

2. 関数の実行時間を 計ってみよう

JupyterLabの起動方法 (Windows)

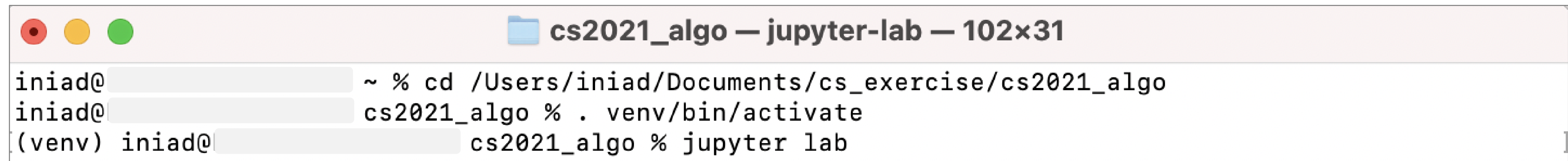
1. PowerShellを起動し、仮想環境を構築した作業フォルダ「cs2024_algo」に移動する
 - `cd C:\Users\iniad\Documents\cs_exercise\cs2024_algo`
※パスの指定方法は、個人の環境に合わせて変えてください。前の手順でもやったように、相対パスでの指定でもかまいません
2. 仮想環境を有効化する
 - `.\venv\Scripts\activate`
3. JupyterLabを起動する
 - `jupyter lab`

```
PS C:\Users\iniad> cd C:\Users\iniad\Documents\cs_exercise\cs2024_algo
PS C:\Users\iniad\Documents\cs_exercise\cs2024_algo> .\venv\Scripts\activate
(venv) PS C:\Users\iniad\Documents\cs_exercise\cs2024_algo> jupyter lab
```

※ PowerShellは、ノートブックを終了するまで閉じないこと！

JupyterLabの起動方法(macOS)

1. Terminalを起動し、仮想環境を構築した作業フォルダ「cs2024_algo」に移動する
 - `cd /Users/iniad/Documents/cs_exercise/cs2024_algo`
※パスの指定方法は、個人の環境に合わせて変えてください。前の手順でもやったように、相対パスでの指定でもかまいません
2. 仮想環境を有効化する
 - `. venv/bin/activate`
3. JupyterLabを起動する **ドットのと後に半角スペースを入れて `venv/bin/activate`**
 - `jupyter lab`



