```
# Scientific Python
'''
Collection of commands uisefull for scientific Python.
'''

#
# Prelude
#
# Python code is interpreted.

# Python code runs on multiple platforms, given the right libraries are insta

# Output
print("Hello World!")


#
# Variables
#
# Variables/Objects are not declared and typed dynamically.
a = 5
b = 7
c = a + b
d = "hello"

print(c)
print(hello)

# Variable names are case sensitive.
# Must start with _ or a letter, (not a number!)

A = 1
c = A + a
print(c)


# A few operators:
2+2
2-1
3*2
5/4 # integer division in Python 2.*
5./4 # float division in Python 2.*
5//4 # integer division
2**3 # power
5 % 4 # remainder of 5/4
```

```
46
47
48
49 #
50 # Strings
51 #
52 my_string = "chicken kidney pie"
53 print(my_string)
54 another_string = 'beautiful blue raincoat'
55 print(another_string)
56
57 # Special characters and commands can be used using the backslash '\'
58 print('Don\'t worry,\nbe happy.')
59
60 # Some String operations.
61 print("It is a shoe! It is a shoe!\n" "It\'s a sandal!")
62
63 # String *slicing*.
64 my_string[0]
65 my_string[3]
66 my_string[-1]
67 my_string[2:4]
68 my_string[::2]
69
70 len(my_string)
71
72
73
74 #
75 # Functions.
76 #
77
78 # Functions return values and do something.
79 a = abs(-5)
80
81 # Get help (IPython):
82 abs?
83 abs??
84
85
86 #
87 # Lists.
88 #
89 sequence = [1, 2, 5, 10, 20, 300]
90 sequence[0]
```

```
sequence[-1]
sequence[-2]
sequence[2:-3]

sequence + [1, 2]

sequence.append(500)
sequence[-1] = [-1, -2, -3]
sequence.append('what a twist!')



#
# Script files.
#

# Create a flie myScript.py.
# We can execude file using.

execfile('myScript.py')
# OR:
runfile('myScript.py')
# OR (Python 3):
exec(open("myScript.py").read())

# Or we can call it from the command line.
python myScript.py

# Comments can be added with the hash symbol.

# Identation matters!!!!!!
# Always 4 spaces!:


#
# Control Flow.
#

if x < 0:
    print("x is negative")
elif x == 0:
    print("x = 0")
else:
    print("x is positive")

```

```
136 # Boolean algebra.
137 if (x > 0) + (x < 0):
138     print("x is not 0")
139 if (x > 0) or (x < 0):
140     print("x is not 0")
141 
142 if (x > 0) * (x < 1):
143     print("x is in the open interval (0, 1)")
144 if (x > 0) and (x < 1):
145     print("x is in the open interval (0, 1)")
146 
147 # Loops
148 my_list = [0, 1, 2, 3]
149 for i in my_list:
150     print(i)
151 
152 my_list = [10, 20, 3, 4]
153 for i, j in enumerate(my_list):
154     print(i, j)
155 
156 for i in range(10):
157     print(i)
158 
159 for i in range(5, 10):
160     print(i)
161 
162 i = 0
163 while i < 5:
164     print(i)
165     i += 1
166 
167 
168 
169 #
170 # User defined functions.
171 #
172 
173 def my_fun(a, b):
174     '''
175     Does something.
176     Always add some documentation.
177     '''
178     a += 1
179     if a > 3:
180         return a + b
```

```
181     else:
182         return a, b
183     print('after return')
184 
185 my_fun(4, 5)
186 my_fun?
187 my_fun??
188 x = 4
189 y = 5
190 print(my_fun(x, y))
191 print(x)
192 
193 # The scope of the variables is the function itself.
194 
195 
196 
197 #
198 # Numpy arrays (easy).
199 #
200 
201 import numpy as np
202 
203 # Create a 1d array from a list.
204 a = np.array([1, 2, 3])
205 # Create a 2d array from a list.
206 b = np.array([[1, 2], [3, 4]])
207 # Access the array's elements.
208 a[0]
209 b[0, 0]
210 # Create array with specific data type.
211 a = np.array([1, 2, 3], dtype=np.float16)
212 
213 # Create a default array.
214 a = np.zeros(10)
215 b = np.zeros([3, 3])
216 e = np.random.random([10, 10])
217 f = np.arange(3, 20, 3)
218 g = np.linspace(0, 1, 11)
219 
220 # Create a meshgrid from two or more 1d arrays.
