consol
Enterprising IT

IT Island Hopping - From Java to Kotlin

Tobias Schneck / Simon Hofmann

21.02.2018

Agenda

- Introduction
- Maven Setup
- Kotlin
    - Variables, Constants
    - Functions
    - Classes
    - Optionals
    - Control Flow
    - Extension Functions
    - Lambdas

Introduction

Why Kotlin?

Inter-operable

Concise

Safe

Tool-friendly

Where to use?

JVM **Android** **Browser** **Native**

Maven Setup

```xml
<dependencies>
   <dependency>
      <groupId>org.jetbrains.kotlin</groupId>
      <artifactId>kotlin-stdlib</artifactId>
      <version>${kotlin.version}</version>
   </dependency>

   <!--TEST Dependencies-->
   <dependency>
      <groupId>org.jetbrains.kotlin</groupId>
      <artifactId>kotlin-test-junit</artifactId>
      <version>${kotlin.version}</version>
      <scope>test</scope>
   </dependency>
   <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>${junit.version}</version>
      <scope>test</scope>
   </dependency>
</dependencies>
```

```xml
<build>
   <sourceDirectory>${project.basedir}/src/main/kotlin</sourceDirectory>
   <testSourceDirectory>${project.basedir}/src/test/kotlin</testSourceDirectory>
   <plugins>
      <plugin>
         <artifactId>kotlin-maven-plugin</artifactId>
         <groupId>org.jetbrains.kotlin</groupId>
         <version>${kotlin.version}</version>
         <configuration/>
         <executions>
            <execution>
               <id>compile</id>
               <phase>compile</phase>
               <goals>
                  <goal>compile</goal>
               </goals>
            </execution>
           <!--TEST -->
            <execution>
               <id>test-compile</id>
               <phase>test-compile</phase>
               <goals>
                  <goal>test-compile</goal>
               </goals>
            </execution>
         </executions>
      </plugin>
   </plugins>
</build>
```

Variables, Constants

```java
String myVariable = "Change me";
final String myConstant = "Can't change me";
```

```kotlin
var myVariable: String = "Change me‟
val myConstant: String = "Can't change me"
```

```java
String myVariable = "Change me";
final String myConstant = "Can't change me";
```

```kotlin
var myVariable: String = "Change me“
val myConstant: String = "Can't change me"
```

```java
String myVariable = "Change me";
final String myConstant = "Can't change me";
```

```kotlin
var myVariable = "Change me“
val myConstant = "Can't change me"
```

Functions

Functions

```
fun bark(times: Int): String {
        return "Wuff".repeat(times)
   }
```

Functions

Function keyword

Return type

```
fun bark(times: Int): String {
      return "Wuff".repeat(times)
  }
```

Functions

Default value

```
fun bark(times: Int = 1): String {
        return "Wuff".repeat(times)
   }
```

Functions

```
fun bark(times: Int): String {
        return "Wuff".repeat(times)
   }
```

Functions

```
fun bark(times: Int = 1) = "Wuff".repeat(times)
```

Function Invocation

```
dog.bark()
dog.bark(3)
```

Function Invocation

```
dog.bark()
dog.bark(3)
dog.bark(times = 3)
```

Classes and Inheritance

Classes

```java
public class JavaDTO{
    private int id;
    private String name;

    public JavaDTO(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(final String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

```kotlin
class KotlinDTO(var id: Int, var name: String)
```

Classes

```
class KotlinDTO(var id: Int, var name: String)
```

Classes

Primary Constructor

```kotlin
class KotlinDTO(var id: Int, var name: String)
```

Classes

```
class KotlinDTO(var id: Int, var name: String) {
    constructor(json: JSONObject) :
            this(json.getInt(”id"), json.getString(”name")) {

