```
def findpeaks01(values, mph): # 'values' is a list, 'mph' is a float
    size = len(values)
    listOfPeaks = []
    for i in range(1, size - 1):
        preval = values[i - 1]
        val = values[i]
        nextval = values[i + 1]
        if val > preval and val > nextval and val > mph:
           listOfPeaks.append(i)
        else:
            continue
    return listOfPeaks
```

it does not handle plateaus ... >= and further controls

if mph is None, there is a comparisong between float and NoneType ... this does not work

this is completely redundant ...

"else don't do anything"

```
def findpeaks02(seq, mph):
    size = len(seq)
    if size > 2:
        peaks = []
        for n in range(1, size-1):
            value = float(seq[n])
            preval = float(seq[n-1])
            nextval = float(seq[n+1])
            diff_prev = value - preval
            diff_next = value - nextval
            if (value > preval) and (value >= nextval):
                if mph == 0:
                    peaks.append(n)
                elif (diff_prev >= mph) and (diff_next >= mph or diff_next == 0):
                    peaks.append(n)
    else:
        peaks = "Input not compliant"
    return peaks
```

if there are less than 2 values, everything would behave in a consisten way anyway

they are expected to be float values from the beginning

do we ask ourself this question everytime, or would it be enough to ask it once, and act accordingly? consider for instance 10000 values ...

what are you returning when size <= 2 ... peaks is not defined

---

```
def findpeaks03(list_in,thres):
    length = len(list_in)
    list_of_peaks = []
    if thres == None:
        for i in range(1, length - 2):
            if list_in[i] > list_in[i - 1]:
                if list_in[i] >= list_in[i + 1]:
                    list_of_peaks.append(i)
    else:
        for i in range(1, length - 2):
            if list_in[i] > thres:
                if list_in[i] > list_in[i - 1]:
                    if list_in[i] >= list_in[i + 1]:
                        list_of_peaks.append(i)
    return list_of_peaks
```

a peak on the element before the last one is not identified

```
def findpeaks04(val,mph):
    peak=[]
    len_val=len(val)-1
    for i in range(1,len_val):
      diff_p= abs(val[i]-val[i-1])
      diff_s=abs(val[i]-val[i+1])
      if diff_p >0 and diff_s >0:
         if mph==None:
             peak.append(i)
         elif val[i]>mph:
             peak.append(i)
      i=i+1
    return peak
```

do we really get the peaks by looking at the absolute difference

do we ask the same question for all items in the list?

```
def findpeaks05(finlist, mph): #returns a list of positions of the peaks in the input file
    peaks=[]
    l=len(finlist)

    if mph == None:
        for i in range(1,l-1):
            peaks.append(finlist[i])
    else:
        for i in range(1,l-1):
            valueafter=finlist[i+1]+mph
            valuebefore=finlist[i-1]+mph
            if finlist[i]>=valueafter and finlist[i]>=valuebefore and finlist[i]>=mph:
                peaks.append(i)

    return peaks
```

adds all points?

if at the end of the plateau there is a higher point, the first point of the plateau is identified as a peak ....

```
def findpeaks06(values,mph):
    peaks=[]
    size=len(values)
    for i in range(1,size-1):
        if values[i]>values[i-1] and values[i]>=values[i+1] and values[i]>mph:
            peaks.append(i)
    return peaks
```

if mph is None, there is a comparisong between float and NoneType ... this does not work

if at the end of the plateau there is a higher point, the first point of the plateau is identified as a peak ....

```
def findpeaks07(ds.temperature, thr):
    fin = open("filenametemp.txt", "w")
    fin.write(ds.temperature)
    fin.close()
    try:
        fileused = open("filenametemp.txt", "r")
    except IOError:
        print(False)
    else:
        vals = []
        thr = raw_input()
        for line in fin:
            peakline = line.strip()
            peakinfo = float(peakline.strip(','))
            vals.append(peakinfo)
        lenv = len(vals)
        listpeaks = []
        if thr != None :
            thr = float(thr)
            for i in range(1, lenv):
                if vals[i] > vals[i-1] and vals[i] > vals[i+1] and vals[i] >= thr:
                    listpeaks.append(i)
                else :
                    continue
        else :
            for i in range(1, lenv):
                if vals[i] > vals[i-1] and vals[i] > vals[i+1] :
                    listpeaks.append(i)
                else :
                    continue
    fileused.close()
    return listpeaks
```

