내용

[What is RiveScript? 2](#_Toc510528401)

[[Tutorial] FIRST STEPS - Hello, Human! 4](#_Toc510528402)

[[Tutorial] FIRST STEPS - Random Replies 7](#_Toc510528403)

[[Tutorial] The Begin File 12](#_Toc510528404)

[[Tutorial] TRIGGERS 16](#_Toc510528405)

[[Tutorial] Arrays in Triggers 20](#_Toc510528406)

[[Tutorial] Priority Triggers 22](#_Toc510528407)

[[Tutorial] Short Discussions 25](#_Toc510528408)

[[Tutorial] Conditionals 26](#_Toc510528409)

[[Tutorial] Topics 30](#_Toc510528410)

[[Tutorial] The Begin Block 32](#_Toc510528411)

[[Tutorial] MORE DEFINITIONS 34](#_Toc510528412)

[[Tutorial] TAGS 37](#_Toc510528413)

[[Tutorial] CONCLUSION 41](#_Toc510528414)

[What is RiveScript?](http://kjw1801.tistory.com/entry/What-is-RiveScript)

**RiveScript란?**

챗봇을 쉽고 빠르게 개발할수 있는 스크립트 기반의 언어라고 정의할 수 있습니다.

**RiveScript의 특징**

**Simple**

RiveScript는 배우기 쉽고 간단 일반 텍스트 스크립팅 언어를 제공 제공합니다.

복잡한 불투명 한 XML 구조 나 임의의 기호 및 일부 다른 챠트 (chatbot) 언어와 같은 라인 노이즈를 망칠 필요가 없습니다!

**Powerful**

RiveScript는 인상적인 챗봇 특성을 구축하기 위해 몇 가지 간단한 규칙을 가지고 있습니다.

한 번에 복잡한 단어 패턴 세트를 간단한 정규 표현 형식으로 작성하십시오.

**Flexible**

RiveScript는 개발과 관련하여 "유닉스와 같은" 접근 방식을 취합니다.

핵심 라이브러리는 작고 독립적이며 지능적인 응답을 제공합니다.

**Open Source**

위와 같은 특징으로 RiveScript의 특징을 설명하고 있지만 RiveScript의 가장 큰 장점은 다양한 언어의 라이브러리를 제공 한다는 점입니다.

**First Party Libraries**

Go

Java

JavaScript

Perl

Python

**Third Party Libraries**

C#

PHP

**진행 방향**

저의 목적은 RiveScript를 배워서 Chatbot을 Android 앱에서 사용하는 방향으로 진행 하고자 합니다.  감사합니다.

RiveScript : <https://www.rivescript.com/>

[Tutorial] FIRST STEPS - Hello, Human!

환경설정은 각각의 환경에 맞춰서 해줘야 되지만 목적은 기본적은 Tutorial을 실행하면서 사용방법을 익히는게 목적이므로 아래 링크에서 온라인으로 테스트가 가능합니다.

온라인 테스트 : <https://play.rivescript.com/>

해당 링크로 접속하면 기본적인 스크립트가 포함된 창이 생성되지만 아래와 같이 나머지 스크립트는 제거해 줍니다.

! version = 2.0

+ hello bot

- Hello, human!

상단에  Run버튼을 클릭해서 왼쪽 하단에 hello bot이라고 입력하면 Hello, human!이라는 응답을 받을 수 있습니다.



**The Code, Explained**

RiveScript는 한줄씩 입력하는 라인 기반의 스크립트 언어고 각 라인의 시작은 각각의 명령 심볼(!, + , -)을 통해서 시작합니다.

! version = 2.0를 해석하자면 버전 2.0의 RiveScript 사양을 따른다라는 의미 입니다. 이렇게 함으로써 어떤 버전을 기준으로 스크립트를 작성했는지 알 수 있고 향후 새로운 버전이나 하위 버전에 대응 할 수 있는 장점이 있습니다.

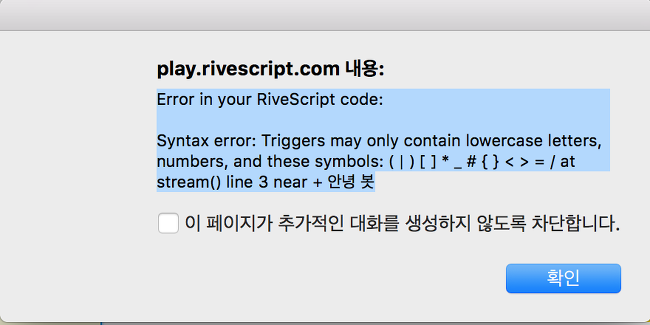
+커멘드는 트리거[1]를 정의하는 방법입니다. 여기서 트리거란 정해진 조건에 만족할 때 자동으로 수행하는 동작을 말하며 사용자의 메시지를 일치시키는데 사용하는 텍스트 행입니다. "hello bot" 이라는 트리거를 정의했고 이에 만족하는 텍스트가 입력 되었을 경우에 동작하도록 합니다.

\* RiveScript는 대소문자 구분없이 문자가 일치했는지 만을 판단합니다. "Hello bot", "hello Bot" 모두 동일한 트리거로 판단.

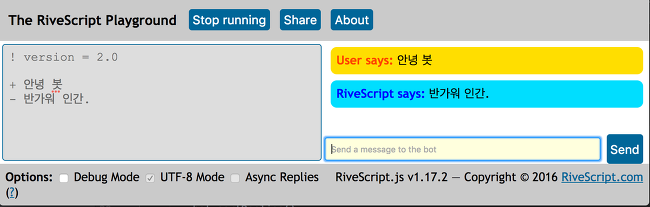
-커멘드는 트리거에 대한 응당을 정의하는 방법입니다. 여기에서 "hell bot"이라는 트리거와 일치하면 "Hello human"이라고 말함으로써 사용자에게 응답하게 됩니다.

