2. Whether or not we score a style point depends on the state of both the arrows and the feet. We can only score a point if the arrow corresponds to a position of a foot already, otherwise we have to move a foot. Thus the only choices we have are of which feet to move. So we'll have to consider the optimal value for if we move the left foot or if we move the right foot. To know whether to award a point, we'll keep track of the foot states (which arrows they are on) in the two cases.

Let $\mathcal{A} = \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$ be the set of arrows. Let $a(n) = \texttt{Arrow[n]}$. Define $\texttt{OPT}^L(n)$ and $\texttt{OPT}^R(n)$ as the max score at step n. Let s_L^L, s_R^L be the states of left and right feet given that the left foot moved most recently; similarly use s_L^R, s_R^R for when the right foot moves. The superscript keeps track of which foot moved recently.

There are some tricky cases to track down.

If either s_L^L or s_R^L is $a(n)$, then we can get a style point if we've just moved the left foot. This would give score $1 + \texttt{OPT}^L(n-1)$ with all states the same. Otherwise, we update $\texttt{OPT}^L(n)$ depending on whether $\texttt{OPT}^L(n-1) \geq \texttt{OPT}^R(n-1)$. If $\texttt{OPT}^L$ is larger, we update s^L assuming we last moved the left foot, otherwise we update s^L assuming we last moved the right foot.

There's a symmetric situation for updating $\texttt{OPT}^R$.

Translation of symbols to code: $s_L^L = \texttt{sLL}, s_L^R = \texttt{sLR}$, etc. Second L/R is superscript.

Pseudocode:

```
a = Arrow[n]

# update OPTL
if sLL == a or sRL == a                         # possible style point
  if (1 + OPTL[n-1]) >= OPTR[n-1]
    OPTL[n] = 1 + OPTL[n-1]                     # get point, don't move
    sLL_new = sLL; sRL_new = sRL
  else
    OPTL[n] = OPTR[n-1]                         # move left foot
    sLL_new = a; sRL_new = sRR
else                                            # have to move left foot
  if OPTL[n-1] >= OPTR[n-1]}
    OPTL[n] = OPTL[n-1]
    sLL_new = a; sRL_new = sRL
  else
    OPTL[n] = OPTR[n-1]
    sLL_new = a; sRL_new = sRR

# update OPTR
if sLR == a or sRR == a                         # possible style point
  if (1 + OPTR[n-1]) >= OPTL[n-1]
    OPTR[n] = 1 + OPTR[n-1]                     # get point, don't move
    sLR_new = sLR; sRR_new = sRR
  else
    OPTR[n] = OPTL[n-1]                         # move right foot
    sLR_new = sLL; sRR_new = a
else                                            # have to move right foot
  if OPTL[n-1] >= OPTR[n-1]}
    OPTR[n] = OPTL[n-1]
    sLR_new = sLL; sRR_new = a
  else
    OPTR[n] = OPTR[n-1]
    sLR_new = sLR; sRR_new = a
```

```
# update states
sLL = sLL_new; sRL = sRL_new; sLR = sLR_new; sRR = sRR_new
```

For initial conditions, you'll have

```
sLL = sLR = LeftArrow
sRL = sRR = RightArrow
OPTL[0] = OPTR[0] = 0
```

The full code would have the above wrapped in a for-loop from symbol 1 to n. You would output the maximum $\max(\mathtt{OPT}^L(n), \mathtt{OPT}^R(n))$.

To know the sequence of arrows, you could backtrack like so:

```
Sequence(OPTL, OPTR, Arrow)
  init Moves[1:n]                              # array of moves to make
  if OPTL[n] >= OPTR[n]
    last = 'L'
  else
    last = 'R'
  for i = n down to 1
    if last == 'L'
      if OPTL[n] == OPTL[n-1] + 1
        Moves[i] = "Vogue!"
      else
        Moves[i] = "Left foot " + Arrow[i]
        if OPTL[n] == OPTR[n-1]
          last = 'R'
    else
      if OPTR[n] == OPTR[n-1] + 1
        Moves[i] = "Vogue!"
      else
        Moves[i] = "Right foot " + Arrow[i]
        if OPTR[n] == OPTL[n-1]
          last = 'L'
```

Overall, the amount of work is $O(1)$ at each iteration, leading to $O(n)$ work overall.

Test case

Style points in red:

```
OPTL = [0, 0, 1, 1, 2, 2, 3, 4, 5]
OPTR = [0, 0, 1, 1, 2, 2, 2, 3, 4]
```

Step	a	sLL	sRL	sLR	sRR	Move
0	–	←	→	←	→	–
1	↑	↑	→	←	↑	Left foot ↑
2	↑	↑	→	←	↑	Vogue!
3	↓	↓	→	↑	↓	Left foot ↓
4	↓	↓	→	↑	↓	Vogue!
5	←	←	→	↓	←	Left foot ←
6	→	←	→	←	→	Vogue!
7	←	←	→	←	→	Vogue!
8	→	←	→	←	→	Vogue!

Stepping through `Sequence(OPTL, OPTR, Arrow)` results in:
`Moves = [Left foot ↑, Vogue!, Left foot ↓, Vogue!, Left foot ←, Vogue!  Vogue!  Vogue!]`