    }
}
```

Has to call
Primary Constructor

Secondary
Constructor

Classes

```
class KotlinDTO(var id: Int, var name: String) {
  override fun toString() {
    return ”ID: $id, Name: $name"
  }
}
```

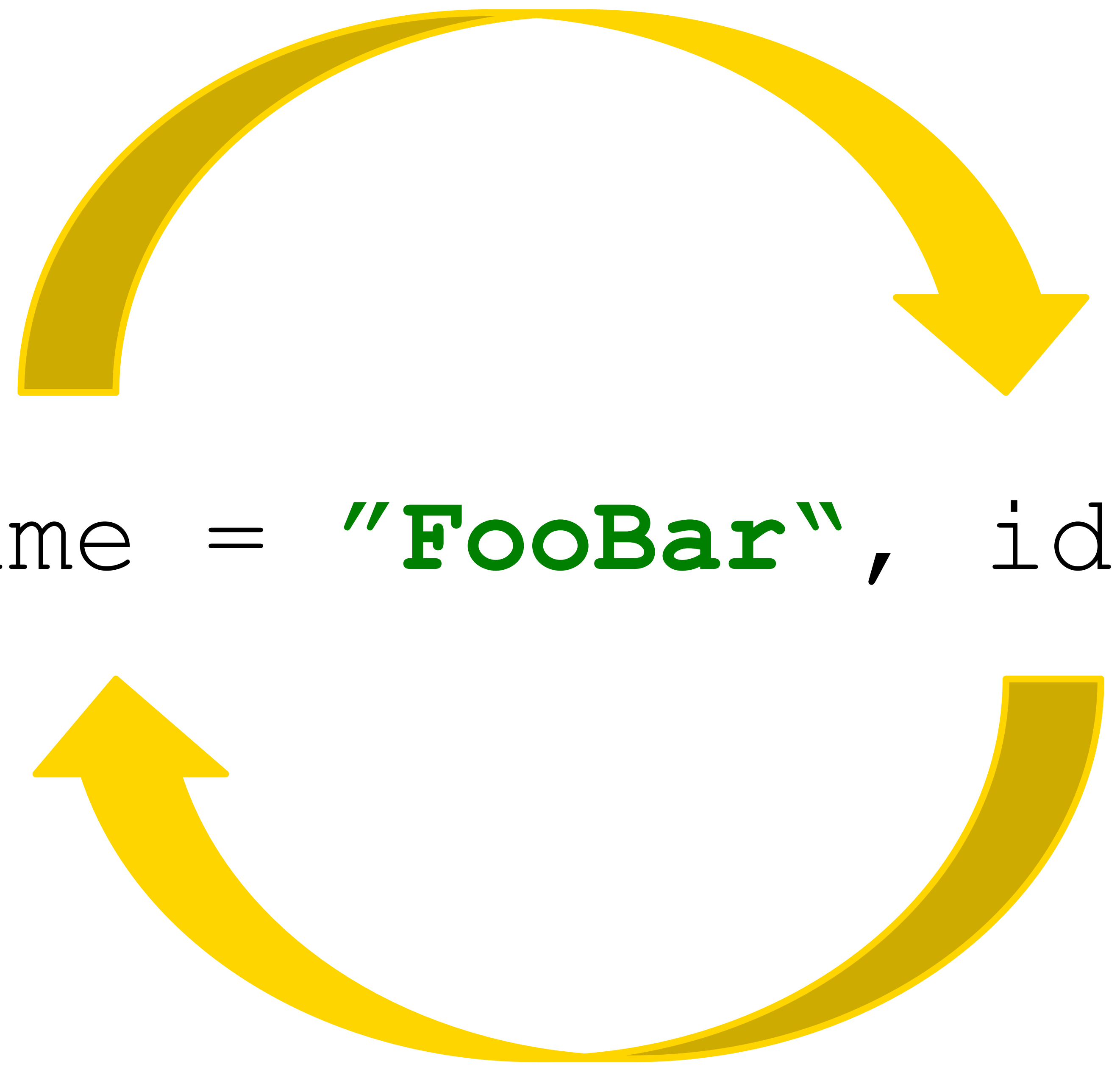
String template

Classes

No more “new”

```
val dto = KotlinDTO(1, ″First DTO")
```

Classes



```
val secondDTO = KotlinDTO(name = ”FooBar“, id = 10)
```

Getters and Setters

```
class Person(var name: String, var age: Int)
```

Default getters and setters are generated

Getters and Setters

```
class Person(name: String, age: Int) {
   var name = name
      set(value) {
         // Special logic
      }
      get() = “ConSoli $field”
   var age = age
}
```

Property Access

```
val human = Person()
human.name = ”Simon“
val personName = human.name
```

Optionals

Java: Nullpointer exceptions

```
public void sayHello(Dog otherDog){
    System.out.println(“Wuff, "+ otherDog.getName());
}
```

Java: Nullpointer exceptions

```
public void sayHello(Dog otherDog){
    System.out.println(“Wuff, "+ otherDog.getName());
}

sayHello(null);
```

Java: Nullpointer exceptions

```
public void sayHello(Dog otherDog){
    System.out.println(“Wuff, "+ otherDog.getName());
}

sayHello(null);
```

Kotlin: Built-in null-safety

```
fun sayHello(otherDog: Dog) {
    println("Hi, " + dog.name);
}
```

Kotlin: Built-in null-safety

```
fun sayHello(otherDog: Dog) {
    println("Hi, " + dog.name);
}

sayHello(null);
```

Kotlin: Built-in null-safety

```
fun sayHello(otherDog: Dog) {
    println("Hi, " + dog.name);
}

sayHello(null);
```

Kotlin: Built-in null-safety

```
var dog: Dog = null
```

Explicit nullable types

```
var dog: Dog? = null
```

Calling Optionals

```
val dog: Dog? = findDog()

dog.bark()
```

Calling Optionals

```
val dog: Dog? = findDog()

if(dog != null){
    dog.bark(3)
}
```

Safe Call Operator

```
val dog: Dog? = findDog()

dog?.bark()
```

For all NPE lovers (The not-null assertion operator)

