
python

Languages

- Solve problems using a computer, give the computer instructions.
- Remember our diaper-changing exercise?

Talk the talk

- Speak its language
  - High-level: Python, C++, Java
  - Low-level: machine language, computers can only execute these
  - High-level languages have to be processed into low-level before the computer can run them
  - But high-level languages can run on different kinds of computers and are easier for humans to write and read, so most programs are written in high-level

Translation

- How does high-level get translated into low-level?
  - Interpreters and compilers!
  - Interpreter processes the program a little bit at a time and runs it
  - Compiler translates everything before running it

What is python ?

Python is a programming language that lets you work more quickly and integrate your systems more effectively.

(in other words, magic!)

http://www.python.org/

Python Vocab

- ***Program*** – A larger file of code that may contain one or more functions.
- ***Variable*** - names that you can assign values to, allowing you to reuse them later on.

  E.g.: `x = 1` or `msg = "Hi, I'm a message!"`
- ***Comments*** – These are notes ignored by the computer. In Python, comments start with a hash mark (#) and end at the end of the line.

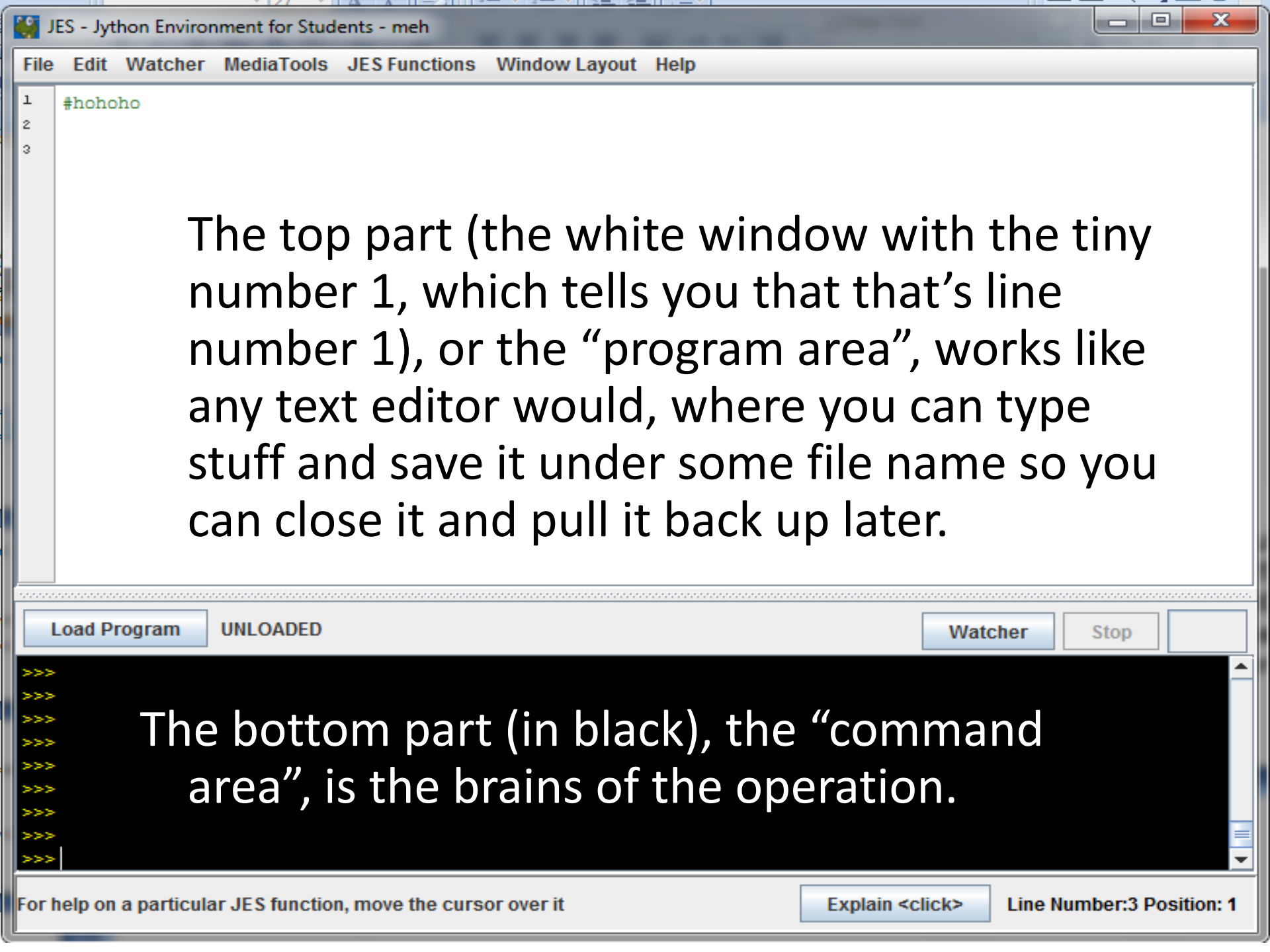
  E.g.: `>>> x + y #both variables store user input`
- ***Operators*** – Mathematical symbols, like +, -, *, and /, but also ** (for exponents).

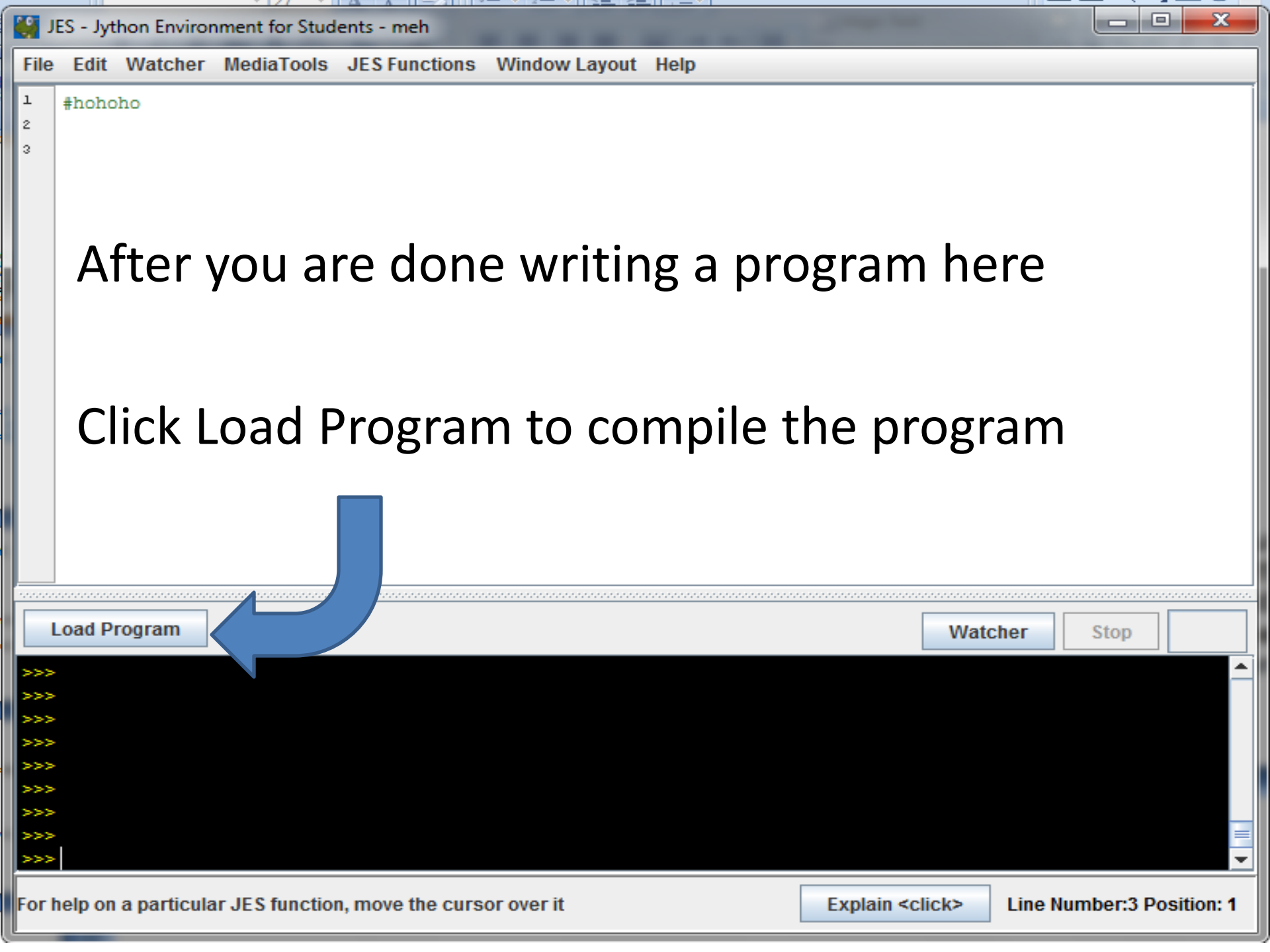
Python Vocab

- ***Keyword*** – Words with meaning/purpose in Python. E.g. “`and`”, “`print`”, and “`if`”.
- ***Function*** – A chunk of code that performs an action. Functions have names and are reusable. Kinds: built-in ones, and “user-defined” ones.
- ***Expression*** – Statements that produce values. Examples include 3 + 5, "Hello world!“.
- ***Error*** – When your program has a problem, the command area will provide an error message.

What is JES?

Jython Environment for Students allows you to program and experiment with python.


JES - Jython Environment for Students - meh
File Edit Watcher MediaTools JES Functions Window Layout Help
1 #hohoho
2
3
The top part (the white window with the tiny number 1, which tells you that that's line number 1), or the "program area", works like any text editor would, where you can type stuff and save it under some file name so you can close it and pull it back up later.
Load Program
UNLOADED
Watcher
Stop
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
The bottom part (in black), the "command area", is the brains of the operation.
For help on a particular JES function, move the cursor over it
Explain <click>
Line Number:3 Position: 1


JES - Jython Environment for Students - meh
File Edit Watcher MediaTools JES Functions Window Layout Help
1 #hohoho
2
3
After you are done writing a program here
Click Load Program to compile the program
Load Program
Watcher
Stop
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
For help on a particular JES function, move the cursor over it
Explain <click>
Line Number:3 Position: 1

Writing Your Program

Header Comments:

```
# file name: circ.py
# author: Durrah Almansour
# description: a program to
calculate the area of a circle.
```

Indentation

- Not an Option, but a Requirement!
- In Python, indenting specifies the “scope” of different chunks of your code.

```
def main():
    print "Hello world!"
```

The print line will be executed when you invoke main().

Writing Your Program

- Always plan what you want your program to do (pseudocode).
- Divide it into parts, it will make it easier to debug and update later.

Writing Your Program

Defining Functions:

```
def main():
    print "Hello world!"
```

- Name the function for the purpose you wrote it for.
- Don’t forget to indent the instructions that go inside the function.

Argument

A value passed to a function or method, assigned to a named local variable in the function body.

Ex.
```
def addUp(a,b):
        print("this is a+b: ", a+b)
```

#a and b are the *parameters*, 1 and 3 are the arguments we passed in.

#The function outputs : this is 1+3: 4

http://docs.python.org/glossary.html#glossary

Calling Functions

```
main()
```

will give the output
Hello world!

- There is no colon when calling a function!
- Use colons with “def”.

- Try this yourself

```
>>>
>>>
>>>
>>> def main():
...     print "Artemis is Awesome"
...
>>> main()
```

DON'T FORGET TO INDENT!

```
>>>
>>>
>>>
>>> def main():
...     print "Artemis is Awesome"
...
>>> main()
Artemis is Awesome
>>>
```

Output

E.g. Print:

```
print "Hello world!"
```

will give the output

Hello world!

Similar formatting, different output

- `print "Hello", "world", "!"`
  will output
  Hello world !

- `print "Hello" + "world" + "!"`
  will output
  Helloworld!

Similar formatting, different output

- ```
  print "Hello"
  print "world"
  print "!"
  ```
- will give the output

  Hello
  world
  !

- ```
  print "Hello",
  print "world",
  print "!"
  ```
- will give the output

  Hello world !

Data Types

The data type of an object determines the values it can hold, and the operations which can be performed on it.

- Numeric Data

Numeric data comes in 2 main flavors:

– Integers (whole numbers) 2, 5, -7, etc.

– Floating Point Numbers (non-integers) 0.2, 5.125, etc.

(CS108 Notes, Prof. Aaron Stevens)

Data Types

- Non-numeric Data Types:

  Those include strings (text), lists, dictionaries, etc..

  Basically anything you can not add up using a simple plus sign (+).

Not a String? Not a Problem!

You can also format outputting *variables* you've defined:

- ```
  x = 42
  print" The value of x is", x, "."
  ```
- will give the output
  The value of x is 42 .


  ```

Not a String? Not a Problem!

- ```
  x = 42
  print "$" + x
  ```
  causes an error.
- So what do we do?

  ```
  x = 42
  print "$" + str(x)
  ```
  will give the output
  $42


  ```

Defining Variables

Rules for naming variables:

- Have to start with a letter or underscore (_)
- Can contain letters, numbers, and underscores
- Can’t contain spaces, punctuation, etc.
- Can’t be Python keywords
- Are case sensitive

Defining Variables

Things that aren't rules, but are worth considering:

- You should give your variables sensible names ("price", "pixelColor, or "samplingRate" instead of "x")
- Just because you technically can start your variable names with underscores doesn't mean you should.

Defining Variables

- For multi-word variable names, two options:
  - start capitalizing each word after the first “myCar”
  - separate words with underscores. For instance, a variable for “Ford Focus” could be “my_car”.
- Abbreviating is common for longer words. So, a variable for “average price” could be “avgPrice” or even “avg”.

Variables

- Variables can hold all kinds of values, including strings, different types of numbers, and user input.
- To assign a string value to a variable, you have to wrap the string in quotes (like usual).

```
firstName = "John"
lastName = "Doe"
mathProblem = "5 + 5"
print lastName, ",", firstName, ";",
mathProblem
```

will give the output

```
Doe , John ; 5 + 5
```

Variables

- Variables can also be assigned new values that are relative to their old values. For example:

```
total = 10
print "Original total:", total
total = total + 4
print "New total:", total
```

will give the output

Original total: 10

New total: 14

Variables

- Remember: A variable has to have been defined on a previous line before it can be used on the right-hand side of an equation, so:
- ```
  total = total + 4
  print "Total:", total
  ```
  causes an error, since there was no mention of the value of “total” before the line trying to redefine it.


  ```

Numeric Operators

- Python built-in numeric operators:
- + addition
- - subtraction
- * multiplication
- / division
- ** exponentiation
- % remainder (modulo)

(CS108 Notes, Prof. Aaron Stevens)

Python Arithmetic

- Try writing the following code in your program area and see what it outputs

```
Def main():
    a = 12
    b = 2
    c = 16
    d = 3
    e = 2.5

    print "the value of a is", a
    print (a / b) * 5
    print a + b * d
    print (a + b) * d
    print b ** d
    print c - e
    a = a + b
    print "the value of a is", a
```

Python Arithmetic

- Is this what you got?

  the value of a is 12
  30
  18
  42
  8
  13.5
  the value of a is 14

Exercise Time!

- Write a program that takes in a birthday (dd, mm, yy) and returns:
  - The age
  - Number of days until next birthday

Taking User Input

Sometimes, instead of passing in arguments,
you can ask for them after calling the function.

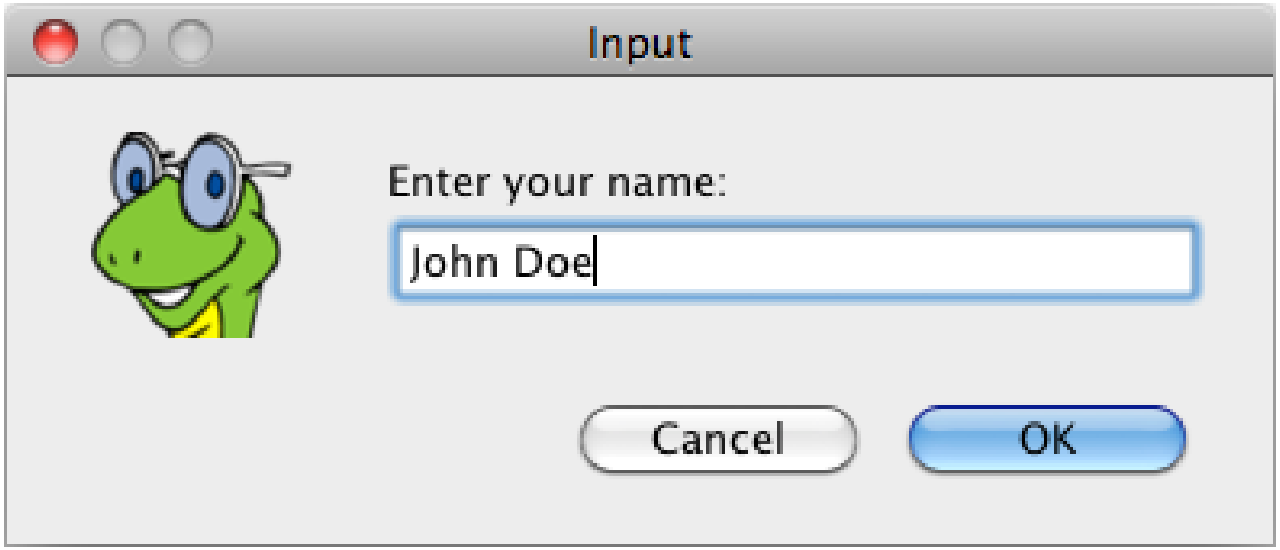
Taking User Input

- *requestNumber()*
- *requestInteger()*
- *requestIntegerInRange()*
- *requestString()*

Taking User Input

```
name = requestString("Enter your name:")
print name
```

first pops up a dialog box (where you can enter a name, say 'John Doe'):



then outputs
John Doe

Ex. Try it with Numbers!

```
num = requestNumber("Enter a number:")
print "Your number:", num
print "Your number squared:", num*num
```

This is where you put the message you want to appear with your input box!

Ex. Try Inputting a String

- Make JES print “<input> is awesome!”

