DYALOG

SimCorp

Workshop:
Advent of Code – day 1

Adám Brudzewsky
Richard Park
Rodrigo Girão Serrão

Day 1

- Array-oriented techniques
- Bingo and Strings

Day 2

- Reading and parsing data from files
- Mathematical insights for array-oriented programming

Problems

- 21.4 Bingo/NoBingo
- 15.5 Nice/Nicer Strings

Topics

- Array-oriented techniques: loops vs doing things "all at once"
- The Rank Operator and other neat tricks

Array-oriented techniques

Procedural programming: decompose the problem into smallest piece which can be expressed, then iterate up to whole data

Array programming: transform data into a form which can be handled "all at once" using known idiomatic constructs

Use the simplest, **flattest** array structure you can

Bingo

Numbers from a list are called one at a time

Mark the occurrence of each number on your board

A board wins when all numbers in a row or column are found

Bingo: First board to win

NoBingo: Last board to win

Bingo

Sub-problems

- Checking if a board is a winner
- Iterating over called numbers
- Index of the first/last winner

Bingo

Checking winners

```
      nums ← 21 18 9 16 8 6 11 0
      ]repr 2⍴⍉boards
(5 5⍴14 21 17 24 4 10 16 15 9 19 18 8 23 26 20 22 11 13 6 5 2 0 12 3 7)

      ⍉←b←nums∊¨3⍉boards
0 1 0 0 0
0 1 0 1 0
1 1 0 0 0
0 1 0 1 0
0 1 0 0 0
```

Bingo

Checking winners

Reduce with Bracket-Axis, Rank

Check one board, loop with each `F⍤⍤` and rank `F⍤⍥k`

On Classic:

```
ö←{α←⊢ ◇ α(αα⌷U2364ωω)ω}
```

Check all boards with rank or axis

Bingo

Checking winners

This function checks if there is a complete row or column in a Boolean matix:

```
Win ← ∨/∧/,∧⍉
```

Exercise: Apply the `Win` function to the entire collection of boards

- `⍵:` Boolean 3D array
- `←:` Integer singleton (scalar, 1-element vector etc.)

Using Each `Win¨` Enclose each matrix using rank `⊂⍤2`

Enclose each matrix using bracket-axis `⊂[2]`

Using Rank `Win⍤2`

Bingo

Checking winners

Exercise: Write the function `Winners` to return a Boolean vector of winners without looping over each board as before

`⍵:` Boolean 3D array

`←:` Boolean vector

Hint: NOT

```
Winners ← Win⍤2
Winners ← Win⍤⊂[2 3]
```

Try reduce with Axis `F/[a]`

Try Reduce with Rank `F≠⍤k`

Bingo

Approaches to iteration

- Loop over each number
- Loop over each board
- Write a "Wins" function or expression which applies to a matrix; loop over each board
- Write a "Wins" function or expression which applies to the whole board; find the index of the winning board(s)
- Check all numbers, find progressive wins with scan

Refactoring a solution

```
winner←nums B1 boards;i;j;called;win
boards←⊂⍤2⊢boards
:For i :In ⍳≢nums
    :For j :In ⍳≢boards
      called←(⊃boards[j])∊i↑nums
      win←∨/(∧/[1]called),∧/[2]called
        :If win
            winner←j
            :Return
        :EndIf
    :EndFor
:EndFor
```

Exercise:
Identify and describe the differences

```
winner←nums B2 boards;called;n
called←0⍨⍤¯1 boards   ⍝ 0⍴⍨⍴boards
:For n :In nums
    called∨←boards=n
    winner←⍸(∨/∧/,∧⌿⍤2) called
    →0⍴⍨0<≢winner
:EndFor
```

Bingo

Iterating through nums

```
1 2⊃⍳(∨/∧/,∧≠⍤2)∨⌿↑boards∘∈⍤nums
```


Bingo

Iterating through nums

Exercise: Simplify the expression

```
↑boards∘∈¨nums
```


Nice and Nicer Strings

Compute the number of strings (lines) in the input which conform to some rule set

