CS65: Introduction to Computer Science

Sequence: List Slicing
Sequence: Tuple
Sequence: List of List
Sequence: String Methods

Md Alimoor Reza
Assistant Professor of Computer Science

Today's Plan

- Sequence
  - String
  - List
    - Useful functions on list (len, max, min, sum)
    - Useful methods of list (sort, index, reverse)
    - Testing membership within a list using in operator
    - Remove an item from list (remove, pop, del)
    - List slicing
  - Tuple
- List of Lists (Nested List)
- String Methods

Exercise #1

Given a list of numbers, can you think of a way to remove all of the numbers from the list that are == 0?

Exercise #2

A programmer wants to write a program that allows the user to enter a rain amount and then remove that if it is in the list.

However, when they test this code, some of the tests cause the program to crash.

Explain the problem and write the code that will fix it (Hint: you might need to add an if statement).

```
rainfall_amounts = [0.0, 0.3, 0.71, 0.0, 0.32, 1.1, 0.4]
val_to_remove = float(input("Enter a value to remove: "))
rainfall_amounts.remove(val_to_remove)
print(rainfall_amounts)
```

Today's Plan

- Sequence
  - String
  - List
    - Useful functions on list (len, max, min, sum)
    - Useful methods of list (sort, index, reverse)
    - Testing membership within a list using in operator
    - Remove an item from list (remove, pop, del)
    - List slicing
  - Tuple
- List of Lists (Nested List)
- String Methods

List Slicing

- Slice is a span of items that are taken from a list (or any sequence)
  - Span is a list containing copies of elements from **start** up to, but not including, **end**
  - Format: *`list_variable_name`***`[start : end]`**
    - **`start :`** starting index — if not specified 0 is used
    - **`end :`** end index — if not specified ***len****(list)* is used

List Slicing

- Format: *list_variable_name*[**start : end**]
  - **start :** starting index — if not specified 0 is used
  - **end :** end index — if not specified ***len****(list)* is used

```
num_list = [10, 11, 12, 13, 14, 15]

print("num_list ", num_list)
print("num_list[0] ", num_list[0])
print("num_list[:1] ", num_list[:1])
print("num_list[1:4] ", num_list[1:4])
print("num_list[1:] ", num_list[1:])
```

```
>>> %Run lec12.py

num_list  [10, 11, 12, 13, 14, 15]
num_list[0]  10
num_list[:1]  [10]
num_list[1:4]  [11, 12, 13]
num_list[1:]  [11, 12, 13, 14, 15]
```

Extra: Consider “parallel” lists

Parallel lists are two or more list variables that contain contain related data – the elements of each list are associated with each other based on their position in the list.

ParallelLists.py

```
# parallel lists

my_list = ["apples", "grapes", "bananas"]
quantity = [ 100, 23, 84]

# print out the value and the quantity
for i in range(len(my_list)):
    print(my_list[i], quantity[i])
```

```
apples 100
grapes 23
bananas 84
```

Extra: Lists and random values

What will this code do?

RandomLists.py

```
2  # random lists
3
4  import random
5
6  random.seed(0)
7  my_list = []
8  for i in range(10):
9      my_list.append(random.randint(1,100))
10
11 print(my_list)
12
```

Today's Plan

- Sequence
  - String
  - List
    - Useful functions on list (len, max, min, sum)
    - Useful methods of list (sort, index, reverse)
    - Testing membership within a list using in operator
    - Remove an item from list (remove, pop, del)
    - List slicing
  - Tuple
- List of Lists (Nested List)
- String Methods

Tuple: another type of a sequence

- We **cannot change/modify** items in a tuple after its creation
  - This property is called **immutability**

- Items are accessed by index (similar to List or String)

Sequence	Example	Syntax	Accessing
String	my_str = “My name is walle”	within enclosing quotation marks, ie, “ ” or ‘ ’	my_str[0] my_str[1]
List	my_list = [1, 2, “a”, “abs”]	within enclosing brackets [ ] and separated by commas	my_list[0] my_list[1]
Tuple	my_tuple = (1, 2, “a”, “abs”)	Within enclosing parenthesis ( ) and separated by commas	my_tuple[0] my_tuple[1]

Mutable Property of List

```
# ---------- mutability of List ------------

my_list = [1, 2, "a", "abs"]

for i in range(len(my_list)):
    print(my_list[i])


# trying to update a location with a new value

my_list[1] = 3

print("modified value of list ", my_list[1])
```

```
>>> %Run lec14demo.py

  1
  2
  a
  abs
  modified value of list  3
```

