Testing

Vorbereitung: Python -> Jupyter notebook

- Mit Jupyter notebook Python interaktiv ausführen
- Gut geeignet für kleine Auswertungen, Plots für die Arbeit oder zum Ausprobieren
- Nicht wirklich geeignet für große Programme (dafür besser mit einer IDE arbeiten (disclaimer: ich selber arbeite normalerweise in Java)

Starting Jupyter notebook environment (1)

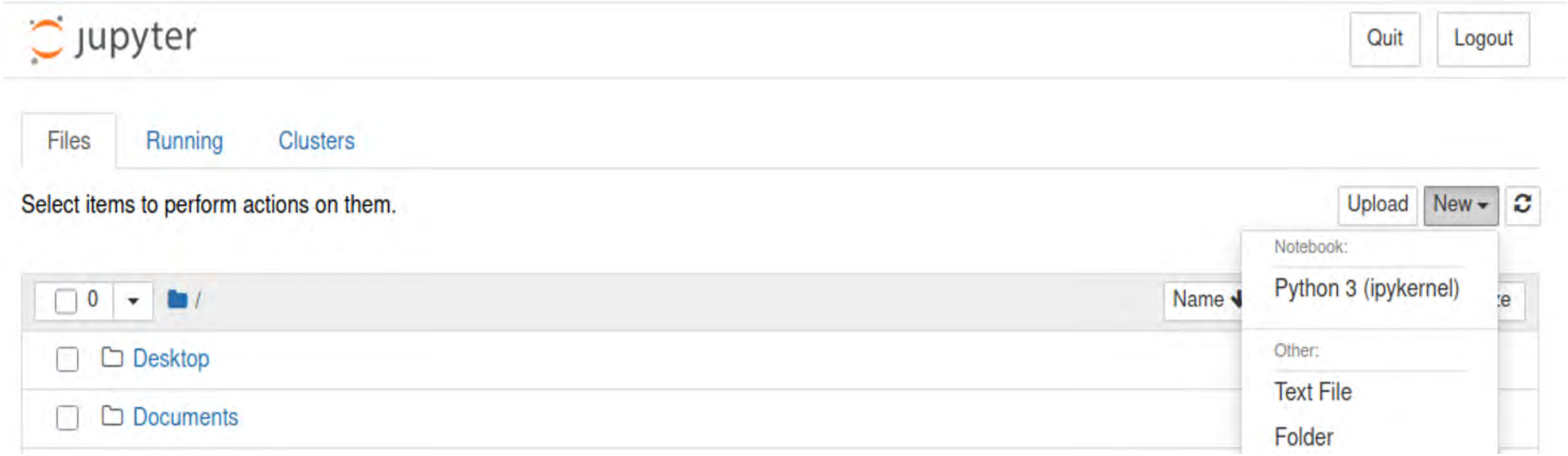
- Loggen Sie sich auf einem der zitipool Rechner ein und starten Sie eine shell (“terminal”)
- Gehen Sie einmal in das Verzeichnis /shares/ziti-opt/ (damit es gemounted wird und Sie Pfad-Vervollständigung in der Shell nutzen können)
- Dann zurück in Ihr Home-Directory
- Starten Sie eine shell in einem apptainer image:
  apptainer shell /shares/ziti-opt/software/nca/jul25.sif
- Überprüfen Sie, ob Sie sich in der apptainer shell in Ihrem home-Directory befinden: pwd

Jupyter notebook environment (2)

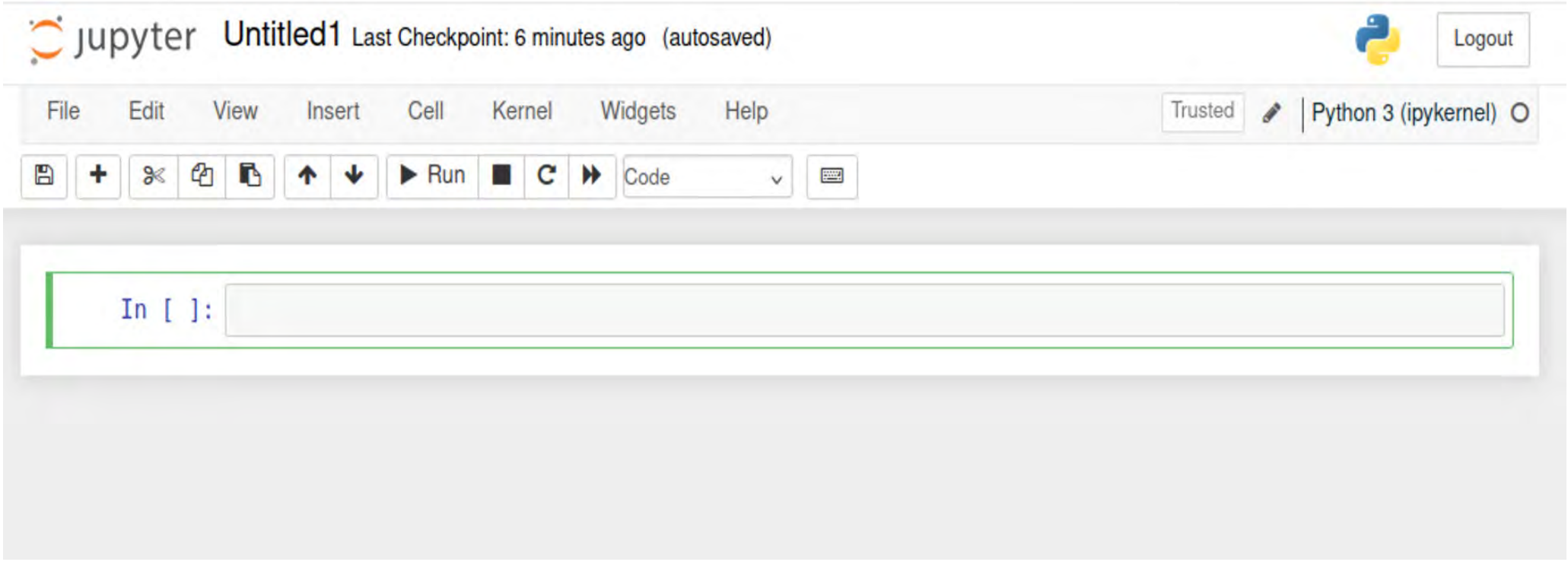
- Starten Sie einen Jupyter Prozess in Ihrer apptainer session:
  jupyter notebook

  Es wird eine URL angezeigt, unter der Sie Ihren Server erreichen können (und es sollte automatisch ein Browser für diese URL gestartet werden)

Im Browser sehen Sie jetzt eine Liste der Dateien in Ihrem home directory



Mit New > Python 3 (ipykernel) können Sie ein neues “Notebook” anlegen



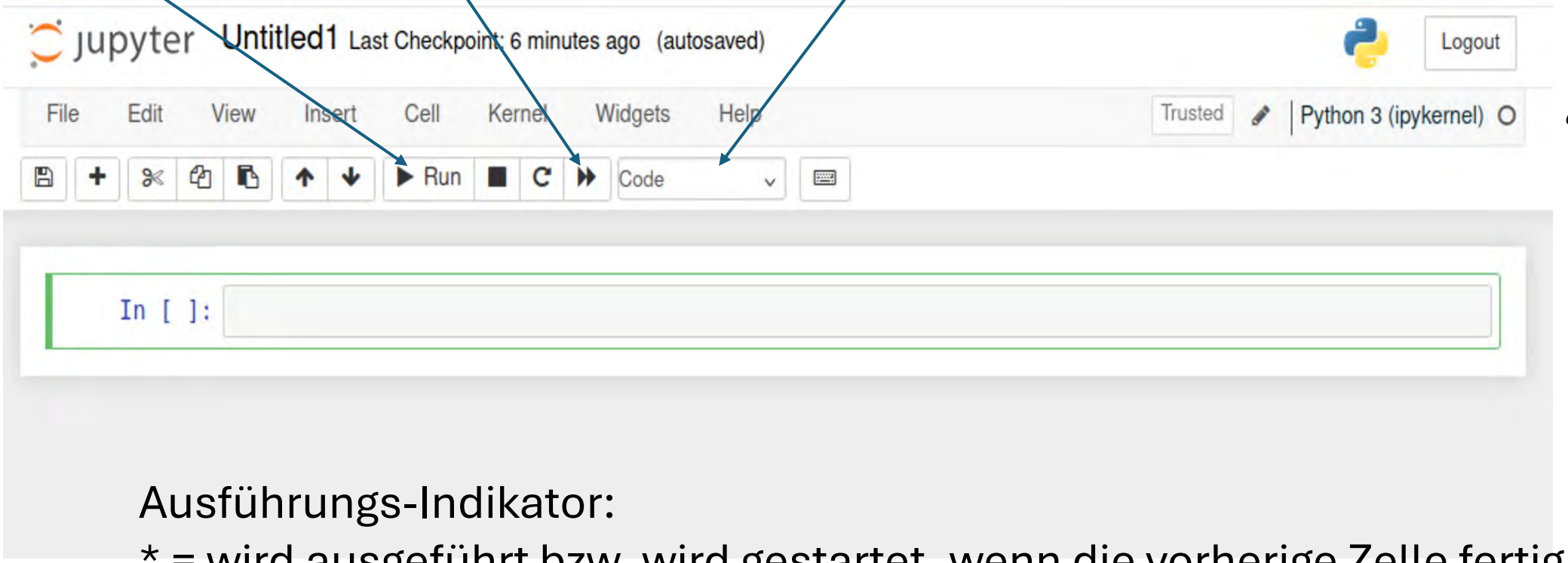
“Kernel” info

Zelle

Kernel neu starten und alle Zellen ausführen

Zelle ausführen

Typ der Zelle (Code, Markdown, ...)

