Tutorial for Pepper

Pepper is a semi-humanoid robot manufactured by SoftBank Robotics. Pepper is equipped with a set of sensors, speakers and cameras enabling Pepper to interact with its environment. Pepper can listen and interpret what you say, detect if you’re happy or sad, walk and move its arms among other things. By creating applications, you can decide how Pepper should interact with its environment. This will be a step-by-step guide for how you can create your own applications.

# What do you need to get going?

You need a set of programs in order to create an application for Pepper. Here’s a list of them. More detailed descriptions along with download links is presented below.

* Python 2.7 32 bit
* Python IDE (Optional but recommended)
* Softbank community account
* Python SDK
* Choregraphe
* Putty
* Git (Optional but recommended)

## Python 2.7 32 bit

You can write code in C++ and Python. We recommend Python as a lot of the available modules for Pepper are written in Python and can be directly copied. Unfortunately, Pepper is not compatible with Python 3.x or Python 64 bit. This is the reason why Python 2.7 32 bit is used.

1. Download the latest version of Python 2.7 from [here](https://www.python.org/downloads/).
2. Install Python.
3. Add installation path to your PATH in “*environmental variables*”. Default path of installation is “C:\Python27”.

## Python IDE

Writing code for Python can be done in any text editor. It’s however easier to use an IDE. An IDE aids in code development with autocorrection, some bug detection and automatic imports and package installations. We use PyCharm and recommend you to do the same.

1. Download PyCharm Community from [here](https://www.jetbrains.com/pycharm/download/#section=windows).
2. Install PyCharm
3. Go to File->Settings->Project interpreter and select your installed version of Python 2.7 as interpreter.

## Softbank account

In order for you to get access to the Python SDK (Software development kit) of Pepper you need a Softbank account. Create the account [here](https://accounts.softbankrobotics.com/#/create). This account might also give you access to the [forum](https://community.ald.softbankrobotics.com/) where you can ask questions. Contact Erik Sjödin otherwise and he might be able to help you. You can try and contact the [support](https://www.softbankrobotics.com/emea/en/support) if you can’t find the answer on the forum or on google. Best practice is to submit your issue, the robot serial number found near the robot’s stop button under the soft cover at the base of the neck (it is the bottom two lines starting with AP.) and the SDK version number.

## Python SDK

You need a Software development kit (SDK) to connect to Pepper via Python. The SDK can be downloaded as either an executable file or as binaries which you extract to a folder. You need a Softbank account in order for you to access the SDK download page.

### Installing SDK via executable file

1. Go to the community [download page](https://community.ald.softbankrobotics.com/en/resources/software/language/en-gb)
2. Find a Python SDK setup file and not a binaries file. You might have to tick the SDK box to the left and scroll down to older versions
3. Download the file and install it
4. Test your installation by importing the naoqi and qi package in PyCharm as  
   import qi

import naoqi

If this works you’ve successfully installed the SDK

### Installing SDK via binaries

1. Go to the community [download page](https://community.ald.softbankrobotics.com/en/resources/software/language/en-gb)
2. Find a Python SDK binaries
3. Download the file to some location
4. Add the following environmental variable in your system properties:  
   name: PYTHONPATH  
   value: path/to/python-SDK/lib
5. Download and install “Microsoft Visual C++ 2010 Redistributable Package (x86)”
6. Test your installation by importing the naoqi and qi package in PyCharm. PyCharm will not detect the packages, so in order to test if everything is installed correctly you have to run the script with the imports. If you get no import error you’ve successfully installed the SDK.  
   import qi

import naoqi

## Choregraphe

Choregraphe is a GUI based development tool for Pepper and other Softbank robotics robots. Choregraphe is similar to Simulink and is a good start for getting to know Pepper and try some simple stuff. You create an application for Pepper by connecting boxes consisting of speech, dialog, movement etc. There are even boxes for Python code where you can add your own code and perform stuff such as fetching data from an API or performing some calculations. The flow of your Application dictates what Pepper will do. An advantage with Chorehraphe is that you can simulate your code on a virtual robot. The disadvantages of Choregraphe is that it's a buggy software and that it limits your flexibility in development. You can download and install the software [here](https://developer.softbankrobotics.com/pepper-naoqi-25/downloads/pepper-naoqi-25-downloads-windows).

### Choregraphe tips

* If you have trouble connecting to the virtual robot or if it doesn’t run your application, simply disconnect using *ctrl+shift+D* and go to *Connection->Connect to Virtual robot*.
* You can also connect to your physical robot through Choregraphe. To do this, go to *Connection->Connect to…* and enter the IP address and port of Pepper. When clicking select, Choregraphe will try to connect to the robot. In the header of Choregraphe you can see the status (*connecting…*, *connected to...* etc.). Make sure that you’re on the same WiFi as Pepper when doing this, otherwise it will fail.
* Once connected to Pepper, you can transfer files to him, e.g. dialog files (\*.top). This is done by going to *Connection->Advanced->File Transfer...* and clicking the upload or download buttons.