it does not handle plateaus ... >= and further controls

this is completely redundant ... "else don't do anything"

this is completely redundant ... "else don't do anything"

```
def findpeaks08(list, mph):
    import math
    output = []
    l = len(list)
    for i in range(1, l-1):
        if list[i] == list[i + 1]:
            j = i
            while list[i]==list[i+1]:
                i=+1
            if list[i + 1] < list[i] and list[i]>mph:
                output.append(j)
                break
            else:
                continue
        elif abs(list[i-1] < list[i]) and abs(list[i] > list[i+1]) and list[i] > mph:
            output.append(i)
        else:
            continue
    return output
```

it does not handle plateaus ... >= and further controls

if mph is None, there is a comparisong between float and NoneType ... this does not work

this is completely redundant ... "else don't do anything"

this is completely redundant ... "else don't do anything"

```
def findpeaks09(list_input, mph):
    list_output=[]
    if mph == None:
        mph = 0
    else:
        mph = float(mph)
    #effective peaks
    for element in range(1,n-1):
        diffprevious = list_input[element] - list_input[element - 1]
        diffnext = list_input[element] - list_input[element + 1]
        if diffprevious > mph and diffnext > mph:
            list_output.append(int(element))
        #plateau
        elif diffprevious > mph and diffnext == 0:
            list_output.append(int(element))
    return list_output
```

filtering out negative peaks ... not what you want to do

already float

missing statement

```
n = len(list_input)
```

if at the end of the plateau there is a higher point, the first point of the plateau is identified as a peak ....

---

two for loops one inside the other?

```
def findpeaks10(lst, mph):
    peaks = []
    size = len(lst) - 1
    i = 1
    for n in range(1, size):
        if mph == 'None':
            for n in range(1, size):
                if lst[n]>lst[n - 1] and lst[n]>=lst[n + 1]:
                    peaks.append(i)
                i+=1
        else:
            for n in range(1, size):
                if lst[n]>mph and lst[n]>=lst[n-1] and lst[n]>=lst[n+1]:
                    peaks.append(i)
                i+=1
    return peaks
```

it is not a string, it is a predefined python type

comparing item in position n and adding position i ?

```
def findpeaks11(listinput, mph):
    size = len(listinput)
    list_peaks = []
    for element in range(1,size-1):
        if listinput[element] >= listinput[element+1] and \
            listinput[element] > listinput[element-1] and listinput[element] >= mph:
            list_peaks.append(element)

    return list_peaks
```

if at the end of the plateau there is a higher point, the first point of the plateau is identified as a peak ....

if mph is None, there is a comparisong between float and NoneType ... this does not work

```
def findpeaks12(values, thr):
    l = len(values)
    peakvalues = []
    for i in range(0, l-2):
        if values[i+1] > values[i] and values[i+1] > thr:
            if values[i+1] == values[i+2]:
                  j = 0
                while values[i+1] == values[i+2+j]:
                  j = j+1
                  if values[i+2+j] > values[i+2+j+1]:
                        peakvalues.append(i+1)
            elif values[i+1] > values[i+2]:
                  peakvalues.append(i+1)
      return peakvalues
```

if thr is None, there is a comparisong between float and NoneType ... this does not work

needs to be fixed like this:
`if values[i+1] > values[i+2+j]:`

it is better to keep this as the first expected condtion, so in the `if` part ...

---

missing second parameter mph

```
def findpeaks13(values):
    peaks=[]      lenseq = len(values)
    if lenseq>2:
        oldvalue=float(values[0])
        value=float(values[0])
        newvalue=float(values[0])
        for i in range(1,lenseq-2):
            oldvalue=float(value)
            value=float(newvalue)
            newvalue=float(values[i])
            #now we check for the peak
            if mph==0 and value>newvalue and value>oldvalue and value>mph:
                peaks.append(i)
            elif mph!=0 and value>newvalue and value>oldvalue and value>mph:
                peaks.append(i)
            elif mph==0 and value<newvalue and value<oldvalue and value>mph:
                peaks.append(i)
            elif mph!=0 and value<newvalue and value<oldvalue and value>mph:
                peaks.append(i)
            else:
                continue
        print(peaks, mph)
    else:
        print(peaks, mph)
```

already float

hmhm strange conditions: finds initial plateau points that are nt peaks

if mph is None, there is a comparisong between float and NoneType ... this does not work

