

Strings in Ruby

- Web site www.ruby-lang.org
- Was used in first implementation of twitter before moving to Scala
- Ranked number 8 by www.langpop.com after C, Java, php, JavaScript, C++, Shell, Python and more popular than objective-C, C#, ...

Ruby Strings

Strings are objects that belong to **String** class

```
>>> "\n\t\".length ↵
```

③

```
>>> "abc".public_methods ↵
```

```
... "*", +, <, <=, ... ==, ..., [], []=, ...,  
"downcase", "downcase!", ..., "insert", ...,  
"each_byte", ..., "replace", ..., "reverse", "reverse!", ...
```

More than 140 methods

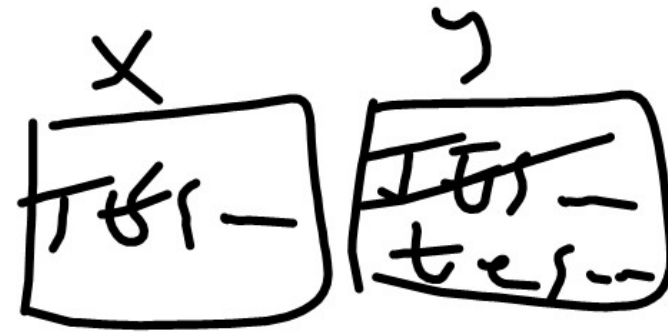
"reverse!" imperative method and changes the string

"reverse" applicative method (the string is not changed)

Ruby Strings are Mutable

Strings can be modified

```
>> x = "testing"
"testing"
```



```
>> y = x #x and y reference the same string
```

```
"testing"
```

```
>> x.upcase!
```

```
"TESTING"
```

```
>> y
```

```
"TESTING"
```

```
>> y = x.dup
```

```
"TESTING"
```

```
>> y.downcase!
```

```
"testing"
```

```
>> y
```

```
"testing"
```

```
>> x
```

```
"TESTING"
```

Ruby Strings Comparison

```
>> s1 = "apple"
"apple"
```

```
>> s2 = "testing"
"testing"
```

```
>> s1 == s2
false
```

```
>> s1 != s2
true
```

```
>> s1 < s2
```

```
true
```

```
>> s1 >= s2
```

```
false
```

```
>> s1 <=> s2
```

```
-1
```

```
>> s2 <=> s1
```

```
1
```

```
>> "x" <=> "x"
```

```
0
```

str ans

String Indexing and Modification

```
>> S = "abc"
```

"abc"

```
>> S[0]
```

97

```
>> S[-1]
```

99

```
>> S[100]
```

nil

```
>> S[0] = 65
```

65

```
>> S
```

"Abc"

```
>> S[1] = "tomi"
```

"tomi"

```
>> S
```

"Atomic"

```
>> S[0] = ?B
```

66

```
>> S
```

"Btomic"

'B'

Substrings

>> S = "replace"
 "replace"

>> S[2,3]
 "pla"

>> S[2,1]
 "p"

>> S[2..-1]
 "place"

>> S[10,10]

nil
 >> S[-4,3]
 "lac"

>> S[1,100]
 "eplace"

>> S[0,2] = ""
 ""

>> S
 "place"

>> S[3..-1] = "naria"
 "naria"

>> S
 "planaria"

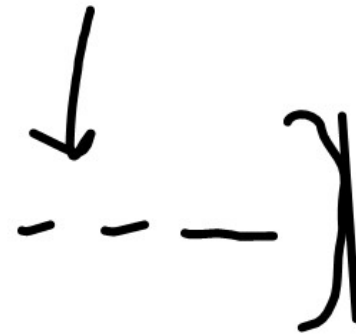
>> S["aria"] = "kton"
 "kton"

>> S
 "plankton"

Evaluation Inside Strings

>> x = 10 ↵

10

{  }

>> y = "twenty" ↵

"twenty"

>> s = "x = #{x}, y + y = #{y + y}" ↵

"x = 10, y + y = twentytwenty" *prints*

>> s = "string methods: #{'ab'.methods} ".length

896

String Append

```
>> s = "just"
```


"just"

```
>> s << "testing" << "this"
```

"justtestingthis"

Non Primitive Types: Enumeration

- All of the possible values, which are symbolic constants, are enumerated in the definition
- C# example



```
enum days {sat,sun,mon, tue, wed, fri};
days d1, d2;
d1=sat;
```

Enum Type Design Issues

- Is a literal constant allowed to appear in more than one type definition? If yes, how is the type of an occurrence of that literal in the program type checked?

☐ Yes in Ada

☐ No in C and C++

X = wed;

enum days {sat, sun, mon, tue, wed, fri};
enum week_end {fri, sat};

- Are enumerated values coerced to integer?

