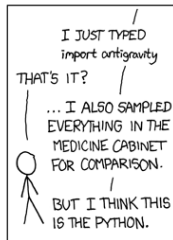
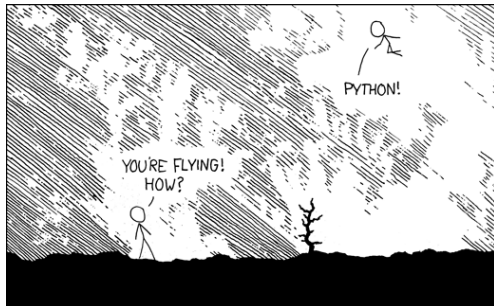


# Programming with Python

## EOAS Software Carpentry Workshop

September 21st, 2016



# Getting started

For our Python introduction we're going to pretend to be a researcher studying inflammation in patients who have been given a new treatment for arthritis.

You need to download some files to follow this lesson:

1. Make a new folder in your Desktop called `python-novice-inflammation`.
2. Download `python-novice-inflammation-data.zip` and move the file to this folder.
3. If it's not unzipped yet, double-click on it to unzip it. You should end up with a new folder called `data`.
4. You can access this folder from the Unix shell with:

```
$ cd && cd Desktop/python-novice-inflammation/data
```

# Launching Jupyter Notebook

There are several ways that we can use Python. We're going to start with a tool called Jupyter Notebook that runs in the browser. In a shell window enter these commands:

```
$ cd
$ cd Desktop/python-novice-inflammation/data
$ jupyter notebook
```

The shell window is now running a local web server for you. Don't close it. You will need to open another shell window to do other command line things. Your browser should open to an "Jupyter: Notebook" page showing a list of directories.

# Analyzing patient data

1. Explain what a library is, and what libraries are used for.
2. Load a Python library and use the things it contains.
3. Read tabular data from a file into a program.
4. Assign values to variables.
5. Select individual values and subsections from data.

- `import numpy`
- `numpy.loadtxt(fname=  
delimiter=)`
- `weight_kg = 55`
- `print('weight in kg:',  
weight_kg)`
- `weight_lb = 2.2 *  
weight_kg`
- `type(data)`
- `data.shape`
- `data[0,0], data[0:1,0:1]`
- `data[0:10:2,1]`
- `data[:3,36:]`

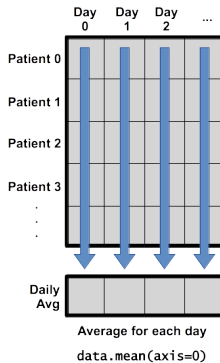
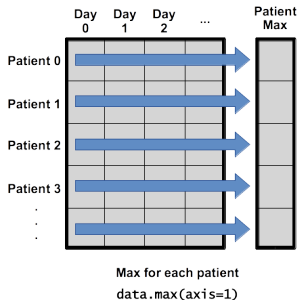
# Analyzing Patient Data cont'd

6. Perform operations on arrays of data.

7. Display simple graphs.

- `data.mean()`
- `data.std()`
- `data.mean(axis=0)`
- `%matplotlib inline`
- `from matplotlib import pyplot`
- `pyplot.imshow(data)`
- `pyplot.show()`
- `pyplot.plot(ave_inflammation)`
- `import matplotlib import pyplot as plt`
- `plt.subplot(1,3,1)`
- `plt.ylabel('average')`
- `plt.show()`

# Operations across an axis



## Exercise

Create a single plot showing 1) the mean for each day and 2) the mean + 1 standard deviation for each day and 3) the mean - 1 standard deviation for each day.



# Repeating actions with loops

1. Explain what a for loop does.
2. Correctly write for loops to repeat simple calculations.
3. Trace changes to a loop variable as the loop runs.
4. Trace changes to other variables as they are updated by a for loop.

- for char in word:

- len('aeiou')

Python has a built-in function called `range` that creates a list of numbers: `range(3)` produces `[0, 1, 2]`, `range(2, 5)` produces `[2, 3, 4]`, and `range(2, 10, 3)` produces `[2, 5, 8]`. Using `range`, write a loop that prints the first three natural numbers:

```
1  
2  
3
```

Python has a built-in function called `range` that creates a list of numbers: `range(3)` produces `[0, 1, 2]`, `range(2, 5)` produces `[2, 3, 4]`, and `range(2, 10, 3)` produces `[2, 5, 8]`. Using `range`, write a loop that prints the first three natural numbers:

One solution:

```
for num in range(1,4,1):  
    print(num)
```

Exponentiation is built into Python:

```
print(5**3)  
125
```

Write a loop that calculates the same result using multiplication (without exponentiation).

