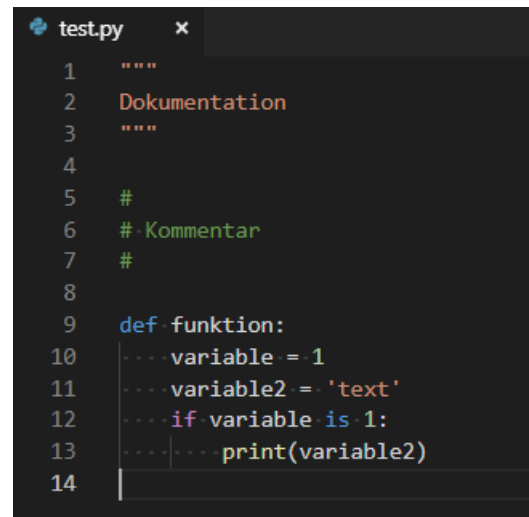


VSCode & Python

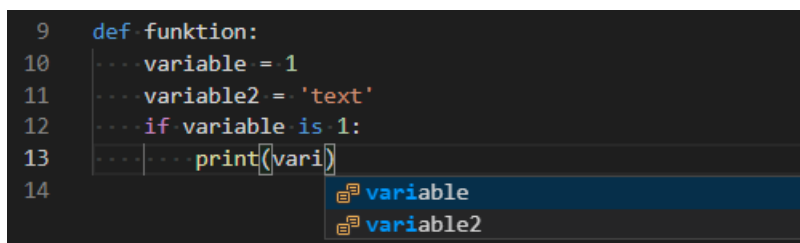
Kodieren ist weitaus komfortabler in einer modernen IDE (Integrated Development Environment) als in einem reinen Textverarbeitungsprogramm (wie z.B. „leafpad“ auf dem Pi). Visual Studio Code hat ausgesprochen guten Plug-In Support (in VSCode „Extensions“ genannt), wodurch es sich hervorragend eignet für unsere Anwendungsbereiche („Use Cases“) anzupassen:

- (a) Syntax-Highlighting (ein bisschen Farbe):



```
test.py x
1  """
2  Dokumentation
3  """
4
5  #
6  # Kommentar
7  #
8
9  def funktion:
10     ... variable = 1
11     ... variable2 = 'text'
12     ... if variable is 1:
13     ...     ... print(variable2)
14
```

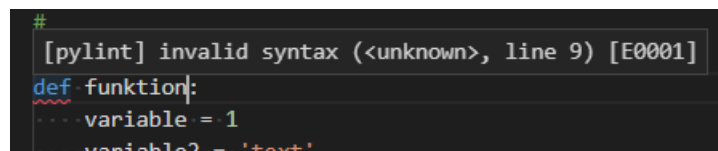
- Auto-Complete („IntelliSense“)



```
9  def funktion:
10     ... variable = 1
11     ... variable2 = 'text'
12     ... if variable is 1:
13     ...     ... print(vari]
14
variable
variable2
```

- Statische Codeanalyse („Linting“)

- (a) FEHLER: In diesem Beispiel fehlt „()“ hinter „funktion“:



```
#
[pylint] invalid syntax (<unknown>, line 9) [E0001]
def funktion:
... variable = 1
... variable2 = 'text'
```

- (b) WARNUNG: In diesem Beispiel wird darauf hingewiesen, dass wir „variable2“ zwar deklarieren und mit „text“ initialisieren, aber nie benutzen:

```
#
[pylint] Unused variable 'variable2' [W0612]
def
... variable2: str
... variable2 = 'text'
... if variable is 1:
...     print(unbekannte_variable)
```

(c) FEHLER: In diesem Beispiel wird eine Variable benutzt, die wir nie deklariert haben:

```
#
def funktion():
... variable = 1
... variable2 = 'text'
... if variabl [pylint] Undefined variable 'unbekannte_variable' [E0602]
...     print(unbekannte_variable)
```