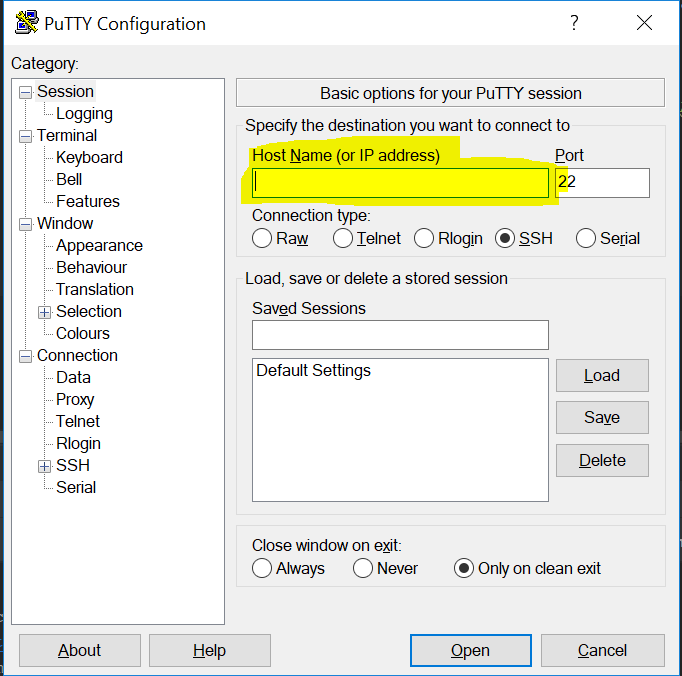
## Putty

You need some SSH client in order to send files and investigate the file system of Pepper. We used Putty, download [here](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html).

### Send files to Pepper using SCP

1. Once Putty has been installed you should be able to use the scp command in the command prompt (cmd) of Windows.
2. Open cmd
3. Use the following command to transfer the file test.py to Peppers /home/nao folder
   1. pscp -scp test.py nao@<IP-address>:/home/nao
4. If you want to transfer folders, you can add the flag -r
   1. pscp -scp -r test-folder nao@<IP-address>:/home/nao

### See Peppers file system using SSH

1. Open Putty
2. Fill in “nao@<IP-address>” where <IP-address> is the IP of Pepper, in the box *Host Name (or IP address)*. If you don’t know the IP address of Pepper, press the button on its stomach. Make sure both of you are connected to the same WiFi.   
   
3. Click open. Fill in *ericsson* as password. Use Linux batch commands, for example *ls* to list files in current directory and *cd* to move between directories . The applications are located in *.local/share/PackageManager/apps*.

## Git

Git is a version control software and is recommended for working with Pepper. Not only do you secure your own code but you can easily download our example code and other examples from GitHub. Download the latest version of Git from [here](https://git-scm.com/downloads).

### Git GUI

You can either work with Git from the *Git bash* or from a GUI. We prefer [Git Extension](https://git-extensions-documentation.readthedocs.io/en/latest/git_extensions.html) (download [here](https://sourceforge.net/projects/gitextensions/)).

### Clone a repository

It’s simple to download a GitHub project using the built in clone function. Either use the GitBash with the command *git clone link-to-repository-on-github* or use Git Extension. [Here’s our repository](https://github.com/TheoBerglin/SummerWithPepper).

# Pepper - basics

Some basic stuff that’s good to know when working with Pepper is listed below.

## Getting Peppers IP

You can obtain Pepper’s IP address by pressing once on the button located on his stomach under the tablet. Note that his IP might change when different WiFi’s are used.

## Robot web page

If you type Pepper’s IP address in the web browser (when being connected to the same WiFi), enter the username *nao* and password *ericsson*, you will access his *Robot web page*. Here, you can adjust the speech volume, change which WiFi to connect to and reboot him, amongst other things.

## Connecting Pepper to WiFi

The easiest way to connect Pepper to a new WiFi is to connect to it while being connected to another one. The reason being, that you can use the Robot web page to choose which network to connect to. If Pepper is not connected to WiFi, you can connect him to internet using an ethernet cable (don’t know where to plug it in though?) or by saying *run/launch/start settings* which will launch the settings on the tablet. Here, you can choose which WiFi to connect to.

## Autonomous mode on/off

You can switch on or off Pepper’s autonomous mode in the following ways:

* Double-click Pepper’s belly button
* Click the heart in Choregraphe when connected to him

# Example application development

Developer guide with tutorials and API: <http://doc.aldebaran.com/2-5/index_dev_guide.html>

Please alter between the tutorial and our developed code to better understand the concepts described below.

## qi-Framework

We’ve used the qi framework when implementing our modules, which we strongly recommend, since it uses the NAOqi API but has made it easier and more intuitive to write your code.

### Applications, sessions and services

To initialize the qi framework, you must create a qi.Application() that connects to Pepper, which in our code is done in *main.py*. Once the connection is set up, you can get the session which in turn can fetch various services that you can utilize when writing your program. Some useful ones that we’ve used are listed below.

### Existing useful modules

There exists a set of useful modules in the NAOqi API that you can use to build your application. Here are some of the most important ones with a brief description of how to use them and some of the pitfalls we have encountered when working with them.

