FirstAlexaSkills Documentation

Release 0.2

Author

Jul 20, 2018

Contents:

`FirstAlexaSkills` is a Python package with example Alexa skills and utilities that will enable you to quickly and confidently develop third party Alexa Skills using AWS Lambda. The tutorials will guide you in developing skills with increasing complexity, starting from basics and using test-driven development to add additional layers of functionality.

CHAPTER 1

Installation

You can install `FirstAlexaSkills` with `pip`:

```
$ pip install firstalexaskills
```

See *Installation* for more information.

CHAPTER 2

Why a package for making Alexa Skills?

If you haven't developed Alexa skills before, the workflow can seem overwhelming. If you have developed skills before, you will agree that having a workflow for developing and testing the Lambda code really makes the difference between happiness and despair. This package addresses:

- Automated testing for Lambdas
- User-friendly generation of fake Alexa events
- Minimizes manual AWS console interactions
- Examples with progressive skill difficulty - from simple to complex
- Beginner friendly commandline tools - perfect for a Hackathon

CHAPTER 3

Layout

`FirstAlexaSkills` consists of three bits: the example_skills, commandline tools that suppport your development and tutorials that you can follow.

3.1 Installation

1. Install the `FirstAlexaSkills` package and its dependencies

   ```
   $ pip install firstalexaskills
   ```

2. Create an Amazon developer account
3. Create an AWS account (the first million AWS Lambda calls are free)
4. Create an IAM user called 'lambdaUser' for running and updating the AWS Lambda functions. The user will require the 'AWSLambdaFullAccess' permissions. Make sure to check "Programmatic access" when creating the user. The AWS website will generate an access key for you, which you can download in .csv file. We will use the credentials in the next step.
5. Configure the AWS CLI to use credentials of your new IAM user

   ```
   $ aws configure --profile lambdaUser
   ```

   Paste the information from credentials.csv into your command line:

   - Access key ID
   - Secret access key
   - an aws region of your choice - example: eu-west-1
   - format: json

6. Create an execution role for AWS Lambda functions.

   Preferably use 'basic_lambda_execute' as name for the role, since the package uses it as default. Unless you expect your function to require special privileges, like access to S3, use the official policy 'AWSLambdaExecute'.

7. Verify that the IAM user is setup correctly:

```
$ aws lambda list-functions --profile lambdaUser
{
    "Functions": []
}
```

3.2 Examples of creating Alexa third-party skills (ASK)

3.2.1 First Alexa Skill (alexa_skill_first)

Challenge: Create your first Alexa third-party skill with the following features:

1. Tell Alexa a specific phrase: "Alexa, ask <my_skill_name> to say something!" and get back a reply of your choice: "I can say whatever you want me to say".
2. Personalize your skill by teaching it facts about your friends: "Alexa, tell me something about <person's name>." and get back a reply: "<person's name> is crazy for pancakes with chocolate and banana."
3. Create a random generator of facts about cool subjects: "Alexa, tell me something cool about <subject>." and get back a random piece of trivia about <subject>: "

To accomplish the above, we will use a working skill which we will deploy to AWS, test and later modify. Rather than start of by learning all the details about AWS Lambda and the Alexa service APIs, we will focus on a simple, but reliable workflow that will allow us to develop much more complex Alexa skills later on.

Before we start:

Let's stop and think how can we do what the challenge is asking for. We will need to create three intents (saying something, person facts, subject trivia), a mapping for sentences and intents and also define some variables (name of the person, etc.) that Alexa needs to pass us alongside the intent name. We will need to create a third-party Alexa skill using ASK and we will also need a service, that will process the intents and generate replies for them. Our example skill comes with all the resources required to define an Alexa skill using ASK - utterances.txt, intent_schema.json. It also comes with working code for the webservice (AWS Lambda) - lambda_function.py. It also comes with tests, test cases and command-line tools that will allow us to easily modify the existing skill code. Let's dive in!

