

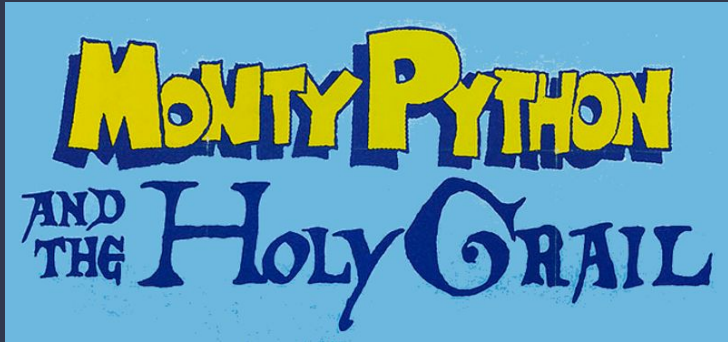
Programming in Python

An introduction
by William Grimes

Overview

- History of Python programming
- Why use Python?
- How to install Python
- Where to run Python?
- Course contents:
 - Data types
 - Boolean conditionals (if/else)
 - Loops
 - Dictionaries
 - Functions
 - Classes and OOP
 - Useful libraries (numpy, matplotlib)

History of Python



- Conceptualised in 1980s by a Dutch programmer Guido van Rossum
- Named after Monty Python, the British TV show
- Python versions:
 - 1990: Python
 - 2000: Python 2.0
 - 2008: Python 3.0

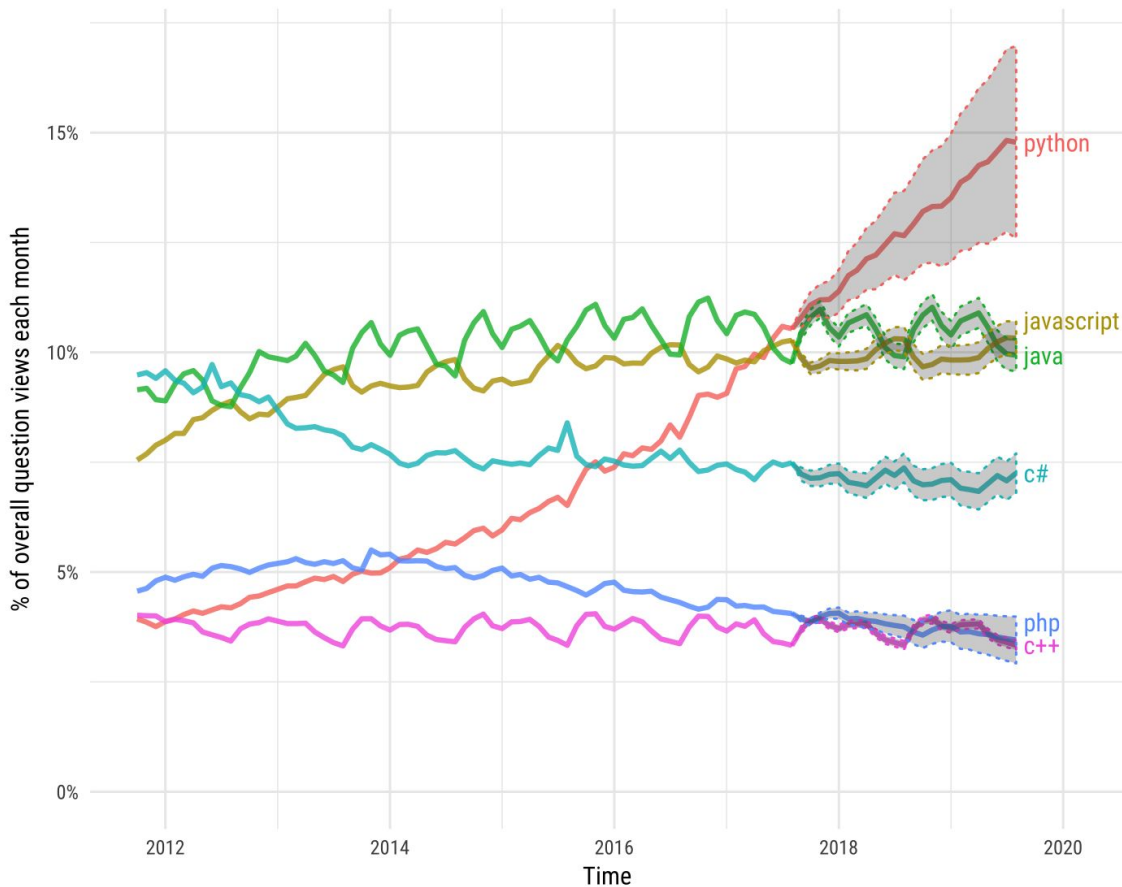
Python is popular



- Python was the most visited tag on Stack Overflow within high-income nations
- Very versatile uses:
 - Finance
 - Sciences
 - Web and internet development
 - Desktop applications
 - Business

Projections of future traffic for major programming languages

Future traffic is predicted with an STL model, along with an 80% prediction interval.



