ENGINEERING
TEXAS A&M UNIVERSITY

Week 6

Lists and List Operations

Learning Objectives

ENGINEERING
TEXAS A&M UNIVERSITY

Lists of Data and List Operations

- Create and identify lists in Python
- Assign or reference a single element within a list
- Find the length of a list using the len() command
- Create and index lists of lists
- Use the list operations of append, insert, and concatenation
- Slice a list to print, delete, or replace portions of a list or create a new list
- Use the shortcut methods of slicing to refer to the start or end of a list (e.g., [:b], [a:], [:])
- Slice a string to print or define a new string
- Recognize limitations of slicing when used on string data types and when errors will occur

Learning Objectives

ENGINEERING
TEXAS A&M UNIVERSITY

Tuples and List Operations

- Create a tuple and use indexing with tuples
- Assign a value from a tuple or slice of a tuple
- Use within a program the functions:
  - len( ), max( ), min( ), sum( )
  - .append( ), .insert( ), .sort( ), .index( ), .pop( )

Dictionaries

- Create and identify dictionaries in Python
- Define and identify key-value pairs of a Python dictionary
- Assign or reference a single element within a dictionary
- Use a for loop with a dictionary in Python
- Use the in command with a dictionary or list in Python

Imagine you want to read in the grades of four quizzes:

```
grade_1 = int(input("Enter grade: "))
grade_2 = int(input("Enter grade: "))
grade_3 = int(input("Enter grade: "))
grade_4 = int(input("Enter grade: "))
```

We need four different variables to store four values.

If we want to find the average:

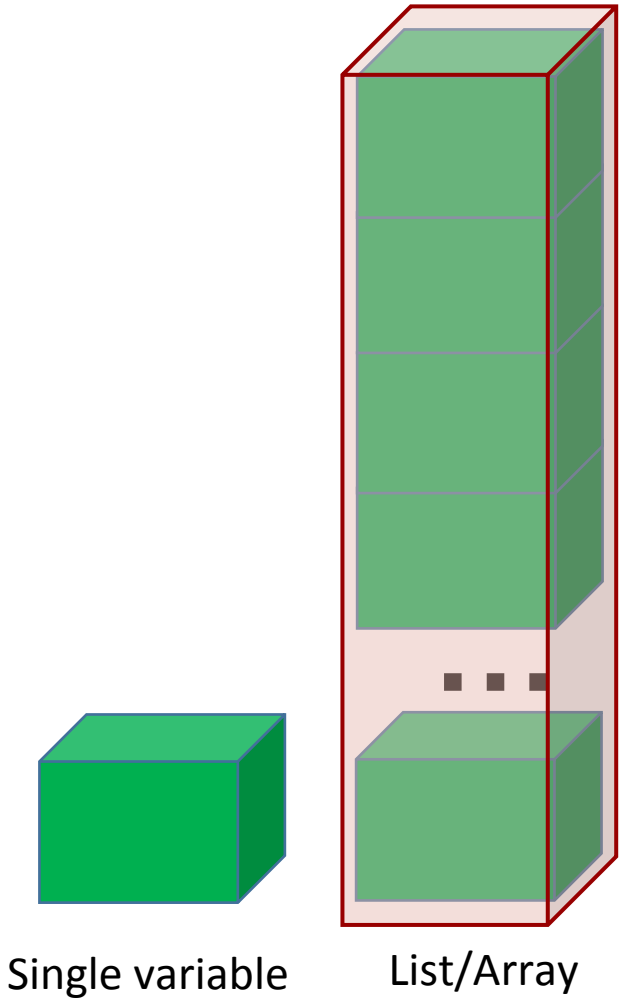
```
average = (grade_1 + grade_2 + grade_3 + grade_4) / 4
```

Similar data

- When we have several data values of the same type, storing them in separate variables doesn't make much sense
- We'd like to find a way to group all those similar data values together in one container.
- In Python, this is done is with a **list**

