ENGG1811 Computing for Engineers

Week 4A: For-loops, list of lists

This week

- For-loops
- In-class project: Counting the number of heartbeats
- List of lists

Recap of for-loops

- You learnt about for-loops last week. An example that we went through was:

```
num_list = [2,-3,4,-5]
new_list_1 = []  # An empty list
for num in num_list:
    new_list_1.append(num**3)

print(new_list_1)
```

num_list = [2, -3, 4, -5]

↓ ↓ ↓ ↓ Cubing each element

new_list_1 = [8, -27, 64, -124]

Doing more with for-loops

- So far, you’ve used a for-loop to apply the same operation to each element individually

- You can do more by ”memorising” some intermediate results

Summing a sequence of numbers

- I will roll a 12-sided die 100 times
- You are not allowed to write any of the numbers down (Hopefully you are not a mnemonist!)
- I want you to tell me what the sum of those 100 numbers are
- How will you do it?
  - Write down the steps that you take to sum up the sequence of numbers
  - In particular, I want you to think whether you find yourselves doing a number of steps repeatedly. If yes, make a note of that in your answer too.

https://openclipart.org/detail/92041/dice

Let us have a go

- We will use an online die to help us

Algorithm

Quiz

- Let us assume that you use a variable called running_total to remember the total so far

Add the number from the die to the number you have remembered (i.e., running_total)

Update the value of the variable running_total

- Question: How will you write the above task using one line of pseudo-code

Summing up the numbers in a list

```
num_list = [5,6,-2,3]
running_total = 0
for k in num_list:
    running_total += k

print(running_total)
```

We will copy the code to Python tutor

http://pythontutor.com/

Note 1: `running_total += k` is a short hand for `running_total = running_total + k`

Note 2: You could have used sum(num_list) but it's good to learn what is behind it

Remark

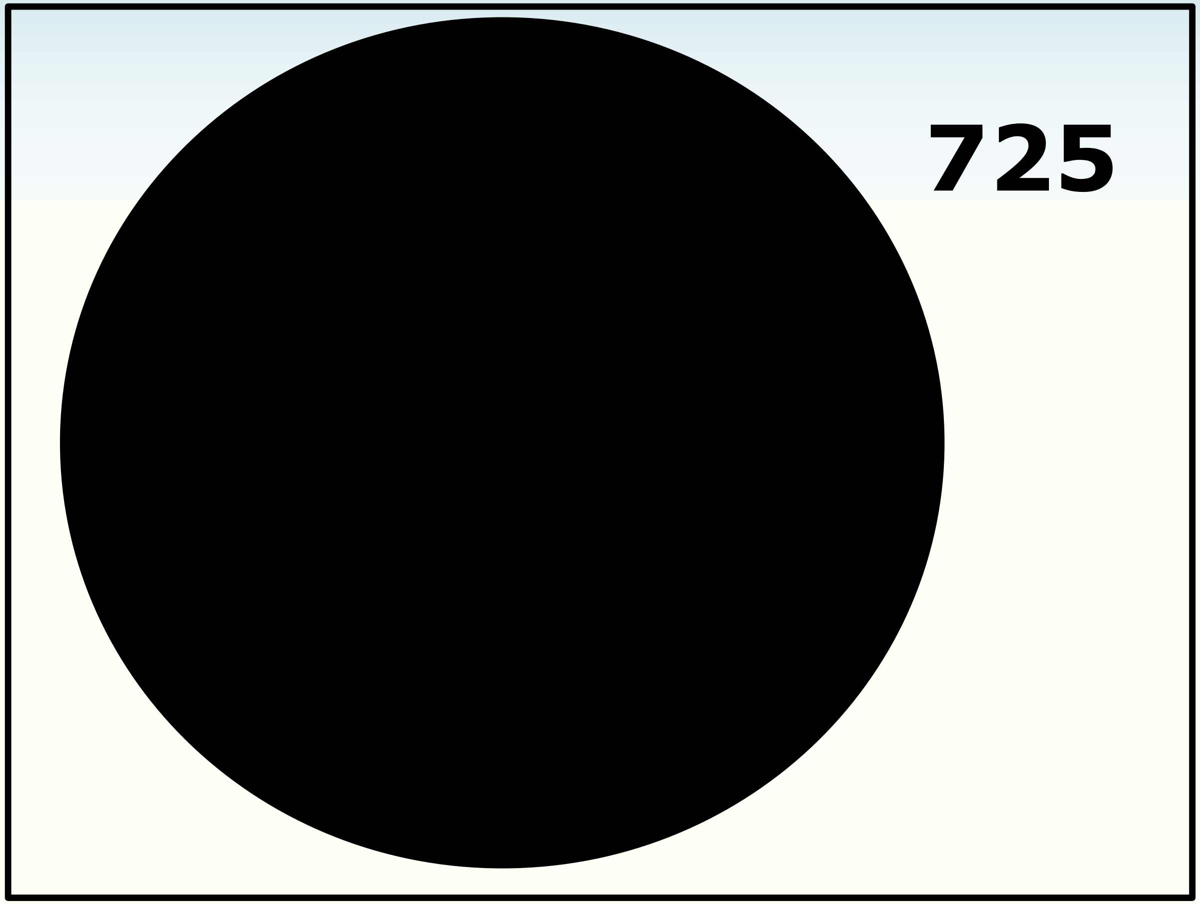
- In addition to +=, Python also has

  -=   *=   /=   //=   **=   %=

Maximum in a sequence of numbers

- I will tell you 100 numbers one by one
- You are not allowed to write any of the numbers down
- After I have told you all the 100 numbers, I want you to tell me what the maximum of those 100 numbers are.
- How will you do it?