#### Memory

The memory module is used to store key information on Pepper. The service which you subscribe to is called [ALMemory](http://doc.aldebaran.com/2-5/naoqi/core/almemory.html). As stated, the memory service lets you store information in variables. The following types are supported as variables

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **type** | integer | boolean | float | list | string |

You can set a value of a variable in the code or in dialogs. The value of a variable is set by inserting data to that variable as *memory.insertData(“****nameOfVariable****”,* ***value****)*. You can access the value of a variable anywhere (that is dialog or methods of your Python code) as long as you continue to be connected with Pepper. The following snippet retrieves the value of a variable: *memory.getData(“****nameOfVariable****”)*.

Another useful method of the memory module is the ability to raise and handle events. You can subscribe to the change of a variable and when the value of the variable is changed execute a specified method. The following code snippet subscribes to the event that the value of the *new\_view* variable is set. When this happens, the method for displaying a new view, *display\_on\_tablet*,will be launched. The value that the variable is set to is also forwarded to the callback function (in this case display\_on\_tablet).

self.new\_view\_subscriber = self.memory.subscriber("new\_view")

self.new\_view\_id = self.new\_view\_subscriber.signal.connect(self.display\_on\_tablet)

Note that you can raise events in the python code, in the dialogs in the \*.top files but also in JavaScript code upon button clicks etc.

#### Text to speech

The text to speech module allows the robot to speak. This module is simple and fun to use. Simply subscribe to the service [*ALTextToSpeech*](http://doc.aldebaran.com/2-5/naoqi/audio/altexttospeech.html) and use the method *say()* to output audio of text. The following code snippet does this:

self.tts = session.service("ALTextToSpeech")

self.tts.say("Hello. What can I help you with?")

#### Dialog

Text to speech is great when you want Pepper to say something. But if you want to interact with Pepper using speech it’s best to use the dialog module. This is done by subscribing to the service [*ALDialog*](http://doc.aldebaran.com/2-5/naoqi/interaction/dialog/aldialog.html?highlight=aldialog) with the following [API](http://doc.aldebaran.com/2-5/naoqi/interaction/dialog/aldialog-api.html). You load a *topic* (.top) containing the conversation you want to have. The topic contains the dialog and is governed by two types of rules: *User rules* and *Proposal rules*. User rules links specific *user input* to possible *robot output*. An example of this is:

u: (Hello Pepper how are you today) Hello human, I am fine thank you and you?

If the user says “Hello Pepper how are you today”, Pepper will respond “Hello human, I am fine thank you and you?”. You can also add triggers to event instead of text as “*e:****nameOfEvent***”. These events could for example be raised in some JavaScript code that handles a click on a logo (like we use in the HumanGreeter). You can also add multiple user inputs to the same robot output. This is done as:

u: ([“user input 1” “user input 2” e:nameOfEvent]) Hello human, I am fine thank you and you?

A Proposal rule triggers a specific robot output without any user output beforehand. Pepper will take the initiative and say something which the user can respond to. An example of this is the following snippet:

**proposal**: Have you seen that guy on the TV yesterday?

u1: (yes) He was crazy, no?

u1: (no) Really, I need to tell you.

As previously mentioned you group these rules in a topic to create an advanced conversation/dialog. You can set variable values within the dialog and you can specify the language in the topic. An example of a more advanced topic can be found [here](http://doc.aldebaran.com/2-5/naoqi/interaction/dialog/aldialog_tuto.html) and another one is also available in our GitHub repository. You have to upload the topic to Pepper via scp or from Choregraphe to be able to load it.

#### Motion

The motion service enables you to move Pepper. Simply subscribe to the service [*ALMotion*](http://doc.aldebaran.com/2-5/naoqi/motion/index.html) and get going. The motion service can for example move Peppers arms or move him around. We have used it to set Pepper in an upright position, rotate and move its arms. The motion API is big and contains multiple functions, please see the Aldebaran documentation for more information.

#### Tablet

Pepper’s tablet can be used to show web pages, play videos etc. To work with the tablet you need to get the service [*ALTabletService*](http://doc.aldebaran.com/2-5/naoqi/core/altabletservice.html). You can then use this service to do various things, specified in the API. Amongst other things you can display .html pages, which is what we’ve mainly used it for. To display pages they need to be located in an html folder somewhere in the apps folder on Pepper: */home/nao/.local/share/PackageManager/apps/*. In python, you can then fetch the address to this folder by getting the robot IP using the .robotIp() method of the *ALTabletService* and using it as follows: *http://<robotIp>/apps*. We’ve created folders for each one of our applications, then created corresponding html folders within each one, where we’ve placed the relevant .html files, e.g. */home/nao/.local/share/PackageManager/apps/humangreeter/html/example-file.html*. We then display this file by calling .showWebview() with the following path: *http://<robotIp>/apps/humangreeter/example-file.html.* Note that you don’t enter the html folder in the path. Our method for displaying web pages looks as follows:

def display\_on\_tablet(self, full\_file\_name):

self.tablet.enableWifi()

