

§5.5-5.10: Arrays

Py ch8: Lists

29 Sep 2006
CMPT14x
Dr. Sean Ho
Trinity Western University

• *devo*

Review of last time (§5.1-5.3)

- Call stack, backtrace
- Abstract Data Types
 - Type hierarchy
- Enumerations
- Arrays

What's on for today (§5.5-5.10, Py ch8)

- Python lists vs. M2/C arrays
- Lists as function parameters
- Multidimensional arrays/lists
- Python-specific list operations
 - Membership (`in`)
 - Concatenate (`+`), repeat (`*`)
 - Delete (`del`), slice (`[s:e]`)
 - Aliasing vs. copying lists

Lists in Python

- Python doesn't have a built-in type exactly like arrays, but it does have **lists**:

```
nelliesWages = [0.0, 25.75, 0.0, 0.0, 0.0]
nelliesWages[1]           # returns 25.75
```

- Under the covers, Python often **implements** lists using arrays, but lists are more **powerful**:
 - Can change **length** dynamically
 - Can store items of different **type**
 - Can **delete/insert** items mid-list
- For now, we'll treat Python lists as **arrays**

Using lists

- We know one way to generate a list: `range()`

```
range(10)          # returns [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Or create directly in square **brackets**:

```
myApples = ["Fuji", "Gala", "Red Delicious"]
```

- We can **iterate** through a list:

```
for idx in range(len(myApples)):
    print "I like", myApples[idx], "apples!"
```

- Even easier:

```
for apple in myApples:
    print "I like", apple, "apples!"
```

Arrays as parameters

```
def average(array):  
    """Return the average of the array's values.  
    pre: array should have scalar values (float, int) and not  
        be empty.  
    """  
    sum = 0  
    for elt in array:  
        sum += elt  
    return sum / len(array)
```

```
myList = range(9)  
print average(myList)           # prints 4
```

- What happens when we pass an empty array? An atomic value?

Type-checking array parameters

- Since Python is **dynamically**-typed, the function definition doesn't specify what **type** the parameter is, or even that it needs to be a **list**
 - Easy way out: state expected type in **precondition**
 - Or do **type checking** in the function:

```
if type(array) != list:  
    print "Need to pass this function a list!"  
    return
```
 - May also want to check for **empty** lists:

```
if len(array) == 0:
```
- **for**, **len()**, etc. don't work on **atomic** types

Array parameters in M2/C/etc.

- In **statically**-typed languages like M2, C, etc., the procedure declaration needs to specify that the parameter is an **array**, and the **type** of its elements:

- **M2:**

```
PROCEDURE Average(myList: ARRAY of REAL) : REAL;
```

- **C:**

```
float average(float* myList, unsigned int len) {
```

- In M2, **HIGH(myList)** gets the **length**
- In C, length is **unknown** (pass in separately)

Multidimensional arrays

- **Multidimensional** arrays are simply arrays of arrays:

```
myMatrix = [ [0.0, 0.1, 0.2, 0.3],  
             [1.0, 1.1, 1.2, 1.3],  
             [2.0, 2.1, 2.2, 2.3] ]
```

- **Accessing**:

```
myMatrix[1][2] = 1.2
```

- **Row-major** convention:



Iterating through multidim arrays

```
def matrix_average(matrix):  
    """Return the average value from the 2D matrix.  
    Pre: matrix must be a non-empty 2D array of scalar  
    values."""  
    sum = 0  
    num_entries = 0  
    for row in range(len(matrix)):  
        for col in range(len(matrix[row])):  
            sum += matrix[row][col]  
            num_entries += len(matrix[row])  
    return sum / num_entries
```

- What if rows are not all equal length?

List operations (Python specific)

```
myApples = [ "Fuji", "Gala", "Golden Delicious" ]
```

- Test for list membership:

```
if "Fuji" in myApples: # True
```

- Concatenate:

```
['a', 'b', 'c'] + ['d', 'e']
```

- Repeat:

```
['a', 'b', 'c'] * 2
```

- Modify list entries (mutability):

```
myApples[1] = "Braeburn"
```

- Convert a string to a list of characters:

```
list("Hello World!") # ['H', 'e', 'l', 'l', 'o', ...]
```

More list operations

- **Delete** an element of the list:

```
del myApples[1]      # [ "Fuji", "Golden Delicious" ]
```

- List **slice** (start:end):

```
myApples[0:1]      # [ "Fuji", "Gala" ]
```

- Assignment is **aliasing**:

```
yourApples = myApples      # points to same array
```

- Use a whole-list slice to **copy** a list:

```
yourApples = myApples[:]
```

```
#[:] is shorthand for [0:-1] or [0:len(myApples)-1]
```

Summary of today (§5.5-5.10, Py ch8)

- Python lists vs. M2/C arrays
- Lists as function parameters
- Multidimensional arrays/lists
- Python-specific list operations
 - Membership (`in`)
 - Concatenate (`+`), repeat (`*`)
 - Delete (`del`), slice (`[s:e]`)
 - Aliasing vs. copying lists

TODO

- Lab 03 due next MTW:
 - M2 ch4 # (24 or 27 or 37)
- Quiz ch4-5 next Mon
- Read through M2 ch5 and Py ch8

- Midterm ch1-5 next week Fri 6Oct
 - See [sample midterm \(with solutions\)](#) under Fall 2005 CMPT14x homepage