BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

MAD2

Swift, Grundlagen

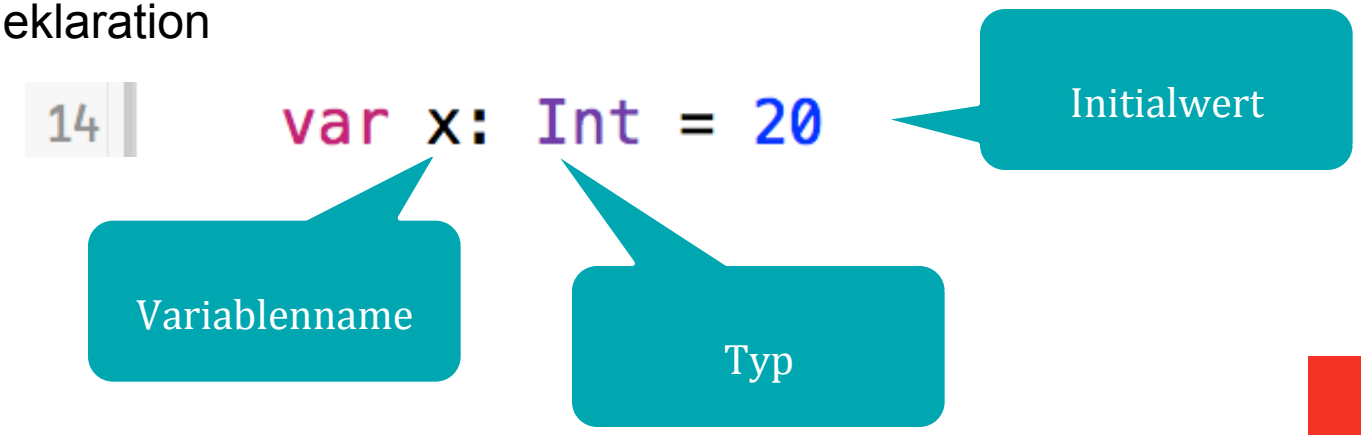
Prof. Dr. Dragan Macos

Typen in Swift

- Int
  - ganze Zahlen, mit oder ohne Vorzeichen
  - Alle Typen in Swift fangen mit Großbuchstaben an.

- UInt8, Uint16, UInt32, UInt64, Int8, Int16, Int32, Int64
  - ganze Zahlen in 8, 16, 32 und 64 Bit Form. Int ist abhängig von der Rechner-Plattform.

  - Typdeklaration



Typen in Swift

- Typdeklaration

```
14    var x: Int = 20
```

Typ

Initialwert

- Das würde auch so gehen

```
14    var x = 20
```

es wird eine neue Variable deklariert

..und gleich auf 20 gesetzt.

..also ihr Typ ist der Typ von 20 →Int.

- Diese Deduktion macht der Compiler.
- *Automatische Typinferenz*.

Optionale Typen in Swift

- Jede eingeführte Variable in Swift muss grundsätzlich initialisiert werden

- Befreiung von der Initialisierung: Optionale Typen
- Zu jedem Typ *T* gehört ein optionaler Typ *T?*

```
14      var x: Int?
```

x muss nicht initialisiert werden

- Bei optionalen Typen können wir abfragen, ob die Variablen gesetzt wurden

Optionale Typen in Swift

- Beispiel

Abfrage, ob x gesetzt ist.

```
24          x = 10;
25          if(x != nil)
26          {
27              println("x ist gesetzt")
28          }
29          else
30          {
31              println("x ist nicht gesetzt")
32          }
```

```
x ist gesetzt
```

Ausgabe

```
25          if(x != nil)
26          {
27              println("x ist gesetzt")
28          }
29          else
30          {
31              println("x ist nicht gesetzt")
32          }
```

```
x ist nicht gesetzt
```

Ausgabe

Wenn der Wert von x an keiner Stelle gesetzt wurde

Zugriff auf optionale Typen

```
x = 10;
if(x != nil)
{
    println("x ist gesetzt" + String(x!))

}
else
{
    println("x ist nicht gesetzt")
}
```

Zugriff auf den Wert mit „!“.
„Ich weiß, dass x gesetzt ist, Du kannst mir ruhig dessen Wert geben.“

„forced unwrapping“

Typen in Swift

- Int
  - ganze Zahlen, mit oder ohne Vorzeichen
  - Alle Typen in Swift fangen mit Großbuchstaben an

- UInt8, Uint16, UInt32, UInt64, Int8, Int16, Uint32, Int64
  - ganze Zahlen in 8, 16, 32 und 64 Bit Form.

