SCALA

THE LANGUAGE

- Statically typed
- Runs on JVM (mix Java and SCALA)
- OO & Functional

SBT (SCALA BUILD TOOL)

- Compile, run, test!
- It comes with REPL (Read-Eval_Print loop).
  - Takes single input, executes it and returns the result of execution.
- To install sbt scala build tool – Run on command line
  - Install Java.
  - Set environment variable (System variables -> Path) to know the path to bin folder of jdk and jre.
  - If JRE doesn't have Java_Home
  - Set User Variable to have JAVA_HOME -> use path for the jdk folder.
  - Then search scala download on google, or go to https://www.scala-lang.org/download/scala2.html
  - The default installation is now version 3! But you can launch version 2.

Install Scala with **cs setup** (recommended)

To install Scala, it is recommended to use `cs setup`, the Scala installer powered by Coursier. It installs everything necessary to use the latest Scala release from a command line:

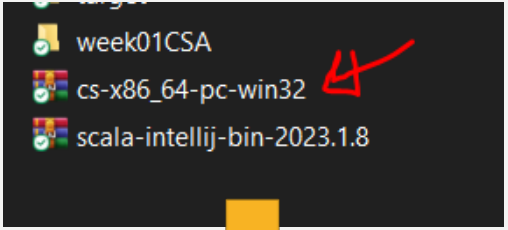
macOS | Linux | Windows | Other

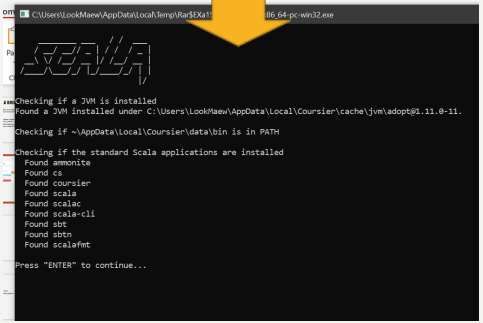
Download and execute the Scala installer for Windows based on Coursier, and follow the on-screen instructions.

Testing your setup

If you are just beginning your journey with Scala, we recommend that you read our getting started guide, which expands upon these details, teaching you how to build your first Scala project:

GET STARTED WITH SCALA





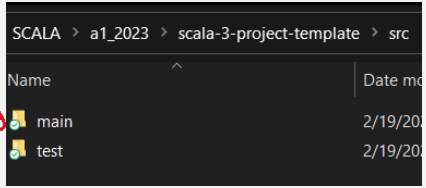

```
E:\Dropbox\teaching\ProgLangSlides\SCALA>scala -version
Scala code runner version 3.2.2 -- Copyright 2002-2023, LAMP/EPFL

E:\Dropbox\teaching\ProgLangSlides\SCALA>
```

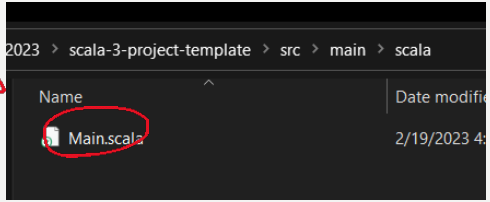
FOR OLD VERSION, YOU MAY NEED TO

- Copy path for bin folder of sbt.
- Set it as environment variable (System variables -> Path).

YOUR FIRST SCALA PROJECT

- Let's create folder a1_2023.
- Cd into the folder then type "sbt new scala/scala3.g8" -> Scala 3 project
- (or "sbt new scala/hello-world.g8" -> Scala 2 project (just in case)
- You have to wait!

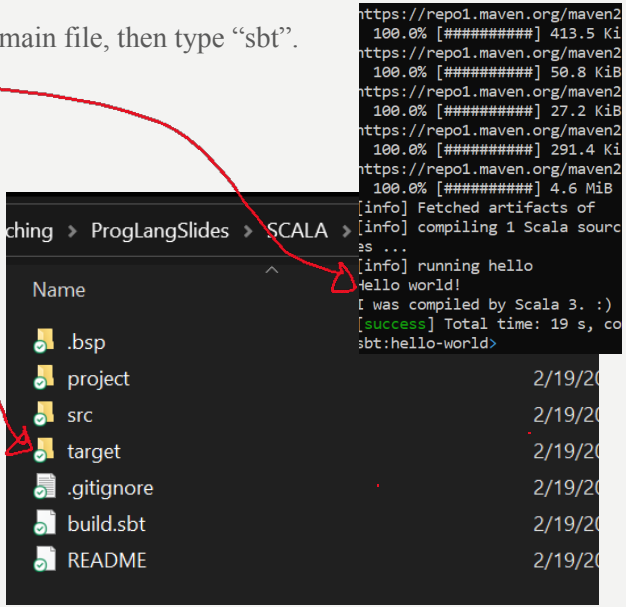




```
@main def hello: Unit =
  println("Hello world!")
  println(msg)

def msg = "I was compiled by Scala 3. :)"
```

GETTING READY TO RUN

- Go into folder of your main file, then type “sbt”.
- Then type “run”
-



Temp files are stored. Scala is a compiled language so it needs to create a code file in order to run.

