UNIVERSITY of DEBRECEN

Dr. József Sütő
Department of IT Systems and Networks

Embedded Systems Laboratory

Lab 2: Introduction into Python programming

Project overview

In this laboratory you will start writing Python programs on Raspberry Pi. Python is the official language for Pi and it is pre-installed on the Raspberry OS. This laboratory content assumes that you have studied at least one programming languages (e.g. C/C++, Java, C#) earlier.

The following topics will be covered in this lab:

- Python tools for Raspberry Pi
- Using the Python command line
- Writing Python programs

Technical requirements

The following components are required to complete this lab:

- Raspberry Pi 3 Model B+ (MicroSD card, power supply, keyboard, mouse)

1. Introduction into Python

- Python is a powerful programming language that is widely used commercially
- Todays it is the most popular programming language on Pi
- Two versions: Python 2 and Python 3 (they are not compatible!)
- Available integrated development environments on the Pi: IDLE, Thonny
- Any text editor (e.g. nano) also can be used for Python programming
- Python is case-sensitive
- **Code blocks are identified according to their indent!!**

1.1. List

- One of the most fundamental possibility to organize your information
- Empty list: **mylist = []**

- Length of the list: **len()**
- Add item to the list: **append()**
- x'th item in the list: **mylist[x]**
- Additional list operations:

Action	**Code**	**Notes**
Sort a list	mylist.sort()	Sorts the list in alphabetical order from low to high
Sort a list in reversed order	mylist.sort(reverse=True)	Sorts in reversed alphabetical order
Delete a list element	del mylist[2]	Delete a list element with the index number specified. List items after it move up the list – no gap
Remove an item from list	mylist.remove("xyz")	Delete a list item that match the parameter. Return with an error if is not exist

1.2. Dictionary

- Another fundamental data structure
- You can access an item using its key
- Dictionary elements definition inside curly brackets: **{key1: value1, key2: value2, …}**
- Example: *feeling = {"happy": "I am happy today!", "sad": "This day is terrible!"}*

1.3. Functions

- A function can receive some information from the rest of the program (zero, one or more *arguments*), work on it, and then send back the result
- Functions enable you to reuse parts of your program
- Functions make easier to update and understand your program
- Function definition: **def *function_name* (*arguments*):**
- Unlike other languages, an argument can be a function
- Arguments may have a default value

- Example: *def grammar_check(message):*

  return „Your message is error free!”

- Observe that the content of the function is shifted by 1 tab relative to the function definition

1.4. Classes

- Python is an object-oriented language
- To create a class, use the **class** keyword: **class Myclass:**
- All classes have a built-in **__init__()** function which is executed at the beginning when the class is initiated (constructor)
- By the **__init__()** function we can assign values to object properties
- An object can contain methods (function that belong to the object)
- The **self** parameter is a reference to the current instance of the class
- **self** can be used to access variables and methods that belong to the class
- It has to be the first parameter of any instance’s method in the class
- You can use any other word instead of self but it is the most commonly used
- Example:

```
class Person:
        def __init__(self, name, age):
                self.name = name
                self.age = age

        def my_func(self):
                print(“My name is ” + self.name )
```

2. Python programming in the Terminal window

An easy way to access the Python interpreter is the Terminal window.

2.1. Determine current Python version:

```
$ python3 -V
```

2.2. Launch Python in terminal

```
$ python3
```

2.3. Import the **datetime** object from the datetime module:

```
>>> from datetime import datetime
```

2.4. Print out the current time and date

```
>>> print(datetime.now())
```

2.5. Exit from the Python interpreter

```
>>> exit()
```

3. Writing Python scripts in text editor

Write a simple Python script in the **nano** text editor and run the script in Terminal

