

Coding Bootcamp Code in Python

PYTHON?

enter, python!

- Python is an *interpreted* language
 - We can code either using the interpreter directly or using *scripts* (text files with python code)
- Python is an *object-oriented language*
 - Each variable is an *object* with a *name*, *value*, and *type*
 - The *type* determines what you can do with the variable
- Current 3.8.2

Python2 or Python3

- There are still a lot of Python2 in Linux.
- Python2 still exists in many commercial solutions
 - Instagram Makes a Smooth Move to Python 3
 - <https://thenewstack.io/instagram-makes-smooth-move-python-3/>
- Python2.7 was retired in 2020
- <https://pythonclock.org/>

Coding Bootcamp Code in Python

HOW TO RUN PYTHON FROM THE TERMINAL?

Say hello to Python: terminal

- Write script using your favorite editor (gedit/vim/emacs) and save to file, e.g., `hello_world.py`
- Run script using Python interpreter

```
$ python hello_world.py  
hello world!
```

- Make script executable

```
$ chmod u+x hello_world.py
```

- Run script directly

```
$ ./hello_world.py  
hello world!
```

For Linux/macOS:
2.7.x comes with
distribution

Interactive: interpreter in terminal

- Useful for experimentation, prototyping

```
$ python
Python 3.6.1 (default, Apr  4 2017, 05:16:07)
[GCC 5.4.0] on linux2
Type "help", "copyright", "credits" or "license"...
>>> t = (3, 7)
>>> a, _ = t
>>> a
3
```

- Quit using `quit()` function or `Ctrl-d`

Interactive: iPython

- More features than standard python shell

```
$ ipython
Python 3.6.1 (default, Apr  4 2017, 15:28:02)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.3 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra
               details.
```

In [1]:

- Not standard, requires install

Interactive: jupyter notebooks

The screenshot shows a Jupyter Notebook interface in a web browser. The top navigation bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', and 'Help'. A toolbar below has icons for file operations like new, open, save, and cell selection. The main area displays a large banner with the text 'INTERACTIVE NOTEBOOKS: SHARING THE CODE' and 'TOOLBOX'. Below the banner, a text cell contains 'Text and $\sqrt{x^2 + y^2}$ '. Code cells show 'In [5]: from math import sqrt' and 'In [6]: distance(3.1, 4.5)'. The output cell shows 'Out[6]: 5.4644304369257'. A sidebar on the right lists 'Python 3' and other kernels. A footer at the bottom right reads '6 NOVEMBER 2014 | VOL 515 | NATURE | 151'. Callout boxes point to specific features: 'Styled text' points to the banner text; 'Typeset LaTeX formulas' points to the mathematical expression; 'Python input' points to the code in cell [5]; and 'Output' points to the numerical result in cell [6]. A box labeled 'Interface in web browser' is at the bottom left.

Styled text

Typeset LaTeX formulas

TOOLBOX

INTERACTIVE NOTEBOOKS:
SHARING THE CODE

In [5]: `from math
def distance(x, y):
 return sqrt(x*x + y*y)`

In [6]: `distance(3.1, 4.5)`

Out[6]: 5.4644304369257

In []:

Python input

Output

Interface in web browser

Jupyter use cases

- Excellent for
 - Explorative programming
 - Data exploration
 - Communication, especially across domains
- Problems
 - What was (re-)executed, what not?
 - Version control?

Use Jupytext

Python help

- Built-in help: interpreter/iPython

```
>>> import sys
>>> help(sys.exit)
Help on built-in function exit in module sys:

exit(...)
    exit([status])

Exit the interpreter by raising SystemExit(status).
If the status is omitted or None, it defaults to ze...
```

- Works also in Jupyter notebooks

Installing & upgrading packages

- Use pip
 - Install new package

```
$ pip install pandas
```

- Upgrade a package

```
$ pip install -U pandas
```

- Install a package only for yourself

```
$ pip install --user -U pandas
```

Conda: environments

- Install anconda (<https://www.anaconda.com/distribution>)
- Create new environment

```
$ conda create -n science python=3 \
    numpy scipy matplotlib
```

environment name

Python version

packages to install

- Use environment
 - Deactivate environment
- ```
$ conda activate science
```
- ```
$ conda deactivate
```

Conda: installing & updating

- Install new package

```
$ conda install holoviews
```

- Update package

```
$ conda update holoviews
```

- Update environment

```
$ conda update --all
```

- Uninstall package

```
$ conda remove holoviews
```

- List all installed packages

```
$ conda list
```

Note: will also install dependencies locally, including non-python libraries

Conda: multiple environments

- Clone environment

```
$ conda create -n data_science --clone science \
    pandas seaborn
```

The diagram illustrates the components of a Conda command for cloning environments. It shows a command line with several parts highlighted by colored boxes and arrows:

- A red box highlights the argument `-n data_science`, with a red arrow pointing to it from the text "environment name".
- A green box highlights the package names `pandas seaborn`, with a green arrow pointing to it from the text "additional packages to install".
- A blue box highlights the argument `--clone science`, with a blue arrow pointing to it from the text "base environment".
- A black box contains the command prefix `$ conda`.
- A black box contains the command suffix `\`.

- List all environments

```
$ conda env list
```

- Remove environment

```
$ conda remove --name data_science --all
```

Conda: sharing environments

- Export environment description
 - Export to YAML file

```
$ conda activate science
$ conda env export > science_environment.yml
```

- Create new environment based on description, portable across systems

```
$ conda env create -n science_env \
-f science_environment.yml
```

Conda: caveats

- Conda installs dependencies
 - Easy & fast
 - System specific distribution: no compiles
 - Library dependencies, e.g., zlib, mpich,...
- Upgrading Python version: clone first!

Further reading

- Conda cheat sheet

https://docs.conda.io/projects/conda/en/latest/_downloads/843d9e0198f2a193a3484886fa28163c/conda-cheatsheet.pdf

- Jupyter notebook tips

<https://www.dataquest.io/blog/jupyter-notebook-tips-tricks-shortcuts/>

Code Pack 01

- A. Hello Python
- B. Travelling to Jupyter
- C. Anaconda can bite you

Coding Bootcamp Code in Python

HOW TO RUN PYTHON USING ANACONDA?

 **ANACONDA.**: Anaconda

- spyder IDE
 - Editor: write scripts/modules
 - Console, i.e., iPython interpreter: execute code snippets, run scripts
 - Object inspector: from editor/console
- Standalone Qt iPython console
- jupyter notebooks
- Manage environments
- Platforms: Windows/macOS/Linux
- License: free for academic use



Anaconda navigator

The screenshot shows the Anaconda Navigator application window. On the left is a sidebar with navigation links: Home (highlighted), Environments (boxed in red), Projects (beta), Learning, Community, Documentation, Developer Blog, and Feedback. Below these are social media icons for Twitter, YouTube, and GitHub. The main area displays a grid of application tiles. The first row contains three tiles: 'jupyter notebook' (version 5.0.0), 'IP[y]: qtconsole' (version 4.3.0), and 'spyder' (version 3.1.3). The second row contains three more tiles, which are partially visible. Each tile includes a 'Launch' button and a gear icon for settings. The top right of the main area has buttons for 'Upgrade Now' and 'Sign in to Anaconda Cloud'. The top bar includes standard window controls (minimize, maximize, close) and the title 'Anaconda Navigator'.

Applications on root Channels Refresh

jupyter notebook 5.0.0

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

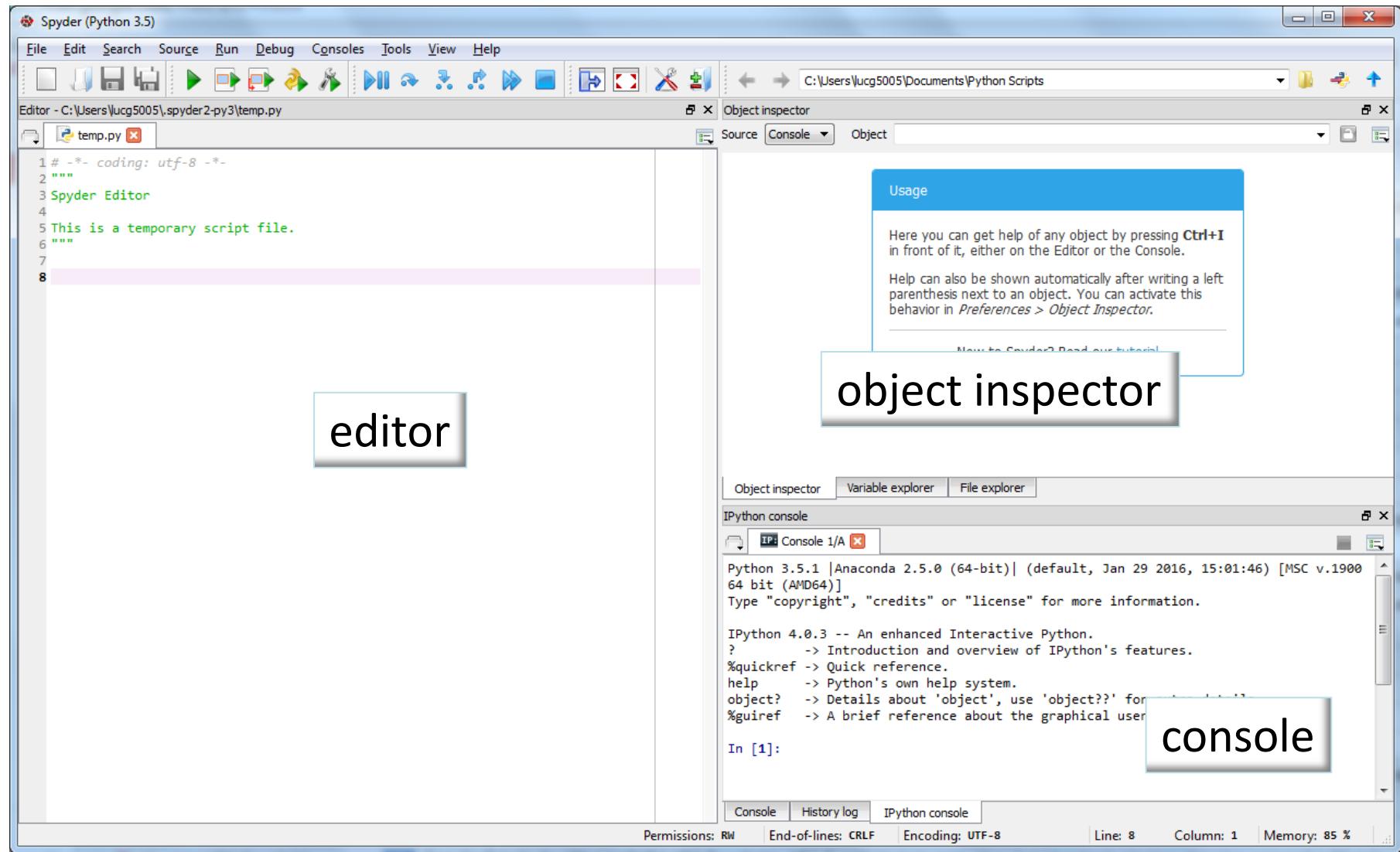
IP[y]: qtconsole 4.3.0

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

spyder 3.1.3

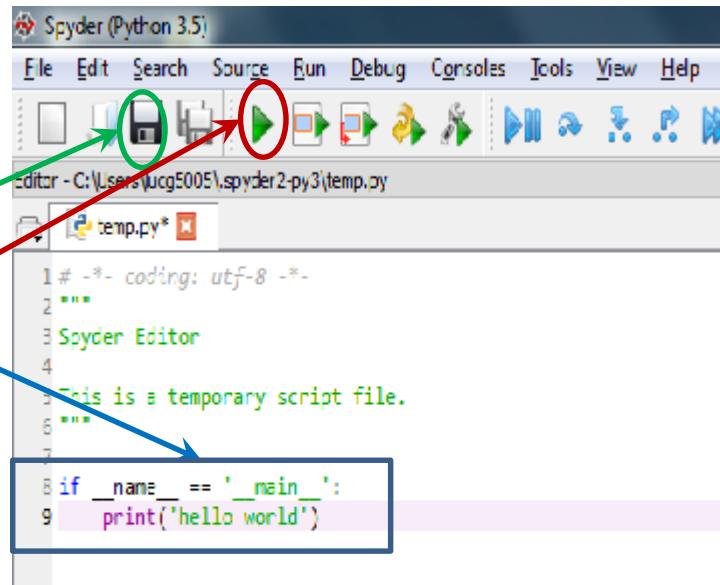
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

spyder



spyder: work cycle

- Repeat until done
 - Edit code
 - Save file
 - Run file
 - Check results



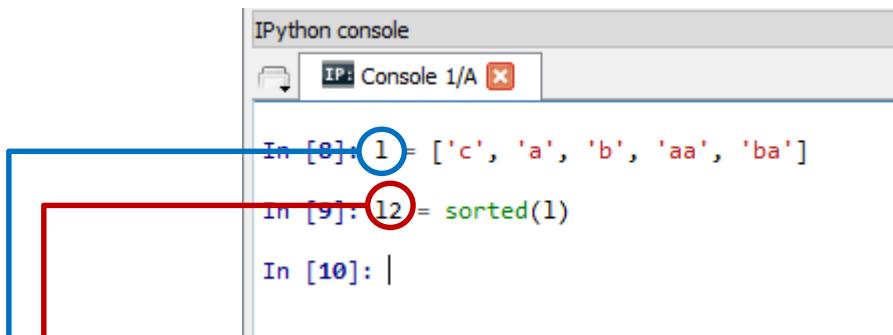
The screenshot shows the IPython console window titled 'IP: Console 1/A'. It displays the Python version information and the IPython help menu. At the bottom, it shows the output of running the script:

```
In [1]: runfile('C:/Users/lucg5005/.spyder2-py3/temp.py', wdir='C:/Users/lucg5005/.spyder2-py3')
hello world
```

A black box highlights the command 'runfile('C:/Users/lucg5005/.spyder2-py3/temp.py', wdir='C:/Users/lucg5005/.spyder2-py3')' in the IPython console. Another black box highlights the output 'hello world'.

spyder: object inspector

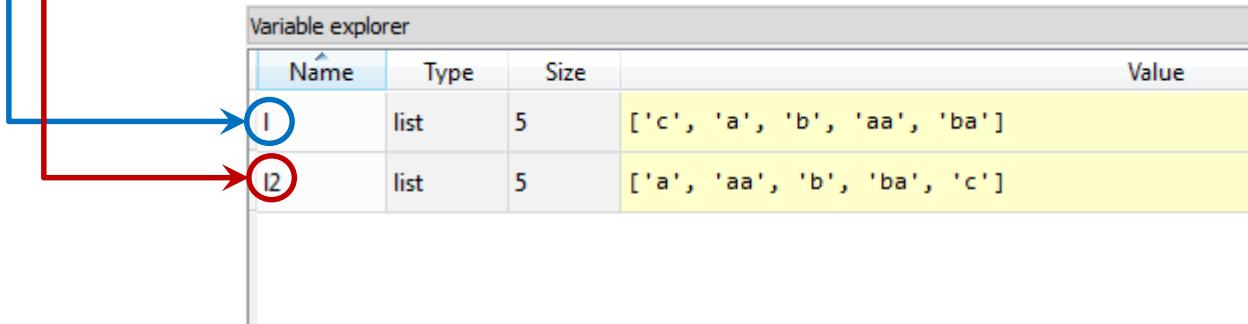
- Executing code snippets



The screenshot shows the IPython console interface. A blue box highlights the first two lines of code. A red box highlights the third line. The code is:

```
In [8]: l1 = ['c', 'a', 'b', 'aa', 'ba']
In [9]: l2 = sorted(l1)
In [10]: |
```

- Variable values in inspector

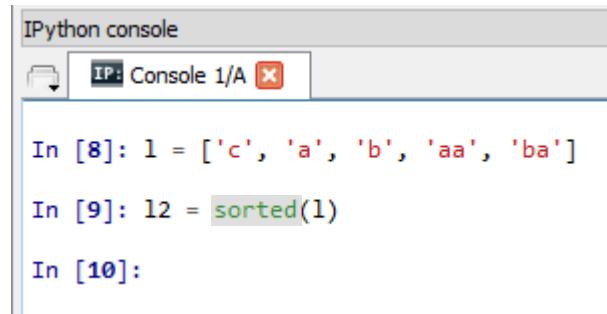


The screenshot shows the Variable explorer interface. A blue arrow points to the entry for 'l1'. A red arrow points to the entry for 'l2'. The table displays the following data:

Name	Type	Size	Value
l1	list	5	['c', 'a', 'b', 'aa', 'ba']
l2	list	5	['a', 'aa', 'b', 'ba', 'c']

spyder: getting help

- Select function/method/... in editor/console, press `ctrl-i`

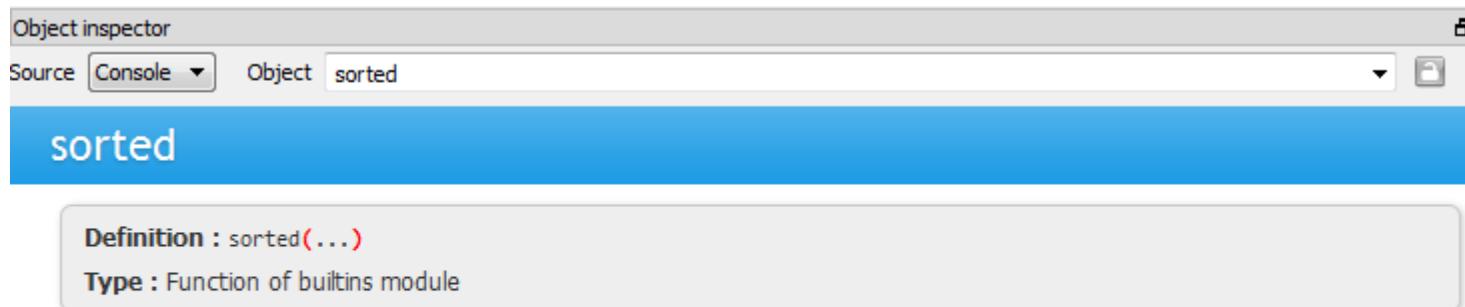


The screenshot shows the IPython console window with the title bar "IPython console" and "IP: Console 1/A". The console area displays the following code:

```
In [8]: l = ['c', 'a', 'b', 'aa', 'ba']
In [9]: l2 = sorted(l)
In [10]:
```

The word "sorted" is highlighted in green, indicating it is being completed.

- Help in object inspector



Return a new list containing all items from the iterable in ascending order.

A custom key function can be supplied to customise the sort order, and the reverse flag can be set to request the result in descending order.

spyder: more features

- Project/file manager
- Debug code
- Profile code

Anaconda environments

Anaconda Navigator

File Help

ANACONDA NAVIGATOR

Upgrade Now Sign in to Anaconda Cloud

Home Environments Projects (beta) Learning Community Documentation Developer Blog Feedback

Search Environments Not installed Channels Update index... Search Packages

Name	Description	Version
bcolz	Provides columnar, chunked and compressable data containers.	1.0.0
bcrypt	Modern password hashing for your software and your servers	3.1.3
binstar		0.12
binstar-build		0.10.7
biopython	Freely available tools for biological computation	1.68
blaze-core		0.9.0
bleach		1.5.0
blist		1.3.6
blosc		1.7.0

856 packages available (root)

Create Clone Import Remove Apply Clear

The screenshot shows the Anaconda Navigator interface. The left sidebar has links for Home, Environments (which is selected and highlighted with a red box), Projects (beta), Learning, Community, Documentation, Developer Blog, and Feedback. Below the sidebar are social media icons for Twitter, YouTube, and GitHub. The main area has tabs for Search Environments, Not installed (which is selected and highlighted with a red box), Channels, and Update index... A search bar for packages is also present. The central part of the screen displays a list of packages with columns for Name, Description, and Version. The 'biopython' package is highlighted with a red box. At the bottom, there are buttons for Create, Clone, Import, Remove, Apply (which is highlighted with a red box), and Clear.

Code Pack 02

A. Spyder for now

Coding Bootcamp Code in Python

PYTHON FUNDAMENTALS: DATA TYPES & STATEMENTS

Hello world!

- Minimal code for Python script

```
if __name__ == '__main__':
    print('hello world!')
```

Python interpreter executes
all code in body

Indentation is relevant!

Code structure

Python Is case-sensitive!

Say hello!

- Save script in file `hello_world.py`
- Run script using Python interpreter

```
$ python hello_world.py  
hello world!
```

- Make script executable

```
$ chmod u+x hello_world.py
```

- Run script directly

```
$ ./hello_world.py  
hello world!
```

That's what the shebang is for:
`#!/usr/bin/env python`

Hello again!

- Encapsulate script in main function

```
import sys

def main():
    print('hello world!')
    return 0

if __name__ == '__main__':
    status = main()
    sys.exit(status)
```

Simple function, no arguments, return status only

Function call

Generating data

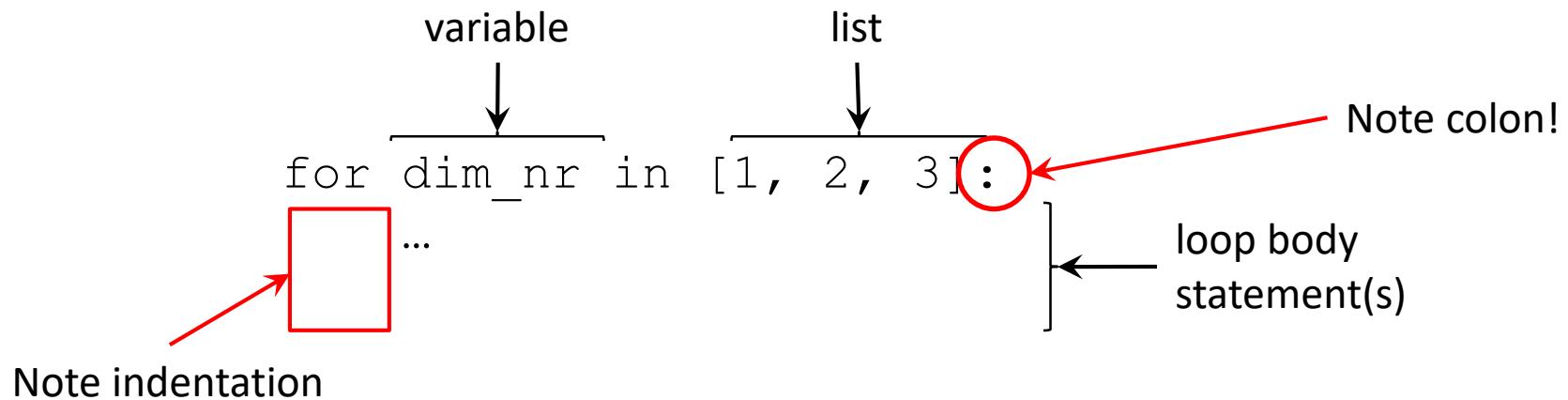
- Need some data?
 - first column, case number: sequential number
 - second column, dimension number: integer 1, 2, 3
 - third column, temperature: float value -0.5, 0.0, 0.5

```
def main():
    print('case', 'dim', 'temp')
    case_nr = 0
    for dim_nr in [1, 2, 3]:
        for temp in [-0.5, 0.0, 0.5]:
            case_nr += 1
            print(case_nr, dim_nr, temp)
    return 0
```

case	dim	temp
1	1	-0.5
2	1	0.0
3	1	0.5
4	2	-0.5
5	2	0.0
...		
9	3	0.5

for loop

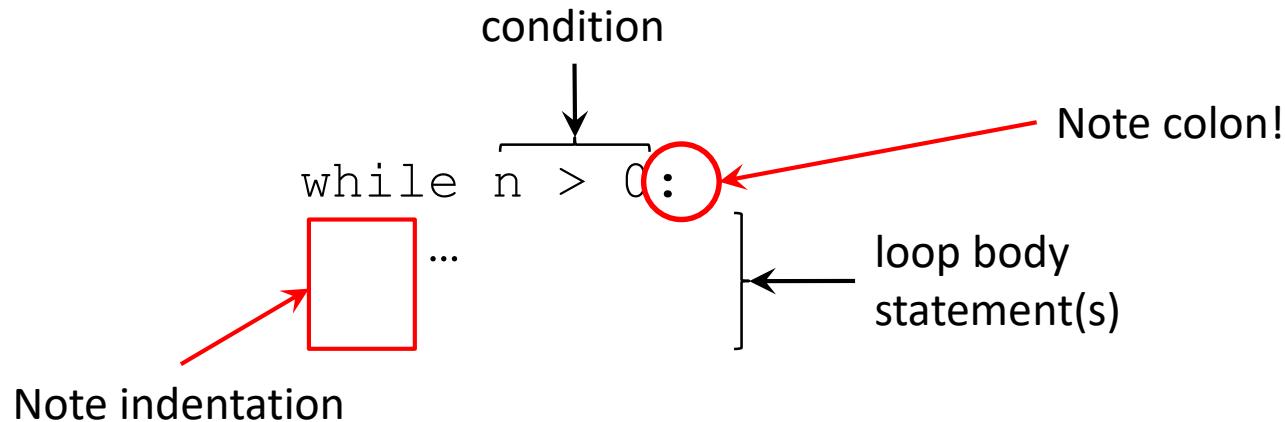
- Semantics: for each element in list do...



- Actually, not only lists, anything one can iterate over (e.g., sets, dictionaries, I/O streams,...)

while loop

- Semantics: while boolean condition holds do...



Skipping and quitting

- Skipping loop iterations: `continue`

```
for n in range(100):  
    if is_prime(n):  
        continue  
    print(n)
```

- Ending loop execution: `break`

```
n = 100  
while n < 1000:  
    if is_prime(n):  
        break  
    n += 1
```

Works for both
`for` and `while` loops

Data types

- str: sequence of characters, e.g., 'temp'
- int: integer, e.g., 2, -1234, 1_203_107
- float: floating point number, e.g., -0.5

3.6+

```
def main():
    print('case', 'dim', 'temp')
    case_nr = 0
    for dim_nr in [1, 2, 3]:
        for temp in [-0.5, 0.0, 0.5]:
            case_nr += 1
            print(case_nr, dim_nr, temp)
```

- complex: complex number, e.g., 1.3 + 4.8j

Lists

- Very useful data structure
- Elements can be of same, or different type
- Literal list
 - `[-0.5, 0.0, 0.5]`
 - `['alpha', 'beta', 'gamma', 'delta']`
- Empty list: `[]` or `list()`
- List constructor
 - `list(range(3))` ≡ `[0, 1, 2]`
 - `list(range(1, 4))` ≡ `[1, 2, 3]`
 - `list(range(1, 8, 2))` ≡ `[1, 3, 5, 7]`
 - `list(range(0, -9, -3))` ≡ `[0, -3, -6]`

Note: explicit list construction can often be avoided,
`range(...)` returns iterable

More list operations

- Example list: `l = ['a', 'b']`
- Number of elements: `len(l) == 2`
- Append to a list:
`l.append('c'), l ≡ ['a', 'b', 'c']`
- Remove last element:
`l.pop() == 'c', l ≡ ['a', 'b']`
- Insert element at position:
`l.insert(1, 'c'), l ≡ ['a', 'c', 'b']`
- Remove element at:
`l.pop(1) == 'c', l ≡ ['a', 'b']`
- Extend a list:
`l.extend(['c', 'd']),
l ≡ ['a', 'b', 'c', 'd']`

Using list elements

- Example list: `l = ['a', 'b', 'c']`
- Use first element: `a = l[0], a == 'a'`
- Use second element: `a = l[1], a == 'b'`

Note: list index is 0-based!

- Use last element: `a = l[-1], a == 'c'`
- One before last: `a = l[-2], a == 'b'`
- Assignment:
`l[2] = 'de', l == ['a', 'b', 'de']`

Slicing & dicing

- Example list: `l = list(range(1, 6))`,
`l ≡ [1, 2, 3, 4, 5]`
- Creating sublists:
 - `l_sub = l[2:4]`, `l_sub ≡ [3, 4]`
 - `l_sub = l[:4]`, `l_sub ≡ [1, 2, 3, 4]`
 - `l_sub = l[2:]`, `l_sub ≡ [3, 4, 5]`
 - `l_sub = l[0:4:3]`, `l_sub ≡ [1, 4]`
 - `l_sub = l[::2]`, `l_sub ≡ [1, 3, 5]`
 - `l_sub = l[4:1:-1]`, `l_sub ≡ [5, 4, 3]`
 - `l_r = l[::-1]`, `l_r ≡ [5, 4, 3, 2, 1]`
- Assigning to slices: `l[::-2] = ['a', 'b', 'c']`,
`l ≡ ['a', 2, 'b', 4, 'c']`

Iterating over lists

- Example list: `data = list(range(1, 6))`
- Straightforward iteration
`for e in data:
 f(e)` 
- Need index?
`for i in range(len(data)):
 g(i, data[i])` 
- Better
`for i, e in enumerate(data):
 g(i, e)` 

Generating data revisited

- Use range (...)
- How to do lists of floats?
 - $[0.5*x \text{ for } x \text{ in } \text{range}(-1, 2)]$
 - list comprehensions: construct list from list

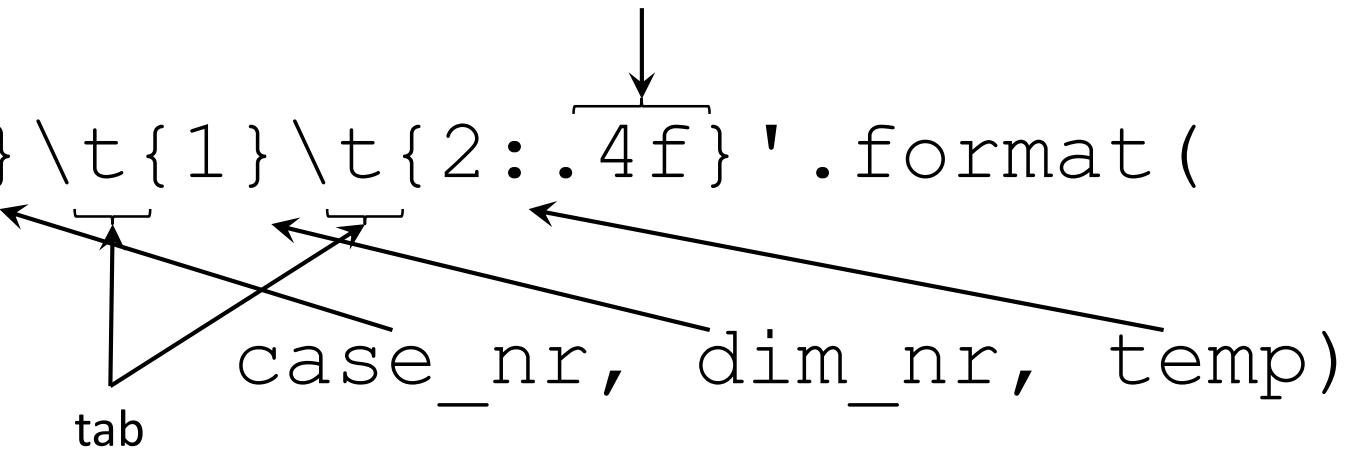
```
def main():
    print('case', 'dim', 'temp')
    case_nr = 0
    for dim_nr in range(1, 4):
        for temp in [0.5*x for x in range(-1, 2)]:
            case_nr += 1
            print(case_nr, dim_nr, temp)
    return 0
```

consider using numpy

Formatting strings

- Use tabs as separator
- Increase number of digits after decimal point to 4

```
' {0} \t{1} \t{2:.4f}'.format(  
    case_nr, dim_nr, temp)
```



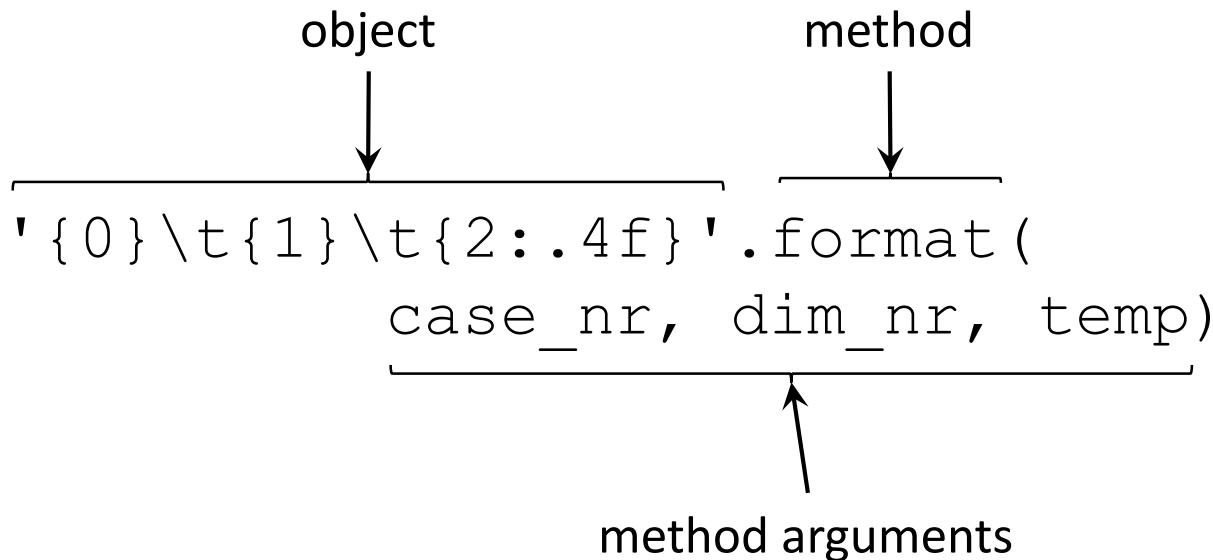
- f-string:

```
f'{case_nr}\t{dim_nr}\t{temp:.4f}'
```

3.6+

Objects & methods

- a string is an object (class str), format is a method on that object



methods on strings produce new strings

Modifying data

- Replace negative temperatures by 0.0

```
case dim temp
1 1 -0.5
2 1 0.0
3 1 0.5
4 2 -0.5
5 2 0.0
...
```



```
case dim temp
1 1 0.0000
2 1 0.0000
3 1 0.5000
4 2 0.0000
5 2 0.0000
...
```

```
import sys
def main():
    print(sys.stdin.readline().rstrip('\r\n'))
    for line in sys.stdin:
        data = line.rstrip('\r\n').split()
        if float(data[2]) < 0.0:
            data[2] = '0.0'
        print('{0} {1} {2:.4f}'.format(
            data[0], data[1], float(data[2])))
```

Code Pack 03

A. Python fundamentals:

1. Primitive Datatypes and Operators
2. Variables and Collections

Coding Bootcamp Code in Python

PYTHON FUNDAMENTALS CONTINUED

Some more str methods: strip

- Getting rid of line endings:

```
rstrip(' \r\n')
```

- method will strip all combinations of \r and \n from right end of string

- Similar methods:

- lstrip: strips from left end of string
 - strip: strips from both ends of string

- no arguments, strips: space, \t, \r, \n, ...

Note that strings are not modified, new string is created!

str method: split

- Splitting string: `split()` returns list of strings
 - no argument: split on (multiple) whitespace
 - otherwise, split on provided string
 - limit number of splits by providing extra argument
- E.g., read file, and print only end times

```
start 1: 2013-03-27 14:20:13
end 1: 2013-03-28 03:05:57
start 2: 2013-03-28 04:30:17
start 3: 2013-03-28 04:30:17
end 2: 2013-03-28 05:45:17
end 3: 2013-03-28 09:15:38
...
```

Split on ':', but note
time format!!!

More str methods: startswith, endswith

- `startswith(prefix)`, `endswith(suffix)`
return True if str starts with prefix/ends with suffix respectively, False otherwise

```
...
for line in sys.stdin:
    event_str = line.strip()
    if event_str.startswith('end'):
        event, time_str = event_str.split(':', 1)
        print(time_str)
```

Only single split, otherwise time is split as well



Even more str methods: is<something>

- Test str **is uppercase/lowercase**:
`s.isupper()/s.islower()`
 - `'ABC'.isupper() == True`
 - `'A19'.isupper() == True`
 - `'Abc'.isupper() == False`
 - `'19'.isupper() == False`
- Test str **has only whitespace**: `s.isspace()`
- Test str **has only digits**: `s.isdigit()`
- Test str **is alphabetic/alphanumeric**:
`s.isalpha()/s.isalnum()`

Searching & replacing in str

- Does str contain substring?
`('ab' in 'ABabCD') == True`
- Find position of first occurrence of substring in str:
`'ABabCD'.find('ab') == 2`
 - returns -1 when not found
 - can search between given start and final position
- Replace all occurrences of substring by other substring
`'3.14'.replace('.',' ',1) == '3,14'`
 - maximum number of replacements can be specified

More methods, but this will do

str operations

- Concatenating strings:

`'abc' + 'def' == 'abcdef'`

- Works for list as well

`[0, 1] + [3, 4] == [0, 1, 3, 4]`

- Multiplying strings:

`'x' * 4 == 'xxxx'`

- Works for list as well

`[0.0] * 4 == [0.0, 0.0, 0.0, 0.0]`

- However, bear in mind that this may *not* always do what you think

Joining list elements

- Often, data contained in list data structure
 - Needs to be represented as delimited string
 - Example:
[3.1745, 18.14, -6.49043]
→ 3.1745, 18.14, -6.49043
- Use list comprehension, str function and str's join(...) method

```
>>> data = [3.1745, 18.14, -6.49043]
>>> print(', '.join([str(number) for number in data]))
3.1745, 18.14, -6.49043
```



type

str & list are sequences

- characters (elements for list) accessed by position, e.g., $s = 'abc'$:
 $s[0] == 'a'$, $s[2] == 'c'$,
 $s[-1] == 'c'$, $s[-2] == 'b'$
- Substrings (slices for list), e.g.,
 $s = 'abcde'$:
 $s[0:3] == 'abc'$, $s[2:4] == 'cd'$,
 $s[1:] == 'bcde'$, $s[:3] == 'abc'$
 $s[::-1] == 'edcba'$

str & list length revisited

- Function `len()` computes str length (number of elements for list)

`len(' ') == 0, len('abc') == 3`

`len([]) == 0, len([3, 5, 7]) == 3`

- Length & truth
 - Empty string is False, non-empty string True
 - Empty list is False, non-empty list is True

```
...  
if len(line.strip()) > 0:  
    ...
```

≡

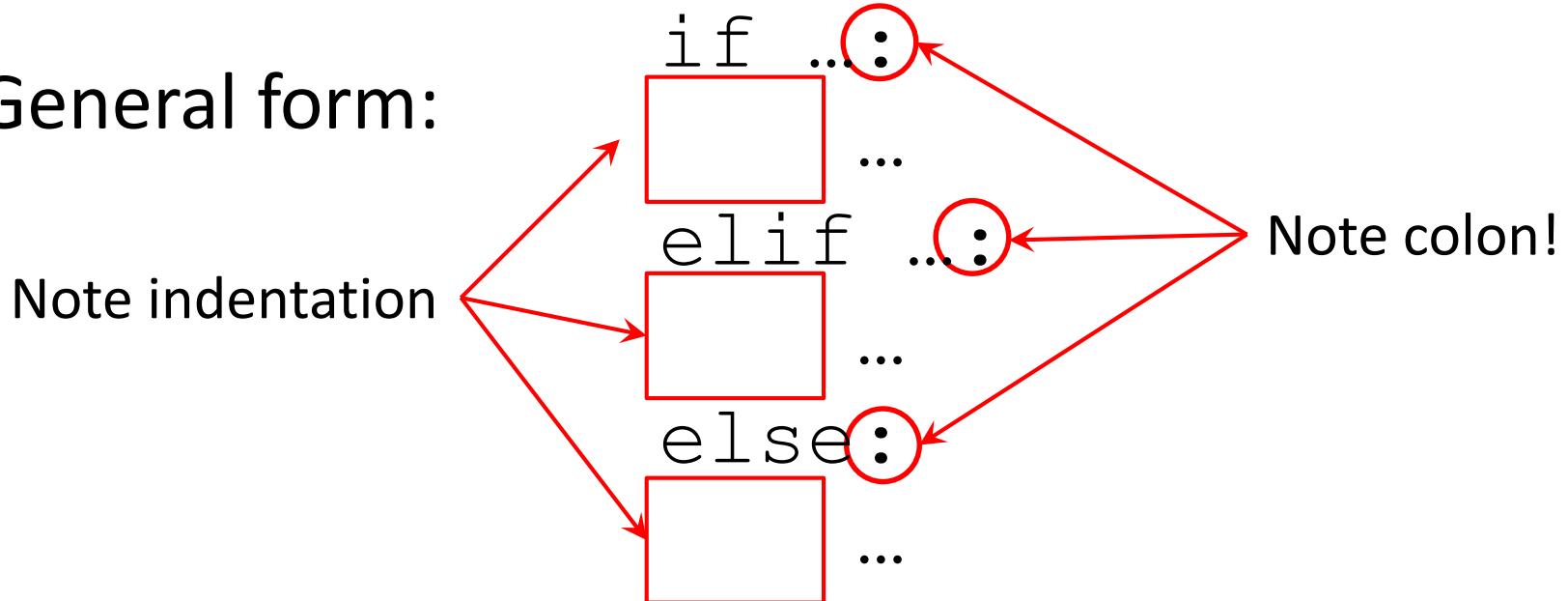
```
...  
if line.strip():  
    ...
```

Type conversion

- Convert str s to floating point: `float(s)`
 - necessary for comparison:
`float(data[2]) < 0.0`
- Convert str s to integer: `int(s)`
- Convert number x to str: `str(x)`
- Convert float x to integer: `int(x)`
 - takes integer part of float, e.g., `int(-3.8) == -3`
- Determining type of expression e: `type(e)`
 - e.g., `type(3 + 0.1) == float`

if statement

- General form:



- Nesting: structure through indentation
- Conditional expression:

```
...
n = 10 if r > 1.0 else 0
...
```

Conditionals

- Boolean values: True, False
- Boolean operators: not, and, or
- Comparison operators: ==, !=, <, <=, >, >=
 - work on str, float, int, ...
- List membership: in, e.g.,
 - 'a' in ['c', 'a', 'd'] == True
 - 'e' not in ['c', 'a', 'd'] == True

Which dimension numbers?

- Which dimension numbers occur in file?

```
import sys
def main():
    sys.stdin.readline()
    dim_nrs = set()
    for line in sys.stdin:
        dim_nrs.add(int(line.rstrip('\r\n').split()[1]))
    print(dim_nrs)
    return 0
```

Yuck, what's that?!?

```
dim_nr = int(line.rstrip('\r\n').split()[1])
```

|||

```
line_str = line.rstrip('\r\n')
data     = line_str.split()
dim_str  = data[1]
dim_nr   = int(dim_str)
```

Python can be terse, but stick to what's comfortable for you!

However, use functions...

Reasonable compromise

- One additional variable simplifies code

```
import sys
def main():
    sys.stdin.readline()
    dim_nrs = set()
    for line in sys.stdin:
        data = line.rstrip('\r\n').split()
        dim_nrs.add(int(data[1]))
    print(dim_nrs)
    return 0
```

Sets

- set is Python data type, acts like set in math
 - empty set: `s = set()`
 - number of elements: `len(s)`
 - add element: `s.add('a')`
 - check membership: `'b' in s`
 - remove element: `s.remove('b')`, `s.discard('b')`
 - remove and return arbitrary element: `s.pop()`
 - iterating over elements:

```
for element in s:  
    ...
```
- No set of sets, set of lists
- Set comprehensions:

```
{i for i in range(3)} ≡ {0, 1, 2}
```

Set operations

```
s1 = {3, 5, 7}
```

```
s2 = {7, 11}
```

- **Intersection:** $s1 \& s2$
`s1.intersection(s2) == {7}`
- **Union:** $s1 | s2$
`s1.union(s2) == {3, 5, 7, 11}` $s1 |= s2$
- **Difference:** $s1 - s2$
`s1.difference(s2) == {3, 5}`
- **Symmetric difference:** $s1 ^ s2$
`s1.symmetric_difference(s2) == {3, 5, 11}`
- **Is subset of?** $s1 \leq s2$
`s1.issubset(s2) == False`
- **Is disjoint from?**
`s1.isdisjoint(s2) == False`

To modify set, use:

```
s1.<op>_update(s2)
```

For union, use:

```
s1.update(s2)
```

More modularity

- Same code copied and pasted, modified

```
...
for line in sys.stdin:
    data = line.rstrip('\r\n').split()
    dim_nr = int(data[1])
...
...
```

- Make it generic

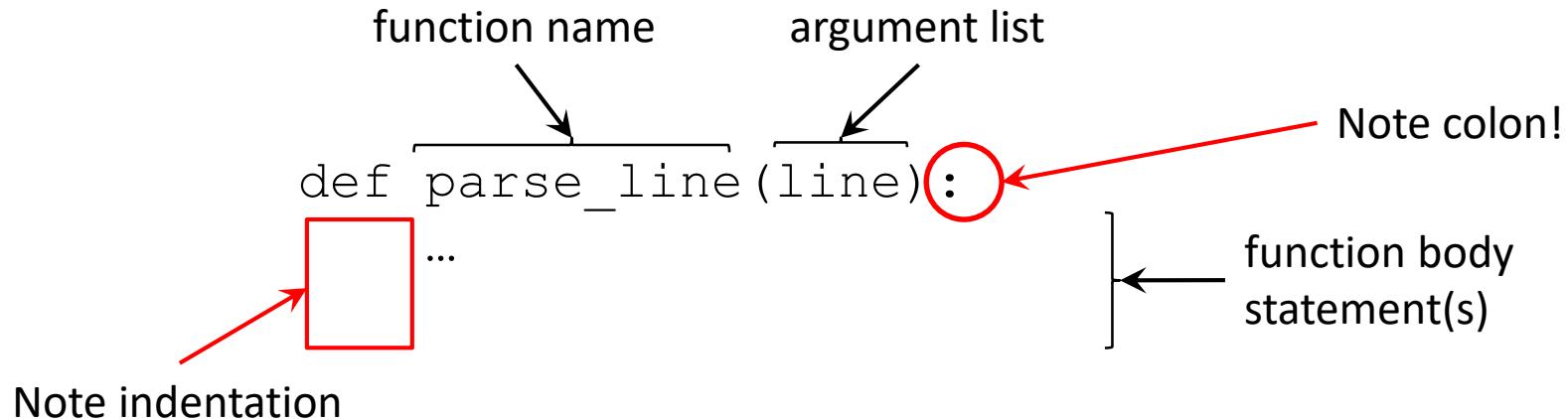
```
def parse_line(line):
    data = line.rstrip('\r\n').split()
    return (int(data[0]), int(data[1]), float(data[2]))
...
for line in sys.stdin:
    case_nr, dim_nr, temp = parse_line(line)
...
...
```

Functions

- Call by reference
 - however, remember that str, int, float et al. are immutable
- Arguments can have default values
- Arguments can be positional, or by keyword
- Higher order
 - functions can have functions as arguments
 - function can return functions (closures)

Anatomy of function definition

- Function definition



- return statement to... return results, if any, and return control to caller

Adding flexibility

- Optional column separator

argument separator default value

```
def parse_line(line, sep=None):  
    data = line.rstrip('\r\n').split(sep)  
    return (int(data[0]), int(data[1]), float(data[2]))
```

...

```
for line in sys.stdin:  
    (case_nr, dim_nr, temp) = parse_line(line)
```

...

call with single argument,
use default for sep (i.e., None)

...

```
(case_nr, dim_nr, temp) = parse_line(line, sep=';')
```

...

Default value pitfall

```
def filter_pos(new_values, values=[]):  
    for new_value in new_values:  
        if new_value > 0:  
            values.append(new_value)  
    return values
```

```
if __name__ == '__main__':  
    value_list = [  
        [1, -3, 5],  
        [13, 33, -15],  
    ]  
    for values in value_list:  
        print(f'filtering {values}')  
        filtered_values = filter_pos(values)  
        print(f'filtered: {filtered_values}')
```

```
def filter(new_values, values=None):  
    if values is None:  
        values = []  
    ...
```



default values are
created on import,
reused for calls:
mutable types == surprise!

```
filtering [1, -3, 5]  
filtered: [1, 5]  
filtering [13, 33, -15]  
filtered: [1, 5, 13, 33]
```

Tuples (YADS ☺)

- tuple is (kind of) fixed length list, *immutable*
- tuple with two elements: `t = ('a', 'b')`
 - first element: `t[0] == 'a'`, second element
`t[1] == 'b'`

```
def parse_line(line, sep=None):  
    data = line.rstrip('\r\n').split(sep)  
    return (int(data[0]), int(data[1]), float(data[2]))  
...  
for line in sys.stdin:  
    case_nr, dim_nr, temp = parse_line(line)  
    ...
```

tuple of int, int, float

1-tuple: ('a',)

3-tuple unpacked into 3 variables

Returning to dimension numbers...