- Implementing the rule set
- Application of the rule set

Nice Strings

- At least three vowels `'aeiou'`
- At least one letter twice in a row
- Does not contain any of `'ab' 'cd' 'pq' 'xy'`

Nice Strings

- At least three vowels `'aeiou'`

  ```
  3≤+/ω∈'aeiou'
  ```

- At least one letter twice in a row
- Does not contain any of `'ab' 'cd' 'pq' 'xy'`

Nice Strings

- At least three vowels `'aeiou'`

```
3≤+/ω∈'aeiou'
```

- At least one letter twice in a row

```
∨/(¯1↓ω)=1↓ω
∨/2=/ω
```

- Does not contain any of `'ab' 'cd' 'pq' 'xy'`

Nice Strings

Does not contain any of `'ab' 'cd' 'pq' 'xy'`

```
                              ∈
bad ← 'ab' 'cd' 'pq' 'xy'

∨/∈  ∈∘string¨bad

∨/   ∨/⍤∈∘string¨bad
```


Nice Strings

Does not contain any of `'ab' 'cd' 'pq' 'xy'`

```
      ⍳
```

Exercise: define `⍺∊⍵` in terms of `⍺⍳⍵`

Nice Strings

Does not contain any of `'ab' 'cd' 'pq' 'xy'`

```
⍳
```

Exercise: define `α∊ω` in terms of `α⍳ω`

```
{(≢ω)≥ω⍳α}
```

Nice Strings

Does not contain any of `'ab' 'cd' 'pq' 'xy'`

```
      ⍳
```

Exercise: create a non-nested 3D character array of pairs of characters (overlapping) from a text matrix

```
ω: Character matrix of shape  n m
←: Character array of shape   n (m-1) 2
```

Nice Strings

Does not contain any of `'ab' 'cd' 'pq' 'xy'`

```
⍳
```

Putting it together:

```
+/~(↑bad){∧/(≢⍺)<⍺⍳0 ¯1↓⍵⌿⍨[2.1]1⌽⍵}⍤1
```


Nice Strings
Toolkit

```
3≤+/ω∈'aeiou'
```

```
∨/2=/ω
```

```
∧/(≠bad)<bad⍳¯1(↓⍤1)ω,[2.5]1⌽ω
```

Nicer Strings

- A pair of letters, found twice without overlapping
- At least one letter repeats with exactly one letter between

Nicer Strings

A pair of letters, found twice (or more) without overlapping

```
{∨/2≤|-/↑,⍳∘.≡⍨2,/⍵}
⍵ ← 'abcxxxz'
⍵ ← 'abcxxxxz'
⍵ ← 'abcxxyxxz'
⍵ ← 'abcxxyxxxz'
```

Nicer Strings

A pair of letters, found twice without overlapping

```
{∨/2≤|-/↑,⍳∘.≡⍨2,/⍵}
⍵ ← ↑'abcxxyxxz' 'defxxxyzz' 'applelppa'
```


Nicer Strings

A pair of letters, found twice without overlapping

```
      (≢'(..).*\1'⍁S 3)¨↓in
ω ← ↑'abcxxyxxz' 'defxxxyzz' 'applelppa'
```


Nicer Strings

Array-element IDs

```
      a←'abcbde'
      ↑a(⍳⍨a)
a b c b d e
1 2 3 2 5 6

      a←'abc' 'cde' 'abc' 'def' 'efg'
      ↑a(⍳⍨a)
```

abc	cde	abc	def	efg
1	2	1	4	5

Nicer Strings
Array-element IDs

```
⌄/2≤|-/↑,⍳⍤⊢.≡⍨⌸2,/⍵

⌄/ 2≤          (⍳⊢/⍴⍵)(-⍤1)      0,⍳⍨⌸⍤1⊢2,/⍵

⌄/ 2≤ ⍘1(↓⍤1) (⍳⊢/⍴⍵)-⍤1 (⍳⍨⌸2) ⍵,[2.5]1⌽⍵
```


Nicer Strings

A pair of letters, found twice without overlapping

```
NicerString←{
      b←{((⊂⍨(0,⍳¯2+≢⍵)+⊂⍳2)[]⍨⊂⍵)⍷⍨⊂⍵}'**',⍵
      b/⍨←2≤+/⍨b
      b←∪b
      b←∨/~0 1 1 0∘(∨/⍷)⍨b
      b∧←⊃∨/(∨/⊢=2∘ϕ)'****',⍵
      b
}
```