Suggested steps:

Deploy and test an existing skill (green boxes)

1. Start by creating a new directory where we will unpack the example Alexa skills

```
$ unpack_example_skills
$ cd example_skills
```

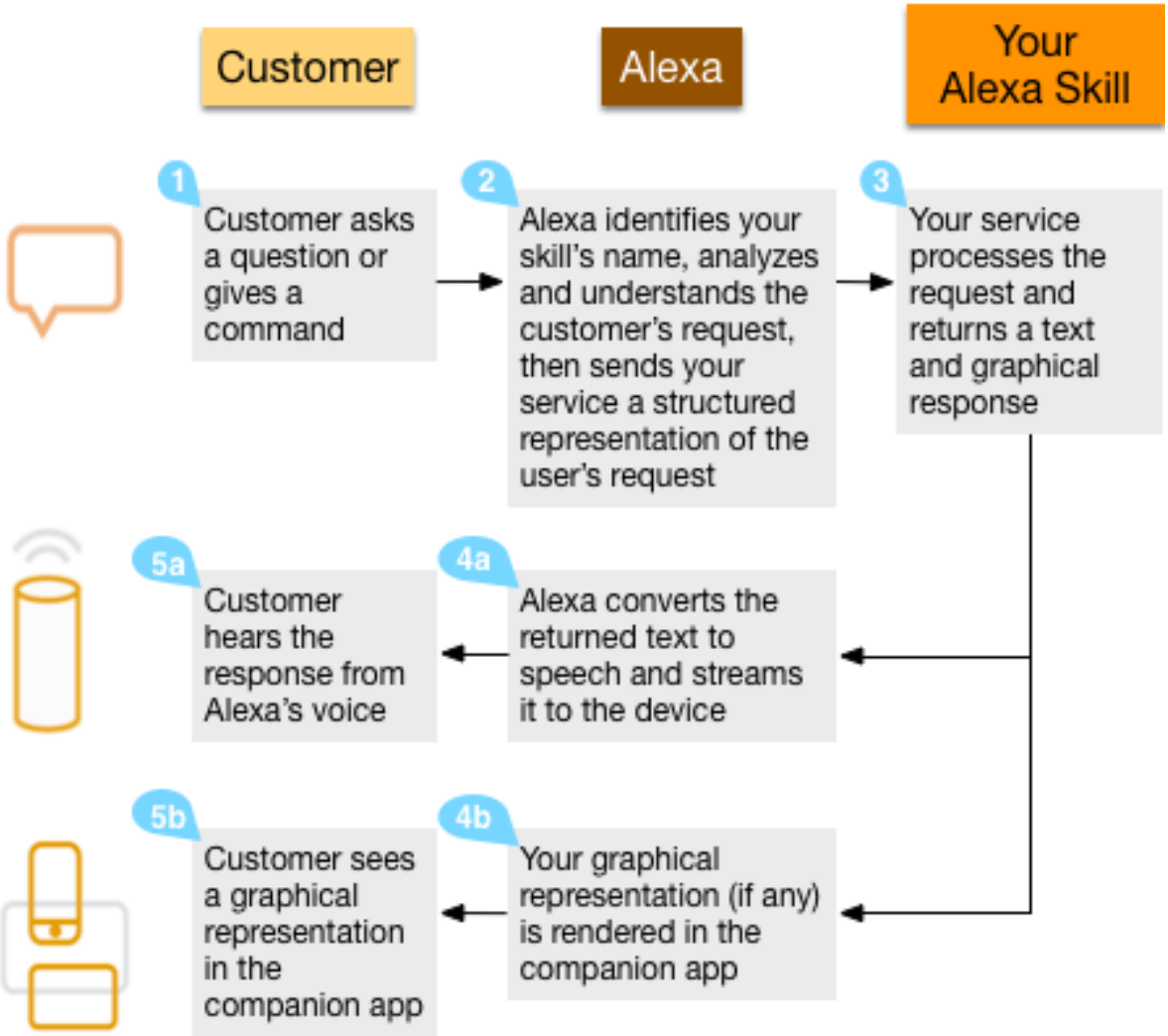
2. Inside we will find two skills, and some additional resources for our Alexa development. Let's have a look at our first skill:

```
$ cd alexa_skill_first
```

3. The above directory contains all you need to create your first Alexa skill. The only file we will focus on at this stage is lambda_function.py, which contains the AWS Lambda code. Start by opening lambda_function.py in an editor (we will be using IDLE for this tutorial) and try to follow the execution flow which starts in the function lambda_handler().

```
$ idle lambda_function.py
```

