**Control Statements in Swift**

Swift has multiple control statements. Some of them are:

1. **for … in … { statements}**

**→** It helps to iterate through an array. Example:

let fruits = ["Apple", "Banana", "Grapes"]

for fruit in fruits{

print("I love \(fruit)!")

}

Output:

I love Apple!

I love Banana!

I love Grapes!

1. **while (condition) {statements}**

**→** You can go through statements until the condition is true

Example:

var count = 0

while (count < 4){

print(count)

count+=1

}

Output:

0

1

2

3

1. **repeat {statements} while {condition}**

**→** It is same as while loop but in while loop the condition has to be true before running the statements but in this loop the statements doesn’t have to be true the first time.

Example:

var count = 5

repeat{

print(count)

count -= 1

}while (count<5 && count>=0)

Output:

5

4

3

2

1

0

1. **if (condition) {statements}**

**else if (condition) {statements}**

**else {statements}**

**→** It executes a set of statements only if the condition is true

Example:

var a\_string = "Woman in STEM"

if (a\_string.contains("C")){

print ("C for coding")

}

else if (a\_string.contains("W")){

print ("Women in STEM rock!")

}

else{

print ("We need more women in STEM")

}

Output:

Women in STEM rock!

**Multi-condition if else statements:**

var a\_number=4

if (a\_number>=0 && a\_number<=2){

print("The number is in between 0 and 2")

}

else if(a\_number>=3 && a\_number<=5){

print("The number is in between 3 and 5")

}

else{

print("The number is greater than 5")

}

Output:

The number is in between 3 and 5

1. **switch (value to consider){**

**case (value):**

**statement**

**default:**

**(otherwise, do something else)**

**}**

**→** It considers a value and compares it with multiple values. We don’t have to use continue or break to exit out of the loop once a case matches. After a case match and it goes through the statements of that case, it automatically leaves the switch statement.

Example:

var a\_number = 4

switch a\_number{

case 2:

print("The number is 2")

case 4:

print ("The number is 4")

case 6:

print("The number is 6")

default:

print("It's a number")

}

Output:

The number is 4

1. **continue** **→** leaves current loop iteration and goes to next iteration without leaving the loop itself

Example:

var count = 1

while (count<6){

if (count==4){

count += 1

continue

}

else{

print(count)

}

count += 1

}

Output:

1

2

3

5

1. **break** **→** leave the control statement immediately

Example:

var a\_number = 4

var count = 0

while(count<10){

if(count==a\_number){

break

}

count+=1

print(count)

}

Output:

1

2

3

4

1. **fallthrough** **→** The switch statement completes its execution as soon as the first matching case is completed. If you don’t want that to happen then you can use fallthrough at the end of every switch case.

Example:

var a\_number = 4

switch a\_number{

case 2:

print("The number is 2")

case 4:

print ("The number is 4")

fallthrough

case 6:

print("The number is 6")

default:

print("It's a number")

}

Output:

The number is 4

The number is 6

1. **return** **→** returns values

Example:

func hello(name: String) -> String{

return ("Hello there \(name)")

}

print (hello(name: "Pragyee"))

Output:

Hello there Pragyee

1. **throw** **→** can be used to throw errors

Swift uses short-circuit evaluation. Which means that it stops evaluating a statement as soon as an outcome is determined. If a statement has two conditions and one condition’s outcome is false then it doesn’t evaluate the second condition. For example:

if (2+3==4 && 3+3==6){

print ("If this prints, it’s wrong");

}

In the above code, if first checks if 2+3 is equal to 4, since that is false, it doesn’t test whether 3+3 is equal to 6 or not.

Since we use if else statement, there might be a possibility of ‘dangling else’ problem but since Swift requires us to have brackets for our if else statements, it is not as big of a problem.

The conditional statements and loops are very similar to other programming languages but one thing that we have in Python and Java and not in Swift is the for(int i=0; i<10; i++) loops. But, a while loop or the other for loop Swift has can be used to emulate the behavior.

References:

* <https://docs.swift.org/swift-book/LanguageGuide/ControlFlow.html>
* <https://medium.com/@cgoldsby/swift-short-circuit-evaluation-28995775fe21>