```
val dog: Dog? = findDog()

dog!!.bark()
```

Elvis Operator

```
val foundDog = findDog()
val myDog = if (foundDog != null) {
    foundDog
} else {
    buyNewDog()
}
```

Elvis Operator

```kotlin
val myDog = findDog() ?: buyNewDog()
```

Elvis Operator

```kotlin
val myDog = findDog() ?: buyNewDog()
```

Automatic casting

```
if(dog != null){
    dog.bark(3)
}

val anyObject: Any = getAnimal()
if(anyObject is Dog){
    anyObject.bark()
}
```

Time to hack!

Time to hack!

On 2_Optionals.kt

Solution

```kotlin
private fun letDogBark(dog: Dog?) {
//      TODO TASK 1
        dog?.bark()
}

private fun getDogName(dog: Dog?): String {
//      TODO TASK 2
        return dog?.name ?: "No dog found"
}

private fun getNameOf(any: Any): String {
//      TODO TASK 3
        return (any as? Dog)?.name ?: "type unknown"
}
```

Control Flow

For Loop

```kotlin
val dogs = getDogs()

for (dog in dogs){
    dog.bark()
}
```

For Loop

```
val dogs = getDogs()

for (index in 0..10){
    dogs[index].bark()
}
```

Range Operator

For Loop

```
for (index in 0..10){
    print(index)
}

>>> 012345678910
```

Ranges are inclusive

For Loop

```
for (index in 10 downTo 1){
      print(index)
}

>>> 10987654321
```

When **switch** becomes more powerful

```
fun getDogPluralString(dogCount: Int): String {




}
```

When **switch** becomes more powerful

```
fun getDogPluralString(dogCount: Int): String {
    when(dogCount){




    }
}
```

When **switch** becomes more powerful

```
fun getDogPluralString(dogCount: Int): String {
    when(dogCount){
        0 -> return ”No dogs"
        1 -> return ”One dog"
        else -> return "$dogCount dogs"
    }
}
```

When **switch** becomes more powerful

```
fun getDogPluralString(dogCount: Int): String {
    when(dogCount){
        0 -> return ”No dogs"
        1 -> return ”One dog"
        else -> return "$dogCount dogs"
    }
}
```

When **switch** becomes more powerful

```
fun getDogPluralString(dogCount: Int): String {
    return when(dogCount){
        0 -> ”No dogs"
        1 -> ”One dog"
        else -> "$dogCount dogs"
    }
}
```

When **switch** becomes more powerful

```
fun getDogPluralString(dogCount: Int): String {
    return when(dogCount){
        0 -> ”No dogs"
        1 -> ”One dog"
        else -> "$dogCount dogs"
    }
}
```

When **switch** becomes more powerful

```
fun getDogPluralString(dogCount: Int) = when (dogCount) {
    0 -> ”No dogs"
    1 -> ”One dog"
    else -> "$dogCount dogs"
}
```

When **switch** becomes more powerful

```
fun getDogPluralString(dogCount: Int) = when (dogCount) {
    0 -> ”No dogs"
    1 -> ”One dog”
    2..4 -> “Many dogs
    else -> ”Too many dogs"
}
```

When without argument

```kotlin
fun evaluatePassword(password: String): String {
    return when {




    }
}
```

When without argument

```kotlin
fun evaluatePassword(password: String): String {
    return when {
        password.isEmpty() -> "Please enter password"
        password.length < 5 -> "Password not long enough"
        !password.containsNumber() -> "Password must contain a number"
        else -> "Password valid"
    }
}
```

consol

Time to hack!

Time to hack!

On 3_ControlFlow.kt

Solution

```kotlin
//  TODO TASK 1
fun findDogOwnerName(dog: Dog): String? {
    return when(dog.name){
        "Bruno" -> "Hans"
        "Ignatz" -> "Peter"
        else -> null
    }
}

//  TODO TASK 2
fun ageToString(dog: Dog): String {
   return when(dog.age){
       0,1 -> "Baby Dog"
       in 2..8 -> "Normal Dog"
       else -> "Old Dog"
   }
}
```

Extensions

Extension Functions

```
fun Int.isEven(): Boolean {
    return this % 2 == 0
}
```

Extension Functions

```
fun Int.isEven(): Boolean {
    return this % 2 == 0
}



println(1.isEven()) // false
println(2.isEven()) // true
println(3.isEven()) // false
```

Time to hack!

Time to hack!

On 4_Extensions.kt

Solution

```kotlin
// TODO TAKS 1
fun String.scream(): String {
    return this.toUpperCase()+"!!!"
}

// TODO TASK 2
private fun applyAllCapsExtension(text: String): String {
    return text.scream()
}