3.1. Create an empty file with **nano** text editor:

```
$ nano test.py
```

3.2. Define the content of the script:

```
a = 10

b = 3

c = a * b

print(c)
```

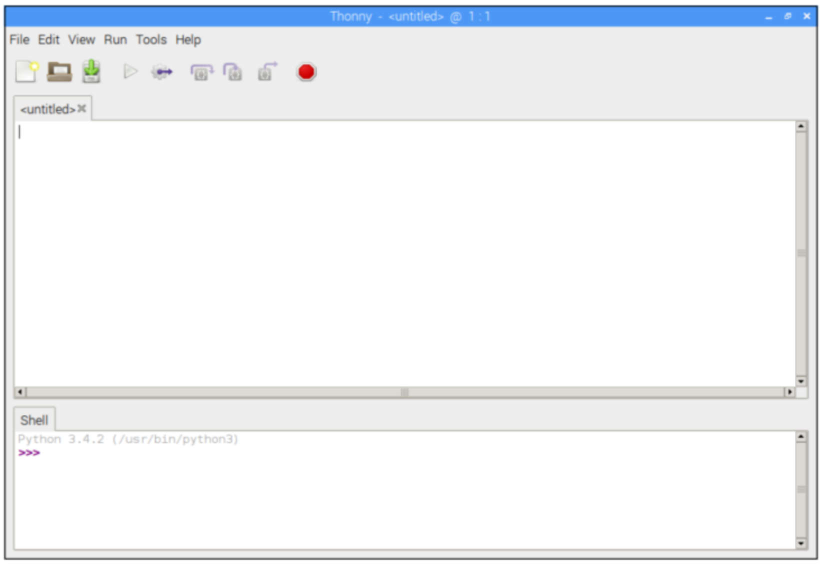
3.3. Exit from nano by CTRL+X and approve changes.

3.4. Run the script:

```
$ python3 test.py
```

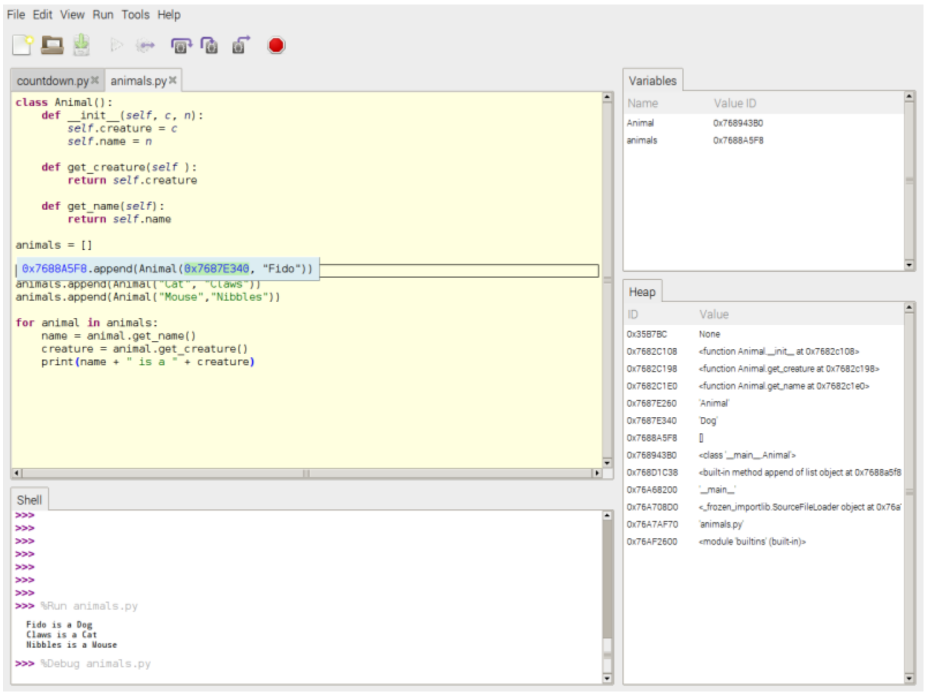
4. The Thonny integrated development environment (IDE)

In most cases, using an IDE is much more convenient than using a simple text editor. The Thonny is a popular, preinstalled IDE in the Raspberry OS which can be found in *Menu > Programming*. When you start Thonny, you will see a similar window as can be seen below. The two main fields of the window are the script editor and the shell. The code in the editor will run in the shell. You can use the shell to interact directly with the program. For example, accessing variables of objects.



```
Python 3.4.2 (/usr/bin/python3)
>>>
```

Optionally, it also has additional panels where the programmer can see variables, objects, and the content of the heap (memory space).



```
class Animal():
    def __init__(self, c, n):
        self.creature = c
        self.name = n

    def get_creature(self ):
        return self.creature

    def get_name(self):
        return self.name

animals = []

0x7688A5F8.append(Animal(0x7687E340, "Fido"))
animals.append(Animal("Cat", "Claws"))
animals.append(Animal("Mouse","Nibbles"))

for animal in animals:
    name = animal.get_name()
    creature = animal.get_creature()
    print(name + " is a " + creature)
```

```
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>> %Run animals.py

  Fido is a Dog
  Claws is a Cat
  Nibbles is a Mouse

>>> %Debug animals.py
```

