```
data = []
"^Leave Blank"
base_data = [2,10,8,4,20,18,9,45,43, 21.5, 107.5,105.5]


"""patterns to try:
if patterns are only 3 long then the iterative linear will pick them up
lin:           [0,2,4,6,8,10]
mult:          [3,-9,27,-81]
compl(*2+2):   [0,2,6,14,30,62,126]
compl(*2+1):   [0,1,3,7,15,31,63,127]
compl(*3-1):   [5,14, 41, 122, 365, 1094]
comp(+2,*5-1):[1, 14, 79, 404, 2029]
comp(+2*3/2): [3, 7.5, 14.25, 24.375] *decimals work
comp(*-5-3):   [2,-13,62, -313, 1562]
alt mult:      [5,-25, 125, -625]
iter(mult):    [2,6,12,36,72,144,432]
iter(lin):     [2,7,5,10,8,13,11,16,14]
compiter(*3+1,-2): [2, 7, 5, 16, 14, 43, 41]
compiter(*-3-1,+2*5): [2,-7,-25, 74, 380, -1141]
compiter(*5,-2,/2): [2,10,8,4,20,18,9,45,43, 21.5, 107.5,105.5]
compiter(+1,*4,/2): [2,3,12,6,7, 28, 14, 15,60,30,31]
[5, 25, 20, 10, 50, 45, 22.5, 112.5, 107.5, 53.75]
"""

def get_data(data_list, base_data):

    "this function gets the input data from the user and makes it a list of single digit integers"
    "(input must be one integer at a timefrom 0-9 and must be separated by blank space. if 0 0 is entered then base_data is set to data)"

    in_data = input("enter any number of digits(pref > 4) \n").split()

    in_data = [int(i) for i in in_data]

```

```
35      if in_data[0] == 0 and len(in_data) == 1:
36          return
37
38      if in_data[0] == 0 and in_data[1] == 0:
39          for items in base_data:
40              data_list.append(items)
41      else:
42          for items in in_data:
43              data_list.append(items)
44      return data_list
45
46  def is_same(data_list):
47
48      "Checks if each element in data_list is the same
    integer, if so returns true. Otherwise returns false"
49      "USED BY find_iterative_difference, checks that
    the repeat is in fact a repeat basically"
50
51      element1 = data_list[0]
52      for item in data_list:
53          if item != element1:
54              return False
55
56      return True
57
58  def print_result(type, in_data):
59      "print the next 5 values given the updated data
    list, also prints the type of pattern given the
    string var"
60      print(type,": ")
61      in_size = len(in_data)-5
62      for i in range(in_size):
63          print(in_data[i], end = ' ')
64      print(" ->", end = " ")
65
66      i = 0
67      while i<5:
68          print(in_data[-5+i], end=' ')
69          i+=1
70
71  "CONSTANT LINEAR FUNCTIONS"
```

```
 72 def list_diff(data_list):
 73
 74     "Finds and returns list difference to the list
    that is inputted by subtracting element i+1 from i"
 75
 76     lin_diff = []
 77     for i in range(len(data_list)-1):
 78         lin_diff.append(data_list[i+1]-data_list[i])
 79     #print("addition list:", end = ' ')
 80     #print(lin_diff)
 81     return lin_diff
 82
 83 """
 84 This method finds the pattern which is a lot more
    difficult then just extending the pattern
 85 def is_repeated(data_list):
 86     "checks if the inputted list has a pattern such
    as 2,3,2,3,2 ect., can be used for lin or mult"
 87     i = 0
 88     pat_list = []
 89     for element in range((len(data_list)//2) +1):
 90         pat_list.append(element)
 91
 92 def check_repeat(data_list,poss_rep):
 93     size = len(poss_rep)
 94     data_len = len(data_list)
 95     for steps in range(int(data_len/size)):
 96 """
 97 def lin_pattern(lin_diff, in_data):
 98
 99     difference = lin_diff[0]
100
101
102     for i in range(5):
103         val = in_data[-1] + difference
104         in_data.append(val)
105         #print(in_data[-1], end = ' ')
106     return True
107
108
109
```

```
110 "CONSTANT MULTIPLIER FUNCTIONS"
111 def list_multiplier(data_list):
112
113     "Finds and returns list multiplier to the list
    that is inputted by dividing element i+1 from i"
114
115     mult_diff = []
116     for i in range(len(data_list)-1):
117         mult_diff.append(data_list[i+1]/data_list[i
    ])
118     #print("multiplier list:", end = ' ')
119     #print(mult_diff)
120     return mult_diff
121
122 def mult_pattern(mult_diff, in_data):
123
124     difference = mult_diff[0]
125
126
127     for i in range(5):
128         val = in_data[-1] * difference
129         in_data.append(val)
130         #print(int(in_data[-1]), end = ' ')
131     return True
132
133
134
135
136 "ITERATIVE FUNCTIONS"
137 def find_iterative_diff(data_list):
138     "Finds the pattern that results in the same
    thing by iteratively skipping certain elements in
    the list"
139     "returns two variables, one a boolean and the
    other a step(which is the important part"
140     "Works for addition differences or
