the number
    binary = digit * remainder + binary

    // move the digit to the left
    digit *= 10

    // remove the last binary digit
    number /= 2
}

binary // 1111011
```

## for loops

Swift provides two kinds of loops that perform a set of statements a certain number of times:

The **for-in** loop performs a set of statements for each item in a range or collection.

Swift also provides two range operators `a..b` and `a..<b`, as a shortcut for expressing a range of values.

```
1...3 // 1, 2, 3
1..<3 // 1, 2
```

```
for value in range {
    statements
}
```

```
// prints 1-10
for i in 1...10 {
    print(i)
}

// prints 0-9
for i in 0..<10 {
    print(i)
}
```

The **for** loop performs a set of statements until a specific condition is met, typically by incrementing a counter each time the loop ends.

```
for initialization ; condition ; increment {
    statements
}
```

```
// ++i is shorthand for i = i + 1
// prints 1-10
for var i = 1; i <= 10; ++i {
    print(i)
}

// prints 10-1
for var i = 10; i > 0; --i {
    print(i)
}
```

## print and terminators

For the drawing exercises below you will need use the `terminator` parameter for the `print` function. The `terminator` refers to the thing that is printed at the end. The default terminator is the new line character `"\n"`.

- `print(value)` will print the value and a new line
- `print(value, terminator: "")` will print the value

```
print("BAT", terminator: "") // prints BAT
print("MAN", terminator: "") // prints MAN
print("") // prints a newline character
// BATMAN

print("BAT")
// BAT
print("MAN")
// MAN
```

## Executing a statement multiple times

Sometimes you just want to execute some statements multiple times but don't care about having an index. A swift convention in `for` loops is to use `_` as the loop variable name when you don't intend to use the variable in the loop.

For example to print "Hello World" 5 times you can use:

```
for _ in 1...5 {
```

```
    print("Hello World")  
}
```

Naming your loop variable `_` is useful because you immediately tell that the variable is not used in the loop.