@Test
fun testAllCapsDogLanguage() {
// TODO TASK 3
    val allCapsDogLanguage = "Ich habe ganz viel Hunger".barkify().scream()

    assertEquals("WUFF WUFF WUFF WUFF WUFF!!!", allCapsDogLanguage)
}
```

Lambdas

Lambda Functions

```
val dog = Dog("Bruno")

val bark = dog::bark

bark(times = 3)
```

Lambda Functions

```kotlin
val dog = Dog("Bruno")

val greetDog: (Dog) -> String = { dog -> "Hey! ${dog.name}" }

println(greetDog(dog))
```

Lambda Functions

Parameter type

Return type

```
val dog = Dog("Bruno")

val greetDog: (Dog) -> String = { dog -> "Hey! ${dog.name}" }

println(greetDog(dog))
```

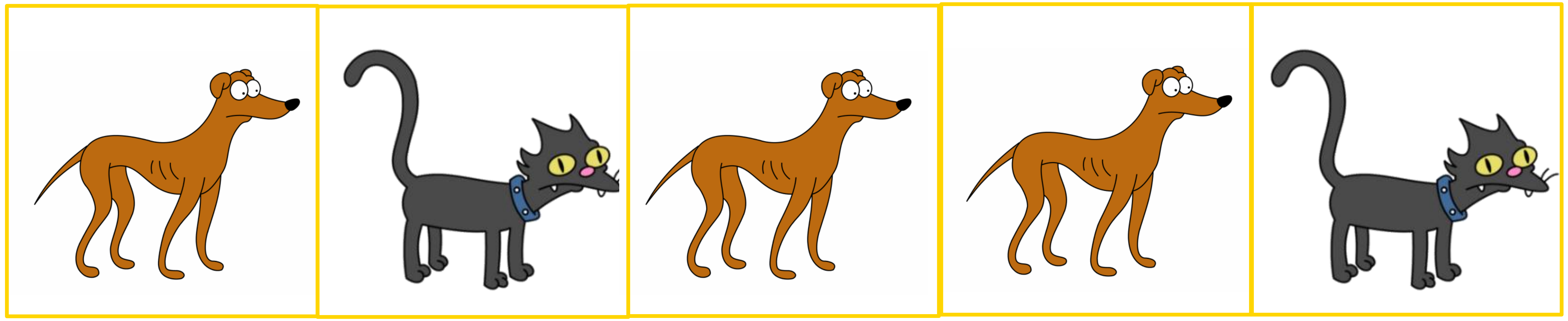
Lambda Functions

Parameter name

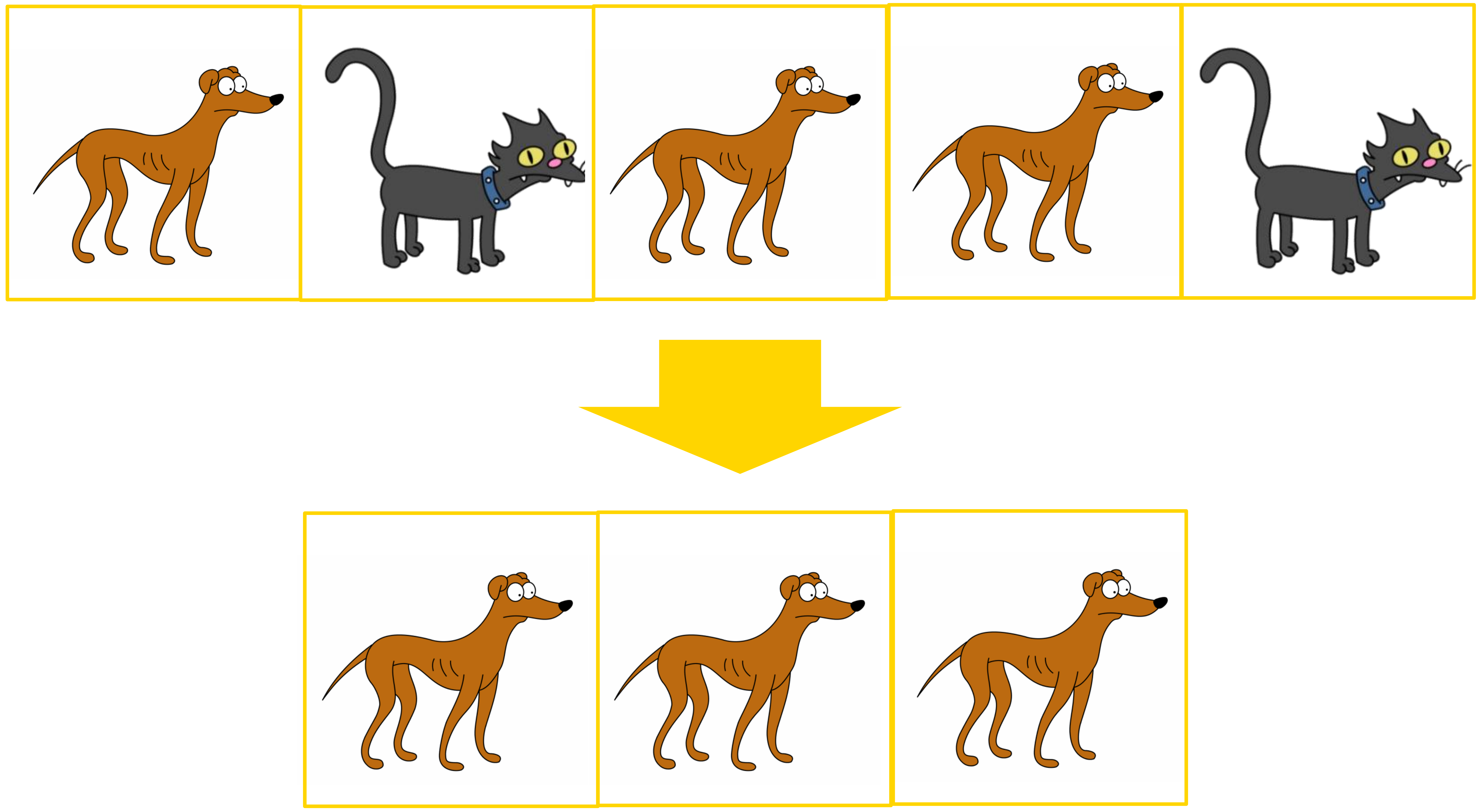
Lambda Body

