COMPUTER PROGRAMMING
LEARNING GUIDE 13

LISTS REVIEWED

Remember lists? To create a list, just assign more than one value to a variable, and put the values in square brackets:

```
>>> x = [1,2]
```

To print an individual element in a list:

```
>>>print (x[0])
```

This number with the item's location is called the *index*. Note that list locations start at zero. So a list or array with 10 elements does not have an element in spot [10]. Just spots [0] through [9]. It can be very confusing to create an list of 10 items and then not have an item 10, but most computer languages start counting at 0 rather than 1. Remember that the first element is index 0.

Remember, there are two sets of numbers to consider when working with a list of numbers: the position and the value. The position, also known as index, refers to *where* a value is. The value is the actual number stored at that location. When working with a list or array, make sure to think if you need the *location* or the *value*.

A program can assign new values to an individual element in a list. In the case below, the first spot at location zero (not one) is assigned the number 22:

```
>>>x[0] = 22
>>>print (x)
[22,2]
```

Also, a program can create a “tuple.” This data type works just like a list, but with two differences. First, it is created with parentheses rather than square brackets. Second, it is not possible to change the tuple once created. See this:

```
>>> x = (1, 2)
>>> print(x)
(1, 2)
>>> print(x[0])
1
>>> x[0] = 22
Traceback (most recent call last):
File "<pyshell#18>", line 1, in <module>
x[0] = 22
TypeError: 'tuple' object does not support item assignment
>>>
```

As can be seen from the output of the code above, we can't assign an item in the tuple a new value. Why would we want this limitation? First, the computer can run faster if it knows the value won't change. Second, some lists we don't want to change, such as a list of RGB colors for red. The color red doesn't change, therefore an immutable tuple is a better choice.

If a program needs to iterate through each item in a list, such as to print it out, there are two types of for loops that can do this.

The first method to iterate through each item in a loop is by using a “for-each” loop. This type of loop takes a collection of items, and loops the code once per item. IMPORTANT: It will take a *copy* of the item and store it in a variable for processing.

The format of the command:

for *item_variable* **in** *list_name*

Type in these examples to see how it works:

```
my_list = [101,20,10,50,60]
for item in my_list:
    print(item)

my_list = [“Spoon”,”Fork”,”Knife”]
for item in my_list:
    print(item)
```

Lists can contain other lists:

```
my_list = [ [2,3],[4,3],[6,7] ]
for item in my_list:
    print(item)
```

The other way to iterate through a list is to use an *index variable* and directly access the list rather than through a copy of each item. To use an index variable, the program counts from 0 up to the length of the list. If there are ten elements, the loop must go from 0 to 9 for a total of ten elements.

The length of a list may be found by using the `len` function. Combining that with the `range` function allows the program to loop through the entire list.

```
my_list=[101,20,10,50,60]
    for i in range(len(my_list)):
        print(my_list[i])
```

This method is more complex, but is also more powerful. Because we are working directly with the list elements, <u>rather than a copy</u>, the list can be modified. The `for-each` loop does not allow modification of the original list.

New items may be added to a list (but not a tuple) by using the `append` command. For example:

```
my_list = [2,4,5,6]
print(my_list)
my_list.append(9)
print(my_list)
```

To create a list from scratch, it is necessary to create a blank list and then use the append function. This example creates a list based upon user input:

```
my_list = [] # Empty list
for i in range(5):
    userInput = input( "Enter an integer: ")
    userInput = int( userInput )
    my_list.append(userInput)
    print(my_list)
```

If a program needs to create an array of a specific length, all with the same value, a simple trick is to use the following code:

```
my_list = [0] * 100
```

SUMMING A LIST (ARRAY)

Creating a running total of a list (array) is common. Here’s how to do it:

```
my_list = [5.76,8,5,3,3,56,5,23]
list_total = 0
for i in range(len(my_list)):
    list_total = list_total += my_list[i]

print (list_total)
```

Using a for loop to iterate the array, rather than count through a range:

```
my_list = [5, 76, 8, 5, 3, 3, 56, 5, 23]
list_total = 0
for item in my_list:
    list_total += item

print(list_total)
```

Numbers in an array can be changed by using a for loop:

```
my_list = [5, 76, 8, 5, 3, 3, 56, 5, 23]
for i in range(len(my_list)):
    my_list[i] = my_list[i] * 2

print(my_list)
```

However version 2 *does not work* at doubling the values in an array. Why? Because `item` is a *copy* of an element in the array. The code below doubles the copy, not the original array element.

```
my_list = [5, 76, 8, 5, 3, 3, 56, 5, 23]
for item in my_list:
    item = item * 2

print(my_list)
```

SLICING STRINGS REFRESHER

Strings are actually lists of characters. They can be treated like lists with each letter a separate item.

