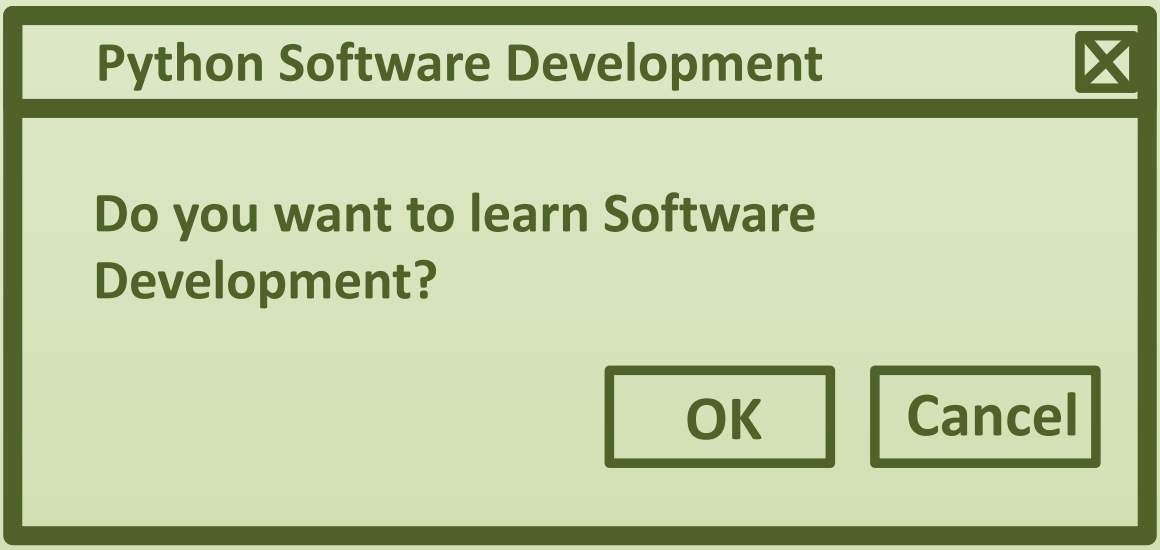
Python for Software Development

Hans-Petter Halvorsen



https://www.halvorsen.blog

Python for Software Development

Python for Software Development

Hans-Petter Halvorsen

2020

Python for Science and Engineering

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/

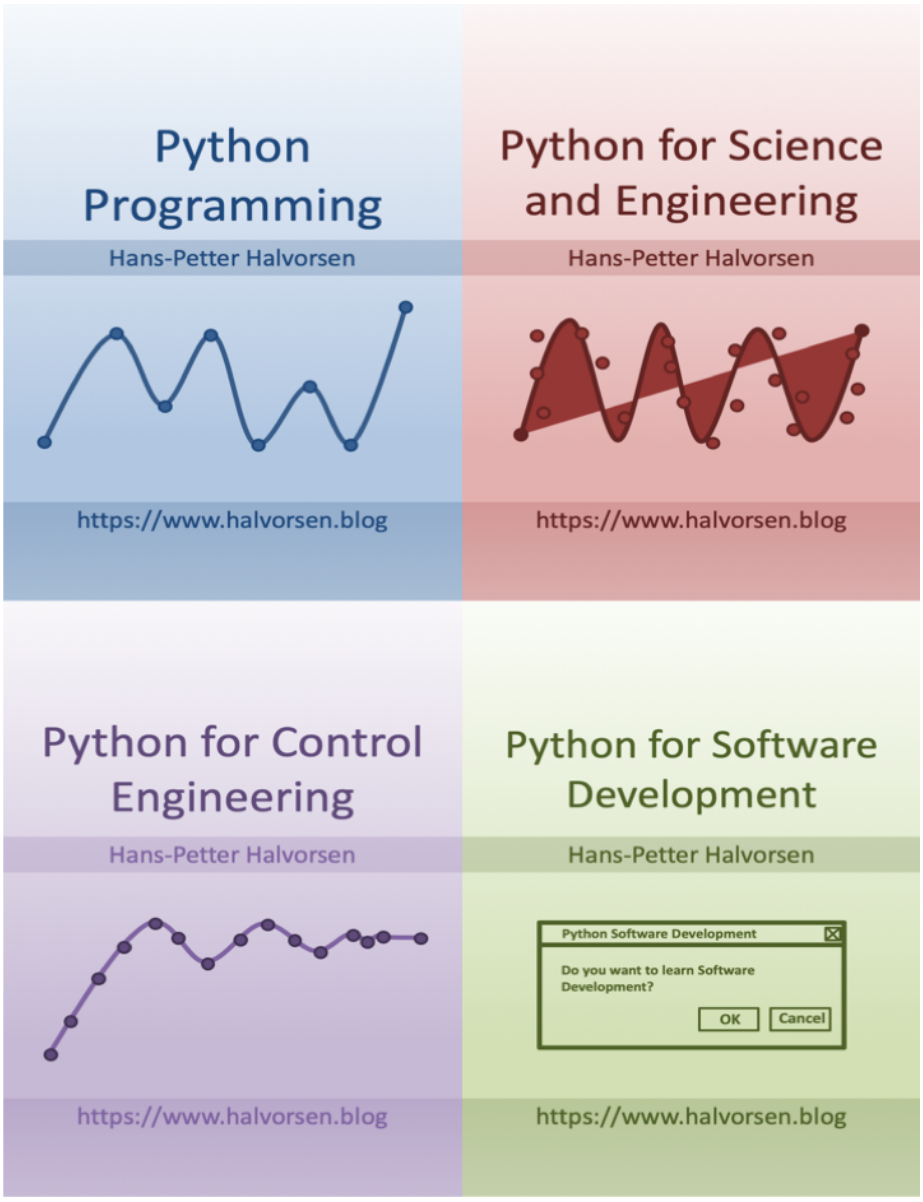
Software Engineering

