07_Transforming_data

September 22, 2019

1 7. Transforming data

1.1 7.1 Types

1.1.1 7.1.1. Basic data types

- _WRITE IT DOWN FROM MEMORY

1.1.2 7.1.2. The type operator

- The non-atomic monadic function (type) can be applied to any entity in q to return its data type expressed as a short.
- It is a "feature" of q that the data type of an atom is **negative** whereas the type of a simple list is **positive**.
- Type operator: returns the type of atoms in number format with the character type at the end and putting a minus sign in front
- Casts that narrow: no data loss
- Casts that widen: data loss
  - real and float data types are rounded when cast to short, integer or long
- When char and string is transformed into a numeric type the underlying ascii code is returned
- String conversion
  - Converting a value to string: string (32;`life`everything;1b)
  - Parse value from string: $ operator
    - You can use any type of characters in a string symbol conversion
    - White spaces are trimmed from the beginning and end of the string, when converted to symbol
    - To parse a char or string numeric value to a numberic type, use the **upper case character notation** of specifying a type
    - Parsing a string function into a q function: value
- Specify type of an empty list: `float$() - In a shorter form: 0#0 or 0#0.0 or 0#`
- **There is no way in q to type nested empty lists!!**

[ ]:
```
0#`
```

```
[ ]: `$"fléskjdféj"
[ ]: string (32;`life`everything;1b)
[ ]: string `Life`the`Universe`and`Everything
[ ]: k_table:([k:`a`b`c] v:10 20 30)
[ ]: k_table
[ ]: s_table:([]k:`a`b`c;v:10 20 30)
[ ]: s_table
[ ]: type k_table
[ ]: k_table[`a]
[ ]: s_table[`k]
[ ]: `int$"23"
[ ]: `char$4
[ ]: `long$"\n"
[ ]: `int$-2.22
[ ]: ceiling 2.22
[ ]: test_date:2019.03.10D15:53:11.123456789
[ ]: `long$test_date
[ ]: `int$0wh
[ ]: `long$0wi
[ ]: "i"$"4"
```

1.2 7.5. Enumerations

- Functional version of enumerated types: casting of values to user-defined target domains
- Purposes of enumerations:
  - Description of arbitrary numbers

  - Type checking: only permissible values are supplied
  - Namespacing: reuse of same names in different domains
  - Normalizing data

1.2.1 7.5.2. Data normalisation

- Data normalization seeks to eliminate duplication, retaining only the minimum amount of data
  - Use case: storing stock ticker data
  - Advantages: simplify storage management, eliminates data duplication
- Syntax
  - Create distinct values from a list of repeated values: enums:distinct symbol_list
  - Enumeration syntax: `enums$symbol_list - where $ does the indexing operation (like` 'enums?symbol_list but creates an enum type)
    * the result is the same as symbol_list only as an enumeration type and with the indexing done

[1]:
```
u:`g`aapl`msft`ibm
v:1000000?u
k:u?v
```

[2]:
```
u
v
k
```

[2]:
```
`g`aapl`msft`ibm
```

[2]:
```
`g`g`msft`aapl`msft`aapl`msft`ibm`msft`aapl`g`ibm`aapl`msft`msft`aapl`g`aapl`..
```

[2]:
```
0 0 2 1 2 1 2 3 2 1 0 3 1 2 2 1 0 1 2 0 1 1 2 3 0 1 2 1 2 1 0 0 1 2 1 2 3 3 0..
```

[ ]:
```
u[k] / reconstruct v from u and k
```

[3]:
```
pr_enum:`u$v
```

[4]:
```
pr_enum
```

[4]:
```
`u$`g`g`msft`aapl`msft`aapl`msft`ibm`msft`aapl`g`ibm`aapl`msft`msft`aapl`g`aa..
```

- There are different types of enumerations:
  - 20h the first type

1.2.2 7.5.6.Updating an enumerated list

- When you update an element in the unique values, it will appear in the enumerations as well.
  - Do not do it, it is dangerous!!!

1.2.3 7.5.7. Dynamically appending an element to an enumeration domain

- Adding a new value to an enumeration is complicated. First you have to append it to the unique list
  - in two steps:
    - 1. symEnum,:`newSym
    - 2. enumList,:`newSym
  - in one step (to update it dynamically):
    - ev,:`sym?twtr (note the ? instead of the dollar sign)
    - using the ? operand checks if the symbol is already in the unique enum list. if not, it appends it
- To convert the enumerated list back to the original list, use the q interpreter (value): value enumeration

[91]:
```
value pr_enum
```

[91]:
```
`g`g`msft`aapl`msft`aapl`msft`ibm`msft`aapl`g`ibm`aapl`msft`msft`aapl`g`aapl`..
```