```
name = requestString("Enter your name:")
    print name, “is awesome!”
```

The For Loop

- Also known as the “definite loop”.
- Allows you to specify a list of items (numbers, words, letters, etc.), and specify action(s) to be performed on each one.
- The official form for the for loop is this:

```
for <var> in <sequence>:
    <body>
```

(Note that the body is indented to in the loop)

The Kittens Need Your Help!

You are working at an animals shelter, you are asked to take a group of kittens, bathe, dry, and feed each one individually.

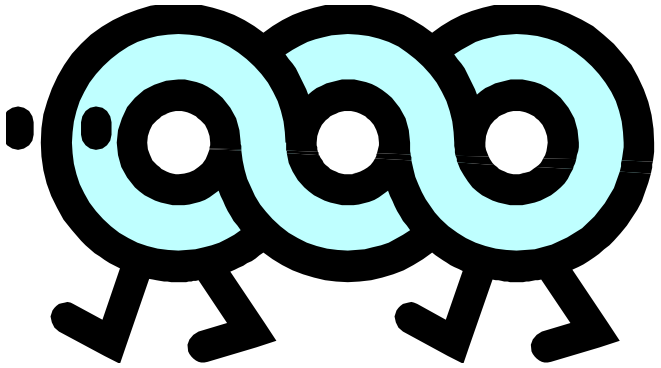
The Kittens Need a Loop!

Using a for-loop type notation, your instructions would look like this:

```
Kittens = [kitty #1, kitty#2,
kitty#3, ...]
for kitty in Kittens:
    bathe kitty
    dry kitty
    feed kitty
```

Ex. For Loop

- ```
  phrase = "Hello world!"
  for letter in phrase:
  print "the next letter is:", letter
  ```
  will give the output
  the next letter is: H
  the next letter is: e
  the next letter is: l
  the next letter is: l
  the next letter is: o
  the next letter is:
  the next letter is: w
  the next letter is: o
  the next letter is: r
  the next letter is: l
  the next letter is: d
  the next letter is: !


  ```

What Just Happened?

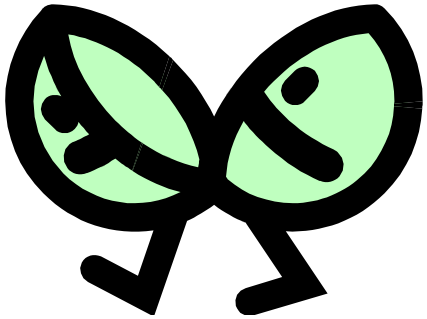
- What Python did was that it went through the line one character at a time, treating the line like a sequence.
- That means that the line can be split into its components (the characters).

split()

Want to work with units of a phrase that aren't characters?

Put something in the <sequence> position that isn't just a string.

The result is a list of all the items in the phrase that are separated by spaces.



split()

```
phrase = "Hello beautiful world!"
for word in phrase.split():
    print "the next word is:", word
```

will give the output

the next word is: Hello

the next word is: beautiful

the next word is: world!

split()

In fact, if you printed phrase.split() to see what it looked like, you'd get ['Hello', 'beautiful', 'world!'], a list containing each "word"

(You will learn about lists tomorrow ☺)

Accumulator Variables

When you're using a for loop, sometimes you want to keep a running total of numbers you're calculating, or re-combine bits of a string.

Accumulator Variables

Steps:

1. Define it for the first time before the for loop starts.

2. Redefine it as itself plus some operation in the body of the for loop.

```
total = 0
  for num in [1,2,4,10,20]:
    total = total + num
  print "Total:", total
```

will give the output

Total: 37

Accumulator Variables

What is the point of accumulator variables?

-Counting.

-keeping score (affects program does).

-debugging.

Conditional Statements

Equals: ==

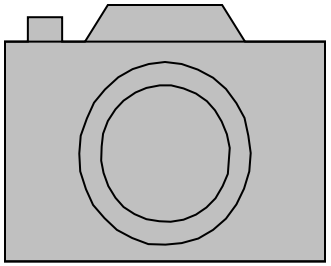
Does not equal: !=

Try this:

```
x = 1
If( x != 2)
    print “Artemis rocks”
```

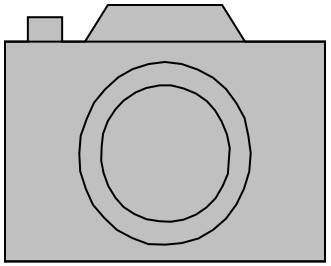
Pictures

File Functions



- pictures live within files.
- You must get your program to go find and read a file that's somewhere else on your computer.

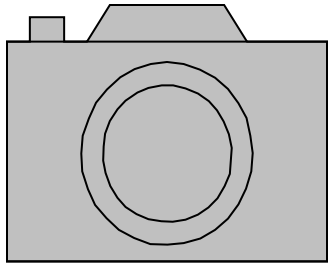
File Functions



Steps:

1. Store the “address” for the file you want as a variable
2. Use functions to read, display, or modify the file at that location.

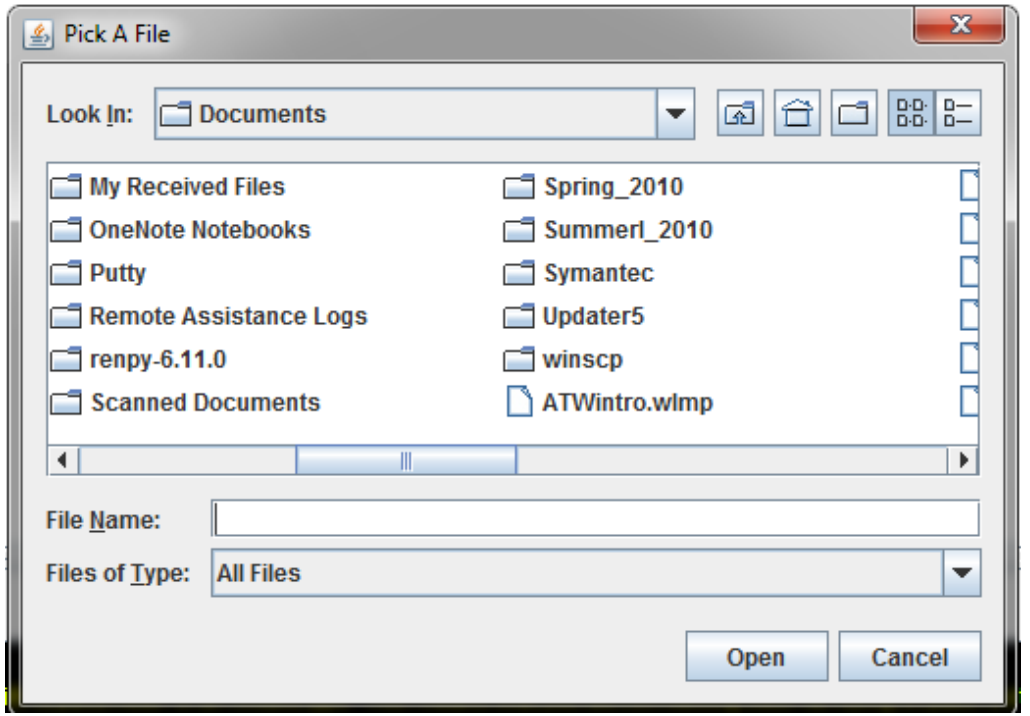
File Functions



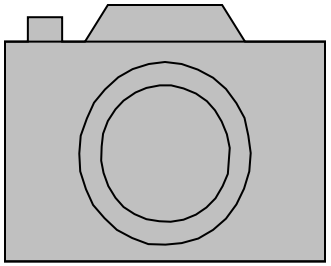
```
myFile = pickAFile()
```

This function notes the 'path name' (as a string) of your file, i.e. the "address" of that file on your computer.

It brings up the 'file selector dialog'.



File Functions



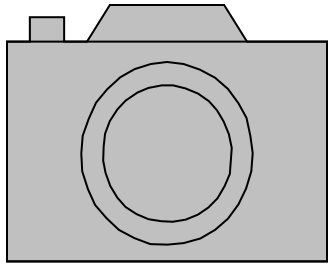
If it is a picture, and you wish to treat it as such:

```
myPic = makePicture(myFile)
```

These functions will not make anything appear.

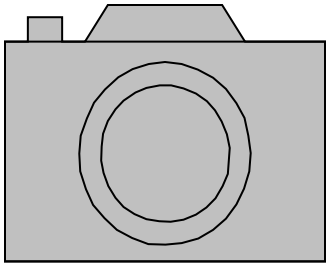
So far, things are just stored in the computer's memory, invisible to the user.

Pictures



After modifying your sound/picture you may want to save it as a new file, since ‘repainting’ a picture or ‘playing’ a sound will simply show you your work, not save it anywhere.

Pictures



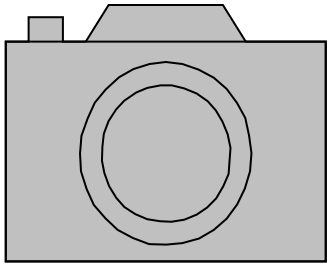
How?

Use *writePictureTo* after you specify the path.

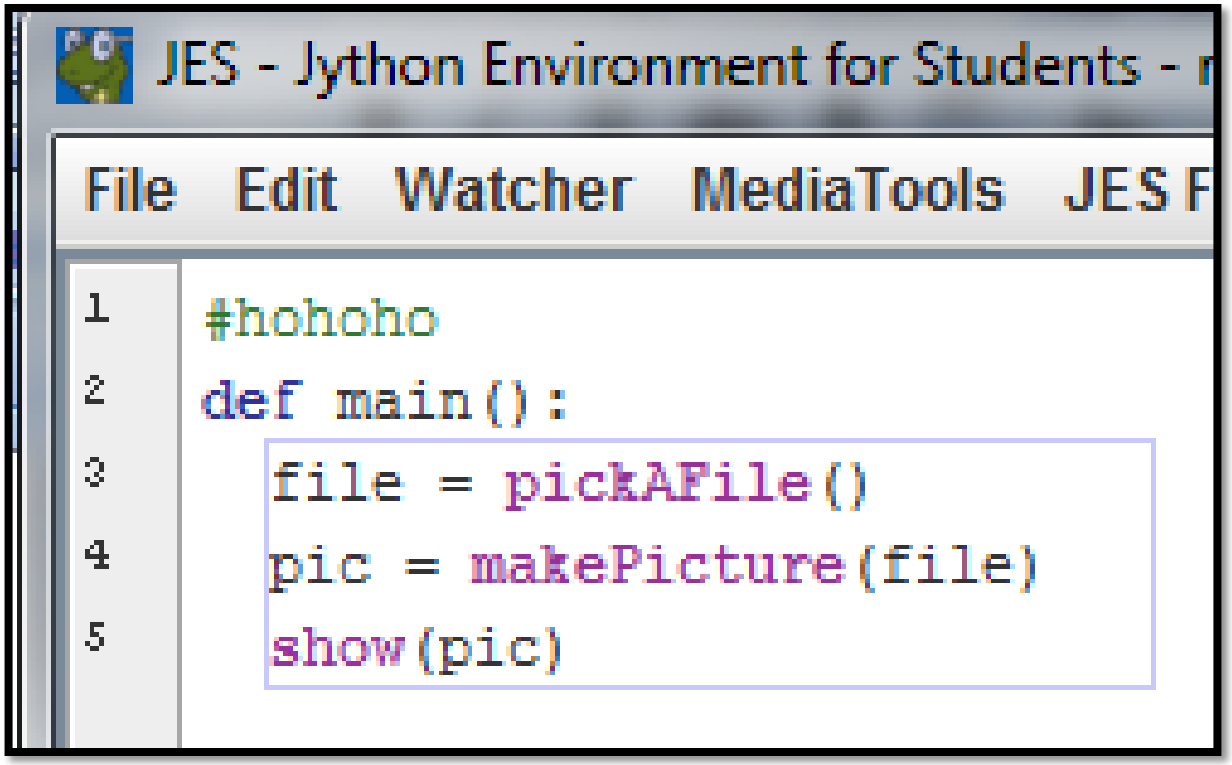
Try this in the command area

```
path ="/Users/Durrah/Documents/stuff/editedpic.jpg"
   writePictureTo(pic,path)
```

Pictures

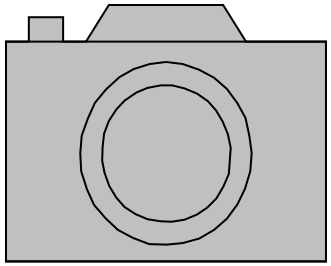


Try writing this simple function



This shows a window containing the picture.

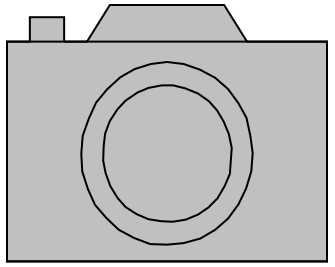
Pictures



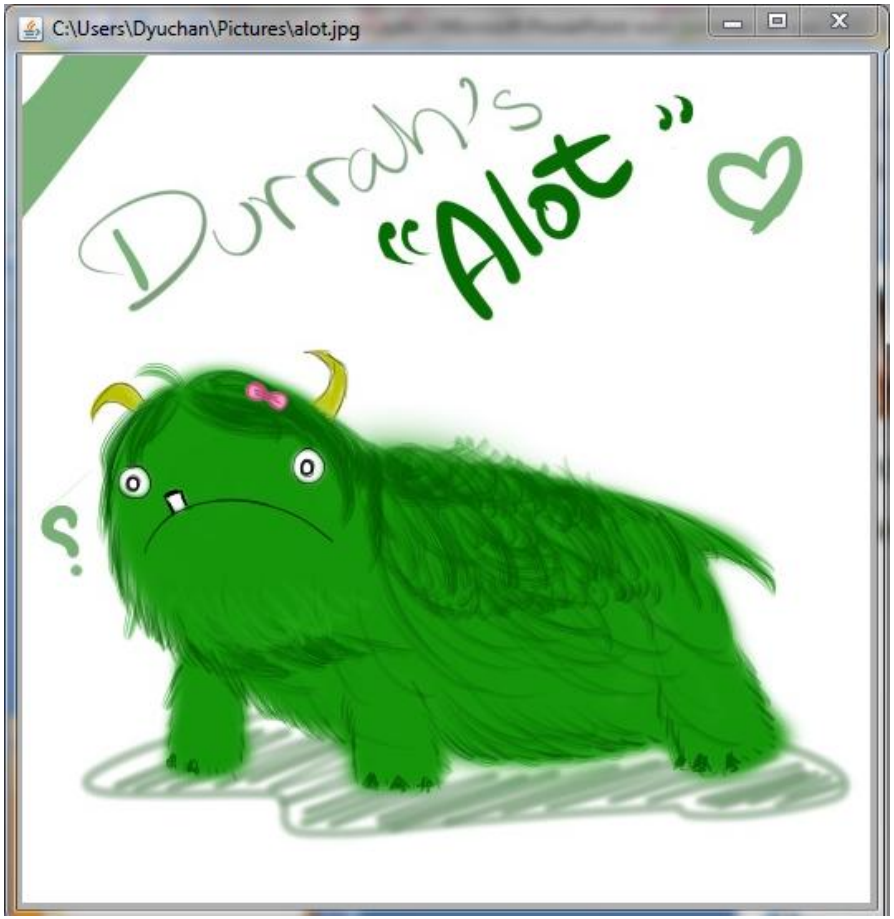
- Find a small picture (use Google)
- Save it in Documents.
- Enter the following

```
>>>
>>> file = pickAFile()
>>> pic = makePicture(file)
>>> show(pic)
>>>
```

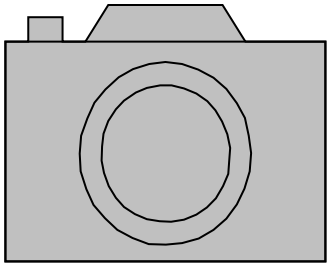
Pictures



Your picture should appear

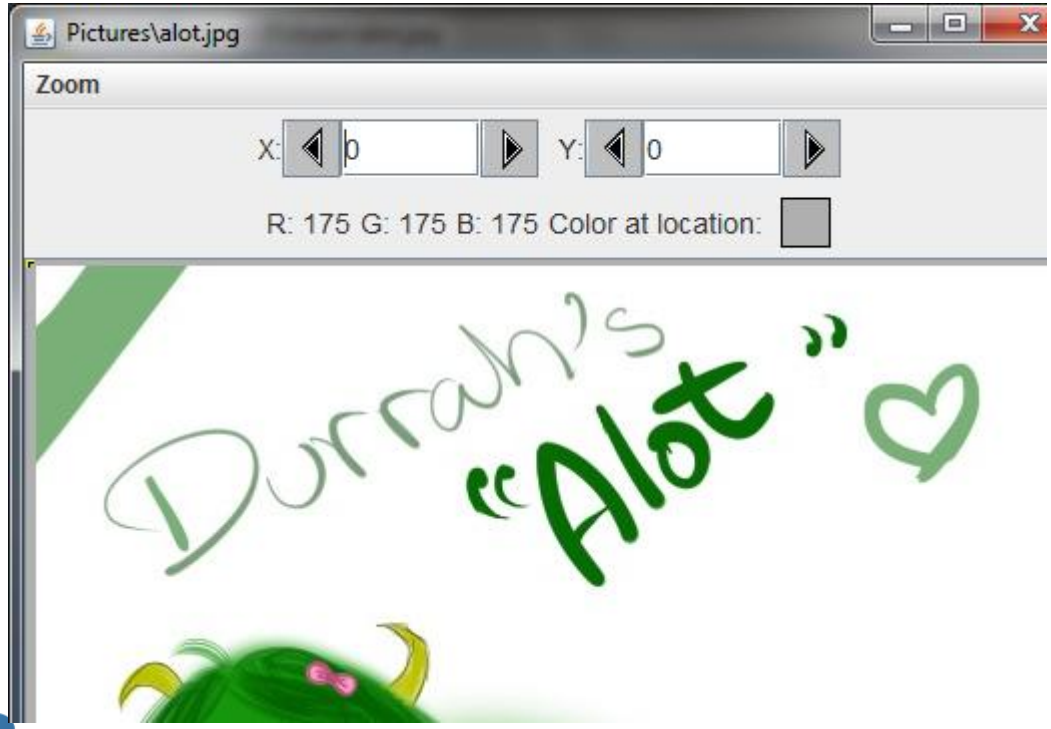


Pictures



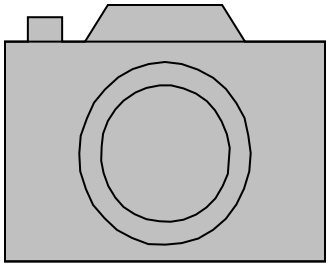
Now enter explore(pic)

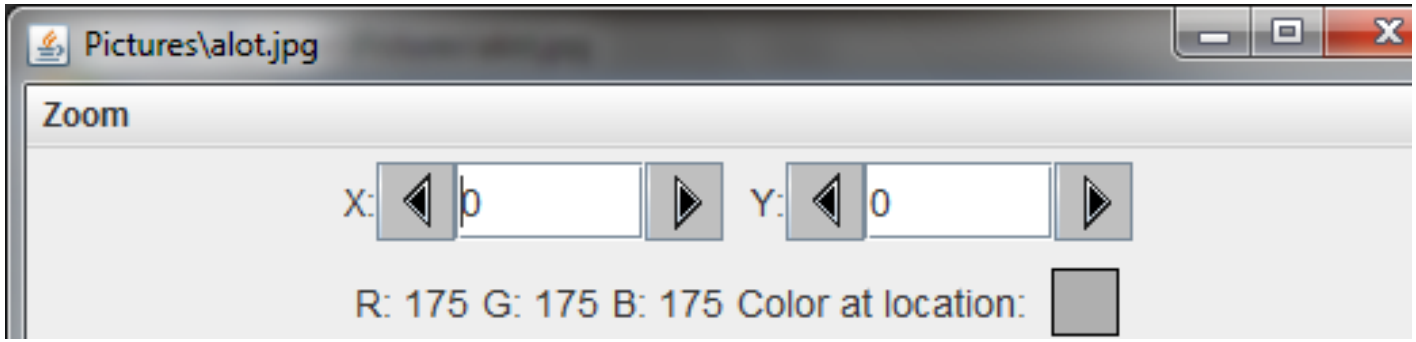
```
>>> show(pic)
>>> explore(pic)
```



Click anywhere in the picture that just popped up and you'll see the X & Y coordinates of the pixel.