```
val dog = Dog("Bruno")

val greetDog: (Dog) -> String = { dog -> "Hey! ${dog.name}" }

println(greetDog(dog))
```

Filter Functions



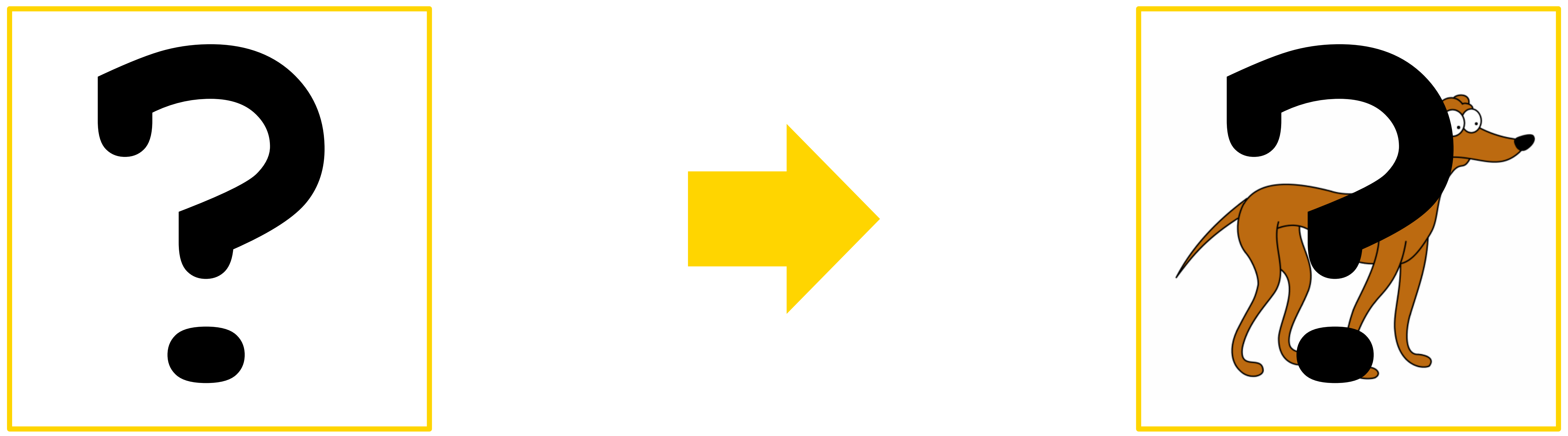
Filter Functions



Filter Functions

```
var dogs: List<Dog> = animals.filter(
        …
)
```

Filter Predicate



```
{ animal -> animal is Dog }
```

Filter Functions

```
var dogs = animals.filter({ animal ->
    animal is Dog
})
```

Filter Functions