4. The first look at lambda_function.py can be confusing since the code is dealing with some objects called "event" and "intent". To understand what they are and when is this code executed, let's look at what happens when you speak to Alexa:



(Source: Alexa blogs)

You tell the Echo something like 'Alexa, ask magic skill to say something'. Your "magic skill" (a third-party Alexa skill) will be used to convert the words 'say something' to a particular intent, let's call it the "saySomethingIntent". Alexa will then trigger your AWS Lambda function in the cloud and supply it with the intent object containing the data of your "saySomethingIntent". Your AWS Lambda function (the code in lambda_function.py) will process the intent according to your own logic (defined in lambda_function.py) and send a reply in a within seconds back to Alexa. Ok, this is still a lot to process. It's enough for now if you understand two things:

(a) In your Alexa skill definition (on developer.amazon.com), you will define which sentences (utterances)

correspond to which intents.

(b) When you speak to Alexa, the AWS Lambda function (the Python code you've been looking at) will receive a specific intent that the sentence triggered and will process it.

(you can close the IDLE window now)

5. We can upload the function as is to the cloud to make sure all works as expected. When we run create_lambda_function, it will zip up this directory and send it to the cloud. We will need the "AWS Lambda function ARN" identifier later to link our skill definition to our processing function.

```
$ create_lambda_function --function-name skill_first --dir .
Function successfully created!
AWS Lambda function ARN: arn:aws:lambda:<your_aws_region>:<your_account_id>
↪:function:skill_first
```

6. Our Lambda function is now in the cloud, ready to be executed. Let's make sure we can indeed run it. One neat way to test our AWS Lambda functions is to generate fake Alexa intents and send them to AWS Lambda. This allows us to skip speaking to an Echo and to automate our tests. We generate a specific intent and send it to our AWS Lambda function as if we were the Alexa service, and in return we will get the text reply the Alexa service would receive. For an example intent json object see: event_template.json. The AlexaFirstSkills package allows us to create these test events by modifying selected fields of the intent object, such as the "name", or any variables ("slots"). Let's have a quick look at our test events:

```
$ cat tests/data/lambda_test_data.json
```

7. Our tests generate 3 different types of intents. We can generate the events and send them to AWS Lambda using test_lambda_function commandline script.

```
$ test_lambda_function --function-name skill_first --test-data tests/data/lambda_
↪test_data.json
```

The tests print the intent and slots sent to our AWS Lambda function and the generated reply. Let's look at them step by step:

```
##########################################
Testing function now!
##########################################
Sending Alexa Intent: FakeIntent and slots:{}
Lambda function replied: hmm not sure how to deal with your request
```

The first test just checks what happens if we send Lambda an intent that it doesn't recognize, we chose the arbitrary name "FakeIntent". It is reassuring to by default, the function would send a meaningful reply back to Alexa. A situation like this can easily occur if you add a skill to your ASK definition without adjusting the Lambda skill functionality.

```
Sending Alexa Intent: saySomethingIntent and slots:{}
Lambda function replied: I say whatever I please
```

Our first test of an existing intent shows that Lambda generates a proper reply. Our first task for this coding challenge involves changing this reply to a string of our choice. Once we have a definition of our skill on developer.amazon.com, we can test this skill out by telling Echo, "Alexa, say something" and Alexa will reply: "I say whatever I please". At that point we won't be surprised however, since we already tested and proved here that the intent handling works!

```
Sending Alexa Intent: personalFactIntent and slots:{}
Lambda function returned an error:
```

(continues on next page)

(continued from previous page)

```
  stackTrace:
 /var/task/lambda_function.py
 208
 lambda_handler
 return on_intent(event['request'], event['session'])

 /var/task/lambda_function.py
 170
 on_intent
 return intent_handler(intent, session)

 /var/task/lambda_function.py
 63
 intent_handler
 person_name = intent['slots']['Person']['value']
   errorMessage: 'Person'
   errorType: KeyError

Lambda function didn't reply!!!
```

Wow! This clearly doesn't look good. Let's try to understand what's going on here. We are sending Lambda a personalFactIntent without any slots. Translated into human, we are supposed to ask Alexa: "Alexa, tell me something about a <NAME>". The problem is that the way our lambda function is written currently, it expects a <NAME> for this intent, which should be passed as a slot variable (is this a good idea? could we fix that??). What we are seeing as a result is the error message from Python ("stack trace") that complains the key "Person" not being present in the intent['slots'] dictionary.