- Refactoring: z.B. „variable2“ umbenennen und es wird automatisch überall umbenannt, wo es im Code benutzt wird.
- Integrierte Kommandozeile („Shell“)

```
9 def funktion():
10     ... variable = 1
11     ... variable2 = 'text'
12     ... if variable is 1:
13         ... print(variable2)
14
15 funktion()
16
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: powershell

Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

PS C:\Users\Tom\Dropbox\Robocup 2018_2019\Material\RaspberryPi Admin\Software Host\VSCode Konfig> python test.py
PS C:\Users\Tom\Dropbox\Robocup 2018_2019\Material\RaspberryPi Admin\Software Host\VSCode Konfig> python test.py
text
PS C:\Users\Tom\Dropbox\Robocup 2018_2019\Material\RaspberryPi Admin\Software Host\VSCode Konfig> |

1 Anforderungen

VSCode direkt auf dem Pi zu installieren ist zwar möglich, aber vermutlich recht lahm, weswegen ich es gar nicht erst versucht habe. Wir wollen es also auf einem Host-Rechner installieren und die Dateien dann auf den Pi spiegeln. Das hat zusätzlich den Vorteil, dass wir vom Code immer ein Backup auf dem Host-Rechner haben und Code-Schnipselchen, die keine Hardware benötigen, auch schnell mal auf dem Host-PC ausprobieren können (so habe ich z.B. für das Display die Beispiel-Renders getestet, obwohl ich das Display nicht da hatte). Wir wollen Syntax-Highlighting, IntelliSense und Linting für Python. Idealerweise kann man den Code sogar debuggen. Idealerweise wollen wir den VSCode Editor so selten wie möglich verlassen (d.h. andere Programme benutzen).

Daraus ergeben sich folgende Anforderungen:

1. Python und alle Python-Libraries, die ohne Hardware auskommen, sollen auf dem Host-PC installiert sein.
2. Python und Python Linting sollen in VSCode aktiviert sein.
3. Python Refactoring soll in VSCode aktiviert sein.
4. Beim Speichern sollen die Quellcode-Dateien automatisch auf den Pi kopiert werden.
5. Python Programme sollen innerhalb VSCode aus einer lokalen Shell (auf dem Host-PC) ausführbar sein.
6. Python Programme sollen innerhalb VSCode aus einer Shell auf dem Pi ausführbar sein.

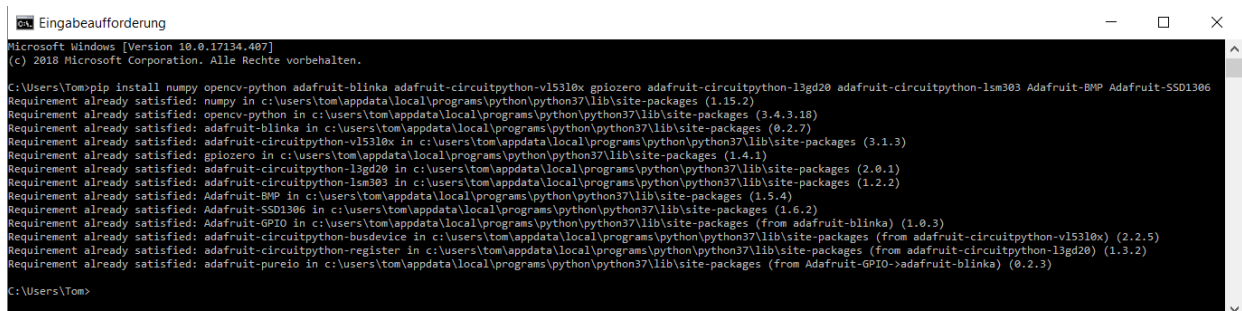
2 Python installieren

1. Auf dem Pi habe ich für Python3 entschieden, also wird das auch auf dem Host installiert:
<https://www.python.org/downloads/>
2. Alle Python-Libraries aus „Anleitung Hardwarezugriff“, Kapitel 1.3, installieren, die keine Hardware benötigen und somit bei der Installation keinen Fehler schmeißen:

In der Windows-Kommandozeile (cmd):

**pip install numpy opencv-python adafruit-blinka adafruit-circuitpython-vl53l0x
gpiozero adafruit-circuitpython-l3gd20 adafruit-circuitpython-lsm303 Adafruit-BMP
Adafruit-SSD1306**

Bei mir ist schon alles installiert:



```

Eingabeaufforderung
Microsoft Windows [Version 10.0.17134.487]
(c) 2018 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Tom>pip install numpy opencv-python adafruit-blinka adafruit-circuitpython-vl53l0x gpiozero adafruit-circuitpython-l3gd20 adafruit-circuitpython-lsm303 Adafruit-BMP Adafruit-SSD1306
Requirement already satisfied: numpy in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (1.15.2)
Requirement already satisfied: opencv-python in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (3.4.3.18)
Requirement already satisfied: adafruit-blinka in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (0.2.7)
Requirement already satisfied: adafruit-circuitpython-vl53l0x in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (3.1.3)
Requirement already satisfied: gpiozero in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (1.4.1)
Requirement already satisfied: adafruit-circuitpython-l3gd20 in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (2.0.1)
Requirement already satisfied: adafruit-circuitpython-lsm303 in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (1.2.2)
Requirement already satisfied: Adafruit-BMP in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (1.5.4)
Requirement already satisfied: Adafruit-SSD1306 in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (1.6.2)
Requirement already satisfied: Adafruit-GPIO in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (from adafruit-blinka) (1.0.3)
Requirement already satisfied: adafruit-circuitpython-busdevice in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (from adafruit-circuitpython-vl53l0x) (2.2.5)
Requirement already satisfied: adafruit-circuitpython-register in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (from adafruit-circuitpython-l3gd20) (1.3.2)
Requirement already satisfied: adafruit-pureio in c:\users\tom\appdata\local\programs\python\python37\lib\site-packages (from Adafruit-GPIO->adafruit-blinka) (0.2.3)