```
var dogs = animals.filter(){ animal ->
    animal is Dog
}
```

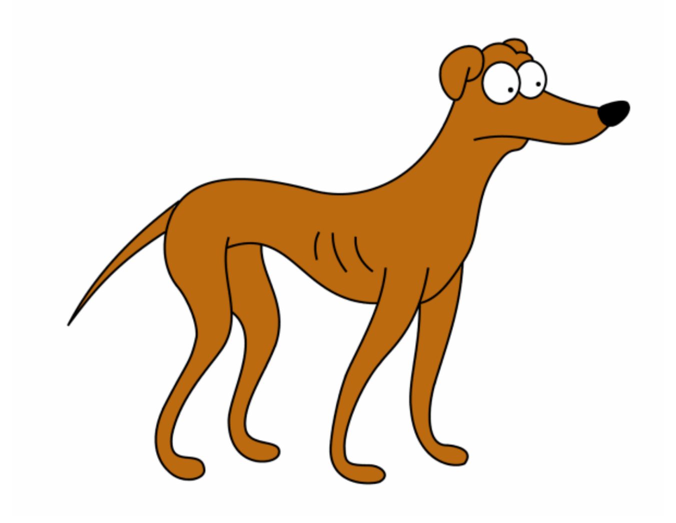
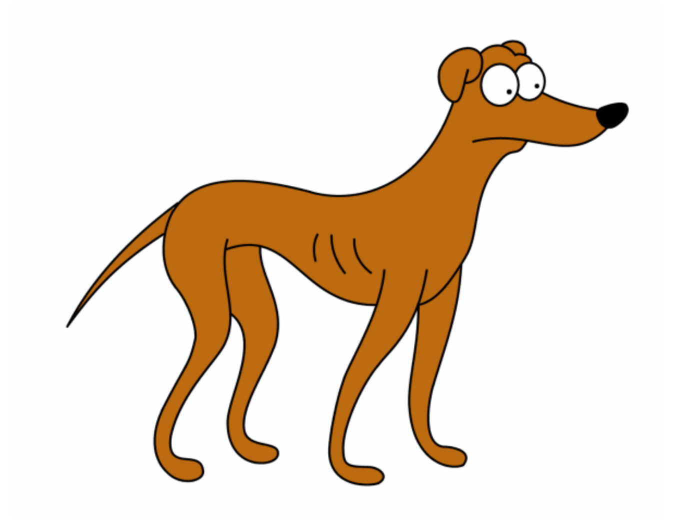
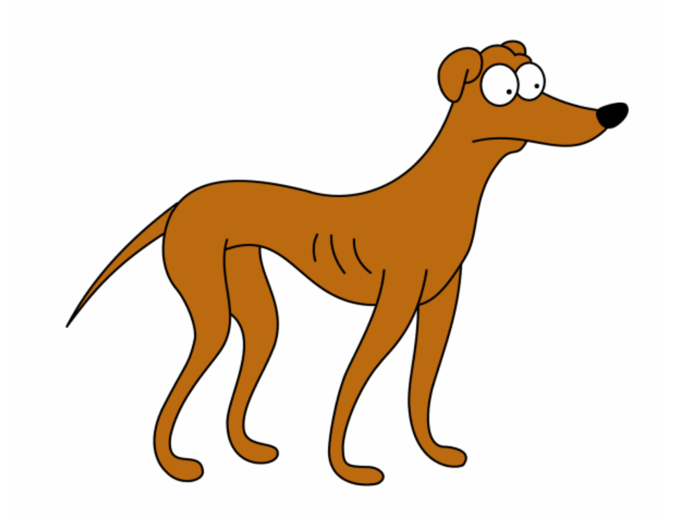
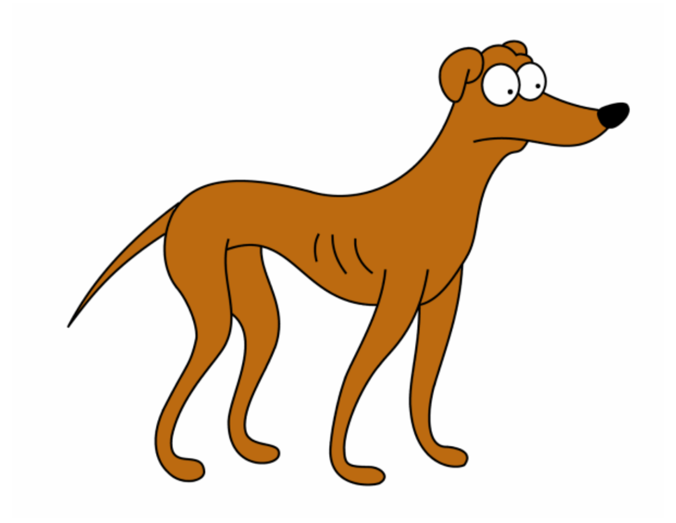
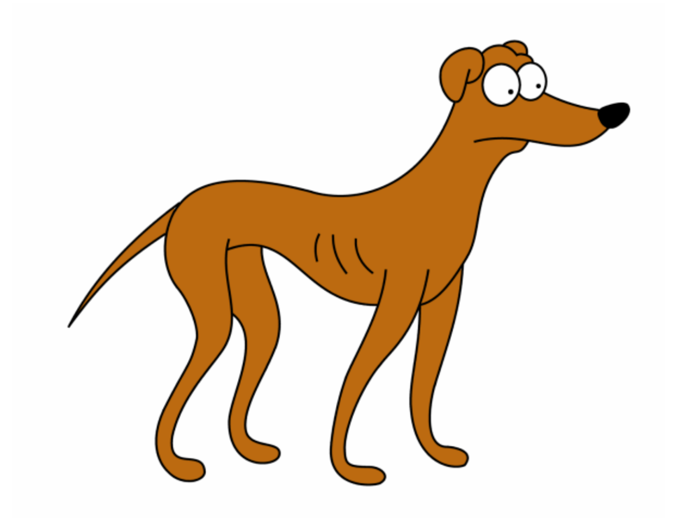
Filter Functions

