Flip-a-coin.md

동전을 던진다  
동전 던지기를 수행하는 opsdroid 기술입니다.

설정

먼저 opsdroid 스킬 디렉터리를 만들어야 합니다. 파일 시스템의 어느 곳에나 위치할 수 있으므로 나중에 위치를 기억하기만 하면 됩니다. 이 예시에서는 ~/opsdroid/skills에 새 폴더를 만듭니다.

$ mkdir -p ~/opsdroid/skills/flip-coin

기술  
그런 다음 이 폴더 안에 \_\_init\_py라는 파일을 만들고 다음 패키지를 가져와야 합니다.

from opsdroid.skill import Skill

from opsdroid.matchers import match\_regex

import random

둘째, 스킬에 대한 클래스를 만듭니다.

class CoinSkill(Skill):

@match\_regex('flip a coin')

async def flip\_a\_coin(self, message):

if random.randint(0, 1):

response = "Heads"

else:

response = "Tails"

await message.respond("{}".format(response))

배열  
셋째, 'configuration.yaml' 파일을 엽니다. opsdroid config edit 명령을 사용하여 이 작업을 자동으로 수행할 수 있습니다.  
  
그런 다음 스킬 섹션에 다음을 추가합니다.

skills:

flip-a-coin:

path: ~/opsdroid/skills/flip-coin

이제 구성을 저장하고 opsdroid를 다시 로드합니다.  
  
더 많은 스킬의 예시는 opsdroid 체크 아웃 예제 섹션을 통해 빌드할 수 있습니다.

how-are-you.md

How are you?  
opsdroid가 "how are you"라는 텍스트에 응답하도록 만드는 기본 스킬을 만들 것입니다. 자신만의 스킬을 만들기 위한 비디오 튜토리얼도 여기에서 이용할 수 있습니다.

도움이 필요하거나 확실하지 않은 사항이 있으면 매트릭스 채널에 가입하여 문의해 보십시오! 우리는 당신을 도울 수 있다면 매우 기쁩니다.

스킬 폴더  
opsdroid 스킬은 skill-<skillname> 라는 이름의 폴더 안에 있습니다. 이 폴더 안에 다음 파일이 있어야 합니다.

LICENSE

README.md

\_\_init\_\_.py

\_\_init\_.py 파일에 파이썬 기능을 모두 작성하겠지만, 스킬 폴더 안에 다른 유용한 파일도 포함할 수 있습니다.

스킬 빌드  
이제 'How are you' 스킬을 쓸 준비가 되었습니다. 우리는 함수를 쓰기 위해 opsdroid regex matcher를 사용할 것입니다. \_\_init\_.py 내부에서 가장 먼저 해야 할 일은 opsdroid에서 상속할 스킬 클래스를 가져오는 것입니다.  
  
스킬 클래스 가져오기

from opsdroid.skill import Skill

Regex Matcher 가져오기  
또한 기술 class의 방법을 사용자 언어의 구나 문장에 연결하는 데 사용할 수 있는 매처도 필요합니다.

from opsdroid.matchers import match\_regex

opsdroid에서 사용할 수 있는 모든 매처는 opsdroid.matchers에서 가져올 수 있습니다.  
  
데코레이터  
매처는 데코레이터로 사용됩니다. 데코레이터는 opsdroid가 그 기능을 이해할 수 있도록 합니다. 그래서 우리는 이 데코레이터를 사용해야 합니다. 그리고 우리는 opsdroid가 반응하는 regex를 사용해야 합니다.

from opsdroid.skill import Skill

from opsdroid.matchers import match\_regex

class MySkill(Skill):

@match\_regex('how are you?')

Regex 매처는 사용자가 보낸 모든 메시지에서 그것을 검색합니다. 만약 사용자가 how are you?를 입력한다면 opsdroid는 match\_regex 데코레이터 아래에 함수를 실행합니다.  
  
참고: 물음표가 없기 때문에 opsdroid가 텍스트와 함께 실행되지 않습니다.  
  
