```
data = []
"^Leave Blank"
base_data = [2,-7,-25, 74, 380, -1141]

"""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]
exp:            [2,4,16,256,65536, 4294967296]
compiter(*3+1,-2): [2, 7, 5, 14, 12, 37, 35]
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]
"""

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) from 1-10 \n").split()

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

    if in_data[0] == 0 and in_data[1] == 0:
        for items in base_data:
```

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

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

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

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

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

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

```
253
254     if is_same(mult_diff1):
255         "This is the block that works for non
    iterative complex such as *2+1/3"
256         difference = mult_diff1[0]
257
258         for i in range(5):
259             "makes comp_dif1 the appropriate amount
    of numbers so that it can then be used " \
260             "to extend in_data"
261             val1 = comp_dif1[-1] * difference
262             comp_dif1.append(val1)
263             val2 = in_data[-1] + comp_dif1[-1]
264             in_data.append(val2)
265
266         return True
267     " NEXT
    PART_____________________________________________
    ________________________"
268
269     "Everything below is for the complex iterative
    pattern, code is a but confusing"
270
271
272     bool_lin, step_lin = find_iterative_diff(
    comp_dif1)
273     bool_mult, step_mult = find_iterative_diff(
    comp_mult1)
274
275     siz = len(in_data)
276     steps = siz // 2 + 1
277     i = 2
278     step = 0
279
280
281
282     while i < steps:
283
284         tr_list = []
285         for t in range(i):
286             temp_pattern = [element for pos, element
```

```
286  in enumerate(in_data) if (pos + t) % i == 0]
287              lin = list_diff(temp_pattern)
288              mult = list_multiplier(lin)
289              tr_list.append(mult)
290              #print("t,lin,mult:",t,mult,lin)
291
292
293              tr_list.append(lin)
294          count = 0
295
296          for y in range(0,len(tr_list),2):
297              if is_same(tr_list[y]) or len(tr_list[y
    ]) == 1:
298
299                  count +=1
300
301          #print(count, i)
302          if count == i:
303              step = i
304              print(step, "STEP!!! and pattern found")
305              i = steps
306              ovr_list = []
307              for items in tr_list:
308
309                  ovr_list.append(items)
310
311
312          i+=1
313
314
315      start = len(in_data) % step
316      if start == step:
317          start = 0
318      #start *= 2
319      num = 0
320      boole = True
321
322      while num < 5:
323
324              if boole:
325                  t = start
```

```
326                 t*=2
327             else:
328                 t+=2
329             if t == step*2:
330                 t = 0
331 
332             if num == 5:
333                 return None
334 
335 
336             new_num = ovr_list[t][-1] * ovr_list[t+1
    ][-1]
337 
338             ovr_list[t+1].append(new_num)
339 
340 
341             in_data.append(new_num + in_data[-step])
342 
343 
344             num +=1
345 
346 
347 
348             boole = False
349 
350 
351 
352     return False
353 
354 
355 
356 
357 
358 def main():
359     orig_in_data = get_data(data,base_data)
360     print(orig_in_data)
361     in_data = orig_in_data
362 
363 
364     lin_diff = list_diff(in_data)
365     "mult_diff = list_multiplier(in_data)"
```

```
366     iter_list_lin, step_lin = find_iterative_diff(
    lin_diff)
367
368     # print("iter_list:",iter_list, "   step:", step
    )
369
370
371     if is_same(list_diff(in_data)):
372         lin_diff = list_diff(in_data)
373         lin_pattern(lin_diff, in_data)
374         print_result("Linear Pattern", in_data)
375
376
377     elif in_data[0] == 0:
378         del in_data[0]
379
380     mult_diff = list_multiplier(in_data)
381     iter_list_mult, step_mult = find_iterative_diff(
    mult_diff)
382
383     if is_same(list_multiplier(in_data)):
384         mult_diff = list_multiplier(in_data)
385         mult_pattern(mult_diff, in_data)
386         print_result("Constant Multiplier", in_data)
387
388
389     elif check_iterative(lin_diff,step_lin):
390         upd_repeat = find_iterative_spot(lin_diff,
    step_lin)
391         print(lin_diff)
392         print(upd_repeat)
393         iterative_pattern_lin(upd_repeat, in_data)
394         print_result("Iterative Linear", in_data)
395
396     elif check_iterative(list_multiplier(in_data),
    step_mult):
397         mult_diff = list_multiplier(in_data)
398         upd_repeat = find_iterative_spot(mult_diff,
    step_mult)
399         iterative_pattern_mult(upd_repeat, in_data)
400         print_result("Iterative Multiplier", in_data
```

```
400 )
401
402     else:
403         comp_pattern(in_data)
404         print_result("Complex Pattern", in_data)
405
406
407 main()
408
409
410
411
412
413
```