Lesson 5

Admin Matters

How many cannot install/use anaconda?

- You cannot use the internet during your mid-term test
- Cutoff the internet on your device
  and see if you can start your favourite IDE
- Install Python 3 IDLE

Programming Quiz For Week 5

- Begins promptly at the start of **Session 3**
- You have **20 minutes**. (If you are late you have less time)
- Check that you are logged in as yourself in Vocareum
- **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).
- Each team should prepare a 4-5 -minute presentation and be ready for Q&A. Maximum **Five slides**.
- Work out the sequence of presentation among yourselves.

Homework

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

Digital World + Chemistry Combined Assignment

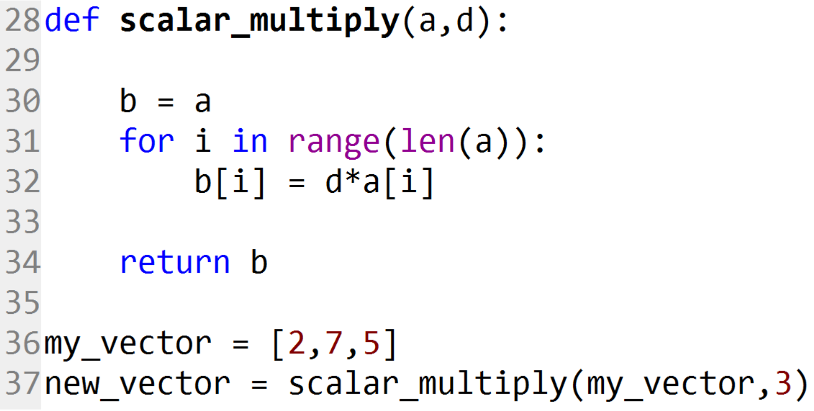
- Please access the wikispaces page.
- Solve the problems progressively
- You can work in groups but you have to submit individually.
- The submission link is at Week 7 of Tutor and problems are due every week.

Pre-Reading

- Read **Week 6** materials before Session 1. I encourage you to post your queries on Piazza.

Recall main points of what you learnt so far

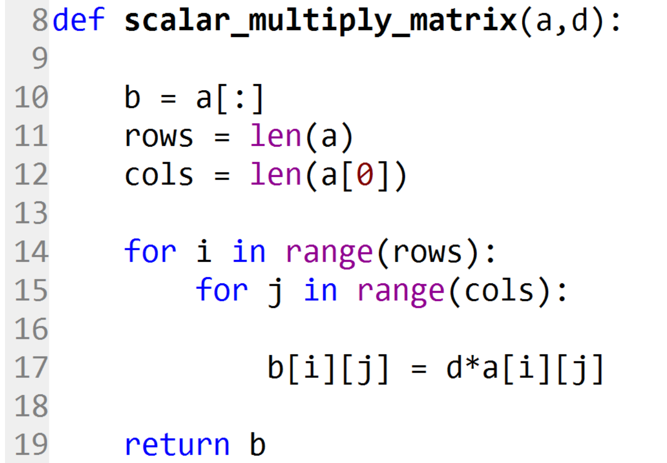
Clicker Question



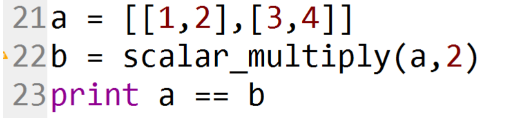
```
def scalar_multiply(a,d):

    b = a
    for i in range(len(a)):
        b[i] = d*a[i]

    return b

my_vector = [2,7,5]
new_vector = scalar_multiply(my_vector,3)
```

`print( new_vector)` gives `[2,7,5].`   True / False

Clicker Question



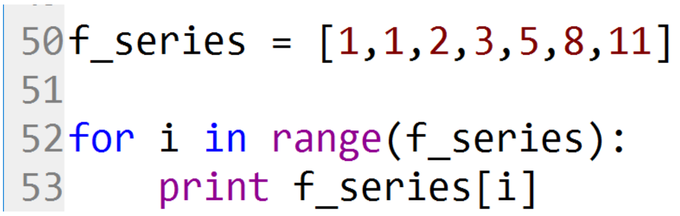
```
 8def scalar_multiply_matrix(a,d):
 9
10    b = a[:]
11    rows = len(a)
12    cols = len(a[0])
13
14    for i in range(rows):
15        for j in range(cols):
16
17            b[i][j] = d*a[i][j]
18
19    return b
```



```
21a = [[1,2],[3,4]]
22b = scalar_multiply(a,2)
23print a == b
```

Assuming the inputs are correct. Line 23 gives True/False?

