

# „Zróbmy to szybko” - Groovy i Grails w akcji

Adam Dudczak  
( [adudczak@gmail.com](mailto:adudczak@gmail.com) )

Poznań Java User Group

# Plan prezentacji

- 1 Wprowadzenie
- 2 Najciekawsze konstrukcje języka
  - Klasy i skrypty
  - Domknięcia
  - Pozostałe
- 3 Making things more Groovy
  - Groovy Builders
  - Operacje I/O
  - GSQL
  - Ant, GAnt i Maven
  - Czy coś jeszcze?
- 4 Grails
  - Quest for the Holy Grails
  - Pierwsza aplikacja
- 5 Zakończenie

*"Groovy is like a super version of Java. It can leverage Java's enterprise capabilities but also has cool productivity features like closures, builders and dynamic typing. If you are a developer, tester or script guru, you have to love Groovy."*

Czym jest Groovy?

- JSR 241: The Groovy Programming Language
- Łatwość programowania, najlepsze pomysły z wielu języków
- „The code look like what it does”
- Łatwy do opanowania dla programistów Javy
- Dynamicznie typowany

Co to znaczy dynamicznie typowany?

- Jeżeli nie zadeklarujesz typu, tak naprawdę nigdy nie wiesz jaki typ przypisało mu Groovy
- Jeżeli dostarczymy informacje o typie, Groovy dopilnuje, żeby wszystko było jak w Javie.

# Plan prezentacji

- 1 Wprowadzenie
- 2 Najciekawsze konstrukcje języka
  - Klasy i skrypty
  - Domknięcia
  - Pozostałe
- 3 Making things more Groovy
  - Groovy Builders
  - Operacje I/O
  - GSQL
  - Ant, GAnt i Maven
  - Czy coś jeszcze?
- 4 Grails
  - Quest for the Holy Grails
  - Pierwsza aplikacja
- 5 Zakończenie

# Piszemy w Groovy?

- Dwa podstawowe sposoby pisania kodu:
  - klasy - zupełnie jak w Javie
  - skrypty - trochę inaczej niż w Javie
- Pakiety, importy, metody - tak samo jak w Javie
- Groovy domyślnie importuje następujące pakiety: `java.lang`, `java.io`, `java.math`, `java.net`, `java.util`, `groovy.lang`, `groovy.util`
- Najważniejsza różnica : Groovy domyślnie zakłada, że wszystko jest `public`.
- Niektóre rzeczy są opcjonalne np. średniki na końcu wyrażenia, nawiasy w wywołaniach metod czy instrukcja `return`.

- Na poziomie JVM klasy Groovy wyglądają tak samo jak klasy Javowe.
- Model dziedziczenia w Groovy jest taki sam jak w Javie.
- Co z tego wynika?
  - Możemy dziedziczyć, implementować, odwoływać się do klas Groovy w Javie.
  - Działa to w obie strony, tak długo jak długo nie występują zależności cykliczne

- Nie musimy tworzyć klas aby pisać w Groovy.

```
println "Witajcie na PJUG'u!"  
>> Witajcie na PJUG'u!
```

- Wszystkie zmienne w skryptach są deklarowane w zakresie lokalnym.
- W skryptach (tylko!) istnieją wiązania (*ang. bindings*)

```
String localVar = "I am a local variable"  
bindingVar = "I am a binding variable"
```

# Zasięg deklarowanych zmiennych – Klasy

## Zasięg zmiennych w klasie

```
class GroovyClass {
    def static prefix = "Mr. "

    def static printer(def name, def surname) {
        println prefix+ " "+ name + " " + surname;
    }

    static void main(args) {
        printer("Name", "Surname")
    }
}
```

## Wynik działania

Mr. Name Surname

## Użycie wiązania

```
prefix = "Mr. "
```

```
def printer(def name, def surname) {  
  println prefix+ " "+ name + " " + surname;  
}
```

```
printer("Name", "Surname")
```

## Wynik działania

```
Mr. Name Surname
```

## Zasięg deklarowanych zmiennych – Skrypty

Wszystkie zmienne deklarowane w skrypcie mają zasięg lokalny!

```
def prefix = "Mr. "  
  
def printer(def name, def surname) {  
    println prefix+ " "+ name + " " + surname;  
}  
  
printer("Name", "Surname")
```

Wynik działania

Exception in thread "main" groovy.lang.MissingPropertyException: No such property: prefix

# Deklarowanie zmiennych

- Deklarując zmienne musimy podać nazwę typu lub użyć słowa kluczowego *def*.
- Użycie *def* oznacza, że: „Nie interesuje mnie jakiego typu jest to zmienna”.
- Aby ułatwić zrozumienie, potraktuj *def* jako Javowy *Object* (!).