한글로 변환

우리의 목적은 한국어 버전의 챗봇으로 영어로 정의한 트리거와 응답을 한국어로 바꿔보겠습니다. + hello bot, - Hello human을 + 안녕 봇, - 반가워 인간 으로 바꿔보겠습니다. 다시 Run 버튼을 눌러주면 아래와 같은 오류가 발생합니다.



한글은 UTF-8 모드로 설정해줘야 정상적으로 동작하고 아래의 UTF-8 Mode 버튼을 클릭해서 다시 Run을 하면 정상적으로 동작합니다. 테스를 하게 되면 정상적으로 출력되는 것을 확인 할 수 있습니다.



[1] 트리거(trigger) : http://terms.naver.com/entry.nhn?docId=860900&cid=42346&categoryId=42346

[Tutorial] FIRST STEPS - Random Replies

우리가 만든 봇이 항상 똑같은 방식으로 반응 한다면 빨리 지루해질 수 있습니다.

이러한 이유로 RiveScript를 사용하면 임의의 응담을 트리거에 쉽게 추가 할 수 있습니다!

Random Replies을 사용하는 방법은 아래와 같이 사용할 수 있습니다.

+ how are you

- I'm great, how are you?

- I'm good, you?

- Good :) you?

- Great! You?

- I'm fine, thanks for asking!

테스트를 하게되면 우리의 봇은 "how are you"라는 트리거에 아래의 5가지 메시지를 랜덤으로 응답하게 됩니다.

또 다른 방법으로는 {random}태그를 사용하여 임의의 응답을 랜덤으로 받는 방법도 있습니다.

+ say something random

- This {random}message|sentence{/random} has a random word.

이렇게 사용입력한 스크립트를 테스트하게 되면 다음과 같은 결과를 얻을 수 있습니다.

{random}과{/random}사이에 |로 문자열을 분리하면 분리된 문자열 내에서 랜덤으로 하나씩 선택해서 응답하게 됩니다.



**A Note About Style**

RiveScript 문서를 멋지고 깔끔하고 읽기 쉽도록 유지하려면 다음과 같은 지침의 가이드 라인을 제공합니다.

**빈 줄을 사용하여 논리 그룹 코드를 분리**하십시오.

예를 들어, 트리거와 그 응답은 함께 그룹화되어야 하며 공백 행은 다른 트리거 및 응답 사이를 구분합니다.

그렇게 하면 아래와 같이 구분되어질 것입니다.

! version = 2.0

+ hello bot

- Hello, human!

+ how are you

- I'm great, how are you?

- I'm good, you?

- Good :) you?

- Great! You?

- I'm fine, thanks for asking!

**Let's Talk About Weight**

랜덤 응답은 확실히 유용하지만, 일부 응답은 다른 응답보다 빈도수를 조정하고 싶을 수도 있습니다.

응답 커멘드에서 {weight}태그를 사용하여 다른 응답에 비해 빈도수를 재정의 할 수 있습니다.

+ greetings

- Hi there!{weight=20}

- Hello!{weight=25}

- Yos kyoco duckeb!

여기에서 각 응답에 가중치(weight)를 할당하고 하나의 횡성수설 한 응답 하나를 남겼습니다. "Hi there!" 은 46번의 응답 중에 20번 선택 됩니다. "Hello!"는 46번의 응답 중 25번 선택 될 것이고, "Yos kyoco duckeb!"는 46번 중 1번만 선택됩니다. 가중치 태그를 포함하지 않는 응답의 가중치는 1입니다. 이렇게 응답의 확률은 가중치를 합으로 나눈 값입니다. (20 + 25 + 1 = 46, 그래서 각각의 응답은 46의 확률 중에서 선택)

\* 가중치는 0이 될 수 없으며 음수가 될 수 없습니다.

\* {random}태그 안에는 가중치를 사용할 수 없습니다.

[Tutorial] FIRST STEPS - Line Breaking  
  
다음은 긴 줄의 RiveScript 코드를 작성할때 여러 줄로 나누어 사용해야 할 때가 있습니다.

이 경우 ^커멘드(**Continuation**)를 사용하면 됩니다. ^커멘드는 자동으로 이전 행의 데이터를 확장합니다.

+ tell me a poem

- Little Miss Muffit sat on her tuffet,\n

^ In a nonchalant sort of way.\n

^ With her forcefield around her,\n

^ The Spider, the bounder,\n

^ Is not in the picture today.

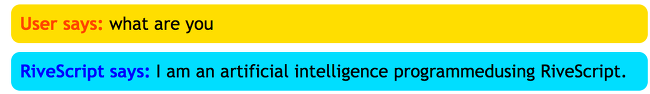
Continuation 커멘드는 이전 행과 연송 행 사이에 공백을 자동으로 삽입하지 않습니다. 다음 예제를 참고하세요.

// There will be no space between "programmed" and "using"!

+ what are you

- I am an artificial intelligence programmed

^ using RiveScript.



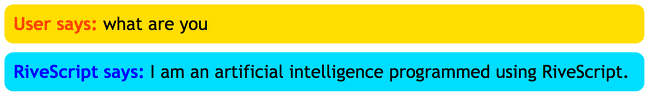
 공백을 표시할 위치에서 \s를 사용하면 정상적으로 공백 처리가 됩니다.

// This one will have a space.

+ what are you

- I am an artificial intelligence programmed\s

^ using RiveScript.



"tell me a poem" 예제에서  \n를 사용하면 공백 대신 줄 바꿈 처리가 됩니다. 그렇지만 연속적인 공백이나 줄바꿈을 매번 입력해줘야 된다면 매우 불편할 것입니다. RiveScript에서는 ! local concat를 통해서 연속적인 처리를 자동으로 수행 할 수 있습니다.

// Tell the parser to join continuation lines with line breaks

! local concat = newline

// Now we don't need to explicitly write the \n characters every time!

+ tell me a poem

- Little Miss Muffit sat on her tuffet,

^ In a nonchalant sort of way.

^ With her forcefield around her,

^ The Spider, the bounder,

^ Is not in the picture today.