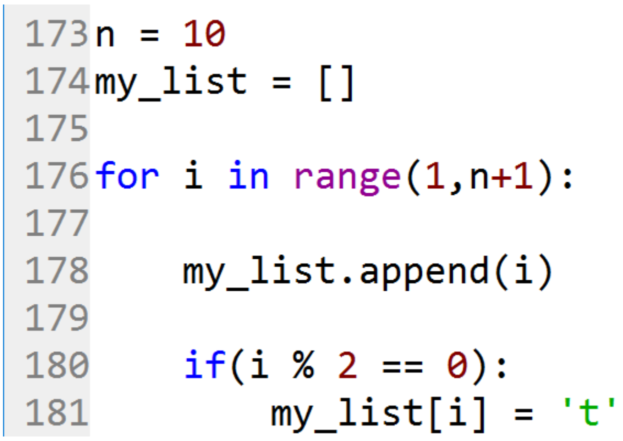
What is wrong with this code?



```
f_series = [1,1,2,3,5,8,11]

for i in range(f_series):
    print f_series[i]
```

Find the one mistake in the code

To display `[1,'t',3,'t',5,'t',7,'t',9,'t']`



```
173 n = 10
174 my_list = []
175 
176 for i in range(1,n+1):
177 
178     my_list.append(i)
179 
180     if(i % 2 == 0):
181         my_list[i] = 't'
```

A) No problem, code works fine

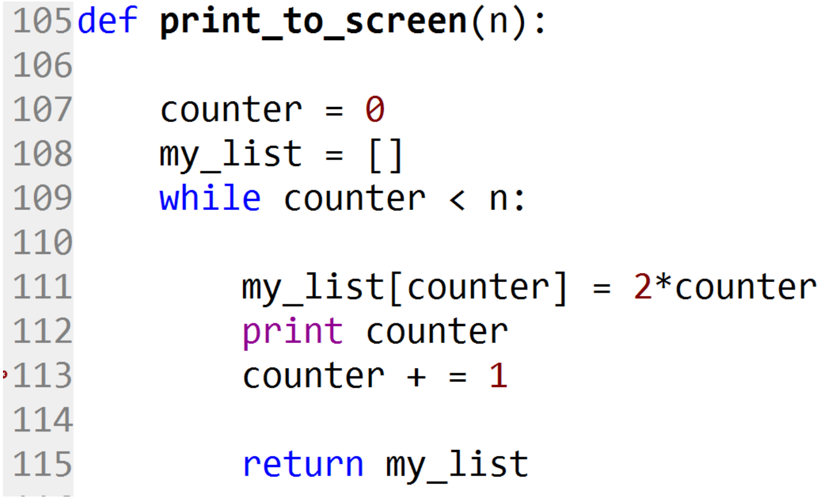
B) Line 176, should be **`for i in range(n)`**

C) Line 178 should be **`my_list[i] = i`**

D) Line 181 will give Index Error

E) Some combination of B,C and D.