221 x = np.linspace(0, 1, 6)
222 y = np.linspace(0, 10, 6)
223 xx, yy = np.meshgrid(x, y)
224 xx, yy = np.meshgrid(x, y, indexing='ij')
225 z = np.linspace(0, 100, 6)
```

```
yy, xx, zz = np.meshgrid(x, y, z)
xx, yy, zz = np.meshgrid(x, y, z, indexing='ij')


# Arrays can be accessed similarly to lists including slicing.
b[-1, -2]
a[1:-3] = 0.3
a[1::2]
a[-1::-2]

# Access some information of the arrays.
e.shape
a.dtype
e.max()
e.min()
e.ndim
e.size

# Array operations are for the most part element wise.
x = np.ones(5)
y = np.zeros(5)
z = x + y

# A few useful function to manipulate arrays.
e.swapaxes(0, 1)
e.reshape(5, 20)

# Like strings and lists, numpy arrays are mutable object.
a = np.array([1, 2, 3])
b = a
b[0] = 5
print(a)
# Works with slicing.
a = np.linspace(0, 1, 11)
b = a[::2]
b[0] = 5
b[1] = 6
print(a)

# To copy, one needs to use 'deep copy'.
a = np.array([1, 2, 3])
b = a.copy()
b[0] = 5
print(a)

```

```
271
272 #
273 # Numpy functions and constants.
274 #
275
276 # Numpy functions mostly act on numpy arrays, which make computation much fas
277 a = np.random(5)
278 np.sin(a)
279 np.cos(a)
280 np.arctan2(x, y)
281 np.max()
282 np.min()
283 a = np.random.random(4)
284 b = np.random.random(4)
285 np.dot(a, b)
286 np.median(a)
287 np.mean(a)
288 np.isnan()
289 np.isinf()
290 np.real(3 + 2j)
291 np.imag(3 + 2j)
292 np.histogram(np.random.random(100))
293 np.sum(a)
294
295 # Some numpy constants.
296 np.NaN
297 np.Inf
298 np.e
299 np.pi
300 np.float128
301 np.int16
302
303
304 # Array masks (Boolean array indexing).
305 a = np.linspace(0, 1, 11)
306 a > 0.5
307 a[a > 0.5]
308 age = np.array([18, 21, 19, 23, 21, 20, 19.5])
309 marks = np.array([1, 5, 2, 4, 1, 3, 4])
310 np.mean(marks[age > 20])
311 # Better: create a Boolean mask array.
312 mask = age > 20
313 np.mean(marks[mask])
314
315 # Find value in array.
```

```
316 np.where(a > 0.5)
317
318
319 #
320 # Numpy IO
321 #
322
323 # Write and read clear text data files.
324 t = np.linspace(0, 10, 11)
325 x = np.random.random(11)
326 y = np.random.random(11)
327 data = np.vstack([t, x, y]).transpose()
328 np.savetxt('time_series.dat', data, delimiter=',', header='t    x    y')
329 my_data = np.loadtxt('time_series.dat', delimiter=',')
330
331 # Write and read binary files.
332 np.save('data', data)
333 my_data = np.load('data.npy')
334
335
336
337 #
338 # plottind 2d
339 #
340
341 import matplotlib.pyplot as plt
342 # or:
343 import pylab as plt
344
345 plt.plot([1, 2, 3, 4])
346 plt.ylabel('some numbers')
347 plt.show()
348
349 # Activate interactive mode (plot automatically drawn).
350 plt.ion()
351 # Deactivate:
352 plt.ioff()
353
354 # For non-interactive need:
355 plt.show() # Show the plot.
356 plt.draw() # (Re)draw the plot.
357 # plt.draw is used in interactive mode to update a figure that has been
358 # altered, but not automatically re-drawn.
359
360 # Change plotting style.
```

```
361 plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
362
363 # Plot multiple data sets with plotting style string.
364 t = np.arange(0, 5, 0.2)
365 plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
366
367 # The plot function has many arguments.
368 plt.plot(t, t**2, alpha=0.8, color='r', linestyle='--', linewidth=2, label='
369 plt.plot(t, np.sin(t), linestyle='', markeredgecolor='black', markeredgewidt
370          markerfacecolor='r', markersize=15, marker='o', label='second')
371
372 # Add some information.
373 plt.grid()
374 plt.legend()
375 plt.xlabel(r'$x$', fontsize=25)
376 plt.ylabel(r'$y^2 + \int_{0}^{\infty}$', fontsize=25)
377 plt.title('Hello Matplotlib!')
378
379
380 #
381 # Save into a file.
382 #
383 fig.savefig('my_plot.eps')
384 fig.savefig('my_plot.png', dpi=120, format='png')
385
386
387
```