// Now change the concat mode to spaces

! local concat = space

// Here we don't have to use \s like in the earlier example.

+ what are you

- I am an artificial intelligence programmed

^ using RiveScript.

// Go back to the default concatenation mode (which doesn't insert ANY

// character when joining lines)

! local concat = none

예제에서 볼 수 있듯이 ! local concat명령을 통해서 다음 그룹에서 처리해야할 명령을 자동으로 설정 할 수 있습니다.

https://play.rivescript.com/ 위의 예제를 테스트를 하면 정상적으로 동작되지 않지만

https://www.rivescript.com/try 에서 테스트를 하면 정상적으로 출력이 됩니다.

The supported options for this are:

* none -- the default, nothing is added when continuation lines are joined together.
* space -- continuation lines are joined by a space character (\s)
* newline -- continuation lines are joined by a line break character (\n)

[[Tutorial] The Begin File](http://kjw1801.tistory.com/entry/Tutorial-The-Begin-File)

이번에는 RiveScript의 brain이 무엇인지 알아보겠습니다.

RiveScript brain이라고 표현하는 이것은 RiveScript 문서 집합을 의미 합니다.

 (a "brain" is a set of RiveScript documents)

RiveScript brain은 begin.rive라는 문서가 포함되어 있어야 합니다.

가장 유용한 기능중에 하나가 이번에 확인 하게 될 내용은 **substitutions** 기능입니다.

우리는 이 substitutions을 이용하여 변환할 단어를 미리 지정할 수 있습니다.

예를 들어 what 's라는 단어가 whats로 변환합니다. 이를 통해 사용자의 메시지에

회신하는 방법을 효과적으로 제어 할 수 있습니다.

가장 먼저 begin.rive 파일부터 아래에서 확인 할 수 있습니다.

! version = 2.0

// Bot variables

! var name = Tutorial

! var age = 5

// Substitutions

! sub i'm = i am

! sub i'd = i would

! sub i've = i have

! sub i'll = i will

! sub don't = do not

! sub isn't = is not

! sub you'd = you would

! sub you're = you are

! sub you've = you have

! sub you'll = you will

! sub what's = what is

! sub whats = what is

! sub what're = what are

! sub what've = what have

! sub what'll = what will

! sub who's = who is

**Definitions**

또한 // 문자가 무엇인지 궁금 할 수 있습니다. 대부분의 프로그래밍 언어 에서처럼 RiveScript를 사용하면 소스 코드에 주석을 표현합니다. // 문자는 주석의 시작을 나타냅니다. 인터프리터는 코드를 읽을 때 이러한 주석을 무시합니다.

이제 이러한 새로운 정의 유형을 살펴 보겠습니다.

먼저 두 개의 Bot 변수를 정의했습니다. ! var name = Tutorial은 이름을 정의하고 ! var age = 5는 나이를 정의합니다.  이후 봇 변수를 사용하여 봇이 사용자에게 자신에 대해 조금이라도 알려줄 수있는 답글을 작성할 수 있습니다.

예를 들어, 제 이름은 {name} 입니다. 나이는 {age} 에요.

그다음은 **substitutions**입니다.  ! sub i'm는 i am로 정의해서 i'm 내용과 i am을 같은 대치어로 정의 할 수 있습니다.

다음 예제를 작성해보겠습니다.

+ what is up

- Not much, you?

- nm, you?

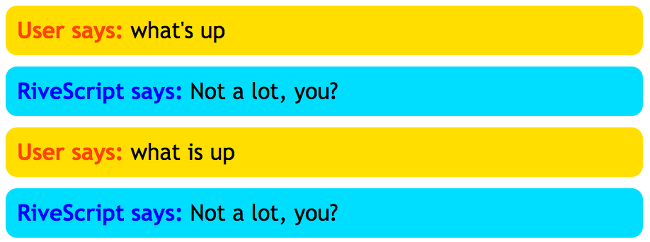
- Not a lot, you?

+ you are a bot

- How did you know I'm a machine?

실제로 실행해서 테스트를 하게 되면 **substitutions**을 정의하기 전에는 "what's up"이라고 입력하면 응답하지 못했지만 begin.rive에 대치어를 정한다음 실행하게 되면 "what's up"과  "what is up"에 대해서 동일한 응답을 하는 것을 확인 할 수 있습니다.

한글로 사용한다면??



한글로 사용한다면??

! version = 2.0

// Substitutions

! sub 나는 = 난

+ 난 멋져

- 나도 그렇게 생각해

해당 응답에서 "난 멋져"라는 메시지 뿐만 아니라 "나는 멋져"라는 응답에 반응 할 수 있고, 동일한 단어를 이렇게 대치어로 해서 훨씬 유연하게 우리가 원하는 챗봇을 만들 수 있습니다.

[[Tutorial] TRIGGERS](http://kjw1801.tistory.com/entry/Tutorial-TRIGGERS)

**Open-Ended Triggers**

기존에는 "hello bot"이라는 트리거가 입력이 되어야 지만 정상적인 응답이 이루어 졌습니다. "hello there" 또는 "hello robot"등과 말 할 수 있지만 오직 "hello bot"이라는 트리거에만 응답합니다.

이번에 배울 기능의 핵심은 바로 와일드카드(Wildcards)입니다. 와일드 카드를 사용한 트리거에 일부 단어를 무한대로 표현할 수 있습니다. 아래 예제를 살펴 보겠습니다.

+ my name is \*

- Nice to meet you, <star1>!

+ \* told me to say \*

- Why would <star1> tell you to say "<star2>"?

- Did you say "<star2>" after <star1> told you to?

+ i am \* years old

- A lot of people are <star1> years old.

예제 트리거를 통해 사용자는 "My name is martin"또는 "I 've 20 years old"라고 말할 수 있으며 RiveScript는 이 메시지를 모두 일치시킬 수 있습니다. 와일드 카드는 이름이나 숫자와 같은 "가변"데이터를 포함 할 수있는 메시지를 일치시키는 데 매우 유용합니다. 아래의 예제를 보겠습니다.