## Przykład

```
def dynamic = 1
dynamic = "String w zmiennej typowanej dynamicznie"
int typed = 2
typed = "Czy da się wrzucić String do zmiennej typu int?"
// throws ClassCastException
```

# Deklarowanie metod

## Deklarowanie metod

```
def concat(def name, def surname) {  
    name + " " + surname; //return i ; są opcjonalne!!  
}  
  
int adder(int first, int second){  
    first + second  
}  
  
void concatVoid(def name, def surname) {  
    name + " " + surname;  
}  
  
println adder(1,3) + " "+ concat("Adam","Dudczak")  
println concatVoid("Adam","Dudczak")
```

## Wynik działania metody

```
4 Adam Dudczak  
null
```

*„A closure in Groovy is an anonymous chunk of code that may take arguments, return a value, and reference and use variables declared in its surrounding scope.”*

- Pod wieloma względami przypomina klasy anonimowe
- Czym się to różni od zwykłego nie nazwanego bloku kodu?
  - Gdy JVM napotka zwykły blok kodu wykonuje go od razu.
  - Domknięcie jest wywoływane dopiero gdy nastąpi do niego odwołanie.
- Składnia wygląda tak:

```
{ [closureArguments->] statements }
```

- Zmienna specjalna *it* jest automatycznie deklarowane wewnątrz domknięcia (*ang. closure*).
- Zostaje jej przypisana wartość pierwszego parametru albo null - jeżeli nie ma parametrów.

```
def c = { it }  
assert c() == null  
assert c(1) == 1
```

- Zagnieżdzone domknięcia i wartość *it*

```
def outer = {  
  def inner = { it+1 }  
  inner(it+1)  
}  
assert outer(1) == 3
```

## Domknięcia – przykłady

### Przypisanie domknięcia do zmiennej

```
def c = {  
    println it;  
}  
//wywołania  
c("test")  
c.call("test2")
```

### Obsługa zdarzeń

```
Button b = new Button ("Push Me");  
b.onClick { buttonClicked() }
```

## Domknięcia – przykłady

### Przekazywanie domknięć do metod

```
(1..3).each { i ->  
  println "Hello ${i}"  
}
```

### Wynik

```
Hello 1  
Hello 2  
Hello 3
```

## Domknięcia – przykłady

### Przekazywanie domknięć do metod

```
def map = [ad:"be", ce:"de"]  
map.each {  
  println it.key +" value "+it.value  
}
```

### Wynik

```
ce value de  
ad value be
```

## Domknięcia – przykłady

Domknięcie mogą być kluczami w mapie

```
f = { println "f called" }  
m = [ (f): 123 ]
```

```
println m.get(f)    // 123  
println m[f]       // 123
```

Domknięcie jako wartości w mapie

```
m = [ f: { println "f called" } ]  
(m.f)()    // f called
```

- Instrukcje warunkowe (if/switch) są kompatybilne z składnią Javy.
- Największa różnica jest w składni switch'a:

```
switch ( x ) {  
  case "foo":  
    result = "found foo"  
    break  
  case [4, 5, 6, 'inList']:  
    result = "list"  
    break  
  case Integer:  
    result = "integer"  
    break  
}
```

Pętla for wygląda trochę inaczej...

```
def x = 0
for ( i in 0..9 ) {
  x += i
}
//iterowanie po znakach w Stringu
def text = "abc"
def list = []
for (c in text) {
  list.add(c)
}
```

- Składnia Groovy oferuje wiele rozszerzeń w stosunku do kolekcji znanych z Javy.

## Kolekcje w Groovy

```
def list = [5, 6, 7, 8]
assert list.get(2) == 7
assert list[2] == 7
assert list instanceof java.util.List

def range = 5..8
assert range.size() == 4

def map = [name:"Gromit", likes:"cheese", id:1234]
def emptyMap = [:]
```

- Operacje na każdym elemencie kolekcji:  
`assert [1, 3, 5] == ['a', 'few', 'words']*size()`
- Przeciążanie operatorów :)
  - Wystarczy, że dodasz do swojego obiektu odpowiednią metodę np. *plus(twój typ a)*.
- Dodawanie metod do obiektów w runtime.