Immutable Property of Tuple

```
# ---------- immutability of Tuple ------------
my_tuple = (1, 2, "a", "abs")

for i in range(len(my_tuple)):
    print(my_tuple[i])


# trying to update a location with a new value

my_tuple[1] = 3

print("modified value of tuple ", my_tuple[1])
```

```
1
2
a
abs
Traceback (most recent call last):
  File "/Users/reza/Class_and_Research/drake_teaching/CS65/c
>
    my_tuple[1] = 3
TypeError: 'tuple' object does not support item assignment
```

Tuple

- **Tuple examples**

```
# tuple examples

tup1 = ()

tup2 = (1,)          # one-tuple needs a comma in Python

tup3 = ("Georg Cantor", "Bertrand Russell", "Kurt Godel")

tup4 = (True, False, True, False)

tup5 = ([1, 2, 3], [4, 5, 6])

tup6 = ((1, 2, 3), (4, 5, 6))
```

- Exercise: Try the examples above in Thonny. Find the items you can modify and which the ones you cannot

Today's Plan

- Sequence
  - String
  - List
    - Useful functions on list (len, max, min, sum)
    - Useful methods of list (sort, index, reverse)
    - Testing membership within a list using in operator
    - Remove an item from list (remove, pop, del)
    - List slicing
  - Tuple
- List of Lists (Nested List)
- String Methods

Fill in the blank

An if statement inside of another if statement is called a NESTED if statement

A loop inside of another loop is called a NESTED loop

A list in which one or more elements is another list (*a list inside of a list*) is called a _____ list

Fill in the blank

An if statement inside of another if statement is called a NESTED if statement

A loop inside of another loop is called a NESTED loop

A list in which one or more elements is another list (*a list inside of a list*) is called a NESTED list

Also known as a two-dimensional list

Nested Lists

A nested list is a ‘list of lists’

- One or more list inside of a list
- Can be thought of as a matrix

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \rightarrow \begin{matrix} [[1, & 2, & 3, & 4], \\ [5, & 6, & 7, & 8], \\ [9, & 10, & 11, & 12]] \end{matrix}$$

```
my_matrix = [ [1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12] ]
```

```
my_matrix = [ [1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12] ]
```

Two-Dimensional Lists

Accessing an element in a two-dimensional list requires two sets of brackets

- The first one is the index into the row
- The second one is the index into the columns.

```
value = my_matrix[1][0]
```

```
my_matrix = [ [1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12] ]
```

Two-Dimensional Lists

Accessing an element in a two-dimensional list requires two sets of brackets

- The first one is the index into the row
- The second one is the index into the columns.

```
value = my_matrix[1][0]
```

row index 1

```
my_matrix = [ [1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12] ]
```

Two-Dimensional Lists

Accessing an element in a two-dimensional list requires two sets of brackets

- The first one is the index into the row
- The second one is the index into the columns.

```
value = my_matrix[1][0]
```

row index 1
col index 0

```
my_matrix = [ [1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12] ]
```

Notice that each integer is enclosed in its own set of brackets

Accessing nested list Elements

```
scores = [[91,92,93,94],[95,96,97,98],
[99,100,101,102]]
```

