

CME 193: Introduction to Scientific Python

Lecture 1: Introduction

Sven Schmit

`stanford.edu/~schmit/cme193`

Contents

- **Administrivia**
- Introduction
- Basics
- Variables
- Control statements
- Exercises

Instructor

Sven Schmit

- from the Netherlands
- Third year PhD student in ICME
- Background in Statistics / Mathematics / CS
- Used Python for data science at Stitch Fix

Feedback

If you have comments, like things to be done differently, please let me know and let me know asap.

Questionnaires at the end of the quarter are nice, but they won't help you.

Content of course

- Variables
- Functions
- Data types
 - Strings, Lists, Tuples, Dictionaries
- File input and output (I/O)
- Numpy, Scipy and Matplotlib
- Classes
- Pandas, Statsmodels and IPython
- Exception handling
- Unit tests
- More packages

Setup of course

- Lectures: first half lecture, second half exercises
- Portfolio
- Final project

If you find you are doing less work than me, you are not learning.

More abstract setup of course

My job is to show you the possibilities, your job is to teach yourself Python!

If you think you can learn Python by just listening to me, you are grossly overestimating my powers.

Exercises

Exercises in second half of the class. Try to finish them in class, else make sure you understand them all before next class.

At the end of the course, hand in a portfolio with your solutions for exercises.

Portfolio

You are required to hand in a portfolio with your solutions to the exercises you attempted.

This is to show your active participation in class

You are expected to do at least 2/3rd of the assigned exercises.

Final project

Besides the portfolio, you are required to submit a project, due one week after the final class.

One paragraph proposal due before lecture 5.

Your opportunity to apply the skills you learn in this class to whatever interests you.

The main objective is to have fun! If you aren't enjoying your project, you are doing something wrong.

You are encouraged to use material not taught in class: we can't cover everything in class.

Workload

On one hand it is a one unit course: so it is relatively easy and relaxed. On the other hand, the only way to learn Python, is by writing Python... **a lot**. So you are expected to put in effort.

I think that the work you put in now is a great investment in your future: Knowing Python will prove extremely valuable whatever the future might bring.

Misc

Website `stanford.edu/~schmit/cme193`

Piazza Use Piazza for discussing problems. An active user on Piazza has more leeway when it comes to portfolio, as it shows involvement.

Office hours After class or by appointment.

References

The internet is an excellent source, and Google is a perfect starting point.

The official documentation is also good, always worth a try:

<https://docs.python.org/2/>.

Course website has a list of useful references.

Last words before we get to it

- You only learn by doing: do the exercises
- Don't fall behind, it is impossible to catch up
- Discuss with your neighbors, you learn most by teaching so help others!
- Try things: Find out what works and what does not, and find out why. Don't ever be stuck. Fail often.
- Test things: Make sure every little part works before moving on. Fail gently.

Contents

- Administrivia
- **Introduction**
- Basics
- Variables
- Control statements
- Exercises

An Introduction to Python

This class:

- Learn the fundamentals of programming
- Gain experience using Python
- Getting confident in using online resources to help you onwards
- Learn about widely used packages for 'computational math', though the basics are more important

Intended for people with limited or no programming experience. If you do have, say, C++ or Java experience, we might go pretty slow.

Feel free to move faster and spend more time discovering on your own!

What is a programming language?

Most of you will hopefully have a good idea:

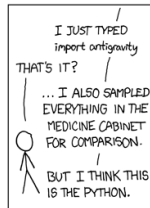
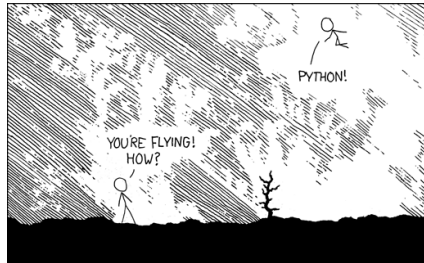
Programming languages translate tasks that we would like to perform on a computer, to a format that a computer understands.

Programming languages

There are many choices for a programming languages

- High level (Python) – low level (Assembly)
- General tasks (Java) – specific tasks (R)
- Different paradigms, etc.

Python



Python

- Easy to learn
- Fast to write code
- Intuitive
- Very versatile
- Less control, less performance
- Less safety handles, responsibility for user

Contents

- Administrivia
- Introduction
- **Basics**
- Variables
- Control statements
- Exercises

How to install Python

Many alternatives, but I suggest installing using a prepackaged distribution, such as Anaconda

`https://store.continuum.io/cshop/anaconda/`

This is very easy to install and also comes with a lot of packages.

See the getting started instructions on the course website for more information.

