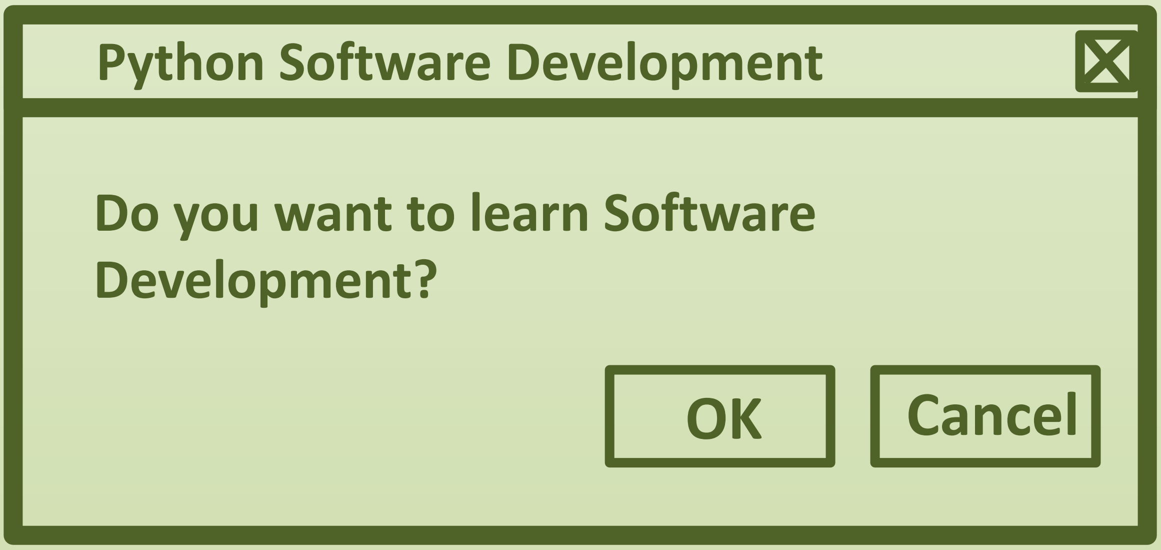
Python for Software Development

Hans-Petter Halvorsen


Python Software Development

Do you want to learn Software Development?

https://www.halvorsen.blog

python™

Python for Software Development

Python for Software Development

Hans-Petter Halvorsen

2020

Python for Science and Engineering

©Hans-Petter Halvorsen

September 9, 2020

Preface

Python is a popular programming language, and it is one of the most used programming languages today.

Python works on all the main platforms and operating systems used today, such Windows, macOS, and Linux.

Python is a multi-purpose programming language, which can be use for simulation, creating web pages, communicate with database systems, etc.

My Blog/Web Site [1]:
https://www.halvorsen.blog

Here you find lots of technical resources about Technology, Programming, Software Engineering, Automation and Control, Industrial IT, etc.

Here you find my Web page with Python resources:

https://www.halvorsen.blog/documents/programming/python/

These resources are a supplement to this textbook. Here you can download the software, download code examples, etc.

This Textbook is written in LaTeXusing Overleaf.

LaTeXis a document preparation system used for the communication and publication of scientific documents.

For more information about LaTeX:
https://www.latex-project.org

Overleaf is a web-bases LaTeXsystem, meaning you can write your LaTeXdocuments in your web browser, you co-work and share documents with others.

For more information about Overleaf:
https://www.overleaf.com

Python Books

You find other Python textbooks within different domains on my Python Web page:
https://www.halvorsen.blog/documents/programming/python/

Python Books:

- **Python Programming** - This is a textbook in Python Programming with lots of Practical Examples and Exercises. You will learn the necessary foundation for basic programming with focus on Python.
- **Python for Science and Engineering** - This is a textbook in Python Programming with lots of Examples, Exercises, and Practical Applications within Mathematics, Simulations, etc. The focus is on numerical calculations in mathematics and engineering. Necessary theory is presented in addition to many practical examples.
- **Python for Control Engineering** - This is a textbook in Python Programming with lots of Examples, Exercises, and Practical Applications within Mathematics, Simulations, Control Systems, DAQ, Database Systems, etc. The focus is on the use of Python within measurements, data collection (DAQ), control technology, both analysis of control systems (stability analysis, frequency response, ...) and implementation of control systems (PID, etc.). Required theory is presented in addition to many practical examples and exercises in Python.
- **Python for Software Development** - This is a textbook in Python Programming with lots of Examples, Exercises, and Practical Applications within Software Systems, Software Development, Software Engineering, Database Systems, Web Application Desktop Applications, GUI Applications, etc. The focus is on the use of Python for creating modern Software Systems. Required theory is presented in addition to many practical examples and exercises in Python.

Programming

The way we create software today has changed dramatically the last 30 years, from the childhood of personal computers in the early 80s to today's powerful devices such as Smartphones, Tablets and PCs.

The Internet has also changed the way we use devices and software. We still have traditional desktop applications, but Web Sites, Web Applications and so-called Apps for Smartphones, etc. are dominating the software market today.

We need to find and learn Programming Languages that are suitable for the New Age of Programming.

We have today several thousand different Programming Languages today. I guess you will need to learn more than one Programming Language to survive in today's software market.

You find lots of Programming Resources here:
https://www.halvorsen.blog/documents/programming/

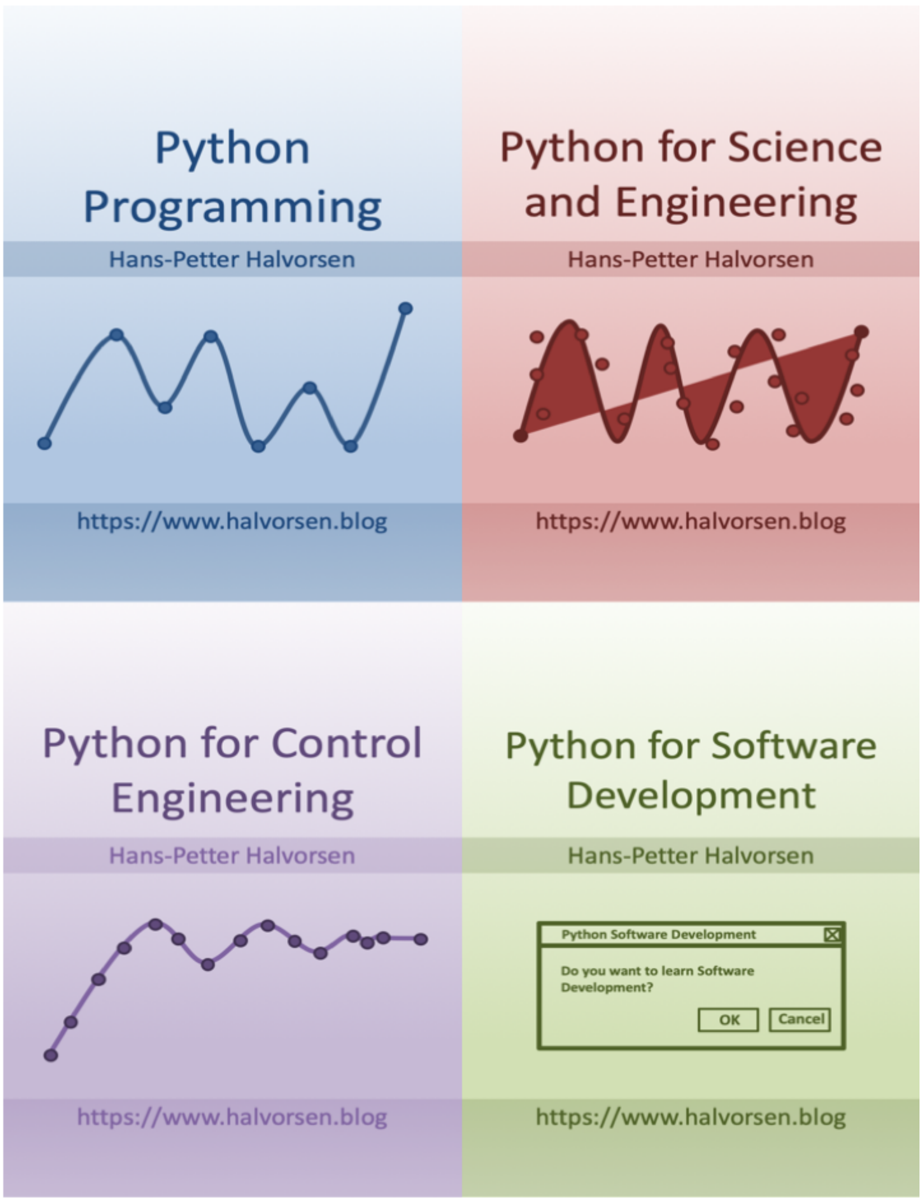
Software Engineering

Software Engineering is the discipline for creating software applications. A systematic approach to the design, development, testing, and maintenance of software.

The main parts or phases in the Software Engineering process are:

- Planning
- Requirements Analysis
- Design
- Implementation
- Testing
- Deployment and Maintenance

You find lots of Software Engineering Resources here:
https://www.halvorsen.blog/documents/programming/software$_engineering/$


Python Programming

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Science and Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Control Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Software Development

Hans-Petter Halvorsen

Python Software Development

Do you want to learn Software Development?

OK Cancel

https://www.halvorsen.blog

Contents

Part I

Getting Started with Python

Chapter 1

Introduction

With this textbook you will learn basic Python programming. The textbook contains lots of examples and self-paced tasks that the users should go through and solve in their own pace.

You will find additional resources on my blog/web site [1].
https://www.halvorsen.blog

My Web Site about Python is:
https://www.halvorsen.blog/documents/programming/python/

See Figure 1.1

1.1 The New Age of Programming

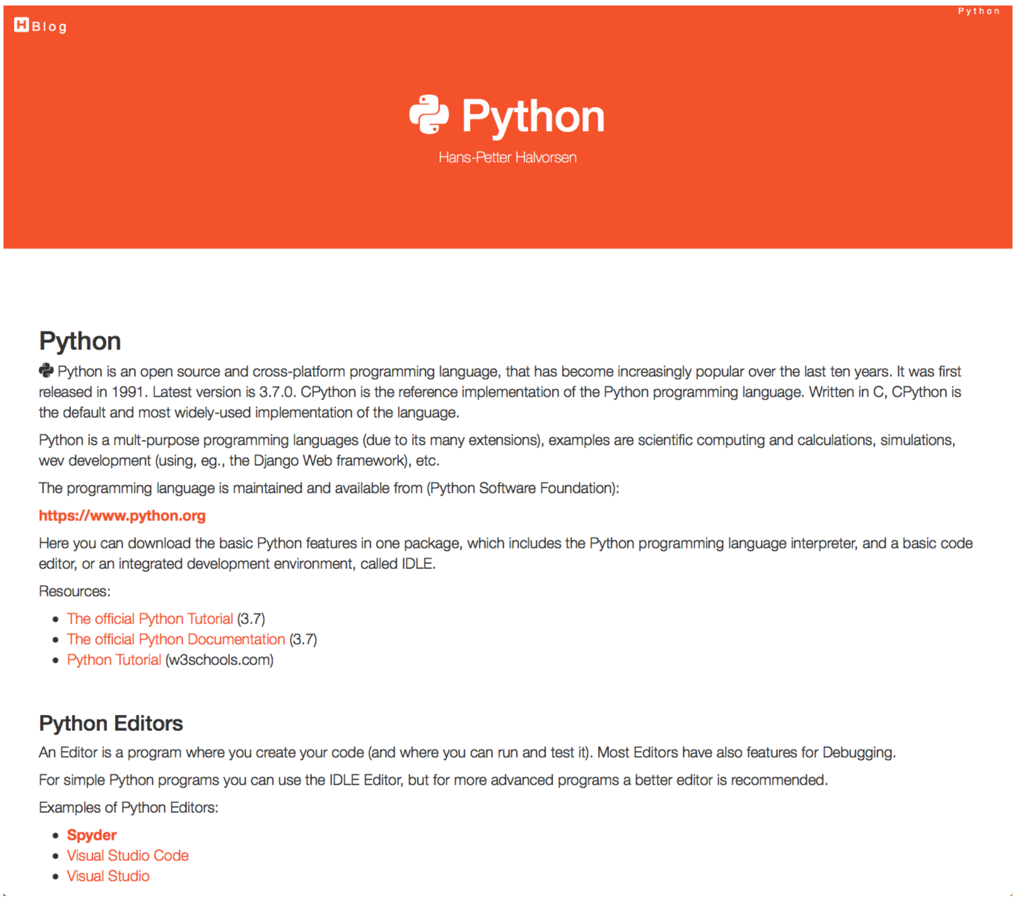
The way we create software today has changed dramatically the last 30 years, from the childhood of personal computers in the early 80s to today's powerful devices such as Smartphones, Tablets and PCs.

The Internet has also changed the way we use devices and software. We still have traditional desktop applications, but Web Sites, Web Applications and so-called Apps for Smartphones, etc. are dominating the software market today.

We need to find and learn Programming Languages that are suitable for the New Age of Programming.

We have today several thousand different Programming Languages, so why should we learn Python? I guess you will need to learn more than one Programming Language to survive in today's software market. Python is easy to learn, so it it a good starting point for new programmers.

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991 [2].



Python

Python is an open source and cross-platform programming language, that has become increasingly popular over the last ten years. It was first released in 1991. Latest version is 3.7.0. CPython is the reference implementation of the Python programming language. Written in C, CPython is the default and most widely-used implementation of the language.

Python is a mult-purpose programming languages (due to its many extensions), examples are scientific computing and calculations, simulations, wev development (using, eg., the Django Web framework), etc.

The programming language is maintained and available from (Python Software Foundation):

https://www.python.org

Here you can download the basic Python features in one package, which includes the Python programming language interpreter, and a basic code editor, or an integrated development environment, called IDLE.

Resources:

- The official Python Tutorial (3.7)
- The official Python Documentation (3.7)
- Python Tutorial (w3schools.com)

Python Editors

An Editor is a program where you create your code (and where you can run and test it). Most Editors have also features for Debugging.

For simple Python programs you can use the IDLE Editor, but for more advanced programs a better editor is recommended.

Examples of Python Editors:

- **Spyder**
- Visual Studio Code
- Visual Studio

Figure 1.1: Web Site - Python

Python is a fairly old Programming Language (1991) compared to many other Programming Languages like C# (2000), Swift (2014), Java (1995), PHP (1995).

Python has during the last 10 years become more and more popular. Today, Python has become one of the most popular Programming Languages.

There are many different rankings regarding which programming language which is most popular. In most of these ranking, Python is in top 10.

One of these rankings is the IEEE Spectrum's ranking of the top programming languages [3].

From this ranking we see that Python is the most popular Programming Language in 2018. See Figure 1.2
As we see in Figure 1.2 they categorize the different Programming Languages into the following categories:

- Web

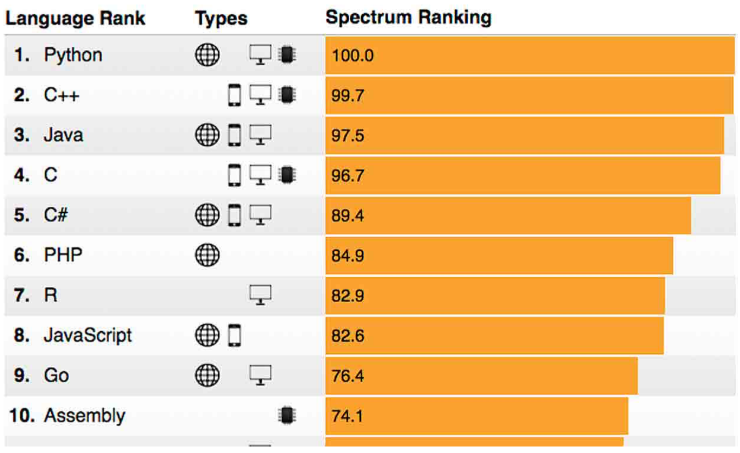


Figure 1.2: The Most Popular Programming Languages

- Mobile
- Enterprise
- Embedded

According to Figure 1.2 we see that Python can be used to program Web Applications, Enterprise Applications and Embedded Applications.

So far Python is not used or not optimized for creating Mobile Applications. We have today 2 major Mobile platforms; iOS Applications are mainly programmed with the Swift Programming language, while Android Applications are mainly programmed with either Java or Kotlin.

Another survey is the ”Stack Overflow Developer Survey 2018” [4]. See Figure 1.3.

As we can see from [5] and Figure 1.4, Python becomes more and more popular year by year.

Based on Figure 1.4, the source [5] try to predict the future of Python, see Figure 1.5.

Based on the surveys and statistics mention above, obviously Python is a programming language that you should learn.

Lets summarize:

- Python is fun to learn and use and it is also named after the British comedy group called Monty Python.
- Python has a simple and flexible code structure and the code is easy to read.

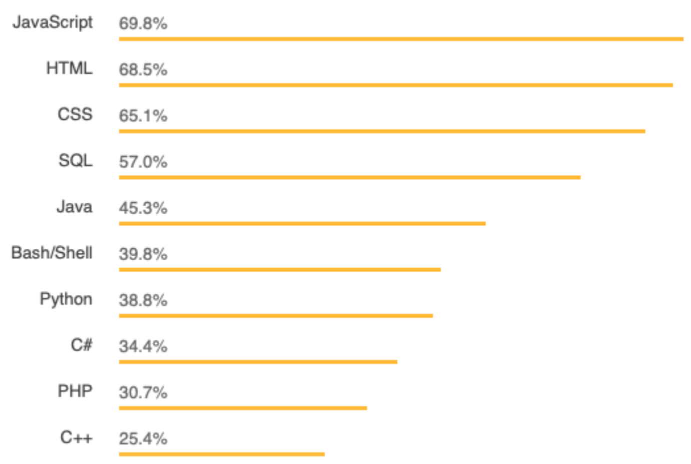


Figure 1.3: The Top Programming Languages - Stack Overflow Survey

- Python is highly extendable due to its high number of free available Python Packaged and Libraries
- Python can be used on all platforms (Windows, macOS and Linux).
- Python is multi-purpose and can be used for to program Web Applications, Enterprise Applications and Embedded Applications, and within Data Science and Engineering Applications.
- The popularity of Python is growing fast.
- Python is open source and free to use
- The growing Python community makes it easy to find documentation, code examples and get help when needed

In general, Python is a multipurpose programming language that can be used in many situations. But there is not one programming language which is best in all kind of situations, so it is important that you know about and have skills in different languages.

My list of recommendations (one of many):

- Visual Studio and C
- LabVIEW - a graphical programming language well suited for hardware integration, taking measurements and data logging
- MATLAB - Numerical calculations and Scientific computing
- Python - Numerical calculations, and Scientific computing, etc.
- Web Programming, such as HTML, CSS, JavaScript and a Server-side framework/programming language like PHP, ASP.NET (C or VB.NET), Django (Python based)

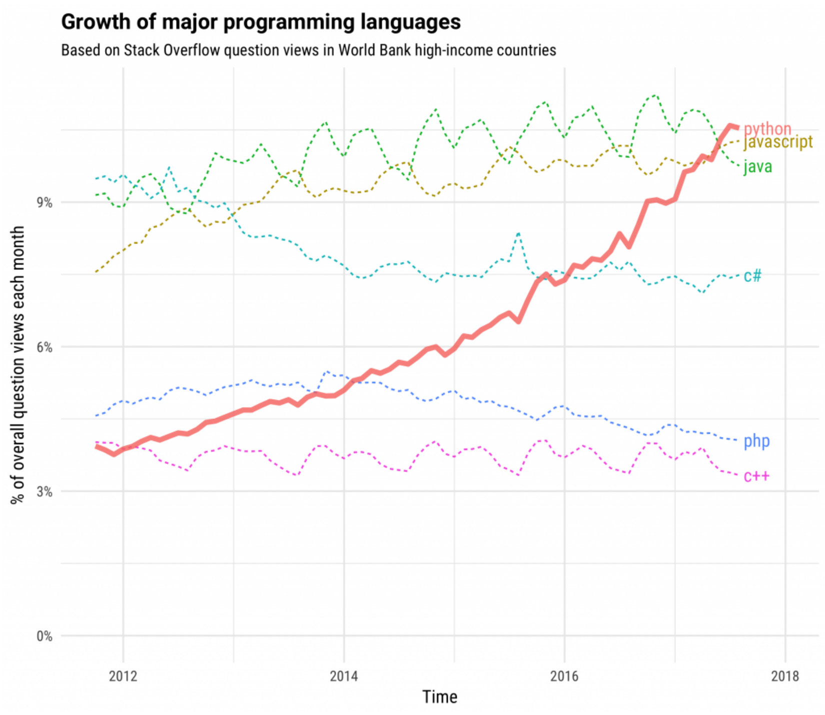


Figure 1.4: The Incredible Growth of Python

- Databases (such as SQL Server and MySQL) and using the Structured Query Language (SQL) or the upcoming NoSQL databases
- App Development for the 2 main platforms iOS (XCode using the Swift Programming Language) and Android (Android Studio using the Java Programming Language or Kotlin Programming language)

If you have skills in most of the tools, programming languages and frameworks mention above, you are well suited for working as a full-time programmer or software engineer.

1.2 MATLAB

If you are looking for MATLAB, please see the following:
https://www.halvorsen.blog/documents/programming/matlab/

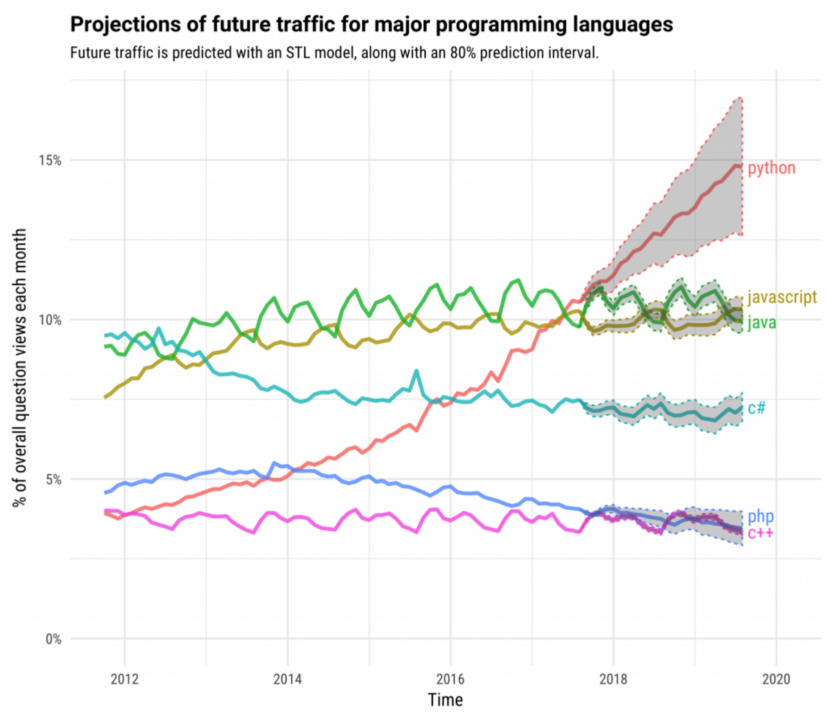


Figure 1.5: The Future of Python

Chapter 2

What is Python?

2.1 Introduction to Python

Python is an open source and cross-platform programming language, that has become increasingly popular over the last ten years. It was first released in 1991. Latest version is 3.7.0. **CPython** is the reference implementation of the Python programming language. Written in C, CPython is the default and most widely-used implementation of the language.

Python is a multi-purpose programming languages (due to its many extensions), examples are scientific computing and calculations, simulations, web development (using, e.g., the Django Web framework), etc.

Python Home Page [6]:
https://www.python.org

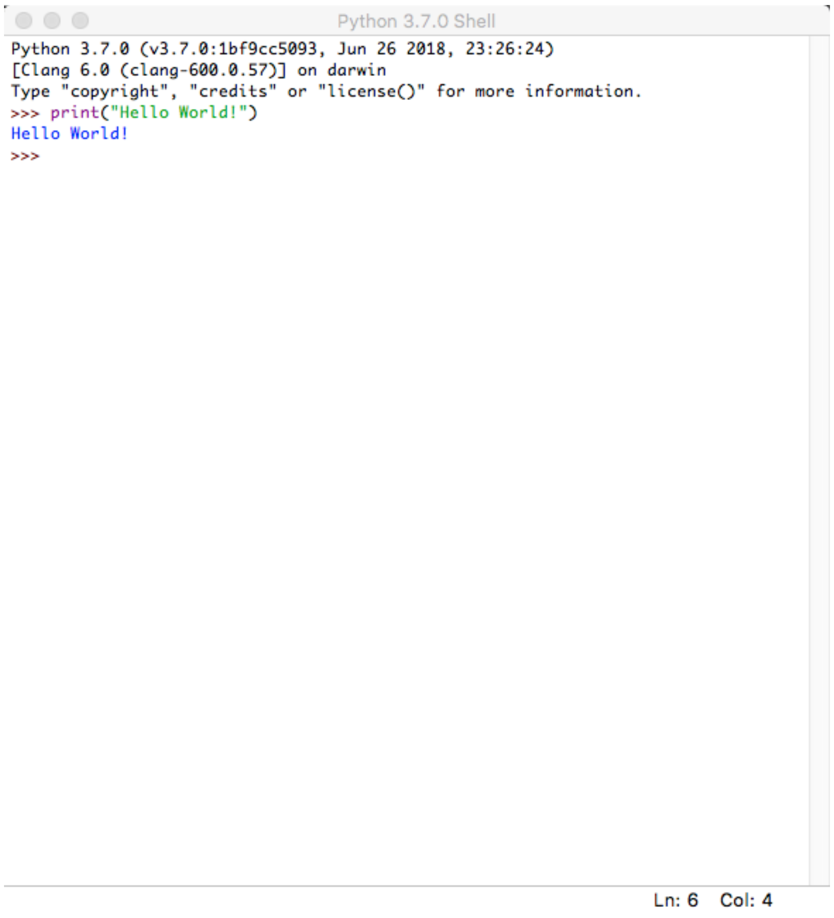
The programming language is maintained and available from (Python Software Foundation): https://www.python.org Here you can download the basic Python features in one package, which includes the Python programming language interpreter, and a basic code editor, or an integrated development environment, called IDLE. See Figure 2.1

But this is just the Python core, i.e. the interpreter a very basic editor, and the minimum needed to create basic Python programs.

Typically you will need more features for solving your tasks. Then you can install and use separate Python packages created by third parties. These packages need to be downloaded and installed separately (typically you use something called PIP), or you choose to use, e.g., a distribution package like Anaconda.

Python is an object-oriented programming language (OOP), but you can use Python in basic application without the need to know about or use the object-oriented features in Python.

Python is an interpreted programming language, this means that as a developer



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello World!")
Hello World!
>>>
```

Figure 2.1: IDLE - Basic Python Editor

you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed. Depending on the Editor you are using, this is either done automatically, or you need to do it manually.

Here are some important Python sources: [6], [7], [8].

2.1.1 Interpreted vs. Compiled

What are the differences between Interpreted programming languages and Compiled programming languages? What kind should you choose, and why should you bother?

Programming languages generally fall into one of two categories: Compiled or Interpreted. With a compiled language, code you enter is reduced to a set of machine-specific instructions before being saved as an executable file.
Both approaches have their advantages and disadvantages.

With interpreted languages, the code is saved in the same format that you entered. Compiled programs generally run faster than interpreted ones because interpreted programs must be reduced to machine instructions at run-time. It is usually easier to develop applications in an interpreted environment because you don't have to recompile your application each time you want to test a small section.

Python is an interpreted programming language, while e.g., C/C++ are translated by running the source code through a compiler, i.e., C/C++ are compiled languages.

Interpreted languages, in contrast, must be parsed, interpreted, and executed each time the program is run.

Another example of an interpreted programming language is PHP, which is mainly used to create dynamic web pages and web applications.

Compiled languages are all translated by running the source code through a compiler. This results in very efficient code that can be executed any number of times. The overhead for the translation is incurred just once, when the source is compiled; thereafter, it need only be loaded and executed.

During the design of an application, you might need to decide whether to use a compiled language or an interpreted language for the application source code.

Interpreted languages, in contrast, must be parsed, interpreted, and executed each time the program is run

Thus, an interpreted language is generally more suited for doing "ad hoc" calculations or simulations, while compiled languages are better for permanent applications where speed is in focus.

2.2 Python Packages

With Python you don't get so much out of the box. Instead of having all of its functionality built into its core, you need to install different packages for different topics.

This approach has advantages and disadvantages. An disadvantage is that you need to install these packages separately and then later import these modules in your code.

This is also typical approach for open source software, because everybody can create their own Python packages and distribute them. In that way you also find Python packages for almost everything, from Scientific Computing to Web Development.

These packages need to be downloaded and installed separately, or you choose to use, e.g., a distribution package like Anaconda, where you typically get the packages you need for scientific computing. With Anaconda you typically get the same features as with MATLAB.

Lots of Python packages exists, depending on what you are going to solve.
We have Python packages for Desktop GUI Development, Database Development, Web Development, Software Development, etc.

See an overview of Applications for Python:
https://www.python.org/about/apps/

See also the Python Package Index (PyPI) web site:
https://pypi.org

Here you can search for, download and install many hundreds Python Packages within different topics and applications. You can also make your own Python Packages and distribute them here.

2.2.1 Python Packages for Science and Numerical Computations

Some important Python Packages for Science and Numerical Computations are:

- **NumPy** - NumPy is the fundamental package for scientific computing with Python [9]
- **SciPy** - SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering. [9]
- **Matplotlib** - Matplotlib is a Python 2D plotting library. [10]
- **Pandas** - Pandas Python Data Analysis Library [11]

These packages need to be downloaded and installed separately, or you choose to use, e.g., a distribution package like Anaconda, where you typically get the packages you need for scientific computing. With Anaconda you typically get the same features as with MATLAB.

2.3 Anaconda

Anaconda is a distribution package, where you get Python compiler, Python packages and the Spyder editor, all in one package.

Anaconda includes Python, the Jupyter Notebook, and other commonly used packages for scientific computing and data science.

They offer a free version (Anaconda Distribution) and a paid version (Enterprise) Anaconda is available for Windows, macOS, and Linux

Web:
https://www.anaconda.com

Wikipedia:
https://en.wikipedia.org/wiki/Anaconda$_(Python_distribution)$

Spyder and the Python packages (NumPy, SciPy, Matplotlib, ...) mention above +++ are included in the Anaconda Distribution.

2.4 Python Editors

An Editor is a program where you create your code (and where you can run and test it). Most Editors have also features for Debugging. For simple Python programs you can use the IDLE Editor, but for more advanced programs a better editor is recommended.

Examples of Python Editors:

- Python IDLE
- Visual Studio Code
- Spyder
- Visual Studio
- PyCharm
- Wing Python IDE
- Jupyter Notebook

These editors are shortly described below and in more detail later in this textbook.

Which editor you should use depends on your background, what kind of code editors you have used previously, your programming skills, what your are going to develop in Python, etc.

2.4.1 Python IDLE

The programming language is maintained and available from (Python Software Foundation): https://www.python.org Here you can download the basic Python features in one package, which includes the Python programming language interpreter, and a basic code editor, or an integrated development environment, called IDLE. See Figure 2.1

Web:
https://www.python.org

2.4.2 Visual Studio Code

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux and macOS.

Web:
https://code.visualstudio.com

Resources: Getting Started with Python in Visual Studio Code

2.4.3 Spyder

Spyder is an open source cross-platform integrated development environment (IDE) for scientific programming in the Python language.

Web:
https://www.spyder-ide.org

Wikipedia:
https://en.wikipedia.org/wiki/Spyder$_(software)$

Spyder is included in the Anaconda Distribution.

2.4.4 Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. The deafult (main) programming language in Visual studio is C, but many other programming languages are supported.

Visual studio is available for Windows and macOS.

Visual Studio (from 2017), has integrated support for Python, it is called "Python Support in Visual Studio".

Web:
https://visualstudio.microsoft.com

Wikipedia:
https://en.wikipedia.org/wiki/Microsoft$_Visual_Studio$

2.4.5 PyCharm

PyCharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is free to use, while the Professional Edition (paid version) has some extra features.

Web:
https://www.jetbrains.com/pycharm/

2.4.6 Wing Python IDE

The Wing Python IDE family of integrated development environments (IDEs) from Wingware were created specifically for the Python programming language.

3 different version of Wing exists [12]:

- **Wing 101** – a very simplified free version, for teaching beginning programmers
- **Wing Personal** – free version that omits some features, for students and hobbyists
- **Wing Pro** – a full-featured commercial (paid) version, for professional programmers

2.4.7 Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and text.

Web:
http://jupyter.org

Wikipedia:
https://en.wikipedia.org/wiki/Project$_Jupyter$

2.5 Resources

Here are some useful Python resources:

- The official Python Tutorial
  - https://docs.python.org/3.7/tutorial/index.html
- The official Python Documentation
  - https://docs.python.org/3.7/index.html
- Python Tutorial (w3schools.com) [13]
  - https://www.w3schools.com/python/

2.6 Installing Python

The Python programming language is maintained and available from (Python Software Foundation):

https://www.python.org

Here you can download the basic Python features in one package, which includes the Python programming language interpreter, and a basic code editor, or an integrated development environment, called IDLE. See Figure 2.1

For basic Python programming this is good enough.

For more advanced Python Programming you typically need a better Code Editor and additional Packages.

For the basic Python examples in the beginning, the basic Python software from:
https://www.python.org is good enough.

I suggest you start with the basic Python software in order to learn the basics, then you can upgrade to a better Editor, install addition Python packages (either manually or or install Anaconda where "everything" is included).

2.6.1 Python Windows 10 Store App

Python 3.7 is also available in the Microsoft Store for Windows 10.

The Microsoft Store version of Python 3.7 is a simplified installer for running scripts and packages.

Microsoft Store version of Python 3.7 is very basic but it's good enough to run the simple scripts.

Python 3.7 Microsoft Store edition will receive all updates automatically when they are released and no manual action is required from your end.

In order to install the Microsoft Store version of Python just open Microsoft Store in Windows 10 and search for Python.

2.6.2 Installing Anaconda

The Spyder Code Editor and the Python packages (such as NumPy, SciPy, matplotlib, etc) are included in the Anaconda Distribution.

Download and install from:
https://www.anaconda.com

2.6.3 Installing Visual Studio Code

Visual Studio Code code is a simple and easy to use editor that can be used for many different programming languages.

Download and install from:
https://code.visualstudio.com

Getting Started with Python in Visual Studio Code:
https://code.visualstudio.com/docs/python/python-tutorial

Chapter 3

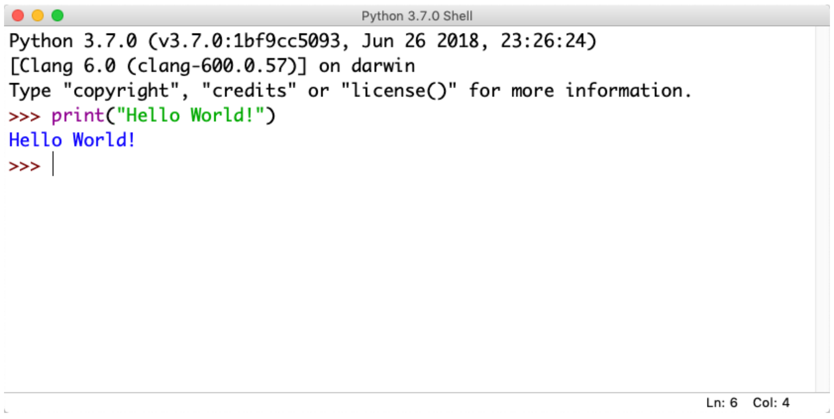
Start using Python

In this chapter we will start to use Python in some simple examples.

3.1 Python IDE

The basic code editor, or an integrated development environment, called IDLE. See Figure 3.1.

Other Python Editors will be discussed more in detail later. For now you can use the basic Python IDE (IDLE) or Spyder if you have installed the Anaconda distribution package.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello World!")
Hello World!
>>>
```

Figure 3.1: Python Shell / Python IDLE Editor

3.2 My first Python program

We will start using Python and create some code examples.