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$_e$*ngineering*/


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].

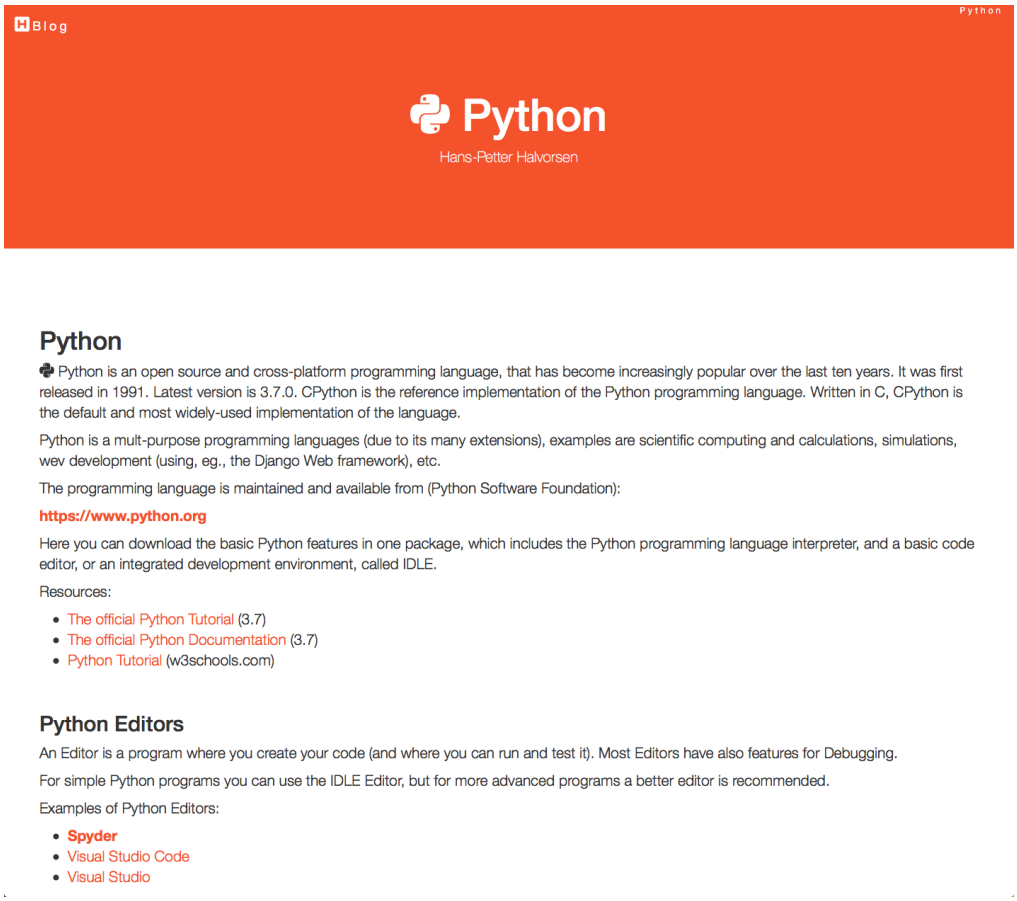


Figure 1.1: Web Site - Python

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

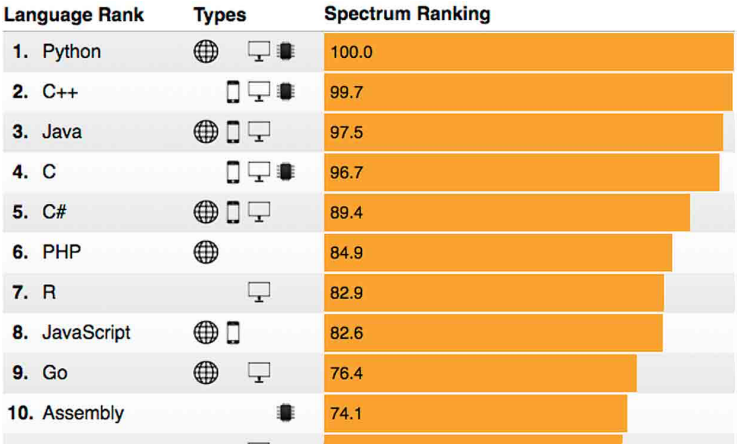


Figure 1.2: The Most Popular Programming Languages

- 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.