- Which dimension numbers occur in file?

```
...
def main():
    _ = sys.stdin.readline()
    dim_nrs = set()
    for line in sys.stdin:
        _, dim_nr, _ = parse_line(line)
        dim_nrs.add(dim_nr)
    for dim_nr in dim_nrs:
        print(dim_nr)
```

`_` is wildcard in tuple unpacking:
tuple elements at those positions are ignored

Named tuples, Python 2.6+

- `collections.namedtuple` *is* tuple,
but elements have names

```
from collections import namedtuple  
...  
Line Data = namedtuple('Line Data', 'case nr dim nr temp')  
  
def parse_line(line, sep=None):  
    data = line.rstrip('\r\n').split(sep)  
    return Line Data(case_nr=int(data[0]),  
                     dim_nr=int(data[1]),  
                     temp=float(data[2]))  
  
...  
for line in sys.stdin:  
    line_data = parse_line(line)  
    dim_nrs.add(line_data.dim_nr)  
    ...
```

Annotations:

- type name**: Points to the class name `Line Data` in the first line.
- element names**: Points to the field names `'case nr dim nr temp'` in the second line.
- constructor**: Points to the call to the constructor `Line Data` in the third line.
- access by name**: Points to the attribute `dim_nr` in the third line.

Named tuples, Python 3.5+

- `typing.NamedTuple` *acts as* tuple, but
 - elements have names
 - elements have type hints
 - can have methods
 - can serve as base class

```
from typing import NamedTuple  
...  
class Line_Data(NamedTuple):  
    case_nr: int  
    dim_nr: int  
    temp: float
```

The diagram illustrates the structure of a Named Tuple. It shows a code snippet with annotations:

- A green box labeled "type name" contains the class name `Line_Data`.
- A blue box labeled "element names + type hints" contains the three field definitions: `case_nr: int`, `dim_nr: int`, and `temp: float`.

Using named tuples

```
...
def parse_line(line, sep=None):
    data = line.rstrip('\r\n').split(sep)
    return Line_Data(case_nr=int(data[0]),
                     dim_nr=int(data[1]),
                     temp=float(data[2]))
...
for line in sys.stdin:
    line_data = parse_line(line)
    dim_nrs.add(line_data.dim_nr)
...
...
```

element values
can be specified
by name in
any order

access by name

Counting dimension numbers

- How many times does a dimension number occur in file?
 - maximum & minimum not known a-priori!

```
...
import sys
def main():
    _ = sys.stdin.readline()
    counter = dict()
    for line in sys.stdin:
        line_data = parse_line(line)
        if line_data.dim_nr not in counter:
            counter[line_data.dim_nr] = 0
        counter[line_data.dim_nr] += 1
    for dim_nr, count in counter.items():
        print('{0}: {1}'.format(dim_nr, count))
```

Dictionaries

- Data structure that maps a key onto a value
 - e.g., map a name to an age

```
ages = {  
    'alice': 35,  
    'bob': 32,  
}  
key, value separated by colon
```

Curly brackets for dict
key/value pair separated by comma

key
value

- Keys can have any (hashable) type (mixed too)
- Values can have any type (mixed too)
- Dictionary comprehensions:
$$\{ k: k**2 \text{ for } k \text{ in range}(3) \} \equiv \{ 0: 0, 1: 1, 2: 4 \}$$

Using dictionaries

```
ages = {  
    'alice': 35,  
    'bob': 32,  
}
```

- Empty dictionary: `{ }` or `dict()`
- Number of key/value pairs: `len(ages)`
- Storing values
`ages['caro'] = 45`
- Retrieving values
`35 == ages['alice']`
- Removing key/value, and return value
`35 == ages.pop('alice')`
- Does `ages` have an age for '`dave`'?
`ages.has_key('dave') ≡ 'dave' in ages`

Iterating over dictionaries

- Iterate over keys:

```
for name in ages.keys():
```

...

```
for name in ages:
```

...

- Iterate over values:

```
for age in ages.values():
```

...

- Iterate over key/value pairs:

```
for name, age in ages.items():
```

...

Note:
creates views

Python 3.6+ *implementation*: keys in insertion order!

Counting again...

- Using `collections.Counter` instead of `dict`: simpler, less error prone

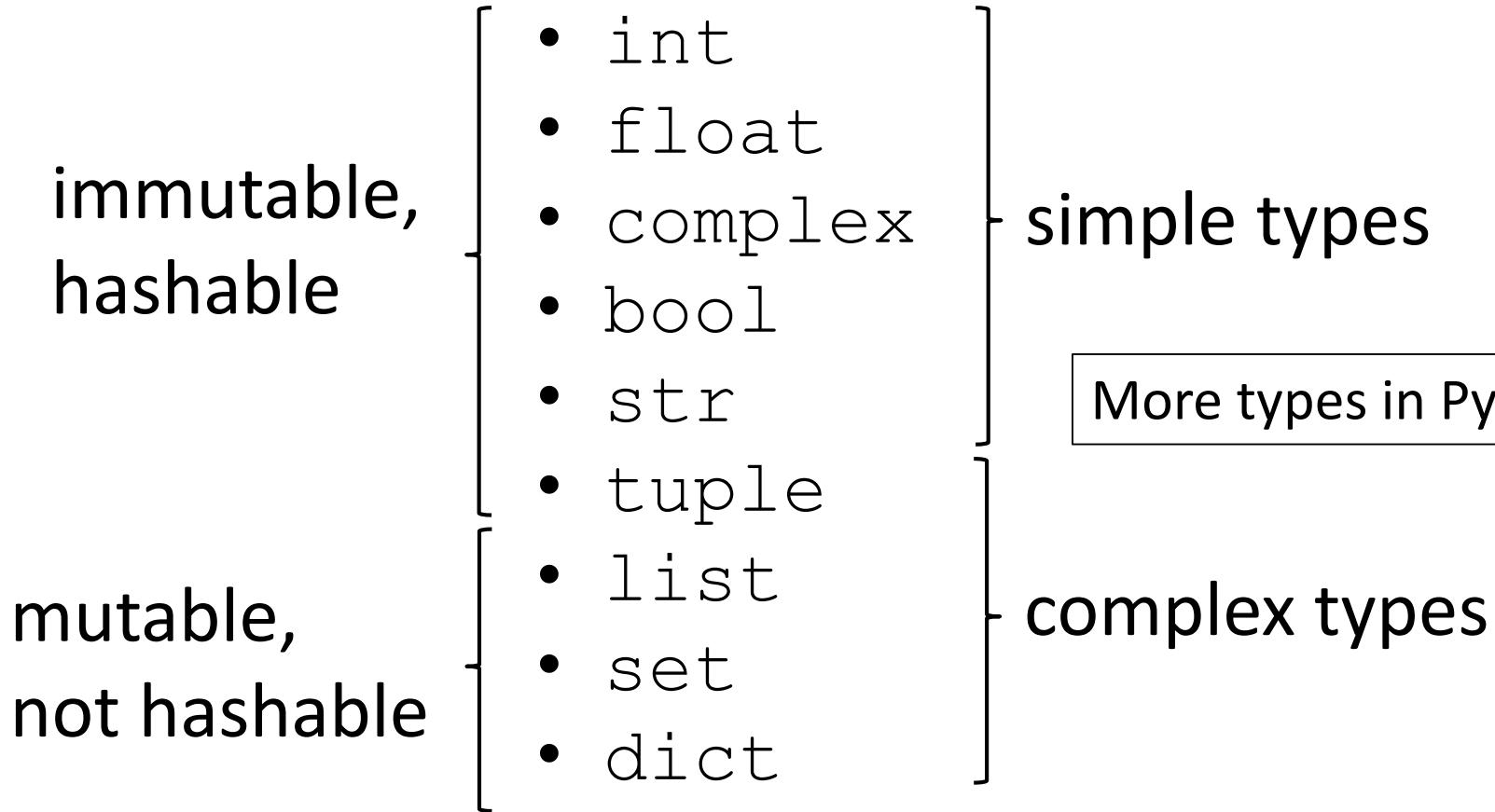
```
...
import collections
import sys
def main():
    _ = sys.stdin.readline()
    counter = collections.Counter()
    for line in sys.stdin:
        line_data = parse_line(line)
        counter[line_data.dim_nr] += 1
    for dim_nr, count in counter.items():
        print('{0}: {1}'.format(dim_nr, count))
```

bonus method: `most_common()`

More special data types

- `collections.namedtuple`: tuples with named elements
- `collections.Counter`: count elements
- `collections.OrderedDict`: remembers insertion order
- `collections.deque`: (bounded) double-ended queue
- `collections.defaultdict`: dictionary with computed default values
- `array.array`: faster than lists, however, use numpy

Summary: data types



Picking the right data type is crucial to produce good code

Summary: control structures

- Conditional statement:

```
if ...:  
    ...  
elif ...:  
    ...  
else:  
    ...
```

- Iteration statements:

- for-loop:

```
for ... in ...:  
    ...
```
- while-loop:

```
while ...:  
    ...
```

Summary: mathematics

- Usual operators: +, -, *, /, %
 - for int, division is floating point division, i.e., `3/5 == 0.6`
- Raise to power: **
 - e.g., `2**4 == 16`
- Floor division: //
- Mathematical functions in module math
 - First import module (usually at top of file):

```
import math
```

Use functions, e.g., `math.sqrt(3.0)`
 - Or import specific function(s):

```
from math import sqrt
```

Use function(s), e.g., `sqrt(3.0)`
- For complex numbers, functions in cmath

changed from 2.x to 3.x!

Code Pack 04

A. Python fundamentals:

1. ~~Primitive Datatypes and Operators~~
2. ~~Variables and Collections~~
3. Control Flow and Iterables
4. Functions

B. Port Scanning

Coding Bootcamp Code in Python

CODE ORGANIZATION

Python modules & packages

- Code organization
 - Functions common to multiple scripts can be put in separate file = module
 - Modules can be organized hierarchically in directory structure = packages

Don't forget `__init__.py` in package directories!

- Python standard library is organized in packages

Example module & use

- Module file:

```
from collections import namedtuple  
Line_Data = namedtuple('Line_Data', 'case_nr dim_nr temp')  
  
def parse_line(line, sep=None):  
    data = line.rstrip('\r\n').split(sep)  
    return Line_Data(case_nr=int(data[0]),dim_nr=int(data[1]),  
                     temp=float(data[2]))
```

- Using the module in script:

```
import data_parsing  
def main():  
    ...  
    for line in sys.stdin:  
        line_data = data_parsing.parse_line(line)  
        ...
```

Importing functions directly

- Importing function `parse_line` from module `data_parsing` in script `counting.py`:

counting.py

```
...
from data_parsing import parse_line

def main():
    ...
    for line in sys.stdin:
        data = parse_line(line)
        ...
    ...


```

More concise, but name clashes can occur!
E.g., `math.sqrt` versus `cmath.sqrt`

```
from math import sqrt
from cmath import sqrt as csqrt
```

Double duty

data_parsing.py

```
from collections import namedtuple
Line_Data = namedtuple('Line_Data',
                      ['case_nr', 'dim_nr', 'temp'])

def parse_line(line, sep=None):
    data = line.rstrip('\r\n').split(sep)
    return Line_Data(case_nr=int(data[0]),
                      dim_nr=int(data[1]),
                      temp=float(data[2]))

if __name__ == '__main__':
    ...
    for line in sys.stdin:
        line_data = parse_line(line)
    ...
```

}

Only executed when
run as script

Package layout & use example

➤ weave.py

➤ vsc

➤ __init__.py

➤ util.py

➤ parameter_weaver

➤ __init__.py

➤ artifact.py

➤ base_formatter.py

➤ c

➤ __init__.py

➤ formatter.py

➤ ...

```
...
from vsc.parameter_weaver.c.formatter import Formatter
...
```

➤ fortran

➤ __init__.py

➤ formatter.py

➤ ...

➤ ...

```
...
from vsc.parameter_weaver.base_formatter import BaseFormatter
...
```

Code Pack 05

A. Enter VS Code

B. Python fundamentals:

1. ~~Primitive Datatypes and Operators~~
2. ~~Variables and Collections~~
3. ~~Control Flow and Iterables~~
4. ~~Functions~~
5. Modules

Coding Bootcamp Code in Python

OBJECT-ORIENTED PYTHON

Object-orientation

- Python types are classes
 - e.g., `(14).bit_length() == 4`
 - 14 is an object of class `int`
 - `bit_length` is object method defined in class `int`
- You are using objects all the time!
- Objects of simple Python types are immutable
 - Operations/methods instantiate new objects

Value versus object identity

- Simple Python types
 - Value identity: `(14 == 14) == True`
 - Object identity: `(14 is 14) == True`
 - However, Python version dependent!
- Other Python types, general classes
 - e.g., two set objects:

```
a = {'alpha'}, b = {'alpha'}
```

 - Value identity: `(a == b) == True`
 - Object identity: `(a is b) == False`

Defining your own classes

- Class definition:

```
class Point:
```

...

- Objects are instances of classes

- instantiated by calling constructor
- have

- attributes
- methods

- Classes have

- attributes
- methods

A simple point...

```
from math import sqrt
```

```
class Point:
```

```
    def __init__(self, x, y):  
        self.x = float(x)  
        self.y = float(y)
```

```
    def distance(self, other):  
        return sqrt((self.x - other.x)**2 +  
                    (self.y - other.y)**2)
```

```
    def __str__(self):  
        return f'({self.x}, {self.y})'
```

} constructor for
Point objects

} method to
compute
distance

creates string representation for Point object

Making a point... or two

```
...
def main():
    p = Point(3, 4)           create Point p at 3, 4
    q = Point(-2, 5)          create Point q at -2, 5
    print(p.x, p.y)          access p's x- and y-coordinates
    print(p, q)               calls __str__ method indirectly
    print(p.distance(q))     on p and q
    p.x = 12.3                compute distance from p to q
    print(p) ...              modifying p
```

```
$ python point_driver.py
3.0 4.0
(3.0, 4.0) (-2.0, 5.0)
5.0990195136
(12.3, 4.0)
```

distance method invoked on Point p, with Point q as argument

More to the point...

- What if points should not be moved?

```
class Point:  
  
    def __init__(self, x, y):  
        self.__x = float(x)  
        self.__y = float(y)  
  
    @property  
    def x(self):  
        return self.__x  
  
    @property  
    def y(self):  
        return self.__y  
  
...
```

} constructor for
Point objects

} getter for object's
__x attribute

} getter for object's
__y attribute

Making a definite point

```
...  
def main():  
    p = Point(3, 4)  
    print(p.x, p.y)  
    p.x = 12.3  
...  
  
create Point p at 3, 4  
try to access p's x-coordinate
```

```
$ python point_driver.py  
3.0 4.0  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
AttributeError: can't set attribute
```

Object attributes

- Make object attributes "private" by hiding them, by convention, use `_` prefix
`self._x = x`
- Create getter/setter method to control access to object attributes

```
@property  
def x(self):  
    return self._x
```

Determine object's state

Object attribute can not accidentally be modified, i.e., read-only

Object attributes: control

- Getter, but no setter

```
...
def main():
    p = Point(3, 4)
    print(p.x)
    p.x = 4.4
    print(p.x)
...
...
```

Protects against modification
of read-only attributes

```
$ python point_driver.py
3.0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
```

Object attribute: setter

- Implementing setters improves control, assignment to attribute is "intercepted" by setter method

```
class Point:  
...  
    @x.setter  
    def x(self, value):  
        self.__x = float(value)  
...
```

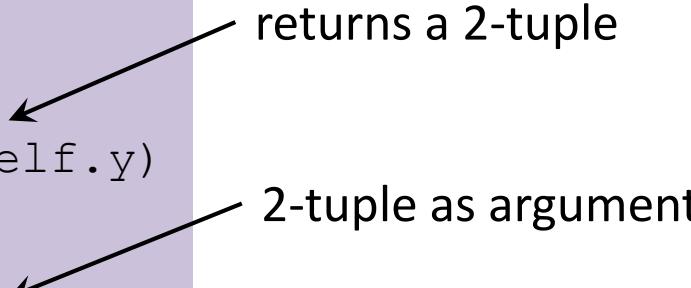
E.g., ensures proper type conversion:

p.x = 3 results in float, not int for __x attribute

Non-trivial getter/setter

- Derived attribute: coordinates as 2-tuple

```
class Point:  
    ...  
    @property  
    def coords(self):  
        return (self.x, self.y)  
  
    @coords.setter  
    def coords(self, value):  
        self.x = value[0]  
        self.y = value[1]  
    ...  
    # Use coords getter/setter  
    print(p.coords)  
    p.coords = (3.5, 7.1)
```



More object methods

```
from math import sqrt, isclose
class Point:
    ...
    def on_line(self, p, q, tol=1.0e-6):
        if not isclose(p.x, q.x, abs_tol=tol):
            a = (q.y - p.y) / (q.x - p.x)
            b = p.y - a*p.x
            return isclose(self.y, a*self.x + b, abs_tol=tol)
        else:
            return isclose(self.x, p.x, abs_tol=tol)
    ...
# check whether r is on line defined by p and q
if r.on_line(p, q):
    ...
```

Python 3.5+

on_line method invoked on Point r, with Point p and q as argument

Object methods

- Used to
 - retrieve information on object
 - modify or manipulate object
 - derive information from object with respect to other objects
 - ...

Determine what objects can do, or can be done with

Static methods

```
...
class Point:
    ...
    @staticmethod
    def all_on_line(p, q, *points):
        for r in points:
            if not r.on_line(p, q):
                return False
        return True
    ...
# check whether p, q, r, v and w are on a line
if Point.all_on_line(p, q, r, v, w):
    ...
...
```

all_on_line method invoked on Point class
with Point p, q, r, v, w as arguments, class ignored

Variable length argument lists

- Arbitrary positional arguments: *args

```
@staticmethod  
def all_on_line(p, q, *points):  
    for r in points:  
        if not r.on_line(p, q):  
            return False  
    return True
```

arguments

available as tuple

- Arbitrary keyword arguments: **kwargs
 - Available as dictionary

Note: not specific to object oriented programming

More elegant solution

- Semantics: True if True for all elements in points

```
@staticmethod  
def all_on_line(p, q, *points):  
    for r in points:  
        if not r.on_line(p, q):  
            return False  
    return True
```

- More elegant: all (...)

```
@staticmethod  
def all_on_line(p, q, *points):  
    return all(r.on_line(p, q) for r in points)
```

- Similar: any (...)

Quick interlude

- What attributes/methods does a class have?

```
>>> from point import Point
>>> p = Point(3.7, 5.1)
>>> dir(p)
['__class__', '__delattr__', '__dict__', '__doc__',
 '__format__', '__getattribute__', '__hash__',
 '__init__', '__module__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__',
 '__sizeof__', '__str__', '__subclasshook__',
 '__weakref__', '__Point__x', '__Point__y',
 'all_on_line', 'coords',
 'distance', 'on_line', 'x', 'y']
```

Inheritance

- Class can extend other class
- For Python 2: make classes inherit from `object`, ensure they can be extended later:
`class Point (object) :`
- New class inherits attributes & methods from parent class
- New class can implement new methods, define new attributes
- New method can override methods of parent class
- New class can inherit from multiple parent classes

Points with mass

```
class PointMass(Point):  
  
    def __init__(self, x, y, mass):  
        super().__init__(x, y)  
        self.__mass = float(mass)  
  
    @property  
    def mass(self):  
        return self.__mass  
  
    def __str__(self):  
        return '{0}: {1}'.format(  
            super().__str__(),  
            self.mass)
```

} constructor
} of Point
} overridden
}
} new object
} method
}
} str method
} of Point
} overridden

PointMass objects have x, y, distance, on_line methods as well
PointMass class has all_on_line methods

Base classes & derivation

```
class PointMass(Point):  
  
    def __init__(self, x, y, mass):  
        super().__init__(x, y)  
        self.__mass = float(mass)
```

create Point is base class for PointMass

first call Point's __init__ method

do PointMass-specific initialization

Point with mass is still Point

```
def main():
    p = PointMass(3, 4, 1)
    q = Point(-2, 5)
    print(p.x, p.y, p.mass)
    print(p.distance(q))
```

create PointMass p at 3, 4
and mass 1

create Point q at -2, 5

p is a Point, so has distance method

```
$ python point_driver.py
3.0 4.0 1.0
5.09902
```

Class attributes

```
class PointMass(Point):  
    __default_mass = 1.0  
  
    def __init__(self, x, y, mass=None):  
        super().__init__(x, y)  
        if mass is not None:  
            self.__mass = float(mass)  
        else:  
            self.__mass = PointMass.__default_mass  
  
    ...  
  
    @classmethod  
    def set_default_mass(cls, mass):  
        cls.__default_mass = float(mass)
```

} class variable
__default_mass

} setter for class'
__default_mass
attribute

Determine state of class

All those methods

- Object methods
 - work on individual objects
 - take object as first argument (`self`)
- Class methods
 - `@classmethod`
 - work at class level
 - take class as first argument (`cls`)
 - `@staticmethod`
 - work at class level
 - ignores object or class it is called on

Code Pack 06

- A. Python fundamentals:
 - ~~1. Primitive Datatypes and Operators~~
 - ~~2. Variables and Collections~~
 - ~~3. Control Flow and Iterables~~
 - ~~4. Functions~~
 - ~~5. Modules~~
 - 6. Classes

Coding Bootcamp Code in Python

GETTING THINGS IN AND OUT: I/O & COMMAND LINE ARGUMENTS

Reading lines from file handles

- Standard file handles:
 - `sys.stdin`: standard input (keyboard, pipe in)
 - `sys.stdout`: standard output (screen, pipe out)
 - `sys.stderr`: standard error (screen, pipe out)
- Reading a single line:
 - `sys.stdin.readline()`: returns str
- Reading all lines at once:
 - `sys.stdin.readlines()`:
returns list of str

Note: line endings,
e.g., `\n` or `\r\n` are
included

Note: `readline()`, `readlines()` are methods on file handles

Reading & memory consumption

- Remember, `readlines()` method reads whole file at once
 - For large files, creates long list = lots of memory
- Avoid:

```
...  
for line in sys.stdin.readlines():  
    ...
```

- Use:

```
...  
for line in sys.stdin:  
    ...
```

Returns iterator, not list
Memory friendly!

Writing to file handles

- print function writes objects to sys.stdout, adds '\n' (or '\r\n') and applies str() conversion function by default
- write (...) method writes str to file handle, e.g.,
 - sys.stderr.write('## error: number is negative\n')
 - sys.stdout.write(output_str)
- flush () method flushes output to disk
 - At least, tells OS to do so

More on print

- `print` has some useful optional arguments
 - `file`: allows to print to any open file handle, e.g.,
`sys.stderr` (**default**: `sys.stdout`)
`print("# error: number should be positive",
 file=sys.stderr)`
 - `sep`: character to separate multiple objects to print
(**default**: ' '), e.g.,
`print('alpha', 3, 5.7, sep='\t')`
 - `end`: character to add when all arguments are printed
(**default**: '\n'), e.g.,
`print('next print will be on same line',
 end='')`
 - `flush`: whether to combine print with a flush on the file
handle (**default**: `False`),
`print('read done', file=sys.stderr,
 flush=True)`

Simple command line arguments

- Script name & command line arguments in `sys.argv`

```
import sys

if __name__ == '__main__':
    print(sys.argv)
```

```
$ python cla_printer.py
['cla_printer.py']
$ python cla_printer.py alpha beta 3.5
['cla_printer.py', 'alpha', 'beta', '3.5']
$ python cla_printer.py 'alpha beta' 3.5
['cla_printer.py', 'alpha beta', '3.5']
```

Note:
all values
are str

Okay for very simple cases, better: use argparse

Code Pack 07

- A. my_repl.py
- B. bot_create_a_story.py
- C. distance.py

Coding Bootcamp Code in Python

WRITING DOCUMENTATION & SIMPLE TESTING

Writing documentation

- Documentation is very important!
 - use DocString

```
def parse_line(line, sep=None):  
    '''Split a line into its fields, convert to the  
    appropriate types, and return as a tuple.'''  
    # using \r, \n should work for Windows & *nix  
    data = line.rstrip('\r\n').split(sep)  
    return (int(data[0]), int(data[1]), float(data[2]))
```

```
>>> import data_parsing  
>>> help(data_parsing.parse_line)  
Help on function parse_line in module validator:  
  
parse_line(line)  
    Split a line into its fields, convert to the  
    appropriate types, and return as a tuple
```

Formatting docstrings

```
def parse_line(line, sep=None):
    '''Split line into fields,
       converted to appropriate types

    Parameters
    -----
    line: str
        line of input to parse
    sep: str
        field separator, default
        whitespace

    Returns
    -----
    tuple (int, int, float)
        data fields: case number,
        dimension number, temperature
    '''

    ...

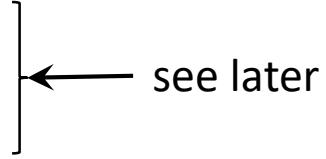
```

Many options

- Google
- reStructured Text
- numpy/scipy

numpy/scipy

What to document and how?

- DocString for
 - functions
 - classes
 - methods
 - modules
 - packages
 - Comments
 - particular code fragments you had to think about
- 
- A bracket diagram consisting of a vertical line with a horizontal arrow pointing left from the text "see later". The bracket is positioned to the right of the first two list items and spans the height of the entire list.
- see later

Assertions

- Testing pre and post conditions
 - Programming by contract

```
def fac(n):  
    assert type(n) == int, 'argument must be integer'  
    assert n >= 0, 'argument must be positive'  
    if n < 2:  
        return 1  
    else:  
        return n*fac(n - 1)
```

Optional

```
$ python -c 'from fac import fac; print(fac(-1))'  
...  
assert n >= 0, 'argument must be positive'  
AssertionError: argument must be positive
```

Assert use cases

- For development only, *not* production!
- *Not* a substitute for error handling, i.e., exception handling
- Run without assertions, run optimized: -O

```
$ python -O -c 'from fac import fac; print(fac(-1))'  
1
```

Useful feature, but don't abuse!

Testing: meeting expectations

- Tests are important!
 - unittest: more features, but harder
 - **doctest**: simple

A program that has not been tested does not work.
— Bjarne Stroustrup

```
def parse_line(line):  
    '''Split a line into its fields, convert to the  
    appropriate types, and return as a tuple.  
    >>> parse_line('5 3 3.7')  
    (5, 3, 3.7)  
    '''  
  
    data = line.rstrip('\r\n').split()  
    return (int(data[0]), int(data[1]), float(data[2]))
```

Statement
to execute

Expected result

- Run tests

No output: hooray, all tests passed!

```
$ python -m doctest data_parsing.py  
$
```

Failing tests

```
def parse_line(line):
    '''Split a line into its fields, convert to the
    appropriate types, and return as a tuple.
    >>> parse_line('5 3 3.7')
    (5, 3, 3.7)
    >>> parse_line('5 3 3')
    (5, 3, 3)
    '''

data = li
return (i
        $ python -m doctest data_parsing.py
*****
File "./data_parsing.py", line 9, in __main__.parse_line
Failed example:
    parse_line('5 3 3')
Expected:
    (5, 3, 3)
Got:
    (5, 3, 3.0)
*****
1 items had failures:
  1 of   2 in __main__.parse_line
***Test Failed*** 1 failures.$
```

Code Pack 08

A. Create and document mymath.py

Coding Bootcamp Code in Python

FILES: I/O AND DATA FORMATS

Reading from files

- Reading from text files, line by line
 - E.g., read file line by line, convert to uppercase, and print

```
1 with open(file_name, 'r') as text_file:  
2     for line in text_file:  
3         print(line.upper())
```

with ...: context manager

- Reading from a binary file, value by value
 - E.g., read doubles (8 bytes) and print

```
1 from struct import unpack  
2 with open(file_name, 'rb') as bin_file:  
3     double_bytes = bin_file.read(8)  
4     while double_bytes:  
5         print(unpack('d', double_bytes)[0])  
6         double_bytes = bin_file.read(8)
```

Not portable!!!:
data type size?
Encoding?
little /big endian?

Libraries & data formats

- Standard library (Python 3.x)
 - Comma separated value files: csv
 - Configuration files: ConfigParser
 - Semi-structured data: json, html1lib, sgmllib, xml
- Non-standard libraries
 - Images: scikit-image
 - HDF5: pytables
 - pandas
 - Bioinformatics: Biopython

Use the "batteries" that are included!

Data formats: CSV

Let Sniffer figure out
CSV dialect (e.g., Excel)

```
0 from csv import Sniffer, DictReader
1 with open(file_name, 'rb') as csv_file:
2     dialect = Sniffer().sniff(csv_file.read(1024))
3     csv_file.seek(0)
4     sum = 0.0
5     csv_reader = DictReader(csv_file, fieldnames=None,
6                             restkey='rest', restval=None,
7                             dialect=dialect)
8     for row in csv_reader:
9         print('{name} --- {weight}'.format(name=row['name'],
10                                            weight=row['weight']))
11        sum += float(row['weight'])
12    print('sum = {}'.format(sum))
```

DictReader uses first
row to deduce field names

Access fields by name,
thanks to DictReader

Drawback: you still need to know field types

Writing to files

- Writing to text files
 - E.g., compute and write to file

Note: '`w`' replaces existing file, use '`x`' to avoid

```
1 with open(file_name, 'w') as text_file:  
2     for i in range(0, 10):  
3         text_file.write('{0}: {1}\n'.format(i, i*i))
```

- Append to text files
 - E.g., add some more squares to same file

```
1 with open(file_name, 'a') as text_file:  
2     for i in range(10, 20):  
3         text_file.write('{0}: {1}\n'.format(i, i*i))
```

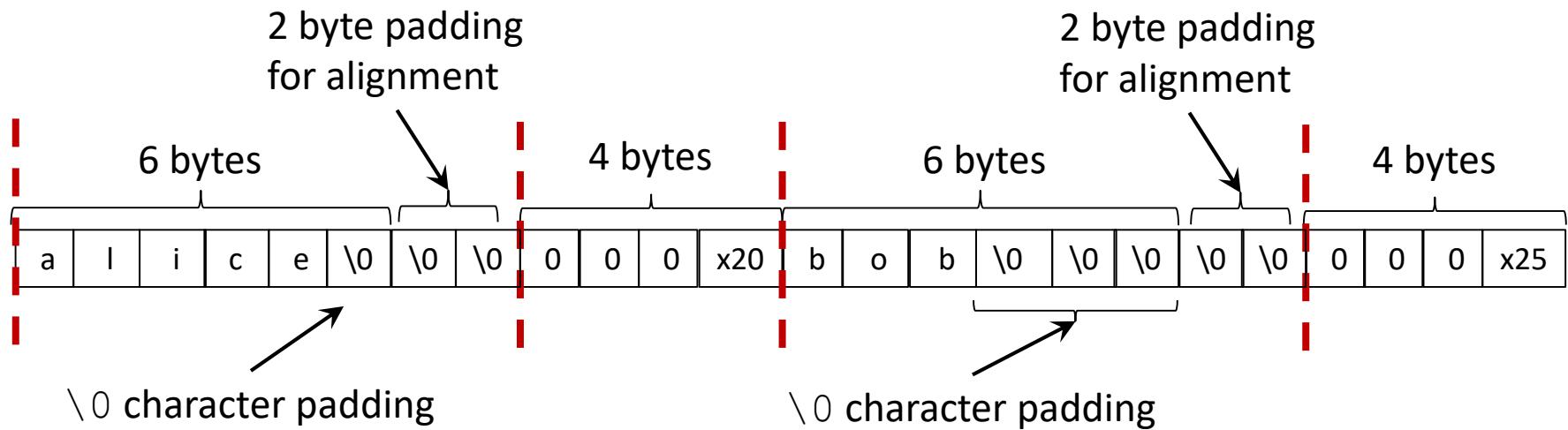
- Writing binary files: don't go there...

... unless you have to

- Writing to binary files

byte representation of
name truncated to 6
characters

```
1 from struct import pack
2 with open(file_name, 'wb') as bin_file:
3     for name, age in people:
4         bin_file.write(pack('6si',
5                             bytes(name, 'ascii'),
6                             age))
```



Data formats: XML output

```
<?xml version="1.0" ?>
<blocks>
  <block name="block_01">
    <item>
      0.1
    </item>
    <item>
      1.1
    </item>
  </block>
  <block name="block_02">
    <item>
      0.2
    </item>
    <item>
      1.2
    </item>
  </block>
</blocks>
```

Data formats: creating XML

```
1 from xml.dom.minidom import Document
2 nr_blocks = 2
3 nr_items = 2
4 doc = Document()
5 blocks = doc.createElement('blocks')
6 doc.appendChild(blocks)
7 for block_nr in range(1, nr_blocks + 1):
8     block = doc.createElement('block')
9     block_name = 'block_{0:02d}'.format(block_nr)
10    block.setAttribute('name', block_name)
11    blocks.appendChild(block)
12    for item_nr in range(0, nr_items):
13        item = doc.createElement('item')
14        text = '{0}.{1}'.format(item_nr, block_nr)
15        text_node = doc.createTextNode(text)
16        item.appendChild(text_node)
17        block.appendChild(item)
18 print(doc.toprettyxml(indent=' '))
```

Code Pack 09

- A. Create a database with Text Files
- B. Working with CSV

Coding Bootcamp Code in Python

WEB SCRAPING: GATHERING DATA FROM THE WEB

Introduction

- Caveat: web scraping code is brittle, typically not robust against
 - page layout changes (unless proper use of CSS)
 - page content changes
 - site redesign

Use site APIs (e.g., REST interface) whenever available!

- Many frameworks available, here **Beautiful Soup**
- However, for tables only, consider **pandas**

Beautiful Soup

- Open web page using `urllib`

```
import urllib
...
page = urllib.request.urlopen(page_url)
```

Note: `urllib2` for Python 2.x

- Cook soup out of opened page

```
from bs4 import BeautifulSoup
...
soup = BeautifulSoup(page, "html5lib")
```

- Eat soup

```
print('looking at {}'.format(soup.title.string))
```

Assumes page has a `title` element

Finding stuff

- First element with tag, e.g., a

```
print('looking at {}'.format(soup.a))
```

- All element with tag, e.g., a

```
for a in soup.find_all('a'):  
    print('a element: {}'.format(a))
```

- Element content, e.g., a

```
for a in soup.find_all('a'):  
    print('link text: {}'.format(a.string))
```

- Element attribute, e.g., href in a

```
for a in soup.find_all('a'):  
    print('link url: {}'.format(a.get('href')))
```

Code Pack 10

A. Try the Web Scraping Code

Coding Bootcamp Code in Python

ERRORS: DEALING WITH EXCEPTIONS

Errors

```
...
def main():
    file_name = sys.argv[1]
    with open(file_name) as in_file:
        for line in in_file:
            print('|{0}|'.format(line.rstrip('\r\n')))
return 0
...
```

exception
thrown

```
$ python quote.py
Traceback (most recent call last):
  File "./quote.0.py", line 13, in <module>
    status = main()
  File "./quote.py", line 6, in main
    file_name = sys.argv[1]
IndexError: list index out of range
```

Either check length of sys.argv, or deal with error!

Playing catch

```
...
def main():
    try:
        file_name = sys.argv[1]
    except IndexError as e:
        sys.stderr.write('### error: no input file\n')
        return 1
    with open(file_name) as in_file:
        for line in in_file:
            print('|{0}|'.format(line.rstrip('\r\n')))
    return 0
...
```

```
$ python quote.py
### error: no input file
```

More trouble

```
$ python quote.py bla
Traceback (most recent call last):
  File "./quote.py", line 17, in <module>
    status = main()
  File "./quote.py", line 11, in main
    with open(file_name) as in_file:
IOError: [Errno 2] No such file or directory: 'bla'
```

exception
thrown

Catching more

```
...
def main():
    try:
        file_name = sys.argv[1]
        in_file = open(file_name)
        with in_file:
            for line in in_file:
                print('|{0}|'.format(line.rstrip('\r\n')))
    except IndexError as e:
        sys.stderr.write('### error: no input file\n')
        return 1
except IOError as e:
    msg = "### I/O error on '{0}': {1}".format(e.filename,
                                                e.strerror)
    sys.stderr.write(msg)
    return 2
return 0
...
```

All handled!

- Now all exceptions are handled

```
$ python quote.py bla  
### I/O error on 'bla': No such file or directory
```

- Note that code size increased from 5 to 16 lines
 - Handling errors takes effort
 - Worthwhile if others are using your software!
- One can create own exceptions, derive class from Exception

Code Pack 11

- A. Python fundamentals:
 - 1. ~~Primitive Datatypes and Operators~~
 - 2. ~~Variables and Collections~~
 - 3. ~~Control Flow and Iterables~~
 - 4. ~~Functions~~
 - 5. ~~Modules~~
 - 6. ~~Classes~~
 - 7. Advanced

Coding Bootcamp Code in Python

THE COLLECTIONS MODULE

ChainMap

- Provides the ability to link multiple mappings together such that they end up being a single unit

```
from collections import ChainMap

car_parts = {'hood': 500, 'engine': 5000, 'front_door': 750}
car_options = {'A/C': 1000, 'Turbo': 2500, 'rollbar': 300}
car_accessories = {'cover': 100, 'hood_ornament': 150,
'seat_cover': 99}
car_pricing = ChainMap(car_accessories, car_options, car_parts)
print (car_pricing['hood'])

#500
```

Counter

- Supports convenient and fast tallies
- Also can run it against most iterables

```
from collections import Counter
print (Counter('superfluous'))
#Counter({'u': 3, 's': 2, 'e': 1, 'l': 1, 'f': 1, 'o': 1, 'r': 1,
'p': 1})

counter = Counter('superfluous')
print (counter['u'])
#3
```

defaultdict

- Subclass of Python's dict that accepts a `default_factory` as its primary argument

```
from collections import defaultdict

sentence = "The red fox jumped over the fence and ran to the zoo
for food"
words = sentence.split(' ')

d = defaultdict(int)
for word in words:
    d[word] += 1

print(d)
```

deque

- Are a generalization of stacks and queues.

```
from collections import deque
import string

d = deque(string.ascii_lowercase)
for letter in d:
    print(letter)
#a
#b
#c
#...
#z
```

namedtuple

- Can use to replace Python's tuple

```
from collections import namedtuple

Parts = namedtuple('Parts', 'id_num desc cost amount')
auto_parts = Parts(id_num='1234', desc='Ford Engine',
                    cost=1200.00, amount=10)

print(auto_parts.id_num)

#1234
```

Code Pack 12

- See the files

1.ChainMap

2.Counter

3.defaultdict

4.deque

5.namedtuple

Coding Bootcamp Code in Python

ITERATORS AND GENERATORS

Iterators

- An iterator is an object that will allow you to iterate over a container.
- The iterator in Python is implemented via two distinct methods:
`__iter__` and `__next__`.
 1. The `__iter__` method is required for your container to provide iteration support. It will return the iterator object itself.
 2. But if you want to create an iterator object, then you will need to define `__next__` as well, which will return the next item in the container.

Generators

- A normal Python function will always return one value, whether it be a list, an integer or some other object.
- But what if you wanted to be able to call a function and have it yield a series of values? That is where generators come in.
- A generator works by “saving” where it last left off (or **yielding**) and giving the calling function a value. So instead of returning the execution to the caller, it just gives temporary control back.
- a generator function requires Python’s **yield** statement.

Code Pack 13

- See the files

1.Iterators

2.Generators

Coding Bootcamp Code in Python

THE ITERTOOLS MODULE

The Infinite Iterators

- The itertools package comes with three iterators that can iterate infinitely
- **count(start=0, step=1)**
- **cycle(iterable)**
- **repeat(object)**

Iterators That Terminate

- Most of the iterators that you create with `itertools` are not infinite
- `accumulate(iterable)`
- `chain(*iterables)`
- `chain.from_iterable(iterable)`
- `compress(data, selectors)`
- `dropwhile(predicate, iterable)`
- `filterfalse(predicate, iterable)`
- `groupby(iterable, key=None)`
- `islice(iterable, start, stop)`
- `starmap(function, iterable)`
- `takewhile(predicate, iterable)`
- `tee(iterable, n=2)`
- `zip_longest(*iterables, fillvalue=None)`

The Combinatoric Generators

- The `itertools` library contains four iterators that can be used for creating combinations and permutations of data.
- **`combinations(iterable, r)`**
- **`combinations_with_replacement(iterable, r)`**
- **`product(*iterables, repeat=1)`**
- **`permutations`**

Code Pack 14

- See the files

1.The_Infinite_Iterators

2.Iterators_That_Terminate

3.The_Combinatoric_Generators

Coding Bootcamp Code in Python

CONTEXT MANAGERS

Context manager in Python

```
# before Python 2.5
f_obj = open(path, 'w')
f_obj.write(some_data)
f_obj.close()
```

```
# after Python 2.5 + with statement
with open(path, 'w') as f_obj:
    f_obj.write(some_data)
```

- **with statement** automatically creates a context manager
- The way this works under the covers is by using some of Python's magic methods: `__enter__` and `__exit__`.

Creating a Context Manager class

```
import sqlite3

class DataConn:
    """
    """

    def __init__(self, db_name):
        """Constructor"""
        self.db_name = db_name

    def __enter__(self):
        """
        Open the database connection
        """
        self.conn = sqlite3.connect(self.db_name)
        return self.conn

    def __exit__(self, exc_type, exc_val, exc_tb):
        """
        Close the connection
        """
        self.conn.close()
        if exc_val:
            raise

if __name__ == '__main__':
    db = 'test.db'
    with DataConn(db) as conn:
        cursor = conn.cursor()
```

Creating a Context Manager using contextlib

```
from contextlib import contextmanager

@contextmanager
def file_open(path):
    try:
        f_obj = open(path, 'w')
        yield f_obj
    except OSError:
        print("We had an error!")
    finally:
        print('Closing file')
        f_obj.close()

if __name__ == '__main__':
    with file_open('test.txt') as fobj:
        fobj.write('Testing context managers')
```

contextlib.closing(thing)

- The difference is that instead of a decorator, we can use the closing class itself in our with statement

```
from contextlib import closing
from urllib.request import urlopen

with closing(urlopen('http://www.google.com')) as webpage:
    for line in webpage:
        # process the line
        pass
```

contextlib.suppress(*exceptions)

- It can suppress any number of exceptions

```
from contextlib import suppress

with suppress(FileNotFoundError):
    with open('fauxfile.txt') as fobj:
        for line in fobj:
            print(line)
```

contextlib.redirect_stdout / redirect_stderr

- The contextlib library has a couple of tools for redirecting stdout and stderr

```
from contextlib import redirect_stdout

path = 'text.txt'
with open(path, 'w') as fobj:
    with redirect_stdout(fobj):
        help(redirect_stdout)
```

ExitStack

- Context manager that will allow to easily programmatically combine other context managers and cleanup functions

```
from contextlib import ExitStack

with ExitStack() as stack:
    file_objects = [stack.enter_context(open(filename))
                    for filename in filenames]
```

Reentrant Context Managers

- If an instance of a context manager try running it twice with Python's with statement.
- The second time it runs, it raises a **RuntimeError**.
- But what if we wanted to be able to run the context manager twice?
- We'd need to use one that is “**reentrant**”.

Code Pack 15

- See the files

- 1.Creating_a_Context_Manager_class
- 2.Creating_a_Context_Manager_using_contextlib
- 3.contextlib.closing(thing)
- 4.contextlib.suppress(exceptions)
- 5.contextlib.redirect_stdout_redirect_stderr
- 6.ExitStack
- 7.Reentrant_Context_Managers

Coding Bootcamp Code in Python

UNIT TESTING

Unit testing

- Key concepts
 - Implementation tested through API
 - Testing should be easy
 - Tests are independent of one another
- Find problems early/fast
- Facilitates change
 - Make small change, run tests
- TDD: Test Driven Development
 - Write tests first, then implement
- Programming framework, e.g., Python's unittest

*"How to test?" is a question that cannot be answered in general.
"When to test?" however, does have a general answer: as early and as often as possible.*

— Bjarne Stroustrup

Test case

- Subclass of `unittest.TestCase`
- Methods `test_<name>` are tests
- `unittest` provides driver for running tests

```
import unittest
from func_lib import fib

class FibTest(unittest.TestCase):
    def test_fib4(self):
        '''test for fib(4)'''
        self.assertEqual(3, fib(4))

if __name__ == '__main__':
    unittest.main()
```

The diagram illustrates the components of a test case in Python code:

- Test case**: Points to the line `class FibTest(unittest.TestCase):`.
- Individual test**: Points to the method definition `def test_fib4(self):`.
- Result to test**: Points to the call `self.assertEqual(3, fib(4))`.
- Expected result**: Points to the value `3` in the `self.assertEqual` call.
- Test driver**: Points to the line `if __name__ == '__main__': unittest.main()`.
- fib_test.py**: Points to the file name at the bottom of the code.

Running tests

- Run Python script

```
$ python ./fib_test.py
F
=====
FAIL: test_fib4 (__main__.FibTest)
test a number computations for small arguments
-----
Traceback (most recent call last):
  File "./fibber.py", line 13, in test_fib4
    self.assertEqual(expected, fib(4))
AssertionError: 3 != 5
-----
Ran 1 test in 0.001s

FAILED (failures=1)
```

Assert methods

- Many methods: provide accurate feedback
 - assertEquals **for** int, str
 - assertAlmostEqual **for** float, complex
 - assertTrue, assertFalse **for** bool
 - assertListEqual, assertSetEqual,
assertDictEqual, assertTupleEqual
 - assertIn
 - assertIsNone
 - assertIsInstance
 - assertRegex

+ negations, e.g.,
assertNotEqual, ...

Checking for expected failure

- Exceptions

```
from func_lib import fib, InvalidArgumentException
...
def test_negative_values(self):
    '''test for call with negative argument'''
    with self.assertRaises(InvalidArgumentException):
        fib(-1)
...
```

- Also useful: assertRaisesRegex
- Warnings: assertWarns

Subtests

- To check for a series of values

```
...
def test_low_values(self):
    '''test a number computations for small arguments'''
    expected = [0, 1, 1, 2, 3, 5, 8, 13]
    for n in range(len(expected)):
        with self.subTest(i=n):
            self.assertEqual(expected[n], fib(n))
...
...
```

Fixtures

- Prepare for test(s), clean up after test(s), e.g.,
 - Open/close a file
 - Open/close a database connection, initialize a cursor
 - Initialize data structures/objects
- Three levels
 - Before/after any test in module is run
 - `setUpModule()`/`tearDownModule()`
 - Before/after any test in test case class is run
 - `setUpClass(cls)`/`tearDownClass(cls)` (mark as `@classmethod`)
 - Before/after each individual test
 - `setUp(self)`/`tearDown(self)`

Module-level

- **setUpModule**: create and fill database

```
import init_db  
...  
def setUpModule():  
    '''create and fill the database'''  
    conn = sqlite3.connect(master_name)  
    init_db.execute_file(conn, 'create_db.sql')  
    init_db.execute_file(conn, 'fill_db.sql')
```

- **tearDownModule**: remove database

```
def tearDownModule():  
    '''remove database file once testing is done'''  
    os.remove(master_name)
```

Test case-level

- `setUpClass`: create copy of database

```
test_name = 'test.db'

@classmethod
def setUpClass(cls):
    '''copy original database'''
    shutil.copyfile(master_name, cls.test_name)
```

Test cases must
be independent!