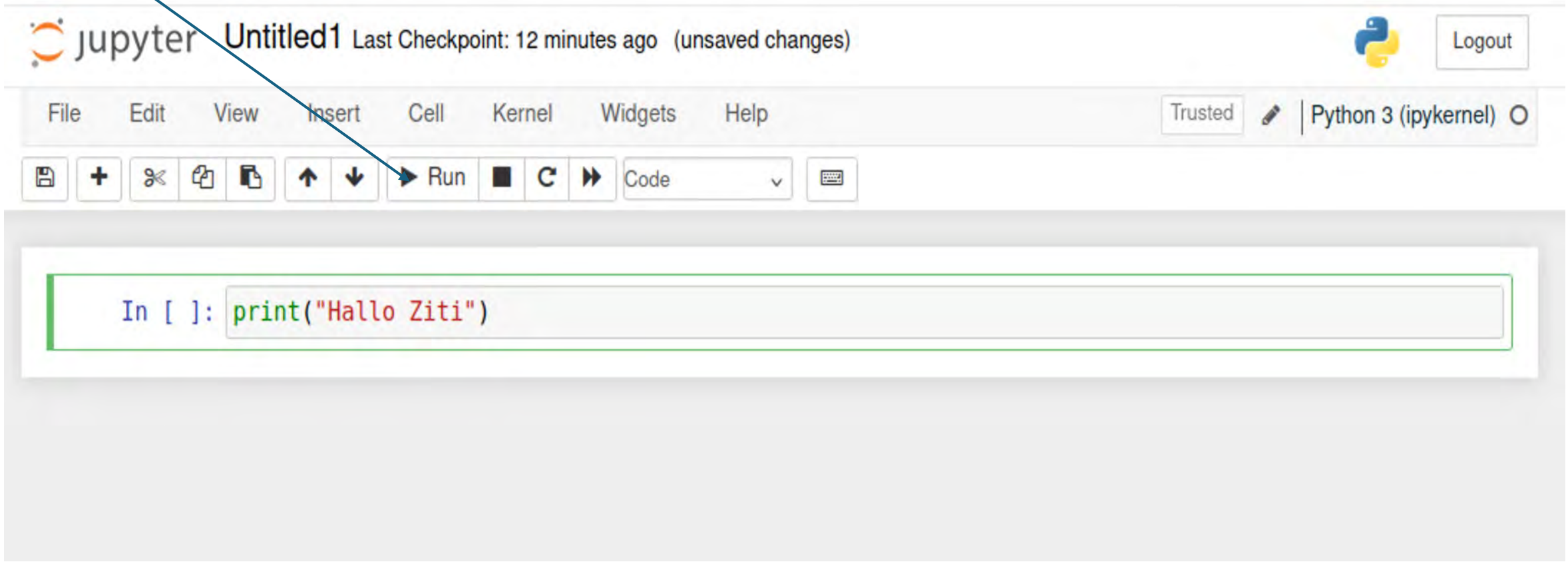


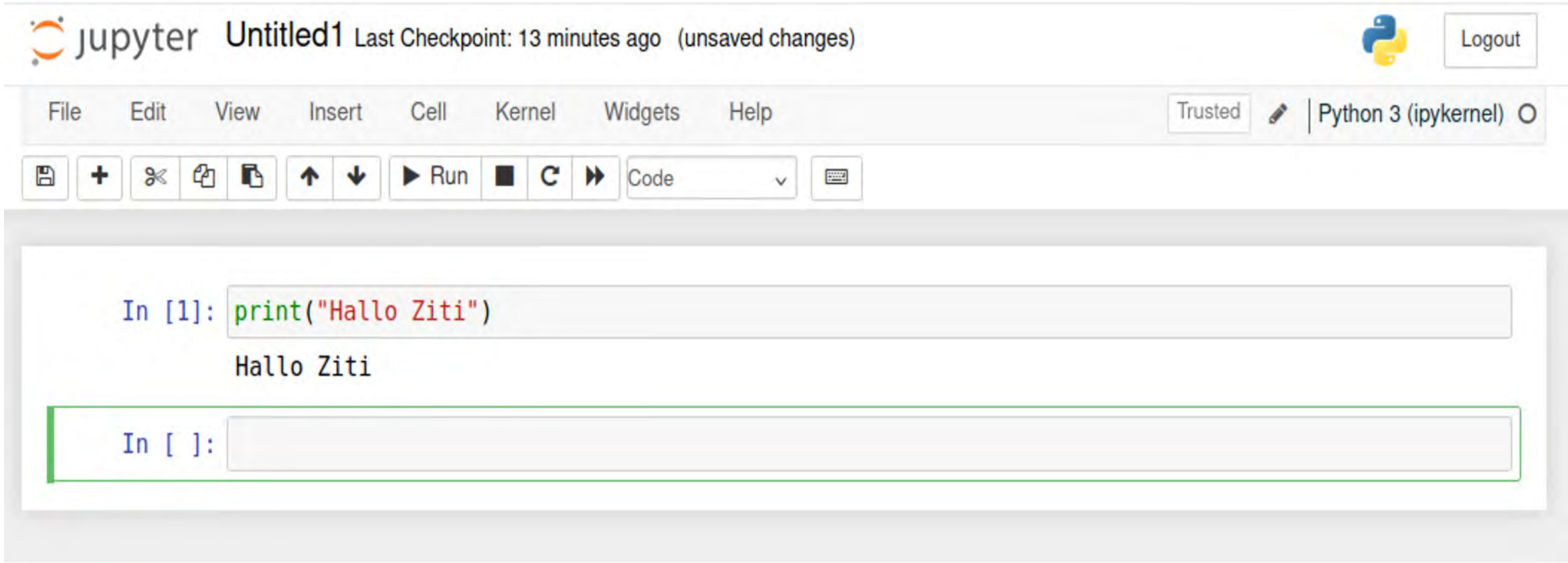
"Kernel" info

Zelle

Ausführungs-Indikator:
* = wird ausgeführt bzw. wird gestartet, wenn die vorherige Zelle fertig ausgeführt wurde
Zahl = fortlaufender Zähler der Ausführungen

Zelle ausführen klicken (oder SHIFT ENTER)




jupyter
Untitled1
Last Checkpoint: 13 minutes ago
(unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3 (ipykernel)
Run
Code
In [1]:
print("Hallo Ziti")
Hallo Ziti
In [ ]:

Jupyter notebook und matplotlib

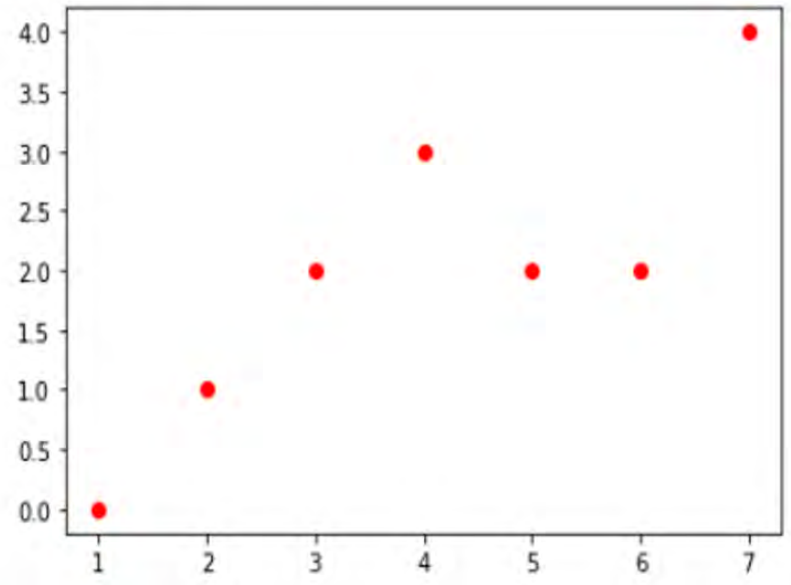
jupyter UsingWithJupyter Last Checkpoint: 2 hours ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

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

In [2]:
```
x=[1,2,3,4,5,6,7]
y=[0,1,2,3,2,2,4]
plt.plot(x,y,"ro")
plt.show()
```

