CS177 Python Programming

Recitation 8:
Lists Comprehension,
Other Sequence of Data Types

What will we see today?

- Lists Comprehension
- Tuples
- 1-D Arrays
- 2-D Arrays (Matrices)
- Dictionaries

List comprehension

Let’s start with some basics. Suppose you want to create an empty list

```
>>> my_list = []
>>> my_list[0] = 10

Traceback (most recent call last):
File "<pyshell#10>", line 1, in <module>
my_list[0] = 10
IndexError: list assignment index out of
range
>>>
```

This breaks because my_list is an empty list so you can’t set an element of an empty list

List comprehension

So, to add to an empty list you have to append the element

```
>>> my_list = []
>>> my_list.append(10)
>>> print(my_list[0])
10
```

The first element appended will be referenced with the index 0. Similarly, the second element will be referenced with the index 1 and son on.

```
>>> my_list = []
>>> my_list.append(10)
>>> my_list.append(20)
>>> print(my_list)
[10, 20]
```

What is the index of this element

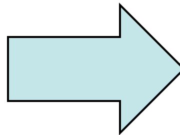
What is the index of this element

List comprehension

Suppose now you want to create a list of squares

A first approach:

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> print(squares)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```



A second approach:

>>>squares = [x**2 **for** x **in** range(10)]

Note that this x is the one in the for loop

List comprehension

A list comprehension consists of brackets containing an expression followed by a **for** clause, then zero or more **for** or **if** clauses. The result will be a new list resulting from evaluating the expression in the context of the **for** and **if** clauses which follow it. Let’s see another example.

List comprehension

Example: This list comprehension combines the elements of two lists if they are not equal:

>>> [(x, y) **for** x **in** [1,2,3] **for** y **in** [3,1,4] **if** x != y]

[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

Mandatory **for** clause

Optional **for** clause

Optional **if** clause

List comprehension

More examples:

```
>>> vec = [-4, -2, 0, 2, 4]
>>> # create a new list with the values doubled
>>> [x*2 for x in vec]
[-8, -4, 0, 4, 8]
>>> # filter the list to exclude negative numbers
>>> [x for x in vec if x >= 0]
[0, 2, 4]
>>> # apply a function to all the elements
>>> [abs(x) for x in vec]
[4, 2, 0, 2, 4]
```

List comprehension

An example containing complex expressions:

```
>>> from math import pi
>>> [str(round(pi, i)) for i in range(1, 6)]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

Tuples

- So far we have two sequence of data types: strings and lists
- As Python evolves, new sequence of data types are added
- A **tuple** is a new sequence of data types with its own characteristics

Tuples

- A **Tuple** consists of a number of values separated by comma
- Tuples are immutable
- Tuples can contain mutable objects as elements
- Elements can be of any type
- Tuples can be nested
- Let’s see some examples

Tuples

Properties

Some examples and properties

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

```
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

Also notice the **round parenthesis**: it different from lists

Tuples

Creating tuples with 0 or 1 item

```
>>> empty = ()
>>> singleton = 'hello',    # <-- note trailing comma
>>> len(empty)
0
>>> len(singleton)
1
>>> singleton
('hello',)
```

1-D Arrays

- I other programming languages, an array is a collection of items of the same data type
- Python does not include such a structure
- In Python arrays are implemented using lists with elements of the same data type