Example 3.2.1. Plotting in Python

Lets open your Python Editor and type the following:

```python
print("Hello World!")
```

Listing 3.1: Hello World Python Example

[End of Example]

An extremely useful command is **help()**, which enters a help functionality to explore all the stuff python lets you do, right from the interpreter. Press q to close the help window and return to the Python prompt.

You can use Python in different ways, either in "interactive" mode or in "Scripting" mode.

The python program that you have installed will by default act as something called an interpreter. An interpreter takes text commands and runs them as you enter them - very handy for trying things out.

Yo can run Python interactively in different ways either using the Console which is part of the operating system or the Python IDLE and the Python Shell which is part of the basic Python installation from https://www.python.org.

3.3 Python Shell

In interactive Mode you use the Python Shell as seen in Figure 3.1.

Here you type one and one command at a time after the ">>>" sign in the Python Shell.

```
>>> print("Hello World!")
```

3.4 Running Python from the Console

A console (or "terminal", or 'command prompt') is a textual way to interact with your OS (Operating System).

The python program that you have installed will by default act as something called an interpreter. An interpreter takes text commands and runs them as you enter them - very handy for trying things out.

Below we see how we can run Python from the Console which is part of the OS.

3.4.1 Opening the Console on macOS

The standard console on macOS is a program called Terminal. Open Terminal by navigating to Applications, then Utilities, then double-click the Terminal program. You can also easily search for it in the system search tool in the top right.

The command line Terminal is a tool for interacting with your computer. A window will open with a command line prompt message, something like this:

```
Last login: Tue Dec 11 08:33:51 on console
computername:~ username
```

Just type python at your console, hit Enter, and you should enter Python's Interpreter.

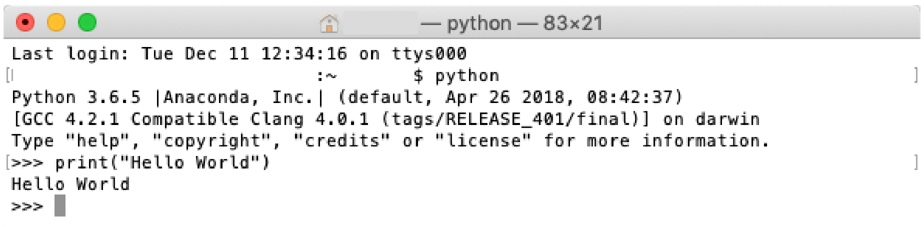
```
Last login: Tue Dec 11 12:34:16 on ttys000
Hans-Petter-Work-MacBook-Air:~ hansha$ python
Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on
    darwin
Type "help", "copyright", "credits" or "license" for more
    information.
>>>
```

The prompt >>> on the last line indicates that you are now in an interactive Python interpeter session, also called the "Python shell". This is different from the normal terminal command prompt!

You can now enter some code for python to run. Try:

```
>>> print("Hello World")
```

Se also Figure 3.2.



```
Last login: Tue Dec 11 12:34:16 on ttys000
                    :~      $ python
Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

Figure 3.2: Console macOS

Try other Python commands, e.g.:

```
>>> a = 5
>>> b = 2
>>> x = 5
>>> y = 3*a + b
>>> y
```

3.4.2 Opening the Console on Windows

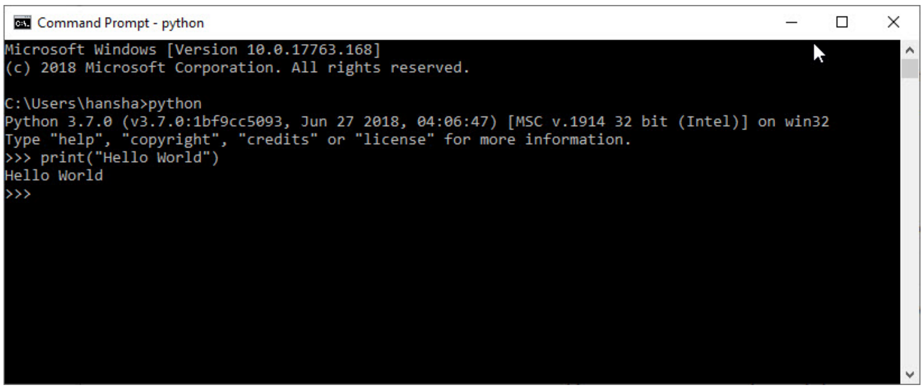
Window's console is called the Command Prompt, named cmd. An easy way to get to it is by using the key combination Windows+R (Windows meaning the windows logo button), which should open a Run dialog. Then type cmd and hit Enter or click Ok.

You can also search for it from the start menu.

It should look like:

```
C:\Users\myusername>
```

Just type python in the Command Prompt, hit Enter, and you should enter Python's Interpreter. See Figure 3.3.



```
Microsoft Windows [Version 10.0.17763.168]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\hansha>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

Figure 3.3: Command Prompt Windows

If you get an error message like this:

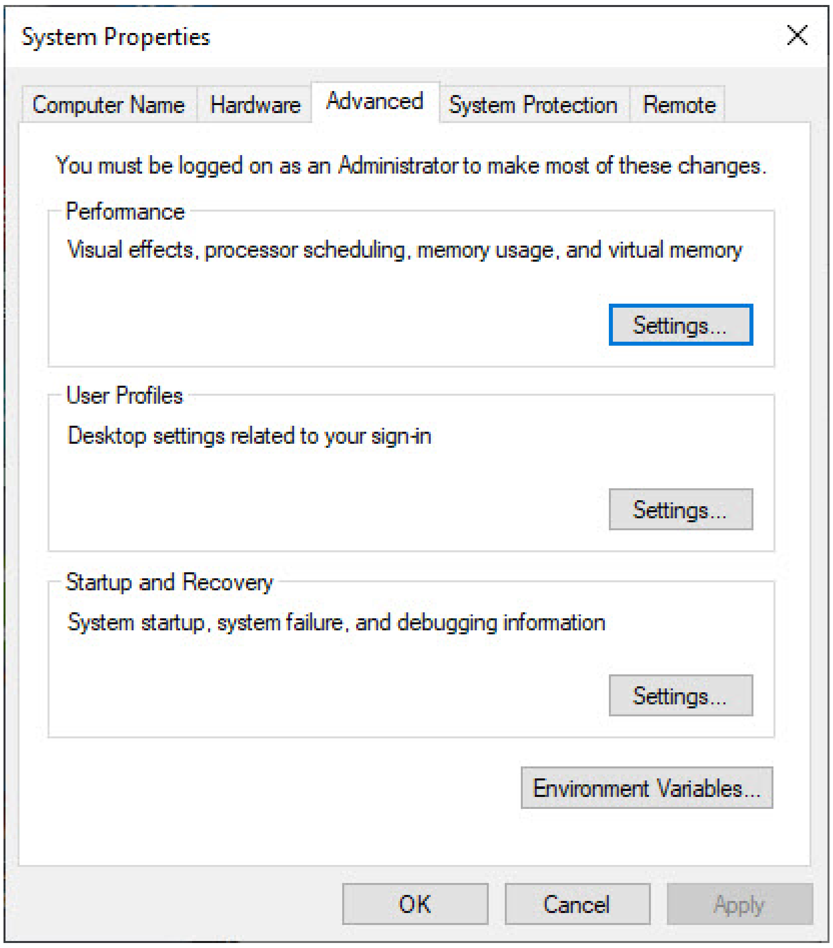
'python' is not recognized as an internal or external command, operable program or batch file.
Then you need to add Python to your path. See instructions below.

Note! This is also an option during the setup. While installing you can select "Add Python.exe to path". This option is by default set to "Off". To get that option you need to select "Customize", not using the "Default" installation.

3.4.3 Add Python to Path

In the Windows menu, search for "advanced system settings" and select View advanced system settings.

In the window that appears, click Environment Variables... near the bottom right. See Figure 3.4.


System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

Settings...

User Profiles

Desktop settings related to your sign-in

Settings...

Startup and Recovery

System startup, system failure, and debugging information

Settings...

Environment Variables...

OK | Cancel | Apply

Figure 3.4: Windows System Properties

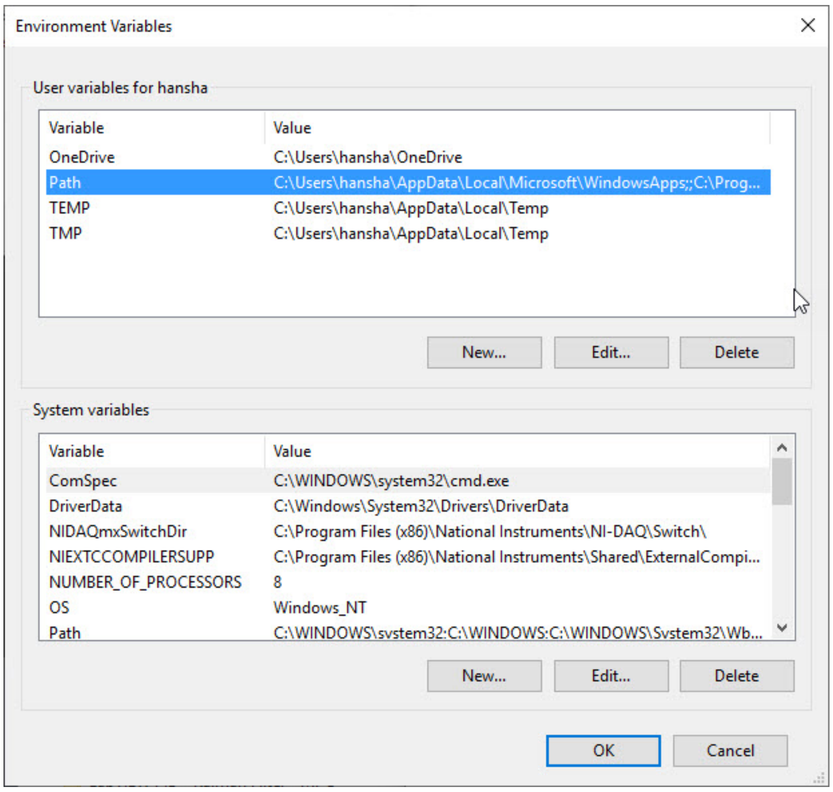
In the next window, find and select the user variable named Path and click Edit... to change its value. See Figure 3.5.

Select "New" and add the path where "python.exe" is located. See Figure 3.6.

The Default Location is:

```
C:\Users\user\AppData\Local\Programs\Python\Python37-32\
```

Click Save and open the Command Prompt once more and enter "python" to verify it works. See Figure 3.3.



Variable	Value
OneDrive	C:\Users\hansha\OneDrive
Path	C:\Users\hansha\AppData\Local\Microsoft\WindowsApps;;C:\Prog...
TEMP	C:\Users\hansha\AppData\Local\Temp
TMP	C:\Users\hansha\AppData\Local\Temp

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NIDAQmxSwitchDir	C:\Program Files (x86)\National Instruments\NI-DAQ\Switch\
NIEXTCCOMPILERSUPP	C:\Program Files (x86)\National Instruments\Shared\ExternalCompi...
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wb...

Figure 3.5: Windows System Properties

3.5 Scripting Mode

In "Scripting" mode you can write a Python Program with multiple Python commands and then save it as a file (.py).

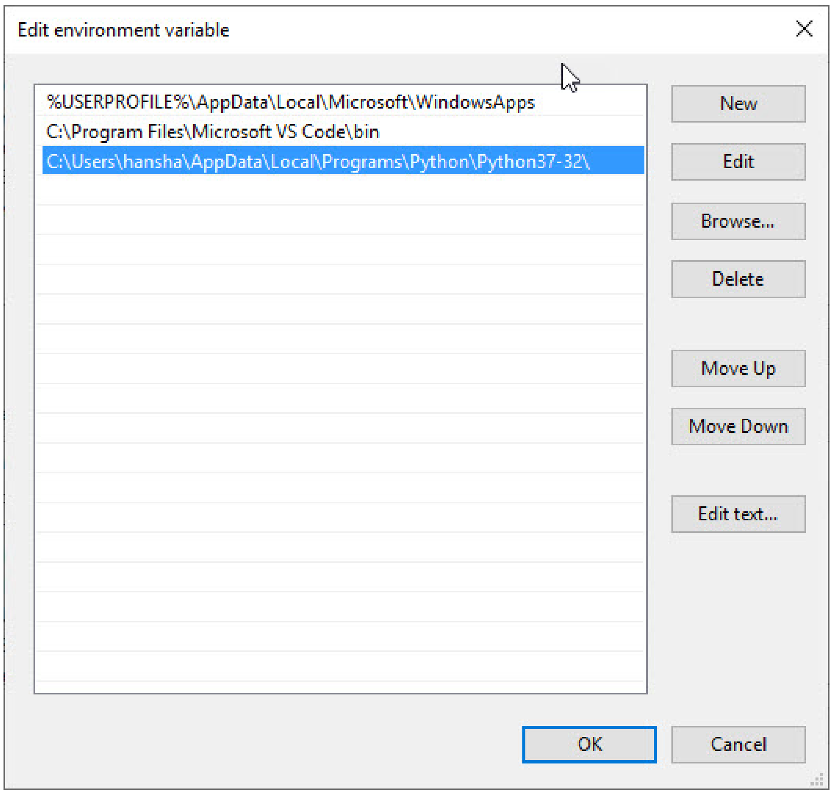
3.5.1 Run Python Scripts from the Python IDLE

From the Python Shell you select File → New File, or you can open an existing Pytho program or Python Script by selecting File → Open...

Lets create a new Script and type in the following:

```python
print("Hello")
print("World")
print("How are you?")
```

In Figure 3.7 we see how this is done. As you see we can enter many Python commands that together makes a Python program or Python script.
From the Python Shell you select Run → Run Module or hit F5 in order to run or execute the Python Script. See Figure 3.8.



```
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Program Files\Microsoft VS Code\bin
C:\Users\hansha\AppData\Local\Programs\Python\Python37-32\
```

Figure 3.6: Windows System Properties

The IDLE editor is very basic, for more complicated tasks you typically may prefer to use another editor like Spyder, Visual Studio Code, etc.

3.5.2 Run Python Scripts from the Console (Terminal) macOS

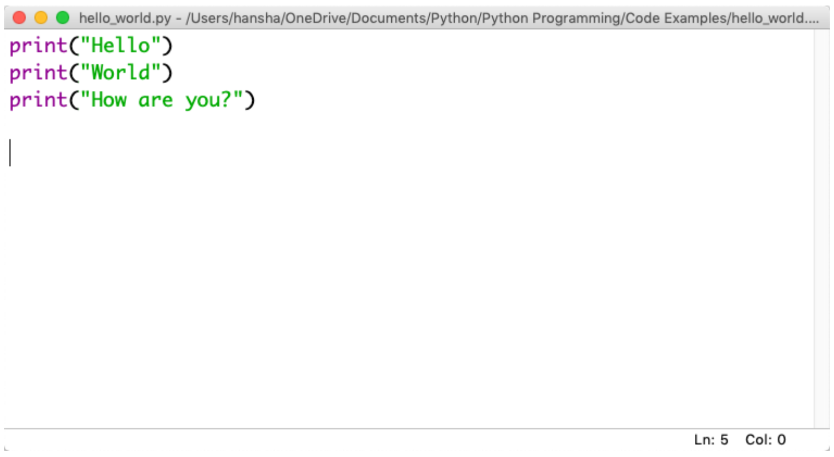
From the Console (Terminal) on macOS:

```
$ cd /Users/username/Downloads
$ python helloworld.py
```

Note! Make sure you are at your system command prompt, which will have $ or > at the end, not in Python mode (which has >>> instead)!

See also Figure 3.9.
Then it responds with:

```
Hello
World
How are you?
```


hello_world.py - /Users/hansha/OneDrive/Documents/Python/Python Programming/Code Examples/hello_world....

```python
print("Hello")
print("World")
print("How are you?")
```

Ln: 5 Col: 0

Figure 3.7: Python Script

3.5.3 Run Python Scripts from the Command Prompt in Windows

From Command Prompt in Window:

```
> cd /
> cd Temp
> python helloworld.py
```

Note! Make sure you are at your system command prompt, which will have > at the end, not in Python mode (which has >>> instead)!

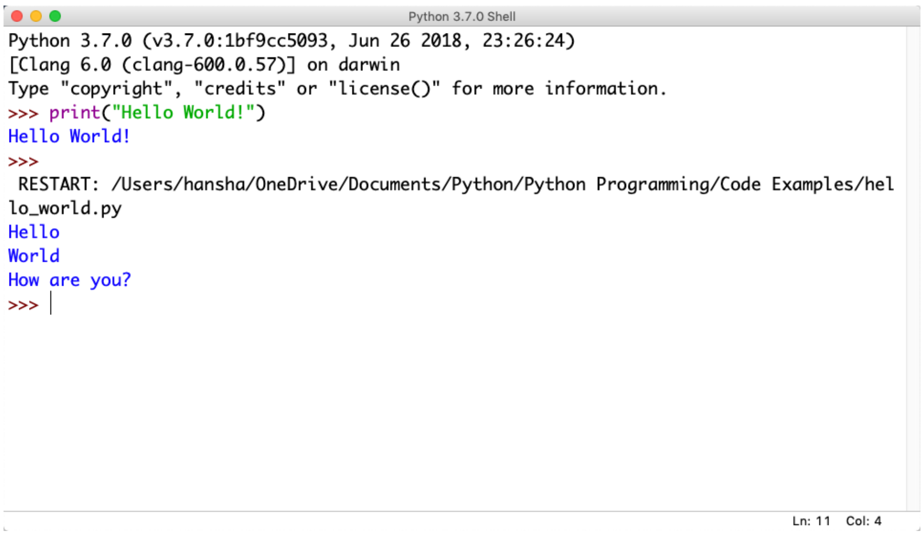
See also Figure 3.10.
Then it responds with:

```
Hello
World
How are you?
```

3.5.4 Run Python Scripts from Spyder

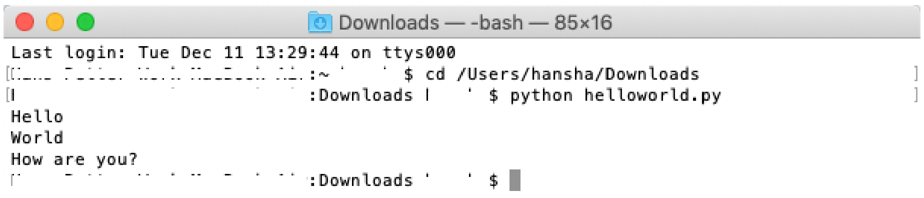
If you have installed the Anaconda distribution package you can use the Spyder editor. See 3.11.

In the Spyder editor we have the Script Editor to the left and the interactive Python Shell or the Console window to the right. See See 3.11.



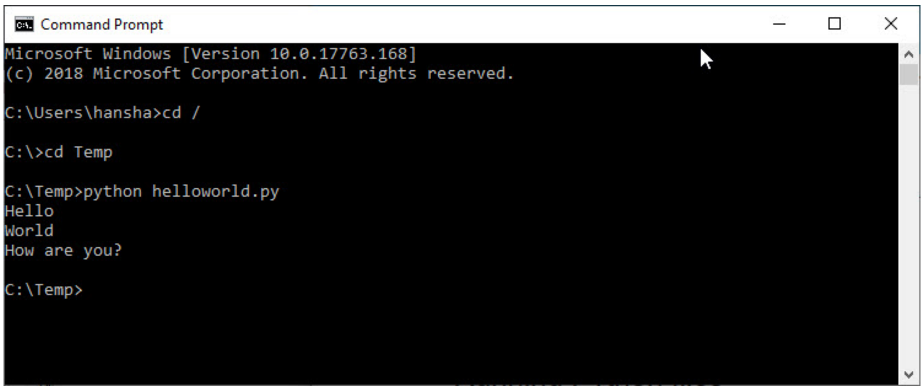
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello World!")
Hello World!
>>>
 RESTART: /Users/hansha/OneDrive/Documents/Python/Python Programming/Code Examples/hel
lo_world.py
Hello
World
How are you?
>>> 
```

Figure 3.8: Running a Python Script



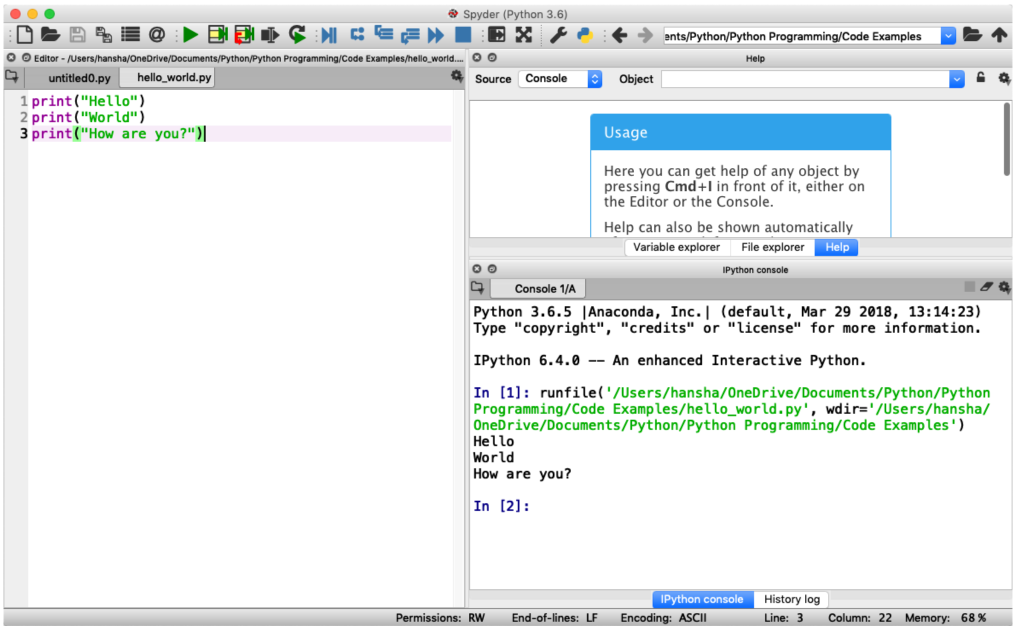
```
Last login: Tue Dec 11 13:29:44 on ttys000
[redacted]:~ [redacted] $ cd /Users/hansha/Downloads
[redacted]:Downloads [redacted] $ python helloworld.py
Hello
World
How are you?
[redacted]:Downloads [redacted] $ 
```

Figure 3.9: Running Python Scripts from Console window on macOS



```
Microsoft Windows [Version 10.0.17763.168]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\hansha>cd /

C:\>cd Temp

C:\Temp>python helloworld.py
Hello
World
How are you?

C:\Temp>
```

Figure 3.10: Running Python Scripts from Console window on macOS



```
print("Hello")
print("World")
print("How are you?")
```

Usage

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically

```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:14:23)
Type "copyright", "credits" or "license" for more information.

IPython 6.4.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/hansha/OneDrive/Documents/Python/Python
Programming/Code Examples/hello_world.py', wdir='/Users/hansha/
OneDrive/Documents/Python/Python Programming/Code Examples')
Hello
World
How are you?

In [2]:
```

Figure 3.11: Running a Python Script in Spyder

Chapter 4

Basic Python Programming

4.1 Basic Python Program

We will start using Python and create some code examples.

We use the basic IDLE editor (or another Python Editor)

Example 4.1.1. Hello World Example

Lets open your Python Editor and type the following:

```
print("Hello World!")
```

Listing 4.1: Hello World Python Example

[End of Example]

4.1.1 Get Help

An extremely useful command is **help()**, which enters a help functionality to explore all the stuff python lets you do, right from the interpreter.

Press q to close the help window and return to the Python prompt.

4.2 Variables

Variables are defined with the assignment operator, “=”. Python is dynamically typed, meaning that variables can be assigned without declaring their type, and that their type can change. Values can come from constants, from computation involving values of other variables, or from the output of a function.

Example 4.2.1. Creating and using Variables in Python

We use the basic IDLE (or another Python Editor) and type the following:

```
>>> x = 3
>>> x
3
```

Listing 4.2: Using Variables in Python

Here we define a variable and sets the value equal to 3 and then print the result to the screen.

[End of Example]

You can write one command by time in the IDLE. If you quit IDLE the variables and data are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead. This is known as creating a script.

Python scripts or programs are save as a text file with the extension **.py**

Example 4.2.2. Calculations in Python

We can use variables in a calculation like this:

```
x = 3
y = 3*x
print(y)
```

Listing 4.3: Using and Printing Variables in Python

We can implement the formula $y = ax + b$ like this:

```
a = 2
b = 5
x = 3

y = a*x + b

print(y)
```

Listing 4.4: Calculations in Python

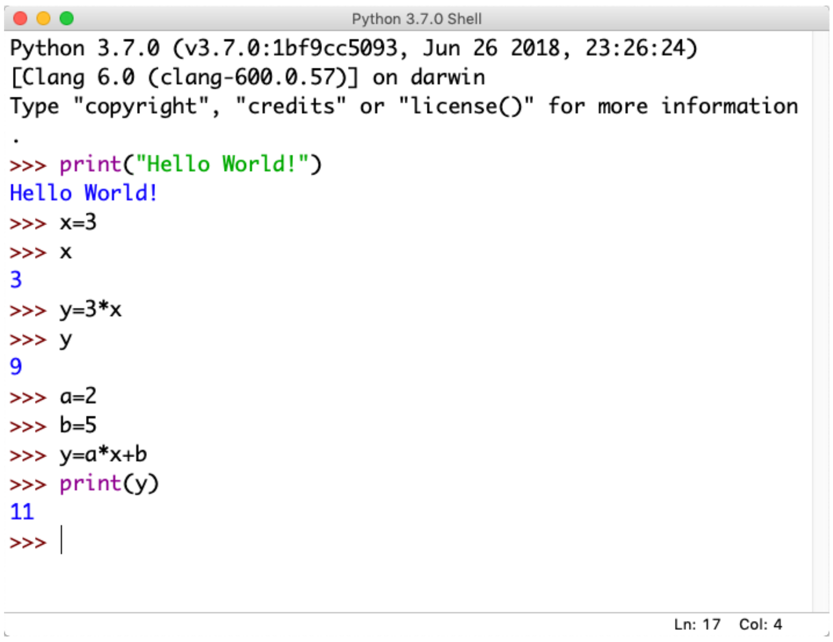
As seen in the examples, you can use the *print()* command in order to show the values on the screen.

[End of Example]

A variable can have a short name (like x and y) or a more descriptive name (sum, amount, etc).

You don need to define the variables before you use them (like you need to to in, e.g., C/C++/C).

Figure 4.1 show these examples using the basic IDLE editor.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information
.
>>> print("Hello World!")
Hello World!
>>> x=3
>>> x
3
>>> y=3*x
>>> y
9
>>> a=2
>>> b=5
>>> y=a*x+b
>>> print(y)
11
>>> |
```

Figure 4.1: Basic Python

Here are some basic rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters (A-z, 0-9) and underscores
- Variable names are case-sensitive, e.g., amount, Amount and AMOUNT are three different variables.

4.2.1 Numbers

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them, so in normal coding you don't need to bother.

Example 4.2.3. Numeric Types in Python

```python
x = 1       # int
y = 2.8     # float
z = 3 + 2j  # complex
```

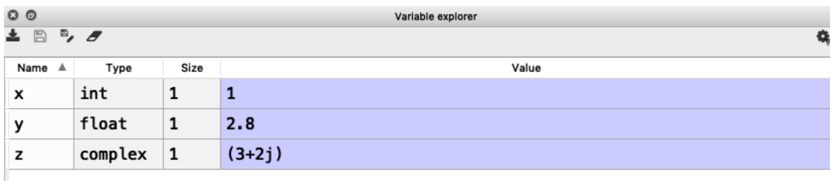
Listing 4.5: Numeric Types in Python

This means you just assign values to a variable without worrying about what kind of data type it is.

```python
print(type(x))
print(type(y))
print(type(z))
```

Listing 4.6: Check Data Types in Python

If you use the Spyder Editor, you can see the data types that a variable has using the Variable Explorer (Figure 4.2):



Name	Type	Size	Value
x	int	1	1
y	float	1	2.8
z	complex	1	(3+2j)

Figure 4.2: Variable Editor in Spyder

[End of Example]

4.2.2 Strings

Strings in Python are surrounded by either single quotation marks, or double quotation marks. 'Hello' is the same as "Hello".
Strings can be output to screen using the print function. For example: print("Hello").

Example 4.2.4. Using Strings in Python

Below we see examples of using strings in Python:

```python
a = "Hello World!"

print(a)

print(a[1])
print(a[2:5])
print(len(a))
print(a.lower())
print(a.upper())
print(a.replace("H", "J"))
print(a.split(" "))
```

Listing 4.7: Strings in Python

As you see in the example, there are many built-in functions form manipulating strings in Python. The Example shows only a few of them.

Strings in Python are arrays of bytes, and we can use index to get a specific character within the string as shown in the example code.

[End of Example]

4.2.3 String Input

Python allows for command line input.

That means we are able to ask the user for input.

Example 4.2.5. String Input in Python

The following example asks for the user's name, then, by using the input() method, the program prints the name to the screen:

```
print("Enter your name:")
x = input()
print("Hello, " + x)
```

Listing 4.8: String Input

[End of Example]

4.3 Built-in Functions

Python consists of lots of built-in functions. Some examples are the print function that we already have used (perhaps without noticing it is actually a Built-in function).

Python also consists of different Modules, Libraries or Packages. These Modules, Libraries or Packages consists of lots of predefined functions for different topics or areas, such as mathematics, plotting, handling database systems, etc. See Section 4.4 for more information and details regarding this.

In another chapter we will learn to create our own functions from scratch.

4.4 Python Standard Library

Python allows you to split your program into modules that can be reused in other Python programs. It comes with a large collection of standard modules that you can use as the basis of your programs.
The **Python Standard Library** consists of different modules for handling file I/O, basic mathematics, etc. You don't need to install these separately, but you need to important them when you want to use some of these modules or some of the functions within these modules.

The math module has all the basic math functions you need, such as: Trigonometric functions: *sin(x)*, *cos(x)*, etc. Logarithmic functions: *log()*, *log10()*, etc. Constants like *pi*, *e*, *inf*, *nan*, etc.

Example 4.4.1. Using the math module

We create some basic examples how to use a Library, a Package or a Module:

If we need only the sin() function, we can do like this:

```python
from math import sin

x = 3.14
y = sin(x)

print(y)
```

If we need a few functions, we can do like this:

```python
from math import sin, cos

x = 3.14
y = sin(x)
print(y)

y = cos(x)
print(y)
```

If we need many functions, we can do like this:

```python
from math import *

x = 3.14
y = sin(x)
print(y)

y = cos(x)
print(y)
```

We can also use this alternative:

```python
import math

x = 3.14
y = math.sin(x)

print(y)
```

We can also write it like this:

```python
import math as mt

x = 3.14
y = mt.sin(x)

print(y)
```

[End of Example]

There are advantages and disadvantages with the different approaches. In your program you may need to use functions from many different modules or packages. If you import the whole module instead of just the function(s) you need you use more of the computer memory.

Very often we also need to import and use multiple libraries where the different libraries have some functions with the same name but different use.

