Scientific Computing using Python

Swaprava Nath

Dept. of CSE
IIT Kanpur

mini-course webpage: https://swaprava.wordpress.com/a-short-course-on-python/

Disclaimer: the contents of this lecture series are based on several texts and online resources

Outline of the Talk

Outline of the Talk

What is Python?

- Python is a general purpose programming language conceived in 1989 by Dutch programmer **Guido van Rossum**

What is Python?

- Python is a general purpose programming language conceived in 1989 by Dutch programmer **Guido van Rossum**
- Python is free and open source, with development coordinated through the Python Software Foundation, `www.python.org`

What is Python?

- Python is a general purpose programming language conceived in 1989 by Dutch programmer **Guido van Rossum**
- Python is free and open source, with development coordinated through the Python Software Foundation, `www.python.org`
- Python has experienced rapid adoption in the last decade, and is now one of the most popular programming languages

What is Python?

- Python is a general purpose programming language conceived in 1989 by Dutch programmer **Guido van Rossum**
- Python is free and open source, with development coordinated through the Python Software Foundation, `www.python.org`
- Python has experienced rapid adoption in the last decade, and is now one of the most popular programming languages
- Typical application domains:
  - scientific computing – simulations
  - web development
  - graphical user interfaces
  - games
  - data processing

What is Python?

- Python is a general purpose programming language conceived in 1989 by Dutch programmer **Guido van Rossum**
- Python is free and open source, with development coordinated through the Python Software Foundation, `www.python.org`
- Python has experienced rapid adoption in the last decade, and is now one of the most popular programming languages
- Typical application domains:
    - scientific computing – simulations
    - web development
    - graphical user interfaces
    - games
    - data processing
- Commercial usage: Google, Dropbox, Reddit, YouTube, Walt Disney Animations, Cisco, Intel etc.

What is Python?

- Python is a general purpose programming language conceived in 1989 by Dutch programmer **Guido van Rossum**
- Python is free and open source, with development coordinated through the Python Software Foundation, `www.python.org`
- Python has experienced rapid adoption in the last decade, and is now one of the most popular programming languages
- Typical application domains:
    - scientific computing – simulations
    - web development
    - graphical user interfaces
    - games
    - data processing
- Commercial usage: Google, Dropbox, Reddit, YouTube, Walt Disney Animations, Cisco, Intel etc.
- Academic usage: companion of many courses – for 101 CS courses or for supportive computing

Relative popularity of Python

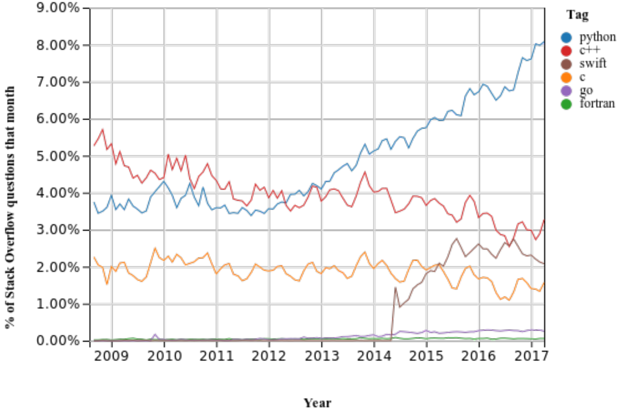


Image courtesy: www.quantecon.org

The plot, produced using Stack Overflow Trends, shows one measure of the relative popularity of Python

Features

- High-level language, useful for rapid development

Features

- High-level language, useful for rapid development
- Relatively small core language – supported by many libraries

Features

- High-level language, useful for rapid development
- Relatively small core language – supported by many libraries
- Multiparadigm language – the style of building the structure and elements of computer programs can be
    - **procedural** – based on routines/subroutines

Features

- High-level language, useful for rapid development
- Relatively small core language – supported by many libraries
- Multiparadigm language – the style of building the structure and elements of computer programs can be
  - **procedural** – based on routines/subroutines
  - **object-oriented** – created on abstract objects

