Lesson 4

Admin Matters

You may use any IDE during your mid-term test

- You cannot use the internet during your mid-term test
-

Programming Quiz

- Begins promptly at the start of **Session 3**
- You have 15 minutes. (If you are late you have less time)
- Check that you are logged in as yourself in Tutor
- **NOTE: We will ONLY accept submissions via Vocareum**

1D Project Proposal

- Begin thinking about what you want to do
- You can bounce your ideas off us before the presentation
- The presentation is on Week 6 Session 3 (after your programming quiz).

Homework

- Homework problems continue and submission is on Tutor.
- Don't forget to press "Check" before you submit.

Digital World + Chemistry Combined Assignment

- Please access the wikispaces page.
- Solve the problems progressively
- You can work in groups but you have to submit individually.
- The submission link is at Week 7 of Tutor and problems are due every week.

Pre-Reading

- Read **Week 5** materials before Session 1. I encourage you to post your queries on Piazza.

Recall Last Week's Lesson

b.socrative.com 593583

List Aliasing

```
 8list_a = ['koala', 'wombat', 'kangaroo']
 9list_b = list_a
10list_a.append('tasmanian devil')
```

`len(list_a) == len(list_b) is True / False.`

List Slices

`my_list = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]`

Write a possible list slice to get the following:

`a = [10, 20, 30, 40, 50, 60, 70, 80, 90]`

`b = [100, 80, 60, 40, 20]`

While Loop

```
19a = 4
20while (a > 3):
21    print a
22    a -= 1
```

You will get an infinite loop. True/False

For loop

```
24for i in range(1,10,2):
25    print i
```

What will you see on the console?

```
27my_string = 'farm'
28for i in my_string:
29    print i
```

What will you see on the console?

Strings

We recall that the following will give an error.

```
32my_string = "old macdonald"
33my_string[3] = 'd'
```

A variable with string data type is said to be immutable.

Because of this, if you decide to modify a string variable,
the result has to be stored in another string variable.

String have their own set of methods.

```
35new_string = my_string.replace('old','young')
```

What is seen on the screen?

For - Loops

For loops vs While loop: Summing 1 + 2 + 3 + 4 ….

Version 1. How many terms do you need to reach a sum that just exceeds 10000?

Version 2. What is the sum of n terms? (Using a loop)

Version 3. Use one line to solve

For-loops can be nested

```
27 for i in range(1,4):
28     for j in range(10,30,10):
29
30         print "i = ", i, "and j = ", j
```

How many lines will be printed out?

A) 2 **B**) 3 **C**) 5 **D**) 6

Clicker Question 1

What is the value of `val` after this code executes?

```
val = 0
for i in 'abc':
  for j in 'def':
    val += 1
```

A) 1 **B**) 3 **C**) 6 **D**) 9 **E**) 27

Lists

List/String operations with overloaded operators

An operator is **overloaded** if its meaning changes for different operands.

If the operands are strings or lists, the operators `+` `*` do very different things

```
part_a = "old macdonald"
part_b = "farm"
line_one = part_a + part_b
line_two = partb*3
```

`print(line_one)` gives `old macdonaldfarm.` True/False?

`print(line_two)` gives `farm farm farm.` True/False?

This example demonstrates **string concatenation**.

String concatenation

Hence you could also format your print statements like this if you'd like.

```
animal = "cats"       #I like cats!
venue = "farm"
quantity = 3
output_string = "there were " + str(quantity) + " " +
animal + "at the " + venue
print(output_string)
```

Check if a list contains a particular entry

```
73 my_list = [10,20,30,40,50,60,70,80]
74 a = 15 in my_list
75 b = 60 in my_list
```

Using list methods

What is the value of a after this code runs?

```
a = [2, 4, 6, 8]
a.remove(4)
a.pop(2)
```

- A. [2, 4]
- B. [6, 8]
- C. [2, 6]
- D. [2, 8]
- E. Nothing; the code produces an error

A list can be nested

The Python Memory Story (5)

A nested list constructed by other lists contains references to those lists, not values.

```
14 a_list = [10,20,30]
15 b_list = ['c','d','e']
16 my_list = [ [45], a_list, b_list]
17
18 my_list[1][2] = 51
19 print my_list[1][2]
20 print a_list[2]
```

The output of line 20 is `30`. True/False

```
14 a_list = [10,20,30]
15 b_list = ['c','d','e']
16 my_list = [ [45], a_list, b_list]
17
18 b_list[1] = 'gg'
19 print my_list[2][1]
```

The output of line 19 is ‘gg’ True/False.

A nested for-loop helps you to access the entries in a nested list

A 2D matrix is like a nested list.

```
79 matrix = [ [10,20,30],[40,50,60],[70,80,90] ]
80
81 for i in range(len(my_list)):
82
83     for j in range(len(my_list[i]):
84
85         print my_list[i][j]
```

Shallow Copy vs Deep Copy

We recall that this statement allows us to copy or clone a list.

```
a = [1,2,3]
b = a[:] #or other options
```

We recall that this is called a shallow copy.

