Functional Programming

Scala Language

Kasun de Zoysa
kasun@ucsc.cmb.ac.lk

UNIVERSITY OF COLOMBO SCHOOL OF COMPUTING

My First Scala Program

You type

```
scala> 1+1
res2: Int = 2
scala>
Scala shows the result at REPL.
```

Last Week

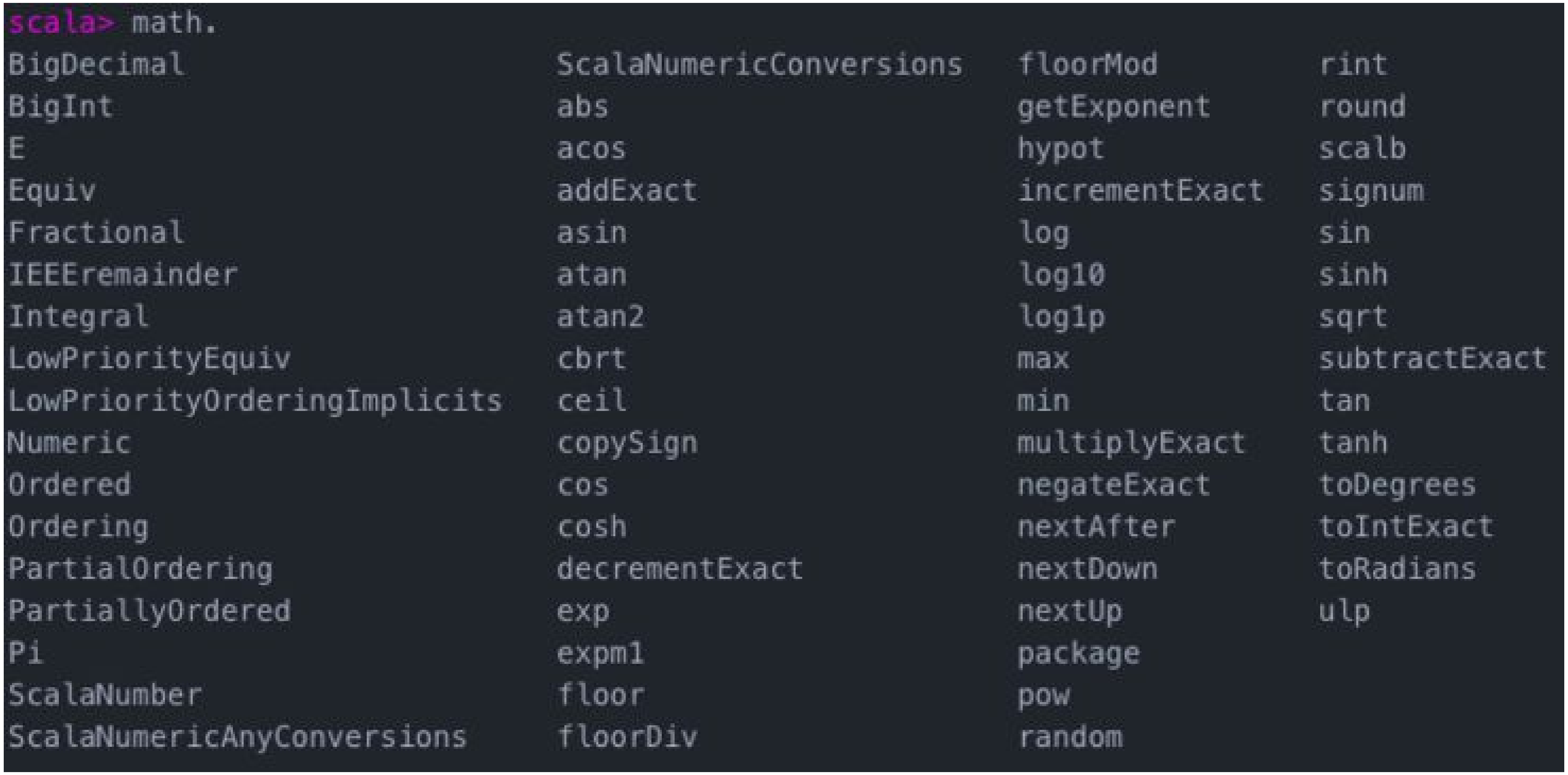
ºF =ºC * 1.8000 + 32.00

```
scala> 35*1.8+32
res0: Double = 95.0
```

The volume of a sphere with radius r is 4/3 Pi r3.
What is the volume of a sphere with radius 5?

```
scala> val r=3.0
r: Double = 3.0
Scala> 4.0/3.0*math.Pi*r*r*r
res1: Double = 113.09733552923255
```

math.<tab>



```
scala> math.
BigDecimal                      ScalaNumericConversions   floorMod         rint
BigInt                          abs                       getExponent      round
E                               acos                      hypot            scalb
Equiv                           addExact                  incrementExact   signum
Fractional                      asin                      log              sin
IEEEremainder                   atan                      log10            sinh
Integral                        atan2                     log1p            sqrt
LowPriorityEquiv                cbrt                      max              subtractExact
LowPriorityOrderingImplicits    ceil                      min              tan
Numeric                         copySign                  multiplyExact    tanh
Ordered                         cos                       negateExact      toDegrees
Ordering                        cosh                      nextAfter        toIntExact
PartialOrdering                 decrementExact            nextDown         toRadians
PartiallyOrdered                exp                       nextUp           ulp
Pi                              expm1                     package
ScalaNumber                     floor                     pow
ScalaNumericAnyConversions      floorDiv                  random
```