Features

- High-level language, useful for rapid development
- Relatively small core language – supported by many libraries
- Multiparadigm language – the style of building the structure and elements of computer programs can be
  - **procedural** – based on routines/subroutines
  - **object-oriented** – created on abstract objects
  - **functional** – treats computation as the evaluation of mathematical functions

Features

- High-level language, useful for rapid development
- Relatively small core language – supported by many libraries
- Multiparadigm language – the style of building the structure and elements of computer programs can be
  - **procedural** – based on routines/subroutines
  - **object-oriented** – created on abstract objects
  - **functional** – treats computation as the evaluation of mathematical functions
- Usually implemented as interpreted as opposed to compiled (e.g. in C)

Features

- High-level language, useful for rapid development
- Relatively small core language – supported by many libraries
- Multiparadigm language – the style of building the structure and elements of computer programs can be
  - **procedural** – based on routines/subroutines
  - **object-oriented** – created on abstract objects
  - **functional** – treats computation as the evaluation of mathematical functions
- Usually implemented as interpreted as opposed to compiled (e.g. in C)
- Syntax and design of a python code makes it easier to read, debug, and develop

Compiled vs Interpreted

- The compiler takes a program as a whole and translates it, but interpreter translates a program statement by statement

Compiled vs Interpreted

- The compiler takes a program as a whole and translates it, but interpreter translates a program statement by statement
- Intermediate code or target code is generated in case of a compiler – interpreter doesn't create intermediate code

Compiled vs Interpreted

- The compiler takes a program as a whole and translates it, but interpreter translates a program statement by statement
- Intermediate code or target code is generated in case of a compiler – interpreter doesn't create intermediate code
- Compiler is comparatively faster than Interpreter as the compiler takes the whole program at one go while interpreter compiles each line of code

Compiled vs Interpreted

- The compiler takes a program as a whole and translates it, but interpreter translates a program statement by statement
- Intermediate code or target code is generated in case of a compiler – interpreter doesn't create intermediate code
- Compiler is comparatively faster than Interpreter as the compiler takes the whole program at one go while interpreter compiles each line of code
- Compiler presents all errors concurrently, and it's difficult to detect the errors – in contrast, interpreter display errors of each statement one by one, and it's easier to detect errors

Compiled vs Interpreted

- The compiler takes a program as a whole and translates it, but interpreter translates a program statement by statement
- Intermediate code or target code is generated in case of a compiler – interpreter doesn't create intermediate code
- Compiler is comparatively faster than Interpreter as the compiler takes the whole program at one go while interpreter compiles each line of code
- Compiler presents all errors concurrently, and it's difficult to detect the errors – in contrast, interpreter display errors of each statement one by one, and it's easier to detect errors
    - Suitable for very large codes – one can check certain snippets of it without running the whole code again

Compiled vs Interpreted

- The compiler takes a program as a whole and translates it, but interpreter translates a program statement by statement
- Intermediate code or target code is generated in case of a compiler – interpreter doesn't create intermediate code
- Compiler is comparatively faster than Interpreter as the compiler takes the whole program at one go while interpreter compiles each line of code
- Compiler presents all errors concurrently, and it's difficult to detect the errors – in contrast, interpreter display errors of each statement one by one, and it's easier to detect errors
    - Suitable for very large codes – one can check certain snippets of it without running the whole code again
    - Step-by-step execution also helps in identifying errors

Suitability for Scientific Computation

- Availability of relevant libraries

Suitability for Scientific Computation

- Availability of relevant libraries
  - `numpy` – numerical computation, e.g., arrays, matrices, and operations on them

Suitability for Scientific Computation

- Availability of relevant libraries
  - `numpy` – numerical computation, e.g., arrays, matrices, and operations on them
  - `scipy` – mathematical operations, e.g., linear algebra, optimization, integration etc.

