# Знакомство с ФП

# Контакты

## Лекторы

- Платонова Наталья; **n.platonova@tinkoff.ru**
- Кобенко Михаил; **m.kobenko@tinkoff.ru**

## Группа в Telegram

- https://clck.ru/FD8CA

# Практика

- для получения автомата по зачету нужно набрать 120 баллов
- за все выполненные задания в одной лекции можно получить максимум 10 баллов
- на сдачу практики по каждой пройденной теме дается 10 дней

# Цели курса

- Научить основам языка Scala
- Понять функциональное программирование через призму Scala
- Развить практические навыки программирования и закрепить функциональное мышление

# В этом курсе НЕ будет

- императивных рычагов управления программой*
- мутабельных коллекций и перезаписываемых ячеек памяти*
- функций, имеющих небезопасную связь с внешним миром*

* позже мы поясним почему и зачем

# Цели занятия

- Получить базовые представления о ФП и о языке
- Подготовить к выполнению практического задания

# Знакомство с ФП

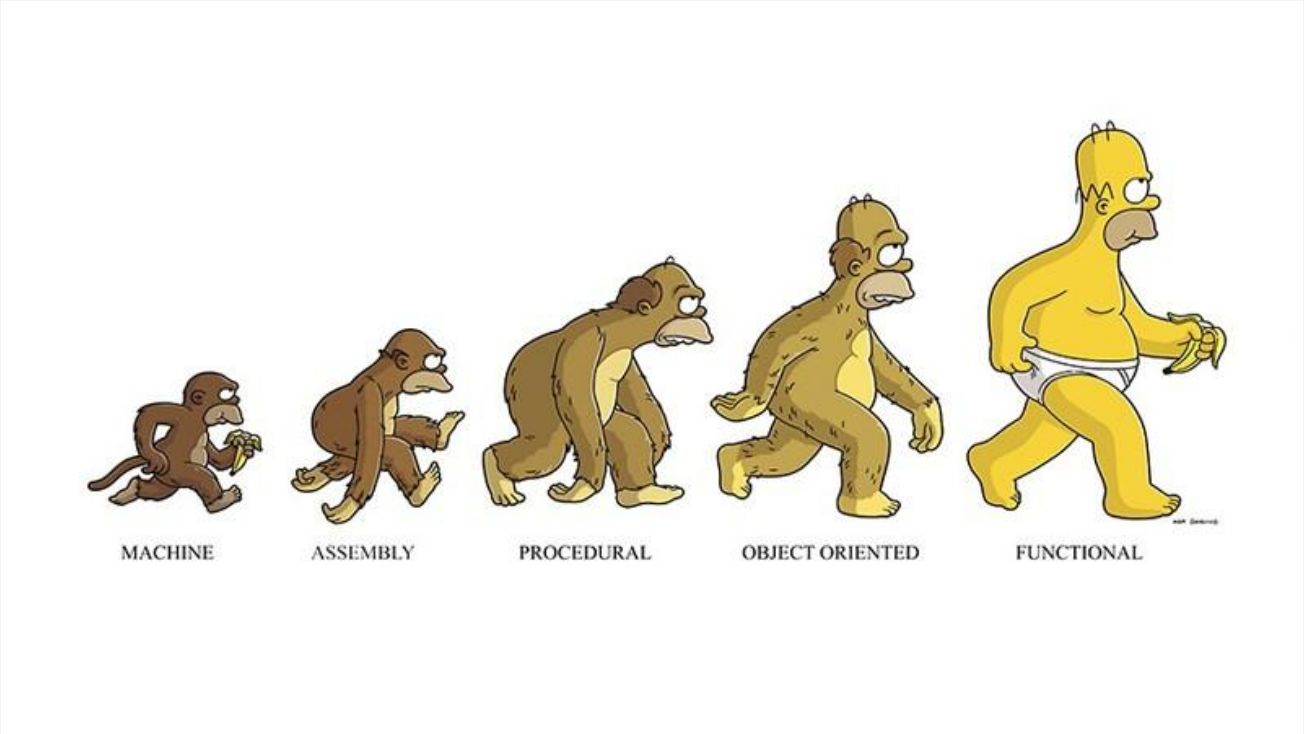
# Зачем мне знать ФП? В чем проблема?