While this is a long and ugly message, this example really shows the value of testing your Lambda function. Rather than just writing our Lambda function and then asking Echo a dozen different questions to check whether the skill works, we can define any situation we would like via the tests/data/lambda_test_data.json and get back the exact reply we would get in spoken language from Alexa. Just fyi, in the above case, if we try this with an Echo: "Alexa, tell me something about", we will get back only silence, as the Lambda function isn't handling this case well at all and errors out before passing a reply to Alexa.

Extra credit: There are at least two ways to fix this. a/ We don't assume the "Person" key will always be present for this intent, that means we add a check for it. b/ It would be really good to add a default reply if things go south, just like we did for the case when Lambda receives an unknown intent. Unexpected problems in Python usually trigger an exception, which causes the program to stop immediately. One thing every skill should consider is to catch these exceptions and rather than simply throw an error, send a default reply to the user notifying them that you cannot provide an answer at this time due to an error.

```
Sending Alexa Intent: personalFactIntent and slots:{u'Person': {u'name': u
↪'personalFactIntent', u'value': u'robogals'}}
Lambda function replied: robogals was founded in 2008 in melbourne, australia
```

We don't need to bother with fixing the Lambda error handling right now. We can simply add a "Person" variable into our test and everything works just fine. The above test corresponds to the utterance: "Alexa, tell me something interesting about Robogals".

```
Sending Alexa Intent: whatsCoolIntent and slots:{u'Subject': {u'name': u
↪'whatsCoolIntent', u'value': u'movie'}}
Lambda function replied: Lord of the rings
```

Our last test checks whether the "whatsCoolIntent" works correctly. One cool thing about this intent is that it always returns a slightly different answer, so don't be surprised if you see a different movie on your output.

Equipped with the test_lambda_function script and our test definitions in tests/data/lambda_test_data.json, we

can confidently go into changing the lambda functionality as we have a simple way to check we are on the right track. In addition to testing the entire Lambda function, we can also test individual bits of Python functionality locally before uploading it to AWS Lambda - we will see more about that later. But now, let's define our skill!

8. We can register a third-party Alexa skill using the Alexa Skills Kit (see this step by step guide). We will only create the skill for testing purposes and will not submit it to the store. The skill directory contains data for the interaction model. intent_schema.json contains the intent schema and utterances.txt contain a single sample utterances. You will need to copy both of them in the appropriate fields. You will need the following information:

   - **Skill Information**
     - Skill Type: Custom
     - Application Id: make one up
     - Name: make one up
     - Invocation Name: make one up
   - **Interaction Model**
     - Intent Schema: copy&paste contents of intent_schema.json
     - Sample Utterances: copy&paste contents of utterances.txt
   - **Configuration**
     - Service Endpoint Type: AWS Lambda ARN (Amazon Resource Name)
     - Pick a geographical region that is closest to your target customers: you have to pick the region where you created the AWS Lambda function (if you followed our setup, this region will be eu-west-1, Europe) copy&paste the AWS Lambda function ARN from the create_lambda_function console output
   - **Test**
     - Service Simulator: type in a sentence to simulate speaking to an Alexa device - 'say something' and check out the reply. If you see a reply appearing, you can use an Alexa device such as an Echo, or Dot to test the skill as well. The device needs to be paired with the same account we used for developing this skill.

9. Test the newly created Alexa skill with a physical device - for example with an Echo Dot. You can invoke the individual skills using any of the utterances we used in our ASK definition (see utterances.txt). The device should give the replies currently coded in lambda_function.py as we have seen during our AWS Lambda testing.

   If you don't have an Alexa device, you can easily test your skill through the developer.amazon.com portal - navigate to where you defined your skill and go the the Test section. You can type a sentence and you should receive text reply from Alexa.

What did we learn?

