# COMPSCI 590N
# Lecture 1: Python Basics

Roy J. Adams

College of Information and Computer Sciences
University of Massachusetts Amherst

## Outline

## What is Python?

Python is an interpreted programming language designed to be readable, compact, and scalable.

## Why use Python?

Python has a number of features that make it particular well suited for research and numerical computing:

## Why use Python?

Python has a number of features that make it particular well suited for research and numerical computing:

- Clear, readable syntax

## Why use Python?

Python has a number of features that make it particular well suited for research and numerical computing:

- Clear, readable syntax
- Interactive mode

## Why use Python?

Python has a number of features that make it particular well suited for research and numerical computing:

- Clear, readable syntax
- Interactive mode
- Modular and portable

## Why use Python?

Python has a number of features that make it particular well suited for research and numerical computing:

- Clear, readable syntax
- Interactive mode
- Modular and portable
- Automatic memory management

## Why use Python?

Python has a number of features that make it particular well suited for research and numerical computing:

- Clear, readable syntax
- Interactive mode
- Modular and portable
- Automatic memory management
- Flexible code time vs. run time tradeoff

## Why use Python?

Python has a number of features that make it particular well suited for research and numerical computing:

- Clear, readable syntax
- Interactive mode
- Modular and portable
- Automatic memory management
- Flexible code time vs. run time tradeoff
- A commonly used, well-documented set of numerical libraries

## Course Goals

The goal of this course is to give the students necessary programming and algorithmic knowledge to be able to write effective numerical programs in Python of the type used in CS 589, 585, etc. Topics we will cover include:

## Course Goals

The goal of this course is to give the students necessary programming and algorithmic knowledge to be able to write effective numerical programs in Python of the type used in CS 589, 585, etc. Topics we will cover include:

- Basic programming concepts in Python.

## Course Goals

The goal of this course is to give the students necessary programming and algorithmic knowledge to be able to write effective numerical programs in Python of the type used in CS 589, 585, etc. Topics we will cover include:

- Basic programming concepts in Python.
- How numbers are represented in a computer and how to work with them.

## Course Goals

The goal of this course is to give the students necessary programming and algorithmic knowledge to be able to write effective numerical programs in Python of the type used in CS 589, 585, etc. Topics we will cover include:

- Basic programming concepts in Python.
- How numbers are represented in a computer and how to work with them.
- Algorithms underlying many basic numerical functions:

## Course Goals

The goal of this course is to give the students necessary programming and algorithmic knowledge to be able to write effective numerical programs in Python of the type used in CS 589, 585, etc. Topics we will cover include:

- Basic programming concepts in Python.
- How numbers are represented in a computer and how to work with them.
- Algorithms underlying many basic numerical functions:
  - linear algebra

## Course Goals

The goal of this course is to give the students necessary programming and algorithmic knowledge to be able to write effective numerical programs in Python of the type used in CS 589, 585, etc. Topics we will cover include:

- Basic programming concepts in Python.
- How numbers are represented in a computer and how to work with them.
- Algorithms underlying many basic numerical functions:
  - linear algebra
  - probability

## Course Goals

The goal of this course is to give the students necessary programming and algorithmic knowledge to be able to write effective numerical programs in Python of the type used in CS 589, 585, etc. Topics we will cover include:

- Basic programming concepts in Python.
- How numbers are represented in a computer and how to work with them.
- Algorithms underlying many basic numerical functions:
    - linear algebra
    - probability
- Documentation, testing, and debugging in Python.

## Prerequisites

- The course has no formal prerequisites; however, an undergraduate level understanding of linear algebra and probability is expected. **No programming experience is expected.**

## Prerequisites

- The course has no formal prerequisites; however, an undergraduate level understanding of linear algebra and probability is expected. **No programming experience is expected.**
- The course will be taught so that students can take it concurrently with other CICS data science courses.

## Text Books

We will use two main sources, both of which can be found online (see the course website for links):

## Text Books

We will use two main sources, both of which can be found online (see the course website for links):

- *Introduction to Python for Computational Science and Engineering*. Hans Fangohr.

## Text Books

We will use two main sources, both of which can be found online (see the course website for links):

- *Introduction to Python for Computational Science and Engineering*. Hans Fangohr.
- *The SciPy Lecture Notes*.

