University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Architecture of large projects in bioinformatics (ADP)

Lecture 04

Łukasz P. Kozłowski

Warsaw, 2024

lukaskoz@mimuw.edu.pl

Outline

1. Data formats in bioinformatics,
2. Popular software libraries (BioPerl, BioPython)
3. Most important bioinformatics databases (UniProt, PDB, RefSeq, GenBank, ENA, InterPro, etc.)
4. Software licensing for scientific purposes. Free-software licensing. Patents.

5. Generic model Organism database (GMOD) project - assumptions, history and usage

6. Genome browsers, problem description and state of the solutions

7. Version control systems (CVS, SVN, git), and online collaboration ad distribution platforms (github, sourceforge).

8. Software testing, automated testing frameworks.

9. Scientific workflow systems - taverna and galaxy. MyExperiment platform. Reproducible research.

10. Literate programming idea and sweave, markdown, software documentation.

11. Interactive scripting platforms, Rstudio, Jupyter.

Outline

1. Data formats in bioinformatics,
2. Popular software libraries (BioPerl, BioPython)
3. Most important bioinformatics databases (UniProt, PDB, RefSeq, GenBank, ENA, InterPro, etc.)
4. Software licensing for scientific purposes. Free-software licensing. Patents.

5. Generic model Organism database (GMOD) project - assumptions, history and usage

6. Genome browsers, problem description and state of the solutions

7. Version control systems (CVS, SVN, git), and online collaboration ad distribution platforms (github, sourceforge).

8. Software testing, automated testing frameworks.

9. Scientific workflow systems - taverna and galaxy. MyExperiment platform. Reproducible research.

10. Literate programming idea and sweave, markdown, software documentation.

11. Interactive scripting platforms, Rstudio, Jupyter.

Outline

1. Data formats in bioinformatics,
2. Popular software libraries (BioPerl, BioPython)
3. Most important bioinformatics databases (UniProt, PDB, RefSeq, GenBank, ENA, InterPro, etc.)
4. Software licensing for scientific purposes. Free-software licensing. Patents.

5. Generic model Organism database (GMOD) project - assumptions, history and usage

6. Genome browsers, problem description and state of the solutions

7. Version control systems (CVS, SVN, git), and online collaboration ad distribution platforms (github, sourceforge).

8. Software testing, automated testing frameworks.

9. Scientific workflow systems - taverna and galaxy. MyExperiment platform. Reproducible research.

10. Literate programming idea and sweave, markdown, software documentation.

11. Interactive scripting platforms, Rstudio, Jupyter.

1. Data formats in bioinformatics,
2. Popular software libraries (BioPerl, BioPython)
3. Most important bioinformatics databases (UniProt, PDB, RefSeq, GenBank, ENA, InterPro, etc.)
4. Software licensing for scientific purposes. Free-software licensing. Patents.

5. Generic model Organism database (GMOD) project - assumptions, history and usage

6. Genome browsers, problem description and state of the solutions

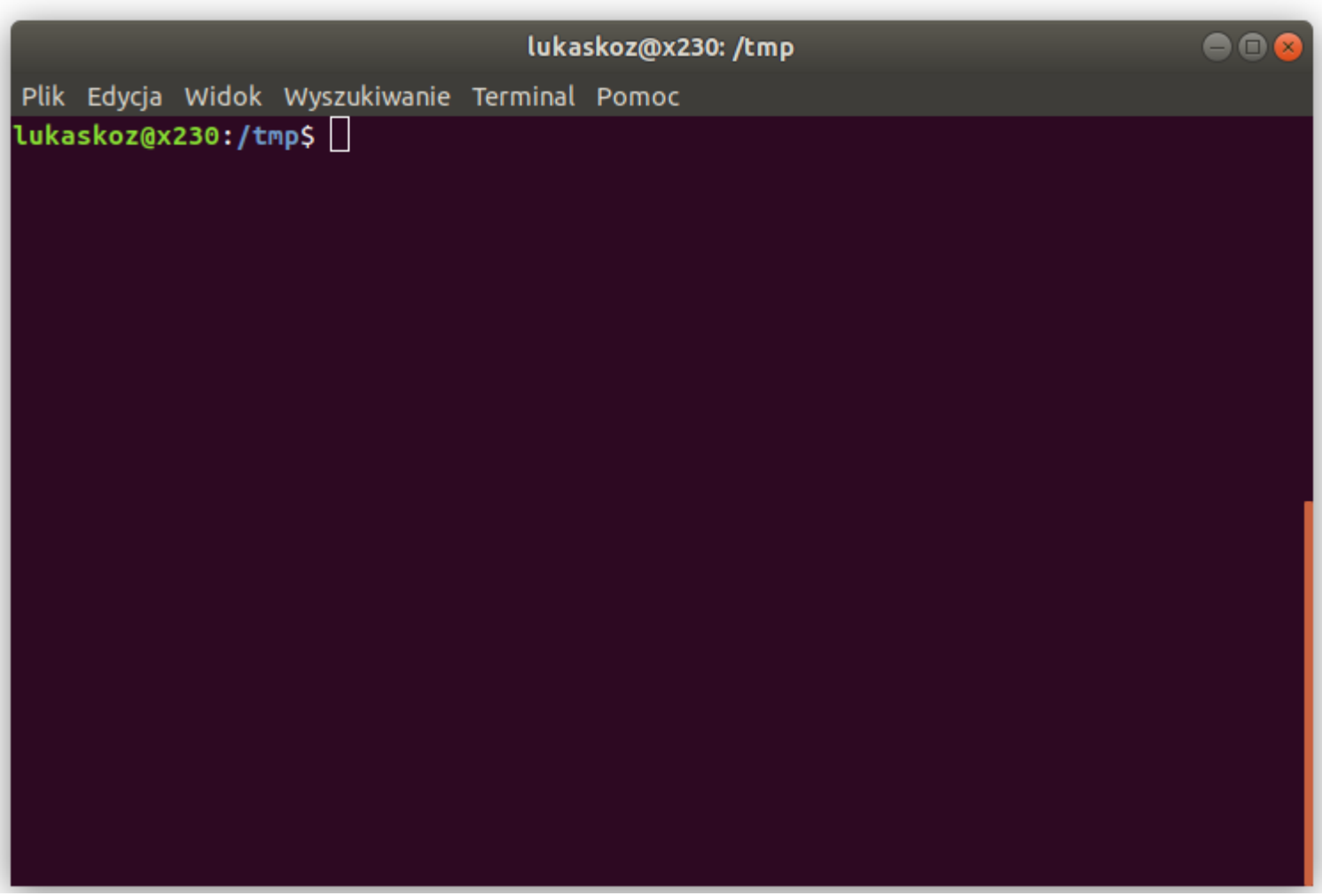
7. Version control systems (CVS, SVN, git), and online collaboration ad distribution platforms (github, sourceforge).

8. Software testing, automated testing frameworks.

9. Scientific workflow systems - taverna and galaxy. MyExperiment platform. Reproducible research.

10. Literate programming idea and sweave, markdown, software documentation.

11. Interactive scripting platforms, Rstudio, Jupyter.



We do as much as possible in PYTHON

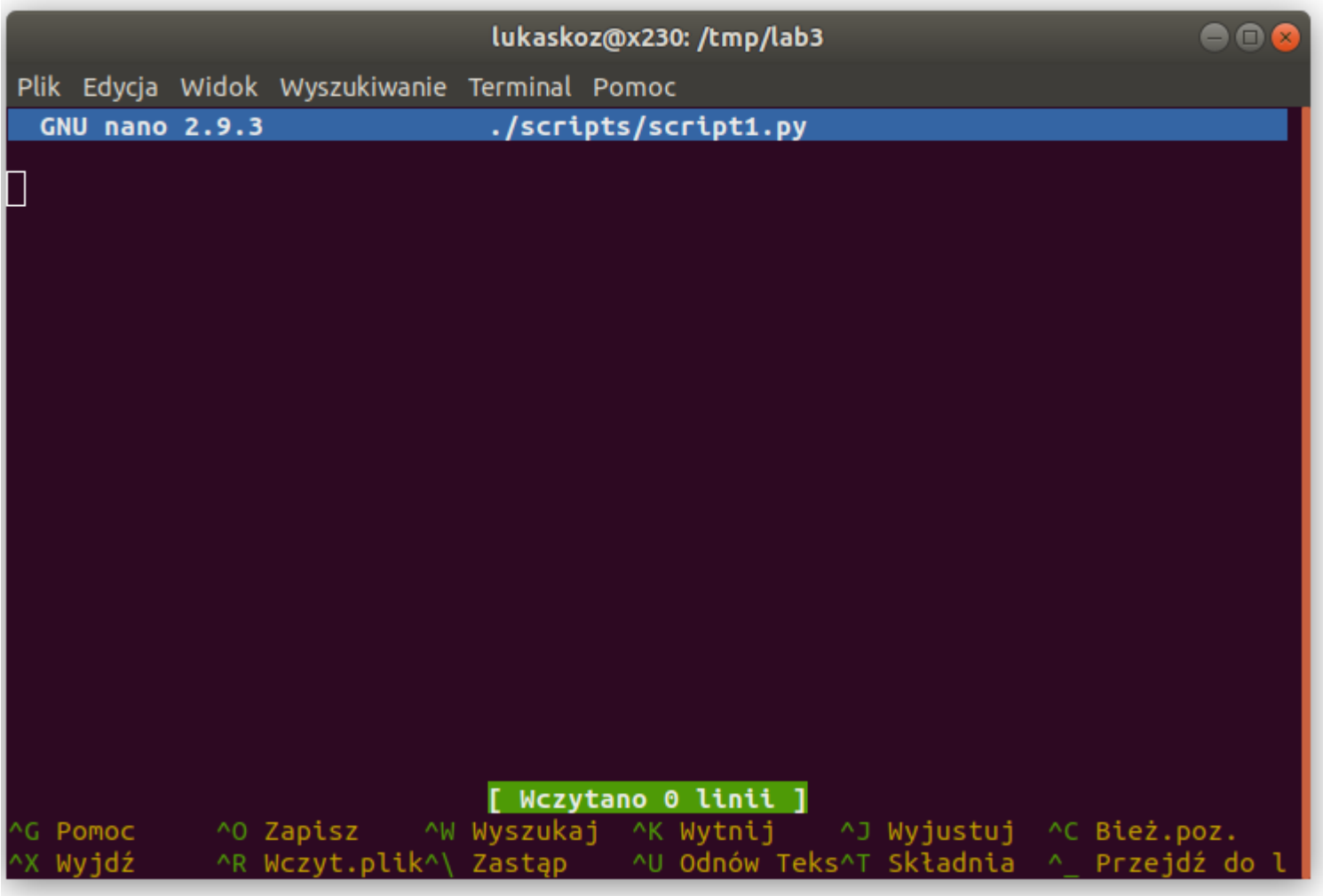
The PYTHON works from command line (CLI) very well
(Terminal/Console is also installed everywhere)

```
lukaskoz@x230: /tmp/lab3
Plik  Edycja  Widok  Wyszukiwanie  Terminal  Pomoc
lukaskoz@x230:/tmp$ mkdir lab3
lukaskoz@x230:/tmp$ cd lab3/
lukaskoz@x230:/tmp/lab3$ mkdir data
lukaskoz@x230:/tmp/lab3$ mkdir images
lukaskoz@x230:/tmp/lab3$ mkdir scripts
lukaskoz@x230:/tmp/lab3$ cd scripts/
lukaskoz@x230:/tmp/lab3/scripts$ touch script1.py
lukaskoz@x230:/tmp/lab3/scripts$ cd ..
lukaskoz@x230:/tmp/lab3$ tree
.
├── data
├── images
└── scripts
    └── script1.py

3 directories, 1 file
lukaskoz@x230:/tmp/lab3$ nano ./scripts/script1.py
```

We do as much as possible in PYTHON

The PYTHON works from command line (CLI) very well
(Terminal/Console is also installed everywhere)



