GRADLE GROOVY DSL CHEAT SHEET

Basic Groovy language features

- it's a scripting language, so write code outside of a class and execute it

```
def myVar = 'Executing as a script'
println myVar //prints 'Executing as a script'
```

- it's dynamic, so use `def` instead of providing a type (see above)
- semicolons at the end of a line are not required (see above)
- brackets are optional when passing parameters to a method, if the method has at least one parameter

```
def multiply(first, second) {
   println first * second
}
multiply 2, 3 //prints '6'
```

Advanced Groovy language features

- define closures using curly brackets `{}`. Closures are blocks of code that can get passed around and executed at a later point.

```
def myClosure = {
   println 'Executing closure'
}
myClosure() //prints 'Executing closure'
```

- if calling a method with brackets, if the last argument is a closure, it can go outside of the brackets (see *Gradle Groovy DSL* for use case).

```
def executeClosure(times, closure) {
   for (int i = 0; i < times; i++) {
       closure()
   }
}
executeClosure(2) { //prints 'Executing closure' twice
   println 'Executing closure'
}
```

Gradle Groovy DSL

- anything you see in the build script operates on the `Project` object

```
version = '0.1.0-SNAPSHOT'
```

Above is a call to the `Project.setVersion(Object version)` method. Groovy knows to call `setVersion` even when we use `=`.

- when you see curly brackets, that's a closure. For example `Project.dependencies(Closure configureClosure)`.

```
dependencies {
    //some dependencies
}
```

- notice above we can leave out brackets when calling dependencies
- when calling methods with zero parameters, include brackets

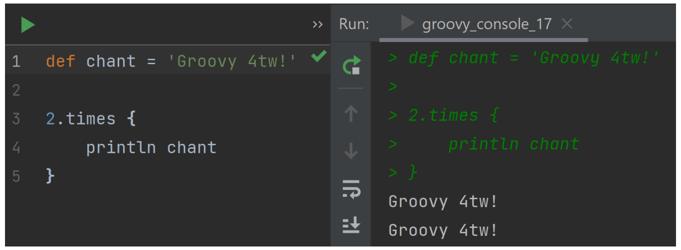
```
repositories {
    mavenCentral()
}
```

- you can put the last closure parameter outside the brackets

```
dependencies {
    implementation('com.google.guava:guava:30.1.1-jre') {
        exclude group: 'com.google.code.findbugs', module: 'jsr305'
    }
}
```

Dive deeper

- try some Groovy in the IntelliJ IDEA Groovy console (*Tools* > *Groovy console*)



```
def chant = 'Groovy 4tw!'

2.times {
    println chant
}
```

```
> def chant = 'Groovy 4tw!'
>
> 2.times {
>     println chant
> }
Groovy 4tw!
Groovy 4tw!
```

Alternatively, just add your Groovy code to *build.gradle* and run `./gradlew` or `gradlew.bat`.

- check out the `org.gradle.api.Project` API documentation to see what methods you can call in the build script https://docs.gradle.org/current/javadoc/org/gradle/api/Project.html
- browse Gradle source in IntelliJ IDEA by control-clicking/pressing F4 on methods in *build.gradle*

ℹ️ to get the wrapper to download the Gradle distribution *with source code*, run `./gradlew wrapper --distribution-type=all`