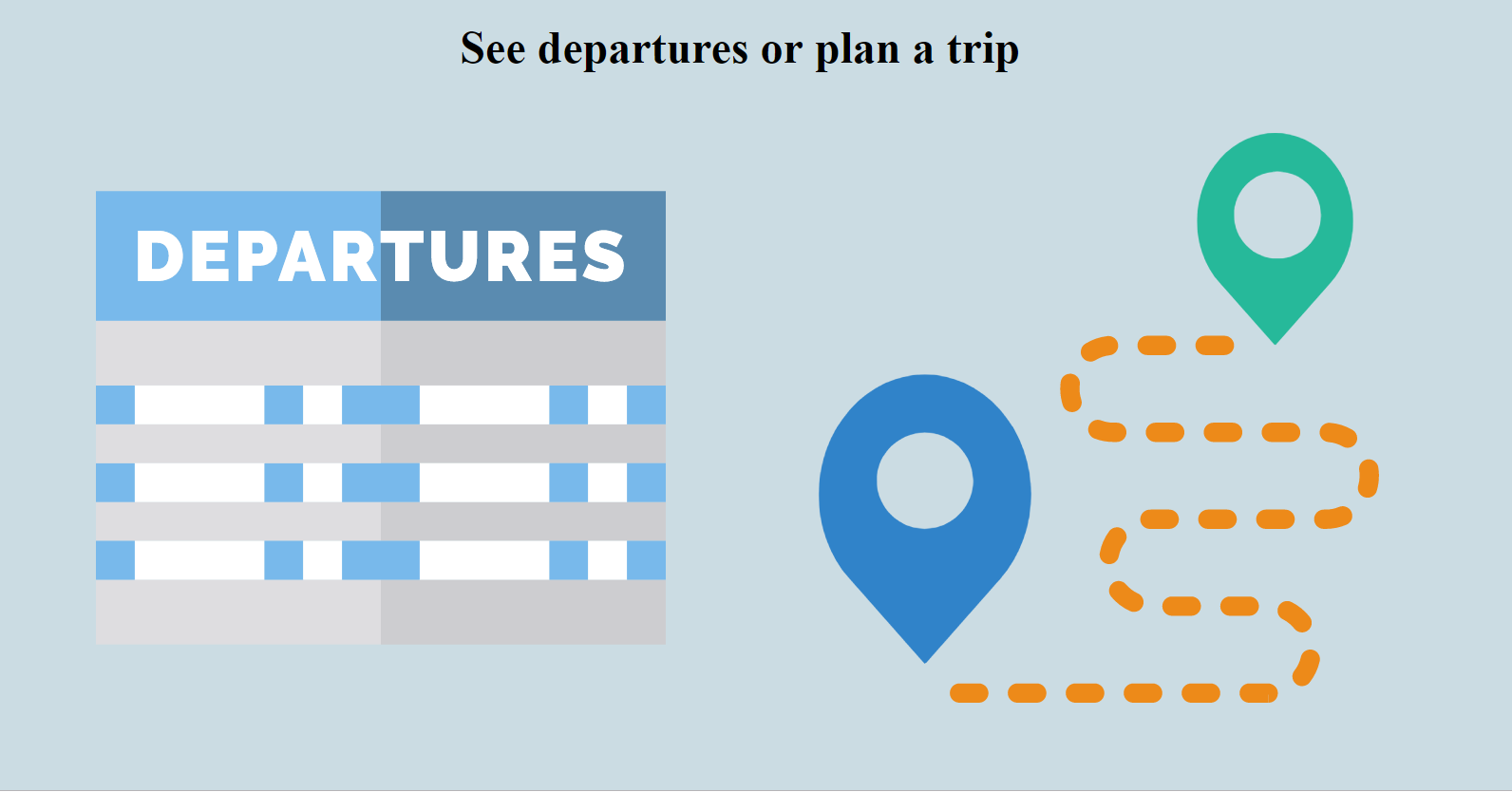
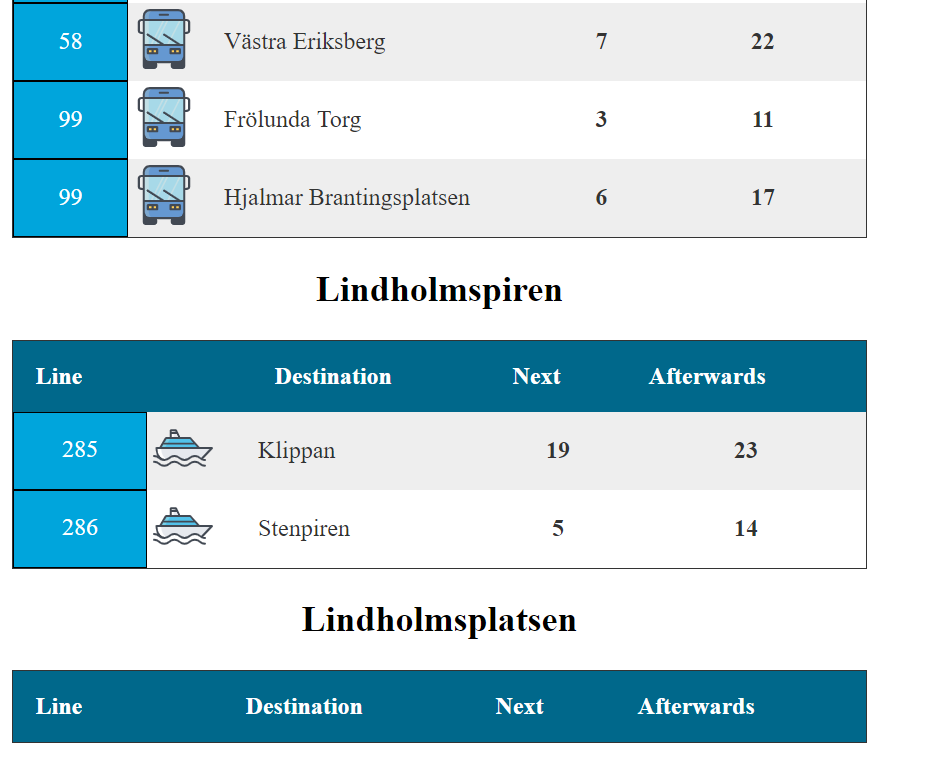