Literal, Values and Variables

- A literal (or literal data) is data that appears directly in the source code, like the number 3, the character A, and the text "Aubowan"

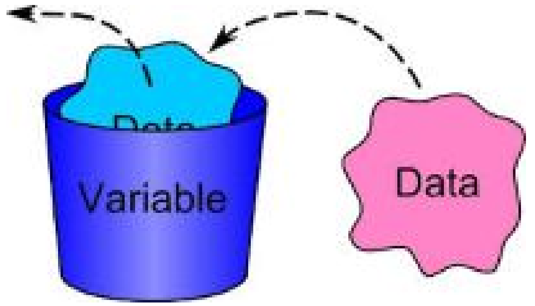
- A value is an **immutable**, typed storage unit. A value can be assigned data when it is defined, but can never be reassigned.

- A variable is a **mutable**, typed storage unit. A variable can be assigned data when it is defined and can also be reassigned data at any time.

Values and Variables

When a number occurs many times in our program(s), we should give it a name using a VALUE /VARIABLE DEFINITION, which associates a name with a value.
Here is how we could give this number a name:

```
scala> val r=3.0
r: Double = 3.0
Scala> 4.0/3.0*math.Pi*r*r*r
res1: Double = 113.09733552923255
```



Define Values

Scala values are defined with the syntax:

val <name>[:<type>] = <literal>

```
scala> val x: Int = 5
x: Int = 5
```

So we will create a value with the **name** x , **type** Int (short for “integer”), and assigned it the **literal** number 5. Value can be defined with or without an explicit type.

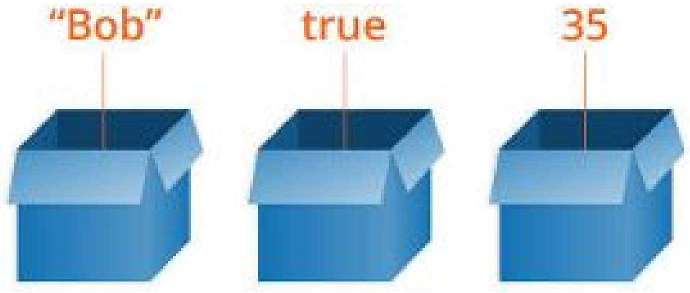
values cannot be reassigned once defined.

```
scala> x=4
<console>:8: error: reassignment to val
       r=4
        ^
```

Define Variables

Syntax: Defining a Variable

```
var <identifier>[: <type>] = <data>

scala> var y: Double = 6.0
y: Double = 6
```



So we will create a variable with the **name** y, **type** Double, and assigned it the **literal** number 6.0.

Variables can be defined with or without an explicit type.

If no type is specified the Scala compiler will use type inference to determine the correct type to assign to your variable.

Values and Variables

- **values cannot be reassigned once defined.**
- **vars can be reassigned.**

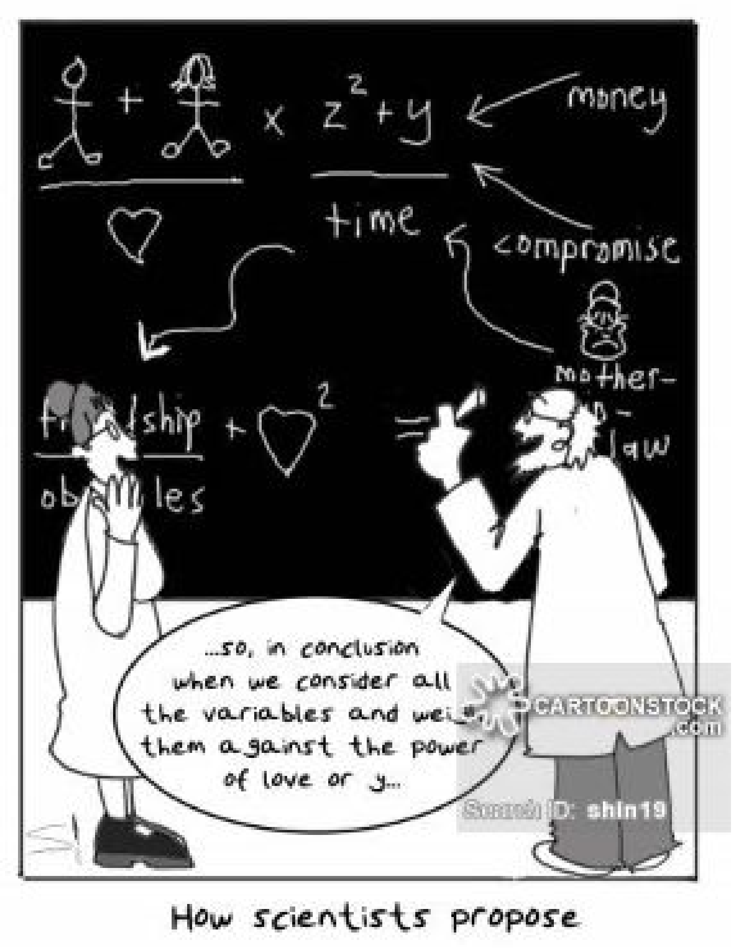
```
scala> var r=3.0

r: Double = 3.0

scala> r=4.5

r: Double = 4.5
```

The data stored in **values** and **variables** in Scala will get automatically deallocated when they are no longer used (garbage collection). There is no ability, or need, to deallocate them manually.