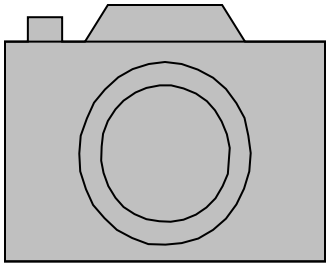
Pictures





This function is useful when you are trying to locate a certain pixel that you want to play with. It gives you the Red, Green and Blue values of that pixel. This is the stuff of which Photoshop is made ☺!

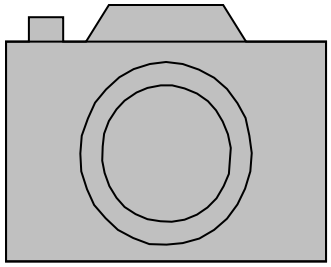
Pixel Functions



So how do you make changes to your picture?

- Target one pixel
- Target all pixel

Getting a Pixel

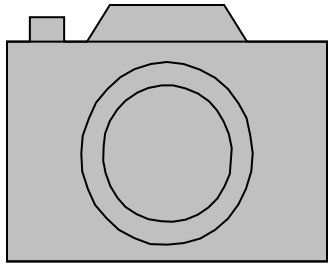


`getPixel` takes three parameters, the picture to take the pixel info from, the x-value of the desired pixel, and the y-value of the pixel, in the form `getPixel(pic, x, y)`

Add this to your command area

```
>>> show(pic)
>>> explore(pic)
>>> aPixel = getPixel(pic, 5, 10)
>>>
```

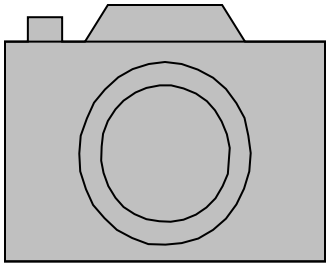
Pixel Functions



`aPixel = getPixel(pic, x, y)` stores the color information of the pixel located at (5, 10) in the picture `pic` in the variable `aPixel`.

But what if we want ALL the pixels?

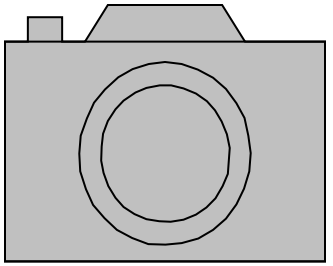
Get Those Pixels!



`getPixels(pic)`

```
>>> show(pic)
>>> explore(pic)
>>> aPixel = getPixel(pic, 5, 10)
>>> allPixels = getPixels(pic)
```

PIXELIZATION!

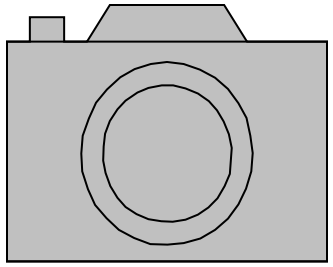


Now we want to change every single pixel… How will we ever do that?

Why of course!

The LOOP to rescue!

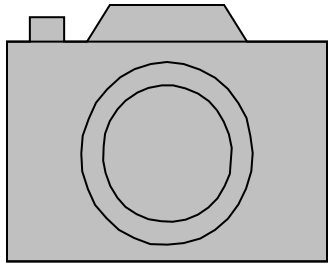
There are New Pixels in Town



Don’t forget to indent!

```
>>> allPixels = getPixels(pic)
>>> for px in getPixels(pic):
...         setRed(px, 0)
...
```

Pictures



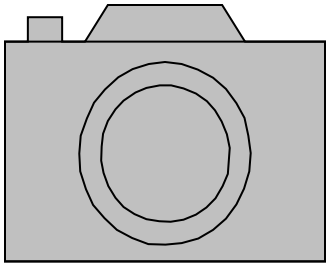
Your picture has not changed yet? That is because you haven't applied the changes to it!

Enter:

```
repaint(pic)
```

```
...         setRed(px, 0)
...
>>> repaint(pic)
>>>
```

THE CHALLENGE

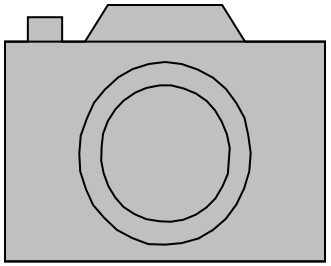


Can you write a program (in the program area) that changes the color of the picture the way we did?

You have all the functions in your command area.

I will give you the pseudocode and you have to code it however you see fit.

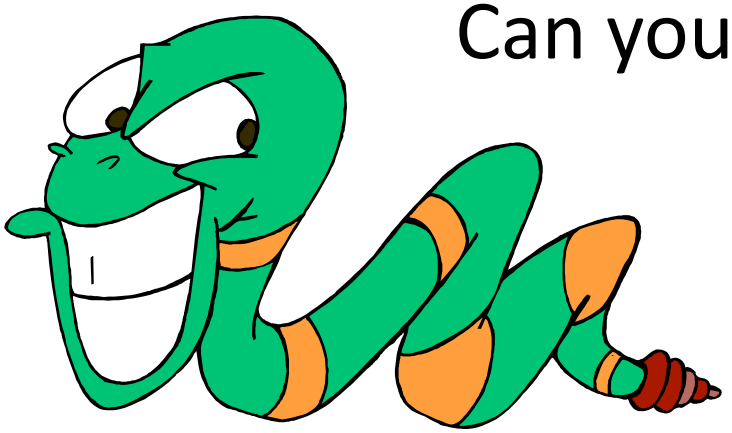
THE CHALLENGE



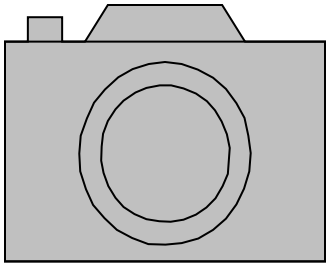
Remember to start with def `main():` and to indent everything to be inside it!

The program should have 9 lines of code. I will show one line of code every TWO minutes

Can you beat the PowerPoint?



Pictures



Get the file

Make it a picture

Display it

Explore it

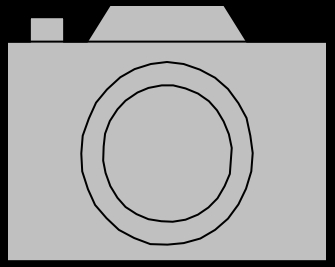
Get one pixel

Change the color of the picture

Apply the changes

```
1  #hohoho
2
3
4
5
6
7
8
9
10
11
```

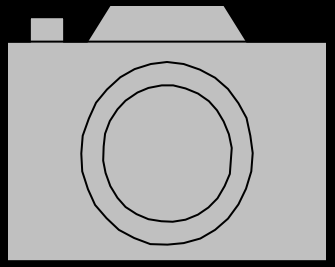
Negative



When creating a negative, we want the opposite of each of the current values for red, green, and blue. If we have a red component of 0, we want 255 instead, If we have 255, we want 0.

In other words, we are getting |255-current| for every pixel in the picture.

Negative

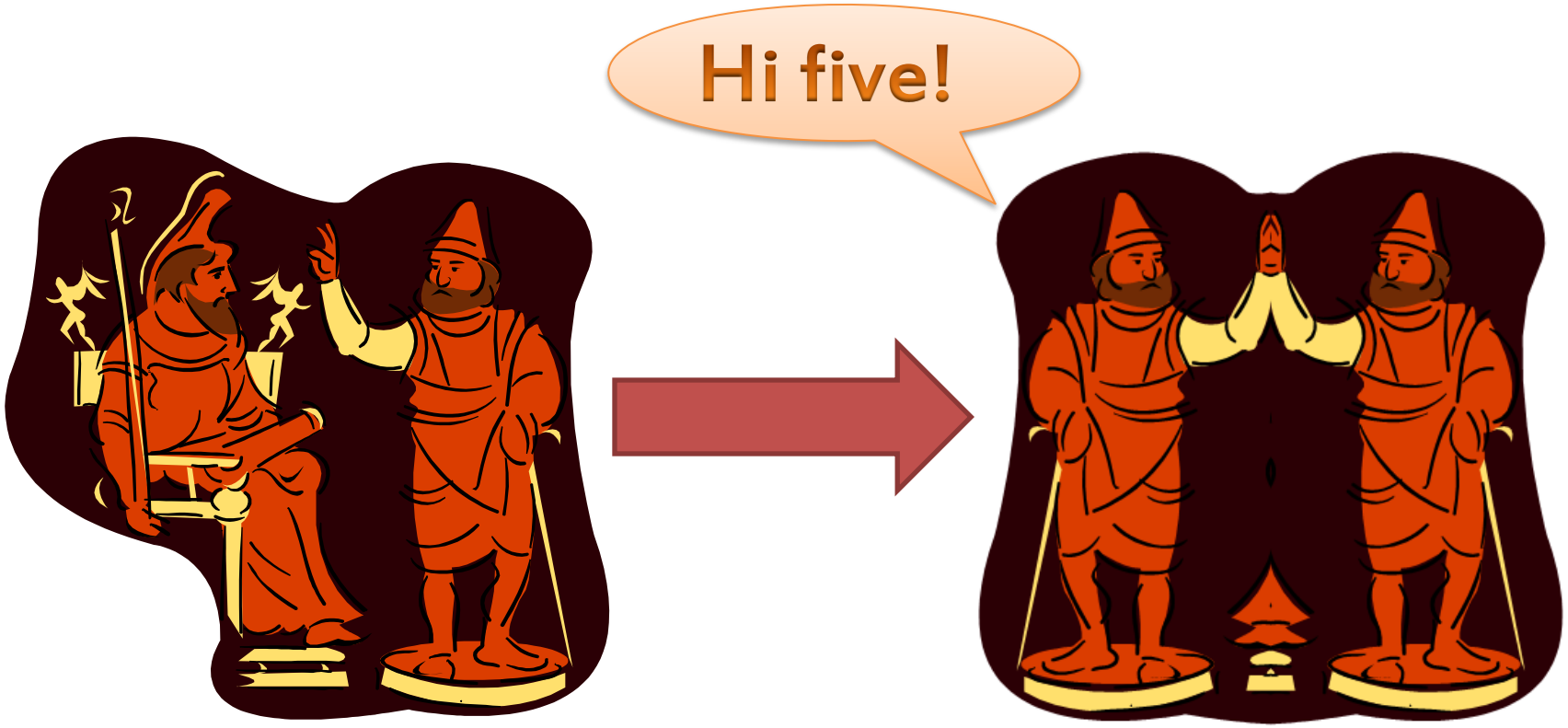


```
Def negative():
   file = pickAFile()
   pic = makePicture(file)
   for px in getPixels(pic):
       red = getRed(px)
       green = getGreen(px)
       Blue = getBlue(px)
       negColor = makeColor(255-red, 255-green, 255-blue)
       setColor(px, negColor)
   repaint(pic)
   show(pic)
```


C:\Users\Dyuchan\Pictures\alot.jpg
Durrah's
"Alot"
C:\Users\Dyuchan\Desktop\alot.jpg
Durrah's
"Alot"

Mirror, Mirror on the Wall

- We can use Python to manipulate more than the colors of the picture. We can do this:



Code, Code on the Screen

```
#Starting with pseudocode
mirrorVertical():
    Get a picture
    Identify its middle.
    in every row
        replace each column with the at the same
        distance from the middle until you reach the
        middle
    apply changes
    show the picture
```

Code, Code on the Screen

```
def mirrorVertical():
  file = pickAFile()
  pic = makePicture(file)
  mirrorpoint = getWidth(pic)/2
  for y in range(1, getHeight(pic)):
    for xOffset in range(1, mirrorpoint):
      pright = getPixel(pic, mirrorpoint+xOffset, y)
      pleft = getPixel(pic, mirrorpoint-xOffset,y)
      c = getColor(pleft)
      setColor(pright, c)
  repaint(pic)
  show(pic)
```

Extra Challenge

- As you can see, the right side mirrored the left side (and that's why the creature has two heads).
- Can you change your code

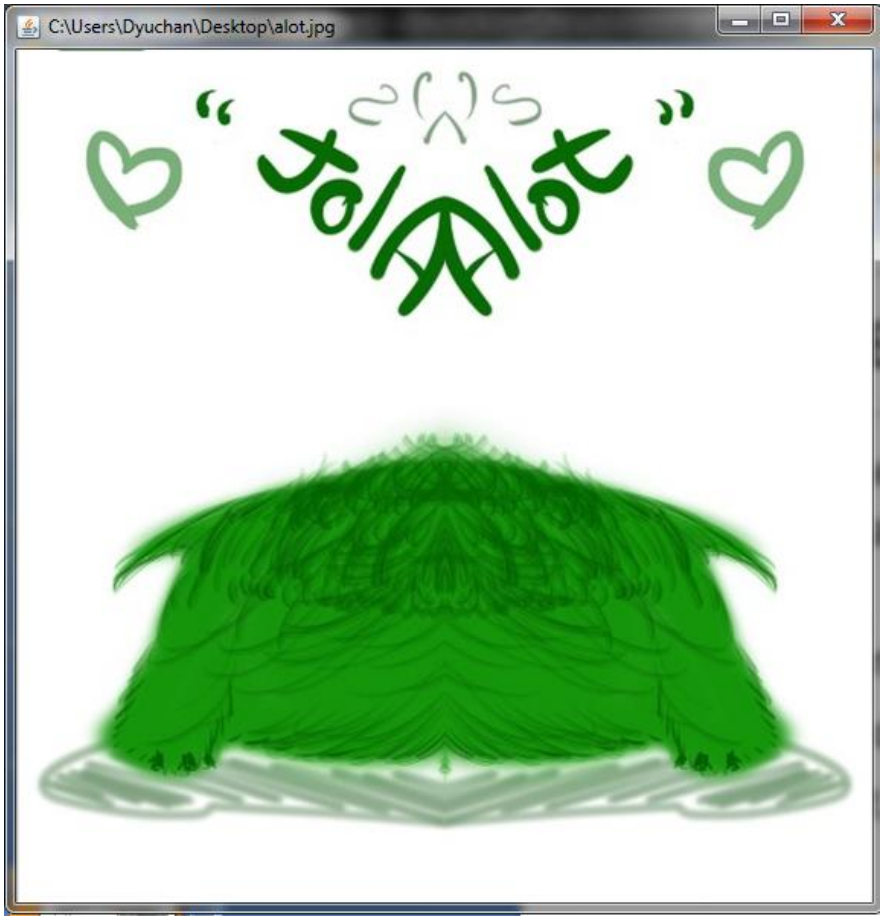
So it does the opposite?

(i.e. let the left side mirror

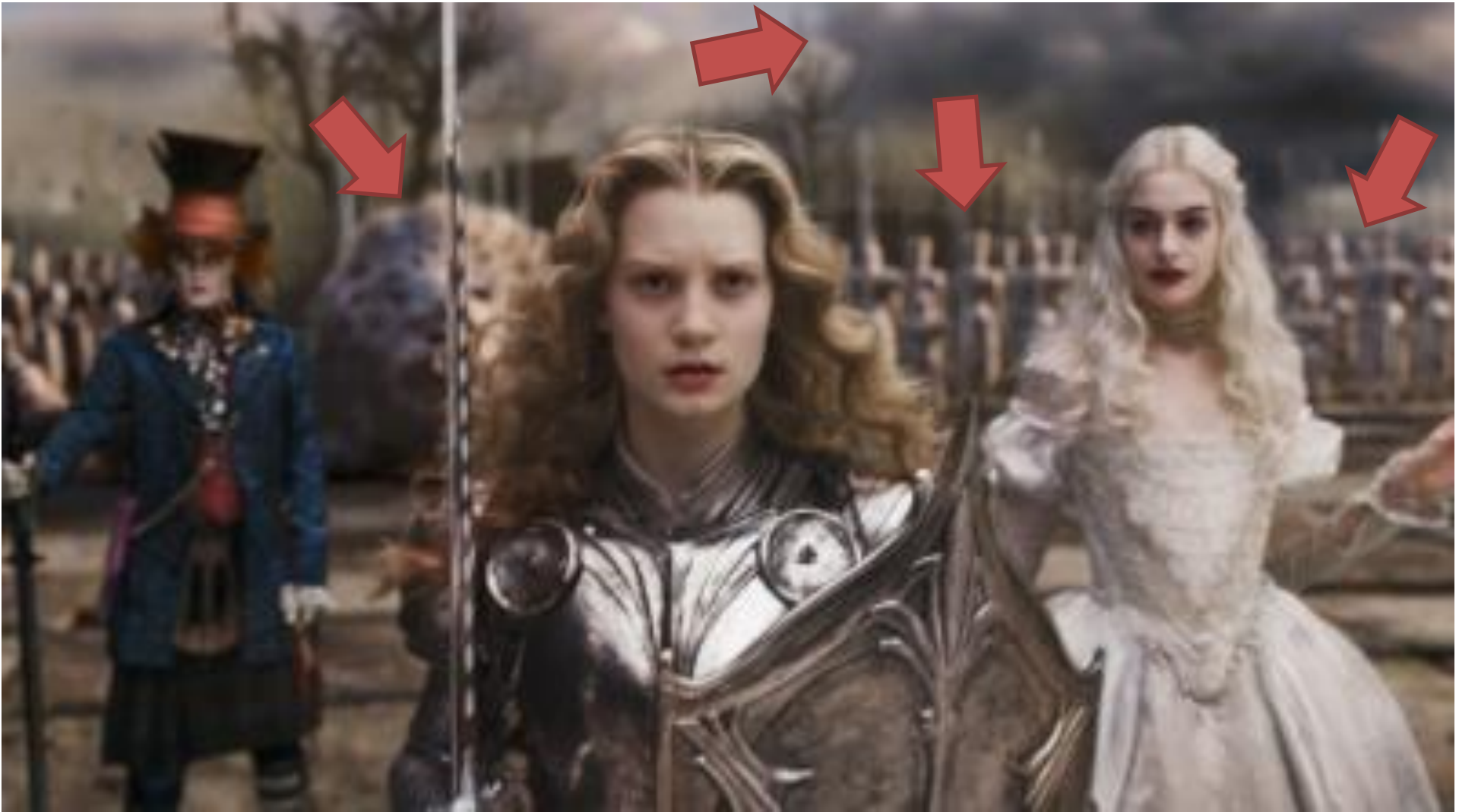
the right side)

