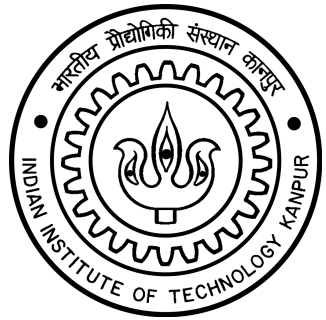
Scientific Computing using Python



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

1. Part 1: Preliminaries of Python
2. Part 2: Scientific Libraries
3. Part 3: Object Oriented Programming

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

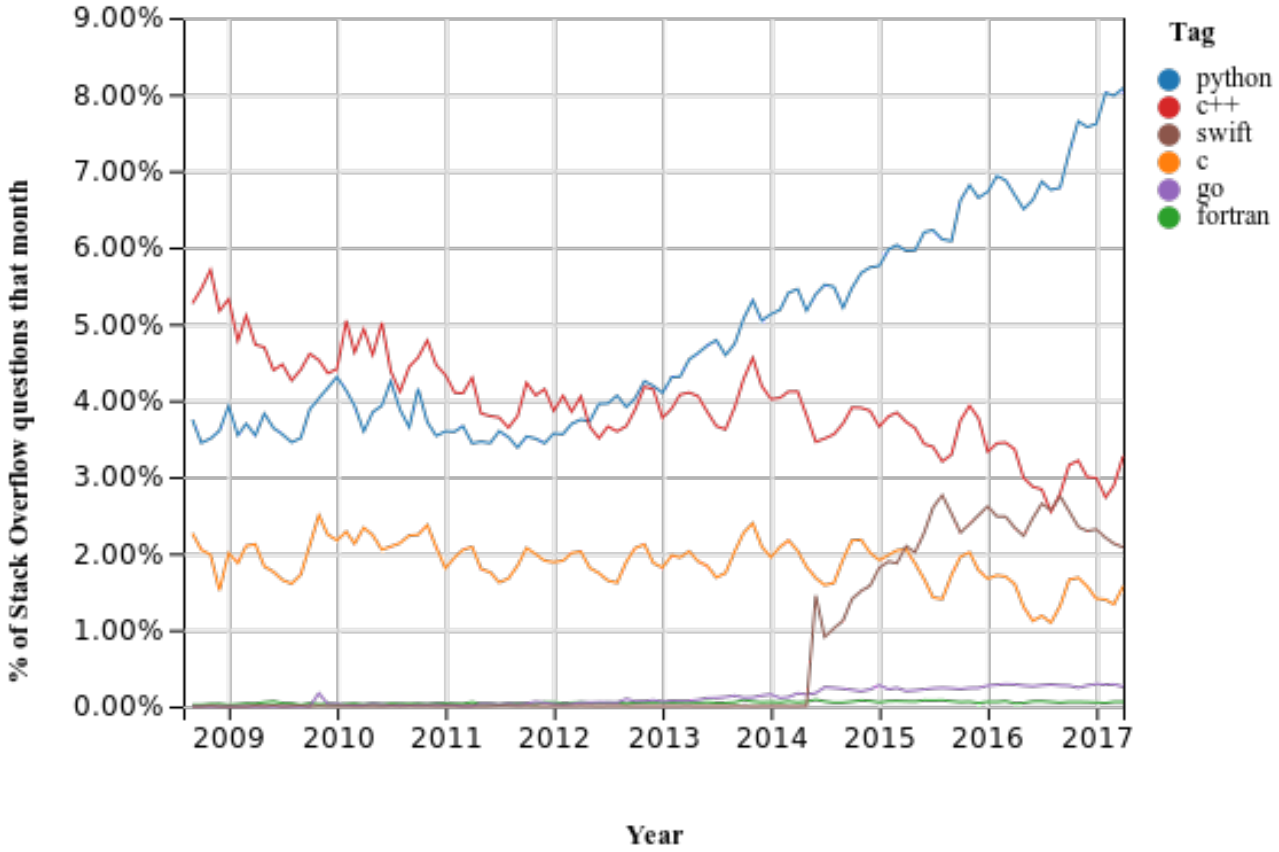


Image courtesy: www.quantecon.org

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