Beispiel:

Würde es ohne Initialisierung gehen?

```
14
15      var meinAlter: Int = 47
16      var grafikCode: UInt32 = 0
17
```

Typen in Swift

- Int
  - ganze Zahlen, mit oder ohne Vorzeichen
  - Alle Typen in Swift fangen mit Großbuchstaben an
- UInt8, Uint16, UInt32, UInt64, Int8, Int16, Uint32, Int64
  - ganze Zahlen in 8, 16, 32 und 64 Bit Form.

Beispiel:

```
16      var meinAlter: Int?
17      var grafikCode: UInt32 = 0
```

Layout-Vorschläge

- Typen fangen mit Großbuchstaben an
- Variablennamen fangen mit Kleinbuchstaben an
  - Beispiel:

```
15      var alter = 50
```

- Wenn Variablennamen Zusammensetzungen mehrerer Worte sind: Jedes neue Wort soll mit einem Großbuchstaben anfangen
  - Beispiel

```
16      var meinAlter = 20
```

- „:“ bei der Typdeklaration soll am Variablennamen „kleben“, danach alles mit einem Leerzeichen trennen
  - Beispiel

```
17      var grafikCode: UInt32 = 0
```

Integer, Grenzen

```
1  let minValue = UInt8.min  // minValue is equal to 0, and is
       of type UInt8
2  let maxValue = UInt8.max  // maxValue is equal to 255, and
       is of type UInt8
```

Float, Double: Reale Zahlen

- Float 32 Bit
- Double 64 Bit

Typsicherheit

- Typprüfung zur Übersetzungszeit: Type Checking
- Swift erlaubt keine Zuweisungen und Ausdrücke mit nicht-kompatiblen Typen.
- Typinferenz (engl. Type Inference) bestimmt die Typen.

- Beispiel

```
1  let meaningOfLife = 42
2  // meaningOfLife is inferred to be of type Int
```

```
1  let pi = 3.14159
2  // pi is inferred to be of type Double
```

```
1  let anotherPi = 3 + 0.14159
2  // anotherPi is also inferred to be of type Double
```

Typkonversion (casting)

```
let twoThousand: UInt16 = 2_000
let one: UInt8 = 1
let twoThousandAndOne = twoThousand + UInt16(one)
```

Das ist erlaubt, damit man die Zahlkonstanten leichter lesen kann. Gleich wie 2000

Aufruf des Initialisierers des Typen UInt16, der von einem UInt8-Ausdruck einen UInt16-Typen erzeugt.
Klappt, weil der Uint16-Initialisierer mit einem UInt8-Parameter definiert ist.

Typ-Alias

```
typealias AudioSample = UInt16

var maxAmplitudeFound = AudioSample.min
```

Ein paar Fehlermeldungen

```
1  let cannotBeNegative: UInt8 = -1
2  // UInt8 cannot store negative numbers, and so this will
           report an error
3  let tooBig: Int8 = Int8.max + 1
4  // Int8 cannot store a number larger than its maximum value,
5  // and so this will also report an error
```

Logische Typen

- Bool
- Konstanten true und false

```
let orangesAreOrange = true
let turnipsAreDelicious = false
```

Bool wird in Konditionalen Befehlen (hier if) verwendet

```
if turnipsAreDelicious {
    println("Mmm, tasty turnips!")
} else {
    println("Eww, turnips are horrible.")
}
// prints "Eww, turnips are horrible."
```

Wo ist der Fehler?

```
let i = 1
if i {
    // this example will not compile, and will report an
        error
}
```

Tupel

- Gruppierung mehrerer Werte zu einem Wert
- Beispiel:

  (404, „Not Found“)
- ein zweistelliges Tupel vom Typ (Int, String)

```
let http404Error = (404, "Not Found")
// http404Error is of type (Int, String), and equals (404,
        "Not Found")
```

