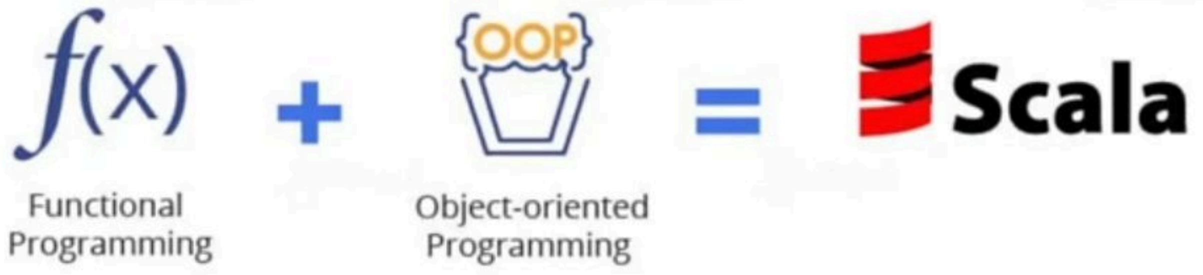
Introduction

Scala is a strong statically typed general-purpose programming language which supports both object-oriented programming and functional programming.

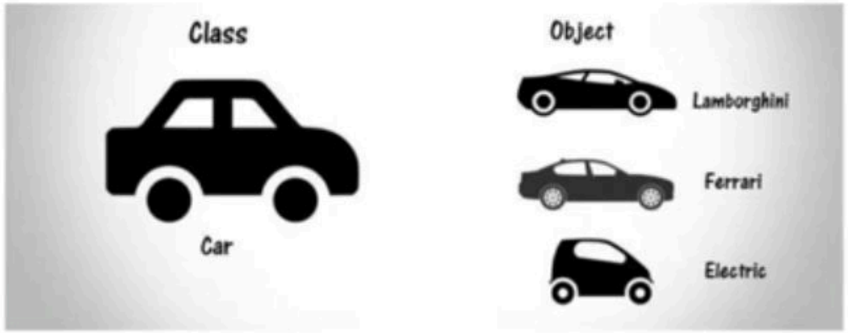
Scala is used in Data processing, distributed computing, and web development. It powers the data engineering infrastructure of many companies.



Classes and Objects

A class is a blueprint for objects. Once you define a class, you can create objects from the class blueprint with the keyword new. Through the object, you can use all functionalities of the defined class.

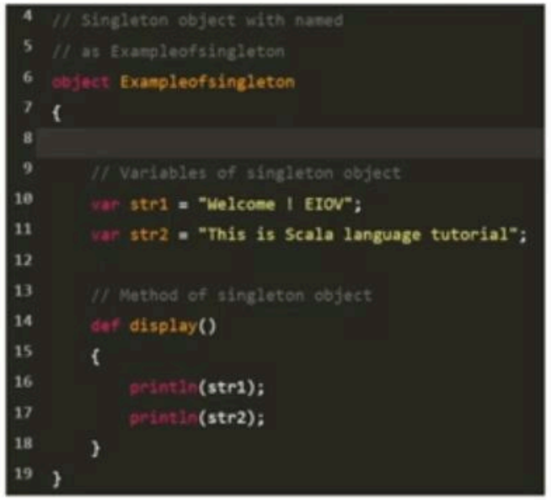
```
class Point(xc: Int, yc: Int) {
 var x: Int = xc
 var y: Int = yc

 def move(dx: Int, dy: Int) {
    x = x + dx
    y = y + dy
    println ("Point x location : " + x);
    println ("Point y location : " + y);
 }
```



Classes and Objects

Singleton Objects:

Scala is more object-oriented than Java because, in Scala, we cannot have static members. Instead, Scala has singleton objects. A singleton is a class that can have only one instance, i.e., Object. You create a singleton using the keyword object instead of the class keyword.



```
// Singleton object with named
// as Exampleofsingleton
object Exampleofsingleton
{

    // Variables of singleton object
    var str1 = "Welcome ! EIOV";
    var str2 = "This is Scala language tutorial";

    // Method of singleton object
    def display()
    {
        println(str1);
        println(str2);
    }
}
```

