

Python for Data Science

Overview of Python
Why Python
Installing Python
Installing Python Modules



Overview of the course

- Assumptions:
 - We are here to learn some new skills
 - We learn new skills by doing
 - We work better with others
 - Python is important
 - It is a glue language
 - Need minimal python skills to use
 - It is interesting on its own
 - It's a modern language with interesting features
 - It's useful where-ever modules don't exist

Python

- Python is an interpreted (scripting) language
 - Source code is compiled into a bytecode representation
 - Executed by Python virtual machine (usually implemented in C or Java)
 - If performance is needed:
 - Can call C-code from Python
 - Use PyPy with Just-In-Time compilation (JIT)

Python

- Why Python:
 - Cool language
 - Extensible through modules
 - Statistics
 - Machine learning
 - Graphics

Python

- Getting Python
 - Can use bundles (anaconda)
 - For the first half: get native Python from Python.org
 - Python 2.7 stable solution (built into MacOS)
 - **Python 3.8.2** the version we need
 - Important : Allow automatic path adjustments on windows
 - This are the defaults

Python

- Using Python:
 - We are going to use IDLE
 - Can create and save scripts
 - Can interact directly in the IDE

Python 3 Modules

- Python comes with many pre-installed modules
- We need to install some modules
 - Use Pip
 - MacOS / Linus
 - In a shell:

```
thomasschwarz@Peter-Canisius Module1 % python3.8 -m pip install matplotlib
```

- Windows:
 - In a command window

```
py -3.8 -m pip install matplotlib
```

Why Python

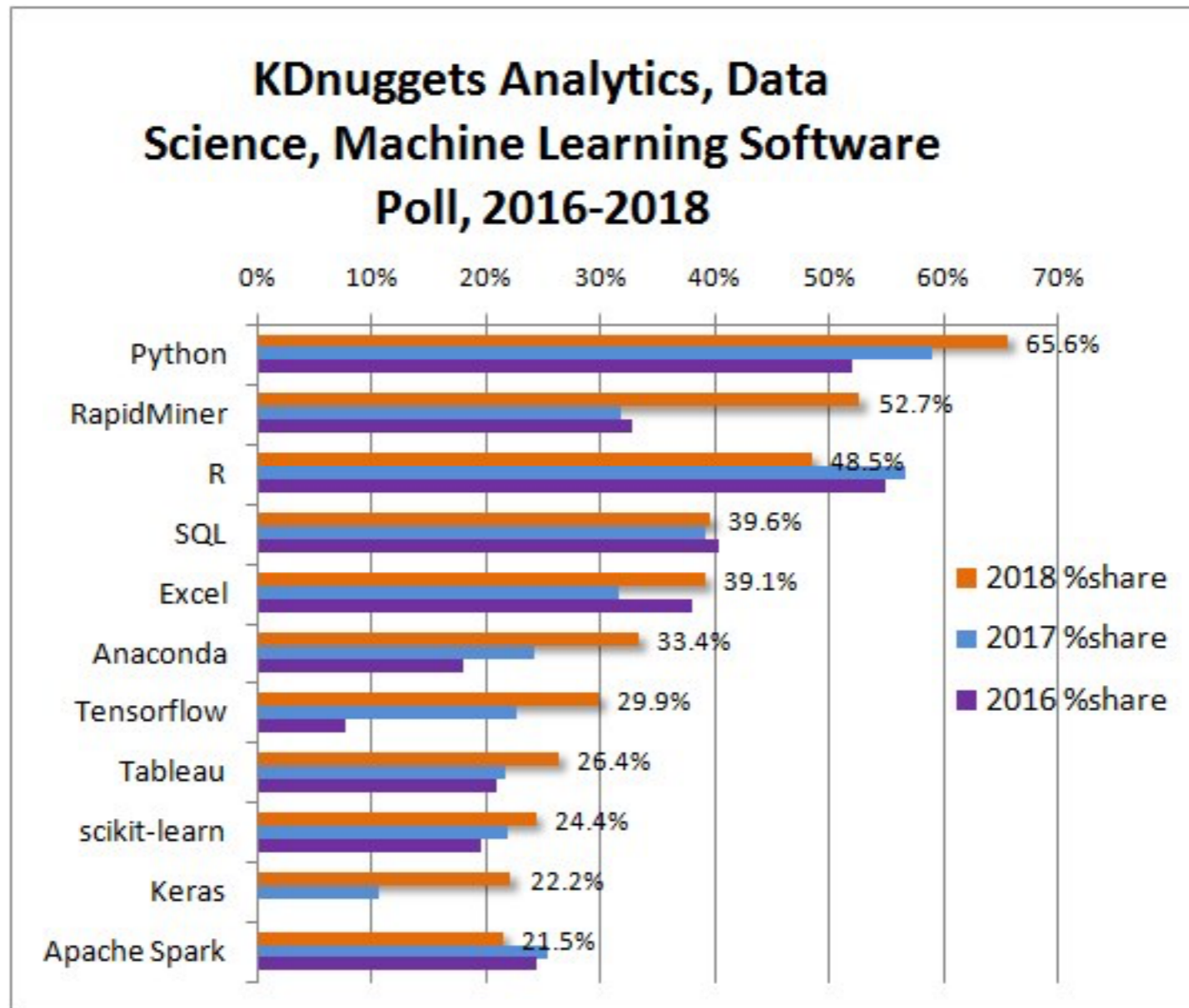
- Universal, accessible language
 - Clear and simple syntax
 - Python philosophy: The frequent should be easy
 - Made for reading
 - Used for fast prototyping
- Excellent support community
 - Help for beginners and experts is easily available