Name

Programmers generally choose names for their variables that are meaningful. They document what the variable is used for.

Variable names can be arbitrarily long. They can contain both letters and numbers, but they have to begin with a letter. It is legal to use uppercase letters, but it is a good idea to begin variable names with a lowercase letter.



Name

- Scala names can use letters, numbers, and a range of special operator characters.

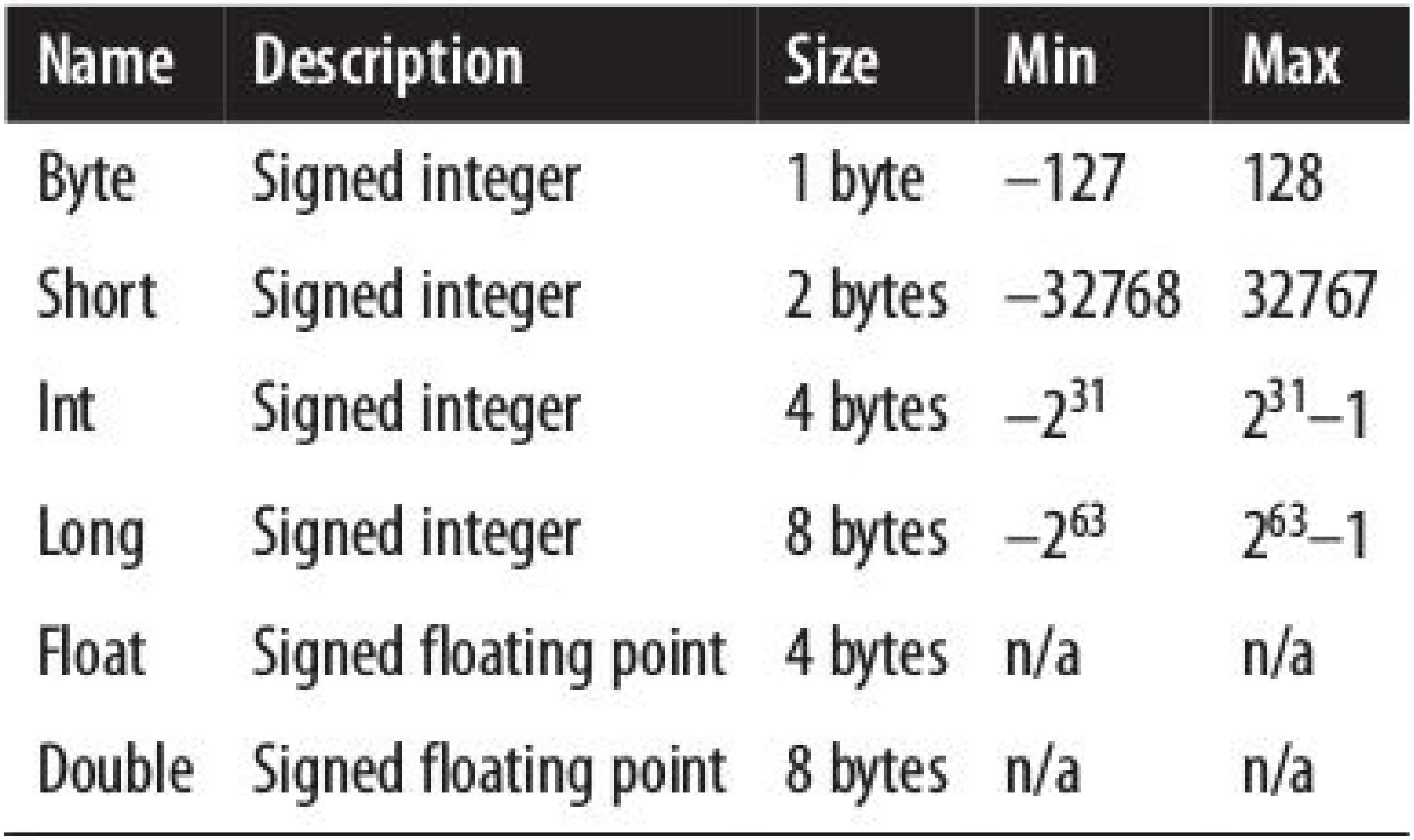
Here are the rules for combining letters, numbers, and characters into valid identifiers in Scala:

- A letter followed by zero or more letters and digits.
- A letter followed by zero or more letters and digits, then an underscore (_), and then one or more of either letters and digits or operator characters.
- One or more operator characters.
- One or more of any character except a backquote, all enclosed in a pair of back quotes.

