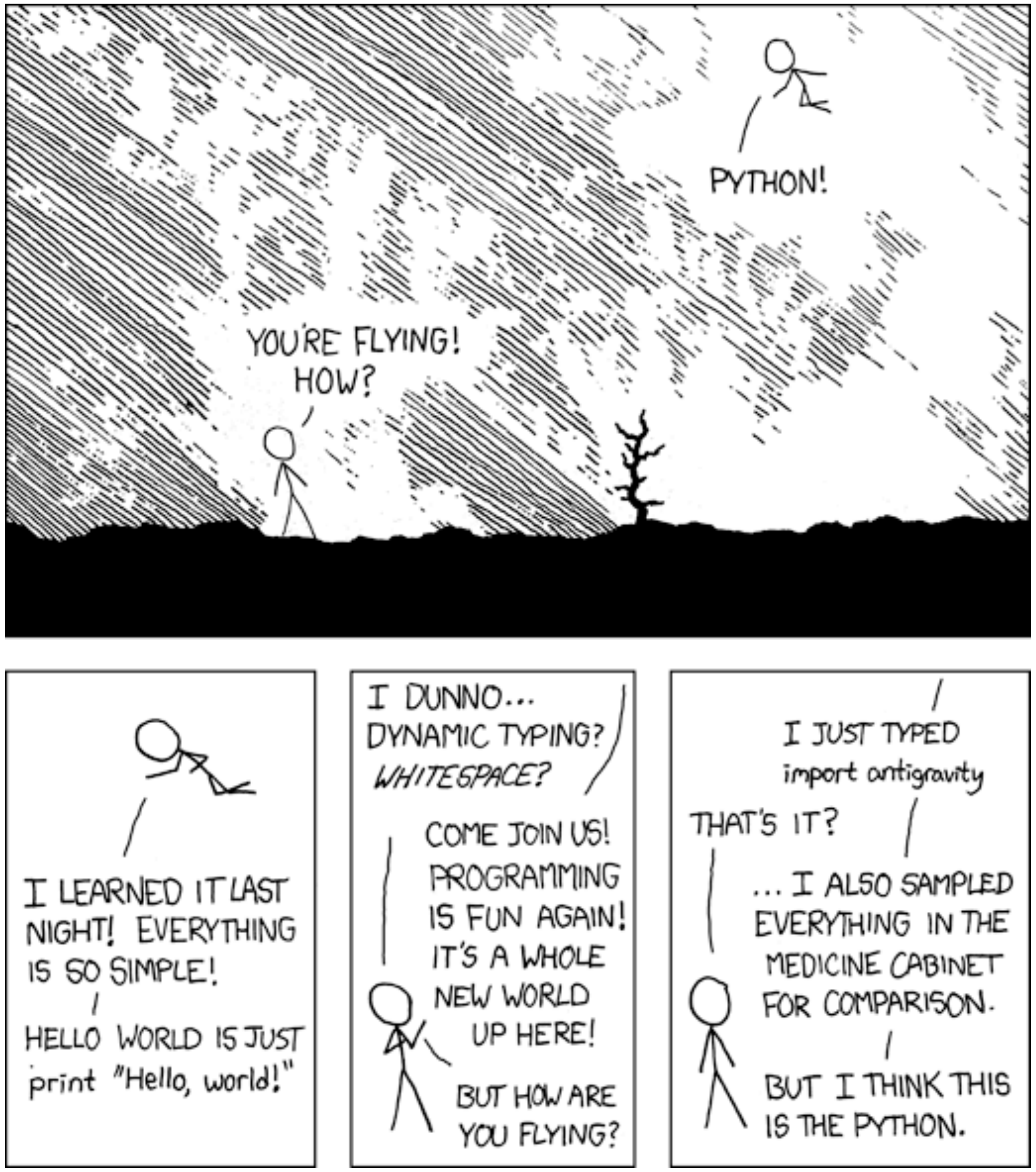
Introduction to Python

GRK / RTG 2753; Methods Academy

Jan Willem de Gee
(j.w.degee@uva.nl)


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

In terminal:

```
cd directory/

git pull https://github.com/jwdegee/intro2python.git

conda create -n intro

conda activate intro

conda install seaborn
```
(installs dependencies: numpy, scipy, matplotlib, pandas)

```
conda install jupyter

jupyter notebook
```

OUTLINE

Today

10:00 - 12:00
L1: Pure Python

12:00 - 13:00
Lunch

13:00 - 16:00
P1: Solve riddle in pure Python

Tomorrow

10:00 - 12:00
L2: Intro to numpy/scipy/matplotlib/pandas

12:00 - 13:00
Lunch

13:00 - 16:00
P2: Behavioral data analysis (in pandas)

Wednesday

09:00 - 10:00
L3: Write an installable program

10:00 - 13:00
P3: Pupil preprocessing

During the practicals: ASK QUESTIONS!