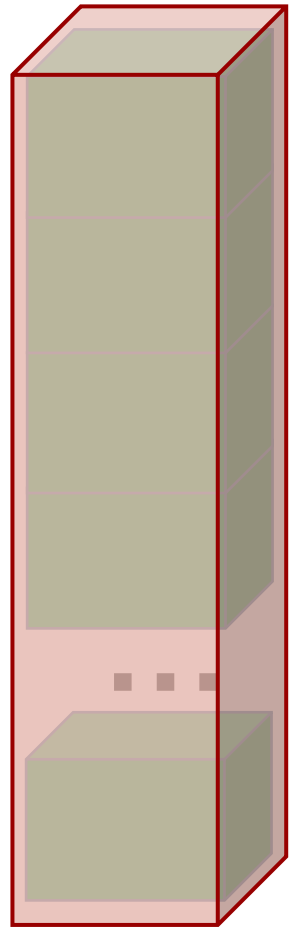
An array in memory

- We can think of a variable as a single “box” of memory
- A list can be thought of as a collection of boxes



Naming lists

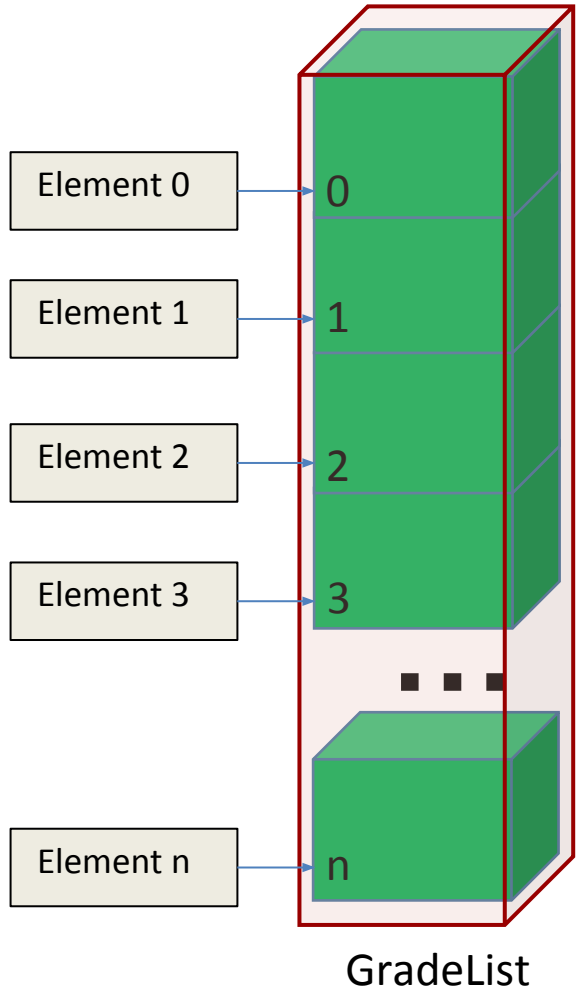
- A list acts like another “type” of variable
- We assign a single name to the entire list as a whole



GradeList

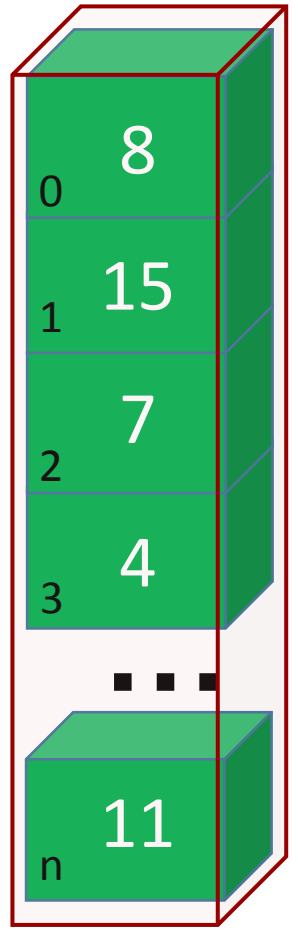
Naming lists

- A list acts like another “type” of variable
- We assign a single name to the entire list as a whole
- Individual elements of a list are numbered, starting at 0



