Introduction to Scientific Python

LIF Neurons & Recap

CCNSS 2017

Overview

Overview

LIF neuron simulation

Overview

LIF neuron simulation

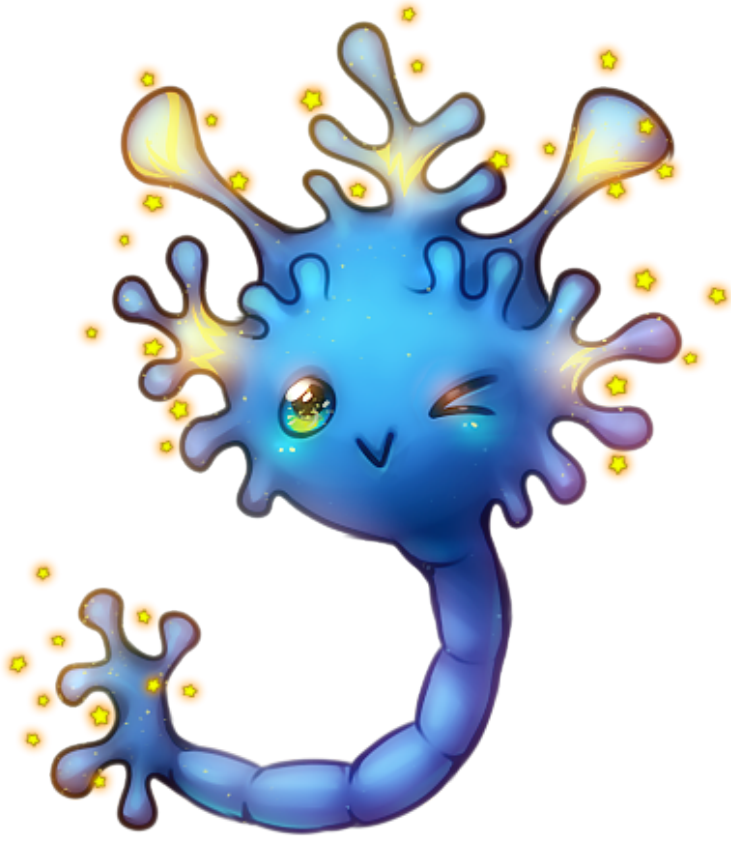
Structure your Python code

Overview

LIF neuron simulation

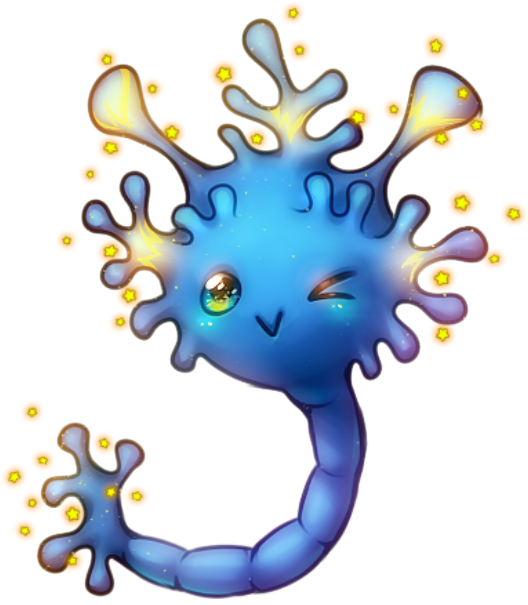
Structure your Python code

Advanced topics



Coding Time!

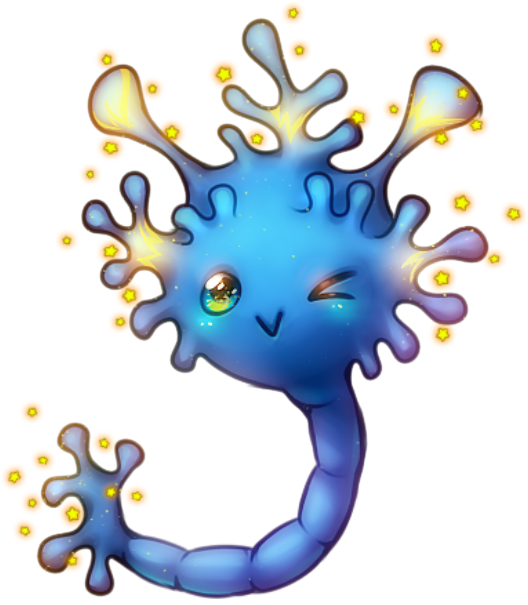
LIF Neuron Exercise



Objective

Implement LIF neuron

LIF Neuron Exercise

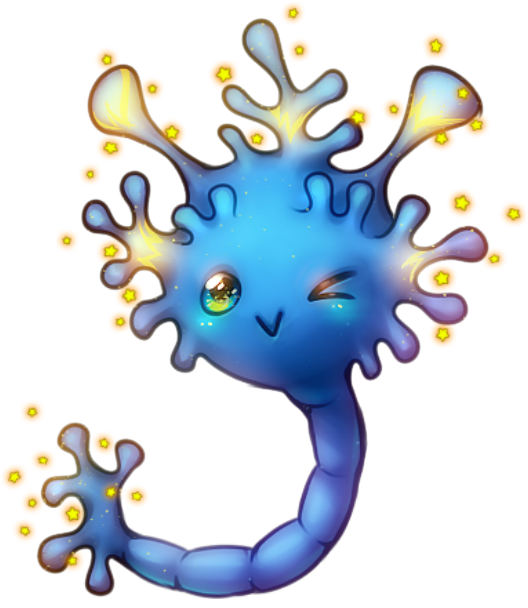


Objective

Implement LIF neuron

Extract ensemble stats

LIF Neuron Exercise



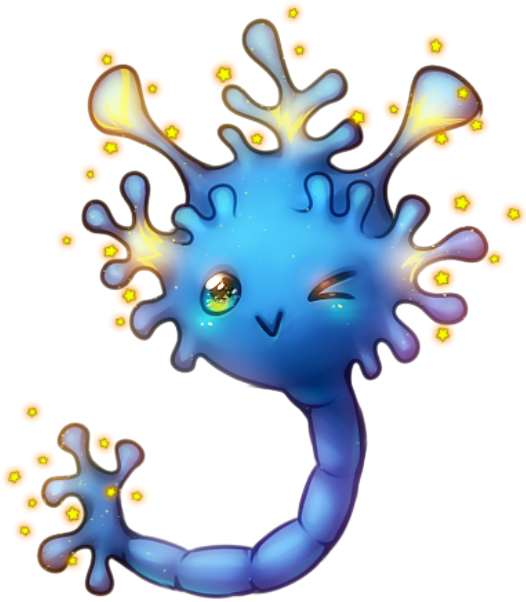
Objective

Implement LIF neuron

Extract ensemble stats

Produce nice graphs!!!

