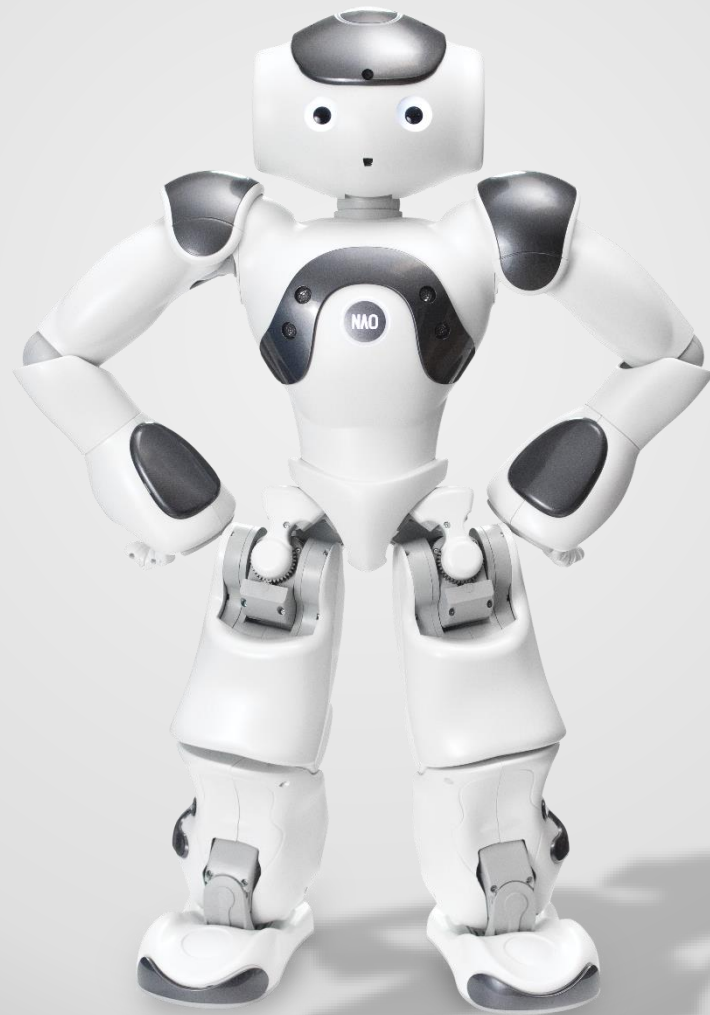


Nao

NAO⁶



語音辨識

20 多種語言

摔倒管理器

檢測摔倒情況並觸發保護

摔倒恢復

能夠獨自站立起來

聲音

4 個全向式麥克風
2 個揚聲器

互聯性

藍牙、乙太網
、Wi-Fi

視覺探測

2 個 500 萬圖元攝像頭

計算能力

CPU 為 ATOM E3845
四核 1.91 GHz
4 GB DDR3 記憶體
32 GB SSD

58 cm

優雅的肢體動作

25個自由度

聲納的探測

4 個檢測障礙物的聲納

自我調整行走

8 個壓力感測器

5.5 kg



硬體部位 介紹



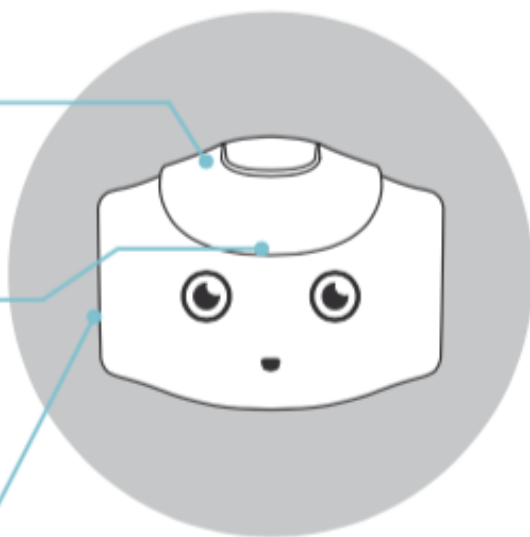
麥克風



攝影機



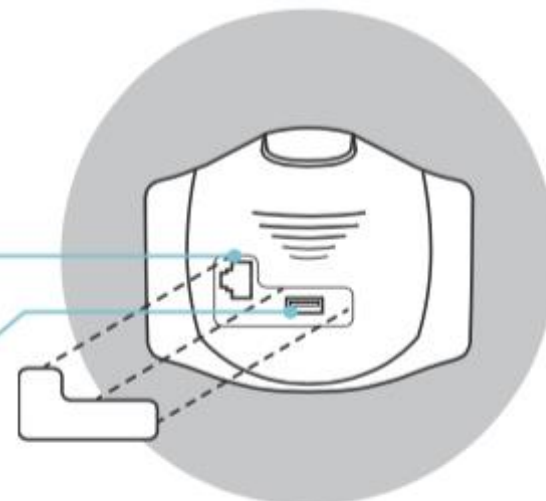
揚聲器



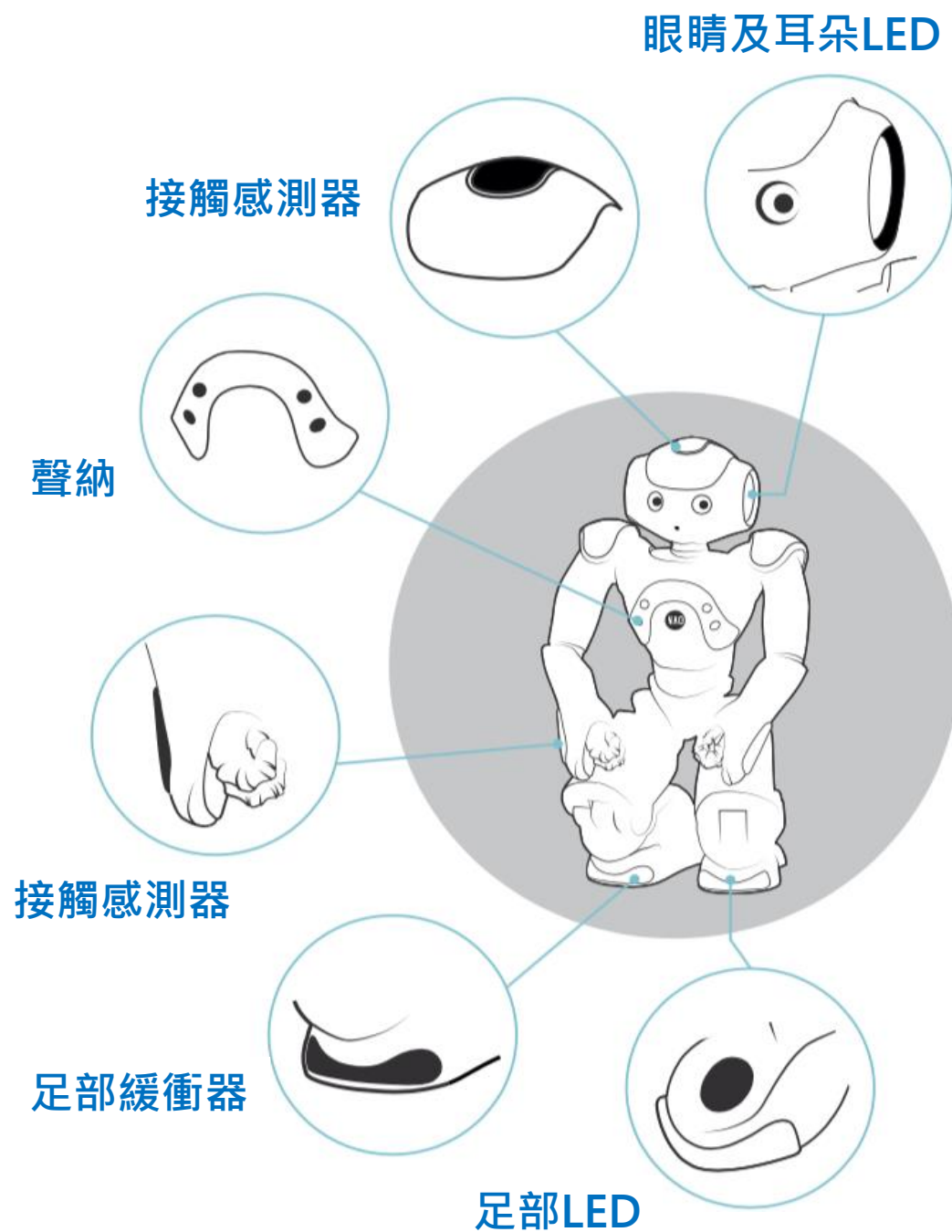
乙太網口



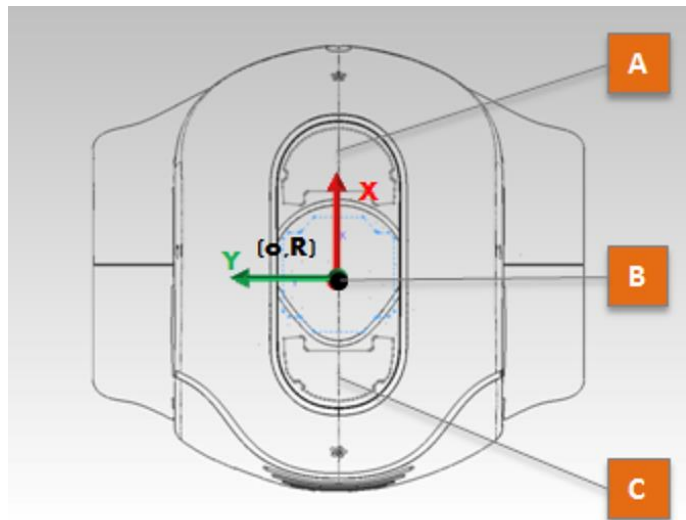
USB口



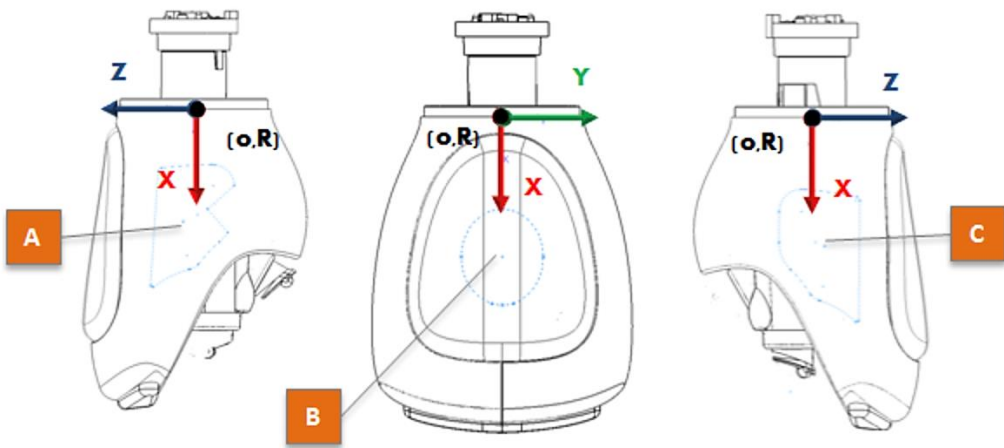
硬體部位介紹



硬體部位介紹



頭頂處**接觸感測器**