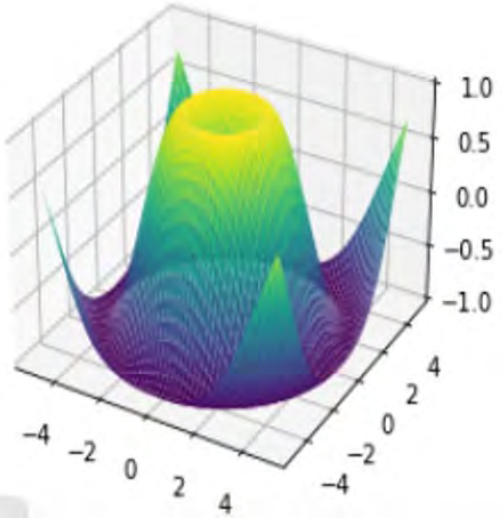


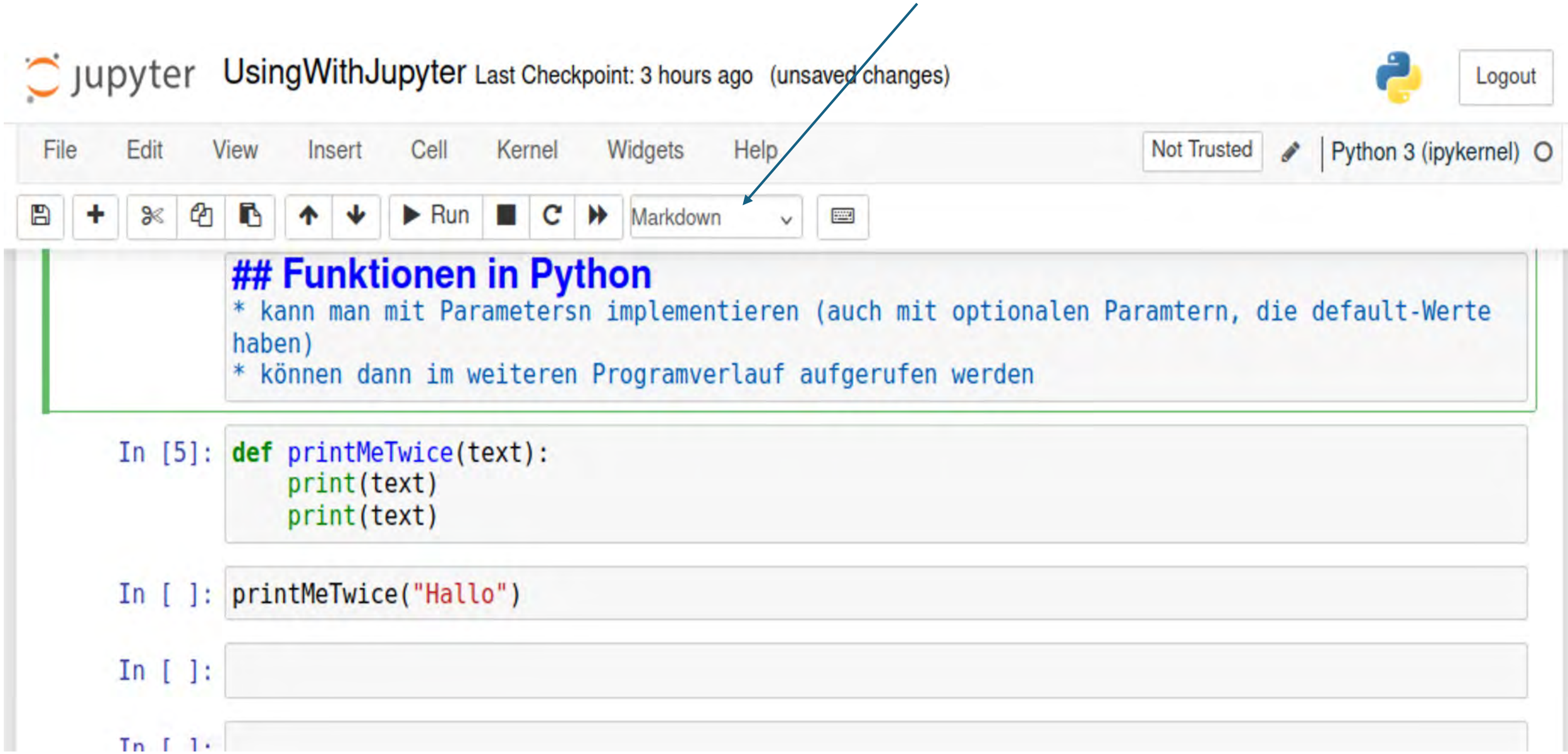
Kann das was?

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

In [2]:
```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
stepsize=0.01
X = np.arange(-5, 5, stepsize)
Y = np.arange(-5, 5, stepsize)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
ax.plot_surface(X, Y, Z,  cmap='viridis')
```

Out[2]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0xffff8c5c7a30>
```




jupyter UsingWithJupyter Last Checkpoint: 3 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Markdown
Funktionen in Python
* kann man mit Parametersn implementieren (auch mit optionalen Paramtern, die default-Werte haben)
* können dann im weiteren Programverlauf aufgerufen werden
In [5]:
def printMeTwice(text):
print(text)
print(text)
In [ ]:
printMeTwice("Hallo")
In [ ]:
In [ ]:

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

Funktionen in Python

- kann man mit Parametersn implementieren (auch mit optionalen Paramtern, die default-Werte haben)
- können dann im weiteren Programverlauf aufgerufen werden

In [7]:
```
def printMeTwice(text):
    print(text)
    print(text)
```

In [8]:
```
printMeTwice("Hallo")
```

```
Hallo
Hallo
```

In [ ]:

Klassen in Python

In [1]:
```
class EineKlasse:
    info="Wert aus der Klasse"

    def __init__(self,wert):
        self.info=wert

    def eineFunktion(self,text):
        print(text)
        print(self.info)
```

In [2]:
```
a=EineKlasse("mit Wert")
a.eineFunktion("ein Text")
```

```
ein Text
mit Wert
```

Testing: test driven development

Grundgedanke:

- Zuerst den Test schreiben (der dann natürlich erst mal fehl schlägt, denn die Funktion gibt es ja noch nicht)
- Erst dann die Funktion implementieren – und zwar nur so weit, so daß der Test erfüllt wird.

Jetzt eine Klasse (mit unittest.TestCase als parent class) mit unittest testen: Methoden mit dem Namen test.... werden ausgeführt

Da es die Methode minus noch nicht gibt schlägt der Test fehl.

In [1]:
```
import unittest
```

In [3]:
```
class TestCal(unittest.TestCase):
    def test_minus(self):
        self.assertEqual(self.minus(1,0,1),1)


unittest.main(argv=['-v'], verbosity=3, exit=False)
```

```
test_minus (__main__.TestCal) ... ERROR

======================================================================
ERROR: test_minus (__main__.TestCal)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/tmp/ipykernel_691111/331199701.py", line 3, in test_minus
    self.assertEqual(self.minus(1,0,1),1)
AttributeError: 'TestCal' object has no attribute 'minus'

----------------------------------------------------------------------
Ran 1 test in 0.005s

FAILED (errors=1)
```

Out[3]:
```
<unittest.main.TestProgram at 0xffff8a32d990>
```

Die minimalste Methode lässt den Test gelingen. Den Fehler in der Methode konnte dieser Test nicht finden. Also muß ein weiterer Test her....

In [4]:
```
class TestCal(unittest.TestCase):
    def test_minus(self):
        self.assertEqual(self.minus(1,0,1),1)

    def minus(self,startWert,minusAmount,anzahl):
        return startWert


unittest.main(argv=['-v'], verbosity=3, exit=False)
```

```
test_minus (__main__.TestCal) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.007s

OK
```

Out[4]:
```
<unittest.main.TestProgram at 0xffff8814bb20>
```

In [5]:
```
class TestCal(unittest.TestCase):
    def test_minus(self):
        self.assertEqual(self.minus(1,0,1),1)
        self.assertEqual(self.minus(1,0.1,1),0.9)

    def minus(self,startWert,minusAmount,anzahl):
        return startWert


unittest.main(argv=['-v'], verbosity=3, exit=False)
```

```
test_minus (__main__.TestCal) ... FAIL

======================================================================
FAIL: test_minus (__main__.TestCal)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/tmp/ipykernel_691111/1021537324.py", line 4, in test_minus
    self.assertEqual(self.minus(1,0.1,1),0.9)
AssertionError: 1 != 0.9

----------------------------------------------------------------------
Ran 1 test in 0.003s

FAILED (failures=1)
```

Out[5]:
```
<unittest.main.TestProgram at 0xffff8814bb50>
```

Diese Methode kann jetzt abziehen ... und die Test-cases klappen

In [6]:
```
class TestCal(unittest.TestCase):
    def test_minus(self):
        self.assertEqual(self.minus(1,0,1),1)
        self.assertEqual(self.minus(1,0.1,1),0.9)

    def minus(self,startWert,minusAmount,anzahl):
        for i in range(0,anzahl):
            startWert-=minusAmount
        return startWert


unittest.main(argv=['-v'], verbosity=3, exit=False)
```

```
test_minus (__main__.TestCal) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
```

Out[6]:
```
<unittest.main.TestProgram at 0xffff8a2d34f0>
```

Was wird jetzt passieren – mit dem 3. Test?

In [10]:

```
class TestCal(unittest.TestCase):
    def test_minus(self):
        self.assertEqual(self.minus(1,0,1),1)
        self.assertEqual(self.minus(1,0.1,1),0.9)
        self.assertTrue(self.minus(1,0.1,10)==0)

    def minus(self,startWert,minusAmount,anzahl):
        for i in range(0,anzahl):
            startWert-=minusAmount
        return startWert
```

In [10]:

```python
class TestCal(unittest.TestCase):
    def test_minus(self):
        self.assertEqual(self.minus(1,0,1),1)
        self.assertEqual(self.minus(1,0.1,1),0.9)
        self.assertTrue(self.minus(1,0.1,10)==0)

    def minus(self,startWert,minusAmount,anzahl):
        for i in range(0,anzahl):
            startWert-=minusAmount
        return startWert

unittest.main(argv=['-v'],verbosity=3, exit=False)
```

```
test_minus (__main__.TestCal) ... FAIL

======================================================================
FAIL: test_minus (__main__.TestCal)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/tmp/ipykernel_65319/3015338277.py", line 5, in test_minus
    self.assertTrue(self.minus(1,0.1,10)==0)
AssertionError: False is not true

----------------------------------------------------------------------
Ran 1 test in 0.002s

FAILED (failures=1)
```

Bessere Fehlermeldung mit assertEqual()

In [11]:

```python
class TestCal(unittest.TestCase):
    def test_minus(self):
        self.assertEqual(self.minus(1,0,1),1)
        self.assertEqual(self.minus(1,0.1,1),0.9)
        self.assertEqual(self.minus(1,0.1,10),0)

    def minus(self,startWert,minusAmount,anzahl):
        for i in range(0,anzahl):
            startWert-=minusAmount
        return startWert

unittest.main(argv=['-v'],verbosity=3, exit=False)
```

```
test_minus (__main__.TestCal) ... FAIL

======================================================================
FAIL: test_minus (__main__.TestCal)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/tmp/ipykernel_65319/14328734.py", line 5, in test_minus
    self.assertEqual(self.minus(1,0.1,10),0)
AssertionError: 1.3877787807814457e-16 != 0

----------------------------------------------------------------------
Ran 1 test in 0.004s

FAILED (failures=1)
```

Und nun?

In [12]:

```
class TestCal(unittest.TestCase):
    def test_minus(self):
        self.assertEqual(self.minus(1,0,1),1)
        #self.assertEqual(self.minus(1,0.1,1),0.9)
        self.assertEqual(self.minus(1,0.1,10),0)

    def minus(self,startWert,minusAmount,anzahl):
        for i in range(0,anzahl):
            startWert-=minusAmount
        return round(startWert)

unittest.main(argv=['-v'],verbosity=3, exit=False)
```

```
test_minus (__main__.TestCal) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.008s

OK
```

Aber mit allen drei Test fällt der round-Trick auf (und damit aus):

In [13]:

```
class TestCal(unittest.TestCase):
    def test_minus(self):
        self.assertEqual(self.minus(1,0,1),1)
        self.assertEqual(self.minus(1,0.1,1),0.9)
        self.assertEqual(self.minus(1,0.1,10),0)

    def minus(self,startWert,minusAmount,anzahl):
        for i in range(0,anzahl):
            startWert-=minusAmount
        return round(startWert)

unittest.main(argv=['-v'],verbosity=3, exit=False)
```

```
test_minus (__main__.TestCal) ... FAIL

======================================================================
FAIL: test_minus (__main__.TestCal)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/tmp/ipykernel_65319/3668040896.py", line 4, in test_minus
    self.assertEqual(self.minus(1,0.1,1),0.9)
AssertionError: 1 != 0.9

----------------------------------------------------------------------
Ran 1 test in 0.003s

FAILED (failures=1)
```

Was dann?

Es kommt auf die Anforderung an:

- Exakt rechnen mit decimal.Decimal
- Statt mit == zu vergleichen prüfen, ob die Abweichung “klein genug” ist

Exact rechnen (mit Decimal)

In [90]:
```
from decimal import Decimal
a1=1
a2=Decimal(1)
for i in range(0,10):
    print(a1," ",a2)
    a1=a1-0.1
    a2=a2-Decimal("0.1")
print(a1," ",a2," is a2 0 ?",a2==Decimal(0))
```

```
1   1
0.9   0.9
0.8   0.8
0.7000000000000001   0.7
0.6000000000000001   0.6
0.5000000000000001   0.5
0.4000000000000001   0.4
0.3000000000000001   0.3
0.2000000000000001   0.2
0.1000000000000001   0.1
1.3877787807814457e-16   0.0  is a2 0 ? True
```

In [20]:

```
import decimal

class TestCal(unittest.TestCase):
    def test_minus(self):
        self.assertEqual(self.minus(1,0,1),1)
        self.assertEqual(self.minus(1,0.1,1),0.9)
        self.assertEqual(self.minus(1,decimal.Decimal('0.1'),10),0)

    def minus(self,startWert,minusAmount,anzahl):
        for i in range(0,anzahl):
            startWert-=minusAmount
        return startWert