Other useful modules in the **Python Standard Library** are **statistics** (where you have functions like *mean()*, *stdev()*, etc.)

For more information about the functions in the **Python Standard Library**, see:
https://docs.python.org/3/library/index.html

4.5 Using Python Libraries, Packages and Modules

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This approach has advantages and disadvantages. A disadvantage is that you need to install these packages separately and then later import these modules in your code.

Some important packages are:

- **NumPy** - NumPy is the fundamental package for scientific computing with Python
- **SciPy** - SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.
- **Matplotlib** - Matplotlib is a Python 2D plotting library

Lots of other packages exists, depending on what you are going to solve.

These packages need to be downloaded and installed separately, or you choose to use, e.g., a distribution package like Anaconda.

Here you find an overview of the **NumPy** library:
https://www.numpy.org

Here you find an overview of the **SciPy** library:
https://www.scipy.org

Here you find an overview of the **Matplotlib** library:
https://matplotlib.org

You will learn the basics features in all these libraries. We will use all of the in different examples and exercises throughout this textbook.

Example 4.5.1. Using libraries

In this example we use the NumPy library:

```python
import numpy as np

x = 3

y = np.sin(x)

print(y)
```

In this example we use both the math module in the Python Standard Library and the NumPy library:

```python
import math as mt
import numpy as np

x = 3

y = mt.sin(x)

print(y)


y = np.sin(x)

print(y)
```

Note! As seen in this example we use a function called sin() which exists both in the math module in the Python Standard Library and the NumPy library. In this case they give the same results. In this case the following code is not recommended:

```python
from math import *
from numpy import *

x = 3

```

```
y = sin(x)

print(y)


y = sin(x)

print(y)
```

In this case it works, but assume you have 2 different functions with the same name that have different meaning in 2 different libraries.

[End of Example]

4.5.1 Python Packages

In addition to the Python Standard Library, there is a growing collection of several thousand components (from individual programs and modules to packages and entire application development frameworks), available from the **Python Package Index**.

Python Package Index (PYPI):
https://pypi.org

Here you can download and install individual Python packages.
An easy alternative is the Anaconda Distribution, where many of the most used Python packages are included.

Anaconda:
https://www.anaconda.com/distribution/

4.6 Plotting in Python

Typically you need to create some plots or charts. In order to make plots or charts in Python you will need an external library. The most used library is **Matplotlib**.

Matplotlib is a Python 2D plotting library

Here you find an overview of the Matplotlib library:
https://matplotlib.org

If you are familiar with MATLAB and basic plotting in MATLAB, using the Matplotlib is very similar.

The main difference from MATLAB is that you need to import the library, either the whole library or one or more functions.
For simplicity we import the whole library like this:

```
import matplotlib.pyplot as plt
```

Plotting functions that you will use a lot:

- plot()
- title()
- xlabel()
- ylabel()
- axis()
- grid()
- subplot()
- legend()
- show()

Lets create some basic plotting examples using the Matplotlib library:

Example 4.6.1. Plotting in Python

In this example we have two arrays with data. We want to plot x vs. y. We can assume x is a time series and y is the corresponding temperature in degrees Celsius.

```python
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

y = [5, 2,4, 4, 8, 7, 4, 8, 10, 9]

plt.plot(x,y)
plt.xlabel('Time (s)')
plt.ylabel('Temperature (degC)')
plt.show()
```

We get the plot as shown in Figure 4.3.

We can also write like this:

```python
from matplotlib.pyplot import *

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = [5, 2,4, 4, 8, 7, 4, 8, 10, 9]

plot(x,y)
xlabel('Time (s)')
ylabel('Temperature (degC)')
show()
```

This makes the code simpler to read. one problem with this approach appears assuming we import and use multiple libraries and the different libraries have some functions with the same name but different use.

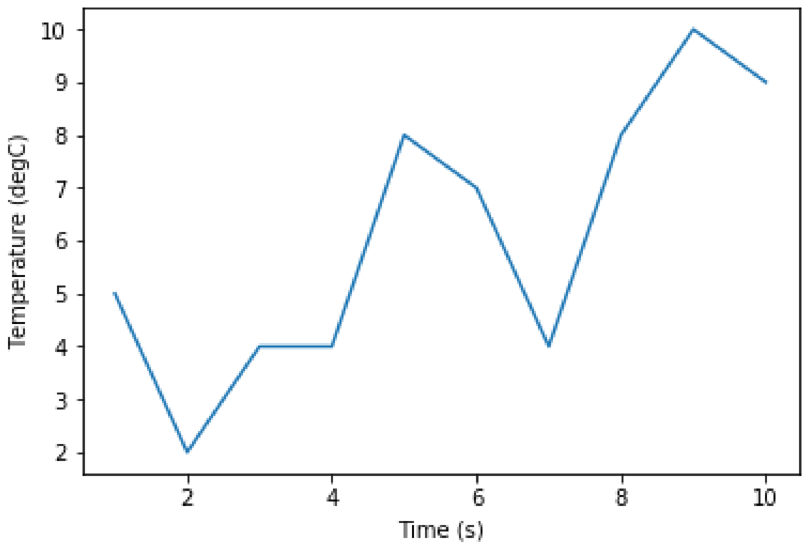


Figure 4.3: Plotting in Python

[End of Example]

We have used 4 basic plotting function in the Matplotlib library:

- plot()
- xlabel()
- ylabel()
- show()

Example 4.6.2. Plotting a Sine Curve

```python
import numpy as np
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4, 5, 6, 7]

y = np.sin(x)

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

This gives the following plot (see Figure 4.4):

A better solution will then be:

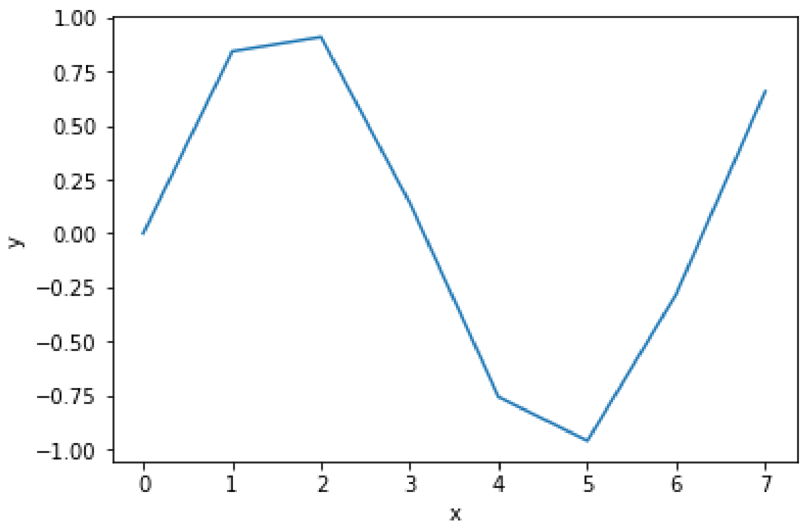


Figure 4.4: Plotting a Sine function in Python

```python
import matplotlib.pyplot as plt
import numpy as np

xstart = 0
xstop = 2*np.pi
increment = 0.1

x = np.arange(xstart,xstop,increment)

y = np.sin(x)

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

This gives the following plot (see Figure 4.5):
If you want grids you can use the grid() function.

[End of Example]

4.6.1 Subplots

The subplot command enables you to display multiple plots in the same window. Typing "subplot(m,n,p)" partitions the figure window into an m-by-n matrix of small subplots and selects the subplot for the current plot. The plots are numbered along the first row of the figure window, then the second row, and so on. See Figure 4.6.

Example 4.6.3. Creating Subplots

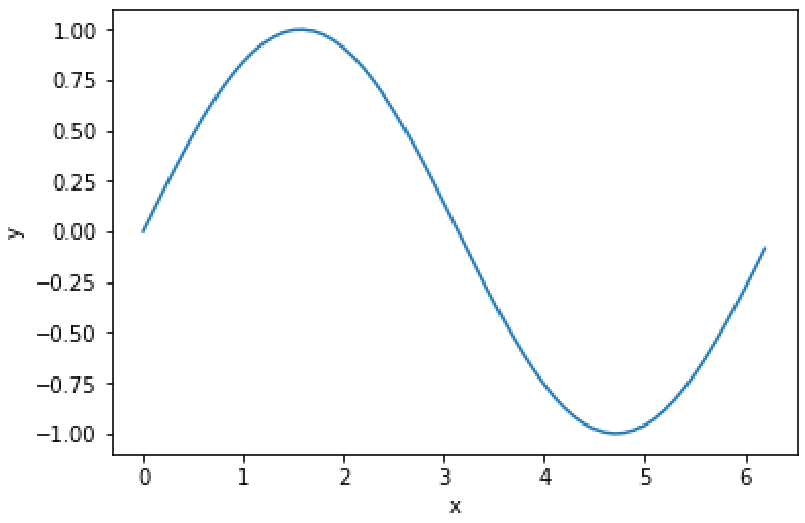


Figure 4.5: Plotting a Sine function in Python - Better Implementation

We will create and plot sin() and cos() in 2 different subplots.

```
import matplotlib.pyplot as plt
import numpy as np

xstart = 0
xstop = 2*np.pi
increment = 0.1

x = np.arange(xstart,xstop,increment)

y = np.sin(x)

z = np.cos(x)


plt.subplot(2,1,1)
plt.plot(x, y, 'g')
plt.title('sin')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.grid()
plt.show()


plt.subplot(2,1,2)
plt.plot(x, z, 'r')
plt.title('cos')
plt.xlabel('x')
plt.ylabel('cos(x)')
plt.grid()
plt.show()
```

[End of Example]

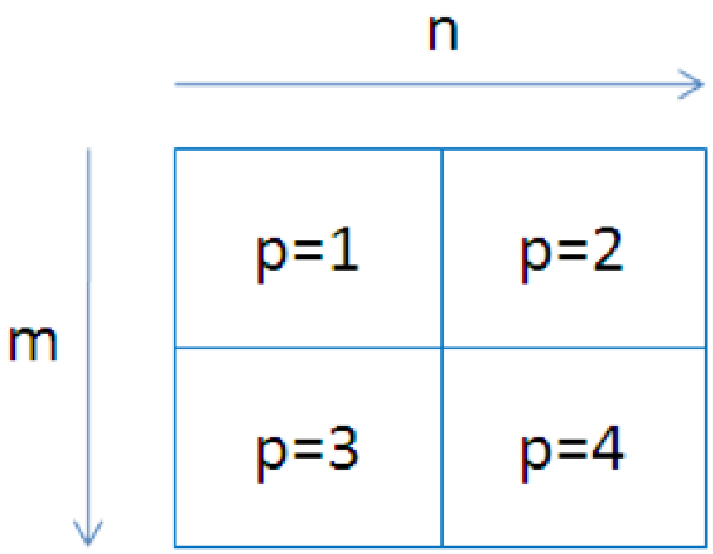


Figure 4.6: Creating Subplots in Python

4.6.2 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 4.6.1. Create sin(x) and cos(x) in 2 different plots

Create sin(x) and cos(x) in 2 different plots.

You should use all the Plotting functions listed below in your code:

- plot()
- title()
- xlabel()
- ylabel()
- axis()
- grid()
- legend()
- show()

[End of Exercise]

Part II

Python Programming

Chapter 5

Python Programming

We have been through the basics in Python, such as variables, using some basic built-in functions, basic plotting, etc.

You may come far only using these thins, but to create real applications, you need to know about and use features like:

- If ... Else
- For Loops
- While Loops
- Arrays ...

If you are familiar with one or more other programming language, these features should be familiar and known to you. All programming languages have these features built-in, but the syntax is slightly different from one language to another.

5.1 If ... Else

An "if statement" is written by using the **if** keyword.

Here are some Examples how you use a If sentences in Python:

Example 5.1.1. Using If ... Else in Python

Using **If**:

```
a = 5
b = 8

if a > b:
    print("a is greater than b")

if b > a:
    print("b is greater than a")

if a == b:
```

```python
    print("a is equal to b")
```

Listing 5.1: If

Try to change the values for a and b.

Using **If - Else**:

```python
a = 5
b = 8

if a > b:
    print("a is greater than b")
else:
    print("b is greater than a or a and b are equal")
```

Listing 5.2: If - Else

Using **Elif**:

```python
a = 5
b = 8

if a > b:
    print("a is greater than b")
elif b > a:
    print("b is greater than a")
elif a == b:
    print("a is equal to b")
```

Listing 5.3: Elif

Note! Python uses "elif" not "elseif" like many other programming languages do.

[End of Example]

5.2 Arrays

An array is a special variable, which can hold more than one value at a time.

Here are some Examples how you can create and use Arrays in Python:

Example 5.2.1. Arrays in Python

```python
data = [1.6, 3.4, 5.5, 9.4]

N = len(data)

print(N)

print(data[2])

data[2] = 7.3

print(data[2])
```

```python


for x in data:
  print(x)


data.append(11.4)


N = len(data)

print(N)


for x in data:
  print(x)
```

Listing 5.4: Using Arrays in Python

You define an array like this:

```python
data = [1.6, 3.4, 5.5, 9.4]
```

You can also use text like this:

```python
carlist = ["Volvo", "Tesla", "Ford"]
```

You can use Arrays in Loops like this:

```python
for x in data:
  print(x)
```

You can return the number of elements in the array like this:

```python
N = len(data)
```

You can get a specific value inside the array like this:

```python
index = 2
x = cars[index]
```

You can use the append() method to add an element to an array:

```python
data.append(11.4)
```

[End of Example]

You have many built in methods you can use in combination with arrays, like sort(), clear(), copy(), count(), insert(), remove(), etc.

You should look into test all these methods.

5.3 For Loops

A For loop is used for iterating over a sequence. I guess all your programs will use one or more For loops. So if you have not used For loops before, make sure to learn it now.

Below you see a basic example how you can use a For loop in Python:

```python
for i in range(1, 10):
  print(i)
```

The For loop is probably one of the most useful feature in Python (or in any kind of programming language). Below you will see different examples how you can use a For loop in Python.

Example 5.3.1. Using For Loops in Python

```python
data = [1.6, 3.4, 5.5, 9.4]

for x in data:
  print(x)


carlist = ["Volvo", "Tesla", "Ford"]

for car in carlist:
  print(car)
```

Listing 5.5: Using For Loops in Python

The range() function is handy to use in For Loops:

```python
N = 10

for x in range(N):
  print(x)
```

The **range()** function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

You can also use the range() function like this:

```python
start = 4
stop= 12 #but not including

for x in range(start, stop):
  print(x)
```

Finally, you can also use the range() function like this:

```python
start = 4
stop = 12 #but not including
step = 2

for x in range(start, stop, step):
  print(x)
```

You should try all these examples in order to learn the basic structure of a For loop.

[End of Example]

Example 5.3.2. Using For Loops for Summation of Data

You typically want to use a For loop for find the sum of a given data set.

```python
data = [1, 5, 6, 3, 12, 3]

sum = 0

#Find the Sum of all the numbers
for x in data:
  sum = sum + x

print(sum)

#Find the Mean or Average of all the numbers

N = len(data)

mean = sum/N

print(mean)
```

This gives the following results:

```
30
5.0
```

[End of Example]

Example 5.3.3. Implementing Fibonacci Numbers Using a For Loop in Python

Fibonacci numbers are used in the analysis of financial markets, in strategies such as Fibonacci retracement, and are used in computer algorithms such as the Fibonacci search technique and the Fibonacci heap data structure.
They also appear in biological settings, such as branching in trees, arrangement of leaves on a stem, the fruitlets of a pineapple, the flowering of artichoke, an uncurling fern and the arrangement of a pine cone.

In mathematics, Fibonacci numbers are the numbers in the following sequence: 0, 1, 1, 2 ,3, 5, 8, 13, 21, 34, 55, 89, 144, . . .

By definition, the first two Fibonacci numbers are 0 and 1, and each subsequent number is the sum of the previous two.

Some sources omit the initial 0, instead beginning the sequence with two 1s.

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

$$f_n = f_{n-1} + f_{n-2} \tag{5.1}$$

with seed values:

$$f_0 = 0, f_1 = 1$$

We will write a Python script that calculates the N first Fibonacci numbers. The Python Script becomes like this:

```python
N = 10

fib1 = 0
fib2 = 1

print(fib1)
print(fib2)

for k in range(N-2):
    fib_next = fib2 +fib1
    fib1 = fib2
    fib2 = fib_next
    print(fib_next)
```

Listing 5.6: Fibonacci Numbers Using a For Loop in Python

Alternative solution:

```python
N = 10

fib = [0, 1]


for k in range(N-2):
    fib_next =  fib[k+1] +fib[k]
    fib.append(fib_next)

print(fib)
```

Listing 5.7: Fibonacci Numbers Using a For Loop in Python - Alt2

Another alternative solution:

```python
N = 10

fib = []

for k in range(N):
     fib.append(0)

fib[0] = 0
fib[1] = 1

```

```python
for k in range(N-2):
    fib[k+2] = fib[k+1] +fib[k]


print(fib)
```
Listing 5.8: Fibonacci Numbers Using a For Loop in Python - Alt3

Another alternative solution:

```python
import numpy as np


N = 10

fib = np.zeros(N)

fib[0] = 0
fib[1] = 1

for k in range(N-2):
    fib[k+2] = fib[k+1] +fib[k]


print(fib)
```
Listing 5.9: Fibonacci Numbers Using a For Loop in Python - Alt4

[End of Example]

5.3.1 Nested For Loops

In Python and other programming languages you can use one loop inside another loop.

Syntax for nested For loops in Python:

```python
for iterating_var in sequence:
    for iterating_var in sequence:
        statements(s)
    statements(s)
```

Simple example:

```python
for i in range(1, 10):
    for k in range(1, 10):
        print(i, k)
```

Exercise 5.3.1. Prime Numbers

The first 25 prime numbers (all the prime numbers less than 100) are:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

By definition a prime number has both 1 and itself as a divisor. If it has any other divisor, it cannot be prime.

A natural number (1, 2, 3, 4, 5, 6, etc.) is called a prime number (or a prime) if it is greater than 1 and cannot be written as a product of two natural numbers that are both smaller than it.

Create a Python Script where you find all prime numbers between 1 and 200.

Tip! I guess this can be done in many different ways, but one way is to use 2 nested For Loops.

[End of Exercise]

5.4 While Loops

The while loop repeats a group of statements an indefinite number of times under control of a logical condition.

Example 5.4.1. Using While Loops in Python

```python
m = 8

while m > 2:
    print(m)
    m = m - 1
```

Listing 5.10: Using While Loops in Python

[End of Example]

5.5 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 5.5.1. Plot of Dynamic System

Given the autonomous system:

$$\dot{x} = ax \tag{5.2}$$

Where:

$$a = -\frac{1}{T}$$

where T is the time constant.

The solution for the differential equation is:

$$x(t) = e^{at}x_0 \tag{5.3}$$

Set T=5 and the initial condition x(0)=1.

Create a Script in Python (.py file) where you plot the solution x(t) in the time interval:

$$0 \leq t \leq 25$$

Add Grid, and proper Title and Axis Labels to the plot.

[End of Exercise]

Chapter 6

Creating Functions in Python

6.1 Introduction

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

Previously we have been using many of the built-in functions in Python

If you are familiar with one or more other programming language, creating and using functions should be familiar and known to you. All programming languages has the possibility to create functions, but the syntax is slightly different from one language to another.

Some programming languages uses the term Method instead of a Function. Functions and Methods behave in the same manner, but you could say that Methods are functions that belongs to a Class. We will learn more about Classes in Chapter 7.

Scripts vs. Functions

It is important to know the difference between a Script and a Function.

Scripts:

- A collection of commands that you would execute in the Editor
- Used for automating repetitive tasks

Functions:

- Operate on information (inputs) fed into them and return outputs
- Have a separate workspace and internal variables that is only valid inside the function

- Your own user-defined functions work the same way as the built-in functions you use all the time, such as plot(), rand(), mean(), std(), etc.

Python have lots of built-in functions, but very often we need to create our own functions (we could refer to these functions as user-defined functions)
In Python a function is defined using the **def** keyword:

```
def FunctionName:
    <statement-1>
    .
    .
    <statement-N>
    return ...
```

Example 6.1.1. Basic Function

Below you see a simple function created in Python:

```
def add(x,y):

    return x + y
```

Listing 6.1: Basic Python Function

The function adds 2 numbers. The name of the function is **add**, and it returns the answer using the **return** statement.

The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Note that you need to use a colon "**:**" at the end of line where you define the function.

Note also the indention used.

```
def add(x,y):
```

Here you see a Python script where we use the function:

```
def add(x,y):

    return x + y


x = 2
y = 5

z = add(x,y)

print(z)
```

Listing 6.2: Creating and Using a Python Function

[End of Example]

Example 6.1.2. Create a Function in a separate File

We start by creating a separate Python File (**myfunctions.py**) for the function:

```python
def average(x,y):

    return (x + y)/2
```

Listing 6.3: Function calculating the Average

Next, we create a new Python File (e.g., **testaverage.py**) where we use the function we created:

```python
from myfunctions import average

a = 2
b = 3

c = average(a,b)

print(c)
```

Listing 6.4: Test of Average function

[End of Example]

6.2 Functions with multiple return values

Typically we want to return more than one value from a function.

Example 6.2.1. Create a Function Function with multiple return values

Create the following example:

```python
def stat(x):

    totalsum = 0

    #Find the Sum of all the numbers
    for x in data:
      totalsum = totalsum + x


    #Find the Mean or Average of all the numbers

    N = len(data)

    mean = totalsum/N


    return totalsum, mean



```

```python
data = [1, 5, 6, 3, 12, 3]


totalsum, mean = stat(data)

print(totalsum, mean)
```
Listing 6.5: Function with multiple return values

[End of Example]

6.3 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 6.3.1. Create Python Function

Create a function **calcaverage** that finds the average of two numbers.

[End of Exercise]

Exercise 6.3.2. Create Python functions for converting between radians and degrees

Since most of the trigonometric functions require that the angle is expressed in radians, we will create our own functions in order to convert between radians and degrees.

It is quite easy to convert from radians to degrees or from degrees to radians.

We have that:

$$2\pi[radians] = 360[degrees] \tag{6.1}$$

This gives:

$$d[degrees] = r[radians] \times (\frac{180}{\pi}) \tag{6.2}$$

and

$$r[radians] = d[degrees] \times (\frac{\pi}{180}) \tag{6.3}$$

Create two functions that convert from radians to degrees (r2d(x)) and from degrees to radians (d2r(x)) respectively.

These functions should be saved in one Python file .py.

Test the functions to make sure that they work as expected.

[End of Exercise]

Exercise 6.3.3. Create a Function that Implementing Fibonacci Numbers

Fibonacci numbers are used in the analysis of financial markets, in strategies such as Fibonacci retracement, and are used in computer algorithms such as the Fibonacci search technique and the Fibonacci heap data structure.
They also appear in biological settings, such as branching in trees, arrangement of leaves on a stem, the fruitlets of a pineapple, the flowering of artichoke, an uncurling fern and the arrangement of a pine cone.

In mathematics, Fibonacci numbers are the numbers in the following sequence:
0, 1, 1, 2 ,3, 5, 8, 13, 21, 34, 55, 89, 144, . . .

By definition, the first two Fibonacci numbers are 0 and 1, and each subsequent number is the sum of the previous two.

Some sources omit the initial 0, instead beginning the sequence with two 1s.

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

$$f_n = f_{n-1} + f_{n-2} \tag{6.4}$$

with seed values:

$$f_0 = 0, f_1 = 1$$

Create a Function that Implementing the N first Fibonacci Numbers

[End of Exercise]

Exercise 6.3.4. Prime Numbers

The first 25 prime numbers (all the prime numbers less than 100) are:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

By definition a prime number has both 1 and itself as a divisor. If it has any other divisor, it cannot be prime.

A natural number (1, 2, 3, 4, 5, 6, etc.) is called a prime number (or a prime) if it is greater than 1 and cannot be written as a product of two natural numbers that are both smaller than it.

Tip! I guess this can be implemented in many different ways, but one way is to use 2 nested For Loops.

Create a Python function where you check if a given number is a prime number or not.

You can check the function in the Command Window like this:

```
number = 4
checkifprime(number)
```

Then Python respond with True or False.

[End of Exercise]

Chapter 7

Creating Classes in Python

7.1 Introduction

Python is an object oriented programming (OOP) language. Almost everything in Python is an object, with its properties and methods.

The foundation for all object oriented programming (OOP) languages are Classes.

To create a class, use the keyword **class**:

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

Example 7.1.1. Simple Class Example

We will create a simple Class in Python.

```
class Car:
    model = "Volvo"
    color = "Blue"


car = Car()


print(car.model)
print(car.color)
```

Listing 7.1: Simple Python Class

The results will be in this case:

```
Volvo
Blue
```

This example don't illustrate the good things with classes so we will create some more examples.

[End of Example]

Example 7.1.2. Python Class

Lets create the following Python Code:

```python
class Car:
    model = ""
    color = ""

car = Car()

car.model = "Volvo"
car.color = "Blue"

print(car.color + " " + car.model)

car.model = "Ford"
car.color = "Green"

print(car.color + " " + car.model)
```

Listing 7.2: Python Class example

You should try these examples.

[End of Example]

7.2 The __init__() Function

In Python all classes have a built-in function called __init__(), which is always executed when the class is being initiated.
In many other OOP languages we call this the Constructor.

Exercise 7.2.1. The __init__() Function

We will create a simple example where we use the __init__() function to illustrate the principle.

We change our previous Car example like this:

```python
class Car:
  def __init__(self, model, color):
    self.model = model
    self.color = color

car1 = Car("Ford", "Green")

print(car1.model)
print(car1.color)


```

```
car2 = Car("Volvo", "Blue")

print(car2.model)
print(car2.color)
```
Listing 7.3: Python Class Constructor Example

Lets extend the Class by defining a Function as well:

```
# Defining the Class Car
class Car:
    def __init__(self, model, color):
        self.model = model
        self.color = color

    def displayCar(self):
        print(self.model)
        print(self.color)


# Lets start using the Class

car1 = Car("Tesla", "Red")

car1.displayCar()


car2 = Car("Ford", "Green")

print(car2.model)
print(car2.color)


car3 = Car("Volvo", "Blue")

print(car3.model)
print(car3.color)

car3.color="Black"

car3.displayCar()
```
Listing 7.4: Python Class with Function

As you see from the code we have now defined a Class "Car" that has 2 Class variables called "model" and "color", and in addition we have defined a Function (or Method) called "displayCar()".

Its normal to use the term "Method" for Functions that are defined within a Class.

You declare class methods like normal functions with the exception that the first argument to each method is *self*.

To create instances of a class, you call the class using class name and pass in whatever arguments its __init__() method accepts.

For example:

```python
car1 = Car("Tesla", "Red")
```

[End of Example]

Exercise 7.2.2. Create the Class in a separate Python file

We start by creating the Class and then we save the code in "Car.py":

```python
# Defining the Class Car
class Car:
    def __init__(self, model, color):
        self.model = model
        self.color = color

    def displayCar(self):
        print(self.model)
        print(self.color)
```

Listing 7.5: Define Python Class in separate File

Then we create a Python Script (testCar.py) where we are using the Class:

```python
# Importing the Car Class
from Car import Car

# Lets start using the Class

car1 = Car("Tesla", "Red")

car1.displayCar()


car2 = Car("Ford", "Green")

print(car2.model)
print(car2.color)


car3 = Car("Volvo", "Blue")

print(car3.model)
print(car3.color)

car3.color="Black"

car3.displayCar()
```

Listing 7.6: Script that is using the Class

Notice the following line at the top:

```python
from Car import Car
```

[language=Python]

[End of Example]

7.3 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 7.3.1. Create Python Class

Create a Python Class where you calculate the degrees in Fahrenheit based on the temperature in Celsius and vice versa.

The formula for converting from Celsius to Fahrenheit is:

$$T_f = (T_c \times 9/5) + 32 \tag{7.1}$$

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \tag{7.2}$$

[End of Exercise]

Chapter 8

Creating Python Modules

As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use a handy function that you have written in several programs without copying its definition into each program.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter (the Python Console window).

8.1 Python Modules

A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended.

Python allows you to split your program into modules that can be reused in other Python programs. It comes with a large collection of standard modules that you can use as the basis of your programs as we have seen examples of in previous chapters. Not it is time to make your own modules from scratch.

Consider a module to be the same as a code library. A file containing a set of functions you want to include in your application.

Previously you have been using different modules, libraries or packages created by the Python organization or by others. Here you will create your own modules from scratch.

Example 8.1.1. Create your first Python Module

We will create a Python module with 2 functions. The first function should convert from Celsius to Fahrenheit and the other function should convert from Fahrenheit to Celsius.

The formula for converting from Celsius to Fahrenheit is:

$$T_f = (T_c \times 9/5) + 32 \tag{8.1}$$

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \tag{8.2}$$

First, we create a Python module with the following functions (**fahrenheit.py**):

```python
def c2f(Tc):

    Tf = (Tc * 9/5) + 32
    return Tf


def f2c(Tf):

    Tc = (Tf - 32)*(5/9)
    return Tc
```

Listing 8.1: Fahrenheit Functions

Then, we create a Python script for testing the functions (**testfahrenheit.py**):

```python
from fahrenheit import c2f, f2c

Tc = 0

Tf = c2f(Tc)

print("Fahrenheit: " + str(Tf))


Tf = 32

Tc = f2c(Tf)

print("Celsius: " + str(Tc))
```

Listing 8.2: Python Script testing the functions

The results becomes:

```
Fahrenheit: 32.0
Celsius: 0.0
```

8.2 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 8.2.1. Create Python Module for converting between radians and degrees

Since most of the trigonometric functions require that the angle is expressed in radians, we will create our own functions in order to convert between radians and degrees.

It is quite easy to convert from radians to degrees or from degrees to radians. We have that:

$$2\pi[radians] = 360[degrees] \tag{8.3}$$

This gives:

$$d[degrees] = r[radians] \times (\frac{180}{\pi}) \tag{8.4}$$

and

$$r[radians] = d[degrees] \times (\frac{\pi}{180}) \tag{8.5}$$

Create two functions that convert from radians to degrees (r2d(x)) and from degrees to radians (d2r(x)) respectively.

These functions should be saved in one Python file .py.

Test the functions to make sure that they work as expected. You can choose to make a new .py file to test these functions or you can use the Console window.

[End of Exercise]

Chapter 9

File Handling in Python

9.1 Introduction

Python has several functions for creating, reading, updating, and deleting files. The key function for working with files in Python is the **open()** function.

The open() function takes two parameters; Filename, and Mode.

There are four different methods (modes) for opening a file:

- ”x” - Create - Creates the specified file, returns an error if the file exists
- ”w” - Write - Opens a file for writing, creates the file if it does not exist
- ”r” - Read - Default value. Opens a file for reading, error if the file does not exist
- ”a” - Append - Opens a file for appending, creates the file if it does not exist

In addition you can specify if the file should be handled as binary or text mode

- ”t” - Text - Default value. Text mode
- ”b” - Binary - Binary mode (e.g. images)

9.2 Write Data to a File

To create a **New** file in Python, use the open() method, with one of the following parameters:

- ”x” - Create - Creates the specified file, returns an error if the file exists
- ”w” - Write - Opens a file for writing, creates the file if it does not exist
- ”a” - Append - Opens a file for appending, creates the file if it does not exist

To write to an **Existing** file, you must add a parameter to the open() function:

- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist

Example 9.2.1. Write Data to a File

```
f = open("myfile.txt", "x")

data = "Helo World"

f.write(data)

f.close()
```

Listing 9.1: Write Data to a File

[End of Example]

9.3 Read Data from a File

To read to an existing file, you must add the following parameter to the open() function:

- "r" - Read - Default value. Opens a file for reading, error if the file does not exist

Example 9.3.1. Read Data from a File

```
f = open("myfile.txt", "r")

data = f.read()

print(data)

f.close()
```

Listing 9.2: Read Data from a File

[End of Example]

9.4 Logging Data to File

Typically you want to write multiple data to the, e.g., assume you read some temperature data at regular intervals and then you want to save the temperature values to a File.

Example 9.4.1. Logging Data to File

```python
data = [1.6, 3.4, 5.5, 9.4]

f = open("myfile.txt", "x")

for value in data:
    record = str(value)
    f.write(record)
    f.write("\n")

f.close()
```

Listing 9.3: Logging Data to File

[End of Example]

Example 9.4.2. Read Logged Data from File

```python
f = open("myfile.txt", "r")

for record in f:
    record = record.replace("\n", "")
    print(record)

f.close()
```

Listing 9.4: Read Logged Data from File

[End of Example]

9.5 Web Resources

Below you find different useful resources for File Handling.

Python File Handling - w3school:
https://www.w3schools.com/python/python$_{f}$*ile*$_{h}$*andling.asp*

Reading and Writing Files - python.org:
https://docs.python.org/3/tutorial/inputoutput.htmlreading-and-writing-files

9.6 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 9.6.1. Data Logging

Assume you have the following data you want to log to a File as shown in Table 9.1.
Log these data to a File.

Create another Python Script that reads the same data.

[End of Exercise]

Exercise 9.6.2. Data Logging 2

Assume you read data from a Temperature sensor every 10 seconds for a period of let say 5 minutes.

Log the data to a File.

You can use the Random Generator in Python. An example of how to use the Random Generator is shown below:

```python
import random
for x in range(10):
    data = random.randint(1,31)
    print(data)
```

Listing 9.5: Read Data from a File

Make sure to log both the time and the temperature value

Create another Python Script that reads the same data.

You should also plot the data you read from the File.

[End of Exercise]

Table 9.1: Logged Data

Time	Value
1	22
2	25
3	28
...	...

