Scala Basics

Was ist der Unterschied zwischen einer Variablendeklaration mit val und var?

- `val` ist eine konstante Referenz, die nicht geändert werden kann
- `var` ist eine variable Referenz, die geändert werden kann

Was ist Currying in Scala?

Dass Funktionen mehrere Argumentenlisten haben können

```
def product(a: Int)(b: Int) = a * b
val p = product(3)(4)
println(p) // 12

val twice : Int => Int = product(2)
val t = twice(5)
println(t) // 10
```

Wie kann man das Problem lösen, dass ein Argument nicht evaluiert werden soll, wenn die Funktion aufgerufen wird und wie macht man das in Scala?

Mögliche Lösung:

- man kann statt dem Argument eine Funktion übergeben, die das Argument zurückgibt

Ziel bei dem Beispiel: `predicate` soll nur evaluiert werden, wenn `assertionEnabled` true ist.

```
def myAssert(predicate: () => Boolean) =
    if (assertionEnabled && !predicate())
        throw new AssertionError
myAssert(() => ref != null)
```

Bessere Lösung: mit => für by-name Parameter

```
def myAssert(predicate: => Boolean) =
    if (assertionEnabled && !predicate)
        throw new AssertionError
myAssert(ref != null)
```

Was ist eine PartialFunction in Scala?

= Funktion, die nicht für alle Eingaben definiert ist, benutzt Pattern-Matching um zu prüfen, ob die Funktion für einen bestimmten Eingabewert definiert ist

```
val squareRoot: PartialFunction[Double, Double] = {
    case x if x >= 0 => Math.sqrt(x) // Nur für x >= 0 definiert
}

println(squareRoot.isDefinedAt(4))   // true
println(squareRoot.isDefinedAt(-4))  // false
```

Was macht das implicit Schlüsselwort in Scala?

- implizite Parameter und Werte werden automatisch in den Kontext eingefügt, wenn sie benötigt werden

```
def printToConsole(s: String)(implicit prompt: String) = println(prompt + " " +
s)
implicit val p = "=>"
printToConsole("hello")("->") // -> hello
printToConsole("world") // => world
```

Was kann man in Scala3 statt implicit verwenden?

- `given` und `using`

```
trait Show[A] {
  def show(a: A): String
}

// Ein `given`-Wert für den Typ `Int`
given Show[Int] with {
  def show(a: Int): String = s"Int: $a"
}

// Eine Funktion, die `Show[A]` benötigt
def printValue[A](a: A)(using showInstance: Show[A]) =
  println(showInstance.show(a))

@main def run() =
  printValue(42) // Nutzt automatisch den `given Show[Int]`
```

Wie macht man eine extesion method in Scala?

- `extension`-Schlüsselwort

```scala
extension (s: String) {
  def toInt: Int = s.toInt
}

val str = "123"
val num = str.toInt // Nutzt die extension method
println(num) // 123
```

Was gibt es bei Scala und Java für Unterschied in Hinsich auf Klassen und Objekte?

- Member sind public by default
- es gibt Operator Overloading
- Methoden können default Werte haben
- Scala erlaubt keine static Member -> stattdessen Companion Objects

Was ist ein Companion Object in Scala?

= ein Objekt, das die gleiche Name wie die Klasse hat und in der gleichen Datei definiert ist

- es kann auf private Member der Klasse zugreifen
- stellt Funktionen zur Verfügung, die auf die Klasse angewendet werden können

```scala
class Person(val name: String, val age: Int) {
  // Instanzmethoden
  def greet(): Unit = {
    println(s"Hello, my name is $name and I am $age years old.")
  }
}

object Person {
  // Companion Object für die Klasse "Person"
  def apply(name: String, age: Int): Person = {
    // Eine Fabrikmethode, um eine Instanz der Klasse "Person" zu erstellen
    new Person(name, age)
  }

  def canVote(age: Int): Boolean = {
    age >= 18
  }
}

// Verwendung des Companion Objects
val person = Person("Alice", 30) // Aufruf der apply-Methode im Companion
Object
person.greet()  // Ausgabe: Hello, my name is Alice and I am 30 years old.

println(Person.canVote(30))  // Ausgabe: true
```

Was ist der Unterschied zwischen einer Extension-Methode und einer Companion-Methode?