DATA TYPES

Java

Boolean	true or false
Byte	8 bit signed value
Short	16 bit signed value
Char	16 bit unsigned Unicode character
Int	32 bit signed value
Long	64 bit signed value
Float	32 bit IEEE 754 single-precision float
Double	64 bit IEEE 754 double-precision float
String	A sequence of characters

Void ~

Unit	Corresponds to no value
Null	null or empty reference
Nothing	subtype of every other type; includes no
Any The	supertype of any type; any object is of
AnyRef	The supertype of any reference type

DECLARING VARIABLES (EXIT AND THEN TYPE "SCALA")

- 1. using **var**
  - **This creates a normal (modifiable) variable.**

Separate type and name of a variable.

Type

```
scala> var a : Int =5
var a: Int = 5
```

Don't need a semicolon at the end !

  - **You can then use variable a in other statements**

value of a

```
scala> a
val res0: Int = 5
```

↳ created new val for "a"

```
scala> a + 30
val res1: Int = 35
```

```
scala> a =25
a: Int = 25
```

must declare + init value !

- Variables need to be initialized when they are created.

```
scala> var c: Int
-- [E067] Syntax Error: ----------------------------------------------------
1 |var c: Int
  |    ^
  |Declaration of variable c not allowed here: only classes can have declared but undefined members
  |
  | longer explanation available when compiling with `-explain`
1 error found
```

- But you do not need to give the data type. It can detect the type by the initial value!

default type

```
scala> var c = 1
var c: Int = 1

scala> var d = false
var d: Boolean = false

scala> var e = 1.25
var e: Double = 1.25
```

```
scala> var g = 4.44f
var g: Float = 4.44
```

- 2. using val : final / const
  - This is defining a constant.

```
scala>   val b :Int = 40
val b: Int = 40

scala> b+10
val res2: Int = 50
```

b is constant, can't change value

```
scala> b=10
-- [E052] Type Error: ----------------------------------------
------
1 |b=10
  |^^^^
  |Reassignment to val b
  |
  | longer explanation available when compiling with `-explain`
1 error found
```

- Initialization of val can be delayed until the first read!

```
scala> lazy val k =8
lazy val k: Int

scala> var v = a + k
var v: Int = 33
```

Not have value yet.

8 now load

The initial value is assigned now.

EXECUTE A BLOCK OF CODE

Return Value : Last line : Return

```
scala> var x = {var h = 22.3; var i = 1; e+h+i}
var x: Double = 24.55
```

Block

```
scala> var y = {var m =5;
     | var n =1;
     | e+m+n}
var y: Double = 7.25
```

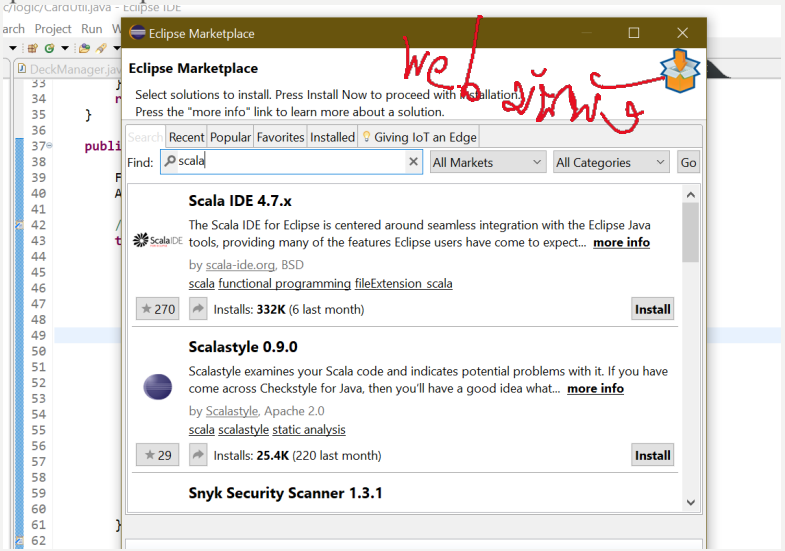
The code can be put on several lines.

```
scala> {val s = 10;
     | a=10
     | s+a
     | }
val s: Int = 10
a: Int = 10
val res3: Int = 20
```

Does not need to give a variable on LHS. It will create a temporary variable to store the result.