Chapter 10

Error Handling in Python

10.1 Introduction to Error Handling

So far error messages haven't been discussed. You could say that we have 2 kinds of errors: syntax errors and exceptions.

10.1.1 Syntax Errors

Below we see an example of syntax errors:

```
>>> print(Hello World)
  File "<ipython-input-1-10cb182148e3>", line 1
    print(Hello World)
                    ^
SyntaxError: invalid syntax
```

In the example we have written print(Hello World) instead of print("Hello World") and then the Python Interpreter gives us an error message.

10.1.2 Exceptions

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called exceptions and are not unconditionally fatal: you will soon learn how to handle them in Python programs. Most exceptions are not handled by programs, however, and result in error messages as shown here:

```
>>> 10 * (1/0)
Traceback (most recent call last):

  File "<ipython-input-2-0b280f36835c>", line 1, in <module>
    10 * (1/0)

ZeroDivisionError: division by zero
```

or:

```
>>> '2' + 2
Traceback (most recent call last):

  File "<ipython-input-3-d2b23a1db757>", line 1, in <module>
    '2' + 2

TypeError: must be str, not int
```

10.2 Exceptions Handling

It is possible to write programs that handle selected exceptions.

In Python we can use the following built-in Exceptions Handling features:

- The **try** block lets you test a block of code for errors.
- The **except** block lets you handle the error.
- The **finally** block lets you execute code, regardless of the result of the try- and except blocks.

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the **try - except** statements.

Some basic example:

```
try:
    10 * (1/0)
except:
    print("The calculation failed")
```

or:

```
try:
  print(x)
except:
  print("x is not defined")
```

You can also use multiple exceptions:

```
try:
  print(x)
except NameError:
  print("x is not defined")
except:
  print("Something is wrong")
```

The finally block, if specified, will be executed regardless if the try block raises an error or not.

Example:

```
x=2

try:
  print(x)
except NameError:
  print("x is not defined")
except:
  print("Something is wrong")
finally:
    print("The Program is finished")
```

In general you should use try - except - finally when you try to open a File, read or write to Files, connect to a Database, etc.

Example:

```
try:
  f = open("myfile.txt")
  f.write("Lorum Ipsum")
except:
  print("Something went wrong when writing to the file")
finally:
  f.close()
```

Chapter 11

Debugging in Python

Debugging is the process of finding and resolving defects or problems within a computer program that prevent correct operation of computer software or a system [14].

Debuggers are software tools which enable the programmer to monitor the execution of a program, stop it, restart it, set breakpoints, and change values in memory. The term debugger can also refer to the person who is doing the debugging.

As a programmer, one of the first things that you need for serious program development is a debugger.

Python has a built-in debugger that can be used if you are coding Python with a basic text editor and running your Python programs from the command line.

A better option is to use the Debugging features integrated in your Python Editor. Debugging is typically integrated with the Python Editor you are using.

See the specific chapter for the different Python Editors.

Chapter 12

Installing and using Python Packages

A package contains all the files you need for a module. Modules are Python code libraries you can include in your project.

Since Python is open source you can find thousands of Python Packages that you can install and use in your Python programs.

You can use a Python Distribution like Anaconda Distribution (or similar Python Distributions) to download and install many common Python Packages as mentioned previously.

12.1 What is PIP?

PIP is a package manager for Python packages, or modules if you like. PIP is a tool for installing Python packages.

If you do not have PIP installed, you can download and install it from this page: https://pypi.org/project/pip/

PIP is typically used from the Command Prompt (Windows) or Terminal window (macOS).

Installing Python Packages:

```
pip install packagename
```

Uninstalling Python Packages:

```
pip uninstall packagename
```

Some Python Editors also have a graphical way of installing Python Packages, like, e.g., Visual Studio.

Part III

Python Environments and Distributions

Chapter 13

Introduction to Python Environments and Distributions

Python comes with many flavours and version.

Python is open source and everybody can bundle and distribute Python and different Python Packages.

A Python environment is a context in which you run Python code and includes Python Packages.

An environment consists of an interpreter, a library (typically the Python Standard Library), and a set of installed packages.

These components together determine which language constructs and syntax are valid, what operating-system functionality you can access, and which packages you can use.

You can have multiple Python Environments on your Computer.

Some of them are:

- CPython distribution available from python.org
- Anaconda
- Enthought Canopy
- WinPython
- etc.

It is easy to start using Python by installing one of these Python Distributions.

But you can also install the core Python from:
https://www.python.org

Then install the additional Python Packages you need by using PIP.
https://pypi.org/project/pip/

13.1 Package and Environment Managers

The two most popular tools for installing Python Packages and setting up Python environments are:

- PIP - a Python Package Manager
- Conda - a Package and Environment Manager (for Python and other languages)

13.1.1 PIP

Web:
https://pypi.org

PIP is typically used from the Command Prompt (Windows) or Terminal window (macOS).

Installing Python Packages:

```
pip install packagename
```

Uninstalling Python Packages:

```
pip uninstall packagename
```

13.1.2 Conda

Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux. Conda installs, runs and updates packages and their dependencies.

The Conda package and environment manager is included in all versions of Anaconda.

Conda was created for Python programs, but it can package and distribute software for any language.

Conda allows you to to also create separate environments containing files, packages and their dependencies that will not interact with other environments.

Web:
https://conda.io/

Conda is part of or integrated with the Anaconda Python Distribution.

Web:
https://www.anaconda.com

13.2 Python Virtual Environments

Python ”Virtual Environments” allow Python packages to be installed in an isolated location for a particular application, rather than being installed globally.

You can have multiple Python Environments on your computer.

Python Virtual Environments have their own installation directories and they don't share libraries with other virtual environments.

Python ”Virtual Environments” is handy when you have different Python applications that needs different versions of Python or different version of the Python Packages you are using.

Chapter 14

Anaconda

Anaconda is not an Editor, but a Python Distribution package. Spyder is included in the Python Distribution package. You can also use Anaconda to install other Editors or Python packages.

It is available for Windows, macOS and Linux.

Web:
https://www.anaconda.com

Wikipedia:
https://en.wikipedia.org/wiki/Anaconda$_(Python_distribution)$

14.1 Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage Python packages. The Anaconda Navigator can search for packages and install them on your computer, run the packages and update them.

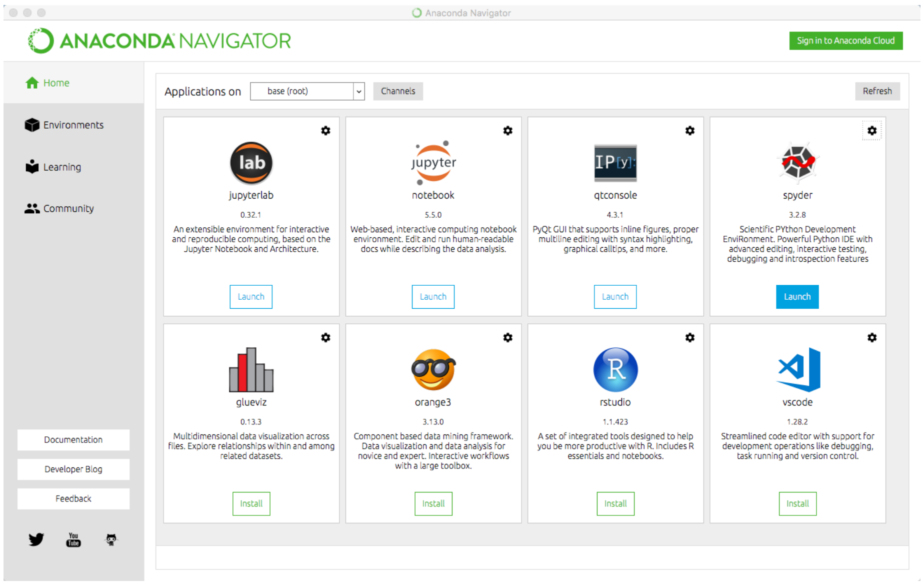
Figure 14.1 shows the Anaconda Navigator.

14.2 Anaconda Prompt

You can use the Anaconda Prompt if you need to install extra Python packages, etc.

Let say you want to install the Python Control Systems Library package. Just enter the following in the Anaconda Prompt:

```
pip install control
```



jupyterlab
0.32.1
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

notebook
5.5.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

qtconsole
4.3.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

spyder
3.2.8
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

glueviz
0.13.3
Multidimensional data visualization across files. Explore relationships within and among related datasets.

orange3
3.13.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

rstudio
1.1.423
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

vscode
1.28.2
Streamlined code editor with support for development operations like debugging, task running and version control.

Figure 14.1: Anaconda Navigator

Python Package Index, or just pip, is a tool used to handle and install Python packages.

For for information about pip and different packages you can install, see the following:

https://pypi.org
Figure 14.2 shows where you can find the Anaconda Prompt. Windows: Search for Anaconda Prompt in the Search field in the start menu.

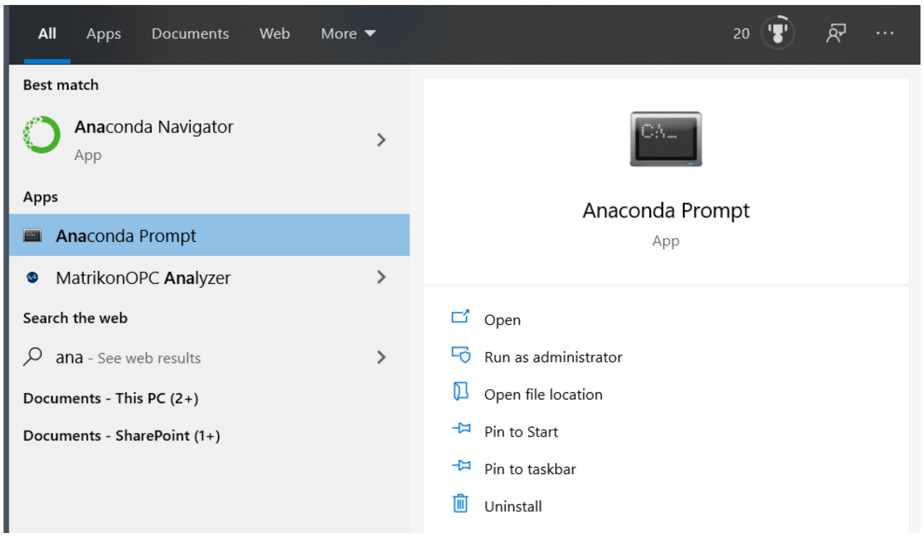


Figure 14.2: Anaconda Navigator

Chapter 15

Enthought Canopy

Enthought Canopy is a Python Platform or Python Distribution for Scientists and Engineers.

It is available for Windows, macOS and Linux.

Canopy is freely available to all users under the Canopy license. Canopy provides access to several hundreds Python packages, including NumPy, SciPy, Pandas, Matplotlib, and IPython.

In addition, we have the Canopy Python Editor.

Enthought Canopy is a competitor to the Anaconda Python Distribution. It is a matter of taste who you prefer.

Web:
https://www.enthought.com/product/canopy/

Part IV

Python Editors

Chapter 16

Python Editors

An Editor is a program where you create your code (and where you can run and test it). Most Editors have also features for Debugging and IntelliSense.

In theory, you can use Windows Notepad for creating Python programs, but in practice it is impossible to create programs without having an editor with Debugging, IntelliSense, color formatting, etc.

For simple Python programs you can use the IDLE Editor, but for more advanced programs a better editor is recommended.

Examples of Python Editors:

- Spyder
- Visual Studio Code
- Visual Studio
- PyCharm
- Wing
- JupyterNotebook

We will give an overview of these Code Editors in the next chapters.

I guess hundreds of different editors can be used for Python Programming, either out of the box or if you install an additional Extension that makes sure you can use Python in that editor.

If you already have a favorite Code Editor, it is a good change you can use that one for Python programming.

Which editor you should use depends on your background, what kind of code editors you have used previously, your programming skills, what your are going to develop in Python, etc.

If you are familiar with MATLAB, Spyder is recommended. Also, if you want to use Python for numerical calculations and computations, Spyder is a good choice.

If you want to create Web Applications or other kinds of Applications, other Editors are probably better to use.

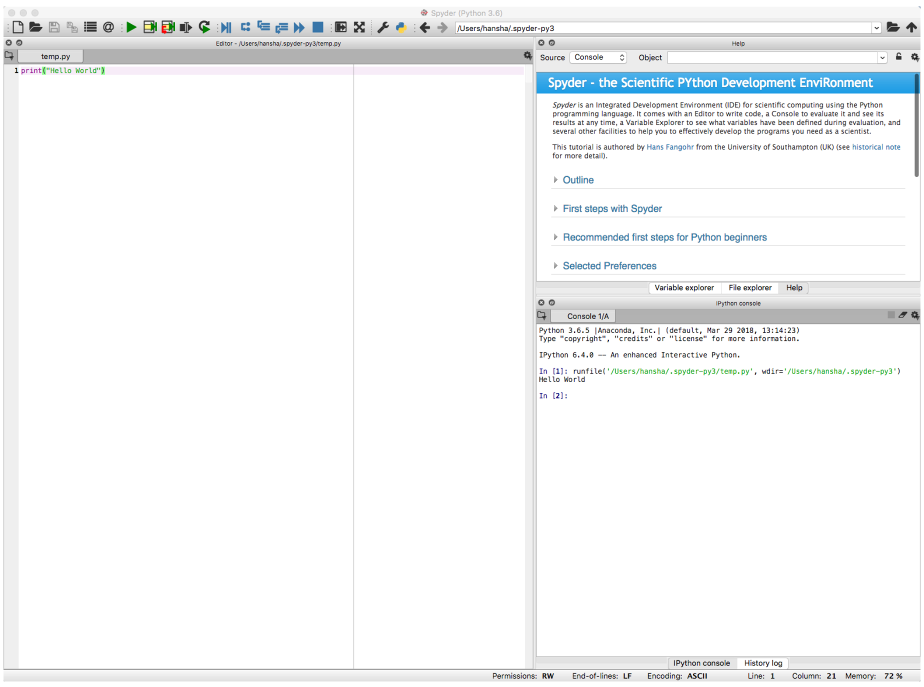
For a list of ”Best Python Editors”, see [15].

Chapter 17

Spyder

Spyder - short for "Scientific PYthon Development EnviRonment".

Spyder is an open source cross-platform integrated development environment(IDE) for scientific programming in the Python language.



```
print("Hello World")
```

Spyder - the Scientific PYthon Development EnviRonment

Spyder is an Integrated Development Environment (IDE) for scientific computing using the Python programming language. It comes with an Editor to write code, a Console to evaluate it and see its results at any time, a Variable Explorer to see what variables have been defined during evaluation, and several other facilities to help you to effectively develop the programs you need as a scientist.

This tutorial is authored by Hans Fangohr from the University of Southampton (UK) (see historical note for more detail).

- Outline
- First steps with Spyder
- Recommended first steps for Python beginners
- Selected Preferences

```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:14:23)
Type "copyright", "credits" or "license" for more information.

IPython 6.4.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/hansha/.spyder-py3/temp.py', wdir='/Users/hansha/.spyder-py3')
Hello World

In [2]:
```

Figure 17.1: Spyder Editor

The Spyder editor consists of the following parts or windows:

- Code Editor window
- iPython Console window

- Variable Explorer
- etc.

Web:
https://www.spyder-ide.org

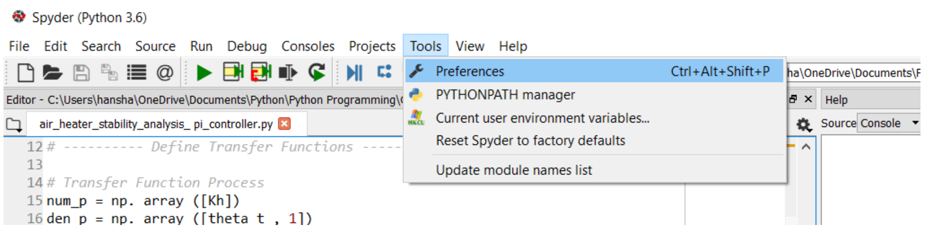
If you have used MATLAB previously or want to use Python for scientific use, Spyder is a good choice. it is easy to install using the Anaconda Distribution.

Web:
https://www.anaconda.com

17.1 Configuration

Typically you want to show figures and plots in separate windows.

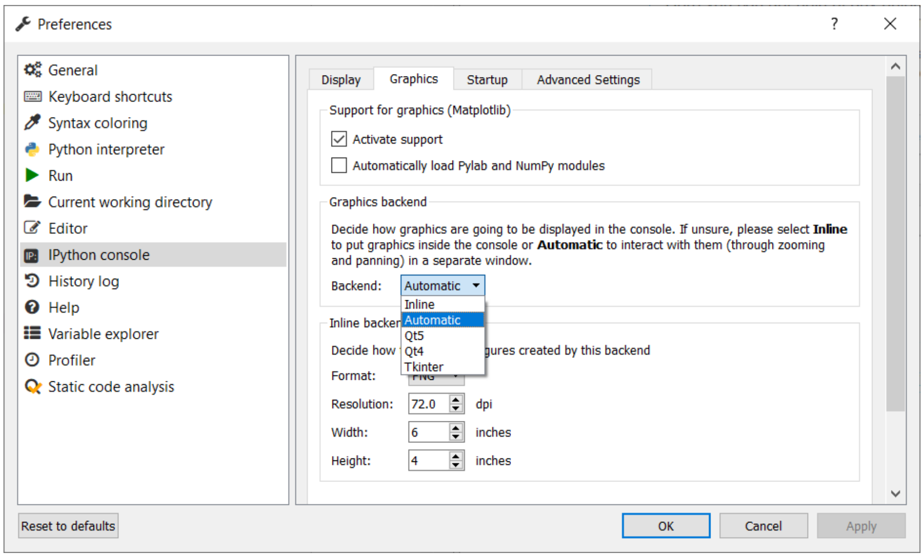
Select Tools-Preferences as shown in Figure 17.2.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Preferences Ctrl+Alt+Shift+P
PYTHONPATH manager
Current user environment variables...
Reset Spyder to factory defaults
Update module names list
Editor - C:\Users\hansha\OneDrive\Documents\Python\Python Programming\
air_heater_stability_analysis_ pi_controller.py
Help
Source Console

```
# ---------- Define Transfer Functions -----

# Transfer Function Process
num_p = np. array ([Kh])
den_p = np. array ([theta_t , 1])
```

Figure 17.2: Python Tools-Preferences

Then select "Automatic" as shown in Figure 17.3.



Preferences

General
Keyboard shortcuts
Syntax coloring
Python interpreter
Run
Current working directory
Editor
IPython console
History log
Help
Variable explorer
Profiler
Static code analysis

Display | Graphics | Startup | Advanced Settings

Support for graphics (Matplotlib)

☑ Activate support

☐ Automatically load Pylab and NumPy modules

Graphics backend

Decide how graphics are going to be displayed in the console. If unsure, please select **Inline** to put graphics inside the console or **Automatic** to interact with them (through zooming and panning) in a separate window.

Backend: Automatic

Inline
Automatic
Qt5
Qt4
Tkinter

Inline backer

Decide how ... gures created by this backend

Format: PNG

Resolution: 72.0 dpi

Width: 6 inches

Height: 4 inches

Reset to defaults | OK | Cancel | Apply

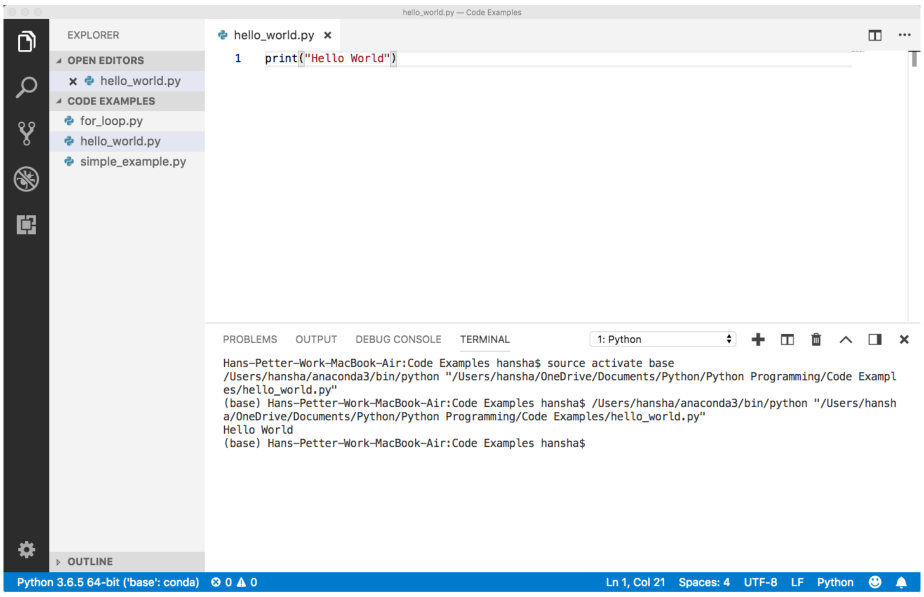
Figure 17.3: Python Preferences window

Chapter 18

Visual Studio Code

18.1 Introduction to Visual Studio Code

Visual Studio Code is a simple and easy to use editor that can be used for many different programming languages.



```python
print("Hello World")
```

```
Hans-Petter-Work-MacBook-Air:Code Examples hansha$ source activate base
/Users/hansha/anaconda3/bin/python "/Users/hansha/OneDrive/Documents/Python/Python Programming/Code Exampl
es/hello_world.py"
(base) Hans-Petter-Work-MacBook-Air:Code Examples hansha$ /Users/hansha/anaconda3/bin/python "/Users/hansh
a/OneDrive/Documents/Python/Python Programming/Code Examples/hello_world.py"
Hello World
(base) Hans-Petter-Work-MacBook-Air:Code Examples hansha$
```

Figure 18.1: Using Visual Studio Code as Python Editor

Right-Click and select ”Run Python File in Terminal”

Web:
https://code.visualstudio.com

Wikipedia:
https://en.wikipedia.org/wiki/Visual$_{studio}{}_{C}ode$

18.2 Python in Visual Studio Code

In addition to Visual Studio Code you need to install the Python extension for Visual Studio Code.

You must install a Python interpreter yourself separately from the extension. For a quick install, use Python from python.org.

https://www.python.org

Python is an interpreted language, and in order to run Python code and get Python IntelliSense within Visual Studio Code, you must tell Visual Studio Code which interpreter to use.

Web:
https://code.visualstudio.com/docs/languages/python

Chapter 19

Visual Studio

19.1 Introduction to Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. The default (main) programming language in Visual studio is C, but many other programming languages are supported.

You could say Visual Studio is the big brother of Visual Studio Code.

Visual studio is available for Windows and macOS.

Visual Studio (from 2017), has integrated support for Python, it is called "Python Support in Visual Studio".

Web:
https://visualstudio.microsoft.com

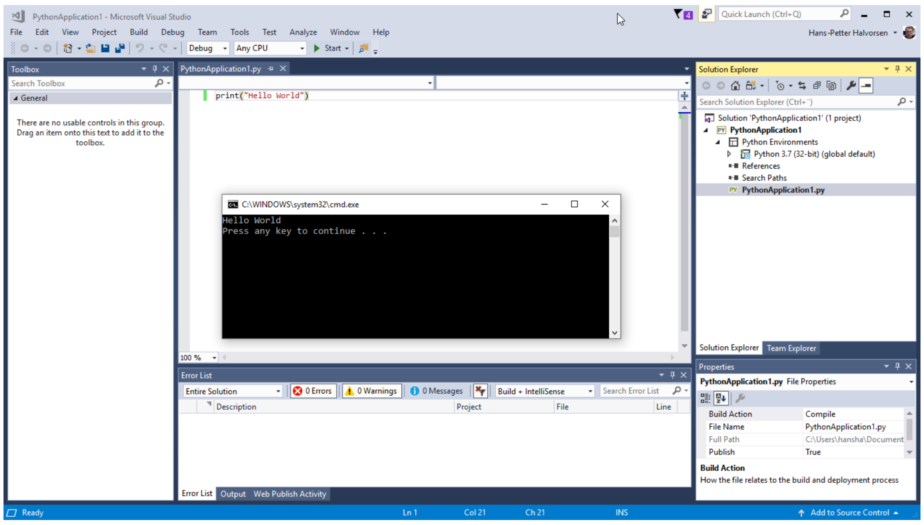
Wikipedia:
https://en.wikipedia.org/wiki/Microsoft$_{V}isual_{S}tudio$

Go to my Web Site to learn more about Visual Studio and C programming:
https://www.halvorsen.blog/

Visual Studio and C:
https://www.halvorsen.blog/documents/programming/csharp/

19.2 Work with Python in Visual Studio

Work with Python in Visual Studio:
https://docs.microsoft.com/visualstudio/python/



```
print("Hello World")
```

```
Hello World
Press any key to continue . . .
```

Build Action	Compile
File Name	PythonApplication1.py
Full Path	C:\Users\hansha\Document
Publish	True

Figure 19.1: Using Visual Studio as Python Editor

19.2.1 Make Visual Studio ready for Python Programming

Visual Studio is mainly for Windows. A MacOS version of Visual Studio do exists, but it has lot less features than the Windows edition.

Note that Python support is available only on Visual Studio for Windows. If you use Mac and Linux, you need to use Visual Studio Code. You could say Visual Studio Code is a down-scaled version of Visual Studio.

Visual Studio (from 2017), has integrated support for Python, it is called "Python Support in Visual Studio". Even if it is integrated, you need to manually select which components you want to install on your computer. Make sure to download and run the latest Visual Studio 2017 installer for Windows.

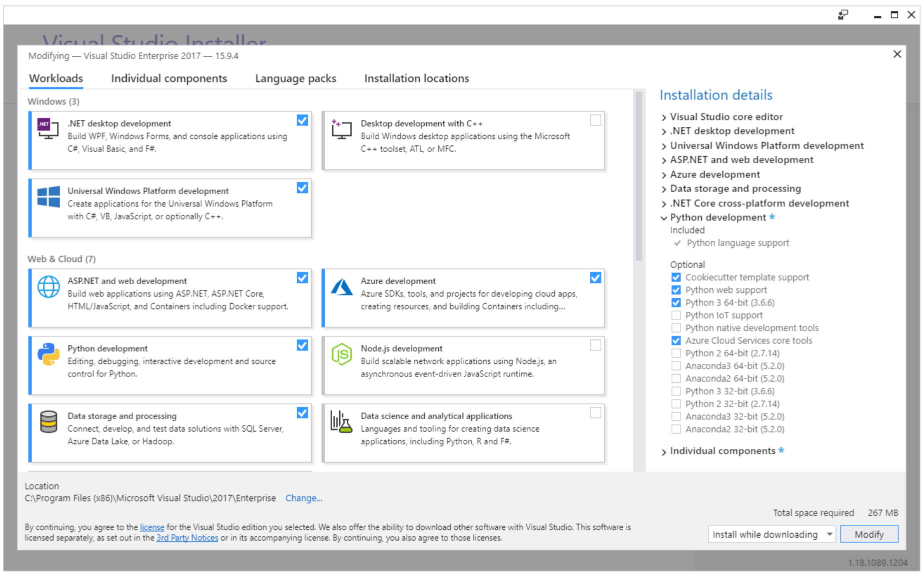
when you run the Visual Studio installer (either for the first time or if you already have installed Visual Studio 2017 and want to modify it) the window shown in Figure 19.2 pops up.
The installer presents you with a list of so called workloads, which are groups of related options for specific development areas. For Python, select the "Python development" workload and select Install (Figure 19.3).

19.2.2 Python Interactive

To quickly test Python support, launch Visual Studio, press Alt+I (or select from the menu: Tools - Python - Python Interactive Window) to open the Python Interactive window. See Figure 19.4.

Lets write something like this:

```
>>> a = 2
```



Modifying — Visual Studio Enterprise 2017 — 15.9.4

Windows (3)

[✓] .NET desktop development
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F#.

[ ] Desktop development with C++
Build Windows desktop applications using the Microsoft C++ toolset, ATL, or MFC.

[✓] Universal Windows Platform development
Create applications for the Universal Windows Platform with C#, VB, JavaScript, or optionally C++.

Web & Cloud (7)

[✓] ASP.NET and web development
Build web applications using ASP.NET, ASP.NET Core, HTML/JavaScript, and Containers including Docker support.

[✓] Azure development
Azure SDKs, tools, and projects for developing cloud apps, creating resources, and building Containers including...

[✓] Python development
Editing, debugging, interactive development and source control for Python.

[ ] Node.js development
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

[✓] Data storage and processing
Connect, develop, and test data solutions with SQL Server, Azure Data Lake, or Hadoop.

[ ] Data science and analytical applications
Languages and tooling for creating data science applications, including Python, R and F#.

Installation details

- Visual Studio core editor
- .NET desktop development
- Universal Windows Platform development
- ASP.NET and web development
- Azure development
- Data storage and processing
- .NET Core cross-platform development
- Python development *
  - Included
    - ✓ Python language support
  - Optional
    - [✓] Cookiecutter template support
    - [✓] Python web support
    - [✓] Python 3 64-bit (3.6.6)
    - [ ] Python IoT support
    - [ ] Python native development tools
    - [✓] Azure Cloud Services core tools
    - [ ] Python 2 64-bit (2.7.14)
    - [ ] Anaconda3 64-bit (5.2.0)
    - [ ] Anaconda2 64-bit (5.2.0)
    - [ ] Python 3 32-bit (3.6.6)
    - [ ] Python 2 32-bit (2.7.14)
    - [ ] Anaconda3 32-bit (5.2.0)
    - [ ] Anaconda2 32-bit (5.2.0)
- Individual components *

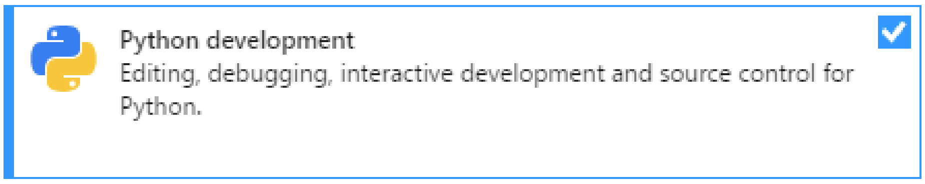
Location
C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 267 MB

1.18.1089.1204

Figure 19.2: Installing Python Extension for Visual Studio



[✓] Python development
Editing, debugging, interactive development and source control for Python.

Figure 19.3: Python Development Workload

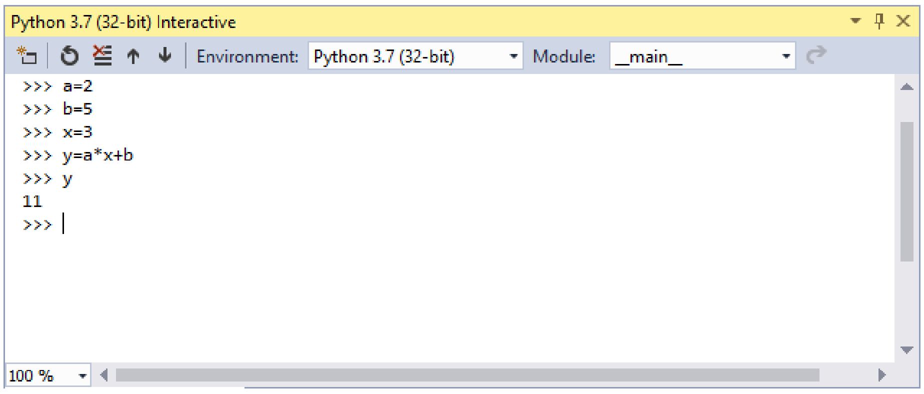
```
>>> b = 5
>>> x = 3
>>> y = a*x + b
>>> y
```

19.2.3 New Python Project

Lets see how we can create a Python Application.

Start by select from the menu: File - New - Project... The New Project window pops up. See Figure 19.5.
We can create an ordinary Python Application (one or more Python Scripts), we can choose to create a Web Application using either Web Frameworks like Django or Flask, or we can create different Desktop GUI applications. We can also create Games.

Example 19.2.1. Python Hello World Application in Visual Studio


Python 3.7 (32-bit) Interactive
Environment: Python 3.7 (32-bit)
Module: __main__
100 %

```
>>> a=2
>>> b=5
>>> x=3
>>> y=a*x+b
>>> y
11
>>> |
```

Figure 19.4: Python Interactive

We start by creating a basic Hello World Python Application. See Figure 19.1. Select File - New - Project... The New Project window pops up. See Figure 19.5.

Name the project, e.g, "PythonApplication1".
In the Project Explorer, open the "PythonApplication1.py" file and enter the following Python code:

```
print("Hello World")
```

Hit F5 (our click the green arrow) in order to run or execute the Python program. You can also right click on the file and select "Start without Debugging".