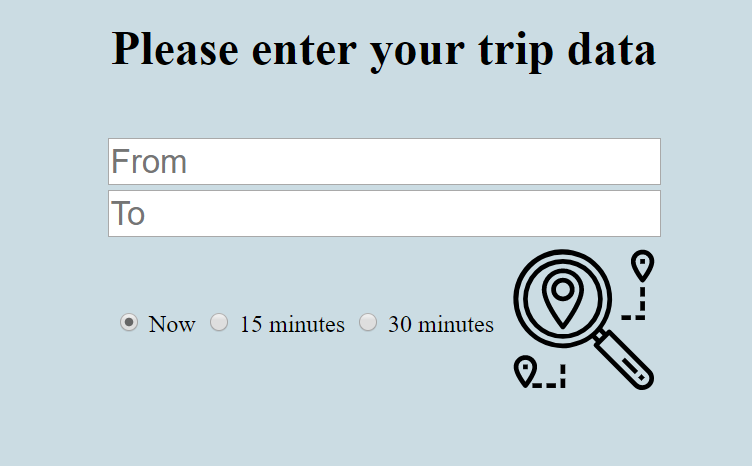
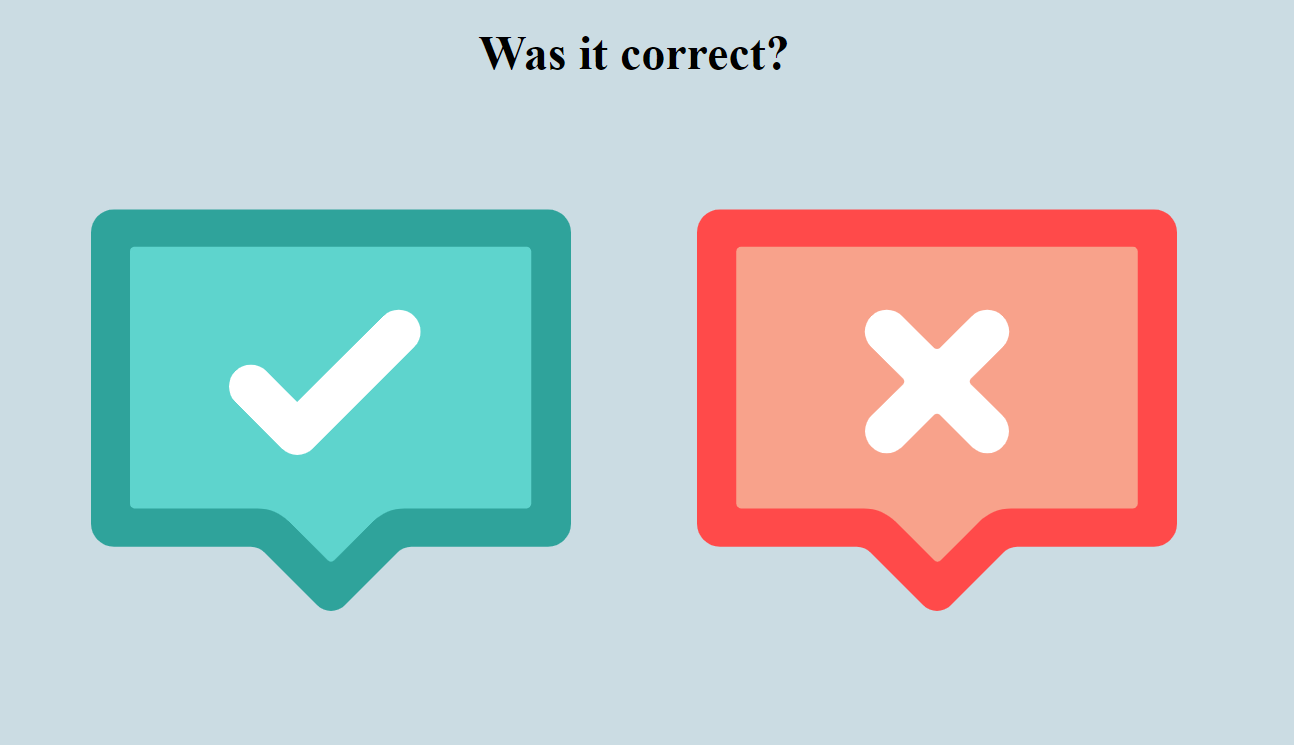
ip = self.tablet.robotIp()