기능들  
opsdroid의 기술은 파이썬 class 메소드입니다. 앞에서 보았듯이, 데코레이터와 메소드가 함께 있으면 여러분이 상상할 수 있는 거의 모든 것을 할 수 있습니다.  
우리의 함수를 적어봅시다. 이로써 사용자가 how are you?라는 단어를 쓸 때 opsdroid가 무엇을 실행할지 알게 됩니다.

from opsdroid.skill import Skill

from opsdroid.matchers import match\_regex

class MySkill(Skill):

@match\_regex('how are you?')

async def how\_are\_you(self, message):

pass

opsdroid는 비동기 방식으로 제작됩니다. 즉, opsdroid가 반응할 모든 함수는 비동기 함수여야 합니다.  
  
참고: 모든 기능에는 self, message라는 두 가지 매개 변수가 사용됩니다.  
  
opsdroid 응답  
이제 우리의 opsdroid는 사용자가 말한 것에 맞추어 기술을 실행할 수 있습니다. 하지만 아직 아무 일도 일어나지 않을 것입니다. 텍스트에 대한 opsdroid 답변을 만들어 봅시다. 자신만의 메시지로 말이죠.

from opsdroid.skill import Skill

from opsdroid.matchers import match\_regex

class MySkill(Skill):

@match\_regex('how are you?')

async def how\_are\_you(self, message):

await message.respond('Good thanks! My load average is 0.2, 0.1, 0.1.')

우리의 기술은 완성되었고 opsdroid는 opsdroid 메인 페이지에서와 같이 반응할 수 있을 것입니다. 마지막으로 opsdroid 구성 파일에 스킬을 추가해 봅시다.  
  
구성에 기술 추가  
단순하게 하기 위해 다음과 같이 가정하겠습니다.  
  
구성 파일은 ~/.opsdroid/configuration.yaml 에 있습니다.  
당신의 스킬은 ~/documents/skill-howareyou 에 있습니다.  
configuration.yaml 파일을 열고 스킬 섹션에서 스킬의 이름과 경로를 추가합니다.

skills:

how-are-you:

path: /Users/username/documents/skill-howareyou

탭 대신 공백을 사용하는 것이 매우 중요합니다. opsdroid를 실행할 때 문제가 있으면 이 두 가지를 모두 확인하십시오. 공간의 문제일 수 있습니다.  
  
더 많은 스킬의 예시는 opsdroid 체크 아웃 예제 섹션을 통해 빌드할 수 있습니다.

index.md

기술 예제  
이 섹션에는 opsdroid에서 스킬을 사용하여 할 수 있는 여러 가지 예제가 포함되어 있습니다. 메뉴에서 예제를 선택하여 전체 소스 및 지침을 확인할 수 있습니다.  
  
각 예시에서는 처음으로 Opsdroid를 설치, 구성 및 실행했다고 가정합니다. 그렇지 않을 경우에는 퀵 스타트 가이드를 참조하십시오. 또한 opsdroid 놀이터에서 시험해 볼 수도 있습니다.  
  
또한 opsdroid 설명서에서 스킬이 어떻게 구성되었는지에 대한 자세한 정보를 찾을 수 있습니다.

.. toctree::

:maxdepth: 1

how-are-you

weather

flip-a-coin

stocks-portfolio

stocks-portfolio.md

주식 포트폴리오

우리는 opsdroid가 특정 주식의 정보를 반환할 수 있는 기술을 만들 것입니다. 그것은 또한 주식 포트폴리오를 저장할 수 있고 각각에 대한 현재 시세와 정보를 가지고 수익률을 올릴 수 있을 것입니다.

이 예시는 YFinace를 사용하여 주식의 정보를 얻고 SQLite3를 사용하여 봇을 닫을 때 포트폴리오를 유지합니다.

도움이 필요하거나 확실하지 않은 사항이 있으면 매트릭스 채널에 가입하여 문의해 주세요! 우리는 당신을 도울 수 있다면 매우 기쁠 것입니다.

스킬 구축

이제 기술 개발을 시작할 준비가 되었습니다! 먼저, 주식 포트폴리오 스킬을 위한 폴더를 만들어야 합니다. 위치를 선택하고 이름을 stocks-portfolio로 지정합니다.

mkdir /path/to/my/stocks-portfolio

디렉토리 내부에 configuration.yaml 파일을 만들 것이며, 다음 섹션에서 이 파일을 사용하여 스킬을 구성할 것입니다.