Why Python?



- Powerful, open-source, and free
- Efficient high level data structures, allowing you to do more in fewer lines of code.
- Both object-oriented and imperative
- It is an interpreted language, rather than compiled, so easier to write code, but execution is slower.
- Fun and easy to be productive



Python installation

- Python 2 vs Python 3
 - Different syntax
 - `print 'hello'`
 - `print('hello')`
 - Not backwards compatible
 - Python 3 is supported now
- Anaconda is a Python distribution
 - Bundled with useful packages
 - Includes a package manager
 - Includes Jupyter Notebooks
- Download Anaconda Python 3.6

<https://www.anaconda.com/download>

```
4 >>> x = {'a': 1, 'b': 2}
5 >>> y = {'b': 3, 'c': 4}
6
7 >>> z = {**x, **y}
8
9 >>> z
10 {'c': 4, 'a': 1, 'b': 3}
```

Where to run Python?

```
4 >>> x = {'a': 1, 'b': 2}
5 >>> y = {'b': 3, 'c': 4}
6
7 >>> z = {**x, **y}
8
9 >>> z
10 {'c': 4, 'a': 1, 'b': 3}
```

- Main ways to access python:

1. **Python Shell**

- terminal/command prompt

2. **Jupyter Notebooks**

- Graphical procedural approach, useful for experimenting

3. **IDE : Integrated Development Environment**

- A software that helps you build code

1) Python Shell

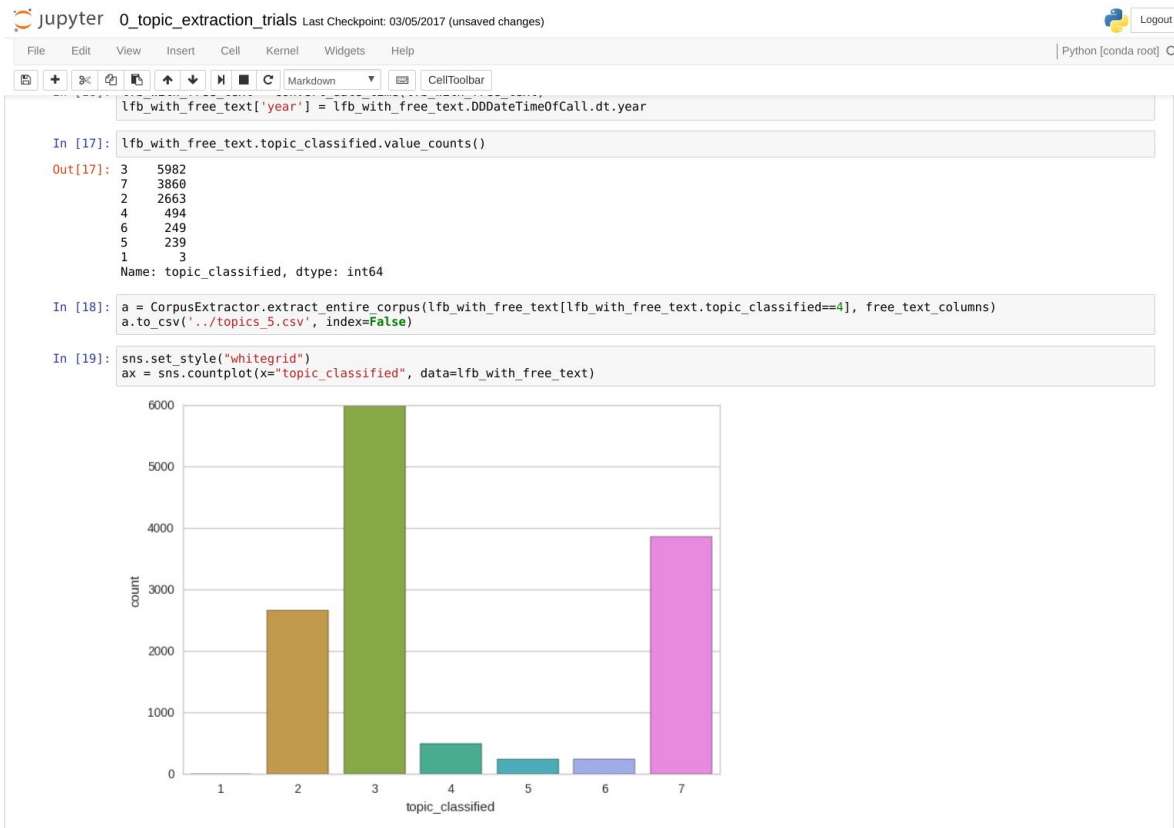


```
~ python
Python 3.5.3 | packaged by conda-forge | (default, May 12 2017, 15:07:14)
[GCC 4.8.2 20140120 (Red Hat 4.8.2-15)] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("this is the python shell")
this is the python shell
>>> python course
```