# Plan prezentacji

- 1 Wprowadzenie
- 2 Najciekawsze konstrukcje języka
  - Klasy i skrypty
  - Domknięcia
  - Pozostałe
- 3 Making things more Groovy
  - Groovy Builders
  - Operacje I/O
  - GSQL
  - Ant, GAnt i Maven
  - Czy coś jeszcze?
- 4 Grails
  - Quest for the Holy Grails
  - Pierwsza aplikacja
- 5 Zakończenie

- Struktury drzewiaste w programowaniu spotykamy na każdym kroku.
- Groovy wspiera:
  - Tworzenie wszelkiego XML'a: skryptów Ant'a, Jelly, XHTML, XSD, konfiguracja Spring.
  - Budowanie GUI: Swing, SWT, sceny Java3d (!).
- Można oczywiście tworzyć własne implementacje, dziedzicząc z *groovy.util.BuilderSupport* - implementujemy w Javie.

## Tworzenie XML'a

```
import groovy.xml.*;

writer = new StringWriter()
builder = new MarkupBuilder(writer)
builder.document(name:"a document") {
    chapter(name:"First one") {
        page1("first page")
        page2("second page")
    }
    chapter(name:"First two") {
        page1("page 1")
        page2("page 2" )
    }
}
println(writer.toString())
```

## Wynik

```
<document name="a document">  
  <chapter name="First one">  
    <page1>first page</page1>  
    <page2>second page</page2>  
  </chapter>  
  <chapter name="First two">  
    <page1>page 1</page1>  
    <page2>page 2</page2>  
  </chapter>  
</document>
```

## Wynik

```
import javax.swing.*;
import java.awt.*;

swing = new groovy.swing.SwingBuilder()
frame = swing.frame(defaultCloseOperation : JFrame.EXIT_ON_CLOSE,
                    title : "pJUG Frame", size : new Dimension(400, 100))
{
    label = label(text:"groovy!", constraints: BorderLayout.NORTH)
    panel {
        button(text: "More groove!",
              actionPerformed : { label.setText("GROOVY!!!")})
    }
}
label.font = label.font.deriveFont(label.font.size2D + 10 as Float)
frame.visible = true
```

## Wynik

```
import javax.swing.*;
import java.awt.*;

swing = new groovy.swing.SwingBuilder()
frame = swing.frame(defaultCloseOperation: JFrame.EXIT_ON_CLOSE,
                    title: "pJUG Frame")
{
    label = label(text: "groovy!")
    panel {
        button(text: "More groove!") {
            actionPerformed {
            }
        }
    }
}
label.font = label.font.deriveFont(label.font.size2D + 10 as Float)
frame.visible = true
```



## Wynik

```
import javax.swing.*;
import java.awt.*;

swing = new groovy.swing.SwingBuilder()
frame = swing.frame(defaultCloseOperation: JFrame.EXIT_ON_CLOSE,
                    title: "pJUG Frame")
{
    label = label(text: "groovy!")
    panel {
        button(text: "More groove!") {
            actionPerformed {
            }
        }
    }
}
label.font = label.font.deriveFont(label.font.size2D + 10 as Float)
frame.visible = true
```



# Operacje na plikach

Czytanie pliku w jednej linii kodu?

```
new File("foo.txt").eachLine { line -> println(line) }
```

Można też tak...

```
new File("foo.txt").withReader { reader ->
  while (true) {
    def line = reader.readLine()
  }
}
```

## Przykład sieciowy

```
import groovy.util.*;

def rssFeed = "http://www.jug.werla.pl/?feed=rss2";
def xmlFeed = new XmlParser().parse(rssFeed);

(0..< xmlFeed.channel.item.size()).each {
    def item = xmlFeed.channel.item.get(it);
    def title = item.title.value[0];
    def link = item.link.value[0];
    println title + " : " + link;
}
```

## Przykład sieciowy – wyniki

\Zróbmy to szybko" - Groovy i Grails : <http://www.jug.werla.pl/?p=16>

Logo PJUG : <http://www.jug.werla.pl/?p=10>

Nowy serwis JUG Poznań : <http://www.jug.werla.pl/?p=5>

Java 6 : <http://www.jug.werla.pl/?p=7>

Szybki przykład na to co zostało z JDBC

```
import groovy.sql.Sql;
import groovy.xml.MarkupBuilder

def xml = new MarkupBuilder()
def sql =
    Sql.newInstance(
        "jdbc:hsqldb:file:e://temp//groovy//eval",
        "sa","", "org.hsqldb.jdbcDriver")

users = xml.users {
    sql.eachRow(
        "select USER_LOGIN, USER_TYPE, USER_ID from USERS") {
        row -> user( id:row['USER_ID'], type:row['USER_TYPE'])
            {
                email(row['USER_LOGIN'])
            }
        }
    }
```