Types

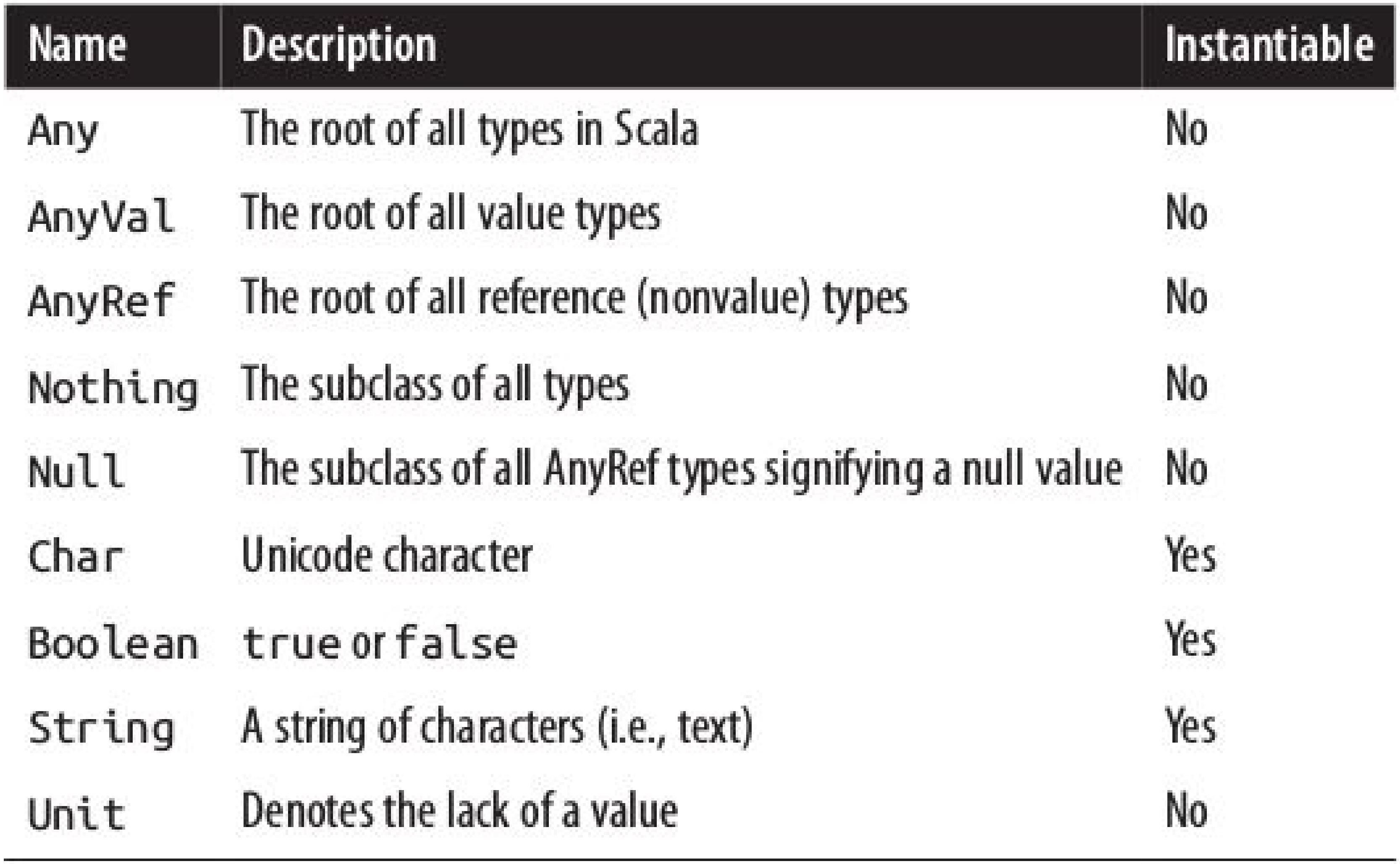
- A type is the kind of data you are working with, a definition or classification of data.
- All data in Scala corresponds to a specific type, and all Scala types are defined as classes with methods that operate on the data.
- Scala has both numeric (e.g., Int and Double) and nonnumeric types (e.g., String) that can be used to define values and variables.
- These core types are the building blocks for all other types including objects and collections, and are themselves objects that have methods and operators that act on their data.

Numeric Data Types



Name	Description	Size	Min	Max
Byte	Signed integer	1 byte	–127	128
Short	Signed integer	2 bytes	–32768	32767
Int	Signed integer	4 bytes	-2^{31}	$2^{31}-1$
Long	Signed integer	8 bytes	-2^{63}	$2^{63}-1$
Float	Signed floating point	4 bytes	n/a	n/a
Double	Signed floating point	8 bytes	n/a	n/a

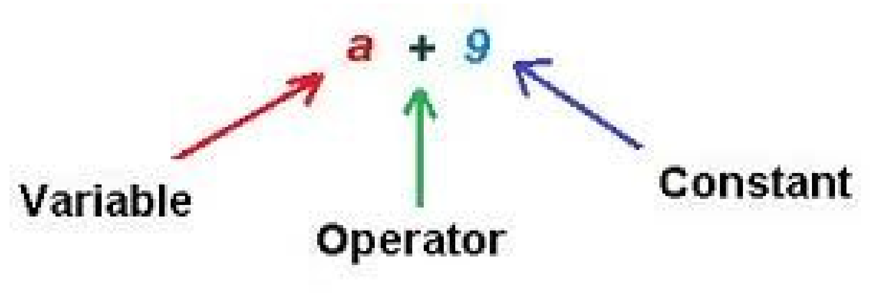
Nonnumeric Data Types



Name	Description	Instantiable
`Any`	The root of all types in Scala	No
`AnyVal`	The root of all value types	No
`AnyRef`	The root of all reference (nonvalue) types	No
`Nothing`	The subclass of all types	No
`Null`	The subclass of all AnyRef types signifying a null value	No
`Char`	Unicode character	Yes
`Boolean`	`true` or `false`	Yes
`String`	A string of characters (i.e., text)	Yes
`Unit`	Denotes the lack of a value	No