Why Python?

Easy syntax

Readability

High-level language

Object oriented

Cross-platform

Free + open source

“Batteries included”

Widely supported

Used by industry

python™

IP[y]: IPython Interactive Computing

jupyter

NumPy

SciPy.org

matplotlib

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

SM StatsModels Statistics in Python

PyMC

pingouin

scikit learn

PyTorch

TensorFlow

MNE

MEG + EEG ANALYSIS & VISUALIZATION

ni learn

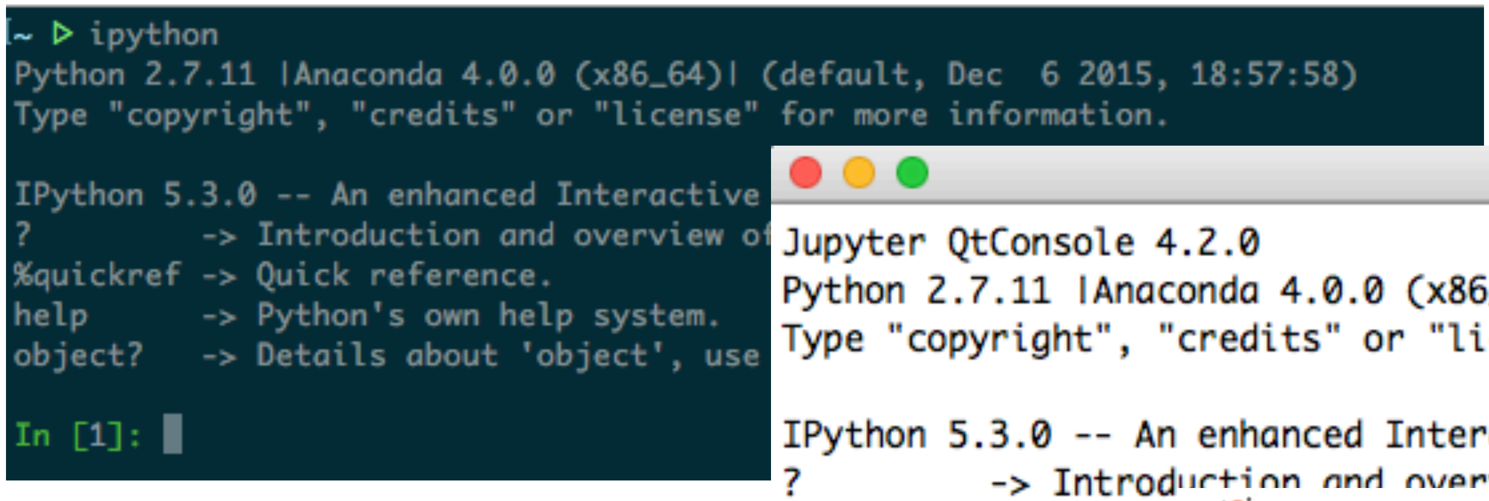
DeepLabCut™: a software package for animal pose estimation