배열

이제 configuration.yaml 파일을 열고 스킬 섹션에 주식 포트폴리오 스킬을 추가해 보겠습니다.

skills:

stock:

path: /path/to/my/stocks-portfolio

필요한 입력 및 클래스

이제 우리의 기술이 Configuration.yaml 파일에 구성되었으므로, 우리는 계속 우리의 stocks-portfolio 폴더 안에 \_\_init\_py 파일을 만들고 스킬 작업을 시작할 것입니다!

우리가 가장 먼저 해야 할 일은 opsdroid에서 스킬 클래스와 regex 매처를 가져오는 것입니다. 그러면 yfinance와 sqlite3 모듈을 가져와야 할 것입니다. \_\_init\_.py 파일은 다음과 같아야 합니다.

import yfinance as yf

import sqlite3

from opsdroid.matchers import match\_regex

from opsdroid.skill import Skill

SQLite3에 연결하기

다음 단계는 \_\_init\_.py 파일을 SQLite 데이터베이스에 연결하는 것입니다. \_\_init\_.py 파일은 다음과 같아야 합니다.

import yfinance as yf

import sqlite3

from opsdroid.matchers import match\_regex

from opsdroid.skill import Skill

conn = sqlite3.connect('/path/to/my/database')

c = conn.cursor()

입력을 수신하기

이제 봇이 받을 내용을 수집해야 합니다. 먼저 스킬 클래스를 상속하는 클래스를 만들어 보겠습니다. 내부에서는 @match\_regex인 특수 opsdroid 함수를 사용할 것입니다. 이 함수는 메시지를 보내고 메시지 옆에 사용할 입력(.\*)을 수집하기 위해 @match\_regex(r"Stock: (.\*)") 과 같이 사용합니다. 그런 다음 message.regex.group(1)을 사용하여 액세스할 것입니다. 이제 봇에서 수신한 내용을 할당할 수 있습니다.

StockSkill 클래스는 다음과 같아야 합니다.

class StockSkill(Skill):

@match\_regex(r"Stock: (.\*)")

async def search\_info\_stock(self, message):

stock\_name = message.regex.group(1)

데이터 가져오기

이제 재미있는 파트입니다! YFinance 모듈을 사용하여 주식에 대한 데이터를 가져오고 모듈은 주식의 기호를 가져와서 정보를 반환할 것입니다. 자세한 내용은 설명서를 참조하십시오.

StockSkill 클래스를 다음과 같이 만듭니다.

class StockSkill(Skill):

@match\_regex(r"Stock: (.\*)")

async def search\_info\_stock(self, message):

stock\_name = message.regex.group(1)

stock = yf.Ticker(stock\_name)

이제 stock은 주식 기호를 입력할 때 YFinance로부터 받는 것입니다.

데이터 사용

데이터를 가져와 변수에 할당한 후 데이터를 사용할 수 있습니다! 이제 커넥터에 정보를 반환할 수 있습니다. 정보 속성을 사용하지만 다른 속성에 관심이 있는 경우 해당 문서를 사용할 수 있습니다.

YFinance 모듈의 데이터는 JSON으로 반환됩니다. 여기 있습니다.

{

"zip":"94103",

"sector":"Technology",

"fullTimeEmployees":3719,

"longBusinessSummary":"Unity Software Inc. operates a real-time 3D development platform. Its platform provides software solutions to create, run, and monetize interactive, real-time 2D and 3D content for mobile phones, tablets, PCs, consoles, and augmented and virtual reality devices. The company offers its solutions directly through its online store and field sales operations in North America, Denmark, Finland, the United Kingdom, Germany, Japan, China, Singapore, and South Korea, as well as indirectly through independent distributors and resellers worldwide. The company was founded in 2004 and is