Tupel-Dekomposition

```
let http404Error = (404, "Not Found")
```

Beispiel 1

```
let (statusCode, statusMessage) = http404Error
println("The status code is \(statusCode)")
// prints "The status code is 404"
println("The status message is \(statusMessage)")
// prints "The status message is Not Found"
```

Beispiel 2

```
let (justTheStatusCode, _) = http404Error
println("The status code is \(justTheStatusCode)")
// prints "The status code is 404"
```

.. hat was mit „pattern matching“ zu tun. Darüber später

Benennung von Tupel-Komponenten

```
let http200Status = (statusCode: 200, description: "OK")
```

Externer Name der Tupel-Komponente

```
println("The status code is \(http200Status.statusCode)")
// prints "The status code is 200"
println("The status message is \
        (http200Status.description)")
// prints "The status message is OK"
```

Tupel weden oft verwendet, wenn Funktionen mehrere Werte liefern sollen

Tupel, einfacher Zugriff auf Elemente

```
var meinTupel = ("mama", "papa")
println("meinTupel .0= \(meinTupel.0)")
println("meinTupel .1= \(meinTupel.1)")
```

```
"meinTupel .0= mama"
"meinTupel .1= papa"
```

Optionale Variablenbindung

```
let possibleNumber = "123"

...

if let actualNumber = possibleNumber.toInt() {
    println("\(possibleNumber) has an integer value of \
        (actualNumber)")
} else {
    println("\(possibleNumber) could not be converted to an
        integer")
}
// prints "123 has an integer value of 123"
```

Außer in „if“-Anweisungen kann die optionale Bindung auch in „while“-Schleifen erfolgen.
.. es geht auch für Variablen...

Operatoren

- .. die meisten sind ähnlich wie in Java.
- Die Differenzen werden im Laufe des Kurses separat angesprochen.
- .. und jetzt ein paar Operatoren..

Nil Coalescing Operator

(a ?? b)

gleich wie

```
a != nil ? a! : b
```

...noch ein paar interessante Operatoren

- Closed Range operator

  (a ...b)

geschlossenes Intervall

```
for index in 1...5 {
    println("\(index) times 5 is \(index * 5)")
}
// 1 times 5 is 5
// 2 times 5 is 10
// 3 times 5 is 15
// 4 times 5 is 20
// 5 times 5 is 25
```

- Half-Open Range Operator

  (a..<b)

halboffenes Intervall

```
let names = ["Anna", "Alex", "Brian", "Jack"]
let count = names.count
for i in 0..<count {
    println("Person \(i + 1) is called \(names[i])")
}
// Person 1 is called Anna
// Person 2 is called Alex
// Person 3 is called Brian
// Person 4 is called Jack
```

String

- String und Zeichen in Swift sind Unicode-kompatibel
- Literale (Stringkonstanten)

```
let someString = "Some string literal value"
```

- Initialisierung

```
var emptyString = ""
var anotherEmptyString = String()
```

Initialisiert durch eine leere Stringkonstante

Initialisiert durch den Aufruf vom Initialisierer.

Sind die beiden Strings gleich??

Strings

- Konkatenation

```
var variableString = "Horse"
variableString += " and carriage"
```

- Geht das?

```
let constantString = "Highlander"
constantString += " and another Highlander"
```

Strings

- Konkatenation

```
var variableString = "Horse"
variableString += " and carriage"
```

- Geht das?

```
let constantString = "Highlander"
constantString += " and another Highlander"
```

Strings

- String ist ein „Value Type“.
- Ein String als Parameter eines Methodenaufrufs und als Wert, der einer Konstante oder Variablen zugewiesen wird, wird kopiert. Immer!
- großer Unterschied zu C und Objective-C

```
for character in "Dog!🐶" {
    println(character)
}
// D
// o
// g
// !
// 🐶
```

„Herausdestillieren“ von Charakter (Zeichen) aus dem String

```
let string1 = "hello"
let string2 = " there"
var welcome = string1 + string2
// welcome now equals "hello there"
```

Konkatenation

append

```
let exclamationMark: Character = "!"
welcome.append(exclamationMark)
// welcome now equals "hello there!"
```

String-Interpolation und –Vergleich

- Erzeugung eines Strings mit Hilfe von Werten von Variablen, Konstanten usw...

```
let multiplier = 3
let message = "\(multiplier) times 2.5 is \
        (Double(multiplier) * 2.5)"
// message is "3 times 2.5 is 7.5"
```

- Stringvergleich mit dem Operator „==“
- Viele andere Operatoren und Funktionen

  hasPrefix, *hasSuffix*, globale Funktion *countElements*, *unicodeScalars*, *value*, *utf8* ...