## Wyniki

```
<users>
  <user type="10" id="1">
    <email>adudczak@gmail.com</email>
  </user>
  <user type="1" id="2">
    <email>test@test.pl</email>
  </user>
</users>
```

## Dataset - przykład

```
def users = sql.dataSet('USERS')
users.each { println it.USER_LOGIN }
users.add(USER_LOGIN: 'jug@poznan.pl',
          USER_TYPE: 1, USER_PASS:'test');
users.each { println it.USER_LOGIN }
```

- Można jeszcze : findAll, createView ...
- <http://groovy.codehaus.org/apidocs/groovy/sql/DataSet.html>

- Jednym z głównych celów do jakich na początku było używane/wymyślane Groovy było usprawnienie budowania projektów.
- Groovy oferuje sporo rozwiązań umożliwiających integrację z Ant'em.
- Groovy Ant Task
- Groovyc Ant Task - zadanie wywołuje kompilator groovyc.
- AntBuilder

- Wymagania : groovy-all-xxx.jar w classpath
- Deklaracja zadania w naszym projekcie:

```
<taskdef name="groovy"  
        classname="org.codehaus.groovy.ant.Groovy"  
        classpathref="my.classpath"/>
```

- Do czego mamy dostęp:
  - instancji AntBuilder'a (obiekt: ant) skonfigurowanego dla naszego projektu
  - obiekty: project, properties, target, task
- Możemy tworzyć skrypt pomiędzy znacznikami groovy korzystając z dostępnych obiektów.
- Wskazać zewnętrzny plik z kodem Groovy.

## Groovy Ant Task - przykłady

```
<groovy>
```

```
    println("Hello World")
```

```
</groovy>
```

```
<groovy>
```

```
    ant.echo("Hello World")
```

```
</groovy>
```

```
<groovy src="/some/directory/some/file.groovy">
```

```
    <classpath>
```

```
        <pathelement location="/my/groovy/classes/directory"/>
```

```
    </classpath>
```

```
</groovy>
```

- Ant Builder pozwala zastąpić xml kodem Groovy.
- Właśnie na tym bazuje GAnt - kod groovy zamiast xml'a.
- *build.gant* zamiast *build.xml*.

## AntBuilder - przykład

```
ant.echo("Hello World!")  
ant.mkdir(dir:"/home/ad/jug")
```

# Ant Builder i Gant

## AntBuilder - przykład

```
includeTargets << gant.targets.Clean //predefined task  
cleanPattern << [ '**/*~' , '**/*.bak' ]  
cleanDirectory << 'build'
```

```
//domyślne zadanie rozponawane jest po nazwie  
task ( 'default' : 'The default target.' ) {  
    println ( 'Default' )  
    depends ( clean )  
    Ant.echo ( message : 'A default message from Ant.' )  
    otherStuff ( )  
}
```

```
task ( otherStuff : 'Other stuff' ) {  
    println ( 'OtherStuff' )  
    Ant.echo ( message : 'Another message from Ant.' )  
    clean ( )  
}
```

*„But then Gant is not about replacing Ant, it is about having a different way of working with the tasks and infrastructure that Ant provides.”*

Więcej informacji można znaleźć :  
<http://groovy.codehaus.org/Gant>

## Pozwala wykonywać i kompilować kod Groovy w Maven2

```
<plugin>
<groupId>org.codehaus.mojo</groupId>
<artifactId>groovy-maven-plugin</artifactId>
<executions>
  <execution>
    <phase>generate-resources</phase>
    <goals>
      <goal>execute</goal>
    </goals>
    <configuration>
      <source> <!-- groovy+ant+maven :) -->
      <body>
        if (project.packaging != "pom") {
          log.info("Copying some stuff...")

          def ant = new AntBuilder()
          def dir = "${project.basedir}/target/classes/META-INF"

          ant.mkdir(dir: dir)
          ant.copy(todir: dir) {
            fileset(dir: "${project.basedir}") {
              include(name: "LICENSE.txt")
              include(name: "NOTICE.txt")
            }
          }
        }
      </body>
    </configuration>
  </execution>
</executions>
</plugin>
```

...