C:\Users\Tom>

```

3. Fürs Linting pylint installieren:

In der Windows-Kommandozeile:

pip install pylint

3 Visual Studio Code installieren

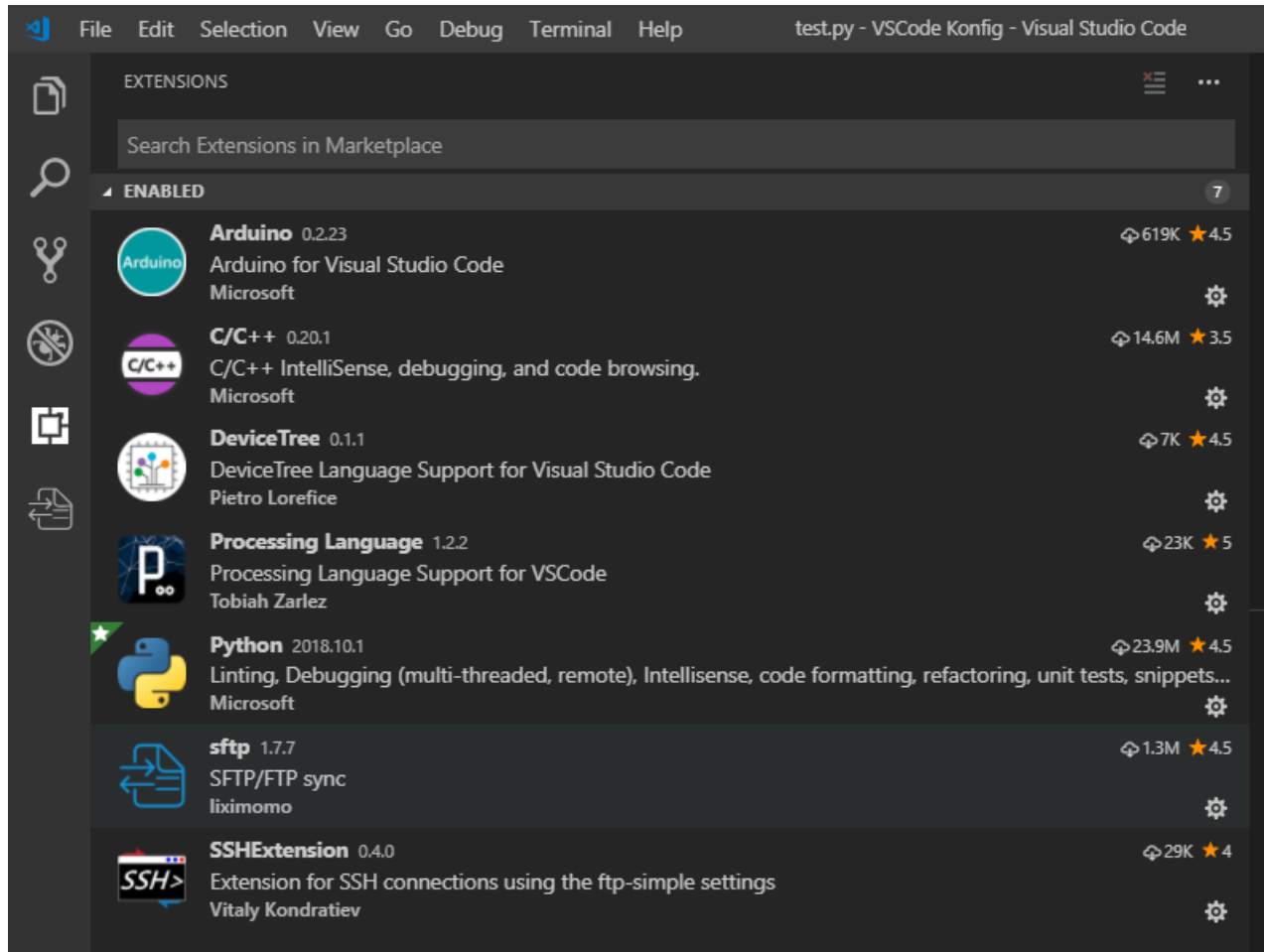
1. <https://code.visualstudio.com/>

3.1 Extensions installieren

1. Für Python Syntax-Highlighting, Code-Completion und Linting: „Python“:

<https://code.visualstudio.com/docs/languages/python>

2. Für das automatische Kopieren der Quellcode-Dateien auf den Pi: „sftp“
3. Für die Shell des Pi direkt in IntelliJ: „SSHExtension“
4. VSCode neu starten.



3.2 Extensions konfigurieren

Das geht natürlich alles innerhalb VSCode, ist aber ein wenig gewöhnungsbedürftig. Einfach ist:

1. Einen Ordner aussuchen/erstellen, in dem alle Python-Skripte abgelegt werden sollen. Das ist unser „Workspace“.
2. In VSCode „File“ → „Open Workspace...“ → Ordner auswählen.
3. In diesem Ordner einen Ordner „vscode“ anlegen, falls er nicht schon existiert. Das geht am besten direkt in VSCode, da Windows sonst rummeckert, dass der Ordner mit einem Punkt losgeht. Der Punkt kommt aus der Linux-Welt. Dort sind alle Dateien/Ordner, die mit einem Punkt beginnen, versteckt.
4. Datei „vscode/settings.json“ erstellen und foldendes hinein kopieren. Ihr findet sie auch in dem „Anleitung Hardwarezugriff ZIP“.

Das sind die Profile für das „SSHExtension“-Plugin. Ihr nutzt mit dem DietPi vmtl. nur „DietPi LAN statisch“ oder „DietPi WLAN statisch“ und könnt die anderen Profile gern löschen. „host“, „port“, „username“ und „password“ entsprechen den Einstellungen aus „Anleitung Netzwerk Raspberry Pi“. „path“ sollte dem Ordner auf dem Pi entsprechen, wo ihr nach dem Einloggen hineinspringen wollt, damit man nicht jedes Mal mit „cd“ dorthin wechseln muss (was allerdings bei mir nicht funktionierte).