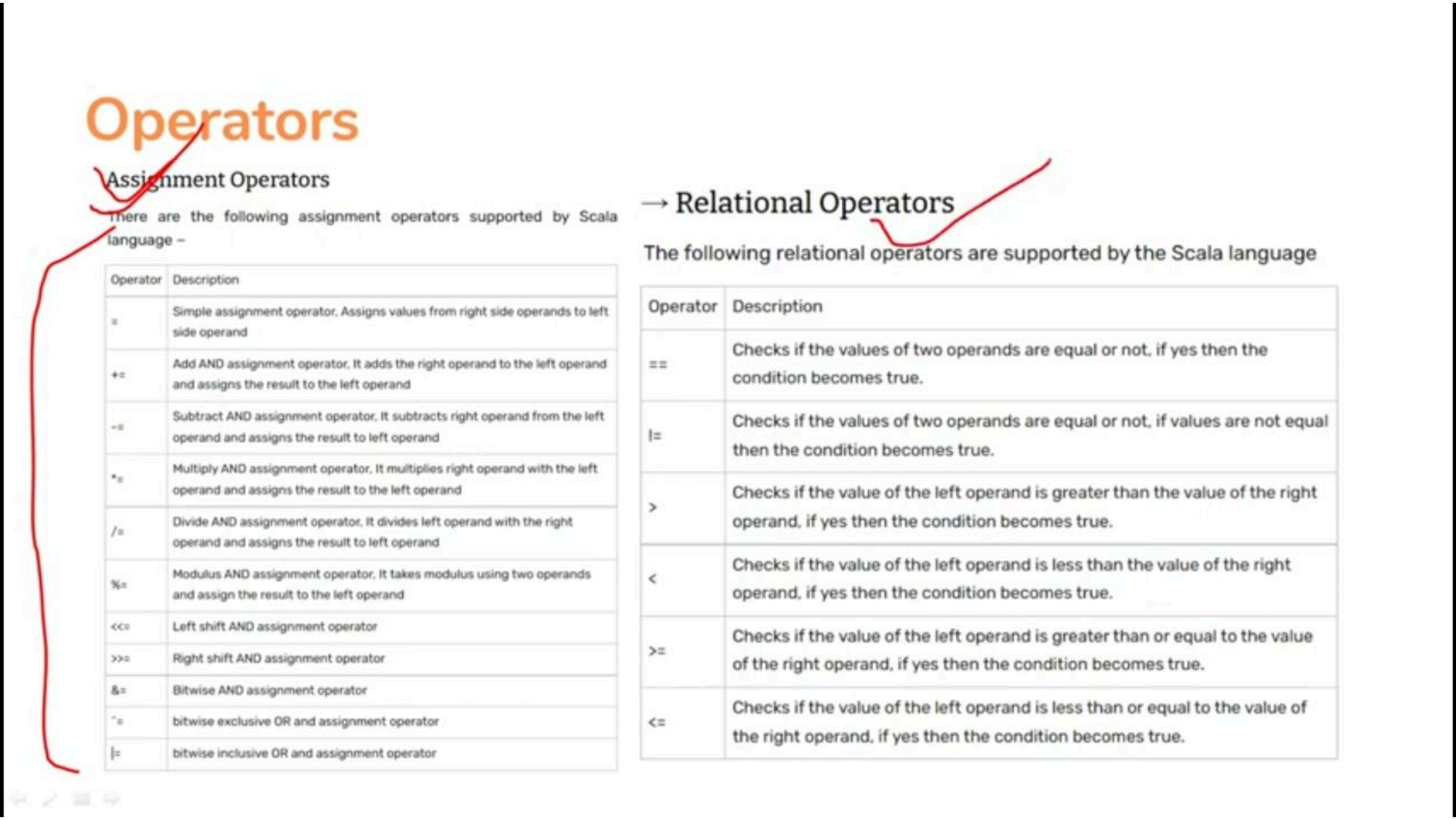
Basic types and Operators

Sr. No.	Data Type	Description
1	Byte	8 bit signed value. Range from -128 to 127
2	Short	16 bit signed value. Range -32768 to 32767
3	Int	32 bit signed value. Range -2147483648 to 2147483647
4	Long	64 bit signed value. -9223372036854775808 to 9223372036854775807
5	Float	32 bit IEEE 754 single-precision float
6	Double	64 bit IEEE 754 double-precision float
7	Char	16 bit unsigned Unicode character. Range from U+0000 to U+FFFF
8	String	A sequence of Chars
9	Boolean	Either the literal true or the literal false
10	Unit	Corresponds to no value
11	Null	null or empty reference
12	Nothing	The subtype of every other type; includes no values
13	Any	The supertype of any type; any object is of type Any
14	AnyRef	The supertype of any reference type

Operators:

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. Scala is rich in built-in operators and provides the following types of operators –

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators



Operators

Assignment Operators

There are the following assignment operators supported by Scala language –

Operator	Description
=	Simple assignment operator. Assigns values from right side operands to left side operand
+=	Add AND assignment operator. It adds the right operand to the left operand and assigns the result to the left operand
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assigns the result to left operand
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assigns the result to the left operand
/=	Divide AND assignment operator. It divides left operand with the right operand and assigns the result to left operand
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to the left operand
<<=	Left shift AND assignment operator
>>=	Right shift AND assignment operator
&=	Bitwise AND assignment operator
^=	bitwise exclusive OR and assignment operator
|=	bitwise inclusive OR and assignment operator

→ Relational Operators

The following relational operators are supported by the Scala language

Operator	Description
==	Checks if the values of two operands are equal or not, if yes then the condition becomes true.
!=	Checks if the values of two operands are equal or not, if values are not equal then the condition becomes true.
>	Checks if the value of the left operand is greater than the value of the right operand, if yes then the condition becomes true.
<	Checks if the value of the left operand is less than the value of the right operand, if yes then the condition becomes true.
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes then the condition becomes true.
<=	Checks if the value of the left operand is less than or equal to the value of the right operand, if yes then the condition becomes true.

Operators

→ Logical Operators

The following logical operators are supported by the Scala language.

Operator	Description
&&	It is called Logical AND operator. If both the operands are non zero then the condition becomes true.
||	It is called Logical OR Operator. If any of the two operands is non zero then the condition becomes true.
!	It is called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then the Logical NOT operator will make it false.

→ Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. The truth tables for &, |, and ^ are as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Operator	Description
&	Binary AND Operator copies a bit to the result if it exists in both operands.
|	Binary OR Operator copies a bit if it exists in either operand.
^	Binary XOR Operator copies the bit if it is set in one operand but not both.
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.
<<	Binary Left Shift Operator. The bit positions of the value of the left operand are moved left by the number of bits specified by the right operand.
>>	Binary Right Shift Operator. The bit positions of the left operand value are moved right by the number of bits specified by the right operand.
>>>	Shift right zero-fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.