+ where is \*

- Where it belongs.

- Where you left it.

- Where the heart is.

이러한 방법은  봇이 주제에 대해 가짜 지식(fake knowledge)을 가질 수 있도록 도와줍니다.

와일드 카드(\*)를 사용하면 일반적인 질문(이름,나이 등)에 대한 트리거를 작성 할 수 있습니다.

또한 트리거에 없는 질문을 있더라도 적어도 그 주제에 맞는 일반적인 응답을 줄 수 있습니다.

예제에서 이미 눈치채셨겠지만 <star1>과 <star2>태그는 응답시에 와일드 카드와 일치하는 단어를 반복 할 수 있습니다.

하나의 와일드 카드만 있는 경우에는 번호를 제거한 <star>태그를 사용할 수 있습니다.

+ who is \*

- I don't know who <star> is.

**Catch-All Trigger**

일치하는 트리거가 없을 때 RiveScript는 "ERR: No Reply Matched"라고 응답합니다. 우리는 와일드 카드를 사용해서 모든 응답에 대응하는 트리거를 작성 할 수 있습니다.

+ \*

- I'm not sure how to reply to that.

- Try asking your question a different way.

- Let's change the subject.

사용자의 메시지가 트리거와 일치하지 않았을때 에러 표시를 제거 할 수 있습니다.

**Specialized Wildcards**

와일드 카드는 유용한 기능입니다. 그렇지만 무조건적인 와일드 카드가 문제가 될 경우가 있습니다. 예를 들어 나이를 물을 때 "i am \* years old"일때 숫자로 24를 표현할 수 있지만 문자로 "I am twenty four years old"도 동일한 응답을 하게 됩니다.

여기에서 우리는 2가지 심볼을 통해서 숫자와 문자를 구별 할 수 있습니다. #는 숫자와 일치하는 와일드 카드이고 \_는 숫자를 제외한 공백과 문자와 일치하는 와일드 카드 심볼입니다. 아래 예제에서 보듯이 똑같아 보이지만 서로 다른 심볼을 통해서 여러 개의 트리거를 가질 수 있으며 원하는 방식대로 동작 가능하게 할 수 있습니다.

+ i am # years old

- A lot of people are <star> years old.

+ i am \_ years old

- Tell me that again but with a number this time.

+ i am \* years old

- Can you use a number instead?

사용하는 심볼 타입에 관계 없이 <star>태그는 사용 가능합니다.

[Tutorial] Alternatives and Optionals  
  
와일드 카드와 같은 것을 사용하고 싶지만 가능한 단어를 몇 가지로 제한하고 싶다면 어떻게해야할까요?

몇 가지 예를 통해서 보도록 하겠습니다.

+ what is your (home|office|cell) number

- You can reach me at: 1 (800) 555-1234.

+ i am (really|very|super) tired

- I'm sorry to hear that you are <star> tired.

+ i (like|love) the color \*

- What a coincidence! I <star1> that color too!

- I also have a soft spot *for* the color <star2>!

- Really? I <star1> the color <star2> too!

- Oh I <star1> <star2> too!

이 예에서 사용자는  "what is your home number"또는 "what is your office number"또는 "what is your cell number"라고 말할 수 있습니다. home, office, cell중에서 일치한다면 이에 대응한 응답을 받을 수 있습니다.

또한, 와일드 카드처럼 <star> 태그도 사용할 수 있다는 것입니다. 하나 이상의 옵션을 포함 할 수 있습니다.

+ how [are] you

- I'm great, you?

+ what is your (home|office|cell) [phone] number

- You can reach me at: 1 (800) 555-1234.

+ i have a [red|green|blue] car

- I bet you like your car a lot.

[options]선택 사항이 없어도 트리거가 실행 되도록 합니다. 그렇기 때문에 <star> 태그를 사용 할 수는 없습니다.

optionals로 작성한 트리거를 키워드(keyword) 라고 합니다. 사용자가 메시지에 맞는 또는 생략 가능한 말에 응답할 것입니다.

+ [\*] the machine [\*]

- How do you know about the machine!?

[\*] 옵션을 사용하여 메시지의 일부를 무시할 수 있습니다.

[[Tutorial] Arrays in Triggers](http://kjw1801.tistory.com/entry/Tutorial-Arrays-in-Triggers)

이번 시간에는 배열을 사용하는 방법에 대해서 배워보겠습니다. 아래의 예제를 살펴보면, "what color is my blue shirt?"라고 물어보는 질문에 color와 와일드 카드를 통해서 bot은 응답하도록 되어있습니다.

+ what color is *my* (red|blue|green|yellow) \*

- Your <star2> is <star1>, silly!

()안에 몇가지 color들이 정의된 것을 볼 수 있습니다. 이렇게 사용해도 되지만 color가 필요한 트리거가 몇가지 더 생길 수가 있습니다. 그럼 그때마다 (red|blue|green|yellow)이 내용을 복사해서 붙여넣기 해야되고 그렇게 한다고 하더라도 나중에 black, white등의 color가 새로 추가 될때에는 기존의 스크립트를 다 찾아서 수정하기에는 어려움이 있습니다. 그래서 RiveScript에서는 배열(**Arrays**)을 통해서 이러한 문제점을 해결 할 수 있습니다.

배열은 ! array 커멘드를 통해서 정의가 됩니다. 이전 시간에 정의하는 모든 커멘드는 begin.rive에 작성되어야 된다고 설명드렸습니다. 그렇게 때문에 배열 또한 begin.rive에 작성되어야 합니다.

! array colors = red blue green yellow

이제 우리는 배열을 정의할 수 있습니다. 다음 예제를 통해서 배열을 어떻게 활용하는지 알아보겠습니다.

+ what color is *my* (@colors) \*

- Your <star2> is <star1>, silly!

- Do I look dumb to you? It's <star1>!

+ i am wearing a (@colors) shirt

- Do you really like <star>?