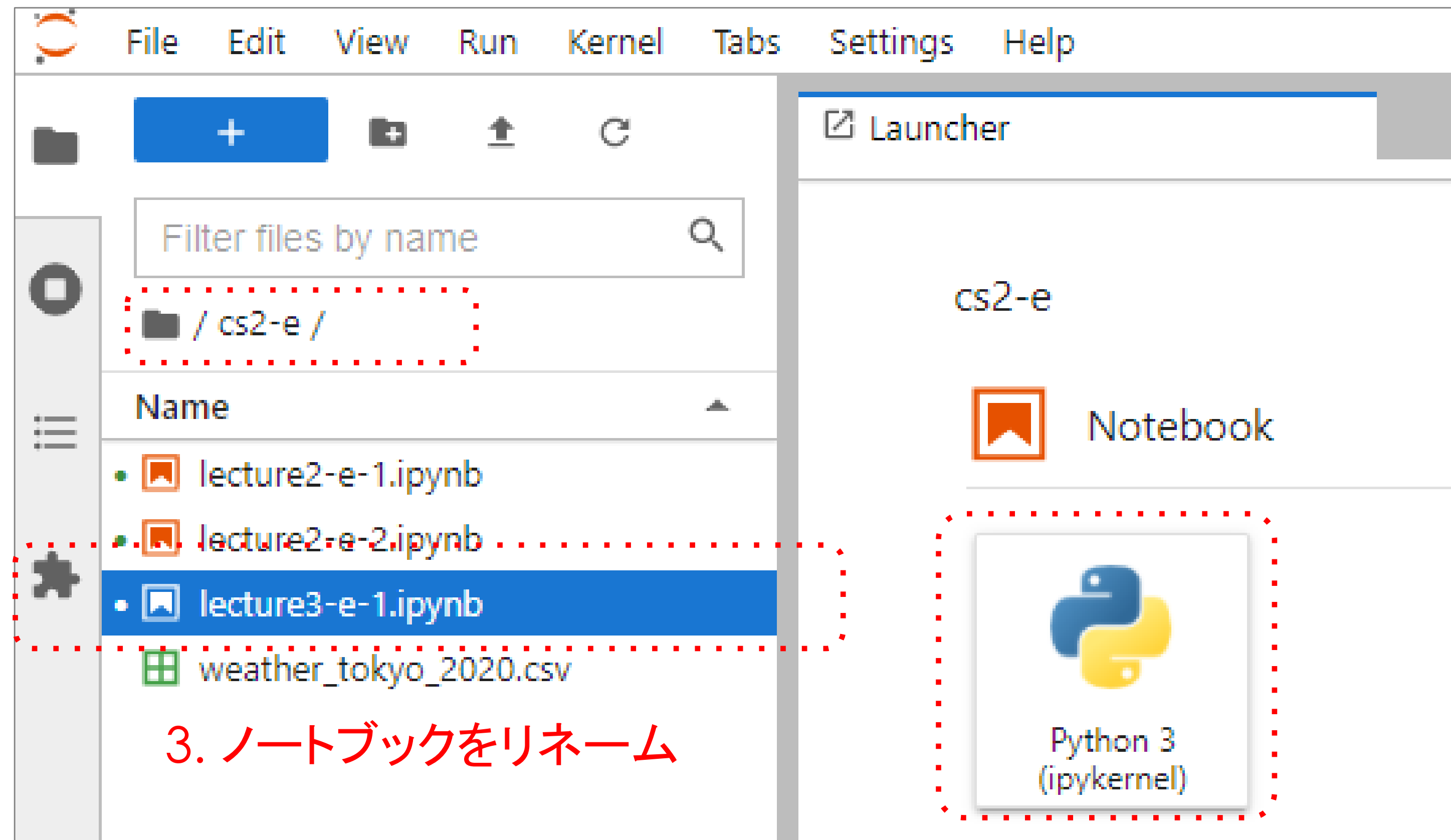
```
cs2021_algo — jupyter-lab — 102x31
iniad@ ~ % cd /Users/iniad/Documents/cs_exercise/cs2021_algo
iniad@ cs2021_algo % . venv/bin/activate
(venv) iniad@ cs2021_algo % jupyter lab
```

※ Terminalは、ノートブックを終了するまで閉じないこと！

演習用のノートブックの作成

- 「cs2-e」の下に、lecture3-e-1.ipynb を作成しましょう

1. cs2-e
フォルダに移動



3. ノートブックをリネーム

2. ノートブックを作成

Matplotlib と timeit モジュールをインポートする

- lecture3-e-1.ipynbに以下のセルを作成します
 - Markdownセル
 - Codeセルを追加し、以下を行います
 - matplotlib.pyplot を plt という別名でインポート
 - 関数の実行時間を計測するため、timeitモジュールをインポート