```
var dogs = animals.filter { animal ->
    animal is Dog
}
```

Filter Functions

```
var dogs = animals.filter { it is Dog }
```

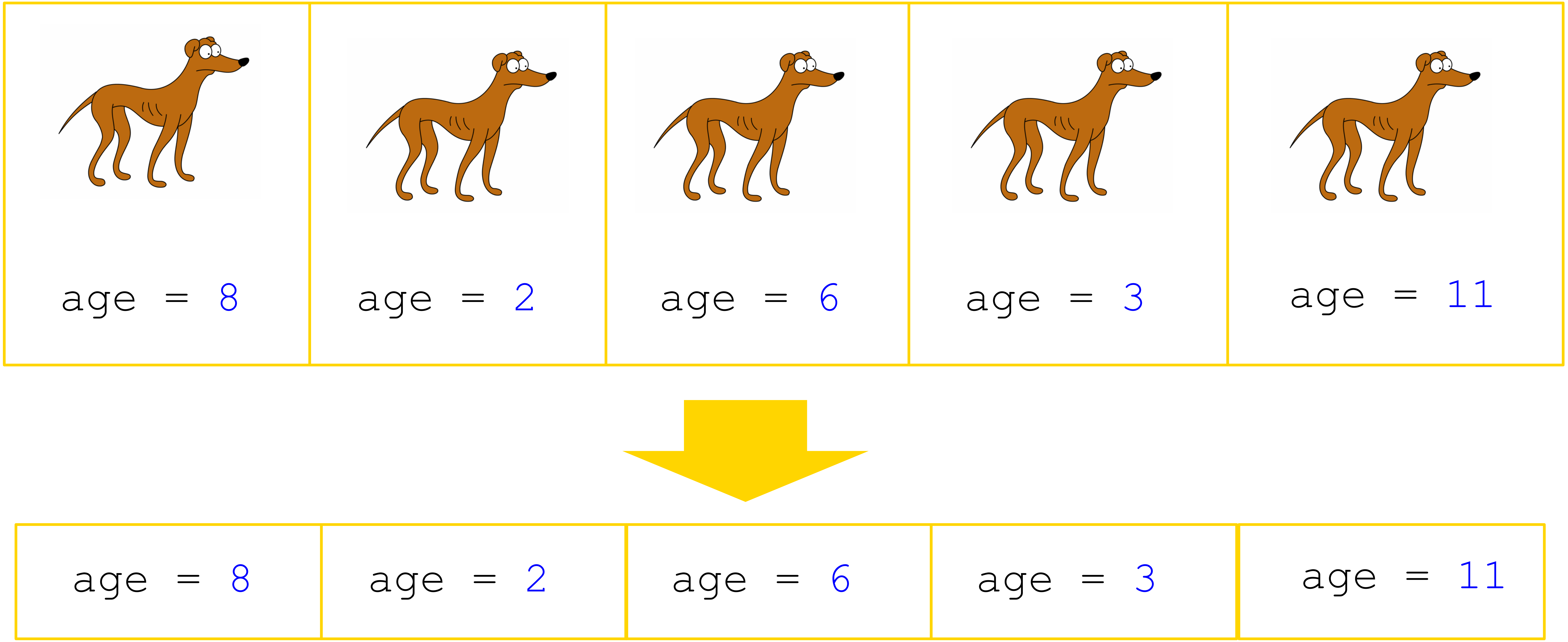
Mapping Functions

`age = 8`	`age = 2`	`age = 6`	`age = 3`	`age = 11`

consol

Mapping Functions



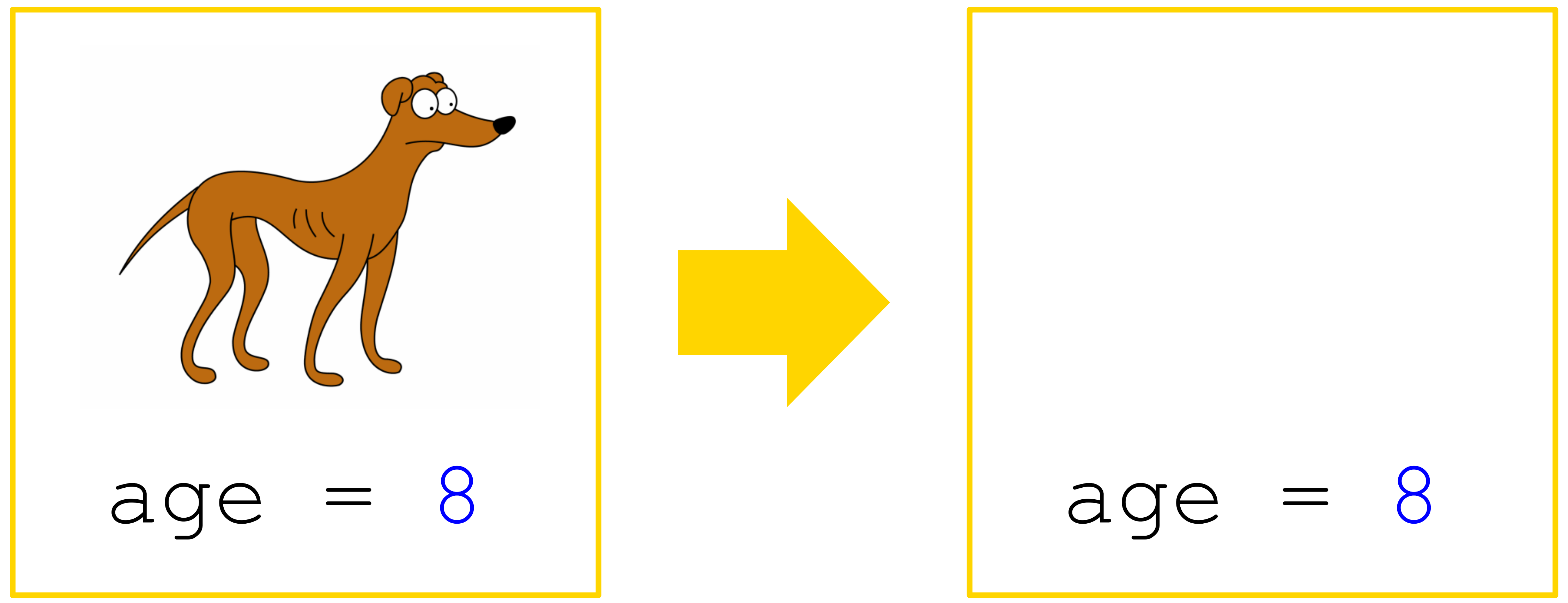
Mapping Functions