## Text Books

We will use two main sources, both of which can be found online (see the course website for links):

- *Introduction to Python for Computational Science and Engineering*. Hans Fangohr.
- *The SciPy Lecture Notes*.

Readings will be posted to Moodle and are intended to be completed before class.

## Other Resources

There exist **many** excellent python resources that may be helpful supplements. A few are:

## Other Resources

There exist **many** excellent python resources that may be helpful supplements. A few are:

- *The Python Tutorial.* https://docs.python.org/3/tutorial/

## Other Resources

There exist **many** excellent python resources that may be helpful supplements. A few are:

- *The Python Tutorial.* https://docs.python.org/3/tutorial/
- *Google's Python Class.* https://developers.google.com/edu/python/

## Other Resources

There exist **many** excellent python resources that may be helpful supplements. A few are:

- *The Python Tutorial.* https://docs.python.org/3/tutorial/
- *Google's Python Class.* https://developers.google.com/edu/python/
- *Fast Lane to Python: A quick, sensible route to the joys of Python coding.* Norm Matloff. http://heather.cs.ucdavis.edu/ matlof-f/Python/PLN/FastLanePython.pdf

## Outline

## Getting Started

Software you will need for this class:

- Python 3.5

## Getting Started

Software you will need for this class:

- Python 3.5
  - Grading will be done in 3.5.

## Getting Started

Software you will need for this class:

- Python 3.5
  - Grading will be done in 3.5.
- Python modules: NumPy, SciPy

## Getting Started

Software you will need for this class:

- Python 3.5
    - Grading will be done in 3.5.
- Python modules: NumPy, SciPy
- Jupyter (optional for in-class demos)

## Getting Started

Software you will need for this class:

- Python 3.5
  - Grading will be done in 3.5.
- Python modules: NumPy, SciPy
- Jupyter (optional for in-class demos)

## Getting Started

Software you will need for this class:

- Python 3.5
  - Grading will be done in 3.5.
- Python modules: NumPy, SciPy
- Jupyter (optional for in-class demos)

Recommendation: **Anaconda** is a complete data science environment that ships with all of the above.
(https://www.continuum.io/downloads)

## The Read-Eval-Print Loop

Python is an interpreted language. This means Python programs are not compiled as a whole (like C/C++), but are read and compiled one line at a time. This process is sometimes called the **Read-Eval-Print Loop**.

## The Read-Eval-Print Loop

Python is an interpreted language. This means Python programs are not compiled as a whole (like C/C++), but are read and compiled one line at a time. This process is sometimes called the **Read-Eval-Print Loop**.

```
(1) Read
(2) Interpret
(3) Compile
(4) Execute
(5) Print
(6) Repeat (1)-(5)
```

## The Read-Eval-Print Loop

Python is an interpreted language. This means Python programs are not compiled as a whole (like C/C++), but are read and compiled one line at a time. This process is sometimes called the **Read-Eval-Print Loop**.

```
(1) Read
(2) Interpret
(3) Compile
(4) Execute
(5) Print
(6) Repeat (1)-(5)
```

This process is extremely flexible and powerful, but can be slow if you are not careful.

## Interactive Python

- One of the most powerful Python features is **Interactive Mode**.

## Interactive Python

- One of the most powerful Python features is **Interactive Mode**.
- Interactive mode allows a user enter lines of code, one at a time, directly into the interpreter.

## Interactive Python

- One of the most powerful Python features is **Interactive Mode**.
- Interactive mode allows a user enter lines of code, one at a time, directly into the interpreter.

## Interactive Python

- One of the most powerful Python features is **Interactive Mode**.
- Interactive mode allows a user enter lines of code, one at a time, directly into the interpreter.

DEMO

## Running a script

Most of what you write, however, will be written in code files sometimes called scripts.

## Running a script

Most of what you write, however, will be written in code files sometimes called scripts.

```
$ python script.py
```

## Running a script

Most of what you write, however, will be written in code files sometimes called scripts.

```
$ python script.py
```

DEMO

## Outline

## Variables and Assignment

- Variables are used to give names to values.

