Macros are functions that are called by the compiler during compilation. Within these functions the programmer has access to compiler APIs. For example, it is possible to generate, analyze, and typecheck code.

Macros are shipping with the official Scala compiler. Since 2.10.0 Scala includes macros enabled with import language.experimental.macros. Numerous commercial and research projects are already using macros.

Macros are good for code generation, static checks and domain-specific languages. Scenarios that traditionally involve writing and maintaining boilerplate can be addressed with macros in a concise and maintainable way.

### Scala Macros

Eugene Burmako, Denis Shabanil

Martin Odersky

### Term generation

def specialized[T]: ClassTag[codec: => Any] - macro 
  def createArray[T: ClassTag](size: Int, el: T) = {
    val a = new Array[T](size)
    for (i <- 0 until size) a(i) = el
  }

### Type generation

def db2(connString: String): Any - macro 
  val db = db2(jdbcDriver:coffee:jdbc.db2)

### Materialization

trait Showable[T] { def show(x: T): String }
  def show(x: T): String = 
    implicit val instShowable = new Showable[Int] {
      def show(x: Int) = x.toString
    }

### Advanced type signatures

trait ClassTag[|a|] request extends Request 
trait Reply 

case class Command, Success extends Reply 

abstract class ActorRef[|a|] def ![|a|]: Any = Unit 

class ChannelRef[|a|] {actor: ActorRef} def <~[::-M:][|a|>:E:][|a|:M] = 

val actor = someActor 
actor! Command // compiles, but fails at runtime

### Advanced static checks

val sender: ActorRef -
  def future[T](body: => T): Future[T] = 
    sender! Response(result) // dangerous capture of a var

### Typing the untypeable

case class Coffee {
  val name: String;
  val price: Decimal
}

type Query(Filt = filter, String = String)

### Language virtualization

// #1: external DSL with strings
val usersWatching = query(String, Int, String){
  "select id, name from users where name = ? 
  when(userWatching("John"))" 
}

// #2: internal DSL with vanilla Scala
case class user(id: Column[Int], name: Column[String]) {
  def filter(p: => Boolean): Queryable[Int] = 
    this.t = List(t).show, translate.execute(t)
}

case class Queryable[T] {val query: Query { 
  def filter: Boolean = Queryable(T) 
  def tutist: List[T] = Query.validate.execute(T)
}

### Internal DSLs

val futureDOB: Future[Response] = 
  $WS"(http://api.day-of-year.today/", get)

val dob: Dynamic = 
  $WS"(http://api.days-left.today/", get)

val daysUntil: Future[Response] = 
  $WS"(http://api.days-left.today/", get)

val flatMap { 
  dob dayOfYear = dayOfYear.map 
  users.filter(_ == dob)
}

### External DSLs

scala> Query"(**)" 
:find $e
  :where
  ?s $e
  $e

val query = Query("/**")