remote\_path = 'http://%s/apps/%s/%s' % (ip, self.folder\_name, full\_file\_name)

self.tablet.showWebview(remote\_path)

## Creating a public transportation planner

We have created a simple public transportation planner for the Gothenburg area. Västtrafik is the provider of public transportation and we’ve used their free REST API. Here’s a quick walkthrough of the application. A brief explanation of how we connect with the API and how the software is built is presented afterwards.

1. Initial screen on Peppers tablet. The application has two possibilities, it can either show the next departures from nearby stops or it can plan a trip. You can start either of the two by clicking the buttons or by speech. Saying “Rides” starts the next ride application and saying “Trip” starts the trip planning.  
   
2. We have two possibilities for the next step of the application.
   1. We display the next rides.  
      
   2. Or we plan a trip. Currently planning a trip doesn’t have its own tablet-view. Instead Pepper continues to show the first tablet view.
      1. Pepper will ask where you want to go and where you want to go from. Simply say the station names. If you have trouble with Peppers speech recognition you can say “text” and be prompted to a text input view.  
         1. Text input page, press go when finished.  
            
      2. Pepper will ask if it understood the specified trip correctly. He will read out the specified trip and the following tablet view will be shown. You can either press the buttons or say “yes”/”no”. Pepper will ask for text input instead if you press/say “no”.  
         
      3. Pepper will fetch the trip data from the Västtrafik API and display the following clickable page. 
3. Pepper will ask if you are satisfied when displaying the final result. If you say “no” Pepper will keep on showing the result. If you say “yes” Pepper will consider that you have received the help you wanted and exit the module and go back to the HumanGreeeter application. *Note* that you can say “exit” in any stage of the application and Pepper will exit the module.

### Connecting with Västtrafik API

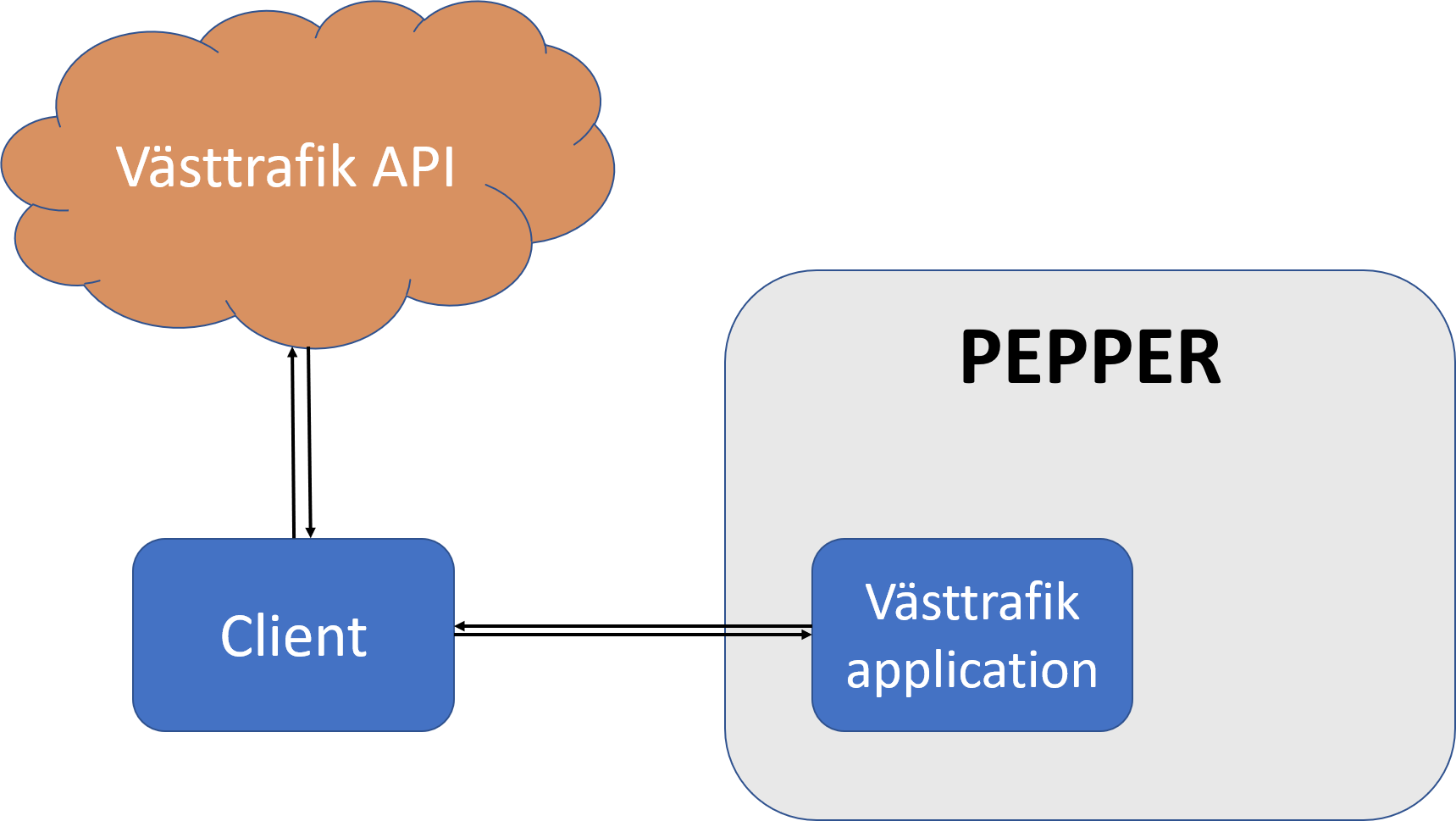
Our example application was a public transportation trip-planner for the Gothenburg area. We connected with the free REST Api of Västtrafik, this is a simple tutorial on how we did that. You can modify our code to connect with other REST Apis’. The API is free, but you need an account and a key for the specific Västtrafik API of interest. We’ve used the API called Reseplaneraren v2.