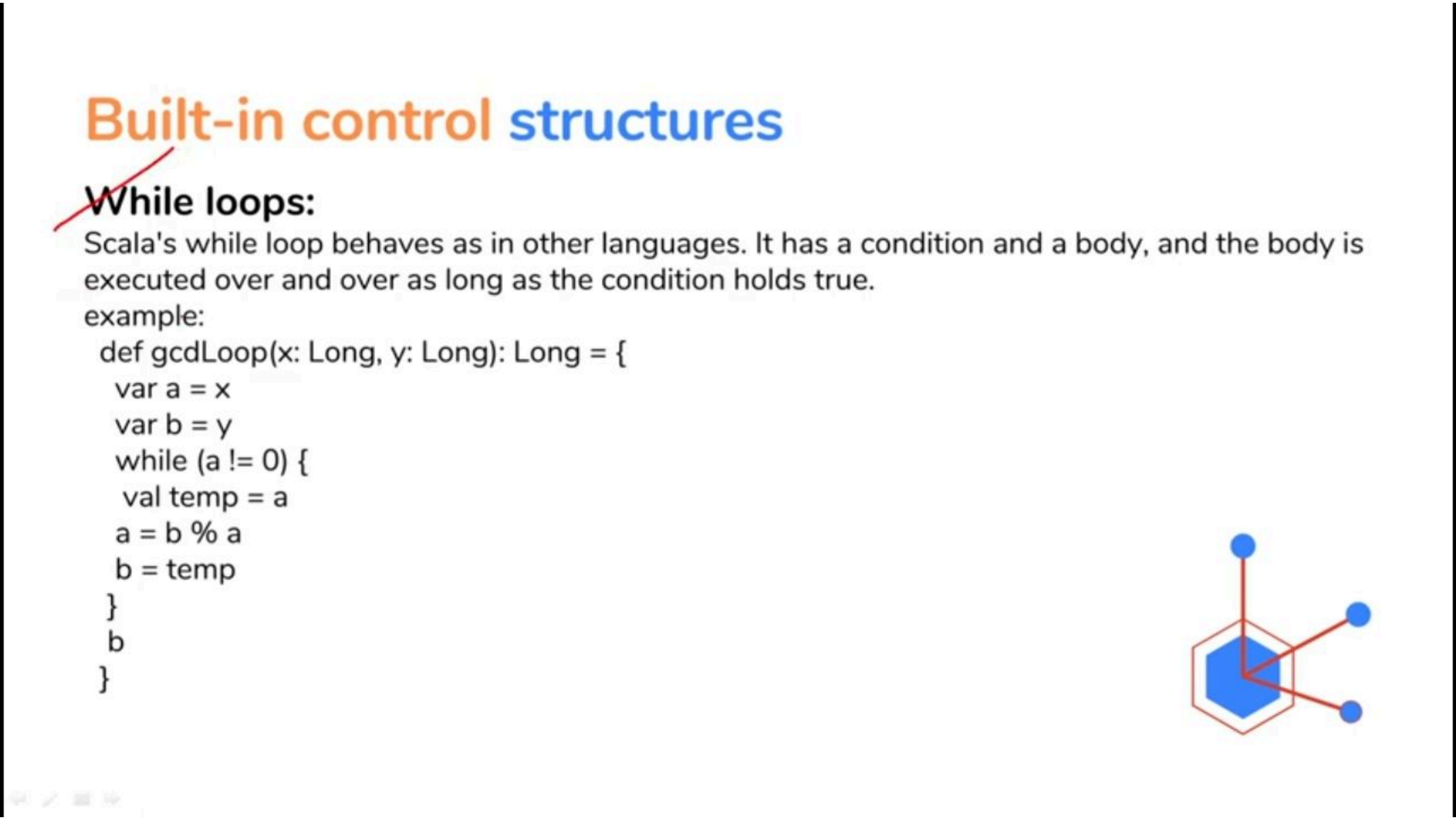
Built-in control structures

- If expressions
- While loops
- For expressions
- Exception handling with try expressions
- Match expressions

If expressions:

- Scala's if works just like in many other languages. It tests a condition and then executes one of two code branches depending on whether the condition holds true. Here is a common example, written in an imperative style:

```
var filename = "default.txt"
  if (!args.isEmpty)
    filename = args(0)
```



Built-in control structures

While loops:

Scala's while loop behaves as in other languages. It has a condition and a body, and the body is executed over and over as long as the condition holds true.

example:

```
 def gcdLoop(x: Long, y: Long): Long = {
  var a = x
  var b = y
  while (a != 0) {
   val temp = a
  a = b % a
  b = temp
 }
 b
}
```

