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

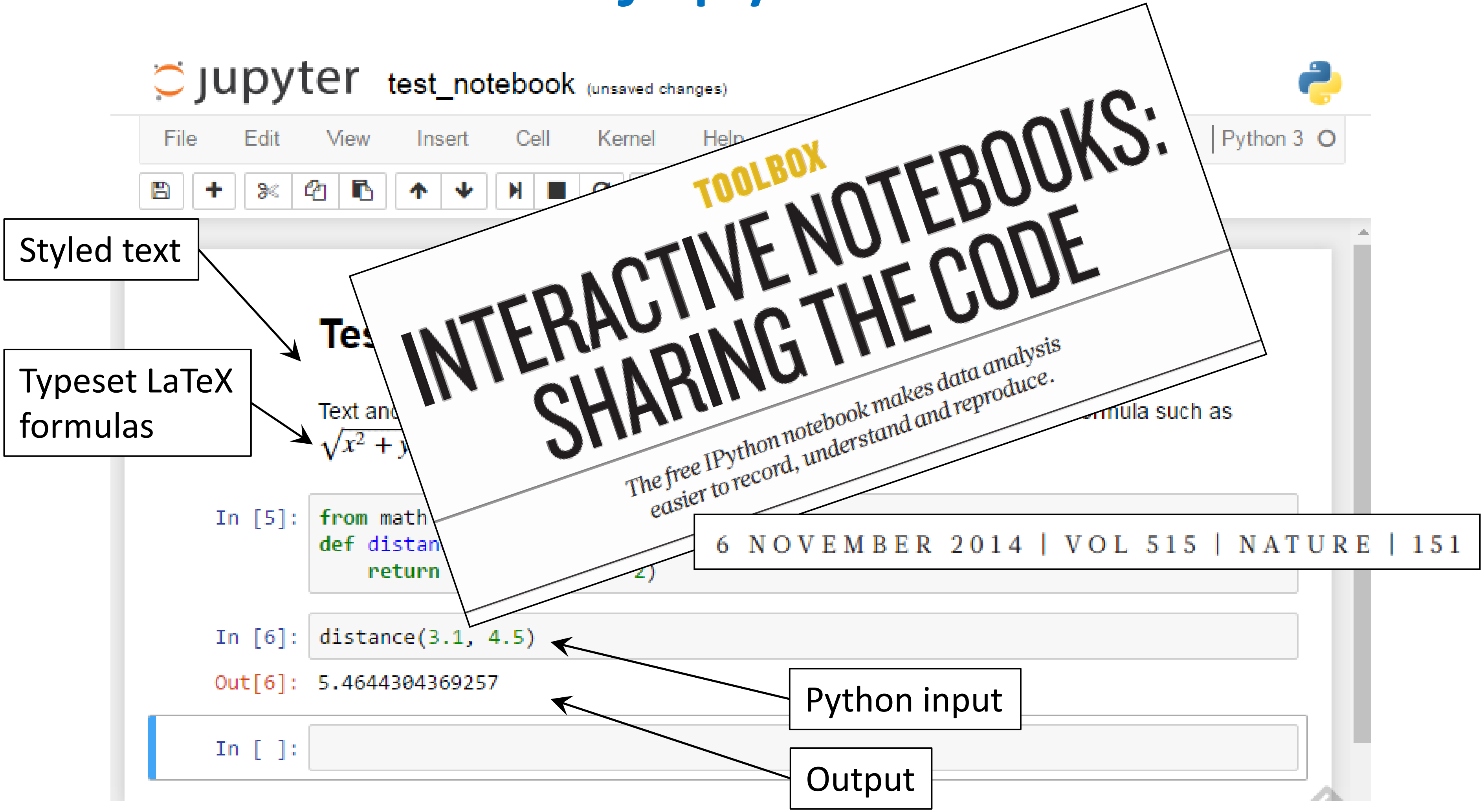
Interactive: iPython

- 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



```
In [5]: from math
        def distan
            return                    2)
```

```
In [6]: distance(3.1, 4.5)
Out[6]: 5.4644304369257
In [ ]:
```

TOOLBOX

INTERACTIVE NOTEBOOKS: SHARING THE CODE

The free IPython notebook makes data analysis easier to record, understand and reproduce.