예제에서 보시는거와 같이 <star>태그를 사용할 수 있으며 사용하지 않는 경우에는 ()를 제거해주면 됩니다.

// Without parenthesis, the array doesn't go into a <star> tag.

+ what color is my @colors \*

- I don't know what color your <star> is.

배열은 optionals에서도 적용할 수 있지만 <star>태그를 안에 사용할 수 는 없습니다.

왜냐하면 옵션은 생략 가능 하기 때문입니다.

// Arrays in an optional

+ i am wearing a [@colors] \*

- Do you really like <star>?

배열을 정의할때 배열 항목을 공백 (단일 단어에 유용함) 또는 파이프 기호 (|)로 구분할 수 있습니다.

// Single word array items

! array colors = red blue green yellow

// Multiple word items

! array blues = light blue|dark blue|medium blue

배열을 정의 할 때 Continuations를 사용하면 각 행의 항목을 공백으로 또는 파이프로 나뉠 수 있습니다.

// A lot of colors!

! array colors = red blue green yellow orange cyan fuchsia magenta

^ light red|dark red|light blue|dark blue|light yellow|dark yellow

^ light orange|dark orange|light cyan|dark cyan|light fuchsia

^ dark fuchsia|light magenta|dark magenta

^ black gray white silver

^ light gray|dark gray

[[Tutorial] Priority Triggers](http://kjw1801.tistory.com/entry/Tutorial-Priority-Triggers)

우리는 지난시간까지 트리거에 대한 대부분의 내용을 학습했습니다.

마지막으로 가중치 또는 우선순위 트리거라 불리는 내용에 대해서 배워 보겠습니다. {weight}태그를 가중치 태그라고 이전 시간에 정의했으므로 우선순위 트리거라고 앞으로 설명하겠습니다.

우선순위 트리거는 다른 트리거보다 우선순위가 높습니다.  이는 트리거가 사용자의 메시지와 얼마나 잘 일치 하는지를 임의로 조정하는 데 유용합니다. 다음과 같은 두 가지 트리거가 있다고 가정 해 보겠습니다.

+ google \*

- Google search: <a href="http://google.com/search?q=<star>">Click Here</a>

+ \* perl script

- You need Perl to run a Perl script.

예제를 실행해서 "google write perl script"라고 메시지를 보내면 우리가 예측하기에는 goolge link를 보여줄 것이라고 생각 할 수 있습니다. 그렇지만 실제를 테스트를 하게 되면 "\* perl script" 트리거로 인식해서 응답하는 것을 볼 수 있습니다.

우리는 "+ google \*"트리거에 {weight}태그를 추가하여 다른 트리거에 비해 더 중요하게 가중치를 추가 할 수 있습니다. 다음은 트리거에 가중치를 준 예제입니다.

+ google \*{weight=10}

- Google search: <a href="http://google.com/search?q=<star>">Click Here</a>

+ \* perl script

- You need Perl to run a Perl script.

기본적으로 가중치를 주지 않은 트리거는 1로 판단됩니다. 그리고 가중치의 값이 높을 수록 더 높은 우선 순위를 가집니다. 0 또는 음수의 값을 가중치로 올 수 없으니 주의 하시기 바랍니다.

[[Tutorial] Redirections](http://kjw1801.tistory.com/entry/Tutorial-Redirections)

이번 시간에는 리디렉션(Redirections)에 대해서 알아보겠습니다.

+ hello

- Hi there!

- Hey!

- Howdy!

+ hey

@ hello

+ hi

@ hello

이 예제에서 보기에 "hey", "hi"로 메시지를 보냈을 때 hello라는 트리거가 동작하도록 리디렉션을 정의하고 있습니다.

+ \* or something{weight=100}

- Or something. {@ <star>}

위의 예제를 합쳐서 "hi or something"라고 메시지를 보내면 "Or something. Hey!" 라고 응답할 것입니다. 와일드 카드로 정의된 트리거에 {@ <star>}태그가 리디렉션을 요청합니다. 하나의 와일드 카드만 사용할 경우에는 <@>과 같이 축약해서도 사용 할 수 있습니다.

+ hello \*

- {@ hello} <@>

+ hello

- Hi there!

+ are you a bot

- How did you know I'm a machine?

다음 예제를 실행하여 "Hello, are you a bot?" 라고 입력하면 "Hi there! How did you know I'm a machine?"라고 응답할 것입니다.

[[Tutorial] Short Discussions](http://kjw1801.tistory.com/entry/Tutorial-Short-Discussions)

이야기를 이어나갈때 어떻게 해야 할까요? 다음은 %(**Previous**)커맨드를 배워보겠습니다.

+ knock knock

- who is there?

+ \*

% who is there?

- <star> who?

+ \*

% \* who

- LOL! <star>! That's funny!

해당 예제를 실행해보면 "knock knock"라고 메시지를 입력했을 때 응답까지는 이해가 될 것입니다. 그다음 %커맨드가 무슨 의미인지를 알아야 되는데. "- who is there?"라는 응답이 왔을 때 그 다음 실행하게 되는 트리거를 정의한 것이라고 볼 수 있습니다. 2번째 트리거는 어떤 단어가 와도 봇은 이것을 가지고 이야기를 이어 나갈 수 있게 됩니다. 다음은 또 다른 예제 입니다.

! array colors = red blue green yellow

+ i have a dog

- What color is it?

+ (@colors)

% what color is it

- That's a silly color for a dog!

예제에서 개를 가지고 있다고 했을때 봇은 어떤 개그 어떤 색인지 물어보고 있습니다. 여기에서 배열에 저장된 칼러를 입력하면 봇은 이어 나갈 수 있지만 엉뚱한 대답을 하게 된다면 봇은 평소와 같은 새 메시지에 반응할 것입니다.

[[Tutorial] Conditionals](http://kjw1801.tistory.com/entry/Tutorial-Conditionals)

**Learning Things**

우리는 RiveScript의 커맨드에 대해서 배웠습니다. 그런데 봇이 우리 이름도 기억하지 못한다면 이 대화가 무슨 소용일까요?