Arrays

- Collection-Typen
- Typen zur Verwaltung mehrerer Werte mit gleichen Typen
- Geordnet, *mutable* oder *immutable* (änderbar, nicht änderbar)
  - Als Variablen und Konstanten umsetzbar
- Syntax:

  Array<irgendeinTyp> oder *[irgendeinTyp]*
- Literale (Konstanten)

  [Wert1, Wert2, Wert3]
- Beispiel

```
var shoppingList: [String] = ["Eggs", "Milk"]

var shoppingList = ["Eggs", "Milk"]
```

Kurzform

Beispiel, Iteration über Array

```
var shoppingList = ["Eggs", "Milk"]


for item in shoppingList {
    println(item)
}
// Six eggs
// Milk
// Flour
// Baking Powder
// Bananas
```

Beispiel, Array-Operationen

```
var shoppingList = ["Eggs", "Milk"]
println("The shopping list contains \
        (shoppingList.count) items.")
// prints "The shopping list contains 2
        items."

shoppingList.insert("Maple Syrup", atIndex: 0)
let mapleSyrup = shoppingList.removeAtIndex(0)
let apples = shoppingList.removeLast()
```

was mach enumerate?

```
for (index, value) in enumerate(shoppingList) {
    println("Item \(index + 1): \(value)")
}
// Item 1: Six eggs
// Item 2: Milk
// Item 3: Flour
// Item 4: Baking Powder
// Item 5: Bananas
```

Wenn das die 5 Elemente des Arrays sind, gibt er sie so aus.

... noch ein bisschen spielen?

```
var threeDoubles = [Double](count: 3, repeatedValue: 0.0)
```

was ist das jetzt??

... noch ein bisschen spielen?

```
var threeDoubles = [Double](count: 3, repeatedValue: 0.0)
```

Der Initialisierer von Array hat eine Variante mit 2 Parametern
count: und *repeatedValue*:

Details kommen noch!

Keine Panik, wenn das unklar ist. Es kommt noch. Es wird glasklar sein.

Dictionary

- Datenstruktur für die Speicherung von Schlüssel-Wert-Paaren
- Typsyntax

  Dictionary<KeyType, ValueType>

oder

  [KeyType:ValueType]

- Beispiel

```
var airports: [String: String] = ["TYO": "Tokyo", "DUB":
        "Dublin"]
```

kürzer

```
var airports = ["TYO": "Tokyo", "DUB":
        "Dublin"]
```

Dictionary: Elemente einfügen

```
airports["LHR"] = "London"

airports["LHR"] = "London Heathrow"

 updateValue(forKey:)


   if let oldValue =
           airports.updateValue("Dublin
           International", forKey: "DUB") {
       println("The old value for DUB was \
           (oldValue).")
   }
   // prints "The old value for DUB was
           Dublin."
```

Wert geändert

Macht das Gleiche

Liefert den alten Wert,, wenn er im Dictionary vorhanden war

Iteration

```
for (airportCode, airportName) in
        airports {
    println("\(airportCode): \
        (airportName)")
}
// LHR: London Heathrow
// TYO: Tokyo
```

1

```
for airportName in airports.values {
    println("Airport name: \
        (airportName)")
}
// Airport name: London Heathrow
// Airport name: Tokyo
```

2

```
for airportCode in airports.keys {
    println("Airport code: \
        (airportCode)")
}
// Airport code: LHR
// Airport code: TYO
```

3

```
let airportCodes = [String]
        (airports.keys)
// airportCodes is ["LHR", "TYO"]

let airportNames = [String]
        (airports.values)
// airportNames is ["London Heathrow",
        "Tokyo"]
```

4

Erzeugung eines leeren Dictionary

```
var namesOfIntegers = [Int: String]()

namesOfIntegers[16] = "sixteen"
// namesOfIntegers now contains 1 key-
        value pair
namesOfIntegers = [:]
```

Wieder ein leeres Dictionary [Int, String]

Wichtig: Key-Typen von Dictionaries können nur Typen sein ,die „hashable“ sind!