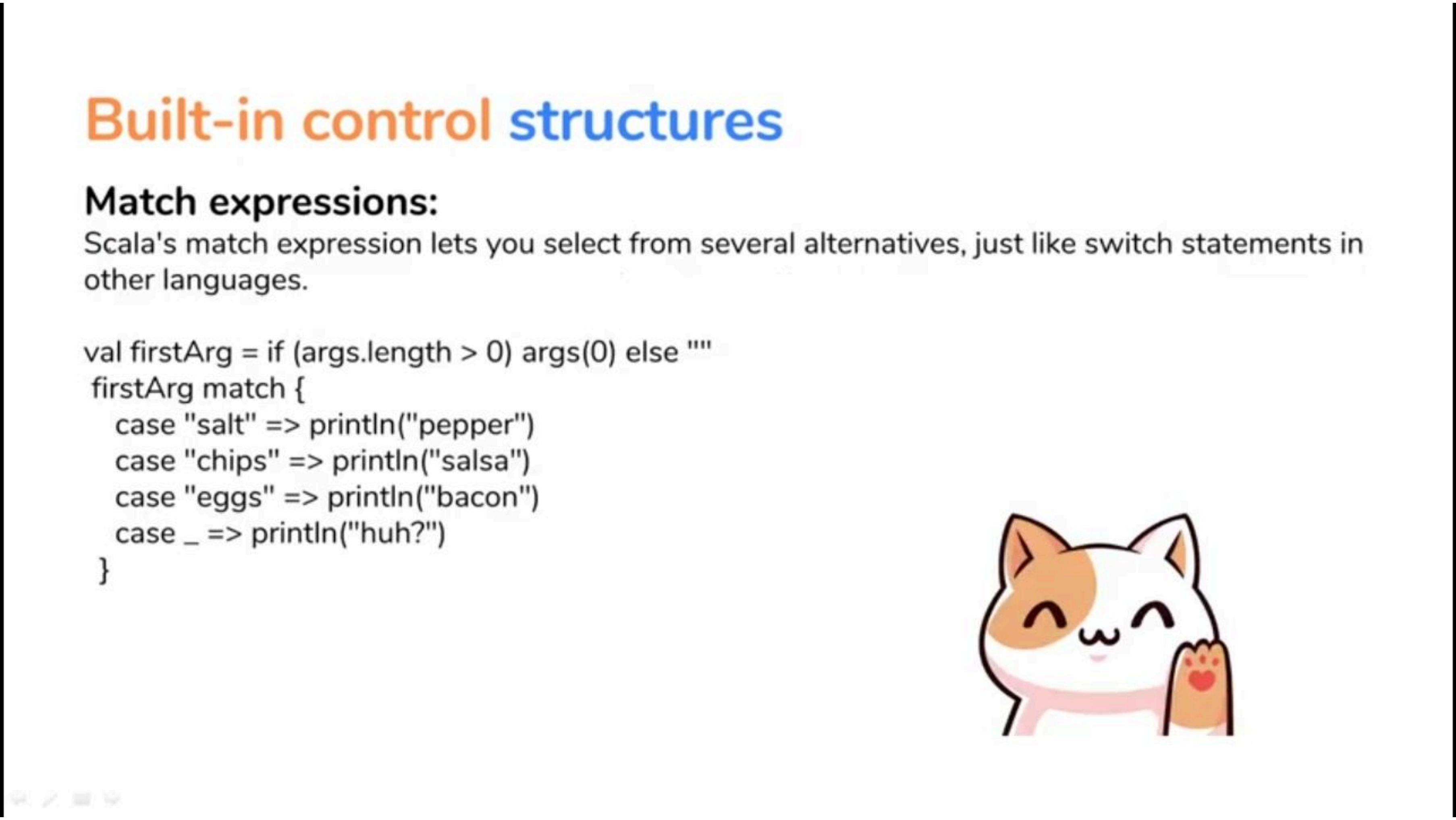
Built-in control structures

For expressions:

```
val filesHere = (new java.io.File(".")).listFiles
     for (file <- filesHere)
          println(file)
```

Exception handling with try expressions:

```
import java.io.FileReader
  import java.io.FileNotFoundException
  import java.io.IOException

 try {
   val f = new FileReader("input.txt") // Use and close file
 } catch {
   case ex: FileNotFoundException => // Handle missing file
   case ex: IOException => // Handle other I/O error
 }
```