A **shallow copy** copies the **contents of the top level** of a nested list. Two ways of shallow copy. From the print statements below, we can see that the inner lists are not copied.

```
import copy
a = [1,2,3]
b = [10, a]
c = copy.copy(b)      #using the copy library
c = b[:]              #or using list slice

print( b is c )           #True/False?
print( c[1] is a )        #True/False?
```

A **deep copy** copies **all the contents** of a nested list. Only the `copy` library helps us to achieve this.

```
a = [1,2,3]
b = [10, a]
c = copy.deepcopy(b)      #using the copy library
print(c[1] is a)          #True/False?
```

More on objects

Recall

Thus far, an object refers to any data in memory.
Each object has an **address** in memory. You can use the `id` function.

You have also learn about **custom data types** using the `class` keyword which have more than one variable that you can treat as one entity or object.

You have seen that we can do operations on a list variable using special commands by somehow "attaching a function to it" using a dot operator.

More on objects

An object can have a set of functions (called **methods**) that define its behaviour.

methods - functions that belong to an object. Each type of object (lists, strings, dictionary) have their own set of **methods**

So, an object is a space in memory that contains data in variable(s) and can also have methods attached to it.

Mutable vs immutable

You have also seen that a string can be treated like a list, but for one big difference.

Objects can be **immutable** (cannot be modified) or **mutable** (can be modified).

You have seen that a list is mutable. Immutable datatypes include string and tuple. Hence in the following lines, which returns a TypeError?

```
my_list = [1, 2, 3, 4, 5]
my_tuple = (1, 2, 3, 4, 5)
my_string = 'abcde'
```

A. `my_list[3] = 20`
B. `my_tuple[3] = 20`
C. `my_string[3] = 'k'`
D. B & C
E. None of the above

The python memory story (4)

Since a string is immutable, why would this code work?

What's the difference between the three lines?

Is data copied?

```
my_string.replace('a','z')
my_string = my_string.replace('a','z')
new_string = my_string.replace('a','z')
```

Dictionary

A dictionary is a non-sequential list

Which of the following is best suited for a dictionary instead of a list?

- A. The order in which people finish a race
- B. The ingredients necessary for a recipe
- C. The names of world countries and their capital cities
- D. 50 random integers

Format of a dictionary

```
my_dict = { key1: value1, key2: value2, (and so on)}
```

The keys of a dictionary need to be immutable

Can be used as keys	**Cannot be used as keys**
Number, string, tuple (more later)	List, dictionary

What methods are available for a dictionary object

```
dir(dict)
```

Some tasks you do to a dictionary

1. Create an empty dictionary, then create a new key-value pair
2. Check the number of key-value pairs
3. Access a value given its key
4. Check if a key is in a dictionary or not
5. Get a **list** of the keys
6. Get a **list** of the values
7. Check if a value is in a dictionary or not
8. Delete a particular key-value pair
9. Clear the entire dictionary

Examples

```
59 #create an empty dictionary
60 countries = {}
61 musical_terms = dict()
62
63 #add entries
64 countries['Germany'] = 'Deutschland'
65 countries['France' ] = 'Francaise'
66 musical_terms['allegro'] = 'fast'
67 musical_terms['andante'] = 'at a walking pace'
68
69 #number of key-value pairs
70 print( len(countries) )
71
72 #access a value given a key
73 print( countries['Germany'])
74
75 #check if a key is in a dictionary
76 print( 'Germany' in countries )
77 print( 'lento' in musical_terms)
78
79 #get a list of all the keys / values
80 a = countries.keys()
81 b = musical_terms.values()
82
83 #printing out a particular value
84 for key in countries:
85     if( countries[key] == 'France' ):
86         print(countries[key])
87
```

Clicker Question using the get method

The get method checks for a key (specified in the first argument).
if the key exists, then it returns the value associated with the key.
if the key does not exist, then it returns the second argument.

```
36my_dd = {'a':5}
37my_dd['b'] = my_dd.get('c',9)
38my_dd['k'] = my_dd.get('a',2)
```

After this code, `my_dd` is

A. {'a': 5, 'b': 9, 'k': 2}
B. {'a': 5, 'b': 9, 'k': 5}
C. {'a': 5, 'b': 5, 'k': 2}
D. {'a': 5, 'b': 5, 'k': 5}
E. Error

```
animals = {'cat': 'meow', 'dog': 'woof', 'rat': 'squeak'}
my_pets = animals
my_pets['duck'] = 'quack'
print( animals['duck'] )
```

The print statement gives a KeyError. A. True B. False

Looping over a dictionary

```
animals = {'cat': 'meow',
           'dog': 'woof',
           'rat': 'squeak'}

#what is a? key or value?
for a in animals:
    print(a)

#what is b? key or value?
for a,b in animals.items():
    print(a,b)

#what data type is c?
for c in animals.items():
    print(c)
```

Clicker Question: difference between dictionary and lists

Which of the following **is** a difference between lists and dictionaries?