RiveScript는 사용자에 대한 변수를 저장하고 반복할 수 있는 기능이 있습니다. 사용자의 변수를 저장하기 위해 <set>태그를 사용하고 변수를 검색하기 위해 <get>태그를 사용합니다. 다음 예제를 통해서 우리는 어떻게 사용자 변수를 저장하고 불러 올 수 있는지 알 수 있습니다.

+ my name is \*

- <set name=<star>>It's nice to meet you, <get name>.

+ what is my name

- Your name is <get name>, silly!

+ i am # years old

- <set age=<star>>I will remember that you are <get age> years old.

+ how old am i

- You are <get age> years old.

예제를 실행시켜보면 봇은 우리의 나이와 이름을 기억하고 있다는 것을 알 수 있습니다. 또한 변수에 대해서 이야기 하고 있다면 begin.rive에 정의한 변수도 이용 할 수 있지 않을까? 라는 궁금증이 생길 수 있습니다. 유사한 방식으로 변수의 값을 불러 올 수 있습니다. 아래의 예제를 참고하세요.

! version = 2.0

// Bot variables

! var name = Tutorial

! var age = 5

// The user can ask the bot its name too!

+ what is your name

- You can call me <bot name>.

- My name is <bot name>.

+ how old are you

- I am <bot age> years old.

이제 봇에게 이름과 나이를 물어보면 변수의 저장된 값을 대답할 것입니다.

**Writing Conditionals**

봇은 사용자의 대한 정보를 학습함으로써, 조건문은 그 변수의 값을 기반으로 응답을 하게 됩니다. 그렇지만 봇에게 내 이름을 말하기전에 내 이름을 물어보게 된다면 "Your name is **undefined**, silly!" 라고 응답할 것입니다. **undefined**이라고 나오는 이유는 변수의 값이 저장되어 있지 않기 때문이죠. 이는 봇이 단순 프로그램처럼 느껴질것입니다.

우리는 조건문을 사용하여 봇이 응답하기 전에 내이름을 알고 있는지 모르는지 모른다면 어떻게 응답해야 되는지를 정의 할 수 있습니다. 아래의 예를 보겠습니다.

+ what is my name

\* <get name> == undefined => You never told me your name.

- Your name is <get name>, silly!

- Aren't you <get name>?

봇에게 이름을 묻는다면 봇은 정의 되지 않는 내용에 대해서 조건문을 수행할 것입니다. 그리고 이름이 입력 되어 있다면 응답중에서 하나를 보여 줄 것입니다.

조건은 변수를 값과 비교하는데 사용됩니다. 변수를 다른 변수와 비교할 수도 있습니다. 다음 예제를 보겠습니다.

+ my name is \*

\* <formal> == <bot name> => Wow, we have the same name!<set name=<formal>>

\* <get name> == undefined => <set name=<formal>>Nice to meet you!

- <set oldname=<get name>><set name=<formal>>

^ I thought your name was <get oldname>?

행이 부족하다면 ^ Continuation커맨드를 사용해서 공간을 확보 할 수 있습니다. 조건은 위에서 아래로 점검됩니다.

다음과 같은 비교 연산자를 사용할 수 있습니다.

== equal to

eq equal to (alias)

!= not equal to

ne not equal to (alias)

<> not equal to (alias)

숫자를 비교하기 위해서는 아래의 비교 연산자를 사용해야 합니다.

< less than

<= less than or equal to

> greater than

>= greater than or equal to

다음은 다양한 비교 연산자를 사용한 예입니다.

+ what am i old enough to do

\* <get age> == undefined => I don't know how old you are.

\* <get age> > 25 => You can do anything you want.

\* <get age> == 25 => You're old enough to rent a car with no extra fees.

\* <get age> > 21 => You're old enough to drink, but not rent a car.

\* <get age> == 21 => You're exactly old enough to drink.

\* <get age> > 18 => You're old enough to gamble, but not drink.

\* <get age> == 18 => You're exactly old enough to gamble.

- You're not old enough to do much of anything yet.

[[Tutorial] Topics](http://kjw1801.tistory.com/entry/Tutorial-Topics)

**LABELED SECTIONS**

레이블이 지정된 섹션에는 세 가지 유형이 있습니다.

레이블은 > 기호를 사용하여 정의되며 <로 끝납니다.

완료되면 라벨이 붙은 모든 섹션이 제대로 닫혀 있어야합니다 (파일의 끝에 있더라도).

스타일을 위해서 레이블이 붙은 섹션의 내용도 들여 써야합니다.

라벨이 지정된 섹션은 서로 안에 삽입 할 수 없습니다.

**Topics**

토픽(Topic)은 트리거의 논리접 그룹을 말합니다. 사용자가 토픽안에 있으면 봇은 토픽 안에 있는 트리거에만 동작합니다.

+ i hate you

- You're really mean! I'm not talking again until you apologize.{topic=sorry}

> topic sorry

// This will match if the word "sorry" exists ANYWHERE in their message

+ [\*] sorry [\*]

- It's OK, I'll forgive you!{topic=random}

+ \*

- Nope, not until you apologize.

- Say you're sorry!

- Apologize!

< topic

해당 예제에서는 "i hate you"라고 트리거가 동작할 때 토픽은 sorry로 넘어가며 해당 토픽에서 빠져 나오기 위해서는 sorry라를 트리거를 작동시켜야지만 빠져 나올 수 있습니다. 태그는 가장 기본적으로 정의되는 {topic=random}토픽이며 sorry 토픽에서 빠져나오기 위해서 {topic=random}태그를 사용하게 됩니다.

토픽은 상속 할 수 있으며 다른 트리거를 포함 시킬 수 있습니다. 그렇지만 이것은 고급 사용자를 위한 것이고 Tutorial 범위에 벗어납니다. 이에 대해 자세한 예를 보고 싶다면 rpg.rive를 참고하시기 바랍니다.