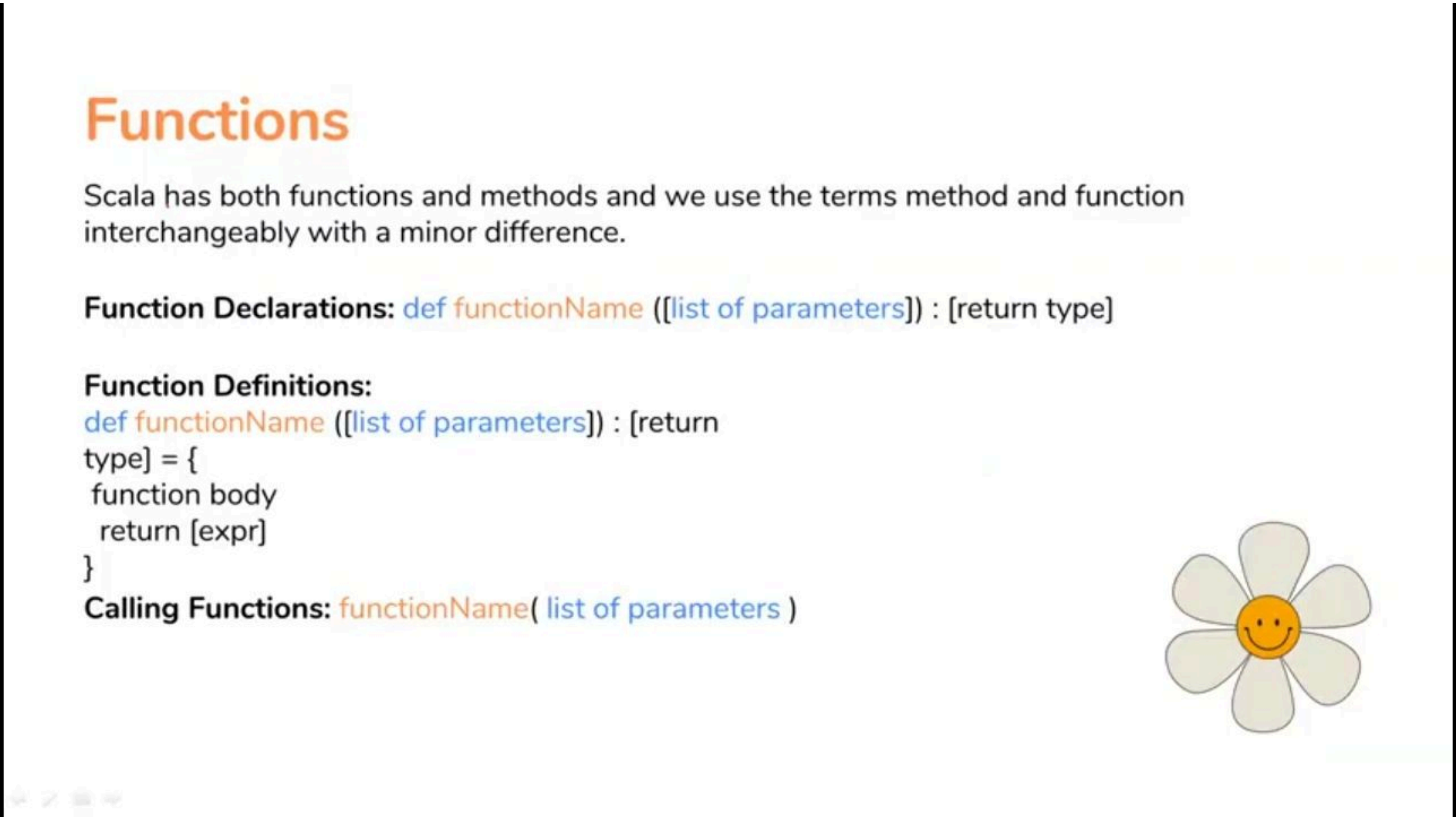


Built-in control structures

Match expressions:

Scala's match expression lets you select from several alternatives, just like switch statements in other languages.

```
val firstArg = if (args.length > 0) args(0) else ""
 firstArg match {
   case "salt" => println("pepper")
   case "chips" => println("salsa")
   case "eggs" => println("bacon")
   case _ => println("huh?")
  }
```



Functions

Scala has both functions and methods and we use the terms method and function interchangeably with a minor difference.

Function Declarations: def functionName ([list of parameters]) : [return type]

Function Definitions:

```
def functionName ([list of parameters]) : [return
type] = {
 function body
  return [expr]
}
```

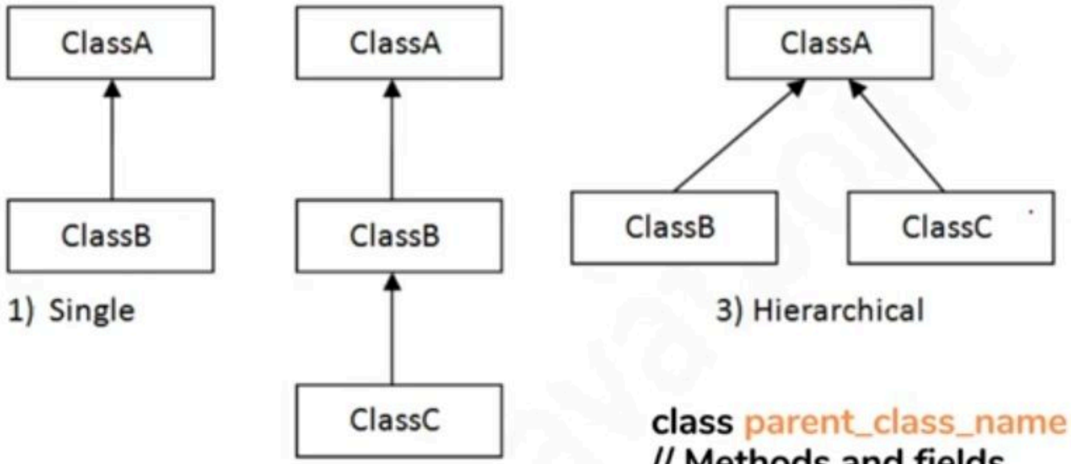
Calling Functions: functionName(list of parameters)

Closures

Difference between a closure function and a normal function is the free variable. A free variable is any kind of variable which is not defined within the function and not passed as the parameter of the function. A free variable is not bound to a function with a valid value. The function does not contain any values for the free variable.

Inheritance

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in Scala by which one class is allowed to inherit the features(fields and methods) of another class.



1) Single

2) Multilevel

3) Hierarchical

```
class parent_class_name extends child_class_name{
// Methods and fields
}
```