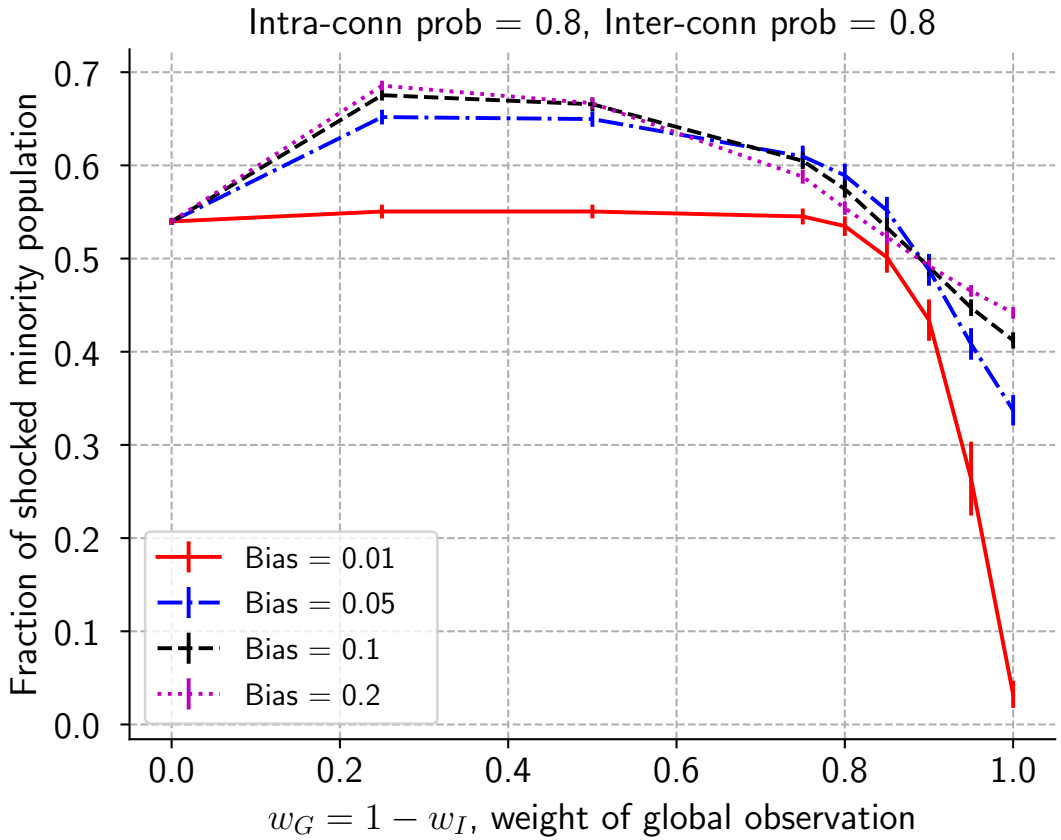
- Availability of relevant libraries
  - `numpy` – numerical computation, e.g., arrays, matrices, and operations on them
  - `scipy` – mathematical operations, e.g., linear algebra, optimization, integration etc.
  - `matplotlib` – all kinds of plotting, 2D and 3D
  - `pandas` – for data handling

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.