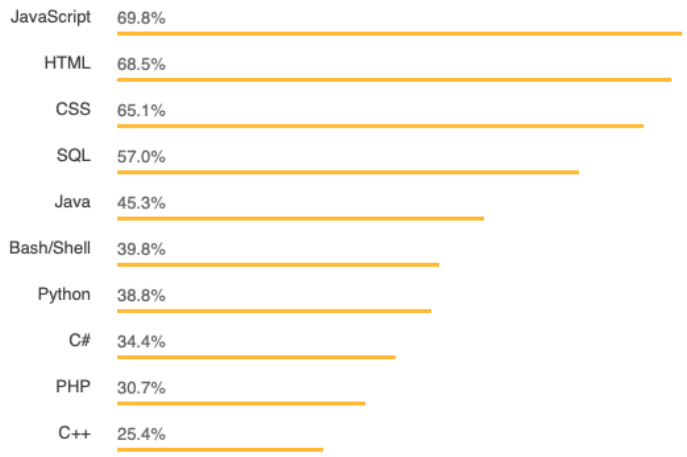


Figure 1.3: The Top Programming Languages - Stack Overflow Survey

- 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)

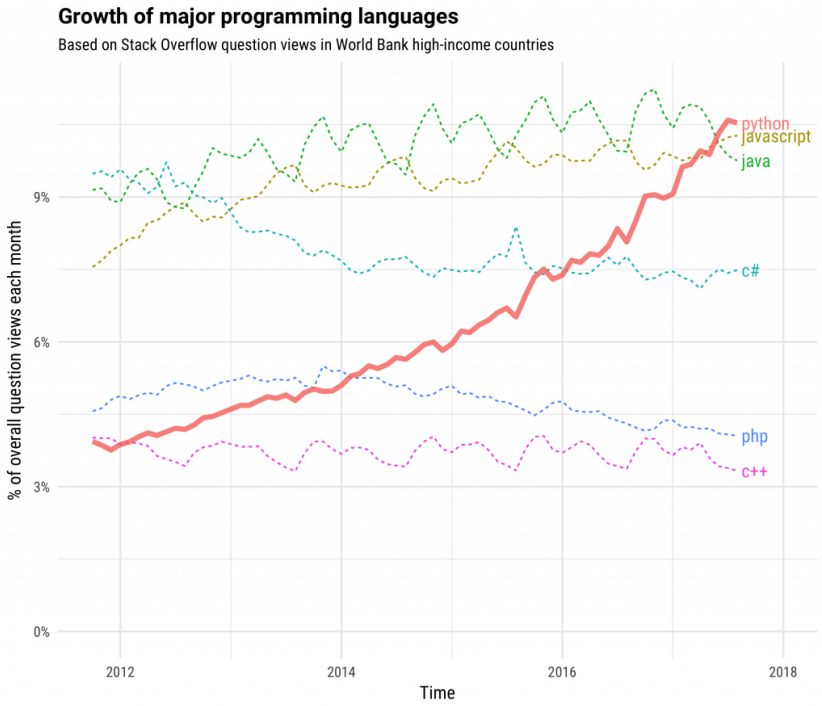


Figure 1.4: The Incredible Growth of Python

- 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/

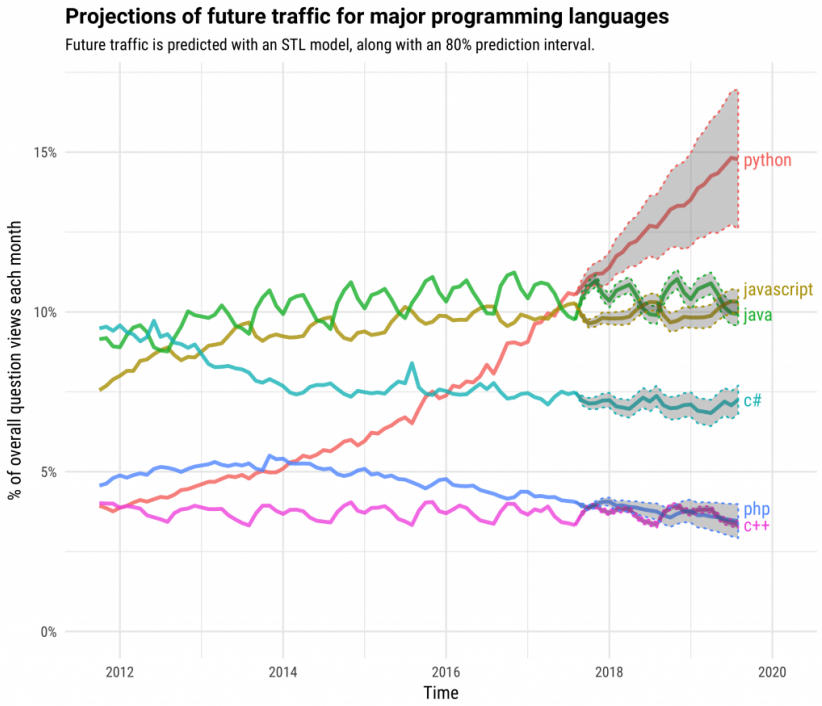


Figure 1.5: The Future of Python

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

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$_{V}isual_{S}tudio$

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.