6 NOVEMBER 2014 | VOL 515 | NATURE | 151

Styled text

Typeset LaTeX formulas

Python input

Output

Interface in web browser

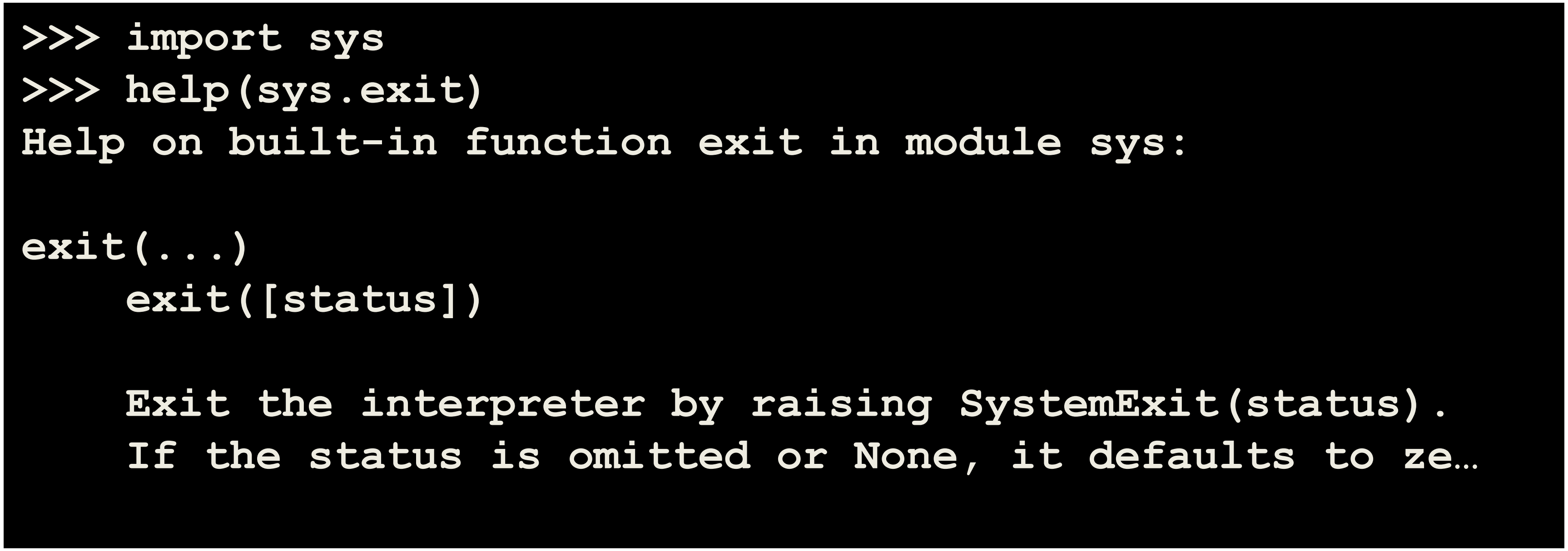
Jupyter use cases

- 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

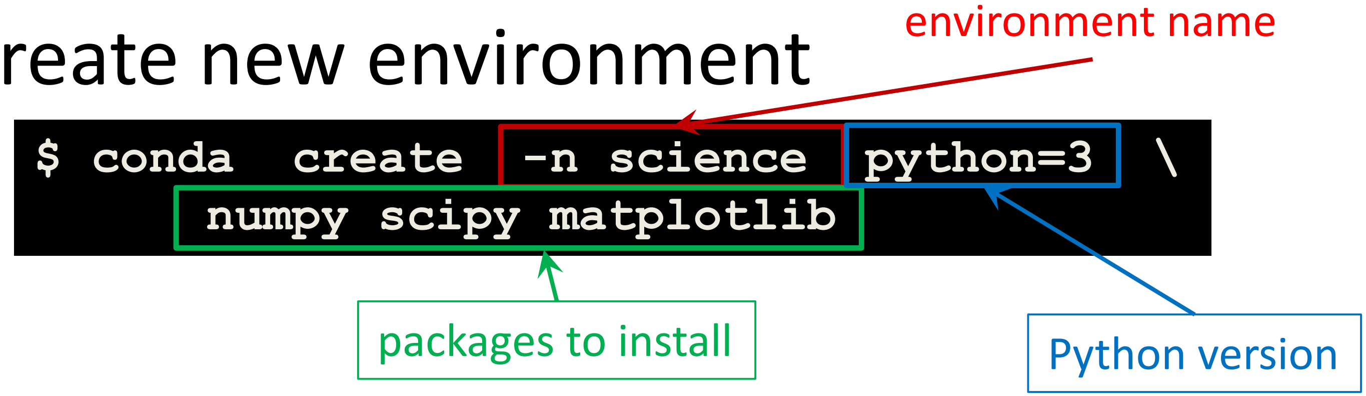
```
$ pip  install  -U  pandas
```

  - 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

packages to install

Python version

- Use environment

```
$ conda activate science
```

- Deactivate environment

```
$ 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
```

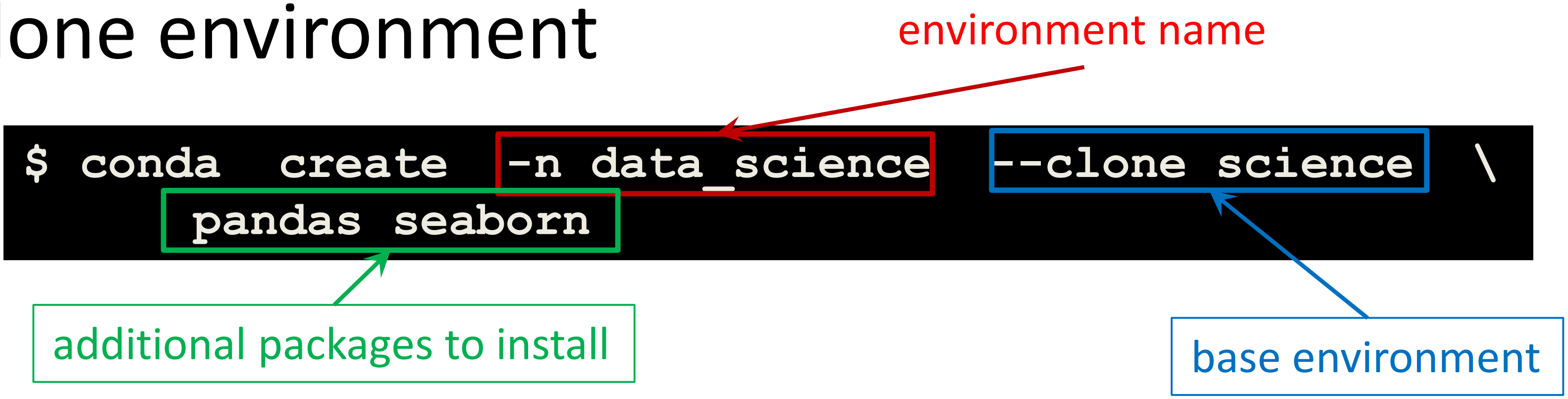
- List all installed packages

```
$ 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
```

