CS65: Introduction to Computer Science

Sequence: String Methods
Introduction to Functions

Md Alimoor Reza
Assistant Professor of Computer Science

Last Lecture

- 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)

Review: Removing Items: `.remove()` Method

- When you know the *value* of the item you want to delete from the list, use the ***remove*** method. Trying to remove a value that isn't in the list will result in an error.

```
num_list = [10, -2, -2, -2, 20, -4, 30]

num_list.remove(-2)
num_list.remove(-2)
num_list.remove(-2)
print("After removing all -2s num_list is ", num_list)
```

```
>>> %Run lec12.py

  After removing all -2s num_list is  [10, 20, -4, 30]
```

- **Heads up!** If the item is not in the list, you will get an error

Review: Removing Items: `.pop()` Method

- When you don't want the *value* of the item you want to delete from the list, use can use index/position of the item using *pop* method. This will only delete the item located at that index.

```
# deleting an item from the list using .pop() method

num_list = [10, -1, -2, -3, 20, -4, 30]

print("Initial num_list is ", num_list)

num_list.pop(0)

print("After removing item at index 0 num_list is ", num_list)

num_list.pop(1)

print("After removing item at index 1 num_list is ", num_list)
```

```
>>> %Run lec12.py

  Initial num_list is  [10, -1, -2, -3, 20, -4, 30]
  After removing item at index 0 num_list is  [-1, -2, -3, 20, -4, 30]
  After removing item at index 1 num_list is  [-1, -3, 20, -4, 30]
```

Review: Removing Items: `del` Keyword

- You can also use index/position of the item using ***`del`*** keyword. This will only delete the item located at that index.

```
# deleting an item from the list

num_list = [10, -1, -2, -3, 20, -4, 30]

print("Initial num_list is ", num_list)

del num_list[0]

print("After removing item at index 0 num_list is ", num_list)

del num_list[1]

print("After removing item at index 1 num_list is ", num_list)
```

```
>>> %Run lec12.py

  Initial num_list is  [10, -1, -2, -3, 20, -4, 30]
  After removing item at index 0 num_list is  [-1, -2, -3, 20, -4, 30]
  After removing item at index 1 num_list is  [-1, -3, 20, -4, 30]
```

Review: 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]
```

Review: 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]

Review: Immutable Property of Tuple

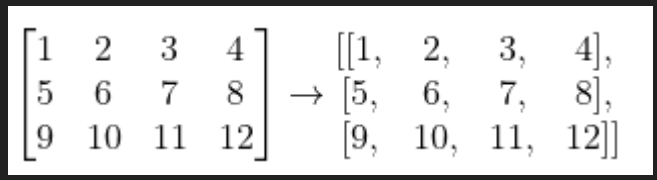
```
# ---------- immutability of Tuple -------------
my_tuple = (1, 2, "a", "abs")

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
```

Review: 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




```
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] ]
```

Review: 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] ]
```

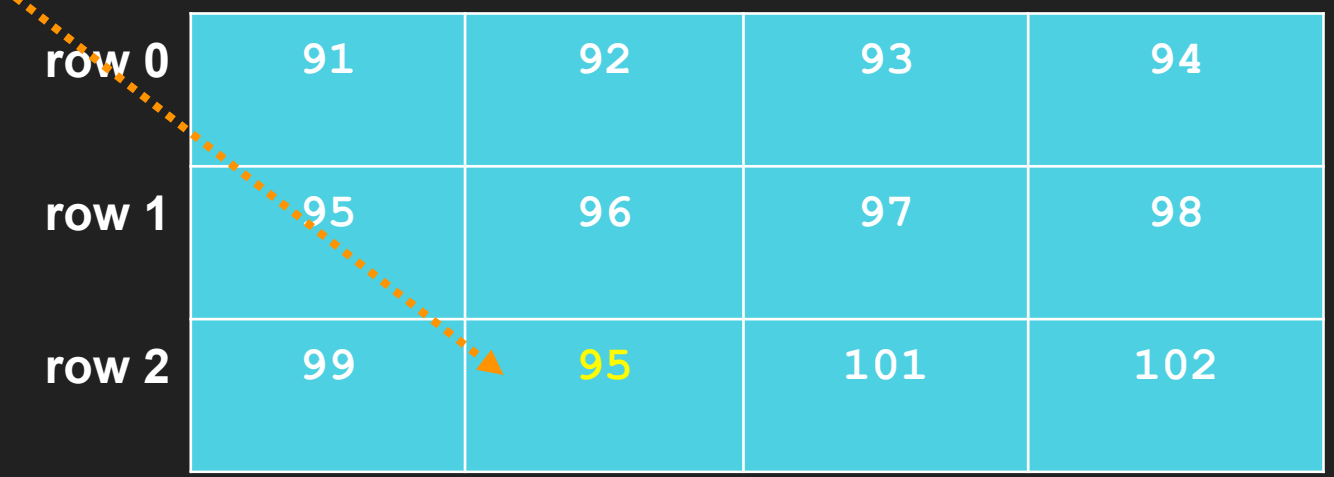
Notice that each integer is enclosed in its own set of brackets