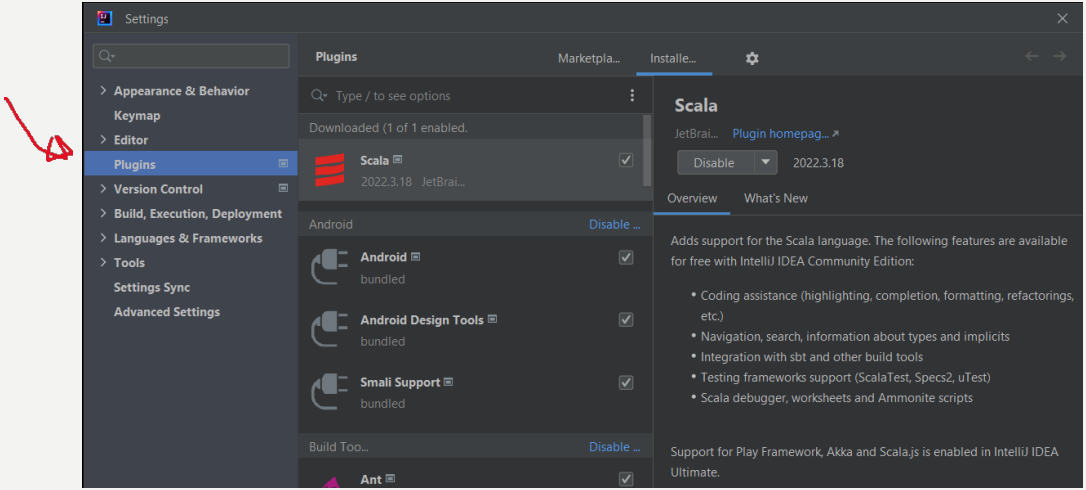
WHAT ABOUT ANY IDE, ECLIPSE?

- Let's install Scala IDE for Eclipse.
- Go to Eclipse Marketplace



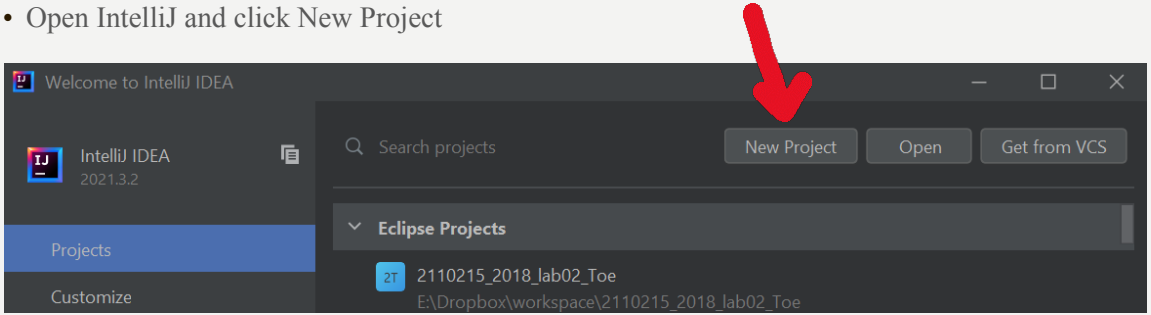
WHAT ABOUT INTELLIJ?