Puzzle



- I want to find the largest number in the rectangle

1. If the largest number behind the circle is 699, what is the largest number in the rectangle?
2. What if the largest number behind the circle is 934?

Finding the maximum

Define a variable called **max_so_far** which is the maximum found so far

```
num_list = [  4,    2,    6,    12,    -3,    17]
```

max_so_far is 6

new_number is 12

Pseudo code:

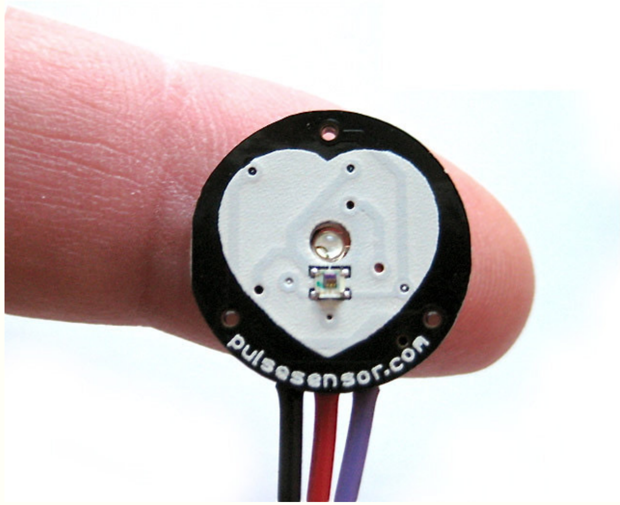
If new_number > max_so_far then

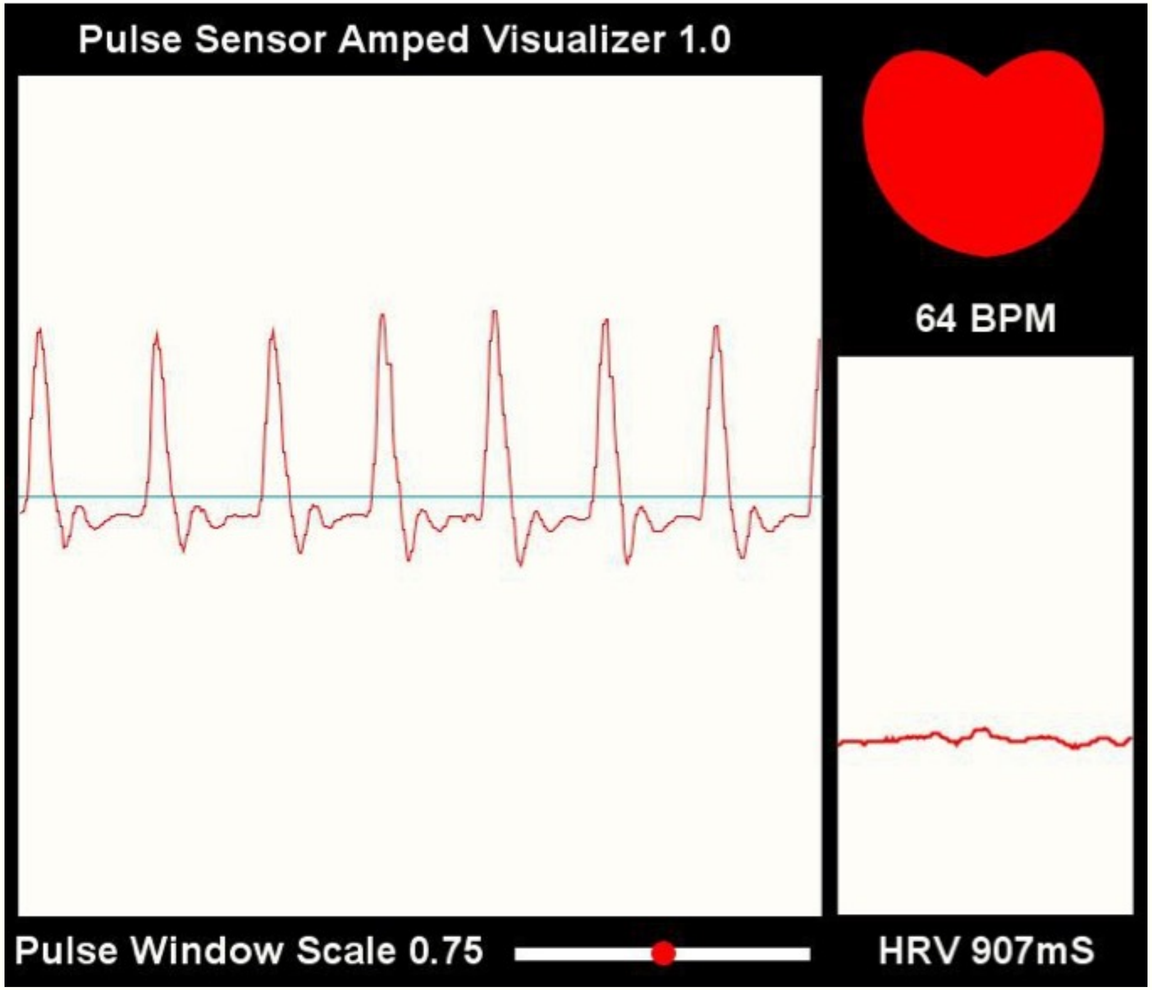
    Update max_so_far to be new_number

Let us finish it in find_max_prelim.py

Counting heart beat automatically

Pulse oximetry sensor



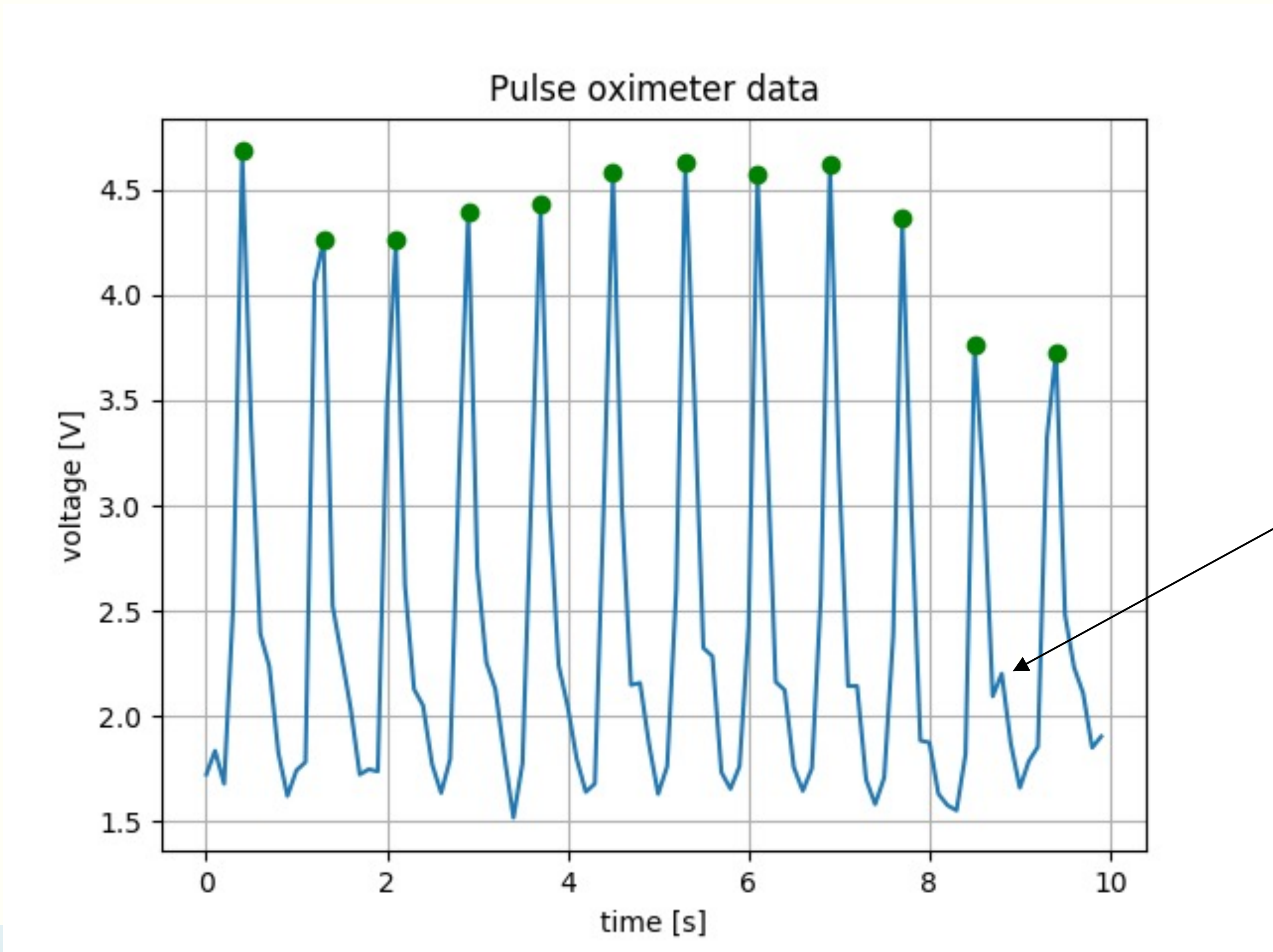


We will use list and for loop to understand how to count heart beat automatically