- Extension-Methode: wird an eine bestehende Klasse angehängt, um neue Methoden hinzuzufügen, ohne die Klasse selbst zu ändern
- Companion-Methode: ist eine Methode im Companion Object einer Klasse, die auf Instanzen dieser Klasse angewendet werden kann und auf private Member der Klasse zugreifen kann

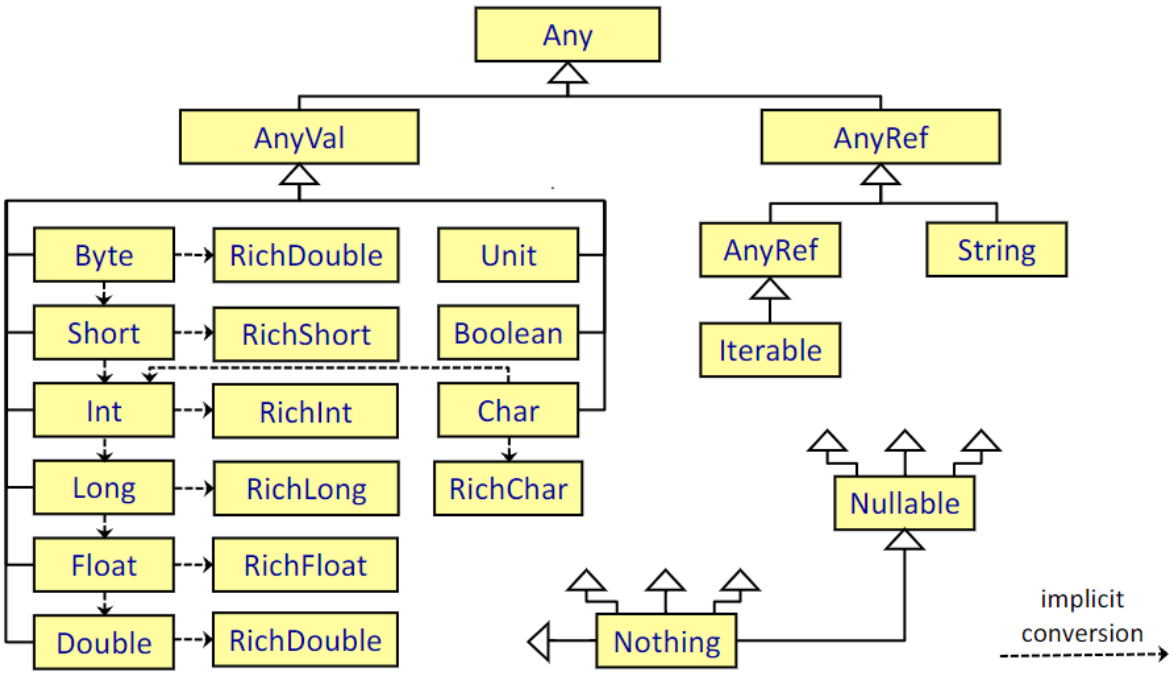
Wie macht man eine Singleton-Klasse in Scala?

- mit dem `object`-Schlüsselwort (wie Companion Object, aber ohne zugehörige Klasse)

Was können Case Classes in Scala?

- sie sind unveränderlich (immutable)
- sie haben automatisch `equals`, `hashCode`, `toString` und `copy` Methoden

Wie sieht die Typhierarchie in Scala aus?



Wie kann man ein Interface in Scala definieren?

- mit dem `trait`-Schlüsselwort

```scala
trait Animal {
  def speak(): Unit  // Abstrakte Methode
}

class Dog extends Animal {
  def speak(): Unit = println("Woof!")  // Implementierung der Methode
}
```

```
val dog = new Dog
dog.speak()  // Ausgabe: Woof!
```

Wie deklariert man Kovarianz und Kontravarianz in Scala?

- Kovarianz: `+T` (z.B. `List[+A]`) -> bedeutet, dass eine Subklasse von `A` auch als Subtyp von `List[A]` betrachtet werden kann
- Kontravarianz: `-T` (z.B. `Function1[-A]`) -> bedeutet, dass eine Subklasse von `A` auch als Supertyp von `Function1[A]` betrachtet werden kann