1-D Arrays

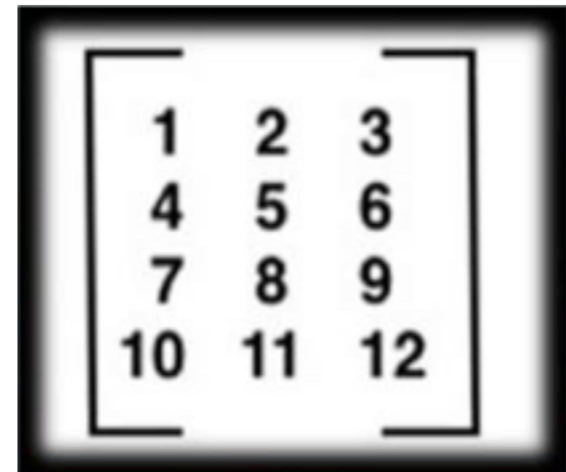
You can implement list comprehension to create and initialize an array

```
>>> my_array = [0 for i in range(10)]
>>> print (my_array)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

This basically just creates an element 0 as many times the for loop runs

2-D Arrays: Matrices

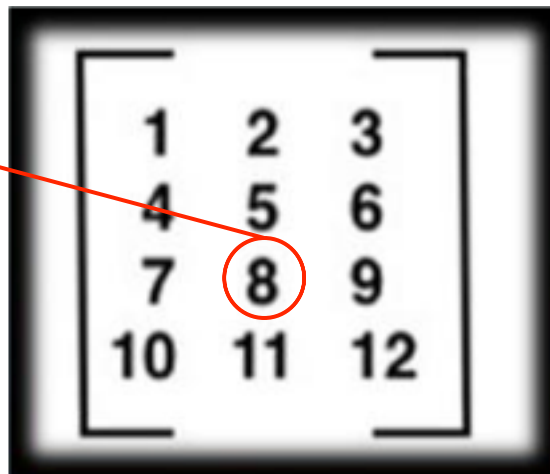
- Matrices in mathematics are arrays of numbers or variables arranged in both rows and columns.
- Each number or variable contained within the matrix can be identified by its position in the row and column.

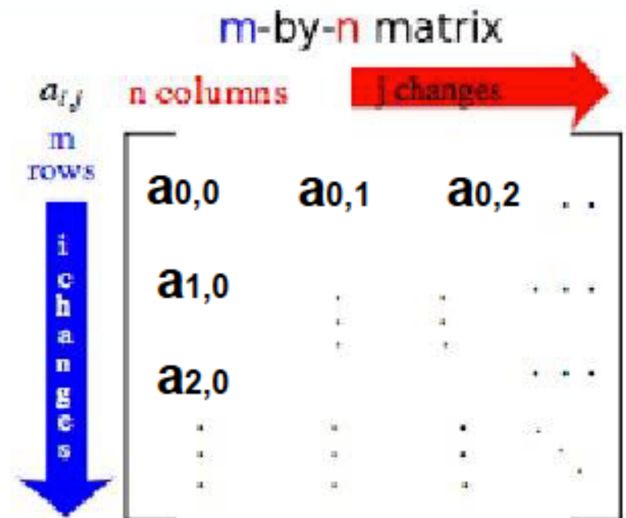


2-D Arrays: Matrices

- Each element of the matrix has a unique position determined by an index i and at index j.
- In the matrix below: the number 1, is defined to be in position 0,0 (located in row index 0 and column index 0)

What are the indexes of number 8 in this matrix?

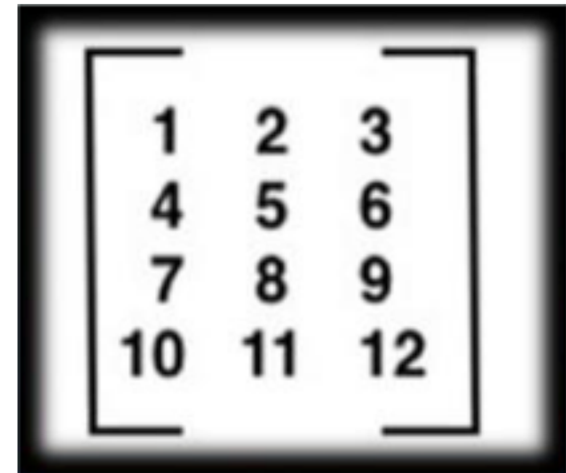




2-D Arrays: Matrices

Matrices as 1-D arrays are encoded in Python using lists

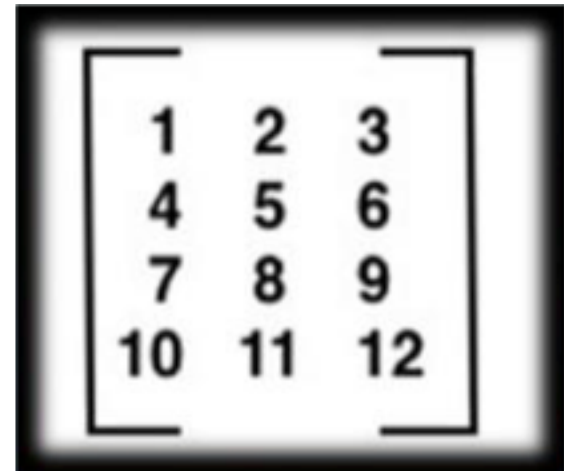
```
>>> myMatrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
>>> numRows = len(myMatrix)
>>> numComumns = len(myMatrix[0])
>>> print(numRows)
4
>>> print(numColumns)
3
```



2-D Arrays: Matrices

Indexing in the matrix:

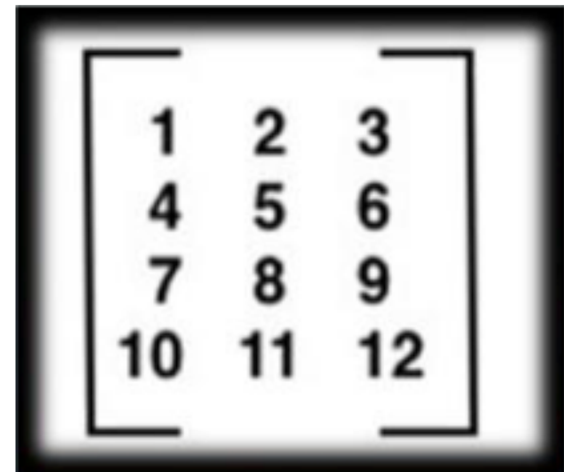
```
>>> myMatrix = [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
>>> print(myMatrix[0][0])
?
>>> print(myMatrix[3][2])
?
>>> print(myMatrix[1][2])
?
>>> print(myMatrix[2][0])
?
>>> print(myMatrix[2][3])
?
```



2-D Arrays: Matrices

Indexing in the matrix:

```
>>> myMatrix = [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
>>> print(myMatrix[0][0])
1
>>> print(myMatrix[3][2])
12
>>> print(myMatrix[1][2])
6
>>> print(myMatrix[2][0])
7
>>> print(myMatrix[2][3])
Traceback (most recent call last):
File "<pyshell#5>", line 1, in <module>
print(myMatrix[2][3])
IndexError: list index out of range
```



2-D Arrays: Matrices

Creating a Matrix: We are going to create a 5 ×4 matrix populated with 0s.

```
>>> #columns creates a list of length 4
>>> columns = 4
>>> rows = 5
>>> #The for loop duplicates that list rownumber of times
>>> x = [[0]*columns for i in range(rows)]
>>> print(x)
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Remember:

```
>>> [0]*4
>>> [0, 0, 0, 0]
```

2-D Arrays: Matrices

Traversing a Matrix: When traversing a matrix you are going to need nested loops to iterate through each row and column.

```
myMatrix = [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
for i in range(0,len(myMatrix)):
    if(i != 0):
        print()
    for j in range(0,len(myMatrix[0])):
        print(str(myMatrix[i][j])+'\t',end = "")
```

Remember:
len(myMatrix) = # rows
lenglen(myMatrix[i]) = # columns

What would be the output?

```
>>>
1	2	3
4	5	6
7	8	9
10	11	12
>>>
```

2-D Arrays: Matrices

Exercise 1:

```
rows = len(myMatrix)
rows = len(myMatrix[0])
>>> for i in range(rows):
        print ('Row : ' + str(i) )
        for j in range(columns):
           print(myMatrix[i][j])
```

What would be the output?

```
Row: 0
1
2
3
Row: 1
4
5
6
Row: 2
7
8
9
Row: 3
10
11
12
```

2-D Arrays: Matrices

Exercise 2: Fill the ‘?’

```
>>> myMatrix2 = [[10,20,33],[40,50,65],[12,2,79]]
>>> exm1 = myMatrix2[1][2]
>>> exm2 = myMatrix2[2][0]
>>> exm3 = myMatrix2[0][3]
>>> print(exm1)
?
>>> print(exm2)
?
>>> print(exm3)
?
```

2-D Arrays: Matrices

Exercise 2: Fill the '?'

```
>>> myMatrix2 = [[10,20,33],[40,50,65],[12,2,79]]
>>> exm1 = myMatrix2[1][2]
>>> exm2 = myMatrix2[2][0]
>>> exm3 = myMatrix2[0][3]
>>> print(exm1)
65
>>> print(exm2)
12
>>> print(exm3)
ERROR!
```

2-D Arrays: Matrices

Exercise 3: Write code for creating a matrix with 5 rows and 4columns. Then make the value 1 only if the row index is equal to the column index.