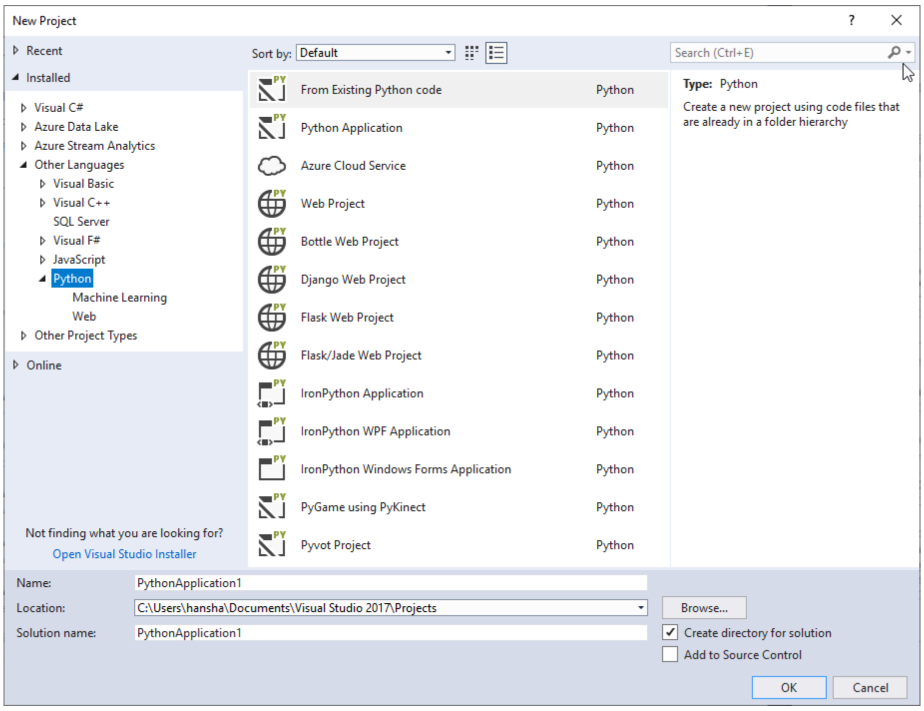
[End of Example]

Example 19.2.2. Visual Studio Python Plotting

Create a new Python File by right click in the Solution Explorer and select Add - New Item... and then select "Empty Python File".

Enter the following Python Code:

```
import matplotlib.pyplot as plt
import numpy as np

xstart = 0
xstop = 2*np.pi
increment = 0.1

x = np.arange(xstart,xstop,increment)

y = np.sin(x)

plt.plot(x, y)
plt.title('y=sin(x)')
```



From Existing Python code	Python
Python Application	Python
Azure Cloud Service	Python
Web Project	Python
Bottle Web Project	Python
Django Web Project	Python
Flask Web Project	Python
Flask/Jade Web Project	Python
IronPython Application	Python
IronPython WPF Application	Python
IronPython Windows Forms Application	Python
PyGame using PyKinect	Python
Pyvot Project	Python

Type: Python

Create a new project using code files that are already in a folder hierarchy

Name: PythonApplication1

Location: C:\Users\hansha\Documents\Visual Studio 2017\Projects

Solution name: PythonApplication1

[✓] Create directory for solution

[ ] Add to Source Control

Figure 19.5: New Python Project

```
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.axis([0, 2*np.pi, −1, 1])
plt.show()
```

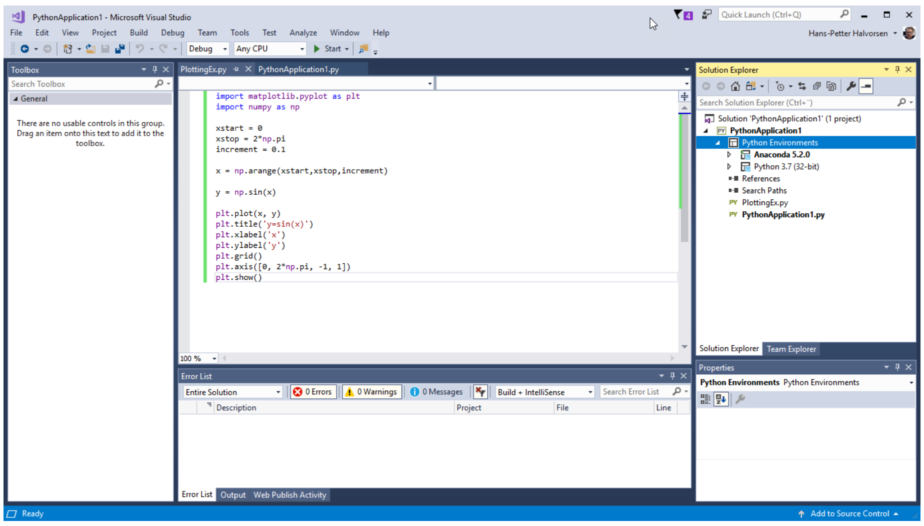
See also Figure 19.6.
Make sure to select proper Python Environment. See Figure (19.7). Visual Studio supports multiple Python Environments.

In this example we use the Matplotlib package for plotting, so we need to have that package installed on the computer. You can install the Matplotlib package in different Python Environments.

I have installed the Matplotlib package as part of the Anaconda distribution setup, so I select "Anaconda x.x.x" in the Python Environments window.

If you haven't installed the Matplotlib package yet (either as part of Anaconda or manually using PIP), you can also easily install Python packages from Visual studio. See Figure 19.8.

You can also easily see which Python Packages that are installed for the different Python Environments. See Figure 19.9.

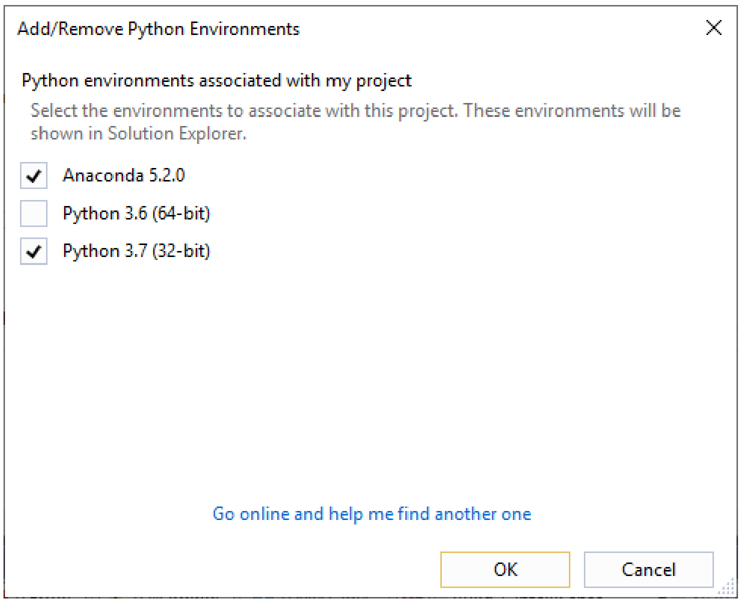


```
import matplotlib.pyplot as plt
import numpy as np

xstart = 0
xstop = 2*np.pi
increment = 0.1

x = np.arange(xstart,xstop,increment)

y = np.sin(x)

plt.plot(x, y)
plt.title('y=sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.axis([0, 2*np.pi, -1, 1])
plt.show()
```

Figure 19.6: Python Plotting Example with Visual Studio

The good thing about using Visual Studio is that you have a graphical user interface for everything, you don't need to use the Command window etc. for installing Python Packages, etc.
Hit F5 (our click the green arrow) in order to run or execute the Python program. You can also right click on the file and select "Start without Debugging".
We get the following results, see Figure 19.10.

[End of Example]


Add/Remove Python Environments

Python environments associated with my project

Select the environments to associate with this project. These environments will be shown in Solution Explorer.

- [✓] Anaconda 5.2.0
- [ ] Python 3.6 (64-bit)
- [✓] Python 3.7 (32-bit)

Figure 19.7: Select your Python Environment

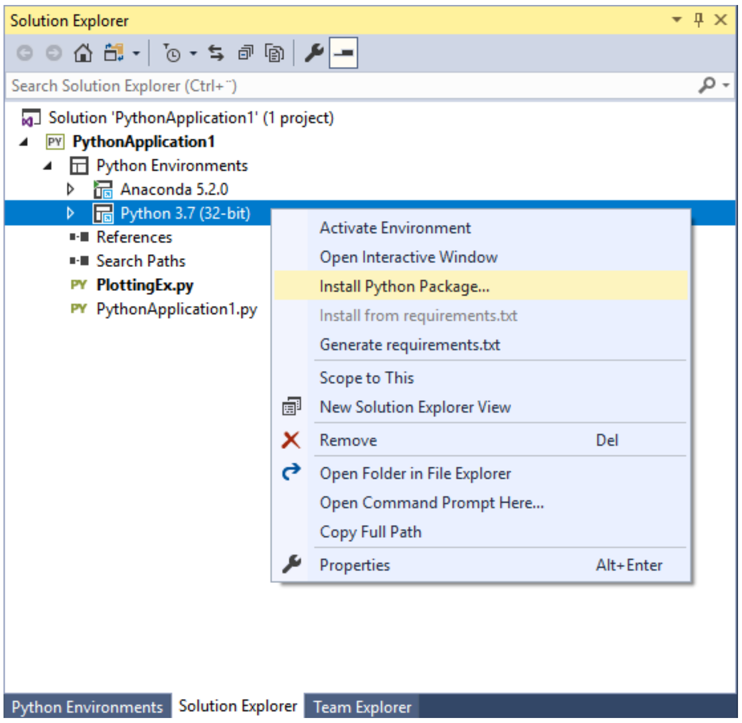
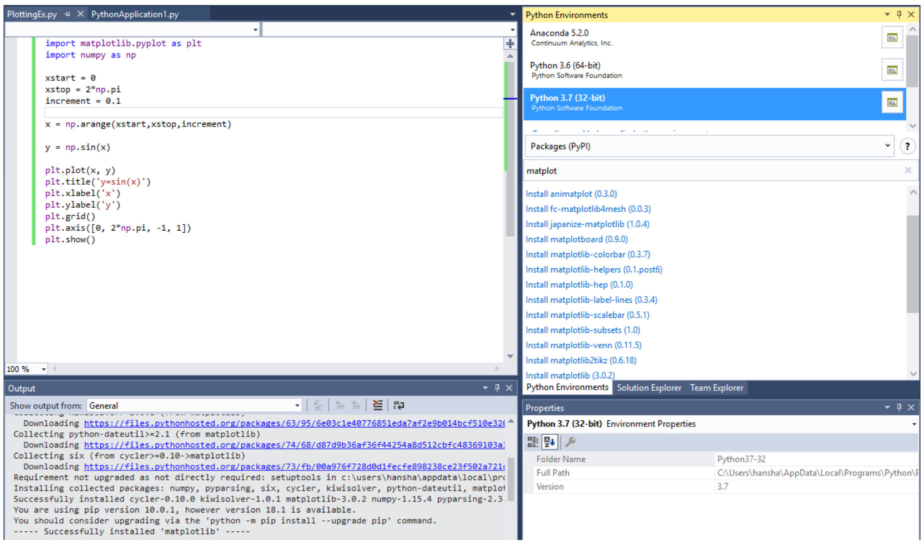


Figure 19.8: Install Python Packages from Visual Studio


PlottingEx.py
PythonApplication1.py
Python Environments
Anaconda 5.2.0
Continuum Analytics, Inc.
Python 3.6 (64-bit)
Python Software Foundation
Python 3.7 (32-bit)
Python Software Foundation
Packages (PyPI)
matplot
Install animatplot (0.3.0)
Install fc-matplotlib4mesh (0.0.3)
Install japanize-matplotlib (1.0.4)
Install matplotboard (0.9.0)
Install matplotlib-colorbar (0.3.7)
Install matplotlib-helpers (0.1.post6)
Install matplotlib-hep (0.1.0)
Install matplotlib-label-lines (0.3.4)
Install matplotlib-scalebar (0.5.1)
Install matplotlib-subsets (1.0)
Install matplotlib-venn (0.11.5)
Install matplotlib2tikz (0.6.18)
Install matplotlib (3.0.2)
Python Environments
Solution Explorer
Team Explorer
100 %
Output
Show output from: General
Properties
Python 3.7 (32-bit) Environment Properties

```python
import matplotlib.pyplot as plt
import numpy as np

xstart = 0
xstop = 2*np.pi
increment = 0.1

x = np.arange(xstart,xstop,increment)

y = np.sin(x)

plt.plot(x, y)
plt.title('y=sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.axis([0, 2*np.pi, -1, 1])
plt.show()
```

```
  Downloading https://files.pythonhosted.org/packages/63/95/6e03c1e40776851eda7af2e9b014bcf510e32
Collecting python-dateutil>=2.1 (from matplotlib)
  Downloading https://files.pythonhosted.org/packages/74/68/d87d9b36af36f44254a8d512cbfc48369103a
Collecting six (from cycler>=0.10->matplotlib)
  Downloading https://files.pythonhosted.org/packages/73/fb/00a976f728d0d1fecfe898238ce23f502a721
Requirement not upgraded as not directly required: setuptools in c:\users\hansha\appdata\local\pro
Installing collected packages: numpy, pyparsing, six, cycler, kiwisolver, python-dateutil, matplot
Successfully installed cycler-0.10.0 kiwisolver-1.0.1 matplotlib-3.0.2 numpy-1.15.4 pyparsing-2.3
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
----- Successfully installed 'matplotlib' -----
```

Folder Name	Python37-32
Full Path	C:\Users\hansha\AppData\Local\Programs\Python\P
Version	3.7

Figure 19.9: Installing Python Packages for different Python Environments from Visual Studio

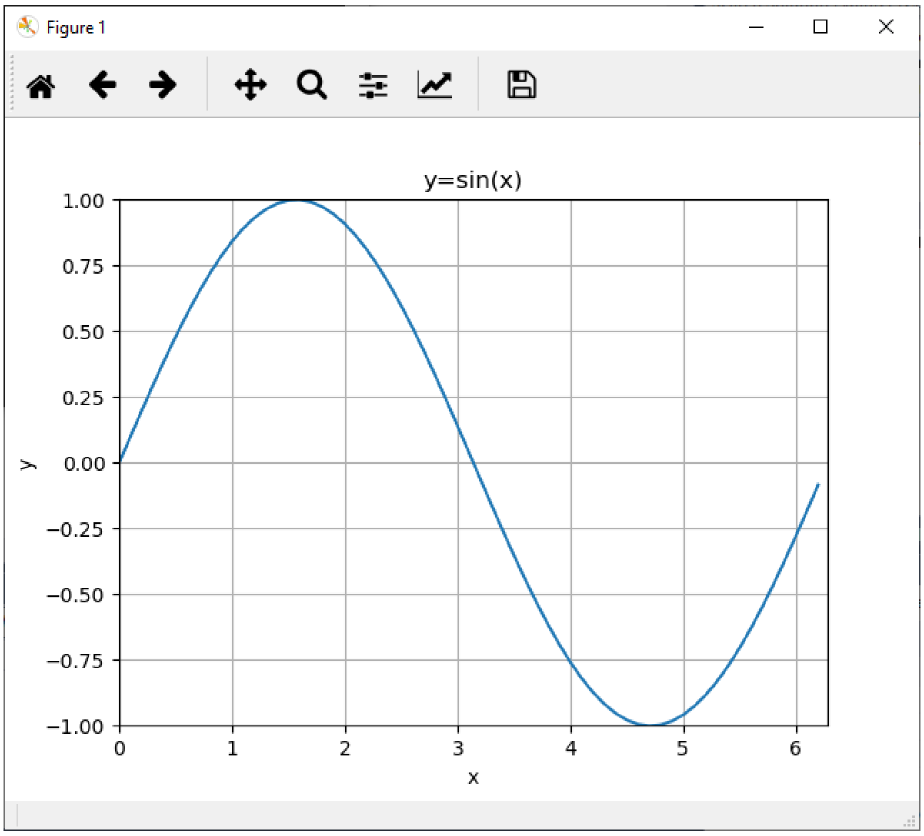


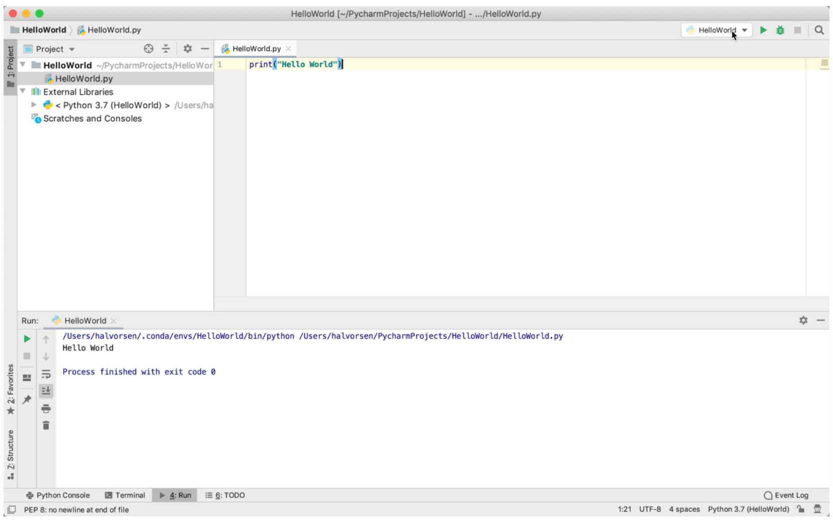
Figure 19.10: Python Plotting Example with Visual Studio

Chapter 20

PyCharm

PyCharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is free to use, while the Professional Edition (paid version) has some extra features.

The PyCharm Editor is shown in Figure 20.1.


HelloWorld [~/PycharmProjects/HelloWorld] - .../HelloWorld.py
HelloWorld
HelloWorld.py
Project
HelloWorld ~/PycharmProjects/HelloWor
External Libraries
< Python 3.7 (HelloWorld) > /Users/ha
Scratches and Consoles
Run:
Python Console
Terminal
4: Run
6: TODO
Event Log
PEP 8: no newline at end of file
1:21 UTF-8 4 spaces Python 3.7 (HelloWorld)

```
print("Hello World")
```

```
/Users/halvorsen/.conda/envs/HelloWorld/bin/python /Users/halvorsen/PycharmProjects/HelloWorld/HelloWorld.py
Hello World

Process finished with exit code 0
```

Figure 20.1: PyCharm Python Editor

Web:
https://www.jetbrains.com/pycharm/

Wikipedia:
https://en.wikipedia.org/wiki/PyCharm

Anaconda and JetBrains also have a collaboration and offer what they call PyCharm for Anaconda. You can download it here:

https://www.jetbrains.com/pycharm/promo/anaconda/

We have code editors like Visual Studio and Visual Studio Code which can be used for many different programming languages by installing different types of plugins.

Editors like Spyder and PyCharm are tailor-made editors for the Python language.

Spyder is light-weight IDE typically used for scientific use. PyCharm on the other hand is full-blown IDE for software development in general by using the Python language. It supports many plugins, it's easier to program Django, etc.

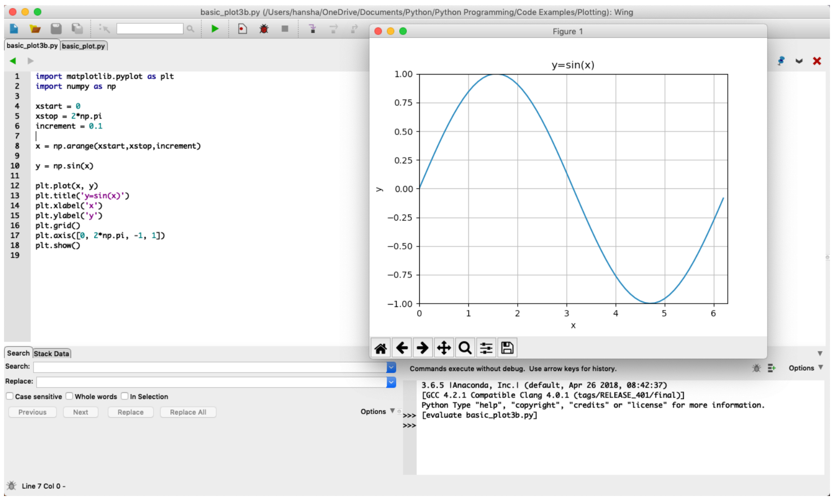
Chapter 21

Wing Python IDE

The Wing Python IDE family of integrated development environments (IDEs) from Wingware were created specifically for the Python programming language.

3 different version of Wing exists [12]:

- **Wing 101** – a very simplified free version, for teaching beginning programmers
- **Wing Personal** – free version that omits some features, for students and hobbyists
- **Wing Pro** – a full-featured commercial (paid) version, for professional programmers



```
import matplotlib.pyplot as plt
import numpy as np

xstart = 0
xstop = 2*np.pi
increment = 0.1

x = np.arange(xstart,xstop,increment)

y = np.sin(x)

plt.plot(x, y)
plt.title('y=sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.axis([0, 2*np.pi, -1, 1])
plt.show()
```

```
3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Python Type "help", "copyright", "credits" or "license" for more information.
>>> [evaluate basic_plot3b.py]
>>>
```

Figure 21.1: Wing Python IDE

Web:
https://wingware.com

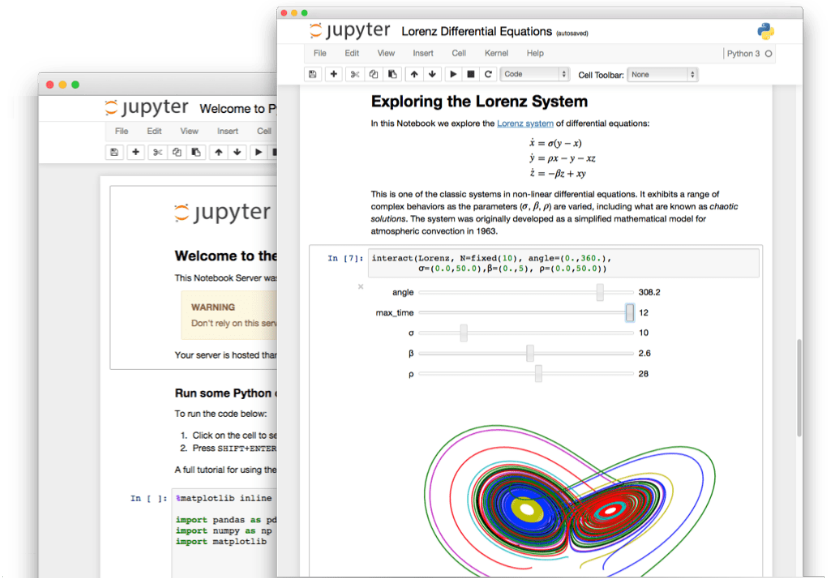
Wikipedia:
https://en.wikipedia.org/wiki/Wing$_IDE$

Chapter 22

Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and text.

The Notebook has support for over 40 programming languages, including Python.



```
In [ ]: %matplotlib inline

        import pandas as pd
        import numpy as np
        import matplotlib
```

Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                 σ=(0.0,50.0),β=(0.,5), ρ=(0.0,50.0))
```

Figure 22.1: Jupyter Notebook [16]

Web:
http://jupyter.org

Wikipedia:
https://en.wikipedia.org/wiki/Project$_Jupyter$

22.1 JupyterHub

JupyterHub is a multi-user version of the notebook designed for companies, classrooms and research labs [17].

JupyterHub runs in the cloud or on your own hardware.

JupyterHub is open-source and designed to be run on a variety of infrastructure. This includes commercial cloud providers, virtual machines, or even your own laptop hardware.

Web:
http://jupyter.org/hub

22.2 Microsoft Azure Notebooks

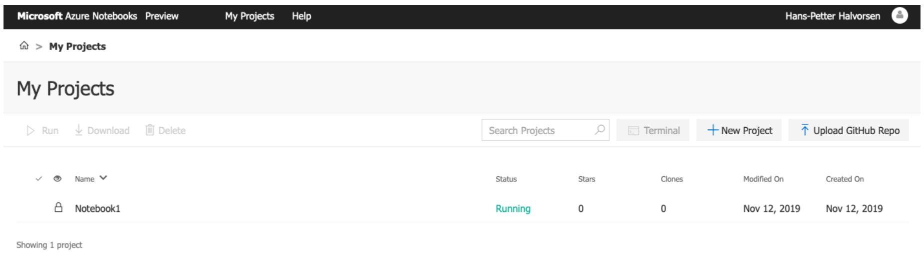
Microsoft Azure Notebooks is a version of Jupyter Notebook from Microsoft.

The good thing about Microsoft Azure Notebooks is that you have the infrastructure and everything up and running ready for you to use. You can use it for free as well.

Web:
https://notebooks.azure.com

Example 22.2.1. Example Name

Figure 22.2 shows an overview of my Azure Notebook Projects.



Name	Status	Stars	Clones	Modified On	Created On
Notebook1	Running	0	0	Nov 12, 2019	Nov 12, 2019

Showing 1 project

Figure 22.2: Azure Notebook Projects

Figure 22.3 shows an overview of my Azure Notebook Project Notebooks.

Figure 22.4 shows an example of a simple Notebook.

[End of Example]

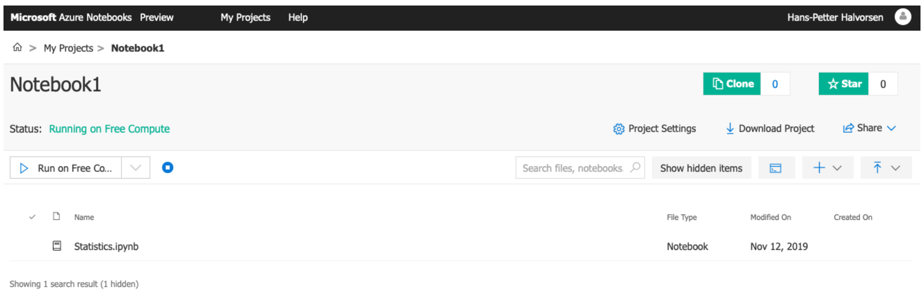
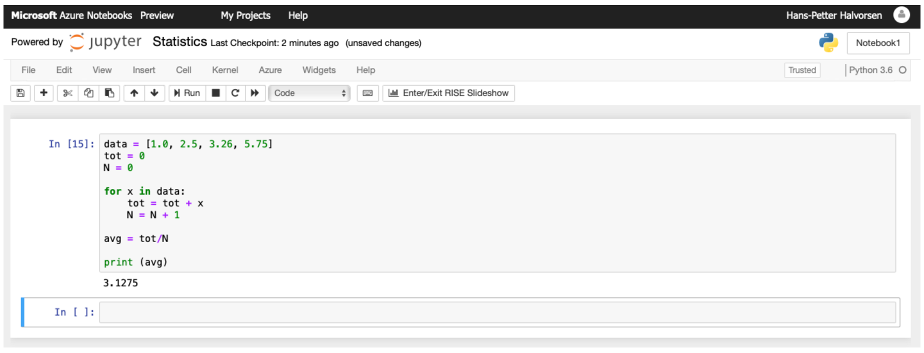


Figure 22.3: Azure Notebook Project Notebooks



```
In [15]: data = [1.0, 2.5, 3.26, 5.75]
         tot = 0
         N = 0

         for x in data:
             tot = tot + x
             N = N + 1

         avg = tot/N

         print (avg)

         3.1275

In [ ]:
```

Figure 22.4: Azure Notebook Example

Part V

Data Acquisition (DAQ) with Python

Chapter 23

Plotting Sensor Data

23.1 Introduction

Typically we want to plot the data from the sensor. We can plot save the data in an array and then plot the data at the end of the program, but more likely we want to plot one value at the time inside the loop, so-called "Real-Time plotting".

In this chapter we only show how you can plot the data from any given sensor using this general approach. Instead of the actual sensor data we just use the random generator in Python.

To read the actual sensor data you typically need a DAQ (Data Acquisition) device connected to you PC or, e.g, a Raspberry Pi device. In all cases you will typically need to install a driver from the vendor of the DAQ device or the sensor you are using.

23.2 Introduction to Real-Time Plotting

You can also use the matplotlib for real-time plotting.

Example 23.2.1. Introduction to Real-Time Plotting

Here is a basic example:

```
import numpy as np
import matplotlib.pyplot as plt

plt.axis([0, 10, 0, 1])

delay = 1 #Seconds

for i in range(10):
    y = np.random.random()
    plt.scatter(i, y)
    plt.pause(delay)

plt.show()
```

Listing 23.1: Real-Time Plotting in Python

We get the following plot as shown in Figure 23.1.

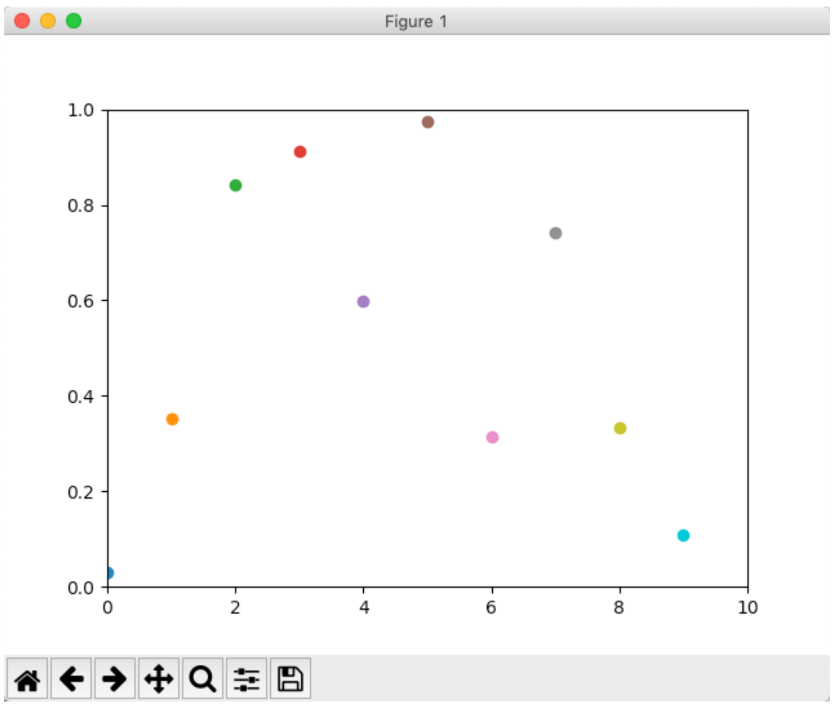


Figure 23.1: Real-Time Plotting with Python

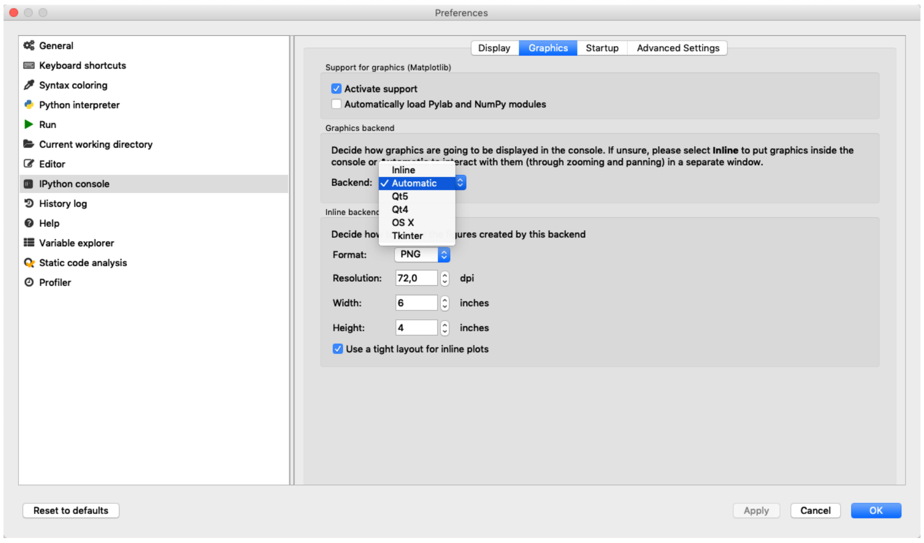
You cannot see the the actual behavior of the plot by watching Figure 23.1, so you need to run the Python program yourself.

If you run the code you see the plot is updated with a new value every second as specified in the code.

[End of Example]

Note! If you use Anaconda and Spyder, you typically need to change the the settings for how graphics are are displayed in Spyder.

Select Preferences from the menu, then IPython console in the list of categories on the left, then the tab Graphics at the top, and change the Graphics back-end from Inline to e.g. Automatic or Qt. See Figure 23.2.



Support for graphics (Matplotlib)

- [✓] Activate support
- [ ] Automatically load Pylab and NumPy modules

Graphics backend

Decide how graphics are going to be displayed in the console. If unsure, please select **Inline** to put graphics inside the console or ~~Automatic to int~~eract with them (through zooming and panning) in a separate window.

Backend: ✓ Automatic

Inline backend

Decide ho~~w to render the fi~~gures created by this backend

Format: PNG

Resolution: 72,0 dpi

Width: 6 inches

Height: 4 inches

- [✓] Use a tight layout for inline plots

Figure 23.2: Change how Graphics are displayed in the Spyder Editor

23.3 Real-Time Plotting with Animation

For more advanced Real-Time plots we should use the animation module in the matplotlib library (matplotlib.animation).

To create a real-time plot, we need to use the animation module in matplotlib. We set up the figure and axes in the usual way, but we draw directly to the axes, ax, when we want to create a new frame in the animation.

We need to use the FuncAnimation function:

```python
ani = animation.FuncAnimation(fig, animate, fargs=(xs, ys),
    interval=1000)
```

FuncAnimation is a special function within the animation module that lets us automate updating the graph. We pass the FuncAnimation() a handle to the figure we want to draw, fig, as well as the name of a function that should be called at regular intervals. We called this function animate() and is defined just above our FuncAnimation() call.

Still in the FuncAnimation() parameters, we set fargs, which are the arguments we want to pass to our animate function (since we are not calling animate() directly from within our own code). Then, we set interval, which is how long we should wait between calls to animate() (in milliseconds).

Note: As an argument to FuncAnimation, notice that animate does not have any parentheses. This is passing a reference to the function and not the result of that function. If you accidentally add parentheses to animate here, animate will be called immediately (only once), and you'll likely get an error

Example 23.3.1. Real-Time Plotting with Animation

Below you find the Python Code for a basic example where we use the animation module in matplotlib.

In the example we update the plot every seconds by setting the interval=1000ms as an input argument to the FuncAnaimation function.