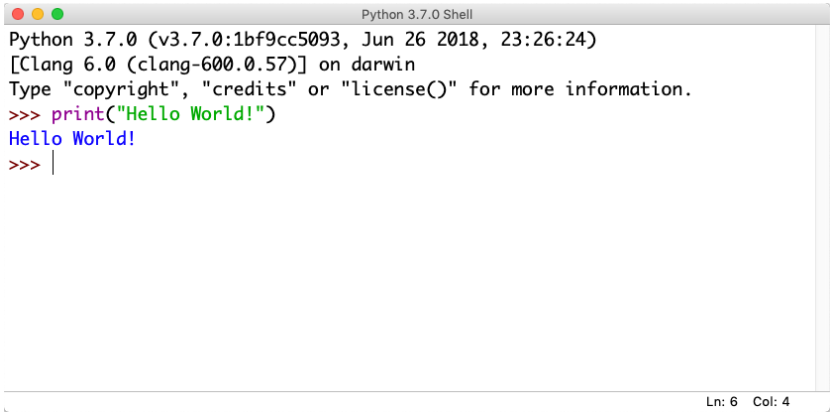


Figure 3.1: Python Shell / Python IDLE Editor

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:

```
1 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.

```
1 >>> 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.

```
1 Last login: Tue Dec 11 12:34:16 on ttys000
2 Hans-Petter-Work-MacBook-Air:~ hansha$ python
3 Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)
4 [GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on
      darwin
5 Type "help", "copyright", "credits" or "license" for more
      information.
6 >>>
```

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.

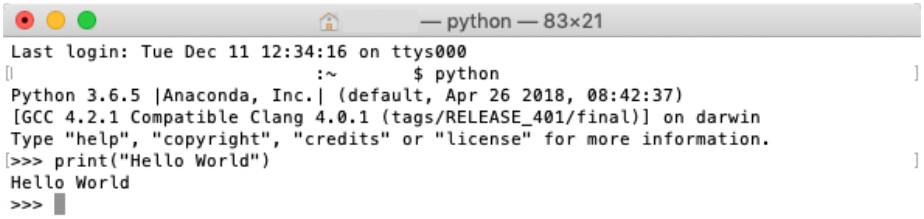


Figure 3.2: Console macOS

Try other Python commands, e.g.:

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

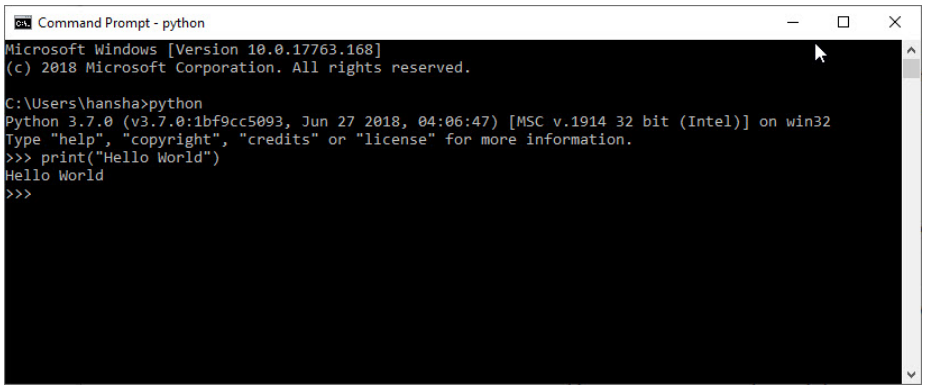


Figure 3.3: Command Prompt Windows

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.