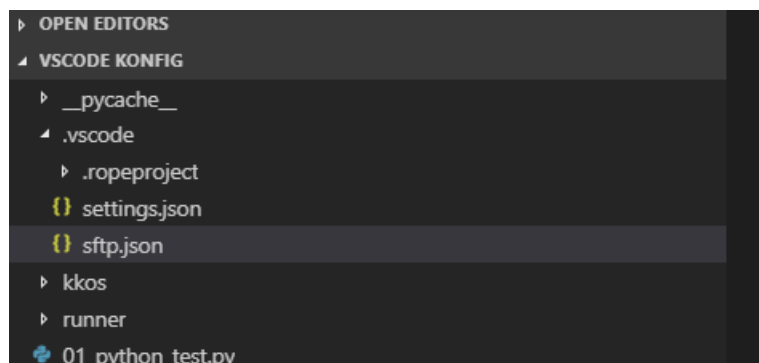
```
{
  "sshexextension.serverList": [
    {
      "name": "DietPi Peer-to-Peer oder WLAN",
      "host": "DietPi.local",
      "port": 22,
      "username": "root",
      "password": "dietpi",
      "path": "/mnt/dietpi_userdata/"
    },
    {
      "name": "DietPi LAN statisch",
      "host": "192.168.0.100",
      "port": 22,
      "username": "root",
      "password": "dietpi",
      "path": "/mnt/dietpi_userdata/"
    },
    {
      "name": "DietPi WLAN statisch",
      "host": "192.168.4.1",
      "port": 22,
      "username": "root",
      "password": "dietpi",
      "path": "/mnt/dietpi_userdata/"
    },
    {
      "name": "Raspbian Peer-to-Peer oder WLAN",
      "host": "raspberrypi.local",
      "port": 22,
      "username": "pi",
      "password": "raspberrypi",
      "path": "/home/pi/"
    }
  ]
}
```

5. Datei „.vscode/sftp.json“ erstellen und folgendes hinein kopieren. Ihr findet sie auch in dem „Anleitung Hardwarezugriff ZIP“.

Das sind die Profile für das „sftp“-Plugin. Ihr nutzt mit dem DietPi vmtl. Nur „DietPi LAN statisch“ oder „DietPi WLAN statisch“ und könnt die anderen Profile gern löschen. „watcher/files“ ist ein Filter für die Dateien, die auf den Pi gespiegelt werden sollen. „**/*“ ist eine Wildcard, d.h. alles wird gespiegelt. Man könnte z.B. auch ausschließlich Dateien spiegeln, die auf „.py“ enden. „autoUpload“ ist das was wir wollen, d.h. beim Speichern wird alles auf dem Pi hochkopiert. „autoDelete“ müsst ihr selber wissen. Wenn man es nicht macht, bleiben Dateien, die man auf dem Host-PC löscht, auf dem Pi noch liegen und man muss sie manuell löschen. Vllt. ist also „true“ auch hier ganz praktisch. Müsst ihr