- Install IntelliJ
- Then install Scala plugin



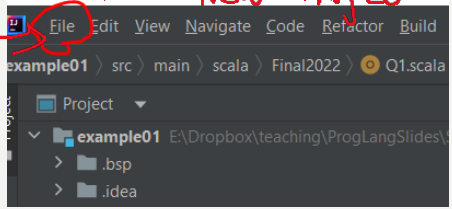
YOUR "HELLO WORLD" PROJECT

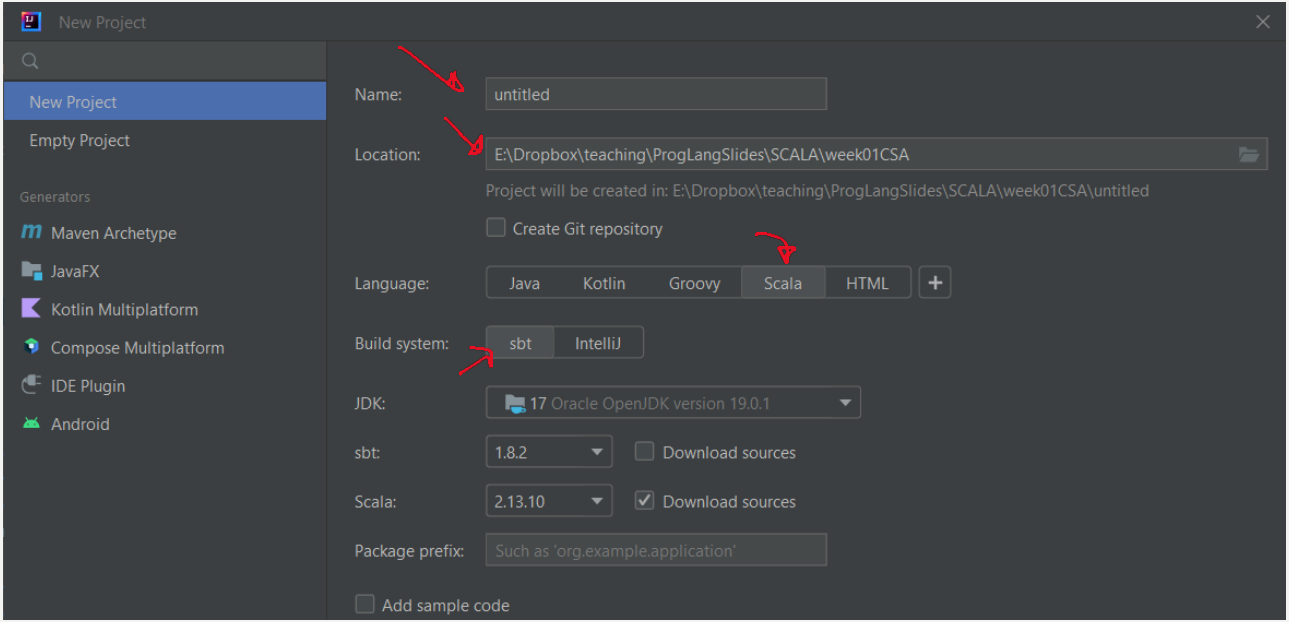
- Open IntelliJ and click New Project

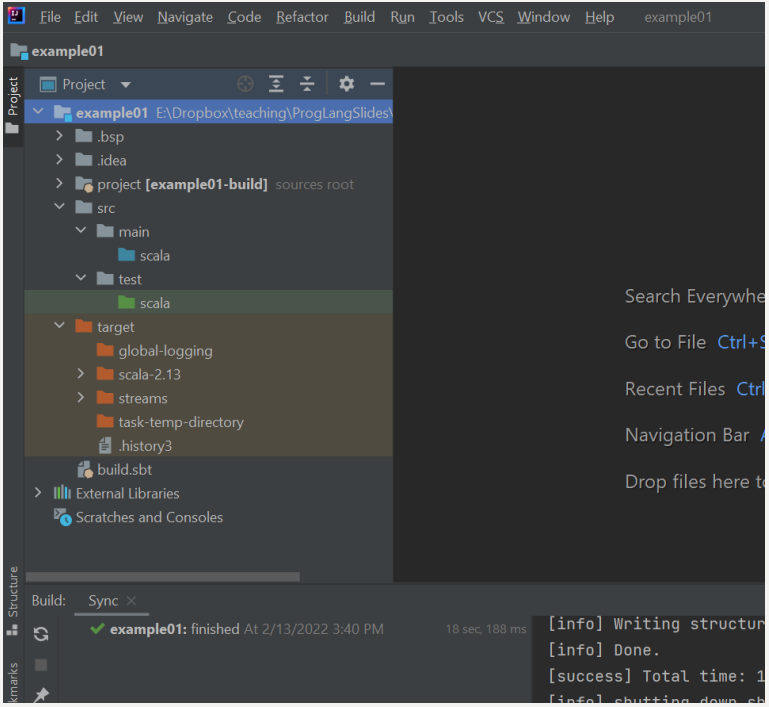


Or

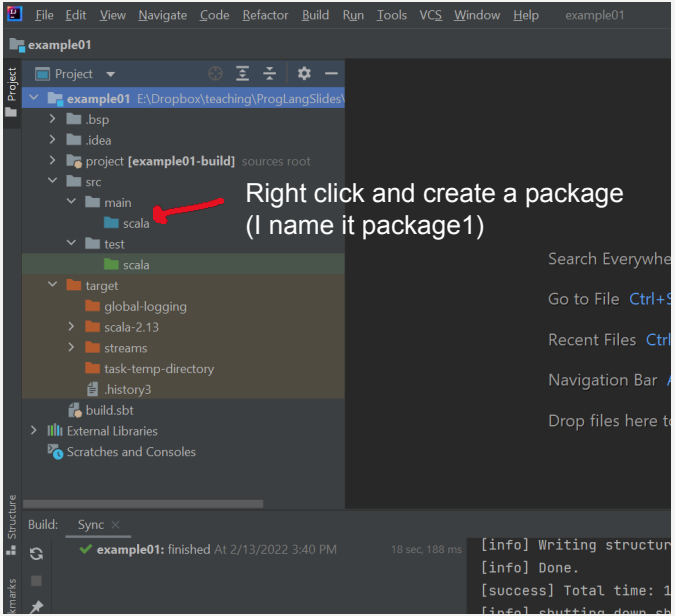
New Project

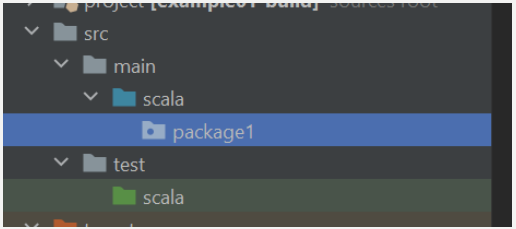



New Project
New Project
Empty Project
Generators
Maven Archetype
JavaFX
Kotlin Multiplatform
Compose Multiplatform
IDE Plugin
Android
Name:
untitled
Location:
E:\Dropbox\teaching\ProgLangSlides\SCALA\week01CSA
Project will be created in: E:\Dropbox\teaching\ProgLangSlides\SCALA\week01CSA\untitled
Create Git repository
Language:
Java
Kotlin
Groovy
Scala
HTML
Build system:
sbt
IntelliJ
JDK:
17 Oracle OpenJDK version 19.0.1
sbt:
1.8.2
Download sources
Scala:
2.13.10
Download sources
Package prefix:
Such as 'org.example.application'
Add sample code

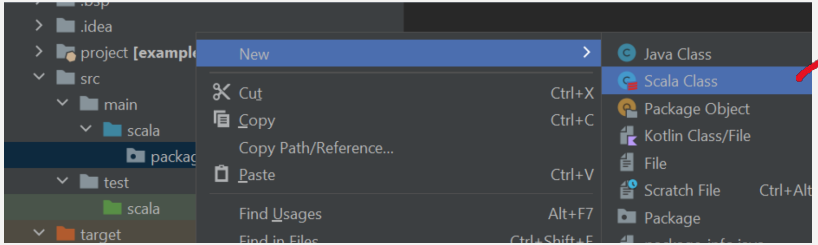

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help example01
example01
Project
example01 E:\Dropbox\teaching\ProgLangSlides\
.bsp
.idea
project [example01-build] sources root
src
main
scala
test
scala
target
global-logging
scala-2.13
streams
task-temp-directory
.history3
build.sbt
External Libraries
Scratches and Consoles
Search Everywhe
Go to File Ctrl+S
Recent Files Ctrl
Navigation Bar
Drop files here t
Build: Sync
example01: finished At 2/13/2022 3:40 PM
18 sec, 188 ms
[info] Writing structur
[info] Done.
[success] Total time: 1

LET'S CREATE A PACKAGE

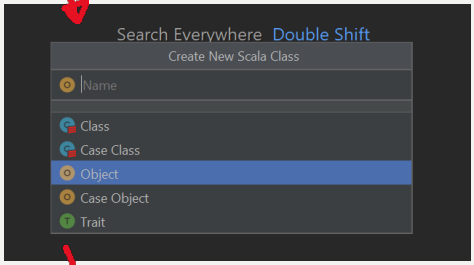


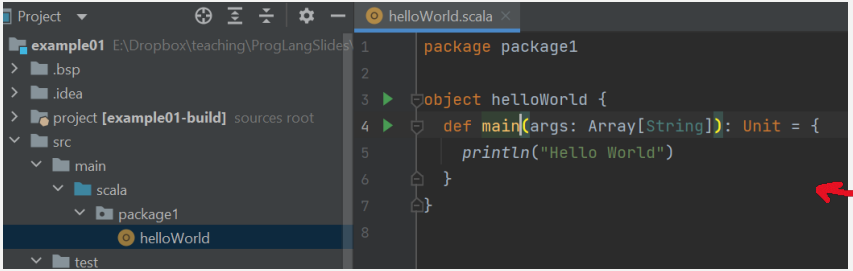


THEN CREATE A SCALA OBJECT THAT HAS “HELLO WORLD”



In the pop up,
change it to object and name it!

