Colab: https://colab.research.google.com/drive/1oc0vgSolFOa2o9igJCa8Jr0-ntj48IRA?usp=sharing

Update from the assessments team on a question in OOPs

In the question Spanoftime class, the return statement's later part including ("The total minutes in time t1 and t2 are: " + totalmin) was earlier expecting two spaces inbetween the string and totalmin. It's fixed now there should only be one space at the end of string i.e. ("The total minutes in time t1 and t2 are: " + totalmin). If you still get the space error reset the code template and complete the two functions it will work fine.

```
!pip install numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
```

Imagine that you are a Data Scientist at Fitbit

Link: https://drive.google.com/file/d/1kXqcJo4YzmwF1G2BPoA17CI49TZVHANF/view?usp=sharing

```
#date   step_count      mood    calories_burned hours_of_sleep  bool_of_active  weight_kg
06-10-2017      5464    200     181     5       0       66
07-10-2017      6041    100     197     8       0       66
08-10-2017      25      100     0       5       0       66
09-10-2017      5461    100     174     4       0       66
10-10-2017      6915    200     223     5       500     66
11-10-2017      4545    100     149     6       0       66
12-10-2017      4340    100     140     6       0       66
13-10-2017      1230    100     38      7       0       66
14-10-2017      61      100     1       5       0       66
15-10-2017      1258    100     40      6       0       65
16-10-2017      3148    100     101     8       0       65
                                        5       0       65
                                        6       500     65
                                        7       0       65
20-10-2017      1580    100     49      5       0       65
21-10-2017      2822    100     86      6       0       65
22-10-2017      181     100     6       8       0       65
23-10-2017      3158    200     99      5       0       65
```

Saved successfully!

```
import numpy as np

# Motivation - Generic

a = [1, 2, 3, 4, 5]

a_sq = [element**2 for element in a] # element-wise operation
```

```
a_np = np.array(a)
```

```
a_np
```

```
array([1, 2, 3, 4, 5])
```

```
a_np ** 2 # only benefit?
```

```
array([ 1,  4,  9, 16, 25])
```

```
l = range(1000000)
```

```
len(l)
```

```
1000000
```

```
%timeit [i**2 for i in l]
```

```
1 loop, best of 5: 271 ms per loop
```

```
l_np = np.arange(1000000)
```

```
len(l_np)
```

```
1000000
```

```
%timeit l_np ** 2
```

```
The slowest run took 6.95 times longer than the fastest. This could mean that
1000 loops, best of 5: 890 µs per loop
```

If you have to perform element-wise operation, Numpy provides

2. Computation Benefit

```
a = np.array([1, 2, 3])
```

```
a*2
```

```
array([2, 4, 6])
```

```
a.ndim
```

```
1
```

```
a.shape
```

```
(3,)
```

```
a = np.array([1,2,3,4,5,6,7,8])
print(a.ndim, a.shape)
```

```
1 (8,)
```

```
a = np.array([1,2,3,4,5,6])
b = len(a)
print(b)
```

```
6
```

```
np.arange(1, 5)
```

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

```
np.arange(1, 5, 2)
```

```
array([1, 3])
```

```
np.arange(1, 5, 0.5) # valid for arnage, not range
```

```
array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

```
# start, end, count
np.linspace(0, 100, 12) # end point is inclusive
```

```
array([  0.        ,   9.09090909,  18.18181818,  27.27272727,
        36.36363636,  45.45454545,  54.54545455,  63.63636364,
        72.72727273,  81.81818182,  90.90909091, 100.        ])
```

Saved successfully!

```
type(a_np)
```

```
numpy.ndarray
```

```
arr = np.array([1, 2, 3, 4])
print(arr)
```

```
[1 2 3 4]
```

```
arr2 = np.array([1, 2, 3, 4.0])
print(arr2)
```

```
[1. 2. 3. 4.]
```

```
np.array([1, 2, 3, 4], dtype="float")
```

```
array([1., 2., 3., 4.])
```

```
100**10
```

```
10000000000000000000
```

```
arr6 = np.array([0, 10, 100])
```

```
arr6**10
```

```
array([                   0,          10000000000, 7766279631452241920])
```

```
np.array([0, 10, 100]).dtype
```

```
dtype('int64')
```

```
type(arr)
```

```
numpy.ndarray
```

```
np.array([1, 2.3, "Anant"])
```

```
array(['1', '2.3', 'Anant'], dtype='<U32')
```

```
# Working with 2D arrays
```

```
m1 = np.array([[1, 2, 3], [4, 5, 6]])
```

```
m1
```

Saved successfully!

```
m1.ndim
```

```
2
```

```
m1.shape
```

```
(2, 3)
```

```
len(np.array([[1, 2, 3], [4, 5, 6]]))
```

```
2
```

```
a = np.array([1, 2, 3,4,5], ndmin = 2)
print(a) # row vector
```

```
[[1 2 3 4 5]]
```

```
a.ndim
```

```
2
```

```
a.shape
```

```
(1, 5)
```

```
m2 = np.arange(1, 13)
```

```
m2.ndim
```

```
1
```

```
m2.shape
```

```
(12,)
```

```
m2.reshape(3, 4)
```

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

```
m2.reshape(4, 3)
```

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

Saved successfully!

```
m2.reshape(12, 1)
```

```
array([[ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11],
       [12]])
```

```
# resize
```

```
a = np.arange(4)
a
```

```
array([0, 1, 2, 3])
```

```
a.reshape(2, 4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-66-842a4f4ea23d> in <module>()
----> 1 a.reshape(2, 4)

ValueError: cannot reshape array of size 4 into shape (2,4)
```

SEARCH STACK OVERFLOW

```
c = np.arange(4)
c
```

```
array([0, 1, 2, 3])
```

```
np.resize(c,(2, 4))
```

```
array([[0, 1, 2, 3],
       [0, 1, 2, 3]])
```

```
d = c
```

```
np.resize(c, (2, 4))
```

```
array([[0, 1, 2, 3],
       [0, 1, 2, 3]])
```

Saved successfully!