Name	Value ID
Animal	0x768943B0
animals	0x7688A5F8

ID	Value
0x35B7BC	None
0x7682C108	<function Animal.__init__ at 0x7682c108>
0x7682C198	<function Animal.get_creature at 0x7682c198>
0x7682C1E0	<function Animal.get_name at 0x7682c1e0>
0x7687E260	'Animal'
0x7687E340	'Dog'
0x7688A5F8	[]
0x768943B0	<class '__main__.Animal'>
0x768D1C38	<built-in method append of list object at 0x7688a5f8
0x76A68200	'__main__'
0x76A708D0	<_frozen_importlib.SourceFileLoader object at 0x76a
0x76A7AF70	'animals.py'
0x76AF2600	<module 'builtins' (built-in)>

5. Multiplication table

Write a multiplication table program in the **Thonny IDE**. The program asks the user which table to generate and show the appropriate table on the screen

5.1. Open the Thonny IDE

5.2. Print out the what is the functionality of this program

```
print('This program calculates times table')
```

5.3. Read the table number from the user:

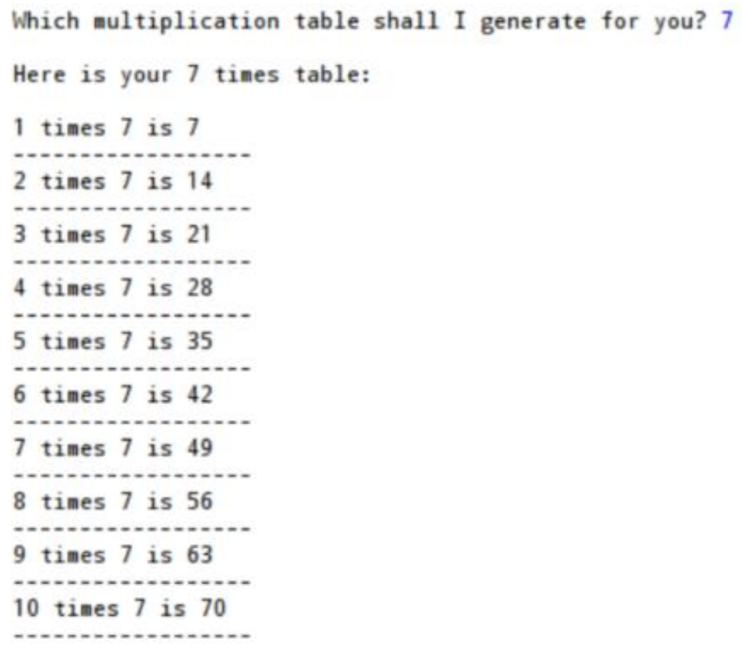
```
tablenum = input('\nWhich multiplication table shall I generate for you?')
```

5.4. Convert *tablenum* to **int**:

```
tablenum = int(tablenum)
```

5.5. Generate and print out the appropriate table in a for loop:

```
print("\nHere is your", tablenum, "times table:\n")
for i in range(1, 11):
        print(i, "times", tablenum, "is", i * tablenum)
        print("------------------")
```



```
Which multiplication table shall I generate for you? 7

Here is your 7 times table:

1 times 7 is 7
------------------
2 times 7 is 14
------------------
3 times 7 is 21
------------------
4 times 7 is 28
------------------
5 times 7 is 35
------------------
6 times 7 is 42
------------------
7 times 7 is 49
------------------
8 times 7 is 56
------------------
9 times 7 is 63
------------------
10 times 7 is 70
------------------
```

Tasks to be solved alone:

5.A. Modify the program to show only the odd lines until line 13 in the table!

6. Oversimplified chatbot

Now you will create an oversimplified chatbot program. The program reads sentences from the user and tries to respond "relevant" sentences.

6.1. Import the **random** module:

```
import random
```

6.2. Create a list with random replies:

```
random_replies = ["Oh really?", "Are you sure about that?",
"Perhaps…", "I don't think so"]
```

6.3. Create a chat dictionary:

```
chat_dict = {"happy": "I'am happy today too", "sad": "Be happy,
life is good", "computer": "Computers will take over the world!"}
```

6.4. Define a function which performs the following tasks:

- Convert the message to lowercase
- Spilt the message into words
- If a word of the message is inside the keys of the dictionary, add the value (sentence) to the *smart_replies* list
- If the replies list is not empty return with a randomly selected item, else return with an empty string