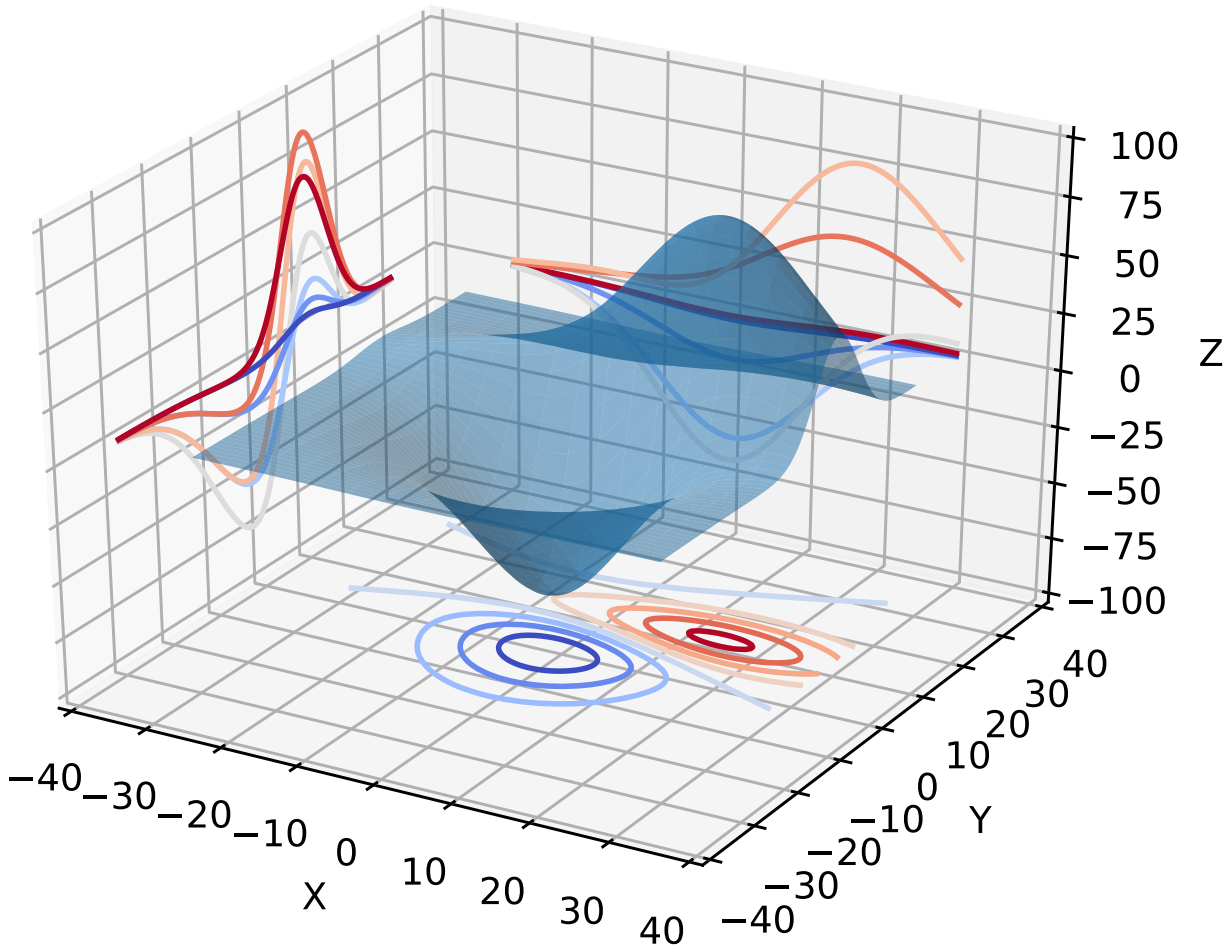
Example in action – 2D plot

- 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

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

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

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

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

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

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

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

- 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

Outline of the Talk

1. Part 1: Preliminaries of Python
2. Part 2: Scientific Libraries
3. Part 3: Object Oriented Programming

Numerical Computation: `numpy`

- Goal: faster array processing

Numerical Computation: `numpy`

- Goal: faster array processing
  - Example: create an array with million entries and process them – add, compute average

Numerical Computation: `numpy`

- Goal: faster array processing
  - Example: create an array with million entries and process them – add, compute average
  - Classic python must use loop which is much slower than `C` or `Fortran`
  - `numpy` does precisely that – accesses the optimized `C` or `Fortran` code by sending the *relevant portion* of the python code