- We have deployed and tested an AWS Lambda function for our skill
- We have seen how speech translates into intents and how intents are processed in Python
- We have defined our Alexa skill using ASK and tested it works

Develop and update skill (blue boxes)

1. The toughest part of Alexa skill creation - the setup, is done and you can give yourself a pet on the back. Now we will turn our attention to modifying the skill. We will do this by first developing functionality and testing it locally. Then updating the AWS Lambda function (uploading the new code to AWS) and testing it using fake Alexa events.

Let's go through the execution flow of our lambda_function.py again, but this time pay special attention to the intent_handler() function. It determines what to do based on the intent name we receive. Our first mission consists of modifying the reply to the "saySomethingIntent" and it will lead us to.. can you guess it?

Yes, it will lead us to the get_something() function. If you have followed the Intro to Python notebook, you should be able to easily modify this function to return 'I can say whatever you want me to say'.

2. At this point, we could simply make the changes we want in lambda_function.py, upload it to the cloud and test it like we did before. But it is much more convenient to test lambda functions as much as we can locally (on our own computer) and only upload them once we are fairly certain our Python functions work. Easy enough, just like we passed fake Alexa events to our AWS Lambda function, we can call our Python functions locally and pass it different parameters. There is a set of tests for you in tests/test_lambda_unit.py.

3. Let's explore the local tests a bit before making any changes to our skill:

```
$ idle tests/test_lambda_unit.py
```

The first test, test_get_something(), simply calls the get_something() function from our lambda skill and checks using an "assert statement" that the function really returns what we would expect. If you haven't changed it, it should return "I say whatever I please".

We can ran all of the tests at once and confirm that our current skill code works as expected (using nose or any other Python test framework):

```
$ nosetests tests/test_lambda_unit.py
----------------------------------------------------------------------
Ran 5 tests in 0.001s
OK
```

4. Now that we know that our intent answering functions are working fine (to the extent we've tested them), we can modify get_something() in our code to return 'I can say whatever you want me to say':

```
$ idle lambda_function.py
```

5. Once we make the change and save the file, it would be interesting to see what happens to our tests - one of them should no longer work since we have just changed the response get_something() is giving us and the test is still expecting to receive 'I say whatever I please'.

```
$ nosetests tests/test_lambda_unit.py
1 of the 5 tests should fail and show the Traceback
```

6. This is easy enough to fix, let's update the correct expected answer in our test_get_something() test and re-run all our local tests. This time, they should all pass again.

```
$ idle tests/test_lambda_unit.py
$ nosetests tests/test_lambda_unit.py
----------------------------------------------------------------------
Ran 5 tests in 0.001s
OK
```

7. Once we are satisfied with the local changes we can confidently update the Lambda in the cloud and test it

```
$ cd alexa_skill_first
$ update_lambda_function --function-name skill_first --dir .
$ test_lambda_function --function-name skill_first --test-data tests/data/lambda_
↪test_data.json
```

8. You can follow steps 4 to 7 to modify the remainder of your code and complete the rest of the challenge.

What did we learn?

- We have learned how to modify and locally test skill functionality

Where to go next?

There is a couple of things you might consider as next steps:

- Add a new intent to this skill - remember you need to add it to both, the ASK definition and to lambda_function.py
- Check out some example Alexa skills for beginners
- Have a look at Amazon's documentation for creating Alexa skills
- Consider skill improvements - improved intent handling (check for dictionary keys before using them, etc.), error handling (catch exceptions, generate default answer), think about security (check skill application id), add more unit tests

3.2.2 Light on! (alexa_iot_skill)

Challenge: Communicate with any Internet capable (IoT) device in your home through Alexa securely (no open ports in your firewall required), instantaneously (1-3 seconds to reach your device) and cheaply (both in terms of $$$ and kW/h). This can include anything from an Arduino to your PC.