Markdownセル →

```
## 3-e-1 関数の実行時間を計ってみよう
```

Codeセル →




```
import matplotlib.pyplot as plt  
import timeit
```


timeitモジュールとは

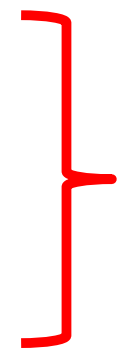
- 関数の実行時間を計測するモジュールです
- timeitモジュールの使い方
 - `timeit.timeit(CODE, globals = globals(), number=NUMBER_OF_ITERATION)`
 - 第一引数 CODE: 実行したいコードを指定する
 - 今回の演習では、引数 globals には `globals()` を指定する
 - 引数 number には実行回数を指定する
 - 戻り値は、CODEをNUMBER_OF_ITERATION回実行した合計の実行時間[s]

timeitモジュールを試してみよう

- Codeセルを作成し、以下のプログラムを書いて実行してみましょう

```
def test():  実行時間を計測したい関数を定義  
    print("Hello World")  
  
time = timeit.timeit('test()', globals = globals(), number=3)  
print(time/3)  計測したい関数を指定  計測回数
```

```
Hello World  
Hello World  
Hello World  
0.00037349999999491956  3回の計測の平均時間を出力
```

 test()関数が3回実行される

二つの関数を計測してグラフ化する手順

1. 関数 f1 について以下を行う

- ループ回数(引数n)を50から450まで50ずつ変化させて計測する
 - 各ループ回数ごとに1000回計測を行って平均実行時間を求める
(n=50で1000回、n=100で1000回.....)
- ループ回数(x軸)と平均実行時間(y軸)をグラフに描画する

```
def f1(n):  
    s = 0  
    for i in range(n):  
        for j in range(n):  
            s += 1  
    return s
```

2. 関数 f2 について以下を行う

- ループ回数(引数n)を1から10まで1ずつ変化させて計測する
 - 各ループ回数ごとに1000回計測を行って平均実行時間を求める
(n=1で1000回、n=2で1000回.....)
- ループ回数(x軸)と平均実行時間(y軸)をグラフに描画する

```
def f2(n):  
    s = 0  
    for i in range(2**n):  
        s += 1  
    return s
```

3. 描画したグラフの形状を、今まで習った計算量のグラフと比較する

1. 関数 f1 を定義する

- Codeセルを作成し、以下のプログラムを書いて実行してみましょう
 - for文の実行回数はどうなりますか？

```
def f1(n):  
    s = 0  
    for i in range(n):  
        for j in range(n):  
            s += 1  
    return s
```