Why Python

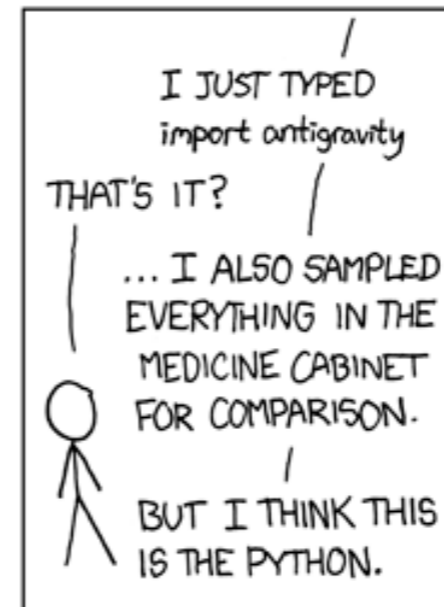
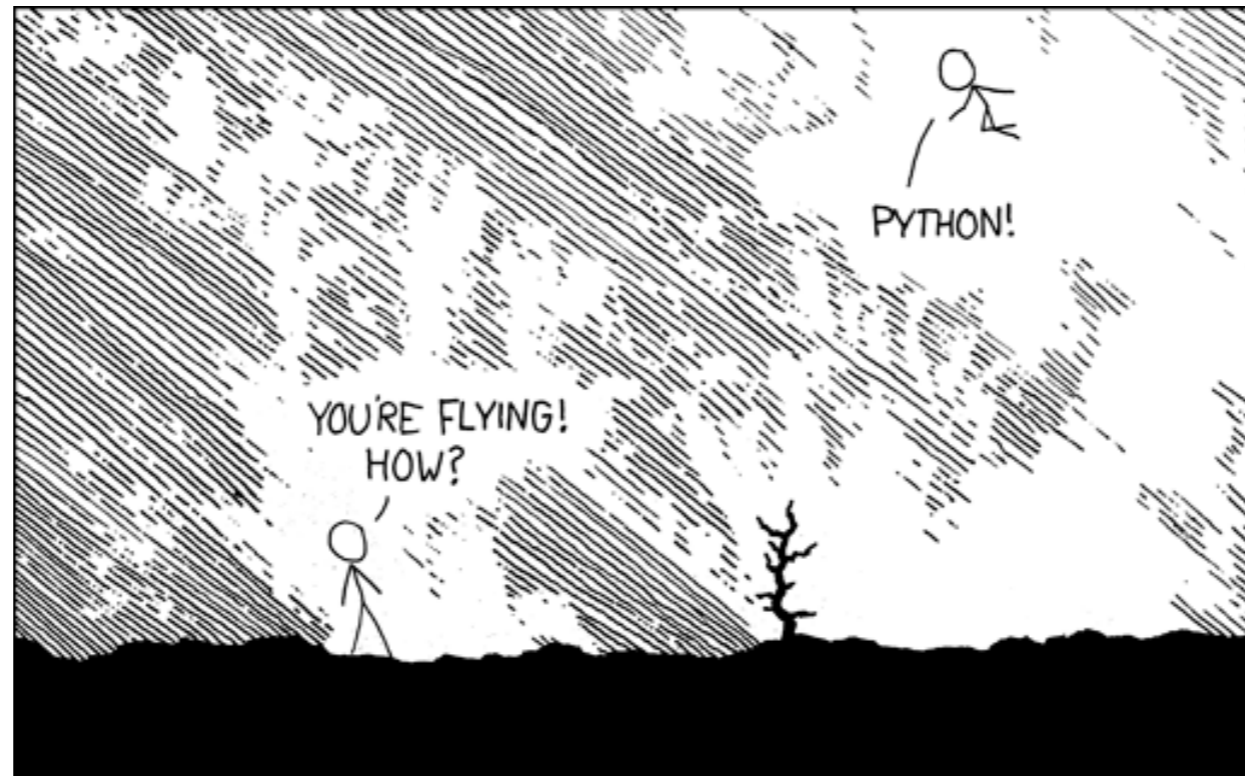
- Universal Language
 - Serves many different constituencies
 - Examples:
 - Gaming: AI engine is usually in Python
 - String processing: Basis for digital humanities and data wrangling
 - Many extension modules
 - With scypy or numpy, fast programs for scientific programming
 - Use pyplot for good quality graphics
 - ...
 - Notebooks based on Python (Jupyter) integrate presentation, data, and programs