unittest.main(argv=['-v'],verbosity=3, exit=False)
```

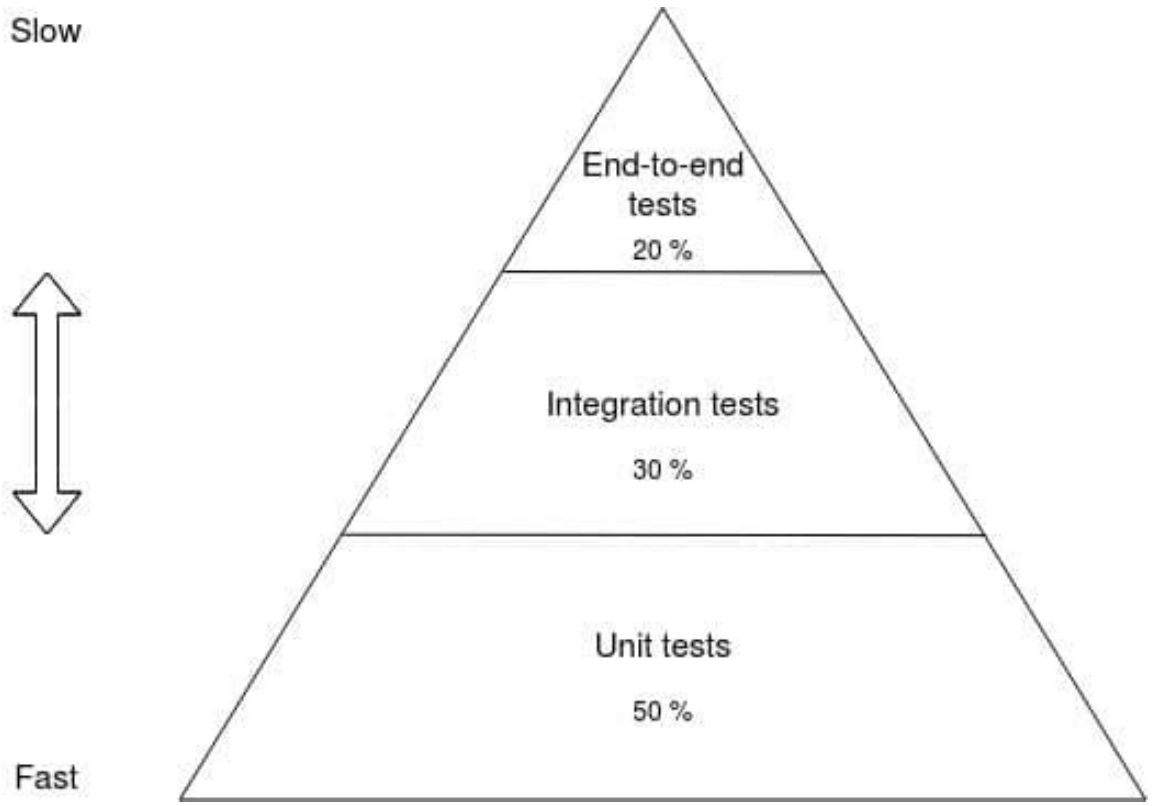
```
test_minus (__main__.TestCal) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
```

Eine Klasse verwenden: Instanz erstellen und Methoden aufrufen

Testing: test driven development



From https://testdriven.io/blog/modern-tdd/

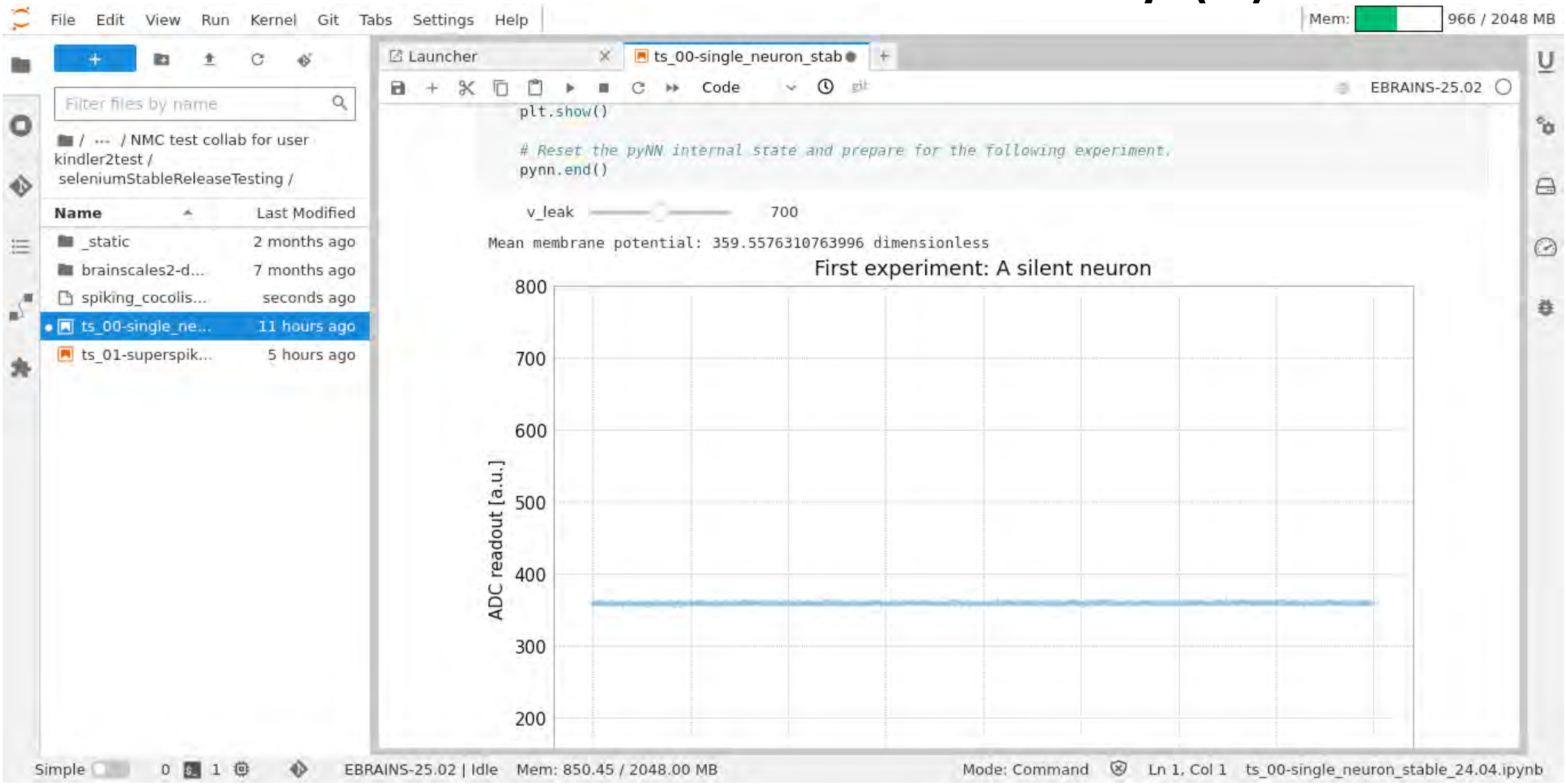
Testing für Software im team

- Lokal entwickeln (mit git für die Verfolgung der Änderungen)
- Änderungen in einem changeset zur Verfügung stellen
- Automatischer Test der Software mit allen Test-Suites
- Manueller code review on top
- Deployment

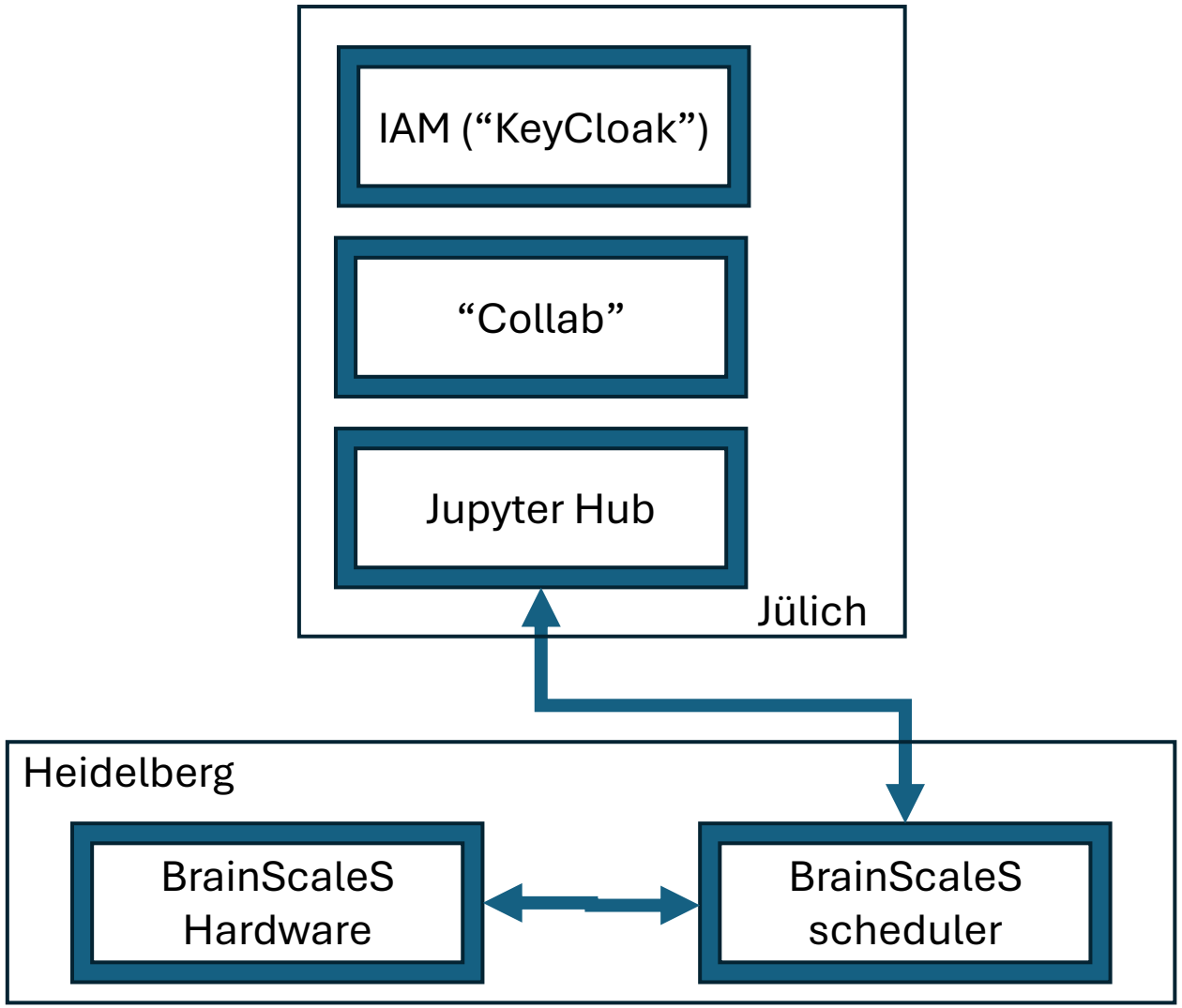
Testing: end-to-end testing of a web service

- Nötig für Services, die Forschungsgruppen im Web für die Forschugnsgemeinschaft anbieten um sicher zu stellen, daß Dinge “weiterhin / immer noch funktionieren”
- Beispiel: Neuromorphic Computing via EBRAINS

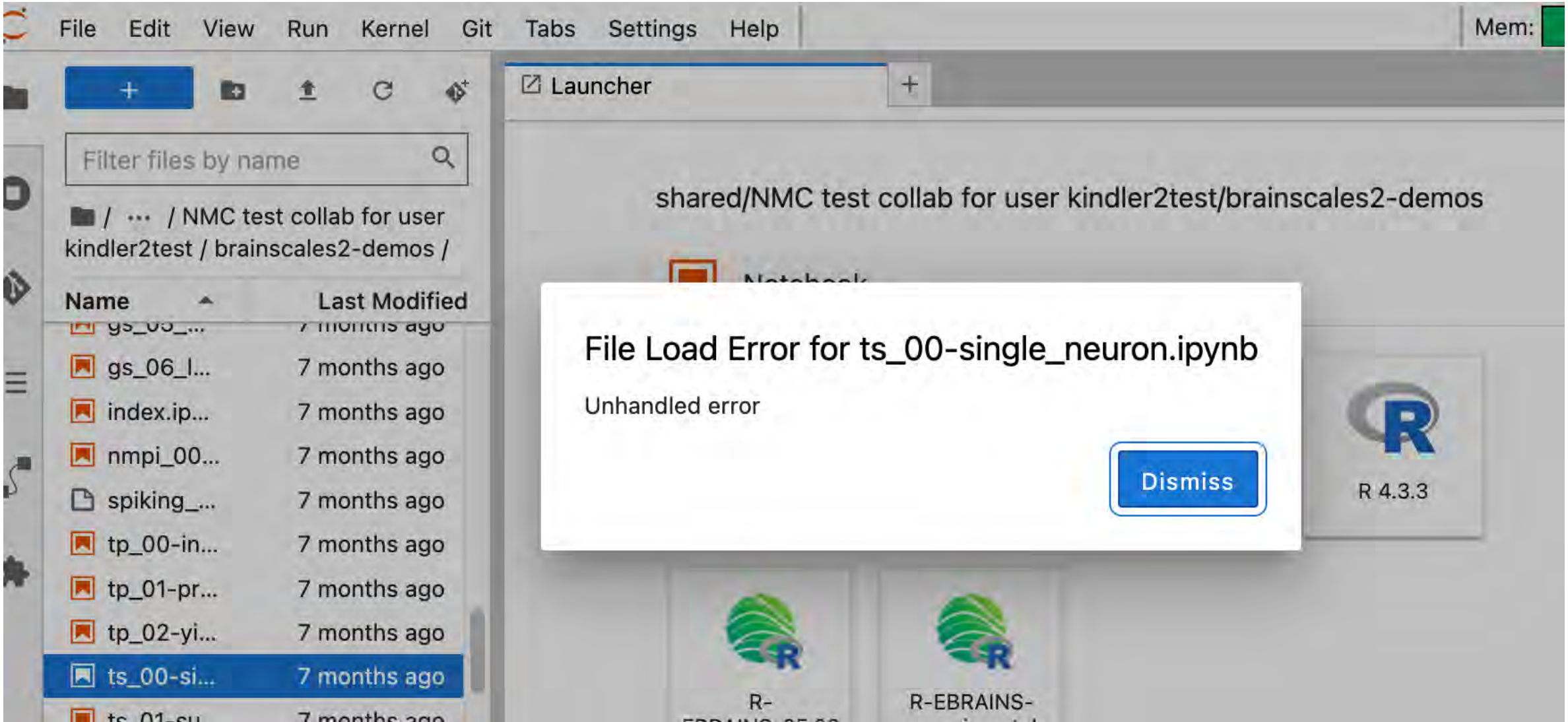
Just execute a notebook cell = easy (?)



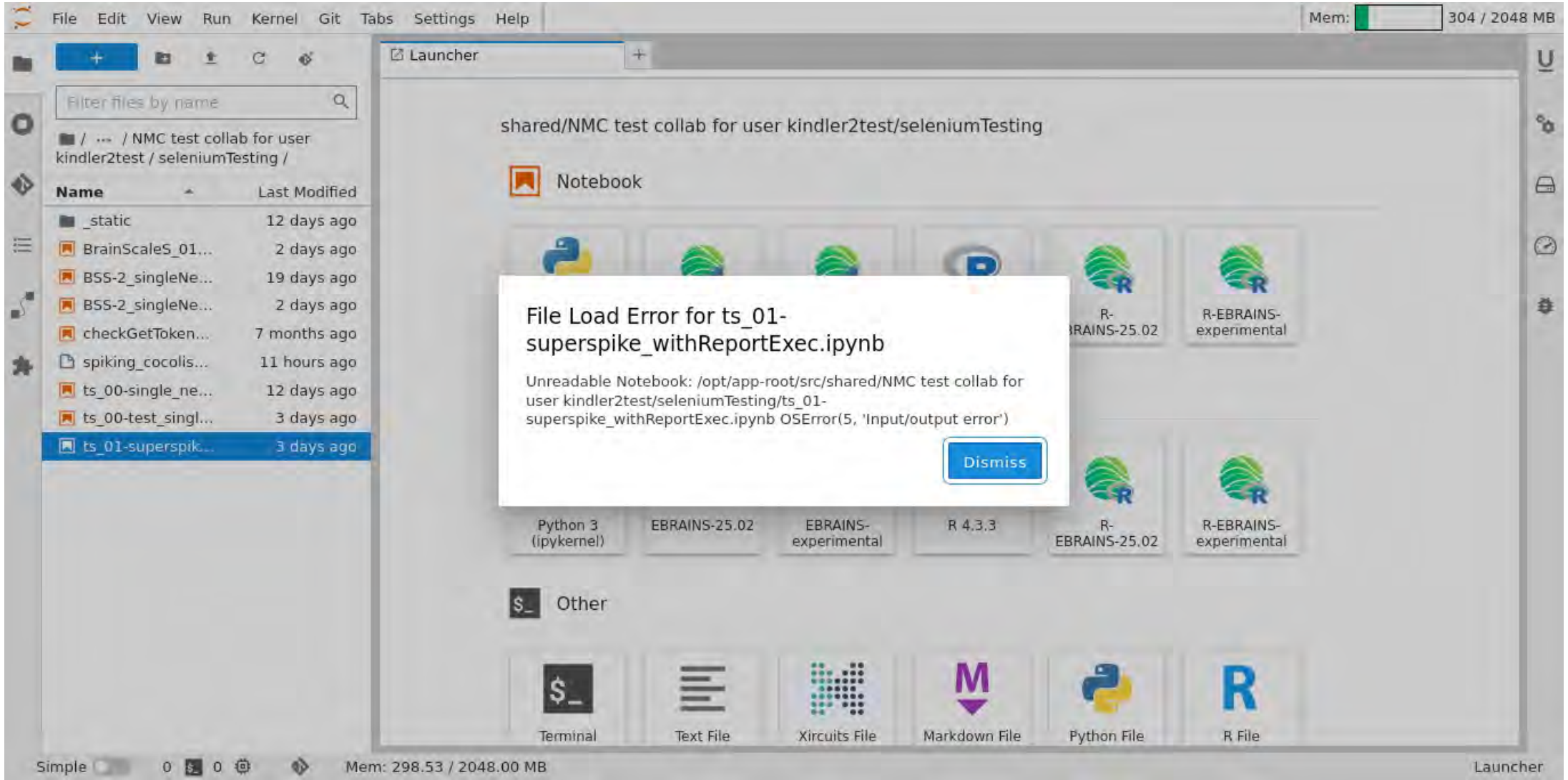
“What could possibly go wrong”



“What could possibly go wrong”



“What could possibly go wrong”



“What could possibly go wrong”

“What could possibly go wrong”

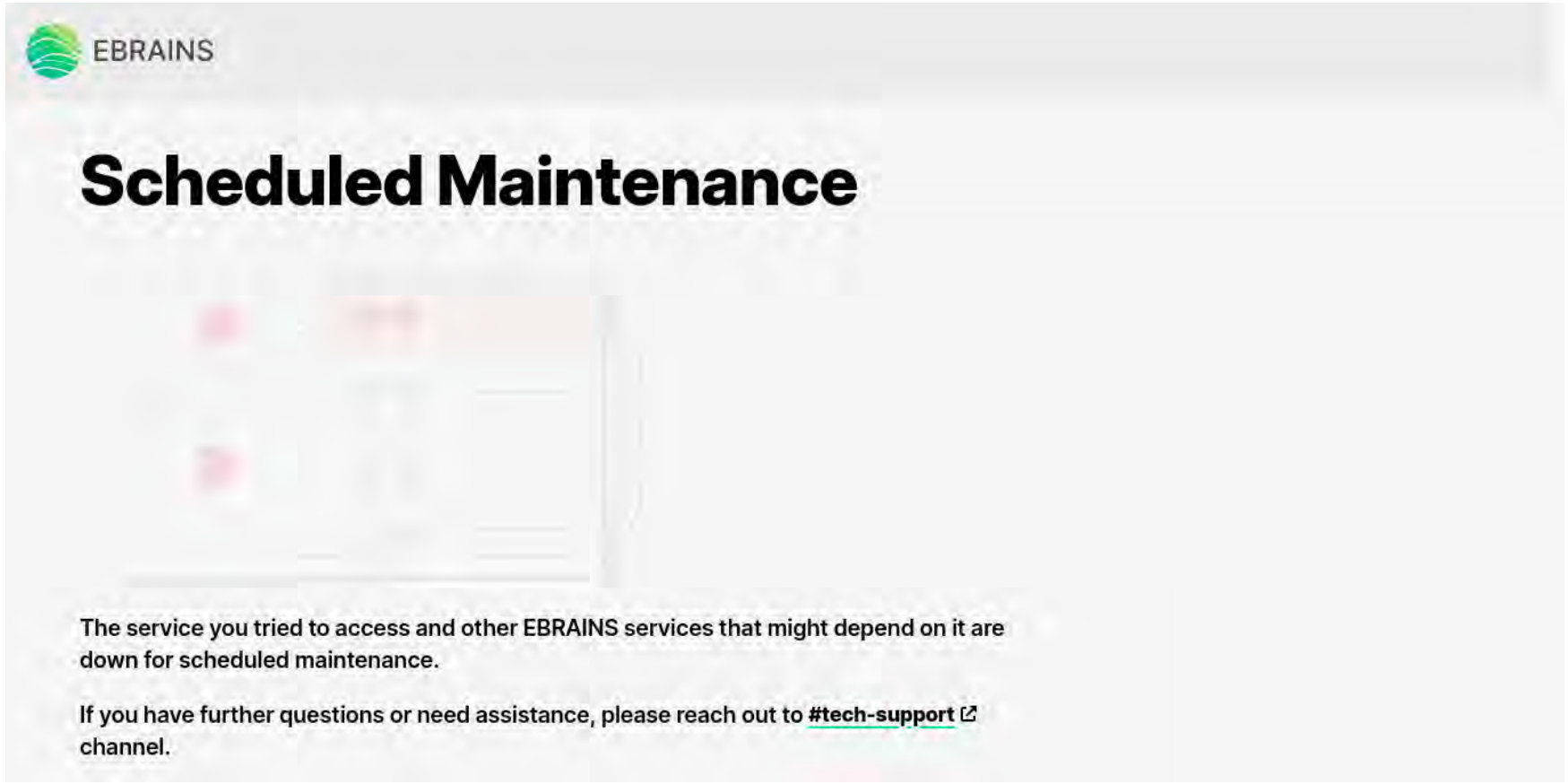
Service Unavailable

The server is temporarily unable to service your request due to maintenance downtime or capacity problems. Please try again later.

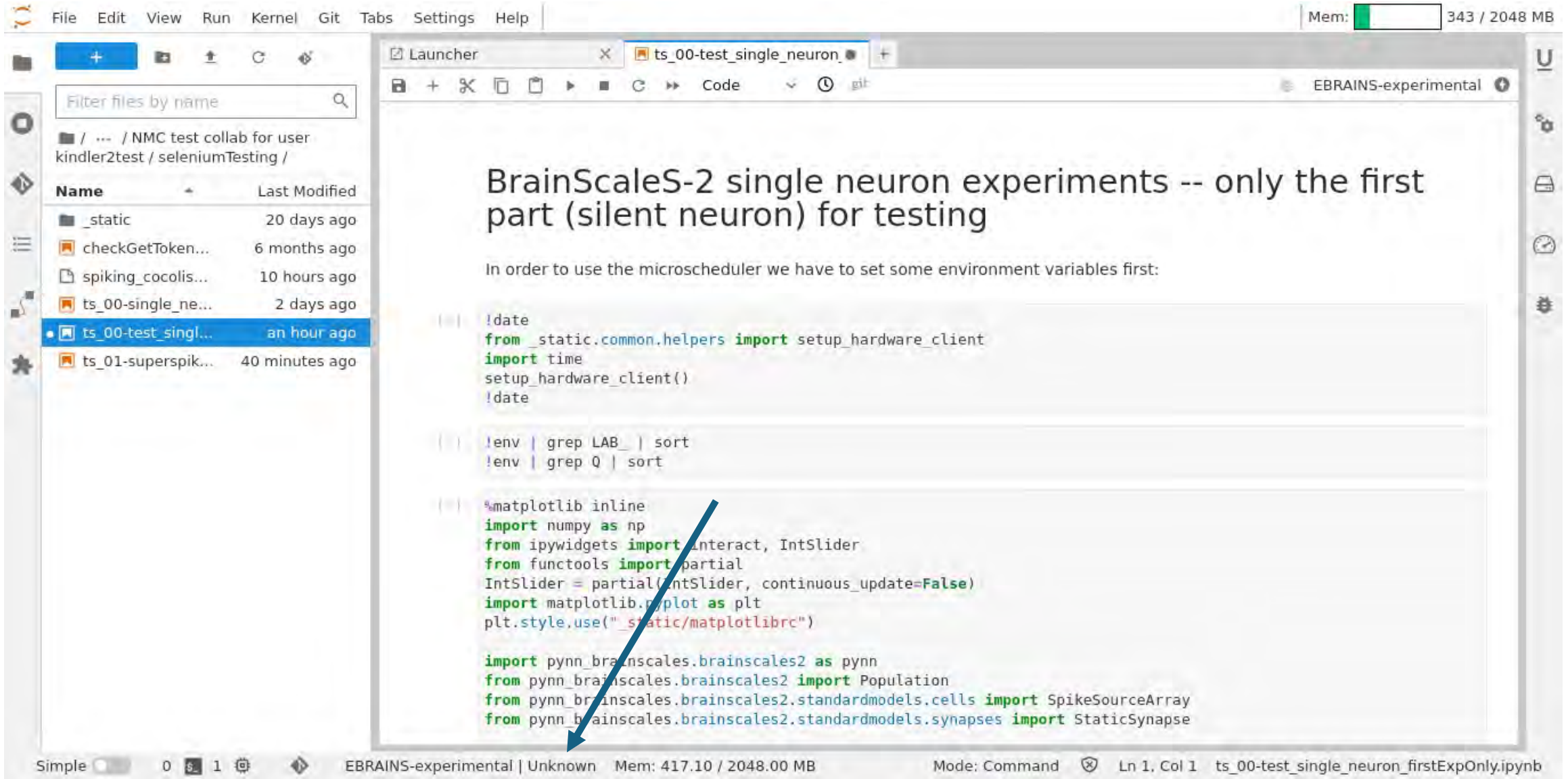
Additionally, a 503 Service Unavailable error was encountered while trying to use an ErrorDocument to handle the request.

Apache/2.4.52 (Ubuntu) Server at iam.ebrains.eu Port 443