☐ Yes in C and C++: sat + 1 = 1

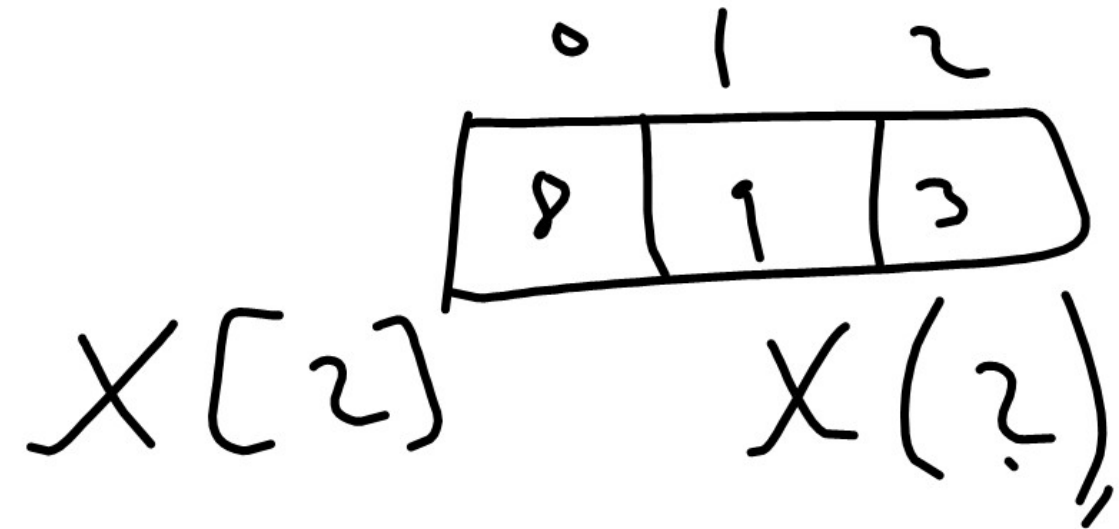
☐ No in C#

Array Type

A homogenous aggregate of data elements in which the individual element is identified by its position in the aggregate relative to the first element

Array Type Design Issues

Index Notation



- Two options for subscript

() Ada, PLI, Fortran

[] C-based languages

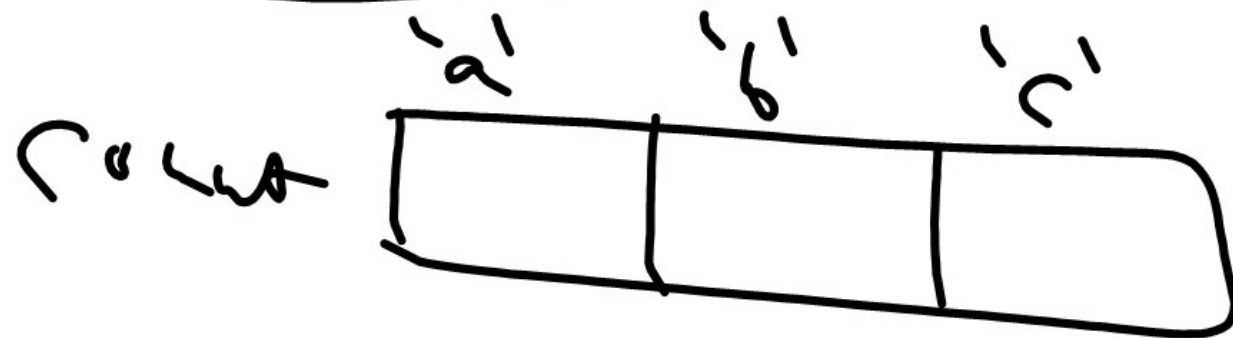
- () notation is motivated by the similarity between functions and arrays.

array name: element index \rightarrow element value

- sum=sum+B(I)** is less readable since B can be an array or a function
- It also requires more work by the compiler

Index Type

- C-based languages indeces are always integers
- Pascal and Ada indeces can be integers, characters, enumerated (ordinal types)



Run-time Checking of Index Range

- C, C++, Perl, Fortran do not specify range checking of subscripts

int X[100];

X[J]=...; // executes even if J>99

- Good for execution time
- Bad for reliability
- Java, Pascal, ML, C# do range checking

Array Initialization

- C and C++ allow initializing arrays at declaration
- Pascal does not

Array Operations

- What are the operations, if any, that operate on arrays
- Ada provides assignment, concatenation, equality and inequality operations
- Fortran 95: +, *, transpose, vector product

APL

- Provides many operations
 - ⊖ V: reverses the elements of vector V
 - ⊖ M: reverses the columns of matrix M
 - ⊖ M: reverses the rows of matrix M
 - ⊖ M: transposes matrix M
 - ÷ M: inverts matrix M
 - |
⊖ ...
- APL requires a special keyboard