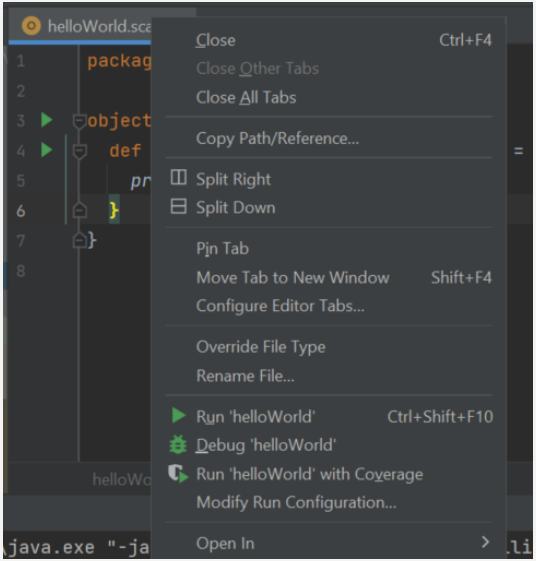




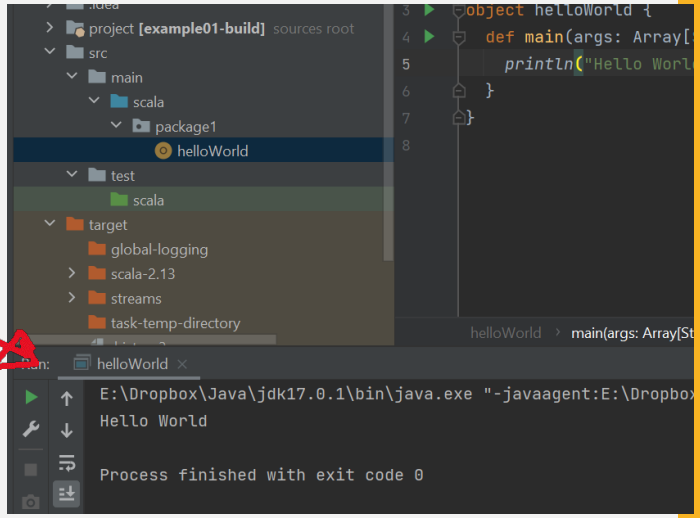
Main method is similar to
Java’s.

NOW WE RUN THE PROGRAM

Right click on the tab or inside the file.
Then choose Run ‘helloWorld’



If you run your program for the first time, it may take a while.



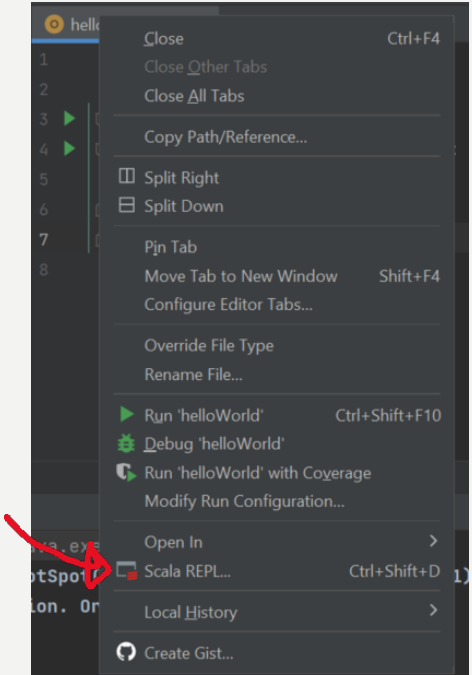
LET'S LOOK AT THE CODE

An instance of class helloWorld. (A class like this cannot have another instance. It is a Singleton!)

```
object helloWorld {
  def main(args: Array[String]){
    println("Hello World")
  }
}
```

Used to define method.

TO RUN REPL IN INTELLIJ




```
scala> object helloWorld {
   def main(args: Array[String]): Unit = {
     println("Hello World")
   }
 }
     |       |       |       | object helloWorld
```

It tells us what is defined.

Include package name if there is one.

Empty array

```
scala> helloWorld.main(Array(""))
Hello World
```

STRING INTERPOLATION

- Concatanation: this is just like Java.

```
object helloWorld {
  def main(args: Array[String]): Unit = {
    var name = "Tanjiro"
    var age  = 15
    println("Hello " + name + ", age =" + age)
  }
}
```

- S string interpolation "$variable"

```
object helloWorld {
  def main(args: Array[String]): Unit = {
    var name = "Tanjiro"
    var age  = 15
    //println("Hello "  + name + ", age =" + age)
    println(s"$name is $age years old.")

  }
}
```

Comment uses //
Or /* */ just like in Java.

```
E:\Dropbox\Java\jdk17.0.1
Tanjiro is 15 years old.
```

- F string interpolation (type safe)