We do as much as possible in PYTHON

The PYTHON works from command line (CLI) very well
(Terminal/Console is also installed everywhere)

nano
joe
gedit
kate
mcedit
…

vi

Emacs

nano
joe
gedit
kate
mcedit
…

Sublime Text

PyCharm

SPYDER

eclipse

eric

Visual Studio

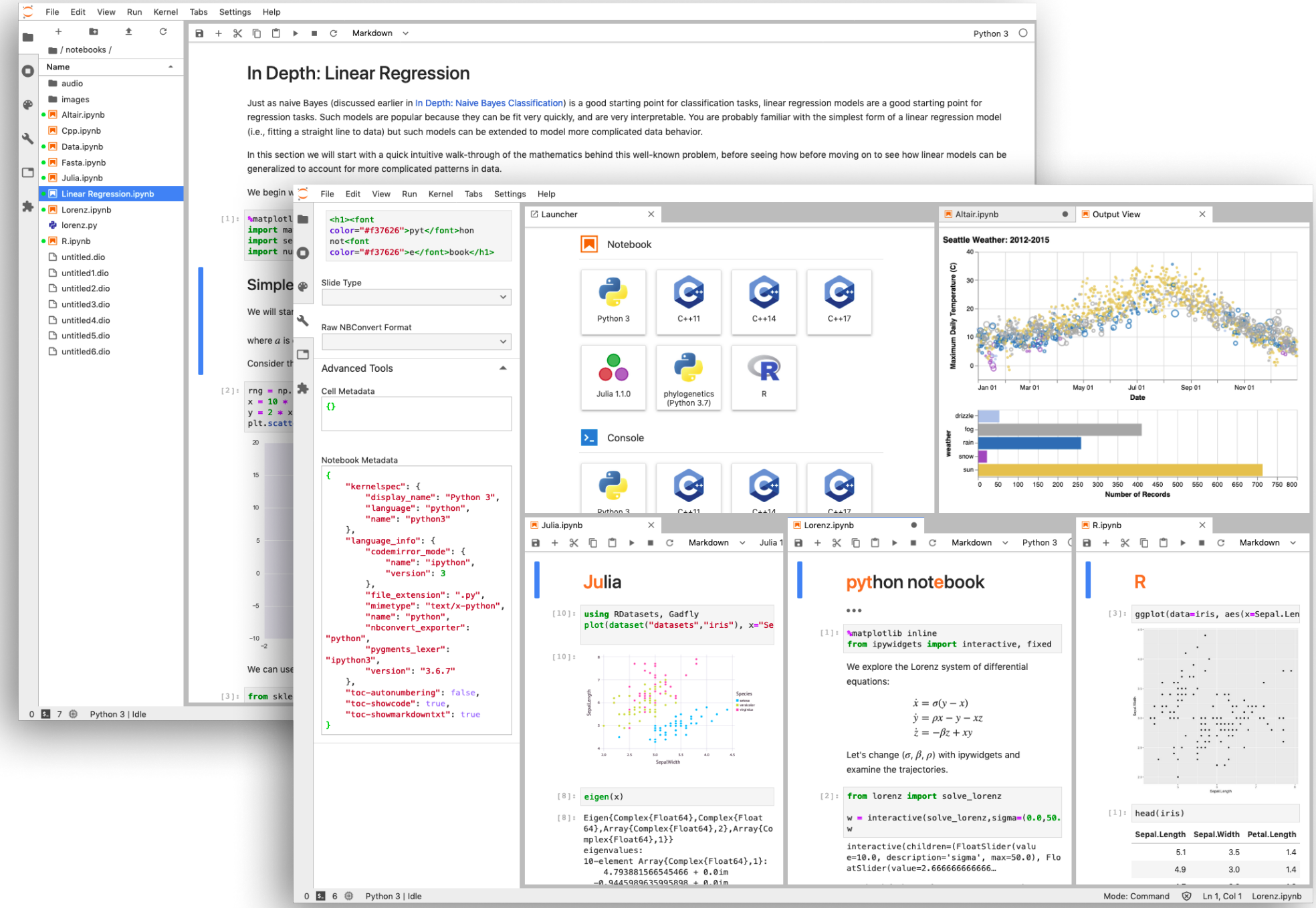
Notepad++

PyDev

jupyter

Interactive scripting platforms, Rstudio, Jupyter

ADP



IPy: IPython
Interactive Computing

A kernel for Jupyter

IPy: IPython
Interactive Computing

IPython: home/lukaskoz

```
lukaskoz@X230:~$ ipython3
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import os

In [2]: os.getcwd
```

```
abc                 environb            F_LOCK              getegid             linesep
abort               error               F_OK                getenv              link
access              EX_CANTCREAT        F_TEST              getenvb             listdir
altsep              EX_CONFIG           F_TLOCK             geteuid             listxattr
chdir               EX_DATAERR          F_ULOCK             getgid              lockf
chmod               EX_IOERR            fchdir              getgrouplist        lseek
chown               EX_NOHOST           fchmod              getgroups           lstat
chroot              EX_NOINPUT          fchown              getloadavg          major
CLD_CONTINUED       EX_NOPERM           fdatasync           getlogin            makedev
CLD_DUMPED          EX_NOUSER           fdopen              getpgid             makedirs
CLD_EXITED          EX_OK               fork                getpgrp             memfd_create
CLD_TRAPPED         EX_OSERR            forkpty             getpid              MFD_ALLOW_SEALING
close               EX_OSFILE           fpathconf           getppid             MFD_CLOEXEC
closerange          EX_PROTOCOL         fsdecode            getpriority         MFD_HUGE_16GB
confstr             EX_SOFTWARE         fsencode            getrandom           MFD_HUGE_16MB
confstr_names       EX_TEMPFAIL         fspath              getresgid           MFD_HUGE_1GB
copy_file_range     EX_UNAVAILABLE      fstat               getresuid           MFD_HUGE_1MB
cpu_count           EX_USAGE            fstatvfs            getsid              MFD_HUGE_256MB
ctermid             execl               fsync               getuid              MFD_HUGE_2GB
curdir              execle              ftruncate           getxattr            MFD_HUGE_2MB
defpath             execlp              fwalk               GRND_NONBLOCK       MFD_HUGE_32MB
device_encoding     execlpe             get_blocking        GRND_RANDOM         MFD_HUGE_512KB
devnull             execv               get_exec_path       initgroups          MFD_HUGE_512MB
DirEntry            execve              get_inheritable     isatty              MFD_HUGE_64KB
dup                 execvp              get_terminal_size   kill                MFD_HUGE_8MB
dup2                execvpe             getcwd              killpg              MFD_HUGE_MASK
environ             extsep              getcwdb             lchown              MFD_HUGE_SHIFT
<unknown>
```

Language of choice

Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala.

Language of choice

Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala.

Share notebooks

Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.

Language of choice

Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala.

Share notebooks

Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.

Interactive output

Your code can produce rich, interactive output: HTML, images, videos, LaTeX, and custom MIME types.

Language of choice

Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala.

Share notebooks

Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.

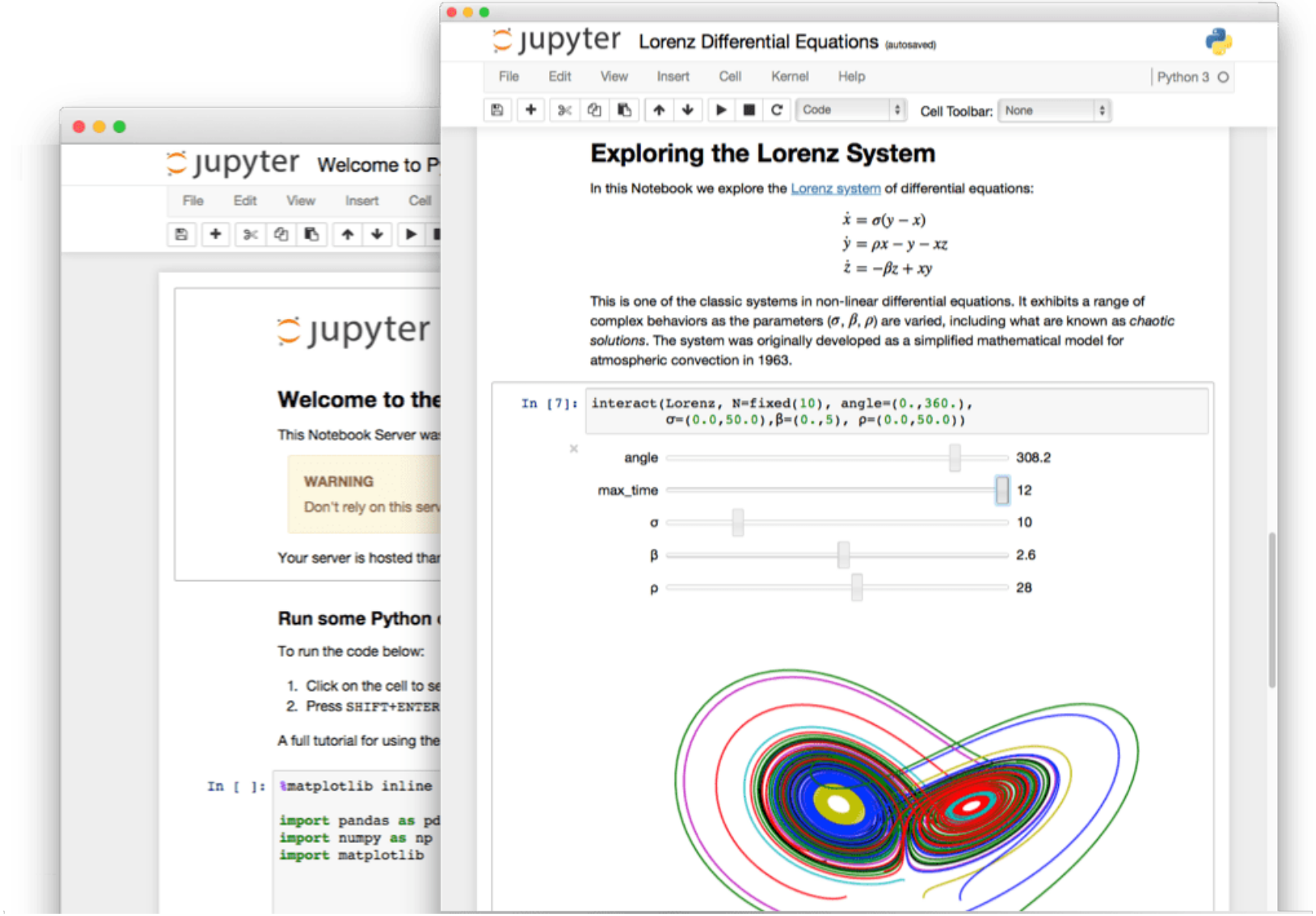
Interactive output

Your code can produce rich, interactive output: HTML, images, videos, LaTeX, and custom MIME types.

Big data integration

Leverage big data tools, such as Apache Spark, from Python, R and Scala. Explore that same data with pandas, scikit-learn, ggplot2, TensorFlow.

Jupyter Notebooks



try Jupyter online

https://jupyter.org/try

jupyter

Install About Us Community Documentation NBViewer JupyterHub

Try Jupyter

You can try Jupyter out right now, without installing anything. Select an example below and you will get a temporary Jupyter server just for you, running on mybinder.org. If you like it, you can install Jupyter yourself.

Try Classic Notebook

A tutorial introducing basic features of Jupyter notebooks and the IPython kernel using the classic Jupyter Notebook interface.

Try JupyterLab

JupyterLab is the new interface for Jupyter notebooks and is ready for general use. Give it a try!

Try Jupyter with Julia

A basic example of using Jupyter with Julia.

try Jupyter online

https://mybinder.org/v2/gh/ipython/ipython-in-depth/master?filep

Error loading ipython/ipython-in-depth/master!
See logs below for details.