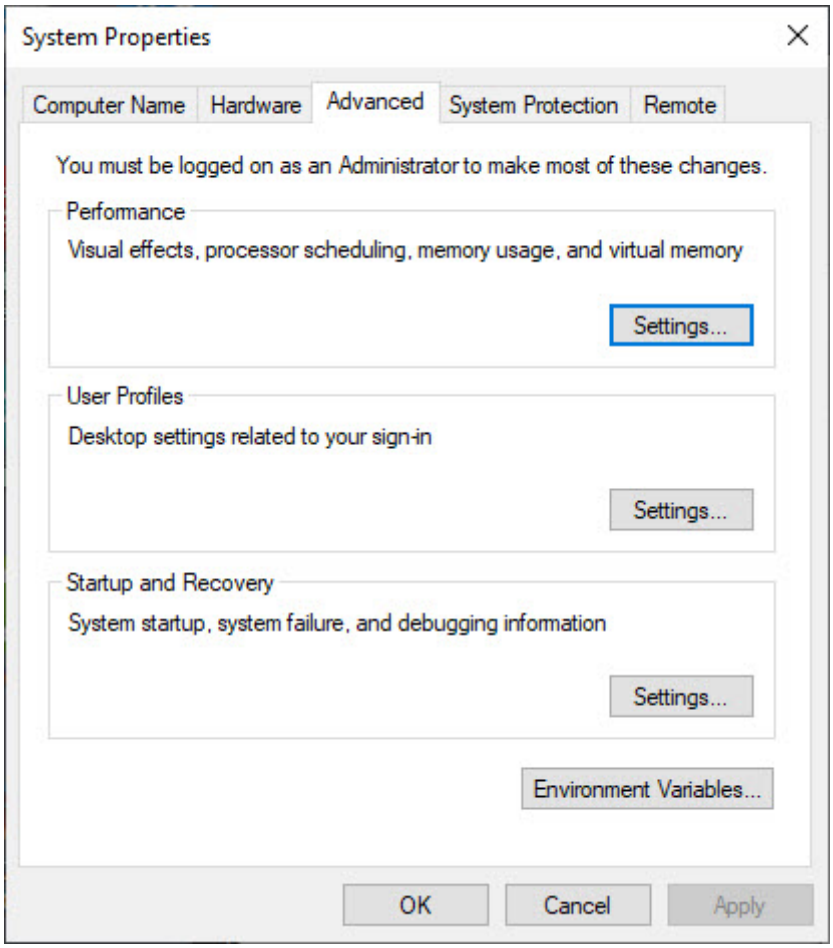


Figure 3.4: Windows System Properties

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.

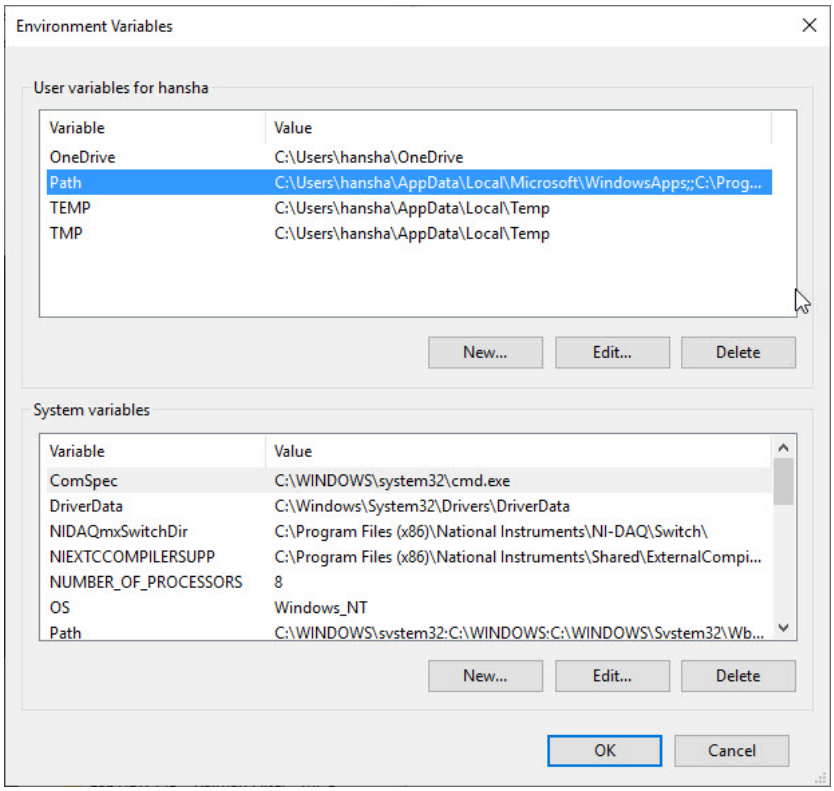


Figure 3.5: Windows System Properties

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:

```
1 print("Hello")
2 print("World")
3 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.

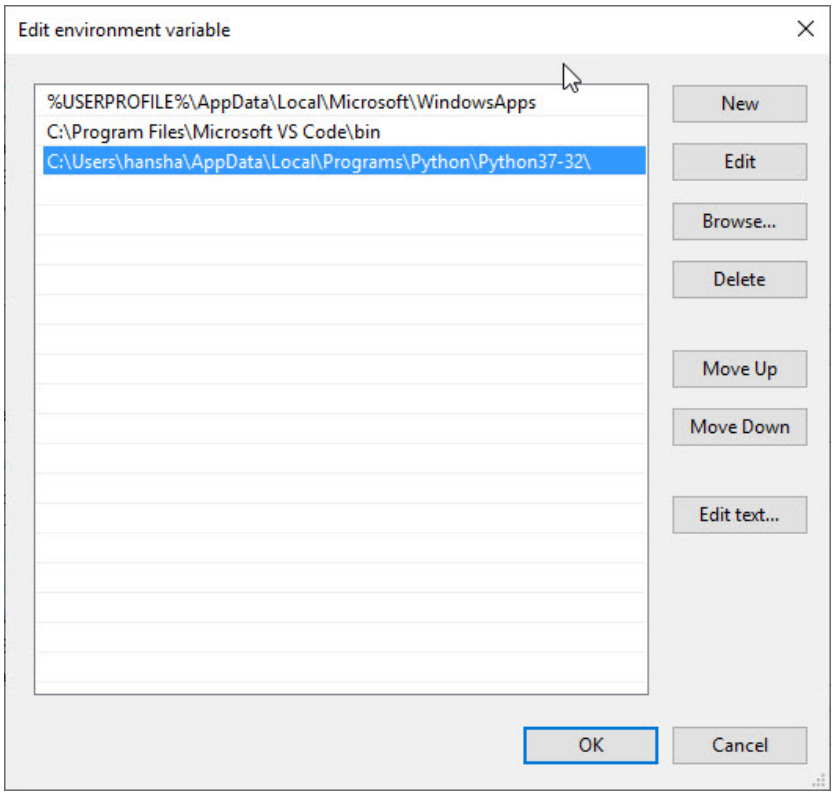


Figure 3.6: Windows System Properties

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?
```