Naming lists

- A list acts like another “type” of variable
- We assign a single name to the entire list as a whole
- Individual elements of a list are numbered, starting at 0
- Each individual element can contain a value



GradeList

Creating a List

- Lists are denoted by brackets, with comma-separated values inside.
- We can assign a list to a variable as normal

```
grades = [87, 93, 75, 100, 82, 91, 85]
names = ['George', 'John', 'Thomas']
```

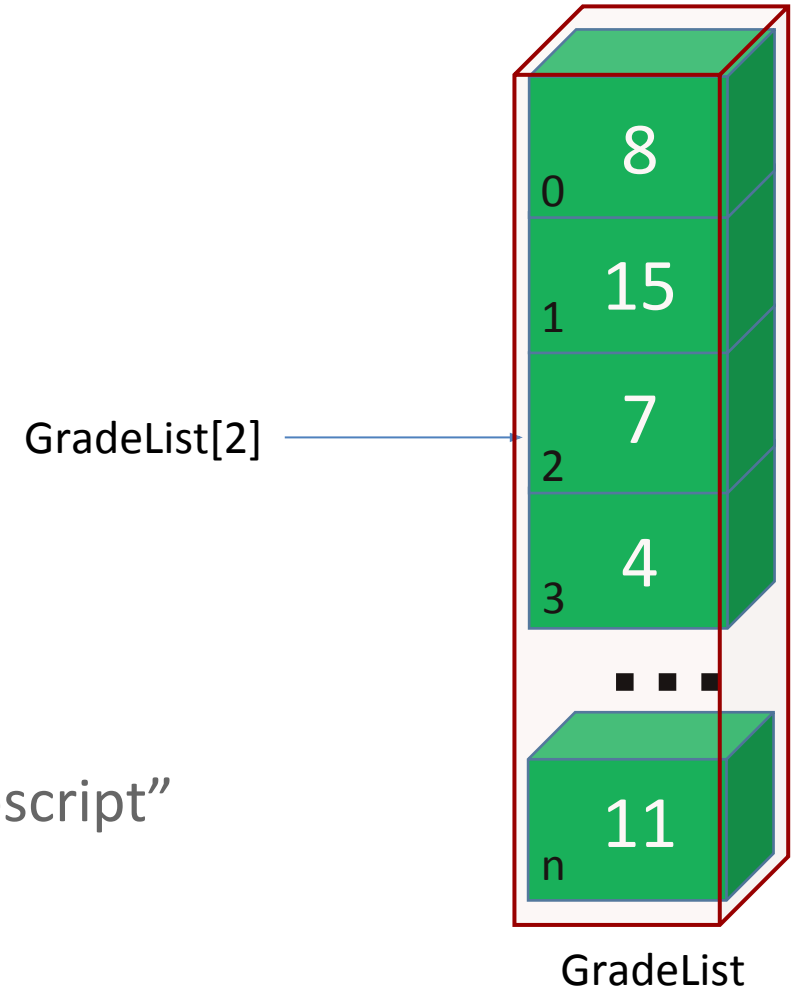
Referring to elements

To get a specific element, we write:

<List name>[<element number>]

For example, **GradeList[2]**

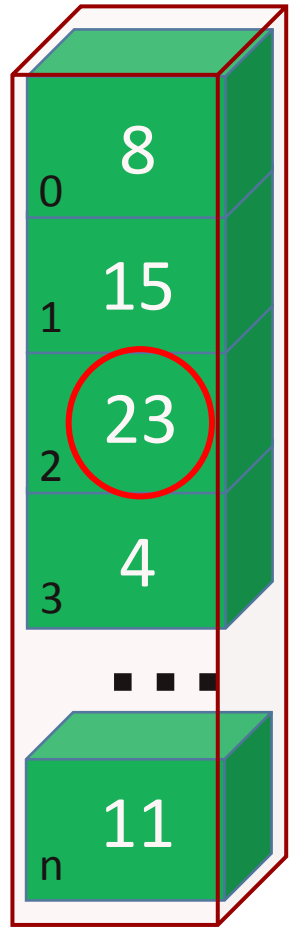
- The list name is GradeList
- We want element 2 (the third element)
- When speaking, it's common to say "sub" as in "subscript" (e.g., "GradeList sub 2")