Review: 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`



Review: 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
```

Review: 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
```

Review: Exercise #3

Given a 2D list, find the summation of all the numbers in a list of lists.

```
my_list2d = [ [1, 2, -3, 4, -1], [10, 20, -30, 40, -1], [60, 70, -80, 90, -1]]
```

Review: Exercise #4

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

```
my_list2d = [ [1, 2, -3, 4, -1], [10, 20, -30, 40, -1], [60, 70, -80, 90, -1]]
```

Today's Plan

- Sequence
  - String
  - List
  - 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
```

Today's Plan

- Sequence
  - String
  - List
  - Tuple
- List of Lists (Nested List)
- String Methods
- Introduction to Functions

Modules

One of the things that makes Python such a great language is that there are many functions you can use that aren't built in.

They're written by other programmers and made available to you.

In order to use them, you need an `import` statement.

Something you import this way is called a module.

A module is some pre-defined code that can be available to a program through the keyword `import`

Modules

One of the things that makes Python such a great language is that there are many functions you can use that aren't built in.

They're written by other programmers and made available to you.

In order to use them, you need an **`import`** statement.

Something you import this way is called a module.

A module is some pre-defined code that can be available to a program through the keyword **`import`**

```
4 from graphics import *
```

Module Examples

```
import random
print( random.randint(1,100) )
```

In random.randint(1,100),

random is the name of the module and **randint** is the name of the *function* we want to use from that module.

1 and 100 are *arguments* we pass to the function that tell it what range of numbers we want our random number to be in.

Module Examples

```
import random
print( random.randint(1,100) )
```

In random.randint(1,100),

random is the name of the module and **randint** is the name of the *function* we want to use from that module.

1 and 100 are ***arguments*** we pass to the function that tell it what range of numbers we want our random number to be in.

```
from graphics import *
```

Notice the use of *

In this case, we don't need to prepend calls to the graphics library with `graphics.function_name()`

Abstraction

Big idea in computer science: **Abstraction**

Abstraction is simplifying and separating the details of how something works to make programming (and understanding) easier

Illustrative example: You don't need to know how to farm to be a chef: they work at different levels of abstraction in food production.

Abstraction in Computer Science

In computer science, you build bigger, more complex things by using the building blocks others (or you) have provided.

Abstraction examples we've already seen:

We've already used several built-in ***functions*** in this class:

- `print("Hello")`
- `val = int(input("please enter a number")`
- `my_list = [2,3,4]`
- `num_elements = len(my_list)`
- `max_num = max(my_list)`

These are "built-in" functions in Python – they are part of the language and you don't need anything else to use them

Calling Functions

- `print("Hello")`
- `val = int(input("please enter a number")`
- `my_list = [2,3,4]`
- `num_elements = len(my_list)`
- `max_num = max(my_list)`

Why are these awesome?

- We don't have to think about the code that makes these work in order to use them (abstraction)
- The code these programmers wrote is reusable
  - the same code they wrote once gets used in many many programs by many different programmers

Defining your own functions

Why should we write our own functions?

Easier problem-solving - break big tasks into small ones

Better organization - easier to find/change the code you're looking for

Easier testing/debugging - you can test each function by itself before integrating into the bigger program

- Testing an individual function by itself is called unit testing

Reusable - you can write code that will be useful many times in this program and possibly later ones

Defining your own functions

The syntax for defining your own function includes the following

- keyword **`def`** (short for define)
- a name/identifier that you pick (same rules as variable names)
- parentheses ( ) (later, we'll put things inside these parentheses)
- colon :
- *indented* block of code

Demo Functions1.py

This function won't run or do anything until you call it. Call it just like any other function you've used.

```
# the following defines a function
def message():
    print("Hello CS 65!")
    print("I'm in a function!")
```

Demo Functions1.py

This function won't run or do anything until you call it. Call it just like any other function you've used.

```
# the following defines a function
def message():
    print("Hello CS 65!")
    print("I'm in a function!")

message() # call the function
```

Functions1.py

Exercise#1

In Thonny, open a new file.

Create a function named **`my_function`**

When **`my_function`** is called, it should print out “THIS IS AWESOME!!”

Use a loop; call the function so it prints out “THIS IS AWESOME!!” 10 times

What is the minimum number of lines of code you can write this program?