File Edit View Insert Format Side Arrange Tools Table Add-ons Help All changes saved

- Also called: shell, terminal, command prompt, interpreter, console
- A basic python interface, without the bells and whistles
- Activate in a terminal window by typing: ***python***
- Or try it here: <https://www.python.org/shell/>

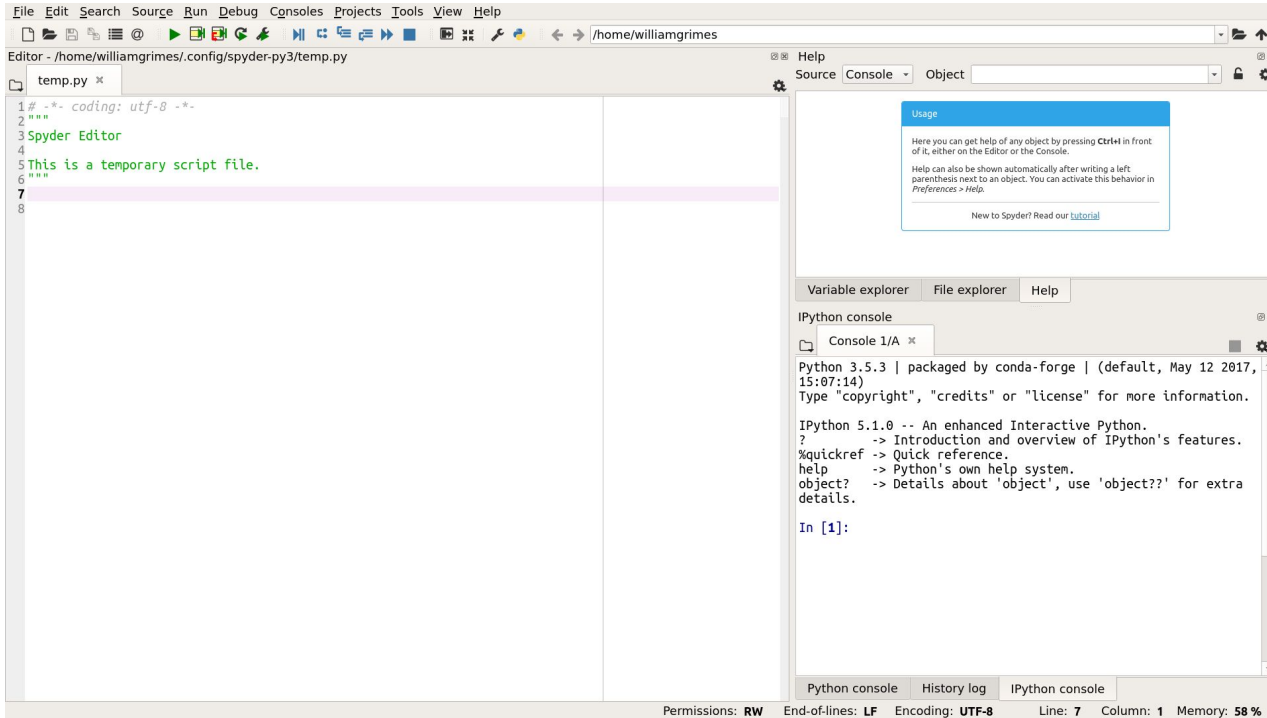
2) IPython and Jupyter Notebooks



- Shell for interactive Python
- Like a scientific laboratory notebook, useful for learning and experimenting
- Jupyter Notebook:
 - Browser based
 - Data visualisation
 - Markdown
 - Live code
- Jupyter comes with Anaconda

3) Integrated Development Environment (IDE)≡

- Includes:
 - Area to edit code
 - Python console
 - Functions to run, debug, and highlight errors
- IDEs:
 - Spyder
 - PyCharm
 - Rodeo
 - ...
- Spyder comes with Anaconda



