C Arrays and Python Lists

21 Oct 2010
CMPT140
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

What's on for today

- Python lists vs. M2/C arrays
  - Indexing, iterating through lists
  - Lists as function parameters
  - Multidimensional arrays/lists
  - List operations

Python type hierarchy (partial)

- Numbers
  - Integral (int, bool): 5, 500000, True
  - Real (float) (only double-precision): 5.0
  - Complex numbers (complex): 5+2j
- Sequences
  - Immutable (str, tuple): “Hello”, (2, 5.0, 'hi')
  - Mutable (lists): [2, 5.0, 'hi']
- Mappings (dict): {"apple": 5, "orange": 8}
- Custom classes (user-defined types): class Student
- *Complete list at docs.python.org*

C Arrays

- Most languages (C, M2, Java, etc.) have arrays:
  - C: `float myWages[5] = {0., 25.75, 0., 0., 0.};`
  - M2: `myWages: ARRAY [0..4] OF REAL;`
- Compound data type, sequential storage
  - Fixed length: must declare length (e.g., 5)
  - Uniform type: same type for all elements
  - Static type: can't change type of elements
- Indexing: `myWages[2] = 15.85;`

Python Lists

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

```
nelliesWages = [0., 25.75, 0., 15., 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

Using lists

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

```
list(range(10))     → [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!")
```

Lists as parameters

```
def average(vec):
    """Average the values in the vector.
    pre: vec should have numeric values
      and not be empty."""
    sum = 0
    for elt in vec:
        sum += elt
    return sum / len(vec)
```

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

- What happens if we pass an empty list? What if we call average(5)?

Type-checking list 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 type in precondition
  - Or do type checking in the function:

```
if type(vec) != type([]):
    print("Need to pass this function a list!")
    return
```

  - May also want to check for empty lists:

```
if len(vec) == 0:
```

- for, len(), etc. don't work on atomic types

average() function, revised

```
def average( vec ):
    """Find average value in vector.
       Pre: vec is a list of numbers.
       Post: returns average value."""
    if type(vec) != type([]) or len(vec) == 0:
        print("average: need a list of nums!")
        return None
    tot = 0
    for elt in vec:
        sum += elt
    return sum / len(vec)
```

shortcut Boolean operator

Array parameters in M2/C/etc.

- In statically-typed languages (M2, C, etc.), the function definition must declare the parameter to be 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
  - Equivalent to Python's len()
- In C, length is unknown (pass in separately)

Multidimensional arrays

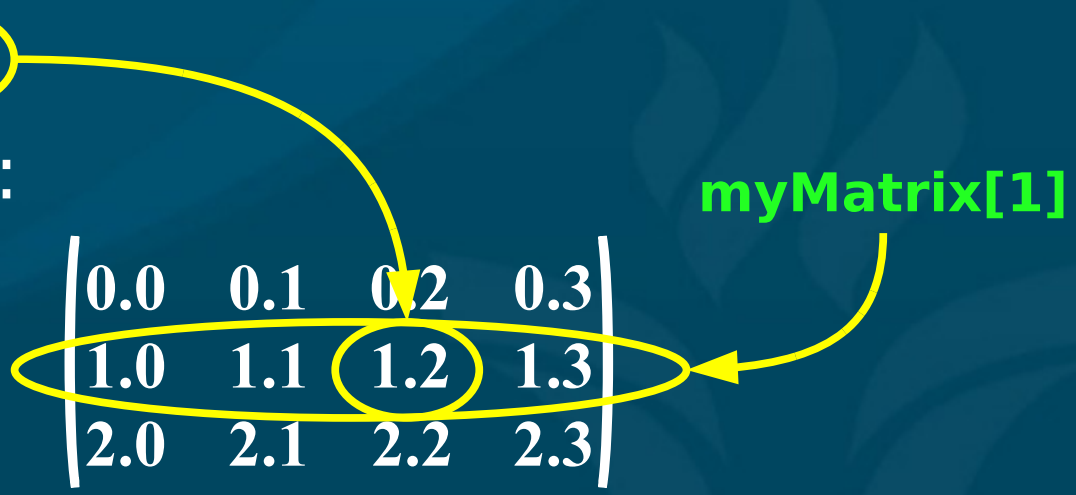
- Multidimensional arrays are simply arrays of arrays:

```
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 in multidim arrays

```
def matrix_average(matrix):
    """Average values from a 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)

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

- Test for list membership:

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

- Concatenate:

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

- Repeat:

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

- Modify list entries (lists are mutable):

```
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" ]
```

- Lists are mutable, so assignment is aliasing:

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

  - Changes to myApples are reflected in yourApples

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

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

  - Shorthand for 0:len(myApples)