    Multiplication differences"
141
142     siz = len(data_list)
143     steps = siz//2 +1
144     i = 2
```

```
145     temp_pattern = []
146 
147     while i < steps:
148         temp_pattern = [element for pos,element in
    enumerate(data_list) if pos % i == 0 ]
149 
150         if is_same(temp_pattern):
151             #print("temp pattern:")
152             #print(temp_pattern)
153             return True, i
154         i+=1
155 
156     return False,i
157 
158 def check_iterative(data_list, step):
159     "Checks that data_list does in fact have a
    pattern with step amount of values"
160     "Returns True if it does, returns false if it
    does not"
161     "Works for linear differences or multiplier
    differences"
162 
163     temp_pattern = []
164     for i in range(step):
165         step_part = [element for pos,element in
    enumerate(data_list) if (pos + i) % step  == 0]
166         temp_pattern.append(step_part)
167     for item in temp_pattern:
168         if is_same(item):
169             print(" ", end = '')
170 
171         else:
172             print(" ", end = ' ')
173             return False
174     return True
175 
176 def find_iterative_spot(data_list,step):
177     "finds what spot in the pattern the given list
    is"
178     "for example if the list is 2,4,2,4,2, it will
    return 4,2 because those are the next items"
```

```
179     "Works for linear differences or multiplication
    differences"
180
181     repeat = data_list[0:step]
182
183
184     new_list = data_list[len(data_list) - step: None
    ]
185
186     i=0
187     while repeat != new_list and i<20:
188         repeat.append(repeat[0])
189         repeat.pop(0)
190
191         i+=1
192     last_repeat = repeat
193
194     return(last_repeat)
195
196 def iterative_pattern_lin(upd_repeat, data_list):
197     "uses te upd_repeat list from
    find_iterative_spot to create the next 5 values in
    the in_data list"
198     #print("iterative pattern")
199     step = len(upd_repeat)
200
201
202     i = 0
203     while i < 5:
204         for t in range(step):
205             if i==5:
206                 return None
207             new_num = data_list[-1] + upd_repeat[t]
208             data_list.append(new_num)
209             i += 1
210
211
212         if i<5 and t == step-1:
213             t=0
214
215
```

```
216 def iterative_pattern_mult(upd_repeat, data_list):
217     "uses te upd_repeat list from
    find_iterative_spot to create the next 5 values in
    the in_data list(for multipliers)"
218     #print("iterative pattern")
219     step = len(upd_repeat)
220
221
222     i = 0
223     while i < 5:
224         for t in range(step):
225             if i == 5:
226                 return None
227             new_num = data_list[-1] * upd_repeat[t]
228             data_list.append(new_num)
229             i+=1
230         if i<5 and t == step-1:
231             t=0
232
233
234
235
236 "COMPLEX PATTERNS/FUNCTIONS"
237 def comp_pattern(in_data):
238     "finds complex pattern such as n = 2(n-1)+1
    using the functions above, and then creates"
239     "the proceding list and prints the result"
240     comp_dif1 = list_diff(in_data)
241     comp_dif2 = list_diff(comp_dif1)
242     comp_dif3 = list_diff(comp_dif2)
243     #print(comp_dif1, "is the first list diff")
244     #print(comp_dif2, "is the second list diff")
245     #print(comp_dif3, "is the third list diff")
246
247     comp_mult1 = list_multiplier(in_data)
248     comp_mult2 = list_multiplier(comp_mult1)
249     diff_mult1 = list_diff(comp_mult1)
250
251     mult_diff1 = list_multiplier(comp_dif1)
252     mult_diff2 = list_multiplier(comp_dif2)
253
```

```
254     #print(mult_diff1, "is the first mult diff of
    the first list diff")
255     #print(mult_diff2, "is the first mult diff of
    the second list diff")
256
257
258     if is_same(mult_diff1):
259         "This is the block that works for non
    iterative complex such as *2+1/3"
260         difference = mult_diff1[0]
261
262         for i in range(5):
263             "makes comp_dif1 the appropriate amount
    of numbers so that it can then be used " \
264             "to extend in_data"
265             val1 = comp_dif1[-1] * difference
266             comp_dif1.append(val1)
267             val2 = in_data[-1] + comp_dif1[-1]
268             in_data.append(val2)
269
270         return True
271     " NEXT
    PART_____________________________________________
    ________________________"
272
273     "Everything below is for the complex iterative
    pattern, code is a but confusing"
274
275
276     bool_lin, step_lin = find_iterative_diff(