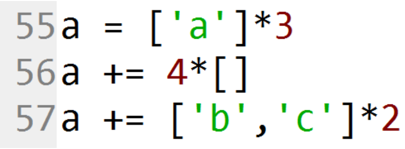
What is wrong with this code?



```
def print_to_screen(n):

    counter = 0
    my_list = []
    while counter < n:

        my_list[counter] = 2*counter
        print counter
        counter + = 1

        return my_list
```

A) No problem, code works fine
B) There is one error
C) There are two errors
D) There are three errors
E) There are four errors

Clicker Question

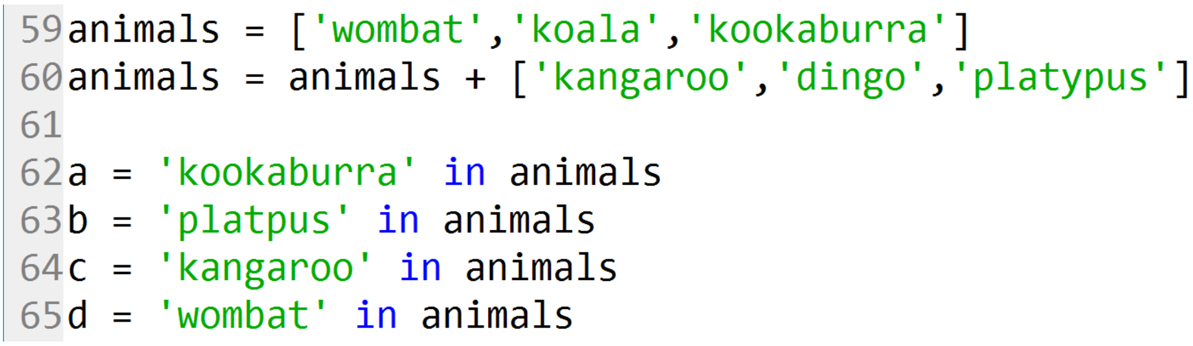


```
a = ['a']*3
a += 4*[]
a += ['b','c']*2
```

`len(a)` is

A) 3 **B)** 5 **C)** 7 **D)** 9 **E)** 11

Clicker Question



```
animals = ['wombat','koala','kookaburra']
animals = animals + ['kangaroo','dingo','platypus']

a = 'kookaburra' in animals
b = 'platpus' in animals
c = 'kangaroo' in animals
d = 'wombat' in animals
```

Which of the following Boolean variables is False?

A) a **B)** b **C)** c **D)** d **E)** all are True

Print Formatting

Consider the following code.

```
country = 'myanmar'

pop = 52.89

d = '{0} has {1} million people'.format(country, pop)

print(d)
```

The placeholder {1} is replaced such that the following string is printed.

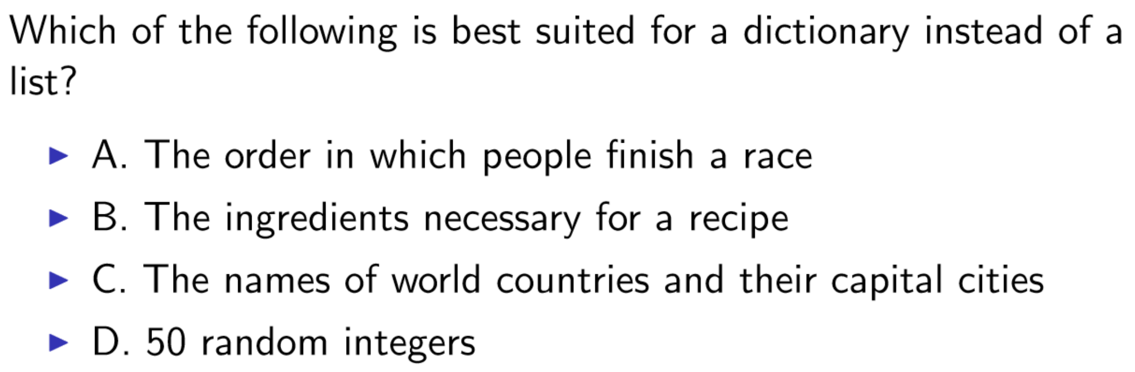
```
1234567890123456789012345678901234567890
  myanmar has   52.890 million people
```

What was {1} replaced by?