Take a look at the full list of configuration files supported by repo2docker.

Build logs hide

```
Found built image, launching...
Too many users running https://github.com/ipython/ipython-in-depth! Try again soon.
```

Getting started with the classic Jupyter Notebook

conda

We recommend installing the classic Jupyter Notebook using the conda package manager. Either the miniconda or the miniforge conda distributions include a minimal conda installation.

Then you can install the notebook with:

```
conda install -c conda-forge notebook
```

pip

If you use `pip`, you can install it with:

```
pip install notebook
```

Congratulations, you have installed Jupyter Notebook! To run the notebook, run the following command at the Terminal (Mac/Linux) or Command Prompt (Windows):

```
jupyter notebook
```

Getting started with JupyterLab

The installation guide contains more detailed instructions

Install with conda

If you use `conda`, you can install it with:

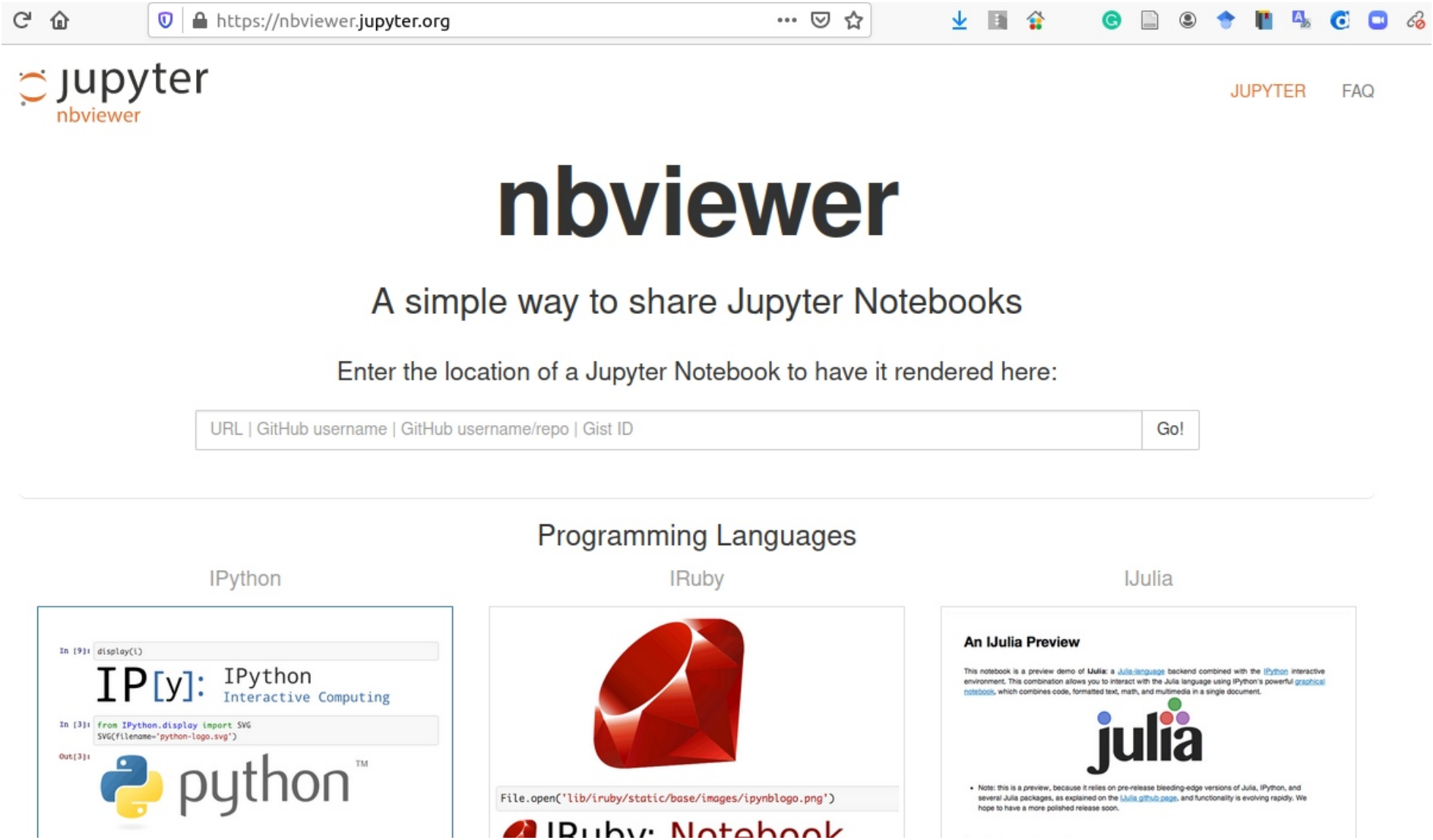
```
conda install -c conda-forge jupyterlab
```

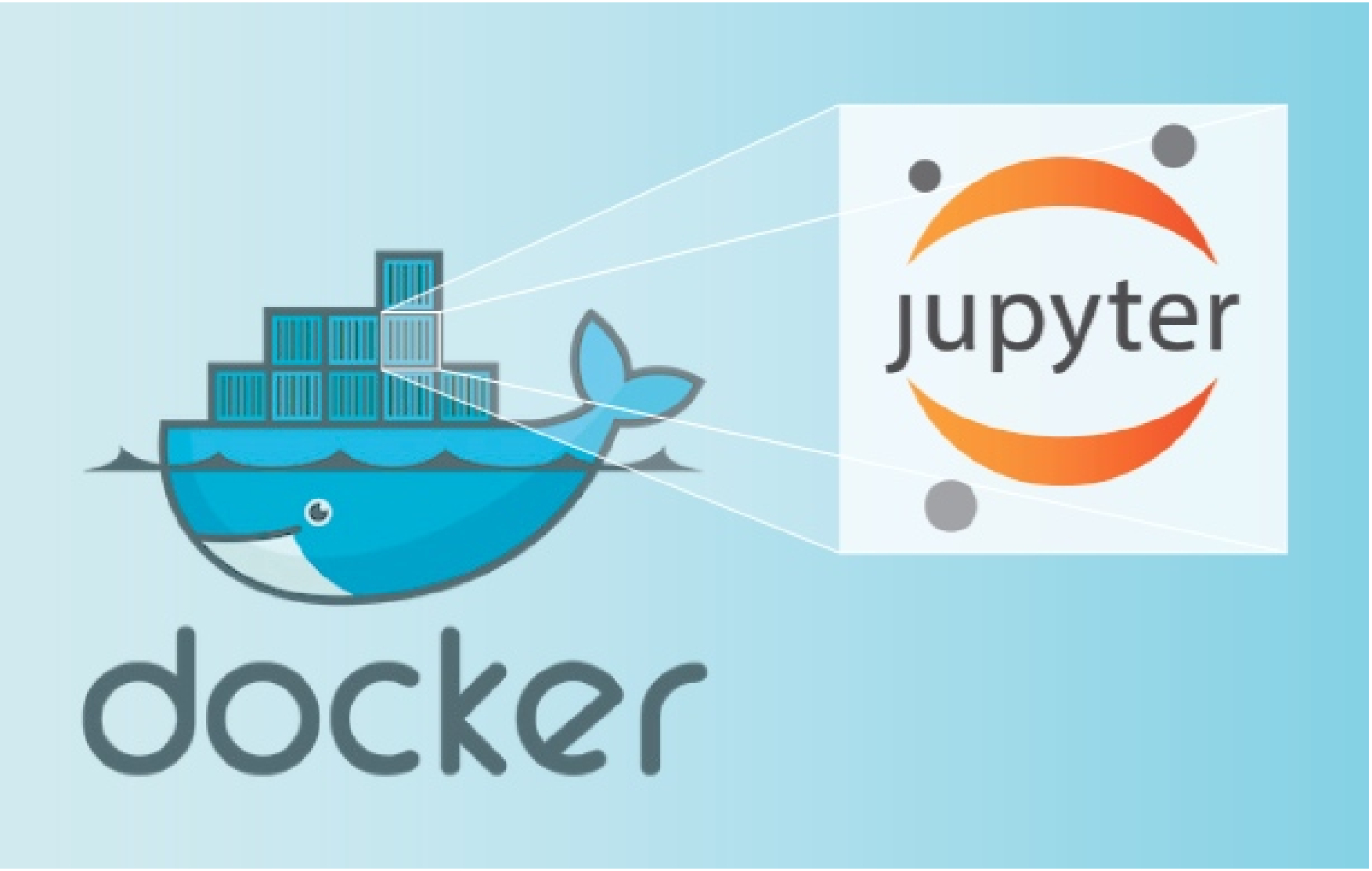
Install with pip

If you use `pip`, you can install it with:

```
pip install jupyterlab
```

If installing using `pip install --user`, you must add the user-level `bin` directory to your `PATH` environment variable in order to launch `jupyter lab`. If you are using a Unix derivative (FreeBSD, GNU / Linux, OS X), you can achieve this by using `export PATH="$HOME/.local/bin:$PATH"` command.


https://nbviewer.jupyter.org
jupyter
nbviewer
JUPYTER
FAQ
nbviewer
A simple way to share Jupyter Notebooks
Enter the location of a Jupyter Notebook to have it rendered here:
URL | GitHub username | GitHub username/repo | Gist ID
Go!
Programming Languages
IPython
IRuby
IJulia
IPython
Interactive Computing
python
File.open('lib/iruby/static/base/images/ipynblogo.png')
An IJulia Preview
julia


jupyter
docker

Simple spectral analysis

An illustration of the Discrete Fourier Transform using windowing, to reveal the frequency content of a sound signal.

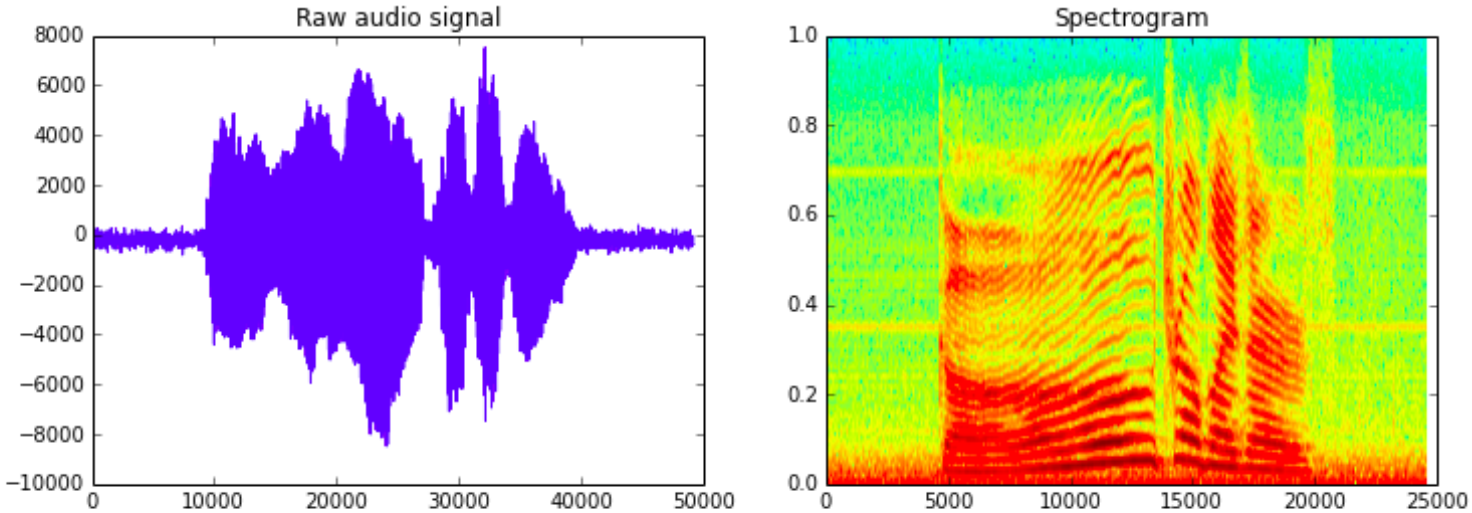
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