http://pulsesensor.com

Counting the number of heart beats

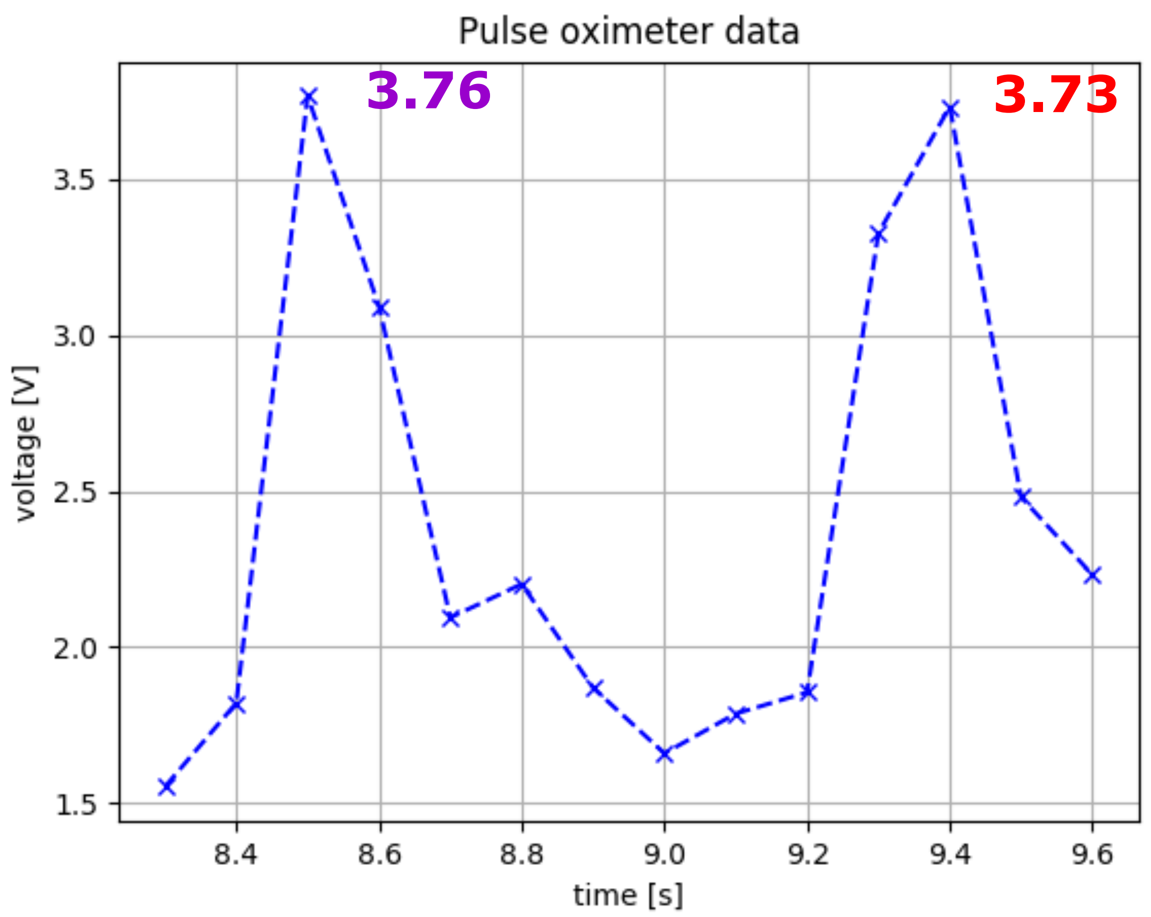
- We will count the number of heart beats by counting the number of tall peaks
  - The tall peaks are marked with green dots



This is a peak but it is not a tall peak

Is it a peak?

To determine whether a point is a peak, you need to look at that point and its two neighbours. Given 3 points, there are 4 possible ways to arrange them

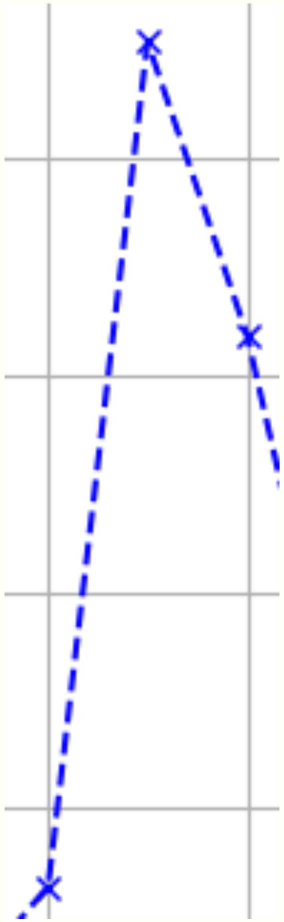


[1.55 1.82 **3.76** 3.09 2.09 2.20 1.87 1.66 1.79 1.86 3.33 **3.73** 2.48 2.23]

Is it a peak?

Can you come out with a logical condition that says whether 3 consecutive points form a peak or not?