A) {1:7f}   B) {1:8f}   C) {1:7.3f}

D) {1:8.3f}   E) {1:3.8f}

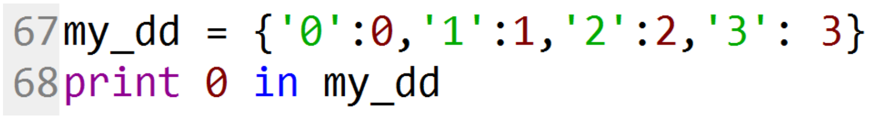
Dictionary



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

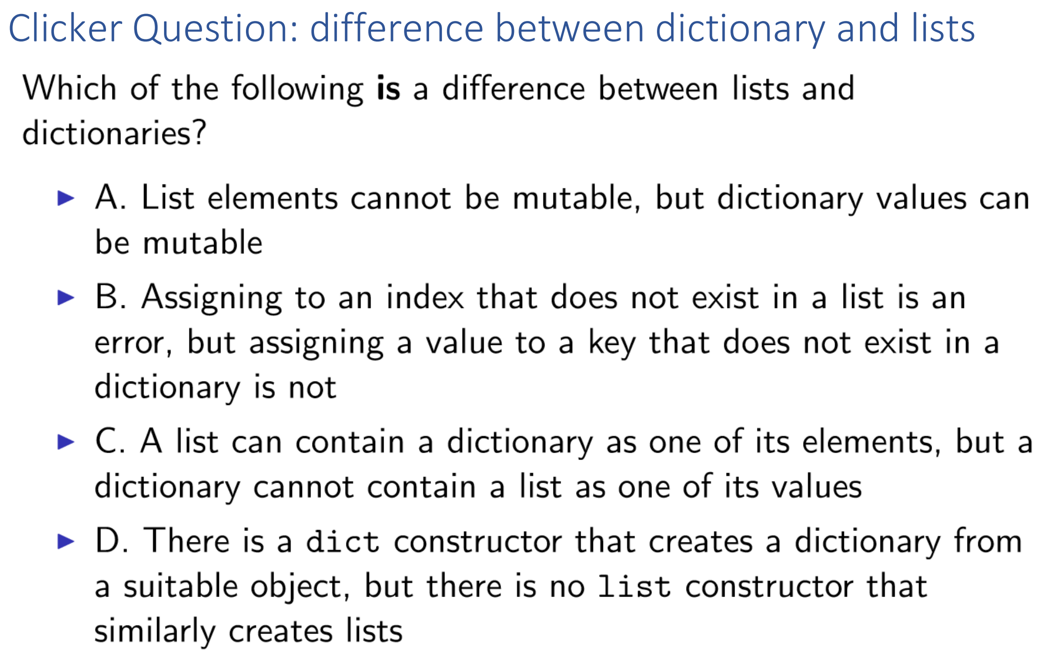
- A. The order in which people finish a race
- B. The ingredients necessary for a recipe
- C. The names of world countries and their capital cities
- D. 50 random integers

Clicker Question



```
67my_dd = {'0':0,'1':1,'2':2,'3': 3}
68print 0 in my_dd
```

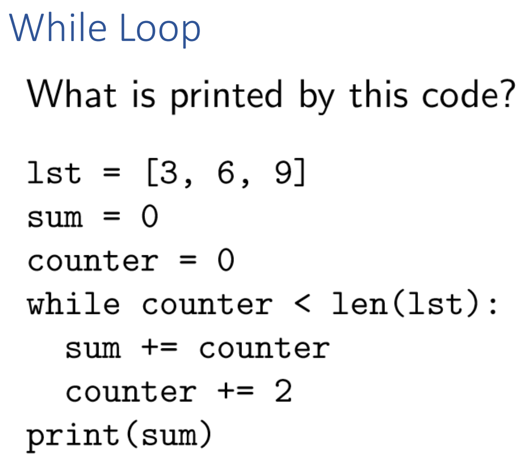
Line 68 gives True/False



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



While Loop

What is printed by this code?

```
lst = [3, 6, 9]
sum = 0
counter = 0
while counter < len(lst):
  sum += counter
  counter += 2
print(sum)
```