- Много фреймворков в других языках
- Разрабатывать приложения все так же сложно
- Нет единого понимания концепций
- Нет надежды на светлое будущее

*“Нам нужен какой-нибудь способ соединять программы как садовые шланги — вкручивать новые сегменты когда необходимо обрабатывать информацию по-другому.”*

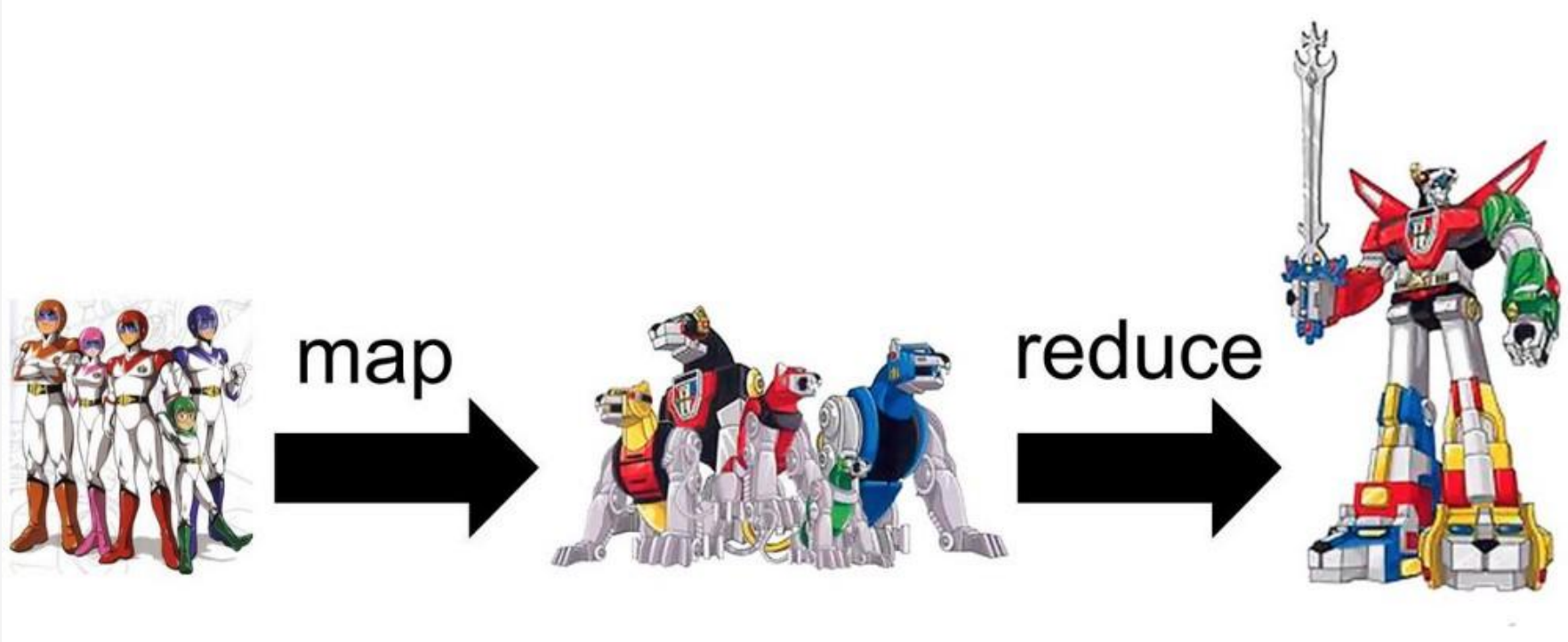
*Doug McIlroy, 1964*

# FP is ...



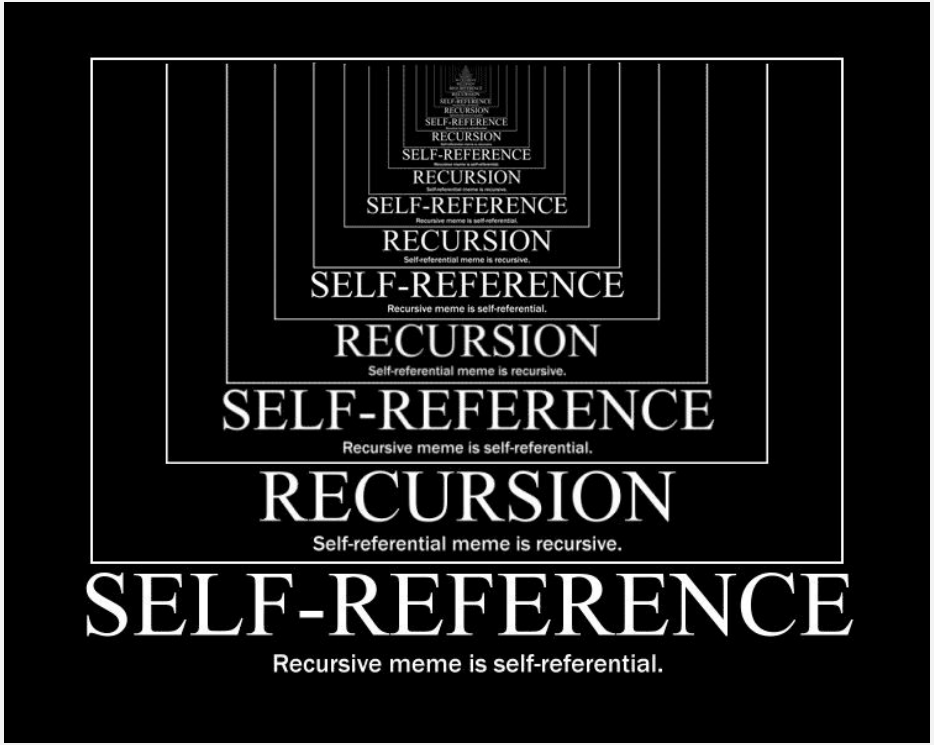

думать не столбик, а в строчку

# FP is ...




композиция совместимых* функций

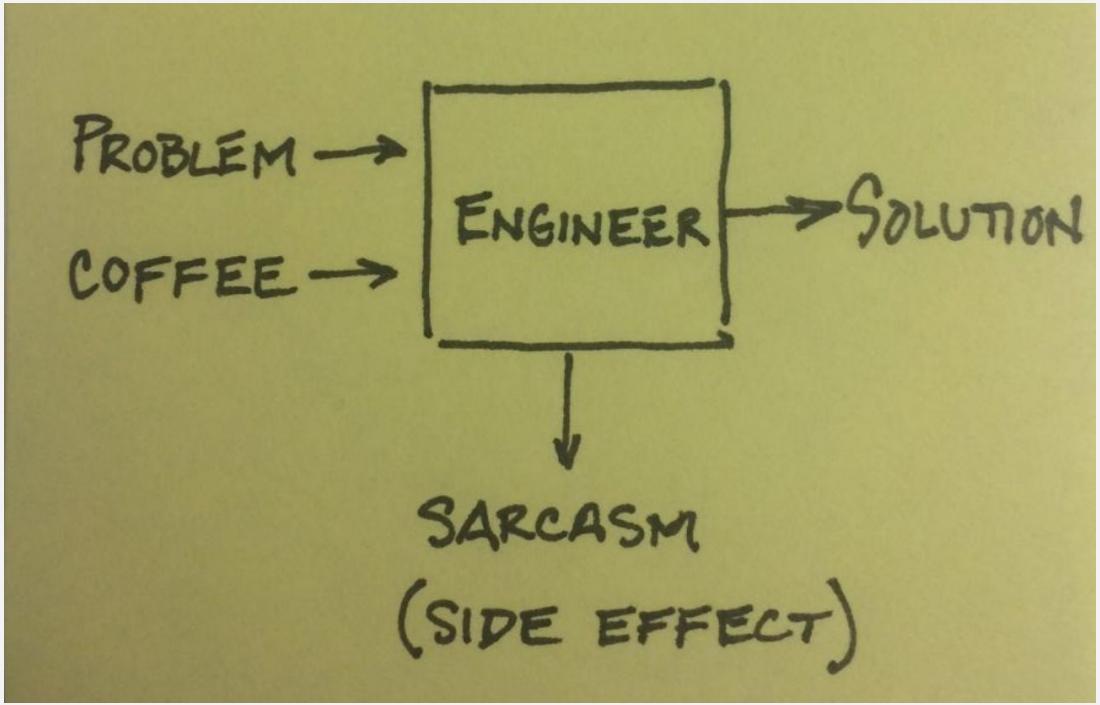
# FP is ...