The **`scores`** variable holds the nested list of integers

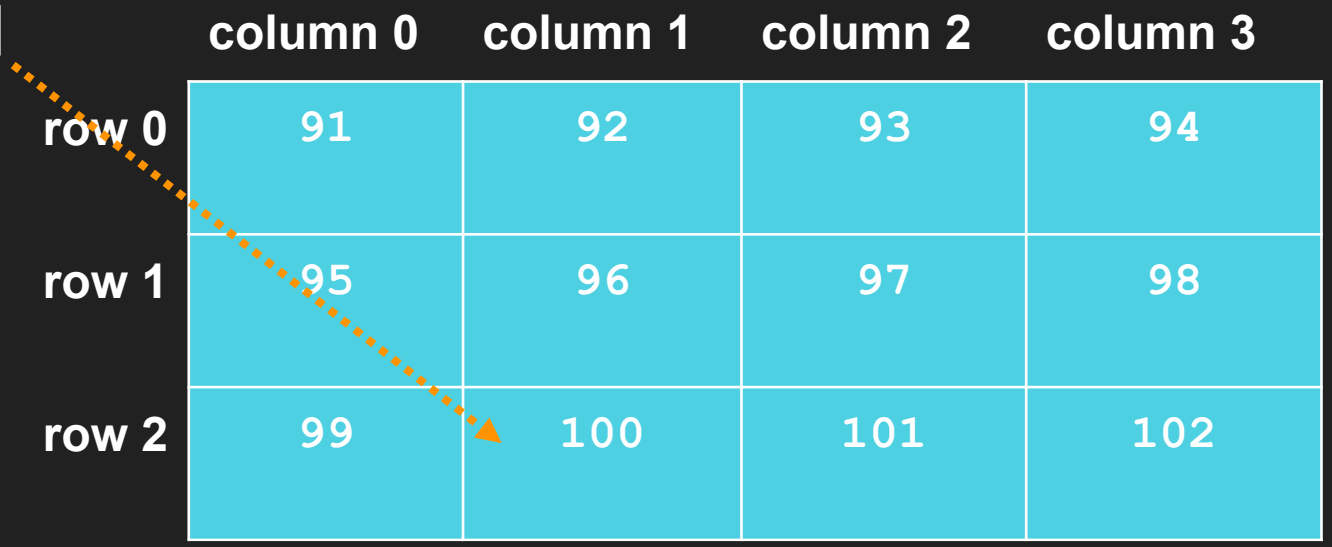
	column 0	column 1	column 2	column 3
row 0	`scores[0][0]`	`scores[0][1]`	`scores[0][2]`	`scores[0][3]`
row 1	`scores[1][0]`	`scores[1][1]`	`scores[1][2]`	`scores[1][3]`
row 2	`scores[2][0]`	`scores[2][1]`	`scores[2][2]`	`scores[2][3]`

Accessing nested list Elements

```
scores = [[91,92,93,94],[95,96,97,98],
[99,100,101,102]]
```

Accessing one of the elements in a two-dimensional list requires the use of both subscripts [row][col]:

```
scores[2][1]
```

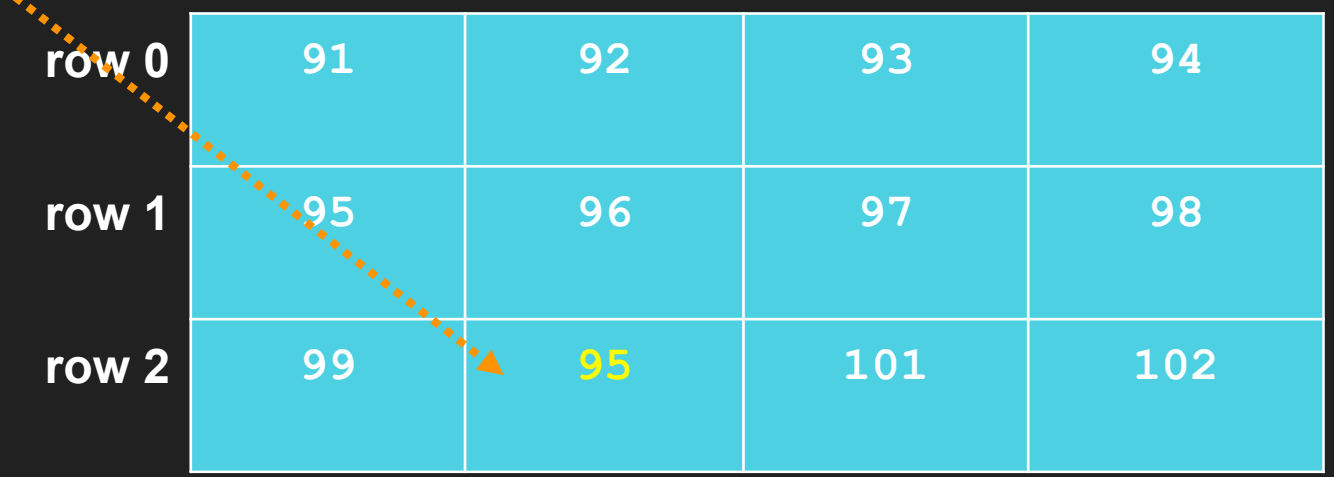


Accessing nested list Elements

```
scores = [[91,92,93,94],[95,96,97,98],
[99,100,101,102]]
```

Accessing one of the elements in a two-dimensional list requires the use of both subscripts [row][col]:

```
scores[2][1] = 95
```



Len of 2D lists

Recall that the **`len`** function gives you the length (number of elements) in a list.

```
my_matrix = [ [1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12] ]
```

What is **`len(my_matrix)`**?

What is **`len(my_matrix[0])`**?