```python
import datetime as dt
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Create figure for plotting
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
xs = []
ys = []


# This function is called periodically from FuncAnimation
def animate(i, xs, ys):

    temp_c = round(np.random.random(), 2)

    # Add x and y to lists
    xs.append(dt.datetime.now().strftime('%H:%M:%S.%f'))
    ys.append(temp_c)

    # Limit x and y lists to 20 items
    xs = xs[-20:]
    ys = ys[-20:]

    # Draw x and y lists
    ax.clear()
    ax.plot(xs, ys)

    # Format plot
    plt.xticks(rotation=45, ha='right')
    plt.subplots_adjust(bottom=0.30)
    plt.title('Temperature Data')
    plt.ylabel('Temperature (deg C)')

# Set up plot to call animate() function periodically
ani = animation.FuncAnimation(fig, animate, fargs=(xs, ys),
    interval=1000)
plt.show()
```

Listing 23.2: Real-Time Plotting with Animation

Figure 23.3 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 23.3, so you need to run the Python program yourself.

[End of Example]

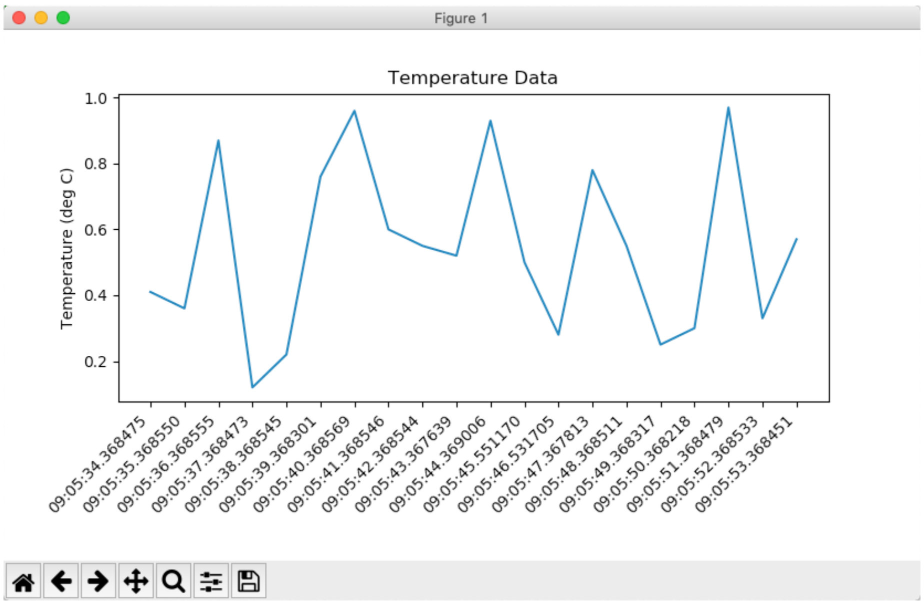


Figure 23.3: Real-Time Plotting with Animation

23.3.1 Speeding Up the Plot Animation

Clearing a graph and redrawing everything can be a time-consuming process in terms of computer time. To remedy that, we are going to use a trick known as blitting.

Blitting is an old computer graphics technique where several graphical bitmaps are combined into one. This way, only one needed to be updated at a time, saving the computer from having to redraw the whole scene every time.
Matplotlib allows us to enable blitting in FuncAnimation, but it means we need to re-write how some of the animate() function works. To reap the true benefits of blitting, we need to set a static background, which means the axes can't scale and we can't show moving timestamps anymore. This means that you have to take the good with the bad. So you have to choose whats most important for you un your simulations.

Example 23.3.2. Real-Time Plotting with Animation with improved Performance

Python Code:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Parameters
x_len = 200          # Number of points to display
y_range = [0, 20]  # Range of possible Y values to display

```

```
# Create figure for plotting
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
xs = list(range(0, 200))
ys = [0] * x_len
ax.set_ylim(y_range)


# Create a blank line. We will update the line in animate
line, = ax.plot(xs, ys)

# Add labels
plt.title('Temperature Data')
plt.xlabel('Samples')
plt.ylabel('Temperature (deg C)')

# This function is called periodically from FuncAnimation
def animate(i, ys):

    rand_val = np.random.random()*20 #Generate Random Values
    between 0 and 20

    temp_c = round(rand_val, 2)

    #print (temp_c)

    # Add y to list
    ys.append(temp_c)

    # Limit y list to set number of items
    ys = ys[-x_len:]

    # Update line with new Y values
    line.set_ydata(ys)

    return line,

# Set up plot to call animate() function periodically
ani = animation.FuncAnimation(fig,
    animate,
    fargs=(ys,),
    interval=100,
    blit=True)
plt.show()
```

Listing 23.3: Real-Time Plotting with Animation

Figure 23.4 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 23.4, so you need to run the Python program yourself.

[End of Example]

For more information about Matplotlib:animations:

https://scipy-cookbook.readthedocs.io/items/Matplotlib$_{A}$*nimations.html*

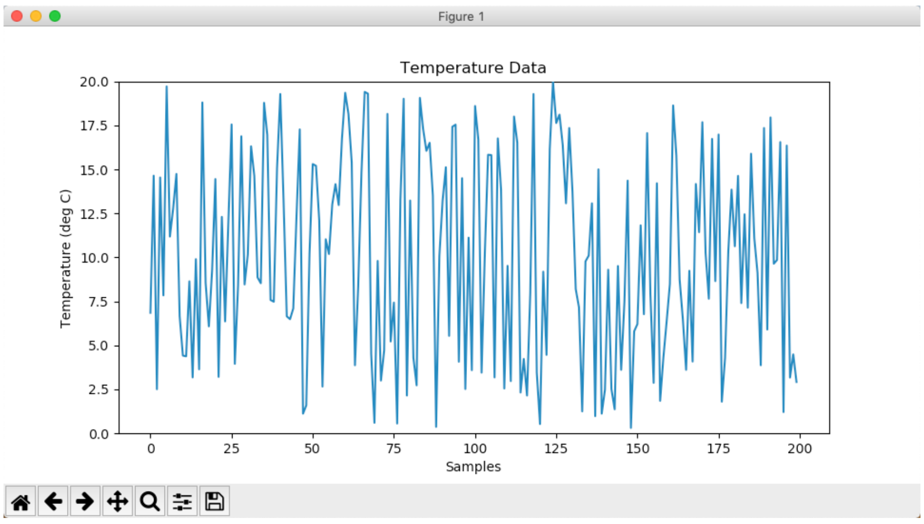


Figure 23.4: Real-Time Plotting with Animation

Other resources:

https://learn.sparkfun.com/tutorials/graph-sensor-data-with-python-and-matplotlib/allplot-sensor-data

https://stackoverflow.com/questions/11874767/how-do-i-plot-in-real-time-in-a-while-loop-using-matplotlib

Chapter 24

Data Acquisition (DAQ) with Python

Python is probably best suited for "ad-hoc" numerical calculations, analysis, simulations, etc., but can be used for many other purposes, even if other programming languages are better suited.

24.1 Introduction to DAQ

To read sensor data you typically need a DAQ (Data Acquisition) device connected to you PC or, e.g, a Raspberry Pi device. In all cases you will typically need to install a driver from the vendor of the DAQ device or the sensor you are using.

A DAQ System consists of 4 parts:

- Physical input/output signals, sensors
- DAQ device/hardware
- Driver software
- Your software application (Application software) - in this case your Python application

Figure 24.1 shows the different steps involved in a DAQ system.

Here you find more information, resources, videos and examples regarding DAQ: https://www.halvorsen.blog/documents/technology/daq/

24.2 Data Acquisition using NI DAQ Devices

Here we will show how we can use Python to retrieve data from the physical world using a DAQ device or an I/O module.

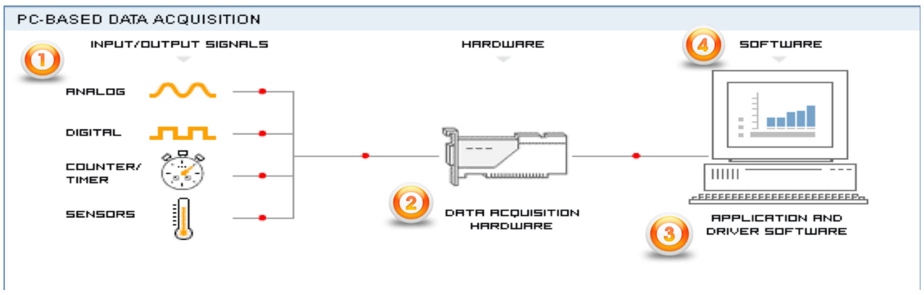

PC-BASED DATA ACQUISITION
1
INPUT/OUTPUT SIGNALS
ANALOG
DIGITAL
COUNTER/ TIMER
SENSORS
HARDWARE
2
DATA ACQUISITION HARDWARE
4
SOFTWARE
3
APPLICATION AND DRIVER SOFTWARE

Figure 24.1: Data Acquisition (DAQ) System

We will use a DAQ device from National Instruments (NI).

Web:
http://www.ni.com/

DAQ hardware: WE will use a NI-USB-600x DAQ device from National Instruments, such as:

- NI-USB-6001
- NI-USB-6008
- NI-USB-6009

They are almost identical and the prices is not so bad either.

USB-6008:
http://www.ni.com/en-no/support/model.usb-6008.html

Figure 24.2 shows the USB-6008 DAQ device from NI.

Streaming Data from NI Data Acquisition (DAQmx) Devices into Python:
https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P8o0SAC

We assume we want to do the following: - We have a USB DAQ system from National Instruments - We want to stream data from my hardware into Python to do data processing - We would like to log data to a file on the hard disk

The best way to do this is to use the NI-DAQmx Python API provided by National Instruments (nidaqmx). The **NI-DAQmx Python API** is hosted on GitHub.

The nidaqmx Python package is a wrapper around the NI-DAQmx C API using the ctypes Python library, and only supports the Windows operating system.

A Python API for interacting with NI-DAQmx (GitHub):
https://github.com/ni/nidaqmx-python

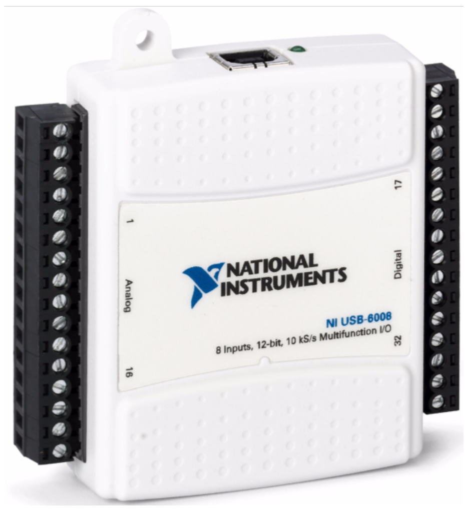


Figure 24.2: USB-6008

For more information about NI DAQ:
ni.com/daq

For more information about Python Resources for NI Hardware and Software:
ni.com/python

Another option is to use the **PyDAQmx** Python package.
This package allows users to use data acquisition hardware from National Instrument with python. It makes an interface between the NIDAQmx driver and python. It currently works only on Windows. The package is not an open source driver from NI acquisition hardware. You first need to install the driver provided by NI.

Web:
https://pypi.org/project/PyDAQmx/

https://pythonhosted.org/PyDAQmx/

24.2.1 NI-DAQmx

NI-DAQmx is the software you use to communicate with and control your NI data acquisition (DAQ) device.

NI-DAQmx supports only the Windows operating system.

You can download NI-DAQmx from this location:

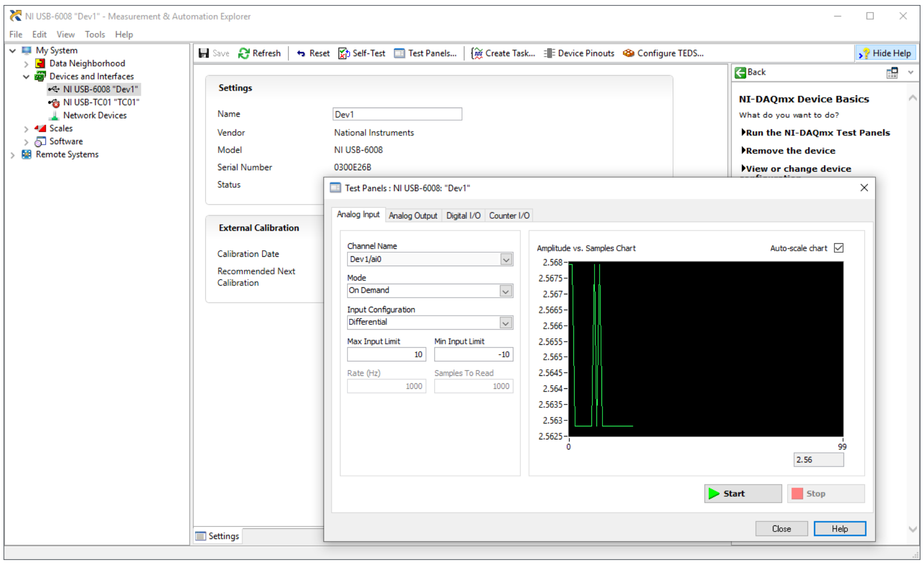
https://www.ni.com/download

24.2.2 Measurement Automation Explorer (MAX)

Measurement Automation Explorer (MAX) is a software you can use it to configure and test the DAQ device before you use it in Python (or other programming languages).

MAX is included with NI-DAQmx software.

Figure 24.3 shows Measurement Automation Explorer (MAX).



Settings

Name: Dev1
Vendor: National Instruments
Model: NI USB-6008
Serial Number: 0300E26B
Status

External Calibration

Calibration Date
Recommended Next Calibration

Channel Name: Dev1/ai0
Mode: On Demand
Input Configuration: Differential
Max Input Limit: 10
Min Input Limit: -10
Rate (Hz): 1000
Samples To Read: 1000
Auto-scale chart ✓
2.56

Figure 24.3: Measurement Automation Explorer(MAX)

With MAX you can make sure your DAQ device works as expected before you start using it in your Python program. You can use the Test Panels to test your analog and digital inputs and outputs channels.

You can also change name of the unit, which you need to use in your Python code.

24.3 NI-DAQmx Python API

In this section we will use the NI-DAQmx Python API provided by National Instruments (nidaqmx). The **NI-DAQmx Python API** is hosted on GitHub.

The nidaqmx Python package is a wrapper around the NI-DAQmx C API using the ctypes Python library, and only supports the Windows operating system.

A Python API for interacting with NI-DAQmx (GitHub):
https://github.com/ni/nidaqmx-python

Other resources:
Control NI DAQ Device with Python and NI DAQmx:
https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019Pf1SAE

Below 4 basic examples will be provided:

- Analog Write using NI DAQ Device
- Analog Read using NI DAQ Device
- Digital Write using NI DAQ Device
- Digital Read using NI DAQ Device

You can easily extend this examples to make them suit your needs. Typically you need to include a while loop where you write and/or read from the DAQ device inside the loop, e.g. read values from one or more sensors that are connected to the DAQ device, you may want to create a control system reading the process value and then later write the calculated control signal (e.g. using a PID controller) back to the DAQ device and the process.

24.3.1 Analog Write

Example 24.3.1. Analog Write using NI DAQ Device

Python code:

```python
import nidaqmx

with nidaqmx.Task() as task:
    task.ao_channels.add_ao_voltage_chan('Dev1/ao0','mychannel'
    ,0,5)

    value = 3
    task.start
    task.write(value)
    task.stop
```

Listing 24.1: Analog Write using NI DAQ Device

Note! The USB-6008 can only output a voltage signal between 0 and 5V.

[End of Example]

24.3.2 Analog Read

Example 24.3.2. Analog Read using NI DAQ Device

Python code:

```python
import nidaqmx

with nidaqmx.Task() as task:
    task.ai_channels.add_ai_voltage_chan("Dev1/ai1")

    value = task.read()
    print(value)
    task.stop
```

Listing 24.2: Analog Read using NI DAQ Device

[End of Example]

Example 24.3.3. Analog Read with RSE

Python code:

```python
import nidaqmx

from nidaqmx.constants import (
    TerminalConfiguration)

with nidaqmx.Task() as task:
    task.ai_channels.add_ai_voltage_chan("Dev1/ai0",
    terminal_config=TerminalConfiguration.RSE)

    value = task.read()
    print(value)
    task.stop
```

Listing 24.3: Analog Read with RSE

[End of Example]

Example 24.3.4. Analog Read with Differential

Python code:

```python
import nidaqmx

from nidaqmx.constants import (
    TerminalConfiguration)

with nidaqmx.Task() as task:
    task.ai_channels.add_ai_voltage_chan("Dev1/ai0",
    terminal_config=TerminalConfiguration.DIFFERENTIAL)

    value = task.read()
    print(value)
    task.stop
```

Listing 24.4: Analog Read with Differential

[End of Example]

24.3.3 Digital Write

Example 24.3.5. Digital Write using NI DAQ Device

Python code:

```python
import nidaqmx

with nidaqmx.Task() as task:
    task.do_channels.add_do_chan("Dev1/port0/line0")

    value = True
    task.start
    task.write(value)
    task.stop
```

Listing 24.5: Digital Write using NI DAQ Device

[End of Example]

24.3.4 Digital Read

Example 24.3.6. Digital Read using NI DAQ Device

Python code:

```python
import nidaqmx

with nidaqmx.Task() as task:
    task.di_channels.add_di_chan("Dev1/port0/line0")

    task.start
    value = task.read()
    print(value)
    task.stop
```

Listing 24.6: Digital Read using NI DAQ Device

[End of Example]

You should use the "nidaqmx.stream_readers" and nidaqmx.stream_writers classes to increase the performance of your application, which accept pre-allocated NumPy arrays.

https://nidaqmx-python.readthedocs.io/en/latest/stream_readers.html#module-nidaqmx.stream_readers

https://nidaqmx-python.readthedocs.io/en/latest/stream_writers.html#module-nidaqmx.stream_writers

24.4 Controlling LEDs

In this section we will see how we can control a LED from Python.

We will need the following equipment:

- PC with Python
- DAQ device
- Breadboard
- LED
- Resistor (e.g., 270ohm)
- Wires for connecting the components and create the circuit

Figure 24.4 shows an overview of LEDs.

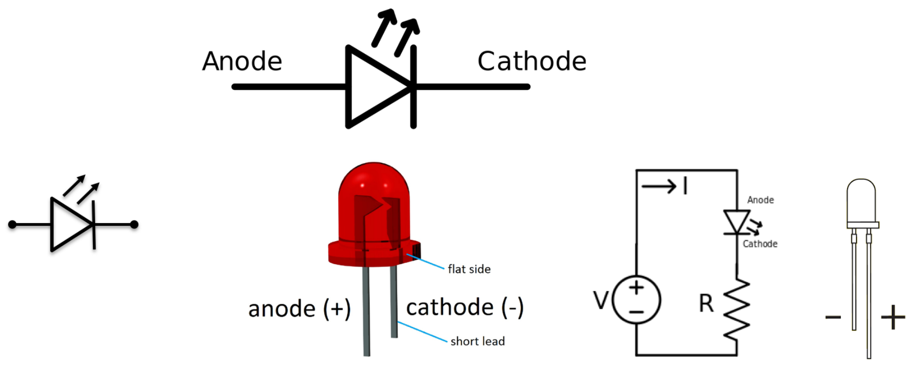


Figure 24.4: LED Overview

A breadboard is used to wire electric components together. Figure 24.5 shows how you should wire a LED using a Breadboard.

Figure 24.6 shows how you wire the LED and connect it to the DAQ device.

Python code for turning on the LED

Example 24.4.1. Controlling a LED from Python

Basic Python code:

```python
import nidaqmx

with nidaqmx.Task() as task:
    task.do_channels.add_do_chan("Dev1/port0/line0")

    value = True
    task.start
    task.write(value)
    task.stop
```

Listing 24.7: Turn on a LED using Python

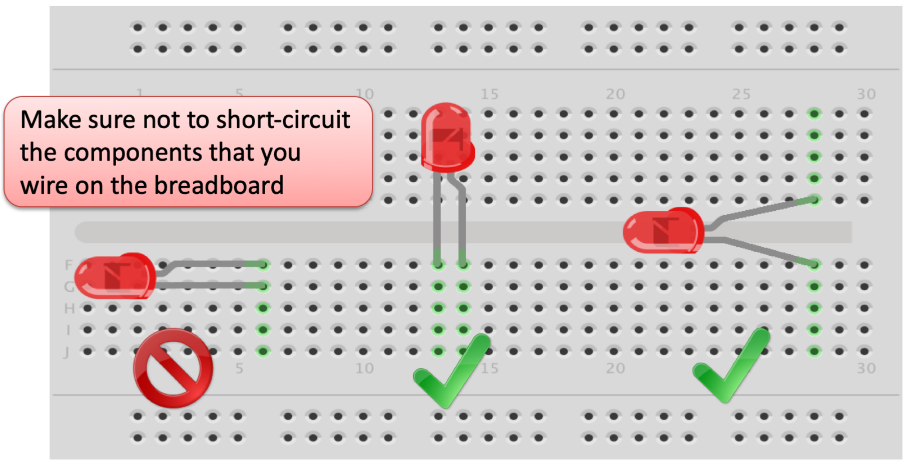

Make sure not to short-circuit the components that you wire on the breadboard

Figure 24.5: How to wire a LED on a Breadboard

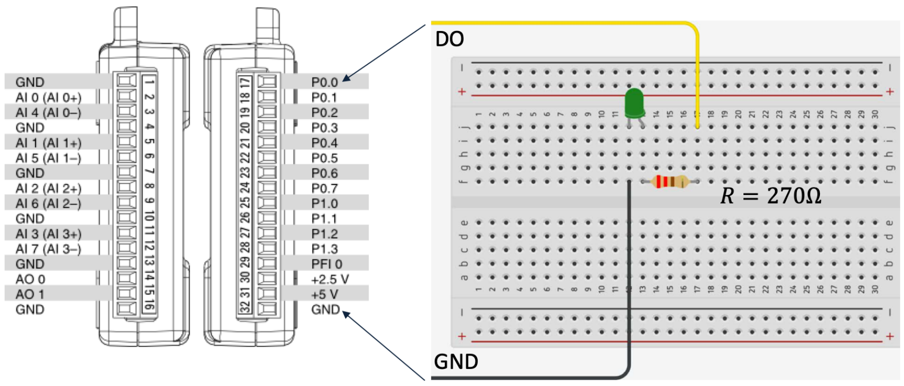


Figure 24.6: Wire the LED and connect to the DAQ device

In this basic example we just turn on the LED.

Below you see an example where we turn the LED on and off inside a loop.

Python code:

```python
import nidaqmx
import time


with nidaqmx.Task() as task:
    task.do_channels.add_do_chan("Dev1/port0/line0")

    value = True
    task.start

    i = 1
    while i < 10:

        task.write(value)
        time.sleep(1)
        value = not value
        task.write(value)
        i = i+1

    task.stop
```
Listing 24.8: Controlling a LED using Python

[End of Example]

24.5 Read Data from Temperature Sensors

In this section we will use Python to read values from a temperature sensor. We will also use Python to plot real-time data from the sensor.

24.5.1 Read Data from TMP36 Temperature Sensor

TMP36 is a small, low-cost temperature sensor and cost about $1 (you can buy it "everywhere").

We will need the following equipment:

- PC with Python
- DAQ device
- Breadboard
- TMP36 Temerature Sensor
- Wires for connecting the components and create the circuit

Figure 24.7 shows the TMP36 sensor.

We connect the TMP36 to LabVIEW using a USB DAQ Device from National Instruments, e.g., USB-6001, USB-6008 or similar. I have used a breadboard for the wiring.
Figure 24.8 show how we can wire the TMP36 together with the USB-6008 DAQ device.

Figure 24.9 shows the TMP3x Datasheet.

We need to convert form Voltage (V) to degrees Celsius.

From the Datasheet (Figure 24.9) we have:

$$(x_1, y_1) = (0.75V, 25^\circ) \tag{24.1}$$

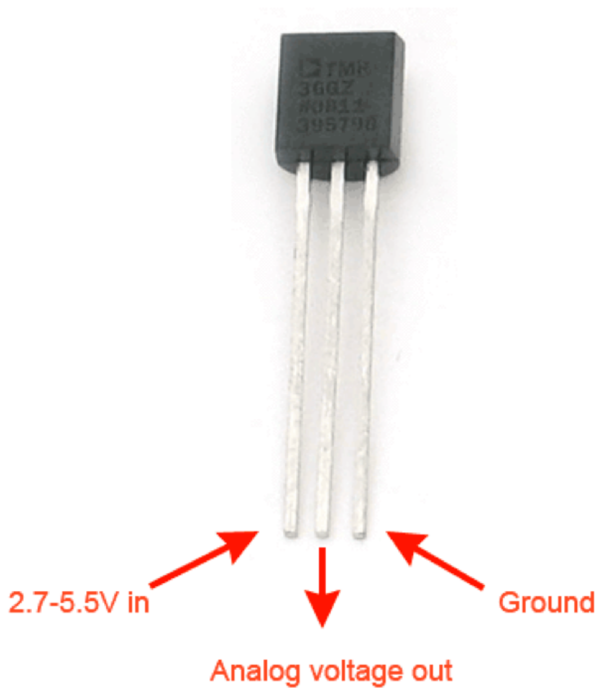


Figure 24.7: TMP36 Temperature Sensor

$$(x_2, y_2) = (1V, 50^\circ) \tag{24.2}$$

From the Datasheet (Figure 24.9) we see that there is a linear relationship between Voltage and degrees Celsius (24.3):

$$y = ax + b \tag{24.3}$$

We can find a and b using the following known formula (24.4):

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \tag{24.4}$$

By putting (24.1) and (24.2) into (24.4) we get:

$$y - 25 = \frac{50 - 25}{1 - 0.75}(x - 0.75) \tag{24.5}$$

Then we get the following formula we can implement in our Python program:

$$y = 100x - 50 \tag{24.6}$$

Example 24.5.1. Read TMP36 Temperature Data

Python code:

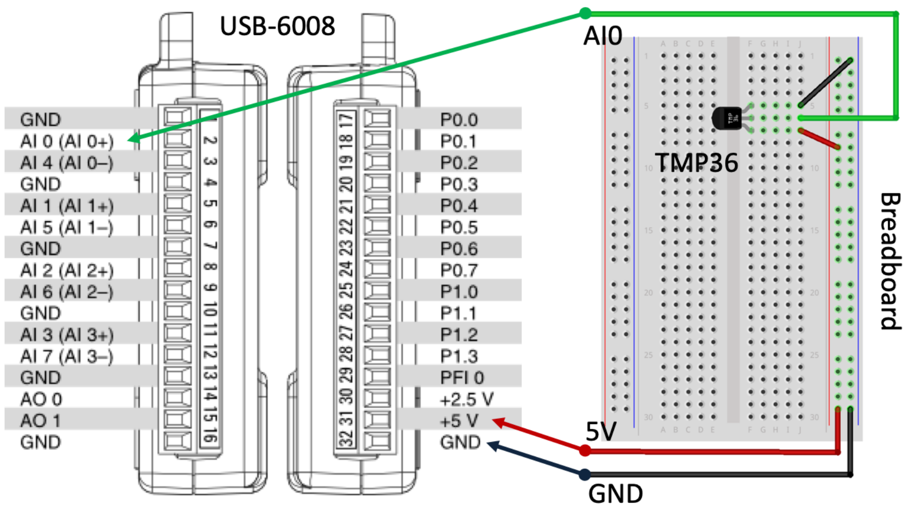


Figure 24.8: TMP36 tmp36 Wiring

```python
import nidaqmx
import time

from nidaqmx.constants import (
    TerminalConfiguration)


with nidaqmx.Task() as task:
    task.ai_channels.add_ai_voltage_chan("Dev1/ai0",
    terminal_config=TerminalConfiguration.RSE)

    i = 0
    while i < 10:

        voltage = task.read()

        degreesC = 100*voltage - 50

        print("Sample:", i)
        print("Voltage Value:", round(voltage,2))
        print("Celsius Value:", round(degreesC,1))
        print("\n")
        time.sleep(1)
        i = i+1

    task.stop
```

Listing 24.9: Read TMP36 Temperature Data

In the example an ordinary while loop in combination with the sleep() function have been used to read one new value each second.

[End of Example]

Example 24.5.2. Real-Time Plotting of Temperature Data

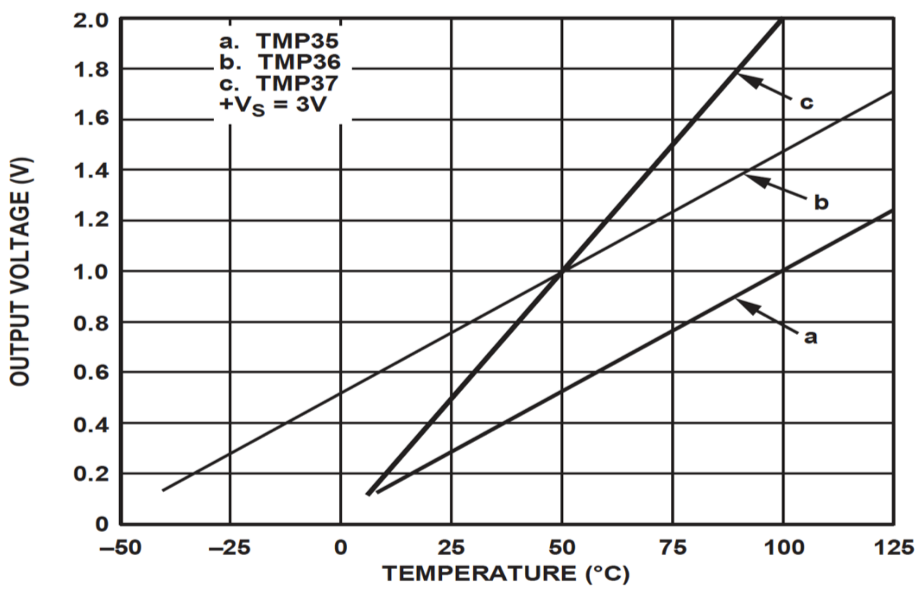


Figure 24.9: TMP3x Datasheet

In this example we will plot the data from the sensor using the Real-time plotting examples shown in Chapter 23.

We want to present the value from the sensor in degrees Celsius:

1. Read Signal from DAQ Device (0-5V)
2. Convert to degrees Celsius using information from the Datasheet
3. Show/Plot Values from the Sensor

The Python code becomes as follows:

```python
import nidaqmx
import time
import datetime as dt
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

from nidaqmx.constants import (
    TerminalConfiguration)


# Create figure for plotting
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
xs = []
ys = []

# Initialize DAQ device
task = nidaqmx.Task()
task.ai_channels.add_ai_voltage_chan("Dev1/ai0", terminal_config=
    TerminalConfiguration.RSE)
```

```
task.start


# This function is called periodically from FuncAnimation
def readdaq(i, xs, ys):

    #Read Value from DAQ device
    voltage = task.read()

    #Convert Voltage to degrees Celsius
    degreesC = 100*voltage - 50
    temp_c = round(degreesC, 1)
    print("Celsius Value:", temp_c)

    # Add x and y to lists
    xs.append(dt.datetime.now().strftime('%H:%M:%S.%f'))
    ys.append(temp_c)

    # Limit x and y lists to 20 items
    xs = xs[-20:]
    ys = ys[-20:]

    # Draw x and y lists
    ax.clear()
    ax.plot(xs, ys)

    # Format plot
    plt.xticks(rotation=45, ha='right')
    plt.subplots_adjust(bottom=0.30)
    plt.title('Temperature Data')
    plt.ylabel('Temperature (deg C)')


# Set up plot to call readdaq() function periodically
ani = animation.FuncAnimation(fig, readdaq, fargs=(xs, ys),
    interval=1000)
plt.show()
task.stop
```

Listing 24.10: Real-Time Plotting of Temperature Data

[End of Example]

24.5.2 Read Data from Thermistor

A Thermistor is an electronic component that changes resistance to temperature, a so-called Resistance Temperature Detectors (RTD). It is often used as a temperature sensor.

Example 24.5.3. Read Thermistor Temperature Data

We will need the following equipment:

- PC with Python
- DAQ device

- Breadboard
- 10kohm Thermistor
- 10kohm Resistor
- Wires for connecting the components and create the circuit

Our Thermistor is a so-called NTC (Negative Temperature Coefficient). In a NTC Thermistor, resistance decreases as the temperature rises.

There is an non-linear relationship between resistance and excitement. To find the temperature we can use the following equation (Steinhart-Hart equation):

$$\frac{1}{T_K} = A + B\ln(R) + C(\ln(R))^3 \tag{24.7}$$

where A, B and C are constants with the following values: $A = 0.001129148, B = 0.000234125, C = 8.76741E - 08$

T_K is the temperature in Kelvin.

We want to solve the equation regarding the Temperature:

$$T_K = \frac{1}{A + B\ln(R) + C(\ln(R))^3} \tag{24.8}$$

The Temperature in degrees Celsius will then be:

$$T_C = T_K - 273.15 \tag{24.9}$$

Wiring...
Figure 24.10 shows how we wire the components together.