1.82 **3.76** 3.09

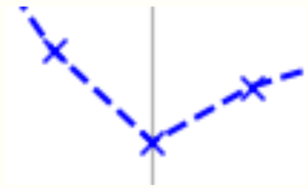


Middle point is a peak

2.20 1.87 1.66

1.66 1.79 1.86

1.87 1.66 1.79



Middle point is NOT a peak

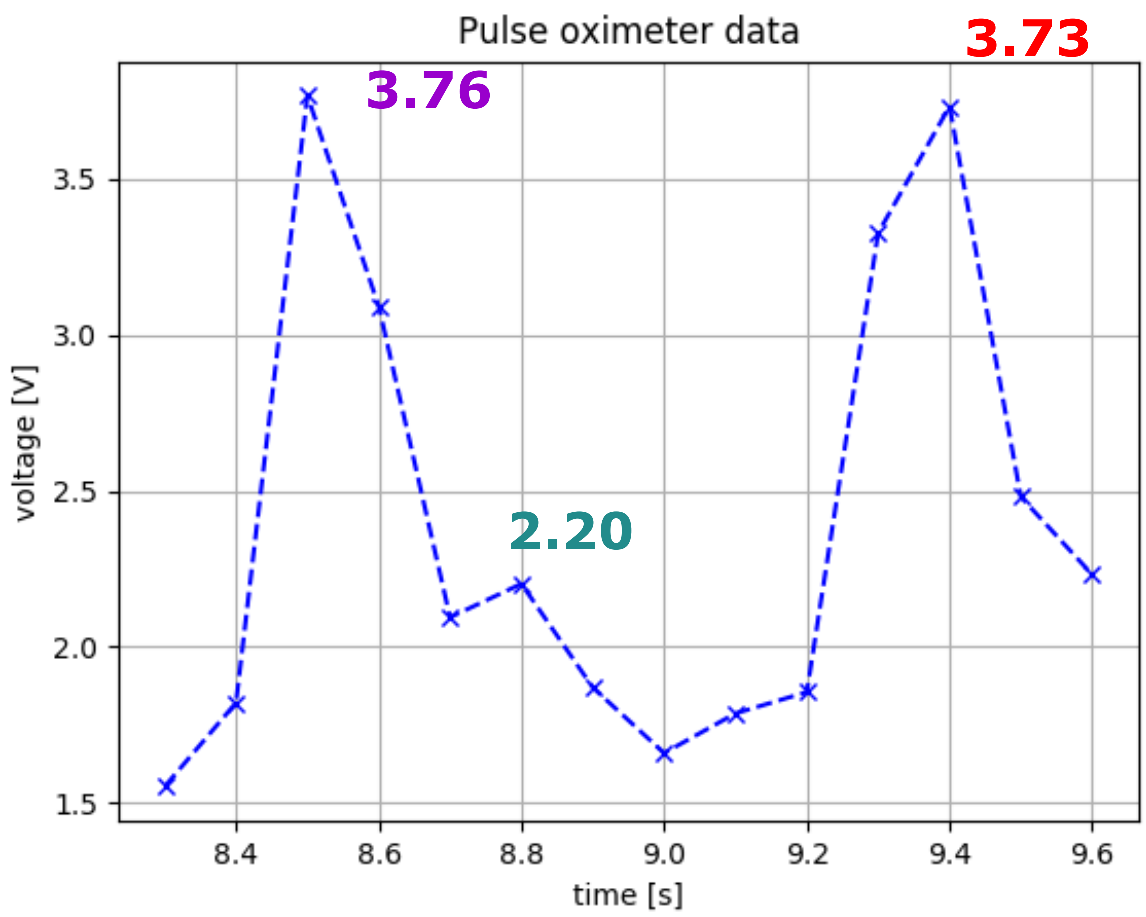
Let us complete the code and check

- Code in heart_beat_prelim.py
- Complete line 41 with the condition to detect a peak

```
40  def is_a_peak(left, middle, right):
41      if :
42          return True
43      else:
44          return False
45
46  # Desired outcomrs:
47  # is_a_peak(1.82, 3.76, 3.09) returns True
48  # is_a_peak(2.20, 1.87, 1.66) returns False
49  # is_a_peak(1.66, 1.79, 1.86) returns False
50  # is_a_peak(1.87, 1.66, 1.79) returns False
51
```

Tall or not tall?

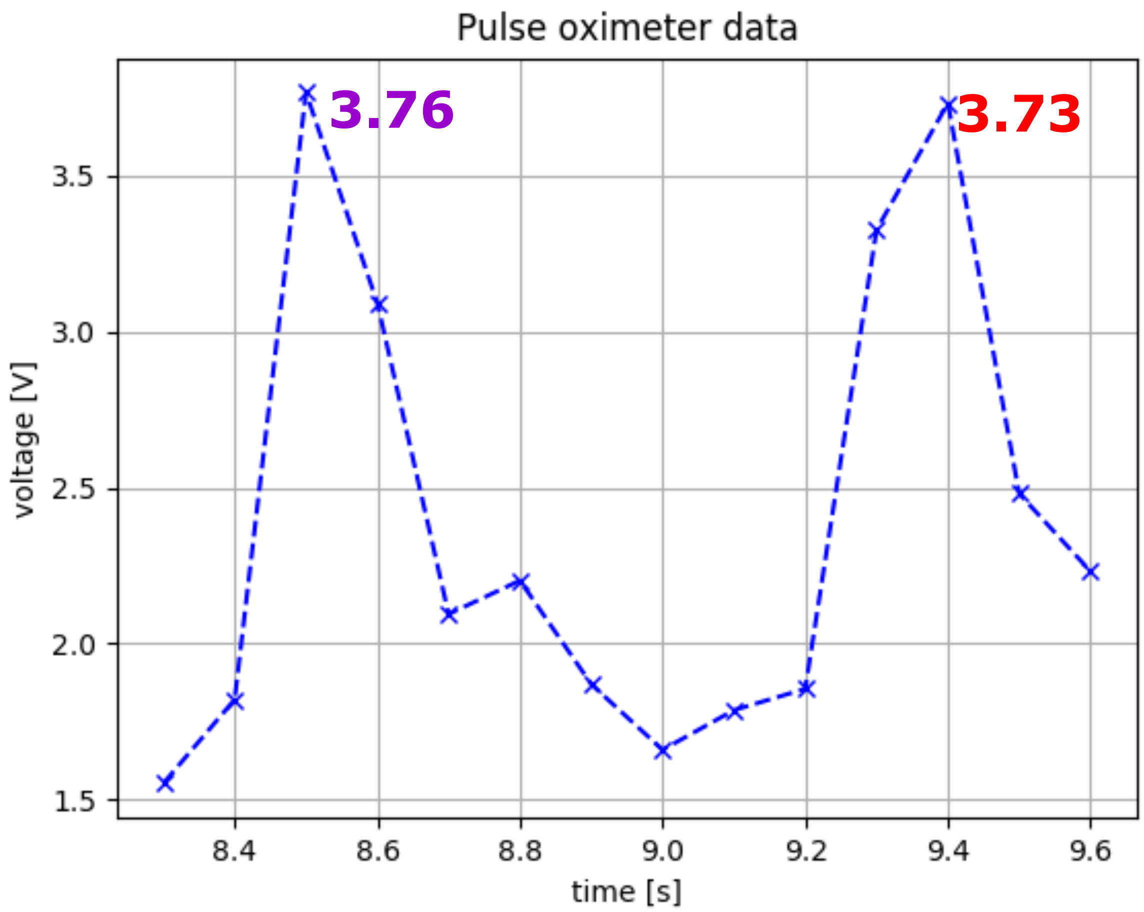
We can set a threshold and require the value at the peak must be greater than or equal to this threshold



[1.55 1.82 **3.76** 3.09 2.09 **2.20** 1.87 1.66 1.79 1.86 3.33 **3.73** 2.48 2.23]

How to count?