Reversed Mirroring

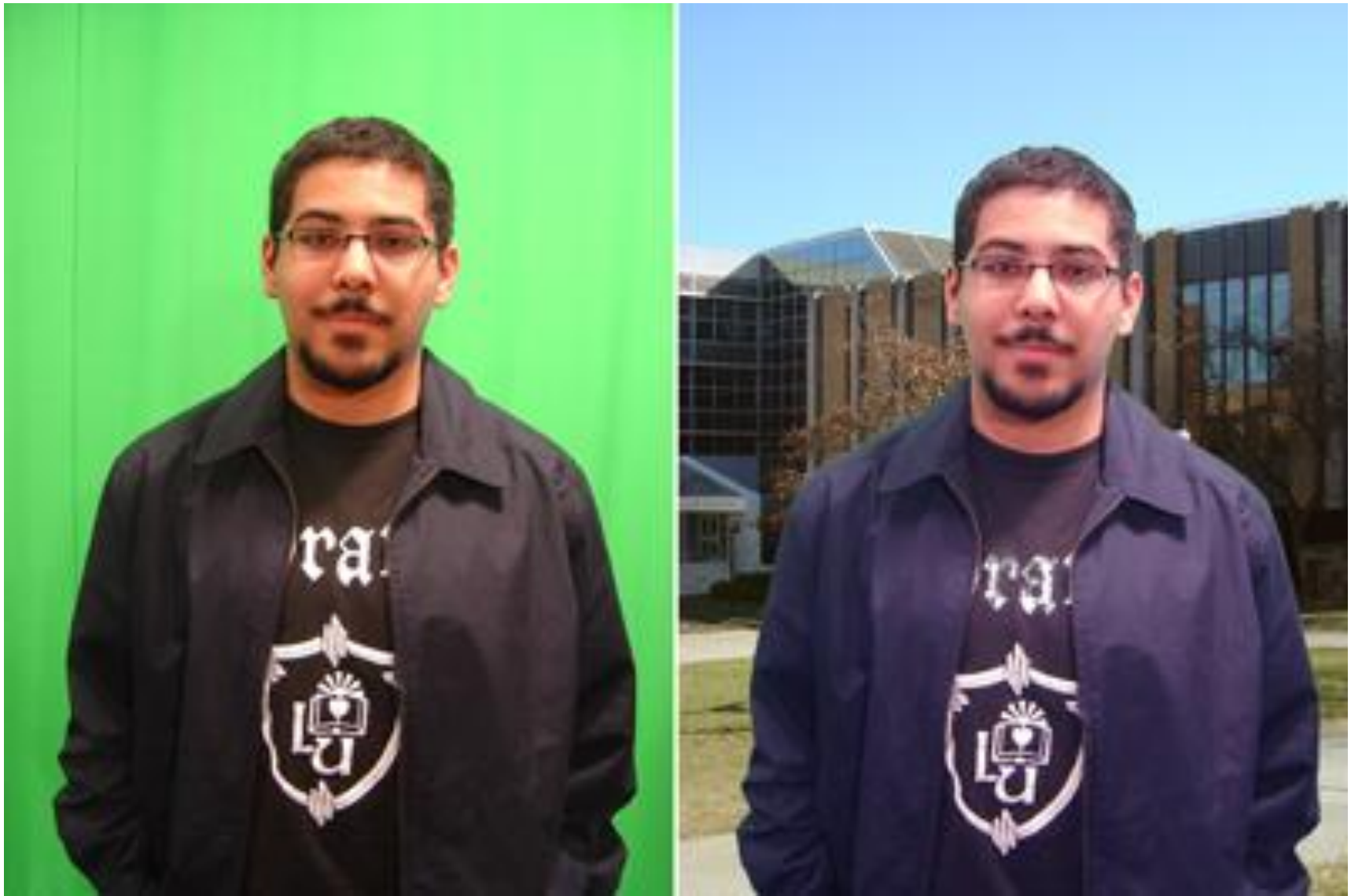
- This is the result I got:



Are the soldiers, clouds, big creature actually there?



Greet the Green☺



The Green Screen

- Placing an object (foreground) in a background of our choice.

The Green Screen

- Tell the computer look at each pixel, and see if its red and blue values are less than its green value.
- If (red+blue < green) then that pixel is likely to belong to the green screen. Now tell the computer to get the pixel at the same location from the background and paint it on the green screen.

The Green Screen

So in a way, you are actually placing the background on the foreground, not the other way around.

Also, it could be a blue screen. (How will the code change?)

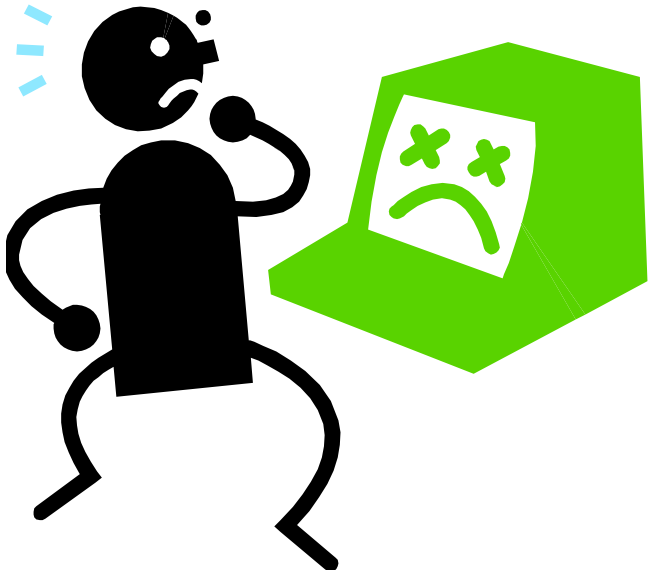
The Green Screen Code

- How do you think the code should look like?

```
def greenScreen(source, bg):
  for p in getPixels(source):
    if(getRed(p)+ getBlue(p) < getGreen(p)):
      setColor(p, getColor(getPixel(bg, getX(p), getY(p))))
  repaint (source)
```

The Green Screen Effect

You do not get perfect results all the time…



Our Journey Ends Here

But yours doesn’t need to!

Want to learn more?

Go to:

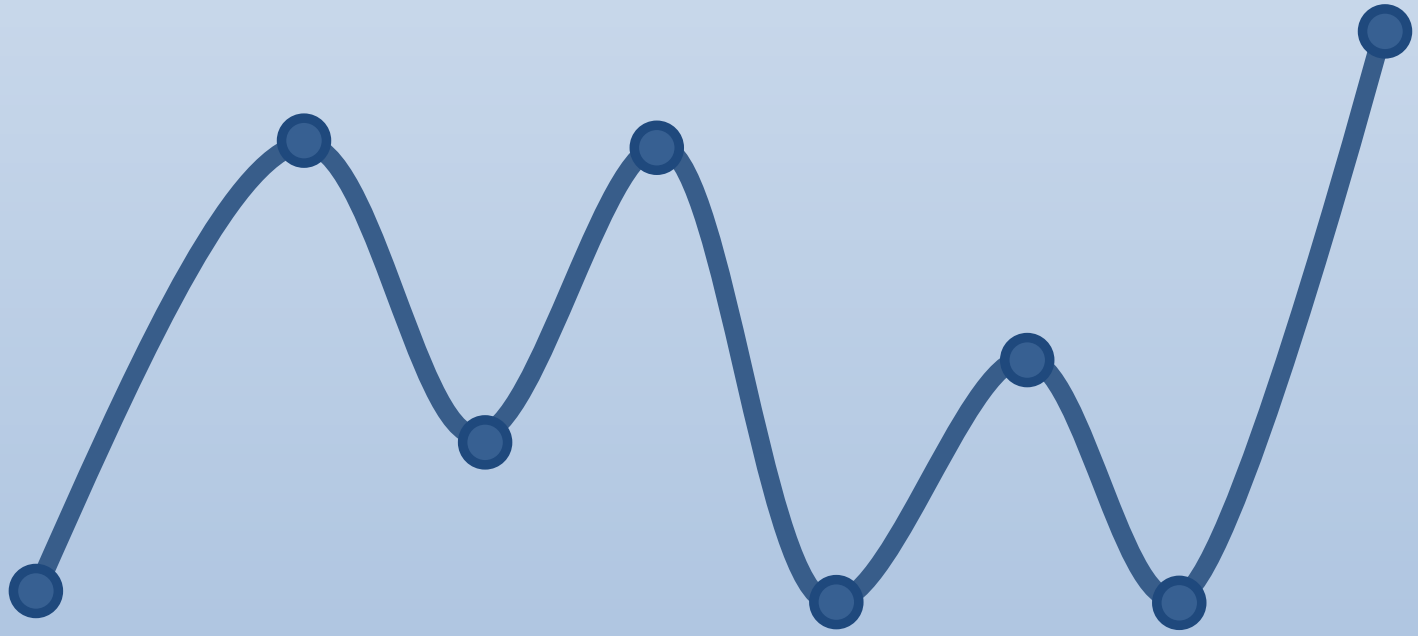
wiki.**python**.org/moin/BeginnersGuide

Questions?

This presentations was based on the CS101 Guide to Python and JES (http://www.cs.bu.edu/courses/cs101b1/jes)

Python Programming

Hans-Petter Halvorsen



https://www.halvorsen.blog

Python Programming

Python Programming

Hans-Petter Halvorsen

2019

Python Programming

August 12, 2020

ISBN:978-82-691106-4-7

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/

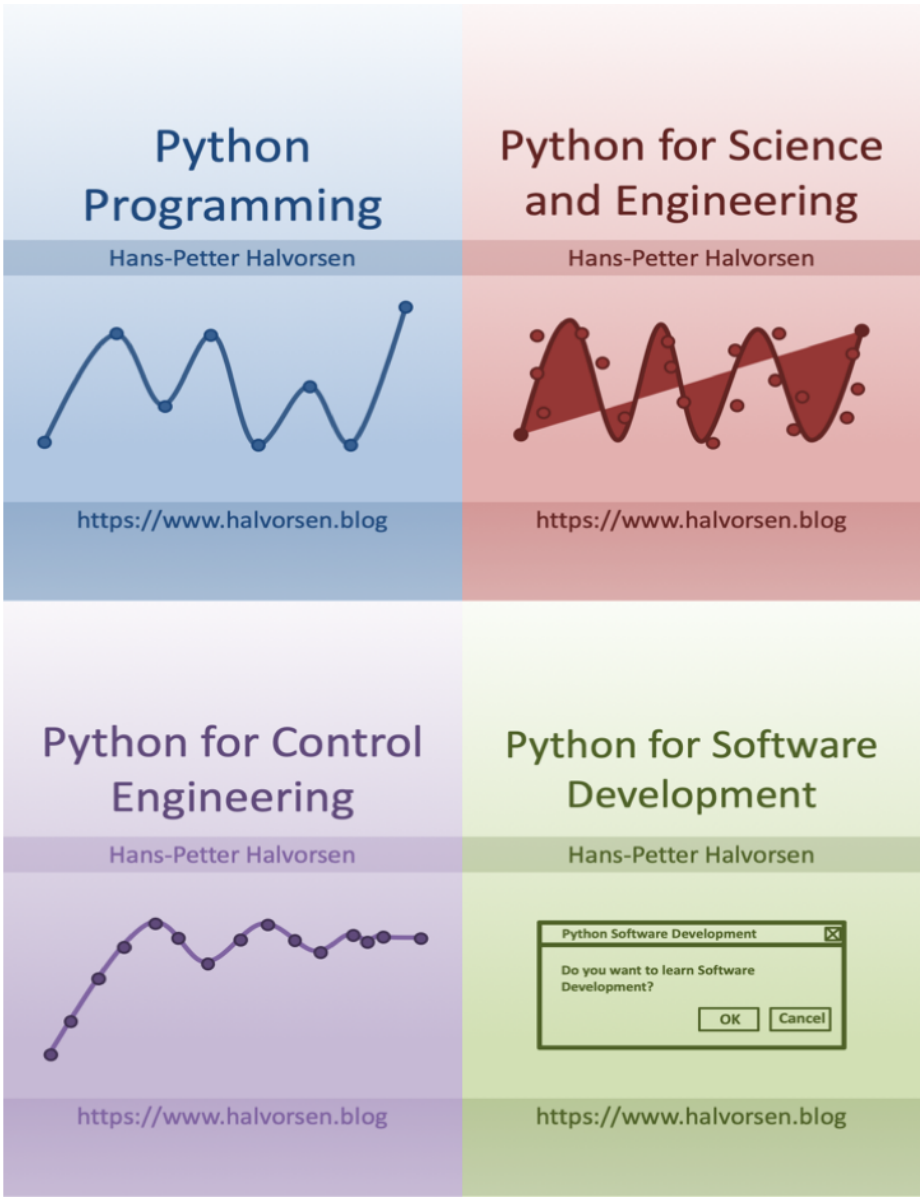
Software Engineering

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].

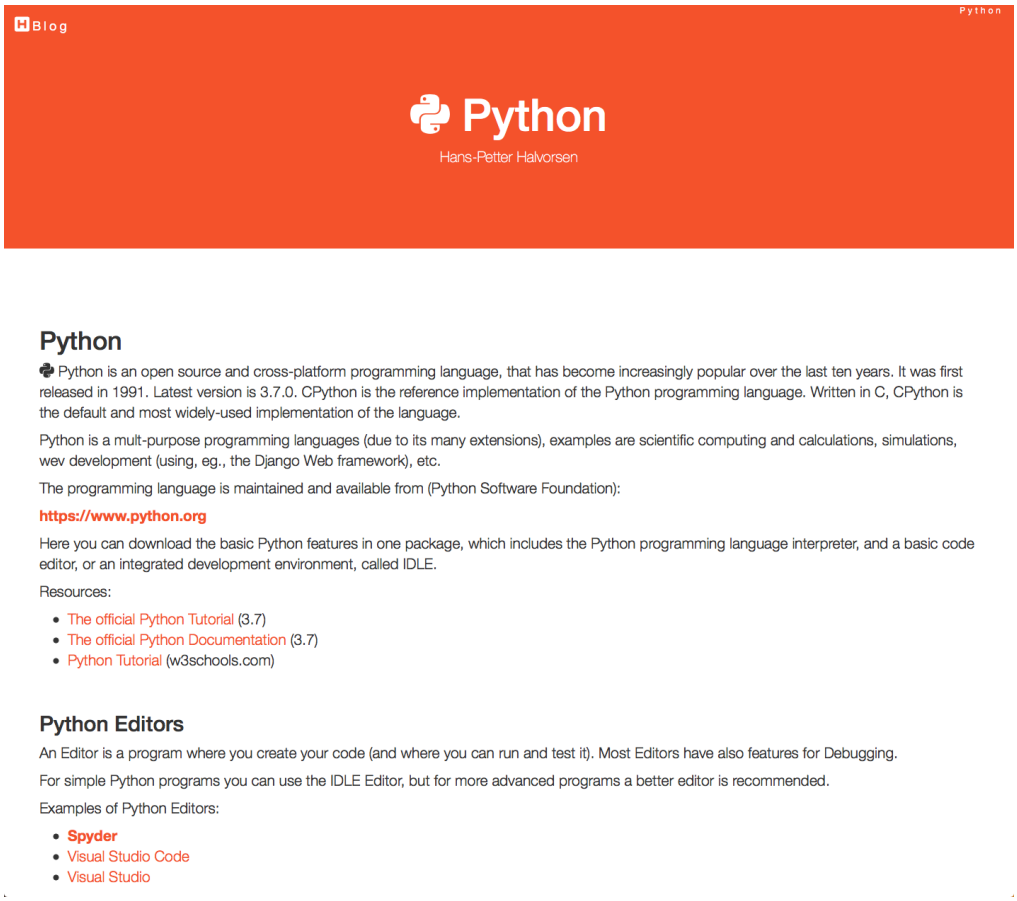


Figure 1.1: Web Site - Python

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

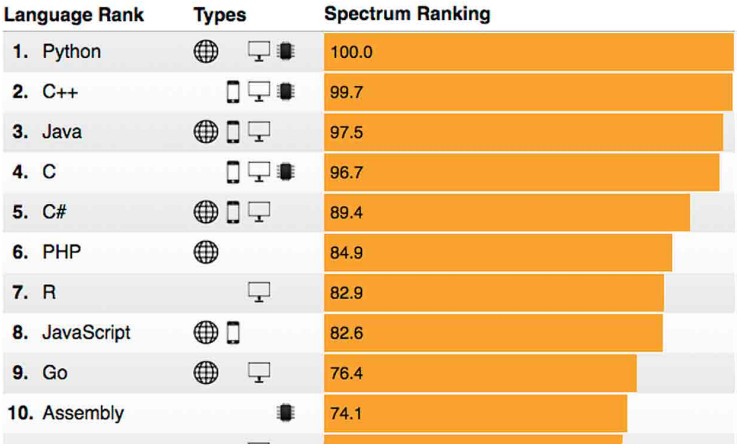


Figure 1.2: The Most Popular Programming Languages

- 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.