environment name

base environment

additional packages to install

- 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

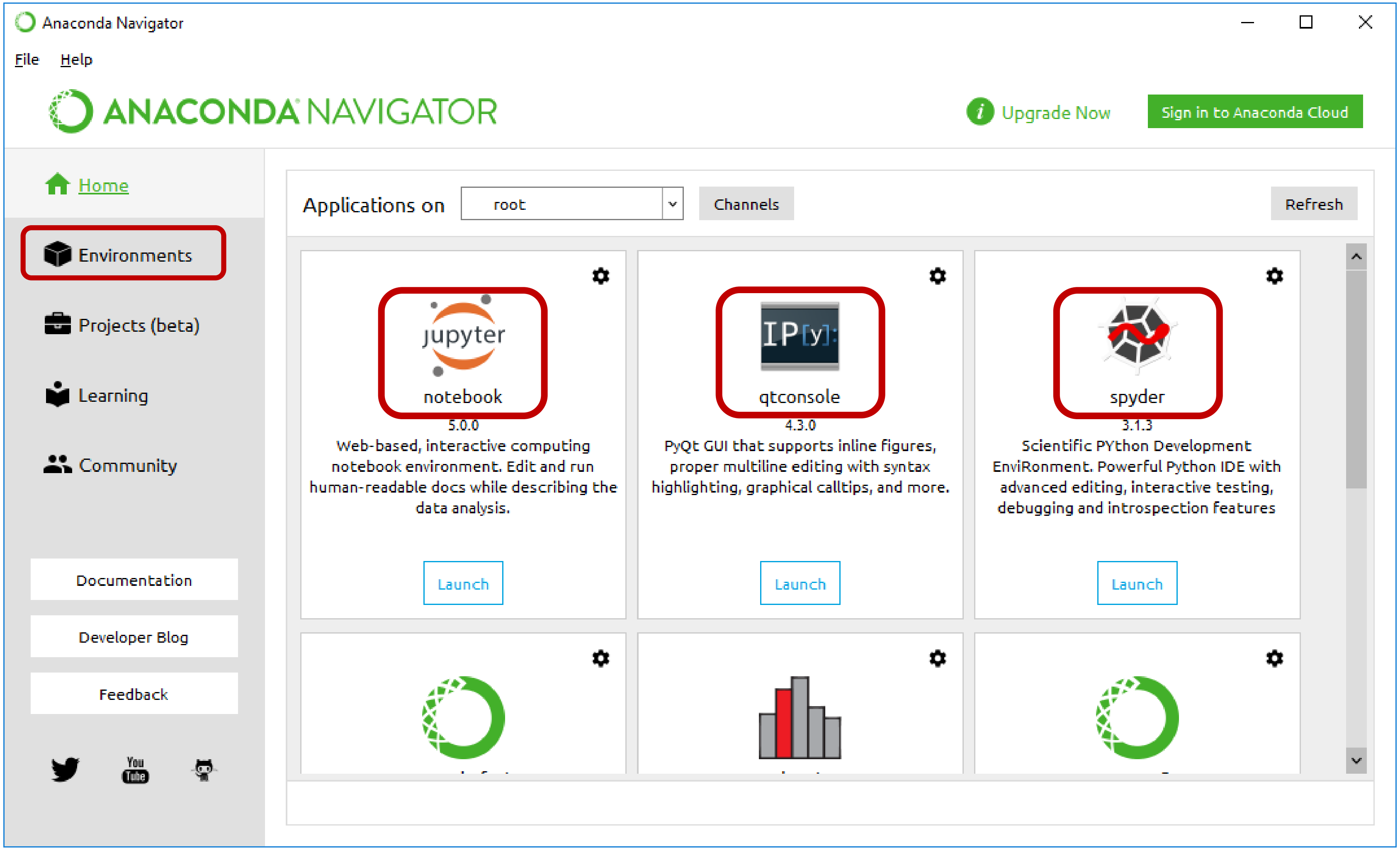
Coding Bootcamp Code in Python

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

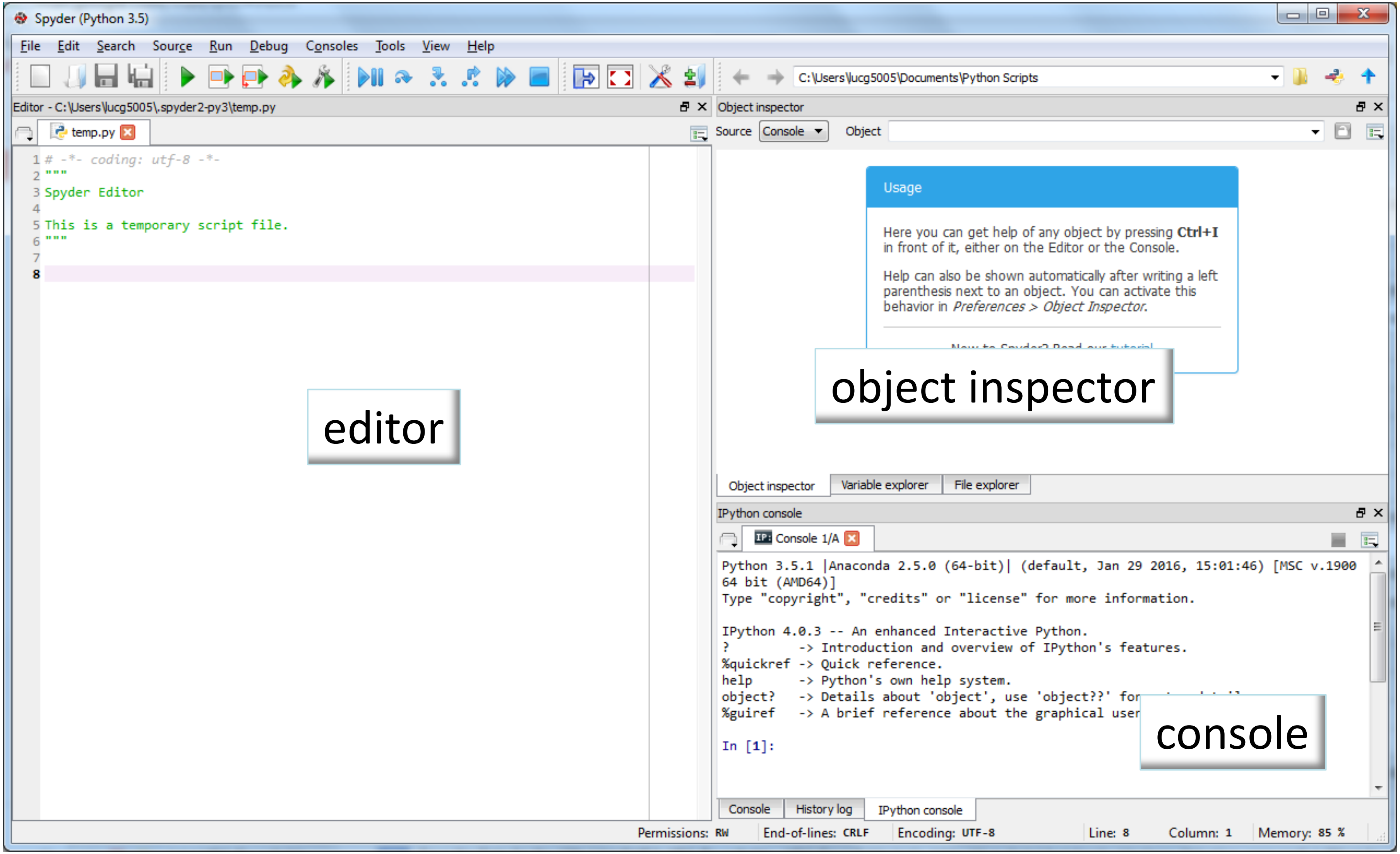


Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

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

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

spyder



C:\Users\ucg5005\Documents\Python Scripts

Editor - C:\Users\ucg5005\.spyder2-py3\temp.py

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

```

Object inspector

Source: Console Object:

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Object Inspector.*

```
Python 3.5.1 |Anaconda 2.5.0 (64-bit)| (default, Jan 29 2016, 15:01:46) [MSC v.1900
64 bit (AMD64)]
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
%guiref   -> A brief reference about the graphical user

In [1]:
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 8 Column: 1 Memory: 85 %

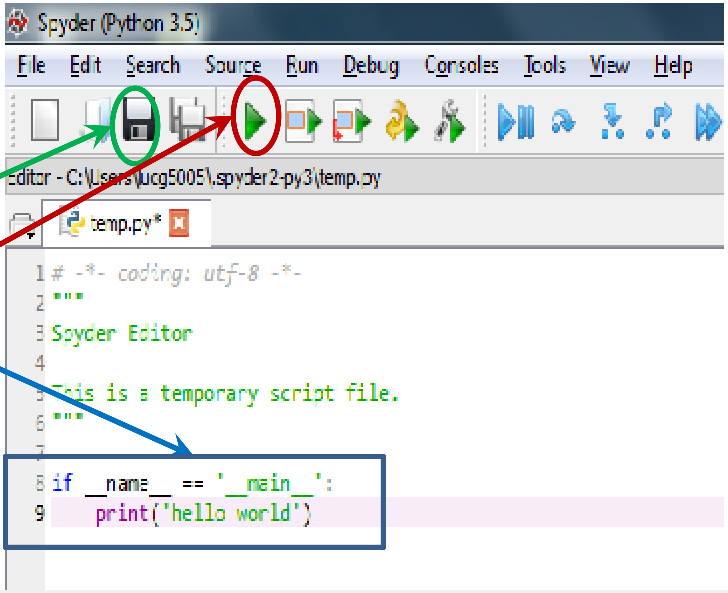
editor

object inspector

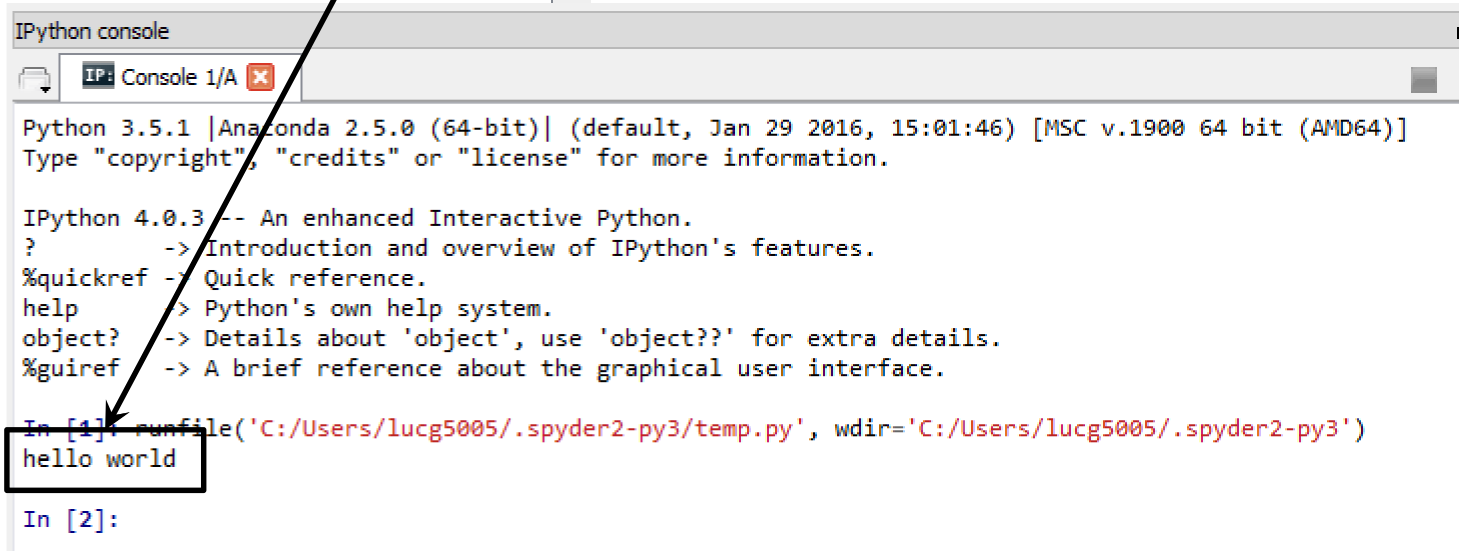
console

spyder: work cycle

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


Spyder (Python 3.5)
File Edit Search Source Run Debug Consoles Tools View Help
Editor - C:\Users\lucg5005\.spyder2-py3\temp.py
temp.py*