Schleifen

- for-in

```
for index in 1...5 {
    println("\(index) times 5 is \(index
        * 5)")
}
// 1 times 5 is 5
// 2 times 5 is 10
// 3 times 5 is 15
// 4 times 5 is 20
// 5 times 5 is 25
```

Hier brauchen wir keinen Index

```
let base = 3
let power = 10
var answer = 1
for _ in 1...power {
    answer *= base
}
println("\(base) to the power of \
        (power) is \(answer)")
// prints "3 to the power of 10 is
        59049"
```

.. noch ein Beispiel

```
let numberOfLegs = ["spider": 8, "ant":
        6, "cat": 4]
for (animalName, legCount) in
        numberOfLegs {
    println("\(animalName)s have \
        (legCount) legs")
}
// spiders have 8 legs
// cats have 4 legs
// ants have 6 legs
```

Dictionary

Zweistelliges Tupel iteriert durch Dictionary. animalName nimmt den Wert von *key* und legCount von *value*.

For-Schleifen

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

```
for var index = 0; index < 3; ++index {
    println("index is \(index)")
}
```

While-Schleifen

While

```
while condition {
    statements
}
```

Do-While

```
do {
    statements
} while condition
```

Bedingte Verzweigungen

IF

```
temperatureInFahrenheit = 90
if temperatureInFahrenheit <= 32 {
    println("It's very cold. Consider
        wearing a scarf.")
} else if temperatureInFahrenheit >= 86 {
    println("It's really warm. Don't
        forget to wear sunscreen.")
} else {
    println("It's not that cold. Wear a
        t-shirt.")
}
// prints "It's really warm. Don't forget
        to wear sunscreen."
```

switch

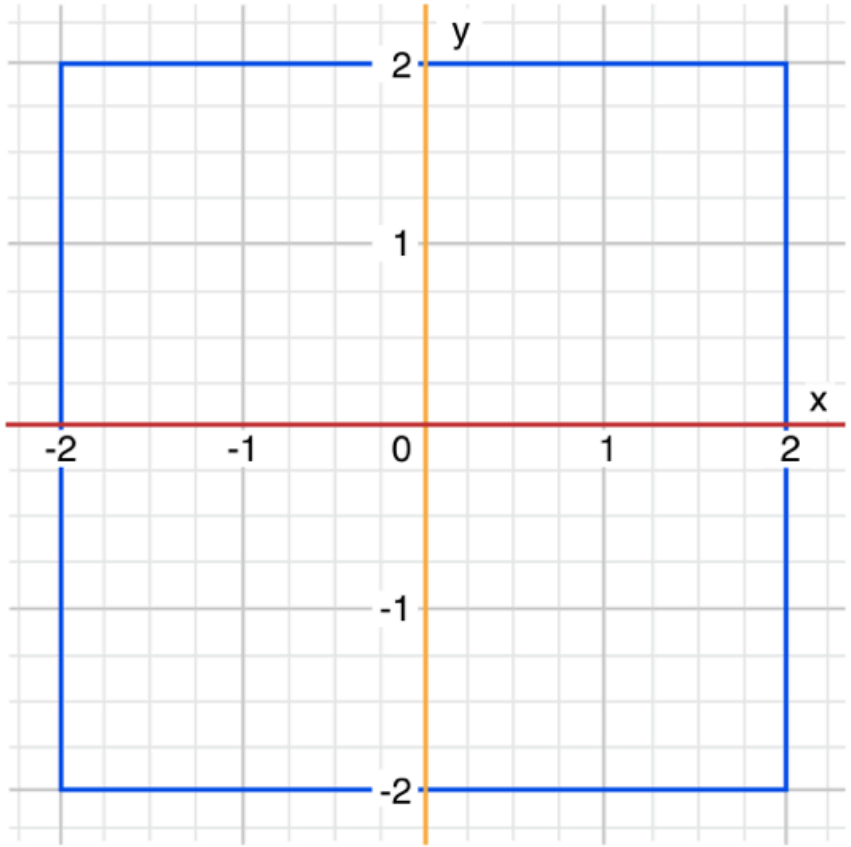
```
switch some value to consider {
case value 1:
    respond to value 1
case value 2,
value 3:
    respond to value 2 or 3
default:
    otherwise, do something else
}
```