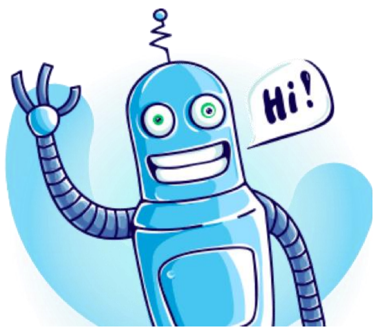
```
def dictionary_check(message):
        message = message.lower()
        user_words = message.split()
        smart_replies = []
        for word in user_words:
                if word in chat_dict:
                        answer = chat_dict[word]
                smart_replies.append(answer)
        if smart_replies:
            reply_chosen = random.randint (1, len(smart_replies)) - 1
            return smart_replies[reply_chosen]
        else:
            return ""
```

6.5. Implement the conversation cycle which is stay alive while the user is not saying “bye”. In the cycle the software reads message from the user and it responds with a “smart” or random response depending on the content of the message

```
    print("What would you like to talk about?")
    user_says = ""
    while user_says != "bye":
        user_says = ""
        while user_says == "":
            user_says = input("Talk to me: ")
        smart_response = dictionary_check(user_says)
        if smart_response:
            print(smart_response)
        else:
            reply_chosen = random.randint (1, len(random_replies)) - 1
            print(random_replies[reply_chosen])
            random_replies[reply_chosen] = user_says
    print("Goodbye. Thanks for chatting today!")
```

Task to be solved alone:

6.A. Improve the chatbot by adding more random and chat sentences!



7. Implement your first class

In this example you will create a class called *CurrentWeather* that holds information about the weather in different cities.

7.1. Implement the *CurrentWeather* class

```
class CurrentWeather:
    weather_data={'Toronto':['13','partly sunny','8 km/h NW'],
                  'Montreal':['16','mostly sunny','22 km/h W'],
                  'Vancouver':['18','thunder showers','10 km/h NE'],
                  'New York':['17','mostly cloudy','5 km/h SE'],
                  'Los Angeles':['28','sunny','4 km/h SW'],
                  'London':['12','mostly cloudy','8 km/h NW'],
                  'Mumbai':['33','humid and foggy','2 km/h S'] }
    def __init__(self, city):
        self.city = city
    def get_temperature(self):
        return self.weather_data[self.city][0]
    def get_weather_conditions(self):
        return self.weather_data[self.city][1]
    def get_wind_speed(self):
        return self.weather_data[self.city][2]
```

7.2. Click to the Run Current Script button to load the code into the Python interpreter

7.3. In the shell command line, create and object from the class

```
weather = CurrentWeather('London')
```

7.4. In the command line, call all of the 3 methods of the object one by one

```
weather.get_temperature()
weather.get_weather_conditions()
weather.get_wind_speed()
```

7.5. Launch the **Object inspector** and **Variables** from the View menu list. In the tables you can see the existing variables and their properties.

7.6. Add the following function to the class:

```
def get_city(self):
    return self.city
```

7.7. Add the following code snippet after the class definition. The first line should have the same indentation as the class definition because this function is independent to the class:

```
if __name__ == "__main__":
    weather = CurrentWeather('Toronto')
    wind_dir_str_len = 2            #length of the wind direction string
    if weather.get_wind_speed()[-2:-1] == ' ':
        wind_dir_str_len = 1
    print("The current temperature in",
        weather.get_city(),"is",
        weather.get_temperature(),
        "degrees Celsius,",
        "the weather conditions are",
        weather.get_weather_conditions(),
        "and the wind is coming out of the",
        weather.get_wind_speed()[-(wind_dir_str_len):],
        "direction with a speed of",
        weather.get_wind_speed()
        [0:len(weather.get_wind_speed()) - wind_dir_str_len])
```

Note: The ***if __name__ == "__main__"*** is true if the file is running directly.

Task to be solved alone:

7.A. Add an additional function to the CurrentWeather class which prints out the wind speed of a city without the wind direction (e.g. NW)!

8. Fibonacci numbers (task to be solved alone)

Write a Python program which uses a loop to print out the first 10 Fibonacci numbers (0, 1, 1, 2, 3, 5, 8, 13, 21, 34). In mathematical terms the sequence $\mathbf{F_n}$ of Fibonacci numbers is defined by the recurrent relation (seed values are $\mathbf{F_0 = 0, F_1 = 1}$):

$$\mathbf{F_n = F_{n-1} + F_{n-2}}$$

References

[1] McManus S, Cook, M. Raspberry Pi for Dummies, 3rd edition, Wiley, 2017.
[2] Monk, S. Raspberry Pi Cookbook, 2nd edition, O'Reilly, 2016.

This work was supported by the construction EFOP-3.4.3-16-2016-00021. The project was supported by the European Union, co-financed by the European Social Fund.