    comp_dif1)
277     bool_mult, step_mult = find_iterative_diff(
    comp_mult1)
278
279     siz = len(in_data)
280     steps = siz // 2 + 1
281     i = 2
282     step = 0
283
284
285
```

```
286     while i < steps:
287         tr_list = []
288         for t in range(i):
289             temp_pattern = [element for pos, element
     in enumerate(in_data) if (pos -t) % i == 0]
290             "note from yesterday: if you look at the
     code it doesn;t make sense that sometimes the
    middle patter"
291             "has less elements than the pattern next
    . So the lin/mult lists must be being added/found in
     the wrong way"
292             "I believe the main problem to be that
    if "
293             print(temp_pattern, i, t, "i,t")
294             lin = list_diff(temp_pattern)
295             mult = list_multiplier(lin)
296             tr_list.append(mult)
297
298
299
300             tr_list.append(lin)
301         count = 0
302
303         for y in range(0,len(tr_list),2):
304             print(tr_list[y])
305             if len(tr_list[y]) == 1:
306                 count +=1
307             elif is_same(tr_list[y]):
308                 count+=1
309         #print(count, i)
310         if count == i:
311             step = i
312             #print(step, "STEP!!! and pattern found
    ")
313             i = steps
314             ovr_list = []
315             print("t,lin,mult:", t, mult, lin)
316             for items in tr_list:
317
318                 ovr_list.append(items)
319
```

```
320
321         i+=1
322
323     for items in ovr_list:
324         print(items, end = ' ')
325     start = ((len(in_data)) % step)
326     print("\n",len(in_data)-1, start)
327
328     #start *= 2
329     num = 0
330     boole = True
331
332     while num < 5:
333
334             if boole:
335                 t = start
336                 t*=2
337
338             else:
339                 t+=2
340             if t == step*2:
341                 t = 0
342
343             if num == 5:
344                 return None
345
346
347             new_num = ovr_list[t][-1] * ovr_list[t+1
    ][-1]
348
349
350             ovr_list[t+1].append(new_num)
351             print(t,new_num, ovr_list[t+1], in_data
    [-step])
352
353             in_data.append(new_num + in_data[-step])
354
355
356             num +=1
357
358
```

```
359
360             boole = False
361
362
363
364     return False
365
366
367
368
369
370 def main():
371     start = input("Would you like to play: y (yes
    ) , n (no) \n")
372     while start == 'y':
373
374
375         orig_in_data = get_data(data,base_data)
376         print(orig_in_data)
377         in_data = orig_in_data
378
379
380         lin_diff = list_diff(in_data)
381         "mult_diff = list_multiplier(in_data)"
382         iter_list_lin, step_lin =
    find_iterative_diff(lin_diff)
383
384         # print("iter_list:",iter_list, "   step:",
    step)
385
386         lin = True
387         if is_same(list_diff(in_data)):
388             lin_diff = list_diff(in_data)
389             lin_pattern(lin_diff, in_data)
390             print_result("Linear Pattern", in_data)
391             lin = False
392
393         if lin:
394             if in_data[0] == 0:
395                 del in_data[0]
396
```

```
397             mult_diff = list_multiplier(in_data)
398             iter_list_mult, step_mult =
    find_iterative_diff(mult_diff)
399
400             if is_same(list_multiplier(in_data)):
401                 mult_diff = list_multiplier(in_data)
402                 mult_pattern(mult_diff, in_data)
403                 print_result("Constant Multiplier",
    in_data)
404
405
406             elif check_iterative(lin_diff,step_lin):
407                 upd_repeat = find_iterative_spot(
    lin_diff,step_lin)
408                 print(lin_diff)
409                 print(upd_repeat)
410                 iterative_pattern_lin(upd_repeat,
    in_data)
411                 print_result("Iterative Linear",
    in_data)
412
413             elif check_iterative(list_multiplier(
    in_data),step_mult):
414                 mult_diff = list_multiplier(in_data)
415                 upd_repeat = find_iterative_spot(
    mult_diff,step_mult)
416                 iterative_pattern_mult(upd_repeat,
    in_data)
417                 print_result("Iterative Multiplier"
    , in_data)
418
419             else:
420                 comp_pattern(in_data)
421                 print_result("Complex Pattern",
    in_data)
422                 return None
423
424         start = input("Again?: y (yes) , n (no) \n")
425 main()
426
427
```

```
428
429
430
431
```