**Lesson 1**

Painless was designed and developed specifically for elasticsearch

* It is fast and secure
* It has a groovy like syntax
* It support all java data types
* It exposes many java classes(eg. MATH) and methods

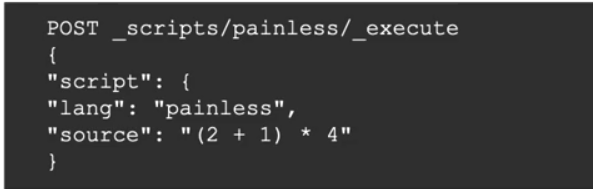
As of Elasticsearch 6.0, painless is the only language supported Other languages, including Groovy, Javascript, and Python are no longer available

**Basics to Painless scripting**

* In this lessonyou will learn to write basic scripting using Painless:
* Scripting Syntax
* Single-Line Expression
* Script parameters
* Statements and blocks
* Primitive data types
* Variables
* Conditionals
* Methods

**Getting Started**

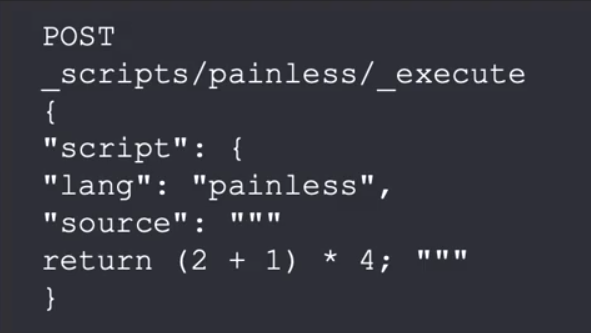
* We can Start writing and running scripts using the \_scripts API with the \_execute end-point



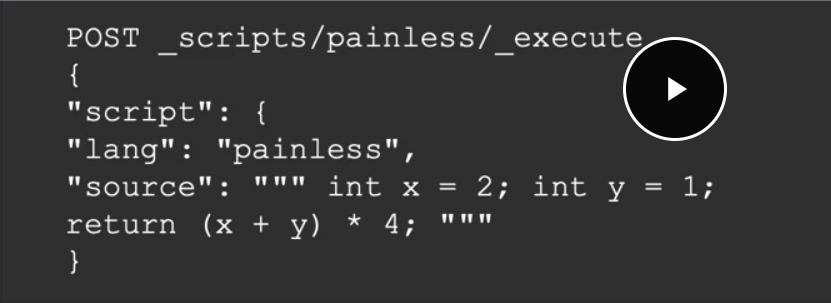
* The source code in thsi case is simple a single-line numerical expression
* Note the use of JSON to encapsulate script - everything in Elasticsearcch is in JSON

**Basics of Painless Scripting**

* Writing complex code as an inline expression is not vey practical
* We can write blocks of code using “”” as the block delimiter. Using a code block, the previous script would now be:



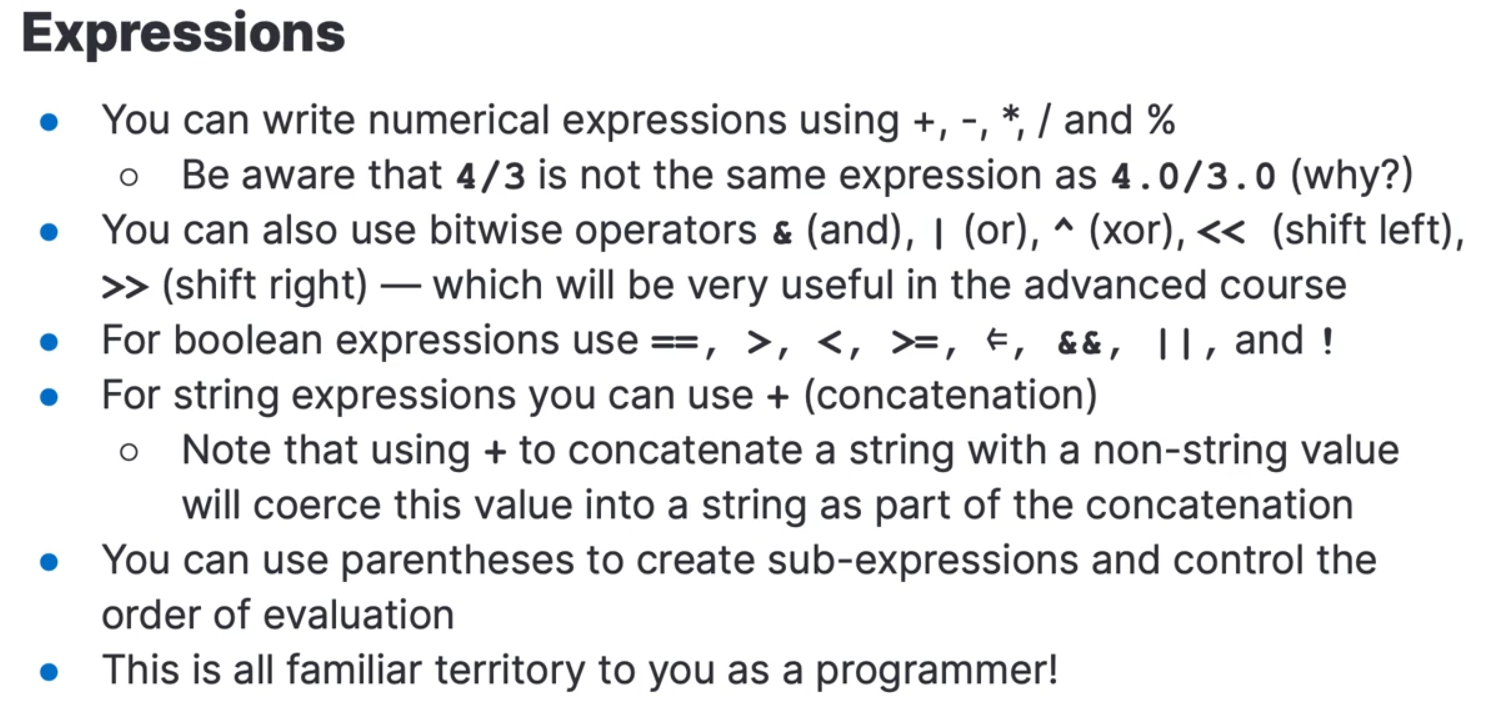
* Variables in painless can be declared by specifying their type, name, and initial value



**Data types**

* Painless has the same data tyes as Java, including
* Primitive types: byte, char, short, int, long, float, double and boolean
* Object wrappers for primitive types: Integer, Long, Float, Double and Boolean String
* Other data types including Date and other
* Data structures
* Arrays, Lists, Maps abd others

**Expressions**



**Maps and ArrayLists**

In Painless, Maps and ArrayLists are particularly easy to build and use — so let's start working with them

* m = ["a": 1, "b": 2] creates a Map m containing two keys, "a" and "b" with values 1 and 2, respectively
* m.get("a") returns 1, the value of key "a"
* m.put("c": 3) adds a new key "c" to m with a value of 3
* m.remove("a") removes the entry in the Map with a key of "a"
* a = [1, 2] creates an ArrayList a containing two values, 1 and 2, in that order

a [0] returns 1, while a [1] returns 2

* a.add (3) adds 3 to the end of a
* a.remove(2) removes the element at position 2 from a, shifting the remaining elements to the left

**Script Parameters**

To make scripts more general and effvient, we can use script parameters

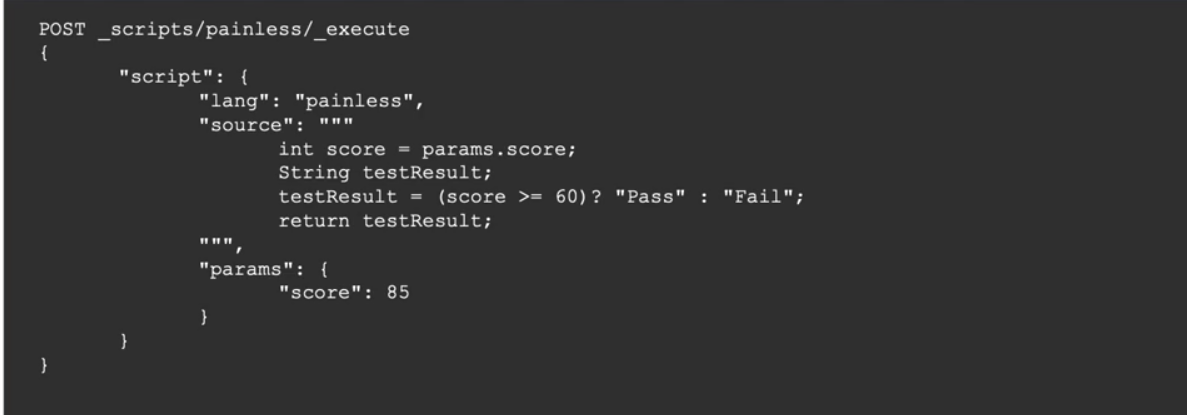
* Value that change from one execution to another should be passed as parameters
* The compiled version of the source will be cached by ES and can be reused with new data

**Conditional Statements**

Painless supports if and if-else conditional statements

**The conditional operator**

Instead of using an if-statement to set the value of a variable, we can use the conditional operator ?