- `tearDownClass`: remove copy of database

```
@classmethod
def tearDownClass(cls):
    '''remove test database'''
    os.remove(cls.test_name)
```

Test-level

- **setUp: create connection & cursor**

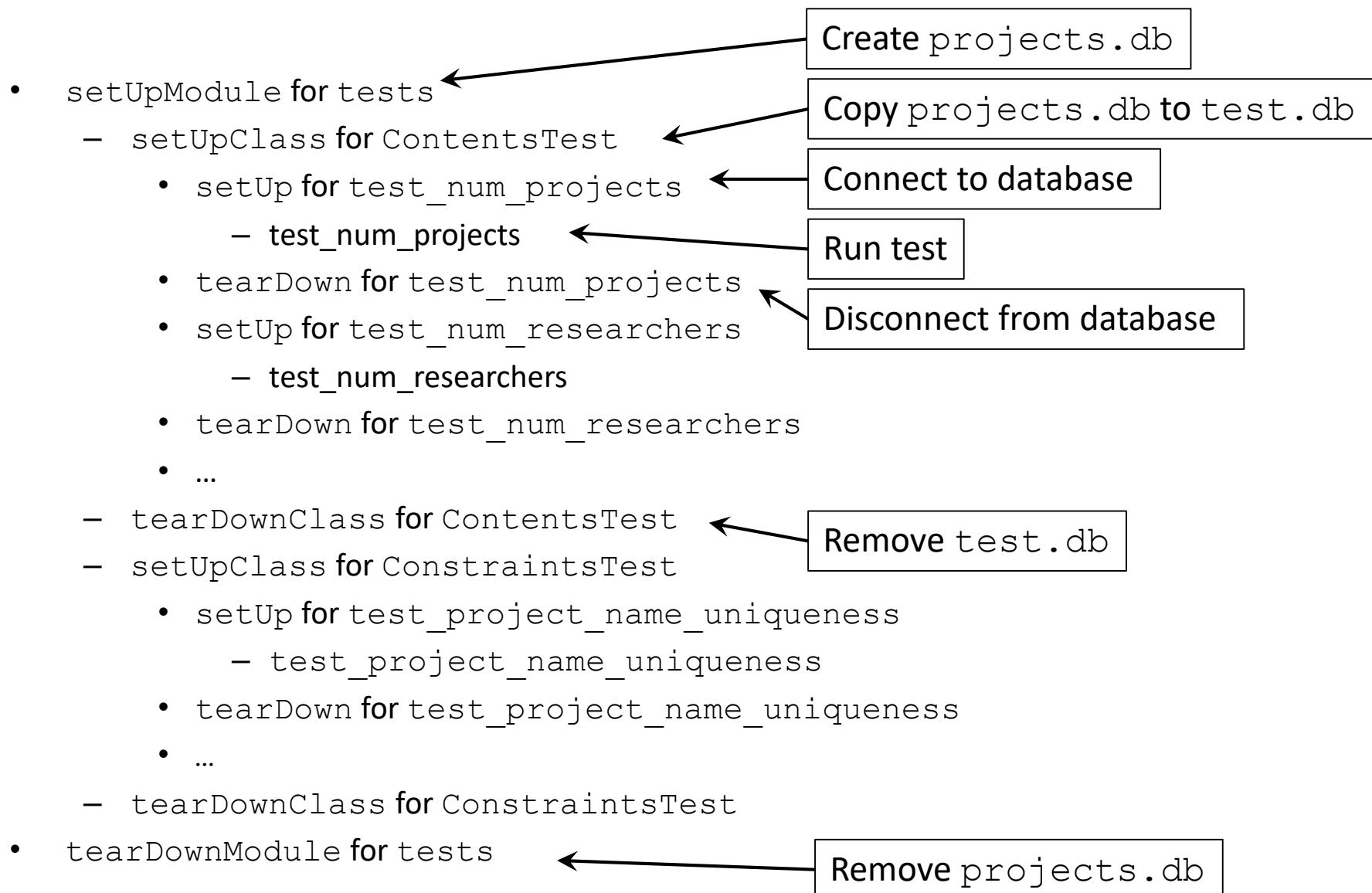
```
def setUp(self):  
    '''open connection, create cursor'''  
    self._conn = sqlite3.connect(self.__class__.test_name)  
    self._conn.row_factory = sqlite3.Row  
    self._cursor = self._conn.cursor()
```

- **tearDown: close connection**

```
def tearDown(self):  
    '''close database connection'''  
    self._conn.close()
```

Tests must be
independent!

Flow for fixtures



Running all tests

- In module

```
...
if __name__ == '__main__':
    unittest.main()
```

fib_test.py

```
$ python ./fib_test.py
```

- In all modules

```
$ python -m unittest discover -p '*_test.py'
```

Test coverage

- Easy to overlook
 - functions/methods
 - code paths
- Use code coverage tool
<https://coverage.readthedocs.io/>
- Steps
 - run code using `coverage run`
 - create detailed report using `coverage annotate`
 - add tests until covered

A program that has not been tested does not work.
— Bjarne Stroustrup

Coverage usage

- Run code

```
$ coverage run ./prog.py  
...
```

- Report

show line numbers missed

```
$ coverage report -m  
coverage report -m  
Name         Stmts    Miss   Cover   Missing  
-----  
functions.py      9       3     67%    2-5  
prog.py         14       2     86%  17-18  
-----  
TOTAL           23       5     78%
```

line numbers
missed

Coverage usage

- Create annotated source code

```
$ coverage annotate -d coverage_report
```

directory for reports

```
...  
>     if options.no_iter:  
>         n = options.max_n  
>         print(f'fac({n}) = {func(n)}')  
!  
else:  
!  
    for n in range(options.max_n + 1):  
!  
        print(f'fac({n}) = {func(n)}')  
...  
...
```

run

not run

- Remove coverage data

```
$ coverage erase
```

Further reading

- B. Kernighan & R. Pike (1999) *The practice of programming*, Addison-Wesley
- M. Fowler (1999) *Refactoring: improving the design of existing code*, Addison-Wesley

Code Pack 16

- See the files:

1.unittest

2.coverage.py

Coding Bootcamp Code in Python

ARGPARSE, CONFIGPARSER

Handling command line arguments

- Many tools start out as short script, evolve into applications used by many
- Model after Unix tools
 - Arguments
 - Flags
 - Options
- Python's argparse benefits
 - Easy to use
 - Self-documenting

Defining command line arguments

- Use argparse library module

```
from argparse import ArgumentParser  
arg_parser = ArgumentParser(description='Gaussian random number generator')
```

- Add positional argument(s)

```
arg_parser.add_argument('nr', metavar='n', type=int, nargs='?', default=1,  
                      help='number of random numbers to generate')
```

- Add flag(s)

```
arg_parser.add_argument('-idx', action='store_true', dest='index',  
                      help='print index for random number')
```

- Add option(s)

```
arg_parser.add_argument('-mu', type=float, default=0.0,  
                      help='mean of distribution')
```

dest='mu' is implicit

- Parse arguments

```
args = arg_parser.parse_args()
```

Using command line arguments

```
for i in range(args.nr):
    if args.index:
        prefix = '{0}\t'.format(i + 1)
    else:
        prefix = ''
    print('{0}{1}'.format(prefix, random.gauss(args.mu, args.sigma)))
```

```
$ ./generate_gaussians -h
usage: generate_gaussians.py [-h] [-mu MU] [-sigma SIGMA] [-idx] [n]
Gaussian random number generator
positional arguments:
  n                  number of random numbers to generate
optional arguments:
  -h, --help          show this help message and exit
  -mu MU             mean of distribution
  -sigma SIGMA       stddev of distribution
  -idx               print index for random number
```

Autogenerated
help message

```
$ ./generate_gaussians -idx 3.0
usage: generate_gaussians.py [-h] [-mu MU] [-sigma SIGMA] [-idx] [n]
generate_gaussians.py: error: argument n: invalid int value: '3.0'
```

ConfigParser configuration files

- Configuration files
 - save typing of options
 - Document runs of applications
- Easy to use from Python: configparser module
- Configuration file (e.g., 'test.conf')

```
[physics]
# this section lists the physical quantities of interest
T = 273.15
N = 1
[meta-info]
# this section provides some meta-information
author = gjb
version = 1.2.17
```

key = value

comments

section physics
section meta-info

Note:
at least one section

Reading & using configurations

- Reading configuration file

```
from configparser import ConfigParser  
cfg = ConfigParser()  
cfg.read('test.conf')
```

- Using configuration values

```
temperature = cfg.getfloat('physics', 'T')  
number_of_runs = cfg.getint('physics', 'N')  
version_str = cfg.get('meta-info', 'version')  
if cfg.has_option('physics', 'g'):  
    acceleration = cfg.getfloat('physics', 'g')  
else:  
    acceleration = 9.81
```

Further reading: argparse

- Argparse tutorial

<https://docs.python.org/3/howto/argparse.html>

Code Pack 17

- See the files:

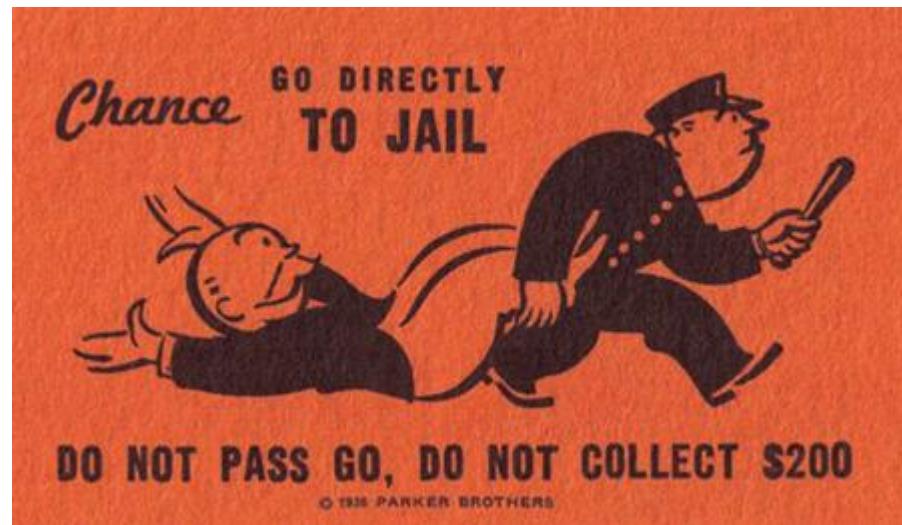
1.argparse

2.configparser

Coding Bootcamp Code in Python

PROFILING

If you don't profile...



Profiling approaches

- Microbenchmarking, i.e., timing functions
 - Easy
 - Can lead to premature optimization
= waste of time
- Profiling with profiling tool
 - Slightly more complicated
 - Identifies true bottlenecks

Both are useful, when used appropriately

Timing functions

- ipython: use magic %time or %timeit

multiple runs

```
In [1]: from primes import primes
In [2]: %timeit result = primes(1000)
10 loops, best of 3: 172 ms per loop
```

↑
timing result

- Command line: use timeit module

module
to use

statements to execute, string per line

```
$ python -m timeit 'from primes import primes' 'primes(1000)'
10 loops, best of 3: 174 msec per loop
```

Don't forget indentation!

Profiler

- Use the cProfile module

module to use sort order



```
$ python -m cProfile -s time primes.py 1000

2914 function calls (2878 primitive calls) in 0.261 seconds

Ordered by: internal time

      ncalls  tottime  percall  cumtime  percall filename:lineno(function)
            1    0.250    0.250    0.251    0.251 primes.py:6(primes)
            1    0.002    0.002    0.002    0.002 {built-in method loads}
        1194    0.001    0.000    0.001    0.000 {'append' of 'list'}
           43    0.001    0.000    0.001    0.000 {'join' of 'str'}
```

Visual profiles: snakeviz

- Use the cProfile module

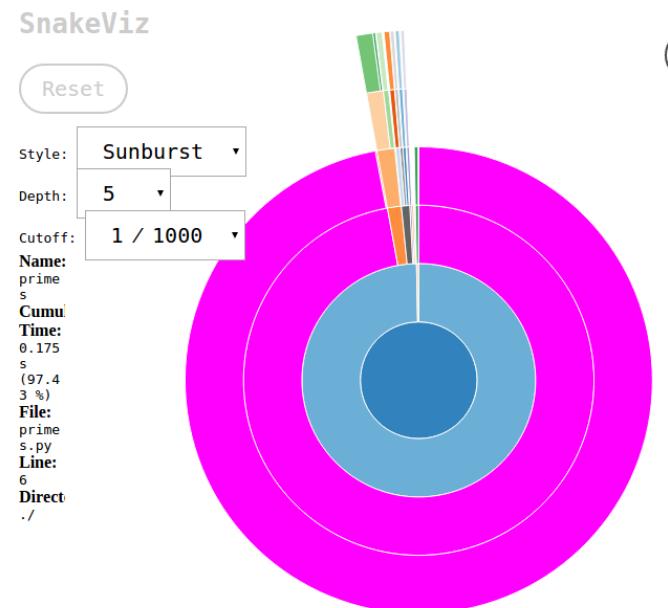
module to use

sort order

output file

```
$ python -m cProfile -s time -o primes.prof \
primes.py 1000
```

```
$ snakeviz primes.prof
```



line_profiler

line by line

show profile on screen

decorate function to profile

```
$ kernprof -l -v primes.py 1000
imer unit: 1e-06 s

Total time: 1.01724 s
File: /home/gjb/Documents/Projects/training-material/Python/Profiling/primes.py
Function: primes at line 4

Line #      Hits          Time  Per Hit   % Time  Line Contents
=====
4                  1           2       2.0     0.0    @profile
5                                max_size = 1000000
6                  1           72903   72903.0    7.2    def primes(kmax):
7                  1           4           4.0     0.0        p = array('i', [0]*max_size)
8                  1           2           2.0     0.0        result = []
9                  1           1           1.0     0.0        if kmax > max_size:
10                 1           0           0.0     0.0            kmax = max_size
11                 1           0           0.0     0.0        k = 0
12                 1           0           0.0     0.0        n = a2
```

Code Pack 18

- See the files:
 - 1.Benchmarking
 - 2.Profiling_Your_Code_with_cProfile

Coding Bootcamp Code in Python

LOGGING

Logging: motivation

- Useful to verify what an application does
 - in normal runs
 - in runs with problems
- Helps with debugging
 - alternative to print statements
- Various levels can be turned on or off
 - see only relevant output

Good practice

Initialize & configure logging

```
import logging
...
logging.basicConfig(level=level, filename=name, filemode=mode,
                    format=format_str)
...
```

- **level**: minimal level written to log
- **filemode**
 - 'w': overwrite if log exists
 - 'a': append if log exists
- **format**, e.g.,
`'{asctime} : {levelname}: {message}'`

Log levels

- CRITICAL: non-recoverable errors
- ERROR: error, but application can continue
- WARNING: potential problems
- INFO: feedback, verbose mode
- DEBUG: useful for developer
- User defined

Selecting log level

- CRITICAL
- ERROR
- WARNING
- INFO
- DEBUG

```
level = logging.ERROR
```

Log messages

- Log to DEBUG level

```
logging.debug('function xyz called with "{0}"'.format(x))
```

- Log to INFO level

ignored at level INFO or above

```
logging.info('application started')
```

- Log to CRITICAL level

ignored at level WARNING or above

```
logging.critical('input file not found')
```

Logging destinations

- File
- Rotating files
- syslog
- ...

Further reading: logging

- Logging how-to

<https://docs.python.org/3/howto/logging.html>

- Logging Cookbook

<https://docs.python.org/3/howto/logging-cookbook.html>

Code Pack 19

- See the files:

Logging

Coding Bootcamp Code in Python

DEBUGGING PYTHON

Errors & warnings: pylint, flake8

- Static code analysis, errors, warnings, code quality suggestions

```
$ pylint add.py
*****
Module add
C: 1, 0: Missing module docstring (missing-docstring)
E: 4,10: Undefined variable 'x' (undefined-variable)
```

Report

=====

3 statements analysed.

...

```
#!/usr/bin/env python
```

```
if __name__ == '__main__':
    print(x + 3)
```

- flake8 can be invoked from vim, as git hook

Use classic debugger

- Bugs are ubiquitous...
- Debugging by print?
 - easy to do
 - takes a long time for complex situations
 - unstructured process
 - pollutes code
- Use debugger (pdb for Python): it can
 - step through code, statement by statement
 - inspect variable values
 - ...

Okay, what's this?!?

```
def main():
    n = int(sys.argv[1])
    matrix = [[0] * n] * n
    for i in range(n):
        for j in range(n):
            matrix[i][j] = i*n + j + 1
    print('\n'.join([' '.join(['{:2d}'.format(e) for e in row])
                    for row in matrix]))
    return 0
```

In your dreams!

```
$ python buggy.py 5
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
```

```
$ python buggy.py 5
21 22 23 24 25
21 22 23 24 25
21 22 23 24 25
21 22 23 24 25
21 22 23 24 25
```

Starting & viewing source

- Starting the debugger

```
$ python -m pdb buggy.py 5
> ./buggy.py(3)<module>()
-> import sys
```

Statement about to be executed

(Pdb)



debugger prompt

- Listing source code: l [<line-nr>] (list)

```
(Pdb) l
1  #!/usr/bin/env python
2
3 ->      import sys
4
5 def main():
6     n = int(sys.argv[1])
```

Stepping

- Execute statement: n (next)

```
(Pdb) n
> ./buggy.py(5)<module>()
-> def main():
(Pdb)
> ./buggy.py(14)<module>()
-> if __name__ == '__main__':
(Pdb)
> ./buggy.py(15)<module>()
-> status = main()
```

- Step into function: s (step)

```
(Pdb) s
> ./buggy.py(5)<module>()
-> def main():
```

r (return): run until current function returns

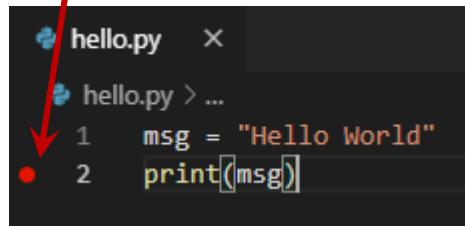
Printing values: variables

- Print variable values: p <var> (print)

```
(Pdb) n
> ./buggy.py(6)main()
-> n = int(sys.argv[1])
(Pdb)
> ./buggy.py(7)main()
-> matrix = [[0] * n] * n
(Pdb) p n
5
(Pdb) n
-> for i in range(n):
(Pdb) p matrix
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

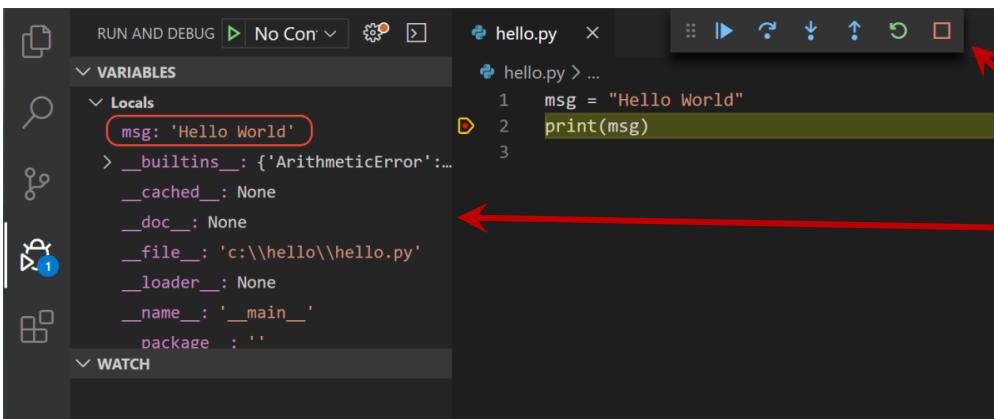
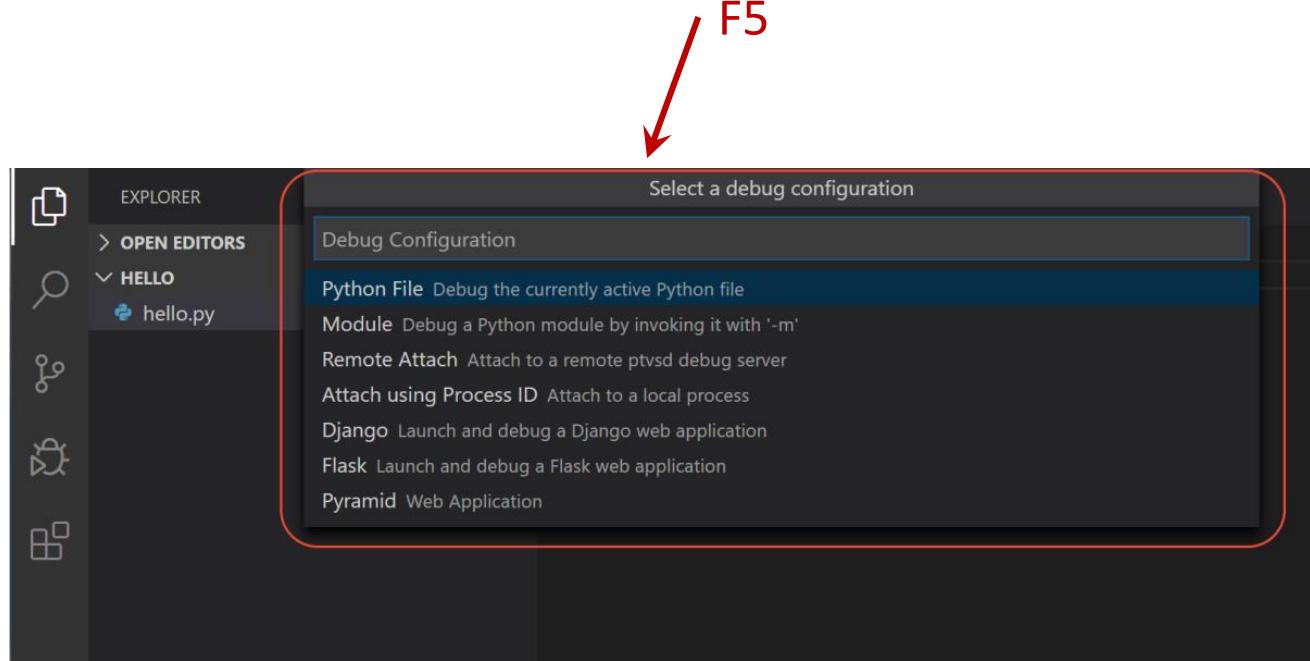
Mouse Click or F9

VSCode Debugger



hello.py

```
1 msg = "Hello World"
2 print(msg)
```

A screenshot of the VSCode editor showing a Python file named 'hello.py'. A red arrow points from the top-left towards the editor area, indicating where to click to set a breakpoint. A red dot is visible on the left margin of the first line of code.

RUN AND DEBUG No Con ▾

VARIABLES

Locals

- msg: 'Hello World'
- _builtins_: {'ArithmetError':...}
- _cached_: None
- _doc_: None
- _file_: 'c:\\hello\\hello.py'
- _loader_: None
- _name_: '__main__'
- package : ''

WATCH

A screenshot of the VSCode 'Run and Debug' view. It shows a toolbar with execution controls like play, step, and stop. Below the toolbar is a code editor with the same 'hello.py' code as the previous screenshot. To the left is a 'VARIABLES' sidebar showing the current state of the 'Locals' scope, with 'msg' highlighted. A red arrow points from the bottom-left towards the 'VARIABLES' sidebar, and another red arrow points from the top-right towards the execution controls.

Control your execution
and the variables

Code Pack 20

A. Debug your code

Coding Bootcamp Code in Python

FILE SYSTEM OPERATIONS: HANDLING FILES AND DIRECTORIES

Working with files in directories

- Directory contains files `data_001.txt`, `data_002.txt`, ...

data_001.txt		
case	dim	temp
1	1	-0.5
2	1	0.0
3	1	0.5
4	2	-0.5
5	2	0.0
6	2	0.5

data_002.txt		
case	dim	temp
7	3	-0.5
8	3	0.0
9	3	0.5
10	4	-0.5
11	4	0.0
12	4	0.5

...



data_all.txt		
case	dim	temp
1	1	-0.5
2	1	0.0
3	1	0.5
4	2	-0.5
5	2	0.0
6	2	0.5
7	3	-0.5
8	3	0.0
9	3	0.5
10	4	-0.5
11	4	0.0
12	4	0.5

Using glob

```
from argparse import ArgumentParser, FileType
from pathlib import Path

...
def main():
    arg_parser = ArgumentParser(description='...')
    arg_parser.add_argument('-o', dest='output_file',
                           type=FileType('w'), help='...')
    arg_parser.add_argument('-p', dest='pattern', help='...')
    options = arg_parser.parse_args()
    is_header_printed = False
    path = Path('.')
    for file_name in path.glob(options.pattern):
        with open(file_name, 'r') as input_file:
            header = input_file.readline()
            if not is_header_printed:
                options.output_file.write(header)
                is_header_printed = True
            for line in input_file:
                if line.strip():
                    options.output_file.write(line)
    return 0
```

Same as in
Bash shell

```
$ python concat_data.py -o data.txt -p 'data_*.txt'
```

Path operations

- Many operations in `pathlib` package

- Current working directory: `Path.cwd()`
 - Create path:

```
path = Path.cwd() / 'data' / 'output.txt'  
path == '/home/gjb/Tests/data/output.txt'
```

Will do the right thing for each OS

- Dissecting paths:

- `filename = path.name`
`name == 'test.txt'`
 - `dirname = path.parent`
`dirname == '/home/gjb/data'`
 - `parts = path.parts`
`parts == ('/', 'home', 'gjb', 'data', 'output.txt')`
 - `ext = path.suffix`
`ext == '.txt'`
 - `dirname = Path('/home/gjb/Tests').name`
`dirname == 'Tests'`
 - `ext = Path('/home/gjb/Tests/').suffix`
`ext == ''`

File system tests

- File tests:
 - `path.exists()`: True if path exists
 - `path.is_file()`: True if path is file
 - `path.is_dir()`: True if path is directory
 - `path.is_symlink()`: True if path is link
 - `pathlib.os.access(path, pathlib.os.R_OK)`:
True if path can be read
 - `pathlib.os.R_OK`: read permission
 - `pathlib.os.W_OK`: write permission
 - `pathlib.os.X_OK`: execute permission

However: ask forgiveness, not permission!

Copying, moving, deleting

- Functions in `os` and `shutil` modules
 - copy file: `shutil.copy(source, dest)`
 - copy file, preserving ownership, timestamps:
`shutil.copy2(source, dest)`
 - move file: `path.replace(dest)`
 - delete file: `path.unlink()`
 - remove non-empty directory: `path.rmdir()`
 - remove directory: `shutil.rmtree(directory)`
 - create directory: `path.mkdir()`

Temporary files

- Standard library `tempfile` package
 - Creating file with guaranteed unique name:
`tempfile.NamedTemporaryFile(...)`

```
import tempfile
...
tmp_file = tempfile.NamedTemporaryFile(mode='w', dir='.',
                                         suffix='.txt', delete=False)
print("created temp file '{0}'".format(tmp_file.name))
with tmp_file.file as tmp:
    ...
    tmp.write(...)
    ...
```

File names such as `tmpD45x.txt`

Walking the tree

- Walking a directory tree: `os.walk(...)`, e.g., print name of Python files in (sub)directories

```
import os
...
for directory, _, file_names in os.walk(dir_name):
    for file_name in file_names:
        _, ext = os.path.splitext(file_name)
        if ext == target_ext:
            print(os.path.join(directory, file_name))
...
...
```

- For each directory, tuple:
 - directory name
 - list of subdirectories
 - list of files in directory

For simple cases, use
`path.rglob(...)`

Code Pack 21

- See the Files

os_module.py

Coding Bootcamp Code in Python

MORE ICING ON APPLICATION: THREADING, MULTIPROCESS, SECURITY

threading

- The threading module makes working with threads much easier and allows the program to run multiple operations at once.
 - Can share memory

```
import threading

def doubler(number):
    """
    A function that can be used by a thread
    """
    print(threading.currentThread().getName() + '\n')
    print(number * 2)
    print()

if __name__ == '__main__':
    for i in range(5):
        my_thread = threading.Thread(target=doubler, args=(i,))
        my_thread.start()
```

multiprocessing

- The Process class is very similar to the threading module's Thread class, but by process.

```
import os
from multiprocessing import Process

def doubler(number):
    result = number * 2
    proc = os.getpid()
    print('{0} doubled to {1} by process id: {2}'.format(
        number, result, proc))

if __name__ == '__main__':
    numbers = [5, 10, 15, 20, 25]
    procs = []

    for index, number in enumerate(numbers):
        proc = Process(target=doubler, args=(number,))
        procs.append(proc)
        proc.start()

    for proc in procs:
        proc.join()
```

The cryptography Package

```
#pip install cryptography

from cryptography.fernet import Fernet
cipher_key = Fernet.generate_key()
print (cipher_key)
#b'APM1JDVgT8WDGOWBgQv6EIhvxl4vDYvUnVdg-Vjdt0o='

cipher = Fernet(cipher_key)
text = b'My super secret message'
encrypted_text = cipher.encrypt(text)
print (encrypted_text)
#(b'gAAAAABXOnV86aeUGADA6mTe9xEL92y_m0_TlC9vcqaF6NzHqRKkjEqh4d21P
InEP3C9HuiUkS9f'
# b'6bdHsSlRiCNWbSkPuRd_62zfEv3eaZjJvLAm3omnya8=' )

decrypted_text = cipher.decrypt(encrypted_text)
print (decrypted_text)
#b'My super secret message'
```

Code Pack 22

- See the files

1.threading

2.multiprocessing

3.The_cryptography_Package

Coding Bootcamp Code in Python

RELATIONAL DATABASES: PYTHON DB API & SQLALCHEMY ORM

Accessing relational databases

- Relational databases:
 - great to store structured data, table-oriented
 - can be accessed easily via command line, programming language, GUI
 - can be queried using **SQL**
 - examples: MySQL, PostgreSQL, Oracle, DB2, SQLite3,...
- Using DB from Python via standard interface
 - Support for sqlite3 built-in, ok for simple applications
- For non-trivial stuff, use SQLAlchemy
 - Object-relational mapping (ORM)
 - Connectors to many RDBMS

SQL

- Create table to store data

```
CREATE TABLE IF NOT EXISTS weather (
    city_name      TEXT      NOT NULL,
    date          TEXT      NOT NULL,
    temperature   REAL      NOT NULL);
```

- Store data

```
INSERT INTO weather (city_name, date, temperature)
VALUES ('London', '2012-03-14', 13.2);
```

- Query data

```
SELECT city_name, AVG(temperature) FROM weather
WHERE date BETWEEN '2012-01-01' AND '2012-01-31'
GROUP BY city_name;
```

- Modify data

```
UPDATE weather SET city_name = 'St. Petersburg'
WHERE city_name = 'Leningrad';
```

Databases - The use of Basic SQL

Syntax

- Packages to use
- ~~adodbapi~~
 - Not maintained
- pyodbc
 - C++
- pypyodbc
 - pure python
- MySQLdb
- psycopg2

```
import psycopg2 #postgresql

conn =
psycopg2.connect(dbname='my_database', user='username')
cursor = conn.cursor()

# execute a query
cursor.execute("SELECT * FROM
table_name")
row = cursor.fetchone()

# close your cursor and connection
cursor.close()
conn.close()
```

Python DB access: inserting data

- Connect to a database & create cursor

```
import sqlite3
conn = sqlite3.connect('weather-db')
cursor = conn.cursor()
```

- Insert data tuples

```
for data in generate_data(nr_cities, start, end):
    cursor.execute('''INSERT INTO weather
                    (city_name, date, temperature)
                    VALUES (?, ?, ?) ''',
                   data)
conn.commit()
cursor.close()
```

tuple

Python DB access: querying

- Compute average temperature for period per city

```
conn = sqlite3.connect('weather-db')
conn.row_factory = sqlite3.Row
cursor = conn.cursor()
cursor.execute(
    '''SELECT city_name, AVG(temperature) AS 'temperature'
       FROM weather WHERE date BETWEEN ? AND ?
      GROUP BY city_name''',
    (start, end))
for row in cursor:
    print('{city}\t{tmp}'.format(city=row['city_name'],
                                 tmp=row['temperature']))
cursor.close()
```

SQLAlchemy: ORM

- Define classes/tables

```
from sqlalchemy import (Column, ForeignKey, UniqueConstraint,  
                        Integer, String, DateTime, Float)  
from sqlalchemy.ext.declarative import declarative_base  
from sqlalchemy.orm import relationship
```

Base = declarative_base()

```
class City(Base):  
    __tablename__ = 'cities'  
    city_id = Column(Integer, primary_key=True)  
    name = Column(String(100), nullable=False, unique=True)
```

object attributes

class ≡ table

class attribute ≡ column definition

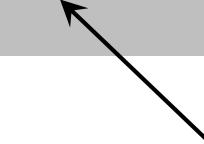
column properties

SQLAlchemy: relationships

- Define relationship

```
class Measurement(Base):  
    __tablename__ = 'measurements'  
    __table_args__ = (  
        UniqueConstraint('time', 'city_id'),  
    )  
    measurement_id = Column(Integer, primary_key=True)  
    time = Column(DateTime, nullable=False)  
    temperature = Column(Float, nullable=False)  
    city_id = Column(Integer, ForeignKey('cities.city_id'))  
    city = relationship(City)
```

table constraint



column properties

relationship for ORM queries

SQLAlchemy: create tables

- To interact, create engine

```
from sqlalchemy import create_engine  
...  
engine = create_engine('sqlite:///{}'.format(db_name))
```

- Creating tables ≡ setting metadata

```
Base.metadata.create_all(engine)
```

That's it!

SQLAlchemy: inserts

- Create engine, session

```
...  
from sqlalchemy.orm import sessionmaker  
...  
engine = create_engine('sqlite:///{}'.format(db_name))  
Base.metadata.bind = engine  
DBSession = sessionmaker(bind=engine)  
db_session = DBSession()  
...
```

- Create and add objects

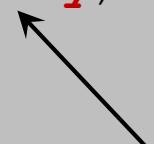
```
...  
for city_name in ['New York', 'Leningrad', 'Paris']:  
    city = City(name=city_name)  
    db_session.add(city)  
db_session.commit()
```

SQLAlchemy: inserting relationships

- Use objects to express relationships

```
...
for city in city_list:
    temperature = measuree_temperature(city)
    date = determine_date()
    measurement = Measurement(time=date,
                                temperature=temperature,
                                city=city)
    db_session.add(measurement)
db_session.commit()
...
```

use actual object



SQLAlchemy: queries

- Queries as method calls

```
...  
city_list = db_session.query(City).all()
```

class ≡ table

- Natural join query

```
...  
measurements = db_session.query(Measurement) \  
    .join('city') \  
    .filter(City.name == city_name,  
            s_date <= Measurement.time,  
            Measurement.time <= e_date) \  
    .all()
```

join on relationship

Note: class attributes!!!

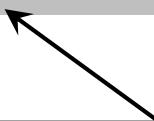
SELECT * FROM ... WHERE ...

SQLAlchemy: updates

- Modify object attribute(s) \equiv update

```
...  
leningrad = db_session.query(City) \  
    .filter(City.name == 'Leningrad') \  
    .one()  
leningrad.name = 'Saint Petersburg'  
db_session.commit()
```

don't forget commit!



Pitfalls

- ORM "hides" database interaction
 - Easy to be inefficient
 - Object creation takes time
 - Can consume a lot of memory
 - Still necessary to understand
 - Relational model
 - How RDBMS works

Further reading: relational databases

- Introduction to relational database design

http://www.ntu.edu.sg/home/ehchua/programming/sql/relational_database_design.html

Code Pack 23

- See the files:

The use of Basic SQL Syntax

Object Relational Mappers

Coding Bootcamp Code in Python

OTHER PYTHON BUILT-INS

any

- Will return True if any element in said iterable is True

```
print (all([0,0,1,0]))  
#False
```

```
print (all([1,1,1,1]))  
#True
```

```
print (any([0,0,0,1]))  
#True
```

```
print (any([0,0,0,0]))  
#False
```

- **all** built-in as it has similar functionality except that it will only return True if every single item in the iterable is True

enumerate

- This returns the position of each item in the iterable as well as the value

```
my_string = 'abcdefg'  
for pos, letter in enumerate(my_string):  
    print (pos, letter)  
  
#0 a  
#1 b  
#2 c  
#3 d  
#4 e  
#5 f  
#6 g
```

eval

- Accepts strings and basically runs them

```
var = 10
source = 'var * 2'
print (eval(source))
#20
```

Can created a major
security breach

filter

- It will take a function and an iterable and return an iterator for those elements within the iterable for which the passed in function returns True

```
def less_than_ten(x):
    return x < 10

my_list = [1, 2, 3, 10, 11, 12]
for item in filter(less_than_ten, my_list):
    print(item)

#1
#2
#3
```

map

- The **map** built-in also takes a function and an iterable and return an iterator that applies the function to each item in the iterable

```
def doubler(x):
    return x * 2

my_list = [1, 2, 3, 4, 5]
for item in map(doubler, my_list):
    print(item)

#2
#4
#6
#8
#10
```

zip

- Takes a series of iterables and aggregates the elements from each of them

```
keys = ['x', 'y', 'z']
values = [5, 6, 7]
print (zip(keys, values))
#<zip object at 0x7faaad4dd848>
```

```
print (list(zip(keys, values)))
#[('x', 5), ('y', 6), ('z', 7)]
```

Code Pack 24

- See the files:

1.any

2.enumerate

3.eval

4.filter

5.map

6.zip

Coding Bootcamp Code in Python

PYTHON FOR SCIENTIFIC COMPUTING: NUMPY

Out of the box

- Python is interpreted
 - Python is slow
 - Python is really slow
- Okay for one-offs, prototypes, short runtimes
- ***Not okay*** for computationally intensive tasks!

Don't use vanilla Python for computations!!!

Python performance

500 × 500 matrices

Python: 0.09 s

C: 0.014 s

Fortran: 0.012 s

Python: 32 s

C: 0.49 s

Fortran: 0.11 s

```
def init_matrix(n):
    m = []
    for i in range(n):
        m.append([])
        for j in range(n):
            m[i].append(random.random())
    return m

def matmul(a, b, c):
    n = len(a)
    for i in range(n):
        for j in range(n):
            c[i][j] = 0.0
            for k in range(n):
                c[i][j] += a[i][k]*b[k][j]
```

Represent matrix as list of lists

Naive!

$$C = A \cdot B$$

$$C_{ij} = A_{i1}B_{1j} + \dots + A_{iN}B_{Nj}$$

Libraries for numeric computation

- numpy
 - Fast arrays
 - Matrix operations (BLAS-like)
 - Linear algebra
 - Fast Fourier Transform
 - Mathematical functions defined on arrays
 - Pseudo-random number generation to initialize arrays
 - Simple statistics
- ...

Python using numpy

500 × 500 matrices

numpy: 0.011 s

numpy: 0.077 s

```
import numpy as np

def init_matrix(n):
    return np.random.uniform(0.0, 1.0, (n, n))

def matmul(a, b):
    return a @ b
```

Language/library	Execution time matrix multiplication (s)
Python	32
C	0.49
Fortran	0.11
Python/numpy	0.077
Fortran/BLAS	0.060



415 ×

Creating array I

convention

```
import numpy as np
```

```
v1 = np.zeros(3)  
v2 = np.ones(3)  
v3 = np.empty(3)
```

```
a = np.zeros((2, 3))  
b = np.ones((2, 3))  
c = np.eye(2)  
d = np.empty((2, 3))
```

create 3-element array, all 0.0

create 3-element array, all 1.0

create 3-element "empty" array
initialize elements later

create 2×3 array, all elements 0.0

create 2×3 array, all elements 1.0

create 2×2 identity array

create 2×3 "empty" array
initialize elements later

Default type: `np.float` = double precision

Creating arrays II

```
e = np.random.uniform(0.0, 1.0, (2, 3))
```

create 2×3 array, elements x
randomly drawn from uniform
distribution such that $x \in [0.0, 1.0[$

```
f = np.array([[3.1, 4.2, -1.1], [-0.3, 1.3, 13.1]])
```

create 2×3 array from a Python
list of lists

```
f = np.genfromtxt('matrix.txt')
```

create 2×3 array
from text file

1.2	2.3	3.4
4.5	5.6	6.7

Creating arrays III

```
e = np.arange(-1.0, 1.0, 0.25)
```

create 8-element array, first element -1.0,
last element less than 1.0, step 0.25

```
[ -1. -0.75 -0.5 -0.25 0. 0.25 0.5 0.75 ]
```

```
f = np.linspace(-1.0, 1.0, 9)
```

create 9-element array, first element -1.0,
last element 1.0, determine step

```
[ -1. -0.75 -0.5 -0.25 0. 0.25 0.5 0.75 1. ]
```

Numpy data types

- **Integers**

`np.int8, np.int16, np.int32, np.int64`

default for `np.int`: `np.int32` on 32-bit, `np.int64` on 64-bit architecture (`np.uint< n >` for unsigned integers)

- **Floating point numbers**

`np.float16, np.float32, np.float64,`
`np.float96`

default for `np.float`: `np.float64`, i.e., double precision

- **Complex numbers**

`np.complex64, np.complex128, np.complex192`
default for `np.complex`: `np.complex128`, i.e., double precision

- **Boolean values:** `np.bool`

- **Characters/** `v = np.zeros(3, dtype=np.int8)`

Accessing array elements

```
a = np.zeros((2, 3))
```

- Array dimensions, strides

```
a.shape == (2, 3)
```

nr. bytes to
next row

```
a.strides == (24, 8)
```

- Assigning to a specific element

```
a[1, 0] = 5.0
```

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ \boxed{5.0} & 0.0 & 0.0 \end{pmatrix}$$

nr. bytes to
next column

- Using an element's value

```
q = a[1, 0] + a[1, 2]
```

Note:

- Implicit conversion of tuple for indexing
- 0-based indexing

Accessing subarrays: slicing

```
a = np.arange(1, 21).reshape(4, 5)
```

- Second column

```
a[:, 1]
```

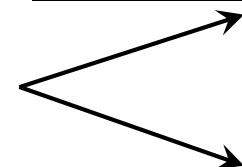
```
[[ 1   2   3   4   5 ]  
 [ 6   7   8   9   10 ]  
 [ 11  12  13  14  15 ]  
 [ 16  17  18  19  20 ]]
```

- Third row

result: 1D array

```
a[2, :]
```

```
[ 2   7   12  17 ]
```



```
[ 11  12  13  14  15 ]
```

- 2D subarray

```
a[1:3, 1:3] = np.eye(2)
```

```
[[ 1   2   3   4   5 ]  
 [ 6   1   0   9   10 ]  
 [ 11  0   1   14  15 ]  
 [ 16  17  18  19  20 ]]
```

cfr. list slicing, but...
array slicing does **not** copy!

Fancy indexing

```
a = np.arange(1, 10, dtype=np.int).reshape(3, 3)
```

```
[[ 1  2  3 ]  
 [ 4  5  6 ]  
 [ 7  8  9 ]
```

- Conditional indexing

```
a[a % 2 == 1] = 0
```

3 × 3 Boolean array

```
[[ 0  2  0 ]  
 [ 4  0  6 ]  
 [ 0  8  0 ]
```

- Conditional assignment

```
np.where(a > 0, 1, -1)
```

3 × 3 Boolean array

```
[[ -1  1  -1 ]  
 [  1  -1  1 ]  
 [ -1  1  -1 ]
```

Operations on arrays

- Scalar-array operations: `+, -, *, /, //, **`

```
a = np.array([[1.0, 3.0], [4.0, 5.0]])
print(3.0 + a)
```

```
[[ 4.   6. ]
 [ 7.   8. ]]
```

- Element-wise operations: `+, -, *, /, //, **`

```
a = np.array([[1.0, 3.0], [4.0, 5.0]])
b = np.array([[2.0, 3.0], [1.0, 0.5]])
print(a*b)
```

```
[[ 2.   9. ]
 [ 4.   2.5 ]]
```

- Matrix product

```
print(np.dot(a, b))
```

```
[[ 5.   4.5 ]
 [ 13.  14.5 ]]
```

– Python 3.5 style

```
print(a @ b)
```

Functions operating on arrays

```
a = np.empty((500, 500))  
b = np.random.uniform(0.0, 1.0, (500, 500))
```

- Avoid

```
for i in range(500):  
    for j in range(500):  
        a[i, j] = math.sqrt(b[i, j])
```

- Use

```
a = np.sqrt(b)
```

140 µs

6.5 µs

21 ×

- Other functions: np.sin, ..., np.sinh, ..., np.exp, np.trace, np.transpose, ...

Some linear algebra

```
a = np.array([[1.0, 3.0], [4.0, 5.0]])
```

- numpy has some linear algebra operations
 - matrix power

```
np.linalg.matrix_power(a, 3)
```

```
[[ 85. 129.]  
 [ 172. 257.]]
```

- matrix inverse

```
np.linalg.inv(a)
```

```
[[-0.71428571 0.42857143]  
 [ 0.57142857 -0.14285714]]
```

- determinant

```
np.linalg.det(a)
```

```
-7.0
```

- eigen values

```
np.linalg.eigvals(a)
```

```
[-1. 7.]
```

References versus copies

- Reshape: different view on same data

```
a = np.array([[1.0, 3.0],  
             [4.0, 5.0]])
```

```
[[ 1.  3.]  
 [ 4.  5.]]
```

```
b = a.reshape((a.size, ))
```

```
[ 1.  3.  4.  5.]
```

```
a[0, 0] = 13.0
```

a

b

```
[[ 13.  3.]  
 [ 4.  5.]]
```

```
[ 13.  3.  4.  5.]
```

- Some operations return copies,
check documentation carefully

numpy data I/O revisited

- Reading text file with 10^9 64-bit floats

- `np.fromtxt(...)`: 57 minutes,

44 GB RAM

- `np.fromfile(..., sep='\\n')`: 4.6 minutes,

8 GB RAM

5 ×

12 ×

All functions are equal, but...
some are more equal than others

- Reading binary file with 10^9 64-bit floats

- `np.fromfile(...)`:

8 seconds,

8 GB RAM

35 ×

Not all data formats are equal: HDF5 to the rescue

Matrices

- Matlab-like initialization

```
a = np.matrix('1.0  3.0; 4.0  5.0')
```

```
[[ 1.   3. ]
 [ 4.   5. ]]
```

- Overloaded * and ** operators

```
a = np.matrix([[1.0, 3.0], [4.0, 5.0]])
b = np.matrix([[2.0, 3.0], [1.0, 0.5]])
print(a*b)
print(a**3)
```

```
[[ 5.    4.5 ]
 [ 13.   14.5 ]]
```

```
[[ 85.   129. ]
 [ 172.  257. ]]
```

- Result is always matrix (2D)

```
a = np.matrix('1.0, 3.0')
b = np.matrix('2.0; 4.0')
print(a*b)
```

```
[[ 14. ]]
```

References

- numpy for MATLAB users
<http://mathesaurus.sourceforge.net/matlab-numpy.html>

Code Pack 25

A. Numpy: Make the code

Coding Bootcamp Code in Python

PYTHON FOR SCIENTIFIC COMPUTING: SCIPY

Libraries for numeric computation

- ...
- **scipy**
 - Dense/sparse linear algebra
 - Solving ordinary differential equations
 - Numerical integration
 - Optimization
 - Interpolation
 - Signal processing
 - Statistics
 - Special mathematical functions
 - Mathematical & physical constants
- ...