headquartered in San Francisco, California.",

"city":"San Francisco",

"phone":"415-539-3162",

"state":"CA",

"country":"United States",

"companyOfficers":[

],

"website":"http://www.unity.com",

"maxAge":1,

"address1":"30 3rd Street",

"industry":"Software—Application",

"previousClose":154.18,

"regularMarketOpen":152.33,

"twoHundredDayAverage":123.33359,

"trailingAnnualDividendYield":"None",

"payoutRatio":0,

"volume24Hr":"None",

"regularMarketDayHigh":156.92,

"navPrice":"None",

"averageDailyVolume10Day":1915112,

"totalAssets":"None",

"regularMarketPreviousClose":154.18,

"fiftyDayAverage":151.33788,

"trailingAnnualDividendRate":"None",

"open":152.33,

"toCurrency":"None",

"averageVolume10days":1915112,

"expireDate":"None",

"yield":"None",

"algorithm":"None",

"dividendRate":"None",

"exDividendDate":"None",

"beta":"None",

"circulatingSupply":"None",

"startDate":"None",

"regularMarketDayLow":147.61,

"priceHint":2,

"currency":"USD",

"regularMarketVolume":1362006,

"lastMarket":"None",

"maxSupply":"None",

"openInterest":"None",

"marketCap":40567812096,

"volumeAllCurrencies":"None",

"strikePrice":"None",

"averageVolume":2200304,

"priceToSalesTrailing12Months":57.123283,

"dayLow":147.61,

"ask":152,

"ytdReturn":"None",

"askSize":800,

"volume":1362006,

"fiftyTwoWeekHigh":174.94,

"forwardPE":"None",

"fromCurrency":"None",

"fiveYearAvgDividendYield":"None",

"fiftyTwoWeekLow":65.11,

"bid":148,

"tradeable":false,

"dividendYield":"None",

"bidSize":1800,

"dayHigh":156.92,

"exchange":"NYQ",

"shortName":"Unity Software Inc.",

"longName":"Unity Software Inc.",

"exchangeTimezoneName":"America/New\_York",

"exchangeTimezoneShortName":"EST",

"isEsgPopulated":false,

"gmtOffSetMilliseconds":"-18000000",

"quoteType":"EQUITY",

"symbol":"U",

"messageBoardId":"finmb\_241908542",

"market":"us\_market",

"annualHoldingsTurnover":"None",

"enterpriseToRevenue":54.824,

"beta3Year":"None",

"profitMargins":-0.35116002,

"enterpriseToEbitda":-213.991,

"52WeekChange":1.1919534,

"morningStarRiskRating":"None",

"forwardEps":"None",

"revenueQuarterlyGrowth":"None",

"sharesOutstanding":270776992,

"fundInceptionDate":"None",

"annualReportExpenseRatio":"None",

"bookValue":"None",

"sharesShort":6390454,

"sharesPercentSharesOut":0.023599999,

"fundFamily":"None",

"lastFiscalYearEnd":1577750400,

"heldPercentInstitutions":0.61965,

"netIncomeToCommon":-317163008,

"trailingEps":"None",

"lastDividendValue":"None",

"SandP52WeekChange":0.14322305,

"priceToBook":"None",

"heldPercentInsiders":0.16246,

"nextFiscalYearEnd":1640908800,

"mostRecentQuarter":1601424000,

"shortRatio":3.2,

"sharesShortPreviousMonthDate":1607990400,

"floatShares":120820703,

"enterpriseValue":38934634496,

"threeYearAverageReturn":"None",

"lastSplitDate":"None",

"lastSplitFactor":"None",

"legalType":"None",

"lastDividendDate":"None",

"morningStarOverallRating":"None",

"earningsQuarterlyGrowth":"None",

"dateShortInterest":1610668800,

"pegRatio":-15.31,

"lastCapGain":"None",

"shortPercentOfFloat":0.048600003,

"sharesShortPriorMonth":5086368,

"impliedSharesOutstanding":"None",

"category":"None",

"fiveYearAverageReturn":"None",

"regularMarketPrice":152.33,

"logo\_url":"https://logo.clearbit.com/unity.com"

}

우리는 ['longName'], ['regularMarketOpen'], ['longBusinessSummary'] 를 원합니다. 이제 우리는 봇이 데이터를 전송하기를 원하므로 await message.respond()를 사용할 것입니다. 다음과 같이 보여야 합니다.

class StockSkill(Skill):

@match\_regex(r"Stock: (.\*)")

async def search\_info\_stock(self, message):

stock\_name = message.regex.group(1)

stock = yf.Ticker(stock\_name)

await message.respond(f"Name: {stock.info['longName']}")

await message.respond(f"Current Price: {stock.info['regularMarketOpen']} {stock.info['currency']}")

await message.respond(f"About: {stock.info['longBusinessSummary']}")

축하합니다! 이제 주식의 이름, 가격 및 정보를 알려주는 봇을 만들었습니다. Stock: <symbol-of-stock> 으로 이것을 할 수 있습니다. 봇의 기본 부분이 완성되면 포트폴리오/절약 섹션을 만들 수 있습니다.