Why Python

- Python in Data Science



Python Modules



Why Python

- Example:
 - Time series data: closing prices of four stock indices
 - given as a cvs file
 - Use Pandas in order to deal with two dimensional data
 - Use matplotlib for graphics

Why Python? Time Series

Example

- Import the modules

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

- Import the csv file as a pandas dataframe

```
raw_data = pd.read_csv('Index2018.csv')
values = raw_data.copy()
```

- The first column should be the index, read as a date

```
values.date = pd.to_datetime(values.date, dayfirst=True)
values.set_index("date", inplace = True)
print(values.describe())
print(values.head())
```

Why Python? Time Series

Example

- Fill in missing values and normalize to start at 100

```
values.spx = values.spx.fillna(method = 'ffill')/values.spx['1994-01-07']*100.0  
values.dax = values.dax.fillna(method = 'ffill')/values.dax['1994-01-07']*100.0
```

- Now display the US Standard & Poor and the German DAX

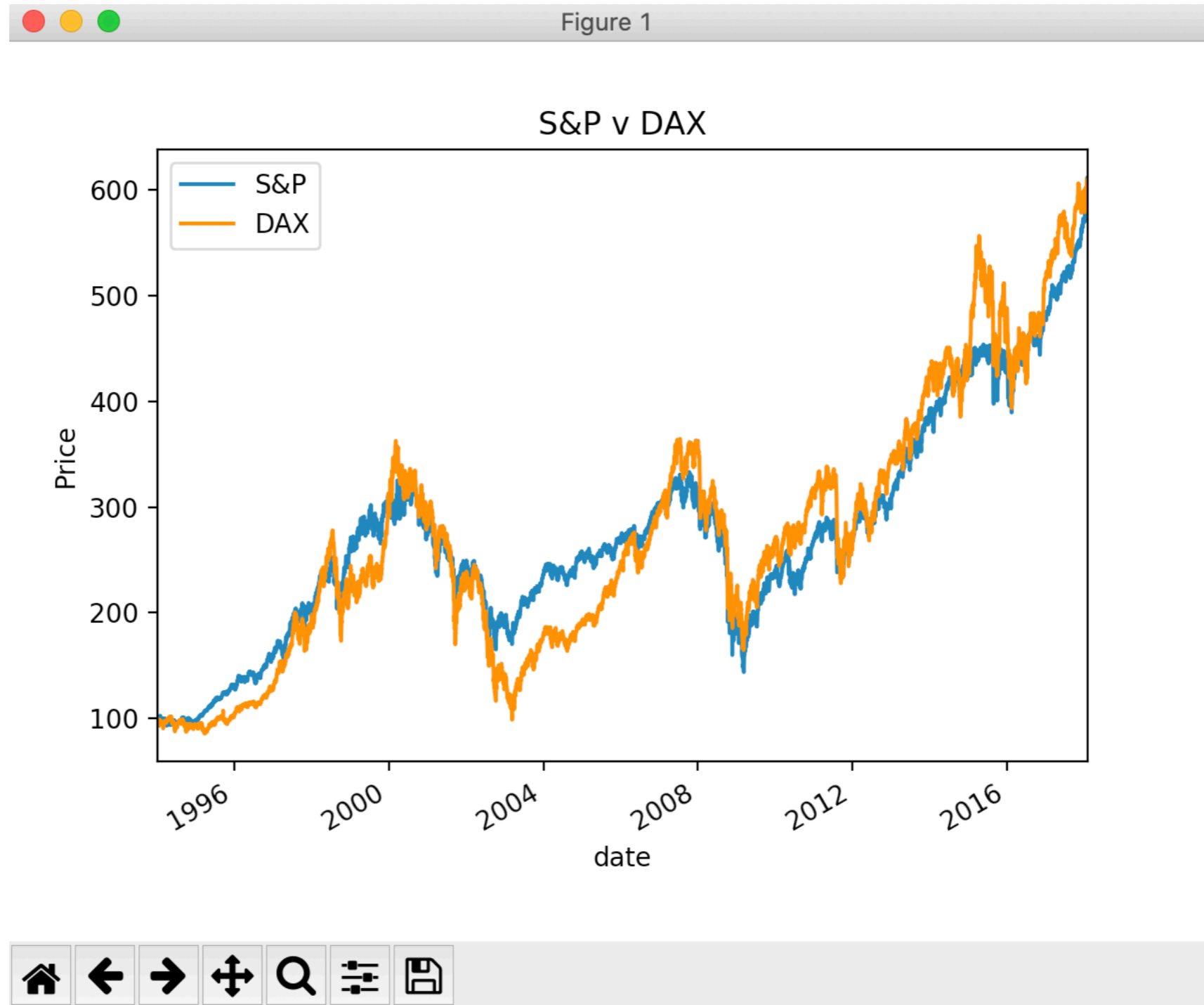
```
values.spx.plot(label='S&P')  
values.dax.plot(label='DAX')
```

- Now annotate the plot and show it

```
plt.title('S&P v DAX')  
plt.xlabel('date')  
plt.ylabel('Price')  
plt.legend()  
plt.show()
```

Why Python? Time Series Example

- Result:



Why Python? Time Series Example

Why Python? Time Series Example


Why Python? Time Series Example

Why Python

- Most of the programming was done for us
- Needed to invoke powerful method
- Majority of the code giving to small tweaks

IDLE