хвостовые рекурсии вместо циклов

# FP is ...

# FP is ...




чистые функции(без побочных эффектов)

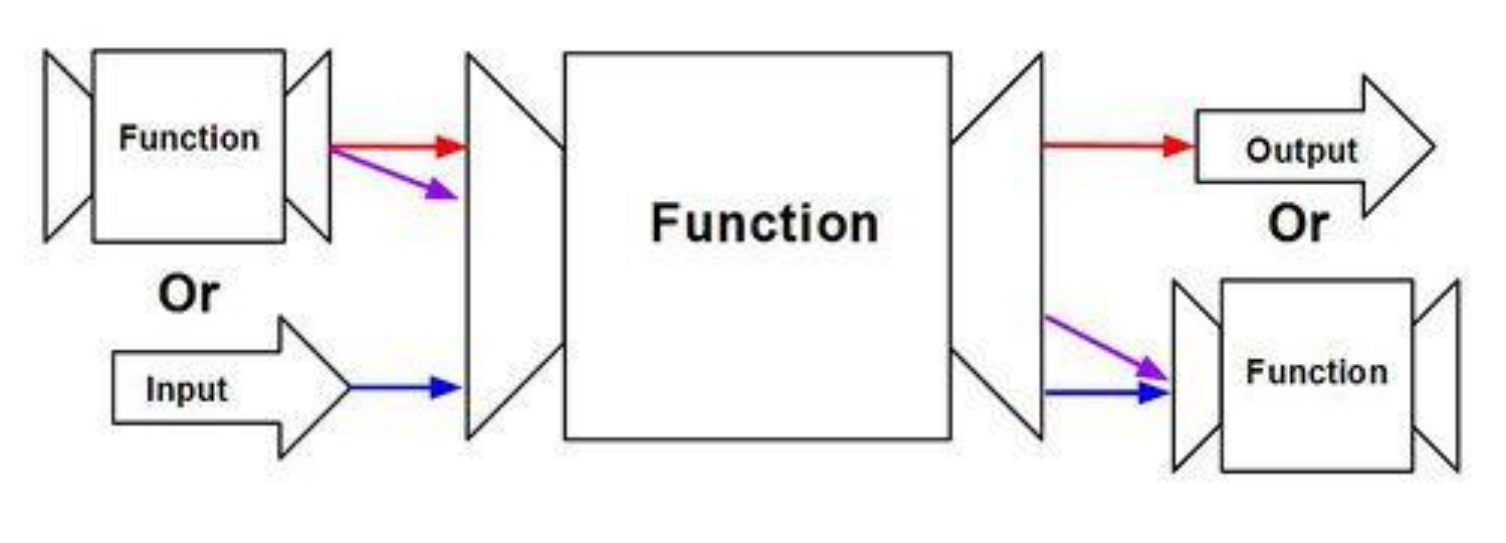
# Преимущества чистых функций

- Переиспользуемость
- Тестируемость
- Проще вылавливать ошибки

# Полезные побочные эффекты

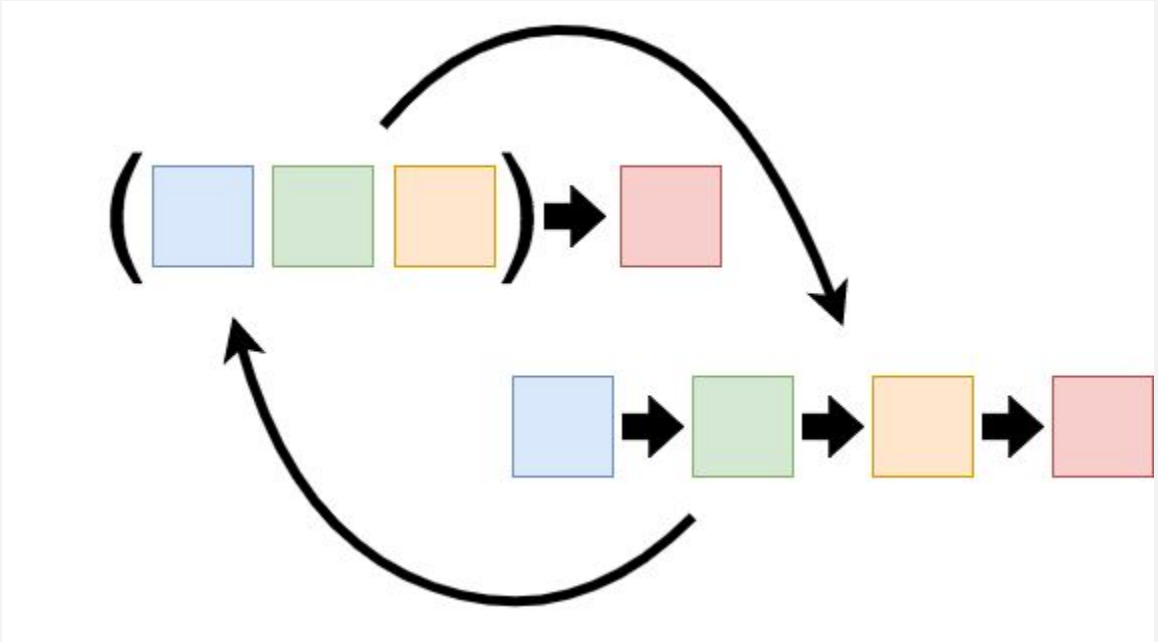
- Запросы к серверу
- Диалог для пользователя
- Вывод на экран