“What could possibly go wrong”

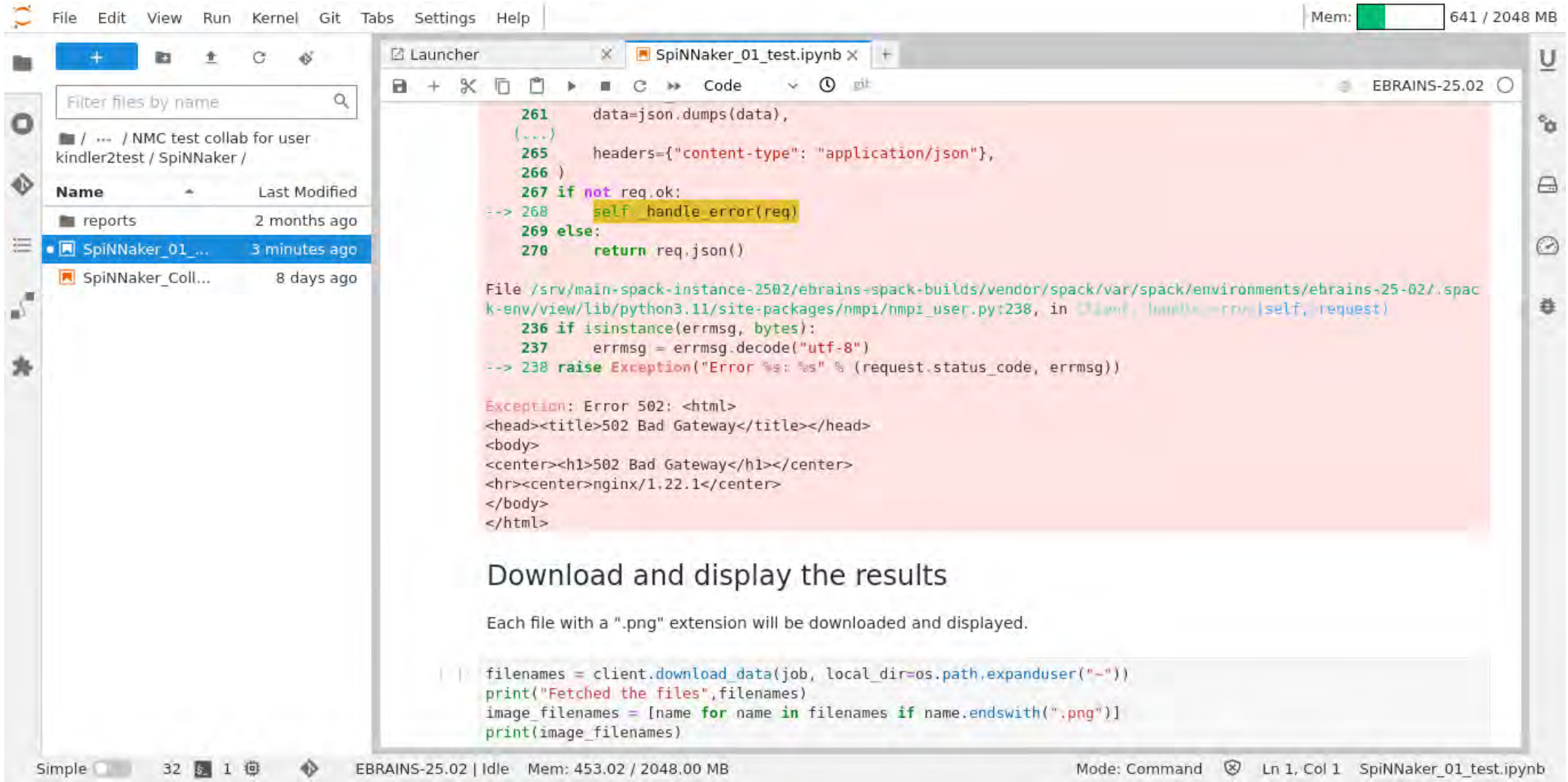


“What could possibly go wrong”



“What could possibly go wrong”

Sometimes it is also our system in Heidelberg….



EBRAINS RI als Beispiel (NMC BrainScaleS)

- Account unter https://ebrains.eu/register erstellen (mit @uni-heidelberg.de Adresse, z.B. der URZID@uni-heidelberg.de)

- Mit dem neuen EBRAINS account https://einc.eu/bss >> “Getting Access” >> “access this URL” aus Punkt 3 wählen

  (die URL muß man 2x aufrufen: Beim ersten Zugriff wird die “Lab session” gestartet, beim zweiten Zugriff wird ein git clone ausgeführt und ein Notebook geöffnet)

  (Vorsicht: diese Stelle ist non-persistent = alle Änderungen gehen beim Ausloggen verloren! Alternative: Collab erstellen und das “Drive” nutzen)

Selenium

Ist eine Libray, mit der man einen browser (Chrome oder Firefox) programmatisch steuern kann.

Damit ist es möglich, einen Web-Service “end-to-end” zu testen, also so, wie ein Nutzer den Service erlebt.

(Ein end-to-end Test sollte natürlich nicht der einzige Test sein. Direkte Service Tests erlauben normalerweise ein einfacheres Debugging, können den end-to-end Test aber nicht ersetzen)

Selenium setup

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
import time
from selenium.webdriver.firefox.service import Service as geckoService
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver import Keys


gecko_service=geckoService("/gecko/geckodriver")
browser_options = webdriver.FirefoxOptions()
if False:
    browser_options.add_argument("-headless")
driver = webdriver.Firefox(service=gecko_service,options=browser_options)
```