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.

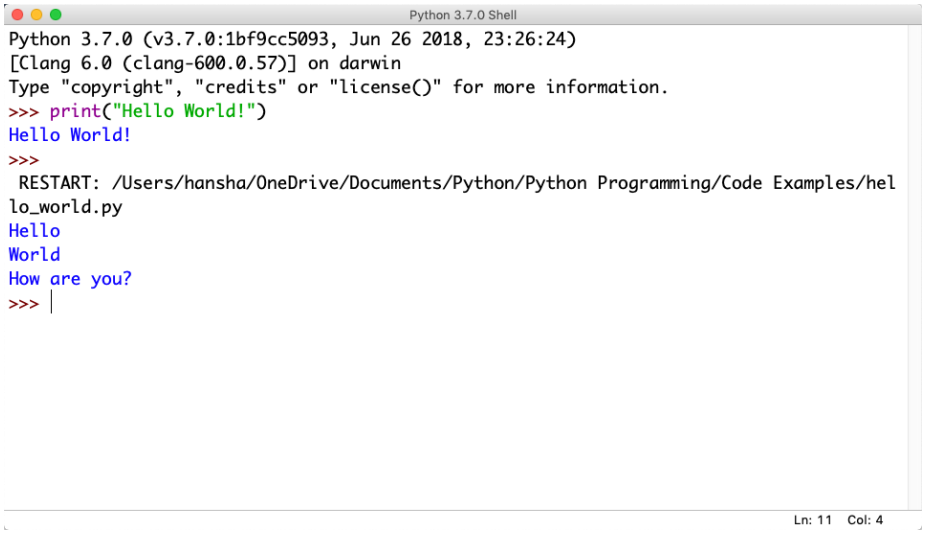


Figure 3.8: Running a Python Script

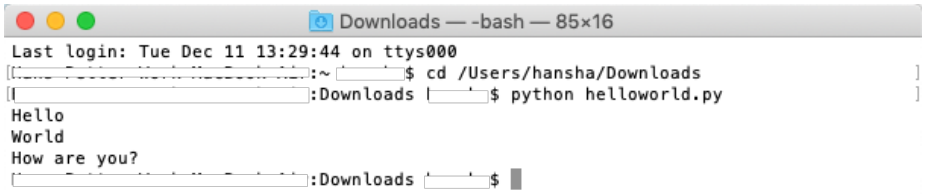


Figure 3.9: Running Python Scripts from Console window on macOS

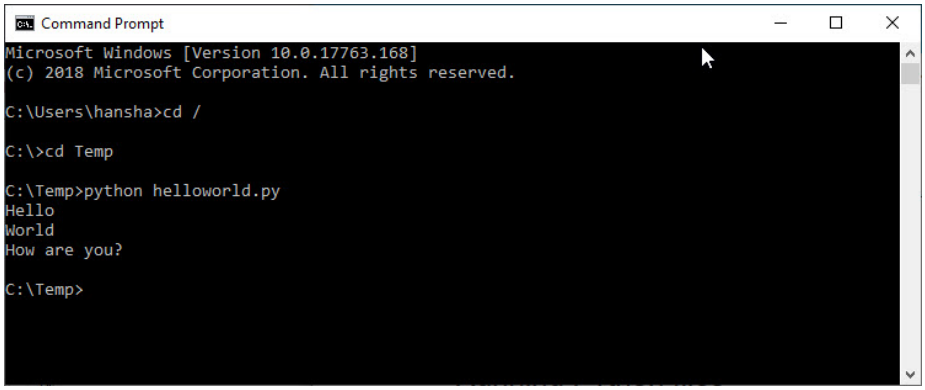


Figure 3.10: Running Python Scripts from Console window on macOS

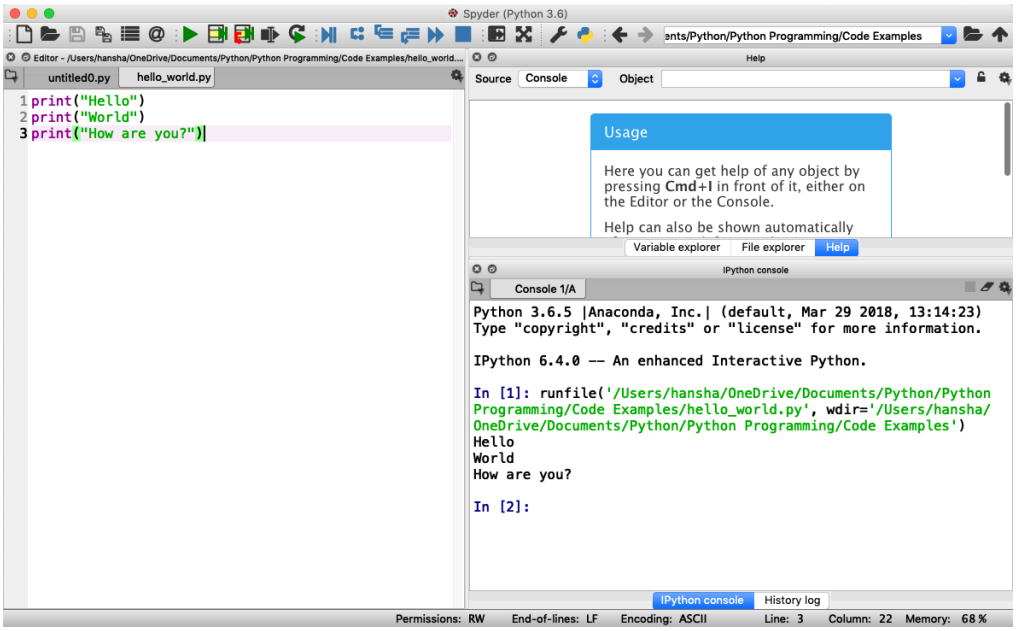


Figure 3.11: Running a Python Script in Spyder

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:

```
1 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.

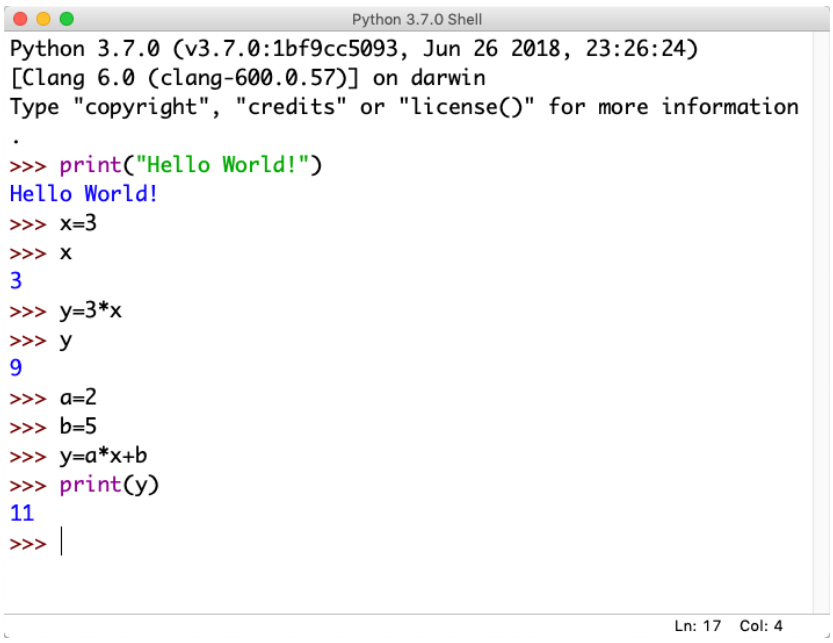


Figure 4.1: Basic Python

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

```
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.

```
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):

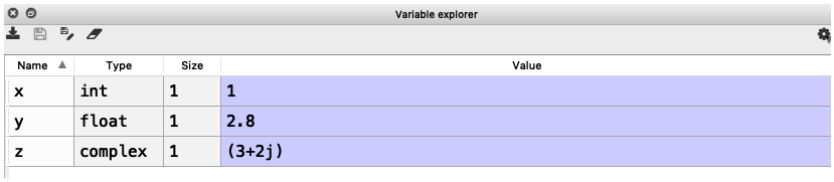


Figure 4.2: Variable Editor in Spyder

[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:

```
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:

```
from math import sin

x = 3.14
y = sin(x)

print(y)
```

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

```
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:

```
from math import *

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

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

We can also use this alternative:

```
import math

x = 3.14
y = math.sin(x)

print(y)
```

We can also write it like this:

```
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:

```
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:

```
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:

```
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.

```
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:

```
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.