```
import scipy as sp
```

convention

```
import scipy.linalg
```

import subpackages as needed

Singular Value Decomposition

- Computing SVD

Should be fast when built against good BLAS/Lapack library

```
import scipy.linalg
a = np.array([[7.3, 5.7], [-1.2, 5.3]])
u, s, v = sp.linalg.svd(a)
```

- Note: s is not a 2D-array, it is a 1D-array

```
s = np.diag(s)
```

- Let's check

```
A = u @ S @ v
delta = A - a
```

```
[[ 8.88178420e-16, 0.00000000e+00],
 [ 4.44089210e-16, 0.00000000e+00]]
```

Linear regression

- Reading data

```
x, y  
0.000e+00, 1.206e+00  
5.263e-02, 1.207e+00  
...  
data.csv
```

```
data = np.genfromtxt('data.csv',  
                     dtype=[np.float64, np.float64],  
                     delimiter=',', names=True)
```

- Linear regression

```
import scipy as sp  
import scipy.stats  
slp, intc, r, _, _ = sp.stats.linregress(data['x'],  
                                         data['y'])
```

Optimization: function definitions

- Minimize $f(x, y) = (x^2 + y^2)^2 - 2x^2 - 2y^2 + 0.1x$
- Define function

```
def f(X):  
    x = X[0]  
    y = X[1]  
    return (x**2 + y**2)**2 - 2*x**2 - 2*y**2 + 0.1*x
```

- Define gradient

```
def grad_f(X):  
    x = X[0]  
    y = X[1]  
    f_x = 4*(x**2 + y**2)*x - 4*x + 0.1  
    f_y = 4*(x**2 + y**2)*y - 4*y  
    return np.array([f_x, f_y])
```

Optimization

- Compute minimum

```
import scipy.optimize

x0 = np.array([1.0, 0.01])
xopt = scipy.optimize.fmin_cg(f, x0, fprime=grad_f,
                             disp=False)
```

- Many methods
 - Powell
 - Conjugate gradient
 - BFGS
 - Newton conjugate gradient
 - ...

Ordinary differential equations

- Rewrite higher order differential equation to set of first order equations

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\theta - q\omega + F_D \sin \Omega_D t \Leftrightarrow \begin{cases} \frac{d\theta}{dt} = \omega \\ \frac{d\omega}{dt} = -\frac{g}{l}\theta - q\omega + F_D \sin \Omega_D t \end{cases}$$

```
def func(t, y, g, l, q, F_D, Omega_D):
    return [
        y[1],
        -(g/l)*y[0] - q*y[1] + F_D*np.sin(Omega_D*t)
    ]
```

$$y[0] \equiv \theta \quad y[1] \equiv \omega$$

Jacobian for equations

- For many methods, convergence improves by specifying Jacobian

$$\begin{cases} f_1(\theta, \omega, t) = \omega \\ f_2(\theta, \omega, t) = -\frac{g}{l}\theta - q\omega + F_D \sin \Omega_D t \end{cases} \quad \begin{bmatrix} \frac{\partial f_1}{\partial \theta} & \frac{\partial f_1}{\partial \omega} \\ \frac{\partial f_2}{\partial \theta} & \frac{\partial f_2}{\partial \omega} \end{bmatrix}$$

```
def jac(t, y, g, l, q, F_D, Omega_D):
    return [
        [0.0, 1.0],
        [-g/l, -q]
    ]
```

Integrate ODEs

- Integrate from t_0 to t_{max} in steps

δt

integration method
RK(4, 5)

```
from scipy.integrate import ode
...
ode_sys = ode(func, jac).set_integrator('dopri5')
ode_sys.set_initial_value([theta0, omega0], t0)
ode_sys.set_f_params(g, l, q, F_D, Omega_D)
ode_sys.set_jac_params(g, l, q, F_D, Omega_D)

while ode_sys.successful() and ode_sys.t < t_max:
    ode_sys.integrate(ode_sys.t + delta_t)
    print(ode_sys.t, ode_sys.y[0], ode_sys.y[1])
```

Signal processing

- Remove noise from sound file (WAV)
 - Read WAV file

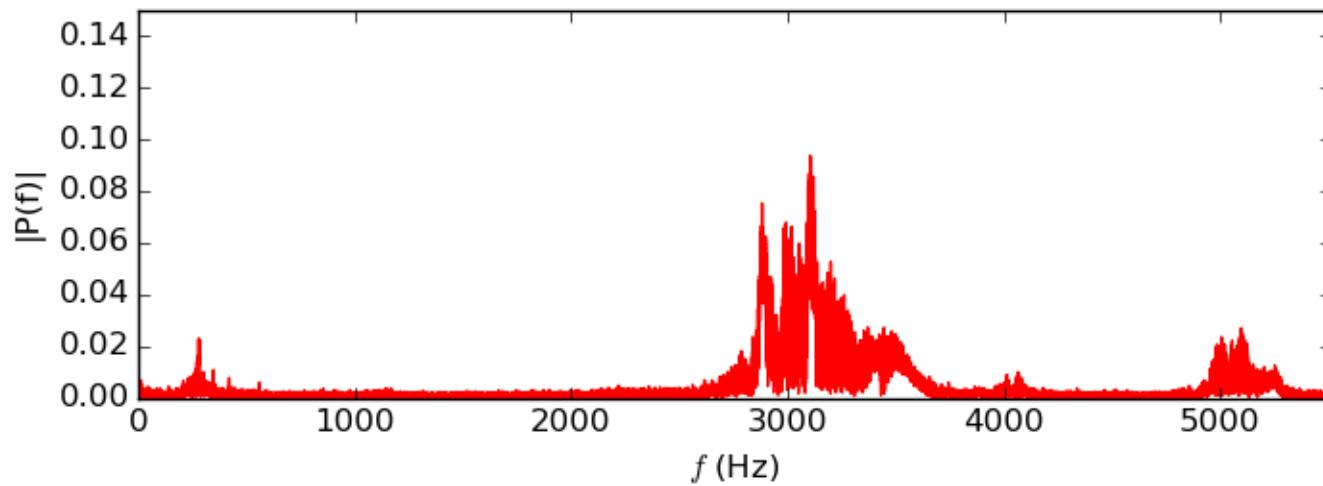
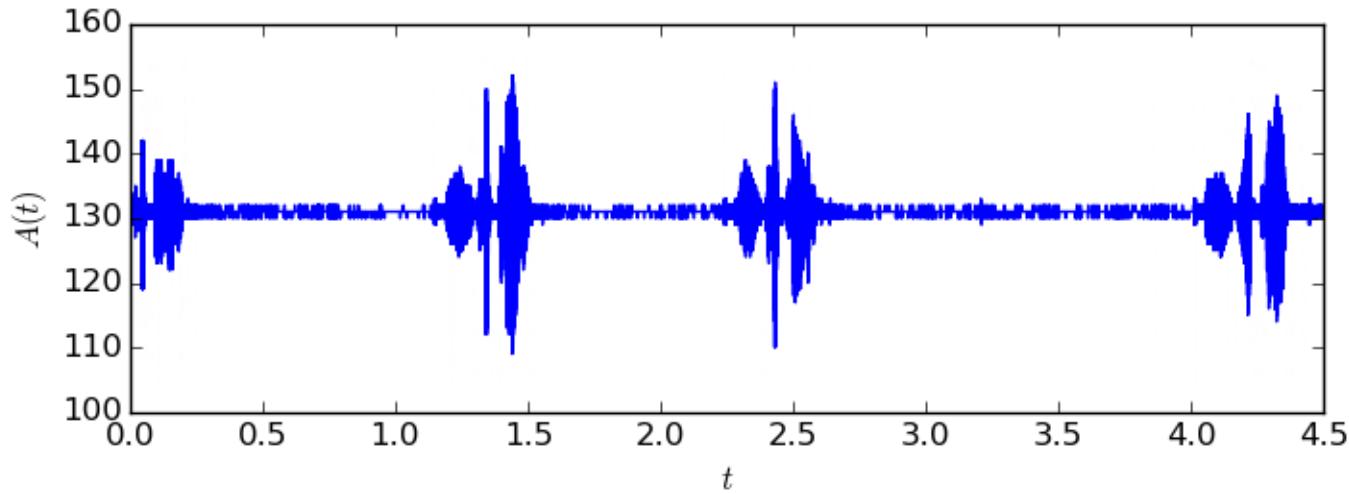
```
...  
from scipy.io import wavfile  
...  
sample_rate, signal = wavfile.read(wav_file_name)
```

- Perform FFT to compute frequency spectrum

```
...  
n = len(signal)  
freq = sample_rate*np.arange(n)/n  
Y = sp.fft(signal)/n
```



Original signal



Create highpass filter

- Import signal processing package

```
...  
import scipy.signal  
...
```

- Create IIR digital filter

```
...  
b, a = sp.signal.iirfilter(17, cutoff,  
                           rs=min_attenuation,  
                           btype='highpass',  
                           analog=False,  
                           ftype='cheby2')  
...
```

order of the filter

fraction of Nyquist frequency

minimum attenuation

filter type

IIR filter type

Filter signal

- Apply filter

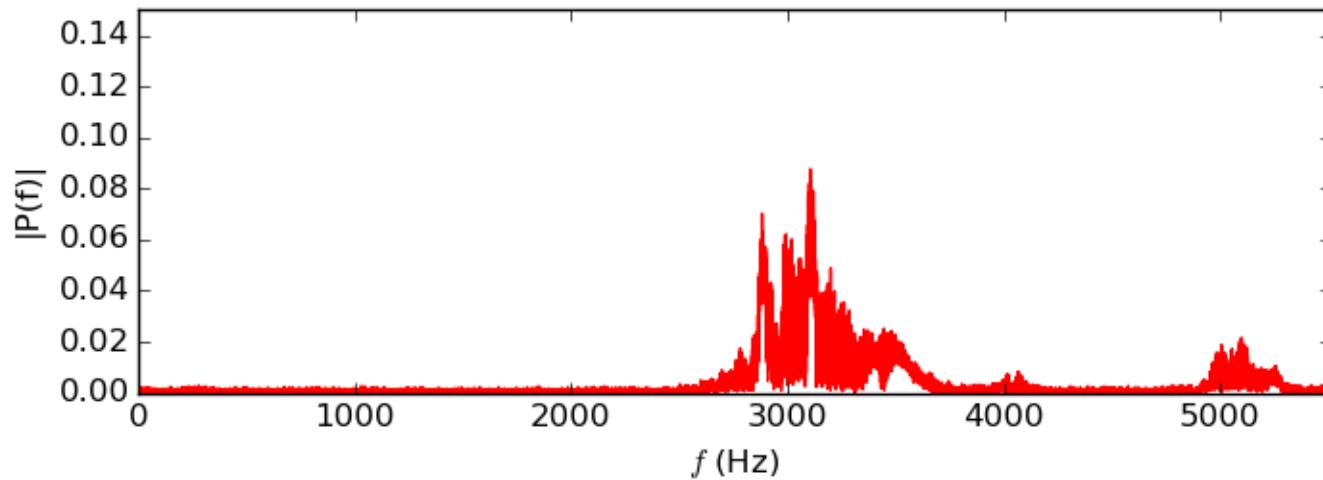
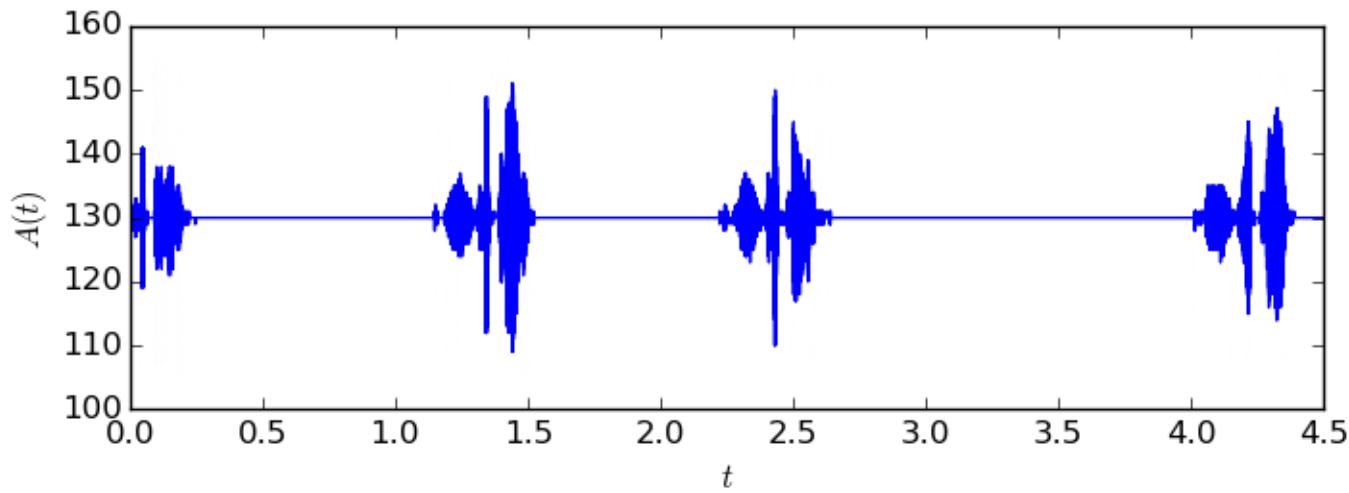
```
...
base = np.uint8(np.mean(signal))
filtered_signal = sp.signal.filtfilt(b, a, signal)
wav_signal = base + np.array(filtered_signal,
                               dtype=np.uint8)
...
```

- Write signal to WAV file

```
...
wavfile.write(filtered_wav_file_name, sample_rate,
               wav_signal)
...
```



Filtered signal



Code Pack 26

A. Scipy: See the code

Coding Bootcamp Code in Python

MATPLOTLIB

And some matplotlib...

- Rich Python plotting library
 - scatter plot
 - line plot
 - bar plot/histogram
 - heatmap
 - 3D surface plot
- Highly customizable plots
 - LaTeX labels/annotation
- Plot to screen, various file formats

Lots of features, this barely scratches the surface.

Convention: `import matplotlib.pyplot as plt`

Simple line plot

- Data: lists or numpy arrays

```
x = np.linspace(0.0, 20.0, 500)
y = np.exp(-mu*x)*np.cos(2.0*np.pi*x)
```

- Add plot

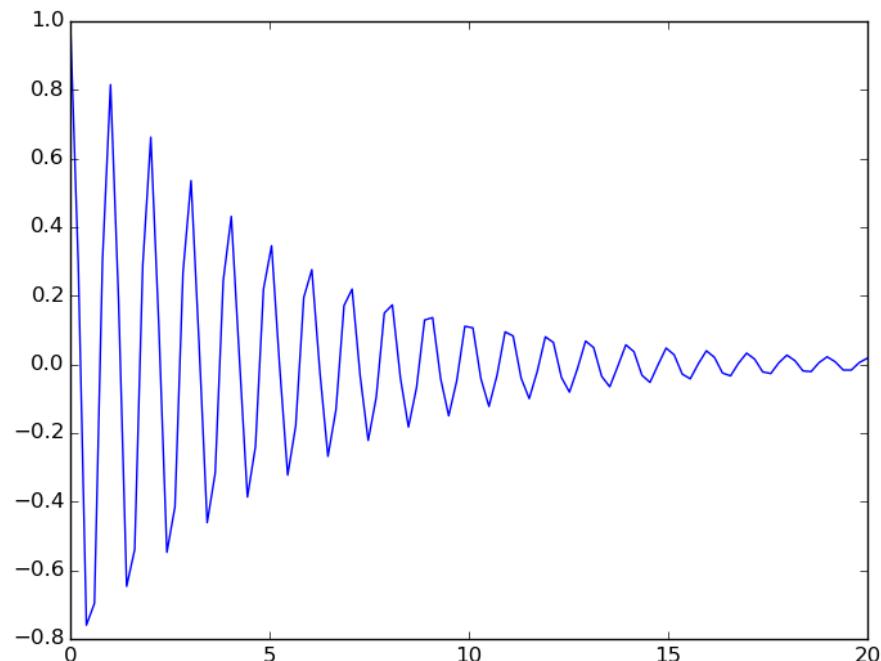
```
plt.plot(x, y)
```

- Show plot

```
plt.show()
```

- Save plot

```
plt.savefig(file_name)
```



Axis labels, annotation

- Add label for x and y axis

```
plt.xlabel(r'$t$', fontsize=14)  
plt.ylabel(r'$\theta(t)$', fontsize=14)
```

- Add annotation

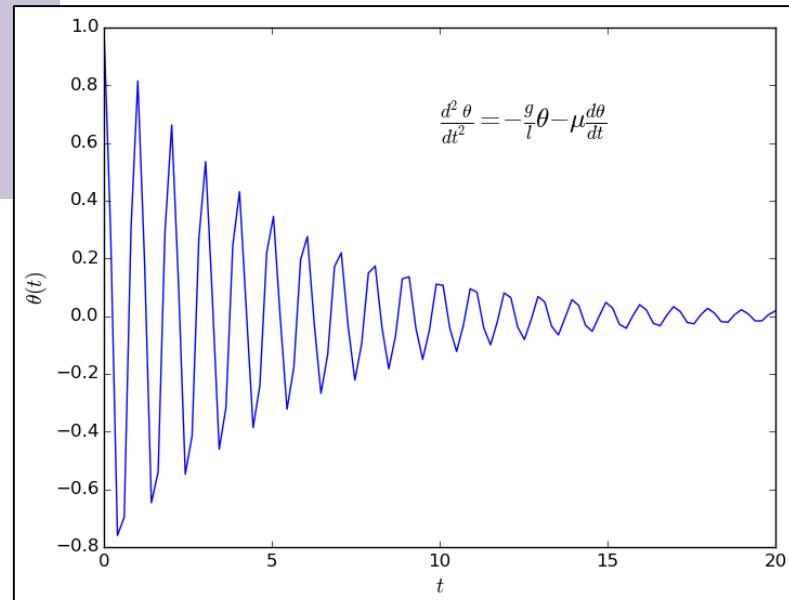
```
eq = (r'$\frac{d^2 \theta}{dt^2} = ' +  
      r'- \frac{g}{l} \theta' +  
      r'- \mu \frac{d\theta}{dt}$')  
plt.text(10.0, 0.65, eq, fontsize=18)
```

LaTeX notation, rendered

construct plot

≡

gradually enrich plt



Multiple functions on line plot

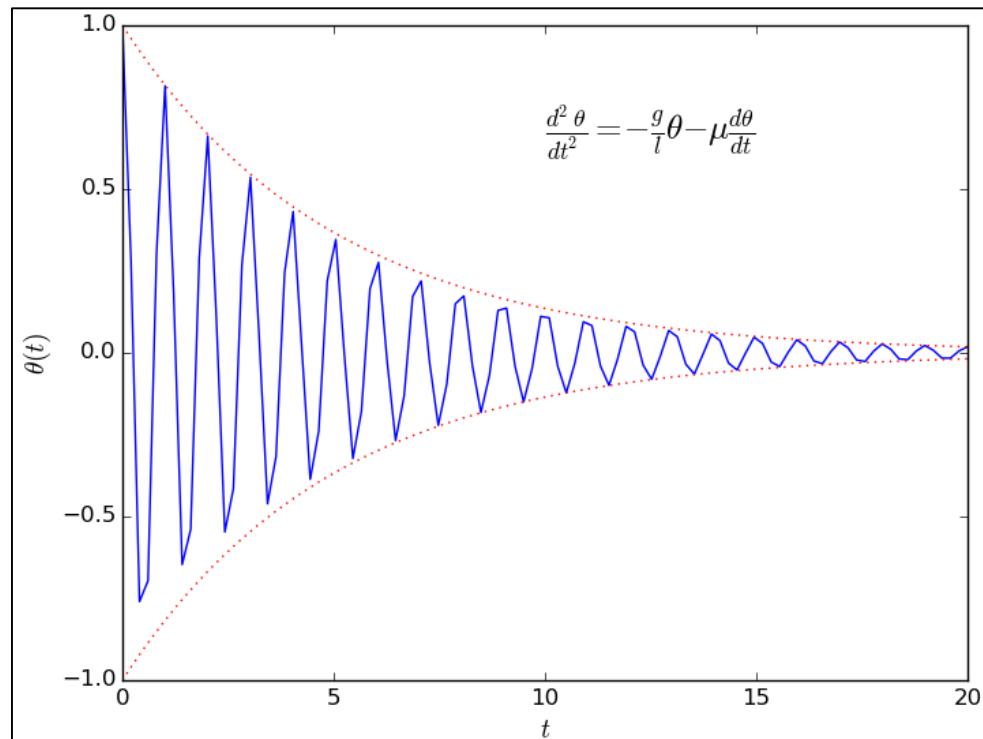
- Data

```
y_plus = np.exp(-mu*x)  
y_min = -np.exp(-mu*x)
```

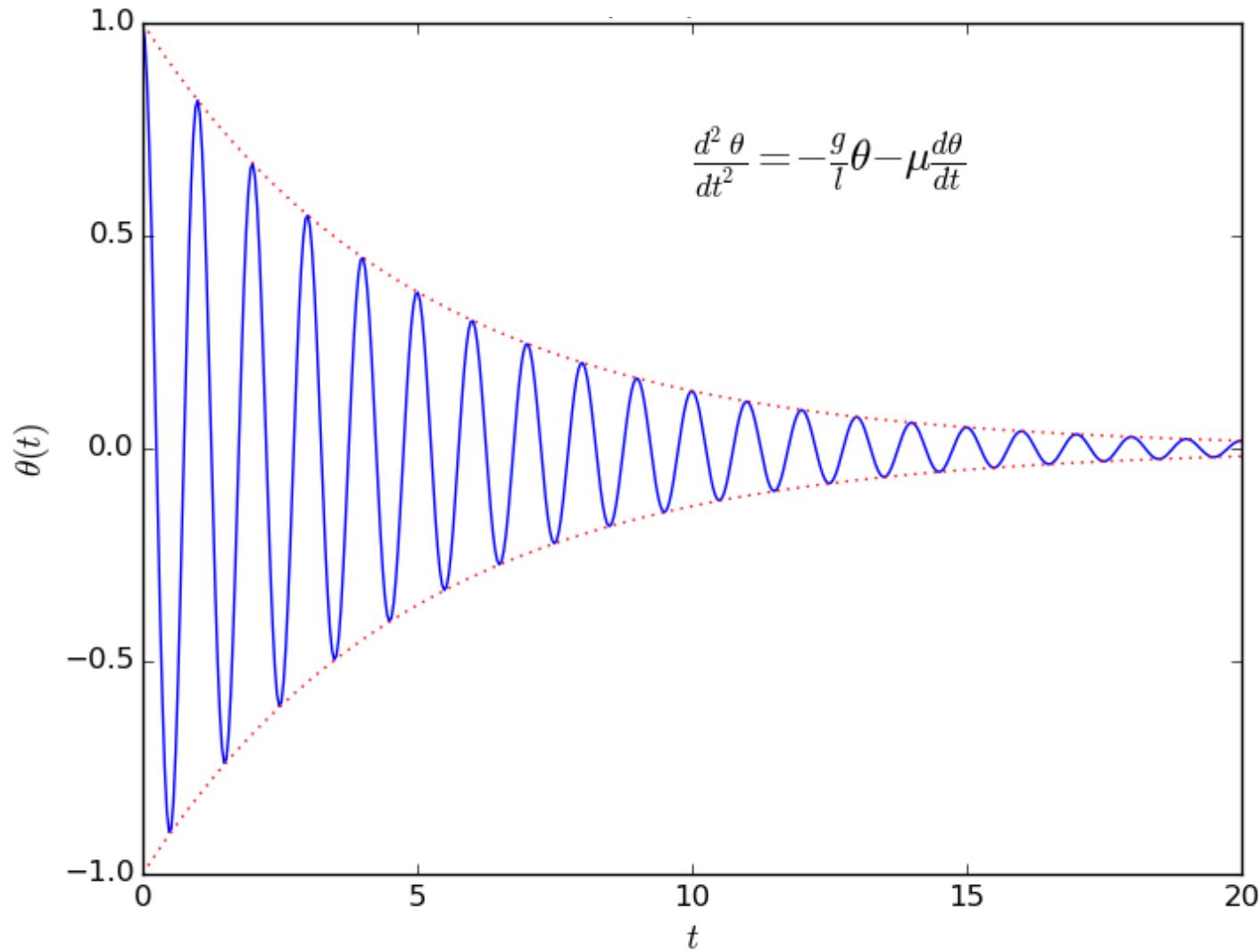
- Add to plot, line style, color

```
plt.plot(x, y_plus,  
         ':', alpha=0.8,  
         color='red',  
         linewidth=0.9)  
  
plt.plot(x, y_min,  
         ':', alpha=0.8,  
         color='red',  
         linewidth=0.9)
```

line type



Complete line plot



Histogram

- Data:

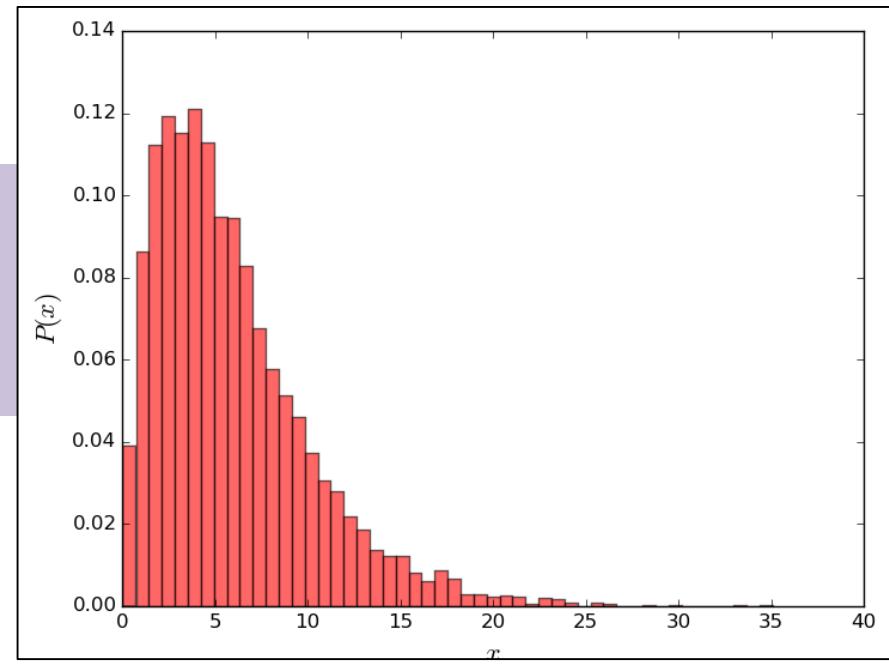
```
values = np.loadtxt('data.txt')  
bins = 50
```

```
13.3146868758  
9.66243828428  
6.44936354431  
2.81183151337  
9.55644862073  
...
```

data.txt

- Plot histogram

```
plt.hist(values, bins, normed=1,  
         color='red', alpha=0.6)  
plt.xlabel('$x$', fontsize=16)  
plt.ylabel('$P(x)$', fontsize=16)
```



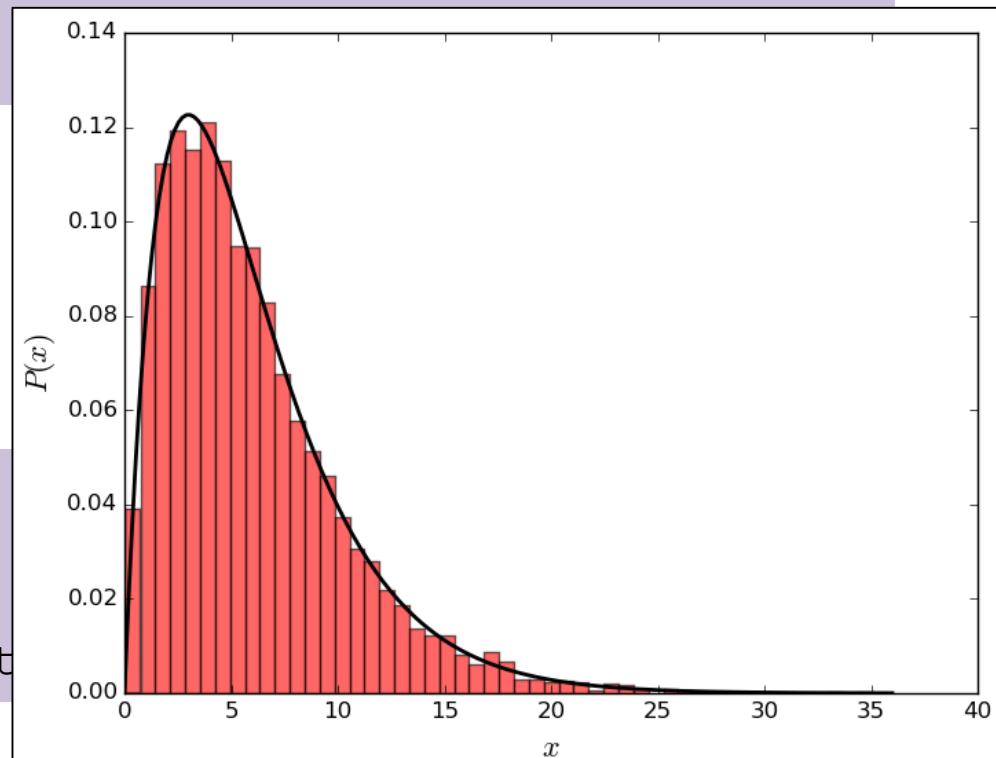
Line plot on histogram

- Data:

```
x = np.linspace(floor(np.min(values)), ceil(np.max(values))),  
        200)  
y = sp.stats.gamma.pdf(x, 2, scale=3.0)  
plt.plot(x, y, linewidth=2.0,  
        color='black')
```

- Reminder

```
import numpy as np  
import scipy as sp  
import scipy.stats  
import matplotlib.pyplot as plt
```



Heat map data

- Function to plot

```
def f(x, y, x0=0.0, freq=1.0, beta=0.5):  
    r = np.sqrt((x - x0)**2 + y**2)  
    return np.exp(-beta*r)*np.cos(2.0*np.pi*freq*r)
```

- Data

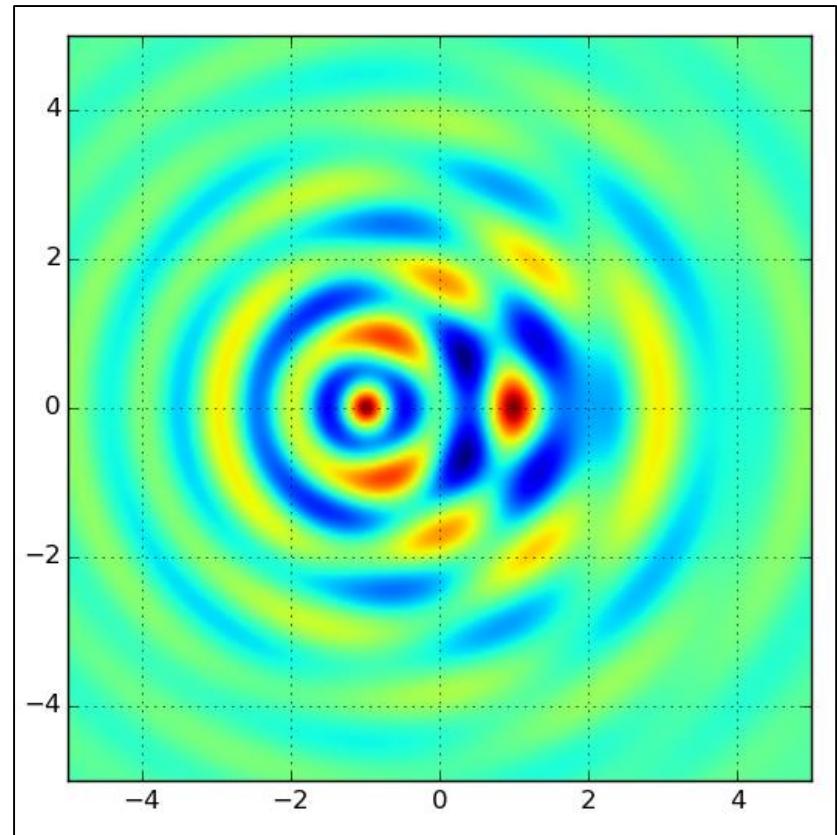
```
x = np.linspace(x_min, x_max, points)  
y = np.linspace(y_min, y_max, points)  
X, Y = np.meshgrid(x, y)  
z = f(X, Y, x0=x0_1, freq=f_1) + f(X, Y, x0=x0_2, freq=f_2)
```

Heat map plot

- Plot map

```
plt.imshow(z, extent=[x_min, x_max, y_min, y_max])  
plt.grid(True)
```

Explore color maps,
helps interpret data!
Brewer schemes



3D surface plot I

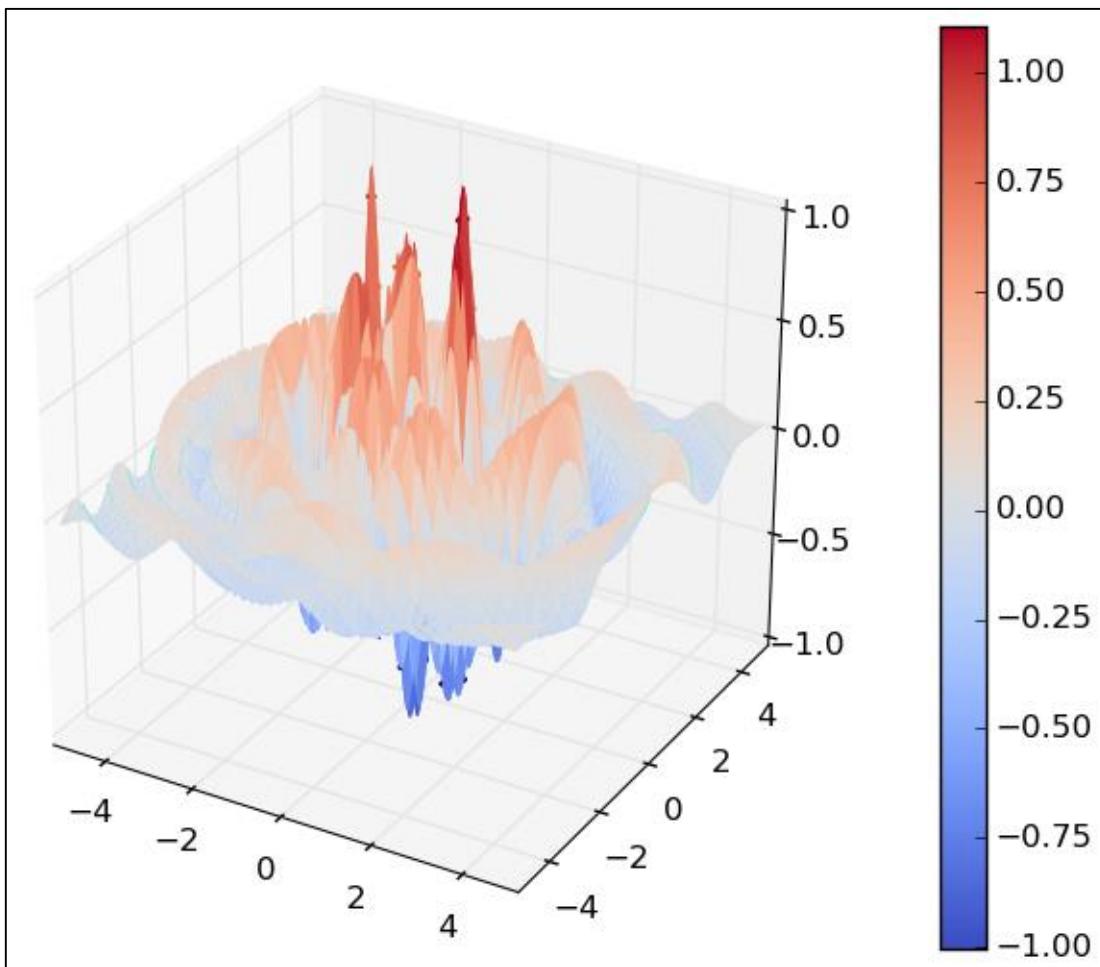
- Importing extra modules

```
from mpl_toolkits.mplot3d import Axes3D  
from matplotlib import cm
```

- Plot

```
figure = plt.figure()  
axes = figure.gca(projection='3d')  
axes.set_xlim(x_min, x_max)  
axes.set_ylim(y_min, y_max)  
axes.set_zlim(z_min, z_max)  
surface = axes.plot_surface(X, Y, z,  
                           rstride=4, cstride=4,  
                           cmap=cm.coolwarm,  
                           linewidth=0)  
figure.colorbar(surface)
```

3D surface plot II



References

- ColorBrewer 2.0: advice on choosing appropriate color maps
<http://colorbrewer2.org/>
- Overview of data visualization types & libraries for Python
<https://python-graph-gallery.com/>

Code Pack 27

A. Matplotlib:

1-Figures_Subplots_and_layouts

2-Plotting_Methods_Overview

3-HowToSpeakMPL

4-Limits_Legends_and_Layouts

5-Artists

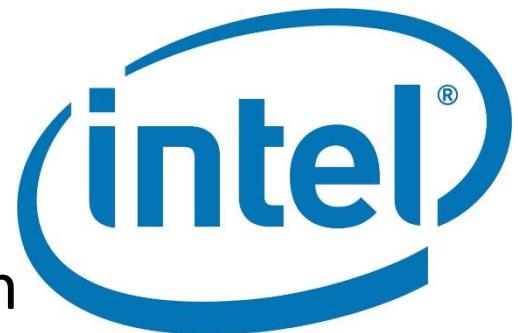
6-mpl_toolkits

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV

OpenCV

- OpenCV is an Image Processing library created by Intel on 1999 and maintained by Willow Garage.
 - First Release was in 2000
- Available for C, C++, and Python
- Newest update is version 4.0
- Open Source and free (3-clause BSD License)
- Easy to use and install



opencv-python

- Officially, OpenCV releases two types of Python interfaces,
- cv and cv2.
- cv (**legacy**): all OpenCV data types are preserved as such.
 - For example, when loaded, images are of format cvMat, same as in C++.
 - For array operations, there are several functions like cvSet2D, cvGet2D, etc. And some discussions say, they are slower.
- cv2 (**latest**): everything is returned as NumPy objects like ndarray and native Python objects like lists, tuples, dictionary, etc.
 - NumPy is a highly stable and fast array processing library.

Easy installation (now!):

- **pip install opencv-python**
(core modules)
- pip install opencv-contrib-python (extra modules)
- And also:
- **pip install numpy**
- pip install matplotlib
- pip install scipy

```
C:\Users\rjamp>python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'4.0.0'
>>>
```

Getting Started with Images

- First:

```
import cv2
```

```
#or then and very common  
import cv2 as cv
```

- Then:

- read an image - cv2.imread()
- display it - cv2.imshow()
- save it - cv2.imwrite()

Importing an image

- Create a Python module and write the following code:

```
import cv2  
  
img = cv2.imread('duomo.jpg',1)  
cv2.imshow("Output Window", img)  
cv2.waitKey()
```

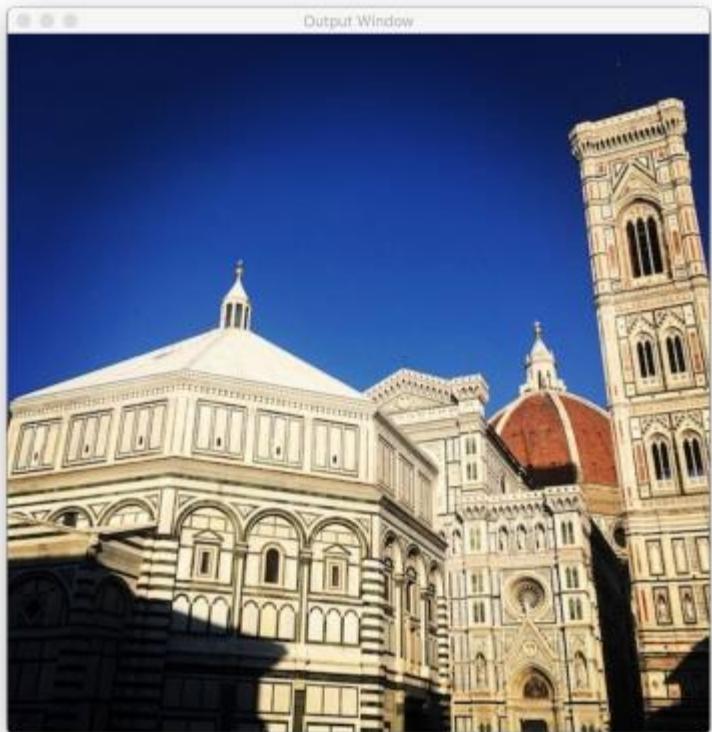
- This code imports an image and outputs it to a window and waits for any user keyboard input to terminate.

cv2.imread() function

- This function is used to load an image and store it into a variable

```
img = cv2.imread('duomo.jpg', 1)
```

- This function accepts 2 parameters:
 1. The filename of the image
 2. Colour Approach:
 - 1: Colour, neglecting transparency
 - 0: Greyscale
 - 1: Colour together with the alpha channel



```
img = cv2.imread('duomo.jpg',1)
```



```
img = cv2.imread('duomo.jpg',0)
```

cv2.imshow() function

- This function is used to display an image in a window. `cv2.imshow("Output Window", img)`
- This function accepts 2 parameters:
 1. The name of the output window
 2. The image to be displayed in the output window
- The window automatically fits the image size.
- **Matplotlib** can be used as an alternative

cv2.waitKey() function

- This is a keyboard binding function

cv2.waitKey()

- A single argument value in milliseconds:
 1. 0 or no argument: wait indefinitely for keyboard interrupt
 2. Any other value: display the window for the duration of that value in ms
- This function returns the ASCII value of the key pressed and if stored in a variable, it can be used to perform subsequent logical operations.

Getting Started with Videos

- cv2.VideoCapture()
- cv2.VideoWriter()

cv2.VideoCapture() Object

- The video capture object allows us to manipulate captured frames from a camera.

```
cap = cv2.VideoCapture(0)
```

- The argument is either the video filename or camera index, 0 for webcam. Allows the handling of each frame.
- After being used, the capture has to be released:

```
cap.release()
```

Using the webcam feed

```
cap = cv2.VideoCapture(0)
while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Our operations on the frame come here
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Display the resulting frame
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
```

Importing a video

```
cap = cv2.VideoCapture('vtest.avi')
while(cap.isOpened()): #returns true when there is another frame
    #to process
    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
```

cv2.VideoCapture() Object

- The video writer object allows us to save a video to an output file.

```
out = cv2.VideoWriter('output.avi', fourcc,  
20.0, (640,480))
```

- FourCC** is a 4-byte code used to specify the video codec. The list of available codes can be found in fourcc.org. It is platform dependent.
- After being used, the capture has to be released:

```
out.release()
```

Drawing Functions in OpenCV

- OpenCV can draw different geometric shapes with these functions:

cv2.line()

cv2.circle()

cv2.rectangle()

cv2.ellipse()

cv2.putText()

...

Handle mouse events in OpenCV

- Draw stuffs with your mouse
- Use it as a Paint-Brush

```
cv2.setMouseCallback()
```

- Can bind trackbar to OpenCV windows
- Trackbar as the Color Palette