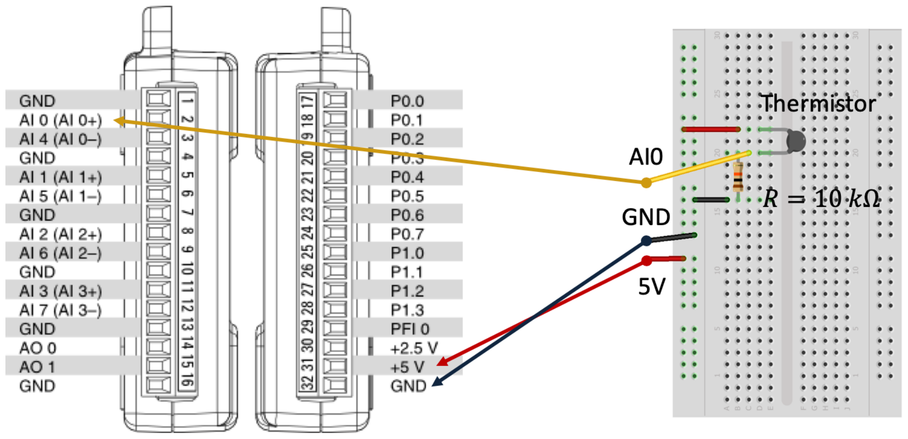


Figure 24.10: Thermistor Wiring

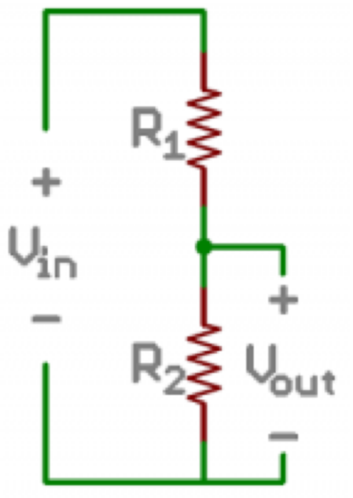


Figure 24.11: Voltage Divider

The wiring is called a "Voltage divider".

Figure 24.11 shows a general Voltage Divider.

A general Voltage Divider has the following equation:

$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2} \tag{24.10}$$

The Voltage Divider for our system becomes as shown in Figure 24.12.

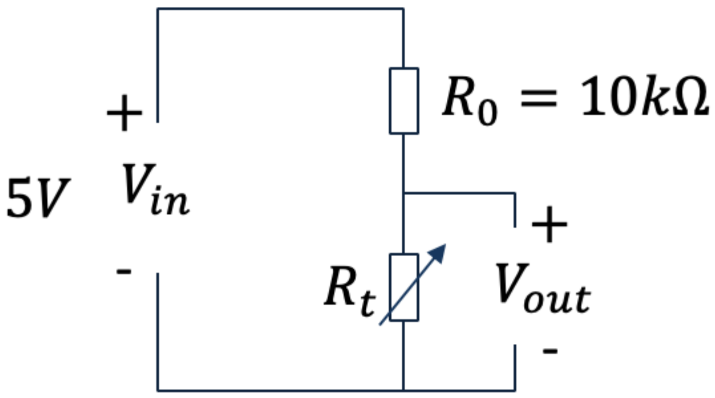


Figure 24.12: Voltage Divider for our System

We then have the following equation:

$$V_{out} = V_{in} \frac{R_t}{R_0 + R_t} \tag{24.11}$$

where R_t is our 10kohm Thermistor and R_0 is an ordinary $10kohm$ Resistor.

V_{in} in our case will be +5V which we get from the USB-6008 DAQ device as shown in the wiring diagram.

V_{out} is the voltage we read using the DAQ device.

Since we need to find the Resistance R_t, which is used in the Steinhart-Hart equation, we reformulate the formula:

$$R_t = \frac{V_{out} R_0}{V_{in} - V_{out}} \tag{24.12}$$

We har no ready to start making the Python program for this example.

The program include the following necessary steps:

1. We wire the circuit on the Breadboard and connect it to the DAQ device
2. We measure V_{out} using the DAQ
3. We calculate R_t using the Voltage Divider equation
   $R_t = \frac{V_{out} R_0}{V_{in} - V_{out}}$
4. We use Steinhart-Hart equation for finding the Temperature
   $T_K = \frac{1}{A + B\ln(R_t) + C(\ln(R_t))^3}$
5. Finally we convert to degrees Celsius
   $T_C = T_K - 273.15$

The Python code then becomes:

```python
import nidaqmx
import numpy as np
import time

from nidaqmx.constants import (
    TerminalConfiguration)


Vin = 5
Ro = 10000 # %10k Resistor


with nidaqmx.Task() as task:
    task.ai_channels.add_ai_voltage_chan("Dev1/ai0",
    terminal_config=TerminalConfiguration.RSE)

    i = 0
    while i < 10:

        Vout = task.read()

        Rt = (Vout*Ro)/(Vin-Vout) # Voltage Divider Equation
        # Rt=10000; Used for Testing. Setting Rt=10k should give
    TempC=25

```

```
        # Steinhart constants
        A = 0.001129148
        B = 0.000234125
        C = 0.0000000876741

        # Steinhart-Hart Equation
        TempK = 1 / (A + (B * np.log(Rt)) + (C * pow(np.log(Rt),3))
    )

        # Convert from Kelvin to Celsius
        TempC = TempK - 273.15

        print("Sample:", i)
        print("Voltage Value:", round(Vout,2))
        print("Celsius Value:", round(TempC,1))
        print("\n")
        time.sleep(1)
        i = i+1

    task.stop
```

Listing 24.11: Read Thermistor Temperature Data

[End of Example]

Example 24.5.4. Real-Time Plotting of Thermistor Temperature Data

Python code:

```
See previous examples
```

Listing 24.12: Real-Time Plotting of Thermistor Temperature Data

[End of Example]

24.5.3 Read Data NI TC-01 Thermocouple Device

In this chapter several examples have been shown using a DAQ device combined with different sensors and components.

Here some examples will be shown using a preset temperature sensor from National Instruments called NI USB-TC01. This is a USB based temperature without need for any kind of wiring, you just plug it in and make your Python program. Since the NI USB-TC01 is compatible with NI-DAQmx, you can program it in the same way as other DAQ devices from NI.

Figure 24.13 shows the TC-01 Thermocouple Device from NI.

Example 24.5.5. Real-Time Plotting of Thermistor Temperature Data

Python code:

Figure 24.13: TC-01 Thermocouple Device

```python
import nidaqmx

task = nidaqmx.Task()

task.ai_channels.add_ai_thrmcpl_chan("TC01/ai0")

task.start()

value = task.read()
print(round(value,1))

task.stop()
task.close()
```

Listing 24.13: TC-01 Thermocouple Python Example

This is just a basic example, which you can easily extend using a while loop or using some kind of plotting, etc..

[End of Example]

24.6 Data Logging

Python has several functions for creating, reading, updating, and deleting files.

Part VI

Python Database Development

Chapter 25

Database Applications with Python

Here we will learn how we can use Python for communication with a Database system such as SQL Server or MySQL. We will learn how we connect to a database, how we can insert data into the database and retrieve data from the database.

A Database is a structured way to store lots of information. The information is stored in different tables. Some of the most popular Database Systems today are:

- SQL Server
- MySQL
- MariaDB
- MongoDB
- etc.

ER Diagram (Entity-Relationship Diagram) is used for design and modeling of databases. It specifies tables and relationship between them (Primary Keys and Foreign Keys)

Figure 25.1 shows an example of an ER diagram consisting of two database tables.

Here you can learn more about Database Systems, download examples and get additional resources, see videos, etc.:
https://www.halvorsen.blog/documents/technology/database/

25.1 Structured Query Language (SQL)

Structured Query Language (SQL) is a database language supported by most of the existing database systems today. You use SQL to interact with the database

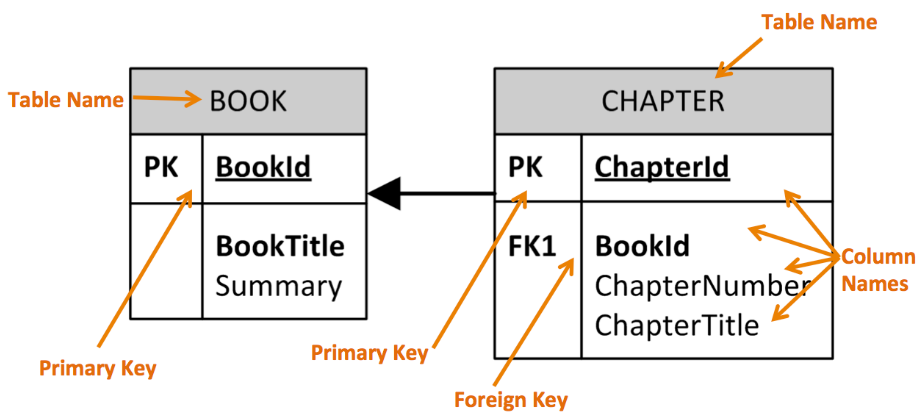


Figure 25.1: ER Diagram Example

system, like insert data into the database and retrieve data from the database.

Here you can learn more about SQL, download examples and get additional resources, see videos, etc.:
https://www.halvorsen.blog/documents/technology/database/

25.2 SQL Server

Here we will see how we can communicate with a SQL Server database from Python.

25.3 MySQL

Here we will see how we can communicate with a MySQL database from Python.

25.4 MongoDB

Here we will see how we can communicate with a MongoDB database from Python.

MongoDB is a so-called NoSQL database. One of the most popular NoSQL database systems is MongoDB.

You can download a free MongoDB database from:
https://www.mongodb.com

Chapter 26

Structured Query Language (SQL)

...

Chapter 27

SQL Server with Python

27.1 Introduction to SQL Server

Here we will see how we can communicate with a SQL Server from Microsoft using the Python programming language.

27.2 SQL Server drivers for Python

There are several python SQL drivers available. However, Microsoft places its testing efforts and its confidence in pyodbc driver. Another driver is pymssql.

pyodbc is an open source Python module that can be used to accessing ODBC databases.

27.3 pyodbc

pyODBC uses the Microsoft ODBC driver for SQL Server.

27.3.1 Installation of pyodbc

Install pyodbc using pip - Python package manager.

pip install pyodbc

27.3.2 ODBC Drivers

Microsoft have written and distributed multiple ODBC drivers for SQL Server: You can use the "ODBC Driver 17 for SQL Server" driver.

This driver supports SQL Server 2008 through 2019.

Note that the "SQL Server Native Client ..." and earlier drivers are deprecated and should not be used for new development.
The connection string can be written like this:

DRIVER=ODBC Driver 17 for SQL Server;SERVER=test;DATABASE=test;UID=user;PWD=password

27.4 SQL Server Python Examples

The cursor.execute function can be used to retrieve a result set from a query against the SQL Database. This function accepts a query and returns a result set.

Example 27.4.1. Basic SQL Server Example

The cursor.execute function is used to retrieve a result set from a query against the SQL Server Database. This function accepts a query and returns a result set, which can be iterated over with the use of cursor.fetchone().

```
import pyodbc

server = "NUCHPH\\SQLEXPRESS"
database = "BOOKS"
username = "sa"
password = "xxx"
conn = pyodbc.connect("DRIVER={ODBC Driver 17 for SQL Server};
    SERVER=" + server + ";DATABASE=" + database + ";UID=" +
    username + ";PWD=" + password )

cursor = conn.cursor()

cursor.execute("SELECT @@version;")
row = cursor.fetchone()
while row:
    print(row[0])
    row = cursor.fetchone()
```

Listing 27.1: Basic SQL Server Example

Example 27.4.2. Getting Data from a Table in SQL Server

Below you find a basic example where data are retrieved from the SQL Server.

```
import pyodbc

server = "NUCHPH\\SQLEXPRESS"
database = "BOOKS"
username = "sa"
password = "xxx"
conn = pyodbc.connect("DRIVER={ODBC Driver 17 for SQL Server};
    SERVER=" + server + ";DATABASE=" + database + ";UID=" +
    username + ";PWD=" + password )

```

```
cursor = conn.cursor()


for row in cursor.execute("select BookId, Title, Isbn from BOOK"):
    print(row.BookId, row.Title, row.Isbn)
```
Listing 27.2: Getting Data from SQL Server

[End of Example]

27.5 Stored Procedures

27.6 Resources

https://github.com/mkleehammer/pyodbc/wiki/Connecting-to-SQL-Server-from-Windows

27.7 pymssql

Resources:

https://pypi.org/project/pymssql/
http://www.pymssql.org/

27.8 Resources

https://docs.microsoft.com/en-us/sql/connect/python/python-driver-for-sql-server

Chapter 28

MySQL with Python

...

Chapter 29

MongoDB with Python

Here we will learn how we can use Python for communication with MongDB. We will learn how we connect to a database, how we can insert data into the database and retrieve data from the database.

29.1 Introduction to MongoDB

Here we will see how we can communicate with a MongoDB database from Python.

MongoDB is a so-called NoSQL database. One of the most popular NoSQL database systems is MongoDB.

You can download a free MongoDB database from:
https://www.mongodb.com

29.2 MongoDB with Python

Here we will see how we can communicate with a MongoDB database from Python.

Python needs a MongoDB driver to access the MongoDB database. Many different drivers do exists, so you just need to choose one.

29.2.1 PyMongo

The PyMongo distribution contains tools for interacting with MongoDB database from Python

https://pypi.org/project/pymongo/

Installation is done using PIP:

```
python -m pip install pymongo
```

29.3 Additional Resources

Tutorials:
https://www.w3schools.com/python/python$_mongodb_getstarted.asp$

Part VII

Python Application Development

Chapter 30

Development of Applications with Python

Python is popular within computation, but can be used within many other applications and can be integrated and used in combination with other programming languages, e.g., for development of Web Applications, etc.

Python is probably best suited for "ad-hoc" numerical calculations, analysis, simulations, etc., but can be used for many other purposes, even if other programming languages are better suited.

Python can be used for creating Web pages (in combination with HTML, CSS, JavaScript). Python is then typically used on the server-side, while HTML, CSS and JavaScript are used on the client-side.

An example is Django, which is a server-side Python framework used for creating dynamic web pages.

Python can also be used for programming and creating Raspberry Pi applications.

In general, Python is a multipurpose programming language that can be used in many situations. But there is not one programming language which is best in all kind of situations, so it is important that you know about and have skills in different languages.

My list of recommendations:

- Visual Studio and C
- LabVIEW - a graphical programming language well suited for hardware integration, taking measurements and data logging
- MATLAB - Numerical calculations and Scientific computing
- Python - Numerical calculations, and Scientific computing, etc.

- Web Programming, such as HTML, CSS, JavaScript and a Server-side framework/programming language like PHP, ASP.NET
- Databases (such as SQL Server and MySQL) and using the Structured Query Language (SQL)
- App Development for the 2 main platforms iOS (XCode using the Swift Programming Language) and Android (Android Studio using the Java Programming Language or Kotlin Programming language)

If you have skills in most of the tools, programming languages and frameworks mention above, you are well suited for working as a full-time programmer or software engineer.
A good resource or starting point for creating Applications for Python is:

Applications for Python [18]:
https://www.python.org/about/apps/

30.1 Mathematics, Science and Engineering

Python is probably best suited for "ad-hoc" numerical calculations, analysis, simulations, etc. These kinds of applications has also been the main focus in this textbook. Other kinds of applications will briefly be covered in this chapter. More details will be covered in later chapters and in other textbooks in my Python textbooks series, which you can read more of in the Preface of this textbook.

The SciPy is a collection of packages for mathematics, science, and engineering, which has been thoroughly reviewed earlier in this textbook.

30.2 Desktop GUI Applications

Python don't come with builtin tools for creating traditional desktop GUI applications, so you need to use an external GUI packages for this purpose.

In my opinion, tools like Visual Studio where you can create professional GUI applications using either the C or VB.NET languages in one integrated packages is a better choice.

Another good alternative is LabVIEW, which has powerful GUI features in combination with extensive hardware integration.

Other tools (Integrated Programming Environments, IDE) and programming languages for GUI applications os Xcode on macOS, which can be used for creating macOS desktop applications and apps for iPhone and iPad.

For Android development you have Android Studio. Here you can use programming languages like Java and Kotlin.

Since this is a Python textbook, lets go back to the options we have if we want to create desktop GUI applications with Python.

Python has different desktop GUI frameworks like:

- PyQt
- Tkinter
- WxPython
- PyGUI
- PySide2
- Kivy

These are just some of the options we have, for a comprehensive overview of GUI frameworks for Python see the following:

https://docs.python.org/3/library/othergui.htmlother-gui-packages
https://wiki.python.org/moin/GuiProgramming

PyQt and wxPython, all have a modern look and feel and more widgets than Tkinter.

This is also a bit of a problem when it comes to desktop GUI development with Python. You have so many choices, and sometimes its better better with one good option than many half good options.

30.2.1 PyQt

PyQt brings together the Qt C++ cross-platform application framework and the cross-platform interpreted language Python. Qt is a cross-platform GUI toolkit.

Qt also includes Qt Designer, a graphical user interface designer. PyQt is able to generate Python code from Qt Designer. It is also possible to add new GUI controls written in Python to Qt Designer.

For more information about PyQt:

https://riverbankcomputing.com/software/pyqt/intro

For more information about Qt:

https://www.qt.io

PyQt Tutorials:

https://likegeeks.com/pyqt5-tutorial/

https://build-system.fman.io/pyqt5-tutorial

30.2.2 PySide2

PySide2 is the official Python module from the Qt for Python project, which provides access to the complete Qt framework.

The originally PySide framework was originally released by Nokia, then owner of Qt. After Nokia sold Qt in 2011, PySide was no longer maintained. Then PySide2 was established and maintained by a community. Finally, in 2016, the Qt company committed to officially support the PySide2 project.
So basically, PySide2 is very similar to PyQt.

The downside with PySide2 (August 2019) is that it is still in "beta" (Technical Preview).

For more information about PySide2:

https://pypi.org/project/PySide2/

https://wiki.qt.io/Qt$_{f}or_{P}ython$

30.2.3 Tkinter

Another popular GUI framework is Tkinter.

For more information about Tkinter:

https://docs.python.org/2/library/tkinter.html

30.2.4 WxPython

WxPython is a cross-platform GUI toolkit for the Python language.

For more information about WxPython:

https://www.wxpython.org

https://wiki.wxpython.org/Getting

30.3 Web Applications

Python can be used for creating Web pages (in combination with HTML, CSS, JavaScript). Python is then typically used on the server-side, while HTML, CSS and JavaScript are used on the client-side.

An example is Django, which is a server-side Python framework used for creating dynamic web pages.

Read more about Django here:

https://www.djangoproject.com
Other popular web application frameworks and programming languages are ASP.NET and PHP.

You may read more about web programming in general here:

https://www.halvorsen.blog/documents/programming/web/

30.4 Database Applications

Python can be used for communication with a database system such as SQL Server or MySQL. Python has different packages for communication with different database systems, both SQL databases (e.g., SQL Server, MySQL, MariaDB, etc.) and so-called NoSQL databases (e.g., MongoDB).

Here you can learn more about Database Systems and SQL, download examples and get additional resources, see videos, etc.:
https://www.halvorsen.blog/documents/technology/database/

30.4.1 SQL Server

SQL Server is a Database System from Microsoft. SQL Server comes in different editions, for basic, personal use SQL Server Express is recommended because it is simple to use and it is free.

Read more about SQL Server here:

https://www.halvorsen.blog/documents/technology/database/sql$_server.php$

30.4.2 MySQL

MySQL is an open-source and widely used Relational Database Management System (RDBMS).

MySQL comes in different editions, both paid (Enterprise) and free versions (Community).

In addition to the Database itself, the MySQL Workbench is nice to have. MySQL Workbench visual database design tool, used for Datbase Modelling, etc. Another handy tool is phpMyAdmin. phpMyAdmin is a free software tool written in PHP, intended to handle the administration of MySQL.

Read more about MySQL here:

https://www.halvorsen.blog/documents/technology/database/mysql.php

30.4.3 MariaDB

MariaDB is a spinoff from the more famous MySQL Database System.

The MariaDB database server is published as free and open source software. MariaDB has compatibility with MySQL in most situations. MariaDB is said to have slightly better performance than MySQL.

Read more about MariaDB here:

https://www.halvorsen.blog/documents/technology/database/mariadb.php

30.4.4 MongoDB

MongoDB is a general purpose, document-based, distributed database.

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program.

You can download MongoDB from:
https://www.mongodb.com

Chapter 31

Python Integration with Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. The deafult (main) programming language in Visual studio is C, but many other programming languages are supported.

Visual studio is available for Windows and macOS.

Visual Studio (from Visual Studio 2017), has integrated support for Python, it is called "Python Support in Visual Studio".

Web:
https://visualstudio.microsoft.com

Wikipedia:
https://en.wikipedia.org/wiki/Microsoft$_Visual_Studio$

Here you can learn more about Visual studio and C, download examples and get additional resources, see videos, etc.:
https://www.halvorsen.blog/documents/programming/csharp/

For an introduction to the Python integration in Visual Studio, see Chapter 19.

Chapter 32

Python Integration with LabVIEW

32.1 What is LabVIEW?

LabVIEW is a graphical programming language well suited for hardware integration, taking measurements and data logging.
Go to my web site in order to learn more about LabVIEW:
https://www.halvorsen.blog

https://www.halvorsen.blog/documents/programming/labview/

Her you find information about LabVIEW, you find lots of resources like training material, videos, code examples, etc.

32.2 Using Python in LabVIEW

LabVIEW is a fully functional programming language which you can use to create many different kinds of applications. In addition it cal also integrate with many other programming languages like MATLAB and Python.

Web:
http://zone.ni.com/reference/en-XX/help/371361R-01/glang/python_pal/

Use the Python functions to call Python code from LabVIEW. See Figure 32.1

Note! LabVIEW supports calling Python version 2.7 and 3.6. Although unsupported versions might work with the LabVIEW Python functions, NI recommends using supported versions of Python only.

Ensure that the bitness of Python corresponds to the bitness of LabVIEW installed on the machine. This means if you have LabVIEW 32 bit, you should use Python 32 bit and if you have LabVIEW 64 bit, you should use Python 64 bit.

To run the Python code, LabVIEW requires the Python shared libraries (DLLs) in the system path.

For Windows: If you install Python 3.6, add the directory containing python36.dll to the system path. If you install Python 2.7, add the directory containing python27.dll to the system path.

For detailed instructions regarding Installing Python for Calling Python Code: http://www.ni.com/product-documentation/54295/en/

LabVIEW functions for dealing with Python: Open Python Session Python Node Close Python Session

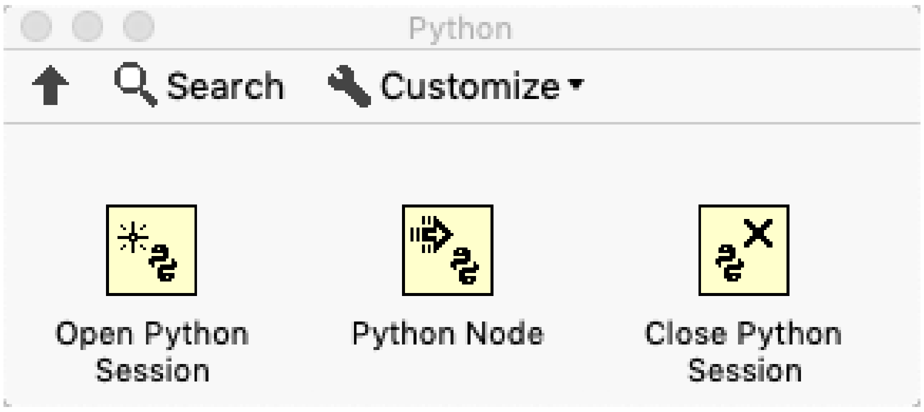


Figure 32.1: Python Integration in LabVIEW

The "Python Node" calls a Python function directly.

Here I will present some examples how we can integrate an existing Python script or Python function.

Example 32.2.1. Python Integration in LabVIEW

We want to use Python to covert between Ceslius and Fahrenheit (and vice versa).

The formula for converting from Celsius to Fahrenheit is:

$$T_f = (T_c \times 9/5) + 32 \qquad (32.1)$$

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \qquad (32.2)$$

First, we create a Python module with the following functions (**fahrenheit.py**):

```python
def c2f(Tc):

    Tf = (Tc * 9/5) + 32
    return Tf


def f2c(Tf):

    Tc = (Tf - 32)*(5/9)
    return Tc
```

Listing 32.1: Python Functions

Then, we create a Python script for testing the functions (**test**$_f ahrenheit.py$) :

```
from fahrenheit import c2f, f2c

Tc = 0

Tf = c2f(Tc)

print("Fahrenheit: " + str(Tf))


Tf = 32

Tc = f2c(Tf)

print("Celsius: " + str(Tc))
```

Listing 32.2: Testing the Functions

The results becomes:

```
Fahrenheit: 32.0
Celsius: 0.0
```

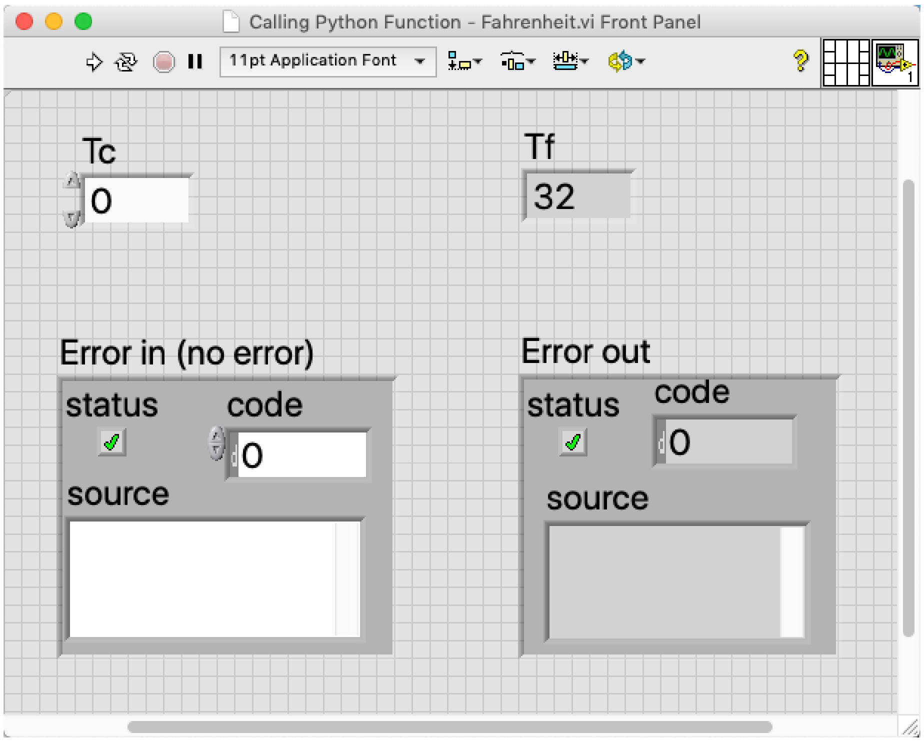
Lets make the LabVIEW program that call these Python functions:

In Figure 32.2 we see the Front Panel.

In Figure 32.3 we see the Block Diagram.

In Figure 32.4 we see LabVIEW Code for calling both Python functions (c2f and f2c) from LabVIEW.

[End of Example]



Tc
0

Tf
32

Error in (no error)
status ✓
code 0
source

Error out
status ✓
code 0
source

Figure 32.2: Python Integration in LabVIEW

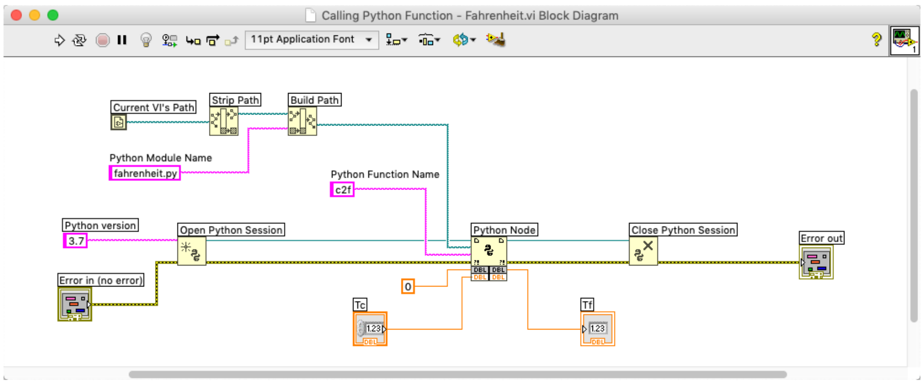


Figure 32.3: Python Integration in LabVIEW

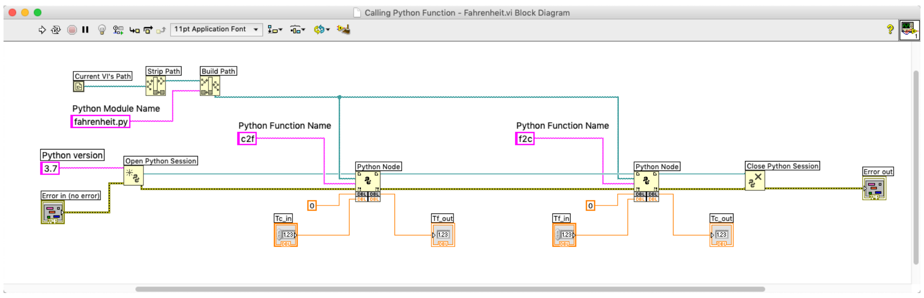


Figure 32.4: Python Integration in LabVIEW

Chapter 33

Raspberry Pi and Python

33.1 What is Raspberry Pi?

The Raspberry Pi is a credit-card-sized computer that plugs into your TV and a keyboard. It is a capable little computer which can be used in electronics projects, and for many of the things that your desktop PC does.

Raspberry Pi is very popular in IoT projects and applications.

For more information and resources regarding Raspberry Pi:

https://www.halvorsen.blog/documents/technology/iot/raspberry_pi.php

Other Resources:

https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/programming-in-python

First, before you start programming Python on a Raspberry Pi device, you need to install an operating system like Raspbian. Raspbian is a Linux distribution tailor made for Raspberry Pi.

Raspbian comes also pre-installed Python.

For more information about Raspbian:

https://www.raspberrypi.org/downloads/raspbian/

Chapter 34

Web Development with Python

34.1 Introduction to Web Development

The basic building blocks for creating Web pages are the following:

- HTML
- CSS
- JavaScript

Here you find more information, resources, videos and examples regarding Web Development:
https://www.halvorsen.blog/documents/programming/web/

34.1.1 HTML

Here you find more information, resources, videos and examples regarding HTML:
https://www.halvorsen.blog/documents/programming/web/html.php

34.1.2 CSS

34.1.3 JavaScript

34.2 Introduction to Web Frameworks

The basic building blocks for creating Web pages are the following:

- HTML
- CSS
- JavaScript

In addition you typically need a Web Framework for creating dynamic web pages that communicate with a Database, etc.

We divide Web Development into 2 parts: Server-side development and client-side development.
On the server-side we have different Web Development Frameworks like:

- PHP
- ASP.NET
- ... Many others, including different Python Web Development Frameworks

34.2.1 PHP

Here you find more information, resources, videos and examples regarding PHP:
https://www.halvorsen.blog/documents/programming/web/php.php

34.2.2 ASP.NET

Here you find more information, resources, videos and examples regarding ASP.NET:
https://www.halvorsen.blog/documents/programming/web/asp$_{n}et.php$

There are different Python Frameworks you can use for creating Web Applications with Python.
Django Bottle Flask

34.3 Django - Python-based Web Framework

Django - Python-based Web Framework.

Chapter 35

Create GUI Applications

Here are some programming environments and programming languages for creating GUI applications presented.

35.1 LabVIEW

Here you find more information, resources, videos and examples regarding LabVIEW:
https://www.halvorsen.blog/documents/programming/labview/

35.2 Visual Studio and C#

Here you find more information, resources, videos and examples regarding Visual Studio and C#:
https://www.halvorsen.blog/documents/programming/csharp/

35.3 Web Development

Here you find more information, resources, videos and examples regarding Web Development:
https://www.halvorsen.blog/documents/programming/web/

Here you find more information, resources, videos and examples regarding HTML:
https://www.halvorsen.blog/documents/programming/web/html.php

Here you find more information, resources, videos and examples regarding PHP:
https://www.halvorsen.blog/documents/programming/web/php.php

Here you find more information, resources, videos and examples regarding ASP.NET:
https://www.halvorsen.blog/documents/programming/web/asp$_{n}et.php$

Chapter 36

Machine Learning with Python

36.1 Introduction to Machine Learning

Here you can learn more about Machine Learning, download examples and get additional resources, see videos, etc.:
https://www.halvorsen.blog/documents/technology/machine$_{l}earning/$

Part VIII

Python PyQt GUI Development

Chapter 37

Getting Started with PyQt

37.1 Introduction

PyQt brings together the Qt C++ cross-platform application framework and the cross-platform interpreted language Python.

Qt is a cross-platform GUI toolkit.

Qt also includes Qt Designer, a graphical user interface designer. PyQt is able to generate Python code from Qt Designer. It is also possible to add new GUI controls written in Python to Qt Designer.

37.2 Introduction to Qt

Qt (pronounced "cute") is a free and open-source widget toolkit for creating graphical user interfaces as well as cross-platform applications that run on various software and hardware platforms such as Linux, Windows, macOS, Android or embedded systems with little or no change in the underlying codebase while still being a native application with native capabilities and speed.

Qt is currently being developed by The Qt Company. Qt was originally created by the Norwegian company Trolltech.

If you want to use Qt inside Python you will need a Python library that let you interface with Qt's C++ API. Many such bindings/libraries do exist. The most used Qt Python library is **PyQt**. We will use PyQt in this chapter.

Another Qt Python library is PySide2. PySide2 is the official Python module from the "Qt for Python" project, which provides access to the complete Qt framework.

The originally PySide framework was originally released by Nokia, then owner of Qt. After Nokia sold Qt in 2011, PySide was no longer maintained. Then

PySide2 was established and maintained by a community. Finally, in 2016, the Qt company committed to officially support the PySide2 project.
So basically, PySide2 is very similar to PyQt.

The downside with PySide2 (August 2019) is that it is still in ”beta” (Technical Preview).

For more information about Qt, see the resources below.