if you print them, it is not possible to use them ... the caller would like to use this information
if the instruction is in both the `if` part and the `else` part, it should be out of the `if` statement.

```
def findpeaks14( Y , mph0 )
if mph0 == str("None
               "):
    mph = 0.0
else:
    mph = float(mph0)

ListOfPeaks = list()
for i in range(1 , Lenght-1):
    Previous = float(Y[i-1])
    Analyzed = float(Y[i])
    Next = float(Y[i+1])
    mphPrevious = Analyzed - Previous
    mphNext = Analyzed - Next

    if (mphPrevious > mph and mphNext > mph) and \
            (Analyzed > Previous and Analyzed > Next) :
        ListOfPeaks.append(i)
    elif (mphPrevious > mph and mphNext >= mph) and \
            (Analyzed > Previous and Analyzed == Next):
      ListOfPeaks.append(i)
    else:
        continue

print(ListOfPeaks)

return ListOfPeaks
```

it is not a string, it is a predefined python type

filtering out negative peaks ... not what you want to do

already float

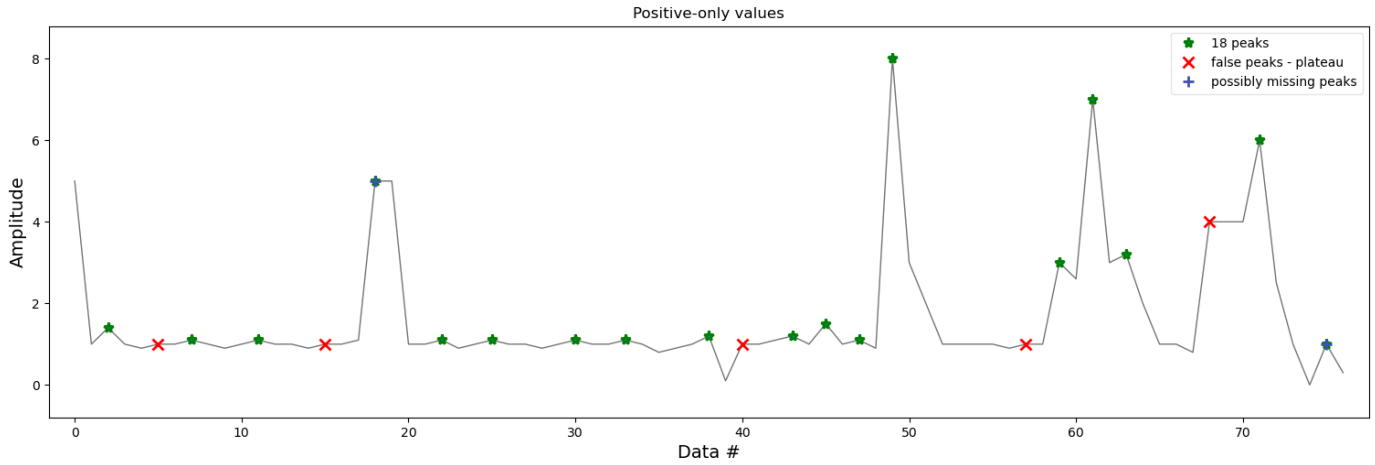
already float

this is completely redundant ...
"else don't do anything"