Len of 2D lists

Recall that the len function gives you the length (number of elements) in a list.

```
my_matrix = [ [1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12] ]
```

What is **`len(my_matrix)`**?

3

What is **`len(my_matrix[0])`**?

4

Len of 2D lists

Recall that the len function gives you the length (number of elements) in a list.

```
# list of lists

num_list = [ [1, 2, 3], [10, 20, 30] ]

list_size_outer  = len(num_list)
print("Size of the outer list ", list_size_outer)

list_size_inner0 = len(num_list[0])
print("Size of the first inner-list ", list_size_inner0)

list_size_inner1 = len(num_list[1])
print("Size of the second inner-list ", list_size_inner1)
```

```
>>> %Run lec12.py

  Size of the outer list  2
  Size of the first inner-list  3
  Size of the second inner-list  3
```

Example – process all elements of a 1D list

What is the total?

```
stock_earnings = [24.5, 19.0,-22.25, -19.75]
```

Can you do it using a loop?

NestedLists1.py

```
# start with 1D list
stock_earnings = [24.5, 19.0,-22.25, -19.75]

total = 0
for row in range(len(stock_earnings)):
    total += stock_earnings[row]

print("total is", total)
```

Example – process all elements of a 2D list

What is the total?

Can you do it using a loop?

```
# a 2D list
stock_earnings = [[24.5, 19.0, -22.25, -19.75],
                  [1.25, -11.5, 102.75, -100.75],
                  [44.5, 41.0, -122.5, -9.25]]
```

Recap: Nested **for** loops

- Putting one loop inside another
  - The first loop is called the outer loop
  - The second loop is called the inner loop

- Here is simpler version:

```
for i in range(3):
    for j in range(3):
        print("i: ", i, "j: ", j)
```

List of Lists and Nested Index Loops

```
# list of lists

num_list = [ [1, 2, 3], [10, 20, 30] ]
```

- **Step 1:** Create an index for each dimension
- **Step 2:** Nest loops
- **Step 3:** Access each element using indexing

List of Lists and Nested Index Loops

```
# list of lists

num_list = [ [1, 2, 3], [10, 20, 30] ]

list_size_i = len(num_list)

for i in range(list_size_i):

    inner_list = num_list[i]      # get one inner list

    list_size_j = len(inner_list) # find the size of the inner list

    for j in range(list_size_j):

        print("num_list[", i, "][", j, "]", num_list[i][j])
```

```
>>> %Run lec12.py

  num_list[ 0 ][ 0 ] 1
  num_list[ 0 ][ 1 ] 2
  num_list[ 0 ][ 2 ] 3
  num_list[ 1 ][ 0 ] 10
  num_list[ 1 ][ 1 ] 20
  num_list[ 1 ][ 2 ] 30
```

List of Lists and Nested Index Loops

- Code simplification
  - Inserting the *len()* inside the range function

```
num_list = [ [1, 2, 3], [10, 20, 30] ]

for i in range(len(num_list)):

    for j in range(len(num_list[i])):

        print("num_list[", i, "][", j, "]", num_list[i][j])
```

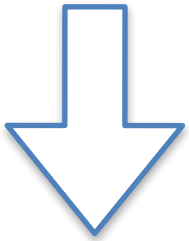
```
>>> %Run lec12.py

  num_list[ 0 ][ 0 ] 1
  num_list[ 0 ][ 1 ] 2
  num_list[ 0 ][ 2 ] 3
  num_list[ 1 ][ 0 ] 10
  num_list[ 1 ][ 1 ] 20
  num_list[ 1 ][ 2 ] 30
```

Exercise 3

- Find the summation of all the numbers in a list of lists

```
num_list = [ [1, 2, 3], [10, 20, 30] ]
```



```
>>> %Run lec12.py

  Total is  66
```

```
num_list = [ [1, 2, 3], [10, 20, 30] ]

for i in range(len(num_list)):

    for j in range(len(num_list[i])):

        print("num_list[", i, "][", j, "]", num_list[i][j])
```

```
>>> %Run lec12.py

  num_list[ 0 ][ 0 ] 1
  num_list[ 0 ][ 1 ] 2
  num_list[ 0 ][ 2 ] 3
  num_list[ 1 ][ 0 ] 10
  num_list[ 1 ][ 1 ] 20
  num_list[ 1 ][ 2 ] 30
```

Example – process all elements of a 2D list

What is the total?