We begin by loading a datafile using SciPy's audio file support:

In [1]:
```
from scipy.io import wavfile
rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin `specgram` routine:

In [2]:
```
%matplotlib inline
from matplotlib import pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(x); ax1.set_title('Raw audio signal')
ax2.specgram(x); ax2.set_title('Spectrogram');
```



Simple spectral analysis

An illustration of the Discrete Fourier Transform using windowing, to reveal the frequency content of a sound signal.

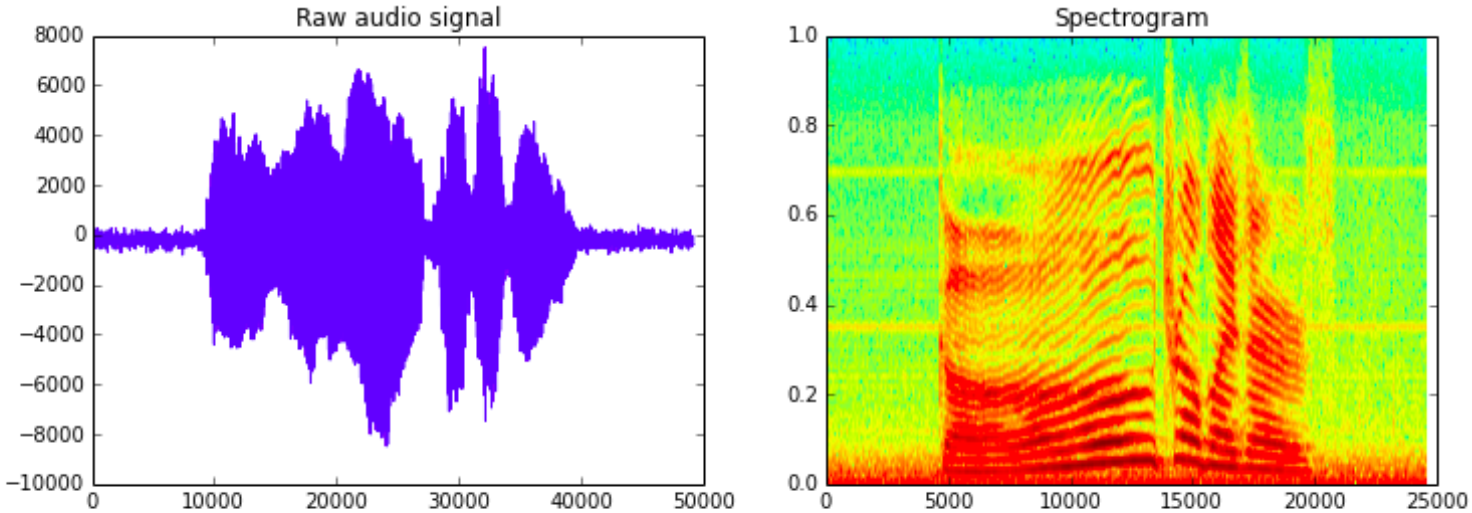
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

We begin by loading a datafile using SciPy's audio file support:

In [1]:
```
from scipy.io import wavfile
rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin `specgram` routine:

In [2]:
```
%matplotlib inline
from matplotlib import pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(x); ax1.set_title('Raw audio signal')
ax2.specgram(x); ax2.set_title('Spectrogram');
```



Besides the `%time` and `%timeit` magics, there are some other magic commands that will surely come in handy:

%pdb	Debug
%prun	Do a performance run
%writefile	Saves the contents of a cell to an external file
%pycat	Shows the syntax highlighted contents of an external file
%who	List all variables of a global scope
%store	Pass variables between notebooks
%load	Insert code from an external script
%run	Execute Python code
%env	Set environment variables

Jupyter tutorials

https://www.dataquest.io/blog/jupyter-notebook-tutorial/

https://www.youtube.com/watch?v=HW29067qVWk

https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook

https://realpython.com/jupyter-notebook-introduction/

and many more

https://nbviewer.jupyter.org/github/ipython/ipython/blob/6.x/examples/IPython%20Kernel/Trapezoid%20Rule.ipynb

jupyter nbviewer

JUPYTER FAQ

ipython / examples / IPython Kernel

Basic Numerical Integration: the Trapezoid Rule

A simple illustration of the trapezoid rule for definite integration:

$$\int_a^b f(x)\,dx \approx \frac{1}{2}\sum_{k=1}^{N}(x_k - x_{k-1})\,(f(x_k) + f(x_{k-1}))\,.$$

First, we define a simple function and sample it between 0 and 10 at 200 points

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
def f(x):
    return (x-3)*(x-5)*(x-7)+85

x = np.linspace(0, 10, 200)
y = f(x)
```

Choose a region to integrate over and take only a few points in that region

Trapezoid Rule.ipynb


```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Basic Numerical Integration: the Trapezoid Rule"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "A simple illustration of the trapezoid rule for definite integration:\n",
    "\n",
    "$$\n",
    "\\int_{a}^{b} f(x)\\, dx \\approx \\frac{1}{2} \\sum_{k=1}^{N} \\left( x_{k} - x_{k-1} \\right) \\left( f(x_{k}) + f(x_{k-1}) \\right).\n",
    "$$\n",
    "<br>\n",
    "First, we define a simple function and sample it between 0 and 10 at 200 points"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {
    "collapsed": false
   },
   "outputs": [],
   "source": [
    "%matplotlib inline\n",
    "import numpy as np\n",
    "import matplotlib.pyplot as plt"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "metadata": {
    "collapsed": false
   },
   "outputs": [],
```


***.ipynb is not HTML file**

Trapezoid Rule.ipynb


```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Basic Numerical Integration: the Trapezoid Rule"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "A simple illustration of the trapezoid rule for definite integration:\n",
    "\n",
    "$$\n",
    "\\int_{a}^{b} f(x)\\, dx \\approx \\frac{1}{2} \\sum_{k=1}^{N} \\left( x_{k} - x_{k-1} \\right) \\left( f(x_{k}) + f(x_{k-1}) \\right).\n",
    "$$\n",
    "<br>\n",
    "First, we define a simple function and sample it between 0 and 10 at 200 points"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {
    "collapsed": false
   },
   "outputs": [],
   "source": [
    "%matplotlib inline\n",
    "import numpy as np\n",
    "import matplotlib.pyplot as plt"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "metadata": {
    "collapsed": false
   },
   "outputs": [],
```


***.ipynb is not HTML file … it is JSON**

Trape


```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Basic Numerical Integration:
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "A simple illustration of the t
    "\n",
    "$$\n",
    "\\int_{a}^{b} f(x)\\, dx \\app
    "$$\n",
    "<br>\n",
    "First, we define a simple func
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {
    "collapsed": false
   },
   "outputs": [],
   "source": [
    "%matplotlib inline\n",
    "import numpy as np\n",
    "import matplotlib.pyplot as pl
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "metadata": {
    "collapsed": false
   },
   "outputs": [],
```


Trapezoid Rule.ipynb


```
   "execution_count": 3,
   "metadata": {
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "a, b = 1, 8 # the left and right boundaries\n",
    "N = 5 # the number of points\n",
    "xint = np.linspace(a, b, N)\n",
    "yint = f(xint)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Plot both the function and the area below it in the trapezoid approximation"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
   "metadata": {
    "collapsed": false
   },
   "outputs": [
    {
     "data": {
      "image/png": [
       "iVBORw0KGgoAAAANSUhEUgAABKcAAAL7CAYAAAAoHObLAAAABHNCSVQICAgIfAhkiAAAAAlwSFlz\n",
       "AAAWJQAAFiUBSVIk8AAAIABJREFUeJzs3Xl8Y3d97//3kSxvs9mzL8lMMjOZSSaTPZokJBOI5tfS\n",
       "ABfK0hPWtKUXSOlKi4ByS2+BB8tFQG9a2gu0hVLupeRQthZStggCpSwKEBISkkkyOvKMxx7b0pHs\n",
       "sbXr/P6QBpQztmexrWPZr+fjcR6yP9+vdD55KCTkPd/v9xiu6wAAAAAADwB4AAAAAAAAAAAAAAAA\n",
       "EU4BAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\n",
       "EU4BAAAAAADAN4RTAAAAAAAA8A3hFAAAAAAAAHxDOAUAAAAAAADfEE4BAAAAAADAN4RTAAAAAAAA\n",
       "8A3hFAAAAAAAAHxDOAUAAAAAAADfEE4BAAAAAADAN4RTAAAAAAAA8A3hFAAAAAAAAHxDOAUAAAAA\n",
       "AADfEE4BAAAAAADANx1z/QDTNLdK+rKkiyzL6j+L+V+S9BxJb7As6+5pxl8i6c2S9kqalHS/pLda\n",
       "lnVkrr0CAAAAAABgcZnTyinTNPdL+r6kVW5ZzH/v0m6vvHrafNN03y9pHskPSDp5ZKiknZL+q5p\n",
       "mjvm0isAAAAAAAAWn/NeOWWa5iFJn5X0mKQvSGVeZ3S/orSW+R9LFpxrdJer+kd+uWW9bam+udU\n",
       "D6s+KOnF59svAAAAAAAFp+5rJy6U9KPJB2WlLYs659mL+Hr/Rhjlda+koKRzqNPBWNKbNO01z1z\n",
       "83l3CwAAAAAgEVnLuHUXZJutyxr8kwTTdO8WNKbJP3hJP3hNKRbzqNPA01z\n",
```

```
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.4.2"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 0
}
```


To open/modify the *.ipynb file you need the whole environment

```
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.4.2"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 0
}
```


To open/modify the *.ipynb file you need the whole environment

Jupter also changed a lot since 2014

Version 3.x **Version 7.x**

Always render *.ipynb file to HTML

```
<body class="nbviewer">

  <!-- These are loaded at the top of the body so they are available to
       notebook cells when they are loaded below. -->
  <script src="/static/components/jquery/dist/jquery.min.js"></script>
  <script src="/static/components/requirejs/require.js"></script>
  <script src="/static/components/moment/min/moment.min.js"></script>
<!-- Navbar
===================================================== -->
  <nav id="menubar" class="navbar navbar-default navbar-fixed-top" data-spy="affix">
    <div class="container">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target=".navbar-collapse">
          <span class="sr-only">Toggle navigation</span>
          <i class="fa fa-bars"></i>
        </button>
        <a class="navbar-brand" href="/">
          <img src="/static/img/nav_logo.svg" width="159"/>
        </a>
      </div>

      <div class="collapse navbar-collapse">
        <ul class="nav navbar-nav navbar-right">
          <li>
            <a class="active" href="https://jupyter.org">JUPYTER</a>
          </li>
          <li>
    <a href="/faq" title="FAQ" >

        <span>FAQ</span>

    </a>
```

RStudio®

RStudio®

```
dens <- density(data, n = npts)
    dx <- dens$x
    dy <- dens$y
    if(add == TRUE)
        plot(0., 0                    main
            ylab                )
    if(orientati        ==        ysc        )
      dx2 <- (dx        min        ax(dx)
        x[1.]
      dy2 <- (dx - min        x(dy)
        y[1.]
      seqbelow <- rep(y[1.], length(dx))
      if(Fill == T)
        confshade(dx2, seqbelow, dy2
```

Discover 2083 software packages available in Bioconductor release 3.14

rsrecovr - master - RStudio

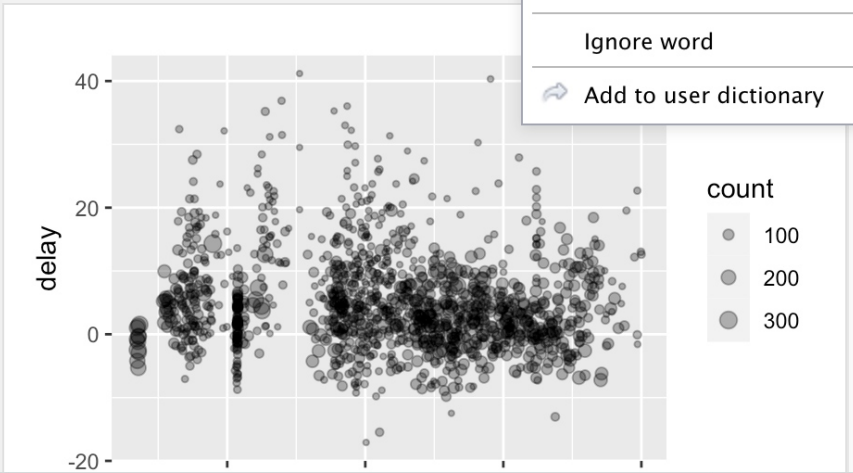
Go to file/function | Addins | rsrecovr — git

projects.R | datavis.Rmd

Knit | Insert | Run