Open page

- driver.get(”URL_to_open")

The URL for the example:

https://lab.jsc.ebrains.eu/hub/user-redirect/git-pull?repo=https%3A%2F%2Fgithub.com%2Felectronicvisions%2Fbrainscales2-demos.git&urlpath=lab%2Ftree%2Fbrainscales2-demos.git%2Fts_00-single_neuron.ipynb&branch=jupyter-notebooks-experimental

A small example

In [87]:
```
driver.get("https://lab.jsc.ebrains.eu/hub/user-redirect/git-pull"\
           +"?repo=https%3A%2F%2Fgithub.com%2Felectronicvisions%2Fbrainscales2-demos.gi
           +"&urlpath=lab%2Ftree%2Fbrainscales2-demos.git%2Fts_00-single_neuron.ipynb"\
           +"&branch=jupyter-notebooks-experimental")
```

In [88]:
```
for e in driver.find_elements(By.CLASS_NAME, "p-MenuBar-itemLabel"):
    print(e.text)
```

```
File
Edit
View
Run
Kernel
Git
Tabs
Settings
Help
```

Elemente auf der Seite finden (Beispiele)

- elems=driver.find_elements(By.TAG_NAME,"h1")
- elems=driver.find_elements(By.CLASS_NAME,"control")
- elems = driver.find_elements(By.ID,"username")

- Oder Elemente unterhalb eines Elementes:

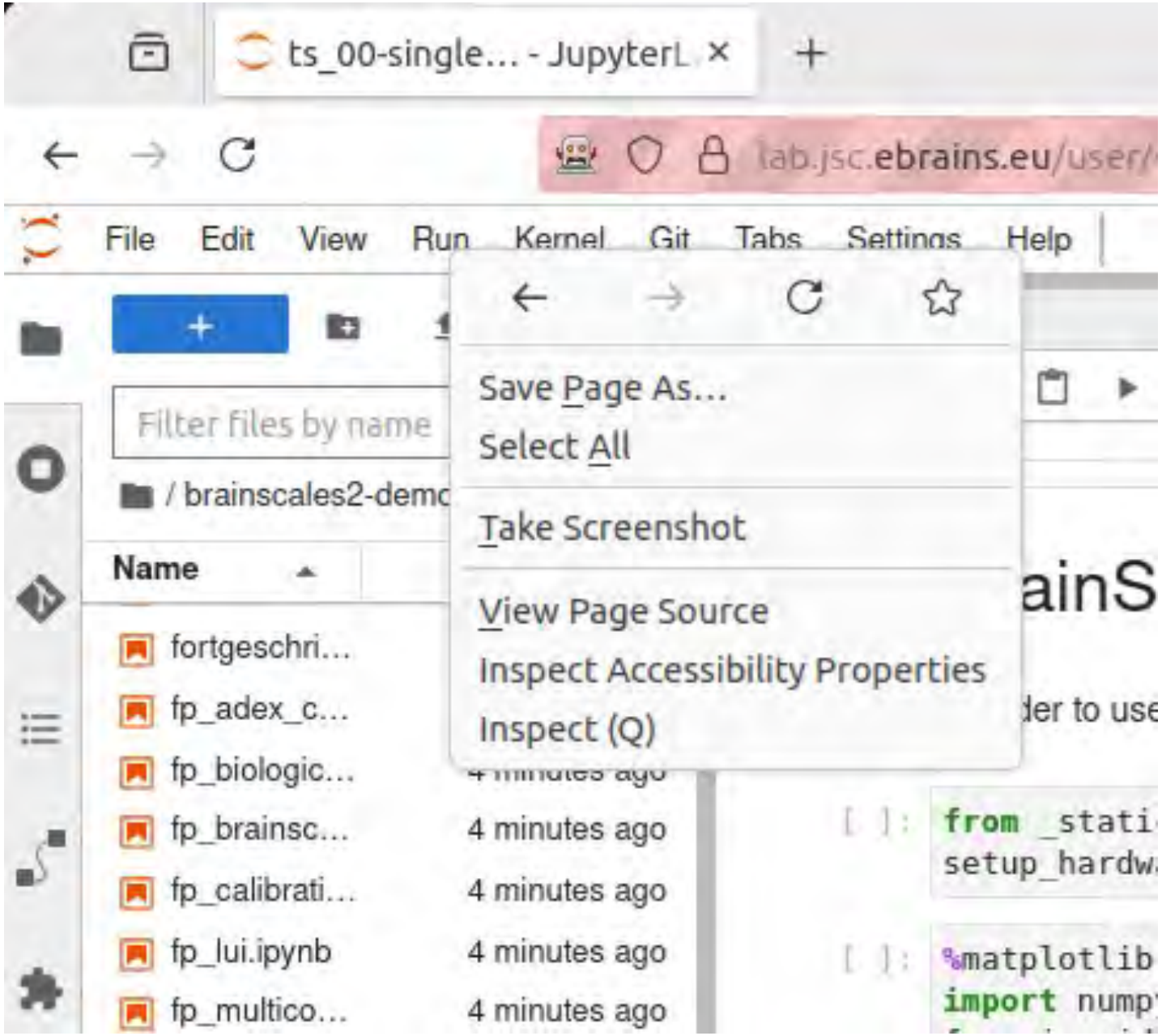
```
elems=driver.find_elements(By.CLASS_NAME,"jp-FileBrowser-crumbs")
if len(elems) > 0:
    el=elems[0].find_elements(By.CSS_SELECTOR,"*")
    for e in el:
        (do something with them)
```

A small example

In [87]:
```
driver.get("https://lab.jsc.ebrains.eu/hub/user-redirect/git-pull"\
           +"?repo=https%3A%2F%2Fgithub.com%2Felectronicvisions%2Fbrainscales2-demos.gi
           +"&urlpath=lab%2Ftree%2Fbrainscales2-demos.git%2Fts_00-single_neuron.ipynb"\
           +"&branch=jupyter-notebooks-experimental")
```

In [88]:
```
for e in driver.find_elements(By.CLASS_NAME, "p-MenuBar-itemLabel"):
    print(e.text)
```

```
File
Edit
View
Run
Kernel
Git
Tabs
Settings
Help
```



Mit “rechter Maustaste” und SHIFT auf ein Element klicken und “Inspect (Q)” auswählen

Dies zeigt im “Inspector” in Firefox das Element im html an - und zum Glück verwendet Jupyter an ganz vielen Stellen class Namen:

A small example

In [87]:
```
driver.get("https://lab.jsc.ebrains.eu/hub/user-redirect/git-pull"\
           +"?repo=https%3A%2F%2Fgithub.com%2Felectronicvisions%2Fbrainscales2-demos.gi
           +"&urlpath=lab%2Ftree%2Fbrainscales2-demos.git%2Fts_00-single_neuron.ipynb"\
           +"&branch=jupyter-notebooks-experimental")
```

In [88]:
```
for e in driver.find_elements(By.CLASS_NAME, "p-MenuBar-itemLabel"):
    print(e.text)
```

```
File
Edit
View
Run
Kernel
Git
Tabs
Settings
Help
```

In [97]:

```
for e in driver.find_elements(By.CLASS_NAME, "p-MenuBar-itemLabel"):
    print(e.text)
    if "Run"==e.text:
        e.click()
        for f in driver.find_elements(By.CLASS_NAME,"p-Menu-itemLabel"):
            print(f.text)
            if f.text=="Run All Cells":
                f.click()
                break
        break
```

Screenshots

- Einen screenshot der aktuellen Seite im Browser machen:
  driver.get_screenshot_as_file(‘/PFAD/test.png’)

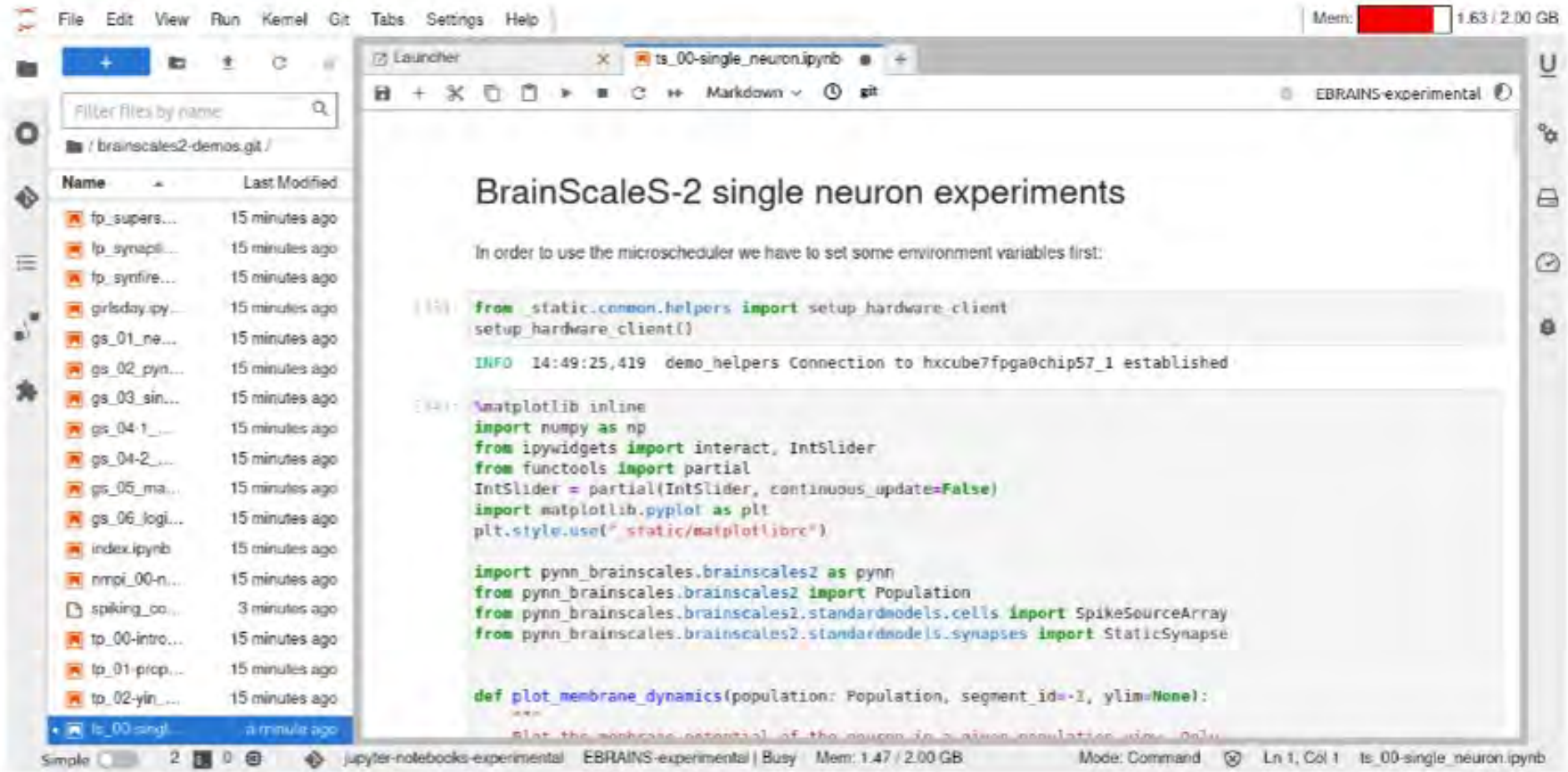
- Screenshots dann im notebook anzeigen:

```
from IPython.core.display import display, HTML
import os
a=os.listdir("/PFAD")
s=""”
for x in a:
    l="screenshots/"+x
    s+='<a href="'+l+'" target=_blank><img src="'+l+'" width=400px></a><br>\n’
    print(x)
print(s)
display(HTML(s))
```

In [105]:
```
1 driver.get_screenshot_as_file("test.png")
```

Out[105]: True

In [111]:
```
1 from IPython.core.display import display, HTML
2 s="<img src=test.png>"
3 display(HTML(s))
```



Übungsaufgabe: log-in hinzu fügen

In [112]:

```
1 import getpass
2 username="guestbktest2july2"
3 password = getpass.getpass()
```

....