Can you do it using a loop?

```
# a 2D list
stock_earnings = [[24.5, 19.0, -22.25, -19.75],
                  [1.25, -11.5, 102.75, -100.75],
                  [44.5, 41.0, -122.5, -9.25]]
```

```
total = 0
for row in range(len(stock_earnings)): # the number of rows
    for col in range(len(stock_earnings[0])): # the number of cols
        total += stock_earnings[row][col]
print(total)
```

Exercise #4

Given a 2D list, print out the number of elements that are < 0?

```
# a 2D list
stock_earnings = [[24.5, 19.0, -22.25, -19.75],
                  [1.25, -11.5, 102.75, -100.75],
                  [44.5, 41.0, -122.5, -9.25]]
```

Today's Plan

- Sequence
  - String
  - List
    - Useful functions on list (len, max, min, sum)
    - Useful methods of list (sort, index, reverse)
    - Testing membership within a list using in operator
    - Remove an item from list (remove, pop, del)
    - List slicing
  - Tuple
- List of Lists (Nested List)
- String Methods

Useful string methods

- **Syntax:** string_expression.method_name($argm_1$, $argm_2$, $argm_n$)

method	purpose	returned value
s.upper() s.lower()	converts letters to upper or lower case	modified copy of s
s.startswith(svar[,start[,stop]]) s.endswith(svar[,start[,stop]])	is svar a prefix/suffix of s?	Boolean value
s.join(iterable)	concatenates strings from iterable, with copies of string s inbetween them	string result of all those concatenations/ interspersings
s.split(sep)	get list of strings obtained by splitting s into parts at each occurrence of sep	list of strings from between occurrences of sep
s.replace(old, new[,count])	replace all (or count) occurrences of old str with new str.	string with replacements performed

Useful String Methods: .upper()

- **Syntax:** string_expression**.**method_name()

  my_str**.upper** ( )

  my_str**.lower**( )

```
#-------------------------------------------------------------
#                    .upper() or .lower()
# ------------------------------------------------------------
my_str       = "drake university"
my_str_upper = my_str.upper()

print("upper(): ", my_str_upper)
print("lower(): ", "HELLO".lower())
```

```
upper():  DRAKE UNIVERSITY
lower():  hello
```

Useful String Methods: .split()

- **Syntax:** string_expression**.**method_name(*argm1*)

  my_str**.split**(*separator*)

```
# ------------------------------------------------------------
#                    .split() method
# ------------------------------------------------------------
my_str         = "computer,science,department"
splitted_items = my_str.split(',')
for val in splitted_items:
    print("splitted strings are: ", val)
```

```
splitted strings are:  computer
splitted strings are:  science
splitted strings are:  department
```

Useful String Methods: .replace()

- **Syntax:** string_expression**.**method_name($argm_1$, $argm_2$)

  my_str**.replace** (old_{str} , new_{str})

```
#-------------------------------------------------------------
#                    .replace()
# ------------------------------------------------------------
my_str      = "A brown quick fox jump over the lazy dog"
new_str     = my_str.replace("lazy", "tired")

print("old : ", my_str)
print("new : ", new_str)
```

```
old :  A brown quick fox jump over the lazy dog
new :  A brown quick fox jump over the tired dog
```

Useful String Methods: .replace()

- **Syntax:** string_expression**.**method_name($argm_1$, $argm_2$, $argm_3$)

my_str**.replace**(old_{str} , new_{str} , *how_many_times*)

```
my_str = "A brown quick fox jump over the lazy lazy lazy dog"
new_str = my_str.replace("lazy", "tired", 2)

print("old : ", my_str)
print("new : ", new_str)
```

```
old :  A brown quick fox jump over the lazy lazy lazy dog
new :  A brown quick fox jump over the tired tired lazy dog
```

Useful String Methods: .find()

- **Syntax:** string_expression**.**method_name($argm_1$)

  my_str**.find** ($str_{you\_are\_looking\_for}$)

```
#-------------------------------------------------------------
#                    .find()
# ------------------------------------------------------------
my_str      = "A brown quick fox jump over the lazy dog"
position    = my_str.find("lazy")

print("string : ", my_str)
print("lazy at position {}", position)
```

```
string :  A brown quick fox jump over the lazy dog
lazy at position {} 32
```

Work on Lab #5 which is due on Saturday, October 25th

Text Reading

https://analytics.drake.edu/~reza/teaching/cs65_fall25/readings/Chapter9.html

Summary

Content quiz#3 will take place next week. Topics are as follows:

- `Sequence: Strings`
- `Sequence: List`
- `Sequence: Tuple`