#### What is a REST API?

REST stands for REpresentational State Transfer. A REST API is an API built on HTTP functions, a well known structure. There are advantages and disadvantages of REST API, but they are not important for this project. What is important is that you communicate with the API through customized web-links depending on what data you want.

#### How we connect with the API

The basics of the connection with the API is that we need to be able to visit websites from Python. We have done this using the python package *request*. You need additional input to the initialization of the request object in order for request to work when you’re behind a firewall/proxy. We haven’t bothered to fix this, but if you’re interested you can read more [here](https://stackoverflow.com/questions/8287628/proxies-with-python-requests-module). Below is an image describing the structure of how we connect with the API.

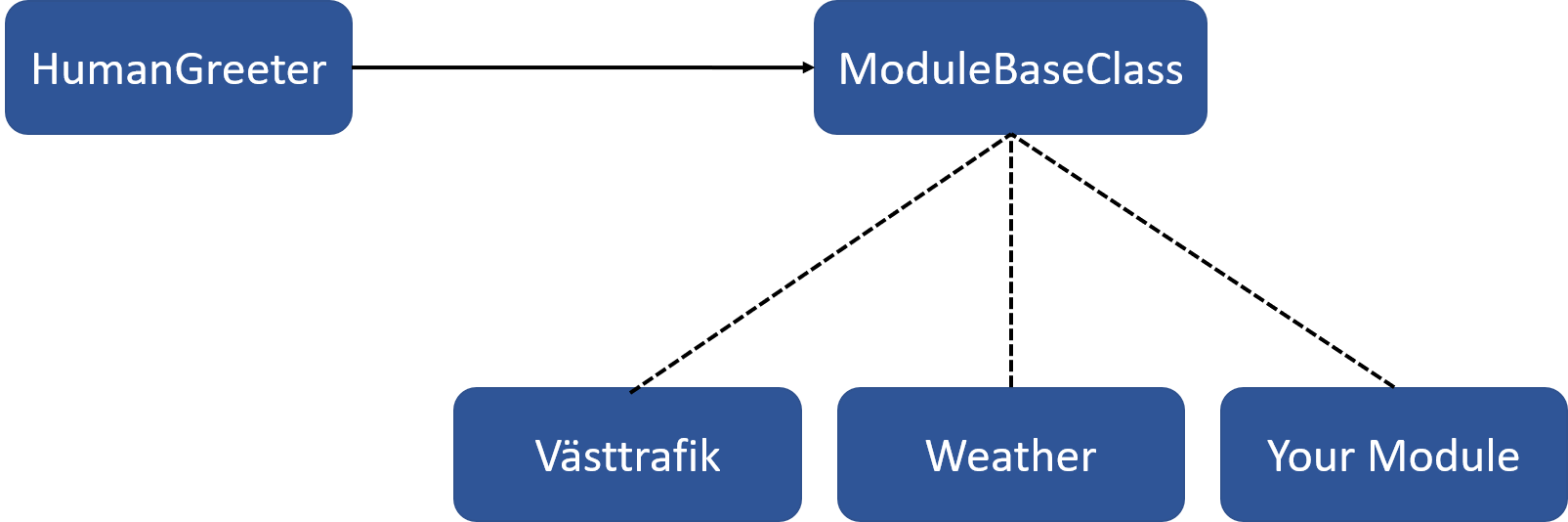


As seen by the image we have a Västtrafik application on Pepper. This application connects with a client, *VasttrafikAPI.py*. The client handles the basic communication with the Västtrafik API. The *client* handles the key for the API and creates the customized request urls. The structure of having a client in the middle of your application and the API is preferable as it limits the dependencies. It also simplifies the process of reusing the API for another application and for merging applications, for example merging the Västtrafik application with a weather application. An important part of the client is its loading of API keys. These are added in the file *api\_settings.yml*.

## Add your own module

We’ve built a module based software consisting of a greeting application from which you start other applications. These applications, or modules, can be the trip planner described above or any other application you can think of. Adding a new application is simple, here’s a tutorial on how you do that.

To do this you need a brief idea of how our software/framework is built. The main application launches an application called *HumanGreeter* (HG). The HG application makes Pepper search for a face of a person which it can interact with. When Pepper finds a face, HG will ask what the person might need help with. Both speech input and click on Peppers tablet will launch the applications. Say “Vasttrafik” or “Vesttrafik” in an english-like accent and HG will launch the trip planner application. The Vasttrafik application is an instance of a *ModuleBaseClass* (MBC) in *module.py*, which all future modules should be as well. MBC initializes some necessary services such as memory and dialog, handles file transfers to Pepper, closing of modules and displaying websites on Pepper. See the below image for an overview of the structure.