## Variables and Assignment

- Variables are used to give names to values.
- Names are assigned using the assignment operator, "="

## Variables and Assignment

- Variables are used to give names to values.
- Names are assigned using the assignment operator, "="

## Variables and Assignment

- Variables are used to give names to values.
- Names are assigned using the assignment operator, "="

```
>>> x = 2
>>> x
2
```

## Variables and Assignment

- Variables are used to give names to values.
- Names are assigned using the assignment operator, "="

```
>>> x = 2
>>> x
2

>>> x = "this is a string"
>>> x
"this is a string"
```

## Variables and Assignment

- Variables are used to give names to values.
- Names are assigned using the assignment operator, "="

```
>>> x = 2
>>> x
2

>>> x = "this is a string"
>>> x
"this is a string"

>>> y = x
>>> y
"this is a string"
```

## Basic Data Types

- Integers, `x = 2`

## Basic Data Types

- Integers, `x = 2`
- Floats (non-integers), `x = 2.5`

## Basic Data Types

- Integers, `x = 2`
- Floats (non-integers), `x = 2.5`
- Bools (True or False), `x = True`

## Basic Data Types

- Integers, `x = 2`
- Floats (non-integers), `x = 2.5`
- Bools (True or False), `x = True`

## Basic Data Types

- Integers, `x = 2`
- Floats (non-integers), `x = 2.5`
- Bools (True or False), `x = True`

```
>>> x = 1
>>> x
1
>>> y = float(x)
>>> y
1.0
>>> z = bool(y)
>>> z
True
```

## Math Operators

- Python support all standard math operators.

## Math Operators

- Python support all standard math operators.
- Arithmetic: +, -, *, /

## Math Operators

- Python support all standard math operators.
- Arithmetic: +, -, *, /
- Exponentiation: **

## Math Operators

- Python support all standard math operators.
- Arithmetic: +, -, *, /
- Exponentiation: **
- Remainder (mod): %

## Math Operators

- Python support all standard math operators.
- Arithmetic: +, -, *, /
- Exponentiation: **
- Remainder (mod): %
- Increment: +=, -=

## Math Operators

- Python support all standard math operators.
- Arithmetic: +, -, *, /
- Exponentiation: **
- Remainder (mod): %
- Increment: +=, -=

## Math Operators

- Python support all standard math operators.
- Arithmetic: +, -, *, /
- Exponentiation: **
- Remainder (mod): %
- Increment: +=, -=

```
>>> 2.0 + 3.0
5.0
>>> 3.0**2.0
9.0
>>> 10.0 % 3.0
1.0
>>> x = 5
>>> x += 3
>>> x
```

## Mixing Types

- Python will automatically choose a type if you mix them.

## Mixing Types

- Python will automatically choose a type if you mix them.

## Mixing Types

- Python will automatically choose a type if you mix them.

```
>>> 2 * 3
6
>>> 2.0 * 3.0
6.0
>>> 2.0 * 3
6.0
```

## Logical Operators

- Python also supports standard logical operators.

## Logical Operators

- Python also supports standard logical operators.
- Comparators: <, <=, ==, >=, >, !=

## Logical Operators

- Python also supports standard logical operators.
- Comparators: <, <=, ==, >=, >, !=
- Logical and/or: `and`, `or`

## Logical Operators

- Python also supports standard logical operators.
- Comparators: <, <=, ==, >=, >, !=
- Logical and/or: `and`, `or`
- Negation: `not`

## Logical Operators

- Python also supports standard logical operators.
- Comparators: <, <=, ==, >=, >, !=
- Logical and/or: `and`, `or`
- Negation: `not`

## Logical Operators

- Python also supports standard logical operators.
- Comparators: <, <=, ==, >=, >, !=
- Logical and/or: `and`, `or`
- Negation: `not`

```
>>> a = 2.0
>>> b = 3.0
>>> a > b
False
>>> not a > b
True
>>> (a > b) or (b == 3.0)
True
```

## Strings

- Strings store store text.

## Strings

- Strings store store text.
- Create strings using quotations: e.g.

```
x = "this is a string"
```

## Strings

- Strings store store text.
- Create strings using quotations: e.g.
  ```
  x = "this is a string"
  ```
- Strings support many of the same operations as numeric types including: +, *, <, <=, ==, ...

## Strings

- Strings store store text.
- Create strings using quotations: e.g.

```
x = "this is a string"
```

- Strings support many of the same operations as numeric types including: +, *, <, <=, ==, ...

## Strings

- Strings store store text.
- Create strings using quotations: e.g.
  `x = "this is a string"`
- Strings support many of the same operations as numeric types including: +, *, <, <=, ==, ...

```
>>> "abc" + "de"
'abcde'
>>> 2*"abc"
'abcabc'
>>> "abc" == 'abc'
True
>>> "a" < "b"
True
```

## Lists and Tuples

- Lists and Tuples store sequences of objects.