(3個) : [Doc](#)



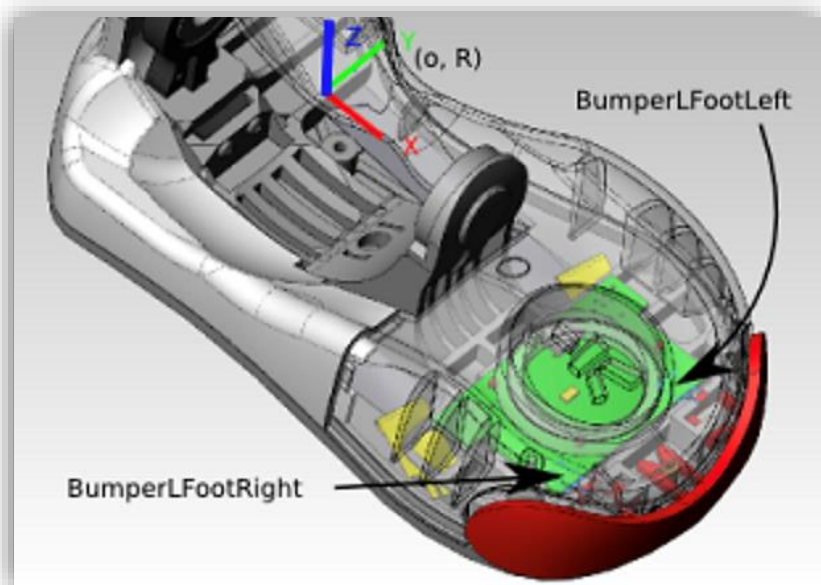
手背處**接觸感測器**
(雙手各3個)
: [Doc](#)

- 主要用來做互動和事件觸發
- 更多詳情 請點選[Doc](#)參閱

足部緩衝器

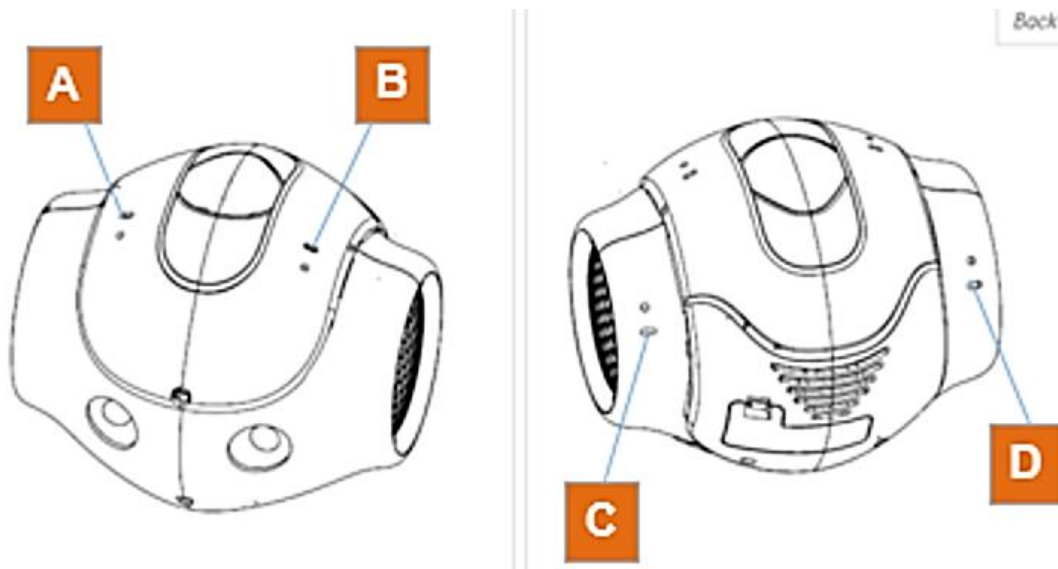
左、右腳 (各2個)

: [Doc](#)



硬體部位介紹

- 主要用來做互動和事件觸發
- 更多詳情 請點選[Doc](#)參閱



麥克風：頭頂周圍
(4個)：[Doc](#)

硬體部位介紹

- 更多詳情 請點選[Doc](#)參閱

硬體部位 介紹

聲納：左右各 1 個
(每一組有一個接收器和發射器)
：[Doc](#)

Frequency: 40kHz

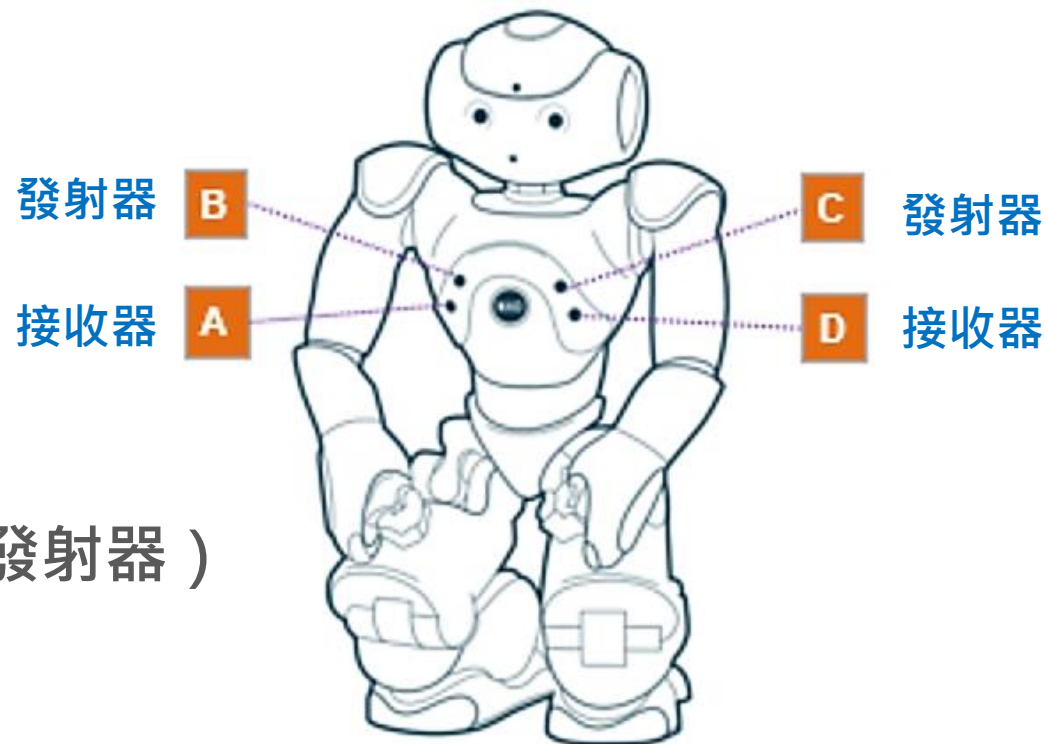
Resolution: 1cm-4cm (depending on distance)

Detection range: 0.20 m - 0.80 m

Under 20 cm there is no distance information, the robot only knows that an object is present.