포트폴리오에 저장 및 추가

표 작성

우리는 이제 이전에 연결했던 SQLite3 데이터베이스 파일을 사용하고 c.execute('''CREATE TABLE stocks (symbol text)''') 를 수행하여 테이블을 추가해야 합니다. 이 명령을 방금 만든 것과 유사한 함수로 실행할 것입니다. 또한 새 표를 만들 때 데이터베이스를 재설정해야 합니다. 다음과 같은 모양이어야 합니다.

@match\_regex(r"Reset Portfolio")

async def new\_table(self, message):

try:

c.execute('''DROP TABLE IF EXISTS stocks''')

c.execute('''CREATE TABLE stocks

(symbol text)''')

except:

await message.respond('Portfolio Creation Failed')

테이블과 데이터베이스 파일을 사용하여 다음 섹션을 준비했습니다!

포트폴리오에 추가

이제 우리는 주식 기호를 데이터베이스에 삽입하는 또 다른 기능을 만들 것입니다. c.execute("SELECT \* FROM stocks")를 통해 데이터베이스에서 정보를 가져오고 c.execute("INSERT INTO stocks VALUES (?)", (params,)) 을 통해 입력할 것입니다. 매개 변수는 주식 이름/기호와 동일합니다. 다음과 같이 보여야 합니다.

@match\_regex(r"Add Stock: (.\*)")

async def add\_stock(self, message):

stock\_name = message.regex.group(1)

params = (stock\_name)

c.execute("SELECT \* FROM stocks")

rows = c.fetchall()

c.execute("INSERT INTO stocks VALUES (?)", (params,))

conn.commit()

await message.respond(f'Stock "{stock\_name}" Added to Portfolio')

여기서 문제는 봇이 동일한 주식의 복제를 데이터베이스에 허용한다는 것입니다. 우리는 루프에 대한 몇 가지 방법으로 이 문제를 해결할 것입니다. 최종 결과는 다음과 같습니다.

@match\_regex(r"Add Stock: (.\*)")

async def add\_stock(self, message):

stock\_name = message.regex.group(1)

params = (stock\_name)

c.execute("SELECT \* FROM stocks")

rows = c.fetchall()

dupe\_name = []

for row in rows:

dupe\_name.append(row[0])

if stock\_name in dupe\_name:

await message.respond(f"Duplicate of {stock\_name}")

else:

c.execute("INSERT INTO stocks VALUES (?)", (params,))

conn.commit()

await message.respond(f'Stock "{stock\_name}" Added to Portfolio')

포트폴리오 나열

이제 쉬운 부분은 데이터베이스에 추가된 스톡을 나열하는 것입니다. 우리는 c.execute("SELECT \* FROM stocks")로 데이터베이스에 추가한 것과 동일한 방식으로 데이터베이스에서 데이터를 가져오게 됩니다. 데이터를 사용하여 데이터베이스의 모든 내용을 루프로 살펴볼 수 있습니다. 다음과 같은 모양이어야 합니다.

@match\_regex(r"List Stocks")

async def list\_info\_stock(self, message):

c.execute("SELECT \* FROM stocks")

rows = c.fetchall()

for row in rows:

stock = yf.Ticker(row[0])

await message.respond(f"Name: {stock.info['longName']}")

await message.respond(f"Price: {stock.info['regularMarketOpen']} {stock.info['currency']}")

await message.respond(f"About: {stock.info['longBusinessSummary']}")

그러나 문제가 있습니다. 사용자가 데이터베이스에 아무것도 없이 스톡 목록을 요청하면 오류가 발생합니다. 데이터베이스의 행 len이 0인지 확인하여 이 문제를 해결할 수 있습니다. 다음과 같이 보여야 합니다.