A) 18 **B**) 6 **C**) 2 **D**) 3 **E**) None of the above

Dictionaries

Getting Familiar with Dictionaries

Given the following dictionary:

```
population = {'australia': 23,
              'thailand': 67,
              'malaysia': 29,
              'south korea': 50,
              'vietnam': 90}
```

Write code to

1. Obtain the population of Thailand.
2. Check if Cambodia is a key in the dictionary, if not, add it to the dictionary with its population.
3. Determine the number of key-value pairs in the dictionary
4. Get a list of countries in the dictionary
5. Get a list of tuples containing the countries and the values
6. Display all the countries in the dictionary in the format
   ```
   The population of australia is 23 million
   The population of thailand is 67 million
   (and so on)
   ```
   Try two ways of doing this.
7. Display all countries that have population larger than 30 million.
8. Remove Australia from the dictionary

Random Module

How to use

```
import random

random.seed()

a = random.randrange(1,7) #simulate rolling a die

b = random.randint(1,6) #does the same thing

c = random.random()

fruit_list = ['apple', 'banana', 'chiku']

fruit = random.choice(fruit_list)
```

Why

A random number generator (RNG) is **pseudorandom**.

To begin the generation, you need a **seed**. Changing the seed ensures that the sequence of random numbers generated is not always the same.

To see the effect of the seed, call the following function as many times as you wish …

```
import random
def rng():
    random.seed(10) #the seed is always the same
    for i in range(10):
        print ( random.randint(1,6) )
```

A RNG has a **period**. The numbers will repeat themselves after the period.

The period of the RNG used by python is 2^{19337}-1. (Invented by two Japanese scientists, search for **Mersenne Twister**)

A RNG with a sufficiently long period is important in computer simulations where random numbers are used to represent real-life events.

The worst RNG I saw had period 2^{32} – 1. You do not want to be generating the same set of numbers over and over again …

Loop With Two Counters Doing different things

(something like the matrix movie, perhaps?)

Write a function **`the_matrix(max_lines, line_width)`** that prints out **`line_width`** random characters per line for **`max_lines`** line, each followed by a blank space.

Hint, you need to use the `sleep(0.01)` command so that there is a noticeable delay when printing each character or line.

`the_matrix(100,14)` gives

```
b 7 X P L i x u 4 V 8 j e
8 W B q y e G T d B A P o
d r Y z Y V d l 5 m u D 3
Z 5 n J G w A T n o x o U
e V Y h f p q D f y f 6 c
R Q M J y K c M w 1 q H A
h c C W t 8 x c 0 D M l P
8 d P i F w C s D w D s H
A I d C 7 s w X C W L d N
9 c k Z 1 h h g 4 3 d m N
r T 3 K 6 H E J I O i x y
```

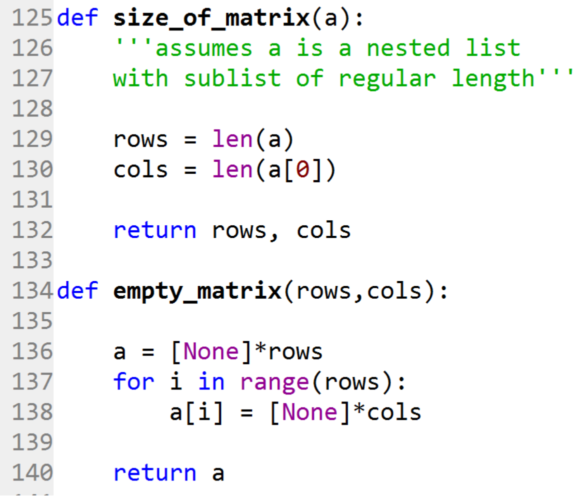
We begin it for you here:

```
import string

import random

import time

def the_matrix(max_lines,line_width):

    lines = 1  #counter for the number of lines

    line_char = 1 #counter for no. of chars per line

     #print my_string to see what this gives you

    my_string = string.ascii_letters + '0123456789'
```