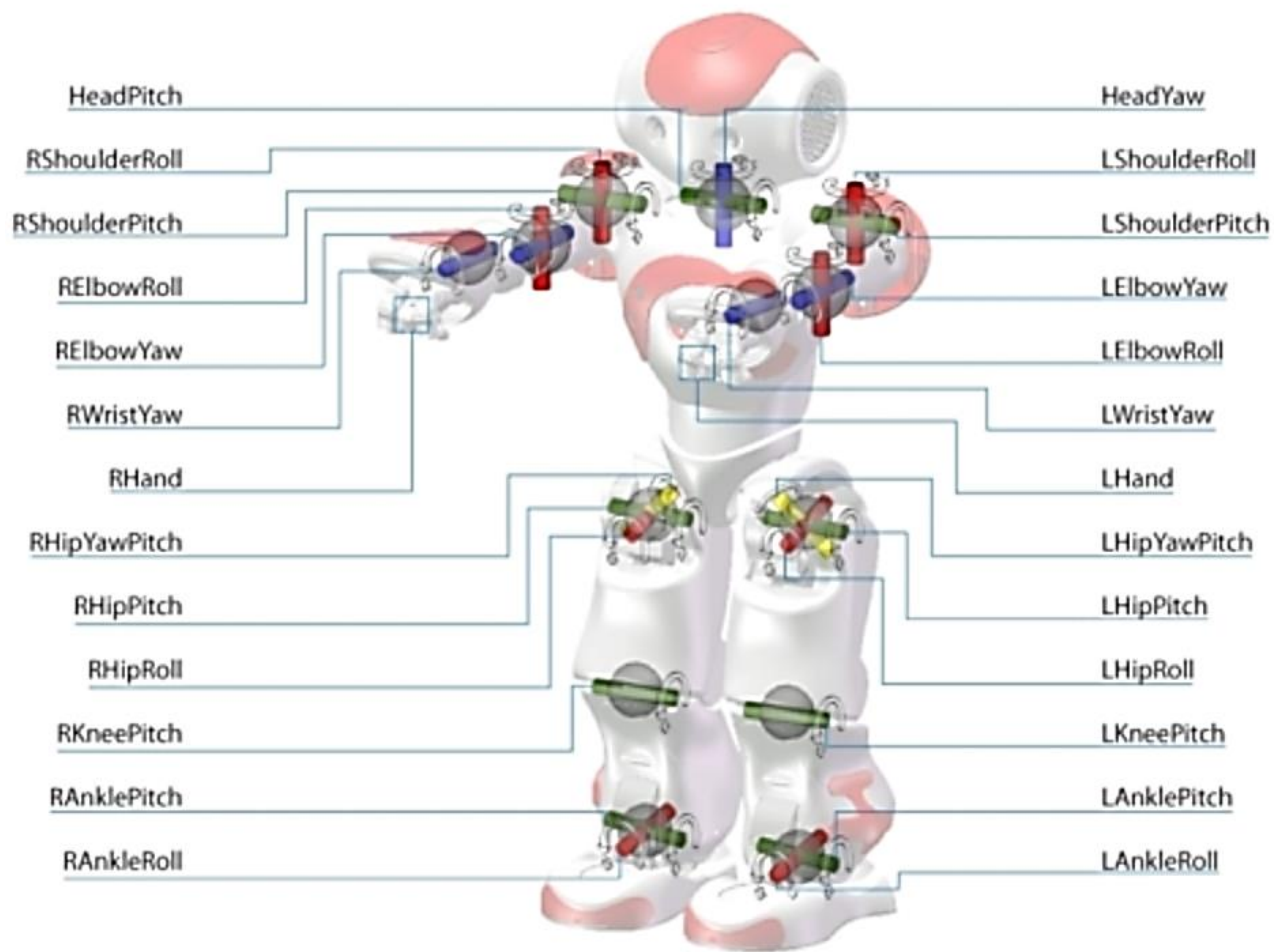
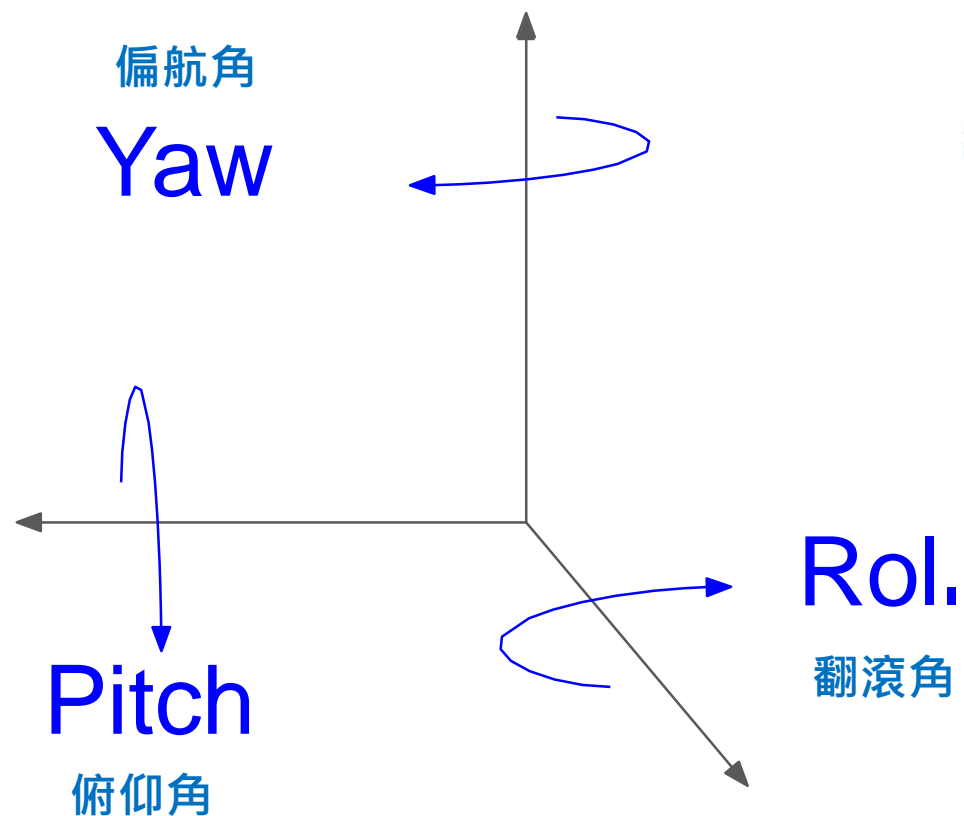
Above 80 cm the value returned is an estimation. For further details, read the important tips in [US/Sensor \(m\)](#).