# Czy coś jeszcze?

- Groovlets
- COM Scripting
- Google Data Support
- Gram - narzędzie al'a xdoclet.
- Groovy Jabber-RPC
- Groovy Monkey - skryptowanie w Eclipse IDE, oparte na Eclipse Jobs API.
- Groovy SOAP
- GSP - o tym jeszcze będzie.
- Native Launcher - tworzenie natywnych plików wykonywalnych (nasz.exe).
- XMLRPC

# Plan prezentacji

- 1 Wprowadzenie
- 2 Najciekawsze konstrukcje języka
  - Klasy i skrypty
  - Domknięcia
  - Pozostałe
- 3 Making things more Groovy
  - Groovy Builders
  - Operacje I/O
  - GSQL
  - Ant, GAnt i Maven
  - Czy coś jeszcze?
- 4 Grails
  - Quest for the Holy Grails
  - Pierwsza aplikacja
- 5 Zakończenie

# Quest for the Holy Grails

- DRY (Don't Repeat Yourself)
- Convention-over-configuration, configuration by exception. . .
- Łatwość dokonywania zmian i ich natychmiastowy efekt
- Wykorzystanie tego wszystkiego co zostało już napisane (w Javie).

- Grails zostało oparte na sprawdzonych rozwiązaniach Javowych:
  - Hibernate
  - Spring
  - SiteMesh
  - GROOVY! – GORM, GSP
  - Ant, Jetty
- Do tego : Prototype, Dojo, Yahoo UI jako wsparcie dla tworzenie aplikacji AJAX'owych.
- Funkcjonalność Grails budują wtyczki - używamy tylko tego co potrzebne.

# Grails prosto z pudełka

- Bezproblemowa instalacja - <http://grails.org/Installation>.
- Możemy praktycznie od razu przystąpić do pracy!
- Grails to wszystko czego nam trzeba - nie ma potrzeby instalować dodatkowego oprogramowania np. Tomcat'a czy bazy danych....
- Grails to narzędzie którym operujemy z konsoli, to tak naprawdę zestaw całkiem sprytnych skryptów Ant/GAnt.

# Pierwsza aplikacja

## Pierwsze kroki...

```
projects> grails create-app
...
create-app:
  [input] Enter application name:
  jug-app
....
BUILD SUCCESSFUL

projects> cd jug-app
jug-app> grail run-app
...
```

## Co się dzieje?

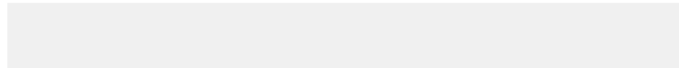
- Grails startuje Jetty,
- przeprowadza deployment naszej aplikacji,
- inicjalizuje i uruchamia...

`http://localhost:8080/jug-app :`



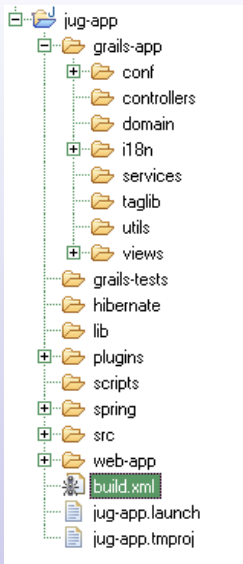
## Welcome to Grails

Congratulations, you have successfully started your first Grails application! At the moment this is the default page, feel free to modify it to either redirect to a controller or display whatever content you may choose. Below is a list of controllers that are currently deployed in this application, click on each to execute its default action:



# Grails – Struktura projektu

Struktura wygenerowanego projektu:



### Tworzenie klasy domenowej

```
jug-app>grails create-domain-class
...
name not specified. Please enter:
Meeting
...
```

### Co zostało wygenerowane?

Stworzone zostaną dwie klasy :

- jug-app/grails-app/domain/Meeting.groovy,
- jug-app/grails-tests/MeetingTests.groovy.

Po utworzeniu są puste.

## Tworzenie klasy domenowej

```
class Meeting {  
    String topic  
    String place  
    static hasMany = [attendies:Attendee]  
}
```

## Static...?

- Za pomocą takich właśnie statycznych pól modeluje się relacje między klasami domenowymi.
- W tym wypadku oznajmiamy, że Meeting może mieć wielu Attendee, a informacja o nich będzie składowana w zmiennej attendies.

