Swift mit Udemy

Switch case:

Syntax: **switch wert {**

**Case1:**

**Code**

**Case2:**

**Code**

**Default:**

**Break**

**}**

**Ich kann auch mit Vergleichsoperatoren arbeiten:**

**Nach dem Case:**

**Case \_ where Wart >/</== etwas**



For-Schleife: In Swift „for in“-Schleifen

Gut wenn man code öfters ausführen will!

Der Unterstrich**\_** in Swift symbolisiert, dass wir den Wert nicht benutzen werden.

Syntax: **for\_in 0…3 { code}**

*Kann aber auch Stadt dem Unterstrich was anderes Schreiben, was genutzt wird*

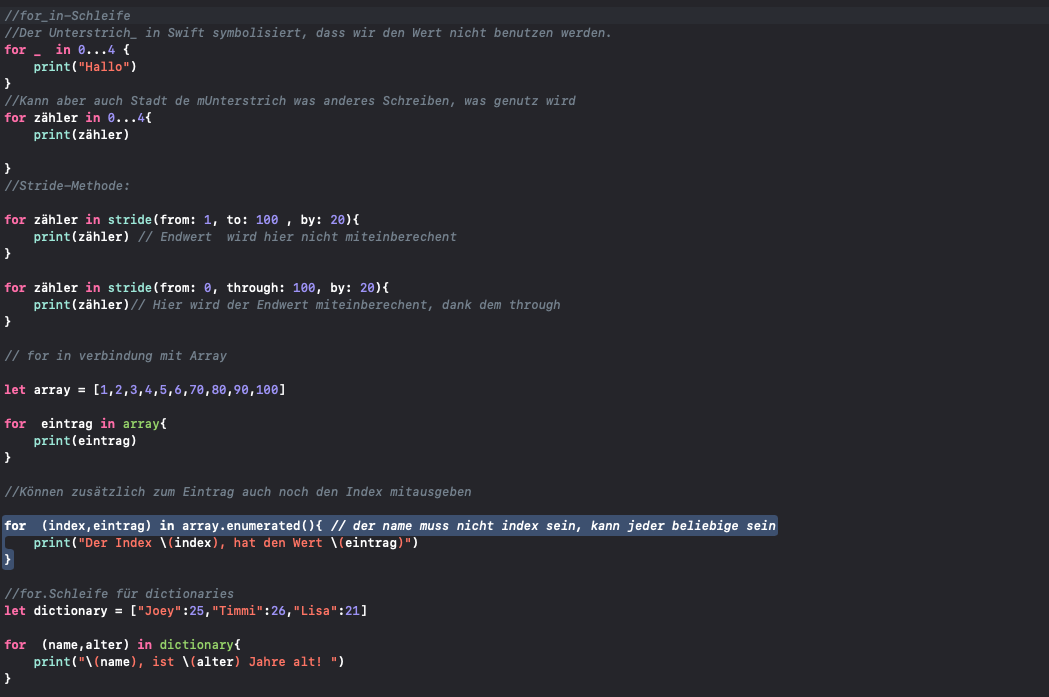
For zähler in 0…4{ print (zähler)}

Stride-Methode:

Geben Schrittweite fest, definieren, von welchem Wert wir starten, zu welchem Wert wir zählen wollen und in welchen Erhöhungen wir das machen wollen.

For zähler in stride (from: Anfang, to: Ende, by: Was möchte ich nach jedem Durchlauf dazuzählen)

Häufigste Anwendung einer >For-Schleife ist das iterieren(durchgehen) durch Arrays

Können zusätzlich zum Eintrag auch noch den Index mitausgeben. Mit arry.enumerated kann ich das Array sortieren

**While:**

Die do-while schleife heißt in switch repeat-while schleife, funktioniert aber sonst ganz gleich



Funktionen:

Wenn ich sie dynamisch machen will, brauche ich paramater, damit im Aufruf der Funktion dann der Parameter mit einem Wert übergeben werden kann. Bei mehreren Parametern werden diese durch ein Komma getrennt

**z.B.: func sagHallo(name:String){**

**print(“Hallo, \(name)”)**

**}**

**Beim aufruf:**

**sagHallo(„Timmy“)**

**Output: Hallo, Timmy**

Können für Parameter 2 Namen festlegen! Einen der in der Funktion steht u einen anderen, der dann beim Aufruf steht!

**Func multi (\_ zahl : Int, und andereZhal: Int){ }**

// der \_ sagt mir hier, dass ich später einen externen Namen für den Aufruf frei wählen kann und das und im 2. Parameter ist der Name, der im Aufruf zu sehen ist. Der 2. Name der Parameter ist der, der in der Funktion verwendet wird. beim Aufrufen, sieht man somit für den ersten Parameter keine Angaben.

**multipliziere(10, und: 20)**

Funktionen mit einen Rückgabewert sind praktisch, das da man sie gleich in einer Variable benutzen kann u so dort etwas ausrechen kann u den Wert gleich in die variable speichern kann. Bei einer Funktion ohne Rückgabewert, kann ich das Ergebnis zwar in eine Variable speichern, diese aber nicht ausgeben

**Func addiere(zahl1:Int, und zahlt2: Int) -> Int {**

**Return zahl1 + zahl2**

**}**

**Let summe = addiere(zahl1:20 und : 30)**

**//summer enthält jetzt den Wert 50**

String Interpolation ist so strak dass ich sogar alles in einem print-Statement aufrufen könnte.

**ar zahl12 = 10**

**var zahl22 = 20**

**print("Die Summe aus \(zahl12) und \(zahl22) ist \(multiplizieren2(zahl12, und: zahl22))")**

Eine Funktion kann aber auch mehrere Rückgabewerte haebn:

**func addierenundSubtrahiere(\_ zahl1: Int,und zahl2: Int)->(summe:Int,differenz: Int){**

**return (zahl1 + zahl2, zahl1 - zahl2)**

**}**

**var ergebnis = addierenundSubtrahiere(20, und: 10)**

**print(ergebnis)**

Kann auch direkt auf einen der beiden Rückgabewerte der Funktion zugreifen:

print("Die Summe ist \(ergebnis.summe) und die Differnenz ist \(ergebnis.differenz)")