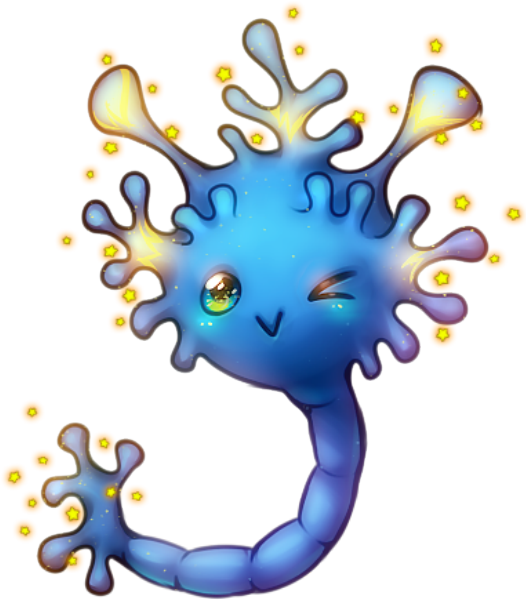
LIF Neuron Exercise



Strategy

No spikes first

LIF Neuron Exercise

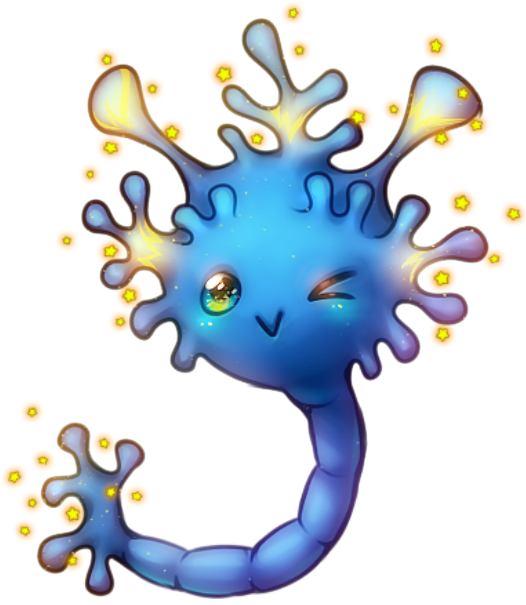


Strategy

No spikes first

Implement ODE integration

LIF Neuron Exercise



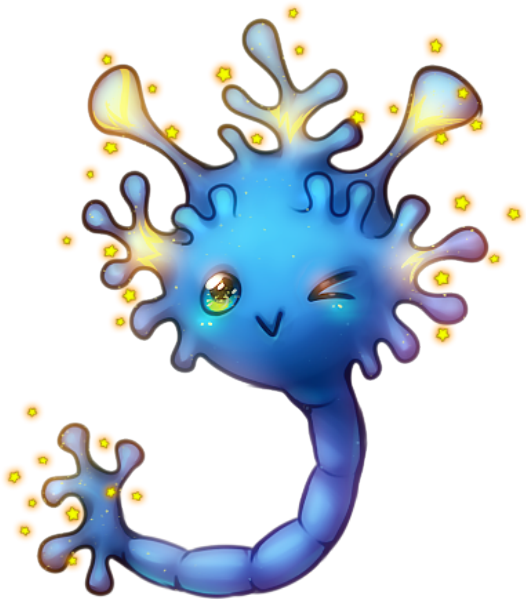
Strategy

No spikes first

Implement ODE integration

Extend to ensemble stats

LIF Neuron Exercise



Strategy

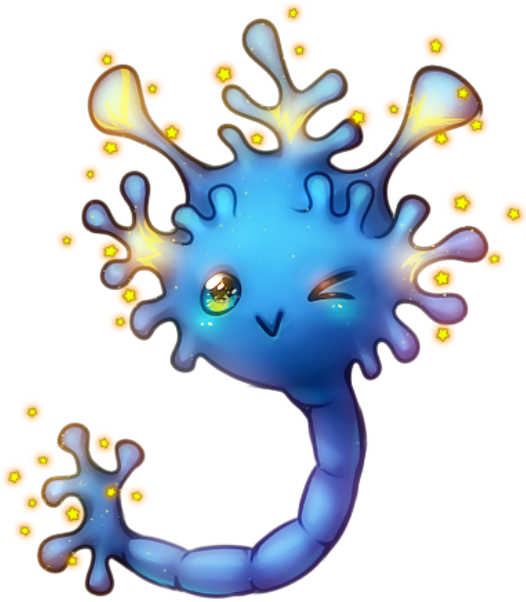
No spikes first

Implement ODE integration

Extend to ensemble stats

Validate stats $\rightleftharpoons$ white noise input

LIF Neuron Exercise



Strategy

No spikes first

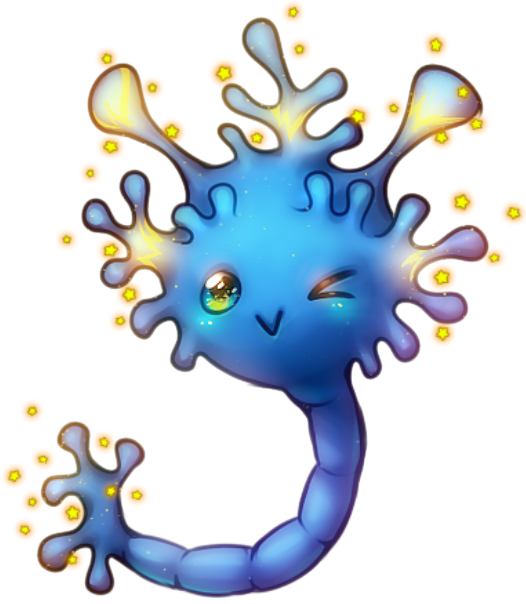
Implement ODE integration

Extend to ensemble stats

Validate stats $\rightleftharpoons$ white noise input

Introduce spikes

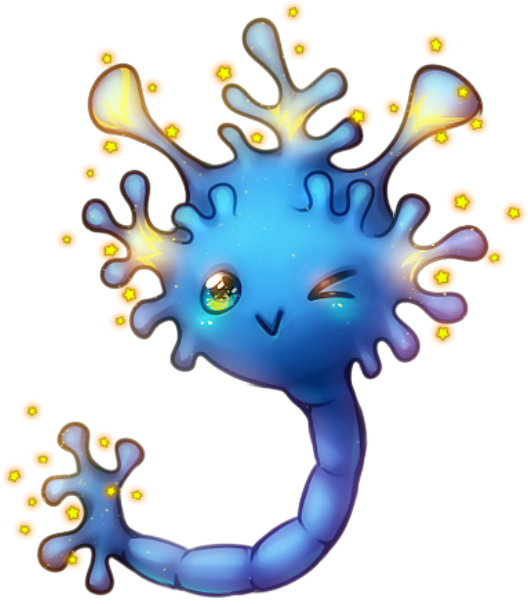
LIF Neuron Exercise



Coding Time!