@match\_regex(r"List Stocks")

async def list\_info\_stock(self, message):

c.execute("SELECT \* FROM stocks")

rows = c.fetchall()

if len(rows) == 0:

await message.respond('No Stocks Added, Please Use Command "Add Stock:"')

for row in rows:

stock = yf.Ticker(row[0])

await message.respond(f"Name: {stock.info['longName']}")

await message.respond(f"Price: {stock.info['regularMarketOpen']} {stock.info['currency']}")

await message.respond(f"About: {stock.info['longBusinessSummary']}")

이제 주식에 대한 특정 정보를 추가, 나열하고 찾을 수 있습니다. 축하합니다! 당신의 opsdroid 여정에 행운을 빌어요! 최종 코드는 다음과 같습니다.

import yfinance as yf

import sqlite3

from opsdroid.matchers import match\_regex

from opsdroid.skill import Skill

conn = sqlite3.connect('stock\_data.db')

c = conn.cursor()

class StockSkill(Skill):

@match\_regex(r"Stock: (.\*)")

async def search\_info\_stock(self, message):

stock\_name = message.regex.group(1)

stock = yf.Ticker(stock\_name)

await message.respond(f"Name: {stock.info['longName']}")

await message.respond(f"Current Price: {stock.info['regularMarketOpen']} {stock.info['currency']}")

await message.respond(f"About: {stock.info['longBusinessSummary']}")

@match\_regex(r"Add Stock: (.\*)")

async def add\_stock(self, message):

stock\_name = message.regex.group(1)

params = (stock\_name)

c.execute("SELECT \* FROM stocks")

rows = c.fetchall()

dupe\_name = []

for row in rows:

dupe\_name.append(row[0])

if stock\_name in dupe\_name:

await message.respond(f"Duplicate of {stock\_name}")

else:

c.execute("INSERT INTO stocks VALUES (?)", (params,))

conn.commit()

await message.respond(f'Stock "{stock\_name}" Added to Portfolio')

@match\_regex(r"List Stocks")

async def list\_info\_stock(self, message):

c.execute("SELECT \* FROM stocks")

rows = c.fetchall()

if len(rows) == 0:

await message.respond('No Stocks Added, Please Use Command "Add Stock:"')

for row in rows:

stock = yf.Ticker(row[0])

await message.respond(f"Name: {stock.info['longName']}")

await message.respond(f"Price: {stock.info['regularMarketOpen']} {stock.info['currency']}")

await message.respond(f"About: {stock.info['longBusinessSummary']}")

@match\_regex(r"Reset Portfolio")

async def new\_table(self, message):

try:

file = open("stock\_data.db", "w")

file.close()

c.execute('''CREATE TABLE stocks

(symbol text)''')

await message.respond('New Portfolio Created')

except:

await message.respond('Portfolio Creation Failed')

weather.md

날씨  
우리는 opsdroid로 마을의 현재 날씨를 알려주는 기술을 만들 것입니다.  
  
이 튜토리얼은 OpenWeatherMap API를 사용하여 도시에서 날씨 정보를 가져옵니다. 무료 버전의 API만 있으면 되므로 등록 후 API 메뉴에서 키를 가져오십시오.  
  
도움이 필요하거나 확실하지 않은 사항이 있으면 매트릭스 채널에 가입하여 문의해 주십시오! 우리는 당신을 도울 수 있다면 매우 기쁠 것입니다.  
  
설정  
튜토리얼에서 날씨 정보를 원하는 도시와 사용할 시스템(매트릭 또는 임페리얼)을 선택할 수 있습니다. 이러한 설정은 opsdroid의 configuration.yaml 파일에 지정됩니다.  
  
우리의 \_\_init\_.py init 파일은 두 가지 기능을 포함합니다. 하나는 OpenWeatherMap API와 상호 작용하는 기능이고 하나는 opsdroid가 우리 도시의 날씨를 알려주는 데 사용할 기능입니다.  
  