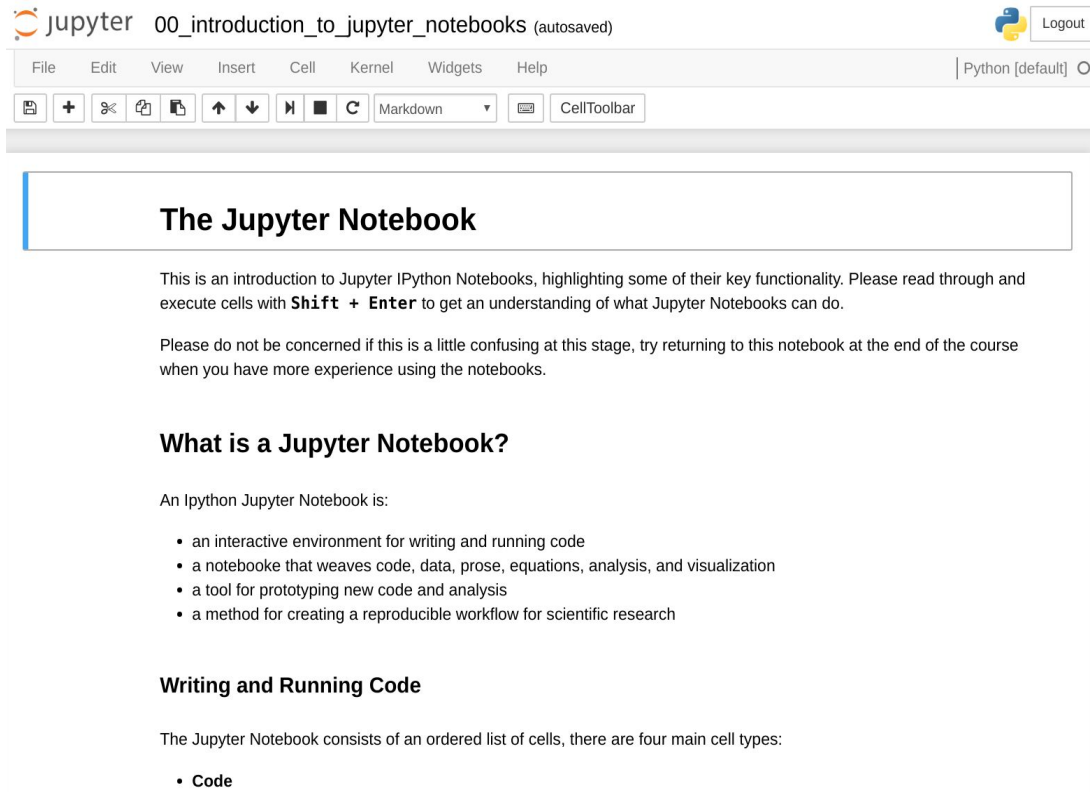
Course overview

- Contents
 - Programming principles:
 - Data types
 - Boolean conditionals (if/else)
 - Loops
 - Dictionaries
 - Functions
 - Classes and OOP
 - Useful libraries
 - Numpy
 - matplotlib
- 14 Jupyter notebooks
- Extra notebooks for more detail
- Walkthroughs with examples

https://github.com/williamgrimes/python_in_a_notebook

Lesson 0:

Jupyter.ipynb



jupyter 00_introduction_to_jupyter_notebooks (autosaved)

File Edit View Insert Cell Kernel Widgets Help Python [default]

Markdown CellToolbar

The Jupyter Notebook

This is an introduction to Jupyter IPython Notebooks, highlighting some of their key functionality. Please read through and execute cells with **Shift + Enter** to get an understanding of what Jupyter Notebooks can do.

Please do not be concerned if this is a little confusing at this stage, try returning to this notebook at the end of the course when you have more experience using the notebooks.

What is a Jupyter Notebook?

An IPython Jupyter Notebook is:

- an interactive environment for writing and running code
- a notebook that weaves code, data, prose, equations, analysis, and visualization
- a tool for prototyping new code and analysis
- a method for creating a reproducible workflow for scientific research

Writing and Running Code

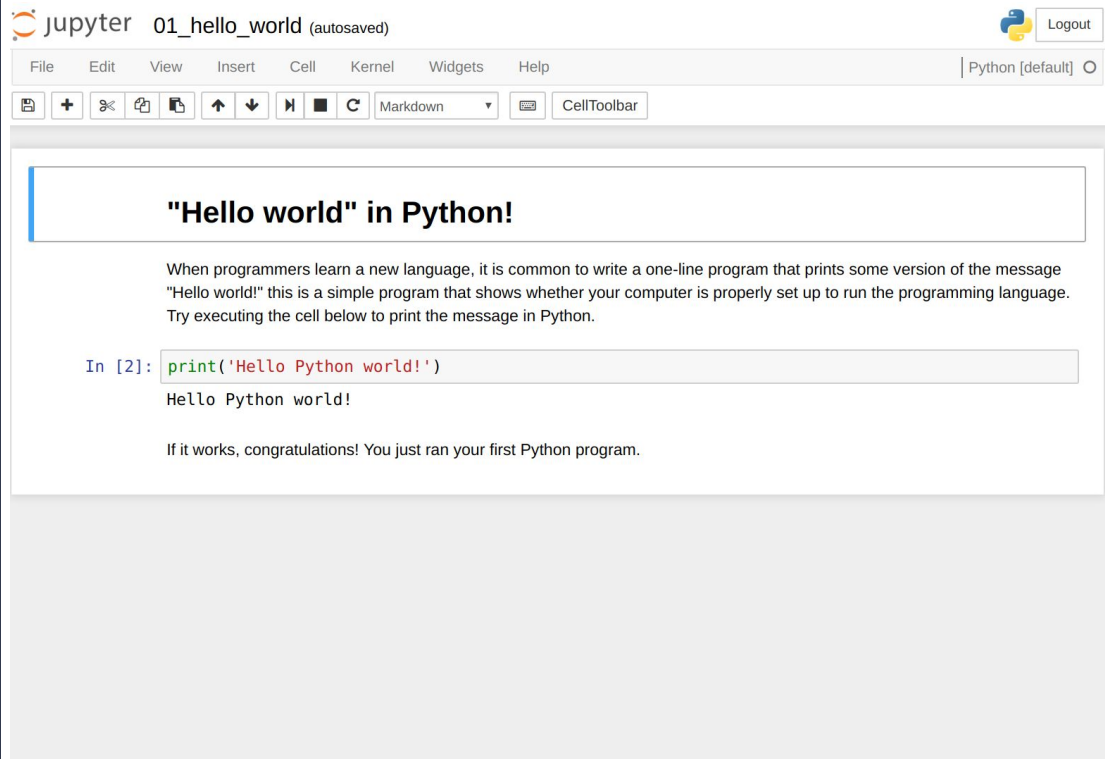
The Jupyter Notebook consists of an ordered list of cells, there are four main cell types:

- Code

- How to work in Jupyter notebooks
- Demonstrates notebook functionality

Lesson 1:

“Hello World”

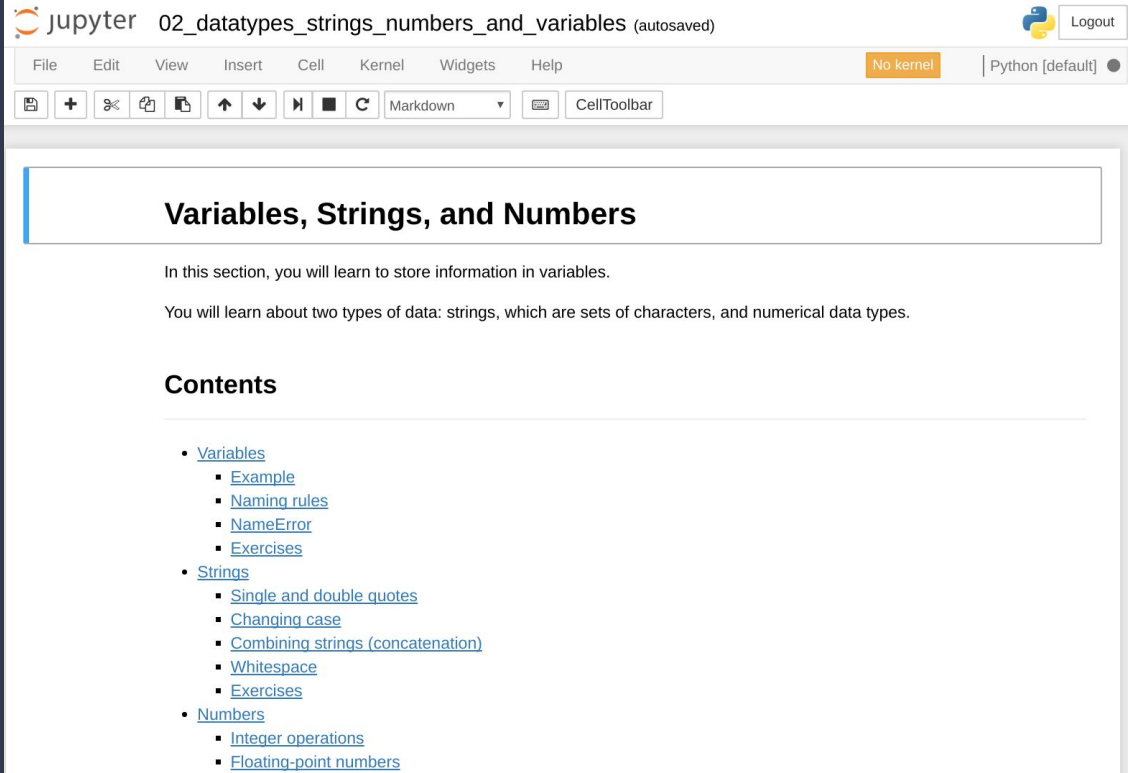


The screenshot shows a Jupyter Notebook window titled "01_hello_world (autosaved)". The interface includes a top menu bar with options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for saving, adding cells, and running code. The main content area has a title "Hello world" in Python! followed by an explanatory paragraph: "When programmers learn a new language, it is common to write a one-line program that prints some version of the message 'Hello world!' this is a simple program that shows whether your computer is properly set up to run the programming language. Try executing the cell below to print the message in Python." Below this is a code cell labeled "In [2]:" containing the code `print('Hello Python world!')`. The output of the cell is "Hello Python world!". A final paragraph states: "If it works, congratulations! You just ran your first Python program."