```
400 ```{r, eval=FALSE}
401 # generate distance/delay scatter astehtic
402 ggplot(delay, aes(x = dist, y = delay)
403   geom_point(aes(size = count), alpha =
404   scale_size_area(max_size = 3)
405 ```
```

aesthetic
aseptic
ascetic
Ignore word
Add to user dictionary



401:40 | Chunk 22 | R Markdown

Console | Terminal | Find in Files | Jobs

Results for "context" in ~/git/rsrecovr | Find | Replace

Replace with: session | Replace All

~/git/rsrecovr/R/all.R

```
37: results ← rbind(results, cbind(project = NA, contextsession = NA,
```

~/git/rsrecovr/R/projects.R

```
10: # list all the contextsession IDs
11: contextsessions ← list.files(state_folder, pattern = "^[a-zA-Z0-9]
13: # recover the sources from each contextsession
14: results ← lapply(contextsessions, function(contextsession_id) {
16: recovred ← recovr_sessions(file.path(state_folder, contextsession_
21: cbind(data.frame(contextsession = contextsession_id), recovred)
```

Environment | History | Connections | Build | Git | Tutorial

Data basics

Start Tutorial

learnr: ex-data-basics

Learn about the base data types in R. Explore R's data frames, and learn how to interact with data frames and their columns.

Filter observations

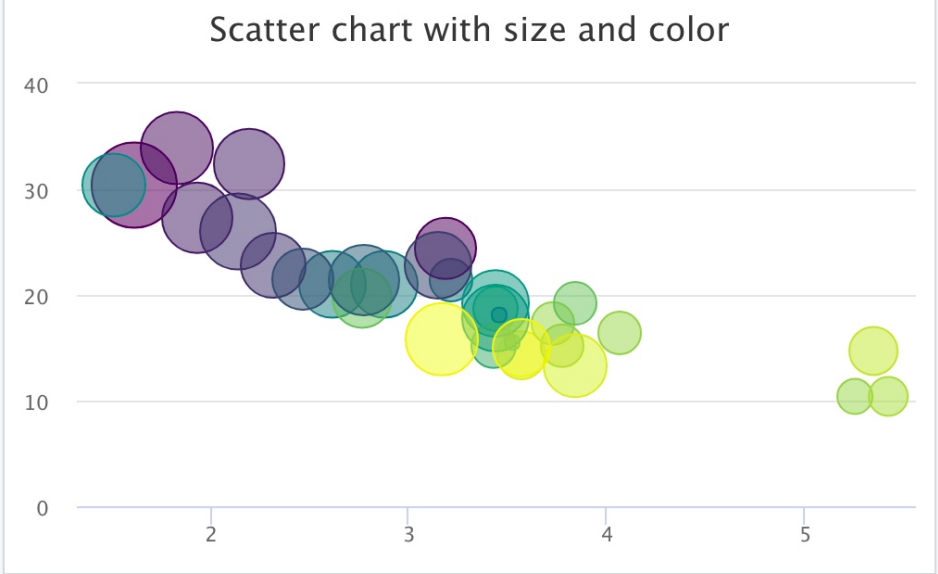
Start Tutorial

learnr: ex-data-filter

Learn how to filter observations in a data frame. Use `filter()` to extract observations from a data frame, and use `&`, `|`, and `!` to write logical tests.

Files | Plots | Packages | Help | Viewer

Zoom | Export | Publish




rsrecovr - master - RStudio
Go to file/function
Addins
rsrecovr — git
projects.R
datavis.Rmd
Knit
Insert
Run
Environment
History
Connections
Build
Git
Tutorial
```{r, eval=FALSE}
# generate distance/delay scatter astehtic
ggplot(delay, aes(x = dist, y = delay)
Data basics
Start Tutorial
ata frames,
columns.
art Tutorial
filter()
, |, and !
Publish
hwy
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv
```


bar.py
Run Script
1 collection = ['hey', 5, 'd']
2 for x in collection:
3 print x
4:1
Python
Console
Terminal
Python (busy)
~/code/crayon
Garys-iMac:crayon gary$ python
Python 2.7.13 (default, Feb 2 2017, 08:57:11)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> collection = ['hey', 5, 'd']
>>> for x in collection:
... print x
...
hey
5
d
>>>

bar.py

```
collection = ['hey', 5, 'd']
for x in collection:
  print x
```

4:1

Console Terminal

Python (busy) ~/code/crayon

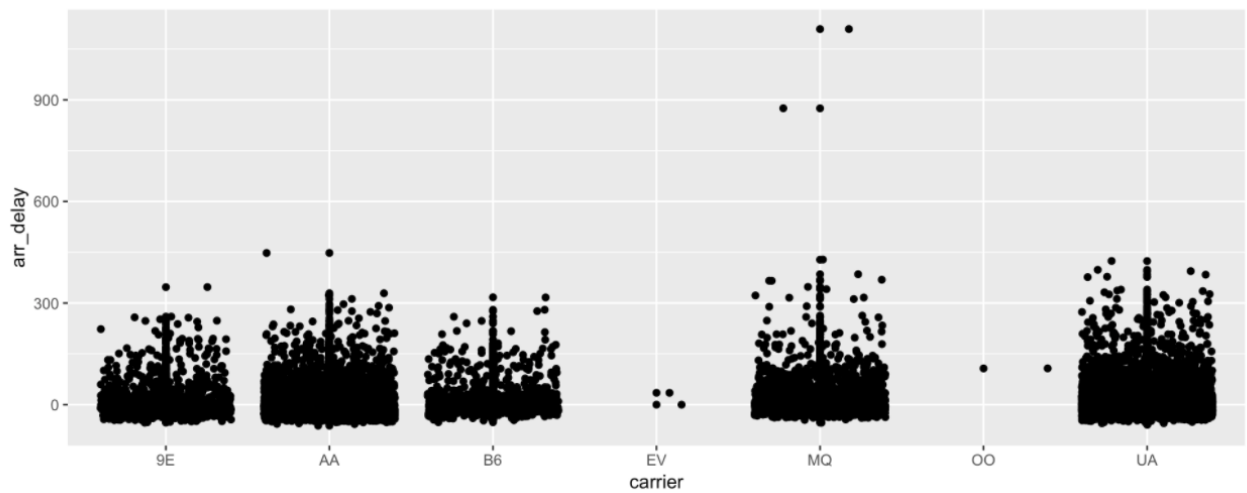
```
Garys-iMac:crayon gary$ python
Python 2.7.13 (default, Feb  2 2017
[GCC 4.2.1 Compatible Apple LLVM 8.
Type "help", "copyright", "credits"
>>> collection = ['hey', 5, 'd']
>>> for x in collection:
...   print x
...
hey
5
d
>>>
```

Preview Insert Run

```{r setup, include=FALSE}
library(reticulate)
use_condaenv("r-reticulate")
```

```{python}
import pandas
flights = pandas.read_csv("flights.csv")
flights = flights[flights["dest"] == "ORD"]
flights = flights[['carrier', 'dep_delay', 'arr_delay']]
flights = flights.dropna()
```

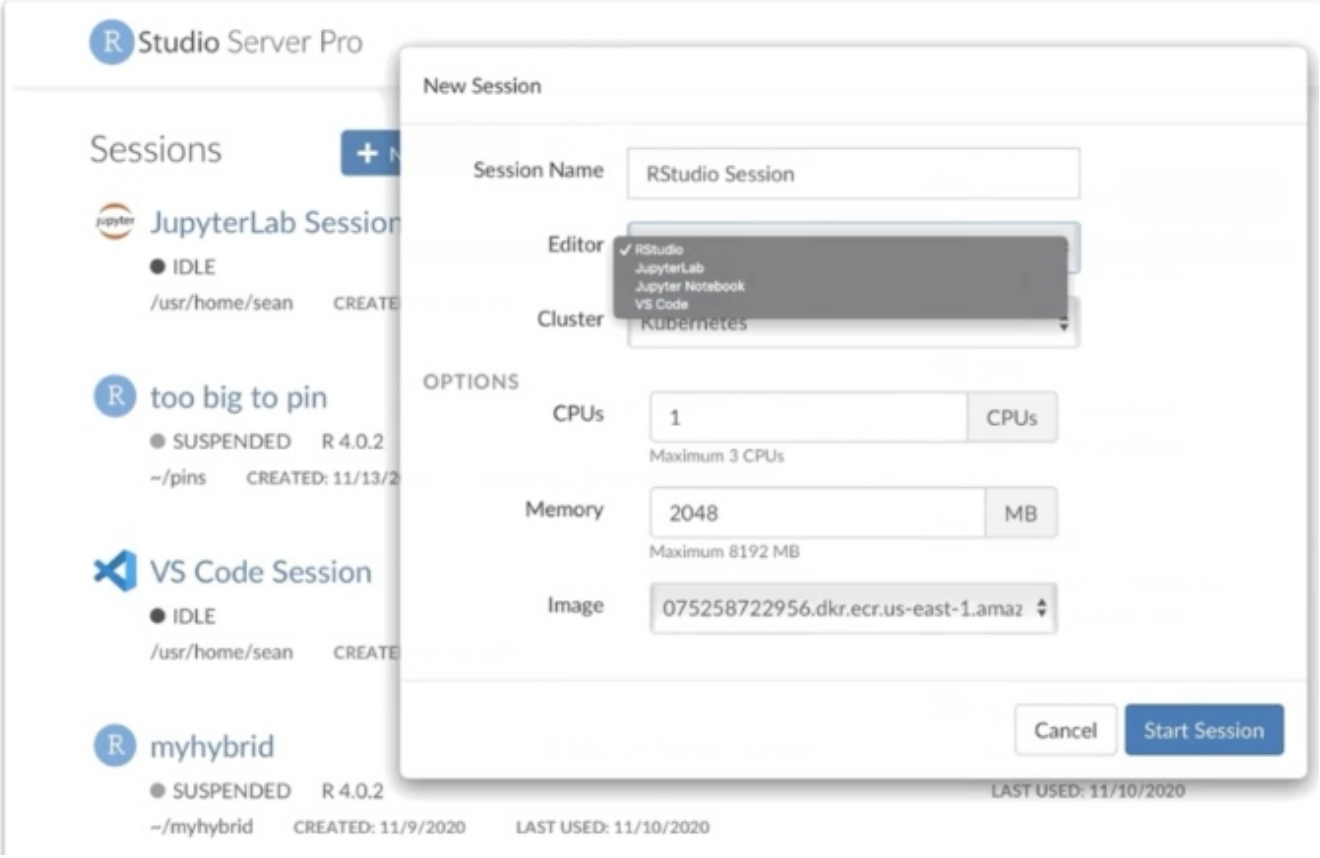
```{r, fig.height=2, fig.width=5}
library(ggplot2)
ggplot(py$flights, aes(carrier, arr_delay)) + geom_point() + geom_jitter()
```



15:34 Chunk 3 R Markdown

For more information about R & python integration using Rstudio watch video at:

https://www.rstudio.com/solutions/r-and-python/ **(15 min)**



Outline

1. Data formats in bioinformatics,
2. Popular software libraries (BioPerl, BioPython)
3. Most important bioinformatics databases (UniProt, PDB, RefSeq, GenBank, ENA, InterPro, etc.)
4. Software licensing for scientific purposes. Free-software licensing. Patents.

5. Generic model Organism database (GMOD) project - assumptions, history and usage

6. Genome browsers, problem description and state of the solutions

7. Version control systems (CVS, SVN, git), and online collaboration ad distribution platforms (github, sourceforge).

8. Software testing, automated testing frameworks

9. Scientific workflow systems - taverna and galaxy. MyExperiment platform. Reproducible research.

10. Literate programming idea and sweave, markdown, software documentation

11. Interactive scripting platforms, Rstudio, Jupyter

1. Data formats in bioinformatics,
2. Popular software libraries (BioPerl, BioPython)
3. Most important bioinformatics databases (UniProt, PDB, RefSeq, GenBank, ENA, InterPro, etc.)
4. Software licensing for scientific purposes. Free-software licensing. Patents.

5. Generic model Organism database (GMOD) project - assumptions, history and usage

6. Genome browsers, problem description and state of the solutions

7. Version control systems (CVS, SVN, git), and online collaboration ad distribution platforms (github, sourceforge).

8. Software testing, automated testing frameworks

9. Scientific workflow systems - taverna and galaxy. MyExperiment platform. Reproducible research.

10. Literate programming idea and sweave, markdown, software documentation

11. Interactive scripting platforms, Rstudio, Jupyter.

Never write the code or the comments in Polish (or any other language than English).

Use ENGLISH only

https://github.com/haichengyi/ACP-DL/blob/master/