} n^2 回繰り返されます

1. 関数 f1 の実行時間を計測し、グラフに描画する

- Codeセルを作成し、以下のプログラムを書いて実行してみましょう

```
num_iter = 1000 → 実行回数

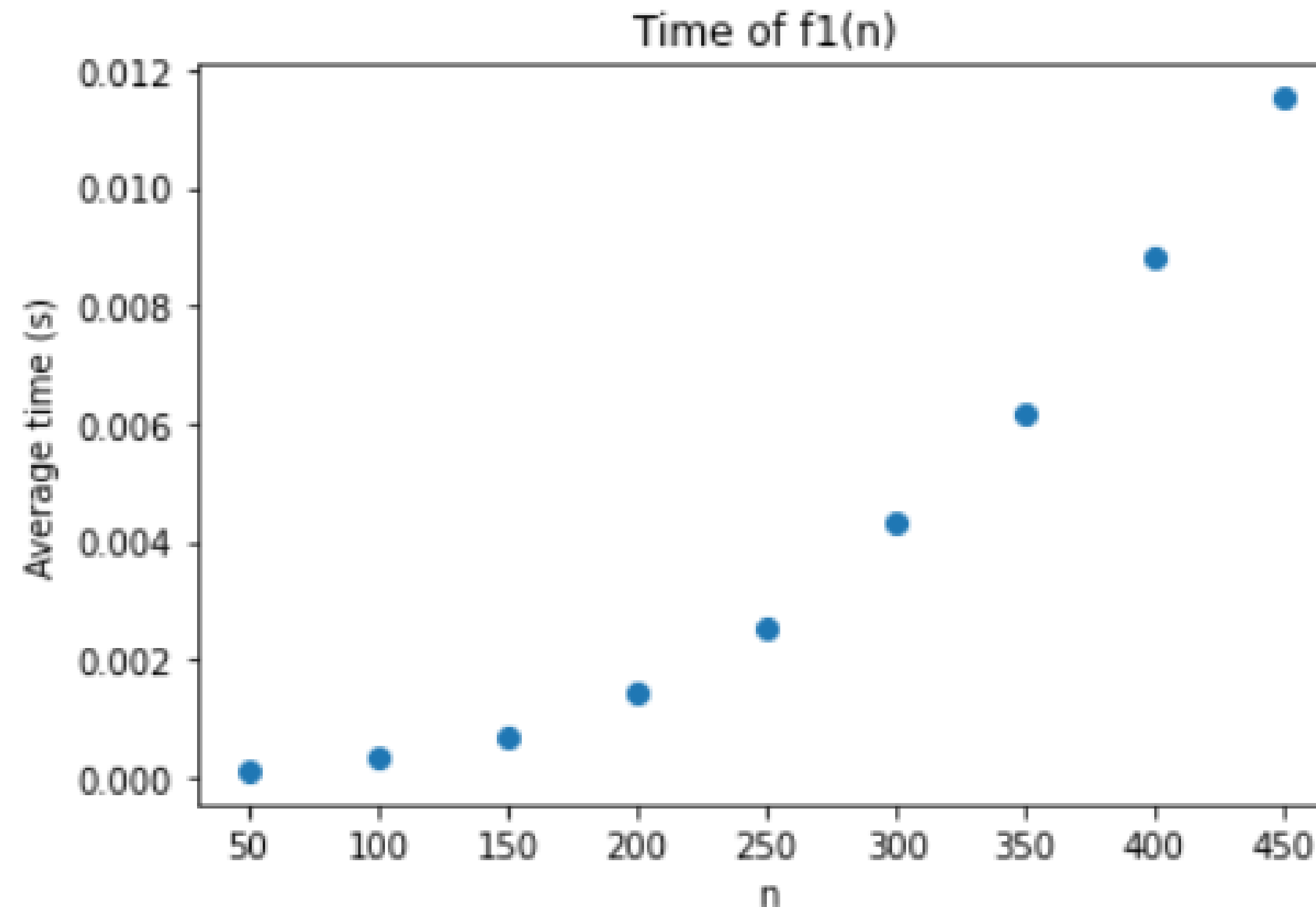
x = range(50, 500, 50) → ループ回数(x軸) [50, 100, 150, ..., 450]
y = [] → 平均時間を格納するためのリスト(y軸)を用意
for n in x:
    time = timeit.timeit('f1(n)', globals = globals(), number=num_iter)
    y.append(time / num_iter)

plt.plot(x, y, 'o')
plt.title('Time of f1(n)')
plt.xlabel('n')
plt.ylabel('Average time (s)')
plt.show()
```

1. グラフの確認

● $O(n^2)$ のグラフが表示されましたか？

■ 見た目が大きく異なる場合は、何回か実行しなおしてみてください



2. 関数 f2 を定義する

- Codeセルを作成し、以下のプログラムを書いて実行してみましょう
 - for文の実行回数はどうなりますか？

```
def f2(n):  
    s = 0  
    for i in range(2**n):  
        s += 1  
    return s
```

} 2^n 回繰り返されます

2. 関数 f2 の実行時間を計測し、グラフに描画する

- Codeセルを作成し、以下のプログラムを書いて実行してみましょう

```
num_iter = 1000

x = range(1, 11, 1)
y = []
for n in x:
    time = timeit.timeit('f2(n)', globals=globals(), number=num_iter)
    y.append(time / num_iter)

plt.plot(x, y, 'o')
plt.title('Time of f2(n)')
plt.xlabel('n', fontsize=9)
plt.ylabel('Average time (s)')
plt.show()
```

2. グラフの確認

● $O(2^n)$ のグラフが表示されましたか？

■ 見た目が大きく異なる場合は、何回か実行しなおしてみてください