Packages

Packages enhance the capabilities of Python, so you don't have to program everything by yourself (it's faster too!).

For example: Numpy is a package that adds many linear algebra capabilities, more on that later

How to install packages

To install a package that you do not have, use `pip`, which is the Python package manager.

such as

```
$ pip install seaborn
```


Python 3

Python 3 has been around for a while and is slowly gaining traction. However, many people still use Python 2, so we will stick with that. Differences are not too big, so you can easily switch.

How to use Python

There are two ways to use Python:

command-line mode: talk directly to the interpreter

scripting-mode: write code in a file (called script) and run code by typing

```
$ python scriptname.py
```

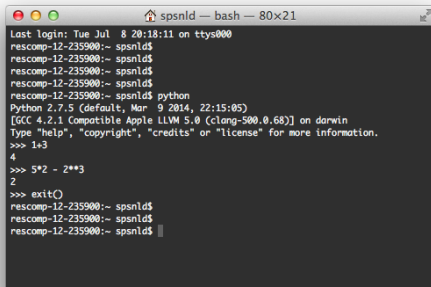
in the terminal.

The latter is what we will focus on in this course, though using the command-line can be useful to quickly check functionality.

The interpreter

We can start the interpreter by typing 'python' in the terminal.

Now we can interactively give instructions to the computer, using the Python language.

A terminal window titled 'spsnld — bash — 80x21' with standard macOS window controls. The terminal shows a user logging in at 'rescomp-12-235900' and typing 'python' to start the Python interpreter. The interpreter displays its version (2.7.5) and the system (darwin). The user enters three commands: '1+3', '5*2 - 2**3', and 'exit()'. The interpreter returns the results '4', '2', and exits the session, returning to the shell prompt.

```
spsnld — bash — 80x21
Last login: Tue Jul 8 20:18:11 on ttys000
rescomp-12-235900:~ spsnld$
rescomp-12-235900:~ spsnld$
rescomp-12-235900:~ spsnld$
rescomp-12-235900:~ spsnld$
rescomp-12-235900:~ spsnld$ python
Python 2.7.5 (default, Mar 9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+3
4
>>> 5*2 - 2**3
2
>>> exit()
rescomp-12-235900:~ spsnld$
rescomp-12-235900:~ spsnld$
rescomp-12-235900:~ spsnld$
```

Scripting mode

A more convenient way to interact with Python is to write a script.

A script contains all code you want to execute. Then you call Python on the script to run the script.

First browse, using the terminal, to where the script is saved

Then call `python scriptname.py`

Scripting mode

Suppose the Python script is saved in a folder `/Documents/Python` called `firstscript.py`.

Then browse to the folder by entering the following command into the terminal

```
$ cd ~/Documents/Python
```

And then run the script by entering

```
$ python firstscript.py
```

Print statement

We can print output to screen using the `print` command

```
print "Hello, world!"
```

Write your first script, that prints `Hello, world!` as above and run it to make sure it works.

Contents

- Administrivia
- Introduction
- Basics
- **Variables**
- Control statements
- Exercises

Values

A value is the fundamental thing that a program manipulates.

Values can be “Hello, world!”, 42, 12.34, True

Values have types. . .

Types

Boolean True/False

String "Hello, world!"

Integer 92

Float 3.1415

Use `type` to find out the type of a variable, as in

```
» type("Hello, world!")
```

```
<type 'str'>
```

Variables

One of the most basic and powerful concepts is that of a *variable*.

A variable *assigns* a name to a value.

```
message = "Hello, world!"  
n = 42  
e = 2.71  
  
# note we can print variables:  
print n # yields 42  
  
# note: everything after pound sign is a comment
```

Try it!

Variables

Almost always preferred to use variables over values:

- Easier to update code
- Easier to understand code (useful naming)

What does the following code do:

```
print 4.2 * 3.5
```

```
length = 4.2  
height = 3.5  
area = length * height  
print area
```

Variables

Almost always preferred to use variables over values:

- Easier to update code
- Easier to understand code (useful naming)

What does the following code do:

```
print 4.2 * 3.5
```

```
length = 4.2  
height = 3.5  
area = length * height  
print area
```

Naming

- Choose meaningful variable names
- Should start with a letter
- Cannot use keywords

Use the Google Python style guide!*

*See References on course website.

Keywords

Not allowed to use keywords, they define structure and rules of a language.

Python has 29 keywords, they include:

- and
- def
- for
- return
- is
- in
- class

Statements, expressions and operators

A statement is an instruction that Python can execute, such as

```
x = 3
```

Operators are special symbols that represent computations, like addition, the values they *operate* on are called operands

An *expression* is a combination of values, variable and operators

```
x + 3
```

Integers

Operators for integers

`+` `-` `*` `/` `%` `**`

Note: `/` uses integer division:

`5 / 2` yields `2`

But, if one of the operands is a float, the return value is a float:

`5 / 2.0` yields `2.5`

Floats

A floating point number approximates a real number.