**Loops**

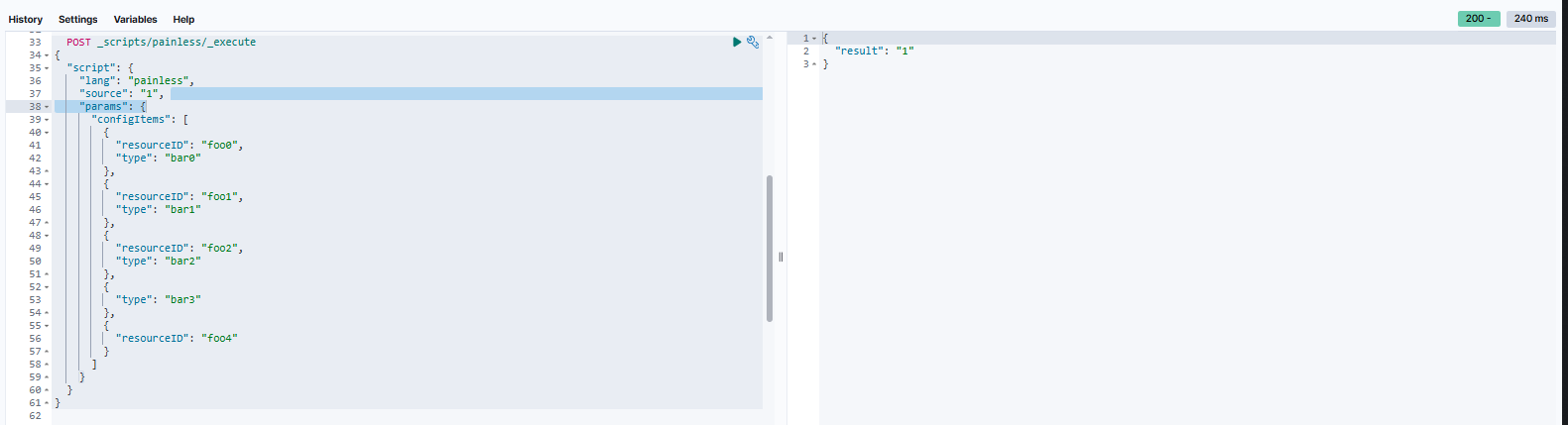
* Paineless supports for, while, do-while and for-each loop
* In the do-while, the condition is evaluated at the end of each iteration - which means it will be run atleast once

**Methods**

* Methods and functions(methods that return value) are supported inside painless scripts
* This script contains a function that find the area of a circle of radius “r”
* Method allow us to write reusable painless code
* Unfortunately, methods cannot be saved outside a script amd must be copy-pasted from script to script

Scripts can be stored in the cluster state and later on invoked by id and with new parameter values

Solved Problem



POST \_scripts/painless/\_execute

{

"script": {

"lang": "painless",

"source": """

Map configItemstoResources(List configItems) {

Map resources = new HashMap();

for (configItem in configItems){

resources.put(

configItem.getOrDefault("resourceID", "Unknow\_resouce\_ID"),

configItem.getOrDefault("type", "Unkwon\_type")

)

}

return resources;

}

return configItemstoResources(params.configItems);

""",

"params": {

"configItems": [

{

"resourceID": "foo0",

"type": "bar0"

},

{

"resourceID": "foo1",

"type": "bar1"

},

{

"resourceID": "foo2",

"type": "bar2"

},

{

"type": "bar3"

},

{

"resourceID": "foo4"