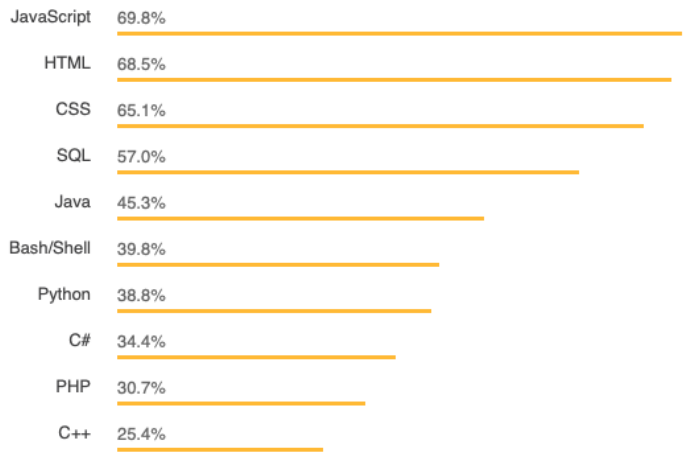


Figure 1.3: The Top Programming Languages - Stack Overflow Survey

- 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)

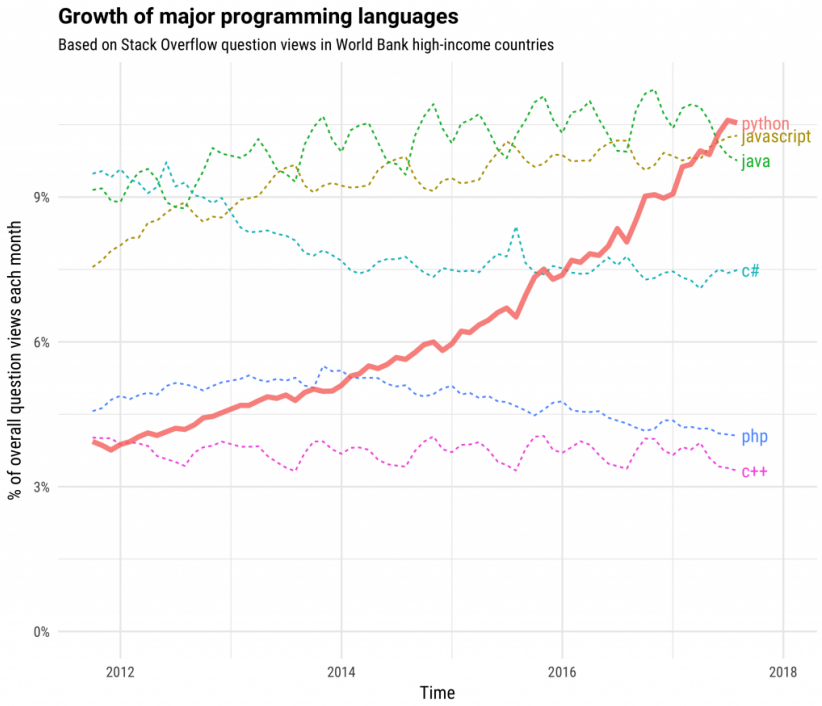


Figure 1.4: The Incredible Growth of Python

- 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/

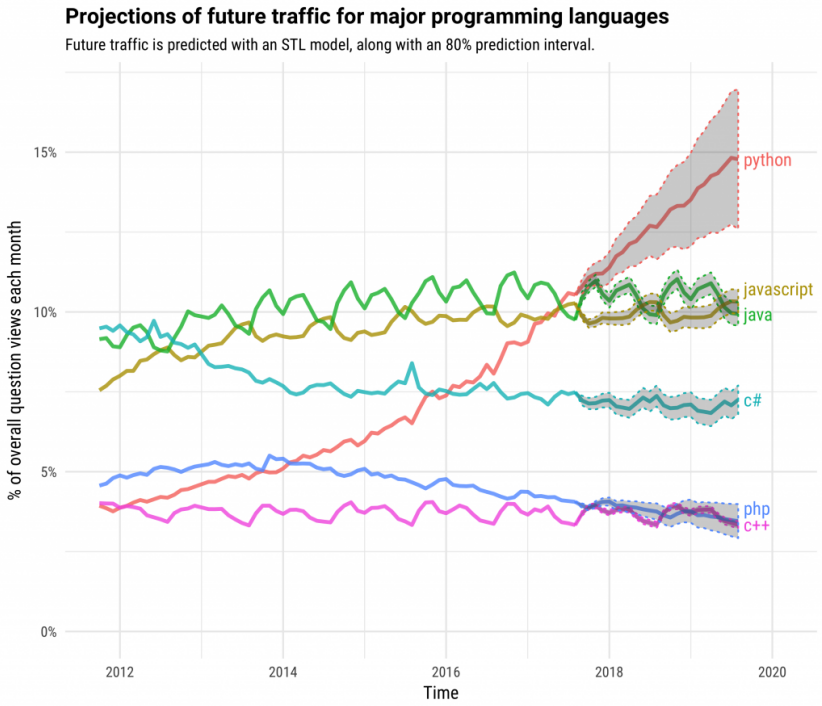


Figure 1.5: The Future of Python

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.

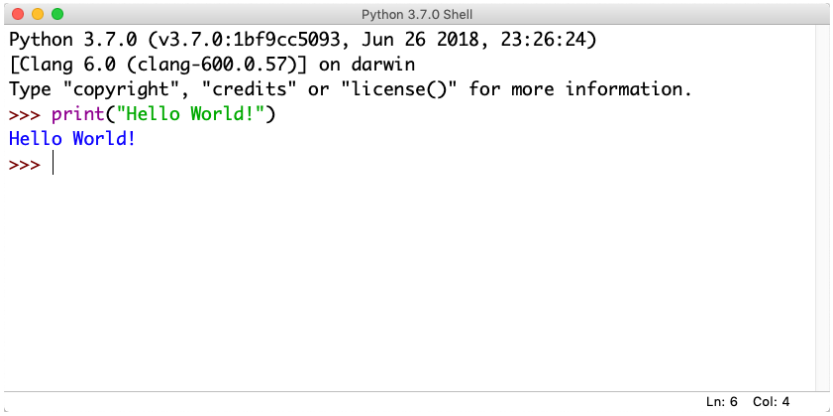


Figure 3.1: Python Shell / Python IDLE Editor

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.

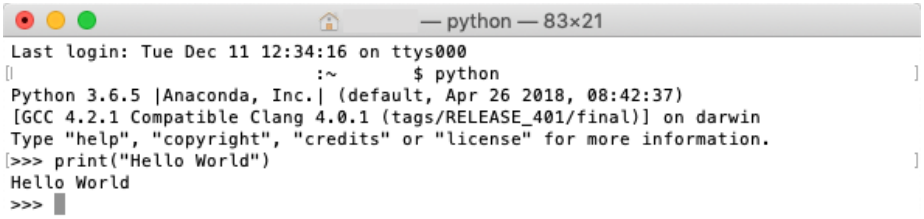


Figure 3.2: Console macOS

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.

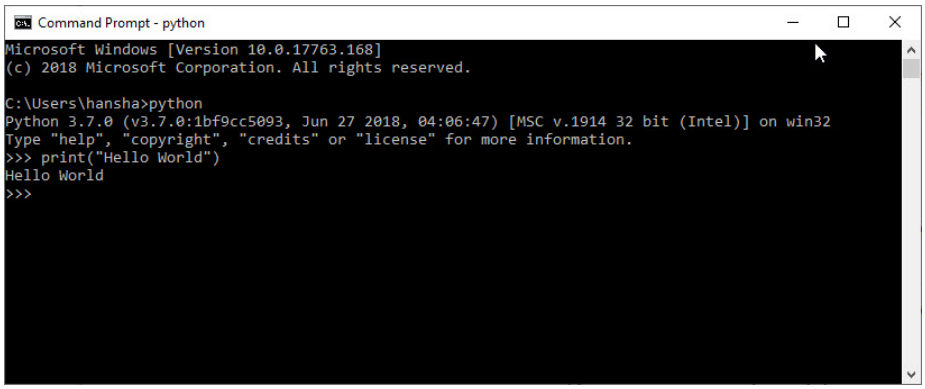


Figure 3.3: Command Prompt Windows

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.

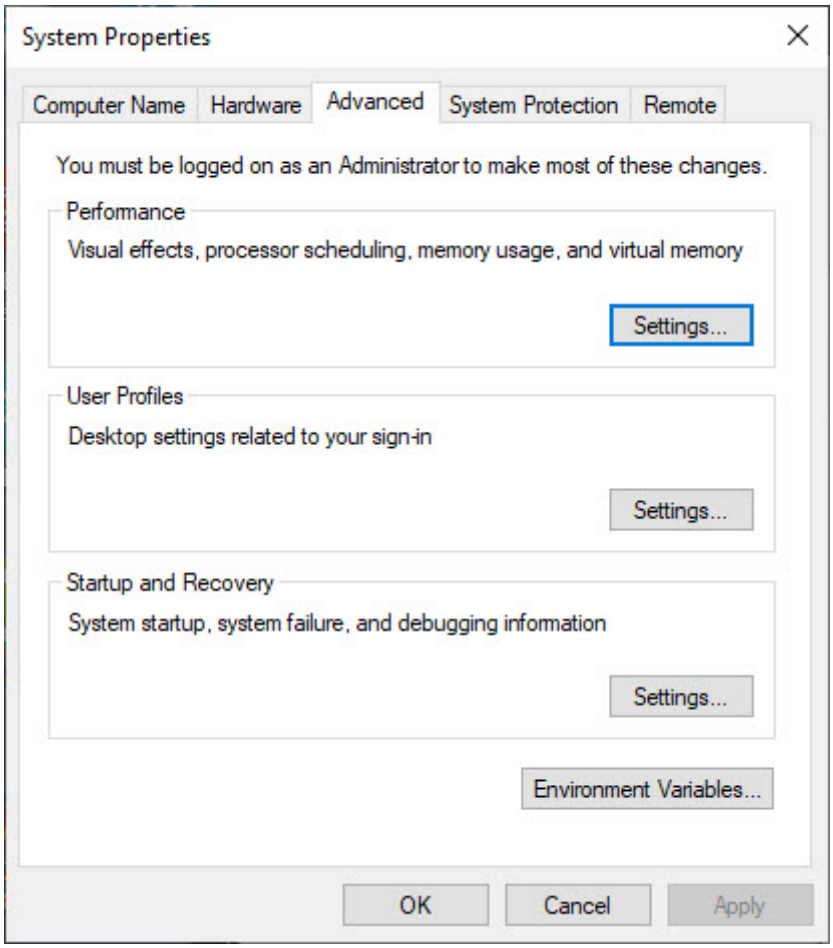


Figure 3.4: Windows System Properties

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.

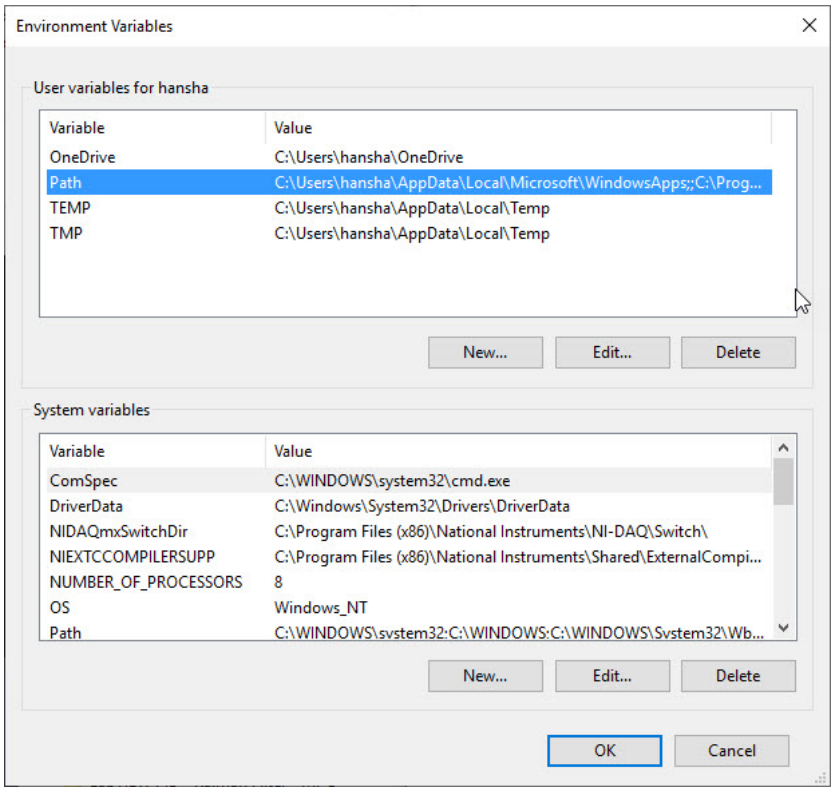


Figure 3.5: Windows System Properties

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.

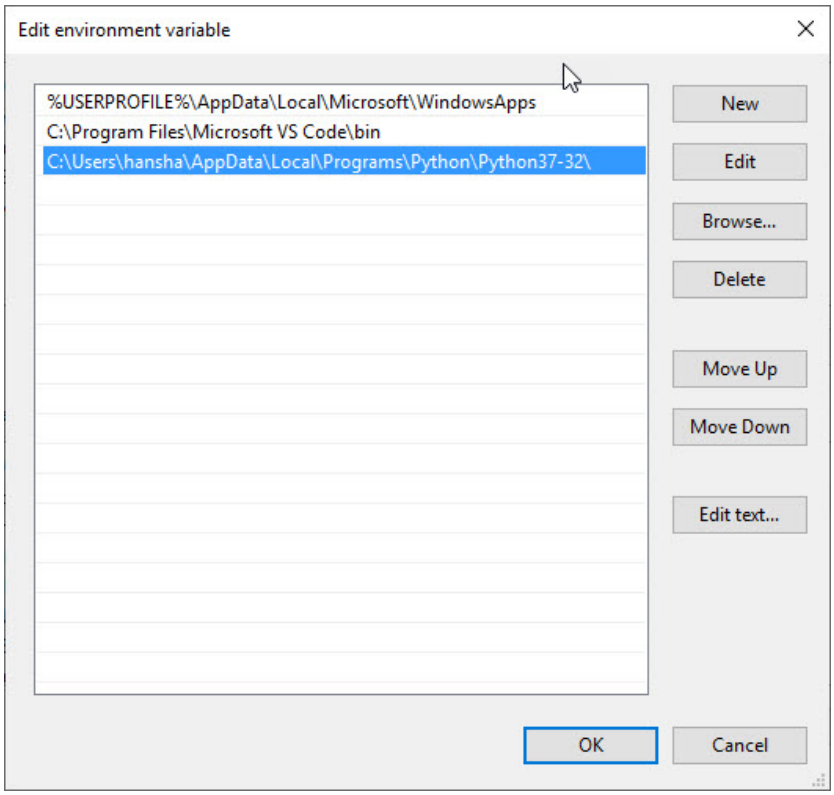


Figure 3.6: Windows System Properties

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.

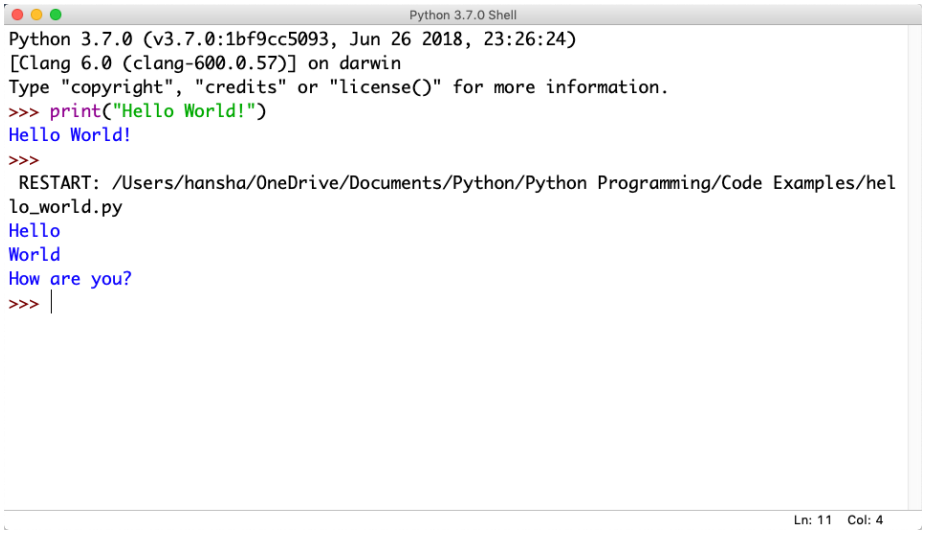


Figure 3.8: Running a Python Script

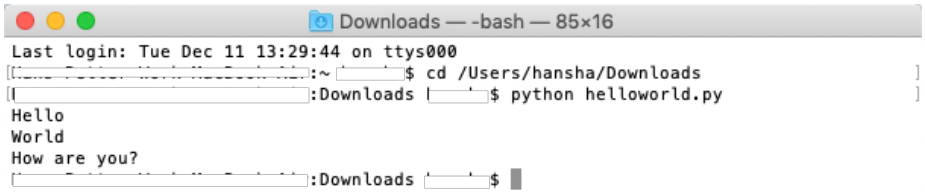


Figure 3.9: Running Python Scripts from Console window on macOS

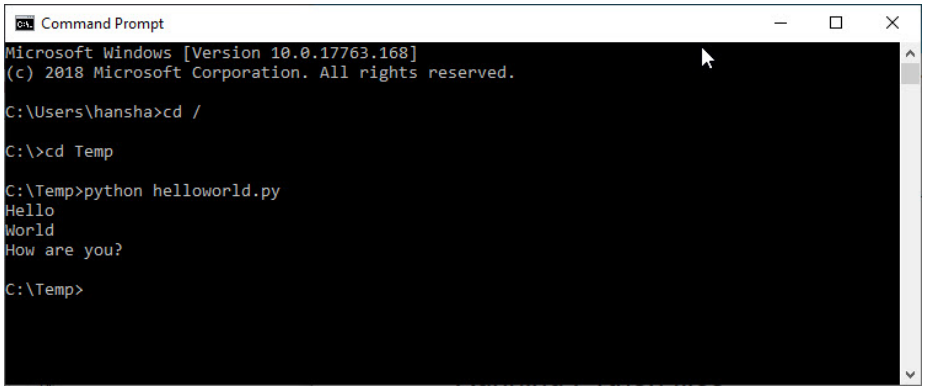


Figure 3.10: Running Python Scripts from Console window on macOS

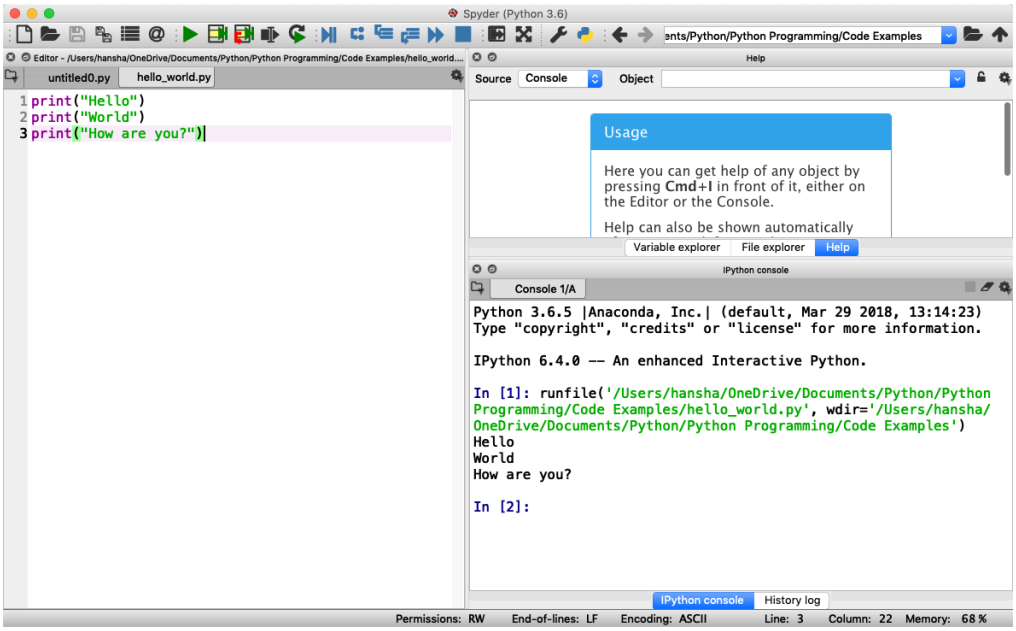


Figure 3.11: Running a Python Script in Spyder

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.
Python

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 implementing 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.