"$variable:l.type"

```
object helloWorld {
  def main(args: Array[String]): Unit = {
    var name = "Tanjiro"
    var age  = 15.5
    //println("Hello "  + name + ", age =" + age)
    //println(s"$name is $age years old.")
    println(f"$name%s is $age%d years old.")
  }
}
```

Note that the type here is not int.

```
example01: build failed At 2/13/2022 7:12 PM with 1  1 sec, 493 ms
  Chart
  helloWorld.scala src\main\scala\package1 1 error
    type mismatch; :9
```

```
E:\Dropbox\teaching\ProgLangSlides\SCALA\exam
type mismatch;
 found   : Double
 required: Int
    println(f"$name%s is $age%d years old.")
```

- Raw string interpolation

raw"——"

```
println("Kamado \nNezuko")
println(raw"Kamado \nNezuko")
```

```
Run:  Scala REPL ×  hello\
Kamado
Nezuko
Kamado \nNezuko
```

IF-ELSE

```scala
object IfElseExample {
  def main(args: Array[String]): Unit = {
    var age = 15
    var x = 3;
    var message = ""
    if(age == 15){
      message = "age is 15"
      x += 1
    } else{
      message = "age is NOT 15"
      x -= 1
    }
    println(message)
    println(x)
  }
}
```

Don’t forget to initialize!

```
E:\Dropbox\Ja
age is 15
4
```

A MORE COMPLEX IF EXAMPLE

```
def main(args: Array[String]): Unit = {
  var a = 15
  var b = 3;
  var c = 20
  if(a<16){
    if(b>3 && c <=20){
      println("case 1.1")
    }else if (b>3 && c ==20){
      println("case 1.2")
    }else if (b>3 && c>20){
      println("case 1.3")
    }else {
      println("case 1.4")
    }
  } else if (a == 16 || b!=4){
    println("case 2.1")
  } else {
    println("case 3.1")
  }
}
```

And, or, not, nested if are just like Java!

IF EXPRESSION

- Very similar to C++

```
object IfExpression {
  def main(args: Array[String]): Unit = {
    var age = 15
    var x = 3;
    var message = ""

    var result = if(age !=15) "age is not 15" else "age is 15"
    println(result)

  }
}
```

if (condition) do1 else do2

```
E:\Dropbox\J
age is 15
```

MATCH (SWITCH STATEMENT)

```
object MatchStatement {
  def main(args: Array[String]): Unit = {
    var x = 45
    x match {
      case 10 => println("x is 10")
      case 20 => println("x is 20")
      case 25 => {
        println("x is 25")
        println("and that's it")
      }
      case 30 => println("x is 30")
      case _ =>
    }
  }
```

default case

- Can be used with other data types like string
- Does not need a "break" statement

doesn't fall through

```
var match {
    case val _ =>
        :
    case _ =>
}
```

Default is doing nothing

MATCH EXPRESSION

```
def main(args: Array[String]): Unit = {
  var x = 25
  var res = x match {
    case 10 => 10.0
    case 20 => 20.0
    case 25 => {
      25.0
      x = 33
    }
    case 30 => 30.0
    case _ =>
  }
  println(res)
  println(x)
}
```

this's return nothing

```
10.0
10
```

```
()
33
```

blank
"no value"

```
()
44
```

MATCH WITH MULTIPLE CASES (FALL THROUGH)

```
object MatchFallThrough {
  def main(args: Array[String]): Unit = {
    var x = 35
    x match {
      case 10 | 20 | 30 | 40 | 50 => println(s"x is $x")
      case 25 | 35 | 45 | 55 => {
        println(s"x is $x")
        println("and that's it")
      }
      case _ =>
    }
  }
}
```

Multiple case "|"

WHILE LOOP

```scala
object WhileLoop {
  def main(args: Array[String]): Unit = {
    var x =0
    while(x<10){
      x += 1     //x++, ++x are NOT allowed in Scala
      println(x)
    }
  }
}
```

DO WHILE

```
def main(args: Array[String]): Unit = {
  var x =0
  do{
    x += 1    //x++, ++x are NOT allowed
    println(x)
  } while(x<0)
}
```

- This loop executes only once!

FOR LOOP

for (x ← val to val) Inclusive

to ↔ until
inclusive exclusive

```
object ForLoop {
  def main(args: Array[String]): Unit = {
    for(x <- 0 ≤ to ≤ 9) { //step by 1 in each iteration
      println(x)   //print 0 to 9
    }
  }
}
```

```
for(x <- 0 ≤ .to( ≤ 9))
```

```
for(x <- 0 ≤ .until( < 10))
```

can also be used.