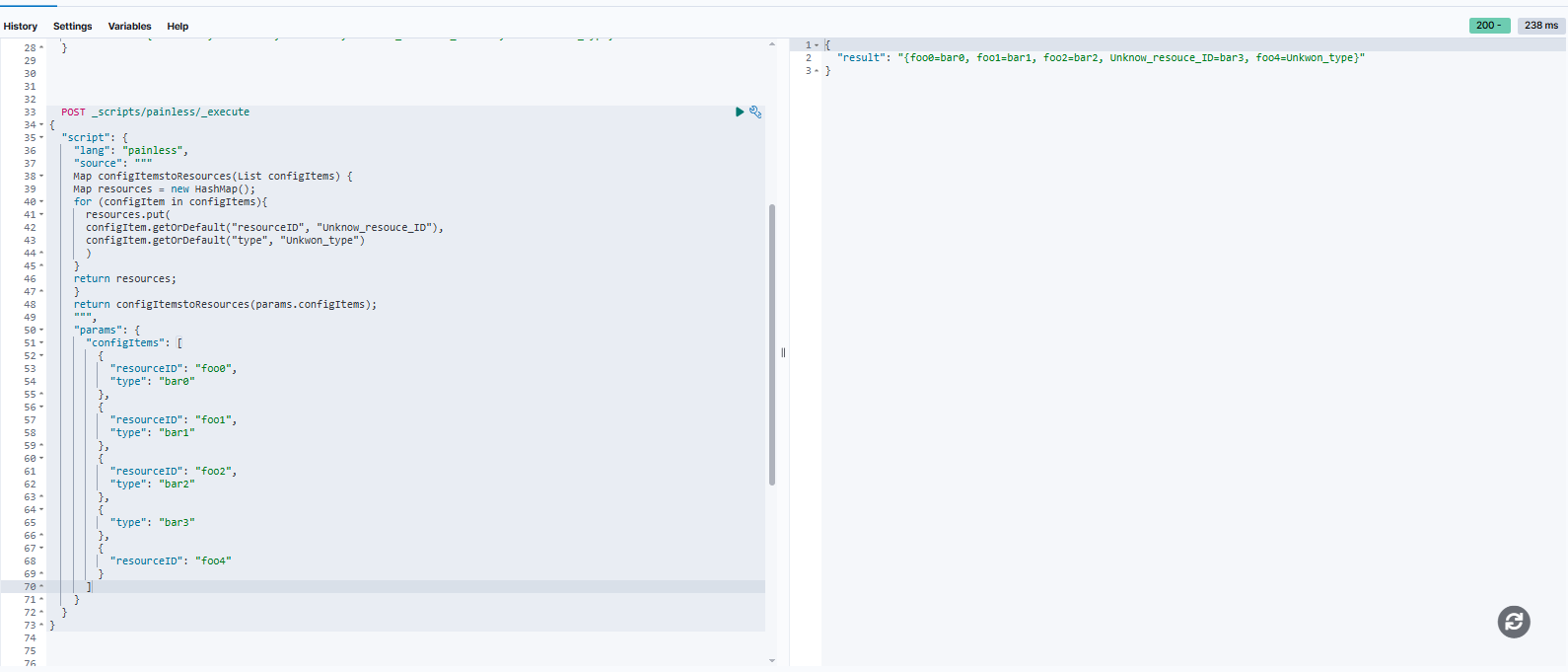
}

]

}

}

}



{

"error": {

"root\_cause": [

{

"type": "script\_exception",

"reason": "compile error",

"script\_stack": [

"""... dObjects){

entries.put(

nestedObject ...""",

" ^---- HERE"

],

"script": """

Map listtoMap(List nestedObjects) {

Map entreis = new HashMap();

for (nestedObject in nestedObjects){

entries.put(

nestedObject.getOrDefault("resourceID", "Unknow\_resouce\_ID"),

nestedObject.getOrDefault("type", "Unkwon\_type")

)

}

return entries;

}

return listtoMap(params.configItems);

""",

"lang": "painless",

"position": {

"offset": 132,

"start": 107,

"end": 157

}

}

],

"type": "script\_exception",

"reason": "compile error",

"script\_stack": [

"""... dObjects){

entries.put(

nestedObject ...""",

" ^---- HERE"

],

"script": """

Map listtoMap(List nestedObjects) {

Map entreis = new HashMap();

for (nestedObject in nestedObjects){

entries.put(

nestedObject.getOrDefault("resourceID", "Unknow\_resouce\_ID"),

nestedObject.getOrDefault("type", "Unkwon\_type")

)

}

return entries;

}

return listtoMap(params.configItems);