```
print(a.ndim)
```

```
2
```

```
import numpy as np
```

```
a = np.arange(4)
a.resize((2, 4))
```

```
a
```

```
array([[0, 1, 2, 3],
       [0, 0, 0, 0]])
```

```
b = np.arange(4)
np.resize(b, (2, 4))
```

```
array([[0, 1, 2, 3],
       [0, 1, 2, 3]])
```

```
c = a
```

```
a.resize((3, 4))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-f7f6576b419d> in <module>()
----> 1 a.resize((3, 4))

ValueError: cannot resize an array that references or is referenced
by another array in this way.
Use the np.resize function or refcheck=False
```

SEARCH STACK OVERFLOW

```
c.resize((3, 4))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-baba613caaad> in <module>()
----> 1 c.resize((3, 4))

ValueError: cannot resize an array that references or is referenced
by another array in this way.
Use the np.resize function or refcheck=False
```

SEARCH STACK OVERFLOW

Saved successfully!

```
a
```

```
array([[0, 1, 2]])
```

```
a.T
```

```
array([[0],
       [1],
       [2]])
```

```
a = np.arange(3)
```

```
a
```

```
array([0, 1, 2])
```

```
a.T
```

```
array([0, 1, 2])
```

```
# nd array --> flatten
```

```
a = np.arange(12).reshape(3, 4)
```

```
a
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
a.flatten()
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
a
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
a.reshape(12).shape
# 1, 12
# 12, 1
# 12
```

```
(12,)
```

Saved successfully!

```
a
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
a.reshape(6, -1)
```

```
array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
```

```
a.reshape(-1, 6)
```

```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

```
# zeros
```

```
np.zeros(3)
```

```
array([0., 0., 0.])
```

```
np.zeros((2, 3))
```

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

```
np.ones(3)
```

```
array([1., 1., 1.])
```

```
np.ones((2, 3))
```

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

```
# (2, 3), all 5s
```

```
np.ones((2, 3)) * 5
```

```
array([[5., 5., 5.],
       [5., 5., 5.]])
```



```
dtype('float64')
```

```
# diagonal matrices
np.diag([1, 2, 3])
```

```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

```
a = np.diag([1,2,3])
b = np.diag(a)
```

```
b
```

```
array([1, 2, 3])
```

```
np.identity(3)
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
# Indexing and slicing
```

```
m1 = np.arange(12)
```

```
m1
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
m1[0]
```

```
0
```

```
m1[12]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-44-0abd94d7097d> in <module>()
----> 1 m1[12]

IndexError: index 12 is out of bounds for axis 0 with size 12
```

SEARCH STACK OVERFLOW

```
m1 = np.arange(1,10).reshape((3,3))
print(m1)
```

Saved successfully!

```
 [4 5 6]
 [7 8 9]]
6
```

```
m1[1, 2] # use this syntax
```

```
6
```

```
m1 = np.array([100,200,300,400,500,600])
```

```
m1[[2,3,4,1,2,2]]
```

```
array([300, 400, 500, 200, 300, 300])
```

```
m1 = np.arange(9).reshape((3,3))
m1
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

```
m1[[0, 0, 1], [0, 1, 2]] # important
```

```
array([0, 1, 5])
```

```
m1 = np.arange(12)
m1
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
m1[:5]
```

```
array([0, 1, 2, 3, 4])
```

```
m1 = np.array([[0,1,2,3],
     [4,5,6,7],
     [8,9,10,11]])
```

```
m1[0, 1:3]
```

```
array([1, 2])
```

```
# [[2, 3],
#  [6, 7],
#  [10,11]]
```

Saved successfully!

```
       [ 6,  7],
       [10, 11]])
```

```
a = np.arange(6)
a
```

```
array([0, 1, 2, 3, 4, 5])
```

```
a[4:] = 10
```

```
a
```

```
array([ 0,  1,  2,  3, 10, 10])
```

```
a = np.array([1,2,3,4,5])
b = np.array([8,7,6])
a[2:] = b[::-1]
a
```

```
array([1, 2, 6, 7, 8])
```

```
# fancy indexing- masking
```

```
m1 = np.arange(12).reshape(3, 4)
```

```
m1
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
m1 < 6 # mask
```

```
array([[ True,  True,  True,  True],
       [ True,  True, False, False],
       [False, False, False, False]])
```

```
m1[m1 < 6]
```

```
array([0, 1, 2, 3, 4, 5])
```

```
m1[m1 % 2 == 0]
```

```
array([ 0,  2,  4,  6,  8, 10])
```

```
a = np.arange(11)
```

Saved successfully!

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
# divisible by 2 or 5
```

```
a[(a %2 == 0) | (a%5 == 0)]
```

```
array([ 0,  2,  4,  5,  6,  8, 10])
```

```
a = np.arange(6)
mask = (a%2 == 0)
a[mask] = -1
print(a)
```

```
[-1  1 -1  3 -1  5]
```

```
a
```

```
array([-1,  1, -1,  3, -1,  5])
```

```
a.reshape(3, -2)
```

```
array([[-1,  1],
       [-1,  3],
       [-1,  5]])
```

```
a = np.array([1, 2, 3])
b = np.array([1, 2, 3])
print (a * b)
```

```
[1 4 9]
```

Saved successfully!

0s completed at 23:47