Wikipedia:
https://en.wikipedia.org/wiki/Qt$_(software)$

The Qt Company web site:
https://www.qt.io

37.3 Introduction to PyQt

PyQt is a set of Python bindings for The Qt Company’s Qt application framework and runs on all platforms supported by Qt including Windows, OS X, Linux, iOS and Android. PyQt5 supports Qt v5.

PyQt is developed and maintened by Riverbank Computing.
For more information about PyQt, see the resources below.

Riverbank Computing web site:
https://riverbankcomputing.com/software/pyqt/

Here are some PyQt Tutorials:

https://likegeeks.com/pyqt5-tutorial/

https://build-system.fman.io/pyqt5-tutorial

https://data-flair.training/blogs/python-pyqt5-tutorial/

https://www.guru99.com/pyqt-tutorial.html

https://pythonspot.com/gui/

37.3.1 PyQtChart

PyQt don’t include any types of charts. In order to use charts or plotting, you can use PyQtChart.

PyQtChart is a set of Python bindings for The Qt Company’s Qt Charts library. The bindings sit on top of PyQt5 and are implemented as a single module.

https://www.riverbankcomputing.com/software/pyqtchart/

37.4 Installing PyQt

37.4.1 Installation of PyQt

Use PIP in order to install PyQt5:

```
pip3 install PyQt5
```

PIP is a tool for installing Python packages.

The Python Package Index (PyPI) is a repository of software for the Python programming language. Package authors use PyPI to distribute their software.

For more information about PIP and installing Python Packages:

https://packaging.python.org/tutorials/installing-packages/
For more information about PyPI:

https://pypi.org

For more information about PyQt5:

https://pypi.org/project/PyQt5/

37.4.2 Installation of Qt Designer

Qt Designer is a tool for quickly building graphical user interfaces with widgets from the Qt GUI framework. It gives you a simple drag-and-drop interface for laying out components such as buttons, text fields, combo boxes and more.

This gives the the plot shown in Figure 37.1.

You can download and install the Qt Designer from this web site:

https://build-system.fman.io/qt-designer-download

37.4.3 Installation of PyQtChart

Use PIP in order to install PyQtChart:

```
pip3 install PyQtChart
```

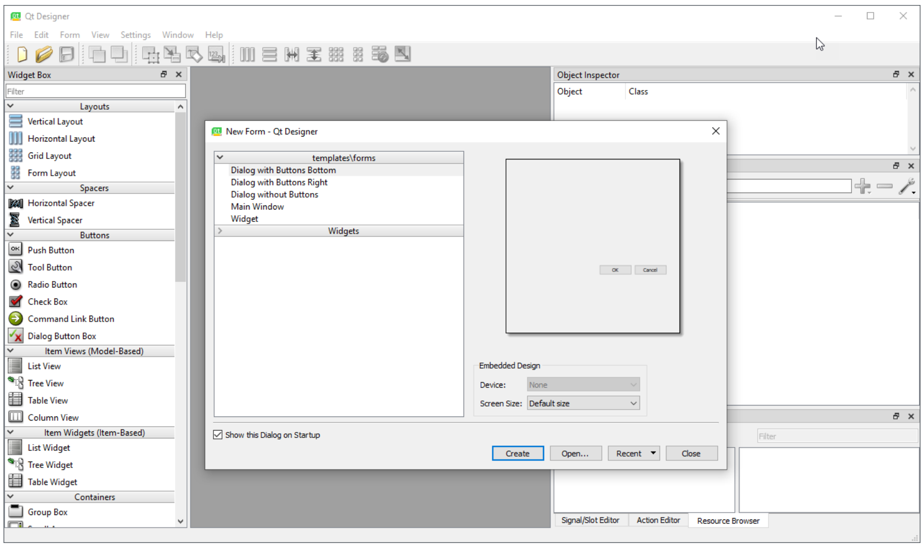


Figure 37.1: Qt Designer

37.5 PyQt Basics

Below we see a basic PyQt application created in Python.

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget
if __name__ == "__main__":
    application = QApplication(sys.argv)
    window = QWidget()
    window.resize(300,200)
    window.setWindowTitle("Hello World")
    window.show()
    sys.exit(application.exec_())
```

Listing 37.1: Basic PyQt Application

If you are using macOS, the Python code above gives the basic application shown in Figure 37.2.

If you are using Windows, the Python code above gives the basic application shown in Figure 37.3.

For the first, with the same Python code we can create cross-platform application.

Lets go through the code in detail:

```
from PyQt5.QtWidgets import QApplication, QWidget
```

This statement imports all the necessary modules you need to create your GUI.

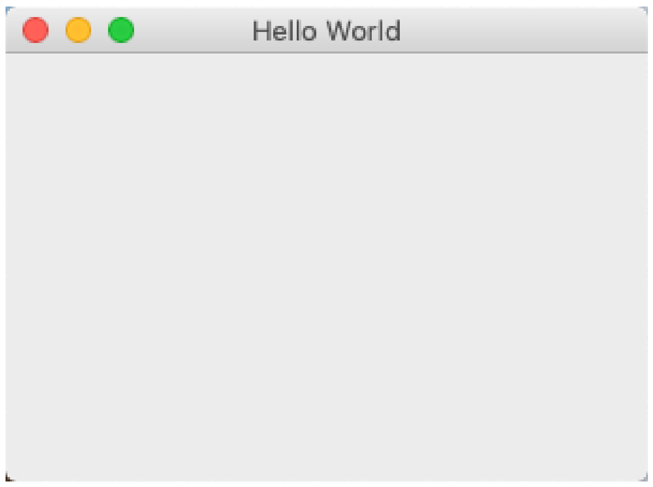


Figure 37.2: Basic PyQt Application on macOS

The QtWidgets module contains all the major widgets that you will be using in this tutorial.

```
application = QApplication(sys.argv)
```

The first thing you need to create is an object of the QApplication class. All PyQt GUI application must create an instance of QApplication.

sys.argv is the list of command-line parameters that you can pass to the application when launching it through the shell or while automating the interface.

If you don't need to pass any arguments to QApplications, you can use [].

```
application = QApplication([])
```

Then we need to create an object of the QWidget class. QWidget is the foundation for all UI objects in Qt. Typically, everything you see in an an PyQt application is a widget, examples: Label, ComboBox, CheckBox, RadioButton, PushButton, etc.

```
window = QWidget()
```

```
window.resize(300,200)
```

The resize method of the QWidget class allows you to set the dimensions of any widget. In this case, you have resized the window to 300px by 200px.

Note that widgets could be nested together, the outermost widget (i.e., the widget with no parent) is called a Window.

Use the **setWindowTitle()** method to set a title for your window.

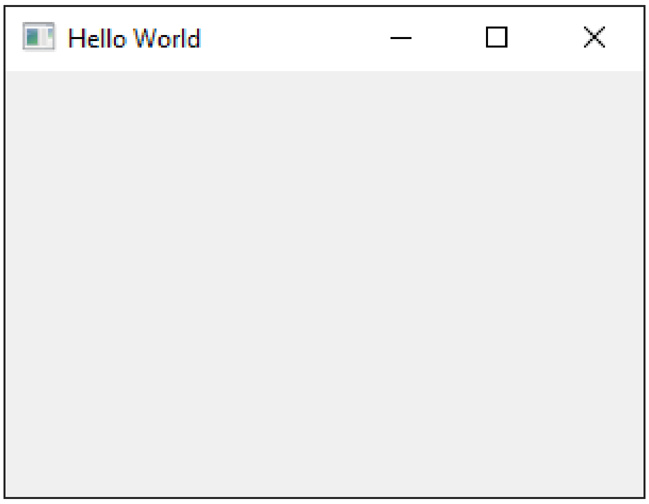


Figure 37.3: Basic PyQt Application on Windows

```
window.setWindowTitle("Hello World")
```

When all the settings are done, we need to show the window on the screen by using the **show()** method:

```
window.show()
```

The final piece of code is:

```
sys.exit(application.exec_())
```

The app.exec_() method starts the event loop inside Qt/C++. PyQt is mainly written in C++ and uses the event loop mechanism to implement parallel execution. app.exec_() passes the control over to Qt which will exit the application only when the user closes it from the GUI.

37.6 PyQt Widgets

QWidget is the foundation for all UI objects in Qt. Typically, everything you see in an an PyQt application is a widget, examples: Label, ComboBox, CheckBox, RadioButton, PushButton, etc.

Lets make some code examples using some of these widgets.

Example 37.6.1. Label (QLabel)

A basic Label example:

```python
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel
if __name__ == "__main__":
    application = QApplication(sys.argv)
    window = QWidget()
    window.resize(300,200)
    window.setWindowTitle("Hello World")

    label = QLabel(window)
    label.setText("This is a Label")
    label.move(100,50)

    window.show()
    sys.exit(application.exec_())
```

Listing 37.2: QLabel Example

This gives the window as shown in Figure 37.4.

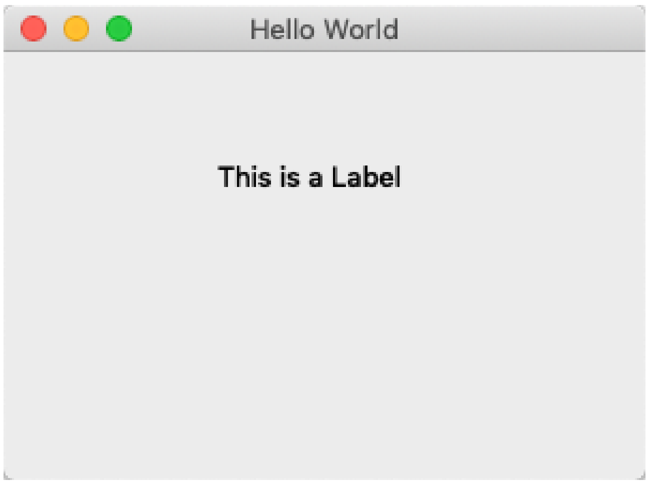


Figure 37.4: QLabel Example

[End of Example]

Example 37.6.2. TextBox (QLineEdit)

A basic TextBox example:

```python
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLineEdit
if __name__ == "__main__":
    application = QApplication(sys.argv)
    window = QWidget()
    window.resize(300,200)
    window.setWindowTitle("Hello World")

    inputbox = QLineEdit(window)
    inputbox.setText("Please enter your Name")
    inputbox.resize(200, 20)
    inputbox.move(20,50)

    window.show()
    sys.exit(application.exec_())
```
Listing 37.3: QLineEdit EXample

This gives the window as shown in Figure 37.5.

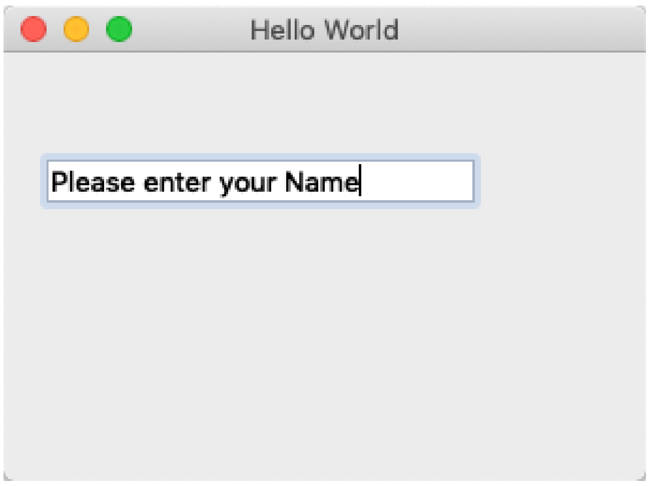


Figure 37.5: QLineEdit Example

You can use the following if you want a read-only field:

```
inputbox.setReadOnly(True)
```

[End of Example]

Example 37.6.3. ComboBox (QComboBox)

A basic ComboBox box example:

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QComboBox
if __name__ == "__main__":
    application = QApplication(sys.argv)
    window = QWidget()
    window.resize(300,200)
    window.setWindowTitle("Hello World")

    combobox = QComboBox(window)
    combobox.addItems(["item one", "item two", "item three"])
    combobox.move(100,50)

    window.show()
    sys.exit(application.exec_())
```
Listing 37.4: QLineEdit Example

This gives the window as shown in Figure 37.6.

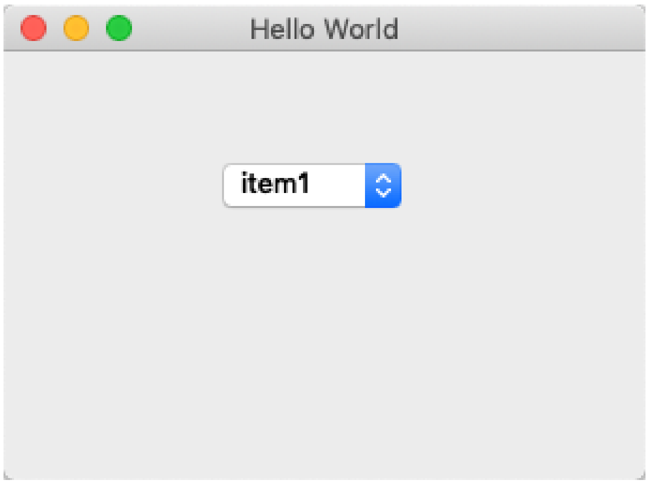


Figure 37.6: QComboBox Example

We can change the visual appearance by using the following:

```
application.setStyle('Fusion')
```

[End of Example]

37.7 Event Handling in PyQt

PyQt uses a mechanism called **signals** to let you react to events such as the user clicking a button.

Example:

```
self.btnShowData.clicked.connect(self.btnShowData_clicked)
```

You need to define a function that is called when the signal occurs. This function is called a **slot** in PyQt.

Example:

```
def btnShowData_clicked(self):
      temp = np.random.randint(20,50)
      self.txtTemp.setText(str(temp))
```

Example 37.7.1. Button Click

Below you see a basic Push Button example:

```python
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton

def button_clicked():
   print("You clicked the Button")

if __name__ == "__main__":
    application = QApplication(sys.argv)
    application.setStyle('Fusion')
    window = QWidget()
    window.resize(300,200)
    window.setWindowTitle("Hello World")

    button = QPushButton(window)
    button.setText("Click Me")
    button.move(100,50)
    button.clicked.connect(button_clicked)

    window.show()
    sys.exit(application.exec_())
```

Listing 37.5: Button Click

[End of Example]

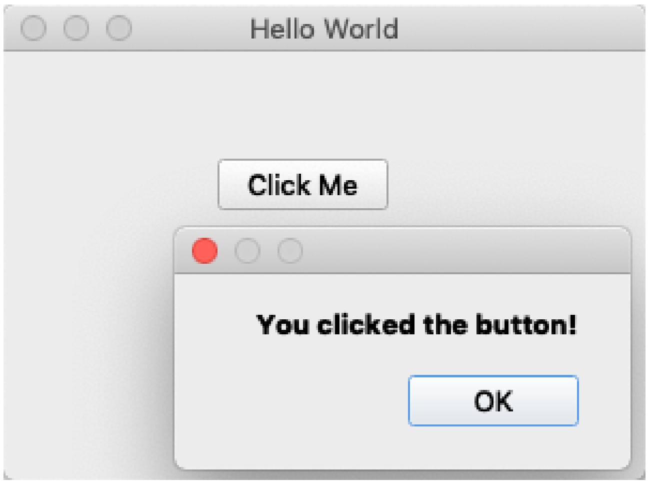
Example 37.7.2. MessageBox (QMessageBox)

MessageBox example:

```python
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton,
    QMessageBox

def button_clicked():
   messagebox = QMessageBox()
   messagebox.setText('You clicked the button!')
   messagebox.exec_()

if __name__ == "__main__":
    application = QApplication(sys.argv)
    application.setStyle('Fusion')
    window = QWidget()
    window.resize(300,200)
    window.setWindowTitle("Hello World")

    button = QPushButton(window)
    button.setText("Click Me")
    button.move(100,50)
    button.clicked.connect(button_clicked)

    window.show()
    sys.exit(application.exec_())
```

Listing 37.6: MessageBox Example

This gives the window as shown in Figure 37.7.



You clicked the button!

Figure 37.7: QMessageBox Example

[End of Example]

Example 37.7.3. Combining different Widgets and Event Handling Example

In this example we combine different Widgets and some Event Handling:

```python
import sys
import numpy as np
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton,
    QLineEdit, QMessageBox

class Application(QWidget):

   def __init__(self):
      super().__init__()
      self.initUI()

   def initUI(self):
      self.setWindowTitle("PyQt Example")
      self.resize(400, 300)


      self.txtName = QLineEdit(self)
      self.txtName.setText("Please enter your Name")
      self.txtName.move(20, 20)
      self.txtName.resize(200, 20)

      self.btnShowName = QPushButton("Show Name", self)
      self.btnShowName.move(15,60)
      self.btnShowName.clicked.connect(self.btnShowName_clicked)

      self.show()


   def btnShowName_clicked(self):
      self.showMessage()

   def showMessage(self):

        name = self.txtName.text()
        text = "Is your Name " + name + "?"
        reply = QMessageBox.question(self, 'Message', text,
    QMessageBox.Yes | QMessageBox.No, QMessageBox.No)

        if reply == QMessageBox.No:
            self.txtName.setText("Please enter your Name")


if __name__ == '__main__':
   app = QApplication(sys.argv)
   app.setStyle('Fusion')
   window = Application()
   sys.exit(app.exec_())
```
Listing 37.7: Combining different Widgets and Event Handling Example

This gives the window as shown in Figure 37.8.

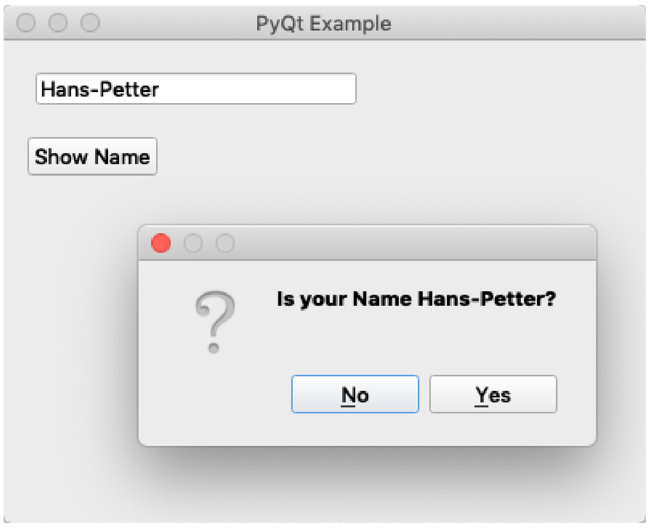


Figure 37.8: PyQt Application Example

[End of Example]

37.8 PyQt Designer

...

37.9 PyQt Applications Examples

Here some real-life examples of PyQt will be provided.

Some examples are:

- Logging Data from a Sensor and presenting the values on the screen
- Simulation of a system and presenting values on the screen
- Control System
- etc.

Example 37.9.1. Example Name

...

```
...
...
```

Listing 37.8: xxx

Example 37.9.2. Example Name

...

```
...
...
```

Listing 37.9: xxx

Example 37.9.3. Example Name

...

```
...
...
```

Listing 37.10: xxx

Part IX

Resources

Chapter 38

Python Resources

Here you find my Web page with Python resources [1]:
https://www.halvorsen.blog/documents/programming/python/

Python Home Page [6]:
https://www.python.org

Python Standard Library [19]:
https://docs.python.org/3/library/index.html

38.1 Python Distributions

Anaconda:
https://www.anaconda.com

38.2 Python Libraries

NumPy Library:
http://www.numpy.org

SciPy Library:
https://www.scipy.org

Matplotlib Library:
https://matplotlib.org

38.3 Python Editors

Spyder:
https://www.spyder-ide.org

Visual studio Code:
https://code.visualstudio.com

Visual Studio:
https://visualstudio.microsoft.com

PyCharm:
https://www.jetbrains.com/pycharm/

Wing:
https://wingware.com

Jupyter Notebook:
http://jupyter.org

38.4 Python Tutorials

Python Tutorial - w3schools.com [13]:
https://www.w3schools.com/python/

The Python Guru [20]:
https://thepythonguru.com

Wikibooks - A Beginner's Python Tutorial:
https://en.wikibooks.org/wiki/A$_Beginner$

TutorialsPoints - Python Tutorial:
https://www.tutorialspoint.com/python/

The Hitchhiker's Guide to Python:
https://docs.python-guide.org

Google's Python Class:
https://developers.google.com/edu/python/

38.5 Python in Visual Studio

Work with Python in Visual Studio
https://docs.microsoft.com/visualstudio/python/

Bibliography

[1] H.-P. Halvorsen, “Technology blog - https://www.halvorsen.blog,” 2018.

[2] H.-P. Halvorsen, “Technology blog - https://en.wikipedia.org/wiki/Python$_{(}programming_{l}anguage),''$ 20

[3] T. . T. P. Languages, “The 2018 top programming languages - https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages,” 2018.

[4] S. Overflow, “Stack overflow developer survey 2018 - https://insights.stackoverflow.com/survey/2018/,” 2018.

[5] stackoverflow.blog, “The incredible growth of python - https://stackoverflow.blog/2017/09/06/incredible-growth-python/,” 2018.

[6] python.org, “python.org - https://www.python.org,” 2018.

[7] python.org, “The python tutorial - https://docs.python.org/3.7/tutorial/,” 2018.

[8] python.org, “Python 3.7.1 documentation - https://docs.python.org/3.7/,” 2018.

[9] scipy.org, “Scipy - https://www.scipy.org,” 2018.

[10] matplotlib.org, “Matplotlib - https://matplotlib.org,” 2018.

[11] pandas, “pandas - http://pandas.pydata.org,” 2018.

[12] Wingware, “Wingware python ide - https://wingware.com,” 2018.

[13] w3schools.com, “Python tutorial - https://www.w3schools.com/python/,” 2018.

[14] Wikipedia, “Debugging - https://en.wikipedia.org/wiki/Debugging,” 2018.

[15] TechBeamers, “Get the best python ide - https://www.techbeamers.com/best-python-ide-python-programming/,” 2018.

[16] Jupyter, “Jupyter - https://jupyter.org,” 2018.

[17] JupyterHub, “Jupyterhub - http://jupyter.org/hub,” 2018.

[18] python.org, "Applications for python - https://www.python.org/about/apps/," 2018.

[19] python.org, "The python standard library - https://docs.python.org/3/library/," 2018.

[20] T. P. Guru, "The python guru - https://thepythonguru.com," 2018.

Part X

Solutions to Exercises

Start using Python

Simulation and Plotting of Dynamic System

Given the autonomous system:

$$\dot{x} = ax \tag{1}$$

Where:

$$a = -\frac{1}{T}$$

where T is the time constant.

The solution for the differential equation is:

$$x(t) = e^{at}x_0 \tag{2}$$

Set T=5 and the initial condition x(0)=1.

Create a Script in Python (.py file) where you plot the solution x(t) in the time interval:

$$0 \leq t \leq 25$$

Add Grid, and proper Title and Axis Labels to the plot.

Python Script:

```
import math as mt
import numpy as np
import matplotlib.pyplot as plt


# Model Parameters
T = 5
a = -1/T

# Simulation Parameters
x0 = 1
t = 0

tstart = 0
```

```
tstop = 25

increment = 1

x = []
x = np.zeros(tstop+1)

t = np.arange(tstart, tstop+1,increment)


# Define the Function
for k in range(tstop):
    x[k] = mt.exp(a*t[k]) * x0


# Plot the Simulation Results
plt.plot(t,x)
plt.title('Simulation of Dynamic System')
plt.xlabel('t')
plt.ylabel('x')
plt.grid()
plt.axis([0, 25, 0, 1])
plt.show()
```

The simulation gives the results as shown in Figure 1.

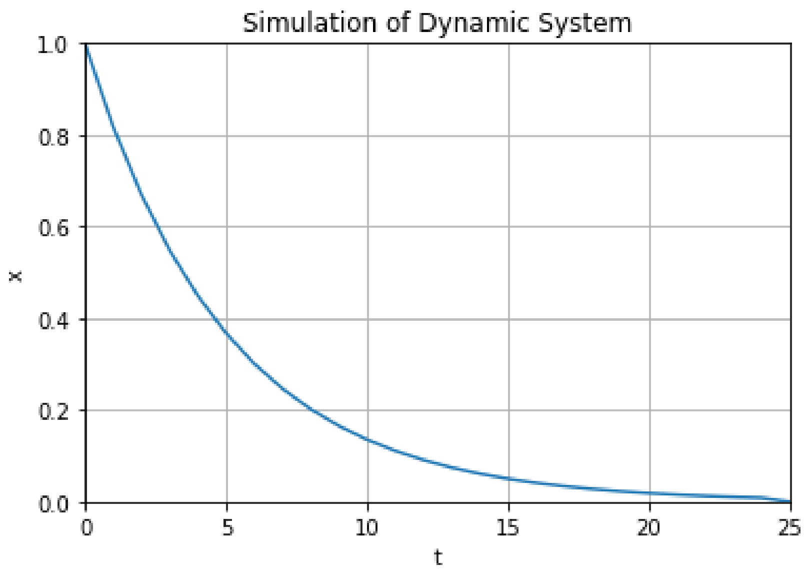


Figure 1: Simulation of Dynamic System

[End of Exercise]

Mathematics in Python

Create Mathematical Expressions in Python

Create a function that calculates the following mathematical expression:

$$z = 3x^2 + \sqrt{x^2 + y^2} + e^{\ln(x)} \tag{3}$$

Test with different values for x and y.

We create a Python Module with a Python Function (mymathfunctions.py):

```python
import math as mt

def calcexpression(x,y):

    z = 3*x**2 + mt.sqrt(x**2 + y**2) + mt.exp(mt.log(x))
    return z
```

Then we can create a Python Script in order to test the function:

```python
import mymathfunctions as mymath

x = 2
y = 2

z = mymath.calcexpression(x,y)

print(z)
```

The results become:

```
16.82842712474619
```

[End of Solution]

Create advanced Mathematical Expressions in Python

Create the following expression in Python:

$$f(x) = \frac{\ln(ax^2 + bx + c) - sin(ax^2 + bx + c)}{4\pi x^2 + cos(x-2)(ax^2 + bx + c)} \tag{4}$$

Given $a = 1, b = 3, c = 5$ Find $f(9)$
(The answer should be $f(9) = 0.0044$)

Tip! You should split the expressions into different parts, such as:

$$poly = ax^2 + bx + c$$

$num = \ldots$
$den = \ldots$
$f = \ldots$

This makes the expression simpler to read and understand, and you minimize the risk of making an error while typing the expression in Python.

When you got the correct answer try to change to, e.g., $a = 2, b = 8, c = 6$

Find $f(9)$

Python Script:

```
...
```

[End of Solution]

Discrete Systems

Bacteria Population

In this task we will simulate a simple model of a bacteria population in a jar.

The model is as follows:

$$\text{birth rate} = bx \tag{5}$$

$$\text{death rate} = px^2 \tag{6}$$

Then the total rate of change of bacteria population is:

$$\dot{x} = bx - px^2 \tag{7}$$

Set b=1/hour and p=0.5 bacteria-hour

We will simulate the number of bacteria in the jar after 1 hour, assuming that initially there are 100 bacteria present.

Find the discrete model using the Euler Forward method by hand and implement and simulate the system in Python using a For Loop.

We can use e.g., the Euler Approximation:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s} \tag{8}$$

T_s - Sampling Interval

Then we get:

$$\frac{x_{k+1} - x_k}{T_s} = bx_k - px_k^2 \tag{9}$$

This gives the following discrete differential equation:

$$x_{k+1} = x_k + T_s(bx_k - px_k^2) \tag{10}$$

Now we are ready to simulate the system.

Python Script:

```python
# Simulation of Bacteria Population
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
b = 1
p = 0.5

# Simulation Parameters
Ts = 0.01
Tstop = 1
xk = 100
N = int(Tstop/Ts) # Simulation length
data = []
data.append(xk)


# Simulation
for k in range(N):
    xk1 = xk + Ts* (b * xk - p * xk**2);
    xk = xk1
    data.append(xk1)

# Plot the Simulation Results
t = np.arange(0,Tstop+Ts,Ts)

plt.plot(t,data)
plt.title('Simulation of Bacteria Population')
plt.xlabel('t [s]')
plt.ylabel('x')
plt.grid()
plt.axis([0, 1, 0, 100])
plt.show()
```

The simulation gives the results as shown in Figure 2.

[End of Solution]

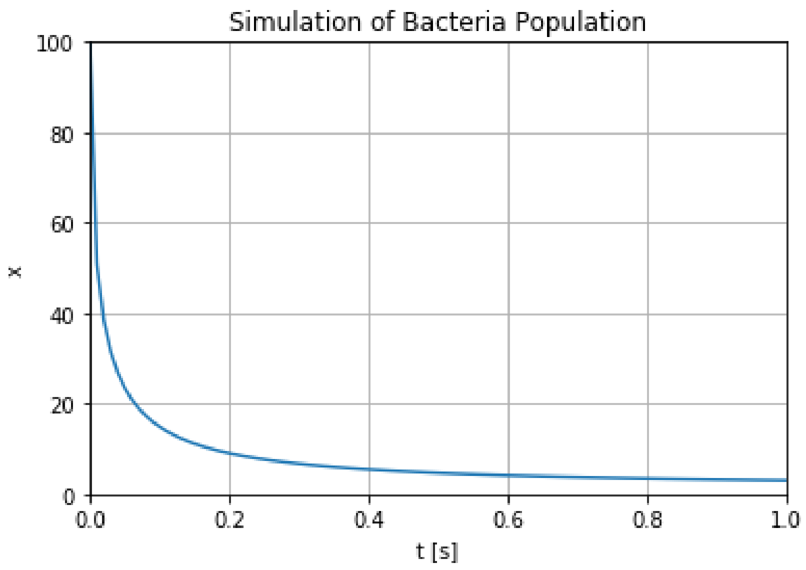


Figure 2: Simulation of Bacteria Population

Simulation with 2 variables

Given the following system:

$$\frac{dx_1}{dt} = -x_2 \tag{11}$$

$$\frac{dx_2}{dt} = x_1 \tag{12}$$

Find the discrete system and simulate the discrete system in Python. Solve the equations, e.g., in the time span [-1 1] with initial values [1, 1].

Python Script:

```python
# Simulation with 2 Variables
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
b = 1
p = 0.5

# Simulation Parameters
Ts = 0.1
Tstart = -1
Tstop = 1
x1k = 1
x2k = 1
N = int((Tstop-Tstart)/Ts) # Simulation length
datax1 = []
```

```
datax2 = []
datax1.append(x1k)
datax2.append(x2k)


# Simulation
for k in range(N):
    x1k1 = x1k - Ts * x2k
    x2k1 = x2k + Ts * x1k

    x1k = x1k1
    x2k = x2k1
    datax1.append(x1k1)
    datax2.append(x2k1)

# Plot the Simulation Results
t = np.arange(Tstart,Tstop+Ts,Ts)

plt.plot(t,datax1, t, datax2)
plt.title('Simulation with 2 Variables')
plt.xlabel('t [s]')
plt.ylabel('x')
plt.grid()
plt.axis([-1, 1, -1.5, 1.5])
plt.show()
```

The simulation gives the results as shown in Figure 2.

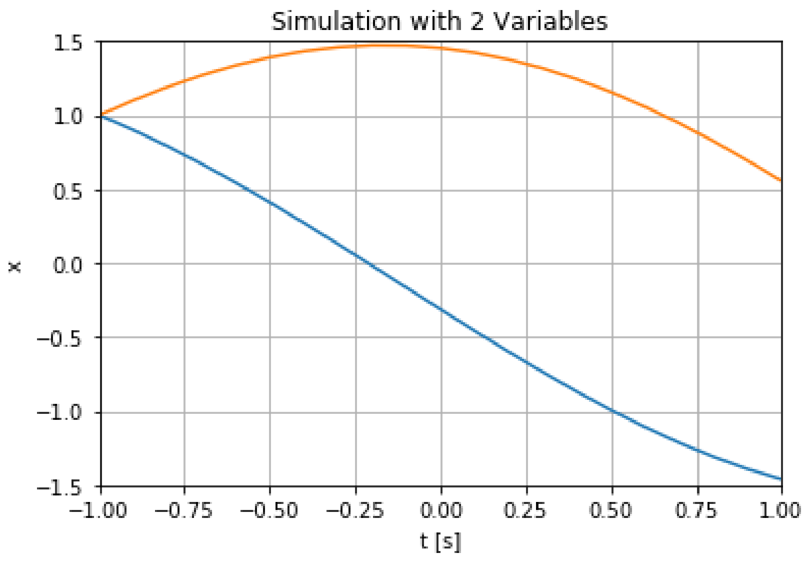


Figure 3: Simulation Example with 2 Variables

Alternative Solution:

```
# Simulation with 2 Variables
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
```

```python
b = 1
p = 0.5

# Simulation Parameters
Ts = 0.1
Tstart = -1
Tstop = 1
N = int((Tstop-Tstart)/Ts) # Simulation length
x1 = np.zeros(N+2)
x2 = np.zeros(N+2)
x1[0] = 1
x2[0] = 1


# Simulation
for k in range(N+1):
    x1[k+1] = x1[k] - Ts * x2[k]
    x2[k+1] = x2[k] + Ts * x1[k]


# Plot the Simulation Results
t = np.arange(Tstart,Tstop+2*Ts,Ts)

plt.plot(t,x1, t, x2)
plt.title('Simulation with 2 Variables')
plt.xlabel('t [s]')
plt.ylabel('x')
plt.grid()
plt.axis([-1, 1, -1.5, 1.5])
plt.show()
```

Choose the approach that fits you. You should also check the time that the simulation take. For larger simulations, this second alternative may be better.

[End of Solution]

python™

Python for Software Development