



Apache Groovy

ÍNDICE

1. Introducción.
2. Instalación
3. Sintaxis.
 - a. Convenciones Groovy.
 - b. Strings.
 - c. Closures.
 - d. Colecciones de datos.
 - e. Estructuras de control.
4. Operadores
5. Referencias y enlaces de interés.



A large, abstract network graph composed of numerous small, glowing blue dots connected by thin, translucent blue lines. The graph forms a complex, organic shape that resembles a stylized brain or a molecular structure. It is set against a dark, solid blue background.

01.

Introducción

Introducción



Groovy es un lenguaje orientado a objetos, estrechamente ligado a Java, funcionando como un complemento a este, no como un reemplazo. Permite el uso de secuencias de comandos, scripts, El código fuente de Groovy se compila en JVM lo que permite que pueda ejecutarse en cualquier plataforma que ejecute Java Virtual Machine.



02. Instalación

Instalación

NOTA: Tomaremos como base de instalación un equipo con sistema operativo Ubuntu 22.04 LTS.

Prerrequisito

JDK8 +. Soporta hasta Java 16

```
$ sudo apt install default-jdk -y
...
$ java -version
openjdk version "11.0.16" 2022-07-19
OpenJDK Runtime Environment (build 11.0.16+8-post-Ubuntu-0ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.16+8-post-Ubuntu-0ubuntu122.04, mixed mode, sharing)
```



Configuración

Variables de entorno:

GROOVY_HOME, PATH, JAVA_HOME

Integración con ID MVS Code

Spanish Language Pack
Code Runner
Groovy Lint, Format and Fix

Instalación

Instalación de Groovy 4.0 a partir de la distribución de los archivos binarios

```
$ wget
https://www.apache.org/dyn/closer.lua/groovy/4.0.4/distribution/apache-groovy-binary-4.0.4.zip?action=download -O apache-groovy-binary-4.0.4.zip
$ unzip apache-groovy-binary-4.0.4.zip
```

Ejecución

Desde consola, scripts

Invocando al ejecutable **groovy**
Líne shebang `#!/usr/bin/env groovy`

Groovy Shell, **groovysh**

Consola Groovy, **groovyconsole**

Integración IDE



A large, abstract network visualization composed of numerous glowing blue dots (nodes) connected by thin lines (edges). The network forms several distinct, flowing, ribbon-like structures that curve across the frame. Some nodes are brighter than others, creating a sense of depth and highlighting certain points of interest. The overall effect is organic and suggests a complex system or data flow.

03. Sintaxis

Sintaxis



La sintaxis de Groovy consiste en una simplificación del Java, se define sólo lo estrictamente necesario para ejecutar la acción deseada.

```
// ¡Hola Mundo! Usando sintaxis Java en Groovy
class HelloWorldJava {

    static void main(String[] args) {
        System.out.println('¡Hola Mundo!')
    }
}
```

```
// ¡Hola Mundo! Usando sintaxis Groovy
println "¡Hola Mundo!"
```

3.a. Convenciones Groovy

El carácter punto y coma al final de línea es opcional.

Si se debe usar para separar varias sentencias en una única línea

Los caracteres paréntesis para llamar a un método con argumentos es opcional.

Son obligatorios cuando se llama a un método sin argumentos, para que el compilador lo diferencie de una variable

Importa por defecto algunos paquetes y clases

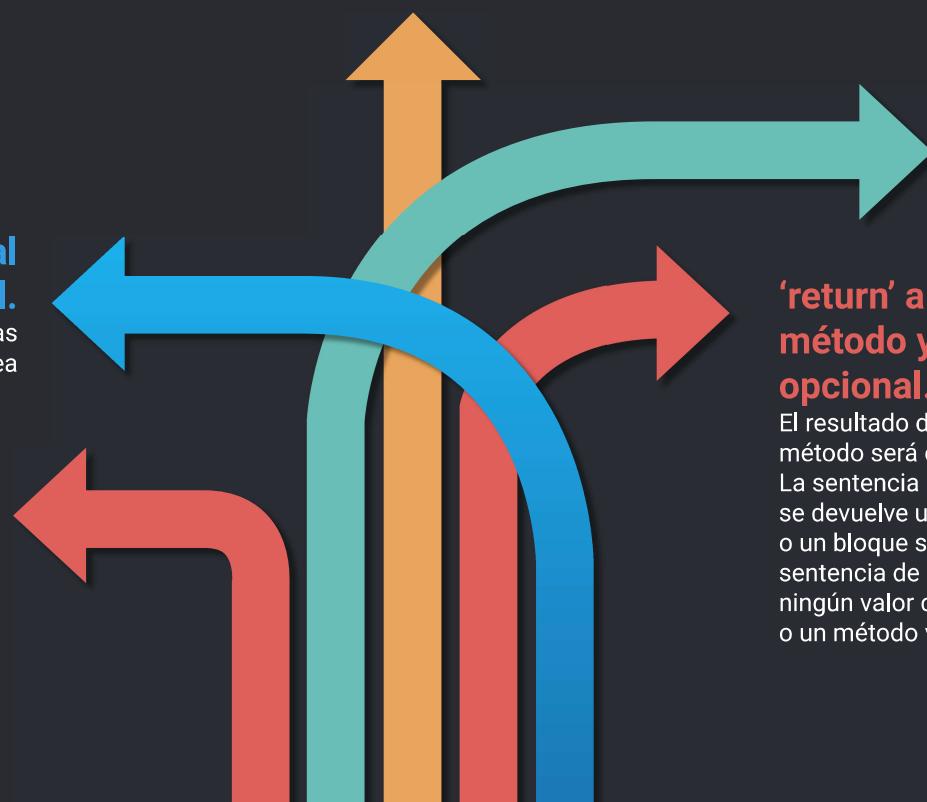
que pueden ser utilizados directamente sin necesidad reflejar la importación en el código

