```
# - Exercises2.py *- coding: utf-8 -*-
"""
Python has lists. The empty list is []. The following is a list of one
item ["a"] and so is [3]. Here is a list with 3 items ["ball",3.14,-2]. Let's
define a list, I'll call it lis and we'll do things with it to illustrate
accessing items in a list. Execute the following cell with Ctrl-Enter.
"""
#%%
lis = ["a","b","c","d","e","f"]
#%%
"""
Exercise:
Some of the things that we can do with lists. Let's try them together.

lis[0] is the first element of the list. (index is 0)
lis[1] is the second element of the list and so on. (index is 1)
The length of the list is len(lis) and is the number of items in the list.
lis[-1] is the last item in the list.
lis[2:4] will list items 2 and 3 (but not 4)
lis[:4] will list items 0, 1, 2, 3 (but not 4); that is all items up to 4
lis[3:] will list all items starting with item 3.
lis.append("g") will append "g" onto the end of the list
"a" in lis          # running this statement will return True
"r" in lis          # running this statement will return False

Everything in Python is an object, whether it is a variable like x or a list
like lis. Objects have methods indicated by the dot. So .append() is a method
of the list object. We'll see more of this.
"""
"""
Here is an example function using a list. We pass in a list of items and
it checks for certain animals or flowers in the list.
We'll try it out on several lists such as ['bear'], ['daisy', lion'], etc.
"""
#%%

def who_is_there(lis):
    if "bear" in lis:
        print("There's a bear.")
    if "lion" in lis:
        print("There's a lion.")
    if "daisy" in lis or "iris" in lis:
        print("There are flowers.")
    if "daisy" in lis and "iris" in lis:
        print("There are at least two flowers.")
    if "donkey" in lis:
        print("There is a donkey.")
    if "horse" not in lis:
        print("There is no horse in the list.")
    print("The list has",len(lis), "items")
#%%
"""
You should make up some lists and pass to 'who_is_there' to see how the if
statements handle various combinations. Some test lists for who_is_there:
"""
#%%
alion = ['lion']
ld = ['lion','daisy']
lbf = ['lion','bear','iris']
#%%
"""
The following function illustrates using lists in 'for' loops. Note that the
loop variable 'let' steps through the list, alist, taking the value of each of
its items in turn.
```

```
"""
#%%
lis = ["a","b","c","d","e","f"]
lis1 = ["a","b","a","r","c","a","a"]
#%%
def count_a(alist):
    ct = 0
    for let in alist:
        if let == 'a':
            ct = ct + 1
    print("There are",ct,"letter a's in the list.")
#%%
"""
Note there is a basic design pattern to these lists. Some variable for
accumulating the results (above it is ct) is initiated before entering the
loop. This variable is updated within the loop. Afterwards that variable is
used (in this case ct is printed out).
"""

"""
Exercise:
Take the following list, nlis, and compute its average. That is, write
a function 'average(numlis)' that uses a 'for' loop to sum up the numbers
in numlis and divide by the length of numlis. Just to be sure that you
got all the numbers in numlis, print each one in your 'for' loop and
print the length of the the list. When using a loop, one always needs to
be careful that it loops as often as is expected. In this case also print out
the number of items in the list.
Caution: Do NOT use the variable nlis in your function. This function should
work on any list of numbers. Just to be sure make sure that your function
(without any changes) works on rlis as well as nlis.
"""
#%%
nlis = [2,4,8,105,210,-3,47,8,33,1]  # average should by 41.5
rlis = [3.14, 7.26, -4.76, 0, 8.24, 9.1, -100.7, 4] # average is -9.215
#%%
# some tests for your function. Be sure your function works for these
average(nlis)
average(rlis)
#%%
"""
Solution:
"""
#%%
def average(numlis):









#%%
"""
End solution
"""
"""
Let me emphasize that you can make a 'for' loop with just a list.
One can simply step through a list to form the loop.

In this example case it is a list of states and we will simply be stepping
through the loop and printing out the states.