PsychoPy

Psychology software in Python

Suite2p

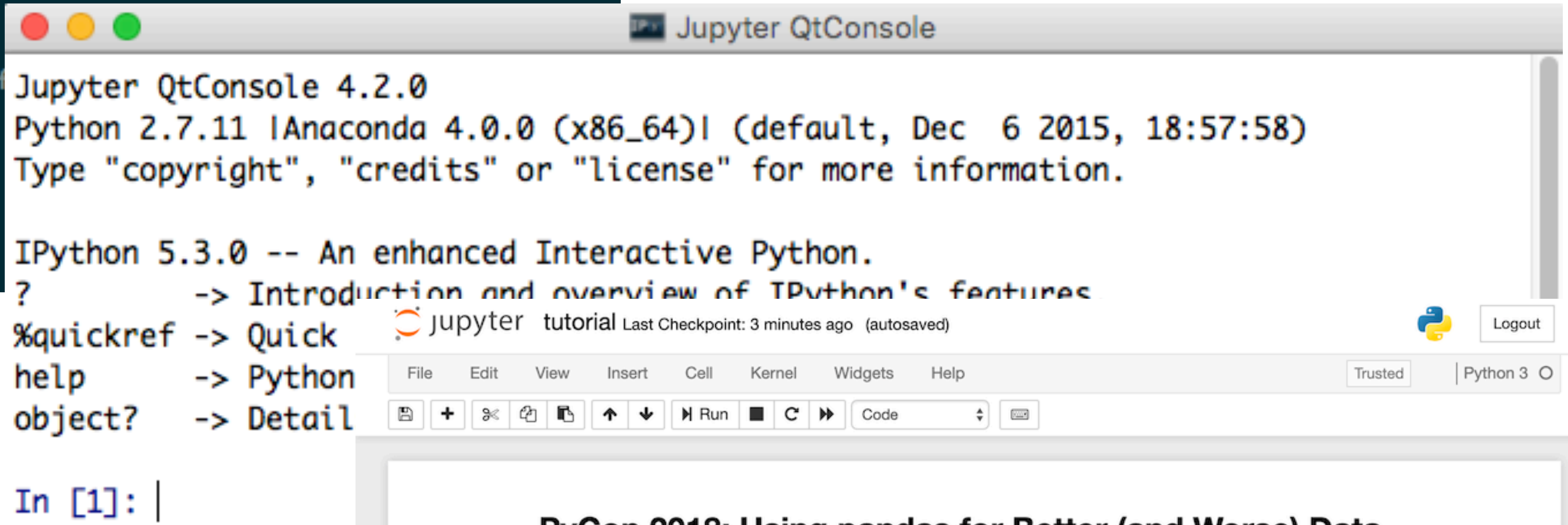
python™ IP[y]: IPython Interactive Computing jupyter



```
~ ▷ ipython
Python 2.7.11 |Anaconda 4.0.0 (x86_64)| (default, Dec  6 2015, 18:57:58)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive
?         -> Introduction and overview of
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use

In [1]:
```