```
cv2.getTrackbarPos()
```

```
cv2.createTrackbar()
```

...

Code Pack 28

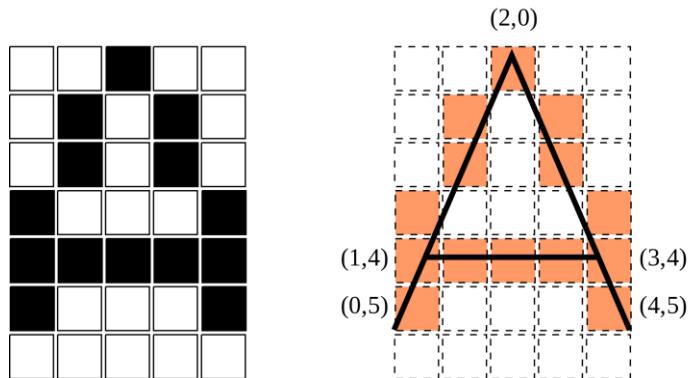
1. Introduction to OpenCV with Python

Coding Bootcamp Code in Python

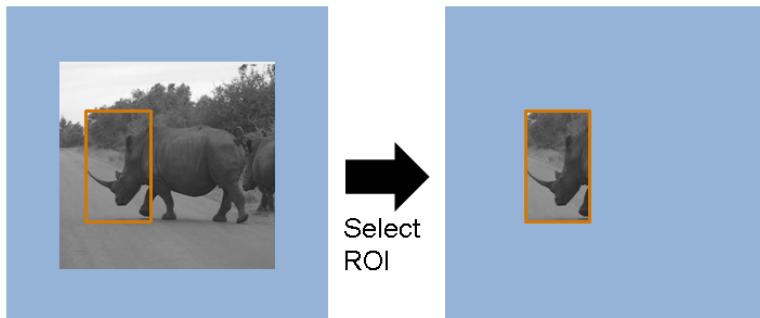
INTRODUCTION TO OPENCV: CORE OPERATIONS

Basic Operations on Images

- Access pixel values and modify them



- Access image properties (`img.shape`, `img.size...`)
- Setting Region of Interest (ROI)



- Splitting and Merging images

Arithmetic Operations on Images

- Addition, subtraction or bitwise operation.
- cv2.add()
- cv2.addWeighted()
- cv2.subtract()
- cv2.bitwise_and()
- cv2.bitwise_or()
- cv2.bitwise_xor()
- cv2.bitwise_not()

[https://docs.opencv.org/4.0.1/d2/de8/group_core_array.html](https://docs.opencv.org/4.0.1/d2/de8/group__core__array.html)

Performance Measurement and Improvement Techniques

- In image processing, it is mandatory that your code is not only providing the correct solution, but also in the fastest manner.
- Useful functions to measure the performance of your code.
- cv2.getTickCount
- cv2.getTickFrequency

Code Pack 28

1. ~~Introduction to OpenCV with Python~~
2. Core Operations

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV: IMAGE PROCESSING IN OPENCV

Changing Colorspaces

In OpenCV is also possible to convert images from one colorspace to another, like:

- BGR ↔ Gray
- BGR↔ HSV
- etc

There are more than 150 colorspace conversion methods available in OpenCV.

- cv2.cvtColor()
- cv2.inRange()
- ...

Changing colorspace makes the simplest method in object tracking:

- Take each frame of the video



Original Frame

- Convert from BGR to HSV color-space



Original Frame

Mask Image

- Threshold the HSV image for a range of blue color



Original Frame

Mask Image

Final Result

- Now extract the blue object alone and get the new position.

Image Thresholding - Simple thresholding

- Thresholding is a method of image segmentation.
- **Simple thresholding** - If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black).



Image Thresholding - Adaptive Thresholding

- **Adaptive Thresholding** - In this, the algorithm calculate the threshold for a small regions of the image.
- So we get different thresholds for different regions.



Image Thresholding - Functions

- cv2.threshold()
- cv2.adaptiveThreshold()

Geometric Transformations of Images

- Scaling

`cv2.resize()`

- Translation

`cv2.warpAffine()`

- Rotation

`cv2.getRotationMatrix2D()`

- Affine Transformation

`cv2.getAffineTransform() then`

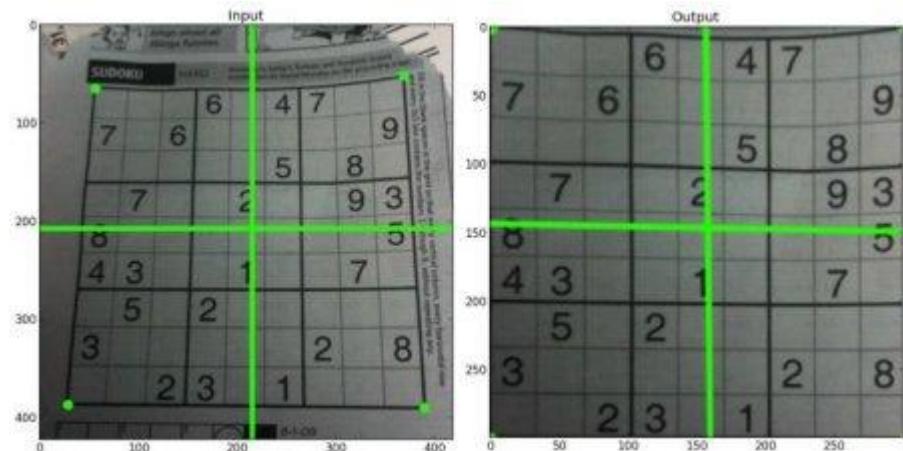
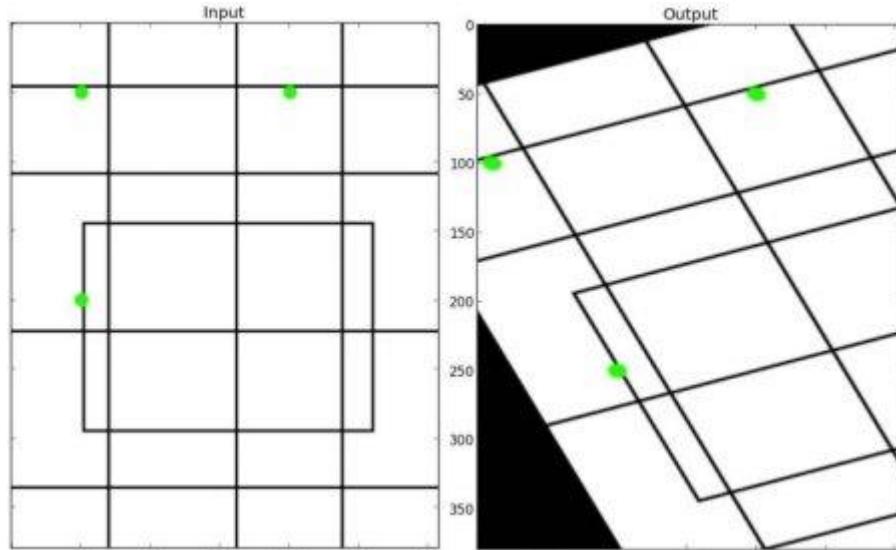
`cv2.warpAffine()`

- Perspective Transformation

`cv2.getPerspectiveTransform() then`

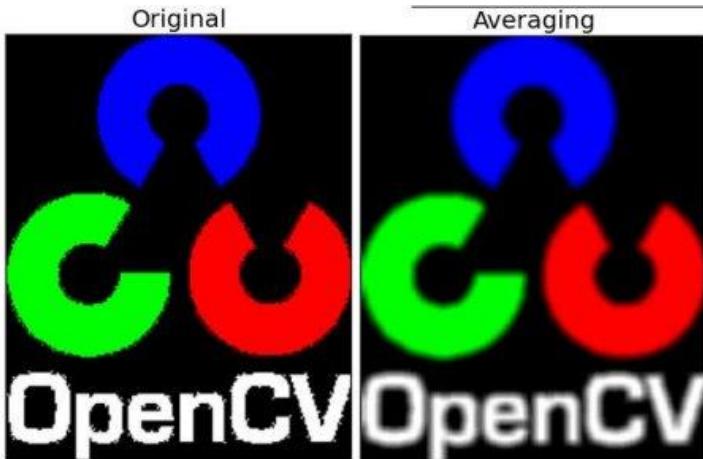
`cv2.warpPerspective()`

Affine Transformation and Perspective Transformation



Smoothing Images - Blur images with various low pass filters.

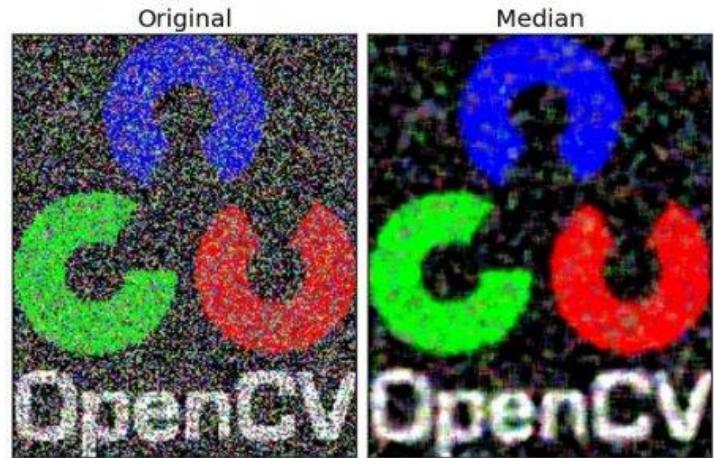
- Averaging – cv2.blur()



- Gaussian - cv2.GaussianBlur()



- Median – cv2.medianBlur()



- Bilateral-
cv2.bilateralFilter()



Morphological Transformations

- Erosion, Dilation, Opening, Closing ...

`cv2.erode()`, `cv2.dilate()`, `cv2.morphologyEx()` ...

Erosion



Base



Closing



Dilation



Opening



Morphological Transformations

Image I



Erosion $I \ominus B$



Dilatation $I \oplus B$



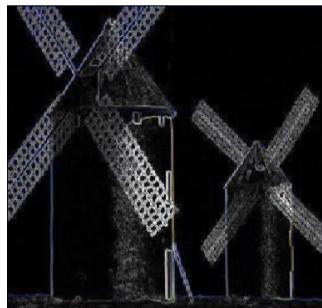
Opening $I \circ B = (I \ominus B) \oplus B$



Closing $I \bullet B = (I \oplus B) \ominus B$



Grad(I)= $(I \oplus B) - (I \ominus B)$



TopHat(I)= $I - (I \ominus B)$

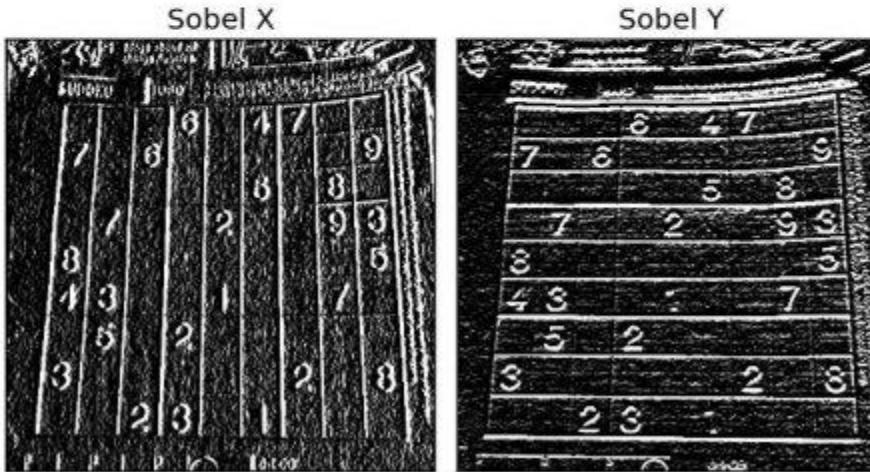
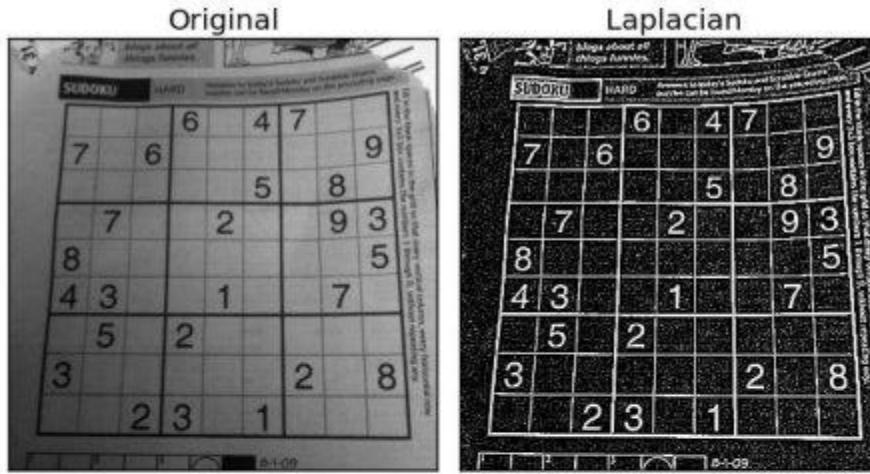


BlackHat(I)= $(I \oplus B) - I$



Image Gradients

- cv2.Sobel(), cv2.Laplacian()



Canny Edge Detection

- Canny Edge Detection is a popular edge detection algorithm.
- It was developed by John F. Canny in 1986.
- cv2 .Canny ()

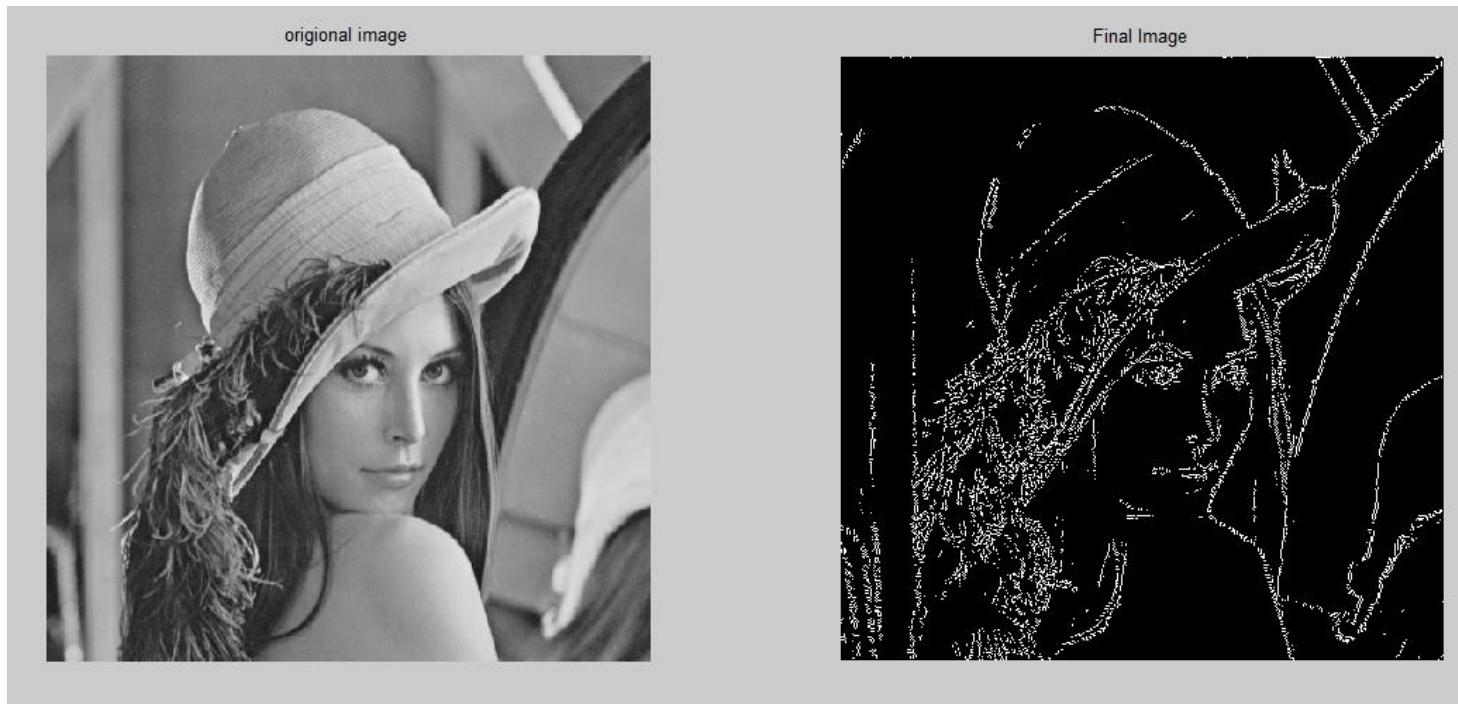


Image Pyramids

- Image segmentation by pyramids - Gaussian and Laplacian pyramids
- **Gaussian pyramid:** Used to downsample images
- **Laplacian pyramid:** Used to reconstruct an upsampled image from an image lower in the pyramid (with less resolution)
- `cv2.pyrUp()`
- `cv2.pyrDown()`

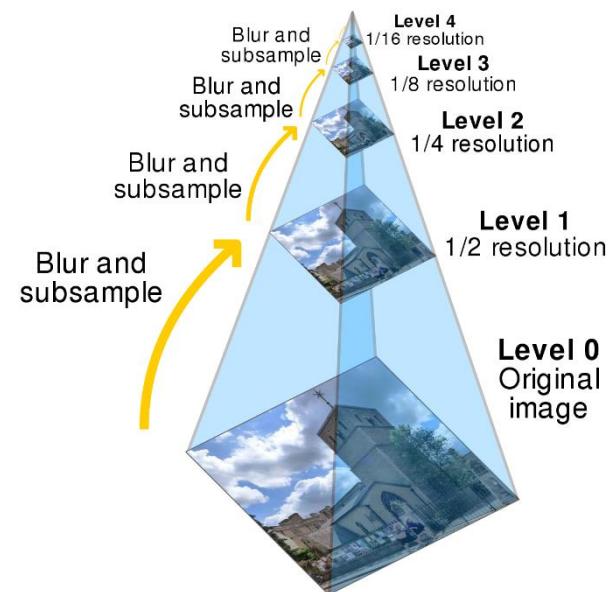


Image Pyramids

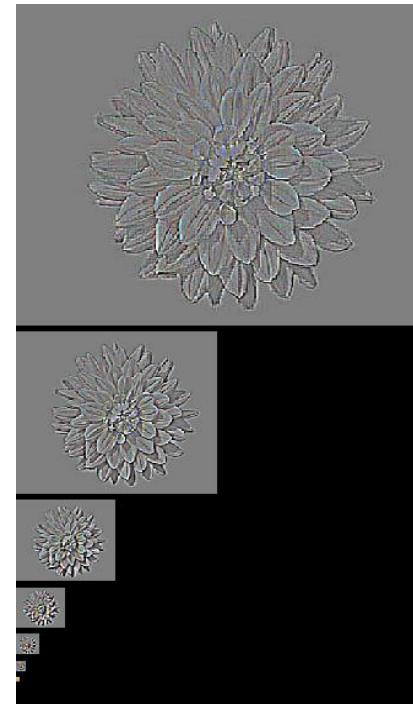
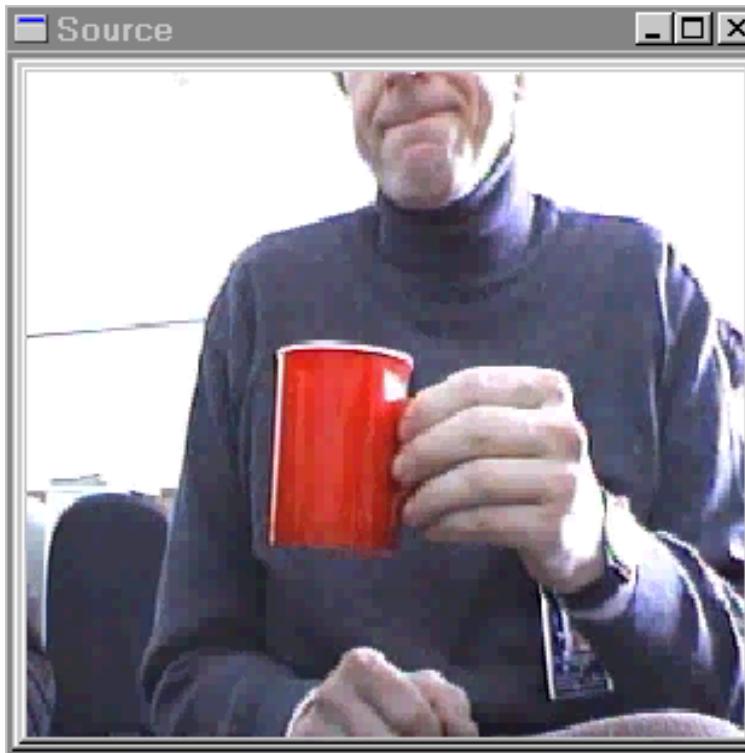


Image Pyramids

On still pictures



And on movies



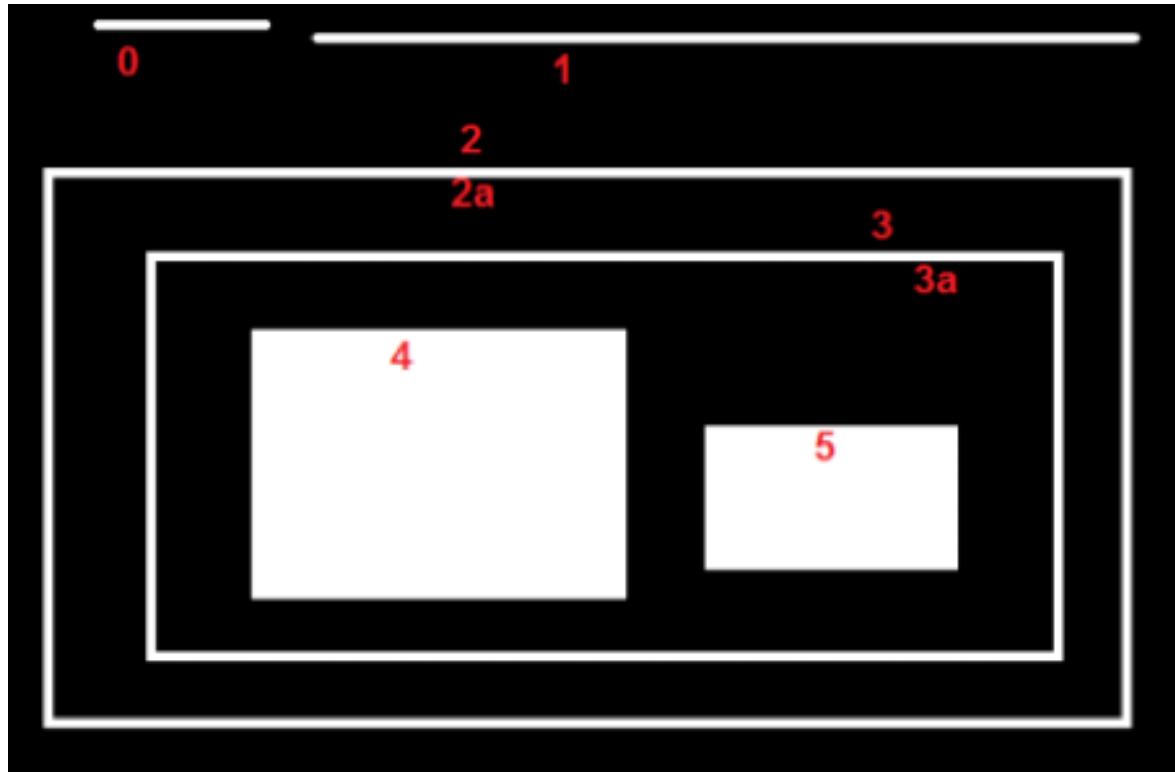
Contours in OpenCV

- Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity.
- The contours are a useful tool for shape analysis and object detection and recognition.

0	0	0	1	1	1	0	0	0	0	2	2	2	2	2	0	3	3	3	3	
0	0	0	0	1	1	1	1	0	0	0	2	2	2	2	2	0	0	3	3	3
0	0	0	1	1	1	1	1	1	0	0	2	2	2	2	2	0	0	3	3	3
0	0	0	1	1	1	1	1	1	1	0	2	2	2	2	2	0	0	3	3	3
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	3	3
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	3
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	3
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1

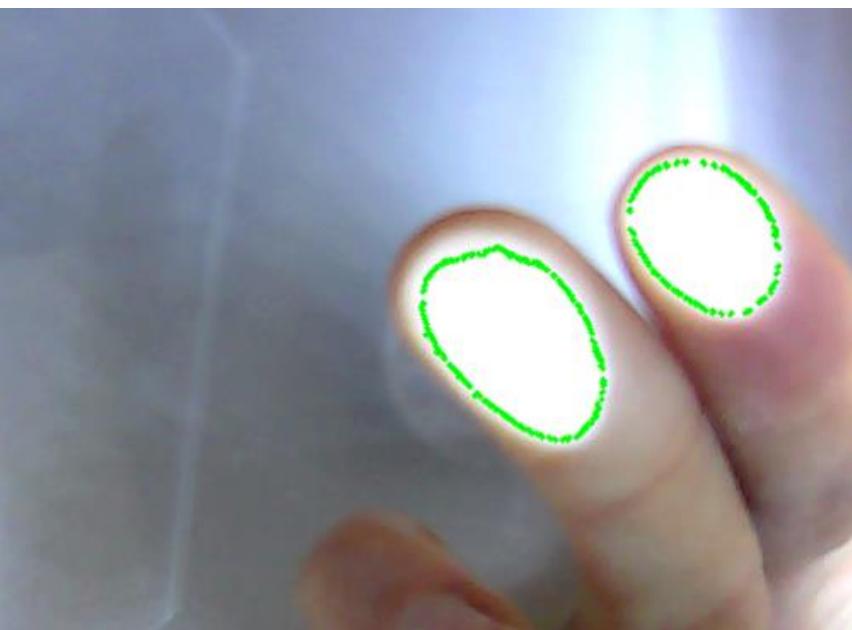
Connected components = find contours

- contours, hierarchy = cv2.findContours()

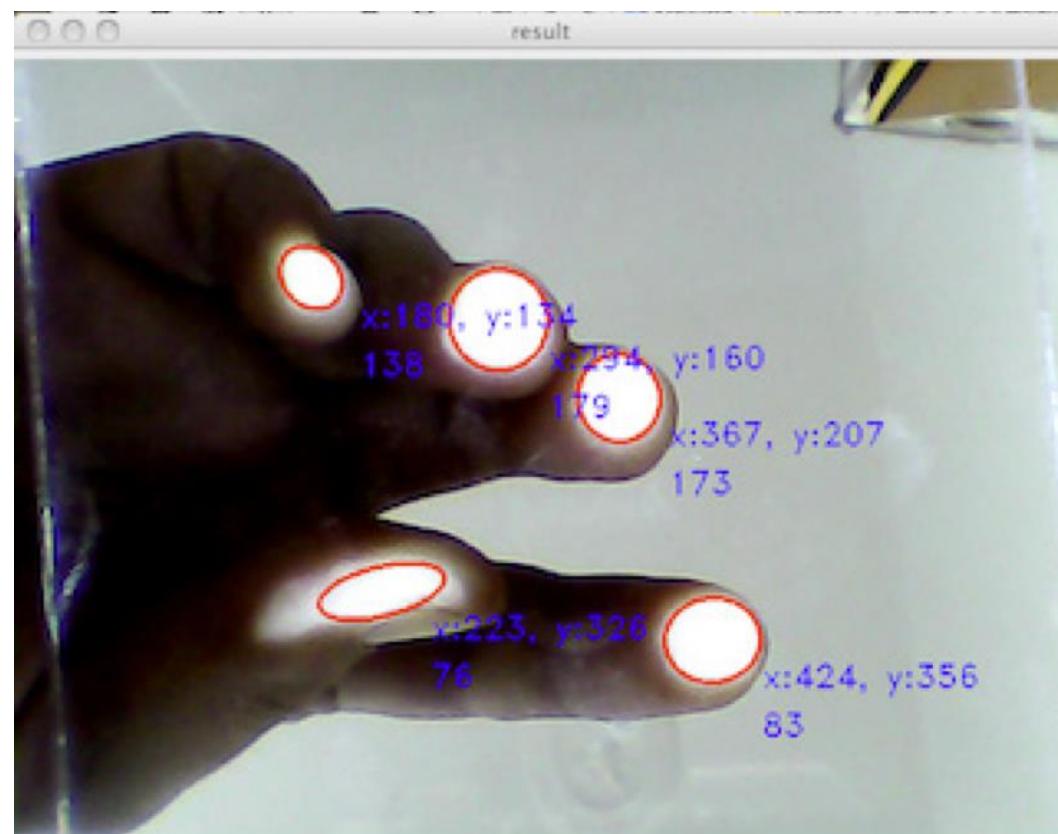


Find center of a contour:

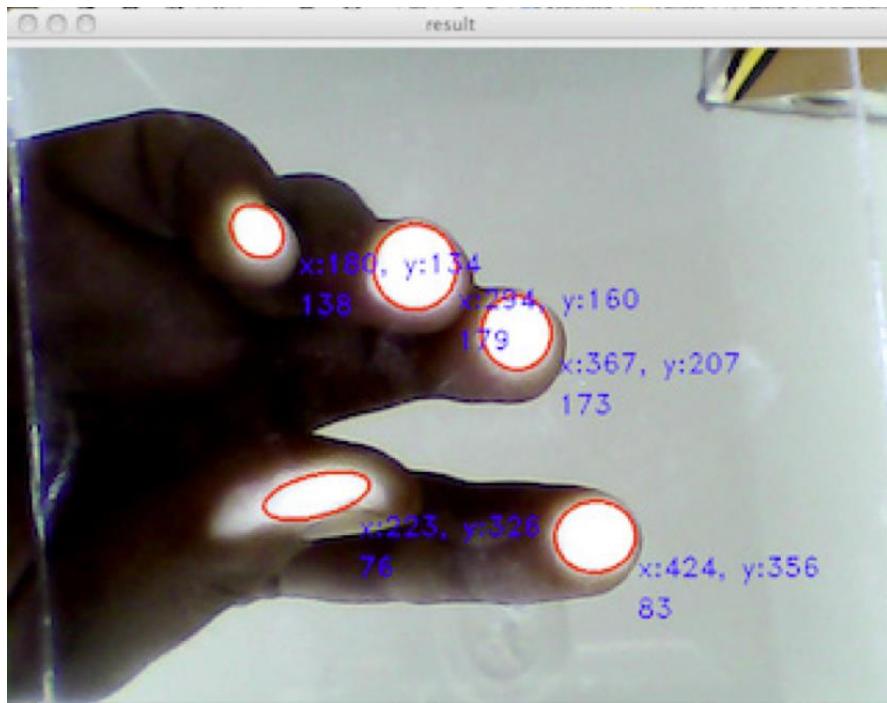
- `cv2.minEnclosingCircle(contour)`



contour = path of points



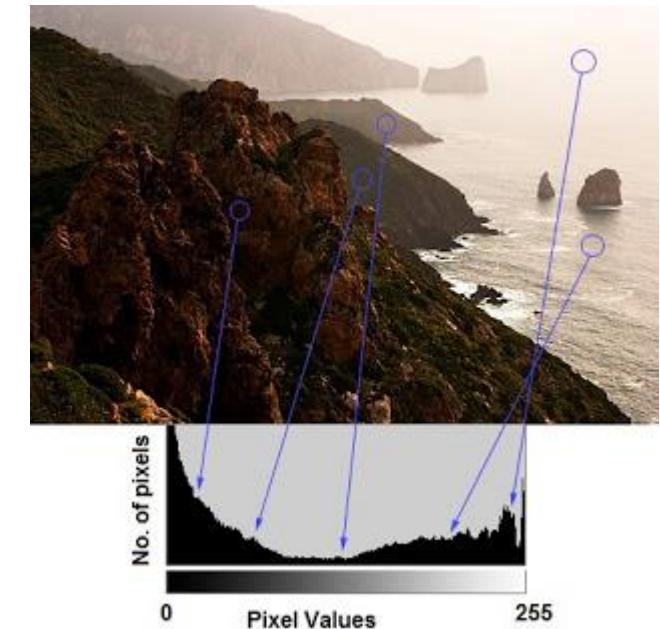
fit circle = center, radius



```
cv2.circle(frame,draw_center,draw_radius,(0,255,0),2)  
  
cv2.putText(frame,textstring,(int(x+50),int(y)),  
           cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,255,255),1)
```

Histograms in OpenCV

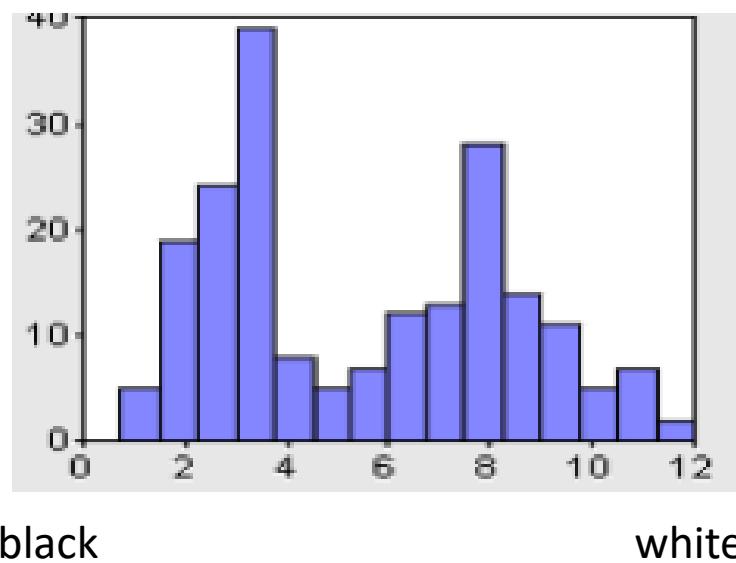
- Histogram is a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.



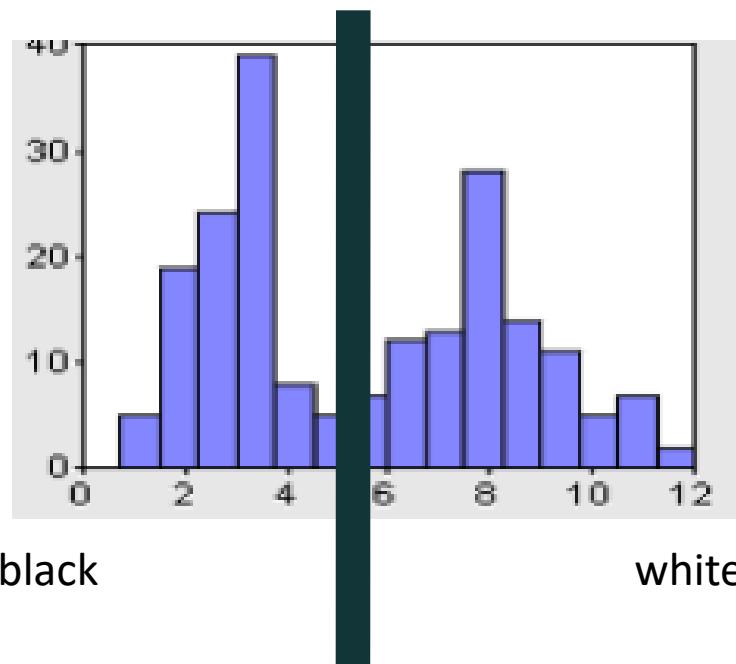
- `cv2.calcHist()`
- `np.histogram()`



let's consider
this image



here's how
the histogram works



where would you place the threshold to find all trees?

Image Transforms in OpenCV

- Fourier Transform of images using OpenCV
- cv2.dft(), cv2.idft()
- There are FFT functions available in Numpy

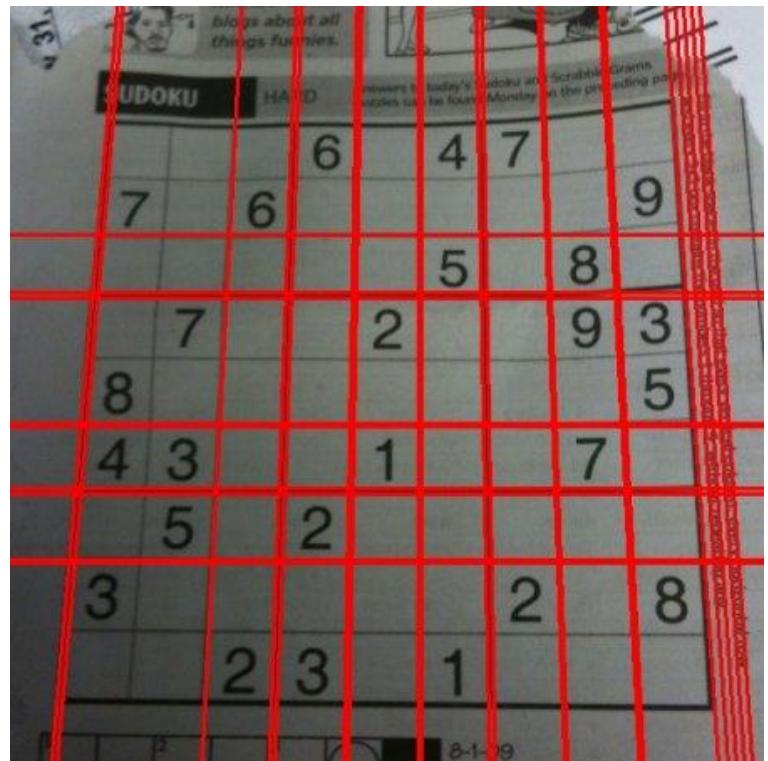
Template Matching

- Template Matching is a method for searching and finding the location of a template image in a larger image.
- OpenCV comes with the `cv2.matchTemplate()`



Hough Line Transform

- The Hough transform is an incredible tool that lets you identify lines and other shapes (circles, angles, etc).
- cv2.HoughLines()
- cv2.HoughLinesP()



Hough Circle Transform

- Use Hough Transform to find circles in an image
- cv2.HoughCircles()



Image Segmentation with Watershed Algorithm

- In geology, a watershed is a divide that separates adjacent catchment basins

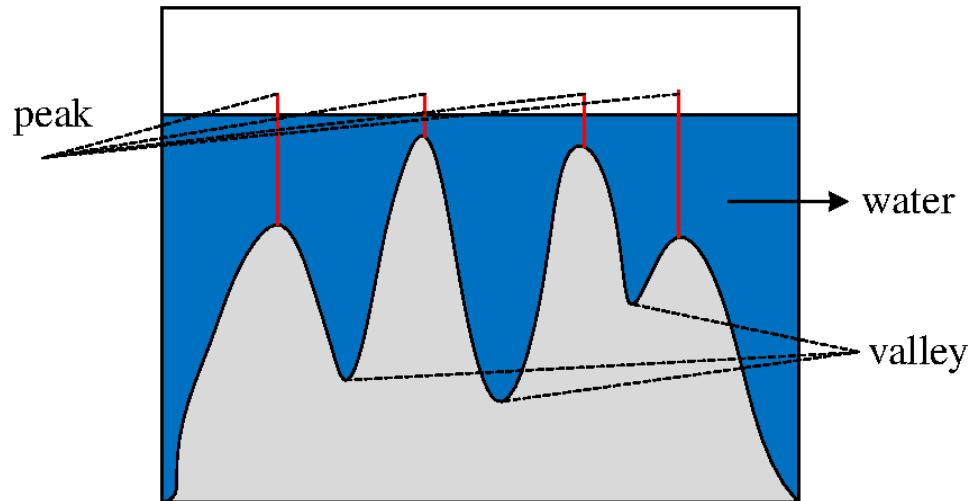
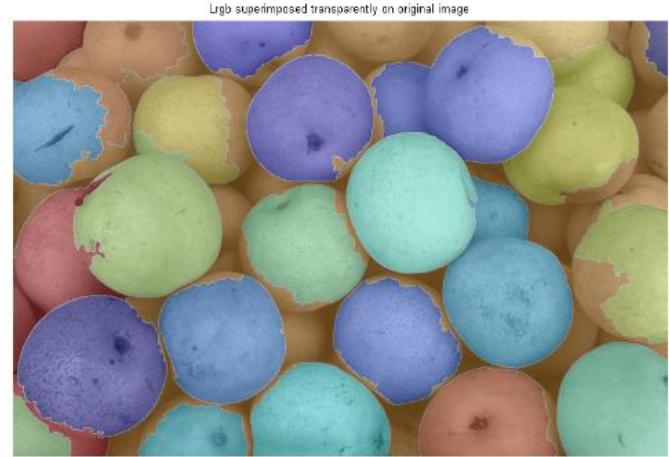


Image Segmentation with Watershed Algorithm

- cv2.watershed()



Or...



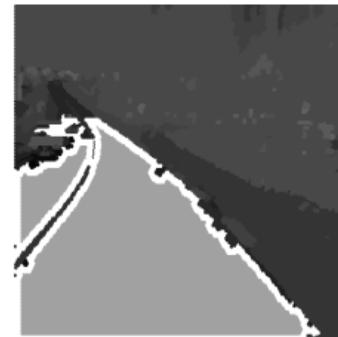
Original
image



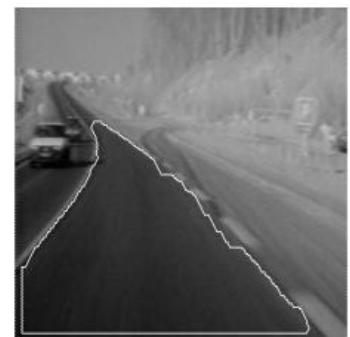
Mosaic-
image



Watershed
of mosaic-
image



Lanes
markers
enhancement

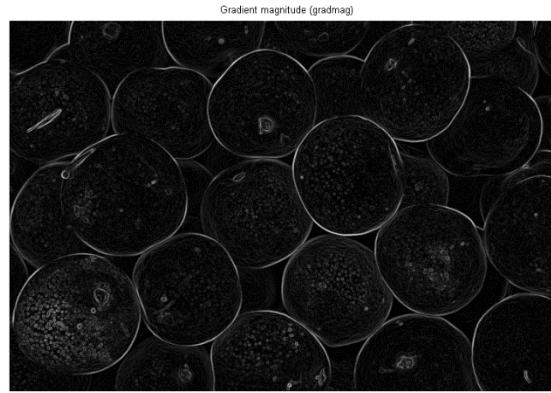


Final result

Watershed Transformation Process



Source: A gray scale image

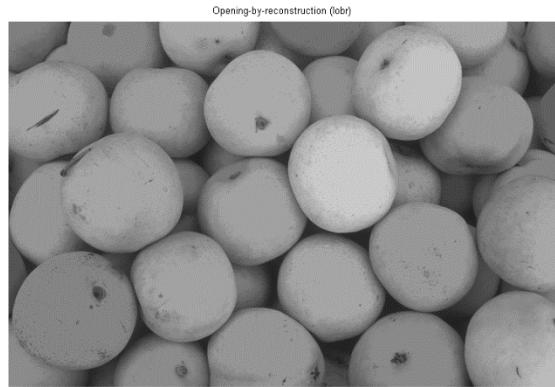


Step 1: Use the Gradient Magnitude as the Segmentation Function -
The gradient is high at the borders of the objects and low (mostly) inside the objects.

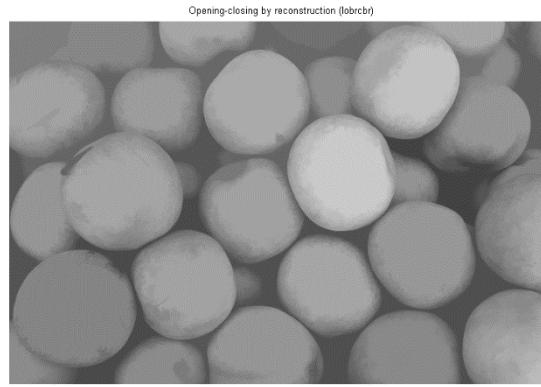


Step 2: Mark the foreground objects

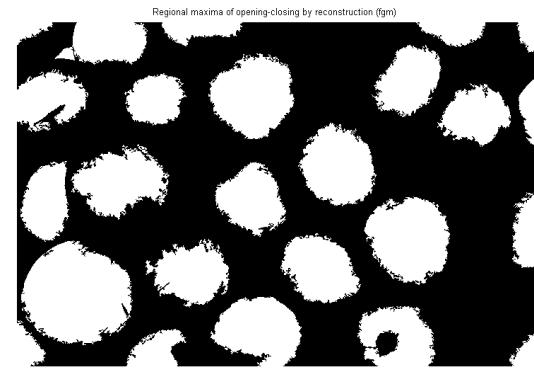
Watershed Transformation Process



Step 3: computing the opening-by-reconstruction of the image

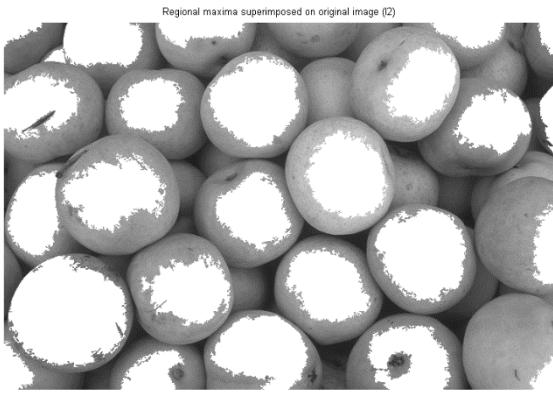


Step 4: Following the opening with a closing can remove the dark spots and stem marks.

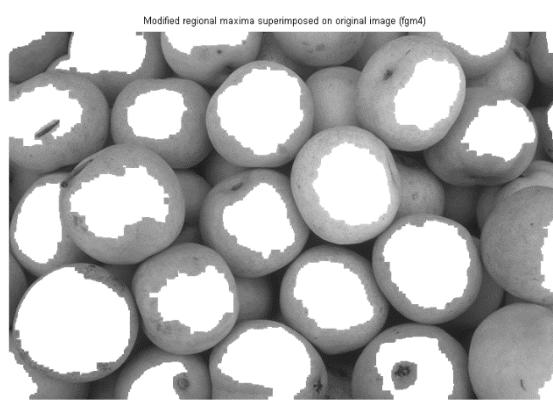


Step 5: Calculate the regional maxima to obtain good foreground markers.

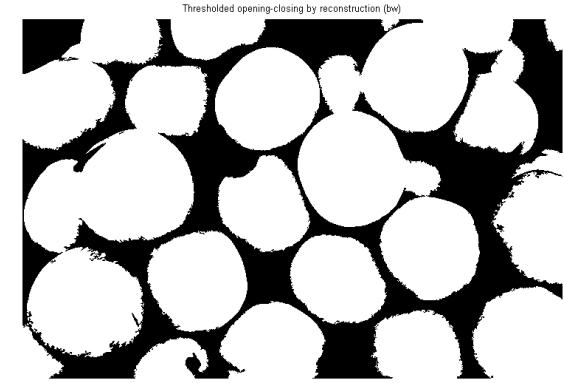
Watershed Transformation Process



Step 6: Superimpose the foreground marker image on the original image, Notice that the foreground markers in some objects go right up to the objects' edge

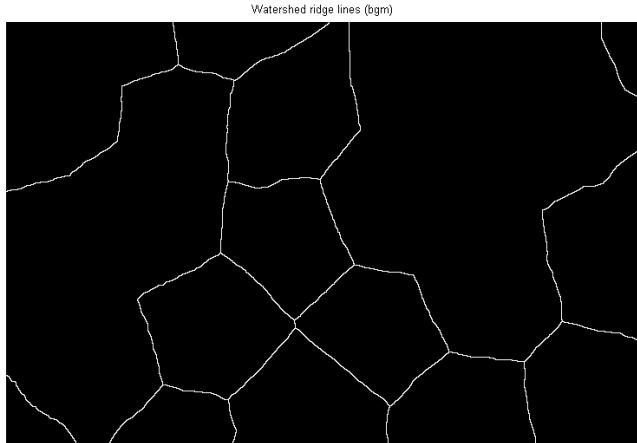


Step 7: cleaning the edges of the marker blobs and then shrinking them a bit

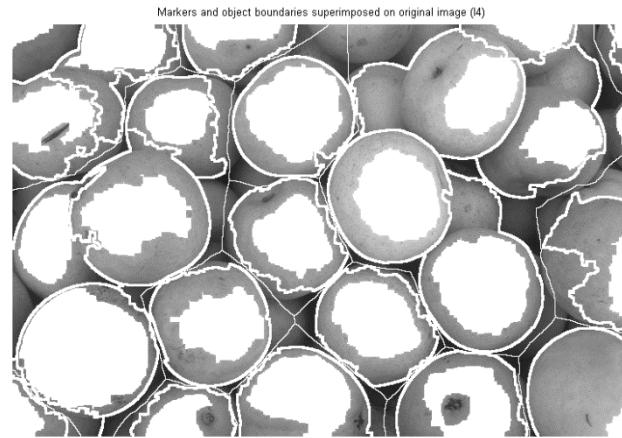


Step 8: Compute Background Markers, Starting with thresholding operation

Watershed Transformation Process

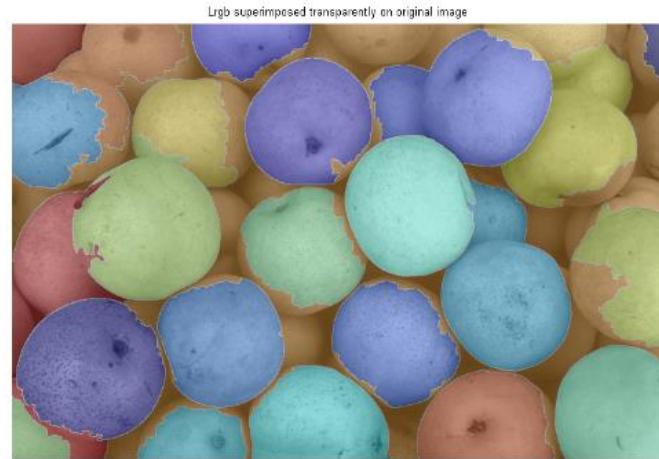
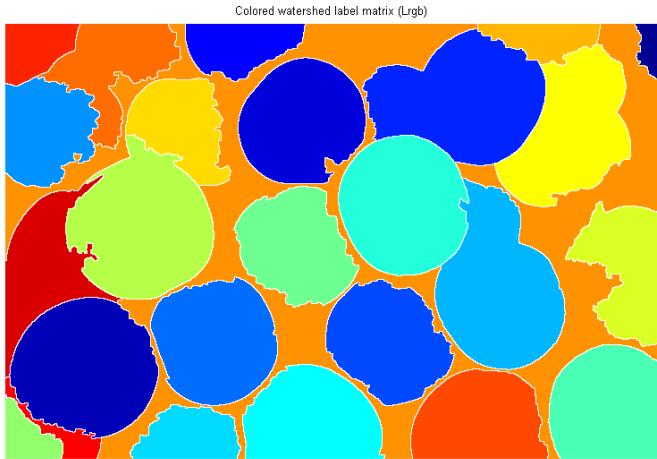


Step 9: Compute Background Markers, using the watershed transform of the distance transform and then looking for the watershed ridge lines of the result



Step 10: Visualize the Result, one of the techniques is to superimpose the foreground markers, background markers, and segmented object boundaries.

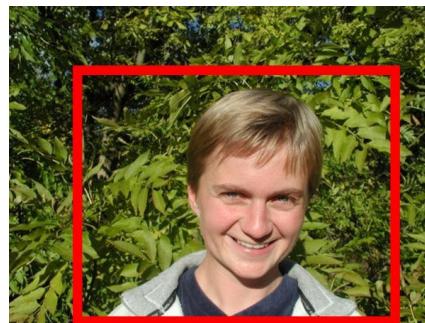
Watershed Transformation Process



*Another useful visualization technique is to display the label matrix as a color image

We can use transparency to superimpose this pseudo-color label matrix on top of the original intensity image.

Interactive Foreground Extraction using GrabCut Algorithm



Code Pack 28

- ~~1. Introduction to OpenCV with Python~~
- ~~2. Core Operations~~
3. Image Processing in OpenCV

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV: FEATURE DETECTION AND DESCRIPTION

Understanding Features

- Features Points or Interest Point are characteristics in an image which differentiates it.
- Like an ID



Feature points are used for

- Image alignment
- 3D reconstruction
- Motion tracking (robots, drones, AR)
- Indexing and database retrieval
- Object recognition
- ...



Features Invariance and covariance

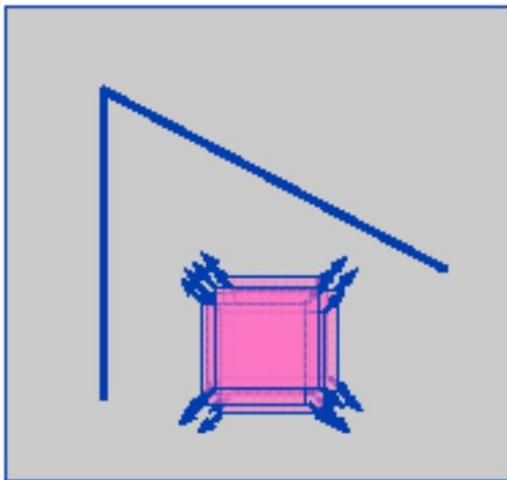
Are locations *invariant* to photometric transformations and *covariant* to geometric transformations?

- **Invariance:** image is transformed and corner locations do not change
- **Covariance:** if we have two transformed versions of the same image, features should be detected in corresponding locations

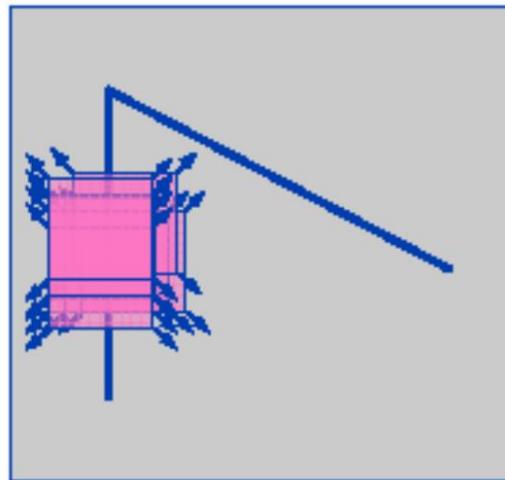


Harris Corner Detection

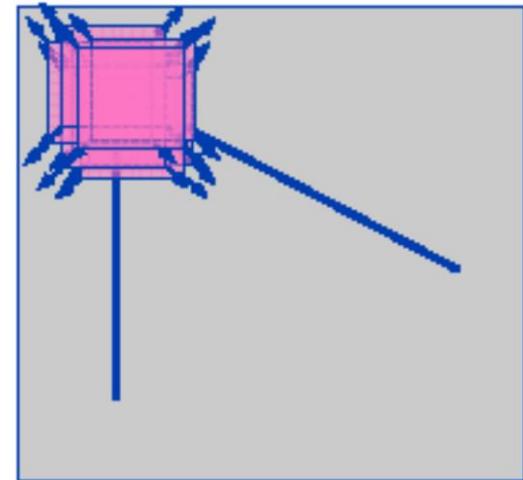
- Harris corner detector gives a mathematical approach for determining which are the corners in a image.
- cv2.cornerHarris()



Flat region:
No change in
all directions



Edge region:
No change along
the edge direction



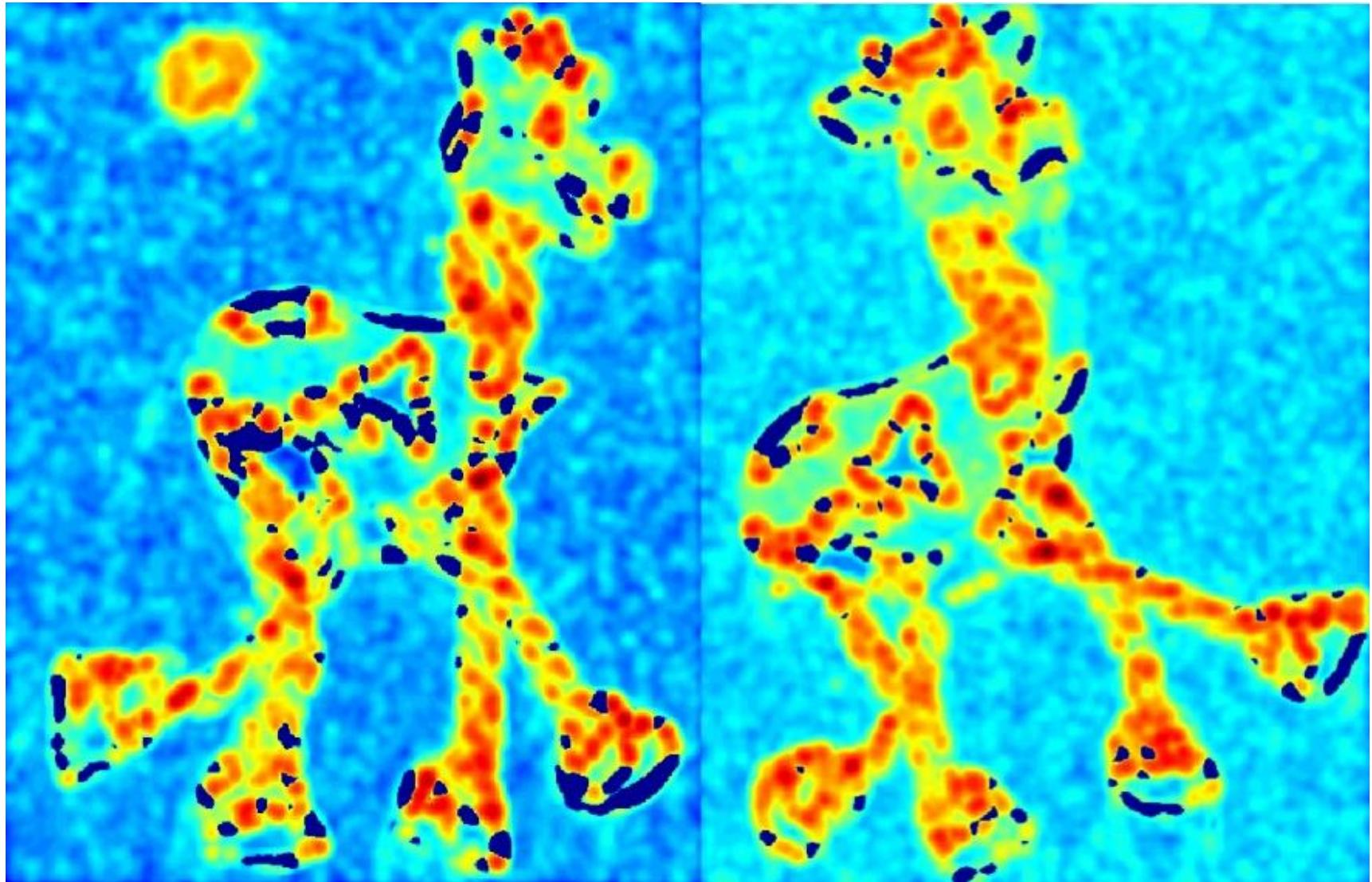
Corner region:
Significant change in
all directions

Harris Corner Detector Algorithm: Begin



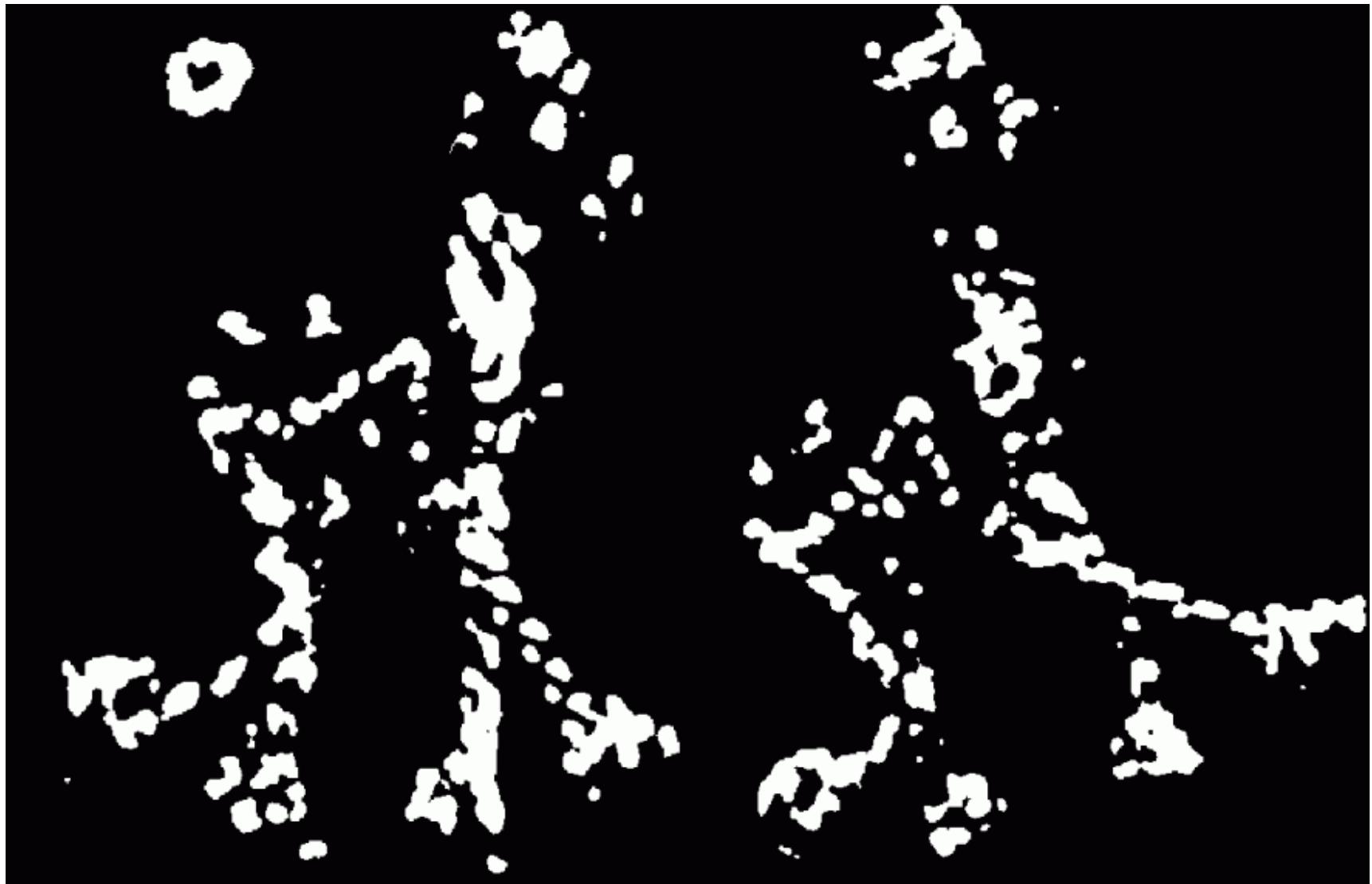
Harris Corner Detector Algorithm: Major Step 1

Compute corner response C



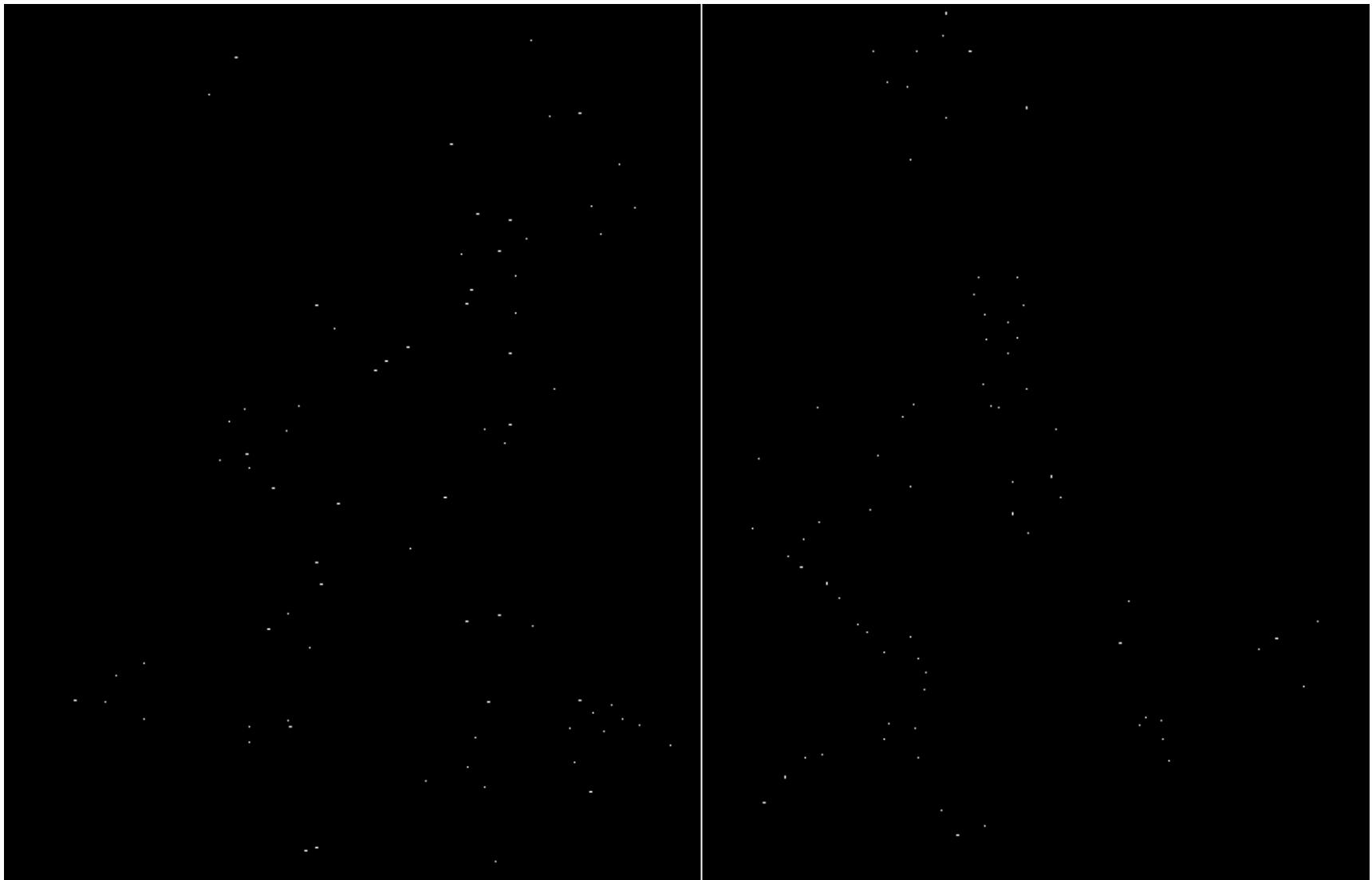
Harris Corner Detector Algorithm: Major Step 2

Find points with large corner response: $C >$ threshold



Harris Corner Detector Algorithm: Major Step 3

Take only the points of local maxima of C



Harris Corner Detector Algorithm: End

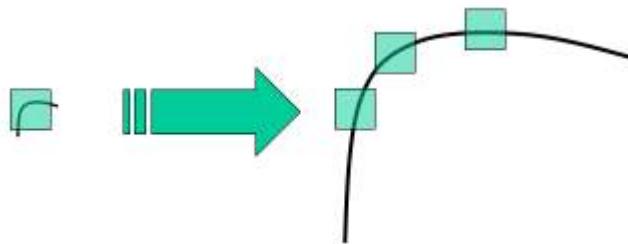


Shi-Tomasi Corner Detector & Good Features to Track

- Another more recent corner detector (1994)
- cv2.goodFeaturesToTrack()

Introduction to SIFT (Scale Invariant Feature Transform)

- Corner Detectors work if image rotate but fail at scaling. X

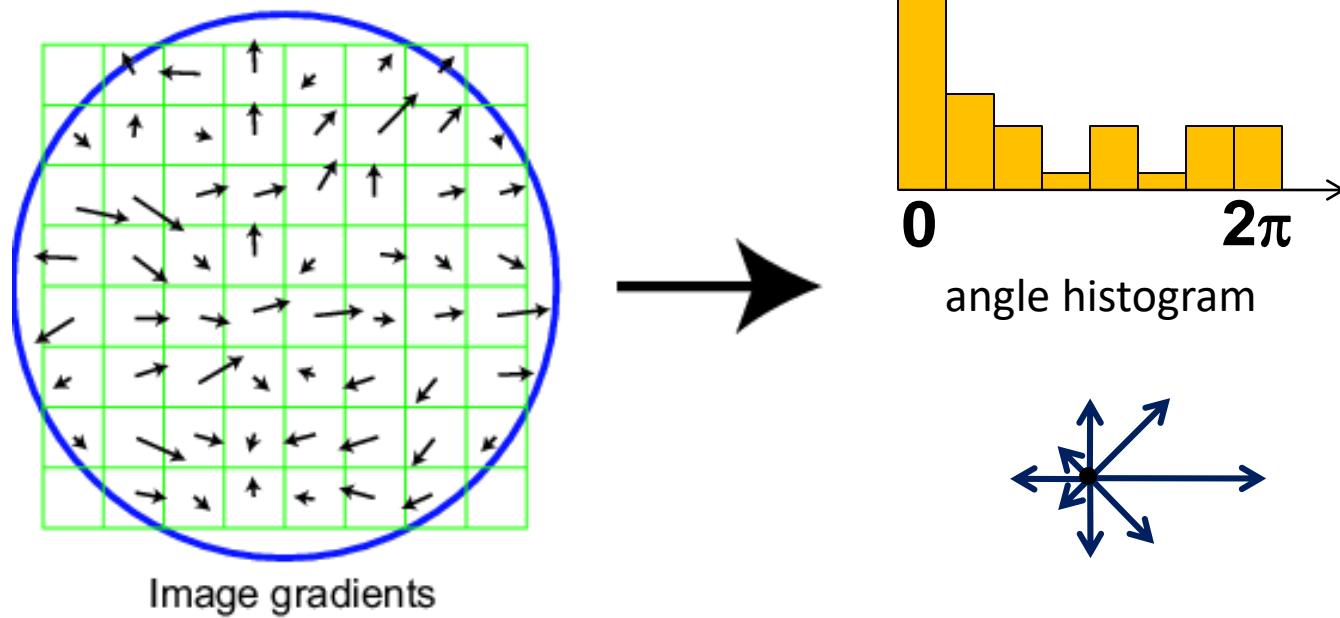


- SIFT (Scale Invariant Feature Transform) solve that. V
- But brings new Concepts:
- **Keypoints** - Features
- **Descriptors** – Encoding of the Features, computed from the keypoint

Scale Invariant Feature Transform

Basic idea:

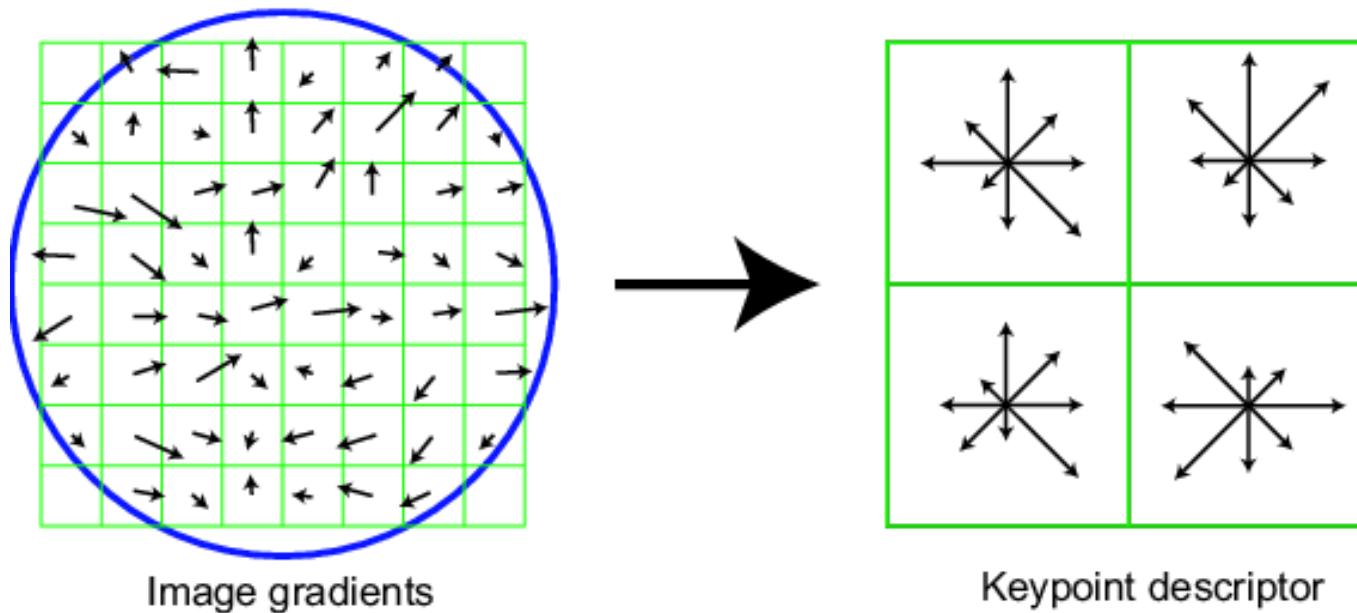
- Take 16x16 square window around detected feature
- Compute gradient orientation for each pixel
- Create histogram over edge orientations weighted by magnitude



SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor

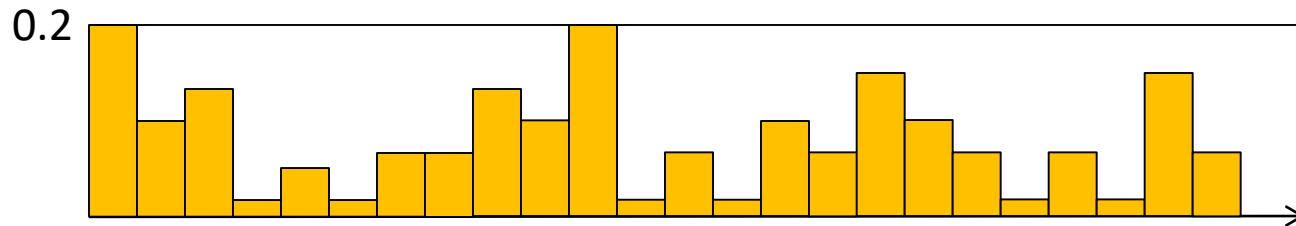


SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor
- Threshold normalize the descriptor:

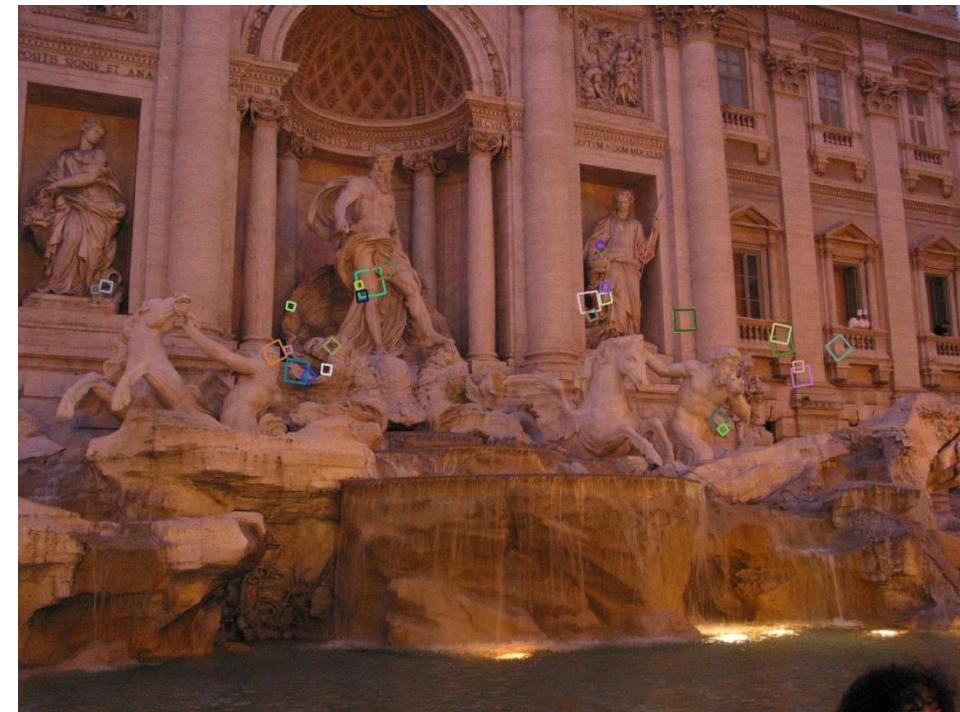
$$\sum_i d_i^2 = 1 \quad \text{such that: } d_i < 0.2$$



Properties of SIFT

Extraordinarily robust matching technique

- Can handle changes in viewpoint
 - Up to about 30 degree out of plane rotation
- Can handle significant changes in illumination
 - Sometimes even day vs. night (below)
- Fast and efficient—can run in real time
- Lots of code available
 - http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT

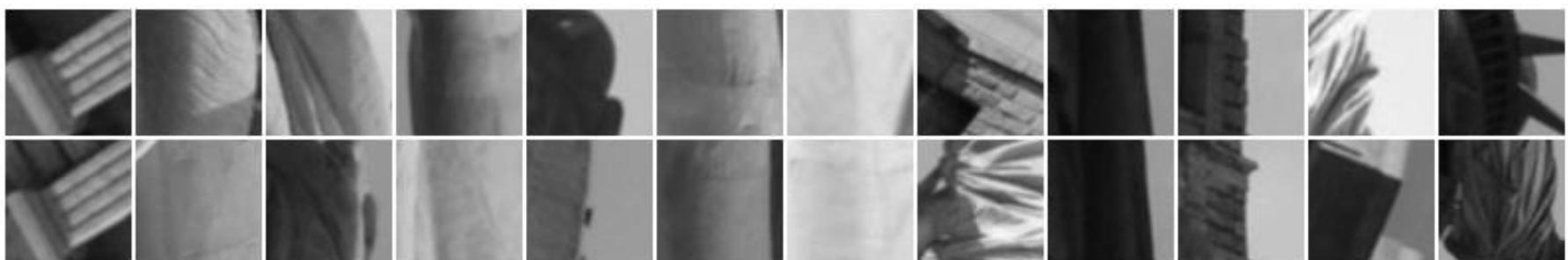
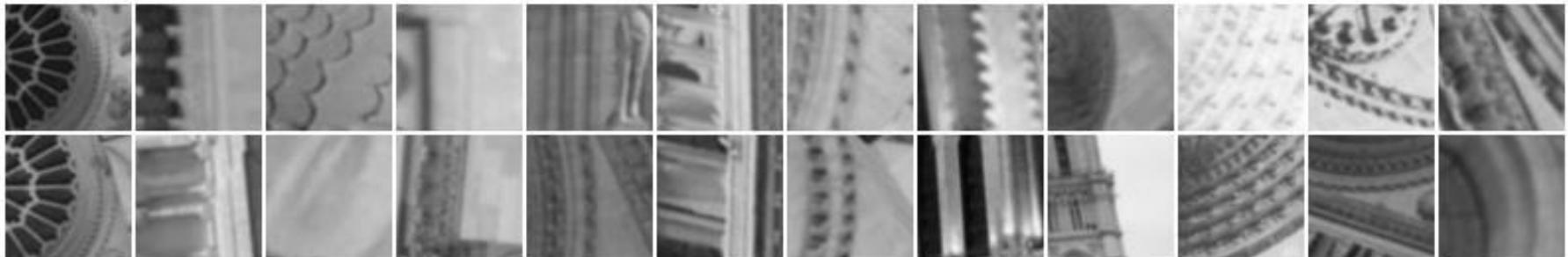


SIFT in OpenCV

- ~~sift = cv2.SIFT()~~
- ~~kp = sift.detect(...)~~
- sift = cv2.xfeatures2d.SIFT_create()
- kp = sift.detect(...)

When does SIFT fail?

Patches SIFT thought were the same but aren't:



Introduction to SURF (Speeded-Up Robust Features)

- For computational efficiency only compute gradient histogram with 4 bins.