Expressions

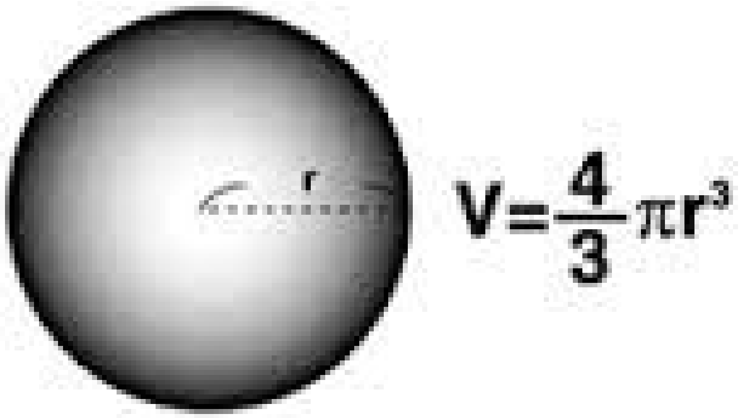
- Expressions provide a foundation for functional programming because they make it possible to return data instead of modifying existing data (such as a variable).
- As noted earlier, an expression is a single unit of code that returns a value.



Expressions

- The entire point of expressions is to return a value that gets captured and used.

```
scala> math.Pi*r*r*r*4/3
res1: Double = 381.7035074111598
```



Cubic volume of a sphere

Expressions Block

- Multiple expressions can be combined using curly braces ({ and }) to create a single expression block.
- An expression has its own scope, and may contain values and variables local to the expression block.
- The last expression in the block is the return value for the entire block.

```
scala> val amount = { val x = 5 * 20; x + 10 }
amount: Int = 110
```

Variables and Expressions

Expressions that contain variables are rules that describe how to compute a number when we are given values for the variables.

A program is such a rule. It is a rule that tells us and the computer how to produce data from some other data.

Large programs consist of many small programs and combine them in some manner. It is therefore important that programmers name each rule as they write it down.

Conditional Statement

In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly.

Conditional statements give us this ability. The simplest forms are the if statement and if .. else statements

The boolean expression after the if statement is called the condition. If it is true, then the indented statement gets executed. If not, nothing happens.

If.. Else Expression Blocks

- The If..Else conditional expression is a classic programming construct for choosing a branch of code based on whether an expression resolves to **true** or **false**.

- **Syntax: Using an If Expression**

  if (<Boolean expression>) <expression>

- **Syntax: Using an If else Expression**

  if (<Boolean expression>) <expression> else <expression>

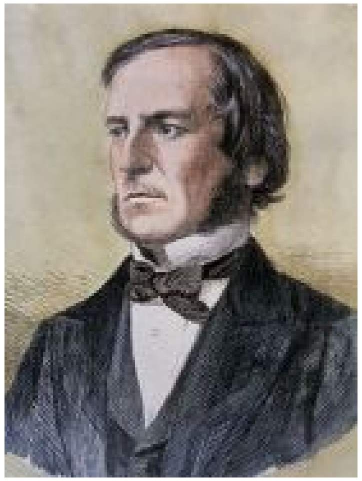
- The term Boolean expression here indicates an expression that will return either true or false .

Examples

```
scala> val x = 10; val y = 20
x: Int = 10
y: Int = 20

scala> val max = if (x > y) x else y
max: Int = 20
```

Boolean Expression

A boolean expression is an expression that is either **true** or **false**.



George Boole
November 2, 1815 - December 8, 1864

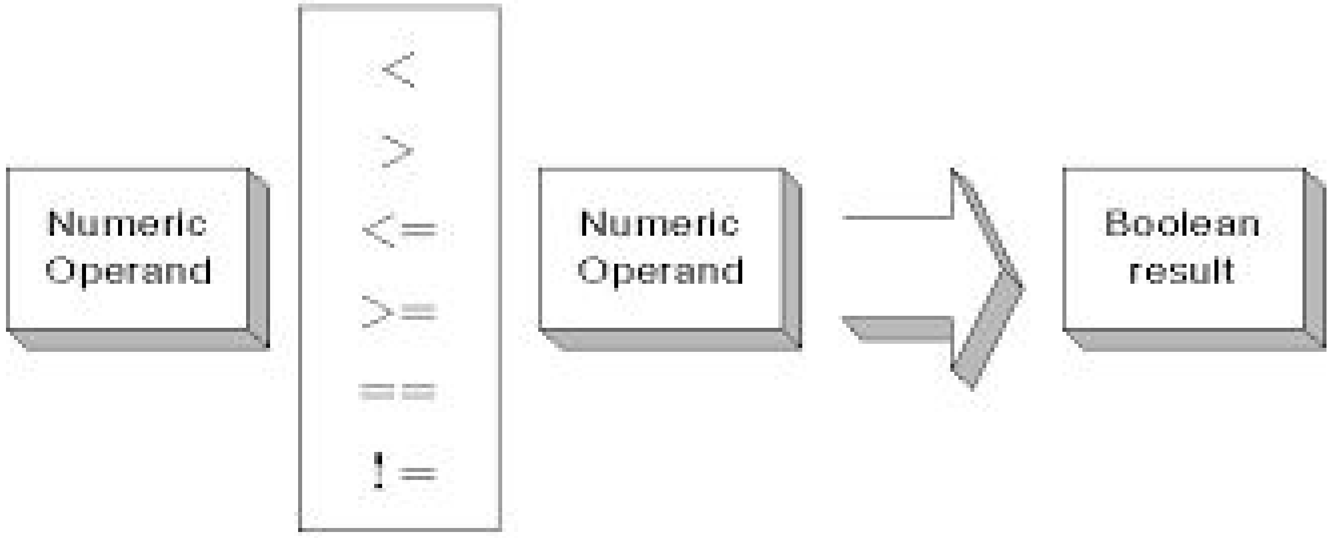
Boolean Expression

The following examples use the Scala **==** operator, which compares two operands and produces **True** if they are equal and **False** otherwise:

```
scala> var x:Int=10
x: Int = 10

scala> var y:Int=9
y: Int = 9

scala> x==y
res4: Boolean = false
```

```
scala> var y:Int=10
y: Int = 10

scala> x==y
res5: Boolean = true
```