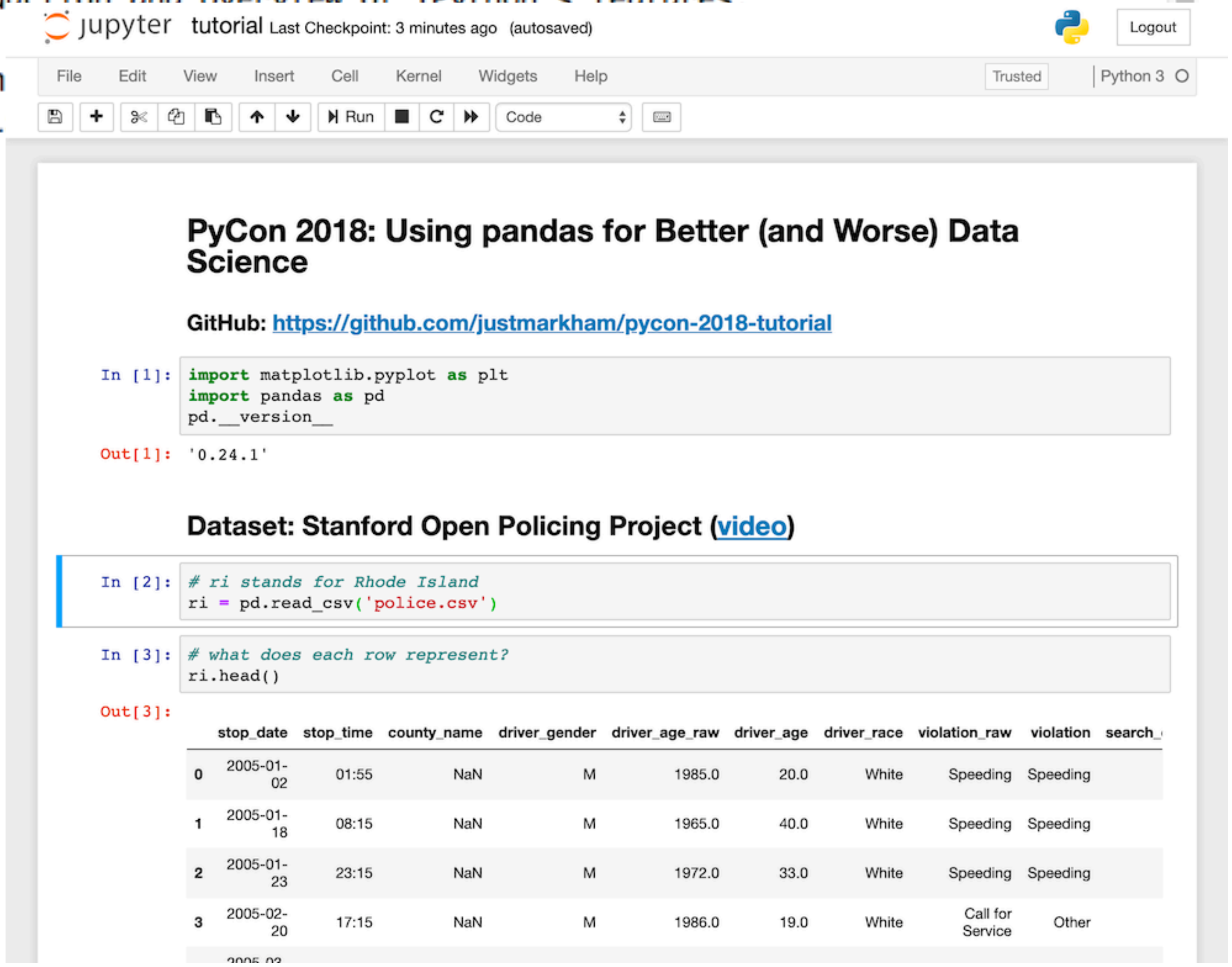


Jupyter QtConsole

```
Jupyter QtConsole 4.2.0
Python 2.7.11 |Anaconda 4.0.0 (x86_64)| (default, Dec  6 2015, 18:57:58)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features
%quickref -> Quick
help      -> Python
object?   -> Detail

In [1]:
```



jupyter tutorial Last Checkpoint: 3 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

PyCon 2018: Using pandas for Better (and Worse) Data Science

GitHub: https://github.com/justmarkham/pycon-2018-tutorial

```
In [1]: import matplotlib.pyplot as plt
        import pandas as pd
        pd.__version__

Out[1]: '0.24.1'
```

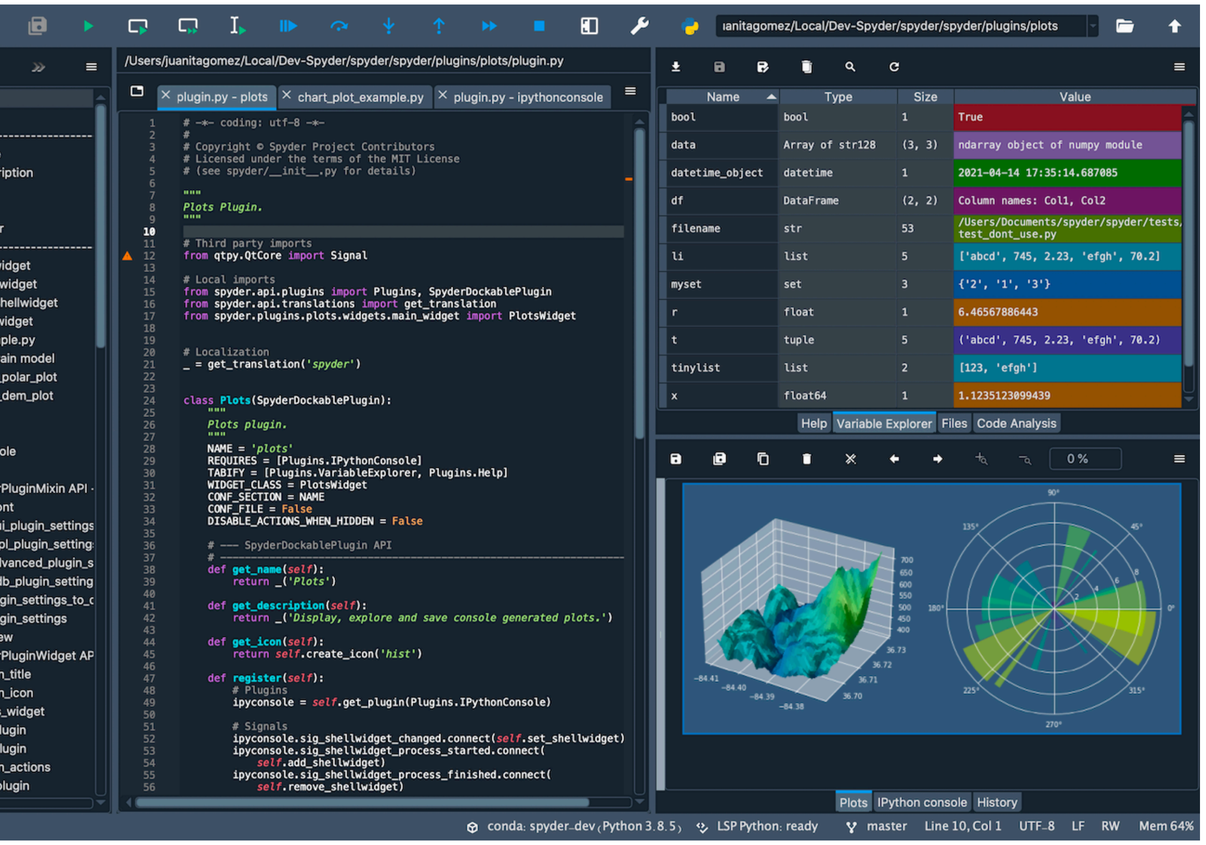
Dataset: Stanford Open Policing Project (video)

```
In [2]: # ri stands for Rhode Island
        ri = pd.read_csv('police.csv')

In [3]: # what does each row represent?
        ri.head()

Out[3]:
```