Numerical Computation: `numpy`

- Goal: faster array processing
  - Example: create an array with million entries and process them – add, compute average
  - Classic python must use loop which is much slower than `C` or `Fortran`
  - `numpy` does precisely that – accesses the optimized `C` or `Fortran` code by sending the *relevant portion* of the python code
  - the relevant portions are sent in batches – therefore your code should be *vectorized* as much as possible
  - and use `numpy` functions for them

Numerical Computation: `numpy`

- Goal: faster array processing
  - Example: create an array with million entries and process them – add, compute average
  - Classic python must use loop which is much slower than `C` or `Fortran`
  - `numpy` does precisely that – accesses the optimized `C` or `Fortran` code by sending the *relevant portion* of the python code
  - the relevant portions are sent in batches – therefore your code should be *vectorized* as much as possible
  - and use `numpy` functions for them
- A ton of useful functions on arrays
  - `linspace`, `amax`, `argmax`, `ones`, `zeros`, `sum`, `mean`, `var`, `std`, `cumsum`, `cumprod` and many more
  - `random` gives a bunch of random variables – `randn`, `beta`, `binomial`, `dirichlet`, `exponential`

Scientific Tools: `scipy`

- `numpy` $\subset$ `scipy`
- built on top of `numpy` for more focused scientific programming, e.g.,
  - linear algebra
  - numerical integration
  - interpolation
  - optimization
  - distributions and random number generation
  - signal processing
- similar ideas of using python over `C` and `Fortran` subroutines
- particularly useful for statistical methods – `scipy.stats`
- two specific functions – `bisect` and `newton`

Plotting: `matplotlib`

Features:

- high quality 2D and 3D plots
- output in all the usual formats (PDF, PNG, EPS, SVG etc.)
- LaTeX integration
- fine grained control over all aspects of presentation
- animation
- and many more

Some example usage of matplotlib

Data Handling: `pandas`

- Pandas is a package of fast, efficient data analysis tools for Python
- Similar to `numpy` that defines the basic array data type and fundamental operations on arrays
- `pandas` defines fundamental *structures* for working with data, and
- endows them with *methods* that facilitate operations such as
  - reading in data
  - adjusting indices
  - working with dates and time series
  - sorting, grouping, re-ordering, slicing, and general data manipulations
  - dealing with missing value
- Two fundamental *data types*: `series` and `dataframe`
  - `series`: an array of (possibly dissimilar) objects *with generalized index* – not as space consuming as dictionaries
  - `dataframe`: a matrix with generalized index – and various methods for data handling

Outline of the Talk

1. Part 1: Preliminaries of Python
2. Part 2: Scientific Libraries
3. Part 3: Object Oriented Programming

Programming in Practice: Simulations

- We typically use simulations to understand the performance/behavior of a system
- Examples:
  1. manufacturing process simulation
  2. stock market simulation
  3. statistical processes simulation

Programming in Practice: Simulations

- We typically use simulations to understand the performance/behavior of a system
- Examples:
  1. manufacturing process simulation
  2. stock market simulation
  3. statistical processes simulation
- Typical steps of simulation:
  - Create a computational environment that mimics the real world
  - Generate (synthetic) or load data from sources
  - Test hypotheses

Programming in Practice: Simulations

- We typically use simulations to understand the performance/behavior of a system
- Examples:
  1. manufacturing process simulation
  2. stock market simulation
  3. statistical processes simulation
- Typical steps of simulation:
  - Create a computational environment that mimics the real world
  - Generate (synthetic) or load data from sources
  - Test hypotheses
- Examples:
  1. simulating a 2D random walk
  2. simulation of a discrete random variable

`object` and `class`

Why do we need an object-oriented code?

- Purpose: reuse a code that we need quite often

`object` and `class`

Why do we need an object-oriented code?