## Tworzenie klasy domenowej – Attendee

```
class Attendee {  
    Meeting meeting  
    String name  
    String surname  
    String address  
  
    static belongsTo = Meeting  
    static optionals = ["address"]  
}
```

## Własności opcjonalne

- Domyślnie wszystkie pola są wymagane.
- Pole statyczne *optionals* zawiera informację o polach opcjonalnych.
- Natomiast pole *belongsTo* indetyfikuje stronę posiadającą w relacji 1 do n.

- Skąd Grails wie co ma być wrzucane do bazy?
- GORM zakłada, że do bazy danych trafia wszystko co znajduje się w katalogu z klasami domenowymi!!
- Domyślną bazą danych jest HSQLDB który jest wbudowany w Grails.
- Można to zmienić w plikach:
  - `grails-app/conf/DevelopmentDataSource.groovy`
  - `grails-app/conf/TestDataSource.groovy`
  - `grails-app/conf/ProductionDataSource.groovy`

## Tworzenie kontrolerów - Meeting

```
jug-app>grails create-controller
...
create-controller:
  [input] Enter controller name:
  Meeting
...
  [echo] Created controller :
         grails-app/controllers/MeetingController.groovy
...
```

Analogicznie postępujemy dla klasy Attendee.

## Scaffold

```
class AttendeeController {  
    def scaffold = Attendee  
}
```

- Scaffold - czyli „rusztowanie” albo „szafot” :)
- Grails powinien dynamicznie wygenerować kontroler (wraz z widokami) pozwalający na realizację podstawowych operacji CRUD ( *ang. create-read-update-delete*)

## Meetings - walidacja

```
class Meeting {  
  //...  
  static constraints = {  
    topic(maxLength:70,blank:false)  
    place(inList:['pub','zoo'], blank:false)  
  }  
}
```

### Meetings - Kolejność kolumn w formularzach

```
class Meeting {  
    //...  
    String toString() {" ${this.topic} : ${this.place} }  
}
```

- Kolejny ciekawy mechnizm w Groovy - GStrings.
- Zmienne z dolarami zostaną podmienione przez odpowiednie wartości.

# Plan prezentacji

- 1 Wprowadzenie
- 2 Najciekawsze konstrukcje języka
  - Klasy i skrypty
  - Domknięcia
  - Pozostałe
- 3 Making things more Groovy
  - Groovy Builders
  - Operacje I/O
  - GSQL
  - Ant, GAnt i Maven
  - Czy coś jeszcze?
- 4 Grails
  - Quest for the Holy Grails
  - Pierwsza aplikacja
- 5 Zakończenie

- Mimo iż Groovy jest projektem dość młodym (Groovy 1.0 w styczniu 2007) to istnieje już spora liczba projektów związanych z tym językiem.
- Grails nie osiągnął jeszcze wersji 0.5, a już sprawdza się w zastosowaniach komercyjnych/produkcyjnych.
- W sieci można znaleźć całkiem sporo przykładów.

- Dokumentacja na stronach projektu jest również bardzo bogata – choć zawiera sporo błędów/zaszłości.
- Eclipse, Netbeans jak i IntelliJ Idea oferują wtyczki do pracy z Groovy
- Ant i Maven są również gotowe do współpracy z G.

- Groovy Project Site :  
<http://groovy.codehaus.org>
- Grails Project Site :  
<http://grails.codehaus.org>
- Groovy Closures :  
<http://groovy.codehaus.org/Closures>
- Practically Groovy: JDBC programming with Groovy by Andrew Glover  
<http://www-128.ibm.com/developerworks/java/library/j-pg01115.html>
- Groovy Ant Scripting :  
<http://docs.codehaus.org/display/GROOVY/Ant+Scripting>
- Groovy Javadocs :  
<http://groovy.codehaus.org/apidocs/index.html>

- Getting Started with Grails by Jason Rudolph  
<http://www.infoq.com/minibooks/grails>
- More Java3D and Groovy Builders  
<http://langexplr.blogspot.com/2007/03/more-java3d-and-groovy-builders.html>
- Persistence Made Easy with Groovy and JPA by Romain Guy  
<http://www.curious-creature.org/2007/03/25/persistence-made-easy-with-groovy-and-jpa/>

- Interview with Dr. Paul King and Jon Skeet about Groovy  
[http://javaposse.com/index.php?post\\_id=175667](http://javaposse.com/index.php?post_id=175667)
- Grails Podcast  
<http://grails.codehaus.org/Grails+Podcast>
- Grails Screencasts  
<http://grails.codehaus.org/Grails+Screencasts>

Pytania?