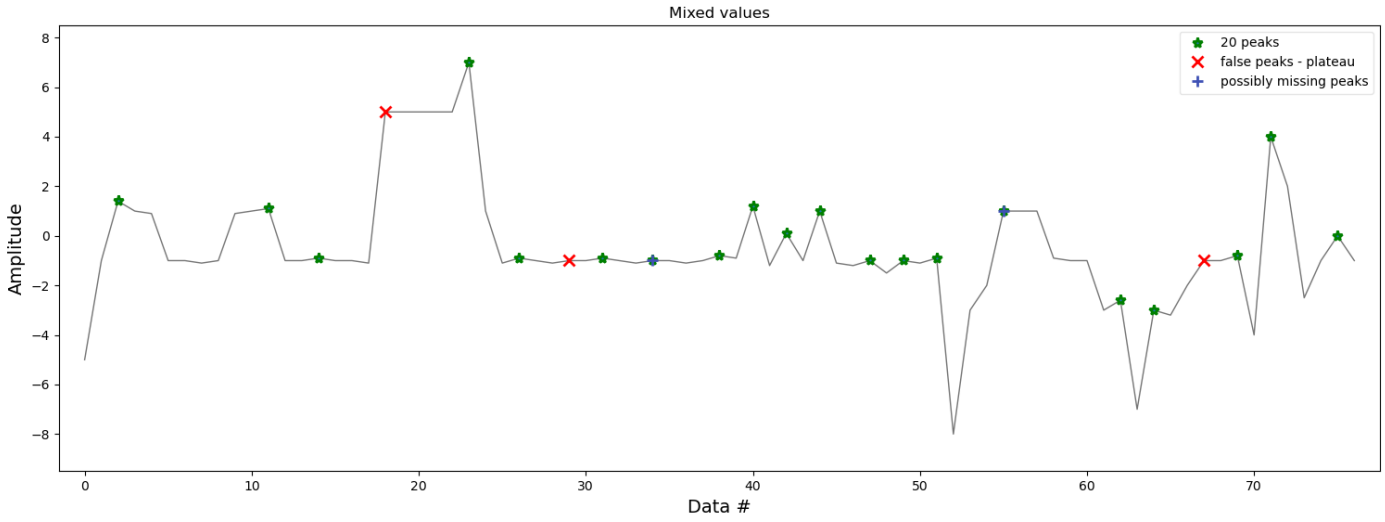
if at the end of the plateau there is a higher point, the first point of the plateau is identified as a peak ....

this is not compatible with the definition of a subprogram ... check the syntax

[5.0, 1.0, 1.4, 1.0, 0.9, 1.0, 1.0, 1.1, 1.0, 0.9, 1.0, 1.1, 1.0, 1.0, 0.9, 1.0, 1.0, 1.1, 5.0, 5.0, 1.0, 1.0, 1.1, 0.9, 1.0, 1.1, 1.0, 1.0, 0.9, 1.0, 1.1, 1.0, 1.0, 1.1, 1.0, 0.8, 0.9, 1.0, 1.2, 0.1, 1.0, 1.0, 1.1, 1.2, 1.0, 1.5, 1.0, 1.1, 0.9, 8.0, 3.0, 2.0, 1.0, 1.0, 1.0, 1.0, 0.9, 1.0, 1.0, 3.0, 2.6, 7.0, 3.0, 3.2, 2.0, 1.0, 1.0, 0.8, 4.0, 4.0, 4.0, 6.0, 2.5, 1.0, 0.0, 1.0, 0.3]



[-5.0, -1.0, 1.4, 1.0, 0.9, -1.0, -1.0, -1.1, -1.0, 0.9, 1.0, 1.1, -1.0, -1.0, -0.9, -1.0, -1.0, -1.1, 5.0, 5.0, 5.0, 5.0, 5.0, 7.0, 1.0, -1.1, -0.9, -1.0, -1.1, -1.0, -1.0, -0.9, -1.0, -1.1, -1.0, -1.0, -1.1, -1.0, -0.8, -0.9, 1.2, -1.2, 0.1, -1.0, 1.0, -1.1, -1.2, -1.0, -1.5, -1.0, -1.1, -0.9, -8.0, -3.0, -2.0, 1.0, 1.0, 1.0, -0.9, -1.0, -1.0, -3.0, -2.6, -7.0, -3.0, -3.2, -2.0, -1.0, -1.0, -0.8, -4.0, 4.0, 2.0, -2.5, -1.0, -0.0, -1.0]



[-5.0, -1.0, -1.4, -1.0, -0.9, -1.0, -1.0, -1.1, -1.0, -0.9, -1.0, -1.1, -1.0, -1.0, -0.9, -1.0, -1.0, -1.0, -1.1, -5.0, -5.0, -1.0, -1.0, -1.1, -0.9, -1.0, -1.1, -1.0, -1.0, -0.9, -1.0, -1.1, -1.0, -1.0, -1.1, -1.0, -0.8, -0.9, -1.0, -1.2, -0.1, -1.0, -1.0, -1.1, -1.2, -1.0, -1.5, -1.0, -1.1, -0.9, -8.0, -3.0, -2.0, -1.0, -1.0, -1.0, -0.9, -1.0, -1.0, -3.0, -2.6, -7.0, -3.0, -3.2, -2.0, -1.0, -1.0, -0.8, -4.0, -4.0, -2.0, -2.5, -1.0, -0.0, -1.0]