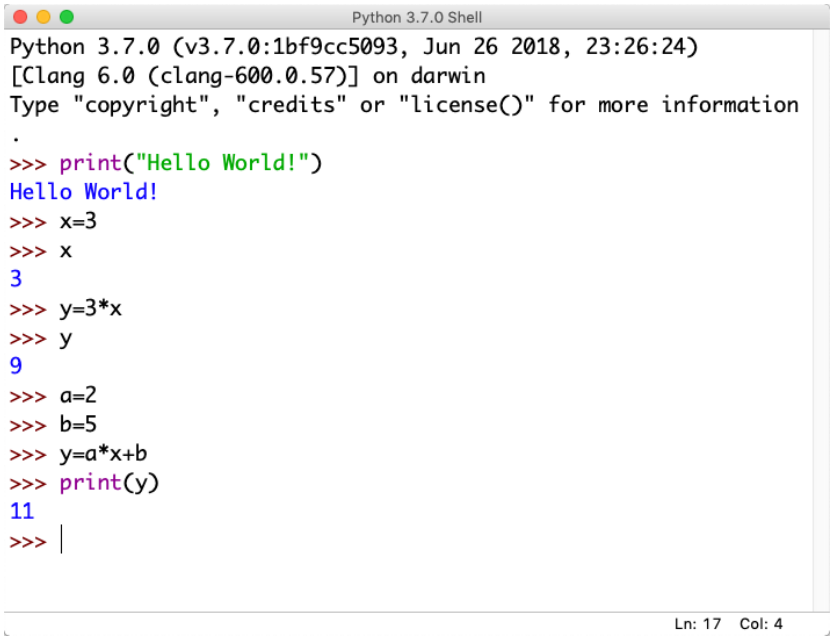


Figure 4.1: Basic Python

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):

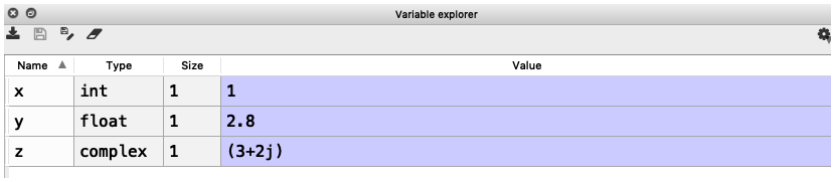


Figure 4.2: Variable Editor in Spyder

[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. Plotting 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. Plotting 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(9 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. 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. An 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:
http://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 to arrays with data. We want to plot x vs. y. We can assume x is a time series and y is the corresponding temperature i 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 following plot:
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.

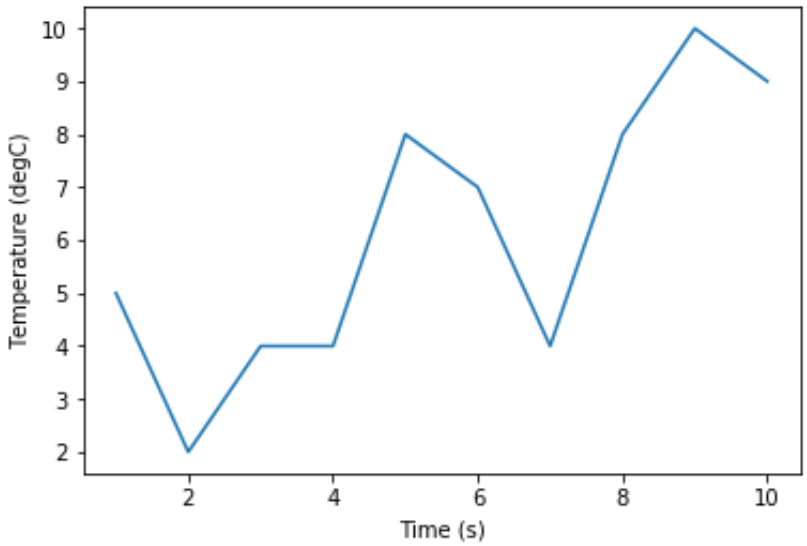


Figure 4.3: Plotting in Python

[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:

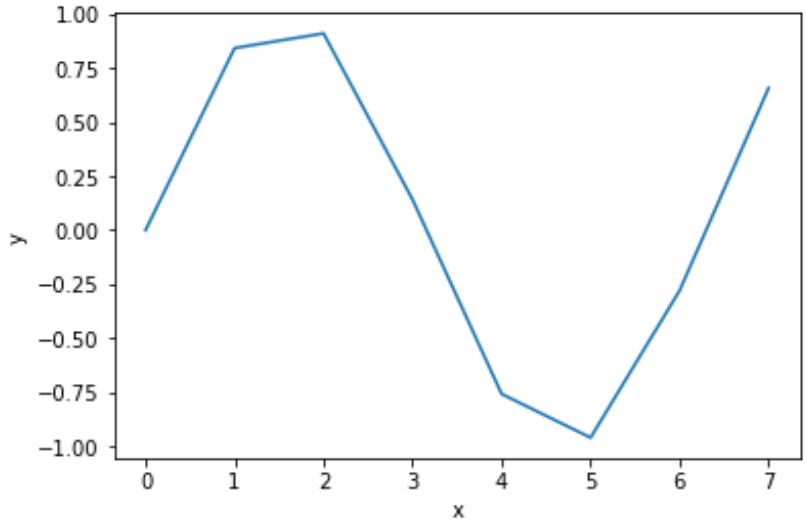


Figure 4.4: Plotting a Sine function in Python

```
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

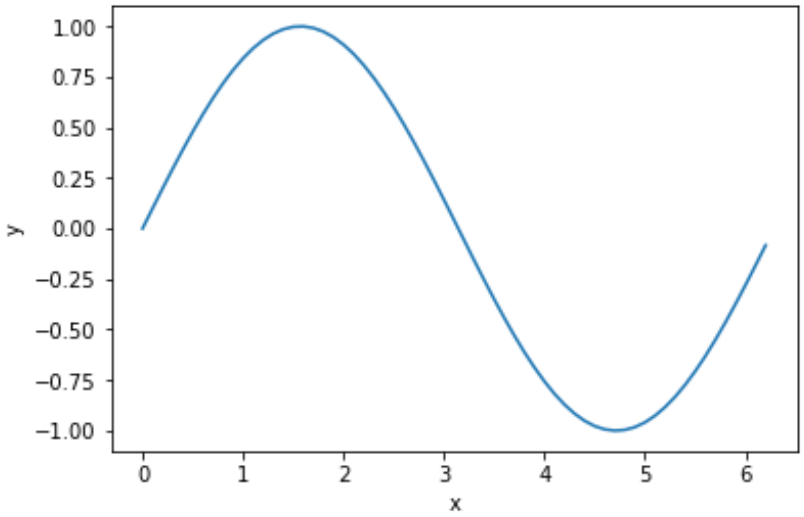


Figure 4.5: Plotting a Sine function in Python - Better Implementation

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]

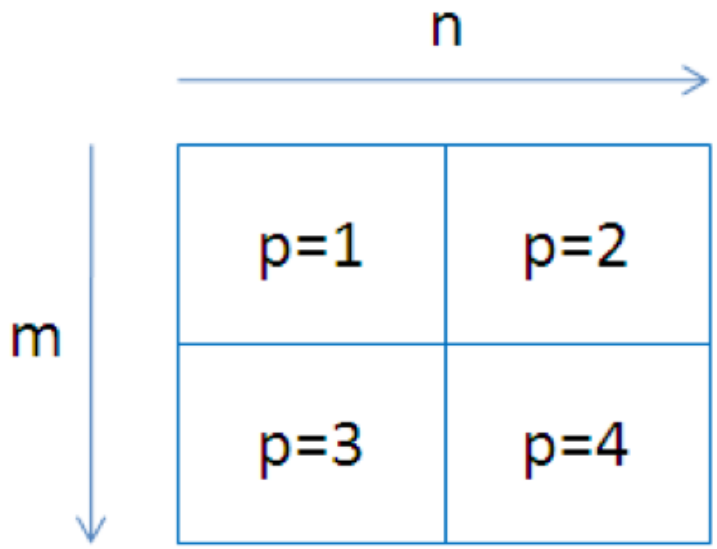


Figure 4.6: Creating Subplots in Python

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 has 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 For Loops in Python

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

Listing 5.1: Using Arrays in Python

Try to change the values for a and b.

Using **If - Else**:

```
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: Using Arrays in Python

Using **Elif**:

```
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: Using Arrays in Python

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. Using For Loops in 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])


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

```
16
17
18 data.append(11.4)
19
20
21 N = len(data)
22
23 print(N)
24
25
26 for x in data:
27   print(x)
```

Listing 5.4: Using Arrays in Python

You define an array like this:

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

You can also use text like this:

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

You can use Arrays in Loops like this:

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

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

```
1 N = len(data)
```

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

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

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

```
1 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 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:

```
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

```
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 yo use in For Loops:

```
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:

```
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:

```
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.

```
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:

```
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:

```
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:

```
N = 10

fib = []

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

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

```

```
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:

```
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:

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

Simple example:

```
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

```
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. Create a Function in a separate File

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:

```
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:

```
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:

```
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



```

```
21 data = [1, 5, 6, 3, 12, 3]
22
23
24 totalsum, mean = stat(data)
25
26 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:

```
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:

```
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:

```
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":

```
# 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:

```
# 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:

```
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**):

```
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**):

```
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

```
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

```
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:

```
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.

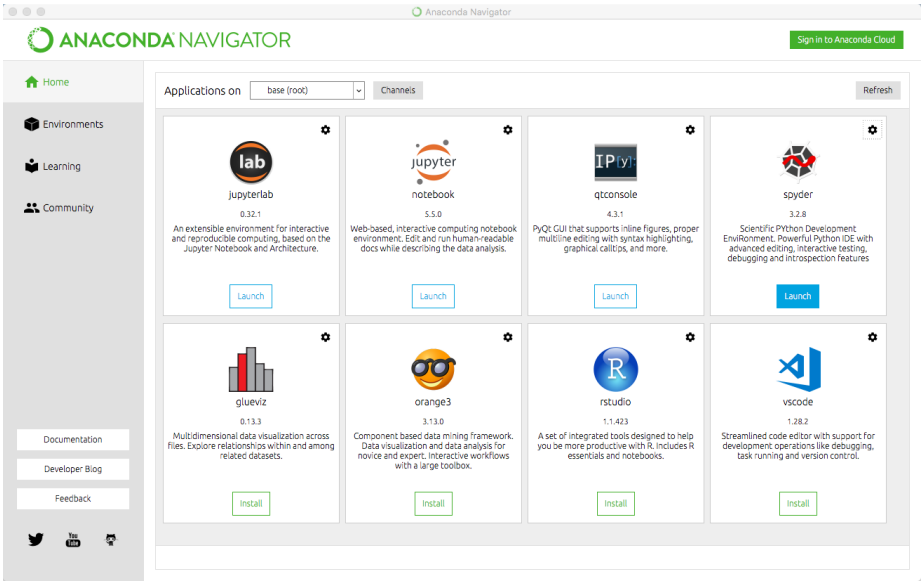


Figure 14.1: Anaconda Navigator

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.

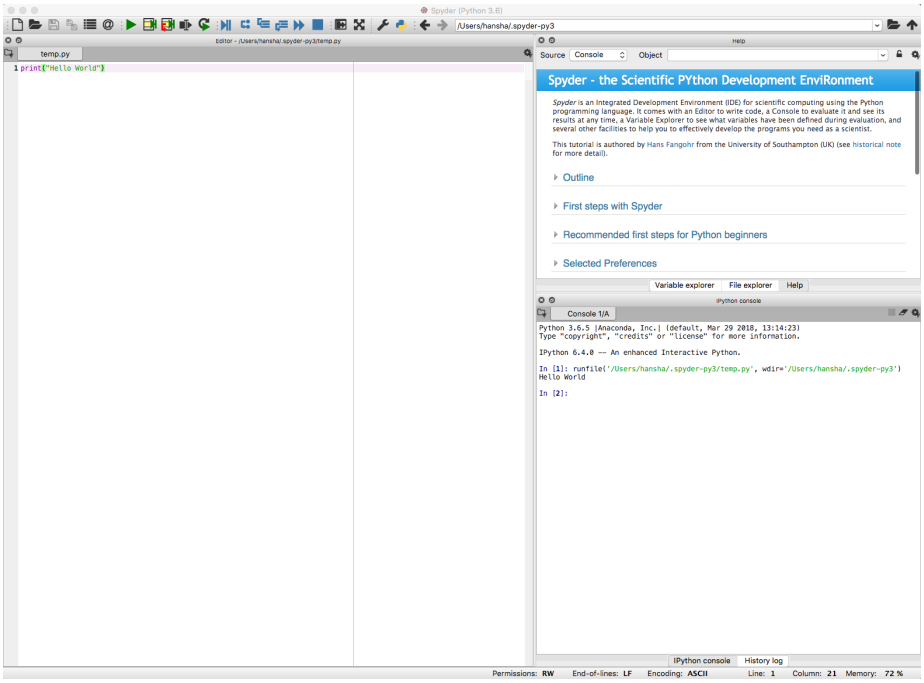


Figure 17.1: Spyder Editor

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

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.

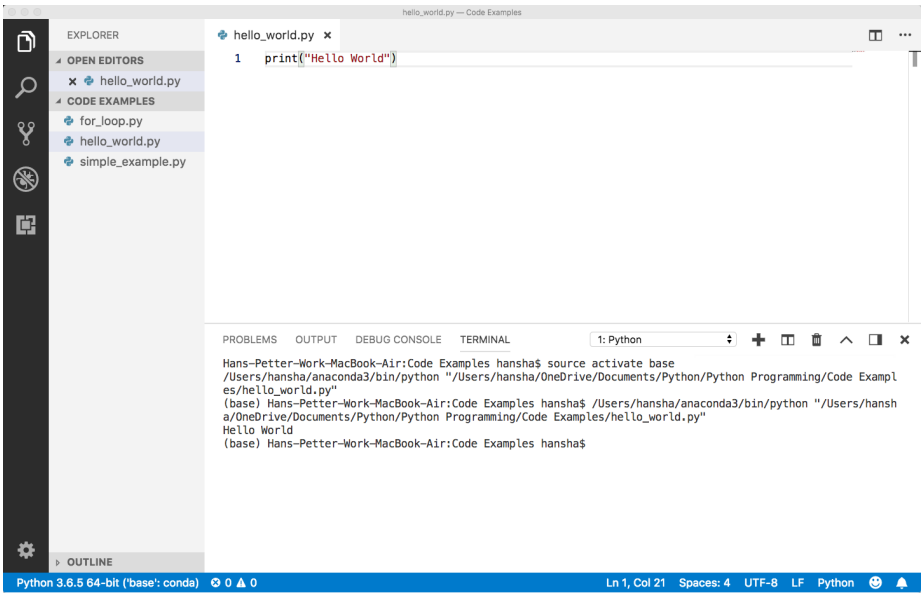


Figure 18.1: Using Visual Studio Code as Python Editor

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

Web:
https://code.visualstudio.com

Wikipedia:
https://en.wikipedia.org/wiki/Visual$_S tudio_C ode$

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/

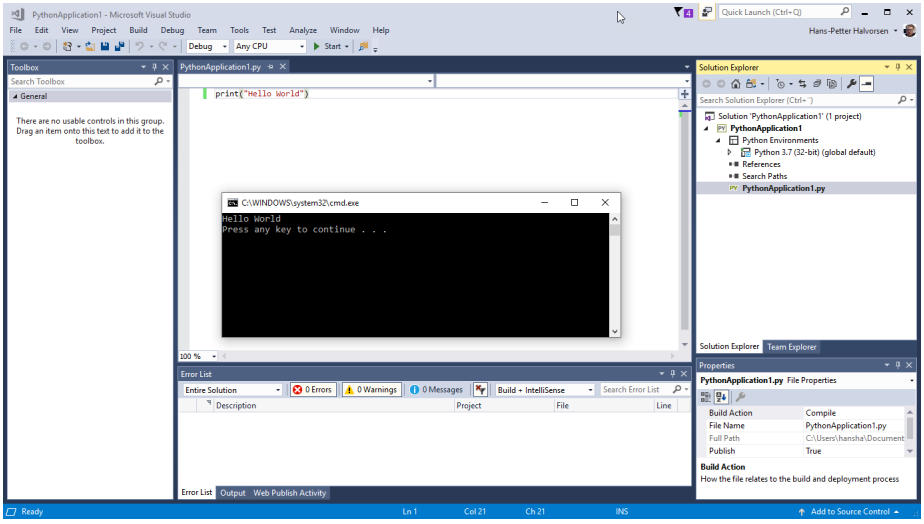


Figure 19.1: Using Visual Studio as Python Editor

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:

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

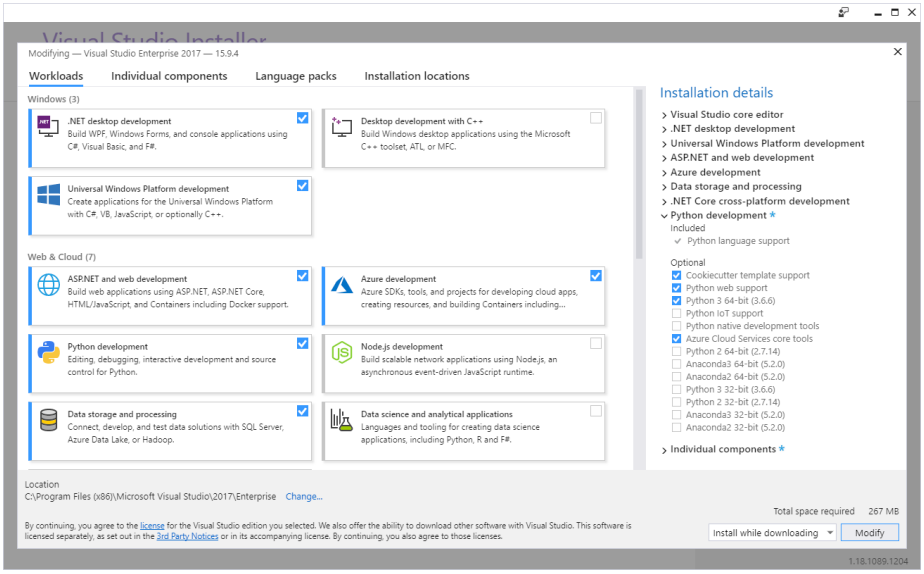


Figure 19.2: Installing Python Extension for Visual Studio

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

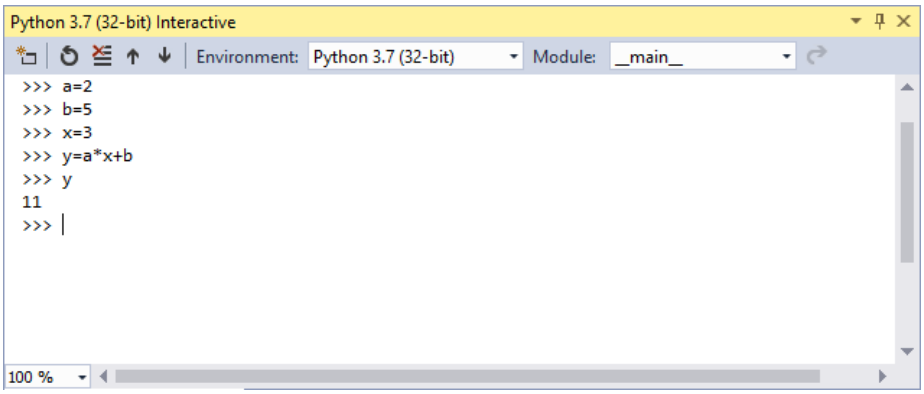


Figure 19.4: Python Interactive

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)')
```

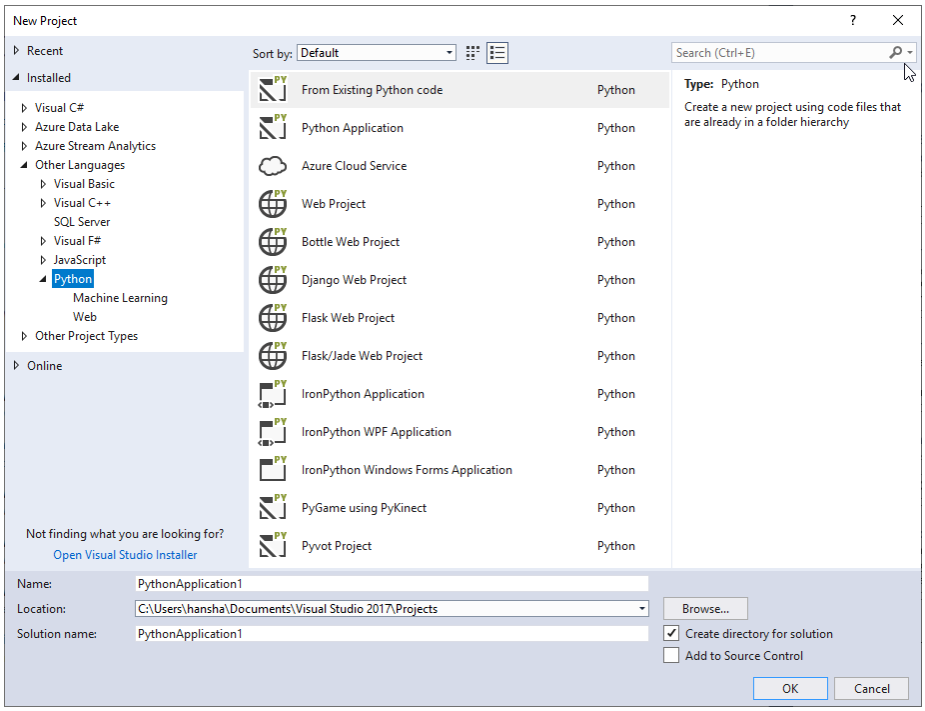


Figure 19.5: New Python Project

```
14 plt.xlabel('x')
15 plt.ylabel('y')
16 plt.grid()
17 plt.axis([0, 2*np.pi, -1, 1])
18 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.

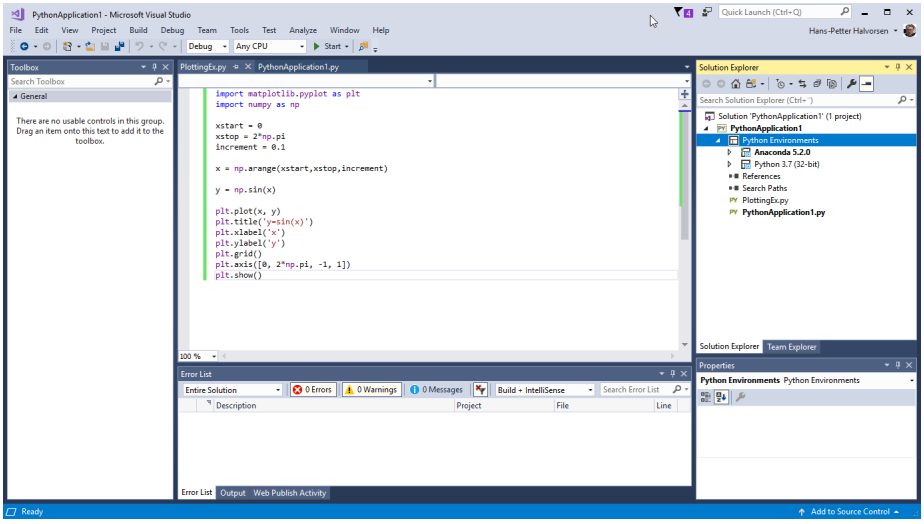


Figure 19.6: Python Plotting Example with Visual Studio

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]

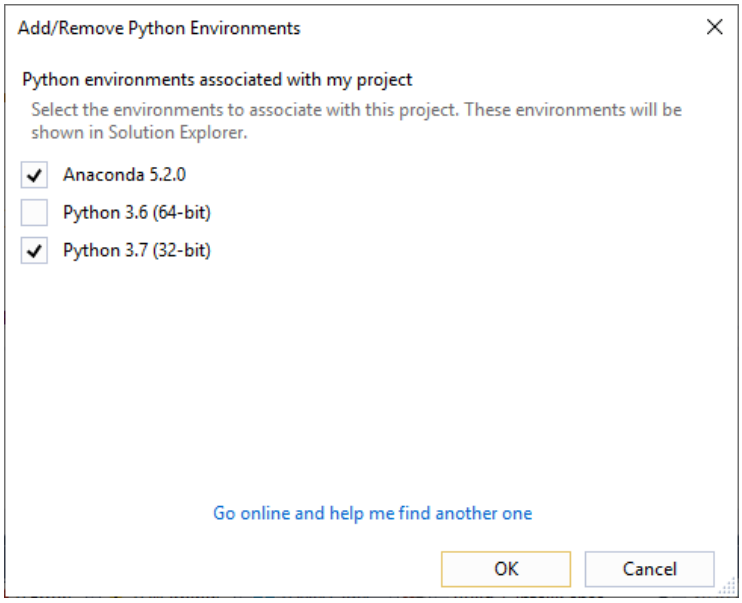


Figure 19.7: Select your Python Environment

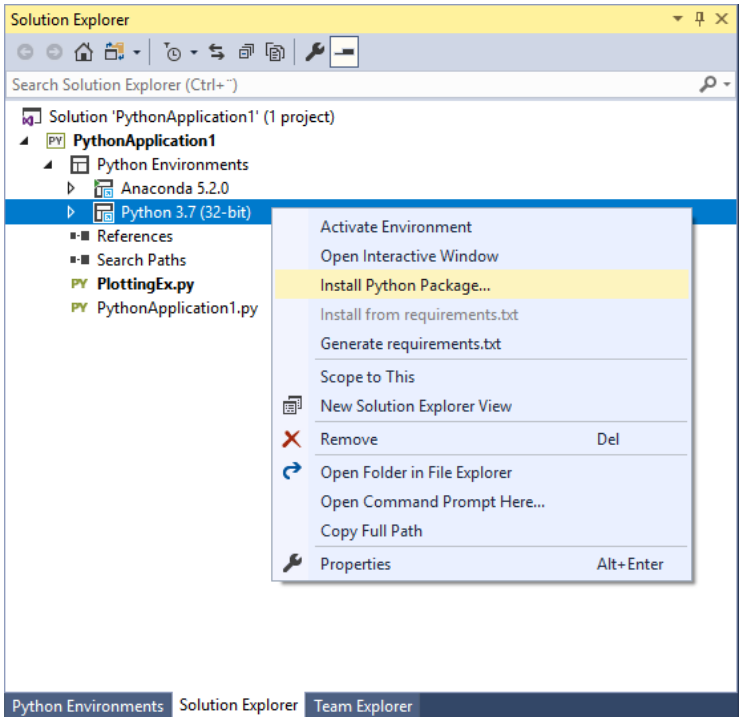


Figure 19.8: Install Python Packages from Visual Studio

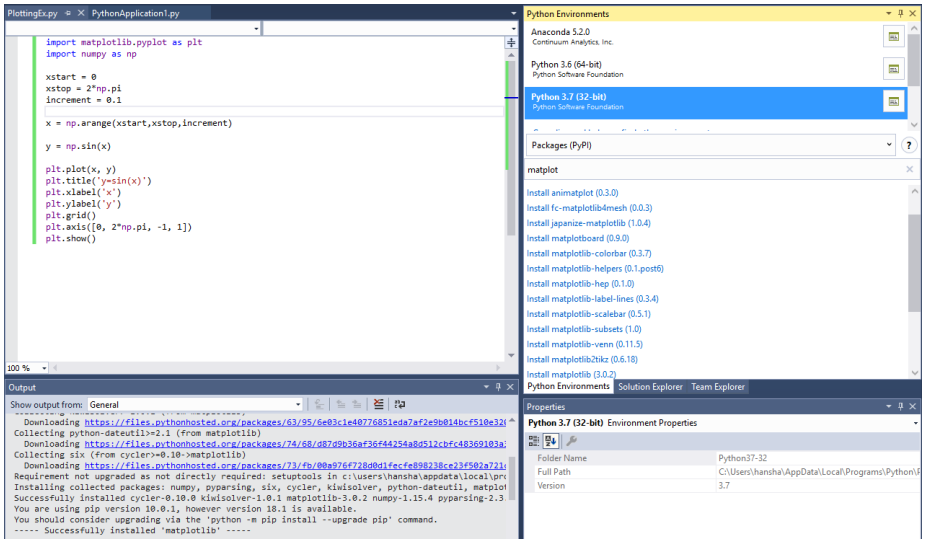


Figure 19.9: Installing Python Packages for different Python Environments from Visual Studio

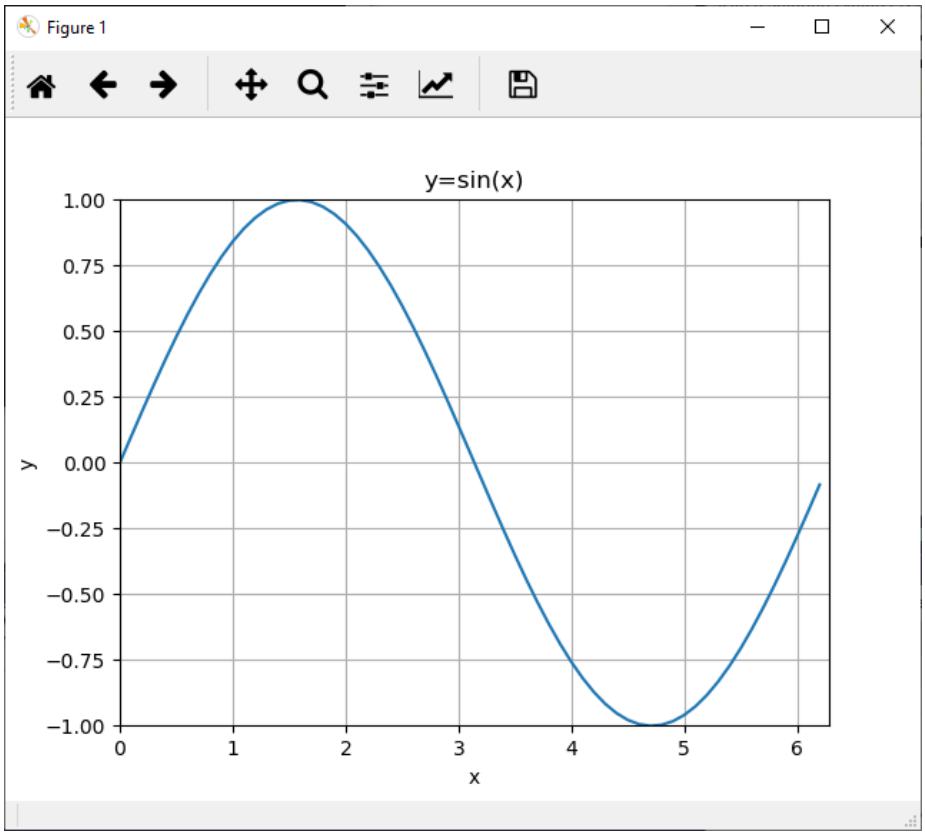


Figure 19.10: Python Plotting Example with Visual Studio

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.

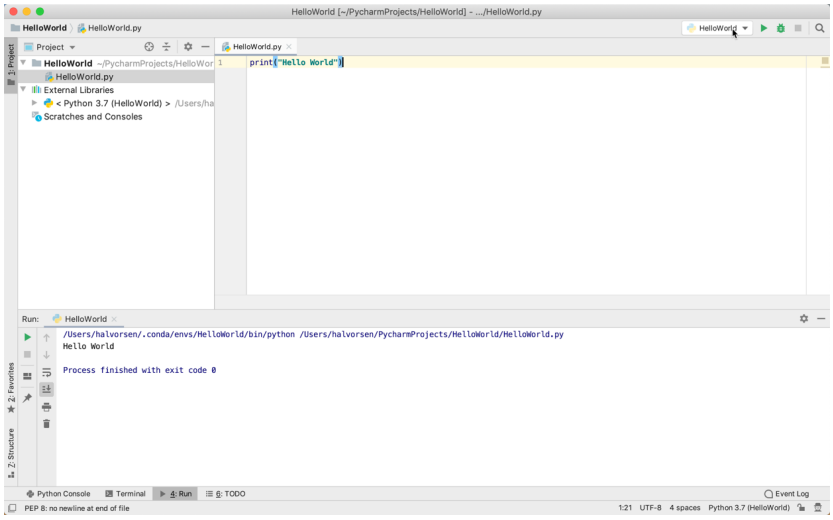


Figure 20.1: PyCharm Python Editor

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

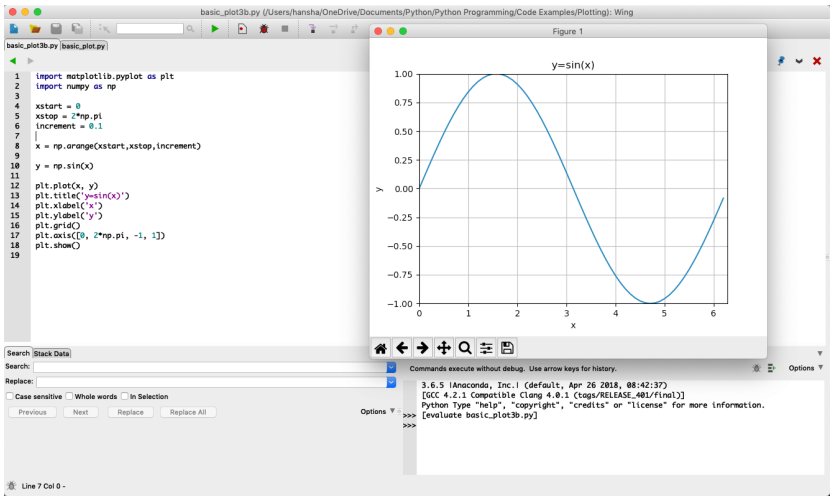


Figure 21.1: Wing Python IDE

Web:
https://wingware.com

Wikipedia:
https://en.wikipedia.org/wiki/Wing$_I DE$

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.

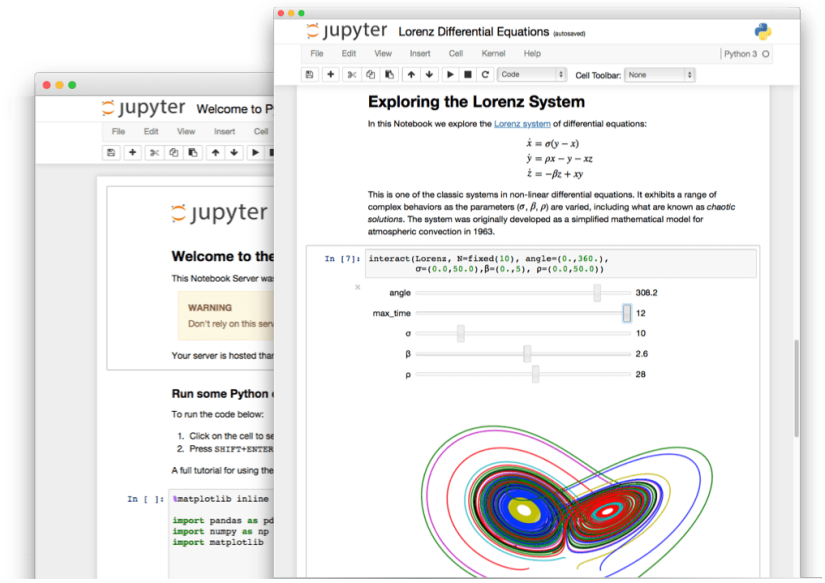


Figure 22.1: Jupyter Notebook [16]

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.

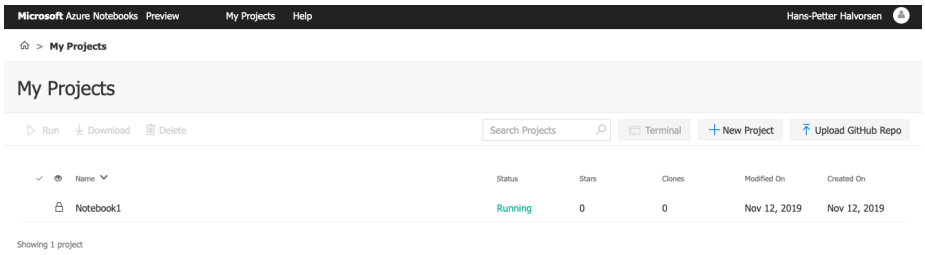


Figure 22.2: Azure Notebook Projects

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]

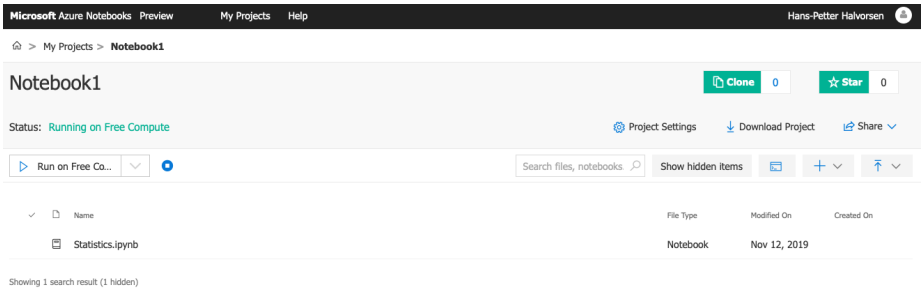


Figure 22.3: Azure Notebook Project Notebooks

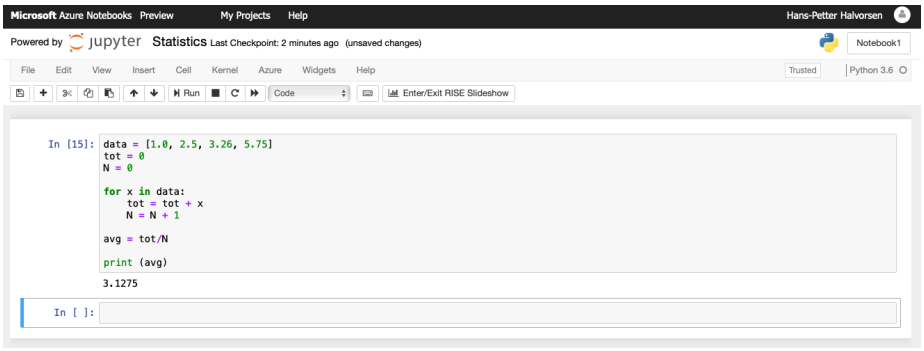


Figure 22.4: Azure Notebook Example

Part V

Python for Mathematics Applications

Chapter 23

Mathematics in Python

Python is a powerful tool for mathematical calculations.