Start IPython Notebook

LIF Neuron Exercise



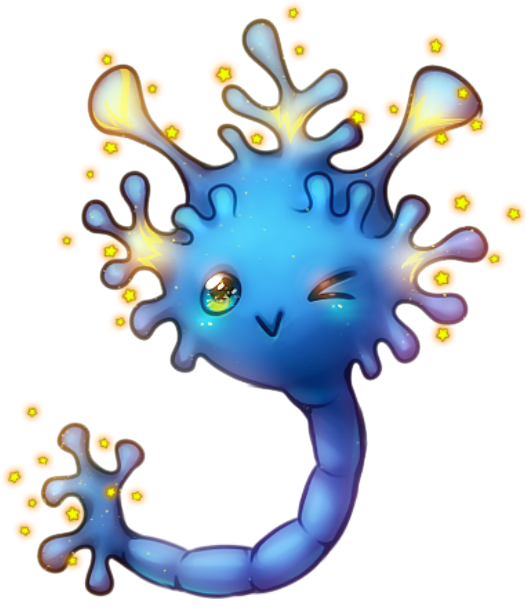
Coding Time!

Start IPython Notebook

(Exercise 1)

Encode simulation parameters

LIF Neuron Exercise



Simulation parameters

```
t_max = 0.15        # second
dt = 1e-3           # second
tau = 20e-3         # second
el = -60e-3         # volt
vr = -70e-3         # volt
vth = -50e-3        # volt
r = 100e-6          # ohm
i_mean = 25e-11     # ampere
```

Control Flow

Control Flow - Loops

While loop

```
t, t_max, dt = 0, 10, 1

while t < t_max:
    print t
    t += dt

print "Finished at value t = ", t
```

Control Flow - Loops

While loop

```
t, t_max, dt = 0, 10, 1

while t < t_max:
    print t
    t += dt

print "Finished at value t = ", t
```

```
0
⋮
9
Finished at value t = 10
```

Control Flow - Loops

For loop

```
for t in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    print t

print "Finished at value t = ", t
```

Control Flow - Loops

For loop

```
for t in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    print t

print "Finished at value t = ", t
```

```
0
⋮
9
Finished at value t = 9
```

Control Flow - Loops

For loop

```
t_max, dt = 10, 1

for t in range(0, t_max, dt):
    print t

print "Finished at value t = ", t
```

Control Flow - Loops

For loop

```
t_max, dt = 10, 1

for t in range(0, t_max, dt):
    print t

print "Finished at value t = ", t
```

```
0
⋮
9
Finished at value t = 9
```

Indentation

Indentation = logical structure

Indentation

Indentation = logical structure

Same spacing = same logical block

Indentation

Indentation = logical structure

Same spacing = same logical block

Use 4 whitespaces (PEP 8)

http://legacy.python.org/dev/peps/pep-0008/

Indentation

```
for t in range(0, t_max, dt):
␣␣␣␣print t
```

Control Flow - Conditional

If statement

```
t_max = 10

if t_max >= 5:
    print "t_max is equal to or more than 5 s"
```

Control Flow - Conditional

If statement

```
t_max = 10

if t_max >= 5:
    print "t_max is equal to or more than 5 s"
```

```
t_max is equal to or more than 5 s
```

Control Flow - Conditional

If-Else statements

```
t_max = 10

if t_max < 5:
    print "t_max is less than 5 s"
else:
    print "t_max is equal to or more than 5 s"
```

Control Flow - Conditional

If-Else statements

```
t_max = 10

if t_max < 5:
    print "t_max is less than 5 s"
else:
    print "t_max is equal to or more than 5 s"
```

```
t_max is equal to or more than 5 s
```

Control Flow - Conditional

If-Elif-Else statements

```
t_max = 10

if t_max < 1:
    print "t_max is less than 1 s"
elif t_max <= 0.5:
    print "t_max is between 1 and 5 s"
else:
    print "t_max is more than 5 s"
```

Control Flow - Conditional

If-Elif-Else statements

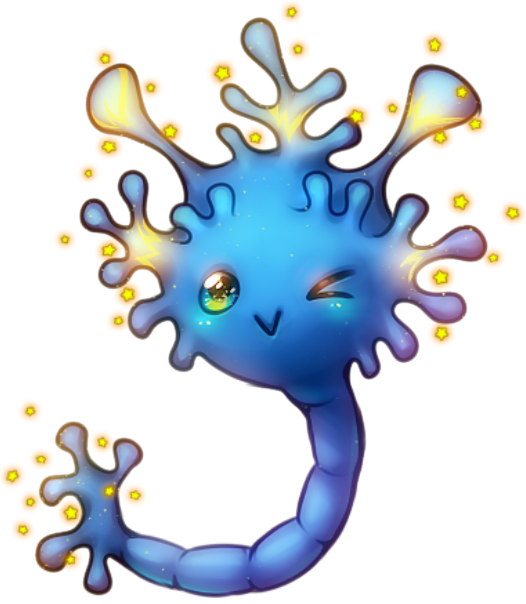
```
t_max = 10

if t_max < 1:
    print "t_max is less than 1 s"
elif t_max <= 0.5:
    print "t_max is between 1 and 5 s"
else:
    print "t_max is more than 5 s"


t_max is more than 5 s
```

Break & Continue

Break and Continue statements

```
t, t_max, dt = 0, 10, 1

while t <= t_max:
    if t > 5:
        print "I'm done!"
        break
    elif t % 2 == 0:
        print t, "is even"
        t += dt
        continue
    t += dt

print "Finished at value t = ", t
```

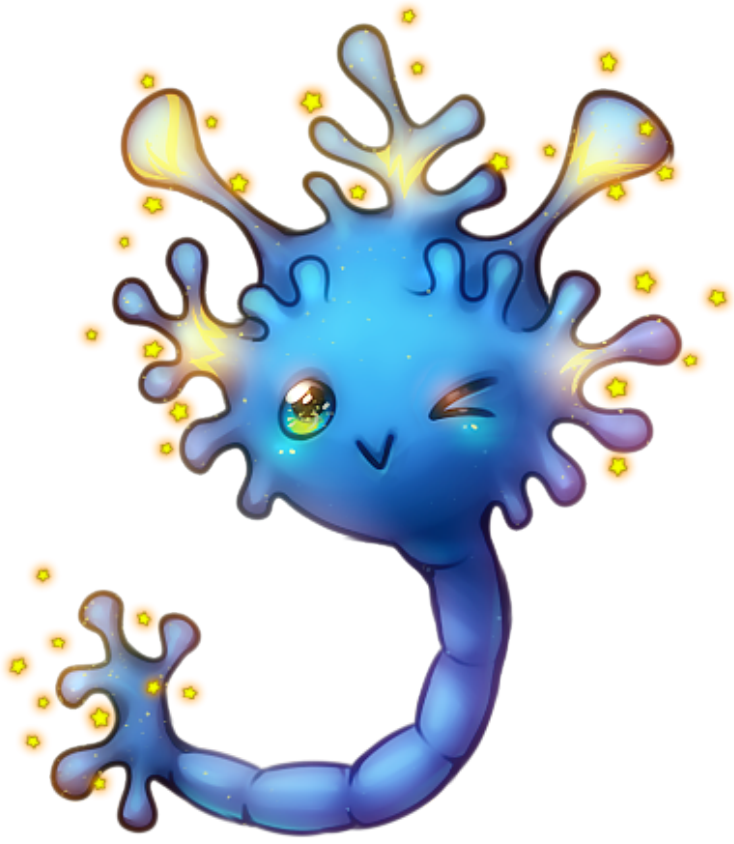
LIF Neuron Exercise



Membrane equation

$$\tau_m \frac{d}{dt} V(t) = E_L - V(t) + RI(t)$$

LIF Neuron Exercise



Coding Time!