"""
#%%
newEngland = ["Maine","New Hampshire","Vermont", "Rhodes Island",
"Massachusetts","Connecticut"]
```

```
def for_state(slis):
    for state in slis:
        print(state)

#%%
"""
Keep in mind that a 'for' loop can step through various kinds of iterators.
"""
"""
Exercise:
Write a function 'print_list(lis)' that prints items of the list lis. Test it
by running the three tests that I give here. This requires writing a function
that includes a loop like the one above, but uses lis for the iterator. Inside
your function you should use lis to represent the list. If you do so, your
function should pass all three tests below.
"""
#%%
letter_list = ['a', 'b', 'c']
cap_list = ['A', 'B', 'C', 'D']
misc_list = ['ball', 3.14, -50, 'university', "course"]
#%%
"""
Solution:
"""
#%%



#%%
"""
End solution
"""
"""
Lets' talk about data types. For starters Python has integers (e.g., 40), float
or real numbers (e.g., 40.0), string ("hello"), list ( ['a','b','c']), bool
(boolean -- that is, True or False). In Python they are called int, float,
str, list, bool. You can tell what type a variable x is by entering type(x).
Here is an example of several:
"""
#%%
x = 17      # integer
y = 3.14    # float
z = "The Walrus and the Carpenter"  # string
z1 = "30"   # string
z2 = '40'   # string
vowels = ['a','e','i','o','u'] # list of strings
nums = ['1','2','3', '3.14'] # list of strings
phrases = ["â€™Twas brillig, and the slithy toves",
    "Did gyre and gimble in the wabe:"] # list of strings (2 strings, in fact)
r = True    # boolean
s = False   # boolean
#%%
"""
Often you can convert one type to another: int(z1) makes and returns an
integer (30); float(z2) returns a float or real number (40.0); str(y) returns
the string "3.14"; etc.
This is important because z1+z2 is not 70 (it is '3040'); however,
int(z1)+int(z2) is 70. Here is a simple program showing when you might want to
use this technique.
"""
#%%
def multiply():
    numstr1 = input("Enter a number: ")
    numstr2 = input("Enter another number: ")
```

```
    num1 = float(numstr1)
    num2 = float(numstr2)
    print("Their product is ", num1 * num2)
#    print("Won't work: ", numstr1 * numstr2)
#%%
"""
Compare list(range(2,20,3)) and range(2,20,3). The first one is a list and
the second one is what Python calls an iterator. The second one dishes out
the next element in the list each time it is called. This is one of the changes
from Python 2 to Python 3. In Python 2 it was a list and there was a function
xrange() for iterating without building the list. That is gone from Python 3.
Can you think of a reason that using range in Python 2 might not be a good idea
with huge lists?
"""
#%%
print(list(range(2,20,3)))
print(range(2,20,3))
#%%
"""
Caution: Notice that large numbers never include commas. Compare these two
examples. In the second, Python thinks that it is printing 3 numbers not 1.
"""
print(12345678)
print(12,345,678)
#%%
"""
Another caution. The following are Python keywords. They have special meaning
and shouldn't be used as variable names:
and         del     from    not     while
as          elif    global  or      with
assert      else    if      pass    yield
break       except  import  print
class       exec    in      raise
continue    finally is      return
def         for     lambda  try

You'll just get a syntax error:
"""
#%%
except = 5
#%%
"""
Note: for readability, if you feel that you need to use one of these as a
variable, you could use an underscore after it. For example, and_, class_, etc.
That makes it different from the Python keyword.
"""
#%%
newEngland = ["Maine","New Hampshire","Vermont", "Rhodes Island",
"Massachusetts","Connecticut"]

def for_state(state_list):
    for state in state_list:
        print(state)

#%%
"""
Let's print a small report. Here is a list of New England states and
their populations. We'll print this as a table or report. Essentially, this
is like the little function above, except that we need to handle the variables
in a more sophisticated way.