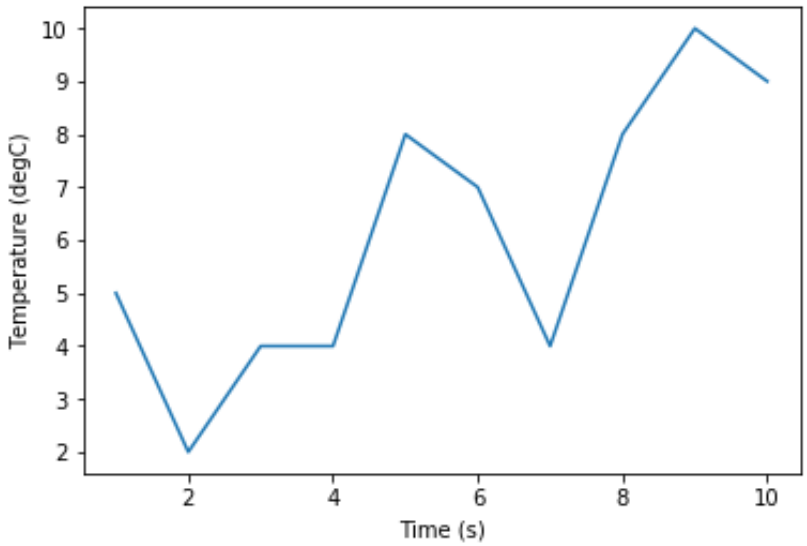


Figure 4.3: Plotting in Python

[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

```
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:

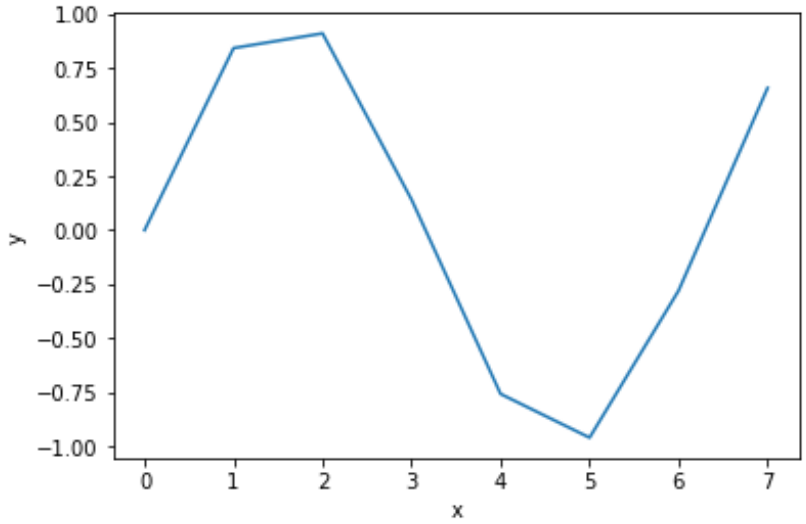


Figure 4.4: Plotting a Sine function in Python

```
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

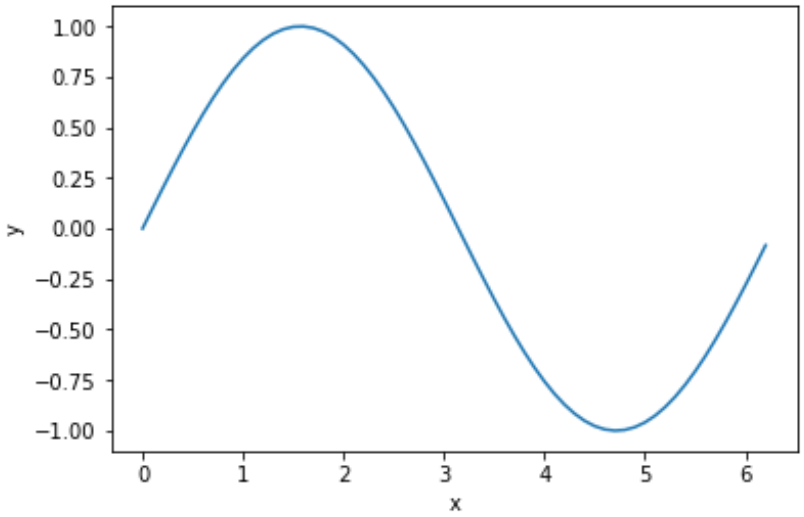


Figure 4.5: Plotting a Sine function in Python - Better Implementation

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]

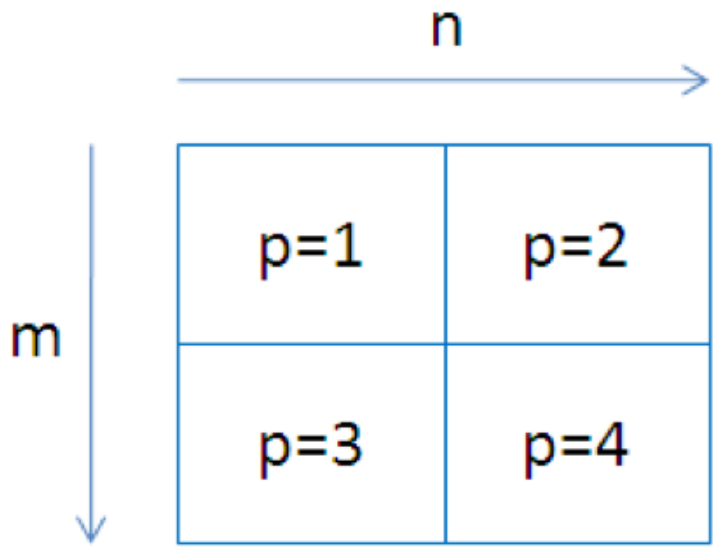


Figure 4.6: Creating Subplots in Python

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:
```