- Purpose: reuse a code that we need quite often
- data and function/method bound together, e.g., a `list`
  - list instance `l`, method `l.append()` or `l.sort()`
  - when the `object` is passed, the methods are passed too

`object` and `class`

Why do we need an object-oriented code?

- Purpose: reuse a code that we need quite often
- data and function/method bound together, e.g., a `list`
  - list instance `l`, method `l.append()` or `l.sort()`
  - when the `object` is passed, the methods are passed too
  - `object` is an **instance** of a `class`

`object` and `class`

Why do we need an object-oriented code?

- Purpose: reuse a code that we need quite often
- data and function/method bound together, e.g., a `list`
  - list instance `l`, method `l.append()` or `l.sort()`
  - when the `object` is passed, the methods are passed too
  - `object` is an **instance** of a `class`
  - example: class `rectangle`, create an object of class rectangle, has parameters/data length and height, function of that data is area, longer side, shorter side, length of diagonal, area of the largest ellipse inscribed inside it

`object` and `class`

Why do we need an object-oriented code?

- Purpose: reuse a code that we need quite often
- data and function/method bound together, e.g., a `list`
  - list instance `l`, method `l.append()` or `l.sort()`
  - when the `object` is passed, the methods are passed too
  - `object` is an **instance** of a `class`
  - example: class `rectangle`, create an object of class rectangle, has parameters/data length and height, function of that data is area, longer side, shorter side, length of diagonal, area of the largest ellipse inscribed inside it
- abstract data types – defined by a `class`

`object` and `class`

Why do we need an object-oriented code?

- Purpose: reuse a code that we need quite often
- data and function/method bound together, e.g., a `list`
  - list instance `l`, method `l.append()` or `l.sort()`
  - when the `object` is passed, the methods are passed too
  - `object` is an **instance** of a `class`
  - example: class `rectangle`, create an object of class rectangle, has parameters/data length and height, function of that data is area, longer side, shorter side, length of diagonal, area of the largest ellipse inscribed inside it
- abstract data types – defined by a `class`
- `object`: collection of data and function, as defined in the `class`

`object` and `class`

Why do we need an object-oriented code?

- Purpose: reuse a code that we need quite often
- data and function/method bound together, e.g., a `list`
  - list instance `l`, method `l.append()` or `l.sort()`
  - when the `object` is passed, the methods are passed too
  - `object` is an **instance** of a `class`
  - example: class `rectangle`, create an object of class rectangle, has parameters/data length and height, function of that data is area, longer side, shorter side, length of diagonal, area of the largest ellipse inscribed inside it
- abstract data types – defined by a `class`
- `object`: collection of data and function, as defined in the `class`
- **Example:** simulation of LUDO game

Simulation using OOP

- discrete random variable with finite state space

Simulation using OOP

- discrete random variable with finite state space
- define the class of discrete random variables
- an instance of the class is a random variable, which has a distribution – probability masses

Simulation using OOP

- discrete random variable with finite state space
- define the class of discrete random variables
- an instance of the class is a random variable, which has a distribution – probability masses

how to draw a random variable of a given distribution

- inverse CDF method – true for discrete random variable too

Summary

This series of lectures discussed

- The basic syntaxes of python and its standard data types, loops and conditionals
- some implementations of numerical techniques

Summary

This series of lectures discussed

- The basic syntaxes of python and its standard data types, loops and conditionals
- some implementations of numerical techniques
- Four important set of scientific libraries

Summary

This series of lectures discussed

- The basic syntaxes of python and its standard data types, loops and conditionals
- some implementations of numerical techniques
- Four important set of scientific libraries
- Object-oriented programming in python

References

- **[online course materials]** Python for scientific computing, by Swaprava Nath: `http://scientificcomputing.is-great.net/`
- **[book]** Introduction to Computation and Programming using Python, by John Guttag, Prentice Hall of India.
- **[online course materials]** Lectures in Quantitative Economics, Thomas J. Sargent and John Stachurski: `https://lectures.quantecon.org/py/index.html`