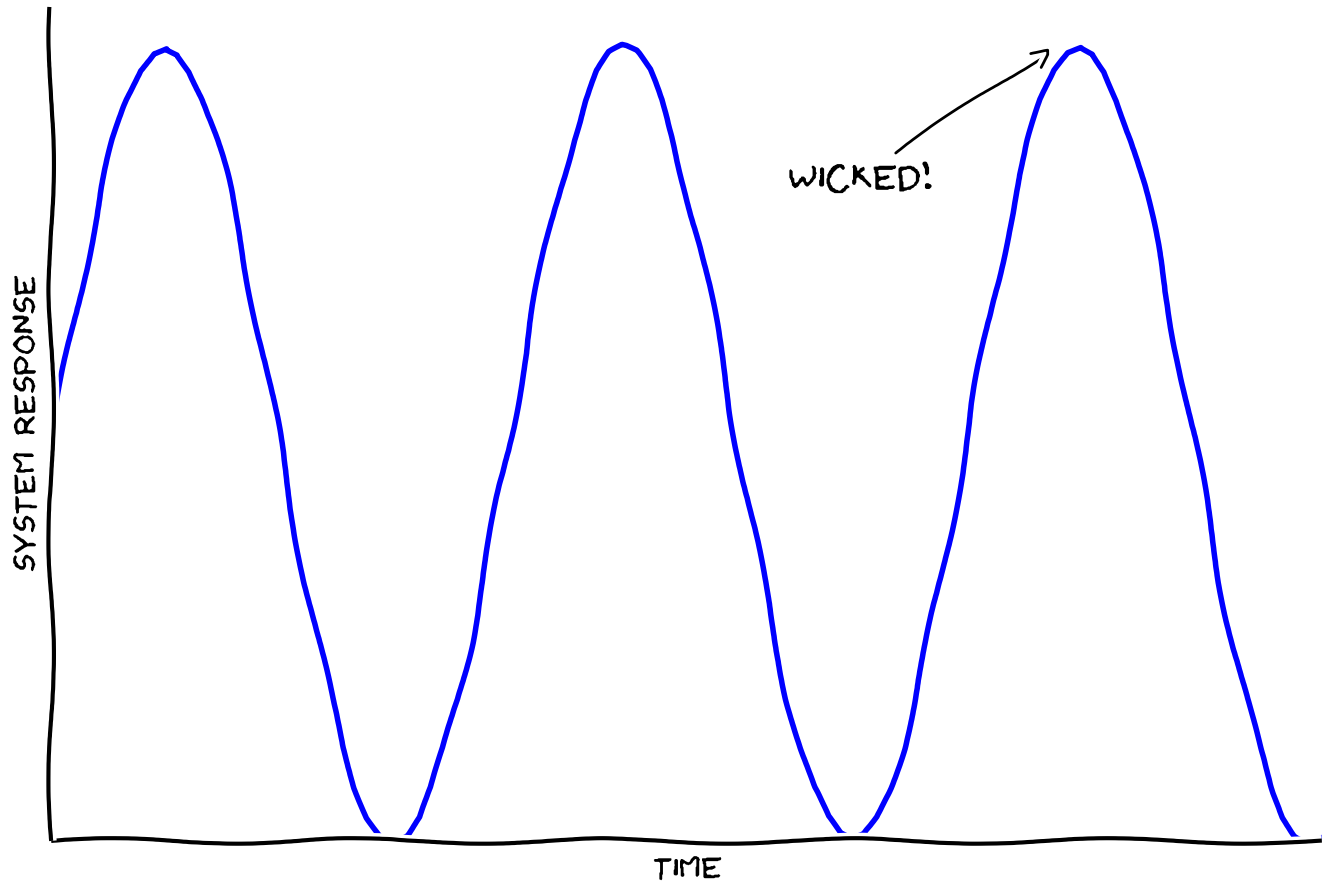
(Exercise 2)

Discrete time integration of $V(t)$

$$V(t+\Delta t) = V(t) + \frac{\Delta t}{\tau_m}(\cdots)$$

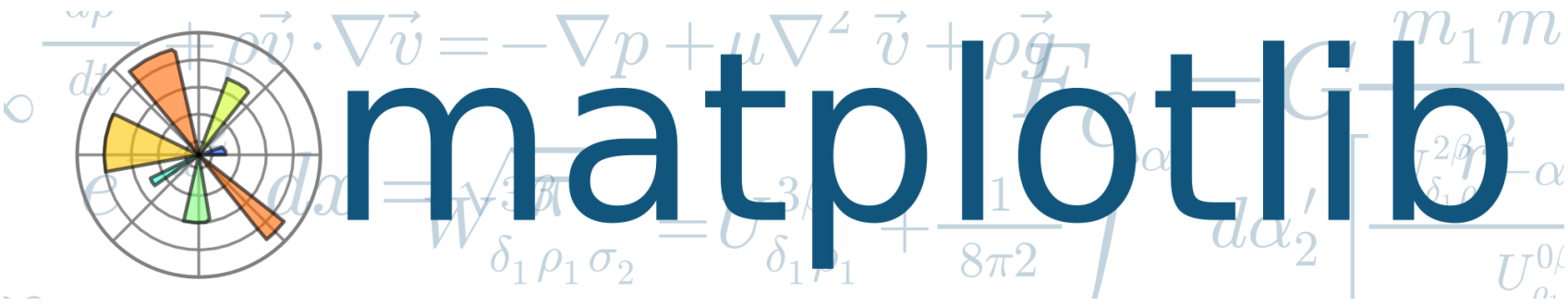
Plotting

Showing Your Stuff



SOME OSCILLATORY SYSTEM

Matplotlib Library



Simple Plot

Key function:

```
plot(x, y, 'r+', label='cross')
```

Simple Plot

Key function:

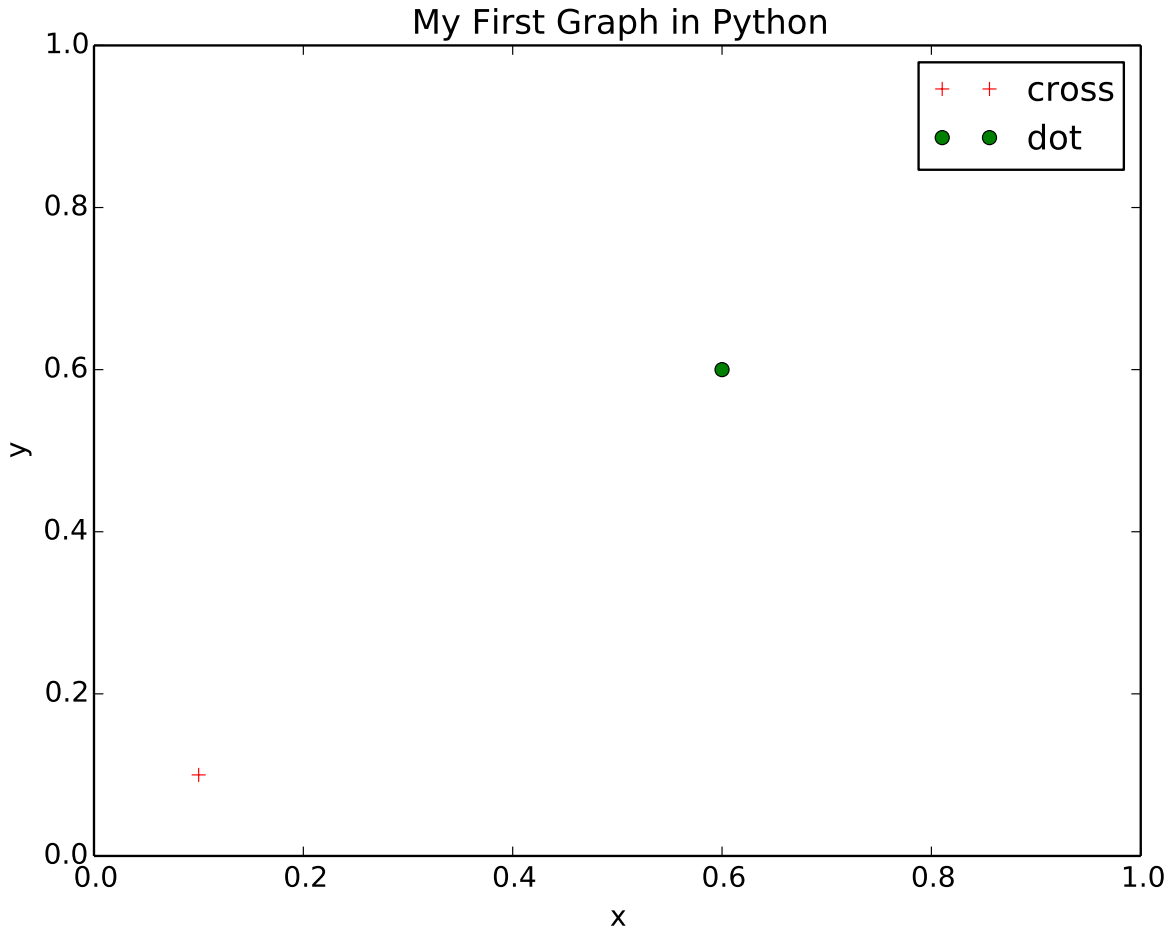
```
plot(x, y, 'r+', label='cross')
```

will plot a red cross at position (x, y) with label 'cross'

Simple Plot

```
import matplotlib.pyplot as plt

x1, y1, x2, y2 = 0.1, 0.1, 0.6, 0.6

plt.figure()
plt.plot(x1, y1, 'r+', label='cross')
plt.plot(x2, y2, 'go', label='dot')

plt.title('My First Graph in Python')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()

plt.show()
```

Simple Plot


My First Graph in Python
cross
dot
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x

Plotting Lists

```
x = range(10)
print x
```

Plotting Lists

```
x = range(10)
print x
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

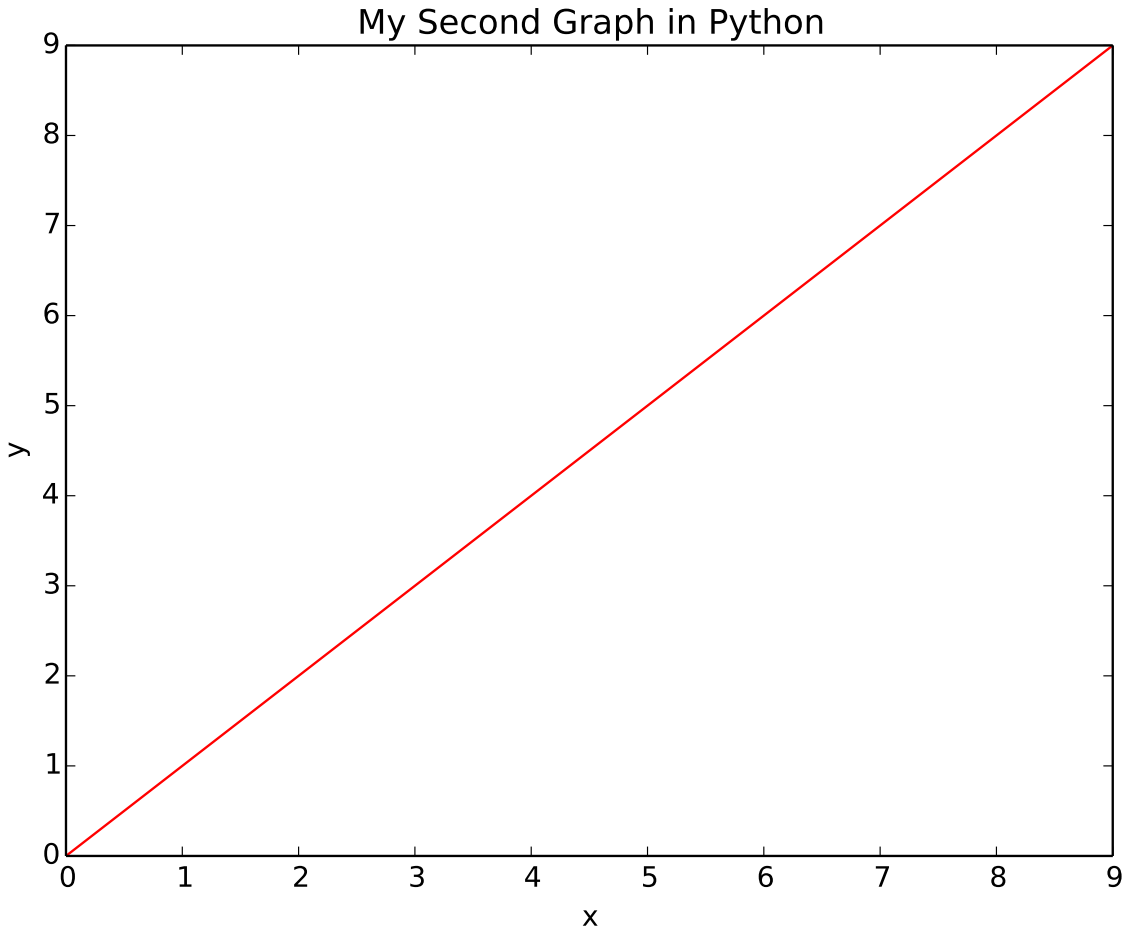
Plotting Lists