Beispiel

```
let count = 3_000_000_000_000
let countedThings = "stars in the Milky
        Way"
var naturalCount: String
switch count {
case 0:
    naturalCount = "no"
case 1...3:
    naturalCount = "a few"
case 4...9:
    naturalCount = "several"
case 10...99:
    naturalCount = "tens of"
case 100...999:
    naturalCount = "hundreds of"
case 1000...999_999:
    naturalCount = "thousands of"
default:
    naturalCount = "millions and millions
        of"
}
println("There are \(naturalCount) \
        (countedThings).")
// prints "There are millions and
        millions of stars in the Milky
        Way."
```

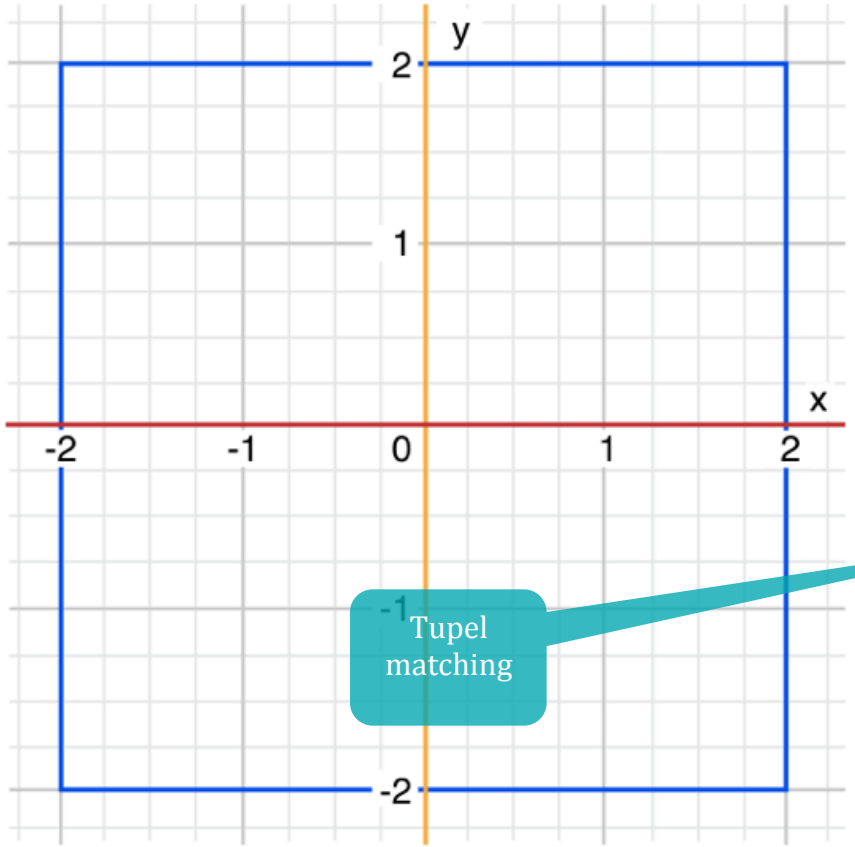
Generell: in Swift ist „break“nicht notwendig.

.. und noch ein Beispiel. Was macht das?




```
let somePoint = (1, 1)
switch somePoint {
case (0, 0):
    println("(0, 0) is at the origin")
case (_, 0):
    println("(\(somePoint.0), 0) is on
        the x-axis")
case (0, _):
    println("(0, \(somePoint.1)) is on
        the y-axis")
case (-2...2, -2...2):
    println("(\(somePoint.0), \
        (somePoint.1)) is inside the box")
default:
    println("(\(somePoint.0), \
        (somePoint.1)) is outside of the
        box")
}
// prints "(1, 1) is inside the box"
```