ausprobieren, ob ihr das mögt. Der Rest ist wie bei 4. „remotePath“ ist der Pfad, in den alles auf dem Pi gespeichert wird.

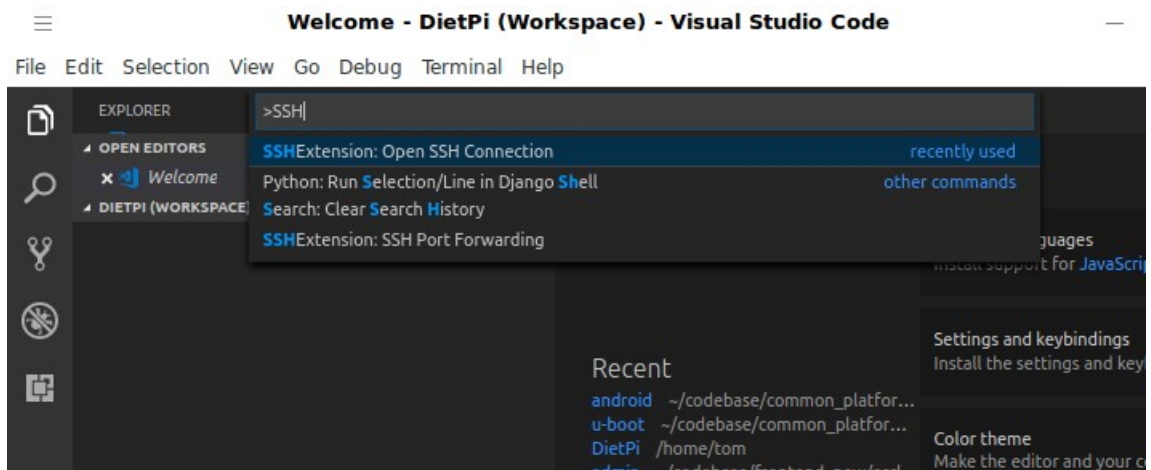
```
{
  "defaultProfile": "DietPi WLAN statisch",
  "name": "Pi",
  "protocol": "sftp",
  "port": 22,
  "watcher": {
    "files": "**/*",
    "autoUpload": true,
    "autoDelete": false
  },
  "profiles": {
    "DietPi Peer-to-Peer oder WLAN": {
      "host": "DietPi.local",
      "username": "root",
      "password": "dietpi",
      "remotePath": "/mnt/dietpi_userdata/"
    },
    "DietPi LAN statisch": {
      "host": "192.168.0.100",
      "username": "root",
      "password": "dietpi",
      "remotePath": "/mnt/dietpi_userdata/"
    },
    "DietPi WLAN statisch": {
      "host": "192.168.4.1",
      "username": "root",
      "password": "dietpi",
      "remotePath": "/mnt/dietpi_userdata/"
    },
    "Raspian Peer-to-Peer oder WLAN": {
      "host": "raspberrypi.local",
      "username": "pi",
      "password": "raspberry",
      "remotePath": "/home/pi/"
    }
  }
}
```



4 SSHExtension benutzen

VSCode neu starten, falls noch nicht geschehen.

Magische Tastenkombination „Strg+Shift+P“ verwenden, um die Befehlsliste zu öffnen und das richtige Profil auswählen (SSHExtension: Open SSH Connection). 2x Enter. Passwort: „dietpi“.

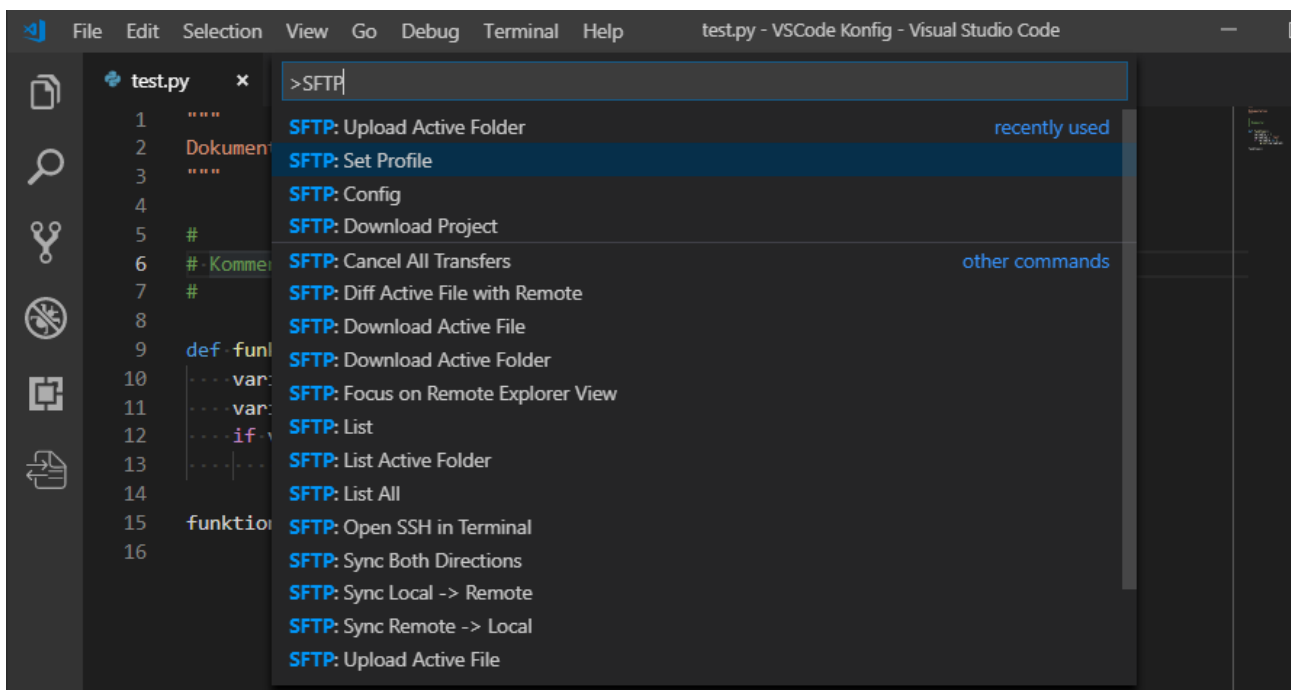


Achtung: Die OpenCV-Preview (imshow) funktioniert aus dieser Shell heraus nicht, da OpenCV dann keine grafische Benutzeroberfläche findet, um das Fenster anzuzeigen. Skripte, die imshow verwenden, müssen immer aus einer Shell im virtuellen Desktop (VNC) gestartet werden.

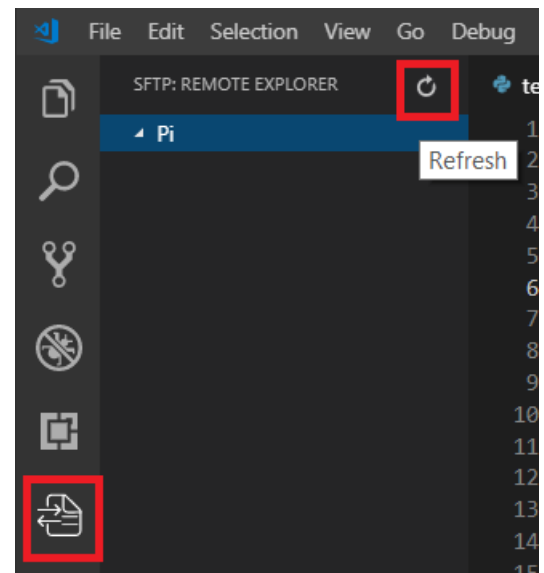
5 sftp benutzen

VSCode neu starten, falls noch nicht geschehen.