Referring to elements

We can refer to elements in a program like we would a variable:

```
GradeList[2] = GradeList[0]+GradeList[1]
```



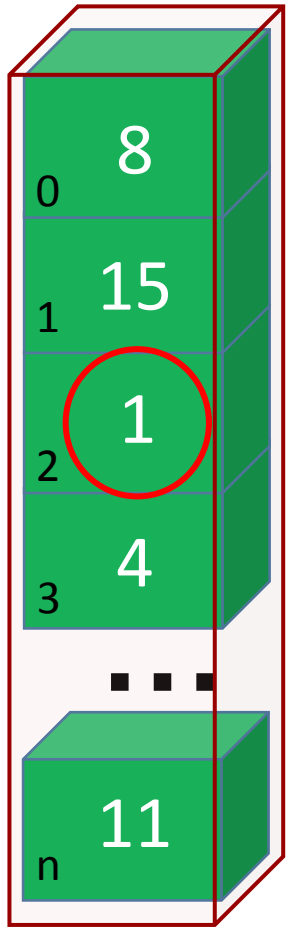
GradeList

Referring to elements

The element number can be a variable.

```
i = 2

GradeList[i] = 1
```



GradeList

What would the following print?

```
grades = [87, 93, 75, 100, 82, 91, 85]
print(grades[0], grades[6])
```

Console

What would the following print?

```
grades = [87, 93, 75, 100, 82, 91, 85]
print(grades[0], grades[6])
```

Console

```
87 85
```

Example: Indexing in Python (2)

ENGINEERING
TEXAS A&M UNIVERSITY

What would the following print?

```
grades = [87, 93, 75, 100, 82, 91, 85]
print(grades[7])
```

Console

What would the following print?

```
grades = [87, 93, 75, 100, 82, 91, 85]
print(grades[7])
```

Trying to access a list element that is past the final element will give an error

Console

```
IndexError: list index out of range
```

What would the following print?

```
grades = [87, 93, 75, 100, 82, 91, 85]
print(grades[-1])
```

Console

What would the following print?

```
grades = [87, 93, 75, 100, 82, 91, 85]
print(grades[-1])
```

Using negative values goes backward in the list!

Console

85

What would the following print?

```
grades = [87, 93, 75, 100, 82, 91, 85]
print(grades[-7])
print(grades[-8])
```

Console

What would the following print?

```
grades = [87, 93, 75, 100, 82, 91, 85]
print(grades[-7])
print(grades[-8])
```

If the length is n, we can index from -n to n-1.
We can go out of range backward, also.

Console

87

```
IndexError: list index out of range
```

What will the following output?

```
grades = [87, 93, 75, 100, 82, 91, 85]
grades[1] = 10
grades[2] -= 90
grades[3] = grades[1] + grades[2]
print(grades[3])
```

What will the following output?

Next

```
grades = [87, 93, 75, 100, 82, 91, 85]
grades[1] = 10
grades[2] -= 90
grades[3] = grades[1] + grades[2]
print(grades[3])
```

Console

What will the following output?

Next

```
grades = [87, 93, 75, 100, 82, 91, 85]
grades[1] = 10
grades[2] -= 90
grades[3] = grades[1] + grades[2]
print(grades[3])
```

Console

grades	
0	87
1	93
2	75
3	100
4	82
5	91
6	85

What will the following output?

```
grades = [87, 93, 75, 100, 82, 91, 85]
grades[1] = 10
grades[2] -= 90
grades[3] = grades[1] + grades[2]
print(grades[3])
```