- Simple program to demonstrate basic syntax
- Customary when learning a new language

Lesson 2:

Data types and variables



The screenshot shows a Jupyter Notebook interface. The top bar includes the Jupyter logo, the file name '02_datatypes_strings_numbers_and_variables (autosaved)', and a 'Logout' button. Below the top bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. To the right of the menu bar is a 'No kernel' status indicator and a 'Python [default]' language selector. Below the menu bar is a toolbar with icons for file operations, cell navigation, and execution. The main content area displays a lesson page with the title 'Variables, Strings, and Numbers'. The text on the page states: 'In this section, you will learn to store information in variables. You will learn about two types of data: strings, which are sets of characters, and numerical data types.' Below this text is a 'Contents' section with a list of links: 'Variables' (with sub-links: 'Example', 'Naming rules', 'NameError', 'Exercises'), 'Strings' (with sub-links: 'Single and double quotes', 'Changing case', 'Combining strings (concatenation)', 'Whitespace', 'Exercises'), and 'Numbers' (with sub-links: 'Integer operations', 'Floating-point numbers').

Variables, Strings, and Numbers

In this section, you will learn to store information in variables.

You will learn about two types of data: strings, which are sets of characters, and numerical data types.

Contents

- [Variables](#)
 - [Example](#)
 - [Naming rules](#)
 - [NameError](#)
 - [Exercises](#)
- [Strings](#)
 - [Single and double quotes](#)
 - [Changing case](#)
 - [Combining strings \(concatenation\)](#)
 - [Whitespace](#)
 - [Exercises](#)
- [Numbers](#)
 - [Integer operations](#)
 - [Floating-point numbers](#)

- Variables hold a value
 - Can be of multiple data types
 - string = "this is a string"
 - integer = 20395

Lesson 3:

Lists, tuples, and sets

- `a_list = ['item_0', 'item_1', 2]`
 - Creation
 - Sorting
 - Looping
 - Slicing
 - List comprehension
- `a_tuple = (0, 'item_1', 12.34)`
 - fixed length
- `a_set = {0, 1, 2, 3}`
 - Membership checking
 - Intersection and union

Lesson 4:

If statements and conditional logic

```
dogs = []

if len(dogs) >= 5:
    print("Lets start a dog hostel!")
elif len(dogs) >= 3:
    print("Wow, we have a lot of dogs here!")
elif len(dogs) >= 1:
    print("Okay, some dogs.")
else:
    print("I wish we had a dog here.")
```

- If, else, else if
- Boolean conditionals
- Test for a condition, and perform action
- N.B. indentation matters in Python

Lesson 5:

Loops for While infinite

```
edibles = ["ham", "spam", "eggs", "nuts"]
for food in edibles:
    if food == "spam":
        print("No more spam please!")
        continue
    print("Great, delicious " + food)
else:
    print("I am so glad: No spam!")
print("Finally, I finished stuffing myself")
```

Output:

```
Great, delicious ham
No more spam please!
Great, delicious eggs
Great, delicious nuts
I am so glad: No spam!
Finally, I finished stuffing myself
```

Lesson 6:

Dictionaries

- Unordered sets
- Key-value pairs
- Implemented as hash tables

```
city_population = {"New York City":8550405,  
                  "Los Angeles":3971883,  
                  "Toronto":2731571,  
                  "Vancouver":631486,  
                  "Boston":667137}
```

Output:

```
>>> city_population["New York City"]  
8550405
```

Lesson 7:

Functions

- Functions are code that is grouped together
 - Subroutines
 - Routines
 - Procedures
 - Methods
 - Subprograms
 - ...

- Allows code to be reused

```
def return_sum(x,y):  
    c = x + y  
    return c
```

```
res = return_sum(4,5)
```

Output:

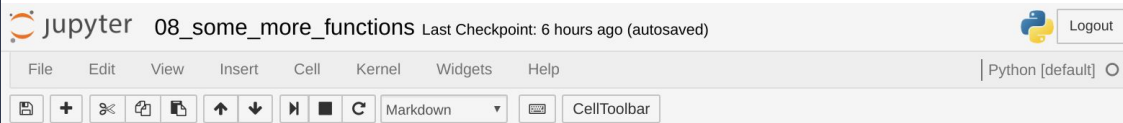
```
>>> print(res)  
9
```

Lesson 8:

More functions

Advanced topic

- The return statement
- If this is too complex return to it later
- Function arguments
 - `*args`
 - `**kwargs`



More Functions

Earlier we learned the most bare-boned versions of functions. In this section we will learn more general concepts about functions, such as how to use functions to return values, and how to pass different kinds of data structures between functions.