Magische Tastenkombination „Strg+Shift+P“ verwenden, um die Befehlsliste zu öffnen und das richtige Profil auswählen: „SFTP: Set Profile“ (sollte aber nicht nötig sein, wenn die Einstellung „defaultProfile“ schon auf das richtige Profil gesetzt war:



Mit dem Pi verbinden und den „SFTP“ Tab aktualisieren:

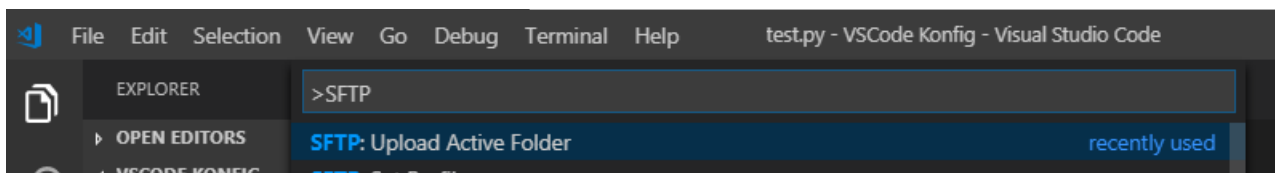


Hier sollten dann alle Dateien auftauchen, die unter „remotePath“ liegen.

Achtung: Die Dateien, die über dieses Menü geöffnet werden, sind nicht editierbar! Dateien immer über den „Explorer“ Tab öffnen.

Änderungen sollten beim Speichern nun automatisch auf den Pi kopiert werden.

Achtung: Wird eine neue Datei im Workspace hinzugefügt, kann es sein, dass die „sftp“-Extension das nicht frisst und die Datei auch beim Speichern nicht auf dem Pi landet. In dem Fall muss der Ordner einmal manuell synchronisiert werden. Probiert mal „Upload Active Folder“ oder „Upload Project“



Dasselbe gilt für neu erstellte Dateien auf dem Pi. Dafür gibt es einen Download-Befehl („Download Active Folder“ oder „Download Project“).

„Open SSH in Terminal“ öffnet eine Shell auf dem Pi. Vielleicht funktioniert das besser als die Shell, die SSHExtension bereit stellt.

6 Python Linter benutzen

Damit eine Datei als Python-Datei erkannt wird, muss sie die Dateierweiterung „.py“ aufweisen. Der Linter läuft immer, wenn man speichert. Fehler werden rot unterstrichen, Warnungen grün. Hält man die Maus darüber, bekommt man eine mehr oder weniger sinnvolle Fehlermeldung. Die Zeile, in der der Fehler auftritt, stimmt meistens. Es kann aber auch sein, dass es ein Folgefehler aus der vorherigen Code-Zeile ist.

Über das Menü View → Problems kann eine Übersicht aller Fehler angezeigt werden:


```
9 def funktion():
10     ... variable = 1
11     ... variable2 = 'text'
12     ... if variable is 1:
13         ... print(nicht_existent)
14
15 funktion()
16
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL Filter.

test.py 2

- [pylint] Undefined variable 'nicht_existent' [E0602] (13, 15)
- [pylint] Unused variable 'variable2' [W0612] (11, 5)

Achtung: Code, der Libraries benutzt, die sich nur auf dem Pi befinden (z.B. RPI.GPIO), wird natürlich immer als Fehler angesehen, da die Library auf dem Host-PC nicht gefunden werden kann.

7 Python Refactoring benutzen

In diesem Beispiel: Rechtsklick auf irgendeine Stelle, an der „variable“ benutzt wird → „Rename Symbol“:

```
def funktion():
    ... variable = 1
    ... variable2 = 'text'
    ... if var
    ... ..pr

funktion()
```

- Go to Definition F12
- Peek Definition Alt+F12
- Find All References Shift+F12
- List All References Shift+Alt+F12
- Rename Symbol F2**
- Change All Occurrences Ctrl+F2

Ich habe die Variable „umbenannt“ genannt. Danach sind alle Vorkommnisse von „variable“ in „umbenannt“ umbenannt worden.

```
9 def funktion():
10     ... umbenannt = 1
11     ... variable2 = 'text'
12     ... if umbenannt is 1:
13         ... print(variable2)
14
```

Wenn das nicht klappt, darauf achten, dass unten rechts im Fenster vllt. ein Hinweis auftaucht, der quittiert werden will. Danach sollte es gehen.

8 Python Debugging

Es ist möglich, den auf dem Pi laufenden Code auf dem Host-PC zu debuggen, d.h. Zeile für Zeile durch das Programm zu schreiten, sich die momentanen Werte von Variablen anzeigen zu lassen usw.

Under construction.

Bis dahin müssen wir uns mit Debug-Code begnügen, z.B.:

```
9 def pausieren(variable):
10     ... input("Variable hat den Wert '%s'. Weiter mit ENTER..." % (variable))
11
12 def funktion():
13     ... umbenannt = 1
14     ... pausieren(umbenannt)
15     ... variable2 = 'text'
16     ... pausieren(variable2)
17     ... if umbenannt is 1:
18         ...     pausieren("")
19     ...     print(variable2)
20
21 funktion()
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell

Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

PS C:\Users\Tom\Dropbox\Robocup 2018_2019\Material\RaspberryPi Admin\Software Host\VSCode Konfig> python test.py

Variable hat den Wert '1'. Weiter mit ENTER...

Variable hat den Wert 'text'. Weiter mit ENTER...

Variable hat den Wert ''. Weiter mit ENTER...

text

PS C:\Users\Tom\Dropbox\Robocup 2018_2019\Material\RaspberryPi Admin\Software Host\VSCode Konfig> □