```
        rects1 = plt.plot(fpr, tpr, label=legend_text)

def ACP_DL():
    # define parameters
    data_dim = 483
    timesteps = 1
    batch_size = 32   # if dataset = acp240, set batch_size = 32; if da
    epochs = 30
    # get data

    #bpf, kmer, label = prepare_feature_acp740()
    bpf, kmer, label = prepare_feature_acp240()
    X = np.concatenate((bpf, kmer), axis=1)  # 1 行拼接 0 默认，列拼接
    #  expected input data shape: (batch_size, timesteps, data_dim)
    X = np.reshape(X, (len(X), timesteps, data_dim))
```

Never write the code or the comments in Polish (or any other language than English).

Use ENGLISH only

Name variables in meaningful way

A = 78	nub_samples = 78
B = [2,4,5,32, …]	samples_tab = [2,4,5,32, …]
A = len(B)	nub_samples = len(samples_tab)

Never write the code or the comments in Polish (or any other language than English).

Use ENGLISH only

Name variables in meaningful way

A = 78	nub_samples = 78
B = [2,4,5,32, …]	samples_tab = [2,4,5,32, …]
A = len(B)	nub_samples = len(samples_tab)

In general, follow ‘PEP 8 -- Style Guide for Python Code’

Never write the code or the comments in Polish (or any other language).

Use ENGLISH only

Name variables in meaningful way

A = 78	nub_samples = 78
B = [2,4,5,32, …]	samples_tab = [2,4,5,32, …]
A = len(B)	nub_samples = len(samples_tab)

In general, follow ‘PEP 8 -- Style Guide for Python Code’

“Comments” – there is never too many comments

“Comments”

Inline comments are unnecessary and in fact distracting if they state the obvious. Don't do this:

```
x = x + 1                 # Increment x
```

But sometimes, this is useful:

Inline Comments

```
x = x + 1                 # Compensate for border
```

“Comments”

Inline comments are unnecessary and in fact distracting if they state the obvious. Don't do this:

```
x = x + 1                 # Increment x
```

But sometimes, this is useful:

Inline Comments

```
x = x + 1                 # Compensate for border
```

PEP 257 -- Docstring Conventions

```
def function(a, b):
    """Do X and return a list."""
```

(Of course "Do X" should be replaced by a useful description!)

“Comments”

SPHINX
Python Documentation Generator

Table of Contents

Previous topic
Text Processing Services

Next topic
re — Regular expression operations

This Page

string — Common string operations

Source code: Lib/string.py

See also: Text Sequence Type — str

String Methods

String constants

The constants defined in this module are:

`string.`**`ascii_letters`**

The concatenation of the `ascii_lowercase` and `ascii_uppercase` constants described below. This value is not locale-dependent.

`string.`**`ascii_lowercase`**

The lowercase letters `'abcdefghijklmnopqrstuvwxyz'`. This value is not locale-dependent and will not

“Comments”

SPHINX
Python Documentation Generator

Python » English 3.9.4 Documentation » The Python Standard Library » Text Processing previous | next | modules | index

Services » Quick search Go |

Table of Contents

Previous topic

Text Processing Services

Next topic

re — Regular expression operations

This Page

string — Common string operations

Source code: Lib/string.py

See also: Text Sequence Type — str

String Methods

biopython

String constants

The constants defined in this module are:

string.**ascii_letters**

The concatenation of the ascii_lowercase and ascii_uppercase constants described below. This value is not locale-dependent.

string.**ascii_lowercase**

The lowercase letters 'abcdefghijklmnopqrstuvwxyz'. This value is not locale-dependent and will not

Code testing

pydoc — Documentation generator and online help system ¶

Doxygen

https://docs.python.org/3/library/argparse.html 90%

Python » English 3.9.4 Documentation » The Python Standard Library » Generic Operating System Services » Quick sear

Table of Contents

argparse — Parser for command-line options, arguments and sub-commands

```
>>> parser.add_argument('integers', metavar='N', type=int, nargs='+',
...                     help='an integer for the accumulator')
>>> parser.add_argument('--sum', dest='accumulate', action='store_const',
...                     const=sum, default=max,
...                     help='sum the integers (default: find the max)')
```

```
$ python myprogram.py --help
usage: myprogram.py [-h] [--foo FOO]