Contents

- [Default argument values](#)
 - [Exercises](#)
- [Positional arguments](#)
 - [Exercises](#)
- [Keyword arguments](#)
 - [Mixing positional and keyword arguments](#)
 - [Exercises](#)
- [Accepting an arbitrary number of arguments](#)
 - [Accepting a sequence of arbitrary length](#)

Lesson 9:

Classes and Object-Oriented Programming (OOP)

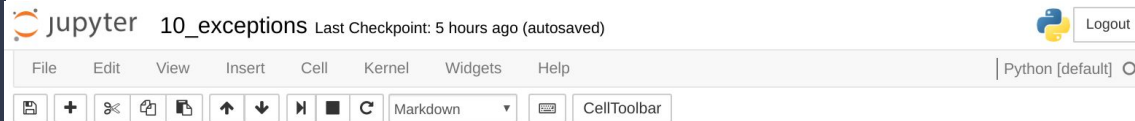
Advanced topic

- Classes combine information and behaviour in objects, aka variables and methods
- A very powerful tool in Python (and other languages e.g. C++, C#, Java, ...)
- Four major principles of OOP:
 - Encapsulation
 - Data Abstraction
 - Polymorphism
 - Inheritance
- Don't worry if this seems abstract at the moment read the notebook anyway. You may not need to use OOP programming, but should be aware of what it is.

Lesson 10:

Exceptions

- Exception is an error during execution
- Modify flow through a program
- Python has built-in support for exception handling
- You may not need this in your code yet but good to be aware how to handle errors.



Exceptions

Exceptions which are events that can modify the *flow* of control through a program.

In Python, exceptions are triggered automatically on errors, and they can be triggered and intercepted by your code.

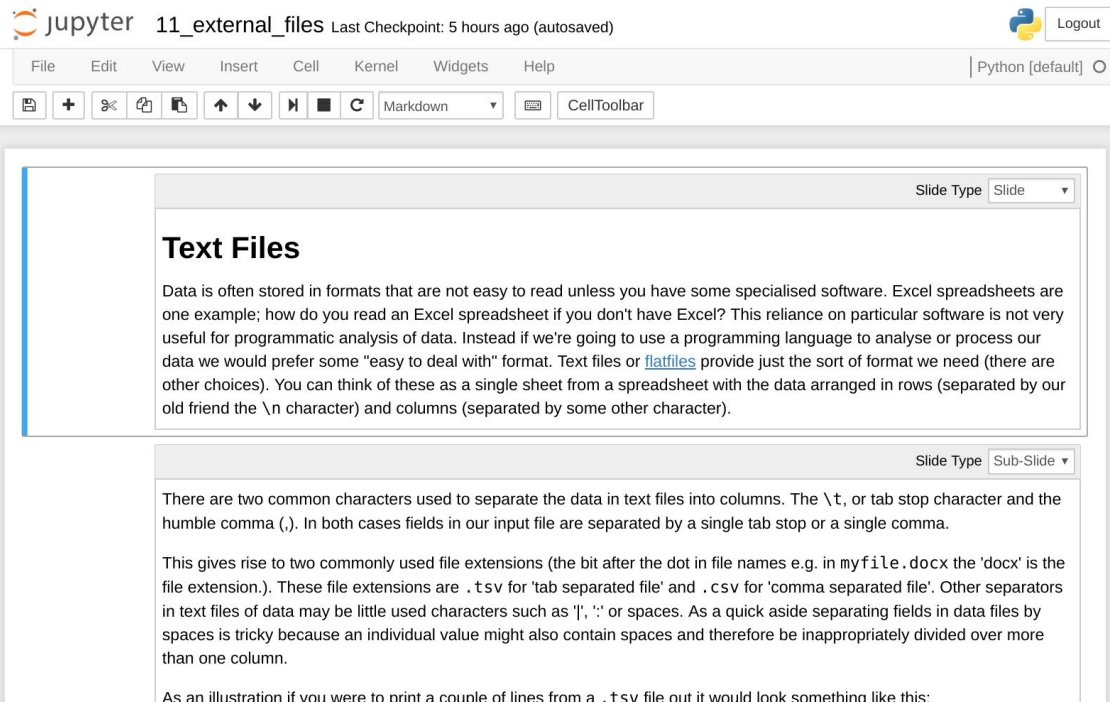
They are processed by **four** statements we'll study in this notebook, the first of which has two variations (listed separately here) and the last of which was an optional extension until Python 2.6 and 3.0:

- try/except:
 - Catch and recover from exceptions raised by Python, or by you
- try/finally:
 - Perform cleanup actions, whether exceptions occur or not

Lesson 11:

External files

- How to read files in and out
- Handling data
- Also look at Numpy and Pandas libraries