.. und noch ein Beispiel



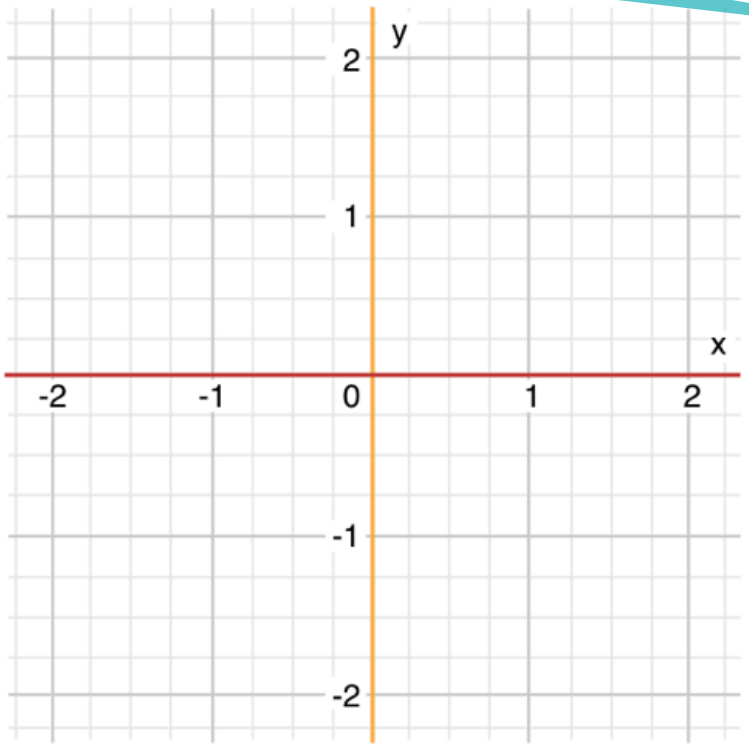
Tupel matching

Range matching

```
let somePoint = (1, 1)
switch somePoint {
case (0, 0):
    println("(0, 0) is at the origin")
case (_, 0):
    println("(\(somePoint.0), 0) is on
        the x-axis")
case (0, _):
    println("(0, \(somePoint.1)) is on
        the y-axis")
case (-2...2, -2...2):
    println("(\(somePoint.0), \
        (somePoint.1)) is inside the box")
default:
    println("(\(somePoint.0), \
        (somePoint.1)) is outside of the
        box")
}
// prints "(1, 1) is inside the box"
```

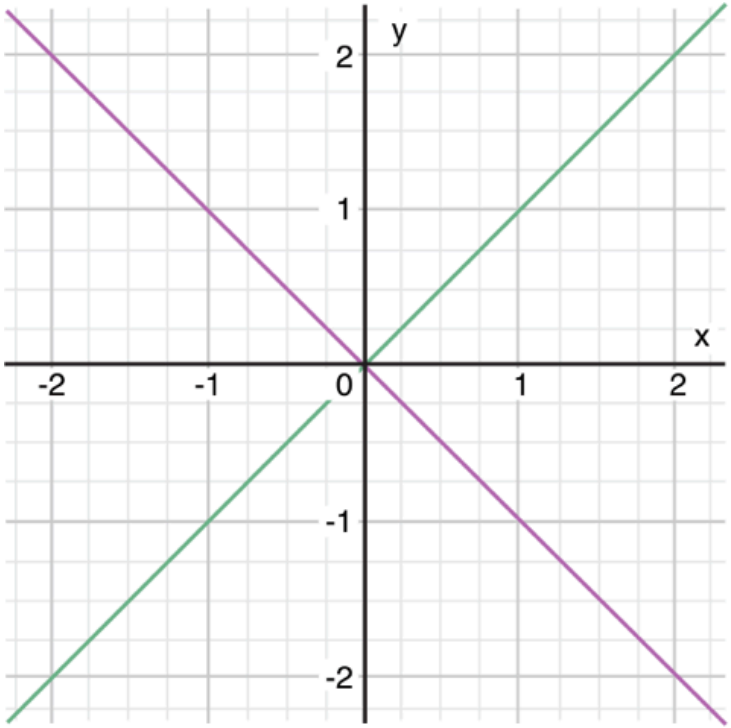
Bindung von Werten in Case

Die erste Komponente nennen wir „x“ und später verwenden wir diese Variable




```
let anotherPoint = (2, 0)
switch anotherPoint {
case (let x, 0):
    println("on the x-axis with an ›
        value of \(x)")
case (0, let y):
    println("on the y-axis with a y
        value of \(y)")
case let (x, y):
    println("somewhere else at (\
        (x), \(y))")
}
// prints "on the x-axis with an x
        value of 2"
```