optional arguments:
 -h, --help  show this help message and exit
 --foo FOO   foo help
```

Code testing

Frequently:

bioinformatic project = public funds (a.k.a Academia)

Code testing

Frequently:

bioinformatic project = public funds (a.k.a Academia)

This imply:

- no money (and time) for proper testing (e.g. writing the unitests)

Try to write in the grant proposal 2 months funding for program testing

Code testing

Frequently:

bioinformatic project = public funds (a.k.a Academia)

This imply:

- no money (and time) for proper testing (e.g. writing the unitests)

Try to write in the grant proposal 2 months funding for program testing

- no testers

Usually, the only testers are … the developer(s) and the users

Code testing

Frequently:

bioinformatic project = public funds (a.k.a Academia)

This imply:

- no money (and time) for proper testing (e.g. writing the unitests)

Try to write in the grant proposal 2 months funding for program testing

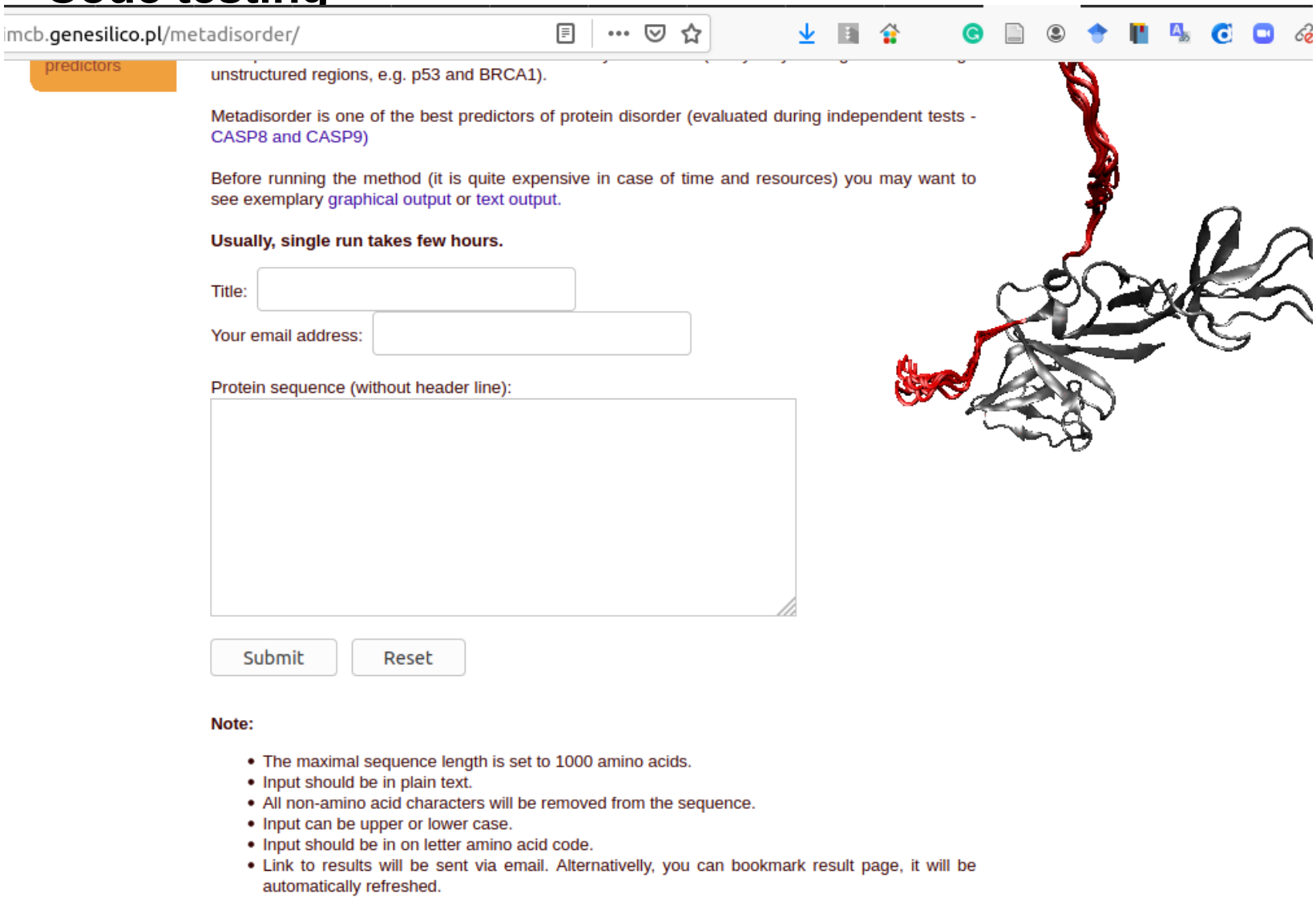
- no testers

Usually, the only testers are … the developer(s) and the users

Advice:

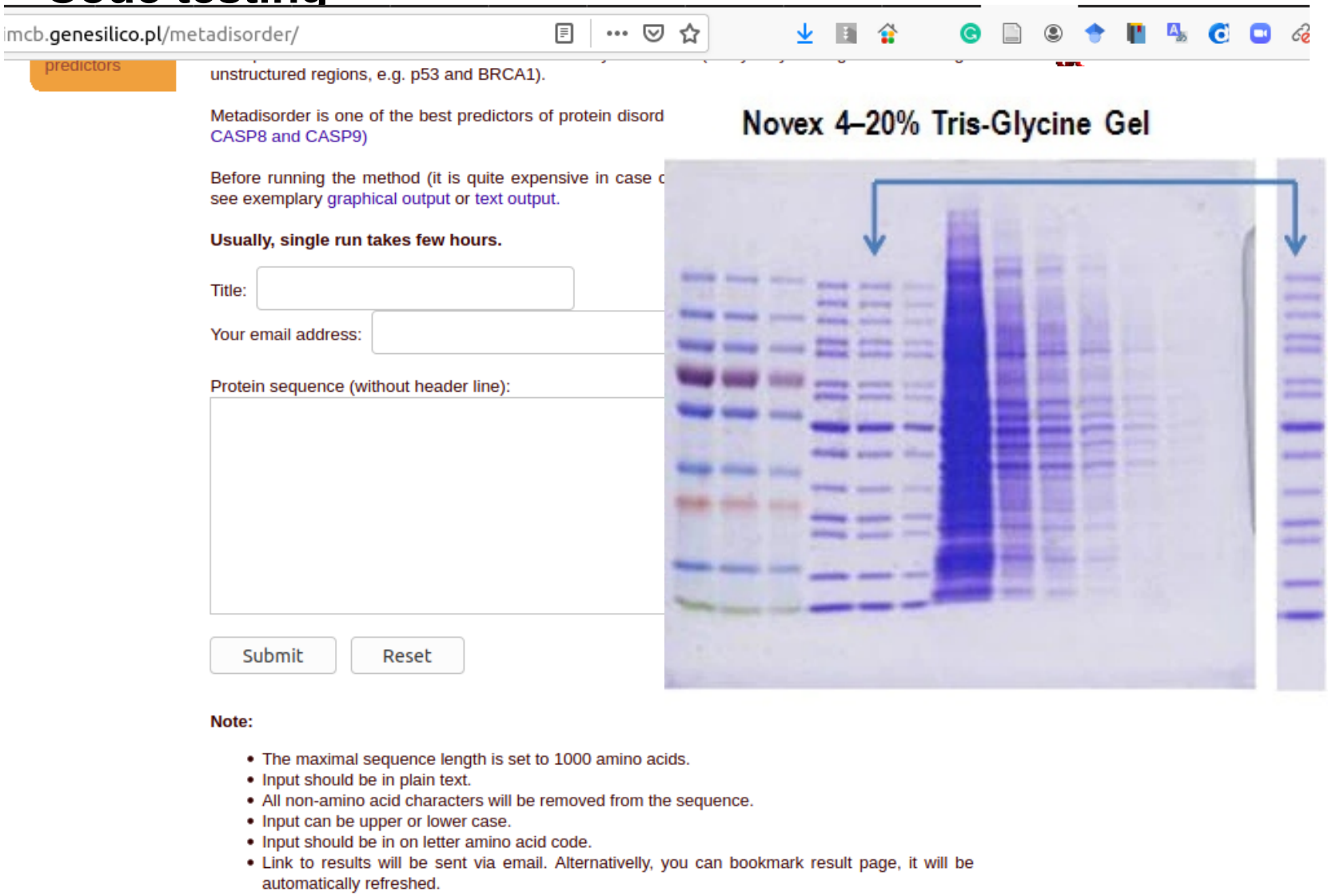
- ask sb who is not bioinformatician (e.g. biologist) to run you software

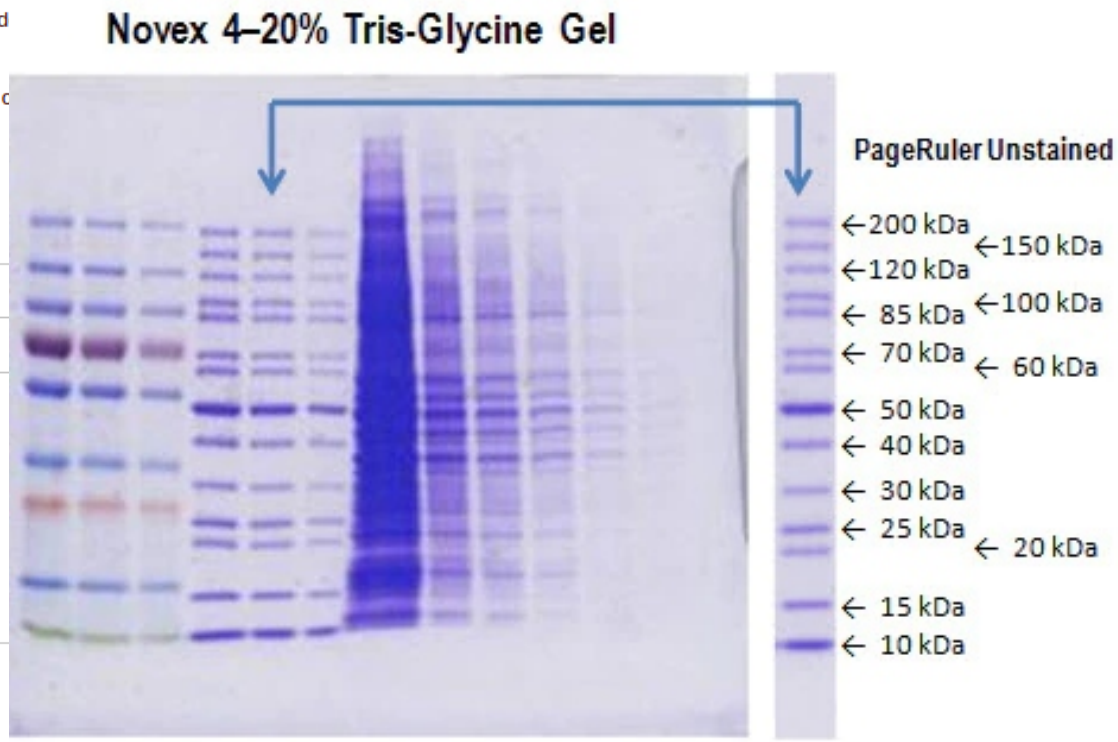
Code testing



- ask sb who is not bioinformatician (e.g. biologist) to run you software, webserver

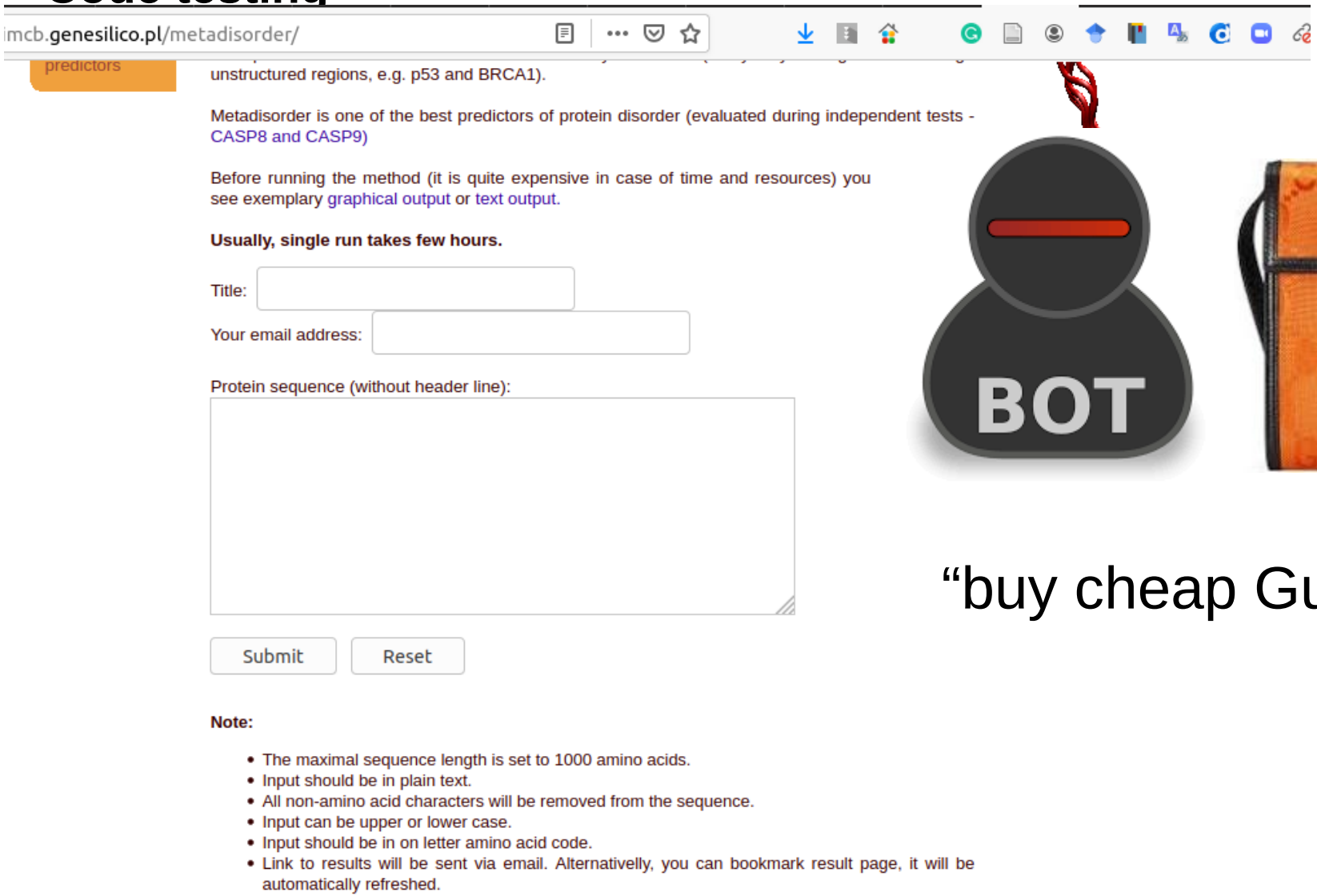
Code testing





- ask sb who is not bioinformatician (e.g. biologist) to run you software, webserver

Code testing



“buy cheap Gucci bags”

- ask sb who is not bioinformatician (e.g. biologist) to run you software

Code testing

Frequently:

bioinformatic project = public funds (a.k.a Academia)

This imply:

- no money (and time) for proper testing (e.g. writing the unitests)

Try to write in the grant proposal 2 months funding for program testing

- no testers

Usually, the only testers are … the developer(s) and the users

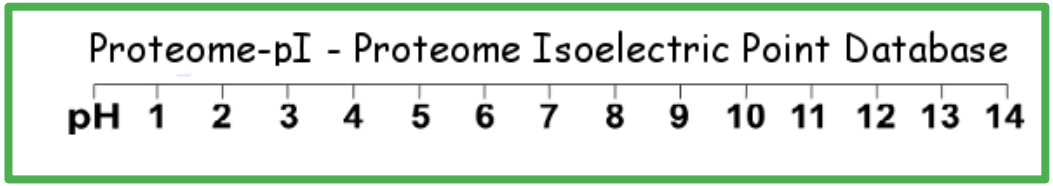
Advice:

- ask sb who is not bioinformatician (e.g. biologist) to run you software
- run your code on thousands or millions of input files

(e.g. process some big database like UniProt)

Code testing

Standalone version of the program



Usually, the only testers are … the developer(s) and the users

Advice:

- ask sb who is not bioinformatician (e.g. biologist) to run you software
- run your code on thousands or millions of input files
  (e.g. process some big database like UniProt)

Code testing

print

Code testing

Source code checkers (static testing)

Execution tests:
– Unit tests
– Integration tests
– Regression tests
– UX tests (GUI, Web, mobile)
– Fuzz tests (random data, shouldn’t crash)
– Mock tests

Code debugging

pdb — The Python Debugger

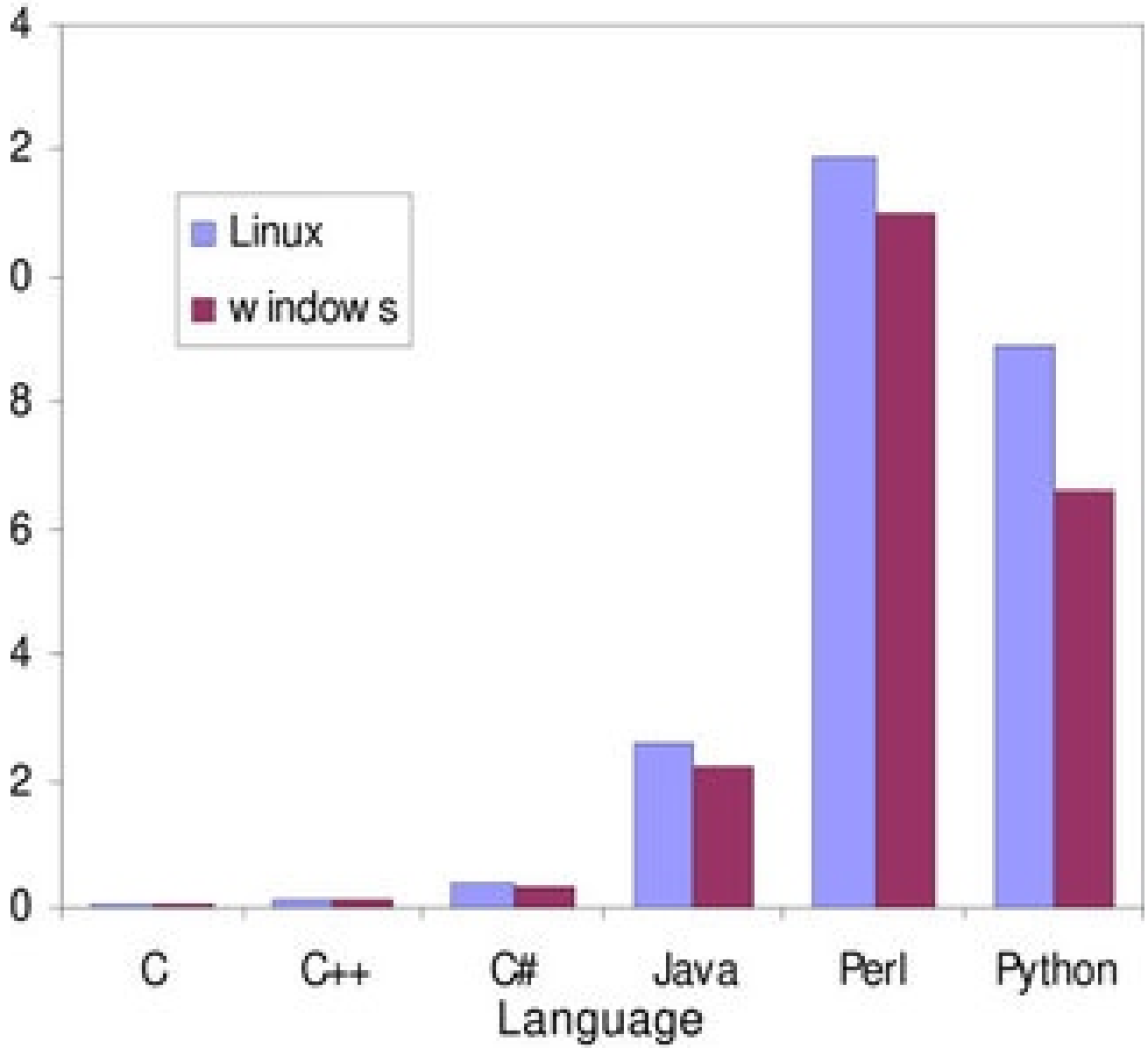
```
>>> import pdb
>>> import mymodule
>>> pdb.run('mymodule.test()')
> <string>(0)?()
(Pdb) continue
> <string>(1)?()
(Pdb) continue
NameError: 'spam'
> <string>(1)?()
(Pdb)
```