```
x = range(10)
print x
```

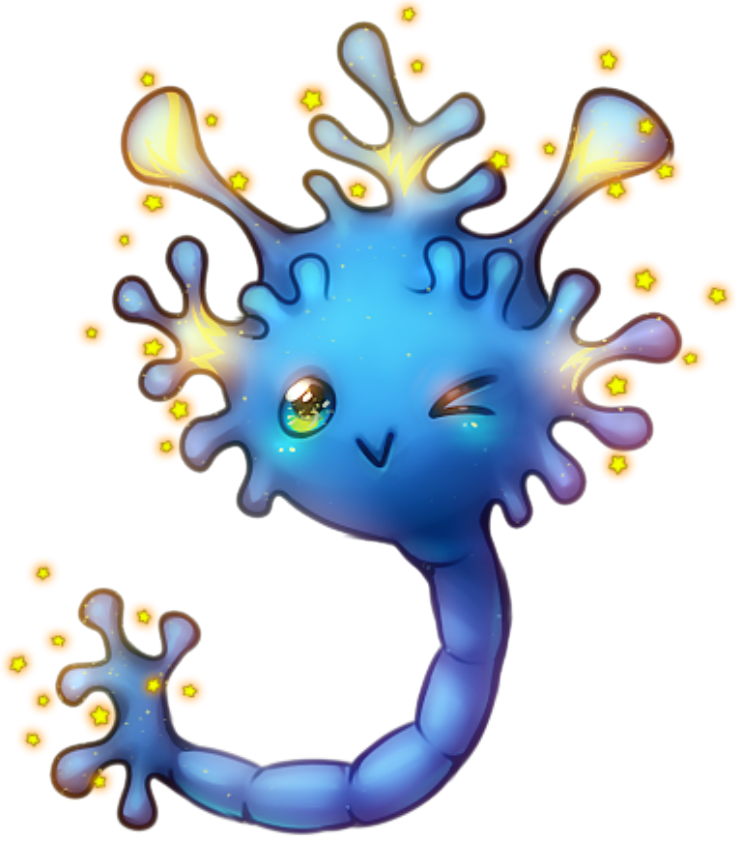
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
plot(x, x, 'ro')
```

Simple Plot II



LIF Neuron Exercise



Coding Time!

(Exercise 3)

Plot $V(t)$ time course

(Exercise 4)

Stochastic input currents

List Indexing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[0]
```

List Indexing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[0]
```

```
100
```

List Indexing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[0]
```

```
100
```

```
mylist = [100, 1000.0, "John", (0.5+0.5j), 10.0]
del mylist[-1]
print mylist
```

List Indexing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[0]
```

```
100
```

```
mylist = [100, 1000.0, "John", (0.5+0.5j), 10.0]
del mylist[-1]
print mylist
```

```
[100, 1000.0, 'John', (0.5+0.5j)]
```

List Slicing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[1:3]
```

List Slicing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[1:3]
```

```
[1000.0, 'John']
```

List Slicing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[1:3]
```

```
[1000.0, 'John']
```

```
print mylist[1:]
```

List Slicing

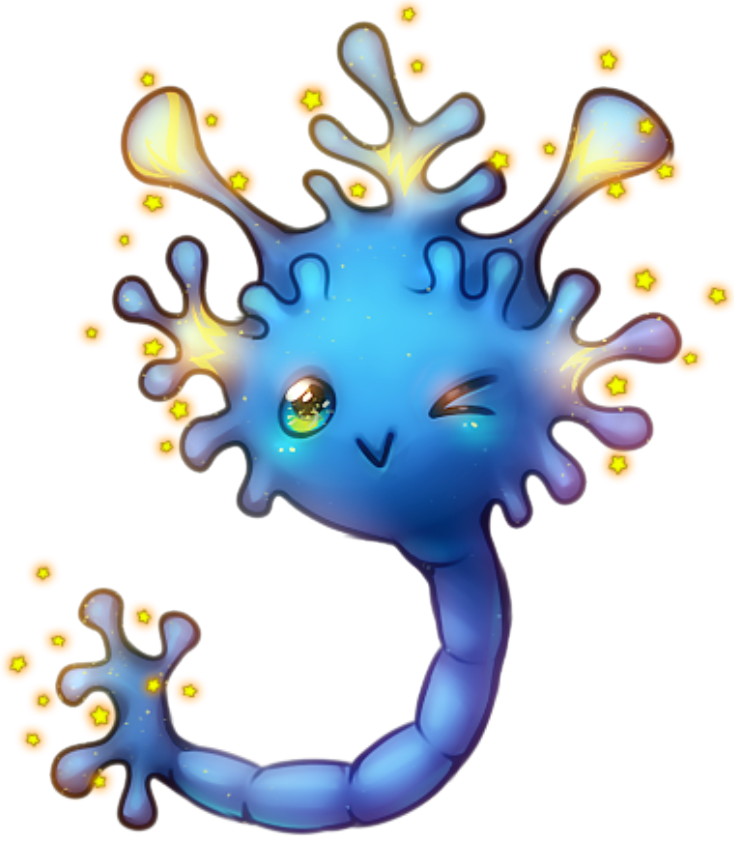
```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[1:3]
```

```
[1000.0, 'John']
```

```
print mylist[1:]
```

```
[1000.0, 'John', 0.5 + 0.5j]
```

LIF Neuron Exercise



Coding Time!

(Exercise 5, 6, 7 and 8)

Ensemble statistics

Standard Variable Types - Dictionary

```
mydict = {'qty':  100, 'person':  "John"}
print mydict
```

Standard Variable Types - Dictionary

```
mydict = {'qty':  100, 'person':  "John"}
print mydict
```

```
{'person':  'John', 'qty':  100}
```

Standard Variable Types - Dictionary

```
mydict = {'qty':  100, 'person':  "John"}
print mydict
```

```
{'person':  'John', 'qty':  100}
```

```
print mydict['person']
```

Standard Variable Types - Dictionary

```
mydict = {'qty':  100, 'person':  "John"}
print mydict
```

```
{'person':  'John', 'qty':  100}
```

```
print mydict['person']
```

```
John
```

Standard Variable Types - Dictionary

```
mydict = {'qty':  100, 'person':  "John"}
print mydict.keys()
```

Standard Variable Types - Dictionary