The screenshot shows a JupyterLab interface with a presentation slide titled "Text Files". The slide content discusses data formats and file extensions. The JupyterLab header shows the file name "11_external_files" and a "Last Checkpoint: 5 hours ago (autosaved)" status. The top menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The top toolbar includes icons for saving, undo, redo, and other editing functions. The slide content is as follows:

Text Files

Data is often stored in formats that are not easy to read unless you have some specialised software. Excel spreadsheets are one example; how do you read an Excel spreadsheet if you don't have Excel? This reliance on particular software is not very useful for programmatic analysis of data. Instead if we're going to use a programming language to analyse or process our data we would prefer some "easy to deal with" format. Text files or [flatfiles](#) provide just the sort of format we need (there are other choices). You can think of these as a single sheet from a spreadsheet with the data arranged in rows (separated by our old friend the `\n` character) and columns (separated by some other character).

There are two common characters used to separate the data in text files into columns. The `\t`, or tab stop character and the humble comma (,). In both cases fields in our input file are separated by a single tab stop or a single comma.

This gives rise to two commonly used file extensions (the bit after the dot in file names e.g. in `myfile.docx` the 'docx' is the file extension.). These file extensions are `.tsv` for 'tab separated file' and `.csv` for 'comma separated file'. Other separators in text files of data may be little used characters such as `'|'`, `':'` or spaces. As a quick aside separating fields in data files by spaces is tricky because an individual value might also contain spaces and therefore be inappropriately divided over more than one column.

As an illustration if you were to print a couple of lines from a `.tsv` file out it would look something like this:

Lesson 12:

Numpy library

- Libraries extend core python functionality
 - see `bonus_importing_modules.ipynb`
- `import numpy as np`
- Numeric Python
 - Matrices/arrays
 - Vectorised
 - Implemented in C for performance

Jupyter 12_numpy_library Last Checkpoint: 4 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Python [conda root] O

Save + Close Copy Paste Up Down Run Stop Refresh Markdown CellToolbar

Numpy - multidimensional data arrays

```
In [1]: %matplotlib inline
import matplotlib.pyplot as plt
```

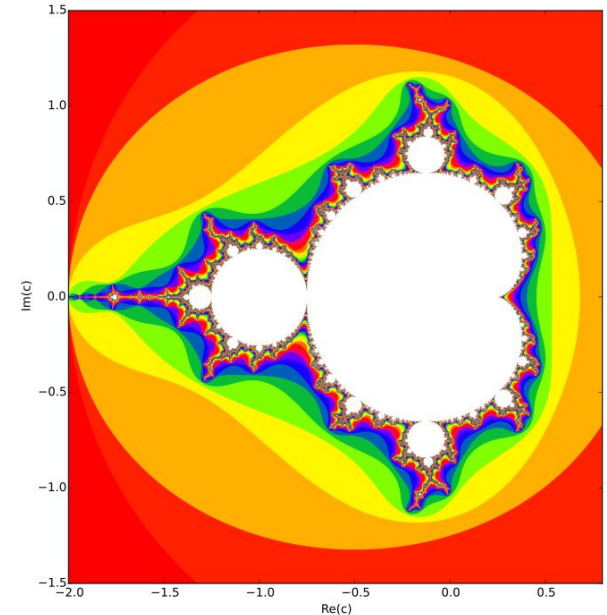
Introduction

The numpy package (module) is used in almost all numerical computation using Python. It is a package that provide high-

Lesson 13:

Matplotlib library

- Primary python plotting library
- Various other plotting libraries
 - Bokeh
 - Plotly
 - Seaborn
 - ggplot
 - ...



Bonus lessons:

Coding Style

Databases and
persistence

Importing
modules

The Zen of
Python

- Recommended bonus lessons:
 - Style guide for python
 - Importing modules
- Bonus lessons for fun:
 - Databases and persistence
 - The Zen of Python
- Other libraries to explore:
 - SciPy
 - Pandas



Summary

- History of Python programming
- Why use Python?
- How to install Python
- Where to run Python?
- Course contents:
 - Data types
 - Boolean conditionals (if/else)
 - Loops
 - Dictionaries
 - Functions
 - Classes and OOP
 - Useful libraries (numpy, matplotlib)

Links and Resources:

https://github.com/williamgrimes/python_in_a_notebook

- Anaconda Install
 - <https://www.anaconda.com/download>
- Online interactive python shell:
 - <https://www.python.org/shell/>
- Extra learning resources:
 - <https://www.learnpython.org/>
 - <https://www.python-course.eu/>
 - <https://wiki.python.org/moin/BeginnersGuide/NonProgrammers>
 - <http://pythontutor.com/>
- Python documentation:
 - <https://docs.python.org/3/>
- Python style guide:
 - <https://www.python.org/dev/peps/pep-0008/>
- Google python class videos:
 - <https://www.youtube.com/watch?v=tKTZoB2Vjuk>