MBC contains a set of abstract methods and some non-abstract methods. The abstract methods are both with pre-written code and completely empty. Both types need to be implemented in the subclass, and can choose to inherit the pre-written code in MBC. The non-abstract methods are of the type that should be consistent throughout the different modules and thus, don’t need to be inherited. The various methods are presented below:

*@abstractmethod* **\_\_init\_\_(self, app, name, pepper\_ip, folder\_name):**

Object initialization. Sets some useful variables of the object. Sets the speech recognition to Swedish and fetches some useful services, such as ALDialog, ALMemory etc. Subscribes to the exit signal (that is raised when a user says exit/quit).

*@abstractmethod* **display\_on\_tablet(self, full\_file\_name):**

Displays the file specified by the parameter *full\_file\_name* on Pepper’s tablet. The file needs to be located in the html folder of the relevant module.

*@abstractmethod* **run(self):**

Empty abstract method that needs to be implemented in the subclass. This method is called in main().

*@abstractmethod* **shutoff(self, \*\_args):**

Empty abstract method that needs to be implemented in the subclass. This method is called when the module is finished. Should unsubscribe to dialogs and other relevant signals.

**exit(self, \*\_args):**

Callback function for when the event *exit* is raised (happens when user says exit/quit). Sets the value of *module\_finished* to True which should break the loop written in *run().*

**transfer\_to\_pepper(self, file\_path):**

Transfers the file specified by the parameter *file\_path* to the html folder of the current module using SSH. This is not an abstract method since it is the same among all modules.

### ExampleModule

We’ve created an example module contained in *example\_module.py* that inherits MBC and shows how a module can be written. This module does nothing more than say that it’s an example module, but you can implement a lot more. Take a look at the VasttrafikModule in *vasttrafik\_module.py* for some inspiration.

#### Important notes

* The python file of the module you want to run needs to be imported into *main.py*
* You must add a row in the file *HumanGreeting\_enu.top* where you specify how to start your module (by speech/event) and set the variable *mod\_to\_run* to the name of your module. **Remember to upload the .top file to /home/nao on Pepper.**
* If you want to be able to start your module by clicking an icon on the tablet, you need to write this code in *introduction.html*, add the corresponding logic (i.e. raising for example an event *example* on click, this event should be the same as the one you specified in the dialog) in the JavaScript file and transferring these files to the folder */home/nao/.local/share/PackageManager/apps/humangreeter/html* on Pepper.
* If you want to display any stuff on Pepper’s tablet, you need to create the views using html code. See the vasttrafik module for inspiration. The folder structure should be as follows
  + html
    - \*.html
    - css (dir)
    - js (dir)
    - images (dir)
* The above folder (html) needs to be transferred to the following folder on Pepper: */home/nao/.local/share/PackageManager/apps/<folder\_name>/* where *<folder\_name>* is the name of a folder you need to create on Pepper, and should be the same as the variable folder\_name that you pass when you create your module.
* Any dialogs you want to use need to be created (either using Choregraphe or a text editor) and transferred to */home/nao* on Pepper, then loaded in your module class. See *vasttrafik\_module.py* for details.

## Run human greeter

* Make sure Pepper is on
* Set him out of autonomous mode
* Make sure you're connected to the same WiFi
* Run program by writing "python main.py" in the terminal. Add the flag "w" if you want Pepper to walk around when searching for humans.
* Add a flag "--ip" followed by the IP of Pepper if it differs from the default one.
* Write "python main.py -h" for a help text of the program
* Quit the program with "Ctrl+c"

## Implemented modules

## 

### Vasttrafik

Public transportation application. You can either plan a trip or see the next departures from close by stations. Planning a trip can be done using speech but also in text by saying "\*text\*" to Pepper. There is no implementation for fetching Peppers current location which means that the coordinates of Pepper is fixed to Lindholmspiren. This can be implemented so that the departures from close by station makes more sense at other locations.

### Weather

Fetches and displays the weather from any location on earth. No speech input apart from exiting module. Displays either hour for hourly or daily.

### Survey

Simple survey application where you click either a green, yellow or red face depending on what you thought about the event. Say "\*Result\*" to be prompted to a result screen. You can go back to the survey from the result screen by saying "\*Go back\*". You can only end the survey from the result screen, this done by saying "\*Thank you Pepper\*", "\*Thanks\*" or "\*Bye Pepper\*".

### News

Pepper will randomly fetch a news story from a selected source and read the title and a summary. Both speech and buttons implemented

### Movie suggestions

Will give you a movie recommendation. Say for example "\*no\*" to get another movie. If you get movies which you think are to old, say "\*new\*" to get newer movies.

### Random fact

Pepper can tell you a random fact. Can be triggered both by speech or button click.

### Tell a joke

Pepper can tell you a Chuck Norris joke. Can be triggered both by speech or button click. Chuck Norris is changed to Zlatan, Zlatan is the best.