```
mydict = {'qty':  100, 'person':  "John"}
print mydict.keys()
```

```
['person', 'qty']
```

Standard Variable Types - Dictionary

```
mydict = {'qty':  100, 'person':  "John"}
print mydict.keys()
```

```
['person', 'qty']
```

```
print mydict.values()
```

Standard Variable Types - Dictionary

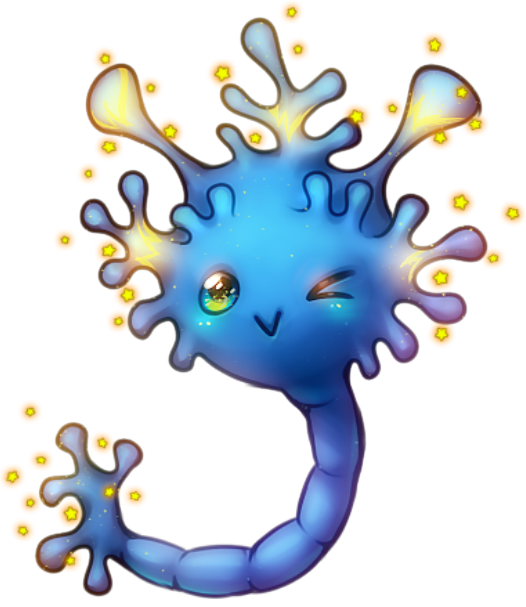
```
mydict = {'qty':  100, 'person':  "John"}
print mydict.keys()
```

```
['person', 'qty']
```

```
print mydict.values()
```

```
['John', 100]
```

LIF Neuron Exercise



Membrane equation
with reset condition

If $V(t) < V_{th}$

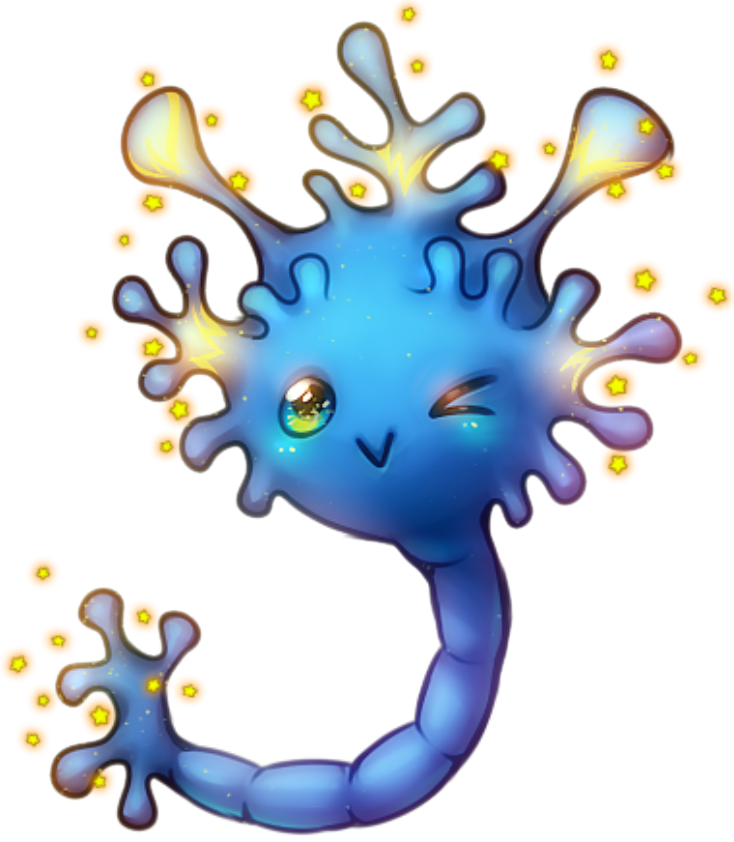
$$\tau_m \frac{d}{dt} V(t) = E_L - V(t) + RI(t)$$

Else

$$V(t) = V_r$$

record spike at time t

LIF Neuron Exercise



Coding Time!

(Exercise 9)

Output spikes

(Exercise 10)

Refractory period
Integration step

Structure your Python code

Structure your Python code

Functions

Structure your Python code

Functions

Modules

Structure your Python code

Functions

Modules

Packages

Functions

Functions

```
def mysum(a, b):
  '''Return a + b'''
  return a + b
```

Functions

```
def mysum(a, b):
  '''Return a + b'''
  return a + b

print mysum(1, 2)
```

```
3
```

Functions

```
def mysum(a, b):
  '''Return a + b'''
  return a + b


print mysum(1, 2)
```

```
3
```

```
help(mysum)
```

```
Help on function mysum in module __main__:

mysum(a, b=2)
 Return a + b
```

Functions

Call by argument names

```
print mysum(a=1, b=2)
```

```
3
```

Functions

Call by argument names

```
print mysum(a=1, b=2)
```

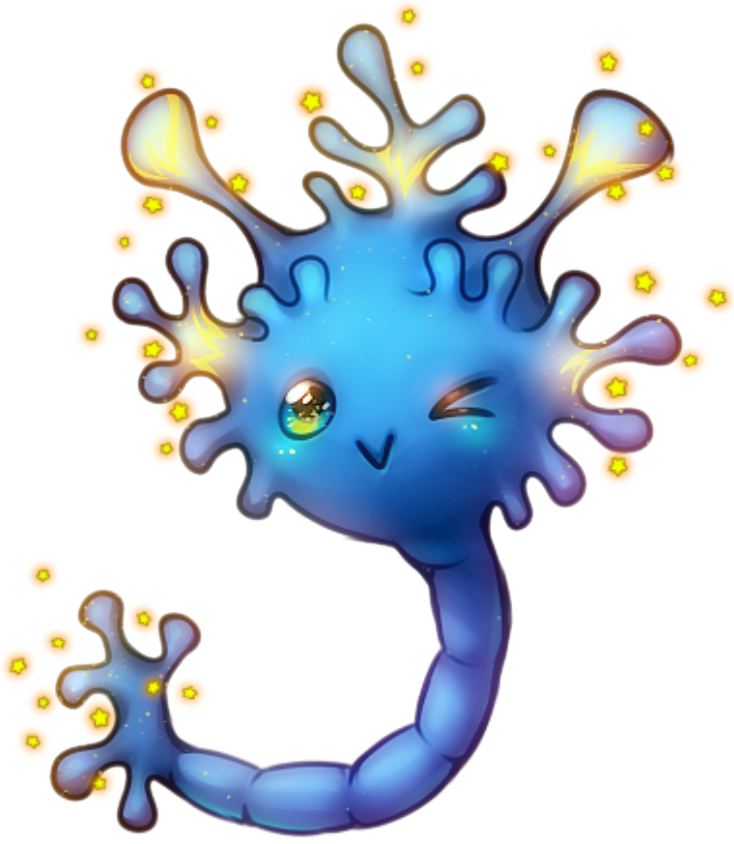
```
3
```

Mandatory arguments vs default values