Exponentiation is built into Python:

```
print(5**3)  
125
```

Write a loop that calculates the same result using multiplication  
(without exponentiation)

One possible answer:

```
ans=1  
for ii in range(1,4,1):  
    ans=ans*5  
print(ans)
```

# Storing Multiple Values in Lists

## Learning Goals

1. Explain what a list is.
2. Create and index lists of simple values.

## Lesson Commands

- `odds = [1, 3, 5, 7]`
- `print(odds[0], odds[-1])`
- `for number in odds:`
- `names[1] = 'Darwin'`
- `odds.append(11)`
- `del odds[0]`
- `odds.reverse()`

# Exercise

## Turn a String into a List

Use a for loop to convert the string 'hello' into a list of letters:

```
['h', 'e', 'l', 'l', 'o']
```

Hint: You can create an empty list like this:

```
my_list = []
```

# Storing Multiple Values in Lists

## Learning Goals

1. Explain what a list is.
2. Create and index lists of simple values.

## Lesson Commands

- `odds = [1, 3, 5, 7]`
- `print(odds[0], odds[-1])`
- `for number in odds:`
- `names[1] = 'Darwin'`
- `odds.append(11)`
- `del odds[0]`
- `odds.reverse()`



# Analyzing Data from Multiple Files

## Learning Goals

1. Use a library function to get a list of filenames that match a simple wildcard pattern.
2. Use a for loop to process multiple files.

## Lesson Commands

- `import glob`
- `filenames = glob.glob('*.*csv')`
- `filenames[0:3]`

# Making Choices

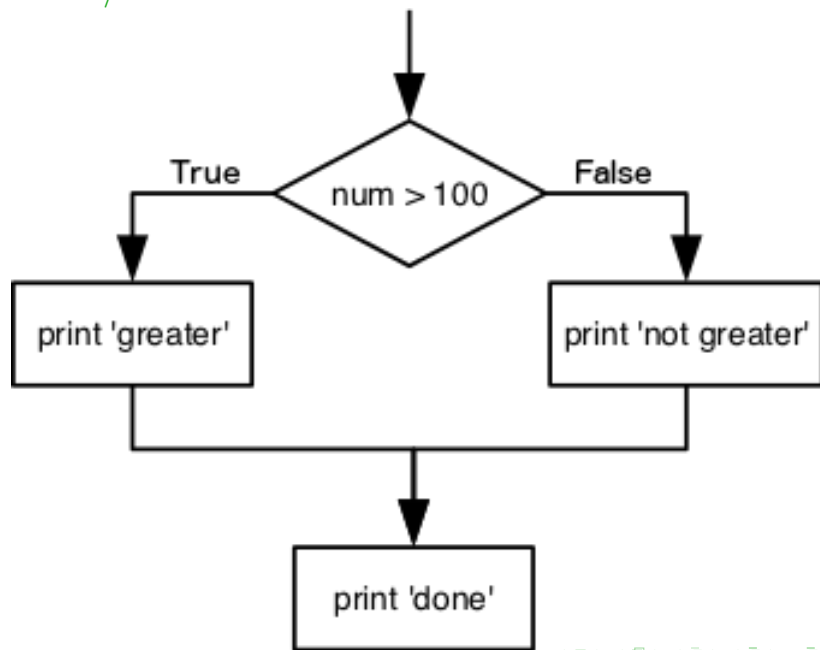
## Learning Goals

1. Write conditional statements including 'if', 'elif', and 'else' branches.
2. Correctly evaluate expressions containing 'and' and 'or'.

## Lesson Commands

- `if num > 100:`
- `else:`
- `if num > 0:`
- `elif num == 0:`
- `and`
- `or`

## Python if/else Flowchart



# Exercise

## How Many Paths?

What will be printed if you run this code:

```
if 4 > 5:  
    print('A')  
elif 4 == 5:  
    print('B')  
elif 4 < 5:  
    print('C')
```

1. A
2. B
3. C
4. B and C

Why did you pick your answer?

# Exercise

## Close Enough

Work with your partner to write some code that will print `True` if the value of variable `a` is within 10% of the value of variable `b` and `False` otherwise. Test your code for positive values, negative values, and values that span zero.

# Making Choices

## Learning Goals

1. Write conditional statements including 'if', 'elif', and 'else' branches.
2. Correctly evaluate expressions containing 'and' and 'or'.

## Lesson Commands

- `if num > 100:`
- `else:`
- `if num > 0:`
- `elif num == 0:`
- `and`
- `or`

# Creating Functions - Defining a Function

## Learning Goals

1. Explain why we should divide programs into small, single-purpose functions.
2. Define a function that takes parameters.
3. Return a value from a function.