```
surf = cv2.SURF(...)  
cv2.xfeatures2d.SURF_create()  
kp, des = surf.detectAndCompute(...)
```

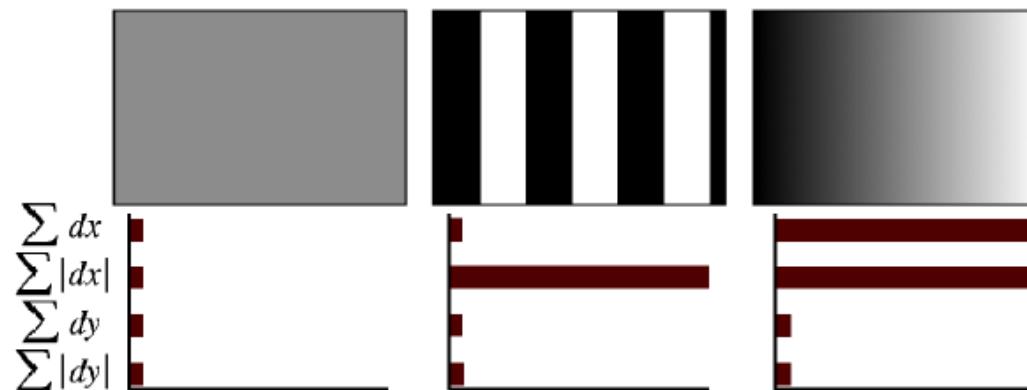


Fig. 3. The descriptor entries of a sub-region represent the nature of the underlying intensity pattern. Left: In case of a homogeneous region, all values are relatively low. Middle: In presence of frequencies in x direction, the value of $\sum |d_x|$ is high, but all others remain low. If the intensity is gradually increasing in x direction, both values $\sum d_x$ and $\sum |d_x|$ are high.

FAST Algorithm for Corner Detection

- Like SURF tries to be a quicker SIFT.
- FAST (Features from Accelerated Segment Test) tries to be a quicker Corner Detection

```
fast = cv2.FastFeatureDetector_create()  
kp = fast.detect(...)
```

BRIEF (Binary Robust Independent Elementary Features)

- SURF → quicker SIFT.
 - FAST → quicker Harris
-
- BRIEF is a easier descriptor to compare and store.
 - Binary-string descriptors

```
brief =  
cv.xfeatures2d.BriefDescriptorExtractor_create()  
  
brief.compute(...)
```

ORB (Oriented FAST and Rotated BRIEF)

- First that came from “OpenCV Labs”
- An efficient alternative to SIFT or SURF in 2011.
- SIFT and SURF are patented and its supposed to pay them for its use.
- ORB is not !!!