```
def mysum(a, b=2):
  '''Return a + 2 or a + b'''
  return a + b

print mysum(1)
```

```
3
```

LIF Neuron Exercise



Coding Time!

(Exercise 11)

Use functions

Modules

Modules

```
def mysum(a, b=2):
  '''Return a + 2 or a + b'''
  return a + b
```

Save as file mymath.py

Modules

```
def mysum(a, b=2):
  '''Return a + 2 or a + b'''
  return a + b
```

Save as file mymath.py

```
import mymath

print mymath.mysum(1, 2)
```

```
3
```

Import Types

```
import mymath as mm

print mm.mysum(1, 2)
```

```
3
```

Import Types

```
import mymath as mm

print mm.mysum(1, 2)
```

```
3
```

```
from mymath import mysum

print mysum(1, 2)
```

```
3
```

Import Types

```
import mymath as mm

print mm.mysum(1, 2)
```

```
3
```

```
from mymath import mysum

print mysum(1, 2)
```

```
3
```

```
from mymath import *

print mysum(1, 2)
```

```
3
```

Modules

Word of advice: **be explicit!**

Modules

Word of advice: **be explicit!**

`np.array, plt.plot`

...you'll get used to it.

Packages

Numpy

Numpy

Fundamental package for scientific computing

Numpy

Fundamental package for scientific computing

Linear algebra, Fourier transform and random numbers

Numpy

Fundamental package for scientific computing

Linear algebra, Fourier transform and random numbers

N-dimensional `array` object

Numpy

Fundamental package for scientific computing

Linear algebra, Fourier transform and random numbers

N-dimensional `array` object

Broadcasting functions

Numpy

Fundamental package for scientific computing

Linear algebra, Fourier transform and random numbers

N-dimensional `array` object

Broadcasting functions

Integrate C/C++ and Fortran code

Scipy

Partner of Numpy package

Scipy

Partner of Numpy package

Fundamental library for scientific computing 

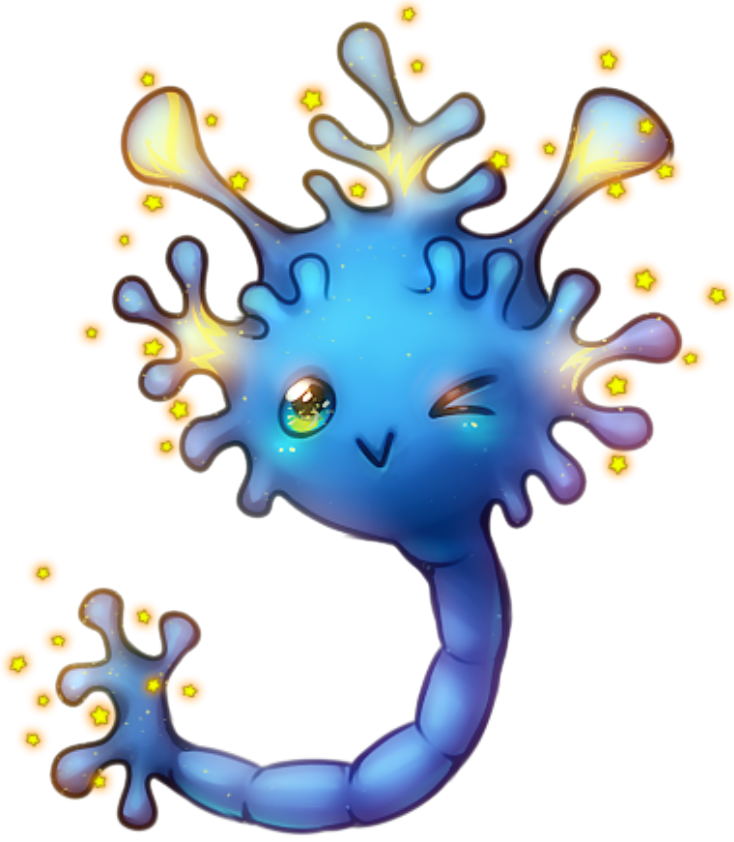
Scipy

Partner of Numpy package

Fundamental library for scientific computing 

Special functions
Integration
Optimization
Interpolation
Signal Processing
Statistics
Multidimensional image processing
...

LIF Neuron Exercise



Coding Time!

(Exercise 12)

Using NumPy

Advanced Topics

List Comprehensions

One-liner `for` loops

```
squares = []
for x in range(5):
    squares += [x**2]
print squares
```

List Comprehensions

One-liner `for` loops

```
squares = []
for x in range(5):
    squares += [x**2]
print squares
```

```
[0, 1, 4, 9, 16]
```

List Comprehensions

One-liner `for` loops

```
squares = []
for x in range(5):
    squares += [x**2]
print squares
```

```
[0, 1, 4, 9, 16]
```

```
squares = [x**2 for x in range(10)]
print squares
```

List Comprehensions

One-liner `for` loops

```
squares = []
for x in range(5):
    squares += [x**2]
print squares
```

```
[0, 1, 4, 9, 16]
```

```
squares = [x**2 for x in range(10)]
print squares
```

```
[0, 1, 4, 9, 16]
```

Enumerate Construct

Returning indexes and elements

```
mylist = ['pyramidal', 'inhibitory', 'glial']
for idx, item in enumerate(mylist):
    print idx, item
```

Enumerate Construct

Returning indexes and elements

```
mylist = ['pyramidal', 'inhibitory', 'glial']
for idx, item in enumerate(mylist):
    print idx, item
```

```
0 pyramidal
1 inhibitory
2 glial
```

Standard Variable Types - Tuples

Tuples are read-only lists

```
mytuple = (100, 1000.0, "John", 0.5 + 0.5j)
print mytuple[0:1]
```

Standard Variable Types - Tuples

Tuples are read-only lists

```
mytuple = (100, 1000.0, "John", 0.5 + 0.5j)
print mytuple[0:1]
```

```
(100,)
```

Standard Variable Types - Tuples

Tuples are read-only lists

```
mytuple = (100, 1000.0, "John", 0.5 + 0.5j)
print mytuple[0:1]
```

```
(100,)
```

```
print mytuple[0:1][0]
```

Standard Variable Types - Tuples

Tuples are read-only lists

```
mytuple = (100, 1000.0, "John", 0.5 + 0.5j)
print mytuple[0:1]
```

```
(100,)
```

```
print mytuple[0:1][0]
```

```
100
```

Standard Variable Types - Sets

Return unique elements of lists and tuples

```
myset = set([1,1,2,3,4])
print myset
```

Standard Variable Types - Sets

Return unique elements of lists and tuples

```
myset = set([1,1,2,3,4])
print myset
```

```
set([1, 2, 3, 4])
```

That's all folks!