Suitability for Scientific Computation

- Availability of relevant libraries
  - `numpy` – numerical computation, e.g., arrays, matrices, and operations on them
  - `scipy` – mathematical operations, e.g., linear algebra, optimization, integration etc.
  - `matplotlib` – all kinds of plotting, 2D and 3D

Suitability for Scientific Computation

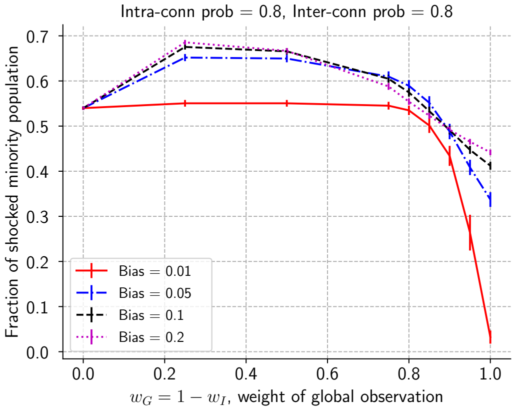
- Availability of relevant libraries
  - `numpy` – numerical computation, e.g., arrays, matrices, and operations on them
  - `scipy` – mathematical operations, e.g., linear algebra, optimization, integration etc.
  - `matplotlib` – all kinds of plotting, 2D and 3D
  - `pandas` – for data handling

Suitability for Scientific Computation

- Availability of relevant libraries
  - `numpy` – numerical computation, e.g., arrays, matrices, and operations on them
  - `scipy` – mathematical operations, e.g., linear algebra, optimization, integration etc.
  - `matplotlib` – all kinds of plotting, 2D and 3D
  - `pandas` – for data handling
- used in data science, machine learning, artificial intelligence, computational biology, computational physics, quantitative economics etc.

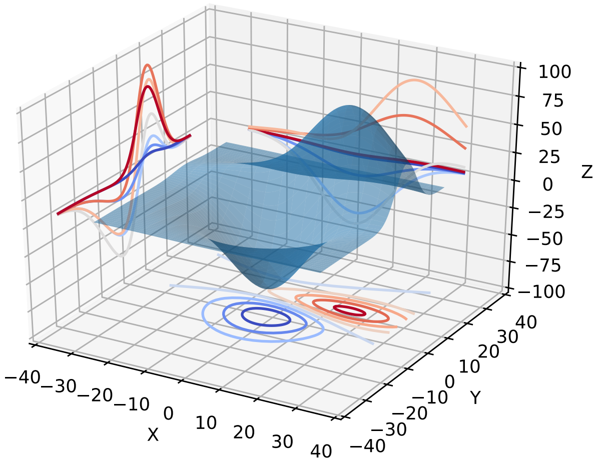
Example in action – 2D plot

- A plot of surprise in the Brexit election



- Used `pandas`, `numpy`, `matplotlib`

Example in action – 3D plot



Setting up Your Python Environment

- Native `python2.7` is available in any linux distribution
- Need to install the libraries

Setting up Your Python Environment

- Native `python2.7` is available in any linux distribution
- Need to install the libraries
- An alternative python distribution is by anaconda – `www.anaconda.com`
- We will use the native python with libraries and an IDE

Setting up Your Python Environment

- Native `python2.7` is available in any linux distribution
- Need to install the libraries
- An alternative python distribution is by anaconda – `www.anaconda.com`
- We will use the native python with libraries and an IDE
- Convenient to use an IDE – integrated development environment
    - `jupyter-notebook`
    - `spyder`

Setting up Your Python Environment

- Native `python2.7` is available in any linux distribution
- Need to install the libraries
- An alternative python distribution is by anaconda – `www.anaconda.com`
- We will use the native python with libraries and an IDE
- Convenient to use an IDE – integrated development environment
    - `jupyter-notebook`
    - `spyder`
- `jupyter-notebook` is useful for testing snippets of code and displaying