- IDLE is an interactive Python interpreter
 - Can be used as a desk calculator
 - Allows you to create new files



```
Python 3.6.5 Shell
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 03:03:55)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: /Users/thomasschwarz/Google Drive/AATeaching/Python 2/Programs/pi.py
0.7831
>>>
RESTART: /Users/thomasschwarz/Google Drive/AATeaching/Python 2/Programs/pi.py
0.7807
0.7853981633974483
|>>>
```

Ln: 11 Col: 0

Variables and Types

- All program languages specify how data in memory locations is modified
- Python: *A variable* is a handle to a storage location
 - The storage location can store data of many types
 - Integers
 - Floating point numbers
 - Booleans
 - Strings

Variables and Types

- Assignment operator = makes a variable name refer to a memory location
- Variable names are not declared and can refer to any legitimate type

```
a = 3.14156432  
b = "a string"
```

a▶ 3.14156432

b▶ "a string"

```
a = b
```

a▶ ~~3.14156432~~

b▶ "a string"

- Create two variables and assign values to them
- Variable *a* is of type floating point and variable *b* is of type string
- After reassigning, both variable names refer to the same value
- The floating point number is garbage collected

Expressions

- Python builds expression from smaller components just as any other programming language
 - The type of operation expressed by the same symbol depends on the type of operands
- Python follows the usual rules of precedence
 - and uses parentheses in order to express or clarify orders of precedence.