```
orb = cv2.ORB()  
kp = orb.detect(...)  
kp, des = orb.compute(...)
```

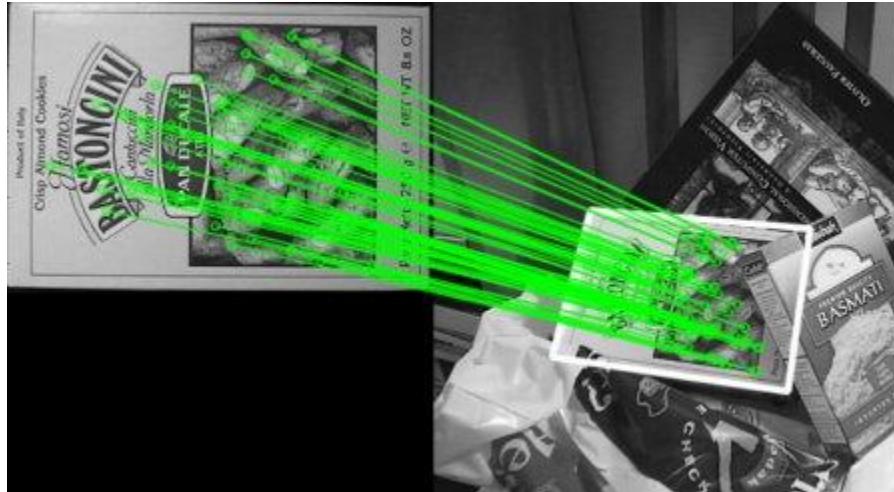
Feature Matching

- Identify if it's the same object (in less than a error)
- Brute-Force Matching with SIFT Descriptors and Ratio Test
`cv2.BFMatcher()`
- FLANN (Fast Library for Approximate Nearest Neighbors)
`cv2.FlannBasedMatcher`



Feature Matching + Homography to find Objects

- Join Techniques and other OpenCV tools to work with complex images



Code Pack 28

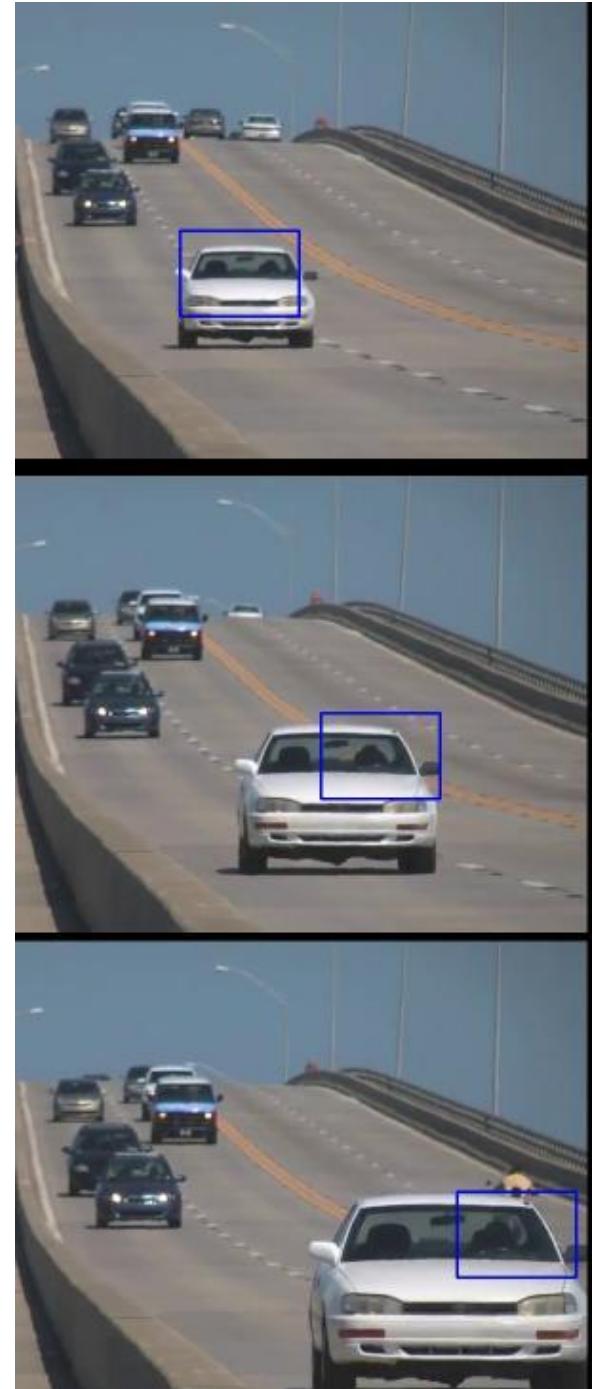
- ~~1. Introduction to OpenCV with Python~~
- ~~2. Core Operations~~
- ~~3. Image Processing in OpenCV~~
4. Feature Detection and Description

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV: VIDEO ANALYSIS

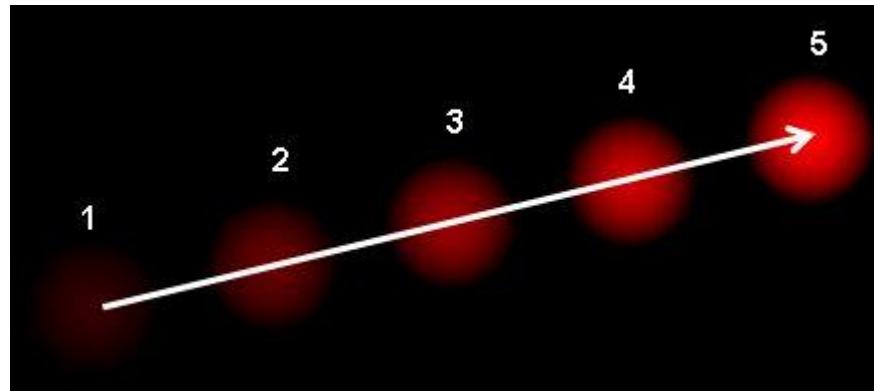
Meanshift and Camshift

- Meanshift and Camshift algorithms are used to find and track objects in videos.
- `cv2.meanShift()` – only used if the object has the same size
- `cv2.CamShif()` – used more if the object resizes in the image



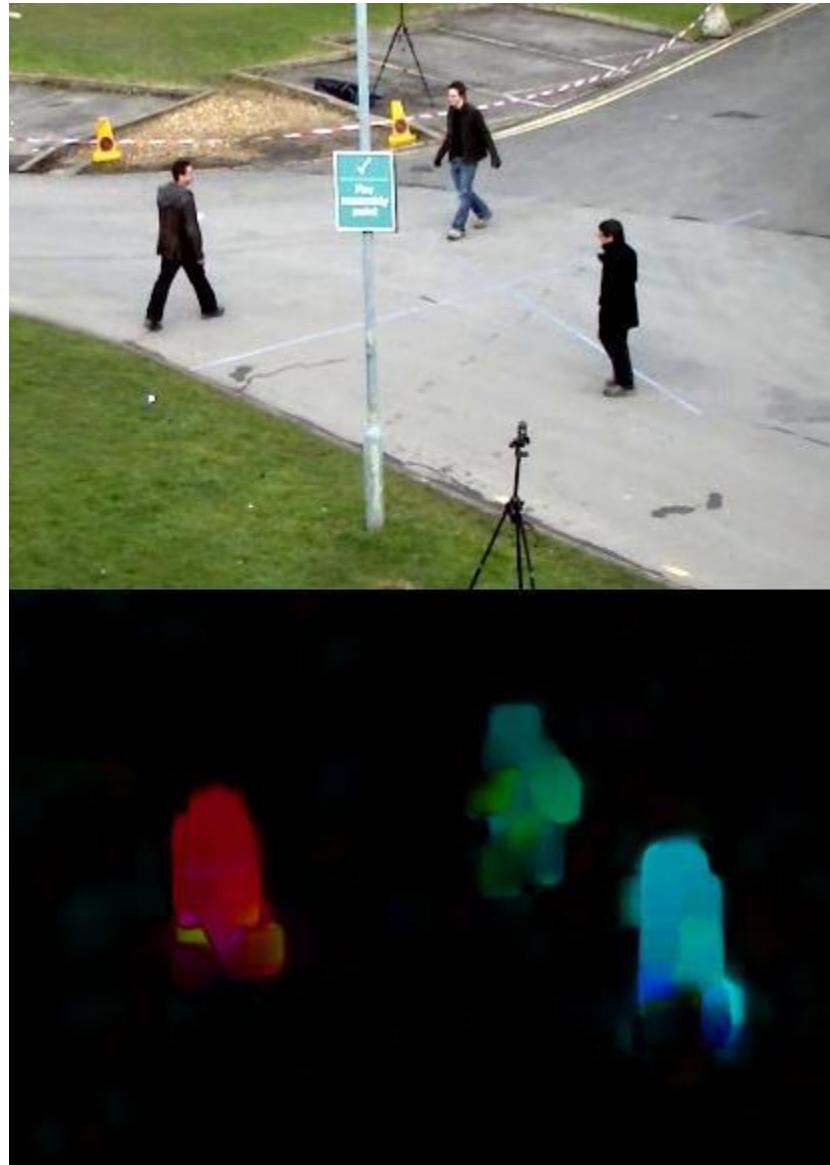
Optical Flow

- Pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera.



- Optical flow has many applications in areas like :
 - Structure from Motion
 - Video Compression
 - Video Stabilization
-
- `cv2.calcOpticalFlowPyrLK()`
 - (Lucas-Kanade method)

Background Subtraction



Code Pack 28

- ~~1. Introduction to OpenCV with Python~~
- ~~2. Core Operations~~
- ~~3. Image Processing in OpenCV~~
- ~~4. Feature Detection and Description~~
5. Video Analysis

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV: CAMERA CALIBRATION AND 3D RECONSTRUCTION

Camera Calibration

- Cheap cameras introduces a lot of distortion to images. Two major distortions are radial distortion and tangential distortion

It his necessary to remap the reading Matrix

- cv2.findChessboardCorners – Dummy Test
- cv2.calibrateCamera()
- cv2.getOptimalNewCameraMatrix()

Pinhole Camera Model: Camera Matrix

- **Camera matrix:** a 3×4 matrix that transforms from 3D scene points (x, y, z) to 2D screen points $(x'/w', y'/w')$:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- How many free parameters in the camera matrix?

Pinhole Camera Model

- Can rewrite the camera matrix in terms of **intrinsic** and **extrinsic** parameters.

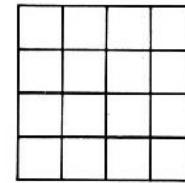
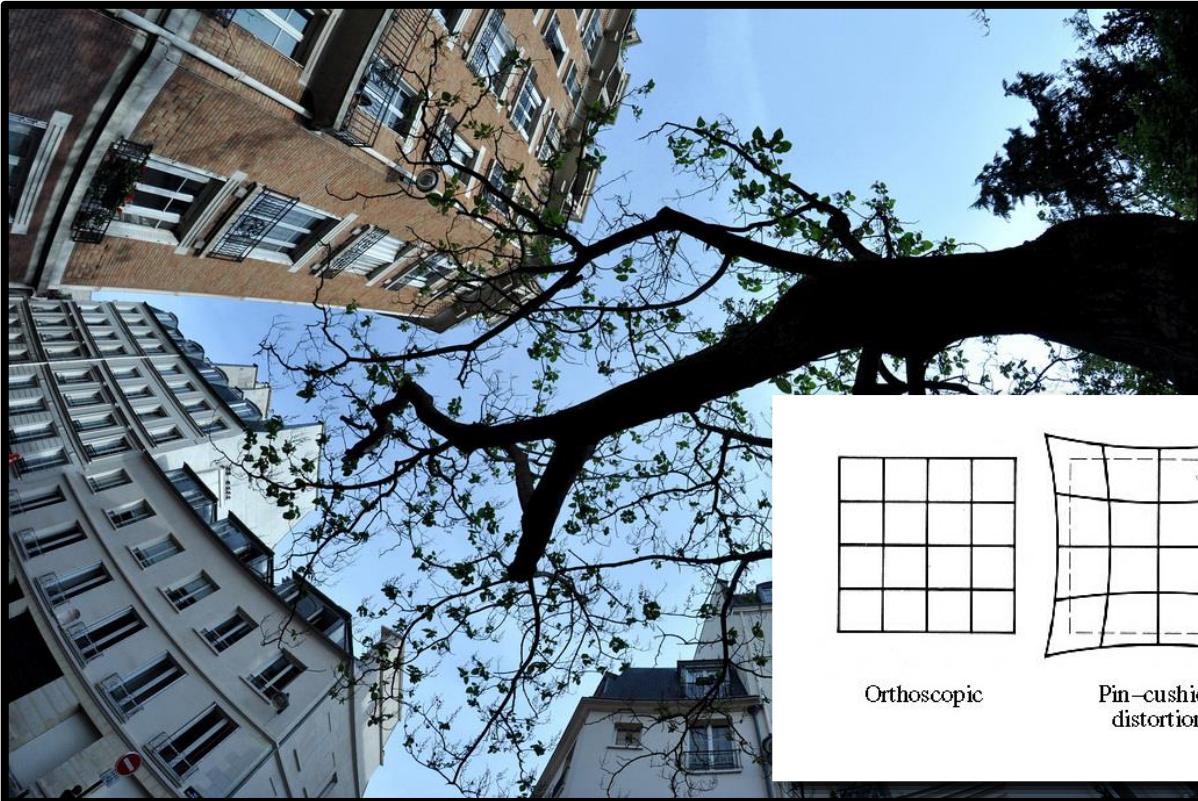
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \mathbf{K} [\mathbf{R} \quad \mathbf{t}] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\mathbf{K} = \begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_x & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

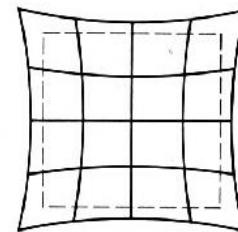
Intrinsic matrix

- How many intrinsic parameters?
- How many extrinsic parameters?
- How many parameters in total?

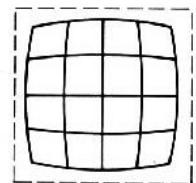
Real Camera Model: Radial Distortion



Orthoscopic



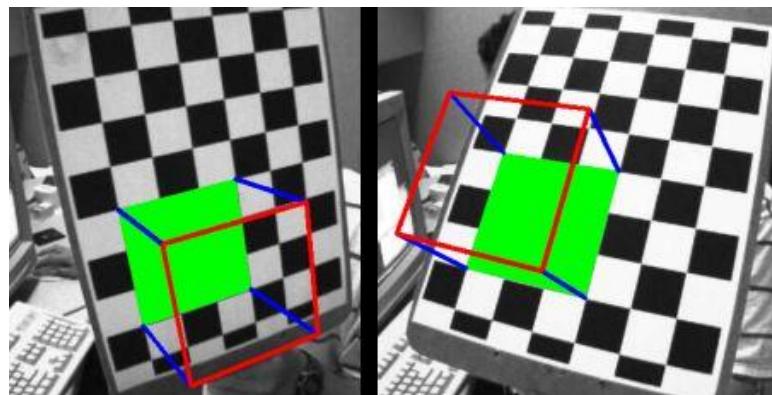
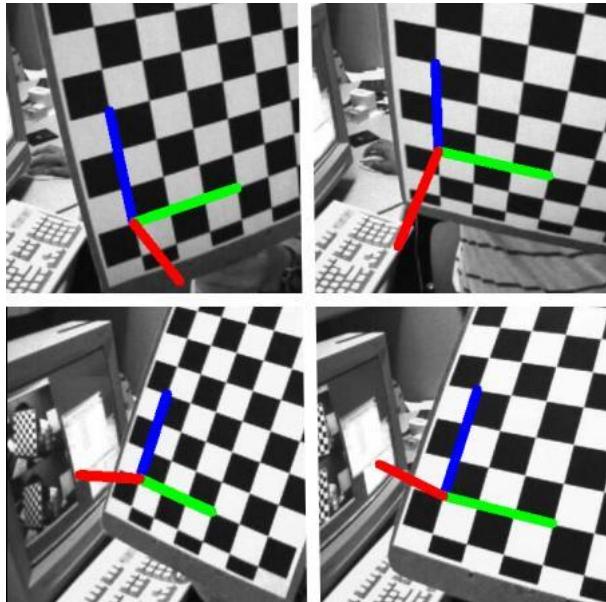
Pin-cushion
distortion



Barrel
distortion

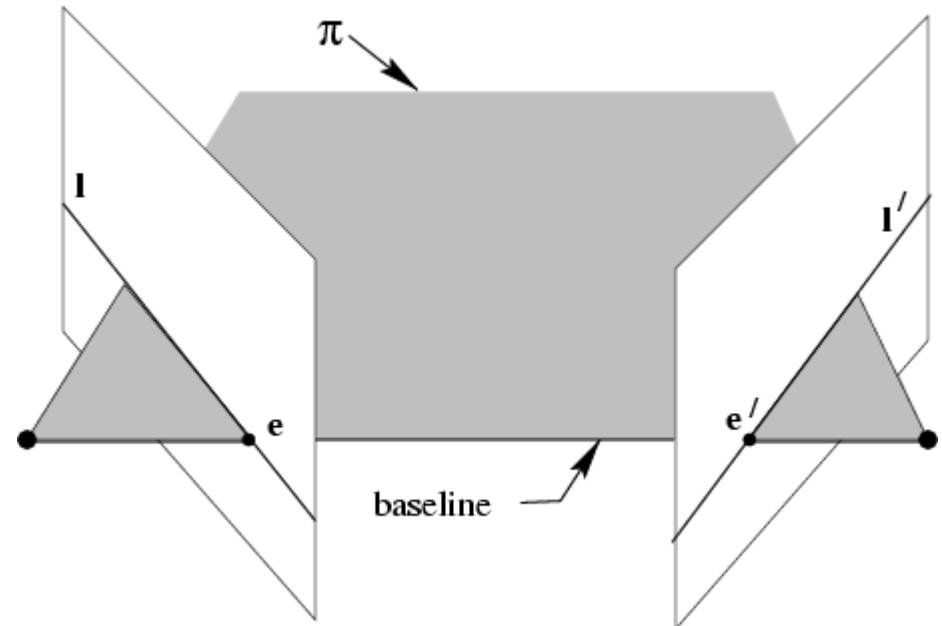
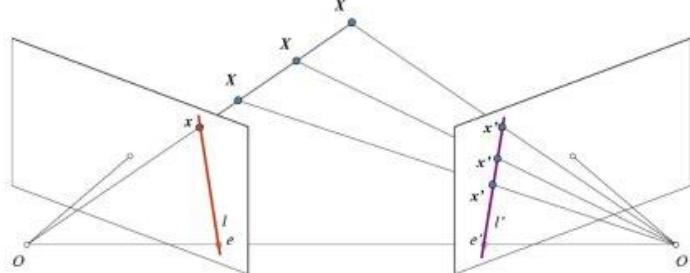
Pose Estimation

- Use the camera matrix and the calib3d module to create some 3D effects in image

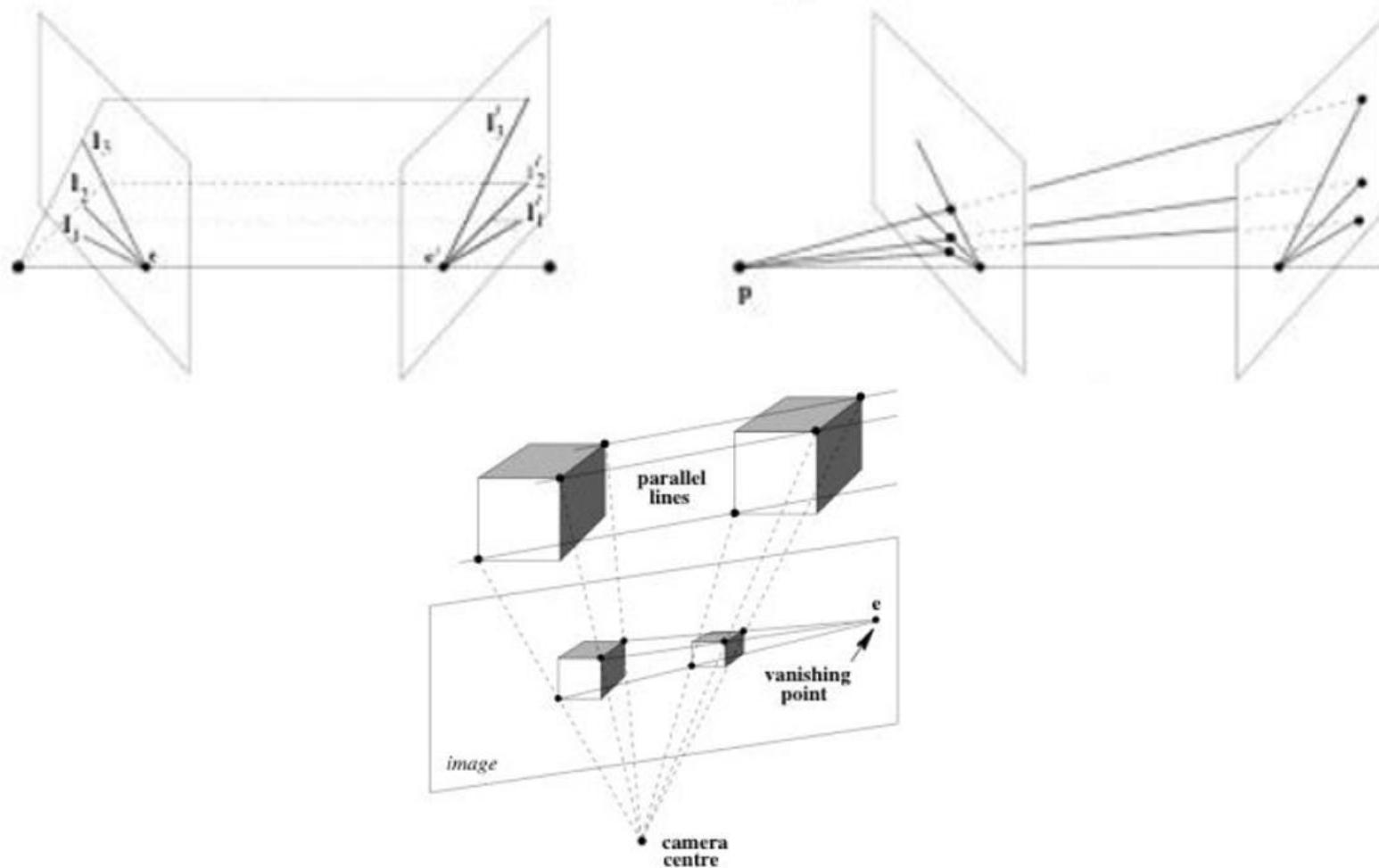


Epipolar Geometry

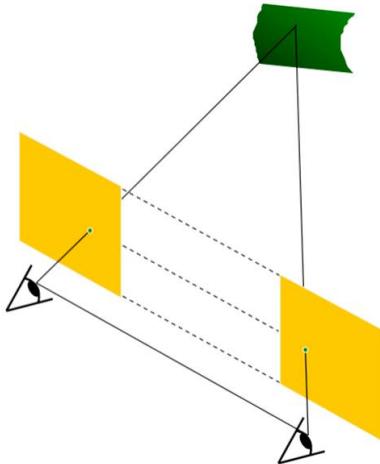
- Baseline: connects two camera centers
- Epipole: point of intersection of baseline with image plane
- Epipole: image in one view of the camera center of the other view.



Epipolar Geometry – More examples

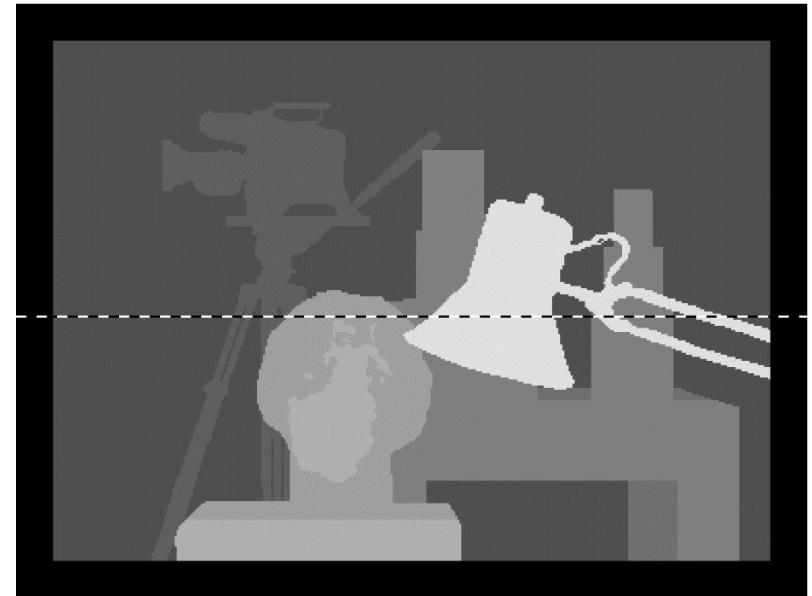


Stereo Images

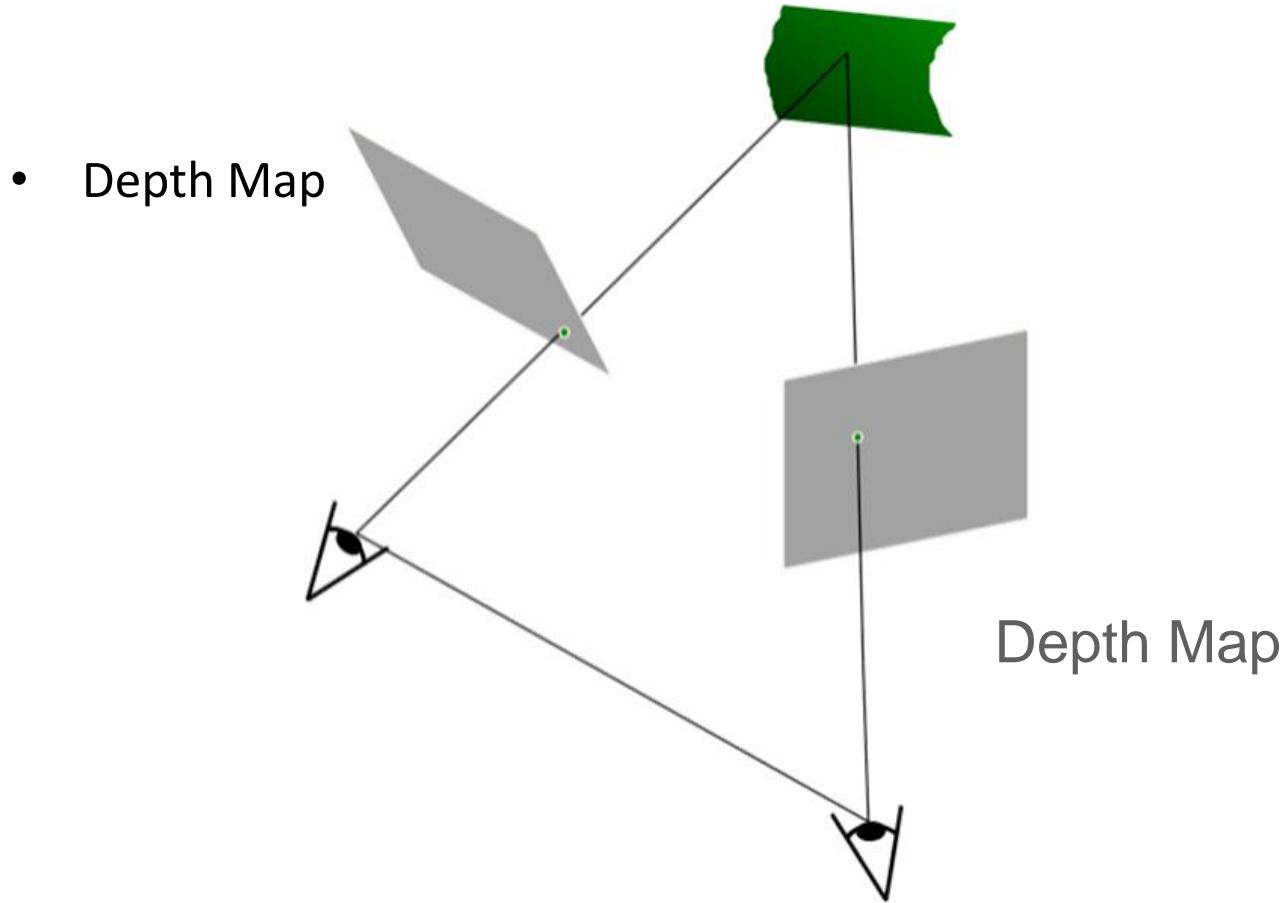


Depth Map from Stereo Images

- Below image contains the original image (left) and its disparity map (right). As you can see, result is contaminated with high degree of noise.
- By adjusting the values of numDisparities and blockSize, you can get better results.



Stereo Rectification



Code Pack 28

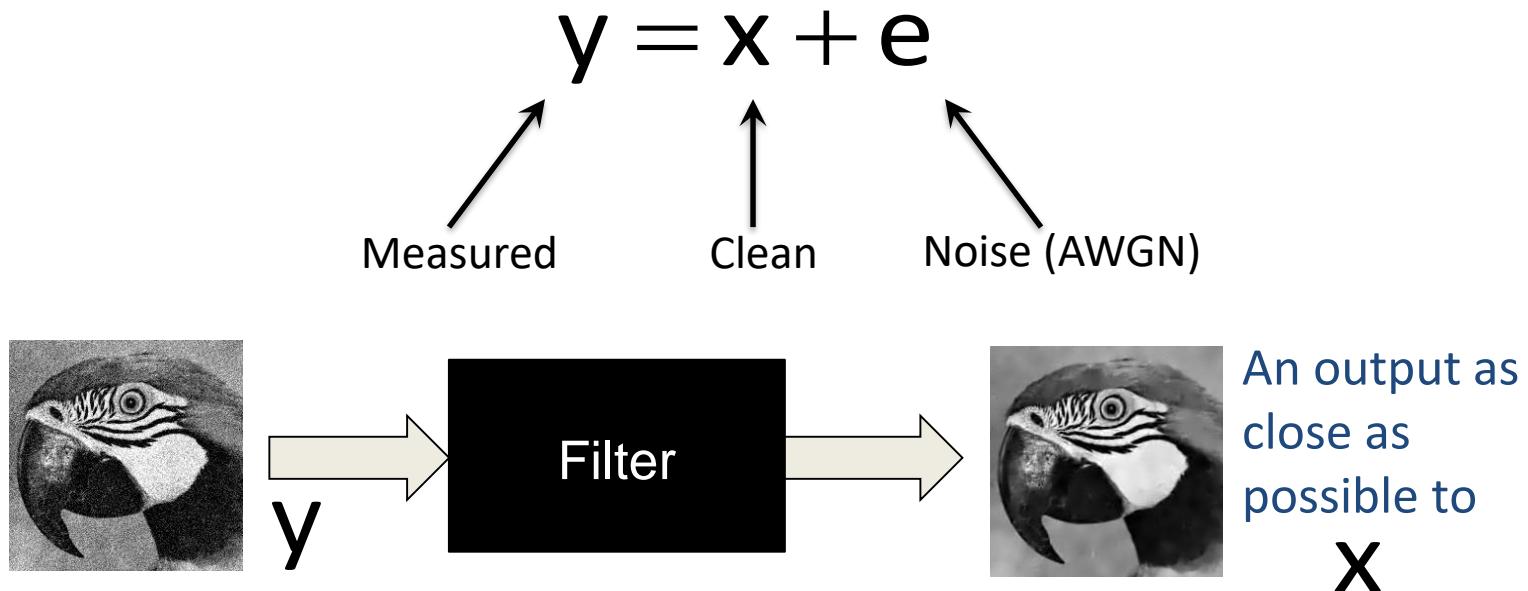
- ~~1. Introduction to OpenCV with Python~~
- ~~2. Core Operations~~
- ~~3. Image Processing in OpenCV~~
- ~~4. Feature Detection and Description~~
- ~~5. Video Analysis~~
- ~~6. Camera Calibration and 3D Reconstruction~~

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV: COMPUTATIONAL PHOTOGRAPHY

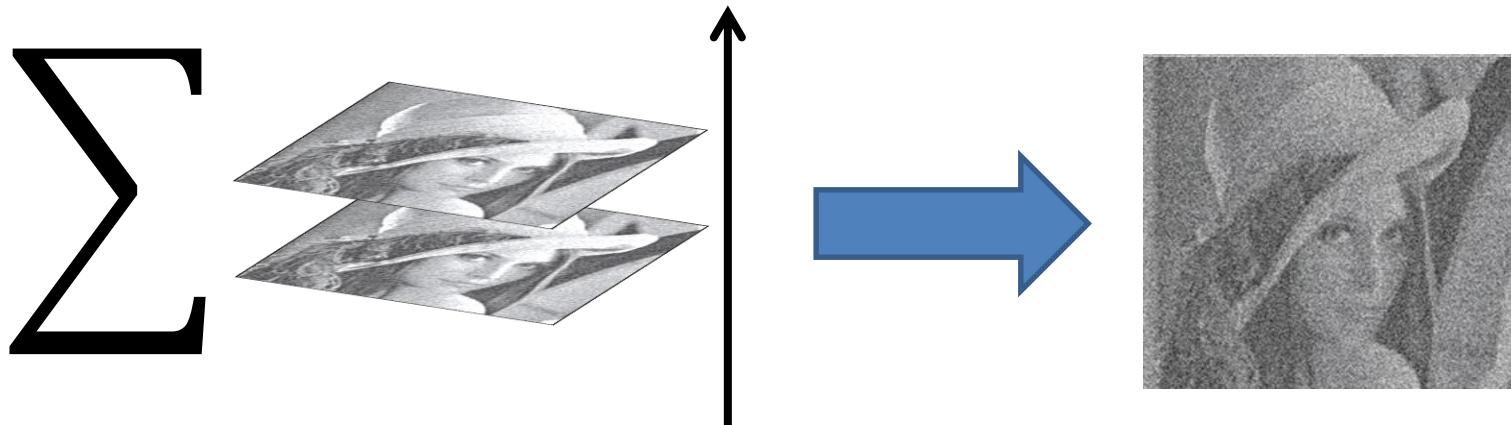
Image Denoising

This is the “simplest” inverse problem



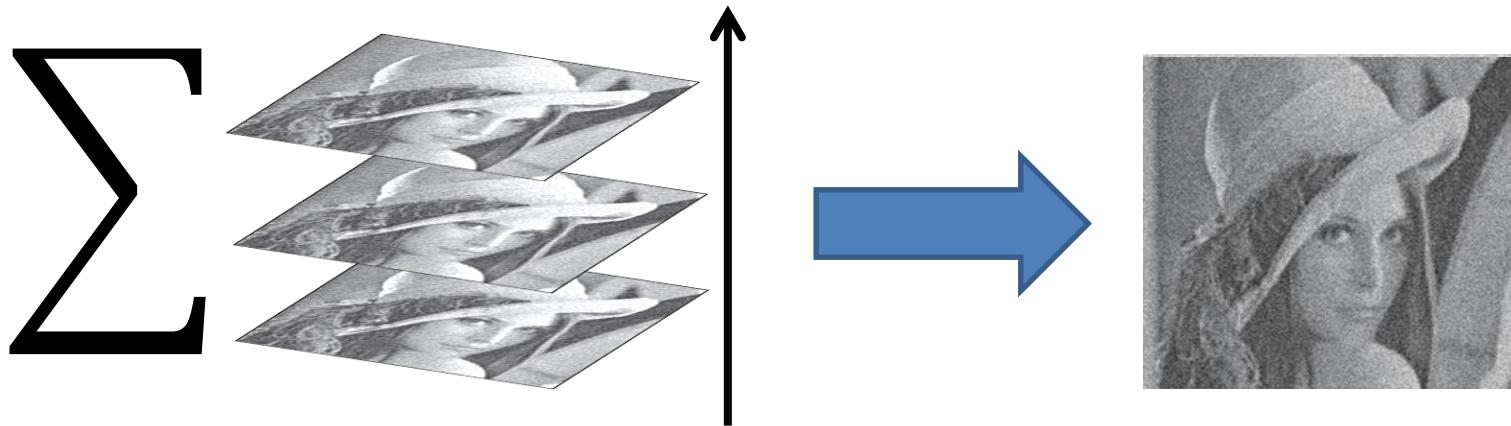
Temporal Denoising

Average multiple images over time



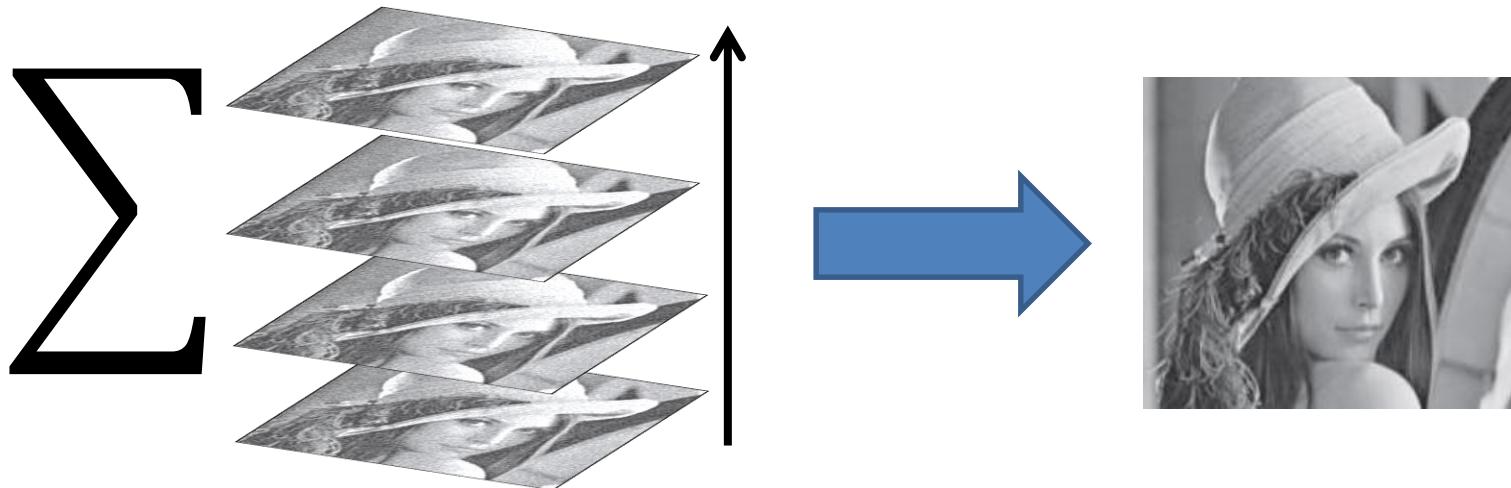
Temporal Denoising

Average multiple images over time

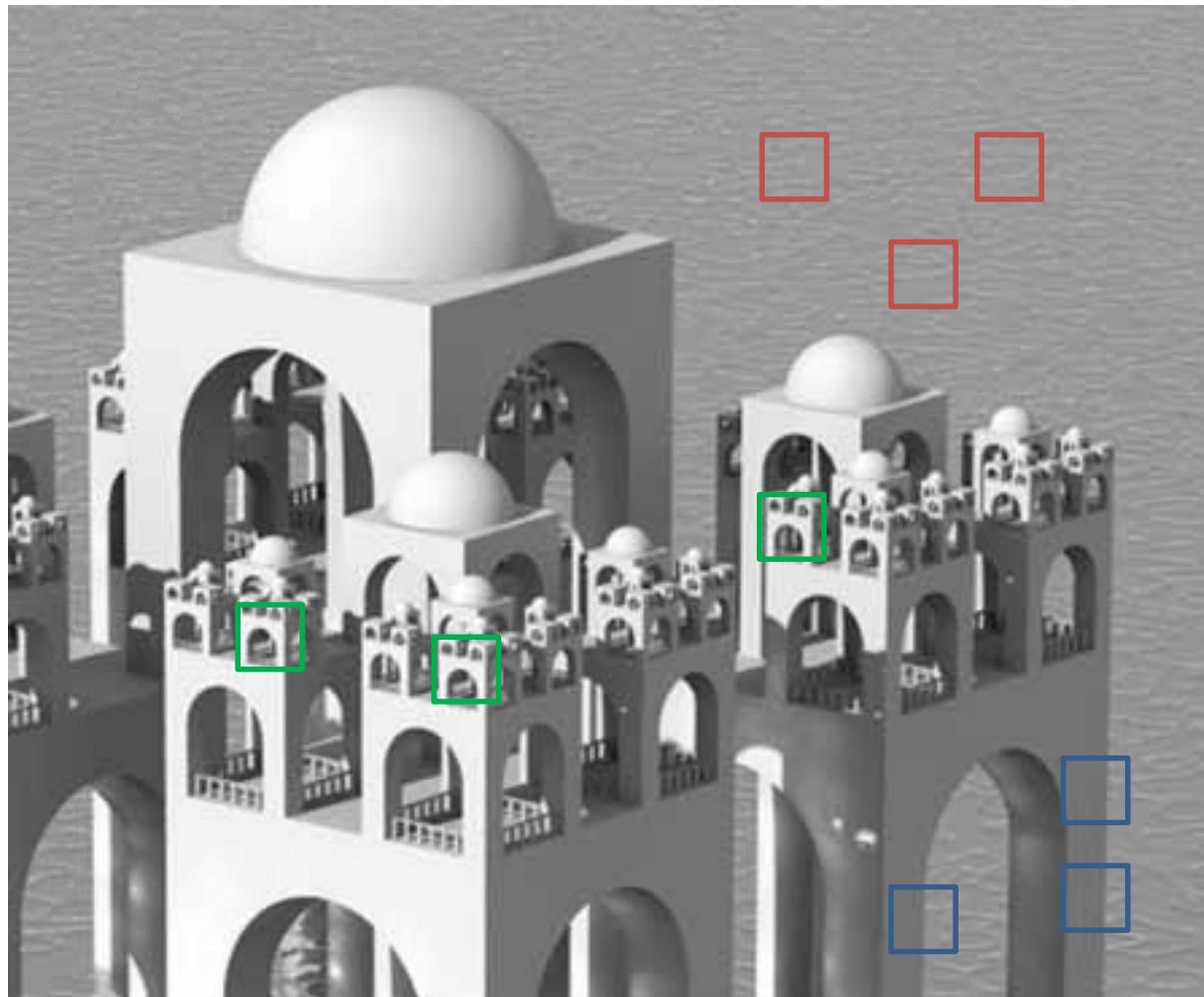


Temporal Denoising

Average multiple images over time



Redundancy in natural images



Stronger Techniques

- Classical Denoising
 - Spatial Methods
 - Transform Methods
- State-of-the-art Methods
 - GSM – Gaussian Scale Mixture
 - NLM – Non-local means
 - BM3D – Block Matching 3D collaborative filtering

Image Denoising in OpenCV

OpenCV provides 4 variations of this technique.

1. `cv2.fastNlMeansDenoising()` - works with a single grayscale images
2. `cv2.fastNlMeansDenoisingColored()` - works with a color image.
3. `cv2.fastNlMeansDenoisingMulti()` - works with image sequence captured in short period of time (grayscale images)
4. `cv2.fastNlMeansDenoisingColoredMulti()` - same as above, but for color images.

Common arguments are:

- `h` : parameter deciding filter strength. Higher `h` value removes noise better, but removes details of image also. (**10 is ok**)
- `hForColorComponents` : same as `h`, but for color images only. (**normally same as h**)
- `templateWindowSize` : should be odd. (**recommended 7**)
- `searchWindowSize` : should be odd. (**recommended 21**)

Image Inpainting

- Inpainting is the process of reconstructing lost or deteriorated parts of images and videos.



Other example Image Inpainting

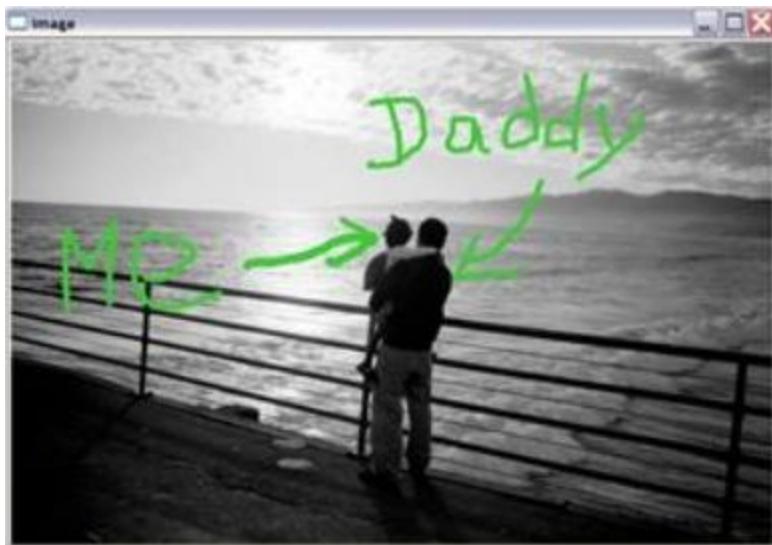


Image Inpainting in OpenCV

- Several algorithms were designed for this purpose and OpenCV provides 2 of them.
- Both can be accessed by the same function:

```
cv2.inpaint()
```

They are:

- cv2.INPAINT_TELEA – based on the paper “An Image Inpainting Technique Based on the Fast Marching Method” by Alexandru Telea in 2004
- cv2.INPAINT_NS.– based on the paper “Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting” by Bertalmio, Marcelo, Andrea L. Bertozzi, and Guillermo Sapiro in 2001

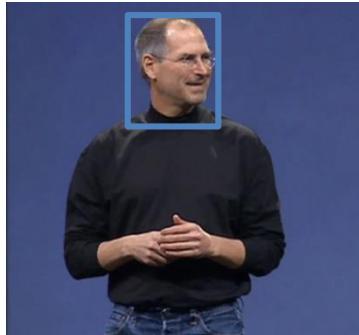
Code Pack 28

1. ~~Introduction to OpenCV with Python~~
2. ~~Core Operations~~
3. ~~Image Processing in OpenCV~~
4. ~~Feature Detection and Description~~
5. ~~Video Analysis~~
6. ~~Camera Calibration and 3D Reconstruction~~
7. Computational Photography

Coding Bootcamp Code in Python

INTRODUCTION TO OPENCV: OBJECT DETECTION

Face detection



Face Detection ≠ Face recognition

- Who is this? →
- ← Where is the face?



Face Detection - detect that is a human face

- Haar Classifier
- LBP Classifier

Alfréd Haar

Face Recognition - detect that is **that** human

- Fisher
- Eigen
- LBPHF

Object Recognition Classifier

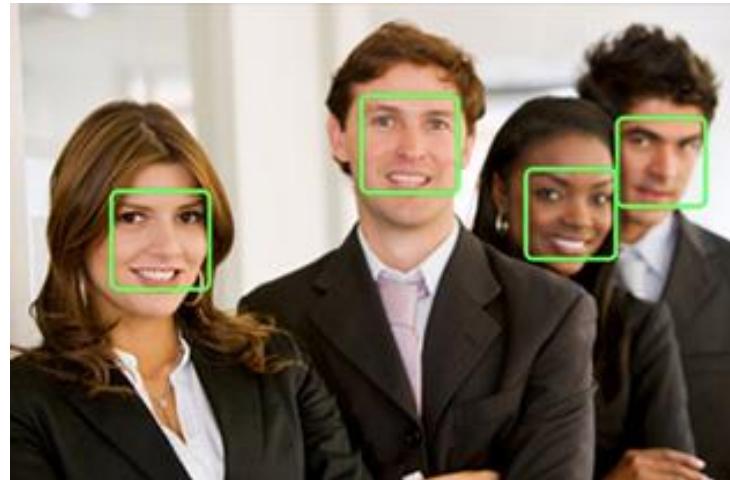
- A program that decides whether an image is a positive image (face image) or negative image (non-face image).
- A classifier is trained on hundreds of thousands of face and non-face images to learn how to classify a new image correctly.

Haar Cascade

- A "cascade" is a series of "Haar-like features" that are combined to form a classifier
- Haar Cascade = Classifier

Face Detection

- The problem of face detection is:
Given an image, say if it contains faces or not.
- The idea of face detection in computer vision is to let the computer learn to detect faces in images, just as a human can do.



Difficulties - Changing lightening

- Affects color, facial feature



Difficulties - Skin Tone

- Large variety of skin tones.



Bisque	Ivory	Light Beige	Satin
Amber	Warm Sienna	Warm Silk	Natural
Radiant	Honey Bronze	Suntan	Golden Glow
Caramel	Latte	Maple	Butternut
Mink			

Difficulties - Facial Expressions

- Affects shape of face and its features



Difficulties – Scaling and Angles



Difficulties - Obstructions

- Obstruction of facial features

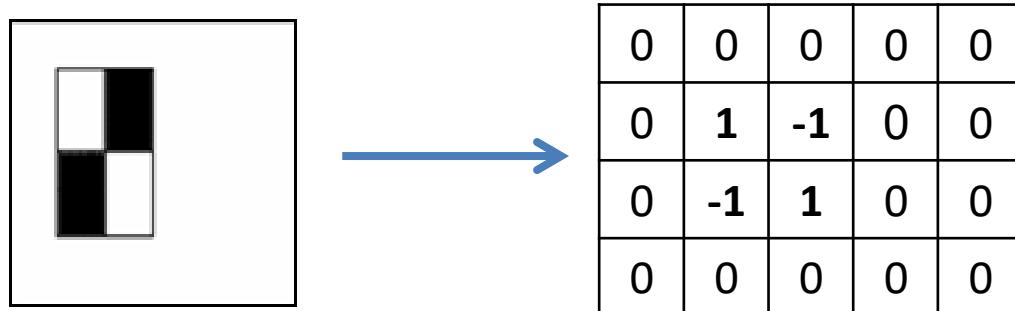


Paul Viola and Michael Jones - 2001

- The Viola–Jones object detection algorithm is the first object detection framework to provide competitive object detection rates in real-time proposed.
- First with speed!
- Although it can be trained to detect a variety of object classes, it was motivated primarily by the problem of face detection.
- The algorithm has 4 stages:
 1. Haar Feature Selection
 2. Creating an Integral Image
 3. Adaboost Training
 4. Final Classifier = Cascading Classifiers

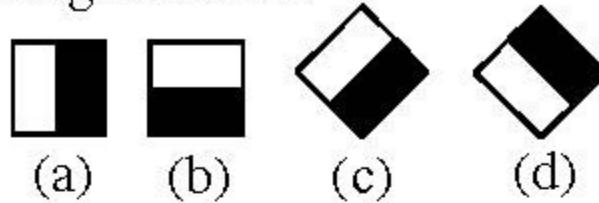
Rectangle Features (or Haar-like Features)

- It look for features inside 24x24 pixels window
- Each feature contains black & white rectangles, the feature value is defined by:
 - $\sum (\text{pixels in white area}) - \sum (\text{pixels in black area})$
 - Other explanation: correlation of the image with a mask with 1 in pixels of white areas, and -1 in pixels of black areas

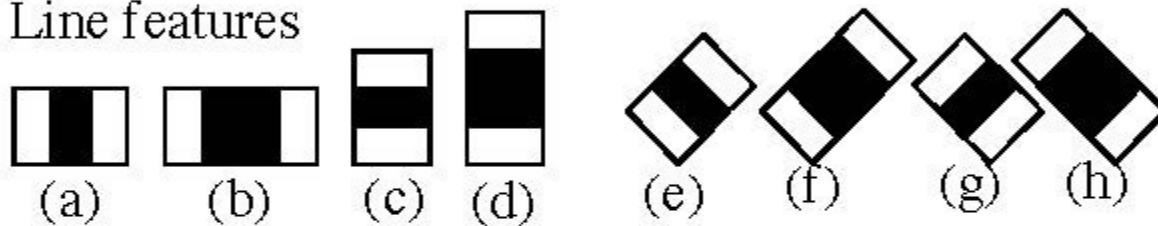


Haar-Like Features

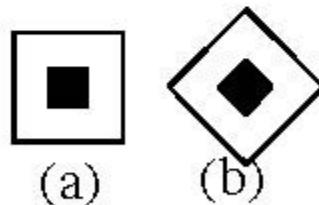
1. Edge features



2. Line features



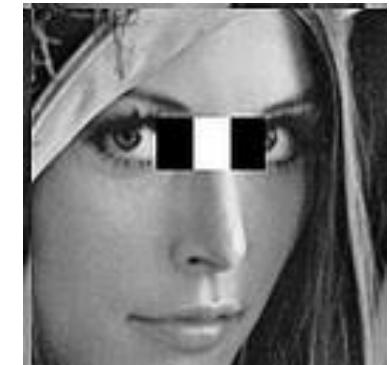
3. Center-surround features



4. Special diagonal line feature used in [3,4,5]

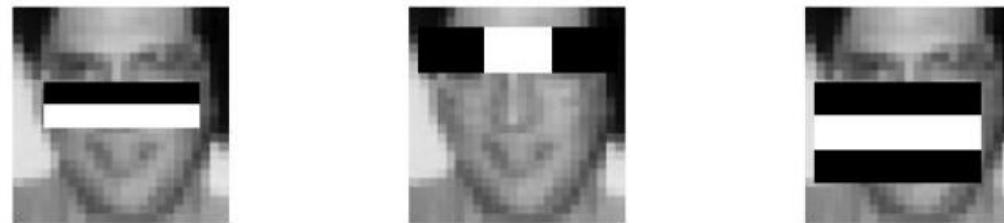


Examples of how the Haar-like Features correspond to common facial features.

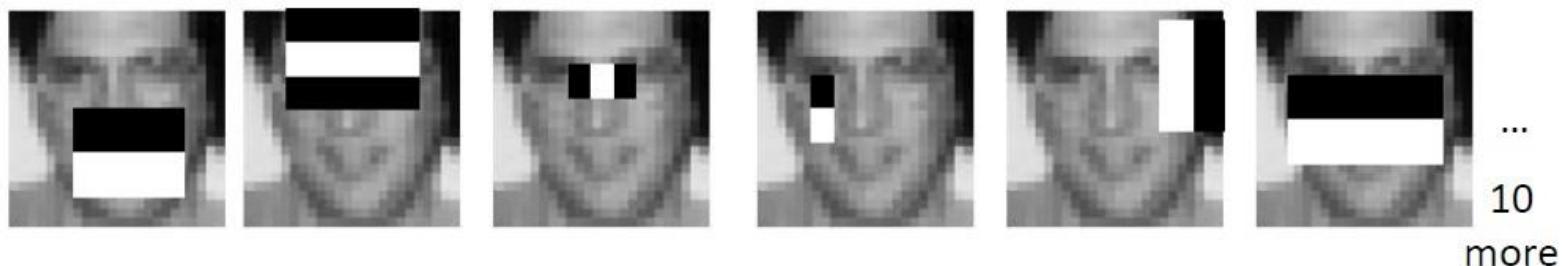


1. Haar Feature Selection

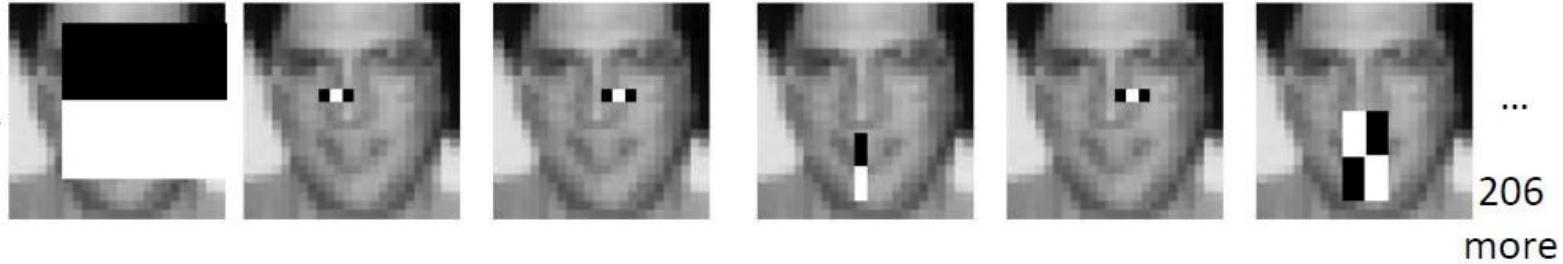
Stage 0



Stage 1

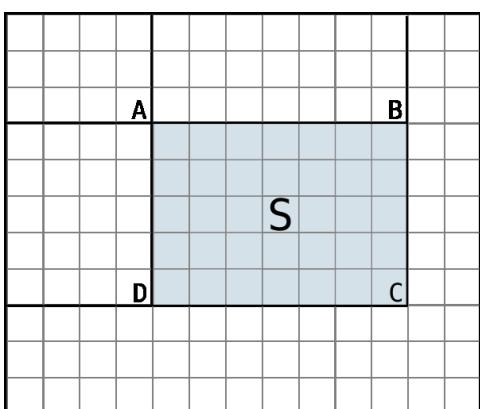


Stage 21



2. Creating an Integral Image

- A "summed area table" is created in a single pass across the image.
- The sum of any region in the image can be computed by a single formula.



Original				
5	2	3	4	1
1	5	4	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

$$5 + 2 + 3 + 1 + 5 + 4 = 20$$

Integral				
5	7	10	14	15
6	13	20	26	30
8	17	25	34	42
11	25	39	52	65
15	30	47	62	81

$$\begin{aligned} &5 + 4 + 2 + \\ &2 + 1 + 3 = 17 \end{aligned}$$

Original				
5	2	3	4	1
1	5	4	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

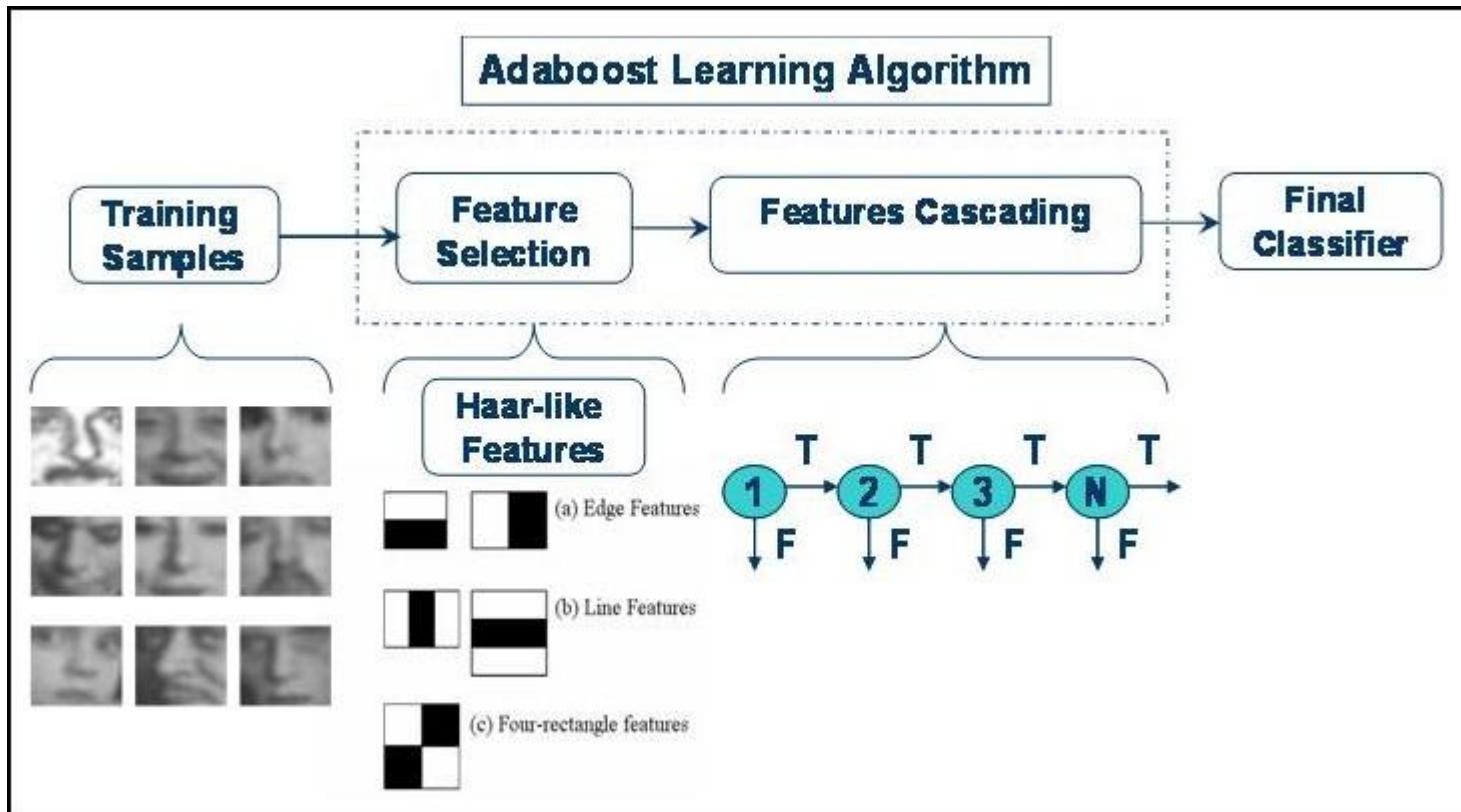
Integral				
5	7	10	14	15
6	13	20	26	30
8	17	25	34	42
11	25	39	52	65
15	30	47	62	81

$$34 - 14 - 8 + 5 = 17$$

3. AdaBoost (Adaptive Boosting)

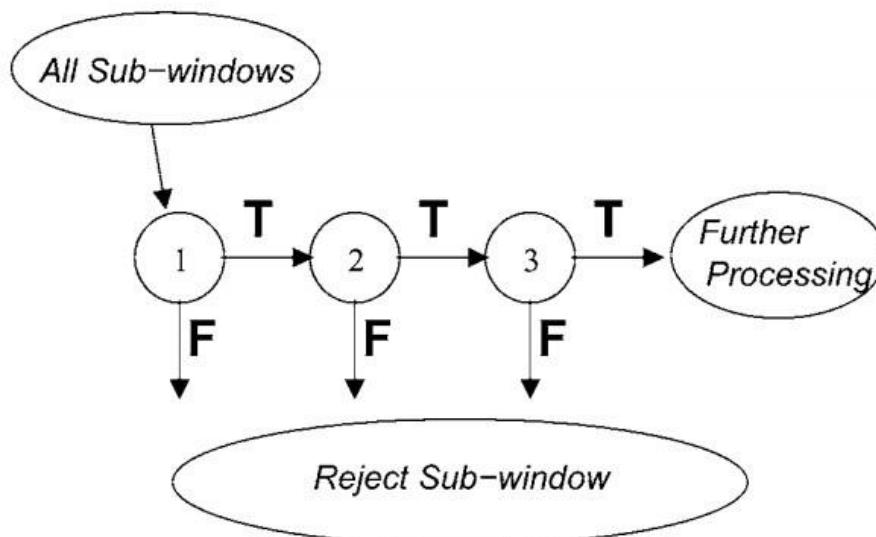
Training

- Adaboost tries out multiple weak classifiers over several rounds, selecting the best weak classifier in each round and combining the best weak classifiers to create a strong classifier



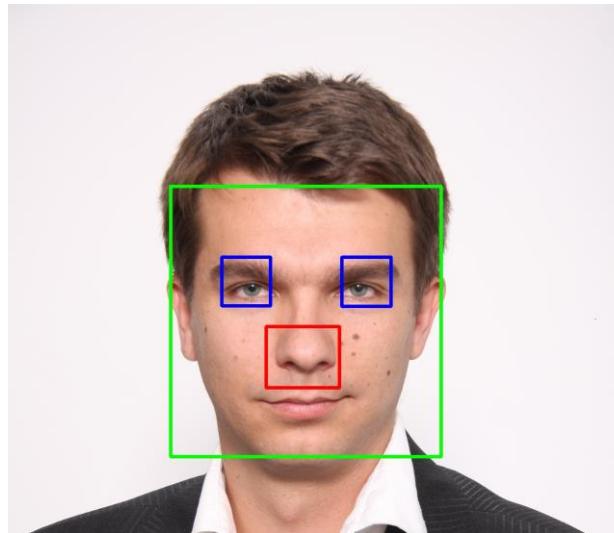
4. Final Classifier = Cascading Classifiers

- Haar cascades consists of a series of weak classifiers - those barely better than 50% correct
- If an area passes a single classifier, go to the next classifier; otherwise, area doesn't match



OpenCV already contains many pre-trained classifiers

- Eyes
- Nose
- Mouth
- Face
- Smile
- Body
- Profile face
- Frontal face
-



Code Pack 28

1. ~~Introduction to OpenCV with Python~~
2. ~~Core Operations~~
3. ~~Image Processing in OpenCV~~
4. ~~Feature Detection and Description~~
5. ~~Video Analysis~~
6. ~~Camera Calibration and 3D Reconstruction~~
7. ~~Computational Photography~~
8. Object Detection

Coding Bootcamp Code in Python

PANDAS

What is it?

- Library for data science
 - defines datastructures
 - Series (1D)
 - DataFrame (2D)
 - defines algorithms
 - selection
 - pivot tables
 - defines utilities
 - visualization, e.g., scatter_matrix
- Backed by numpy
- Nice to experiment with data, jupyter notebooks

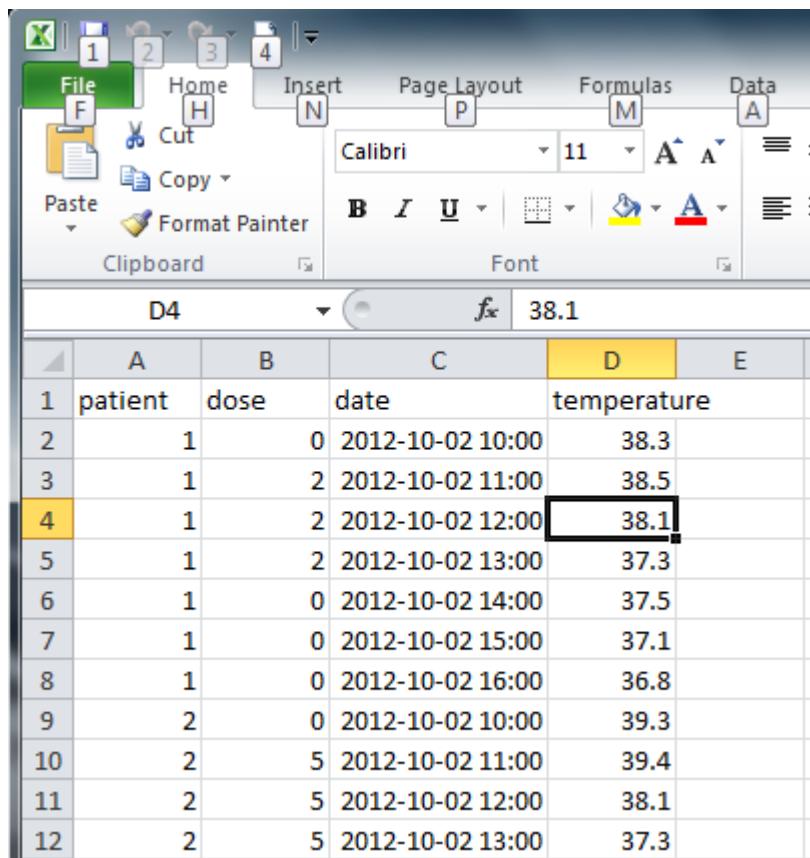
```
import pandas as pd
```

convention

R dataframes
for Python

Example data

- Microsoft Excel spreadsheet



A screenshot of a Microsoft Excel spreadsheet. The ribbon menu is visible at the top, showing tabs for File, Home, Insert, Page Layout, Formulas, and Data. The Home tab is selected. The font toolbar below the ribbon shows Calibri, 11pt, bold, italic, underline, and alignment options. The formula bar at the top indicates the active cell is D4, containing the value 38.1. The spreadsheet contains the following data:

	A	B	C	D	E
1	patient	dose	date	temperature	
2		1	0	2012-10-02 10:00	38.3
3		1	2	2012-10-02 11:00	38.5
4		1	2	2012-10-02 12:00	38.1
5		1	2	2012-10-02 13:00	37.3
6		1	0	2012-10-02 14:00	37.5
7		1	0	2012-10-02 15:00	37.1
8		1	0	2012-10-02 16:00	36.8
9		2	0	2012-10-02 10:00	39.3
10		2	5	2012-10-02 11:00	39.4
11		2	5	2012-10-02 12:00	38.1
12		2	5	2012-10-02 13:00	37.3

Read dataframe

- Create DataFrame from Excel file
 - Also tabular data, CSV, HDF5, SQL query, HTML page,...
- Show in notebook