Exercise: Start with this code:

```
months = "JanFebMarAprMayJunJulAugSepOctNovDec"
n = int(input("Enter a month number: "))
```

Trap input for integers from 1 to 12. Print out the month corresponding to the number the user enters.

SECRET CODES

This code prints out every letter of a string individually:

```
plain_text = "This is a test. ABC abc"
for c in plain_text:
    print(c, end=" ")
```

Computers do not actually store letters of a string in memory; computers store a series of numbers. Each number represents a letter. The system that computers use to translate numbers to letters is called *Unicode*. The full name for the encoding is Universal Character Set Transformation Format 8-bit, usually abbreviated `UTF-8`

The Unicode chart covers the Western alphabet using the numbers 0-127. Each Western letter is represented by one byte of memory. Other alphabets, like Cyrillic, can take multiple bytes to represent each letter. Google a Unicode chart to have a look at the values.

This next set of code converts each of the letters in the prior example to its ordinal value using UTF-8:

```
plain_text = "This is a test. ABC abc"
for c in plain_text:
    print(ord(c), end=" ")
```

This next program takes each UTF-8 value and adds one to it. Then it prints the new UTF-8 value, then converts the value back to a letter:

```
for c in plain_text:
    x = ord(c)
    x = x + 1
    c2 = chr(x)
    print(c2, end="")
```

The next code listing takes each UTF-8 value and adds one to it, then converts the value back to a letter:

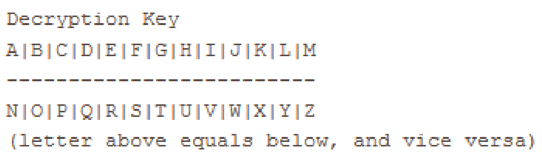
```
plain_text = "This is a test. ABC abc"
encrypted_text = ""
for c in plain_text:
    x = ord(c)
    x = x + 1
    c2 = chr(x)
    encrypted_text = encrypted_text + c2
print(encrypted_text)
```

And, of course, if we *encrypt* something, we need to *decrypt* it:

```
encrypted_text = "Uijt!jt!b!uftu/!BCD!bcd"
plain_text = ""
for c in encrypted_text:
    x = ord(c)
    x = x - 1
    c2 = chr(x)
    plain_text = plain_text + c2
print(plain_text)
```

ASSIGNMENTS

There is a "hide and go seek" game played the world over called "geocaching". People hide things, and enter the GPS co-ordinates on a web page on www.geocaching.com. Other "geocachers" then go out and find the spot. Sometimes, the caches are hard to find and the hider will leave a hint on the geocaching page, but will encrypt it in case people don't want to see the "spoiler". Geocaching.com uses a system called ROT-13. Here is how to encrypt and decrypt the clues.



```
Decryption Key
A|B|C|D|E|F|G|H|I|J|K|L|M
-------------------------
N|O|P|Q|R|S|T|U|V|W|X|Y|Z
(letter above equals below, and vice versa)
```

Write a program that takes user text input and encrypts it using ROT 13. Ensure that the user only enters letters and a space. No other characters are allowed.

Write a program to decrypt user text input as well.

Another assignment on the next page(s)

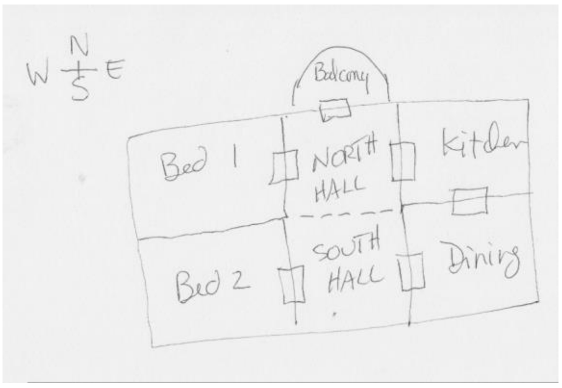
One of the first "adventure" games ever played was a text adventure called Colossal Cave Adventure. You can play the game on-line to get an idea what text adventure games are like. Arguably the most famous of this genre of game is the Zork series, referred to in episodes of "The Big Bang Theory".