MULTIPLE RANGE FOR LOOP

```
object MultipleRangeLoop {
  def main(args: Array[String]): Unit = {
    for(x <- 0 ≤ .until( < 5); i <- 0 ≤  to  ≤ 4) {
      println(s"$x , $i")
    }
  }
}
```

Multiple Nested with ";"

```
0 , 0
0 , 1
0 , 2
0 , 3
0 , 4
1 , 0
1 , 1
1 , 2
1 , 3
1 , 4
```

This is like a nested loop.

LOOP ON A LIST

```
object LoopOnList {
  def main(args: Array[String]): Unit = {
    var mylist = List(1,3,5,7)
    for(m <- mylist) {
      println(m)
    }
  }
}
```

each value in list

```
E:\Dropbox\Ja
1
3
5
7
```

FOR LOOP WITH BOOLEAN CONDITION

```
object LoopWithCondition {
  def main(args: Array[String]): Unit = {
    for(x <- 0 ≤ until < 5; if x%2==0) {
      println(x)
    }
    println("------------------")
    var mylist = List(1,3,5,7) do if !
    for(m <- mylist; if m >= 3 ) {
      println(m)   All loop , pass when
    }
  }
}
```

```
0
2
4
------------------
3
5
7
```

- It goes through every value, but only execute code inside the loop if the condition is satisfied.

FOR LOOP EXPRESSION

```
def main(args: Array[String]): Unit = {
  var r1 = for{x <- 0 ≤ until < 5; if x%2==0} yield {
    x
  }
  println(r1)
  println("-------------------")
  var mylist = List(1,3,5,7)
  var r2 = for{m <- mylist; if m >= 3 } yield {
    m
  }
  println(r2)
}
```

express this in containers

return this to container*

! default type is Vector, Unless origin specified !

```
E:\Dropbox\Java\jdk1
Vector(0, 2, 4)
-------------------
List(3, 5, 7)
```

HOW TO WRITE YOUR OWN FUNCTION

```
object Function {
  def main(args: Array[String]): Unit = {
      println(area( width = 2, height = 3))
      println(areaScale(4,5))
  }

  def area(width: Int, height: Int): Int = {
    width * height
  }
  def areaScale(w: Int, h: Int): Int ={
    val w2 = w+1   // w += 1 is not allowed
    val h2 = h+1
    w2*h2  //last statement will be returned (you can use "return")
  }
}
```

Short function

```
def add(x:Int,y:Int):Int = x+y
```

Return type can even be removed if it is known for sure.

FUNCTION BELONGS TO AN OBJECT

```
object Function {
  object Math {
    def addM(x:Int,y:Int):Int = x+y
  }

  def main(args: Array[String]): Unit = {
    println(Function.area( width = 2, height = 3))
    println(areaScale(4,5))
    println(Math.addM(5,3))
  }

  def area(width: Int, height: Int): Int = {
    width * height
  }
```

Nested Object

Class.Method

You can use + here. It's not operator overload. It's just that it can be a function name. And it is used just like a function of object Math.

In fact +, -, * ,/ are not an operator in Scala. They are functions. !

FUNCTION WITH 1 ARGUMENT

```
object Function {
  object Math {
    def addM(x:Int,y:Int):Int = x+y
    def squareM(x:Int):Int = x*x
  }

  def main(args: Array[String]): Unit = {
      println(Function.area( width = 2, height = 3))
      println(areaScale(4,5))
      println(Math.addM(5,3))  //function of object Math
      println(Math squareM 3)  //one argument function call
  }
```

↳ single parameter

FUNCTION CAN HAVE DEFAULT ARGUMENT VALUE

```
object FunctionDefaultArg {
  object Math {
    def addM(x:Int =1, y:Int =1):Int = x+y
    def squareM(x:Int = 1):Int = x*x
  }

  def main(args: Array[String]): Unit = {
    println(Math.addM())
    println(Math.squareM())
  }
}
```

```
E:\Dropb
2
1
```

You can provide some first parameters too

```
println(Math.addM(5))
```

FUNCTION THAT DOES NOT RETURN VALUE

```
object FunctionNotReturnValue {
  def f1(x:Int):Unit ={
    println(s"x is given = $x")
  }
  def main(args: Array[String]): Unit = {
    f1(3)
  }
}
```

void (unit)

FUNCTION AS VARIABLE (ANONYMOUS FUNCTION)

```
object FunctionAsVariable {
  def main(args: Array[String]): Unit = {
    var x = (a:Int, b:Int) => a+b
    var z = (a:Int, b:Int) => {
      var c = a+b
      c*c
    }
    println(x(5,7))
    println(z(2,3))
  }
}
```

Use via value from parameters

```
E:\Drop
12
25
```