- A. List elements cannot be mutable, but dictionary values can be mutable
- B. Assigning to an index that does not exist in a list is an error, but assigning a value to a key that does not exist in a dictionary is not
- C. A list can contain a dictionary as one of its elements, but a dictionary cannot contain a list as one of its values
- D. There is a `dict` constructor that creates a dictionary from a suitable object, but there is no `list` constructor that similarly creates lists

Tuples

When your functions return two values (or more),
you are actually returning another data type called a **tuple**. A tuple is an immutable data type.

```
def simple(a):
    return a*3, a + 5

#you would have done this----
value1, value2 = simple(10)
print(value1, value2)

#or this--------------------
values = simple(10)
print( type(values) )
print( values[0], values[1] )

values[0] = 1  #is this possible?
```

What are the methods associated with a tuple objects?

The overloaded operators for lists are available to tuples

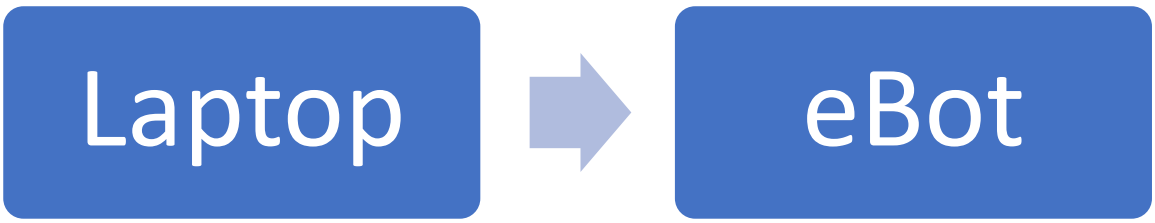
```
a = (10, 20)
b = (33, 55)
c = a*3 + b
print( len(c) )
```

You can make a tuple a key of a dictionary. Why?

```
dd=dict()
values = (30, 10)
dd[values] = 10  #making a tuple a key
```

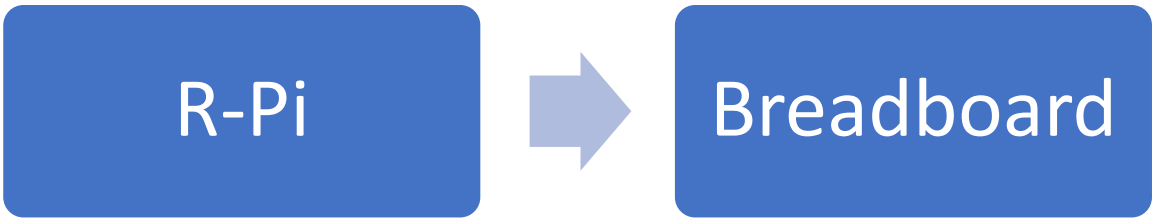
1D Mini Project

1D Mini Project 1 Recap




```
15      ebot.wheels(speed_left, speed_right)
16      sleep(duration)
```

1D Mini Project 2 Recap

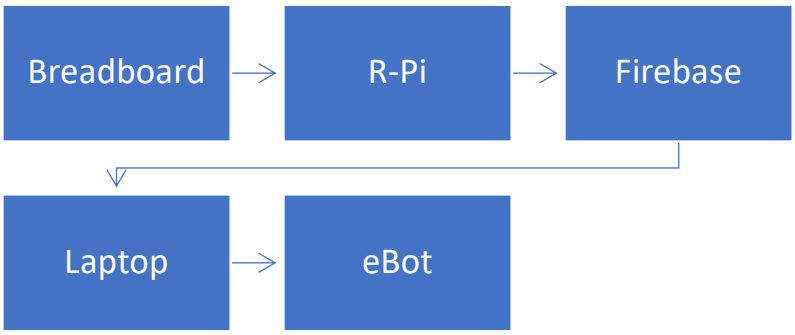


Within an infinite while loop:

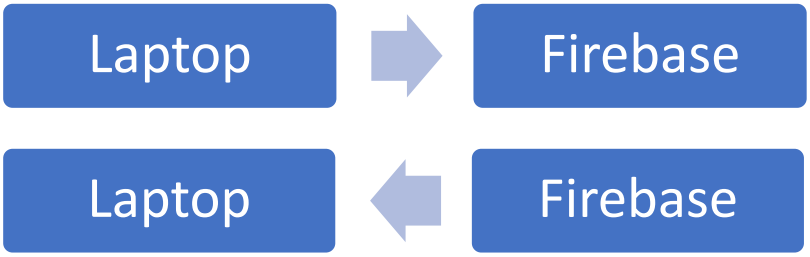
```
36      if GPIO.input(switch) == GPIO.HIGH:
37          GPIO.output(led[1], GPIO.LOW)
38          blink(led[0], 1)
39      else:
40          GPIO.output(led[0], GPIO.LOW)
41          blink(led[1], 1)
```

1D Mini Project 3

Overview



wk4_firebasebasics.py



Get used to putting and retrieving

- Integers, floats
- Lists, nested lists
- Dictionaries, nested dictionary

wk4_raspberrypi.py

Write code to capture button presses

and upload the data to Firebase

wk4_ ebot.py

Write code to retrieve data from firebase
and command the ebot to move according to the instructions