Relational Operators

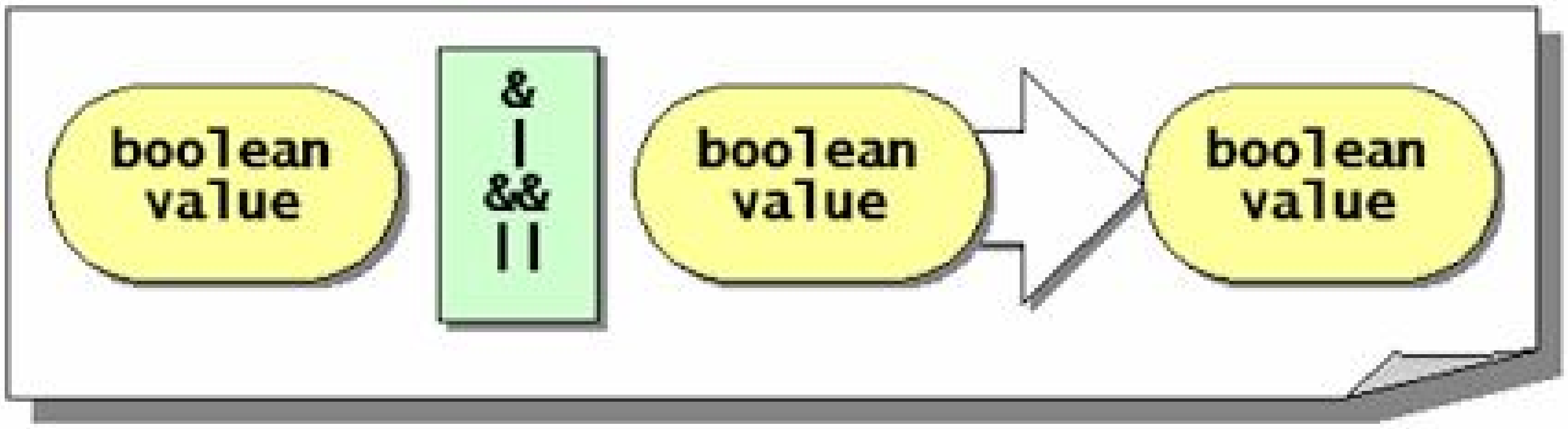


In computer science, a relational operator is a programming language construct or operator that tests or defines some kind of relation between two entities.

Scala: Relational Operators

- == Checks whether the two operands are equal or not and returns true if they are equal.
- != Checks if the two operands are equal or not and returns true if they are not equal.
- > Checks if the first operand is greater than the second and returns true if the first operand is greater than the second operand.
- < Checks if the first operand is lesser than the second and returns true if the first operand is lesser than the second operand.
- >= Checks whether the first operand is greater than or equal to the second operand and returns true if the first operand is greater than or equal to the second operand.
- <= Checks whether the first operand is lesser than or equal to the second operand and returns true if the first operand is lesser than or equal to the second operand.