	stop_date	stop_time	county_name	driver_gender	driver_age_raw	driver_age	driver_race	violation_raw	violation	search_
0	2005-01-02	01:55	NaN	M	1985.0	20.0	White	Speeding	Speeding	
1	2005-01-18	08:15	NaN	M	1965.0	40.0	White	Speeding	Speeding	
2	2005-01-23	23:15	NaN	M	1972.0	33.0	White	Speeding	Speeding	
3	2005-02-20	17:15	NaN	M	1986.0	19.0	White	Call for Service	Other	



ıanitagomez/Local/Dev-Spyder/spyder/spyder/plugins/plots
/Users/juanitagomez/Local/Dev-Spyder/spyder/spyder/plugins/plots/plugin.py
plugin.py - plots
chart_plot_example.py
plugin.py - ipythonconsole

```
# -*- coding: utf-8 -*-
#
# Copyright © Spyder Project Contributors
# Licensed under the terms of the MIT License
# (see spyder/__init__.py for details)

"""
Plots Plugin.
"""

# Third party imports
from qtpy.QtCore import Signal

# Local imports
from spyder.api.plugins import Plugins, SpyderDockablePlugin
from spyder.api.translations import get_translation
from spyder.plugins.plots.widgets.main_widget import PlotsWidget


# Localization
_ = get_translation('spyder')


class Plots(SpyderDockablePlugin):
    """
    Plots plugin.
    """
    NAME = 'plots'
    REQUIRES = [Plugins.IPythonConsole]
    TABIFY = [Plugins.VariableExplorer, Plugins.Help]
    WIDGET_CLASS = PlotsWidget
    CONF_SECTION = NAME
    CONF_FILE = False
    DISABLE_ACTIONS_WHEN_HIDDEN = False

    # --- SpyderDockablePlugin API
    # ------------------------------------------------------------------
    def get_name(self):
        return _('Plots')

    def get_description(self):
        return _('Display, explore and save console generated plots.')

    def get_icon(self):
        return self.create_icon('hist')

    def register(self):
        # Plugins
        ipyconsole = self.get_plugin(Plugins.IPythonConsole)

        # Signals
        ipyconsole.sig_shellwidget_changed.connect(self.set_shellwidget)
        ipyconsole.sig_shellwidget_process_started.connect(
            self.add_shellwidget)
        ipyconsole.sig_shellwidget_process_finished.connect(
            self.remove_shellwidget)
```