case mit where




```
let yetAnotherPoint = (1, -1)
switch yetAnotherPoint {
case let (x, y) where x == y:
    println("(\(x), \(y)) is on the
        line x == y")
case let (x, y) where x == -y:
    println("(\(x), \(y)) is on the
        line x == -y")
case let (x, y):
    println("(\(x), \(y)) is just
        some arbitrary point")
}
// prints "(1, -1) is on the line x
        == -y"
```

Programfluss-Steuerung in Schleifen

- continue
- break
- fallthrough
- return

wir mögen das nicht so....

Demo Playground

Aufgaben

1. Schreiben Sie ein Swift-Programm, das aus einem [String] einen [(Int, String)] erzeugt, wobei die Zahl Int die Anzahl der Vorkommen eines Strings im Array darstellt.

   Beispiel: Aus [„Peter“, „Willi“, „Peter“] wird
   [(2, „Peter“), (1, „Willi“)] erzeugt.

Die beiden Arrays können Sie als globale Variablen deklarieren und initialisieren, z. B.:

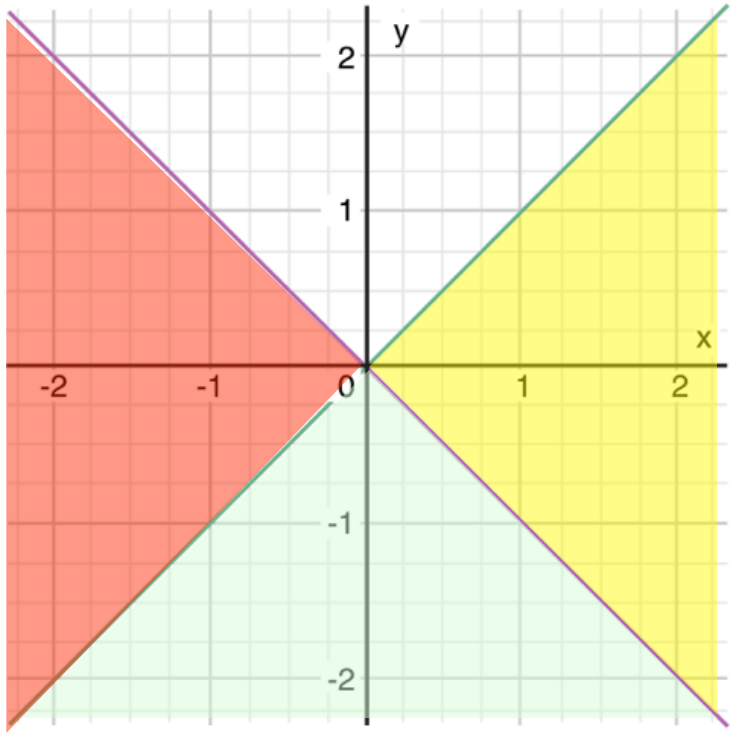
```
var strArray: [String] = ["Peter", "Willi", "Peter"]
var arrayTupel: [(Int, String)]?
```

Die Aufgabe können Sie auch im Playground lösen.

2. Lösen Sie die Aufgabe 1 mit einem Dictionary. ☺

Aufgaben

- Ermitteln Sie mit Hilfe eines case-Befehls, ob ein Punkt im roten, grünen, gelben oder keinem der aufgelisteten Bereiche ist (s. Aufgabe auf der Seite 50)



Quellen

Die meisten Sourcecode-Beispiele und die Sprachdefinition der Sprache Swift wurden aus:

pple Inc. „The Swift Programming Language.“ iBooks. https://itun.es/de/jEUH0.l

genommen.

Eventuelle andere Quellen bzw. eigene Beispiele werden an den entsprechenden Stellen direkt angegeben bzw. gekennzeichnet.