Without vectorization

```
for j in range(0,16):
    f = f + w[j] * x[j]
```

t_0 `f + w[0] * x[0]`

t_1 `f + w[1] * x[1]`

...

t_{15} `f + w[15] * x[15]`

Vectorization

```
np.dot(w,x)
```

t_0

w[0]	w[1]	...	w[15]

in parallel * * ... *

x[0]	x[1]	...	x[15]

t_1

w[0]*x[0] + w[1]*x[1] +...+ w[15]*x[15]

efficient → scale to large datasets

→ It is the difference between vector. vs without vectorization computation.

→ Vectorization works in parallel so, it takes less time to compute.

Gradient descent

$\vec{w} = (w_1 \quad w_2 \quad \cdots \quad w_{16})$ ~~b~~ parameters

derivatives $\vec{d} = (d_1 \quad d_2 \quad \cdots \quad d_{16})$

```
w = np.array([0.5, 1.3, ... 3.4])
d = np.array([0.3, 0.2, ... 0.4])
```

learning rate α

compute $w_j = w_j - 0.1 d_j$ for $j = 1 \ldots 16$

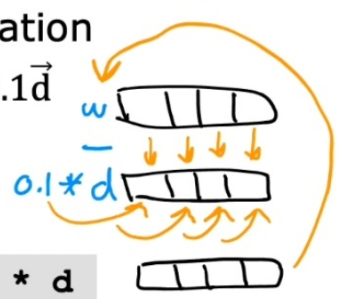
Without vectorization

$$w_1 = w_1 - 0.1 d_1$$
$$w_2 = w_2 - 0.1 d_2$$
$$\vdots$$
$$w_{16} = w_{16} - 0.1 d_{16}$$

```
for j in range(0,16):
    w[j] = w[j] - 0.1 * d[j]
```

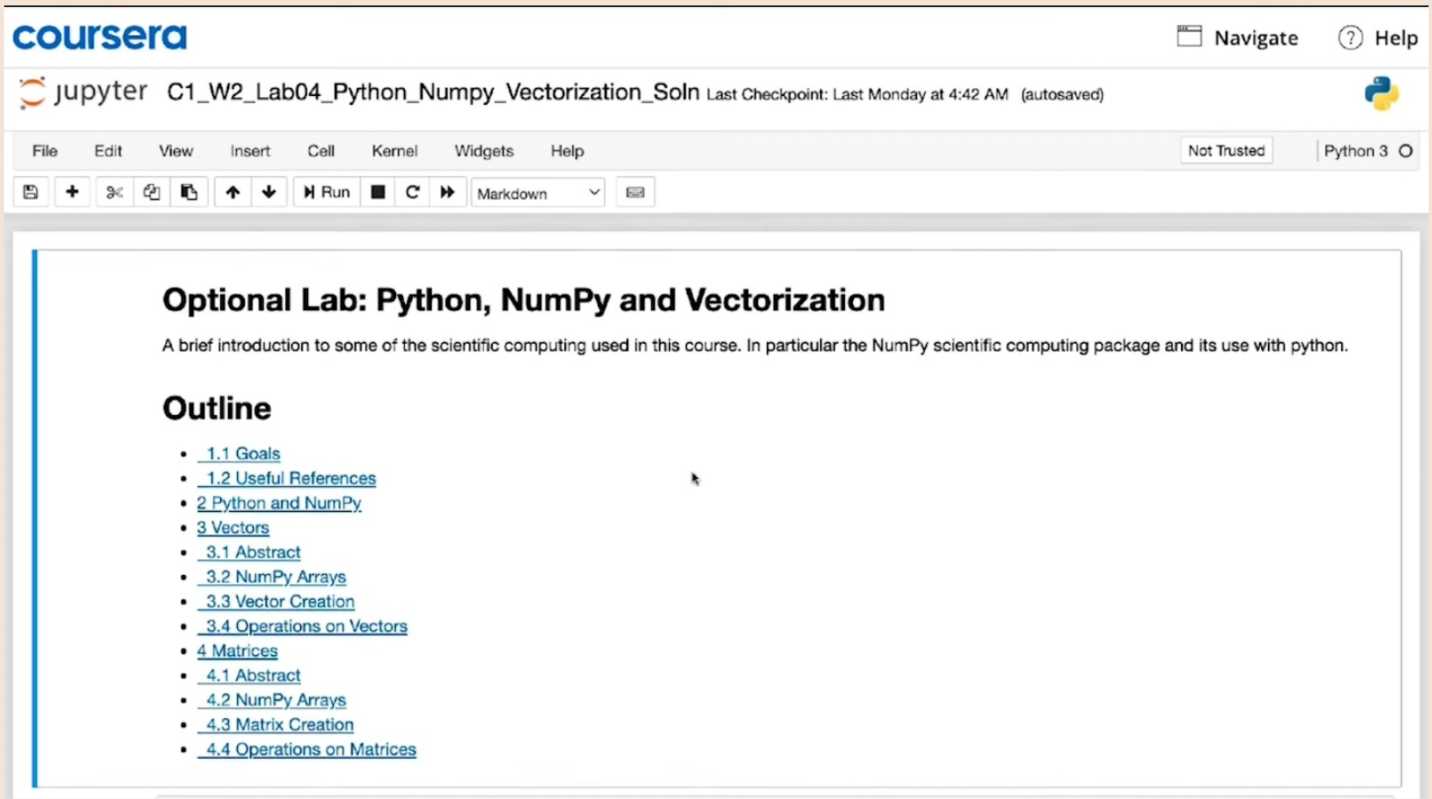
With vectorization

$$\vec{w} = \vec{w} - 0.1\vec{d}$$




```
w = w - 0.1 * d
```

→ Here how vectorization takes less time working in parallel in the gradient descent algorithm.



→ optional lab numpy vectorization practice.