Name	Type	Size	Value
bool	bool	1	True
data	Array of str128	(3, 3)	ndarray object of numpy module
datetime_object	datetime	1	2021-04-14 17:35:14.687085
df	DataFrame	(2, 2)	Column names: Col1, Col2
filename	str	53	/Users/Documents/spyder/spyder/tests/test_dont_use.py
li	list	5	['abcd', 745, 2.23, 'efgh', 70.2]
myset	set	3	{'2', '1', '3'}
r	float	1	6.46567886443
t	tuple	5	('abcd', 745, 2.23, 'efgh', 70.2)
tinylist	list	2	[123, 'efgh']
x	float64	1	1.1235123099439

```
conda: spyder-dev (Python 3.8.5)   LSP Python: ready   master   Line 10, Col 1   UTF-8   LF   RW   Mem 64%
```

Dynamic typing

No need to declare variable types.
But: Python keeps track of types.
Need explicit casts
(e.g. int() or str())

Code Indentation

No {} or end statements. Code is grouped by indentation. Use 4 spaces and no tabs.

Operators

```
= assignment
== comparison
+-*/ math
```

Control flow

```
if statement:
    foo()
elif statement:
    bar()
else:
    foobar()
```

Loops

```
for i in collection:
    foobar()

while statement:
    foobar()
```

(remember continue and break)

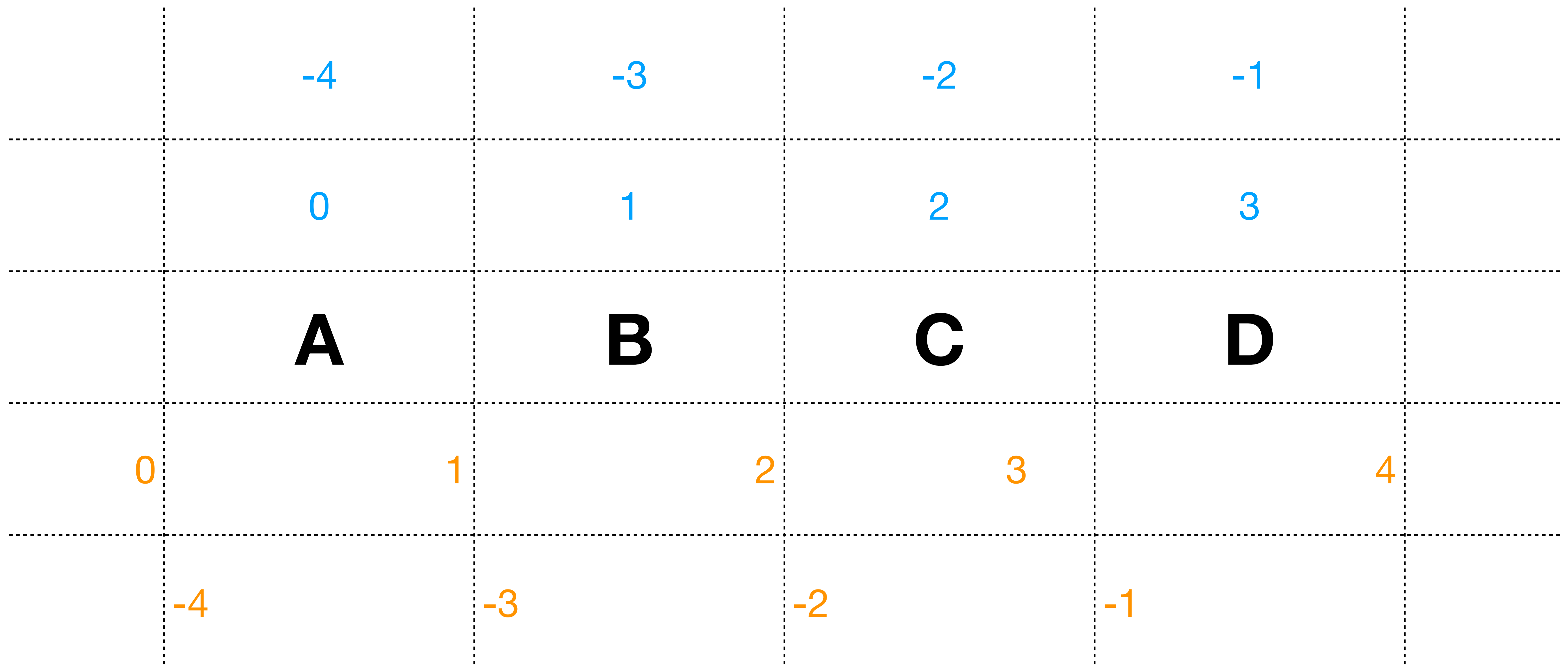
Errors

```
try:
    foo()
except Exception as e:
    #fix error
finally:
    #cleanup
```