"""
#%%
newEngland = [["Massachusetts",6692824],["Connecticut",3596080],
              ["Maine",1328302],["New Hampshire",1323459],
              ["Rhode Island",1051511],["Vermont",626630]]
```

```
#%%
"""
Exercise:
Before writing the function, let's understand this list of lists better.
Try this out.
What is the first item of newEngland? (i.e., the one of index 0)
What is the second item?
What is the name of the state in the second element? How do we get that?
What is the population of the state in the second element?
"""
"""
Solution:
"""
#%%









#%%
"""
End solution
"""
#%%
newEngland = [["Massachusetts",6692824],["Connecticut",3596080],
              ["Maine",1328302],["New Hampshire",1323459],
              ["Rhode Island",1051511],["Vermont",626630]]

def report1(state_data):
    """ prints population report """
    print("Population          State")
    for state_item in state_data:
        print(state_item[1], "        ", state_item[0])

#%%
"""
Note: that because we pass the list into the function by way of the argument
state_data, the above works on the following mid-Atlantic list. Execute the
following cell to define midAtlantic in IPython and try it:
"""
#%%
midAtlantic = [["New York",19746227],["New Jersey",8938175],
               ["Pennsylvania",12787209]]
#%%
"""
Note that we don't use 19,746,227 as the population of New York. Why?
"""
"""
Another way to do it.
"""
#%%
newEngland = [["Massachusetts",6692824],["Connecticut",3596080],
              ["Maine",1328302],["New Hampshire",1323459],
              ["Rhode Island",1051511],["Vermont",626630]]

def report2(state_data):
    """ prints population report """
    print("Population          State")
    for i in range(0,len(state_data)):
        print(state_data[i][1], "        ", state_data[i][0])

#%%
```

```
"""
Find the sum of the populations of the New England states. Print
out how many there are. Use a basic loop design pattern.
"""
#%%
newEngland = [["Massachusetts",6692824],["Connecticut",3596080],
              ["Maine",1328302],["New Hampshire",1323459],
              ["Rhode Island",1051511],["Vermont",626630]]

def population(state_data):
    """ Sums state populations """
    sum_ = 0
    num_states = len(state_data)
    for i in range(0,num_states):
        one_state = state_data[i]
        pop = one_state[1]
        sum_ = sum_ + pop
    print("The total population of this list of states is",sum_)
    print("There are",num_states,"states in this list of states.")
#%%

"""
Version using more syntactic sugar -- the variables have better and more
meaningful names. This may read better in a bigger program.
"""
#%%
def population(state_data):
    """ Sums state populations """
    population = 1
    sum_ = 0
    num_states  = len(state_data)
    for state in range(0,num_states):
        sum_ = sum_ + state_data[state][population]
    print("The total population of this list of states is",sum_)
    print("There are",num_states,"states in this list of states.")
#%%

"""
Exercise:
Write a function 'average(nlis)' that uses a 'for' loop and 'range()' to sum up
the numbers in nlis and divide by the length of nlis. Just to be sure that you
have used all the numbers in nlis, print each one in your 'for' loop and print
the length of the list. Do not use the variable numlis in your function! If you
change to a different list will it work? For numlis, the output should look
like:

65 44 3 56 48 74 7 97 95 42
the average is 53.1
"""
#%%
numlis = [65, 44, 3, 56, 48, 74, 7, 97, 95, 42]  # test on this list
numlis2 = [4,6,8,12,2,7,19]     # test on a second list to be sure
#%%
"""
Solution Starter:
"""
#%%
def average(nlis):
    pass  # delete this and enter your code starting here
```

```
#%%
"""
End solution
"""
"""
Libraries. Python is a "small" language in the sense that many tools
that are available are not automatically included when you run it. Many of
these tools are in modules called libaries and can be loaded into your program
only when you need them, keeping your programs smaller when they aren't needed.
A typical way of doing that is

import random

which will load the library named random.