Effective cone: 60°



● 更多詳情 請點選[Doc](#)參閱

硬體部位 介紹



- 根據關節和姿態方向來命名電機

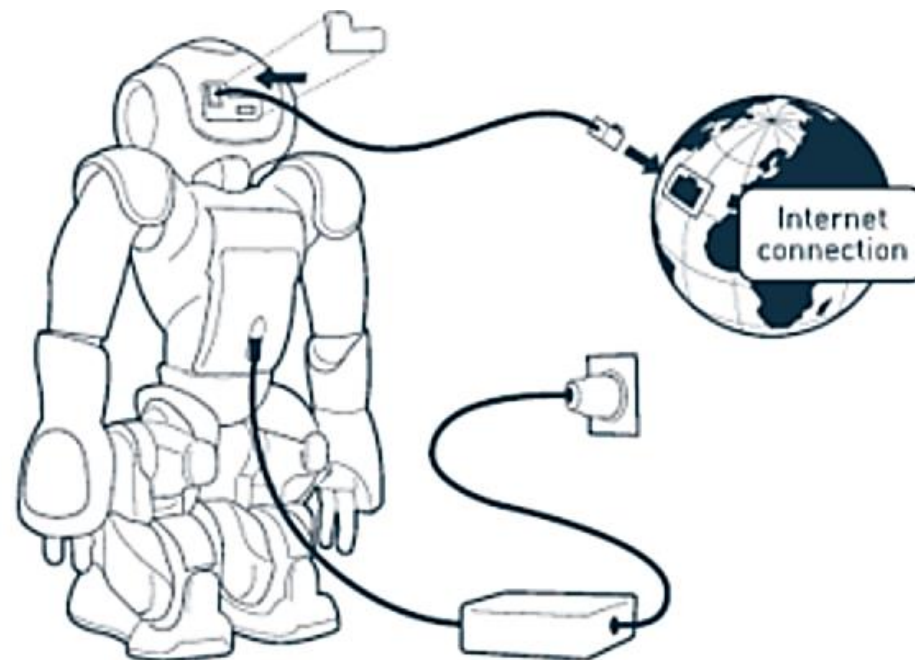
硬體部位 介紹

乙太網連接

初次配置網路首先需要用有線連接

網路連接方式

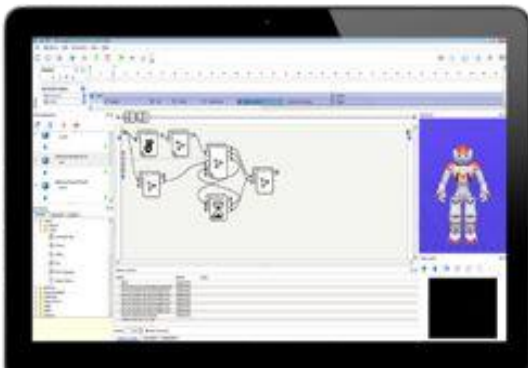
無線網連接：Wi-Fi 和藍牙4.0



軟體



軟體-在電腦上的開發工具



Choregraphe 2.8

簡易視覺化程式設計工具



Monitor

監視機器人

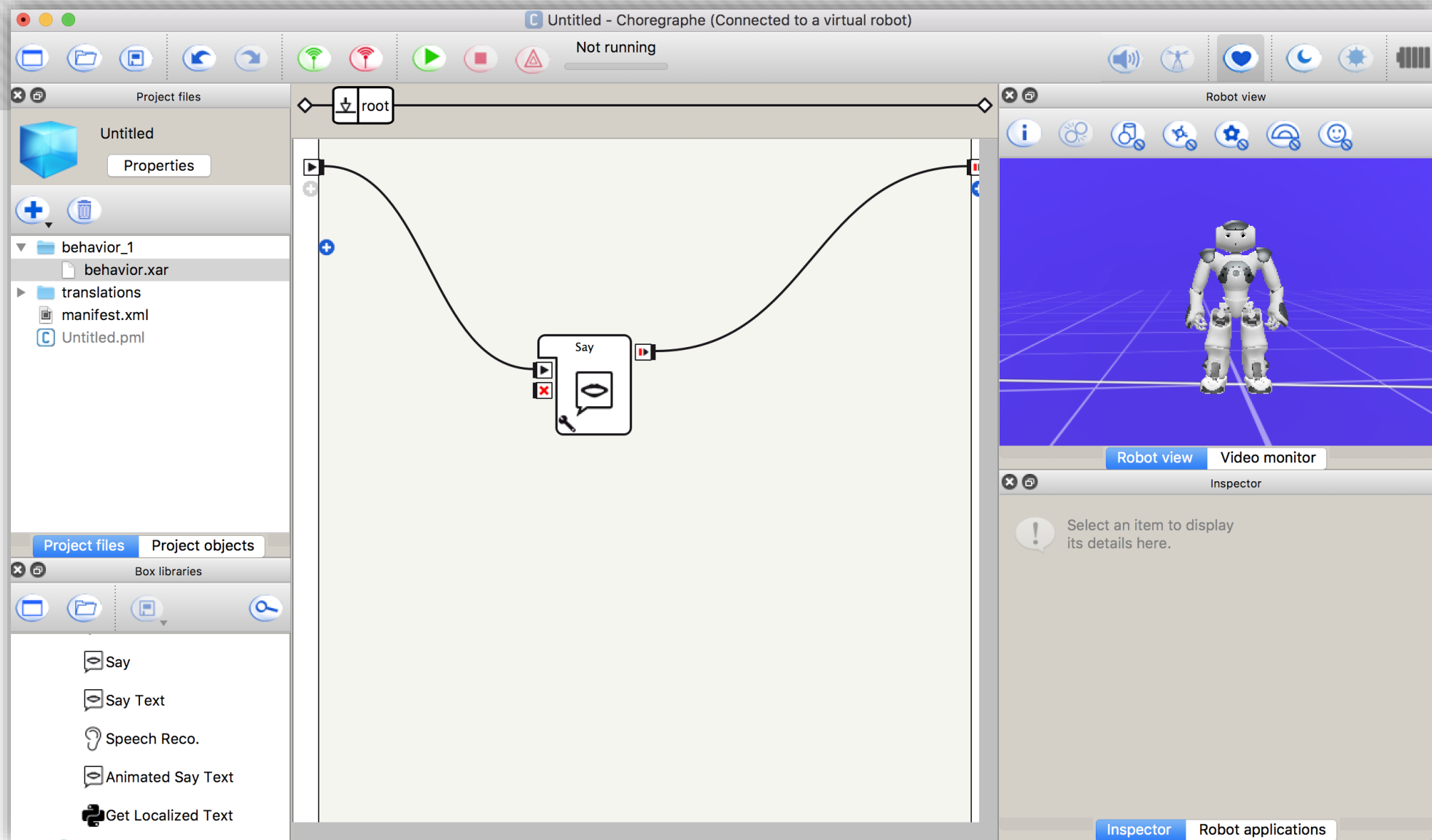
內部感測器資料等



**Software
Development Kit**

提供給C++ 或者
Python的綜合API

軟體-Choregraphe



軟體- Choregraphe



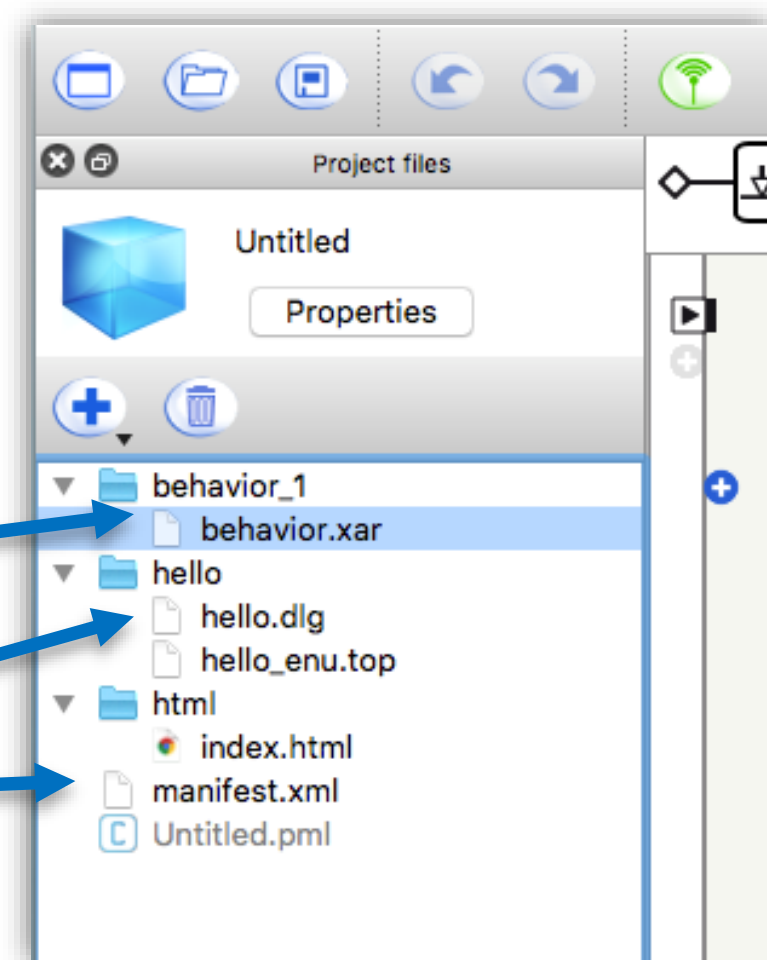
- 創建動作、行為和對話方塊。
- 在虛擬機器人上或直接在實體機器人上測試。
- 監視和控制你的機器人。
- 使用您自己的 **Python** 代碼編程。

Choregraphe 允許您創建對話、服務和行為的應用程式，
例如與人互動、跳舞、發送電子郵件，而無需編寫一行代碼。

軟體- Choregraphe

一個應用程式(APP)包含什麼?

- 行為 (.xar): NAO能做什麼
- 對話主題 (.dlg / .top): NAO可以談論什麼
- 屬性 (圖示, 名字, ...)