스킬 빌드  
우리는 이제 기술을 연습할 준비가 되었습니다. 먼저 날씨 스킬을 위한 폴더를 만들어야 합니다. 위치를 선택하고 날씨 기술 이름을 지정하십시오.

mkdir /path/to/my/weather-skill

배열

이제 opsdroid의 configuration.yaml 파일을 열고 날씨 기술을 스킬 섹션에 추가해 보겠습니다.

skills:

weather:

city: <Your city, your country> # For accuracy use {city},{country code}

units: <metric/imperial> # Choose metric/imperial

api-key: <Your Api Key>

# Developing the skill

path: /path/to/my/weather-skill

no-cache: true

참고: 모든 opsdroid 실행에 스킬을 설치하도록 하려면 no-cache를 true로 설정해야 합니다.

날씨 스킬

이제 configuration.yaml 파일에 모든 구성 세부 정보가 설정되었으므로 날씨 스킬 폴더 안에 \_\_init\_py를 만들고 스킬 작업을 시작하겠습니다.

우리가 가장 먼저 해야 할 일은 opsdroid와 aiohttp 모듈에서 스킬 클래스와 regex\_matcher를 가져오는 것입니다. \_\_init\_.py 파일은 다음과 같아야 합니다.

from opsdroid.skill import Skill

from opsdroid.matchers import match\_regex

import aiohttp

날씨 데이터 가져오기

이제 OpenWeatherMap에서 날씨 데이터를 가져와야 합니다. 날씨 데이터를 얻기 위해 비동기 함수를 만들어 봅시다.

OpenWeatherMap의 현재 날씨 데이터 문서를 읽어보면, 우리가 사용해야 하는 API 끝부분이 http://api.openweathermap.org/data/2.5/weather?q= 이라는 것을 알 수 있을 것입니다.

\_\_init\_.py 파일을 다음과 같이 만듭니다.

from opsdroid.skill import Skill

from opsdroid.matchers import match\_regex

import aiohttp

async def get\_weather(config):

api\_url = "http://api.openweathermap.org/data/2.5/weather?q="

다음 사항을 OpenWeatherMap에 전달해야 합니다.

도시

단위

API 키

이러한 세부 정보는 이미 opsdroid 구성에 있으므로 구성에서 가져올 수 있습니다.

기능을 다음과 같이 만드십시오.

async def get\_weather(config):

api\_url = "http://api.openweathermap.org/data/2.5/weather?q="

parameters = "{}&units={}&appid={}".format(

config['city'], config['units'], config['api-key'])

api\_url과 파라미터에 가입하면 날씨 데이터를 얻는 데 필요한 전체 URL을 얻을 수 있습니다. 이제 aiohttp를 사용하여 세션을 시작하고 OpenWeatherMap에서 데이터를 가져올 것입니다.

이제 기능은 다음과 같아야 합니다.

async def get\_weather(config):

api\_url = "http://api.openweathermap.org/data/2.5/weather?q="

parameters = "{}&units={}&appid={}".format(

config['city'], config['units'], config['api-key'])

async with aiohttp.ClientSession() as session:

response = await session.get(api\_url + parameters)

우리의 날씨 기능이 점점 좋아지고 있습니다. 이제 JSON 형식으로 응답을 반환해야 합니다. 다행히도 aiohttp는 그것을 꽤 쉽게 만들어 주므로, 우리가 해야 할 일은 우리의 반응에 .json()을 추가하는 것입니다.

기능을 다음과 같이 만드십시오.

async def get\_weather(config):

api\_url = "http://api.openweathermap.org/data/2.5/weather?q="

parameters = "{}&units={}&appid={}".format(

config['city'], config['units'], config['api-key'])

async with aiohttp.ClientSession() as session:

response = await session.get(api\_url + parameters)

return await response.json()

이제 get\_weather 함수를 호출하면 aiohttp는 OpenWeatherMap API에서 모든 데이터를 가져와 파이썬 사전으로 우리에게 돌려줍니다.

response.json()은 다음과 같은 내용을 제공합니다.