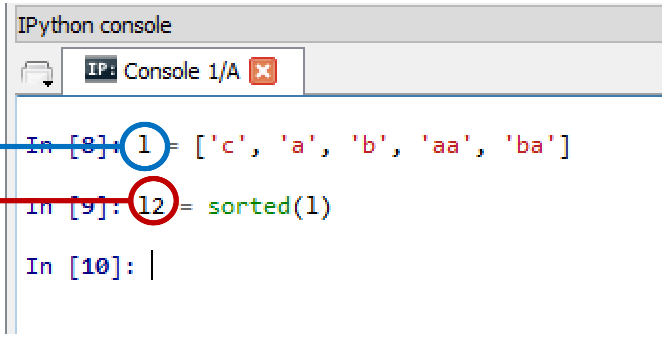
```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

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



```
Python 3.5.1 |Anaconda 2.5.0 (64-bit)| (default, Jan 29 2016, 15:01:46) [MSC v.1900 64 bit (AMD64)]
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.
%guiref   -> A brief reference about the graphical user interface.

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

In [2]:
```

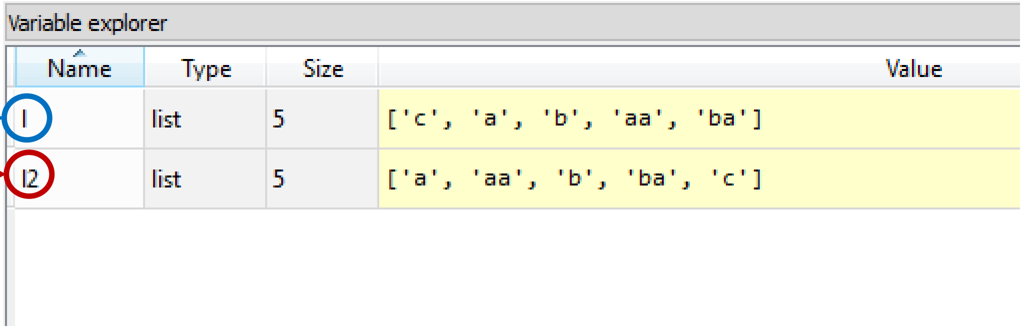
spyder: object inspector

- Executing code snippets



```
In [8]: l = ['c', 'a', 'b', 'aa', 'ba']

In [9]: l2 = sorted(l)

In [10]: |
```

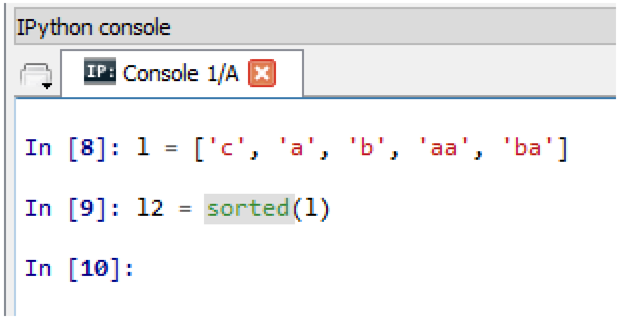
- Variable values in inspector



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

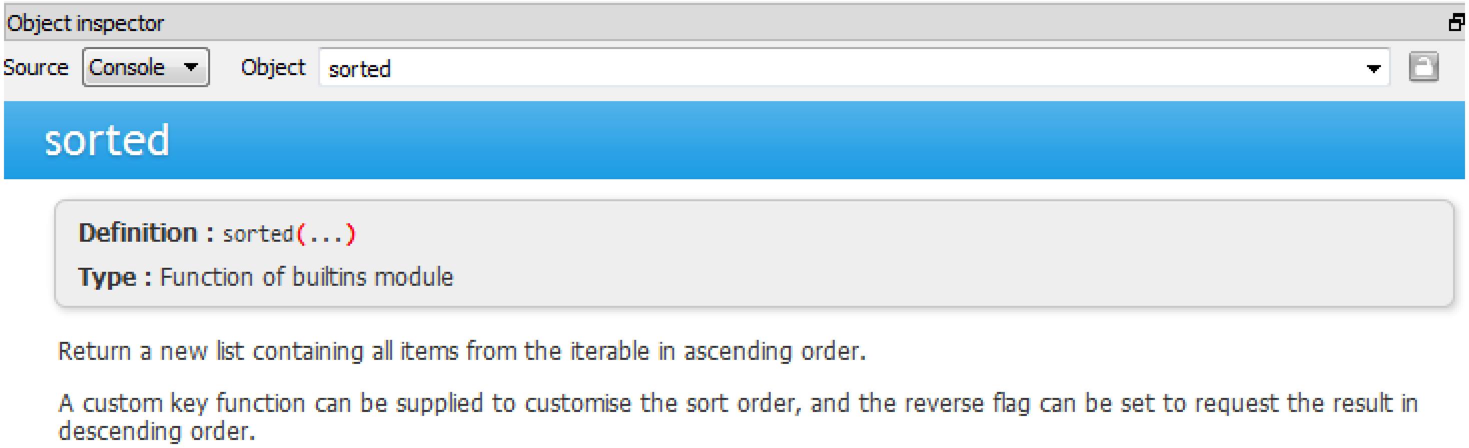
spyder: getting help

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