존재하지 않는 토픽을 지정하면 더이상 대화를 할 수 없냐고 물어 볼 수 있지만 RiveScript는 이를 감지해서 자동으로 기본 토픽("random")으로 되돌립니다.

[[Tutorial] The Begin Block](http://kjw1801.tistory.com/entry/Tutorial-The-Begin-Block)

Begin Block은 이전에 저희가 살펴봤던 RiveScript brain의 선택적 기능입니다.

brain에 관한 자세한 내용은 [[Tutorial] The Begin File](http://kjw1801.tistory.com/entry/Tutorial-The-Begin-File) 에서 설명하고 있습니다.

Begin Block또한 begin.rive에 포함되고 토픽과 비슷한 형태로 동작합니다. 예를 들어 보겠습니다.

> begin

+ request

- {ok}

< begin

Begin Block 응답을 가져오기 위한 전처리와 후처리로 사용됩니다.

Begin Block이 있으면 사용자가 말하는 각 메시지에 대해 요청 트리거가 작동됩니다.

응답에 {ok}태그가 있으면 사용자 메시지에 대한 응답을 가져오게 됩니다.

> begin

// If we don't know their name, set the new\_user topic and continue.

+ request

\* <get met> == undefined => <set met=true>{topic=new\_user}{ok}

- {ok}

< begin

> topic new\_user

+ \*

- Hi! I'm <bot name>! I'm a chatbot written in RiveScript.\s

^ What is your name?{topic=asked\_name}

< topic

> topic asked\_name

+ #

- Your name is a number?

+ \*

- I only want your first name.

+ \_

- <set name=<formal>>Nice to meet you, <get name>!{topic=random}

< topic

request는 어떤 메시지는 들어 왔을 때 {topic=new\_user}로 넘어가게 됩니다. 그리고 이름을 받도록 토픽을 설정해 놨는데 숫자나 특수문자가 아닌 <formal>문자가 들어 왔을 때 이름을 저장하고 토픽 {topic=random}으로 넘어오게 됩니다.

[[Tutorial] MORE DEFINITIONS](http://kjw1801.tistory.com/entry/Tutorial-MORE-DEFINITIONS)

begin.rive로 다시 돌아가서 유용한 2가지 타입이 있습니다.

**Global Variables**

전역 변수(Global Variables)는 봇 변수(bot variables)와 비슷하지만 봇과 관련은 없습니다.

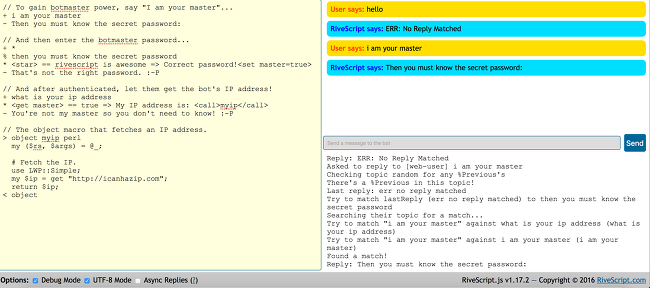
전역 변수는 RiveScript 인터프리터에 의해 정의 될 수 있습니다. 이번에는  RiveScript 인터프리터에 직접 영향을주는 두 가지 특수한 전역 변수에 대해서 배워 보겠습니다.

**Debug Mode**

! global debug = true

! global debug = false

이 전역 변수는 RiveScript 인터프리터에서 디버그 모드를 켜거나 끌 수 있습니다. 디버그 모드는 https://play.rivescript.com/에서 Debug Mode를 클릭하면 실제 어떻게 출력 되는지 볼 수 있습니다.



**Recursion Depth**

! global depth = 50

이 전역 변수는 RiveScript 응답은 서로 리다이렉트 할 수 있기 때문에 무한 재귀를 피할 수있는 보호 기능이 있습니다 (예를 들어, 다른 트리거로 리다이렉트되고 그 트리거가 처음 트리거로 리다이렉트하는 등).

기본적으로 제한 값으로 50으로 설정이 됩니다. 즉, 50개의 리디렉션 후에 응답을 찾을 수 없으면 "ERR: Deep Recursion Detected"라고 응답합니다.  또한 이 값은 0보다 큰수여야 합니다.

**Person Substitutions**

1인칭 대명사와 2인칭 대명사를 교환하기 위한 기능입니다. brain에서 정의해기 때문에 begin.rive에서 작성 되어야 합니다. 대체를 정의하기 위한 방법은 다음과 같습니다.

! person i am = you are

! person you are = i am

! person i'm = you're

! person you're = I'm

! person my = your

! person your = my

! person you = I

! person i = you

대체를 호출 하려면 <person>태그(<star>대치 가능)를 사용하거나 {person}...{/person}로 사용할 수 있습니다. 실제 예를 통해서 테스트를 해보겠습니다.

+ say \*

- Umm... "<person>"

실제로 구동 상태에서 "say I am the greatest"라고 메시지를 보내면 봇은 "Umm... "you are the greatest"라고 응답할 것입니다. 만약 대치 하지 않았다고 한다면 봇은 "i am the greatest"라고 말했을 것입니다.

[[Tutorial] TAGS](http://kjw1801.tistory.com/entry/Tutorial-TAGS)

지금까지 우리는 RiveScript의 TAG에 대해서 공부 했습니다.

우리가 공부했던  태그의 예는 <star>, <get>, {topic}입니다.

실제로는 이 보다 더 많은 태그가 있고 우리는 이를 사용할 수 있습니다!

이번에는 우리가 배우지 않은 태그와 태그 사용 방법을 배워보겠습니다.

태그는 일반적으로 화살괄호(<>)안에 태그를 입력하는 형태로 사용이 되며 중괄호({})는 주변의 텍스를 수정하는데 사용이 됩니다. 태그는 일반적으로 특별히 명시된 경우를 제외하고 모든 RiveScript 명령과 함께 사용할 수 있습니다.