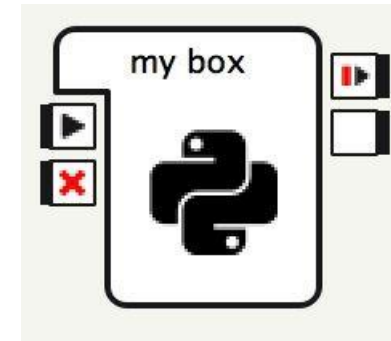


軟體- Choregraphe

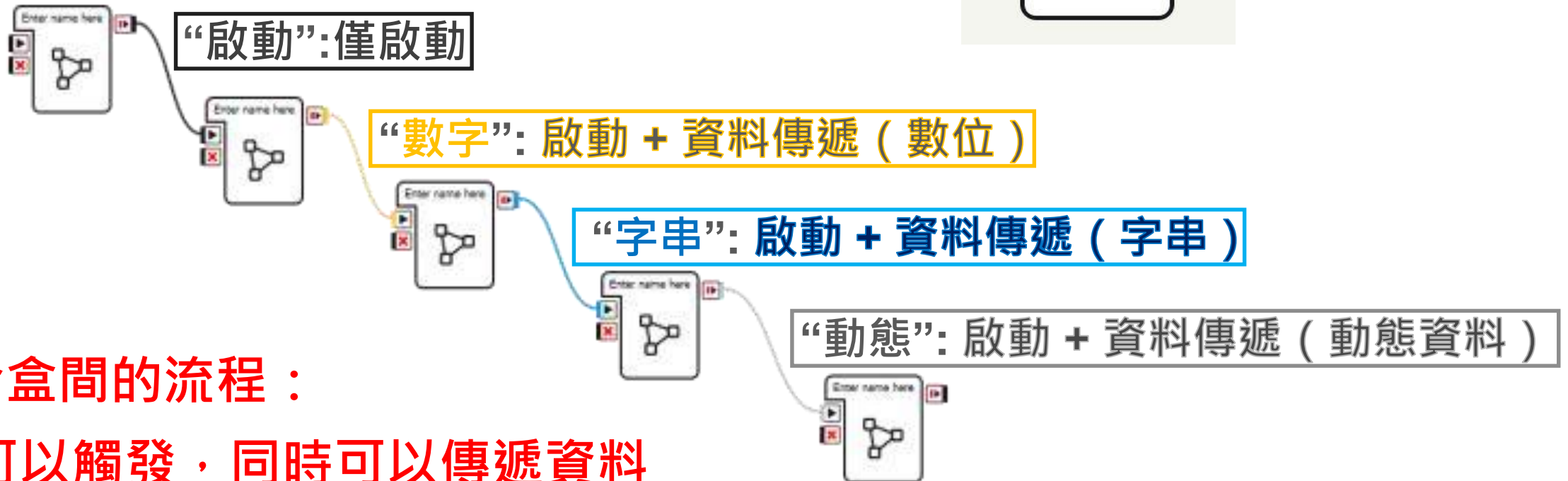
指令盒 (Choregraphe程式設計的基本組件)

指令盒-輸入輸出口

開始輸入口
停止輸入口



停止輸出口
非停止輸出口



指令盒間的流程：

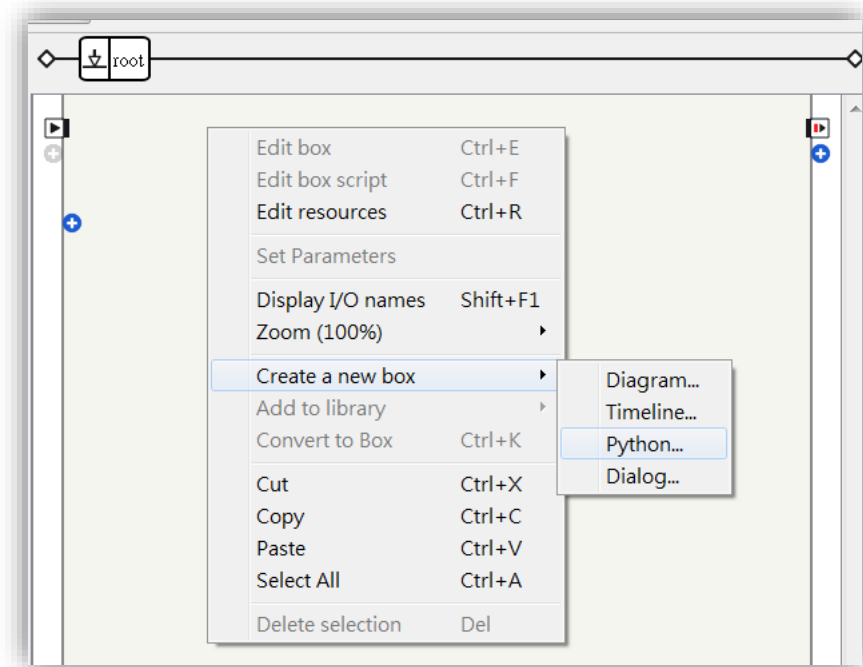
可以觸發，同時可以傳遞資料

軟體 - 編寫自己的指令盒

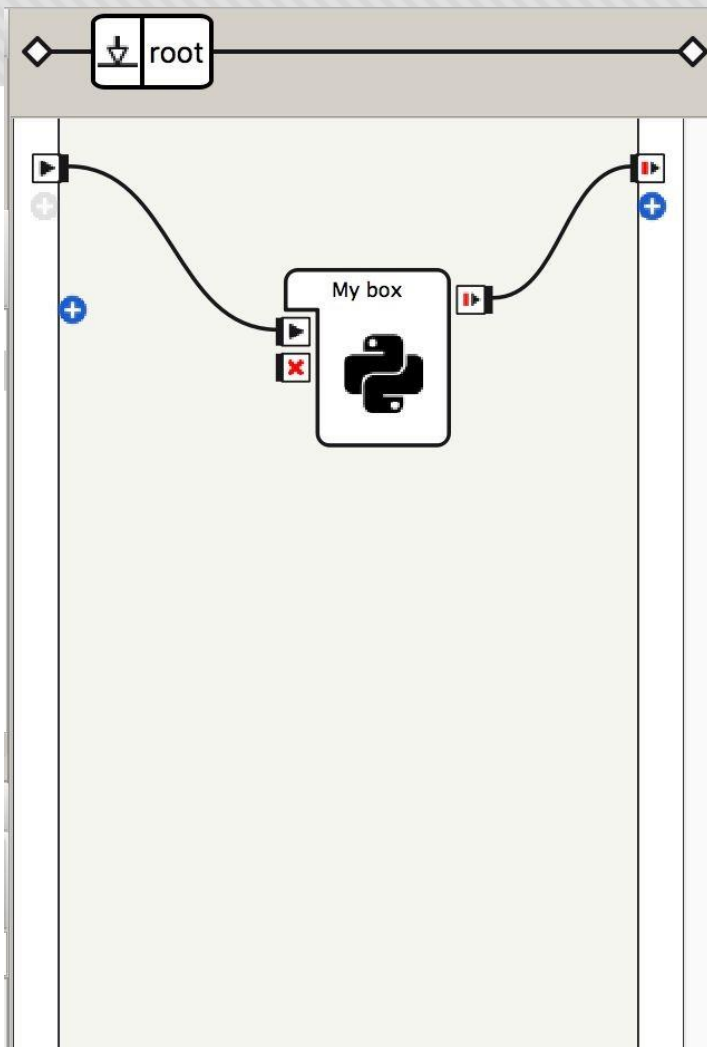


右擊程式設計區域空白處
選擇”添加新指令盒”
... Python !