```
In [3]: data = pd.read_excel('data/patients.xlsx')
data
```

	patient	dose	date	temperature
0	1	0	2012-10-02 10:00:00	38.3
1	1	2	2012-10-02 11:00:00	38.5
2	1	2	2012-10-02 12:00:00	38.1
3	1	2	2012-10-02 13:00:00	37.3
4	1	0	2012-10-02 14:00:00	37.5
5	1	0	2012-10-02 15:00:00	37.1
6	1	0	2012-10-02 16:00:00	36.8
7	2	0	2012-10-02 10:00:00	39.3
8	2	5	2012-10-02 11:00:00	39.4
9	2	5	2012-10-02 12:00:00	38.1
10	2	5	2012-10-02 13:00:00	37.3
11	2	0	2012-10-02 14:00:00	36.8
12	2	0	2012-10-02 15:00:00	36.8

Transform dataframe

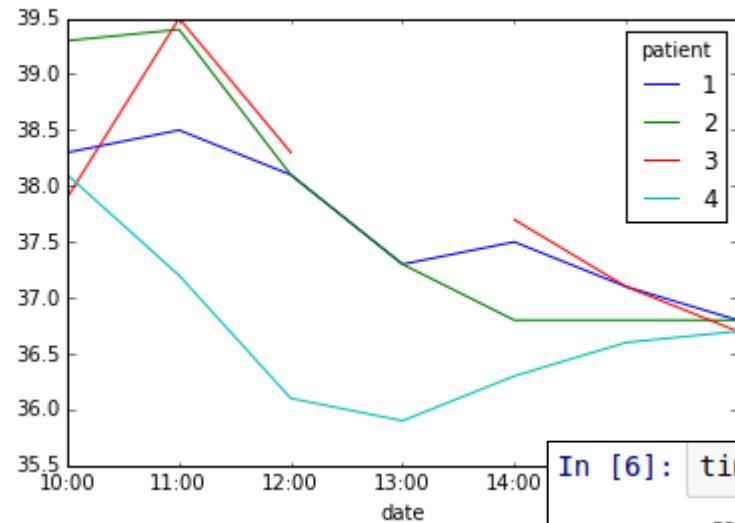
- Patient data as columns, date as index:
pivot table

		dose				temperature			
patient		1	2	3	4	1	2	3	4
date									
2012-10-02 10:00:00	0	0	0	0		38.3	39.3	37.9	38.1
2012-10-02 11:00:00	2	5	2	5		38.5	39.4	39.5	37.2
2012-10-02 12:00:00	2	5	5	5		38.1	38.1	38.3	36.1
2012-10-02 13:00:00	2	5	2	0		37.3	37.3	NaN	38.9
2012-10-02 14:00:00	0	0	2	Nan		37.5	36.8	37.7	36.3
2012-10-02 15:00:00	0	0	2	0		37.1	36.8	37.1	36.6
2012-10-02 16:00:00	0	0	0	0		36.8	36.8	36.7	36.7

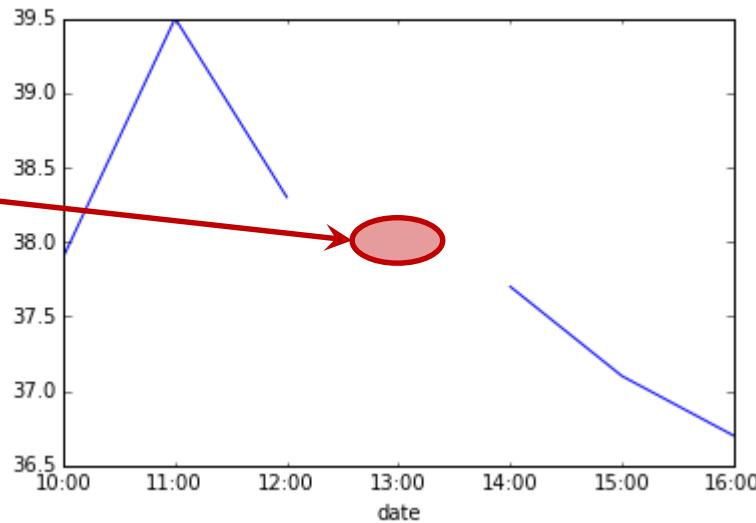
missing value

Plot data

```
In [5]: timeseries['temperature'].plot();
```



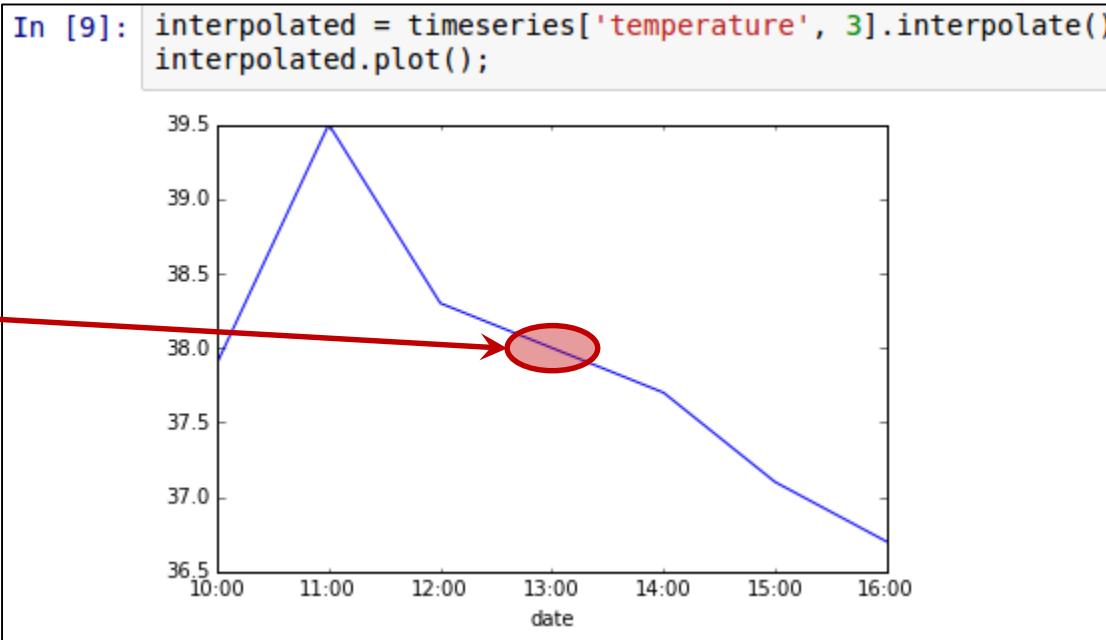
```
In [6]: timeseries['temperature', 3].plot();
```



missing value

Missing data: NaNs

- Can be filled with 0, other value, or interpolated



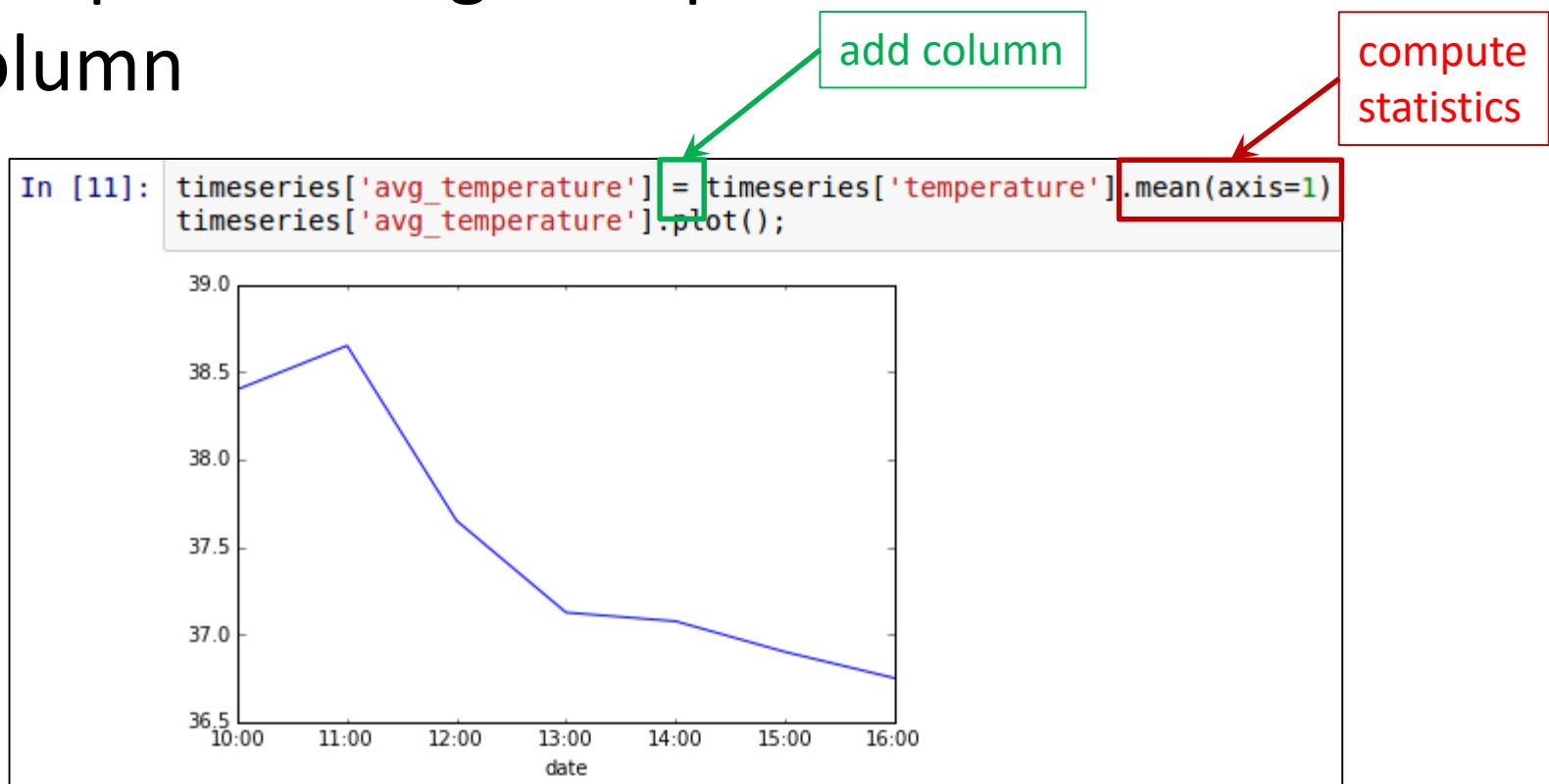
In-place changes

- Changes produce new DataFrame
 - good to experiment
 - bad for performance/memory usage
- Use in place

In [10]: timeseries.interpolate(inplace=True) timeseries									
patient	dose				temperature				avg_temperature
	1	2	3	4	1	2	3	4	
date									
2012-10-02 10:00:00	0	0	0	0	38.3	39.3	37.9	38.1	38.400000
2012-10-02 11:00:00	2	5	2	5	38.5	39.4	39.5	37.2	38.650000
2012-10-02 12:00:00	2	5	5	5	38.1	38.1	38.3	36.1	37.650000
2012-10-02 13:00:00	2	5	2	0	37.3	37.3	38.0	35.9	36.833333
2012-10-02 14:00:00	0	0	2	0	37.5	36.8	37.7	36.3	37.075000
2012-10-02 15:00:00	0	0	2	0	37.1	36.8	37.1	36.6	36.900000
2012-10-02 16:00:00	0	0	0	0	36.8	36.8	36.7	36.7	36.750000

Statistics & adding columns

- Compute average temperature and add as column



Cumulative sum

- Dose is better expressed cumulatively

compute
sums

modify column

cumsum()

In [16]:

```
for patient_id in timeseries['dose'].columns:  
    timeseries['cum_dose', patient_id] = timeseries['dose', patient_id].  
timeseries
```

Out[16]:

	dose				temperature				avg_temperature	cum_dose			
patient	1	2	3	4	1	2	3	4		1	2	3	4
date													
2012-10-02 10:00:00	0	0	0	0	38.3	39.3	37.9	38.1	38.400	0	0	0	0
2012-10-02 11:00:00	2	5	2	5	38.5	39.4	39.5	37.2	38.650	2	5	2	5
2012-10-02 12:00:00	2	5	5	5	38.1	38.1	38.3	36.1	37.650	4	10	7	10
2012-10-02 13:00:00	2	5	2	0	37.3	37.3	38.0	35.9	37.125	6	15	9	10
2012-10-02 14:00:00	0	0	2	0	37.5	36.8	37.7	36.3	37.075	6	15	11	10
2012-10-02 15:00:00	0	0	2	0	37.1	36.8	37.1	36.6	36.900	6	15	13	10
2012-10-02 16:00:00	0	0	0	0	36.8	36.8	36.7	36.7	36.750	6	15	13	10

More pivot & query

- `pivot_table` and `query` are powerful!

```
In [28]: data.pivot_table(index=['patient'], values=['dose', 'temperature'],
                        aggfunc={'dose': np.sum, 'temperature': np.max})
```

Out[28]:

	dose	temperature
patient		
1	6	38.5
2	15	39.4
3	13	39.5
4	10	38.1

What is total dose versus maximum temperature for each patient?

```
In [43]: data.query('temperature > 39.0').loc[:, ['patient', 'date', 'temperature']]
```

Out[43]:

	patient	date	temperature
7	2	2012-10-02 10:00:00	39.3
8	2	2012-10-02 11:00:00	39.4
15	3	2012-10-02 11:00:00	39.5

Which patients had temperature > 39, and when?

Reading HTML tables

```
In [14]: genes_data = pd.read_html('data/genes.html', index_col=0, header=0)
genes = genes_data[0]
genes.columns = [int(x) for x in genes.columns]
genes
```

Out[14]:

	1	2	3	4
FXDG	0.010199	0.042988	0.831946	-0.023656
VTUR	0.466012	0.494679	0.806661	0.518115
OVAH	0.783847	0.754150	0.208352	0.826368
AKSE	0.922433	0.929716	0.662824	0.959443
SJNN	0.325089	0.302198	0.522847	0.356346
LVKM	0.554702	0.531044	0.138988	0.600229

column
names

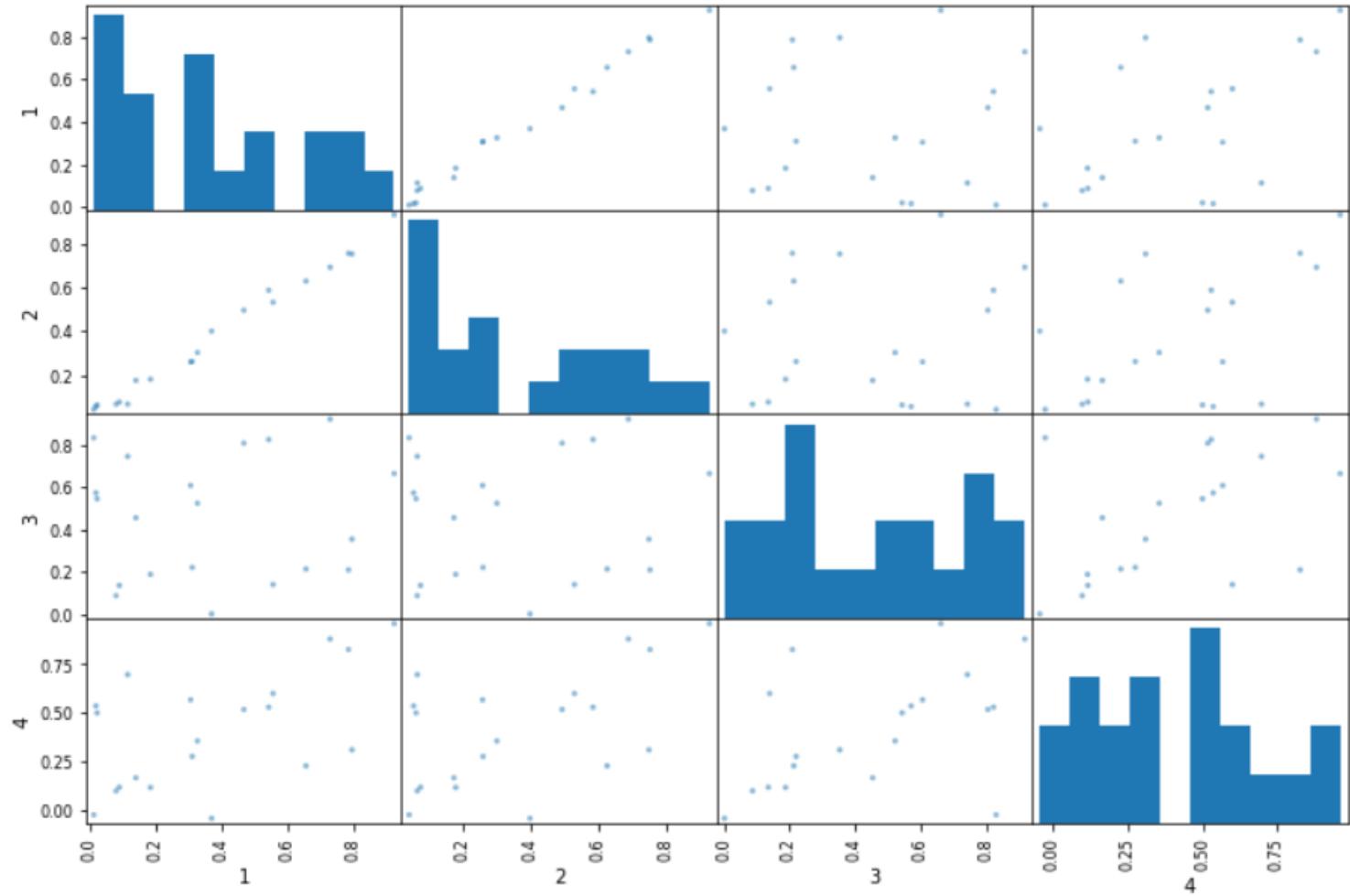
index
column

read_html produces list of DataFrame,
one per HTML table on page

Column names are str by default,
converted to int for consistency
with running example

Scatter matrix

```
In [24]: pd.plotting.scatter_matrix(genes, figsize=(12, 8));
```



Computing correlations

```
In [17]: indices = list(genes.columns)
genes_corr = pd.DataFrame(index=indices, columns=indices)
genes_p_value = pd.DataFrame(index=indices, columns=indices)
for id1 in genes.columns:
    for id2 in genes.columns:
        genes_corr[id1][id2], genes_p_value[id1][id2] = stats.pearsonr(genes[id1], genes[id2])
```

Creating
DataFrames
by hand

```
In [18]: genes_corr
```

```
Out[18]:
```

	1	2	3	4
1	1	0.9936518	0.04567655	0.5474551
2	0.9936518	1	0.0728375	0.5373312
3	0.04567655	0.0728375	1	0.4850267
4	0.5474551	0.5373312	0.4850267	1

Note:

```
import scipy.stats as stats
```

```
In [19]: genes_p_value
```

```
Out[19]:
```

	1	2	3	4
1	0	1.554131e-18	0.8483549	0.01247343
2	1.554131e-18	0	0.7602363	0.01455602
3	0.8483549	0.7602363	0	0.03018841
4	0.01247343	0.01455602	0.03018841	0

Code Pack 29

Pandas

Not this one:



Coding Bootcamp Code in Python

MACHINE LEARNING

Introduction

- Machine learning is making great strides
 - Large, good data sets
 - Progress in algorithms
 - Many libraries
 - scikit-learn
 - PyTorch
 - TensorFlow
 - Keras
 - ...
-
- The diagram consists of three rectangular boxes with arrows pointing to them. A bracket on the left groups the library names 'scikit-learn', 'PyTorch', 'TensorFlow', 'Keras', and '...'. An arrow points from the top box, labeled 'classic machine learning', to the top of the bracket. Another arrow points from the middle box, labeled 'deep learning frameworks', to the bottom of the bracket. The bottom box, labeled 'Fast-evolving ecosystem!', encloses the entire bracketed area.

scikit-learn

- Nice end-to-end framework
 - data exploration (+ pandas + holoviews)
 - data preprocessing (+ pandas)
 - cleaning/missing values
 - normalization
 - training
 - testing
 - application
- Classic only?

Sure, but don't jump into deep end
unless you can swim!

Machine learning tasks

- Supervised learning
 - **regression**: predict numerical values
 - **classification**: predict categorical values, i.e., labels
- Unsupervised learning
 - **clustering**: group data according to "distance"
 - association: find frequent co-occurrences
 - link prediction: discover relationships in data
 - data reduction: project features to fewer features
- Reinforcement learning

Data set

- World happiness index 2015 & 2016 (<http://worldhappiness.report/>)
- Happiness score for country based on
 - economic factors (GDP)
 - family situation (social network)
 - health care (life expectancy)
 - freedom
 - trust (government corruption)
 - generosity
 - dystopia residual
- Geographical region, e.g., Western Europe, Southern Asia,...

training: 2015

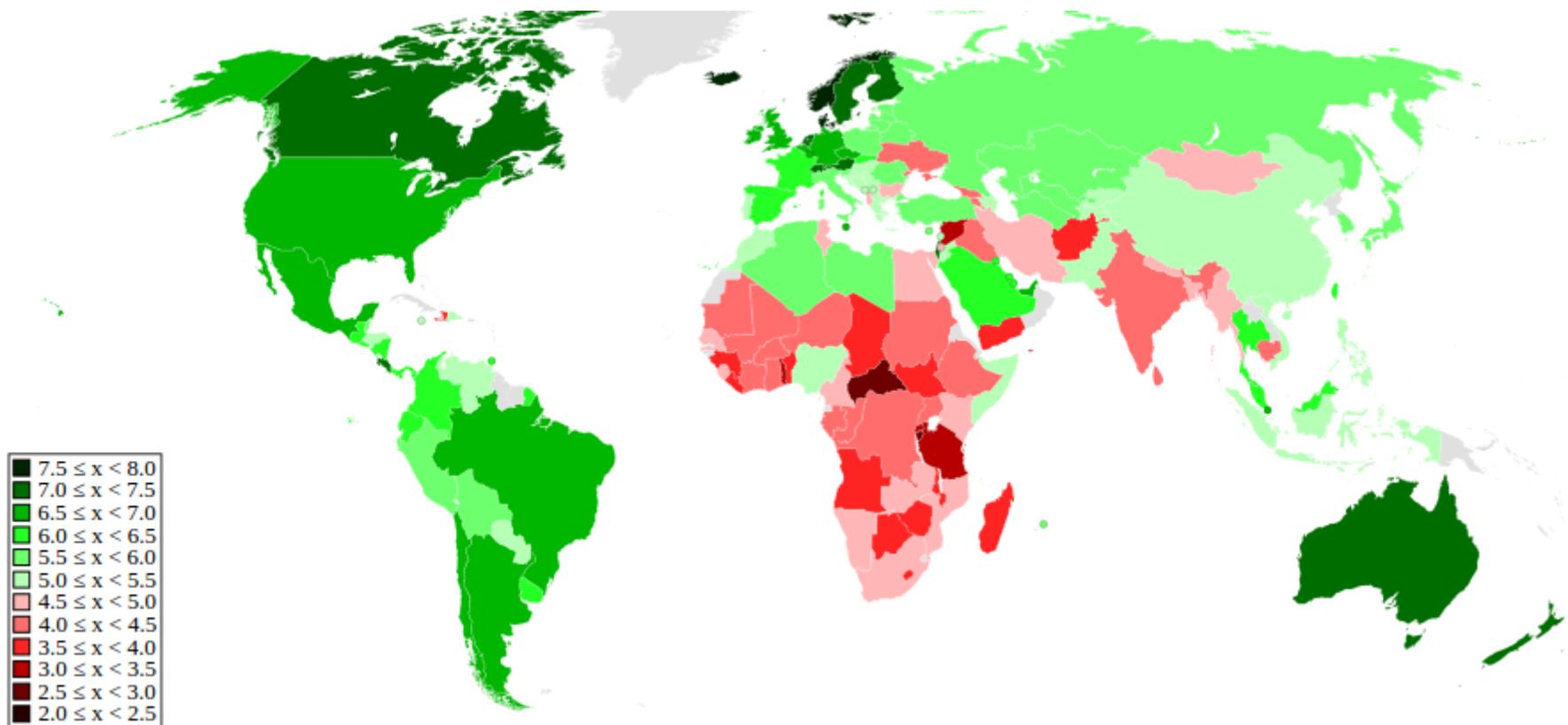
test: 2016

don't touch!

quantitative data

categorical data

World happiness



https://en.wikipedia.org/wiki/World_Happiness_Report

Task 1

- Given
 - economy
 - family
 - health
 - freedom
 - trust
 - generosity
 - dystopia residual
 - region
- Predict happiness score

Regression

Let's peek...

In [4]: `data_2015.describe()`

Out[4]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generc
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615	0.143422	0.237
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693	0.120034	0.126
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330	0.061675	0.150
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515	0.107220	0.216
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092	0.180255	0.309
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	0.551910	0.795

Rescaling

Missing values?

```
In [5]: data_2015.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158 entries, 0 to 157
Data columns (total 12 columns):
Country          158 non-null   object
Region           158 non-null   object
Happiness Rank   158 non-null   float64
Happiness Score  158 non-null   float64
Standard Error   158 non-null   float64
Economy (GDP per Capita) 158 non-null   float64
Family           158 non-null   object
Health (Life Expectancy) 158 non-null   float64
Freedom          158 non-null   object
Trust (Government Corruption) 158 non-null   object
Generosity        158 non-null   float64
Dystopia Residual 158 non-null   float64
dtypes: float64(9), int64(1), object(2)
memory usage: 14.9+ KB
```

```
In [63]: data_2015.count()
```

```
Out[63]: Country          158
Region           158
Happiness Rank   158
Happiness Score  158
Standard Error   158
Economy (GDP per Capita) 158
Family           158
Health (Life Expectancy) 158
Freedom          158
Trust (Government Corruption) 158
Generosity        158
Dystopia Residual 158
dtype: int64
```

No NaNs, otherwise, use Imputer

Extracting data

- Part of machine learning pipeline
 - fit + transform
- Extracting columns from pandas data frame
- Transform data
 - scale numerical features to [0, 1]
 - one-hot encoding for regions

Numerical attributes pipeline

- Extract, then scale

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler

num_attrs = data_2015.columns[5:13]

num_attrs_transformer = ColumnTransformer([
    ('minimax', MinMaxScaler(), num_attr_names)
])
```

name of operation

operation constructor

column names



Categorical attribute pipeline

- Extract, then create one-hot attributes

Country	Region
Belgium	Western Europe
Canada	North America
Germany	Western Europe
...	...

Country	Western Europe	Northren America	...
Belgium	1	0	...
Canada	0	1	...
Germany	1	0	...
...



```
from sklearn.preprocessing import OneHotEncoder

region_transformer = ColumnTransformer([
    ('one_hot_encoder', OneHotEncoder(categories='auto'),
     ['Region'])
])
```

Combining pipelines & execution

- Both pipelines must be executed, results combined

```
from sklearn.pipeline import FeatureUnion

preparation_pipeline = FeatureUnion(transformer_list=[  
    ('region_attr', region_attr_transformer),  
    ('num_attrs', num_attrs_transformer),  
])
```

- Run data through pipeline

```
train_data = preparation_pipeline.fit_transform(data_2015)
```

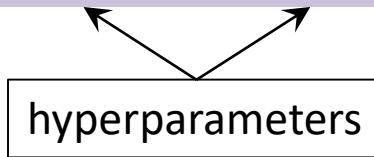
numpy array

Ready to start training!

Training & prediction

- Create learning algorithm

```
from sklearn.linear_model import Ridge  
  
ridge_regr = Ridge(alpha=0.5, fit_intercept=False)
```



- Train

```
X = train_data  
Y = np.array(data_2015['Happiness Score'])  
ridge_regr.fit(X, Y);
```

???

- Predict

```
Y_ridge_regr = ridge_regr.predict(X)
```

Score & errors

- Score

```
ridge_regr.score(X, Y) → 0.9984
```

- Better: cross validation

```
scores = cross_val_score(  
    ridge_regr, X, Y,  
    scoring='neg_mean_squared_error',  
    cv=10  
)
```

10-fold

```
np.sqrt(-scores)
```



0.1954,	0.0215,
0.0579,	0.0478,
0.0551,	0.0649,
0.0591,	0.0451,
0.0729,	0.0803

Fine tuning

- Define hyper parameter search space

```
grid_params = [  
    {'alpha': [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5],  
     'fit_intercept': [True, False]},  
]
```

- Define grid searcher

```
grid_search = GridSearchCV(ridge_regr, grid_params, cv=10,  
                           scoring='neg_mean_squared_error')
```

- Search

```
grid_search.fit(X, Y)
```

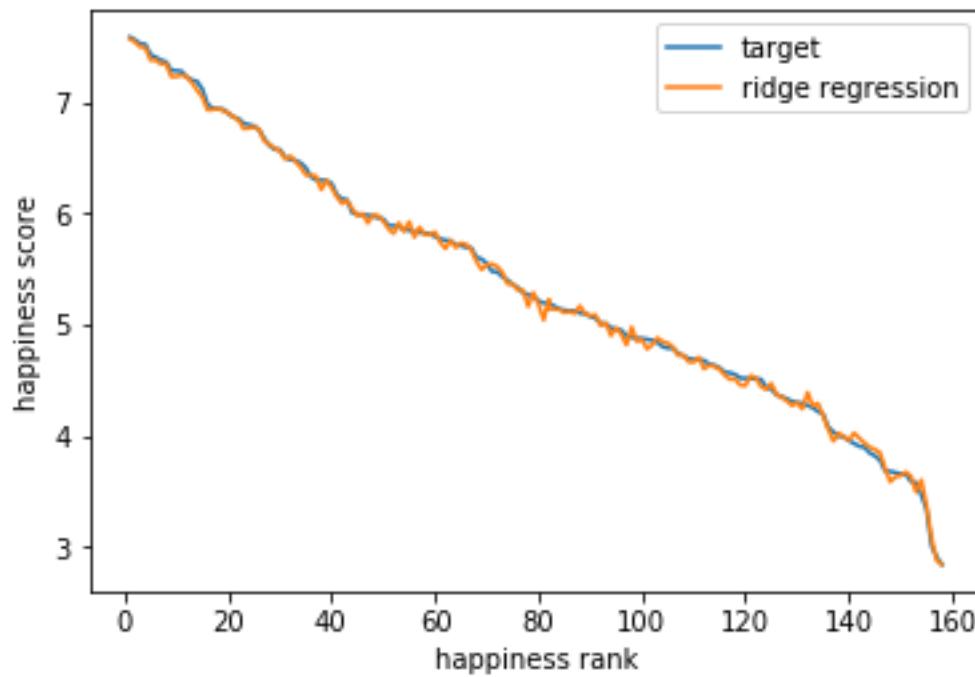
- Best hyper parameters

```
grid_search.best_params_
```



```
{'alpha': 0.0005, 'fit_intercept': True}
```

Training result



Testing the model

- Use pipeline

```
test_data = preparation_pipeline.transform(data_2016)
```

Note: only transform, no fit_transform!

- Predict

```
Y_test = ridge_regr.predict(X_test)
```

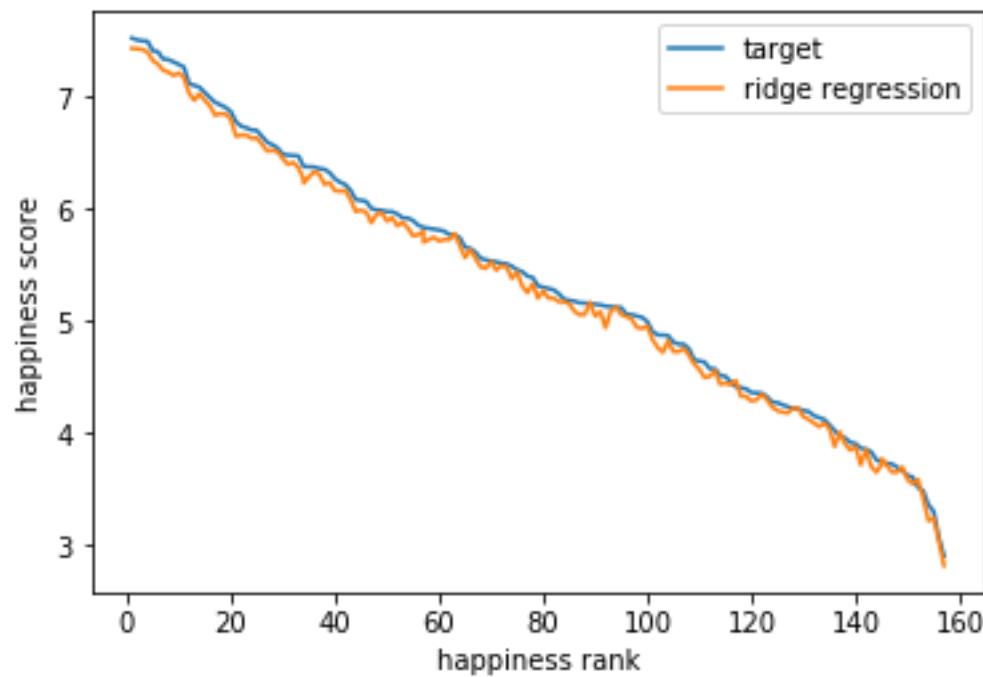
- Score

```
ridge_regr.score(X_test, Y_test)
```



0.9950

Test result



Code Pack 30

1.regression_world_happiness

Task 2

- Given
 - economy
 - family
 - health
 - freedom
 - trust
 - generosity
 - dystopia residual
- Predict whether country
in Western Europe

Classification

Data preparation

- Numerical pipeline is same as in task 1
 - select from pandas DataFrame
 - scale

```
prepared_data = num_attrs_pipeline.fit_transform(data_2015)
```

- Add column for class
 - True if in Western Europe, False otherwise

```
data_2015['Western Europe'] = (data_2015['Region'] ==  
                                'Western Europe')
```

Training & scoring

- Create learning algorithm

```
from sklearn.naive_bayes import GaussianNB
```

```
nb = GaussianNB()
```

- Train

attributes: approx. Gaussian distr.

```
nb.fit(prepared_data, data_2015['Western Europe'])
```

- Scoring

```
nb.score(prepared_data, data_2015['Western Europe'])
```

Seems reasonable



0.9367

Testing the model

- Use pipeline

```
test_data = num_attrs_pipeline.transform(data_2016)
```

Note: only transform, no fit_transform!

- Predict

```
Y_test = nb.predict(test_data)
```

- Score

```
nb.score(test_data, Y_test)
```



0.8662

Not so good

Actually...

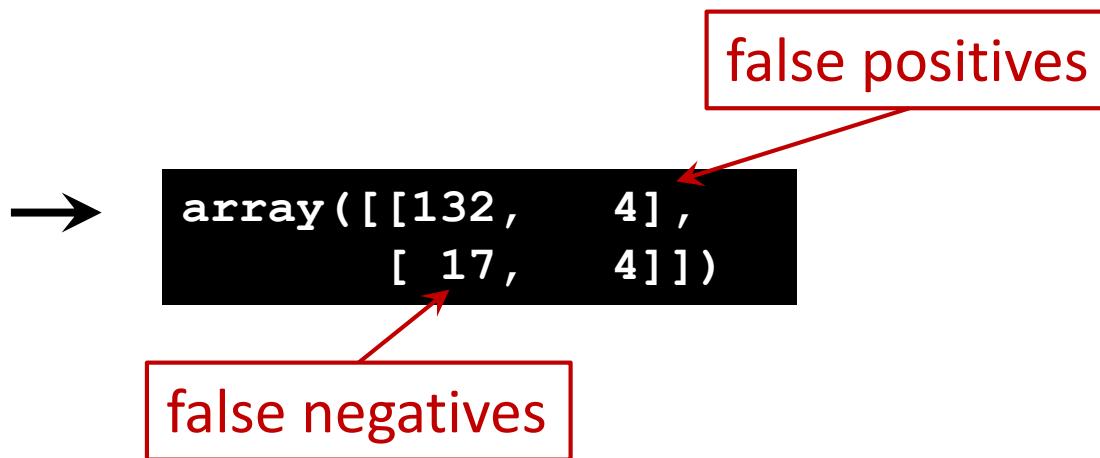
- Compute confusion matrix

```
confusion_matrix(data_2016['Western Europe'],  
                  nb.predict(test_data))
```

→ **array([[132, 4],
 [17, 4]])**

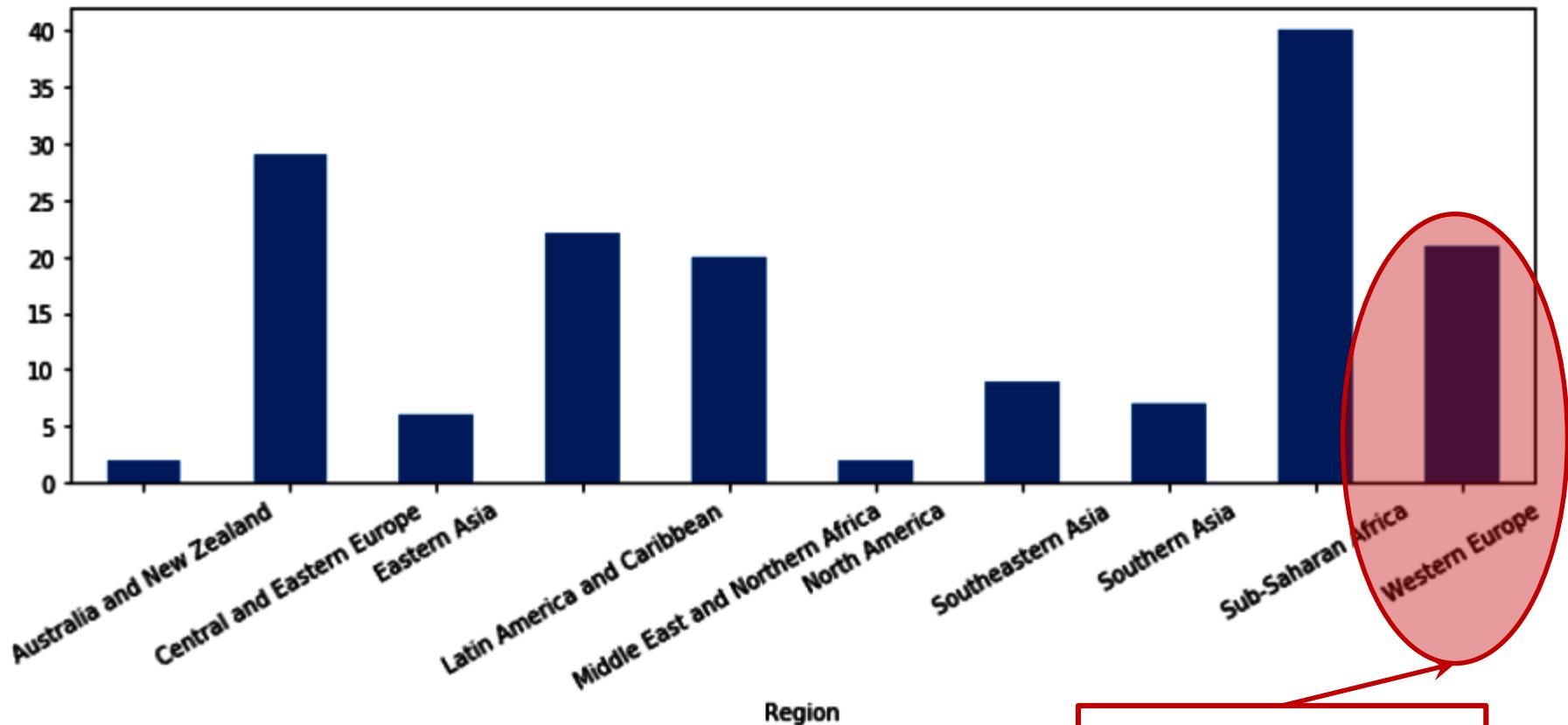
false positives

false negatives



Massive exit from Western Europe: 17 countries just left!

Data set properties



small class,
many detractors

Code Pack 30

~~1.regression_world_happiness~~

2.classification_finding_regions

Task 3

- Given
 - economy
 - family
 - health
 - freedom
 - trust
 - generosity
 - dystopia residual
- Find countries that are "close"

Clustering

Data preparation & Training

- Data preprocessing same as task 2
- Create learning algorithm

```
from sklearn.cluster import KMeans  
k_means = KMeans(n_clusters=3)
```

How many clusters?

- Train

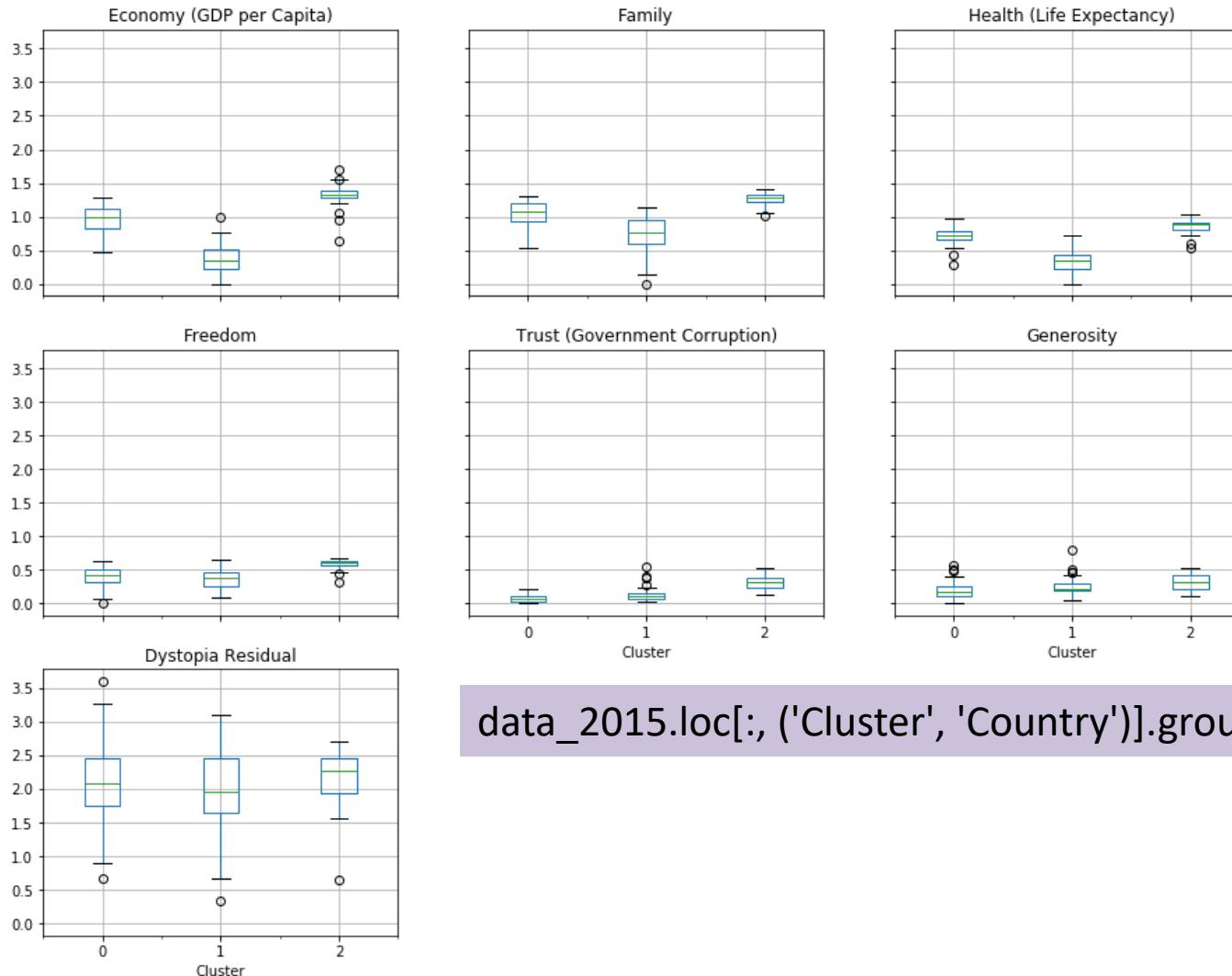
```
k_means.fit(prepared_data)
```

- Add cluster label to data set

```
data_2015['Cluster'] = 0  
for i in range(3):  
    data_2015.loc[clusterer.labels_ == i, 'Cluster'] = i
```

Examine clusters

```
data_2015.boxplot(by='Cluster', column=num_attr_names)
```



```
data_2015.loc[:, ('Cluster', 'Country')].groupby('Cluster').count()
```

Country	Cluster	Count
0	0	77
1	1	50
2	2	31

Code Pack 30

1.~~regression_world_happiness~~

2.~~classification_finding_regions~~

3.cluster_countries

Coding Bootcamp Code in Python

EXPOSING YOUR MODEL: FLASK

Introduction

- Micro-services
 - single user web application
 - can act as API (REST interface)
 - can act as GUI (HTML/Javascript)
- Many frameworks, e.g., Flask
 - easy to install
 - nice for simple applications
- Note: security!

Did I mention that you should really, *really* know what you are doing?

Do not (and I mean never ever)
use this for production or die!



Hello Flask

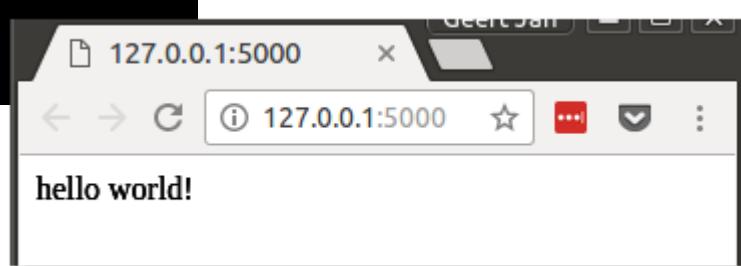
- Web application

```
from flask import Flask  
  
app = Flask(__name__)  
  
@app.route('/')  
def hello_world():  
    return 'hello world!'  
  
if __name__ == '__main__':  
    app.run()
```

Handles request for /

- Run

```
$ python ./hello_world.py  
* Running on http://127.0.0.1:5000/  
(Press CTRL+C to quit)
```

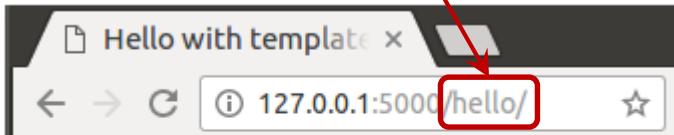


Hello yourself

- Routing with variables

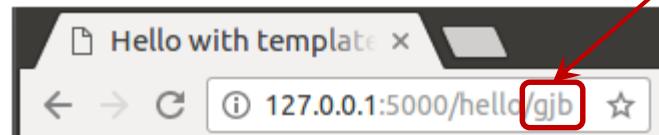
```
@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return 'hello {0}!'.format(name if name else 'world')
```

default route: name == None



Hello world!

variable



Hello gjb!

Variable types: int, float, path, uuid

Templates

- Templates: Flask uses jinja2

```
<!doctype html>                                templates/hello.html

<title>Hello with templates</title>

<body>
  {% if name %}
    <h1>Hello {{name}}!</h1>
  {% else %}
    <h1>Hello world!</h1>
  {% endif %}
  <p> This is a jinja2 template. </p>
</body>
```

if name defined → **variable substitution**

if name undefined → **variable substitution**

- Using template

```
from flask import render_template
...
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

Form template

- Form template (ugly, use CSS for style)

```
<!doctype html>
<title>EasySum</title>
<body>
    <h1>EasySum</h2>
    {%
        if result %
            <p> The result of {{op1}} + {{op2}} is {{result}}. </p>
        % else %
            <p> This web application computes the sum of two numbers
                , the can even be floating point values. </p>
        <form method="POST">
            First operand: <input type="text" name="op1"/> <br/>
            Second operand: <input type="text" name="op2"/> <br/>
            <input type="submit" value="Compute"/>
        </form>
    % endif %
</body>
```

templates/sum.html

Form handling

- Use request & specify methods

```
from flask import Flask, request, render_template  
...  
@app.route('/', methods=['GET', 'POST'])  
def start():  
    if request.method == 'POST':  
        op1 = float(request.form['op1'])  
        op2 = float(request.form['op2'])  
        result = op1 + op2  
        return render_template('sum.html', op1=op1, op2=op2,  
                             result=result)  
    else:  
        return render_template('sum.html')
```

The code is annotated with three callout boxes:

- A box labeled "Methods to handle" has an arrow pointing to the `@app.route('/')` line.
- A box labeled "Check method" has an arrow pointing to the `if request.method == 'POST':` line.
- A box labeled "Retrieve input values" has an arrow pointing to the `float(request.form['op1'])` line.

- Note: *always* validate input!

Sessions

- Keep track of application state with session

```
app.secret_key = 'A1Zr38g/3yZ R~XGH!j1N]3WX/,?RT'
```

```
@app.route('/reset')
def reset():
    session['fib'] = 1
    session['fib_prev'] = 1
    session['number'] = 0
    return render_template('fib.html', number=None)
```

Persistent variables

Note: Flask server is single session
⇒ no multi-user, multi-session!

```
@app.route('/')
def start():
    if 'number' in session:
        session['number'] += 1
        fib = session['fib']
        session['fib'] += session['fib_prev']
        session['fib_prev'] = fib
        return render_template('fib.html',
                               number=session['number'],
                               fibonacci=session['fib'])
    else:
        return reset()
```

Check state

Retrieve/update
session values

Further reading

- Flask
<http://flask.pocoo.org/docs/latest>
- Jinja2
<http://jinja.pocoo.org/docs/latest>
- Did I mention security is an issue?
http://flask.pocoo.org/docs/latest/advanced_foreword/#develop-for-the-web-with-caution



There be dragons!

pickle module

Python pickle module is used for serializing and de-serializing a Python object structure.

- Any object in Python can be pickled so that it can be saved on disk.
- The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.

```
import pickle

def storeData():
    # initializing data to be stored in db
    Ricardo = {'key' : 'Ricardo', 'name' : 'Ricardo Profile',
               'age' : 43, 'pay' : 40000}
    Pinto = {'key' : 'Pinto', 'name' : 'Pinto Profile',
             'age' : 50, 'pay' : 50000}

    # database
    db = {}
    db['Ricardo'] = Ricardo
    db['Pinto'] = Pinto

    # Its important to use binary mode
    dbfile = open('examplePickle', 'ab')

    # source, destination
    pickle.dump(db, dbfile)
    dbfile.close()

def loadData():
    # for reading also binary mode is important
    dbfile = open('examplePickle', 'rb')
    db = pickle.load(dbfile)
    for keys in db:
        print(keys, '=>', db[keys])
    dbfile.close()

if __name__ == '__main__':
    storeData()
    loadData()
```

Code Pack 31

Full App