# FP is ...




функции высшего порядка

# FP is ...



каррирование и частичное применение

# Знакомство с языком

# Саммари языка

- Компилируется в Java bytecode
  - Бежит в JVM, т.е. “везде”
  - совместим с jvm-языками
  - можно переиспользовать Java-инструментарий
- Впитал ФП-концепции из Haskell
- Изобретен синтетически, в Швейцарии, Его Величеством - Мартином Одерски
- Имеет дружелюбное комьюнити
- Любит, когда его контрибьютят (много по-настоящему качественных библиотек)
- Сильные IT-компании

# Компании

- LinkedIn
- Twitter
- Foursquare
- Netflix
- Tumblr
- The Guardian
- Sony
- СберТех
- Яндекс
- AirBnB
- Apple (added Sep., 2013)
- Verizon
- SoundCloud
- Morgan Stanley
- AT&T
- Bloomberg
- eBay
- Tinkoff.ru

# Строгая типизация

- Проверка адекватности кода во время компиляции
- Упрощение рефакторинга
- Дополнительная безопасность за счет паттерн-матчинга
- Гибкость за счет продвинутой системы типов

# Вывод типов

```
val question: String =
 "question of life"

val answer = 42

trait Pet
class Cat extends Pet
class Dog extends Pet

val pets = List(new Cat, new Dog)
```

```
question: String = question of life


answer: Int = 42

defined trait Pet
defined class Cat
defined class Dog

pets: List[Pet] =
 List(Cat@5fffb692, Dog@3086f480)
```

# Гарантия* корректности

```scala
object Program extends App {
  val answer: String = 42
}
```

```
type mismatch;
[error]  found   : Int(42)
[error]  required: String
[error]   val answer: String = 42
[error]                        ^
```

# Pattern-matching

```
trait Pet
class Cat extends Pet {
  def meow(): String = "meow-meow"
}
class Dog extends Pet {
  def bark(): String = "woof-woof"
}

val pet: Pet = new Cat
pet.meow
pet match {
  case cat: Cat => cat.meow()
  case dog: Dog => dog.bark()
}
```

```
defined trait Pet
defined class Cat


defined class Dog



pet: Pet = Cat@46a795de
meow is not a member...
res0: String = meow-meow
```

# Продвинутая система типов

```
trait Pet {
  type Voice
  def voice: Voice
}
class Cat extends Pet {
  type Voice = String
  def voice = "meow-meow"
}
class Robot extends Pet {
  type Voice = Int
  def voice = 31337
}

new Robot().voice
```

```
defined trait Pet


defined class Cat


defined class Robot


res0: Int = 31337
```

# Парадигмы программирования

- Императивное программирование
  - Процедурное программирование
  - Структурное программирование
  - Объектно-ориентированное программирование
- Декларативное программирование
  - Функциональное программирование
  - Логическое программирование

# Императивное

```
define main = { // процедура
 alloc x = 0 // состояние памяти
 x = x + 1 // инструкции
 x = x * 5 // присваивание
 if (x > 2) // ветвление
   return x // произвольная
            // точка выхода
 x = x + 3
 return x
}
```

# Декларативное

```
define main(x: Int): Int = {
// функция
 define increased = x + 1
// выражения
 define multiplied = increased * 5
// объявление а не присваивание
 define bumped = multiplied + 3
 define result =
  if (x > 2) multiplied else bumped
// тернарный if
 result
}
```

* Псевдокод

# Обещание

Scala — мультипарадигмальный язык программирования, спроектированный кратким и типобезопасным для простого и быстрого создания компонентного программного обеспечения, сочетающий возможности функционального и объектно-ориентированного программирования.