It is easy to start an adventure like this. It is also a great way to practice using lists. Our game for this guide will involve a list of rooms that can be navigated by going north, east, south, or west. Each room will be a list with the room description, and then what rooms are in each of the directions. Here's a sample run:

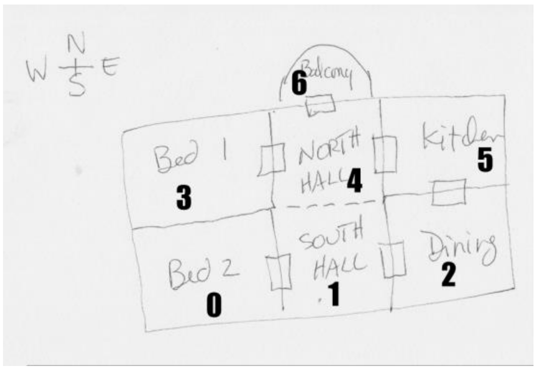
```
You are in a dusty castle room.
Passages lead to the north and south.
What direction? N

You are in the armory.
There is a room off to the south.
What direction? S

You are in a dusty castle room.
Passages lead to the north and south.
What direction? S

You are in a torch-lit hallway.
There are rooms to the east and west.
What direction? E

You are in a bedroom. A window overlooks the castle courtyard.
A hallway is to the west.
What direction? W

You are in a torch-lit hallway.
There are rooms to the east and west.
What direction? W

You are in the kitchen. It looks like a roast is being made for supper.
A hallway is to the east.
What direction? W

Can't go that way.

You are in the kitchen. It looks like a roast is being made for supper.
A hallway is to the east.
What direction?
```

Creating Your Castle

Before you start, sketch out the castle (world) that you want to create. It might look something like this:



Next, number all of the locations starting at zero.



Use the sketch to figure out how all the rooms connect. In my example, room 0 connects room 1 to the east, and no room to the north, south or west. Room 5 connects to room 4 to the east, and room 2 to the south.

Step-by-step Instructions

1. Create an empty array called room_list.
2. Create a variable called room. Set it equal to an array with five elements. For the first element, create a string with a description of your first room. The last four elements will be the number of the next room if the user goes north, east, south, or west. Look at your sketch to see what numbers to use. Use None if no room hooks up in that direction. (Do not put None in quotes. Also, remember that Python is case sensitive so none won't work either. The keyword None is a special value that represents "nothing." Because sometimes you need a value, other than zero, that represents nothing)
3. Append this room to the room list.
4. Repeat the prior two steps for each room you want to create. Just re-use the room variable.
5. Create a variable called current_room. Set it to zero.
6. Print the room_list variable. Run the program. You should see a really long list of every room in your adventure.
7. Adjust your print statement to only print the first room (element zero) in the list. Run the program and confirm you get output similar to:
   ```
   ['You are in a room. There is a passage to the north.', 1, None, None, None]
   ```
8. Using current_room and room_list, print the current room the user is in. Since your first room is zero, the output should be the same as before.
9. Change the print statement so that you only print the description of the room, and not the rooms that hook up to it. Remember if you are printing a list in a list the index goes after the first index. Don't do this: [current_room[0]], do [current_room][0]
   ```
   You are in a room. There is a passage to the north.
   ```
10. Create a variable called done and set it to False. Then put the printing of the room description in a while loop that repeats until done is set to True.
11. Before printing the description, add a code to print a blank line. This will make it visually separate each turn when playing the game.
12. After printing the room description, add a line of code that asks the user what they want to do.
13. Add an if statement to see if the user wants to go north.
14. If the user wants to go north, create a variable called next_room and get it equal to room_list[current_room][1], which should be the number for what room is to the north.
15. Add another if statement to see if the next room is equal to None. If it is, print "You can't go that way." Otherwise set current_room equal to next_room.
16. Test your program. Can you go north to a new room?
17. Add elif statements to handle east, south, and west. Add an else statement to let the user know the program doesn't understand what she typed.
18. Add several rooms, at least five. It may be necessary to draw out the rooms and room numbers to keep everything straight. Test out the game. You can use **\n** if you have a multi-line room description.
19. Optional: Add a quit command. Make sure that the program works for upper and lower case directions. Have the program work if the user types in "north" or "n" or "N" or "NORTH".

Spend a little time to make this game interesting. Don't simply create an "East room" and a "West room." That's boring.

Also spend a little time to double check spelling and grammar. Without a word processor checking your writing, it is important to be careful.

Use **\n** to add carriage returns in your descriptions so they don't print all on one line. Don't put spaces around the **\n**, or the spaces will print.

Some enhancements:. Using all eight cardinal directions (including "NorthWest"), along with "up" and "down" is rather easy. Managing an inventory of objects that can exist in rooms, be picked up, and dropped is also a matter of keeping lists. Some games like this use two word commands, the first is a verb "go", "get","drop","kill" etc, and the second word is the noun "north","sword","bucket","key".

If you enjoy creating this adventure game, you can expand it endlessly and it could represent your final project.