然後按兩下指令盒...撰寫代碼



軟體 - 編寫自己的指令盒



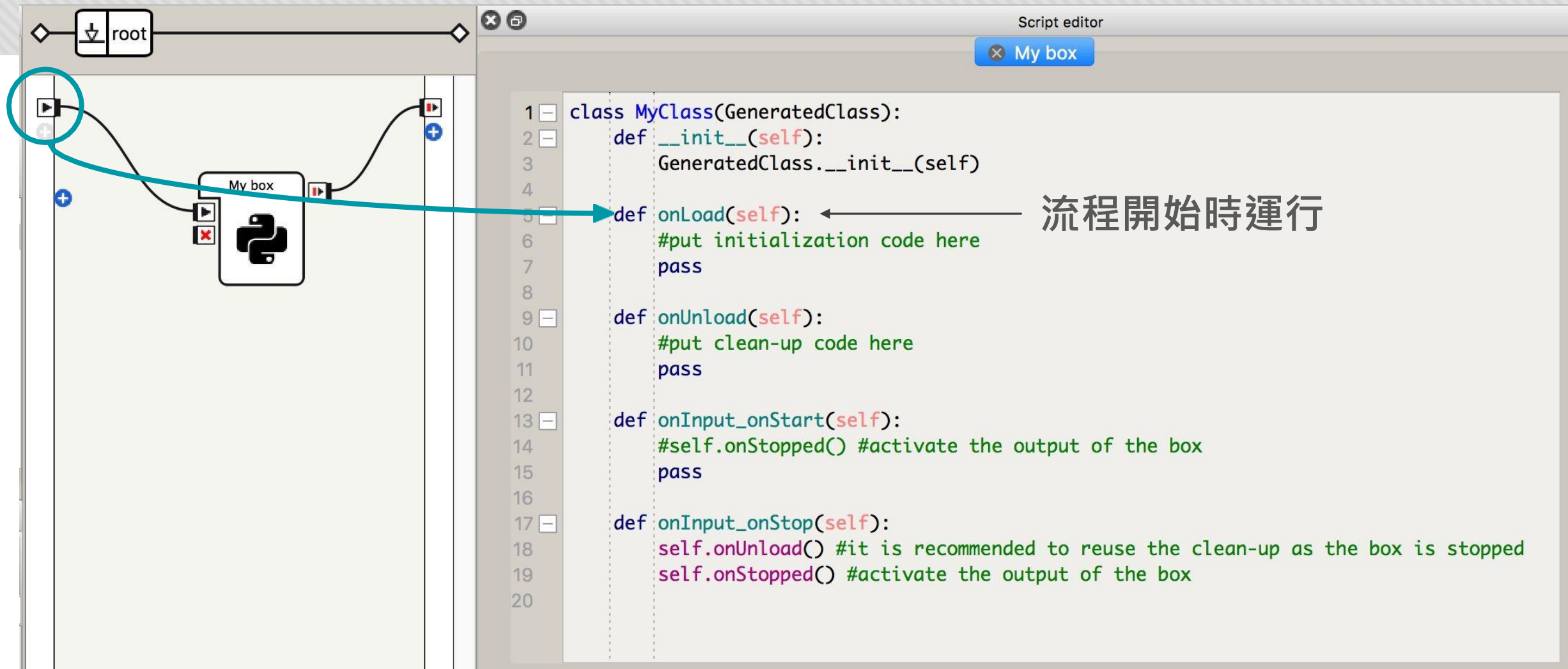
Script editor

My box

```
1 class MyClass(GeneratedClass):
2     def __init__(self):
3         GeneratedClass.__init__(self)
4
5     def onLoad(self):
6         #put initialization code here
7         pass
8
9     def onUnload(self):
10        #put clean-up code here
11        pass
12
13    def onInput_onStart(self):
14        #self.onStopped() #activate the output of the box
15        pass
16
17    def onInput_onStop(self):
18        self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
19        self.onStopped() #activate the output of the box
20
```

APP啟動時即會運行

軟體 - 編寫自己的指令盒

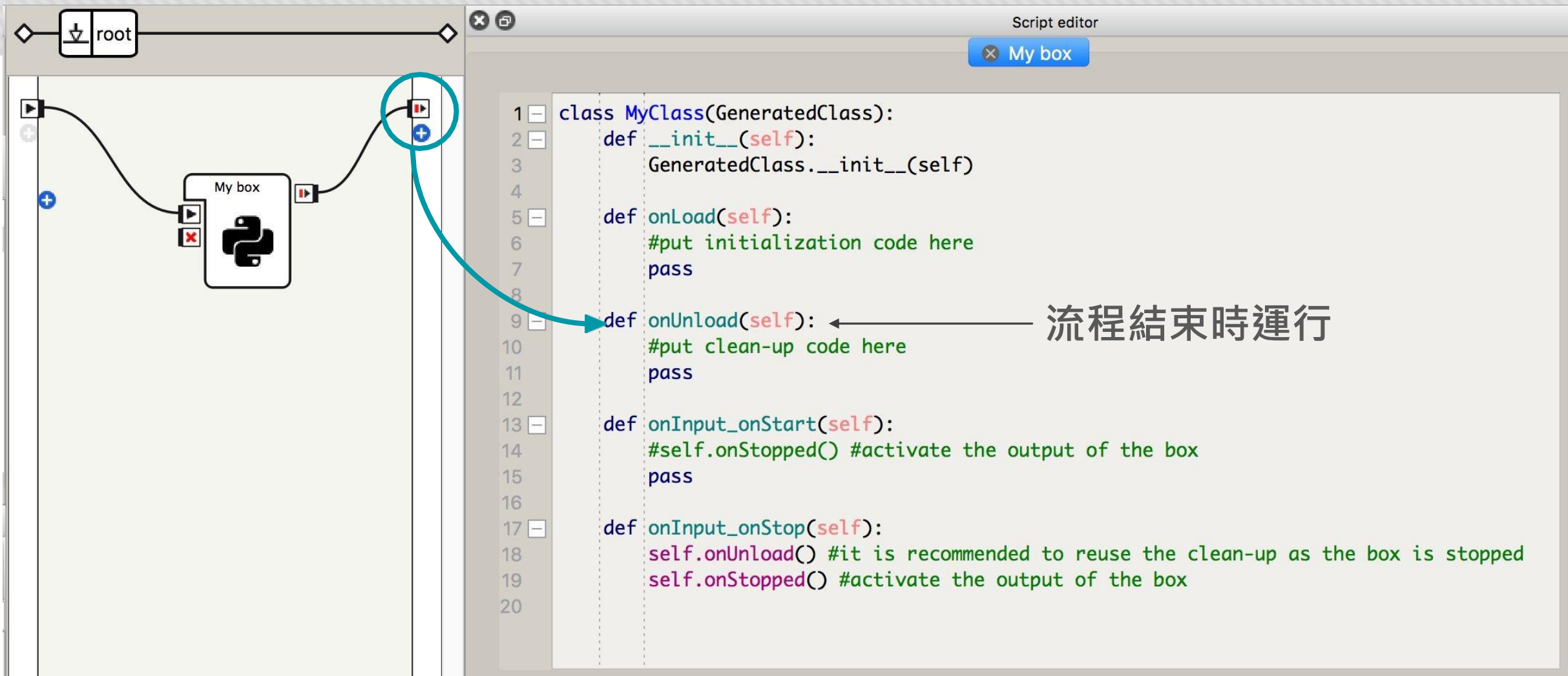


Script editor

My box

```
1 class MyClass(GeneratedClass):
2     def __init__(self):
3         GeneratedClass.__init__(self)
4
5     def onLoad(self): ← 流程開始時運行
6         #put initialization code here
7         pass
8
9     def onUnload(self):
10        #put clean-up code here
11        pass
12
13    def onInput_onStart(self):
14        #self.onStopped() #activate the output of the box
15        pass
16
17    def onInput_onStop(self):
18        self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
19        self.onStopped() #activate the output of the box
20
```

軟體 - 編寫自己的指令盒



The image displays a software development environment with two main components: a block diagram on the left and a script editor on the right.

Block Diagram: The diagram shows a flow starting from a 'root' block. A line connects to a 'My box' block, which contains a Python logo. From 'My box', the flow continues to a terminal block (represented by a red square with a white 'P' and a blue plus sign). This terminal block is circled in blue, and a blue arrow points from it to the 'onUnload' method in the script editor.

Script Editor: The editor window is titled 'Script editor' and 'My box'. It contains the following Python code:

```
1 class MyClass(GeneratedClass):
2     def __init__(self):
3         GeneratedClass.__init__(self)
4
5     def onLoad(self):
6         #put initialization code here
7         pass
8
9     def onUnload(self):
10        #put clean-up code here
11        pass
12
13    def onInput_onStart(self):
14        #self.onStopped() #activate the output of the box
15        pass
16
17    def onInput_onStop(self):
18        self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
19        self.onStopped() #activate the output of the box
20
```

A blue arrow points from the 'onUnload' method (line 9) to the text '流程結束時運行' (Run when the process ends).