Next

Console

grades	
0	87
1	10
2	75
3	100
4	82
5	91
6	85

What will the following output?

```
grades = [87, 93, 75, 100, 82, 91, 85]
grades[1] = 10
grades[2] -= 90
grades[3] = grades[1] + grades[2]
print(grades[3])
```

Next

Console

grades	
0	87
1	10
2	-15
3	100
4	82
5	91
6	85

What will the following output?

```
grades = [87, 93, 75, 100, 82, 91, 85]
grades[1] = 10
grades[2] -= 90
grades[3] = grades[1] + grades[2]
print(grades[3])
```

Next

Console

grades	
0	87
1	10
2	-15
3	-5
4	82
5	91
6	85

What will the following output?

```
grades = [87, 93, 75, 100, 82, 91, 85]
grades[1] = 10
grades[2] -= 90
grades[3] = grades[1] + grades[2]
print(grades[3])
```

Console

-5

grades	
0	87
1	10
2	-15
3	-5
4	82
5	91
6	85

Printing a list

Printing a list will display the list with square brackets, with the values of the elements separated by commas.

```
grades = [87, 93, 75, 100, 82, 91, 85]
print(grades)
```

Console

```
[87, 93, 75, 100, 82, 91, 85]
```

Finding the length of a list

`len(x)` returns the number of elements in list x

```
grades = [87, 93, 75, 100, 82, 91, 85]
print(len(grades))
```

Console

7

range(x) is a list

- range(x) is just a way to generate a list of all elements from 0 to x-1
- So, the following are equivalent:

```
for i in range(10):
    print(i)
```

```
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    print(i)
```

Lists of lists

We can make lists of lists


```
grid = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(grid[0][0])
print(grid[1][2])
```

Console

Lists of lists

We can make lists of lists

```
grid = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(grid[0][0])
print(grid[1][2])
```

Next

grid[0] is the list [1, 2, 3]
So, element [0] of grid[0] is 1

Console

1

Lists of lists

We can make lists of lists

```
grid = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(grid[0][0])
print(grid[1][2])
```

Next

grid[1] is the list [4, 5, 6]
So, element [2] of grid[1] is 6

Console

1

6

List operations

Until today, variables had a single assigned value

- To change the value, you assign a new value.

With lists, many values are held within the same list. Many times we don't wish to reassign everything within the list. We may wish to:

- add a value
- remove a value
- change a single value
- change a range of values

These type of changes are performed through **list operations**

List operations: append

We often want to add an element to a list

- For example, taking input from a user and adding the value to the end of an existing list.

The format of append is different from things we've seen to this point.

<list name>.append(<thing to add>)

List operations: append

We often want to add an element to a list

- For example, taking input from a user and adding the value to the end of an existing list.

The format of append is different from things we've seen to this point.

<list name>.append(<thing to add>)

Start with the name of the list

List operations: append

We often want to add an element to a list

- For example, taking input from a user and adding the value to the end of an existing list.

The format of append is different from things we've seen to this point.

<list name>.append(<thing to add>)

Then, there is a period and the word append

List operations: append

We often want to add an element to a list

- For example, taking input from a user and adding the value to the end of an existing list.

The format of append is different from things we've seen to this point.

<list name>.append(<thing to add>)

Then, inside of parentheses is the thing you want to add to the list

ENGINEERING
TEXAS A&M UNIVERSITY

Example: appending a new grade

```
grades = [87, 93, 75, 100, 82, 91, 85]
grades.append(80)
print(grades)
print(len(grades))
```

Console

```
[87, 93, 75, 100, 82, 91, 85, 80]
8
```

List operations: insert

We often want to add an element to a list

- Sometimes we don't want the new value at the end of the list.

The format of insert is similar to append, but we also specify where to place the new value.

<list name>.insert(<index>,<thing to add>)

The first value in the parentheses is the index value you want the inserted value to have once added.

List operations: insert

We often want to add an element to a list

- Sometimes we don't want the new value at the end of the list.

The format of insert is similar to append, but we also specify where to place the new value *(the index the new value will have after being inserted)*

<list name>.insert(<index>,<thing to add>)

The second value is the thing you want to add to the list.