```kotlin
val dogs = listOf(Dog("Bello",age = 8),Dog("Rex",age = 2),
       Dog("Lessi",age = 6),Dog("Bruno",age = 3),Dog("Bello",age = 11))
```

Mapping Functions

```kotlin
val dogs = listOf(Dog("Bello",age = 8),Dog("Rex",age = 2),
     Dog("Lessi",age = 6),Dog("Bruno",age = 3),Dog("Bello",age = 11))


val dogAges = dogs.map {      …      }
```

Map Predicate



```
{  dog  ->  dog.age  }
```

Mapping Functions

```
val dogs = listOf(Dog("Bello",age = 8),Dog("Rex",age = 2),
     Dog("Lessi",age = 6),Dog("Bruno",age = 3),Dog("Bello",age = 11))


val dogAges = dogs.map { dog -> dog.age }
```

Mapping Functions

```
val dogs = listOf(Dog("Bello",age = 8),Dog("Rex",age = 2),
      Dog("Lessi",age = 6),Dog("Bruno",age = 3),Dog("Bello",age = 11))


val dogAges = dogs.map { it.age }
```

Mapping Functions

```
val dogs = listOf(Dog("Bello",age = 8),Dog("Rex",age = 2),
        Dog("Lessi",age = 6),Dog("Bruno",age = 3),Dog("Bello",age = 11))


val dogAges = dogs.map { it.age }

val average = dogAges.average()      (= 6)
```

Map / Filter Combinations

```kotlin
val oldDogAgeAverage = dogs.map { it.age }.filter { it > 5 }.average()
```

Time to hack!

Time to hack!

On 5_Lambdas.kt

Solution

```
// TODO TASK 1
private fun findDogNames(dogs: List<Dog>): List<String> {
    return dogs.map { it.name }
}

// TODO TASK 2
private fun findOldDogs(dogs: List<Dog>): List<Dog> {
    return dogs.filter { it.age > 5 }
}

// TODO TASK 3
private fun findNamesOfOldDogs(dogs: List<Dog>): List<String> {
    return dogs.filter { it.age > 5 }.map { it.name }
}
```

Any questions?

Thank you!

ConSol
Consulting & Solutions Software GmbH

Franziskanerstr. 38
D-81669 Munich
Germany
Tel.: +49-89-45841-100
info@consol.de
www.consol.com
Twitter: @consol_de