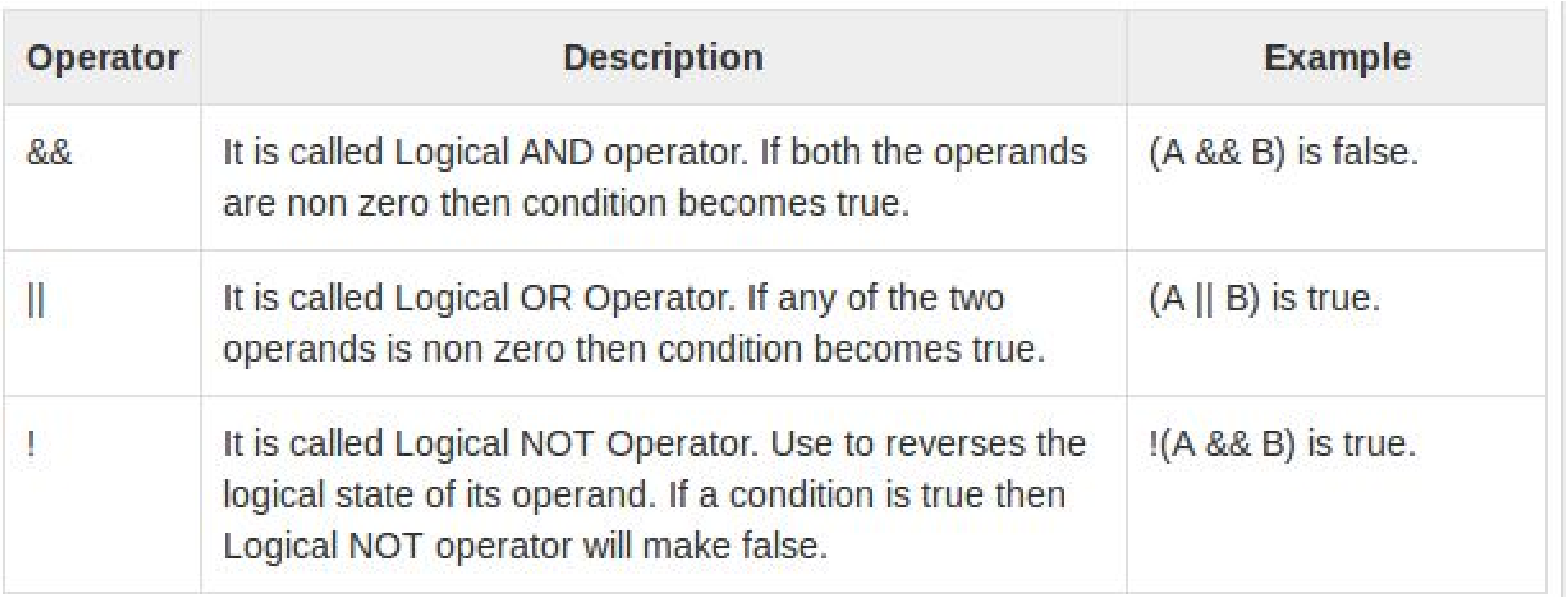
Logical Operators



Operator	Meaning
Not	Test if value is NOT something
And	Test for more than one condition
Or	Test if the value is either OR something
Xor	Test if one and only one value is true

Scala: Logical Operators

The following logical operators are supported by Scala language. For example, assume variable A holds 1 and variable B holds 0, then :



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

What are the results of the following Scala conditions?

```
4>3 && 10 <=100
4 < 3 || 10 == 100
!(2==3)
```

What are the results of the following Scala conditions if x =5 and y=10?

```
var x=5; var y=10
x: Int = 5
y: Int = 10

x-5<=0 || x==y
x<y && y>100
x*x == y+15 && y <100
```

What are the results of the following Scala conditions?

```
scala> 4>3 && 10 <=100
res0: Boolean = true

scala> 4 < 3 || 10 == 100
res2: Boolean = false

scala> !(2==3)
res5: Boolean = true
```

```
scala> var x=5; var y=10
x: Int = 5
y: Int = 10

scala> x-5<=0 || x==y
res17: Boolean = true


scala> x<y && y>100
res7: Boolean = false

scala> x*x == y+15 && y <100
res16: Boolean = true
```

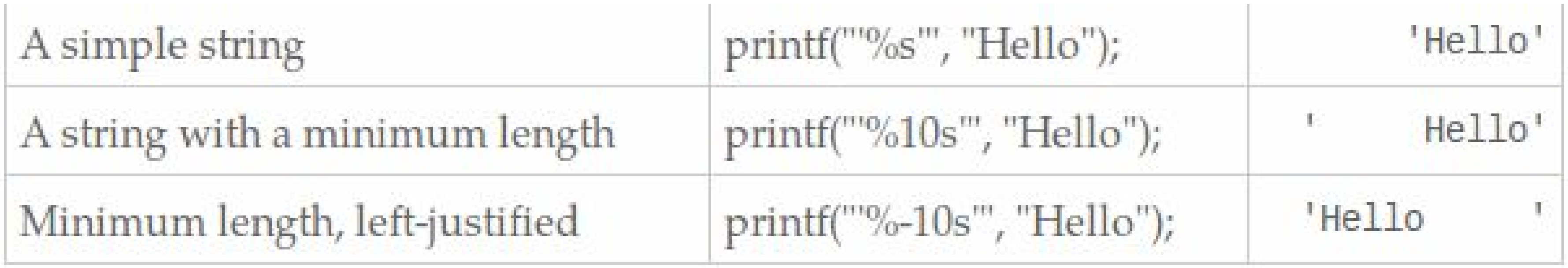
printf

The most commonly used format specifiers are:

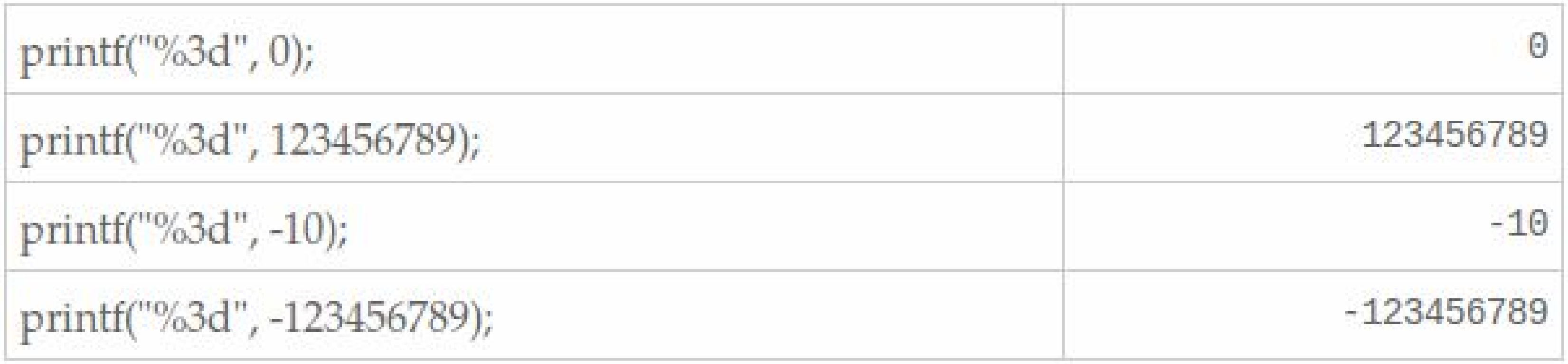
- %s for strings
- %b for bools
- %i for ints
- %f for floats