{

"coord":

{

"lon": -0.13,

"lat": 51.51

},

"weather":

[

{

"id": 800,

"main": "Clear",

"description": "clear sky",

"icon": "01n"

}

],

"base": "stations",

"main":

{

"temp": 3.37,

"pressure": 1022,

"humidity": 86,

"temp\_min": 2,

"temp\_max": 5

},

"visibility": 10000,

"wind":

{

"speed": 2.1,

"deg": 310

},

"clouds":

{

"all": 0

},

"dt": 1511076000,

"sys":

{

"type": 1,

"id": 5089,

"message": 0.1668,

"country": "GB",

"sunrise": 1511076308,

"sunset": 1511107601

},

"id": 2639545,

"name": "London",

"cod": 200

}

이제 우리의 기능을 호출하면 날씨 데이터를 얻을 수 있을 것입니다. 다음 단계는 opsdroid가 현재의 날씨를 알려주도록 만드는 것입니다.

날씨 알려주기

이 기술로 우리의 get\_weather 기능을 호출한 다음 날씨 데이터를 통해 자세한 정보를 얻을 수 있습니다.

우리는 또한 우리가 선택한 매처(이 경우 regex)로 기술을 꾸며야 합니다.

\_\_init\_.py 파일을 다음과 같이 만듭니다.

from opsdroid.skill import Skill

from opsdroid.matchers import match\_regex

import aiohttp

async def get\_weather(config):

api\_url = "http://api.openweathermap.org/data/2.5/weather?q="

parameters = "{}&units={}&appid={}".format(

config['city'], config['units'], config['api-key'])

async with aiohttp.ClientSession() as session:

response = await session.get(api\_url + parameters)

return await response.json()

class WeatherSkill(Skill):

@match\_regex()

async def tell\_weather(self, message):

pass

우리는 날씨를 알려주기 위해 무엇이 opsdroid를 실행시킬지 선택해야 합니다. How's the weather? 라고 입력할 때의 opsdroid 실행기를 만들어 봅시다.

class WeatherSkill(Skill):

@match\_regex("How's the weather?")

async def tell\_weather(self, message):

pass

이제 우리는 기술을 실행시킬 방법을 얻었으므로 날씨 데이터를 얻기 위해 get\_weather 기능을 사용할 것입니다. 가독성을 위해 일부 날씨 데이터도 저장할 수 있는 변수를 만들 것입니다.

기능을 다음과 같이 만드십시오.

class WeatherSkill(Skill):

@match\_regex("How's the weather?")

async def tell\_weather(self, message):

weather\_data = await get\_weather(self.config)

temp = weather\_data['main']['temp']

humidity = weather\_data['main']['humidity']

city = weather\_data['name']

이제 남은 일은 opsdroid가 당신의 도시 기온을 나타내도록 만드는 것입니다. 우리는 message.respond()를 호출하여 그것을 할 수 있습니다.

tell\_weather 기능을 다음과 같이 변경합니다.

class WeatherSkill(Skill):

@match\_regex("How's the weather?")

async def tell\_weather(self, message):

weather\_data = await get\_weather(self.config)

temp = weather\_data['main']['temp']

humidity = weather\_data['main']['humidity']

city = weather\_data['name']

await message.respond("It's {} and {}% humidity in {}.".format(temp, humidity, city))

이제 여러분이 How's the weather를 opsdroid에 입력할 때마다 현재의 날씨를 알려줄 것입니다.

마지막으로, 우리는 개인적인 방법으로 get\_weather 기능을 스킬 클래스로 이동시킬 수 있는데, 이것은 우리가 그 구성에 대해 논쟁할 것이 없게 할 것입니다. 이것은 우리에게 결과적으로 다음과 같은 기술으로 남을 것입니다.

from opsdroid.skill import Skill

from opsdroid.matchers import match\_regex

import aiohttp

class WeatherSkill(Skill):

async def \_get\_weather(self):

api\_url = "http://api.openweathermap.org/data/2.5/weather?q="

parameters = "{}&units={}&appid={}".format(

self.config['city'], self.config['units'], self.config['api-key'])

async with aiohttp.ClientSession() as session:

response = await session.get(api\_url + parameters)

return await response.json()

@match\_regex("How's the weather?")

async def tell\_weather(self, message):

weather\_data = await self.\_get\_weather()

temp = weather\_data['main']['temp']

humidity = weather\_data['main']['humidity']

city = weather\_data['name']

await message.respond("It's {} and {}% humidity in {}.".format(temp, humidity, city))