Expressions

- Arithmetic Operations between integers / floating point numbers:
 - Negation (-), Addition (+), Subtraction (-), Multiplication (*), Division (/), Exponentiation (**)
 - Integer Division //
 - Remainder (modulo operator) (%)

Expressions

- IF we use `/` between two integers, then we always get a floating point number
- If we use `//` between two integers, then we always get an integer
 - `a//b` is the integer equal or just below `a/b`

Expressions

- Strings are marked by using the single or double quotation marks
- You can use the other quotation mark within the string
- Some symbols are given as a combination of a forward slash with another symbol
 - Examples: `\t` for tab, `\n` for new line, `\'` for apostrophe, `\"` for double quotation mark, `\\` for backward slash
 - We'll get to know many more, but this is not the topic of today

Expressions

- Strings can be concatenated with the +
- They can be replicated by using an integer and the * sign
- Examples:
 - `"abc"+"def" → 'abcdef'`
 - `'abc\"'+ 'fg' → 'abc"fg'`
 - `3*"Hi '" → "Hi 'Hi 'Hi '"`

Change of Type

- Python allows you to convert the contents of a variable or expression to an expression with a different type but equivalent value
 - Be careful, type conversion does not always work
- To change to an integer, use `int()`
- To change to a floating point, use `float()`
- To change to a string, use `str()`

Example

- Input is done in Python by using the function `input`
 - Input has one variable, the prompt, which is a string
 - The result is a string, which might need to get processed by using a type conversion (aka **cast**)
 - The following prints out the double of the input (provided the user provided input is interpretable as an integer), first as a string and then as a number

```
user_input = input("Please enter a number ")
print(2*user_input)
print(2*int(user_input))
```

```
Please enter a number 23
2323
46
|
```

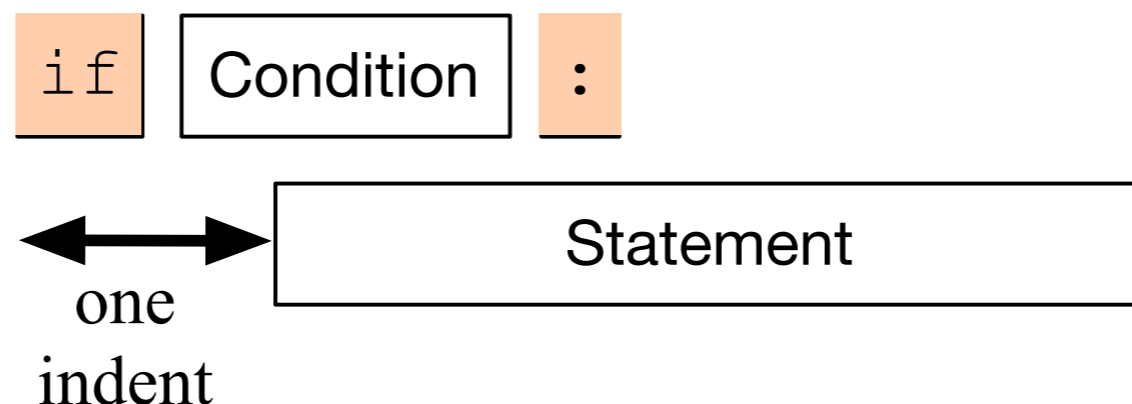
Example

- Python does not understand English (or Hindi) so giving it a number in other than symbolic form does not help
- It can easily understand “123”
- It does not complain about the expression having the same type.

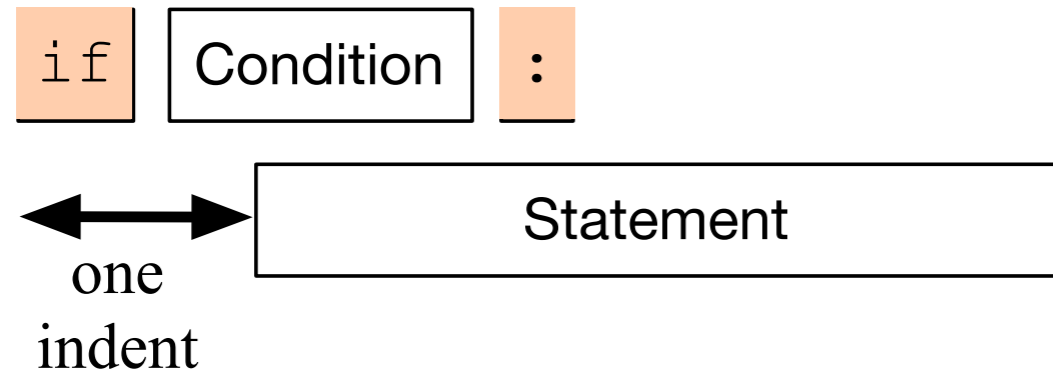
```
>>> int("two")
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    int("two")
ValueError: invalid literal for int() with base 10: 'two'
>>> float("123")
123.0
>>> int(24)
24
>>> |
```