Overview:

The goal of this example is to automate as much as possible behind the scenes and allow you to focus on your IoT logic, that means handling of the intents on the device and formulation of the replies. We will use MQTT for communicating messages between our AWS Lambda function and our device, use AWS IoT to keep track of devices and get access to a ton of additional functionality (like rules and notifications). We have selected a Raspberry Pi as our IoT device, but feel free to pick anything that can run Python and can talk to the Internet. There are certain bits and pieces of the setup that you will have to go through though:

Here is what we are going to do:

1. Use a third party Alexa skill (ASK) to route certain Alexa interactions (intents) to your device - using a special invocation
2. Use a AWS Lambda function as a forwarder between Alexa and your device (they are bits of nicely formatted and well-defined JSON)
3. You will create a "thing" on AWS IoT to represent your IoT device
4. The Python Lambda function will use MQTT (add link) to securely communicate with your device using AWS IoT - no need to change it
5. You will use a Python client on your home device to listen for messages from our Lambda function and parse the forwarded Alexa intents
6. Everything was building up to this point, since now you can handle the Alexa intent on your device, and the best bit is that you can immediately send a reply, which will be forwarded back to Alexa and magic! The Echo will reply you.

3.3 Skillathon - a hackathon for Alexa Skills

3.3.1 PC Setup for the skillathon

1. Create an Amazon developer account and an AWS account for the event. Follow the steps outlined in the 'Setup and requirements' section of the README.
2. Check that you have a working internet connection

```
$ ping google.com
```

3. Install IDE and dependencies on the PC

```
$ sudo apt-get update && sudo apt-get upgrade
$ sudo apt-get install python3.6 python3-pip python3-dev idle
$ sudo pip3 install --upgrade pip
$ sudo pip3 install firstalexaskills jupyter nose
$ aws configure --profile lambdaUser # use the API credentials from AWS
test the setup:
$ aws lambda list-functions --profile lambdaUser # there should be a list␣
↪(possibly empty) of functions
$ unpack_example_skills
$ jupyter notebook ~/example_skills/python_intro/python_intro.ipynb # does the␣
↪notebook work?
$ rm -rv example_skills
```

4. Depending on the experience level of the participants, you might decide to define the Alexa skills in advance, in order to save time and focus on coding, rather than filling out forms. Each ASK Alexa skill needs to have a single service endpoint. Therefore each team developing a Lambda service will require an Alexa skill of their own.

   Note: It can be practical to create a list of team names based on the expected number of attendees before the event. Define a skill in the development portal and use the team name as the name of the skill. That way, during the main event, the participants don't have to interact with the developer.amazon.com portal at all. Each team can then invoke the skill saying: "Alexa, ask team blue to tell me something". Please follow Alexa's advice on suitable words for skill invocation when using this! Similarly, each team will need to use their team name as part of the AWS Lambda function name they will create.

5. Finally, pair your Alexa devices (Echo, Echo Dot, Echo Show) with the Amazon developer account you've created and test the whole setup.

3.3.2 Things to consider

Problem:

- Echo's speech recognition accuracy decreases in proportion to the distance of the speaker and amount of noise in the room. A large room full of people screaming gentle conversation invitations to multiple Alexa devices at once can result into a painful experience.

Workaroud:

- Rely more on software testing and limit the Echo interactions for the final "ahaaa" moments rather than continuous testing
- Split teams into smaller rooms

Problem:

- The Alexa devices require wifi to communicate with Alexa and many wifi devices (smartphones etc.) trying to connect concurrently to the same network can make the interactions more sluggish, or outright stall them.

Workaroud:

- Simples, dedicate a wifi for the Alexa devices and consider not using other wifi access points on the premises to avoid signal interference.

3.3.3 Agenda for the skillathon

In order for the participants to have fun and make the most of their creative ideas, they will need three things: Python basics, a conceptual idea of how Alexa skills work and tools to code and test the skills.

1. Intro to Alexa and third-party Alexa skills (ASK)
2. Intro to Python (note: consider to split the participants into groups for people with and without coding experience). People that understand the concept of variables and functions should work through the Intro to Python notebook. People that don't know those concepts or haven't programmed before should first get an explanation of what programs are, what is execution flow and how variables and functions work. (see the 15 minute Intro to Python for non-programmers)

```
$ cd ~
$ unpack_example_skills # will create a directory alexa_skills with skills and␣
↪resources
$ jupyter notebook ~/example_skills/python_intro/python_intro.ipynb
```

3. Quick demo of a working Alexa skill
4. Split into teams and choose a team name
5. Work through the AlexaFirstSkills tutorial

3.4 Python intro

If it just so happens that you are interested in Alexa, but don't have much Python experience, don't worry. There isn't that much to know about Python to be able to do an awfull lot.