"""
#%%
import random
#%%
# Each run of the following gives a different random number between 0 and 1
print(random.random())
#%%
# Each run of the following gives a different random integer between 3 and 8
print(random.randint(3,8))
#%%
"""
The following example builds a sentence using various parts of speech.
It randomly chooses words from a list by using random.choice(), a function
or method imported from a library called random. We have used a method of the
string data type to capitalize the first letter of the sentence.
"""
#%%
import random

verbs=["goes","cooks","shoots","faints","chews","screams"]
nouns=["bear","lion","mother","baby","sister","car","bicycle","book"]
adverbs=["handily","sweetly","sourly","gingerly","forcefully","meekly"]
articles=["a","the","that","this"]

def sentence():
    article = random.choice(articles)
    noun = random.choice(nouns)
    verb = random.choice(verbs)
```

```
    adverb = random.choice(adverbs)

    our_sentence = article + " " + noun + " " + verb + " " + adverb + "."
    our_sentence = our_sentence.capitalize()

    print(our_sentence)
#%%
"""
Exercise:
Adapt this function to write a four line poem. Call it simple_poem().
Essentially you have to write a loop around this so that you get 4 lines.
Remember that the inside or scope of the loop has to be indented 4 spaces.
Note: The Edit menu has a quick way to indent a series of lines. The function
is repeated here for your convenience in modifying it.
"""
"""
Solution (modify the copy below to be your simple_poem function):
"""
#%%
import random

verbs=["are","is","goes","cooks","shoots","faints","chews","screams"]
nouns=["bear","lion","mother","baby","sister","car","bicycle","book"]
adverbs=["handily","sweetly","sourly","gingerly","forcefully","meekly"]
articles=["a","the","that","this"]

def simple_poem():
    article = random.choice(articles)
    noun = random.choice(nouns)
    verb = random.choice(verbs)
    adverb = random.choice(adverbs)

    our_sentence = article + " " + noun + " " + verb + " " + adverb + "."
    our_sentence = our_sentence.capitalize()

    print(our_sentence)
```



```
#%%
"""
End Solution:
"""
"""
Let's look at a couple of loop design patterns.
"""
"""
Example: Add numbers until you get a blank one. This initializes a variable
sum_ and adds to it each time through the loop. Afterwards sum_ is used in a
print statement.
"""
#%%
def add_up():
    sum_ = 0
```

```
    while True:             # will loop forever
        num = int(input("Enter a number, input 0 to quit: "))
        if num == 0:
            break           # breaks out of while loop
        sum_ = sum_ + num
    print(sum_)
#%%
"""
Building lists - recall the .append() method
"""
#%%
baseball = []
baseball.append("ball")
baseball.append("bat")
baseball.append("mitt")
baseball
#%%
"""
Let's write a program to build a list of the numbers. Before we initialized
sum_ to 0. The equivalent for a list is to set it to the empty list. Adding to
the sum has its equivalent in appending to the list.
"""
#%%
def store_up():
    num_lis = []
    while True:
        nextnum = int(input("Enter a number, 0 to quit: "))
        if nextnum == 0:
            break
        num_lis.append(nextnum)
    print(num_lis)
#%%
"""
Exercise:
Write a function diner_waitress() that asks for you order. First start an empty
list, call it order. Then use a while loop and an input() statement to gather
the order. Continue in the while loop until the customer says "that's all".
Onne way to end the loop is to use 'break' to break out of the loop when
"that's all" is entered.
Recall that you can add to a list by using the list's .append() method; suppose
that your list is called order. To create an empty list you can use
order = []. You are going to have to input one food at a time and append it
to the order list.
Then print out the order. Here is my run:

diner_waitress()
Hello, I'll be your waitress. What will you have?

menu item: eggs

menu item: bacon

menu item: toast

menu item: jelly

menu item: that's all
You've ordered:
['eggs', 'bacon', 'toast', 'jelly']

"""
#%%
"""
Solution:
"""
```

#%%

#%%