Wikipedia.org

# Scala объектно-ориентированная

- Все значения это объекты, все операции это вызовы методов
- Трейты для композиции объектов

# Все значения это объекты

```
1 + 2
1.+(2)

"Scala".charAt(0)
"Scala" charAt 0

"Scala".replace("a", "_")
"Scala" replace ("a", "_")
```

```
res0: Int = 3
res1: Int = 3

res2: Char = S
res3: Char = S

res4: String = Sc_l_
res5: String = Sc_l_
```

# Трейты для композиции объектов

```
trait Полет {
 def крылья: String
 def лететь(): String =
   "Хлопаю моими " + крылья
}

trait Птица {
 def крылья: String =
   "крыльями"
}

class Голубь extends Птица with Полет
class Пингвин extends Птица
```

```
class Дракон extends Полет {
 def крылья: String =
   "стеклянными крыльями"
}

object Икар extends Полет {
 def крылья: String =
   "восковыми крыльями"
}

Икар.лететь()
```

# Обещание

Scala — мультипарадигмальный язык программирования, спроектированный кратким и типобезопасным для простого и быстрого создания компонентного программного обеспечения, сочетающий возможности функционального и объектно-ориентированного программирования.

Первые версии языка созданы в 2003 году коллективом лаборатории методов программирования Федеральной политехнической школы Лозанны под руководством Мартина Одерски, язык реализован для платформ Java и JavaScript. По мнению Джеймса Стрэчена, создателя языка программирования Groovy, Scala может стать преемником языка Java.

Wikipedia.org

# Всё возвращает значение

```
val conditionalValue =
 if (1 > 2) 13 else 42

val value = print("42")
```

```
conditionalValue: Int = 42


value: Unit = ()
```

# Операции над функциями

```
def square(x: Int): Int = x * x
val cude = { x: Int => x * x * x }

val numbers = List(1, 2, 3)
numbers.map(square)

def mul(m: Int): Int => Int =
  x => x * m

val double = mul(2)
double(6)
```

```
square: square[](val x: Int) => Int
cude: Int => Int = Lambda$...

numbers: List[Int] = List(1, 2, 3)
res0: List[Int] = List(1, 4, 9)

mul: mul[](val i: Int) => Int => Int


double: Int => Int = Lambda$...
res1: Int = 12
```

# Иммутабельные структуры данных

```
val fruit = "apple"
fruit = "orange"
var answer = -1
answer = 42

val laws = List("#1", "#1")
laws(0) = "#0"

val updated =
  laws.updated(0, "#0")
laws

val newLaws = "#-1" :: updated
```

```
fruit: String = apple
// reassignment to val
answer: Int = -1
answer: Int = 42

laws: List[String] = List(#1, #1)
// update is not a member...

updated: List[String] = List(#0, #1)

res0: List[String] = List(#1, #1)

newLaws: List[String] =
 List(#-1, #0, #1)
```

# Базовый синтаксис

# Литералы

```
1
1 + 2
3.14
"Scala string"


"""You can write multiline strings
  | with triple " symbols.
  |Leading spaces will be removed
  | by stripMargin method
""".stripMargin
```

```
res0: Int = 1
res1: Int = 3
res2: Double = 3.14
res3: String = Scala string

res2: String =
You can write multiline strings
 with triple " symbols.
Leading spaces will be removed
 by stripMargin method
```

# Объявление значений

```
val a = 1

val b = a + 2

a = 5



val pi: Double = 3.14

val e: Int = 2.72
```

```
a: Int = 1

b: Int = 3

error: reassignment to val
 a = 5
 ^

pi: Double = 3.14

error: type mismatch;
 found   : Double(2.72)
 required: Int
```

# Объявление функций

```
def square(x: Int): Int = x * x

def complexStuff(x: Int) = {
 def multiply(m: Int) = x * m
 def increase(x: Int) = x + 500

 increase(multiply(1000))
}

complexStuff(100)
```

```
square: square[](val x: Int) => Int

complexStuff:
 complexStuff[](val x: Int) => Int




res0: Int = 100500
```