Consider a small section of data. Each voltage value in the list below corresponds to a 'x' in the graph.



[1.55 1.82 **3.76** 3.09 2.09 2.20 1.87 1.66 1.79 1.86 3.33 **3.73** 2.48 2.23]

We will consider 2 methods.

- Method 1 creates a list of marking. Similar to creating a list from another list that we used in Week 3
- Method 2 memorises the result.

How to count?

Let us say we mark a tall peak with Y and non-tall peak with N

– Note: We won't mark the two ends because there is not enough information to tell they are peaks or not

[1.55 1.82 **3.76** 3.09 2.09 2.20 1.87 1.66 1.79 1.86 3.33 **3.73** 2.48 2.23]

N **Y** N N N N N N N N **Y** N

- We can count the number of tall peaks by counting the number of Y's
- How can you mark a list? The list is stored in the computer memory and is not accessible by a pen
- If you stare at this line of markings for a while, you may have an idea

A list of markings

[1.55 1.82 **3.76** 3.09 2.09 2.20 1.87 1.66 1.79 1.86 3.33 **3.73** 2.48 2.23]

N **Y** N N N N N N N N **Y** N

- The markings ‘Y’ and ‘N’ is a sequence so we can store them in a list
- Python has a function to count the occurrence of a certain value in a list. We can do this.
- Let us explore an alternative

An alternative way to mark the list

[1.55 1.82 **3.76** 3.09 2.09 2.20 1.87 1.66 1.79 1.86 3.33 **3.73** 2.48 2.23]

N **Y** N N N N N N N N **Y** N

- Instead of using ‘Y’ and ‘N’ to mark the list, I would like to ask you to mark the list in a different way
  - I want you to use numbers to mark the list
  - If you choose the numbers in a certain way, then the sum of the sequence of numbers is also the number of tall peaks
- Any suggestions?

An alternative way to mark the list (cont'd)

[1.55 1.82 **3.76** 3.09 2.09 2.20 1.87 1.66 1.79 1.86 3.33 **3.73** 2.48 2.23]

0 **1** 0 0 0 0 0 0 0 0 **1** 0

- You can mark with the integers 1 and 0
  - 1 means the point is a tall peak
  - 0 means it is not
- If you can come out such a list of 1's and 0's, then you can find the number of tall peaks

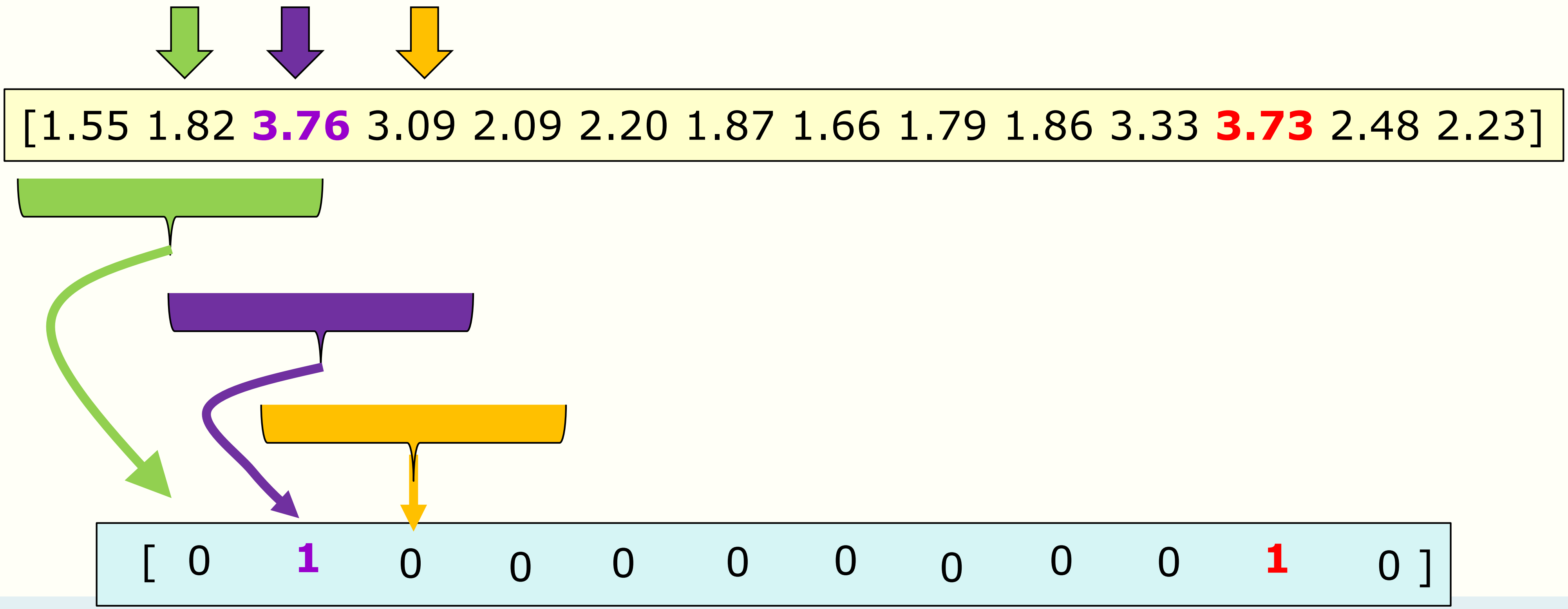
Need to sweep through the list

Given list

[1.55 1.82 **3.76** 3.09 2.09 2.20 1.87 1.66 1.79 1.86 3.33 **3.73** 2.48 2.23]