ENGINEERING
TEXAS A&M UNIVERSITY

Example: inserting a new grade

```
grades = [87, 93, 75, 100, 82, 91, 85]
grades.insert(1,80)
print(grades)
print(len(grades))
```

Console

```
[87, 80, 93, 75, 100, 82, 91, 85]
8
```

List operations: delete

Sometimes we want to remove a value from a list

The delete command removes a specific item from the list, by item index:

del <list name>(<index>)

Notice this is a different format than other operators. (It's also only **del**, not the full word.)

List operations: .pop

Sometimes we want to remove a value from a list

The operation .pop returns the last item from the list, but also removes it from the list:

```
a_list = [0, 1, 2]
print(a_list.pop())
print(a_list)
```

Console

```
2
[0, 1]
```

List operations: concatenation

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list3 = list1 + list2
print(list3)
```

Console

```
[1, 2, 3, 4, 5, 6]
```

list1	
0	1
1	2
2	3

list2	
0	4
1	5
2	6

list3	
0	1
1	2
2	3
3	4
4	5
5	6

Notice that list 3 is a new list, in different memory locations than list1 and list2

Concatenating lists vs. appending

- *must* add lists (not elements)

```
grades = [87, 93, 75, 100, 82, 91, 85]
grades += [80]
j = 95
grades += [j]
print(grades)
```

Notice we convert the value 80 and the value in the variable j to a list, by adding [ ]

Console

```
[87, 93, 75, 100, 82, 91, 85, 80, 95]
```

List slicing

In Python, you can “slice” a list, or pull out a subpart of it

Slicing has the form:

<list name>[a:b]

a indicates the element to start with in the list

b indicates the element to stop **before** in the list

(In other words, you start “infinitesimally before” element a, and stop “infinitesimally before” element b)

Example: List Slicing

ENGINEERING
TEXAS A&M UNIVERSITY

```
grades = [87, 93, 75, 100, 82, 91, 85]

print(grades[0:3])
```

Console

Example: List Slicing

ENGINEERING
TEXAS A&M UNIVERSITY

```
grades = [87, 93, 75, 100, 82, 91, 85]

print(grades[0:3])
```

Console

Example: List Slicing

ENGINEERING
TEXAS A&M UNIVERSITY

```
grades = [87, 93, 75, 100, 82, 91, 85]

print(grades[0:3])
```

Console

```
[87, 93, 75]
```

Using lists in a loop structure

There are many methods to create a list using a loop, here's one:

```
for i in range(90)
    new_item = math.sin(math.rad(i))
    existing_list.append(new_item)
```

Here's something else—what's this do?:

```
for i in range(len(list_2))
    print(list_2.pop())
```

More on List slicing

Reminder—Slicing has the form:

<list name>[a:b]

a indicates the element to start with in the list

b indicates the element to stop **before** in the list

(In other words, you start “infinitesimally before” element a, and stop “infinitesimally before” element b)

Example: List Slicing

ENGINEERING
TEXAS A&M UNIVERSITY

```
grades = [87, 93, 75, 100, 82, 91, 85]

print(grades[4:5])
print(grades[-3:-1])
```

Console

Example: List Slicing

ENGINEERING
TEXAS A&M UNIVERSITY

```
grades = [87, 93, 75, 100, 82, 91, 85]

print(grades[4:5])
print(grades[-3:-1])
```

Console

```
[82]
[82, 91]
```

More list slicing

- If you leave off the starting value (a), it means “start at the beginning”
- If you leave off the ending value (b), it means “go to the end”

Example: More List Slicing

ENGINEERING
TEXAS A&M UNIVERSITY

```
grades = [87, 93, 75, 100, 82, 91, 85]

print(grades[:3])
print(grades[4:])
print(grades[:])
```

Console

```
[87, 93, 75]
[82, 91, 85]
[87, 93, 75, 100, 82, 91, 85]
```

Example: More List Slicing

ENGINEERING
TEXAS A&M UNIVERSITY

Slicing does not give out-of-range errors

- If you try to go past the beginning/end of a list when slicing, you just get the beginning/end of the list.