## Lists and Tuples

- Lists and Tuples store sequences of objects.
- Contents can be mixed types.

## Lists and Tuples

- Lists and Tuples store sequences of objects.
- Contents can be mixed types.

## Lists and Tuples

- Lists and Tuples store sequences of objects.
- Contents can be mixed types.

```
>>> a = [1,2,'a','b'] # Lists use []
>>> a
[1, 2, 'a', 'b']
>>> b = (1,2,a,'string') # Tuples use ()
>>> b
(1, 2, [1, 2, 'a', 'b'], 'string')
>>> len(b) # Length of a sequence
4
```

## Accessing sequences

- Individual entries of a List, Tuple, or String can be accessed using indexing.

## Accessing sequences

- Individual entries of a List, Tuple, or String can be accessed using indexing.
- Access groups of elements using a ":".

## Accessing sequences

- Individual entries of a List, Tuple, or String can be accessed using indexing.
- Access groups of elements using a ":".

## Accessing sequences

- Individual entries of a List, Tuple, or String can be accessed using indexing.
- Access groups of elements using a ":".

```
>>> a = [1,2,'a','b']
>>> a[0] # Indexing is zero based
1
>>> a[2]
'a'
>>> a[1:4] # Returns positions 1 to 3
[2,'a','b']
>>> a[-1]
'b'
```

## Dictionaries

- Dictionaries store key/value pairs.

## Dictionaries

- Dictionaries store key/value pairs.
- Values are accessed using the keys.

## Dictionaries

- Dictionaries store key/value pairs.
- Values are accessed using the keys.
- Created using { }.

## Dictionaries

- Dictionaries store key/value pairs.
- Values are accessed using the keys.
- Created using { }.

## Dictionaries

- Dictionaries store key/value pairs.
- Values are accessed using the keys.
- Created using { }.

```
>>> a = {'one':1,'two':2,'three':3}
>>> a['two']
2
>>> a.keys()
['three', 'two', 'one']
>>> a.values()
[3, 2, 1]
>>>
```

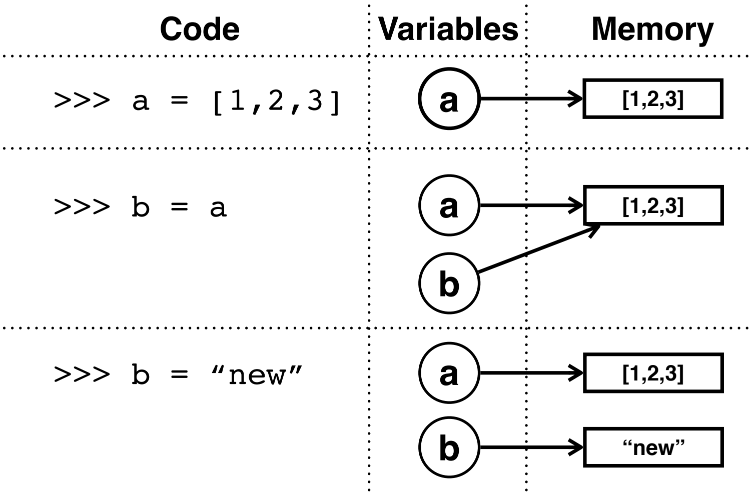
## Variables as Names

- A Python variable can be though as a name for an object.

## Variables as Names

- A Python variable can be though as a name for an object.
- Multiple names may refer to the same object.

## Variables as Names



## Variables as Names

DEMO

## Mutability

- A mutable object can have it's value changed. An immutable object cannot.

## Mutability

- A mutable object can have it's value changed. An immutable object cannot.
- Mutable: List, Dictionary

## Mutability

- A mutable object can have it's value changed. An immutable object cannot.
- Mutable: List, Dictionary
- Immutable: Int, Float, Bool, String, Tuple

## Mutability

- A mutable object can have it's value changed. An immutable object cannot.
- Mutable: List, Dictionary
- Immutable: Int, Float, Bool, String, Tuple

## Mutability

- A mutable object can have it's value changed. An immutable object cannot.
- Mutable: List, Dictionary
- Immutable: Int, Float, Bool, String, Tuple

DEMO