* <botstar>, <botstar1> - <botstarN>

이 <star>태그는 와 비슷하지만 % Previous에 있는 와일드 카드를 가져옵니다. 다음은 그 예입니다.

+ i bought a new \*

- Oh? What color is your new <star>?

+ (@colors)

% oh what color is your new \*

- <star> is a pretty color for a <botstar>.

실제로 실행해보면 <botstar>태그는 이전의 트리거에 표시된 즉 %에 사용된 와일드카드를 가지고 오게 됩니다. "i bought a new car"라고 입력하게 되면 car = <star>가 됩니다. 그리고 "blue"라고 컬러 값을 입력 했을 때 blue = <star>가 되지만 이전에 car = <botstar>가 되게 됩니다. 잘 이해가 안 될 수도 있지만 자주 사용하다보면 어떤 식으로 구동하는지 잘 알 수 있을 것입니다. 물론 와일드 카드 수에 따라서 <botstar1>로 사용할 수 있습니다.

* <input>, <reply>

<input>, <reply> 태그는 사용자 및 봇이 각각 보낸 이전 메시지를 표시하는데 사용됩니다. 이전의 9개 메시지까지가 저장이 되며 <input1> 에서 <input9>까지 <reply1>에서 <reply9>까지 사용이 가능합니다. 다음은 태그가 사용되는 몇가지 예제입니다.

// If the user repeats the bot's previous message

+ <reply>

- Don't repeat what I say.

// If the user keeps repeating themselves over and over.

+ <input1>

\* <input1> == <input2> => That's the second time you've repeated yourself.

\* <input1> == <input3> => If you repeat yourself again I'll stop talking.

\* <input1> == <input4> => That's it. I'm not talking.{topic=sorry}

- Please don't repeat yourself.

// An example that uses both tags

+ why did you say that

- I said, "<reply>", because you said, "<input>".

* <id>

<id>는 기본적으로  localuser이며 RiveScript는 사용자 ID를 통해서 여러 사용자를 구분합니다. 동일한 RiveScript 인터프리터를 사용하여 여러 사용자에 대한 응답을 제공 할 수 있으며 ID를 기반으로 사용자 변수를 별도로 유지합니다. 다음은 일반 사용자와 관리자를 구분하는 방법의 예입니다.

! var master = kirsle

+ am i your master

\* <id> == <bot master> => Yes, you are. Hi Kirsle!

- No, <bot master> is my master, and you are <id>.

* <bot>

<bot>태그는 봇 변수를 검색하는데 사용됩니다. 또한 봇 변수를 설정하는데 사용할 수도 있습니다. 봇 변수는 RiveScript 인터프리터 인스턴스에 대해 "전역"으로 간주 될 수 있습니다. 즉, 봇의 이름을 Aiden으로 설정하면 사용자의 ID와 상관없이 요청하는 모든 사람의 이름이 Aiden이됩니다. 이것은 특정 사용자 ID에 연결된 사용자 변수와 대조됩니다.

+ what is your name

- You can call me <bot name>.

+ tell me about yourself

- I am <bot name>, a chatterbot written by <bot master>.

// Setting a bot variable dynamically

+ i hate you

- Aww! You've just ruined my day.<bot mood=depressed>

* <env>

<env> 태그는 전역 변수를 검색하는 데 사용됩니다. 또한 전역 변수를 설정할 때도 사용할 수 있습니다.

예를 들어 RiveScript 인터프리터가 모든 환경 변수를 RiveScript 전역으로 복사하면 CGI 기반 RiveScript 봇이 사용자에게 자신의 IP 주소를 알릴 수 있습니다.

+ what is my ip

- Your IP address is: <env REMOTE\_ADDR>

다음은 전역변수를 설정하는 방법에 대한 예제입니다. 이 예제에서 관리자는 디버그 모드를 켜거나 끌 수 있습니다.

+ set debug mode (true|false)

\* <id> == <bot master> => <env debug=<star>>Debug mode set to <star>.

- You're not my master.

* <add>, <sub>, <mult>, <div>

이 태그는 숫자 변수를 각각 더하거나 빼거나 곱하거나 나눌 수 있습니다.

+ give me 5 points

- <add points=5>You have been given 5 points. Your balance is: <get points>.

정의되지 않는 변수 예를 들어 points를 사용하면 기본 0으로 초기화됩니다. 그리고 숫자가 아닐시에는 오류가 발생합니다. 또한 이 태그는 + Trigger와 함께 사용할 수는 없습니다.

<formal>, <sentence>, <uppercase>, <lowercase>

<formal>은 각 단어의 첫글자를 대문자로 만듭니다.

<sentence>는 첫 단어를 대문자로 만듭니다.

<uppercase>, <lowercase>전체 문자열을 대문자 또는 소문자로 만듭니다.

* <call>

이 태그는 매크로를 호출하는 데 사용됩니다.

// Call a macro named "reverse" and give it an argument

+ say \* to me in reverse

- <call>reverse <star></call>

다만 이 예제는 우리가 배우지 않은 ["Object Macros"](https://www.rivescript.com/docs/tutorial#object-macros)에 해당 되는 내용이기 때문에 자세한 내용은 다음에 다루도록 하겠습니다.

[[Tutorial] CONCLUSION](http://kjw1801.tistory.com/entry/Tutorial-CONCLUSION)

RiveScript 기본적인 Tutorial을 끝냈습니다. 지금까지 RiveScript 코드 작성 방법에 대해 배웠으며 이를 토대로 chatterbot을 개발하는데 도움이 되었으면 합니다. 우리가 다루지 못한 내용으로는 Object Macros가 있지만 이에 대한 자세한 내용은 https://www.rivescript.com/docs/tutorial#object-macros 을 참조하십시오. 우리는 튜토리얼에서 배운 내용을 바탕으로 안드로이드에서 사용 가능한 챗봇을 개발하는 부분에 대해서 실습할 수 있도록 하겠습니다. 감사합니다.