If you are looking for similar using MATLAB, please take a look at these resources:
https://www.halvorsen.blog/documents/programming/matlab/

23.1 Basic Math Functions

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.

In this chapter we will focus on the math module that is part of the Python Standard Library.

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. etc.

Example 23.1.1. 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/

23.1.1 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 23.1.1. Create Mathematical Expressions in Python

Create a function that calculates the following mathematical expression:

$$z = 3x^2 + \sqrt{x^2 + y^2} + e^{\ln(x)} \tag{23.1}$$

Test with different values for x and y.

[End of Exercise]

Exercise 23.1.2. Create advanced Mathematical Expressions in Python

Create the following expression in Python:

$$f(x) = \frac{\ln(ax^2 + bx + c) - sin(ax^2 + bx + c)}{4\pi x^2 + cos(x - 2)(ax^2 + bx + c)} \tag{23.2}$$

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 $f(9)$

[End of Exercise]

Exercise 23.1.3. Pythagoras

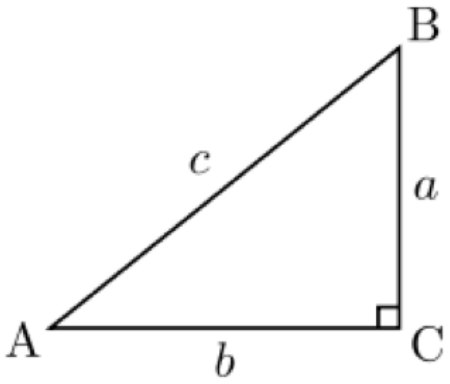


Figure 23.1: Right-angled triangle

Pythagoras theorem is as follows:

$$c^2 = a^2 + b^2 \tag{23.3}$$

Create a function that uses Pythagoras to calculate the hypotenuse of a right-angled triangle (Figure 23.1), e.g.:

```
def pythagoras(a,b)
    ...
    ...
    return c
```

[End of Exercise]

Exercise 23.1.4. Albert Einstein

Given the famous equation from Albert Einstein:

$$E = mc^2 \tag{23.4}$$

The sun radiates $385x10^{24} J/s$ of energy.

Calculate how much of the mass on the sun is used to create this energy per day.

How many years will it take to convert all the mass of the sun completely? Do we need to worry if the sun will be used up in our generation or the next? justify the answer.

The mass of the sun is $2x10^{30} kg$.

[End of Exercise]

Exercise 23.1.5. Cylinder Surface Area

Create a function that finds the surface area of a cylinder based on the height (h) and the radius (r) of the cylinder. See Figure **??**.

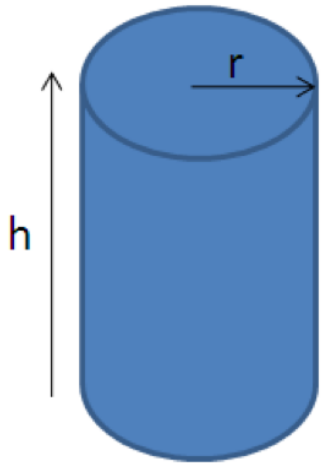


Figure 23.2: cylinder

[End of Exercise]

23.2 Statistics

23.2.1 Introduction to Statistics

Mean or average:
The mean is the sum of the data divided by the number of data points. It is commonly called "the average",

Formula for mean:

$$\bar{x} = \frac{x_1 + x_2 + x_3 + ... + x_N}{N} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{23.5}$$

Example 23.2.1. Mean

Given the following dataset: 2.2, 4.5, 6.2, 3.6, 2.6

Mean:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{2.2 + 4.5 + 6.2 + 3.6 + 2.6}{5} = \frac{19.1}{5} = 3.82 \tag{23.6}$$

[End of Example]

Variance:

Variance is a measure of the variation in a data set.

$$var(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{23.7}$$

Standard deviation:
The standard deviation is a measure of the spread of the values in a dataset or the value of a random variable. It is defined as the square root of the variance.

$$std(x) = \sigma = \sqrt{var} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{23.8}$$

We typically use the symbol σ for standard deviation.

We have that $\sigma^2 = var(x)$

23.2.2 Statistics functions in Python

Mathematical statistics functions in Python:
https://docs.python.org/3/library/statistics.html

statistics is part of the The Python Standard Library.

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

Example 23.2.2. Statistics using the statistics module in Python Standard Library

Below you find some examples how to use some of the statistics functions in the statistics module in Python Standard Library:

```
import statistics as st

data = [-1.0, 2.5, 3.25, 5.75]

#Mean or Average
m = st.mean(data)
print(m)

# Standard Deviation
st_dev = st.stdev(data)
```

```
print(st_dev)

# Median
med = st.median(data)
print(med)

# Variance
var = st.variance(data)
print(var)
```

Listing 23.1: Statistics functions in Python

[End of Example]

IMPORTANT: Do not name your file "statistics.py" since the import will be confused and throw the errors of the library not existing and the mean function not existing.

You can also use the **NumPy** Library. NumPy is the fundamental package for scientific computing with Python.

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

Example 23.2.3. Statistics using the NumPy Library

Below you find some examples how to use some of the statistics functions in NumPy:

```
import numpy as np

data = [-1.0, 2.5, 3.25, 5.75]

#Mean or Average
m = np.mean(data)
print(m)

# Standard Deviation
st_dev = np.std(data)
print(st_dev)

# Median
med = np.median(data)
print(med)

# Minimum Value
minv = np.min(data)
print(minv)

# Maxumum Value
maxv = np.max(data)
print(maxv)
```

Listing 23.2: Statistics using the NumPy Library

[End of Example]

Exercise 23.2.1. Create your own Statistics Module in Python

Using the built-in functions in the Python Standard Library or the NumPy library is straightforward.

In order to get a deeper understanding of the mathematics behind these functions and to learn more Python programming, you should create your own Statistics Module in Python.

Create your own Statistics Module in Python (e.g., "mystatistics.py) and then create a Python Script (e.g., "testmystatistics.py) where you test these functions.

You should at least implement functions for mean, variance, standard deviation, minimum and maximum.

[End of Exercise]

23.3 Trigonometric Functions

Python offers lots of Trigonometric functions, e.g., sin, cos, tan, etc.

Note! Most of the trigonometric functions require that the angle is expressed in radians.

Example 23.3.1. Trigonometric Functions in Math module

```
1  import math as mt
2
3  x = 2*mt.pi
4
5  y = mt.sin(x)
6  print(y)
7
8  y = mt.cos(x)
9  print(y)
10
11 y = mt.tan(x)
12 print(y)
```

Listing 23.3: Trigonometric Functions in Math module

Here we have used the Math module in the Python Standard Library.

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

[End of Example]

Example 23.3.2. Plotting Trigonometric Functions

In the example above we used some of the trigonometric functiosn in basic calculations.

Lets see if we are able to plot these functions.

```
import math as mt
import matplotlib.pyplot as plt

xdata = []
ydata = []

for x in range(0, 10):
    xdata.append(x)
    y = mt.sin(x)
    ydata.append(y)

plt.plot(xdata, ydata)
plt.show()
```

Listing 23.4: Plotting Trigonometric Functions

In the example we have plotted sin(x), we can easily extend the program to plot cos(x), etc.

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

[End of Example]

Example 23.3.3. Trigonometric Functions using NumPy

The problem with using the Trigonometric functions in the the Math module from the Python Standard Library is that they don't handle an array as input.

We will use the NumPy library instead because they handle arrays, in addition to all the handy functionality in the NumPy library.

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

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()

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

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

Listing 23.5: Trigonometric Functions using NumPy

This Python script gives the plots as shown in Figure 23.3.

[End of Example]

Exercise 23.3.1. Create Python functions for converting between radians an 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:

$$2\pi[radians] = 360[degrees] \tag{23.9}$$

This gives:

$$d[degrees] = r[radians] \times (\frac{180}{\pi}) \tag{23.10}$$

and

$$r[radians] = d[degrees] \times (\frac{\pi}{180}) \tag{23.11}$$

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 23.3.2. Trigonometric functions on right triangle

Given right triangle as shown in Figure 23.4.

Create a function that finds the angle A (in degrees) based on input arguments (a,c), (b,c) and (a,b) respectively.

Use, e.g., a third input "type" to define the different types above.

Use you previous function r2d() to make sure the output of your function is in degrees and not in radians.

Test the function to make sure it works properly.

Tip! We have that:

$$\sin(A) = \frac{a}{c} \rightarrow A = \arcsin(\frac{a}{c}) \tag{23.12}$$

$$\cos(A) = \frac{b}{c} \rightarrow A = \arccos(\frac{b}{c}) \tag{23.13}$$

$$\tan(A) = \frac{a}{b} \rightarrow A = \arctan(\frac{a}{b}) \tag{23.14}$$

We may also need to use the Pythagoras' theorem:

$$c^2 = a^2 + b^2 \tag{23.15}$$

```
>>> a=5
>>> b=8
>>> c = sqrt(a**2 + b**2)

>>> A = right_triangle(a,c,'sin')
A =
    32.0054

>>> A = right_triangle(b,c,'cos')
A =
    32.0054
>>> A = right_triangle(a,b,'tan')
A =
    32.0054
```

We also see that the answer in this case is the same, which is expected.

[End of Exercise]

Exercise 23.3.3. Law of Cosines

Given the triangle as shown in Figure 23.5.

Create a function where you find c using the **law of cosines**.

$$c^2 = a^2 + b^2 - 2ab\cos(C) \tag{23.16}$$

Test the functions to make sure it works properly.

[End of Exercise]

Exercise 23.3.4. Plotting Trigonometric Functions

Plot $sin(\theta)$ and $cos(\theta)$ for $0 \leq \theta \leq 2\pi$ in the same plot (both in the same plot and in 2 different subplots).

Make sure to add labels and a legend and use different line styles and colors for the plots.

[End of Exercise]

23.4 Polynomials

A polynomial is expressed as:

$$p(x) = p_1 x^n + p_2 x^{n-1} + ... + p_n x + p_{n+1} \tag{23.17}$$

where $p_1, p_2, p_3, ...$ are the coefficients of the polynomial.

We will use the Polynomial Module in the NumPy Package.

Web:
https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.polynomials.polynomial.html

Other Resources:

Python Advanced Course Topics - Polynomials:
https://www.python-course.eu/polynomial$_c lass_i n_p ython.php$

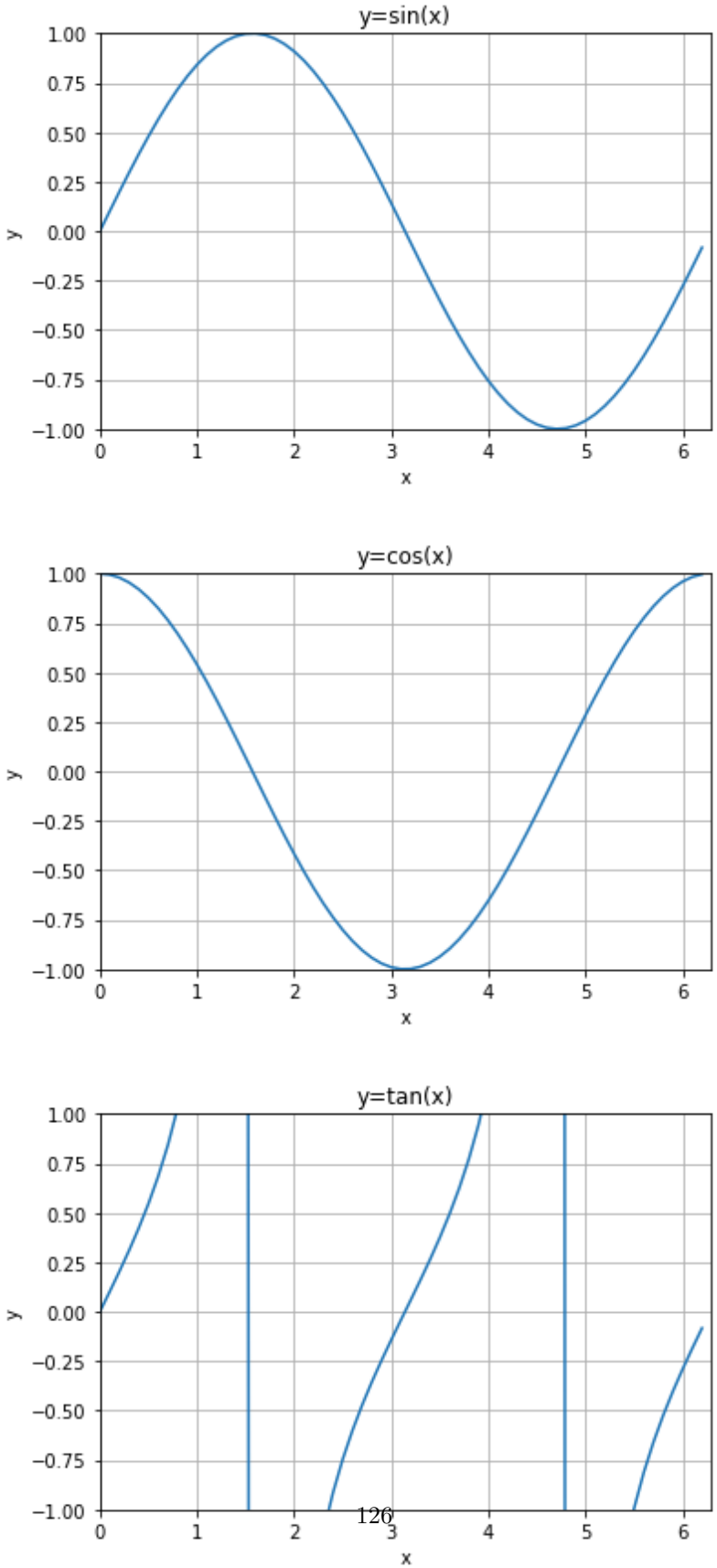


Figure 23.3: Trigonometric Functions

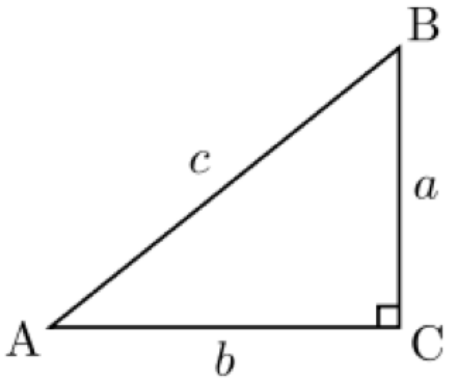


Figure 23.4: Right Triangle

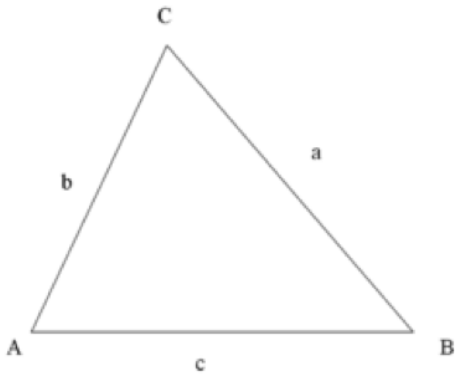


Figure 23.5: Law of Cosines

Part VI

Resources

Chapter 24

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 [18]:
https://docs.python.org/3/library/index.html

24.1 Python Distributions

Anaconda:
https://www.anaconda.com

24.2 Python Libraries

NumPy Library:
http://www.numpy.org

SciPy Library:
https://www.scipy.org

Matplotlib Library:
https://matplotlib.org

24.3 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

24.4 Python Tutorials

Python Tutorial - w3schools.com [13]:
https://www.w3schools.com/python/

The Python Guru [19]:
https://thepythonguru.com

Wikibooks - A Beginner's Python Tutorial:
https://en.wikibooks.org/wiki/A$_B$*eginner*

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/

24.5 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$_{(programming language)}$," 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, "The python standard library - https://docs.python.org/3/library/," 2018.

[19] T. P. Guru, "The python guru - https://thepythonguru.com," 2018.

Part VII

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.

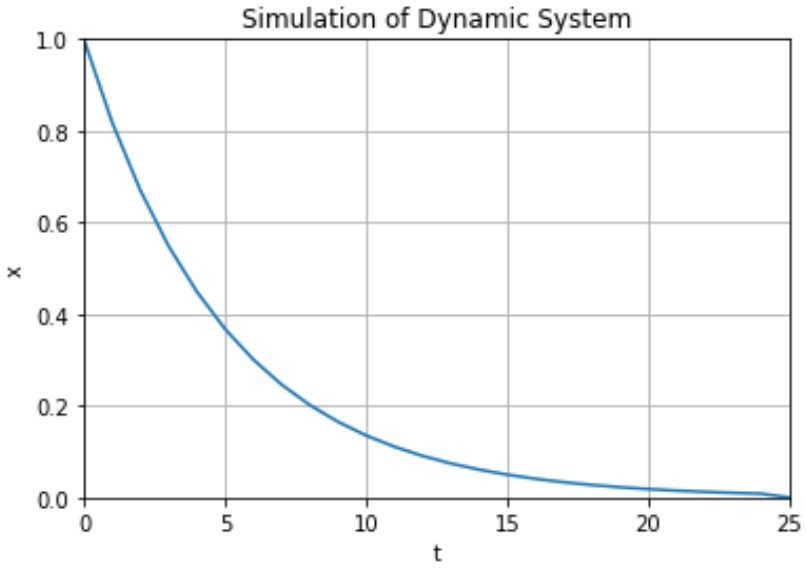


Figure 1: Simulation of Dynamic System

[End of Exercise]

Python Programming

August 12, 2020

ISBN:978-82-691106-4-7

Python Programming