3.4.1 the 1 minute intro to Python for programmers

Python has variable types such as ints, floats, strings, bytes, lists, dicts and others. Python requires you to use offsets (tabs, or 4-spaces, not mixed though!) for scoping the code.

```
>>> # this is a comment              # which doesn't get executed
>>> a = [1, 2, 3]                   # creates a list
>>> a = a + [4, 5]                  # appends 4 and 5 to the list
>>> print(a[0])                     # prints the first element of a
>>> d = {"my_key": "my_value"}      # creates a dict (hash map)
>>> d["another_key"] = 42           # adds another entry into d
>>> print(d["my_key"])              # prints "my_value"
>>> def polite_function(x):         # defines a function returning a string "hi"
>>>     a = "hi"                    # for a call: polite_function("hello")
>>>     if x == "hello"             # compares the value of x against "hello"
>>>         return a                # and if True, greets you back
>>>     else:
>>>         return "Pffff!"
```

Ready for a puzzle? Try and define a new list "my_list" a new dictionary "my_dict" and new function "my_fun" so that:

```
>>> print(my_list[2] + " is " + my_dict["what"] + " " + str(my_fun() + 2) + "! :)") #␣
↪prints "Python is number one! :)"
```

Now go and change some Alexa skills!

3.4.2 the 15 minute Intro to Python for non-programmers

One beautiful thing about programming is that it's very reliable! The computer isn't like a puppy, it will always do what you tell it to, not just maybe, and not just sometimes. Now, if that would be true literally, you wouldn't have to read this introduction and you could just have chat with it about what is it you want. Unfortunately, while Alexa is getting smarter, you still have to program your computer to do whatever you want it to do (as of 2017). But in a sense programming is like a chat. You give instructions, and observe what the computer does with them. If you don't like what you see, you modify your instructions, until you are satisfied. These "instructions" are computer code and writing them is what people call "programming". Think of them as cooking recipe that you give the computer. It takes your recipe and provided it understands all the instructions, it executes each instruction in the order you've given it, from start to finish. And that's really it. To start programming in Python you need to know only two things: there are variables (of different types) and there are functions. You can think of "variables" in computer code as the same "variables" you learned about in your mathematics class.

You can give it a try by creating a file (recipe) by creating a file called "first_script.py" in IDLE. Now write something like:

```
>>> a = 1
>>> print(a)
>>> a = 2
>>> print(a)
>>> b = 3
>>> a = b
>>> print(a)
```

Save the file and execute it in the command line by running:

```
$ python first_script.py
```

What is going on here? We create a variable called "a" and set it to 1. Then we print the value of "a". We can also create a variable "b" and then assign "a" to have the value of "b". So far so good. Right? Now comes a tricky bit:

```
>>> a = 1
>>> a = a + 1
>>> print(a)
```

Ok. So if you replace the variable "a" with its value in the second line you get "1 = 1 + 1" which does not make much sense. Here is where the difference to "variables" you know from school becomes apparent. In mathematics the equal sign "=" means "is equal", while in programming it means "make it equal". So the first line above reads: "Take the value of "a" and make it equal to 1." and the second line reads: "Take the value of "a" and make it equal to the sum of "a" (which is currently 1, set in the previous line) and 1." Hence we are setting "a" to be equal to 2 and then printing it in the last line.

The last bit you need to understand are functions. Fortunately they are super simple to wrap your head around. Think of them as mini-programs (or mini-cooking recipes). You can put them in whichever place you want and then you can execute them whenever you like. Let's try to think of a recipe for making pancakes. We will be adding ingredients and mixing them together. We will create a "mixing" function that we will call at different times:

```
>>> def mixing():
>>>     print("Mixing ingredients now...")
>>>     print("Now they are well mixed again!")
>>> a = "flour"
>>> print("adding " + a)
>>> b = "milk"
>>> print("adding " + b)
>>> mixing()
>>> c = "eggs"
>>> print("adding " + c)
>>> mixing()
```