2-D Arrays: Matrices

Exercise 3: Write code for creating a matrix with 5 rows and 4columns. Then make the value 1 only if the row index is equal to the column index.

```
rows = 5
columns = 4
Matrix = [[0]*columns for i in range(rows)]
for i in range(rows):
    for j in range(columns):
        if (i == j):
            Matrix[i][j] = 1
```

2-D Arrays: Matrices

Exercise 4: What should be the output of the following code?

```
rows = 5
columns = 5
M = [[0]*columns for i in range(rows)]
for i in range(rows):
    if(i!= 0):
        print()
    for j in range(columns):
        if i+j == 4:
            M[i][j] = 1
            print(str(M[i][j])+'\t',end = "")
```

2-D Arrays: Matrices

Exercise 4: What should be the output of the following code?

```
>>>
0       0       0       0
0       0       0       1
0       0       1       0
0       1       0       0
1       0       0       0
```

2-D Arrays: Matrices

Matrix Multiplication:

$$\begin{bmatrix} 1 & 0 \\ -3 & 2 \end{bmatrix} \cdot \begin{bmatrix} -1 & 4 \\ 3 & 5 \end{bmatrix}$$

1st Row Times 1st Column

$$(1)(-1)+(0)(3) \quad -1+0 \quad -1$$

1st Row Times 2nd Column

$$(1)(4)+(0)(5) \quad 4+0 \quad 4$$

2nd Row Times 1st Column

$$(-3)(-1)+(2)(3) \quad 3+6 \quad 9$$

2nd Row Times 2nd Column

$$(-3)(4)+(2)(5) \quad -12+10 \quad -2$$

Final Answer: $\begin{bmatrix} -1 & 4 \\ 9 & -2 \end{bmatrix}$

2-D Arrays: Matrices

Matrix Multiplication

```
A = [[1,0][-3,2]]
x = [[-1,4][3,5]]
rows = 2
columns = 2
M = [[0]*columns for iin range(rows)]
#iterate through rows of A
For i in range(rows):
    #iterate through columns of x
    for j in range(columns):
        #iterate through rows of x
        for k in range(rows):
            M[i][j] += A[i][k]*x[k][j]
print(M)
```

$$\begin{bmatrix} 1 & 0 \\ -3 & 2 \end{bmatrix} \cdot \begin{bmatrix} -1 & 4 \\ 3 & 5 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 \\ -3 & 2 \end{bmatrix} \cdot \begin{bmatrix} -1 & 4 \\ 3 & 5 \end{bmatrix}$$

1st Row Times 1st Column: $(1)(-1)+(0)(3)$, $-1+0$, -1

1st Row Times 2nd Column: $(1)(4)+(0)(5)$, $4+0$, 4

2nd Row Times 1st Column: $(-3)(-1)+(2)(3)$, $3+6$, 9

2nd Row Times 2nd Column: $(-3)(4)+(2)(5)$, $-12+10$, -2

$$\begin{bmatrix} 1 & 0 \\ -3 & 2 \end{bmatrix} \cdot \begin{bmatrix} -1 & 4 \\ 3 & 5 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 \\ -3 & 2 \end{bmatrix} \cdot \begin{bmatrix} -1 & 4 \\ 3 & 5 \end{bmatrix}$$

Final Answer: $\begin{bmatrix} -1 & 4 \\ 9 & -2 \end{bmatrix}$

Dictionaries

- Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by *keys*
- Keys must be of any immutable type
- Strings and numbers can always be keys
- Tuples can be used as keys if they contain only strings, numbers, or tuples
- If a tuple contains any mutable object either directly or indirectly, it cannot be used as a key
- Lists cannot be used as keys

Dictionaries

- It is best to think of a dictionary as an unordered set of *key: value* pairs, with the requirement that the keys are unique (within one dictionary).
- A pair of braces creates an empty dictionary: {}
- Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary;
- This is also the way dictionaries are written on output.

Dictionaries

- The main operations on a dictionary are storing a value with some key and extracting the value given the key.
- It is also possible to delete a key:value pair with **del**.
- If you store using a key that is already in use, the old value associated with that key is forgotten.
- It is an error to extract a value using a non-existent key.

Dictionaries

Examples

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
```

Creating a dictionary

Adding a new pair key:value

Reading the value of an existing pair

Deleting an existing parir

Getting the keys of the dictionary

Validating the existence of a key:value pair in the dictionary

Dictionaries

The dict() constructorbuilds dictionaries directly from sequences of key-value pairs:

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

Dictionaries

In addition, dict comprehensions can be used to create dictionaries from arbitrary key and value expressions:

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

Thank you!