```
In [8]: l = ['c', 'a', 'b', 'aa', 'ba']

In [9]: l2 = sorted(l)

In [10]:
```

- Help in object inspector



Source Console

Object sorted

sorted

Definition : `sorted(...)`

Type : Function of builtins module

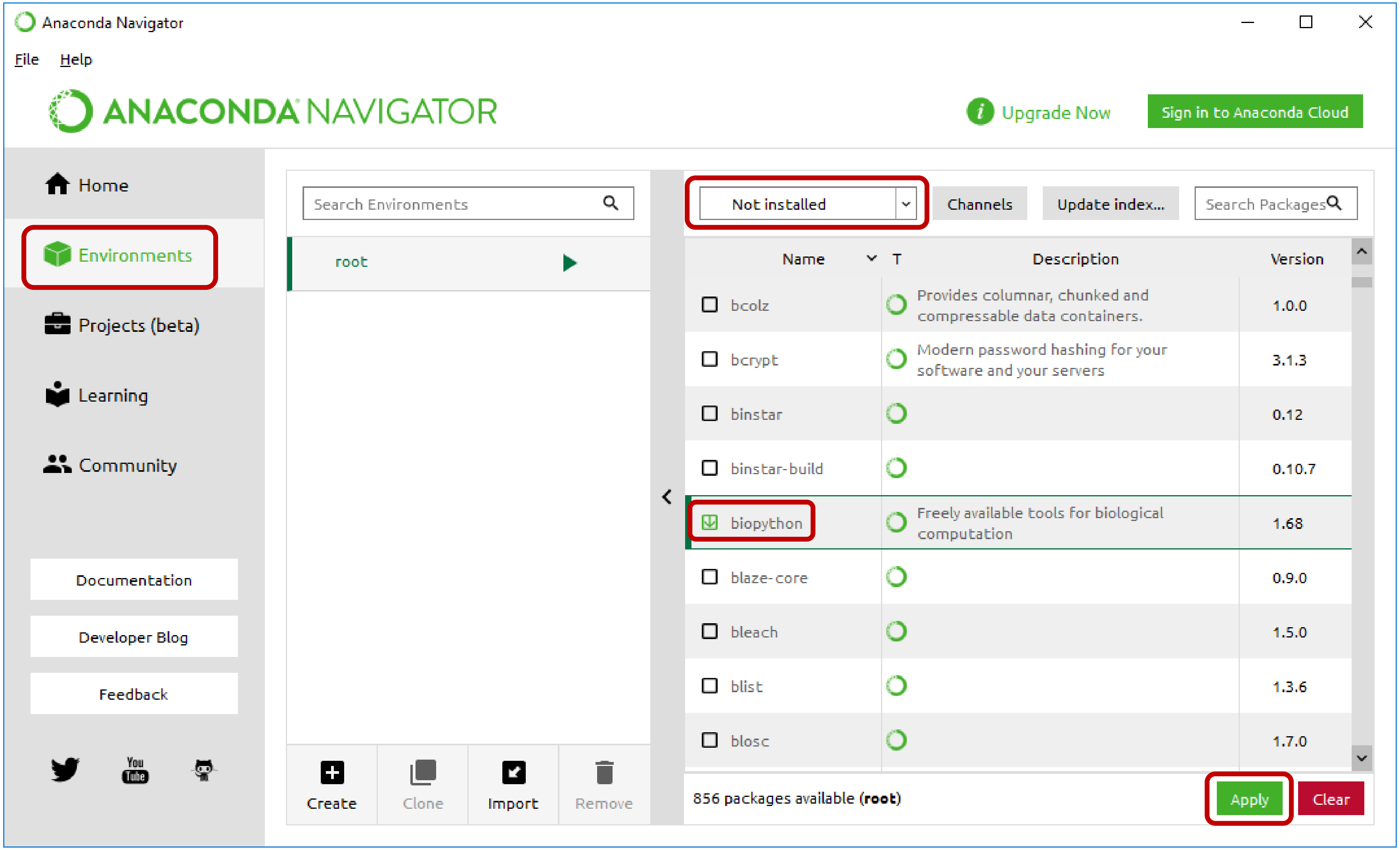
Return a new list containing all items from the iterable in ascending order.

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



Not installed

Name	T	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)**

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
```