Conditional Statements

- Sometimes a statement (or a block of statements) should only be executed if a condition is true.
- Conditional execution is implemented with the if-statement
- Form of the if-statement:



Conditional Statements



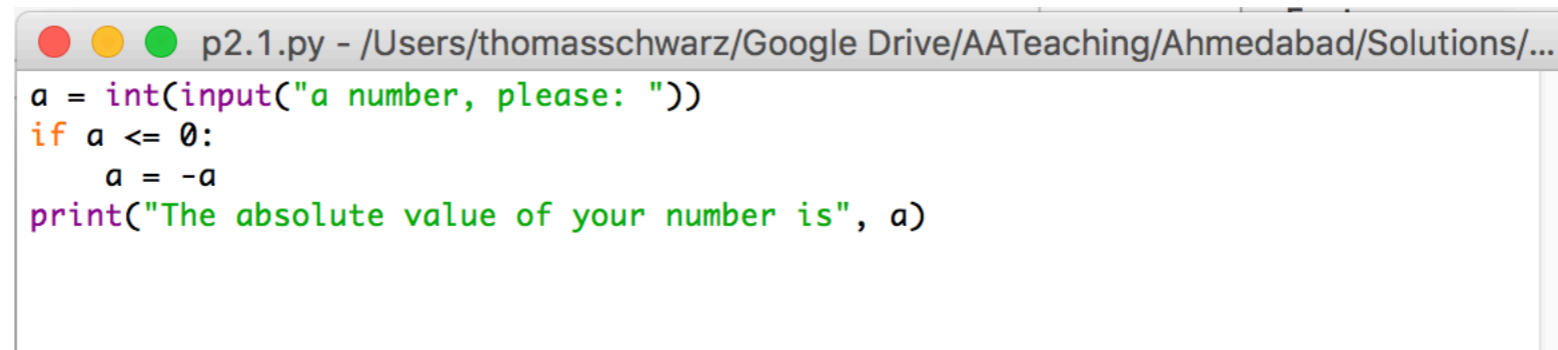
- `if` — is a keyword
- Condition: a Boolean, something that is either True or False
- Statement: a single or block of statements, all indented
 - Indents are tricky, you can use white spaces or tabs, but not both. Many editors convert tabs to white spaces
 - The number of positions for the indent is between 3 and 8, depending on the style that you are using. Most important, keep it consistent.

Example

```
● ● ● p2.1.py - /Users/thomasschwarz/Googl  
a = int(input("a number, please: "))  
if a < 5:  
    print("that is a small number.")
```

- First line asks user for integer input.
- Second line checks whether user input is smaller than 5.
- In this case only, the program comments on the number.