軟體 - 編寫自己的指令盒

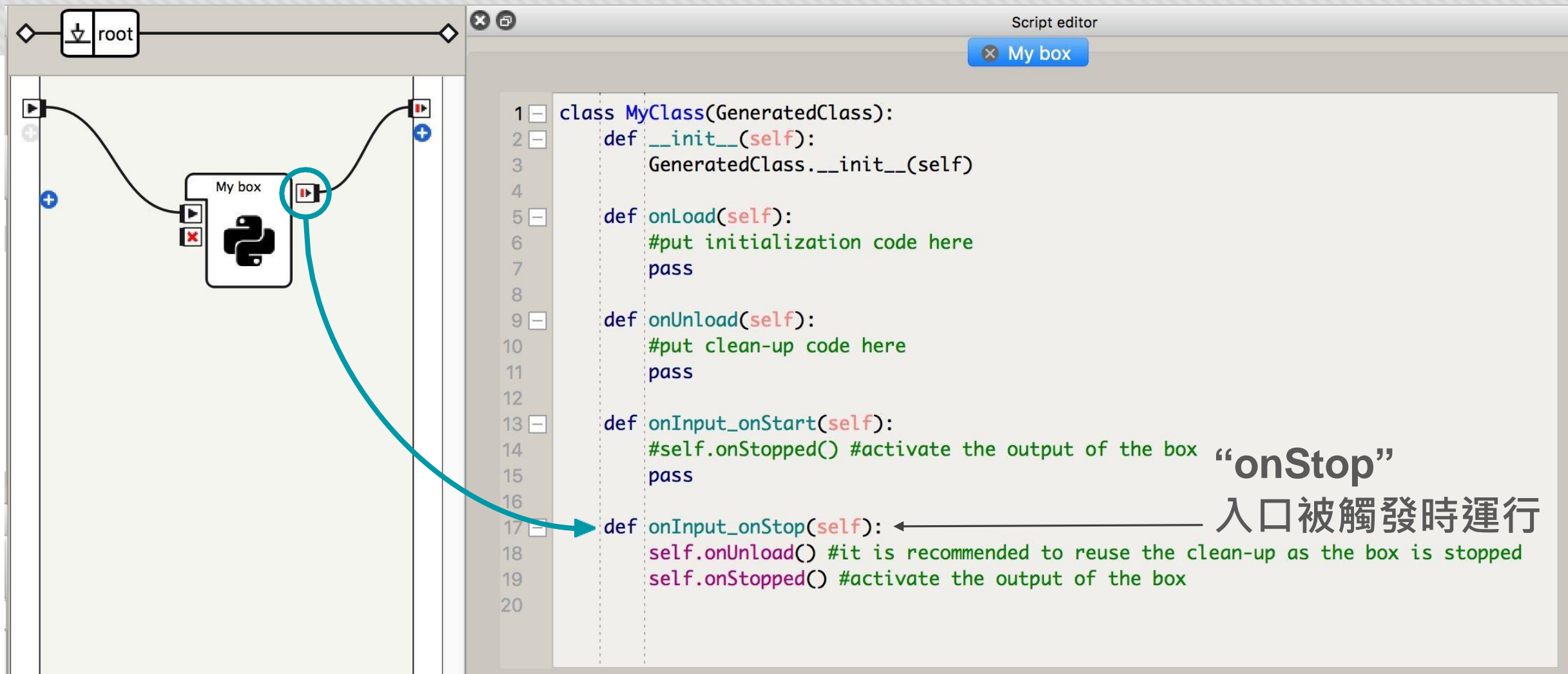
Script editor

My box

```
1 class MyClass(GeneratedClass):
2     def __init__(self):
3         GeneratedClass.__init__(self)
4
5     def onLoad(self):
6         #put initialization code here
7         pass
8
9     def onUnload(self):
10        #put clean-up code here
11        pass
12
13    def onInput_onStart(self):
14        #self.onStopped() #activate the output of the box
15        pass
16
17    def onInput_onStop(self):
18        self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
19        self.onStopped() #activate the output of the box
20
```

“onStart”
入口被觸發時運行

軟體 - 編寫自己的指令盒



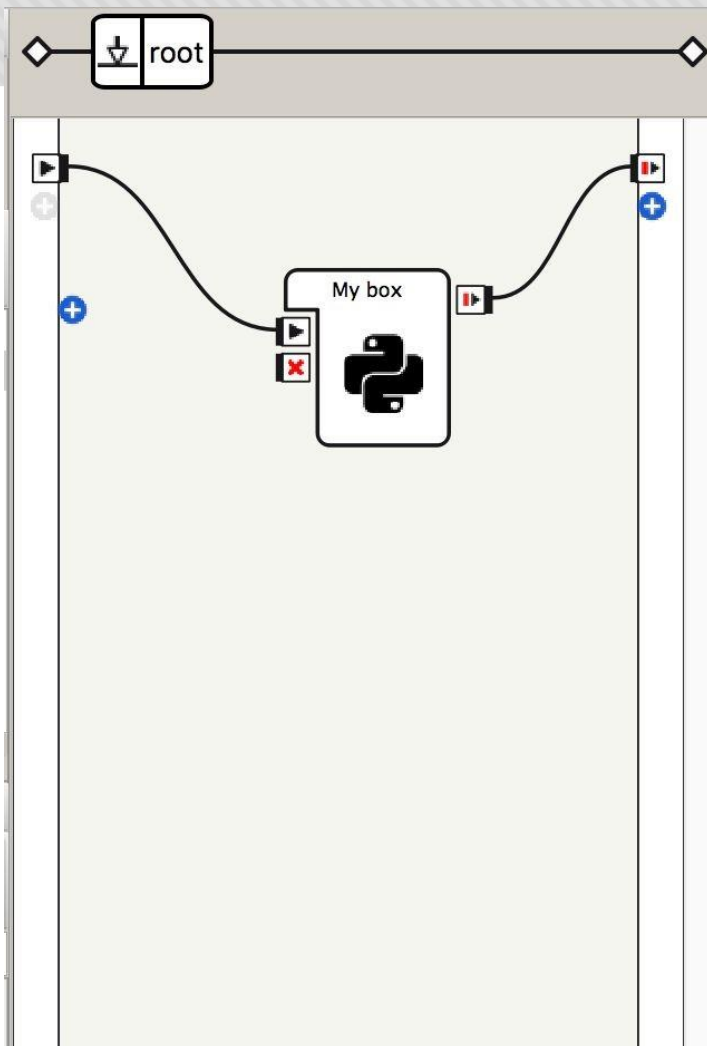
Script editor

My box

```
1 class MyClass(GeneratedClass):
2     def __init__(self):
3         GeneratedClass.__init__(self)
4
5     def onLoad(self):
6         #put initialization code here
7         pass
8
9     def onUnload(self):
10        #put clean-up code here
11        pass
12
13    def onInput_onStart(self):
14        #self.onStopped() #activate the output of the box
15        pass
16
17    def onInput_onStop(self):
18        self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
19        self.onStopped() #activate the output of the box
20
```

“onStop”
入口被觸發時運行

軟體 - 編寫自己的指令盒



Script editor

My box

```
1 class MyClass(GeneratedClass):
2     def __init__(self):
3         GeneratedClass.__init__(self)
4
5     def onLoad(self):
6         #put initialization code here
7         pass
8
9     def onUnload(self):
10        #put clean-up code here
11        pass
12
13    def onInput_onStart(self):
14        #self.onStopped() #activate the output of the box
15        pass
16
17    def onInput_onStop(self):
18        self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
19        self.onStopped() #activate the output of the box
20
```

APP啟動時即會運行

流程開始時運行

流程結束時運行

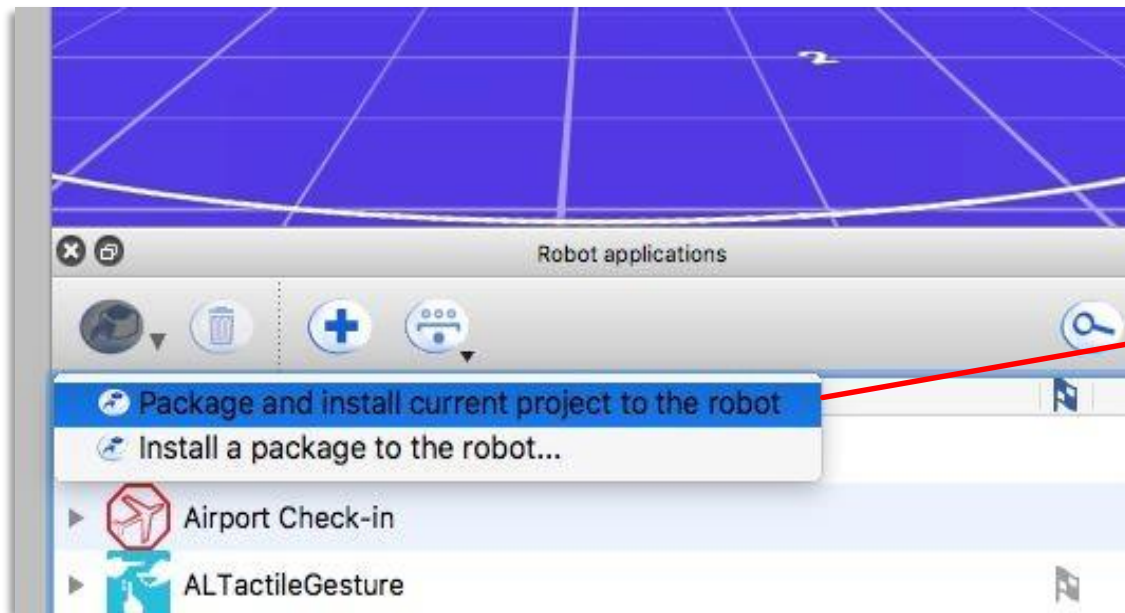
“onStart”
入口被觸發時運行

“onStop”
入口被觸發時運行

軟體- Choregraphe

運行App

→ 所有單次運行的app都有一個統一的名字
“**.lastUploadedChoregrapheBehavior**”
(暫存檔案，不保存。)



將當前專案
保存並安裝到機器人上

軟體- Choregraphe

如果想要匯出pkg文件包! (APP安裝包)

注意: 填寫屬性欄各項屬性:

Application:

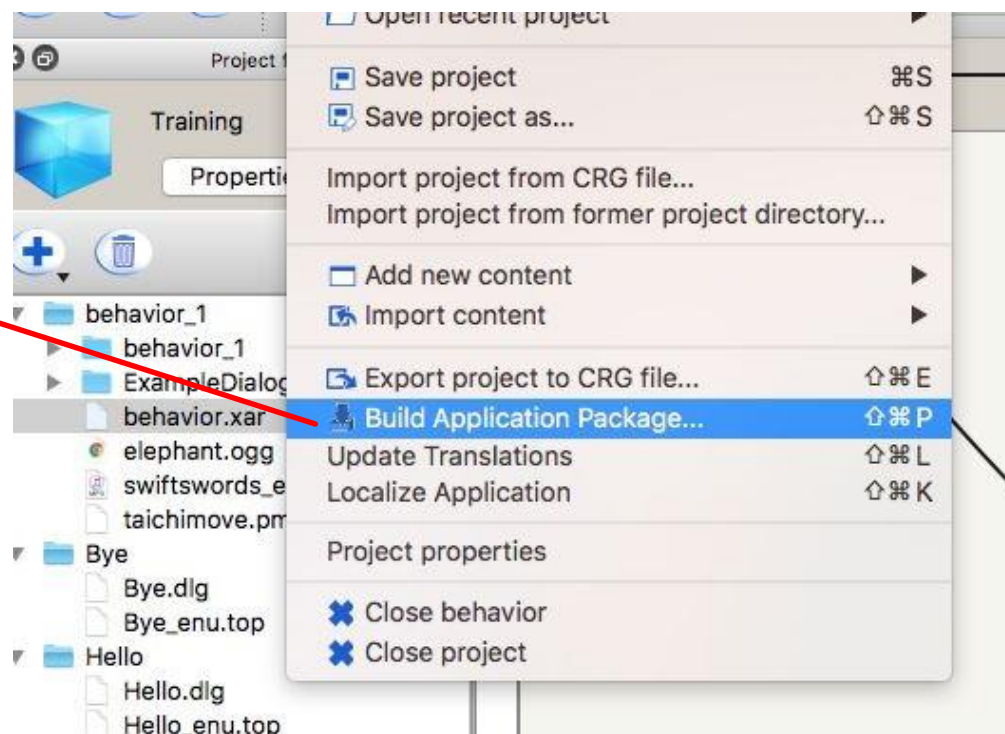
- Name
- ID
- Description...

Behaviors:

- Name
- Trigger sentences

Dialogs:

- Availability from autonomous life

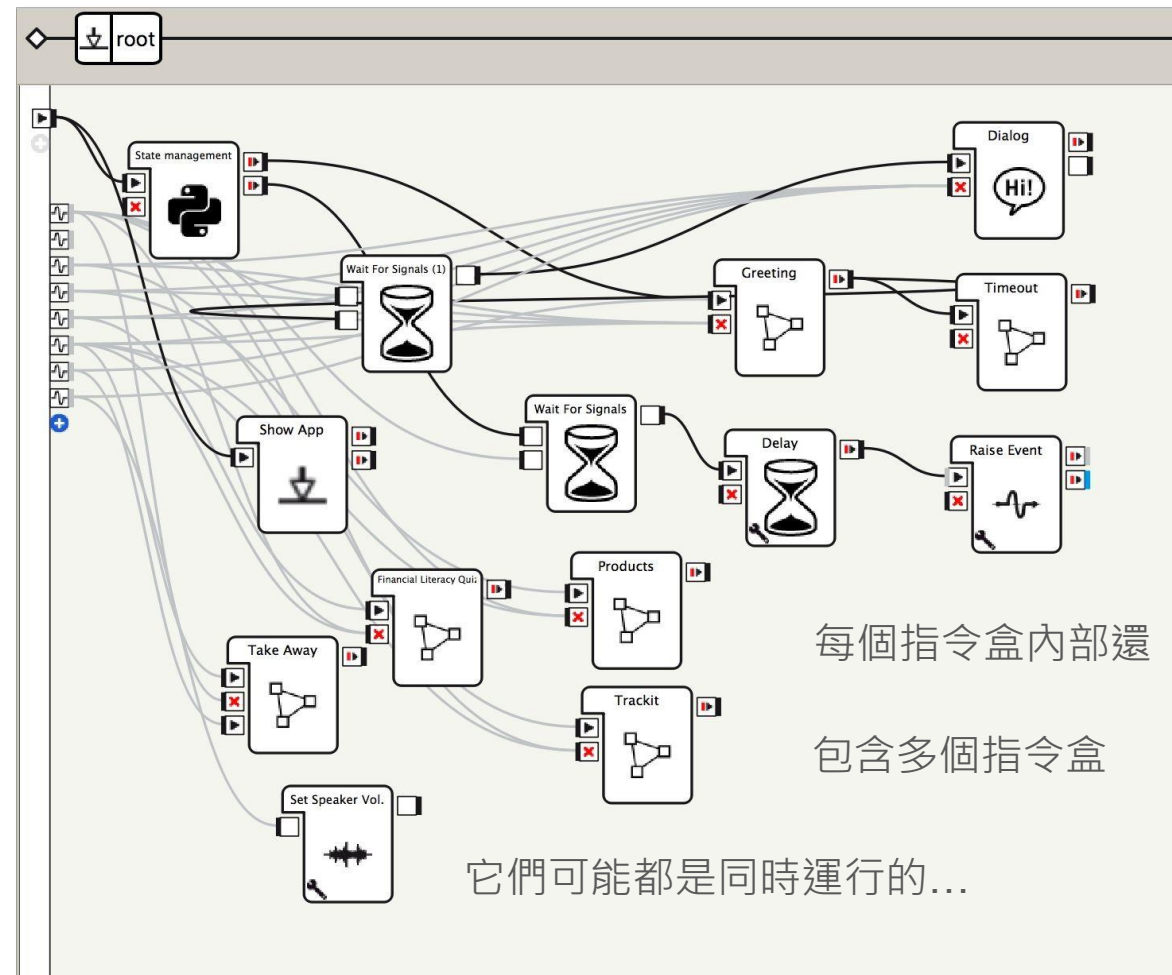


軟體- Choregraphe

一個“正式”的App，通常有2000行以上的代碼，如果全用指令盒的話，很快就會形成一個“蜘蛛網”...

使程式有以下缺點：

- 難以閱讀
- 很難debug
- 沒有清晰的邏輯結構



每個指令盒內部還

包含多個指令盒

它們可能都是同時運行的...

所以：
⇒ 還是用Python吧！

軟體-其他

你可以在機器人上用哪些軟體呢？

- 訪問檔案系統檔：**SFTP**
- 訪問終端：**SSH**

當你在機器人上工作時，機器人頭部處理你的所有命令

- 錄音存儲在頭部
- 日誌存儲在頭部



軟體-檔案系統

測試：拍照

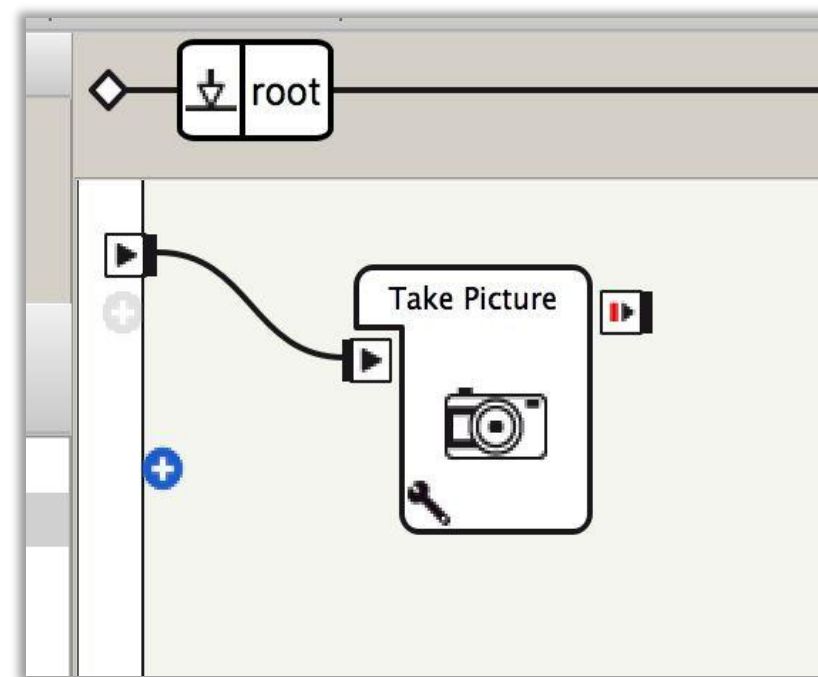
用**SFTP**工具連接到機器人，
並查看“recordings”資料夾下的文件

Host: **IP Address** （ 機器人ip ）

Username: **nao**

Password: **Robot Password (default nao)**

Port: **22**



軟體-SSH

讓我們使用SSH訪問機器人的頭部！

工具：

“putty”（Windows系統）

“ssh”命令列工具（Linux / Mac系統）

用帳號密碼登錄，預設都是 **nao**

ssh nao@<IP-of-robot>

默認密碼：**nao**

測試：

```
nao [0] ~ $ [type here:] say hello
```



補充說明

寫Nao和Python 複雜程式時還可以使用範本直接創建：

<https://github.com/aldebaran/robot-jumpstarter>

安裝Python SDK時請確保電腦已經安裝32位python2.7版本



**Thank you for your
attention.**