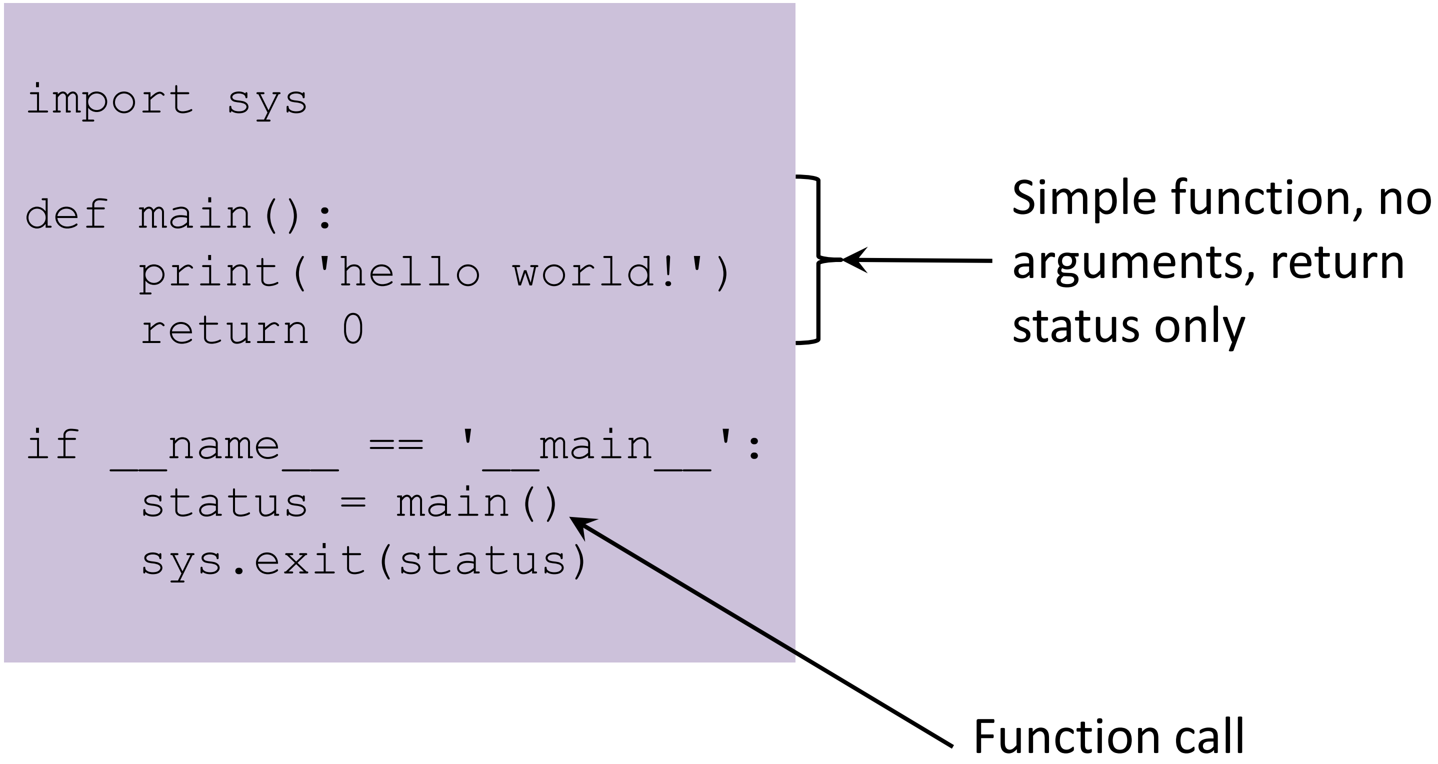
- Run script directly

```
$ ./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

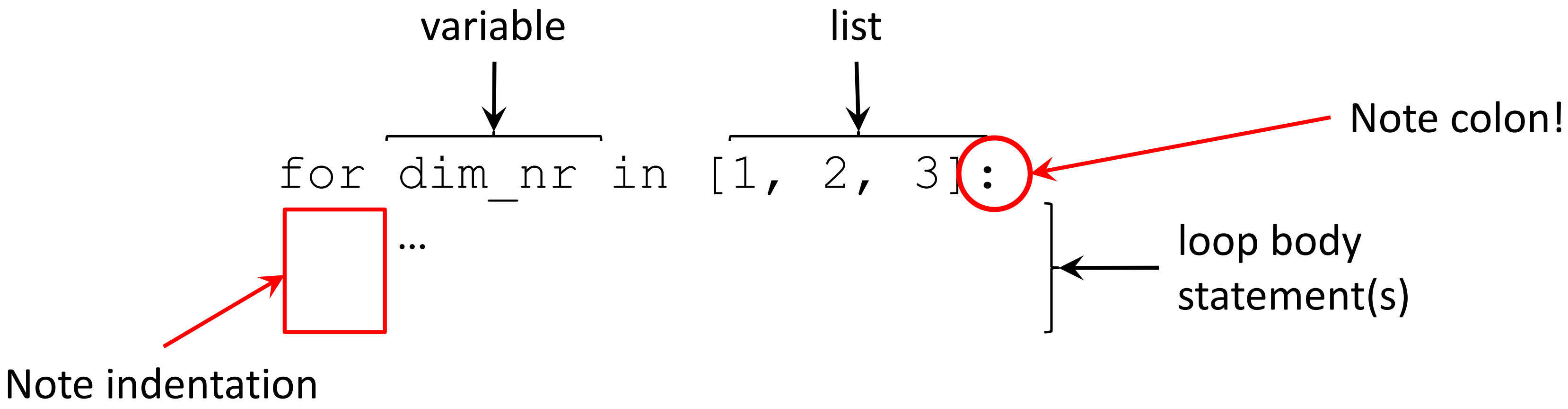
- Need some data?
  - first column, case number: sequential number
  - second column, dimension number: integer 1, 2, 3
  - third column, temperature: float value -0.5, 0.0, 0.5

```
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…



variable

list

Note colon!

```
for dim_nr in [1, 2, 3]:
    …
```

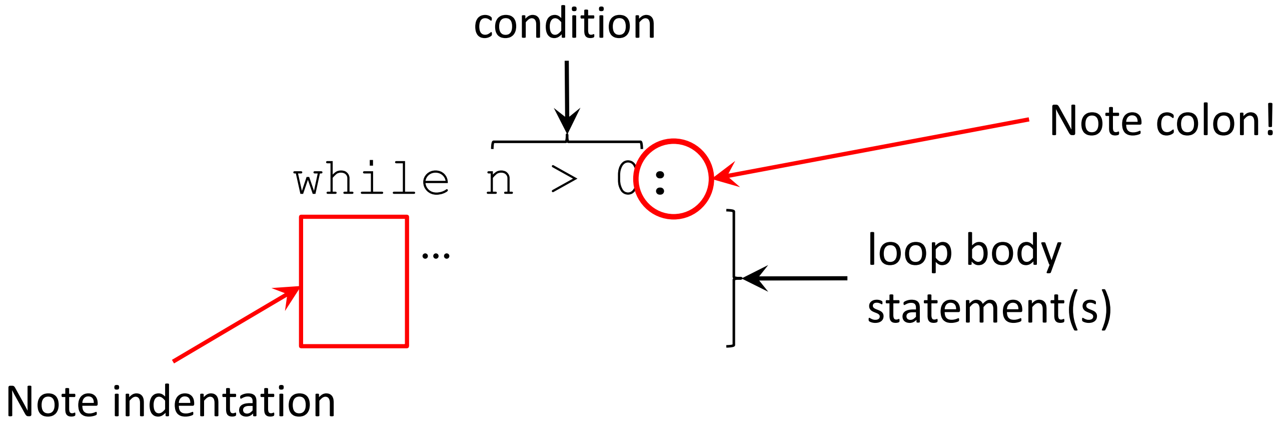
loop body statement(s)