Nicer Strings

A pair of letters, found twice without overlapping

```
NicerString←{
      b←{((⊂⍨(0,⍳¯2+≢⍵)+⊂⍳2)⌷⍨⊂⍵)∈⍨⊂⍵}'**',⍵
      b/⍨←2≤+/⍨b
      b←∪b
      b←∨/~0 1 1 0∘(∨/∈)⍨b
      b∧←⊃∨/(∨/⊢=2∘ϕ)'**',⍵
      b
}
```

Exercise:
Simplify this

Nicer Strings

A pair of letters, found twice without overlapping

```
RepeatWithoutOverlap←{
      p←(∪2,/⍵)⍷¨⊂⍵
      p/⍨←2≤+/¨p
      ∨/¨0 1 1 0∘(∨/⍷)¨0,¨p
}
```

Exercise:

This code is meant to check if character vector ⍵ contains a character pair that appears twice but not overlapped.

Can you find an argument for which this is true, but this function returns 0?

Nicer Strings

A pair of letters, found twice without overlapping

```
z←NicerString t;m;b;c
m←↓t
c←≢⊃m
b←{p←⍨/⍤2=/2,/⍵ ⋄ (∧/p)∨2≤+/0=p}⍤⍤m
b∧←{c≥≢∪2,/⍵}⍤⍤m
b∧←{∨/=/(↑(3,/⍵))[;1 3]}⍤⍤m
z←+/b
```


Nicer Strings

A pair of letters, found twice without overlapping

```
z←NicerString t;m;b;c
m←↓t
c←≠⊃m
b←{p←⍵≡¨2=/2,/⍵ ⋄ (∧/p)∨2≤+/0=p}¨m
b∧←{c≥≠∪2,/⍵}¨m
b∧←{∨/=/(↑(3,/⍵))[;1 3]}¨m
z←+/b
```

Exercise:

This code is meant to check if character vector `⍵` contains a character pair that appears twice but not overlapped.

Can you find a string for which this is **true** but the function returns 0?

Can you find a string for which this is **false** but the function returns 1?

Nicer Strings

A letter repeated with exactly one in between

```
'abcxyxfg'
```

Nicer Strings

A letter repeated with exactly one in between

```
'abcxyxfg'
```

Nicer Strings

A letter repeated with exactly one in between

```
abcxyxfg
abcxyxfg
```

Nicer Strings

A letter repeated with exactly one in between

a b c x y x f g

a b c x y x f g

Nicer Strings

A letter repeated with exactly one in between

```
abcxyxfg
   abcxyxfg
```

Nicer Strings

A letter repeated with exactly one in between

```
      abcxyxfg
         abcxyxfg
      {↑(2↓⍵)(¯2↓⍵)}'abcxyxfg'
cxyxfg
abcxyx
      {(2↓⍵)=(¯2↓⍵)}'abcxyxfg'
0 0 0 1 0 0
```

Why can we not use
`3=/⍵` ?

Nicer Strings

A letter repeated with exactly one in between

```
      abcxyxfg
         abcxyxfg
      {↑(2↓⍵)(¯2↓⍵)}'abcxyxfg'
cxyxfg
abcxyx
      {(2↓⍵)=(¯2↓⍵)}'abcxyxfg'
0 0 0 1 0 0
```

What could we use 3⍰⍵ ?

Nice and Nicer Strings

Solution toolkit

```
3≤+/⍵∈'aeiou'
```

```
∨/2=/⍵
```

```
∧/(≢bad)<bad⍳1(↓⍤1)⍵,[2.5]1⌽⍵
```

```
∨/2≤|-/↑,⍨∘.=⍨⍳⍨⍵
```

```
∨/3(⊣/=⊢/)⍵
```


Nice and Nicer Strings

Three high-level approaches

1) Solve for a single string, and apply on each string
2) Apply each condition to every string, then AND results
3) Filter input for each condition, and count remaining