Miscellaneous

```
# Comments
“““Multi line strings“““
```

Indexing & Slicing



lst[0] = ‚A‘ lst[-3] = ‚B‘ lst[2:] = [‚C‘, ‚D‘] lst[-1:] = ‚D‘

lst[::2] = [‚A‘, ‚C‘] lst[::-2] = [‚D‘, ‚B‘] lst[-1:-3] = []

Comprehensions

[statement loop conditional]

```
lst = []
for y in range(10):
    for x in range(10):
        if x > y:
            lst.append(‚#‘)
        else:
            lst.append(‚.‘)
```

```
lst = []
for x in range(10):
  if x > 5:
     lst.append(‚#‘)
```

$\downarrow$

```
['#' for x in range(10) if x > 5]
```

$\downarrow$

```
[['#' if x > y else '.' for x in range(10)] for y in range(10)]
```

Indexing

Starts at 0. Think interval. negative indices start from the end

Slicing

Slice and step through lists

Unpacking

Containers can be unpacked into variables:

```
a,b = [1,2]
```

References!

```
a = b = [1, 2]
a.extend([5])
b != [1,2]
```

Tuples

Immutable lists.

Dictionaries

Easy key-value store. (The thing that Matlab users didn’t know they were missing)

Comprehensions

Compact statement of simple for loops. Not shown: set comprehensions.

Anatomy of function!

```
def name(argument):
    """
    Doc string
    """
    body
    return value
```

Anatomy of function!

```
def name(a, *args, b=1, **kw):
    '''
    Doc string
    '''
    body
    return value1, value2, …
```

Return values

return a, b, c -> tuple
omitting a return = return None

Tuple unpacking

a, b, c = (1,2,3)
func(*(1,2,3), {‚a‘:1})

Keyword arguments

def func(a=1, b=2, c=3)

Variable #arguments

def func(a, *args, b=1, **kw):

Functions are objects!

Have properties and can be assigned to other variables.

lambdas

Simple functions that map statement to output.

Decorators

Replace a function with a function that takes original function as input.
Logging and Memoize/caching.

Task one

The riddle: 50 prisoners are in solitary cells, unable to see, speak or communicate in any way from those solitary cells with each other. There's a central living room with one light bulb; the bulb is initially off. No prisoner can see the light bulb from his own cell. Everyday, the warden picks a prisoner at random, and that prisoner goes to the central living room. While there, the prisoner can toggle the bulb if he or she wishes. Also, the prisoner has the option of asserting the claim that all 50 prisoners have been to the living room. If this assertion is false (that is, some prisoners still haven't been to the living room), all 50 prisoners will be shot for their stupidity. However, if it is indeed true, all prisoners are set free. Thus, the assertion should only be made if the prisoner is 100% certain of its validity.

Before the random picking begins, the prisoners are allowed to get together to discuss a plan. So - what plan should they agree on, so that eventually, someone will make a correct assertion?

Question: How can the prisoners tell, with certainty, that all 50 of them have visited the central living room with the light bulb?

Task: Once you've decided on a strategy simulate how many turns the prisoners will have to take.

```
from random import choice # Choose a random number

def prisonser(N=50):
    """
    Computes how many turns the prisoners need before being freed.

    Arguments:
        N : int, default=50

    Returns:
        The number of turns required by the prisoners.

    """

    # Implement your solution here

    return turns


nr_prisoners = 100
turns = prisonser(N=nr_prisoners)
print( "total turns (days) required: {}".format(turns))
```

OUTLINE

Today

10:00 - 12:00
L1: Pure Python

12:00 - 13:00
Lunch

13:00 - 16:00
P1: Solve riddle in pure Python

Tomorrow

10:00 - 12:00
L2: Intro to numpy/scipy/matplotlib/pandas

12:00 - 13:00
Lunch

13:00 - 16:00
P2: Behavioral data analysis (in pandas)

Wednesday

09:00 - 10:00
L3: Write an installable program

10:00 - 13:00
P3: Pupil preprocessing

During the practicals: ASK QUESTIONS!