[0 **1** 0 0 0 0 0 0 0 0 **1** 0]

Want to create



heart_beat_prelim.py

Method 1: Create a list of marking

- Similar to creating a list from another list in Week 3, except
  - You use 3 consecutive elements at a time
  - The new list is shorter list compared to the given list

```
68  is_peak_list = []
69
70  for k in range( ):
71      if is_a_tall_peak(voltage_list[k],voltage_list[ ],voltage_list[ ]):
72          is_peak_list.append(1)
73      else:
74          is_peak_list.append(0)
75
76  print('Method 1: The number of heart beats is:',sum(is_peak_list))
```

Fill in Lines 70, 71 so that the code will create the list of marking.

Hint: The aim of the loop is to check:

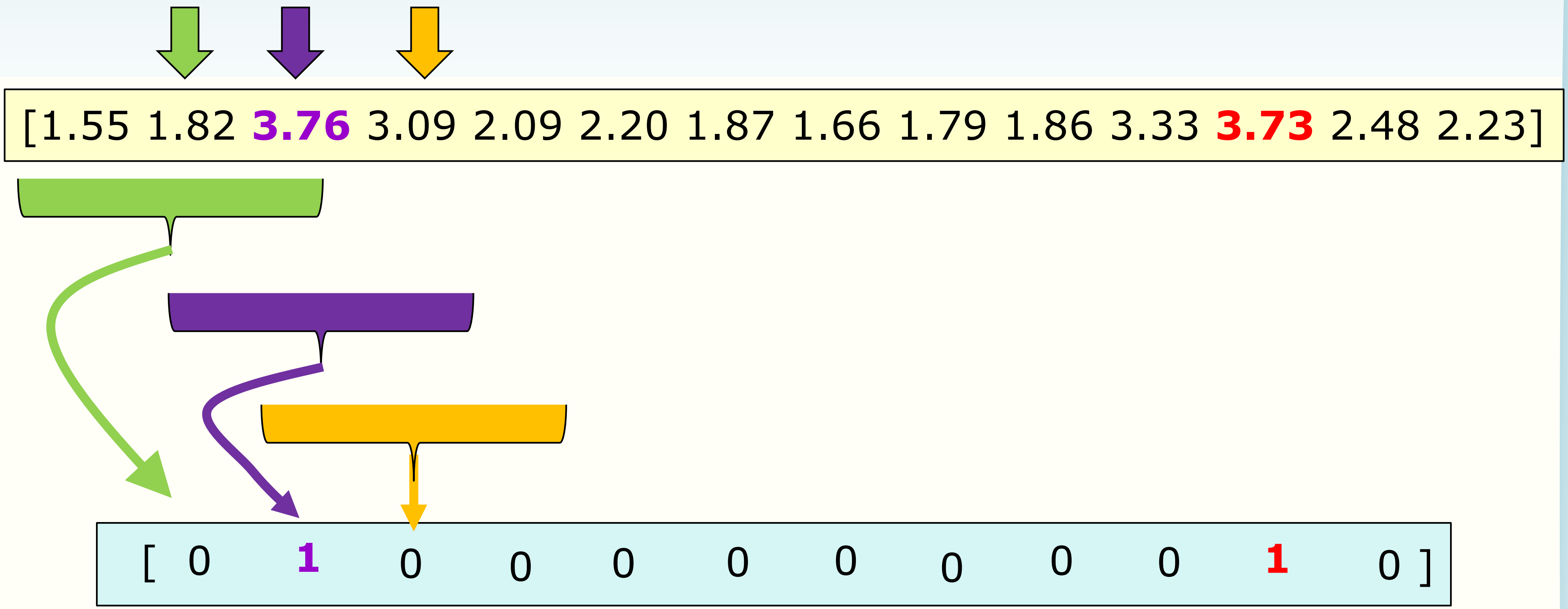
is_a_tall_peak(voltage_list[0], voltage_list[1], voltage_list[2])

is_a_tall_peak(voltage_list[1], voltage_list[2], voltage_list[3])

is_a_tall_peak(voltage_list[2], voltage_list[3], voltage_list[4])

...

Method 2: ”Memorise” some result



- You can get the number of heart beats by summing up this list
- You summed up a list of numbers on p.9 using a for-loop
- Here, you only need to sum up the 1’s and you can ignore the zeros

Method 2: Memorise the result

- File: heart_beat_prelim.py
- Fill in Lines 88 and 89 in the same way as you did for method 1
- Complete Line 90 so that the variable num_heart_beat_so_far will contain the number of heart beats at the end of the loop
  - Consult p.9 if you need

```
86 num_heart_beat_so_far = 0
87
88 for k in range( ):
89     if is_a_tall_peak(voltage_list[k],voltage_list[ ],voltage_list[ ]):
90         num_heart_beat_so_far
91
92 print('Method 2: The number of heart beats is:',num_heart_beat_so_far)
```

What wrong with this code?

- Let us have a look at the code in mean_abs_bad.py

- What the code wants to do is:
  - For each list
    - Compute the absolute value of each element
    - Sum up the absolute values
    - Divide the sum by the number of elements to obtain the mean

- Why do you think the code is bad?
  - How would you fix it?