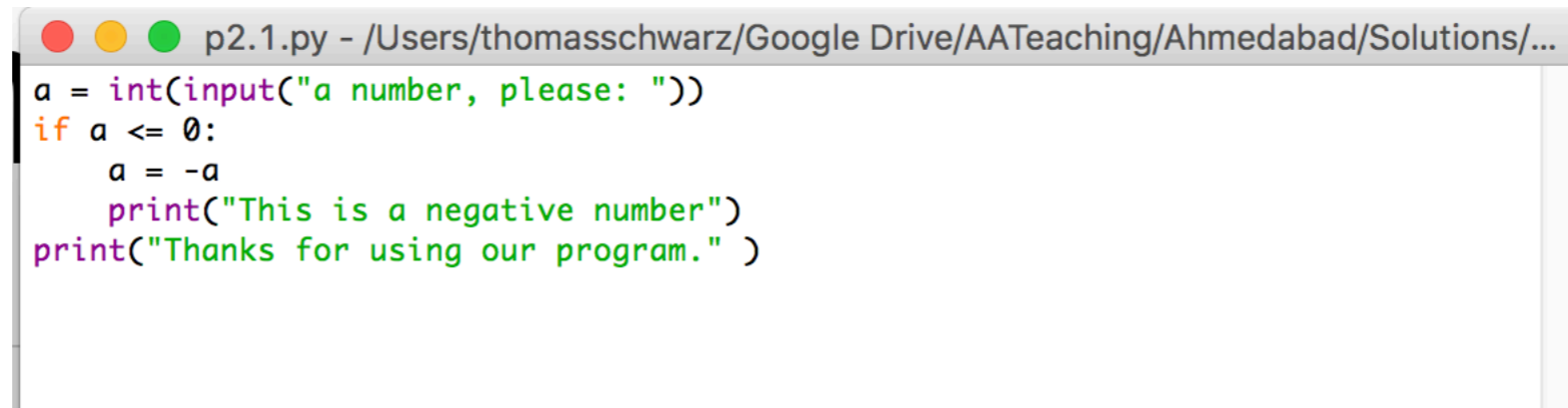
Example

A screenshot of a text editor window showing a Python script. The window title is "p2.1.py - /Users/thomasschwarz/Google Drive/AATeaching/Ahmedabad/Solutions/...". The code is as follows:

```
a = int(input("a number, please: "))
if a <= 0:
    a = -a
print("The absolute value of your number is", a)
```

- Here we calculate the absolute value of the input.
- The third line is indented.
- The fourth line is not, it is always executed.

Example



```
p2.1.py - /Users/thomasschwarz/Google Drive/AATeaching/Ahmedabad/Solutions/...  
a = int(input("a number, please: "))  
if a <= 0:  
    a = -a  
    print("This is a negative number")  
print("Thanks for using our program." )
```

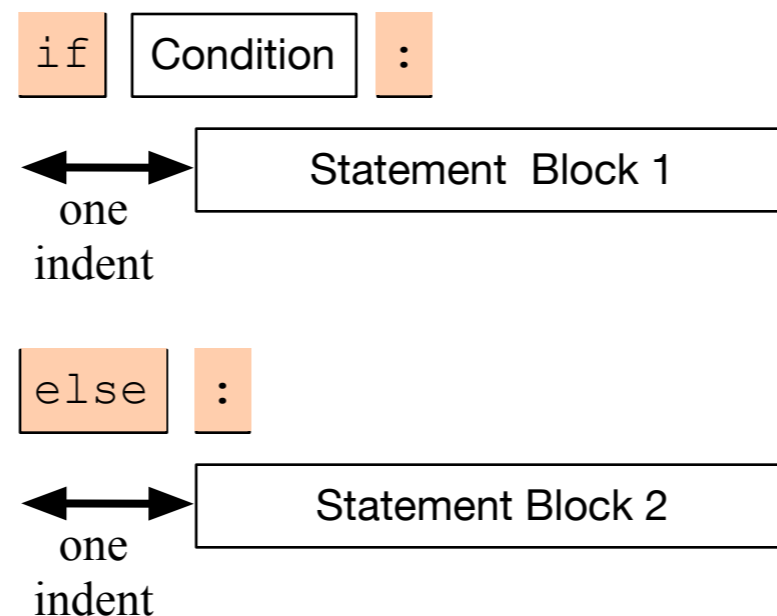
- Here, lines 3 and 4 are indented and are executed if the input is a negative integer.
- The last line, line 5, is always executed since it is not part of the if-statement

Alternative statements

- Very often, we use a condition to decide which one of several branches of execution to pursue.
- The else-statement after the indented block of an if-statement creates an alternative route through the program.

Alternative Statements

- The if-else statement has the following form:



- We add the keyword `else`, followed by a colon
- Then add a second set of statements, indented once
- If the condition is true, then Block 1 is executed, otherwise, Block 2.

Examples

- I can test equality by using the double = sign.
- To check whether a number n is even, I take the remainder modulo 2 and then compare with 0.

```
p2.2.py - /Users/thomasschwarz/Google Drive/AATeaching/Ahmedabad/Solutions/...
number = int(input("Enter a number: "))
if number%2 ==0:
    print("The number is even.")
    print("Its square is", number**2)
else:
    print("The number is odd.")
    print("Its square-root is", number**0.5)
|
```

Alternative Statements

- Often, we have more than two alternative streams of execution.
- Instead of nesting if expressions, we can just use the keyword “elif”, a contraction of else if.