# class

```
class Vehicle(description: String) {
  println("constructing...")
  val name = description + " vehicle"
  def print(): Unit = println(name)
}

class Car extends Vehicle("heavy")

val car = new Car


car.print()
```

```
defined class Vehicle




defined class Car

constructing...
car: Car = Car@4b765e92

heavy vehicle
res0: Unit = ()
```

# object

```
class Vehicle(description: String) {
  def print(): Unit =
    println(description)
}

object Vehicle {
  def produce(): Vehicle =
    new Vehicle("from factory")
}

Vehicle.produce().print()

object Car extends Vehicle("heavy")
Car.print()
```

```
defined class Vehicle



defined module Vehicle



from factory

defined module Car
heavy
```

# package/import

```
package my.pakage

import another.pakage.Data

object Main {
 def main(args: Array[String]): Unit = {
   val d = new Data(1, 2)
   println("Hello world")
 }
}
```

# package/import

```
package my.pakage

object Main {
 def main(args: Array[String]): Unit = {
   import another.pakage.Data

   val d = new Data(1, 2)
   println("Hello world")
 }
}
```

# List

```
val elements =
 List("fire","water","earth","air")

elements.size
elements(1)

elements(2) = "darkness"

var newElements =
 List("fire","water","earth","air")
newElements =
 newElements.updated(2, "darkness")
```

```
elements: List[String] =
 List(fire, water, earth, air)

res0: Int = 4
res1: String = water

error: value update is not a member
of List[String]
newElements: List[String] =
 List(fire, water, earth, air)
newElements: List[String] =
 List(fire, water, darkness, air)
```

# Map

```
val capitals = Map(
 "Russia" -> "Moscow",
 "Great Britain" -> "London"
)

capitals.keys

capitals("Great Britain")

capitals("Great Britain") =
 "Landan"
```

```
capitals: Map[String,String] =
 Map(Russia -> Moscow, Great Britain
-> London)


res0: Iterable[String] =
 Set(Russia, Great Britain)
res1: String = London

value update is not a member of
Map[String,String]
```

# if

```
val age = 13

if (age < 18) println("Underaged")


if (age < 18) println("Underaged")
else println("Adult")


val label =
 if (age < 18)
  "Underaged"
 else
  "Adult"
```

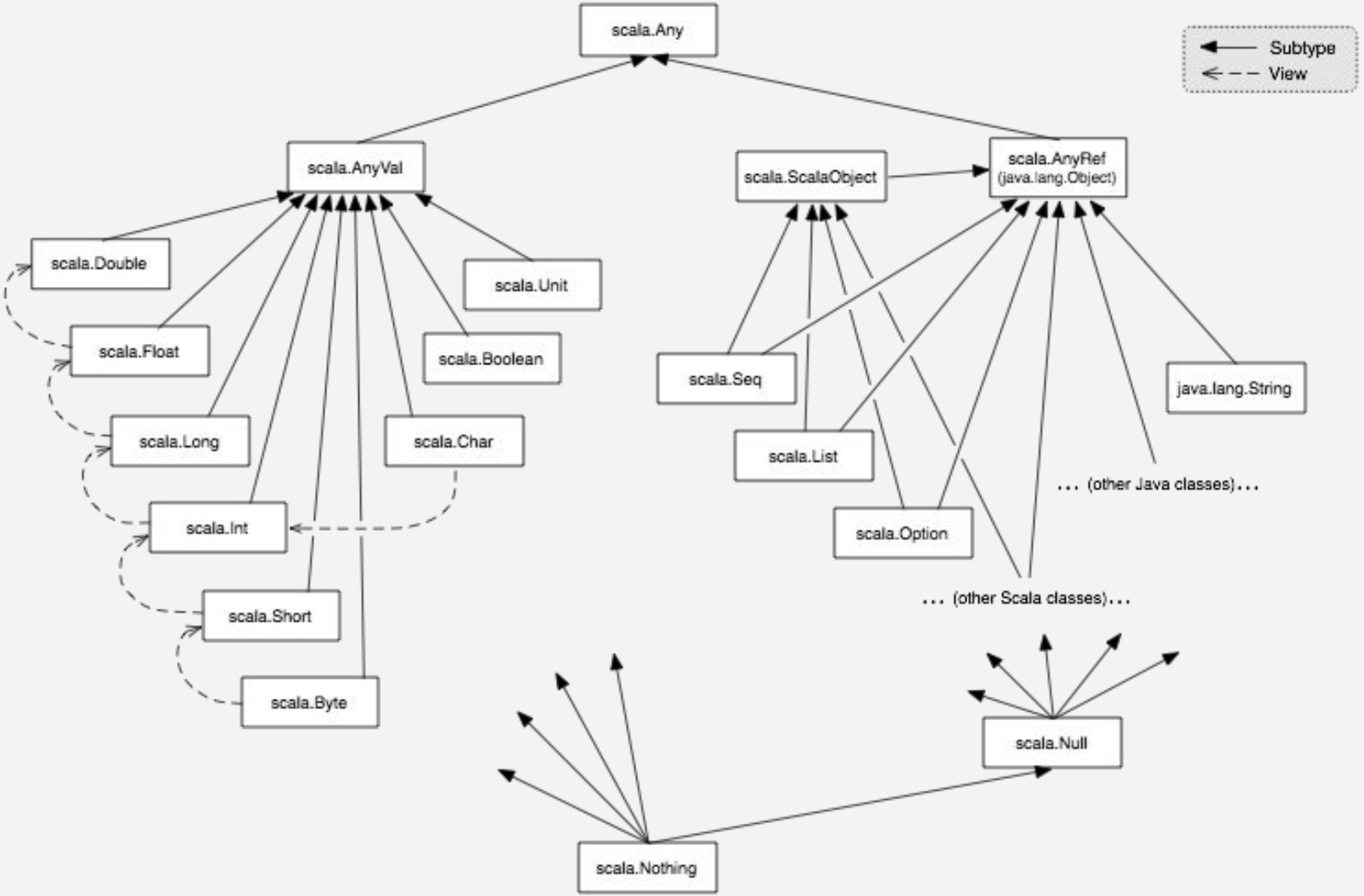
```
age: Int = 13

Underaged
res0: Unit = ()

Underaged
res1: Unit = ()


label: String = Underaged
```