Avoid repeating code

```
17 # dataset0
18 total = 0
19 for datum in dataset0:
20     total += abs(datum)
21 mean_abs0 = total / len(dataset0)
22 print('Dataset 0 average = ',mean_abs0)
23
24 # dataset1
25 total = 0
26 for datum in dataset1:
27     total += abs(datum)
28 mean_abs1 = total / len(dataset1)
29 print('Dataset 1 average = ',mean_abs1)
30
31 # dataset2
32 total = 0
33 for datum in dataset2:
34     total += abs(datum)
35 mean_abs2 = total / len(dataset2)
36 print('Dataset 2 average = ',mean_abs2)
```

The code for computing the average of the absolute value is repeated a few times:

Lines 18-22, 25-29

32-36, 39-43

Why repeating code is bad? Say you want to compute mean rather than mean of absolute value, you need to edit all the code.

Using function to hide details

```python
def mean_abs(data_list):
    total = 0
    for datum in data_list:
        total += abs(datum)
    mean_abs_value = total / len(data_list)
    return mean_abs_value
```

```python
# dataset0
mean_abs0 = mean_abs(dataset0)
print('Dataset 0 average = ',mean_abs0)

# dataset1
mean_abs1 = mean_abs(dataset1)
print('Dataset 1 average = ',mean_abs1)

# dataset2
mean_abs2 = mean_abs(dataset2)
print('Dataset 2 average = ',mean_abs2)
```

Code in mean_abs_improved1.py

All the computation of mean absolute value now goes in a function

The code looks less messy and is easier to understand.

We can improve this code further.

List of lists

```
In [9]: a = [[23,24,25,26],[31,32,33]]

In [10]: a[0]
Out[10]: [23, 24, 25, 26]

In [11]: a[1]
Out[11]: [31, 32, 33]

In [12]: a[1][2]
Out[12]: 33
```

Using list of lists to improve the code (1)

```
# dataset0
mean_abs0 = mean_abs(dataset0)
print('Dataset 0 average = ',mean_abs0)

# dataset1
mean_abs1 = mean_abs(dataset1)
print('Dataset 1 average = ',mean_abs1)

# dataset2
mean_abs2 = mean_abs(dataset2)
print('Dataset 2 average = ',mean_abs2)
```

This part is repetitive. In order to use the for-loop, we need to use list of lists for the original data.

```
# 4 data sets
dataset0 = [-1.6, 1.8,-1.8,-2.0, 1.5]
dataset1 = [ 1.8,-1.6, 1.6,-1.8,-2.2]
dataset2 = [-1.6,-1.8,-1.9, 2.3,-2.1]
dataset3 = [ 1.6, 1.7, 2.0, 2.4]
```

Lesson: How you store your data can make your code cleaner!

We will do this in class. Improved code on the next page.

Using list of lists to improve the code (2)

```
# %% Define a function to compute mean absolute value of
# a list of numbers
def mean_abs(data_list):
    total = 0
    for datum in data_list:
        total += abs(datum)
    mean_abs_value = total / len(data_list)
    return mean_abs_value

# %%
# 4 data sets
dataset0 = [-1.6, 1.8,-1.8,-2.0, 1.5]
dataset1 = [ 1.8,-1.6, 1.6,-1.8,-2.2]
dataset2 = [-1.6,-1.8,-1.9, 2.3,-2.1]
dataset3 = [ 1.6, 1.7, 2.0, 2.4]
# Turn the datasets into a list of lists
datasets = [dataset0, dataset1, dataset2, dataset3]

# Loop through the datasets
# The function mean_abs computes the mean of the absolute
# value of the elements in a list
for k in range(len(datasets)):
    mean_abs_value = mean_abs(datasets[k])
    print('Dataset',k,'average = ',mean_abs_value)
```

Changes

Function reuse

- There are two reasons why functions are important. You can reuse them and abstraction.

- You have developed the function mean_abs() and you can re-use it in any of your program by simply importing it

- This is the beauty of software. Code once and use forever and whenever.

Abstraction

- Abstraction hides details
- It allows us to use a piece of software of code as if it were a black box, i.e. something whose interior details we cannot see, don't need to see or shouldn't even want to see
  - For example, many of us use math.cos(), math.sin() etc. as a black box

- Quoted from: John V. Guttage, "Introduction to Computation and Programming Using Python", MIT Press. [Note: The code in the book is written in Python 2.]

Graph plotting is abstraction in action!

- You can view Lines 17-25 as commands for plotting graphs
- It's important to realise that each line calls a function
  - Line 18: The plot function has two inputs. The first is the data in the x-axis. The second input is the data in the y-axis.
    - Where is the output of this function?
  - Line 21: The input is a string which is the text of the title of the graph
- You are using the fruit of abstraction and don't you love it!

```
17 fig1 = plt.figure()            # create a new figure
18 plt.plot(load,length)          # plot(data in x-axis, data in y-axis)
19 plt.xlabel('load [lbf]')       # label for x-axis
20 plt.ylabel('length [inches]')   # label for y-axis
21 plt.title('Tensile strength test') # title of the graph
22 plt.grid()  # display the grid
23 plt.show()  # to display the graph
24 fig1.savefig('tensil_test.png') # save the graph as a PNG file
25 fig1.savefig('tensil_test.pdf') # save the graph as a PDF file
```

Summary

- For-loops
  - Remembering intermediate results
  - Applications: sum, max
- Counting heart beats
  - An example of using programming to solve a computational problem
  - An example of processing a data sequence which puts together a few skills that you've learnt: list, for-loop, conditional, logical expression
- List of lists
  - Moral: Use data type to reduce replications
- Function reuse and abstraction