Note: only finite precision, and finite range (overflow)!

Operators for floats

+ addition

- subtraction

* multiplication

/ division

% modulo operator

** power

Booleans

Boolean expressions:

`==` equals: `5 == 5` yields `True`

`!=` does not equal: `5 != 5` yields `False`

`>` greater than: `5 > 4` yields `True`

`>=` greater than or equal: `5 >= 5` yields `True`

Similarly, we have `<` and `<=`.

Logical operators:

`True and False` yields `False`

`True or False` yields `True`

`not True` yields `False`

Booleans

Boolean expressions:

`==` equals: `5 == 5` yields `True`

`!=` does not equal: `5 != 5` yields `False`

`>` greater than: `5 > 4` yields `True`

`>=` greater than or equal: `5 >= 5` yields `True`

Similarly, we have `<` and `<=`.

Logical operators:

`True and False` yields `False`

`True or False` yields `True`

`not True` yields `False`

Modules

Not all functionality available comes automatically when starting python, and with good reasons.

We can add extra functionality by importing modules:

```
»» import math
```

```
»» math.pi
```

```
3.141592653589793
```

Useful modules: `math`, `random`, and as we will see later `numpy`, `scipy` and `matplotlib`.

More on modules later!

Final words

No need to remember the details: everything is well documented online!

However, knowing about the tools that exist will help you look them up!

Always try the Python documentation first if you forgot something.

Contents

- Administrivia
- Introduction
- Basics
- Variables
- **Control statements**
- Exercises

Control statements

Control statements allow you to do more complicated tasks.

- If
- Else
- For
- While

If statements

Using `if`, we can execute part of a program conditional on some statement being true.

```
if traffic_light == 'green':  
    move()
```


Indentation

In Python, blocks of code are defined using indentation.

This means that everything indented after an `if` statement is only executed if the statement is `True`.

If the statement is `False`, the program skips all indented code and resumes at the first line of unindented code

```
if statement:
    # if statement is True, then all code here
    # gets executed but not if statement is False
    print "The statement is true"
    print "Else, this would not be printed"
# the next lines get executed either way
print "Hello, world,"
print "Bye, world!"
```

If-Else statement

We can add more conditions to the If statement using else and elif (short for else if)

```
if traffic_light == 'green':  
    drive()  
elif traffic_light == 'orange':  
    accelerate()  
else:  
    stop()
```

For loops

Very often, one wants to repeat some action. This can be achieved by a for loop

```
for i in range(5):  
    print i**2,  
# 0 1 4 9 16
```

Here, `range(n)` gives us a *list* with integers $0, \dots, n - 1$. More on this later!

While loops

When one does not know how many iterations are needed, we can use `while`.

```
i = 1
while i < 100:
    print i**2,
    i += i**2 # a += b is short for a = a + b
# 1 4 36 1764
```

Continue

`continue` continues with the next iteration of the smallest enclosing loop.

```
for num in range(2, 10):  
    if num % 2 == 0:  
        print "Found an even number", num  
        continue  
    print "Found an odd number", num
```

from: Python documentation

Continue

`continue` continues with the next iteration of the smallest enclosing loop.

```
for num in range(2, 10):  
    if num % 2 == 0:  
        print "Found an even number", num  
        continue  
    print "Found an odd number", num
```

from: Python documentation

Break

The break statement allows us to jump out of the smallest enclosing for or while loop.

Finding prime numbers:

```
max_n = 10
for n in range(2, max_n):
    for x in range(2, n):
        if n % x == 0: # n divisible by x
            print n, 'equals', x, '*', n/x
            break
    else: # executed if no break in for loop
        # loop fell through without finding a factor
        print n, 'is a prime number'
```

from: Python documentation

Break

The break statement allows us to jump out of the smallest enclosing for or while loop.

Finding prime numbers:

```
max_n = 10
for n in range(2, max_n):
    for x in range(2, n):
        if n % x == 0: # n divisible by x
            print n, 'equals', x, '*', n/x
            break
    else: # executed if no break in for loop
        # loop fell through without finding a factor
        print n, 'is a prime number'
```

from: Python documentation

Pass

The `pass` statement does nothing, which can come in handy when you are working on something and want to implement some part of your code later.

```
if traffic_light == 'green':  
    pass # to implement  
else:  
    stop()
```

Contents

- Administrivia
- Introduction
- Basics
- Variables
- Control statements
- Exercises

Exercises