

SWIFT TRAINING - WEEK 2-3

- 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 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 `lowerBound...upperBound` and `lowerBound.., as a shortcut for expressing a range of values.`

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

```
1..
```

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

```
}
```

```
// prints 1-10
```

```
for i in 1...10 {  
    print(i)
```

```
}
```

```
// prints 0-9
```

```
for i in 0..    print(i)
```

```
}
```

```
// this will crash - don't do it! :)
```

```
for i in 10...1 {  
    print(i)
```

```
}
```

If you want to loop on a range in reverse order you can use the `reversed` range method:

```
// this will print the numbers from 10 to 1
```

```
for i in (1...10).reversed() {  
    print(i)
```

```
}
```

STRIDE

`stride` is a function from the swift standard library that returns the sequence of values `start`, `start + stride`, `start + 2 * stride`, ... `end`) where `end` is the last value in the progression that is less than `end`.

The `stride` function works with any kind of number:

```
stride(from: 1, to: 10, by: 2) // 1, 3, 5, 7, 9
```

```
stride(from: 1, to: 2, by: 0.1) // 1.0, 1.1 ... 1.9
```

Let's take for example a program that counts from 1 to 10 by 3:

```
for i in stride(from: 1, to: 10, by: 3) {
    print(i)
}
```

You can use `stride` to create decreasing sequences if the `stride` parameter is negative:

```
for i in stride(from: 3, to: 1, by: -1) {
    print(i)
}
// prints: 3 2 1
```

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 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.

Exercises

3.1 Chalkboard

Write a program that writes "I will not skip the fundamentals!" N times. Complete this with a **while** and with a **for**.

```
var N = 10
```

```
// your code here
```

3.2 Squares

Print the first N square numbers. A square number, also called perfect square, is an integer that is obtained by squaring some other integer; in other words, it is the product of some integer with itself (ex. $1, 4 = 2 * 2, 9 = 3 * 3 \dots$).

```
// your code here
```

3.3 Powers of 2

Print the powers of 2 that are less than or equal to N .

```
var N = 10
```

```
var power = 2
```

```
// your code here
```

3.4 Alternate Counting

Write all the numbers from 1 to N in alternative order, one number from the left side (starting with one) and one number from the right side (starting from N downto 1).

For example if $N = 4$, you should write 1, 4, 2, 3. And for $N = 9$ you should write 1, 9, 2, 8, 3, 7, 4, 6, 5. Use **while** $left < right$ {

```
var N = 9
```

```
var left = 1
```

```
var right = N
```

3.5 Square

Given an integer N draw a square of $N \times N$ asterisks, like the ones below:

```
N = 1
*
```

```
N = 2
**
**
```

```
N = 3
```

```
***
***
***
```

3.5 Advanced - Use an array and display with N Sizes 1, 2, 3, 10

HINT: for N in [1, 2, 3, 10] {

3.6 Rectangle

Given two integers `row` and `column` draw a rectangle of `row` x `column` asterisks, like the ones below:

row = 1 column = 3

```
***
```

row = 2 column = 2

```
**
**
```

row = 3 column = 7

```
*****
*****
*****
```

Advanced - Use an array and display with different Sizes 1 x 3, 2 x 2, 3 x 7, 10 x 3

HINT: for (row, column) in [(1, 3), (2, 2), (3, 7), (10, 3)] {

3.7 Triangle

Given an integer `size` draw a triangle of asterisks. The triangle should have `size` lines, the *i*-th line should have *i* asterisks on it.

size = 10

```
*
**
***
****
*****
*****
*****
*****
*****
*****
```

Advanced - Use an array and display with different Sizes 1 x 3, 2 x 2, 3 x 7, 10 x 3

HINT: for Size in [1, 2, 3, 10] {

3.8 Pyramid - Evaluate the given code, test it and resolve how it functions.

Given an integer `N` draw a pyramid of asterisks, like the ones below:

`N = 10`

```

      *
     ***
    *****
   *********
  ***********
 *****
*****
*****
*****
*****
*****
*****
*****

```

```

var N = 10
print("N = \(N)")

for i in 1...N {
    for j in 0..<(N-i) {
        print(" ", Terminator: "")
    }

    for j in 1...2*i-1 {
        print(" ", Terminator: "")
    }
    print()
}

```

3.9 GCD

You are given two numbers `a` and `b`. Find the greatest common divisor of `a` and `b`. Evaluate the given code, test it and resolve how it functions.

```

var a = 24
var b = 12
var maxDiv = a

if b < maxDiv {
    maxDiv = b
}

var gcd = 1

for i in 1...maxDiv {
    if (a % i == 0) && (b % i == 0){
        gcd = i
    }
}

```

```
}  
}  
print(gcd)
```