Setting up Your Python Environment

- Native `python2.7` is available in any linux distribution
- Need to install the libraries
- An alternative python distribution is by anaconda – `www.anaconda.com`
- We will use the native python with libraries and an IDE
- Convenient to use an IDE – integrated development environment
  - `jupyter-notebook`
  - `spyder`
- `jupyter-notebook` is useful for testing snippets of code and displaying
- `spyder` is useful for a long codebase development
- Examples

Some Introductory Python Programs

- Task 1: finding if a number is even/odd – `if-else` clause
- Task 2: finding the smallest of three numbers
- Task 3: finding if a natural number is prime or not – `while` loop and `for` loop

- Notice the indentation and absence of any braces or brackets
- Makes the code clutter-free and more readable

More Examples of Loops and Conditionals

- Finding square root of a non-negative integer n – handling exceptions

More Examples of Loops and Conditionals

- Finding square root of a non-negative integer n – handling exceptions

$$x_{t+1} = \frac{1}{2}\left(x_t + \frac{n}{x_t}\right)$$

More Examples of Loops and Conditionals

- Finding square root of a non-negative integer n – handling exceptions

$$x_{t+1} = \frac{1}{2}\left(x_t + \frac{n}{x_t}\right)$$

- Finding r-th root of any non-negative number n – bisection method

More Examples of Loops and Conditionals

- Finding square root of a non-negative integer n – handling exceptions

$$x_{t+1} = \frac{1}{2}\left(x_t + \frac{n}{x_t}\right)$$

- Finding r-th root of any non-negative number n – bisection method

$$h = n, l = 0, guess = \frac{h+l}{2}$$
$$\texttt{if} \;\; guess^r > n : h = guess$$
$$\texttt{else} \;\; : l = guess$$

More Examples of Loops and Conditionals

- Finding square root of a non-negative integer n – handling exceptions

$$x_{t+1} = \frac{1}{2}\left(x_t + \frac{n}{x_t}\right)$$

- Finding r-th root of any non-negative number n – bisection method

$$h = \max\{1, n\}, l = 0, guess = \frac{h+l}{2}$$

```
if  $guess^r > n$ : $h = guess$
    else  : $l = guess$
```

More Examples of Loops and Conditionals

- Finding square root of a non-negative integer n – handling exceptions

$$x_{t+1} = \frac{1}{2}\left(x_t + \frac{n}{x_t}\right)$$

- Finding r-th root of any non-negative number n – bisection method

$$h = \max\{1, n\}, l = 0, guess = \frac{h+l}{2}$$
$$\texttt{if } guess^r > n : h = guess$$
$$\texttt{else } : l = guess$$

- Generalization of the square-root finding algorithm – Newton-Raphson method for finding a real root of a polynomial f

More Examples of Loops and Conditionals

- Finding square root of a non-negative integer n – handling exceptions

$$x_{t+1} = \frac{1}{2}\left(x_t + \frac{n}{x_t}\right)$$

- Finding r-th root of any non-negative number n – bisection method

$$h = \max\{1, n\}, l = 0, guess = \frac{h+l}{2}$$

$$\texttt{if}\ \ guess^r > n : h = guess$$

$$\texttt{else}\ \ : l = guess$$

- Generalization of the square-root finding algorithm – Newton-Raphson method for finding a real root of a polynomial f

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

Python Standard Data Types

- Python data types: mutable and immutable
- Primitive data types: `int, float, bool`
- Lists, Tuples, Strings
- Dictionaries, Sets

Python Standard Data Types

- Python data types: mutable and immutable
- Primitive data types: `int, float, bool`
- Lists, Tuples, Strings
- Dictionaries, Sets

Functions

- Example of a function – the root finding algorithm as a function
- Example of recursion of a function – checking a palindrome
- **Exercise**: solve the ‘Tower of Hanoi’ problem using recursion