La declaración 'public' de los métodos es opcional.

De forma predeterminada, las clases y los métodos de Groovy son públicos.

'return' al final de un método y de un script es opcional.

El resultado de la última sentencia del método será el valor devuelto por este. La sentencia return es necesaria cuando se devuelve un valor dentro de un bucle o un bloque switch. Si la última sentencia de un método no devuelve ningún valor como una sentencia println o un método void, se devolverá null.



3.a. Convenciones Groovy

Todo es un objeto.

Incluso los tipos primitivos también son tratados como objetos.

Se permite la sobrecarga de algunos operadores.

Operador de referencia segura (?)

Si tenemos una referencia **null** a un objeto no obtendremos **NullPointerException** si usamos el operador de referencia segura (?) al acceder a sus métodos o propiedades. En su lugar obtendremos null.

No es necesario declarar el tipo de una variable cuando le asignamos un valor.

Se asignará el tipo según el valor del objeto

Implícitamente crea métodos getters y setters y proporciona argumentos a los constructores
No es necesario capturar las excepciones.

Todas las expresiones numéricas con decimales son por defecto de tipo 'BigDecimal'.

Comportamiento destinado a evitar la inexactitud de Float y Double. Clases que siguen disponibles si son instanciadas explícitamente.

El concepto de verdadero y falso se extiende a multitud de situaciones.

Cuando se evalúa un valor cero, null, un string vacío, una colección vacía, un array de longitud cero o un StringBuilder/StringBuffer vacío, se obtiene false. En cualquier otra situación, se obtiene true.



3.b. Strings

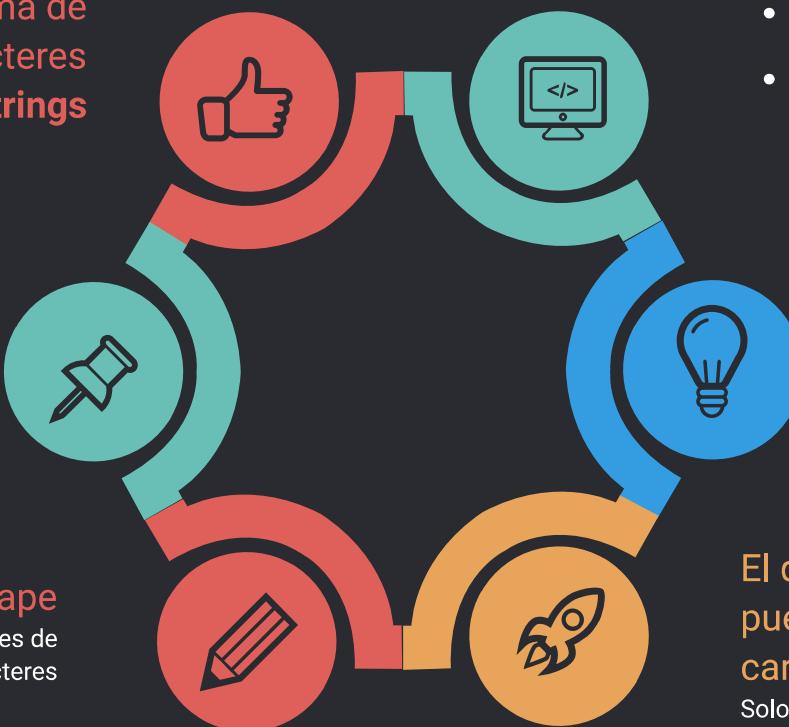
Los literales de texto se representan en forma de cadenas de caracteres llamadas **strings**

Con tres comillas simples o tres comillas dobles

Definimos strings de varias líneas.
Útil para almacenar en variables contenido de archivo de texto.
No es necesario emplear el carácter de escape
Las tres comillas simples no son interpolables

Secuencias de escape

Disponibles dentro de las definiciones de cadenas de caracteres



Dos clases principales de strings

- La clase **java.lang.String**, constructor por defecto comillas simples y dobles.
- La clase **GString**, cadenas interpoladas, reemplazan el marcador de posición por su valor. \${}, \$ para expresiones inequívocas.

El cambio de tipo (casting) lo realiza Groovy automáticamente.

El dólar y la barra inclinada pueden emplearse como caracteres

Solo son necesarios cuando surgen conflictos con el uso especial de esos caracteres.