Note indentation

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

`while` loop

- Semantics: while boolean condition holds do…



condition

Note colon!

```
while n > 0:
    …
```

loop body statement(s)

Note indentation

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` 3.6+
- `float`: floating point number, e.g., `-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)
```

- `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 for x in range(-1, 2)]`
  - list comprehensions: construct list from list

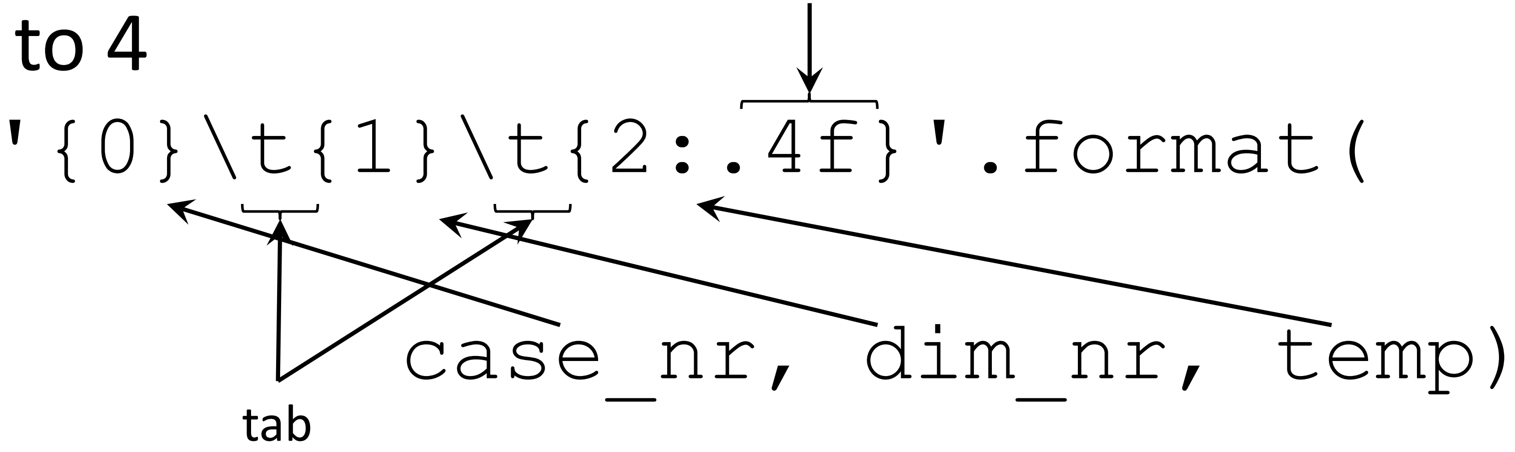
looks a lot like math

```
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)
```

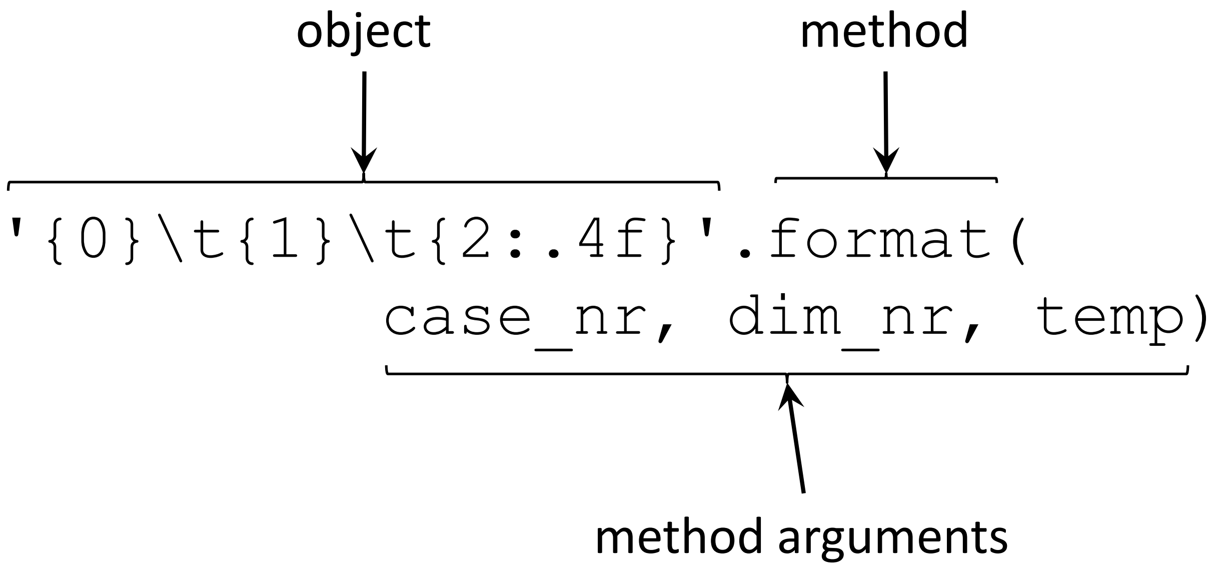
- f-string:

```
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



object

method

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

method arguments

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