# for

```
val numbers = List(1, 2, 3)
val returned =
 for (n <- numbers) println(n)


val squares =
 for (n <- numbers) yield n * n

val otherNumbers = List(4, 5, 6)

val complexStuff = for {
 low <- numbers
 high <- otherNumbers
} yield low * high
```

```
numbers: List[Int] = List(1, 2, 3)
1
2
3
returned: Unit = ()
squares: List[Int] = List(1, 4, 9)


otherNumbers: List[Int] =
 List(4, 5, 6)
complexStuff: List[Int] =
 List(4, 5, 6, 8, 10, 12, 12, 15, 18)
```

# Иерархия типов


scala.Any
Subtype
View
scala.AnyVal
scala.ScalaObject
scala.AnyRef
(java.lang.Object)
scala.Double
scala.Unit
scala.Float
scala.Boolean
scala.Seq
java.lang.String
scala.Long
scala.Char
scala.List
... (other Java classes)...
scala.Int
scala.Option
... (other Scala classes)...
scala.Short
scala.Byte
scala.Null
scala.Nothing

# AnyVal

Value типы

- Byte
- Short
- Char
- Int
- Long
- Float
- Double
- Boolean
- Unit

# AnyRef

Ссылочные типы, аналог Object из Java.

Примеры:

- String
- Tuple
- List
- Map
- etc

# Bottom types

- Null - наследуется (то есть совместим) со всем, что имеет в предках AnyRef, применяется для null ссылок. Существует один инстанс - null.
- Nothing - наследуется (то есть совместим) со всем, применяется как “терминальное” значение. Инстансов не существует.

# Арифметичиские операции

- +
- -
- *
- /
- %

# Логические операции

- >
- <
- >=
- <=
- !
- && и &
- || и |

# Побитовые операции

- &
- |
- ^
- ~

# RichWrappers

| | |
|---|---|
| Byte | scala.runtime.RichByte |
| Short | scala.runtime.RichShort |
| Int | scala.runtime.RichInt |
| Long | scala.runtime.RichLong |
| Char | scala.runtime.RichChar |
| Float | scala.runtime.RichFloat |
| Double | scala.runtime.RichDouble |
| Boolean | scala.runtime.RichBoolean |
| String | scala.collection.immutable.StringOps |

# Лень

```
val x = { println("x"); 15 }
lazy val y = { println("y");
13 }
```

```
x
x: Int = 15
y: Int = <lazy>
```