## Example Code

- ```
def fahr_to_kelvin(temp):  
    return ((temp - 32) * (5/9)) + 273.15
```
- ```
def kelvin_to_celsius(temp):  
    return temp - 273.15
```
- ```
def fahr_to_celsius(temp):  
    temp_k = fahr_to_kelvin(temp)  
    result = kelvin_to_celsius(temp_k)  
    return result
```

## Exercise

Write a function called `analyze` that takes a filename as a parameter and displays the three graphs produced in the previous lesson, i.e., `analyze('inflammation-01.csv')` should produce the graphs already shown, while `analyze('inflammation-02.csv')` should produce corresponding graphs for the second data set. Hint: a function can just “do” something. It doesn’t necessarily need to return anything.



## Solution

```
def analyze(filename):  
    data = np.loadtxt(fname=filename, delimiter=',')  
    fig = plt.figure(figsize=(10.0, 3.0))  
  
    axes1 = fig.add_subplot(1, 3, 1)  
    axes2 = fig.add_subplot(1, 3, 2)  
    axes3 = fig.add_subplot(1, 3, 3)  
  
    axes1.set_ylabel('average')  
    axes1.plot(data.mean(axis=0))  
  
    axes2.set_ylabel('max')  
    axes2.plot(data.max(axis=0))  
  
    axes3.set_ylabel('min')  
    axes3.plot(data.min(axis=0))  
  
    fig.tight_layout()  
    plt.show(fig)
```

## Defining a Function

```
def detect_problems(filename):  
  
    data = np.loadtxt(fname=filename, delimiter=',')  
  
    if data.max(axis=0)[0] == 0 and data.max(axis=0)[20] == 0:  
        print('Suspicious looking maxima!')  
    elif data.min(axis=0).sum() == 0:  
        print('Minima add up to zero!')  
    else:  
        print('Seems OK!')
```

# Testing and Documentation

## Learning Goal

3. Test and debug a function.

## Example Code

- ```
def centre(data, desired):  
    return (data - data.mean()) + desired
```
- ```
z = numpy.zeros((2,2))
```
- ```
print centre(z, 3)
```
- ```
print data.std() - centred.std()
```
- ```
def center(data, desired):  
    '''Return a new array containing the original data  
    centered around the desired value.'''  
    return (data - data.mean()) + desired
```
- ```
help(centre)
```

# Defining Defaults

## Learning Goals

6. Set default values for function parameters.

## Example Code

- `def center(data, desired = 0):`
- `def display(a=1, b=2, c=3):`  
    `print 'a:', a, 'b:', b, 'c:', c`  
    `print 'no parameters:'`  
    `display()`  
    `print 'one parameter:'`  
    `display(55)`  
    `print 'two parameters:'`  
    `display(55, 66)`
- `help(numpy.loadtxt)`

## Exercise

“Adding” two strings produces their concatenation: `'a' + 'b'` is `'ab'`. Write a function called `fence` that takes two parameters called `original` and `wrapper` and returns a new string that has the wrapper character at the beginning and end of the original. A call to your function should look like this:

```
print(fence('name', '*'))  
*name*
```

## Exercise

“Adding” two strings produces their concatenation: `'a' + 'b'` is `'ab'`. Write a function called `fence` that takes two parameters called `original` and `wrapper` and returns a new string that has the wrapper character at the beginning and end of the original. A call to your function should look like this:

```
print(fence('name', '*'))  
*name*
```

## Solution

```
def fence(original, wrapper):  
    """Returns a string with charcter wrapper added to the  
       beginning and end of string original."""  
  
    return wrapper + original + wrapper
```

# Tracebacks and Exceptions

## Learning Goals

1. Read a traceback, and determine the following relevant pieces of information:
  - ▶ The file, function, and line number on which the error occurred
  - ▶ The type of the error
  - ▶ The error message
2. Describe the types of situations in which the following errors occur:
  - ▶ `SyntaxError` and `IndentationError`
  - ▶ `NameError`
  - ▶ `IndexError`
  - ▶ `FileNotFoundError`

## Exercise

Does this code raise an exception? If so, what is the name of the exception?

```
for x in range(10, -10, -1):  
    print('inverse of', x, 'is', 1/x)
```

Can you modify the code so that it does what is intended, but avoids the exception?



# Try/Except Blocks

## Learning Goals

1. Write error handling Python code using try and except statements.

## Lesson Commands

```
try:  
    # something that might go wrong  
except SomeError:  
    # handle the error
```

# Command-line programs

## Learning goals

1. Use the values of command-line arguments in a program.
2. Handle flags and files separately in a command-line program.
3. Read data from standard input in a program so that it can be used in a pipeline.

## Commands and functions

`sys.version`

`sys.argv`

`sys.stdin`

# Switching to shell commands

\$ in front of a command that tells you to run that command in the shell rather than the Python interpreter

- Rewrite `readings.py` so that it uses `-n`, `-m`, and `-x` instead of `--min`, `--mean`, and `--max` respectively. Is the code easier to read? Is the program easier to understand?
- Separately, modify `readings.py` so that if no action is given it displays the means of the data.