M^{-1}

Jagged Arrays

- A rectangular array is a multidimensional array in which all of the rows have the same number of elements and all of the rows have the same number of elements and so on
- A jagged array is a multidimensional array in which rows have different size, this also applies to columns and other dimensions
- C# supports both kinds

C# Example

```
bool [ ][ ] X = new bool[2][ ];
```

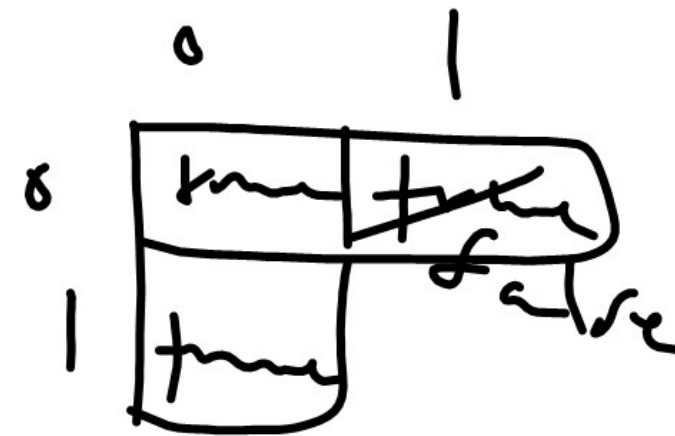
```
X[0] = new bool[2];
```

```
X[1] = new bool[1];
```

```
X[0][0] = true;
```

```
X[0][1] = false;
```

```
X[1][0] = true;
```



Slices

- An array slice is a substructure of the array
- Often languages that provide slices allow operating on the slices as a single unit
- Fortran 95 example

Integer vector(1:10), mat(1:3, 1:3), cube(1:3, 1:3, 1:4)

vector(3:6) is a 4-element array and a slice of vector

3 - 6

mat(1:3, 2) refers to the second column in mat

mat(3,1:3) refers to the third row of mat

mat 3

(cube(1:3,1:3,2)) could be assigned to mat

vector(2:10:2) is a one dimensional array consisting of second, fourth, sixth, eighth and tenth elements of vector

Associative Arrays

- Unordered collection of data elements that are indexed by an equal number of values called keys
- Both keys and data values are stored in the array
- In Perl associative arrays are called hashes. Every hash variable name begins with %
`%salaries=("ahmad"=>900, "jamal"=>400, "sami"=>500);`
`$salaries{"ahmad"}=900;`
- The size of the hash table is dynamic
`delete $salaries{"jamal"};`
- `If (exists $salaries{"sami"}) ...`
- *Each* operator is used to loop over hash elements

Array categories

Based on index range and storage allocation:

- Static arrays
- Fixed stack-dynamic arrays
- stack-dynamic arrays
- Stack-dynamic arrays
- Heap-dynamic arrays

Static Arrays

- Subscript ranges are statically bound
- Allocation before run-time
- Global arrays in C
- Efficient for execution time

Fixed stack-dynamic arrays

- Subscript ranges are statically bound
- Memory Allocation during run-time
- Stored on run-time stack
- Local array in a C function
- More memory efficient than static arrays

Stack-dynamic arrays

- Subscript ranges are dynamically bound
- Allocation during run-time when declaration is elaborated
- Once allocated, array size stays fixed

- Ada example

```
Get(x);
```

```
Declare
```

```
  A: array[1..x] of integer;
```

```
begin
```

```
  ...
```

```
end;
```

- Stored on run-time stack
- More flexible

Fixed heap-dynamic array

- Binding of subscript range and storage when the user requests them rather than at elaboration time
- Storage is allocated in heap
- Once allocated memory the size does not change
- Dynamic arrays in C++ using new and delete operations

Heap-dynamic arrays

- Allocation during run-time
- Size can change any number of times during execution
- C# example
 - ArrayList X=new ArrayList();**
 - X.Add(elem);** // extend array and store elem at end
- Perl example
 - @X=(2,5,18);**
 - push(@X, 15 , 20);**
 - Array becomes (2, 5, 18, 15, 20)

Array Implementation

- In a statement $X[J] = \dots$ the compiler replaces $X[J]$ with a code to compute its address
- Assuming a one-dimensional array index starts from 0:
$$\text{address}(X[i]) = \text{address}(X[0]) + i * \text{element_size}$$
- The $+$ and $*$ operations need to be performed during run-time. Less efficient than variables

Implementation of matrixes

- A matrix can be stored in row major, as in most languages, or as a column major, as in Fortran
- Given matrix $X[10][20]$ and assuming row and column numbers start from 0, and row-major ordering is used:

$$\text{Address}(X[i][j]) = \text{address}(X[0][0]) + (i * 20 + j) * \text{element_size}$$

- The 3 arithmetic operations are performed during execution
- This explains why defining a C function $f(\text{int } X[10][\])\{\dots\}$ is invalid