This Week’s Big Idea

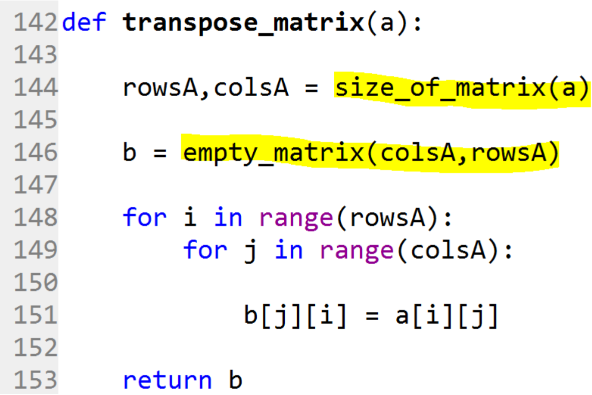
- Functions can call other functions
- You can write functions to tackle smaller tasks
- You can then write (bigger) functions to call these other functions to solve your problems

Simpler functions

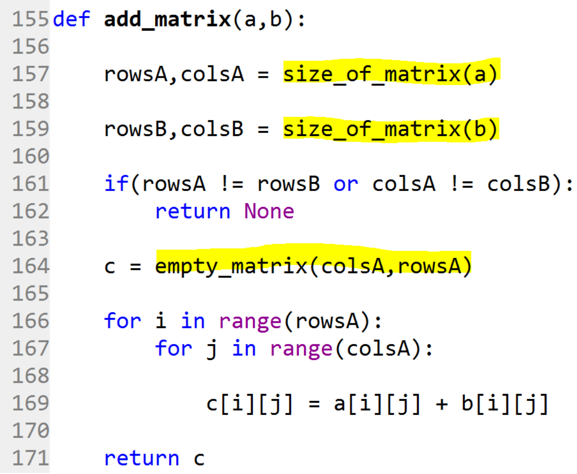


```python
def size_of_matrix(a):
    '''assumes a is a nested list
    with sublist of regular length'''

    rows = len(a)
    cols = len(a[0])

    return rows, cols

def empty_matrix(rows,cols):

    a = [None]*rows
    for i in range(rows):
        a[i] = [None]*cols

    return a
```

Bigger functions



```python
def transpose_matrix(a):

    rowsA,colsA = size_of_matrix(a)

    b = empty_matrix(colsA,rowsA)

    for i in range(rowsA):
        for j in range(colsA):

            b[j][i] = a[i][j]

    return b
```



```python
def add_matrix(a,b):

    rowsA,colsA = size_of_matrix(a)

    rowsB,colsB = size_of_matrix(b)

    if(rowsA != rowsB or colsA != colsB):
        return None

    c = empty_matrix(colsA,rowsA)

    for i in range(rowsA):
        for j in range(colsA):

            c[i][j] = a[i][j] + b[i][j]

    return c
```

Composition of functions

In the following example, the list returned by `add_matrix` is passed to `transpose_matrix`.

```
a = [ [1,2], [3,4] ]

b = [ [5,6], [7,8] ]

d = transpose_matrix(  add_matrix(a,b)  )
```

Tuples

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

```
def simple(a):

    a = 2.0*a
    b = a/0.6

    return a, b

#you probably did this
value1, value2 = simple(10)
print value1
print value2

#let's explore more
values = simple(10)
print type(values)
print values
print values[0]
print values[1]
```

A tuple is like a list. However, a tuple is **immutable**.

You should also know how to use the console to determine the methods that a tuple object has.

Recursion

What is recursion?

When a function calls itself.

What does a recursive function need?

the **base case**: the function only knows how to solve the simplest case

else, the function divides itself into two parts

- one that it knows how to solve
- one that it does not – **this should resemble the original problem**

Recursion thus involves

- calling the function with simpler and simpler versions of the problem
- the series of function calls terminates at the base case.

Factorial Example

Base case: $0! = 1$

General formula: $n! = n \times (n - 1)!$

Fibonacci Series

Base case $a_0 = 0$ and $a_1 = 1$

General formula $a_n = a_{n-1} + a_{n-2}$