3.c Closures

Son bloques de código independiente, puede tomar argumentos y devolver valores. Asignables a variables.

Siguen la siguiente sintaxis:

[closureParameters ->] statements]

[closureParameters->] lista opcional de parámetros separados por comas.

statements declaraciones, con cero o más declaraciones.

Con listas de parámetros, requiere el carácter `->` para separar los argumentos de las declaraciones.

Ejemplos de definiciones de closures

```
{ item++ }
{ -> item++ }
{ println it }
{ it -> println it }
{ name -> println name }
{ String x, int y ->
    println "hey ${x} the value is ${y}"
}
{ reader ->
    def line = reader.readLine()
    line.trim()
}
```

Permite preestablecer el valor de un parámetro, de modo que al invocar el closure admite un argumento menos (Técnica de Currying)

Los closures pueden ser pasados como parámetros de función.

```
// Clouser con solo un parámetro
def saludaPlaneta = { nombre ->
    println "¡Hola ${nombre}!" }
saludaPlaneta 'Venus'
// Precarga de parámetro
def planeta = saludaPlaneta.curry('Mercurio')
planeta()
// Otro clouser con solo un parámetro
def cuadrado = { factor ->
    factor * factor
}
```

```
// Clouser con dos parámetros
def multiplicar = { factor01, factor02 ->
    factor01 * factor02
}
// Precarga de parámetro
def doble = multiplicar.curry(2)
def triple = multiplicar.curry(3)
println cuadrado(8)
println doble(8)
println triple(8)
```

```
// Ejemplo de clouser como parámetro
def ssolar = ['Mercurio', 'Venus', 'Tierra', 'Marte', 'Júpiter', 'Saturno', 'Urano',
'Neptuno', '{Plutón?'}]
def mensajePlanetas(sistema, Closure mensaje) {
    for (cceleste in sistema) {
        mensaje cceleste
    }
}
mensajePlanetas(ssolar, saludaPlaneta)
def despidePlaneta = { nombre ->
    println "¡Adios ${nombre}!" }
mensajePlanetas(ssolar, despidePlaneta)
```

3.d Colecciones de datos



3.e Estructuras de control



i. Estructuras condicionales

```
if (...) {...} else if (...) {...} else {...}
```

```
switch (...) {  
    case ...:  
    ...  
    default:  
    ...  
}
```

Operador `? ?:`



ii. Estructuras de bucle

- for
- for con asignación múltiple
- for in
- while
- do/while



iii. Manejo de excepciones

```
try / catch / finally
```



04. Operadores.

Operadores.

Aritméticos.

De asignación.

Relacionales.

Lógicos.

Binarios.

Condicionales

De objetos

De expresiones regulares.

Otros operadores.

El operador de operador especial	Special operator	(*)
Igual que el operador AND	Equal to the AND operator	&
ANDador de patrón	Pattern AND operator	(..)
Banco	Bank	++
Proporciona una forma sencilla de crear una nave espacial.	Provides a simple way to create a spaceship.	(<=>)
Descomponer	Decompose	
XOR	XOR	^
Safari navigation	Safari navigation	?{1}
Subscript	Subscript	[]
Memoria	Memory	[]<=
Transpuesto	Transposed	?{1}^
NOT	NOT	!~
Operador de función de búsqueda	Search function operator	~
División	Division	[]>
Exponente	Exponentiation	[]^
Asociatividad de la memoria	Memory associativity	?{1}^n
Método permuto	Method permuto	<< . &
Número igual que	Number equal to	%>=
Identidad	Identity	?{1}
Matriz inversa	Inverse matrix	[]^-1
Identidad del operador	Identity of the operator	*: =
ROPER	ROPER	[]<=
A la derecha sin signo	No sign on the right	>>
Identificador booleano	Boolean identifier	[]+
requiere una coincidencia	requires a match	[]*
llamada de la cadena de	Call of the string	[]()
entrada:	input:	[]()

05. Referencias y enlaces de interés.



Referencias y enlaces de interés.

- <http://groovy-lang.org/documentation.html>
- <http://groovy-lang.org/semantics.html>
- <https://www.eficode.com/blog/jenkins-groovy-tutorial>
- <http://www.davidmarco.es/archivo/tutorial-groovy>
- <http://www.cheatography.com/davechild/cheat-sheets/regular-expressions/pdf/>
- <https://regexlib.com/CheatSheet.aspx>
- <https://www.ocpsoft.org/tutorials/regular-expressions/java-visual-regex-tester/>



keep in touch

CONTACTO:

Antonio Lledó
alledo@devcenter.es

T. +34 965 68 87 02

www.devcenter.es

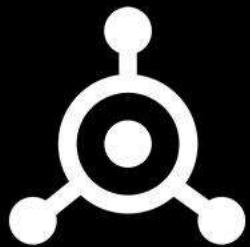
Head Office:

Avenida Cartagena 1, Entlo.
03195 Altet, Alicante

TechHub:

Parque Científico UMH QUORUM IV
Avinguda de la Universitat d'Elx,
03202 Alicante





KEEP
CALM
DEVCENTER
IS
WORKING
ON IT