\b	backspace
\f	form feed
\n	newline, or linefeed
\r	carriage return
\t	tab
\\	backslash

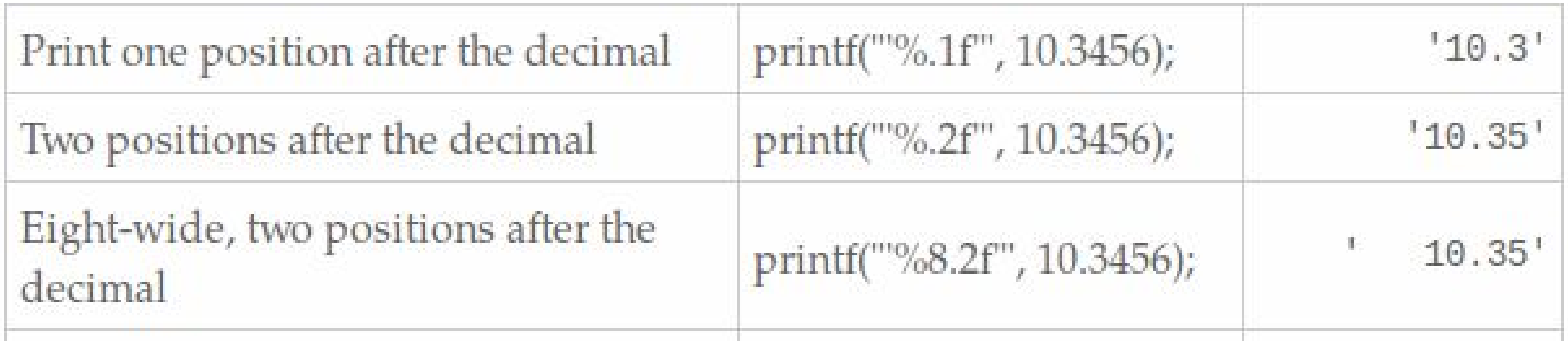
printf



A simple string	printf("'%s'", "Hello");	`'Hello'`
A string with a minimum length	printf("'%10s'", "Hello");	`'     Hello'`
Minimum length, left-justified	printf("'%-10s'", "Hello");	`'Hello     '`



printf("%3d", 0);	`  0`
printf("%3d", 123456789);	`123456789`
printf("%3d", -10);	`-10`
printf("%3d", -123456789);	`-123456789`



Print one position after the decimal	printf("'%.1f'", 10.3456);	`'10.3'`
Two positions after the decimal	printf("'%.2f'", 10.3456);	`'10.35'`
Eight-wide, two positions after the decimal	printf("'%8.2f'", 10.3456);	`'   10.35'`

Reading from the Terminal

To read input from the terminal, there are a number of readXXX-methods.

- **readLine()** reads a line and returns it as a string.
- **readLine(prompt)** prints a prompt and then reads a line.
- **readInt()** reads a line and returns it as an integer.
- **readDouble()** reads a line and returns a floating point number.
- **readBoolean()** reads a line and returns a Boolean ("yes", "y", "true", and "t" for true, and anything else for false).
- **readChar()** reads a line and returns the first character.

The methods are part of the **scala.io.StdIn** object, so you have to import them before you can use them

readLine()

```
import scala.io.StdIn.readLine

object ReadName extends App {
      printf("Enter Your Name => ")
      var name=readLine()
        printf("%s %s \n","Ayubowan",name)
}
```

String

- The String type represents “**strings**” of text, one of the most common core types in any programming language.
- Like numeric types, the String type supports the use of math operators. For example, use the equals operator (==) to compare two String values.

- **What do you expect this to print?**

```
scala> val s1="Aubowan "+"Kasun"

scala> val s2="Aubowan "*5+"Kasun"

scala> s1==s2
```

String Methods

```
+                     concat              isInstanceOf          startsWith
asInstanceOf          contains            lastIndexOf           subSequence
charAt                contentEquals       length                substring
chars                 endsWith            matches               toCharArray
codePointAt           equalsIgnoreCase    offsetByCodePoints    toLowerCase
codePointBefore       getBytes            regionMatches         toString
codePointCount        getChars            replace               toUpperCase
codePoints            indexOf             replaceAll            trim
compareTo             intern              replaceFirst
compareToIgnoreCase   isEmpty             split
```

```
scala> var s1=scala.io.StdIn.readLine("Enter Your Name : ")
Enter Your Name : s1: String = kasun de zoysa

scala> s1.toUpperCase()
res1: String = KASUN DE ZOYSA

scala> s1.length()
res2: Int = 14

scala> s1.contains("kasun")
res3: Boolean = true
```

Comments

As programs get bigger and more complicated, they get more difficult to read. Formal languages are dense, and it is often difficult to look at a piece of code and figure out what it is doing, or why.

For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing.

These notes are called comments , and they start with The // symbol for Scala.

Discussion