```
grades = [87, 93, 75, 100, 82, 91, 85]
print(grades[4:300])
print(grades[-100:-3])
```

Console

```
[82, 91, 85]
[87, 93, 75, 100]
```

List slicing operations

- List slicing does not make a copy of the sublist until it is assigned somewhere.
- So, you can actually change/add to a list by reassigning to a sliced region.
  - This is easiest to see by example

Making a 'sublist'

```
grades = [87, 93, 75, 100, 82, 91, 85]
sublist = grades[1:4]
print(grades)
print(sublist)
```

Console

```
[87, 93, 75, 100, 82, 91, 85]
[93, 75, 100]
```

Making a 'sublist' (2)

```
grades = [87, 93, 75, 100, 82, 91, 85]
sublist = grades[1:4]
sublist = []
print(grades)
print(sublist)
```

sublist is a copy of part of the grades list, so adjusting sublist did not affect grades

Console

```
[87, 93, 75, 100, 82, 91, 85]
[]
```

Deleting a slice

```
grades = [87, 93, 75, 100, 82, 91, 85]
sublist = grades[1:4]
grades[1:4] = []
print(grades)
print(sublist)
```

We took the part of grades from element 1 through element 3, and replaced it by an empty list!

sublist is unaffected

Console

```
[87, 82, 91, 85]
[93, 75, 100]
```

Inserting a slice

```
grades = [87, 93, 75, 100, 82, 91, 85]
sublist = grades[1:4]
grades[5:5] = [65, 50]
print(grades)
print(sublist)
```

We can also add values to the list.

Notice that [5:5] refers to the location **just before** element 5.

Console

```
[87, 93, 75, 100, 82, 65, 50, 91, 85]
[93, 75, 100]
```

Strings as lists

Strings are essentially lists of characters! (with some key differences…)

We can slice strings just like we slice lists (but we can't assign to a sliced region)

```
name = "Texas A&M University"
print(name[6:9])
name[6:9] = []
```

Console

```
A&M
TypeError: 'str' object does not support item assignment
```

Some last notes on lists

It is a good idea (but not required) for lists elements to be the same type

- It allows you to loop through, doing the same operation to each element

Remember lists are not vectors

- Adding (concatenating) lists puts one list on the end of the other

There are many more list operations and things you can do with slicing, but these are a good start.

Tuples

A tuple is like a list, with one important distinction: it cannot change values

- The term used is that it's **immutable**
- We can still use it to compute with

Where lists use brackets [ ], tuples use parentheses or comma-separation:

```
my_tuple = (1, 2, 3)
```

← Tuples

```
my_tuple = 1, 2, 3
```

```
my_list = [1, 2, 3]
```

← List

Tuple operations

```
my_tuple_A = (1, 2, 3)
my_tuple_B = 1, 2, 3
my_list = [1, 2, 3]
print(my_tuple_A)
print(my_tuple_B)
print(my_list)
```

Output

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

← Tuples

← List

Tuple operations

Most operations are just like those for lists:

```
my_tuple = (1, 2, 3, 4)
print(my_tuple[2])
print(my_tuple[1:3])
print(len(my_tuple))
```

Output

```
3
(2, 3)
4
```

Tuple operations

We can assign values *from* a tuple:

```
my_tuple = (1, 2, 3)
a, b, c = my_tuple
print(a)
print(b)
print(c)
```

Output

```
1
2
3
```

Tuple operations

But, assignment is not allowed:

```
my_tuple = (1, 2, 3, 4)
my_tuple[2] = 5
```

Output

```
TypeError: 'tuple' object does not support item assignment
```

Returning tuples

A function will only return **one** value

- But, that value can be a tuple
- Then, *you* can assign the parts of a tuple to various values

Example:

```
a, b = some_function()
```

Then, those tuple values are assigned to the two variables, a and b

Some Useful Functions

There are some built-in functions that are particularly useful for lists:

- `len(A)` – for the list / tuple A, gives the size (we already saw this)
- `max(A)` – for the list / tuple A, gives the maximum element
- `min(A)` – likewise for the minimum element
- `sum(A)` – the sum of all the elements in a list / tuple A

A Brief Introduction to Dictionaries

Dictionaries and lists are both “containers” in Python, with similarities and differences.

Dictionaries associate key —value pairs

- Every key has a single value
- Multiple keys can have the same value

In a “real” dictionary, each “key” is a word, and the “value” is …

- the definition, or
- the pronunciation, or
- the synonyms, etc...

Dictionary

Dictionaries in Python are created by using braces { }

An empty dictionary is just the empty braces { },

e.g.: `age = { }`

Initial elements can be described as `<key>:<value>`, separated by commas, inside the braces { }

e.g: `age = {'John' : 21, 'James' : 25}`

Dictionary Elements

Access a dictionary element using [ ]

- Using [ ] is analogous to an index in a list, but...
- The value in the [ ] is the key
- The key does *not* have to be an integer!

Example:

```
age['John'] = 23
age['Jill'] = 21
age['James'] = 20
age['Jessica'] = 23
```

Operations on Dictionaries and Lists

For loops work with a dictionary similar to a list

```
for <iterator> in <dictionary/list>:
    #Iterator takes on the value of the KEY in a
    #  dictionary, or a value in a list
```

"in" command can test whether an element is in a list

```
if <item> in <dictionary/list>:
    # True if the item is a key in a dictionary, or a
    #   value in a list
```

Example

```
age = {'John' : 21, 'Jill' : 21}
age['James'] = 20
age['Jessica'] = 23
print(age)
for i in age:
    print("key", i, ", value",age[i])
if 'James' in age:
    print("Yes for James")
else:
    print("No for James")
if 'Joe' in age:
    print("Yes for Joe")
else:
    print("No for Joe")
```

Output

Example

```
age = {'John' : 21, 'Jill' : 21}
age['James'] = 20
age['Jessica'] = 23
print(age)
for i in age :
    print("key", i, ", value",age[i])
if 'James' in age:
    print("Yes for James")
else:
    print("No for James")
if 'Joe' in age:
    print("Yes for Joe")
else:
    print("No for Joe")
```

Output

```
{'John': 21, 'Jill': 21, 'James': 20, 'Jessica': 23}
key John , value 21
key Jill , value 21
key James , value 20
key Jessica , value 23
Yes for James
No for Joe
```

Current Assignments

ENGINEERING
TEXAS A&M UNIVERSITY

Complete the ETID module on the eCommunity page

- Due 10/06, by end-of-day

Lab Assignment 6	Due 10/06 by end-of-day
Lab Assignment 6b	Due 10/06 by end-of-day

Preactivity: Complete zyBook chapters 7.8+ *(excluding optional sections and 7.22)* prior to class next week

Exercise

If we have a string named city_name, how would we get the first 4 and last 4 characters of the city name?

Exercise

If we have a string named city_name, how would we get the first 4 and last 4 characters of the city name?

```
city_name[:4]
city_name[-4:]
```

Exercise

If we have a string named city_name, how would we get the first 4 and last 4 characters of the city name?

```
city_name[:4]
city_name[-4:]
```

But, what if the name was only 3 characters long?

Exercise

If we have a string named city_name, how would we get the first 4 and last 4 characters of the city name?

```
city_name[:4]
city_name[-4:]
```

But, what if the name was only 3 characters long? (Try it out…)

```
city_name = "Arp"
print(city_name[:4])
print(city_name[-4:])
```

Exercise

If we have a string named city_name, how would we get the first 4 and last 4 characters of the city name?

```
city_name[:4]
city_name[-4:]
```

But, what if the name was only 3 characters long? (Try it out…)

```
city_name = "Arp"
print(city_name[:4])
print(city_name[-4:])
```

Console

```
Arp
Arp
```

Remember, when slicing lists, we don't get out of range errors!