Alternative Statements

`if` `Condition 1` `:`

↔ `Statement Block 1`
one indent

`elif` `Condition 2` `:`

↔ `Statement Block 2`
one indent

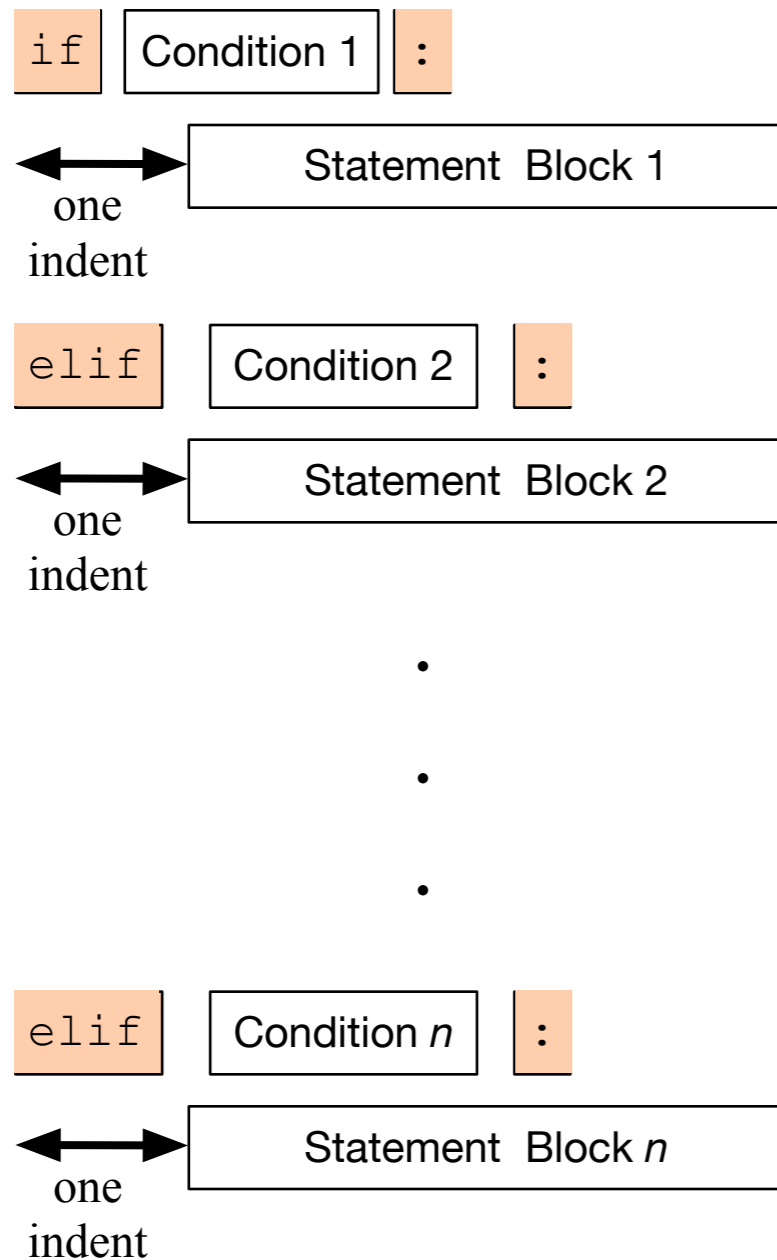
.
. .
.

`else` `:`

↔ `Statement Block n`
one indent

- One of the statement blocks is going to be executed
- The else block contains the default action, if none of the conditions are true

Alternative Statements



- Here, there is no else statement, so it is possible that none of the blocks is executed.

Examples

- Categorization of temperatures

```
if temperature < -25.0:
    feeling = "arctic"
elif temperature < -10.0:
    feeling = "Wisconsin in winter"
elif temperature < 0.0:
    feeling = "freezing"
elif temperature < 15.0:
    feeling = "cold"
elif temperature < 25.0:
    feeling = "comfortable"
elif temperature < 35.0:
    feeling = "hot"
elif temperature < 45.0:
    feeling = "Ahmedabad in the summer"
else:
    feeling = "hot as in hell"
```

Boolean Expressions

- Nested loops to implement decision tree:

```
if x<10:  
    if y<3:  
        if x<2:  
            result=0  
        else:  
            result=1  
    else:  
        result=0  
else:  
    if y<2:  
        result=1  
    else result=0
```