See how useful the "mixing" function is? We defined it at the very top and then called it whenever we needed it by running "mixing()". This was a very simple function, it didn't do much other than print the same thing, over and over. Functions are great because we can give them our variables and receive a variable in return. Let's try to do that:

```
>>> def mixing(a, b, c):
>>>     return "We've made great pancake dough using: " + a + ", " + b + " and " + c␣
↪+ " :)"
>>> a = "flour"
>>> print("adding " + a)
>>> b = "milk"
>>> print("adding " + b)
>>> c = "eggs"
>>> print("adding " + c)
>>> result = mixing(a, b, c)
>>> print(result)
```

If you have made it this far, you should be very proud! You've just seen what is the essence of writing a program. There is much more to learn but most of that deals with what are the right instructions for getting the computer to do what you want. Once you have truly understood why writing code is like writing recipes, you can always google how to make it do what you want. If there are bits you are unclear about, just go back and make small modifications to see how the computer reacts.

I guess it's also time to let you in on two secrets:

Firstly, this guide was never about Python - that's what "the 1 minute intro to Python" is all about. It was a bit about cooking and recipes but mostly about what really happens between you and the computer when you code.

And secondly, now that you've learned how coding works, you should learn what to code. Go and fearlessly explore "the 1 minute intro to Python"!

3.5 FirstAlexaSkills package

3.5.1 Modules

3.5.2 FirstAlexaSkills.lambda_utils module

FirstAlexaSkills.lambda_utils.**create_lambda**(*client*, *function_name*, *dir_path*, *role*)

FirstAlexaSkills.lambda_utils.**create_zipfile**(*dir_path*)

FirstAlexaSkills.lambda_utils.**generate_testevents**(*test_data*, *event_template*)

FirstAlexaSkills.lambda_utils.**get_account_id**(*profile_name*, *region=None*)

FirstAlexaSkills.lambda_utils.**get_eventtemplate_fn**()

FirstAlexaSkills.lambda_utils.**get_execrole_arn**(*execrole_name*, *profile_name*, *region=None*)

FirstAlexaSkills.lambda_utils.**get_skills_archive_fn**()

FirstAlexaSkills.lambda_utils.**list_lambda_functions**(*client*)

FirstAlexaSkills.lambda_utils.**print_nested**(*obj*, *nested_level=0*, *output=<open file '<stdout>', mode 'w'>*)

FirstAlexaSkills.lambda_utils.**replace_nested_dict_val**(*orig_dict*, *k*, *v*)

FirstAlexaSkills.lambda_utils.**run_all_tests**(*client*, *function_name*, *alexa_event_data*, *event_template*)

FirstAlexaSkills.lambda_utils.**test_lambda**(*client*, *function_name*, *event*)

FirstAlexaSkills.lambda_utils.**update_lambda**(*client*, *function_name*, *dir_path*)

3.5.3 Module contents

3.6 Develop

This assumes that you went through the installation and setup notes. To develop the example skills and this package will require working AWS credentials.

Caution: Some of the tests in this project require AWS credentials as they perform end-to-end testing including cloud upload. Specifically tests in tests/scripts (test the commandline tools) tests/functional (end-to-end tests of skills functionality). During development, tests should only be run using `$ python setup.py test -a '-e cloud'`. Travis CI will only runs local tests using `$ python setup.py test`. That means that Travis does not guarantee that your commit won't break the build and the resposnsibility for cloud tests lies with you!!

Here are a few steps to get you started:

```
$ git clone https://github.com/means-to-meaning/FirstAlexaSkills
$ mkvirtualenv python36fasdev --no-site-packages
$ activate python36fas
$ cd FirstAlexaSkills
$ python setup.py develop # or pip install -e .
__________ do some work __________
$ python setup.py test -a '-e cloud' # this will run all the tests including end-end
↪cloud stuff - requires valid IAM credentials for 'lambdaUser'
$ python setup.py test -a '-e docs' # test documentation
$ python setup.py test # run the local tests
$ restview docs/develop.rst # view docs in browser while modifying them
# $ sphinx-apidoc -F -o docs FirstAlexaSkills # generate sphinx autodoc documentation,
↪ don't run this every time as it tends to add all modules including the tests
$ python setup.py develop --uninstall # or pip uninstall FirstAlexaSkills
$ python setup.py test
$ deactivate
# update the commit and push!
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index