""",

"lang": "painless",

"position": {

"offset": 132,

"start": 107,

"end": 157

},

"caused\_by": {

"type": "illegal\_argument\_exception",

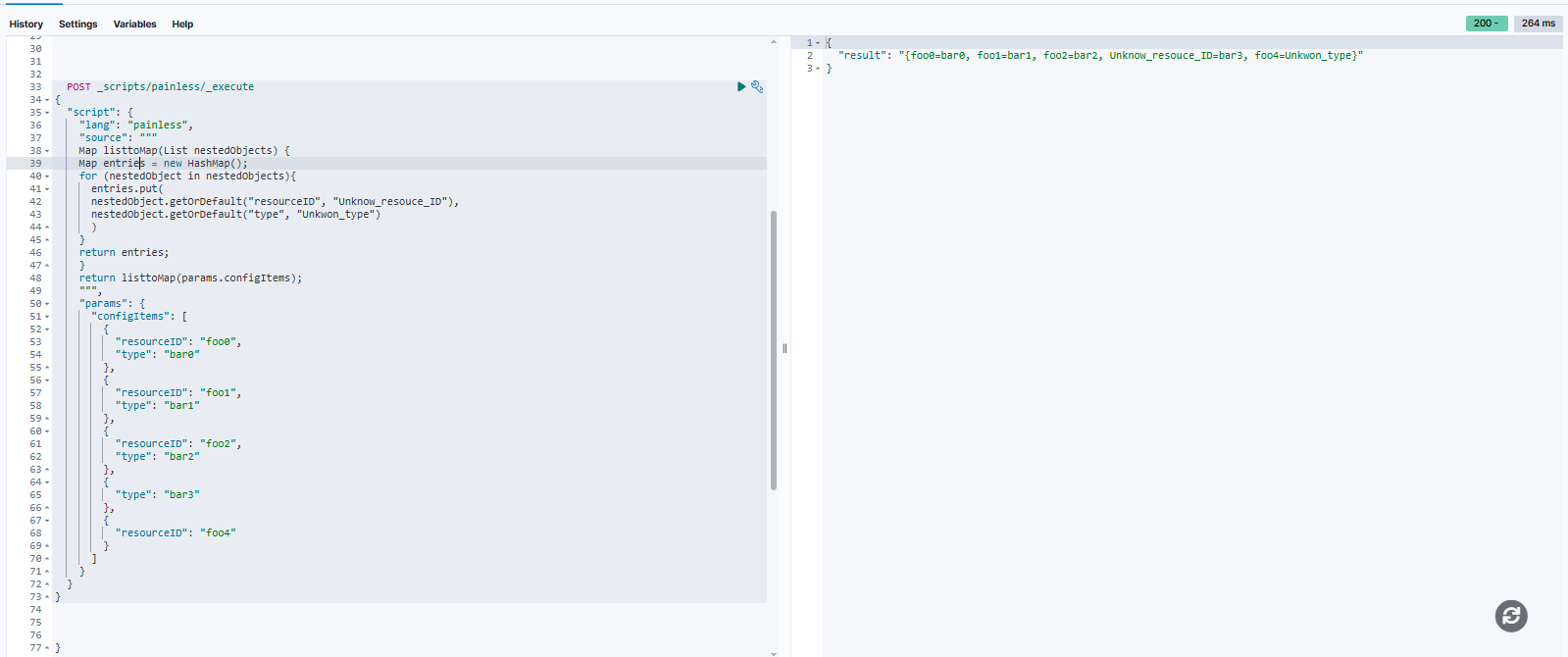
"reason": "cannot resolve symbol [entries]"

}

},

"status": 400

}



**FINAL CODE OF PROBEM**

POST \_scripts/painless/\_execute

{

"script": {

"lang": "painless",

"source": """

Map listtoMap(List nestedObjects, String keyField, String valueField) {

Map entries = new HashMap();

for (nestedObject in nestedObjects){

entries.put(

nestedObject.getOrDefault(keyField, "Unknow\_key"),

nestedObject.getOrDefault(valueField, "Unkwon\_value")

)

}

return entries;

}

return listtoMap(params.configItems, params.keyField, params.valueField);

""",

"params": {

"keyField" : "resourceID",

"valueField" : "type",

"configItems": [

{

"resourceID": "foo0",

"type": "bar0"

},

{

"resourceID": "foo1",

"type": "bar1"

},

{

"resourceID": "foo2",

"type": "bar2"

},

{

"type": "bar3"

},

{

"resourceID": "foo4"

}

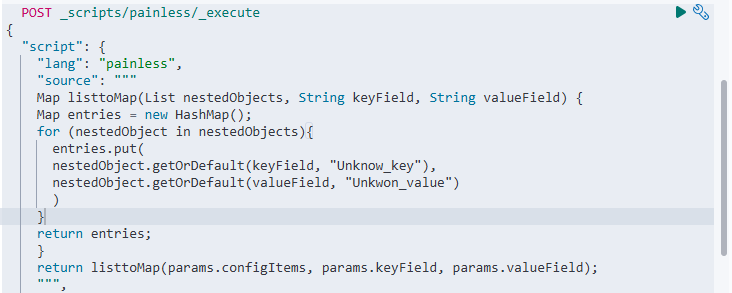
]

}

}

}

}



**OUTPUT**