```
python3 -m pdb myscript.py
```

Code profiling

time()

Code profiling



Code profiling

Python <--> Cpython <--> Cyton <--> PyPy

Code profiling

https://realpython.com/build-python-c-extension-module/

C

```c
static PyObject *method_fputs(PyObject *self, PyObject *args) {
    char *str, *filename = NULL;
    int bytes_copied = -1;

    /* Parse arguments */
    if(!PyArg_ParseTuple(args, "ss", &str, &filename)) {
        return NULL;
    }

    FILE *fp = fopen(filename, "w");
    bytes_copied = fputs(str, fp);
    fclose(fp);

    return PyLong_FromLong(bytes_copied);
}
```

Code profiling

https://realpython.com/build-python-c-extension-module/

C

```c
static PyObject *method_fputs(PyObject *self, PyObject *args) {
    char *str, *filename = NULL;
    int bytes_copied = -1;

    /* P
    if(

    }

    FILE
    byte
    fclo

    retu
}
```

Python >>>

```python
>>> import fputs
>>> fputs.__doc__
'Python interface for the fputs C library function'
>>> fputs.__name__
'fputs'
>>> # Write to an empty file named `write.txt`
>>> fputs.fputs("Real Python!", "write.txt")
13
>>> with open("write.txt", "r") as f:
>>>     print(f.read())
'Real Python!'
```

Code profiling

`timeit` — Measure execution time of small code snippets

Basic Examples

The following example shows how the Command-Line Interface can be used to compare three different expressions:

```
$ python3 -m timeit '"-".join(str(n) for n in range(100))'
10000 loops, best of 5: 30.2 usec per loop
$ python3 -m timeit '"-".join([str(n) for n in range(100)])'
10000 loops, best of 5: 27.5 usec per loop
$ python3 -m timeit '"-".join(map(str, range(100)))'
10000 loops, best of 5: 23.2 usec per loop
```

This can be achieved from the Python Interface with:

```
>>> import timeit
>>> timeit.timeit('"-".join(str(n) for n in range(100))', number=10000)
0.3018611848820001
>>> timeit.timeit('"-".join([str(n) for n in range(100)])', number=10000)
0.2727368790656328
>>> timeit.timeit('"-".join(map(str, range(100)))', number=10000)
0.23702679807320237
```

A callable can also be passed from the Python Interface:

```
>>> timeit.timeit(lambda: "-".join(map(str, range(100))), number=10000)
0.19665591977536678
```

Code profiling

Table of Contents

Previous topic

pdb — The Python Debugger

Next topic

timeit — Measure execution time of small code snippets

The Python Profilers

Source code: Lib/profile.py and Lib/pstats.py

Introduction to the profilers

`cProfile` and `profile` provide *deterministic profiling* of Python programs. A *profile* is a set of statistics that describes how often and for how long various parts of the program executed. These statistics can be formatted into reports via the `pstats` module.

The Python standard library provides two different implementations of the same profiling interface:

1. `cProfile` is recommended for most users; it's a C extension with reasonable overhead that makes it suitable for profiling long-running programs. Based on `lsprof`, contributed by Brett Rosen and Ted Czotter.
2. `profile`, a pure Python module whose interface is imitated by `cProfile`, but which adds significant overhead to profiled programs. If you're trying to extend the profiler in some way, the task might be easier with this module. Originally designed and written by Jim Roskind.

Code profiling

To profile a function that takes a single argument, you can do:

```
import cProfile
import re
cProfile.run('re.compile("foo|bar")')
```

(Use `profile` instead of `cProfile` if the latter is not available on your system.)

The above action would run `re.compile()` and print profile results like the following:

```
      197 function calls (192 primitive calls) in 0.002 seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000    0.001    0.001 <string>:1(<module>)
     1    0.000    0.000    0.001    0.001 re.py:212(compile)
     1    0.000    0.000    0.001    0.001 re.py:268(_compile)
     1    0.000    0.000    0.000    0.000 sre_compile.py:172(_compile_charset)
     1    0.000    0.000    0.000    0.000 sre_compile.py:201(_optimize_charset)
     4    0.000    0.000    0.000    0.000 sre_compile.py:25(_identityfunction)
   3/1    0.000    0.000    0.000    0.000 sre_compile.py:33(_compile)
```

Code profiling

Instead of printing the output at the end of the profile run, you can save the results to a file by specifying a filename to the `run()` function:

```
import cProfile
import re
cProfile.run('re.compile("foo|bar")', 'restats')
```

The `pstats.Stats` class reads profile results from a file and formats them in various ways.

The files `cProfile` and `profile` can also be invoked as a script to profile another script. For example:

```
python -m cProfile [-o output_file] [-s sort_order] (-m module | myscript.py)
```

unittest — Unit testing framework

https://realpython.com/python-testing/

Python

```python
def test_sum():
    assert sum([1, 2, 3]) == 6, "Should be 6"

def test_sum_tuple():
    assert sum((1, 2, 2)) == 6, "Should be 6"

if __name__ == "__main__":
    test_sum()
    test_sum_tuple()
    print("Everything passed")
```

Python

```python
def test_sum():
    assert sum([1, 2, 3]) == 6, "Should be 6"

def test_sum_tuple():
    assert sum((1, 2, 2)) == 6, "Should be 6"

if __name__ == "__main__":
    test_sum()
    test_sum_tuple()
    print("Everything passed")
```

Python

```python
import unittest


class TestSum(unittest.TestCase):

    def test_sum(self):
        self.assertEqual(sum([1, 2, 3]), 6, "Should be 6")

    def test_sum_tuple(self):
        self.assertEqual(sum((1, 2, 2)), 6, "Should be 6")

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

Python

```python
def test_sum():
    assert sum([1, 2, 3]) == 6, "Should be 6"
```

Shell

```shell
$ python -m unittest test
F.
======================================================================
FAIL: test_list_fraction (test.TestSum)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 21, in test_list_fraction
    self.assertEqual(result, 1)
AssertionError: Fraction(9, 10) != 1

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

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

Other tests runners

An example of a simple test:

```python
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(3) == 5
```

To execute it:

```
$ pytest
============================= test session starts ==============================
platform linux -- Python 3.x.y, pytest-6.x.y, py-1.x.y, pluggy-0.x.y
cachedir: $PYTHON_PREFIX/.pytest_cache
rootdir: $REGENDOC_TMPDIR
collected 1 item

test_sample.py F                                                         [100%]

================================= FAILURES =================================
_______________________________ test_answer ________________________________

    def test_answer():
>       assert inc(3) == 5
E       assert 4 == 5
E        +  where 4 = inc(3)

test_sample.py:6: AssertionError
========================== short test summary info ==========================
FAILED test_sample.py::test_answer - assert 4 == 5
============================== 1 failed in 0.12s ===============================
```

Other tests runners

nose

is nicer testing for python

pytest

Other tests runners

Hypothesis

Test faster, fix more

```
@given(st.lists(st.floats(allow_nan=False, allow_infinity=False), min_size=1))
def test_mean(xs):
    assert min(xs) <= mean(xs) <= max(xs)
```

```
Falsifying example: test_mean(
  xs=[1.7976321109618856e+308, 6.10239004302755e+303]
)
```

Other tests runners

tox is a tool for automating test environment management and testing against multiple interpreter configurations

```
# content of: tox.ini , put in same dir as setup.py
[tox]
envlist = py27,py36

[testenv]
# install pytest in the virtualenv where commands will be executed
deps = pytest
commands =
    # NOTE: you can run any command line tool here - not just tests
    pytest
```

Other tests runners

`unittest.mock` — mock object library

unittest.mock is a library for testing in Python. It allows you to replace parts of your system under test with mock objects and make assertions about how they have been used.

unittest.mock

Python

```python
from datetime import datetime

def is_weekday():
    today = datetime.today()
    # Python's datetime library treats Monday as 0 and Sunday as 6
    return (0 <= today.weekday() < 5)

# Test if today is a weekday
assert is_weekday()
```

Shell

```shell
$ python my_calendar.py
```

Shell

```shell
$ python my_calendar.py
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    assert is_weekday()
AssertionError
```

mock

Python

```python
import datetime
from unittest.mock import Mock

# Save a couple of test days
tuesday = datetime.datetime(year=2019, month=1, day=1)
saturday = datetime.datetime(year=2019, month=1, day=5)

# Mock datetime to control today's date
datetime = Mock()

def is_weekday():
    today = datetime.datetime.today()
    # Python's datetime library treats Monday as 0 and Sunday as 6
    return (0 <= today.weekday() < 5)

# Mock .today() to return Tuesday
datetime.datetime.today.return_value = tuesday
# Test Tuesday is a weekday
assert is_weekday()
# Mock .today() to return Saturday
datetime.datetime.today.return_value = saturday
# Test Saturday is not a weekday
assert not is_weekday()
```

Thank you for your time
and
See you at the next lecture

Any other
questions & comments

lukaskoz@mimuw.edu.pl