Orange County Community College

Computer Science and Technology Department

csc 138 - Scripting

Laboratory Exercise 4

Name: ____Alex DeGhetto_______________

Date: _____03/15/2023_______________

Objectives:

- Strings and slices
- Understand the function and operation of Lists in Python.
- Create lists of various data types.
- Demonstrate the ability to add and remove list elements.
- Demonstrate knowledge of list slicing.

Indexes (Indices) and Slicing

The individual characters (letters, etc.) that make up a string can be referenced by their position in the sequence, also known as their ***index***. Remember, however, that the computer starts counting at zero rather than one. Consider the following string declaration:

    applicant = 'Ed Norton'

0	1	2	3	4	5	6	7	8
E	d		N	o	r	t	o	n

Note that the character 'E' is placed at index 0 in the string, the character 'd' is placed at index 1, etc. Thus is we issue the statement:

    print(applicant[3])

This command asks the interpreter to print out the character at index 3 in the string applicant. The shell should respond with the letter 'N'. Ensure that this is the case by creating the string and issuing the print statement in your Python shell.

1. Write a Python statement to create a string containing the name 'Ralph Kramden'.

applicant = 'Ralph Kramden'

2. Write a statement that will print out the letter 'm' from the string you created above.

print(applicant[9])

A ***substring*** or ***slice*** is a smaller portion of a string. In the 'applicant' example above, 'Ed' is a substring of the entire name. In Python, we can take a substring by specifying the index we want to start at and the index we wish to end at. We can specify these two indices using the same bracket notation as above, but separating the starting and ending indices with a colon. Thus the statement:

    applicant[4:7]

will produce the substring:

    ort

Note that the ending index in Python is ***exclusive*** – that means that the substring stops *before* the ending index and will *not* include the character *at* that index. Issue the statement above and ensure that it produces the above output.

If we omit the starting index, the slice will start at the beginning of the string (that is, at index zero).

3. At the Python shell, issue the following statement:

    print(applicant[:2])

What is the output of this statement?

(im using Ralph Kramden not Ed Norton)

Ra

Why?

it printed the first 2 indexes, 0 and 1.

If we omit the ending index of a slice, the interpreter will return everything from the starting index to the end of the string

4. At the Python shell, issue the following statement:

```
print( applicant[3:] )
```

What is the output of this statement?

ph Kramden

Why?

It went from the 3rd index until the end.

5. Given the following string variable declaration:

```
word  =  'Antidisestablishmentarianism'
```

What would the following slice statements print?   Make sure to type the above statement and each print statement in to test your responses.

```
print ( word[4:7] )
```

dis

```
print( word[7:16] )
```

establish

```
print( word[16:20] )
```

ment

```
print( word[7:] )
```

establishmentarianism

```
print( word[:20] )
```

Antidisestablishment

```
print( word[2:5] + word[7:9])
```

tides

```
print( word[9:15] )
```

tablis

```
print( word[11:16] )
```

blish

```
print( word[19:24] )
```

6. Given the following string variable declaration:

```
text  =  'The quick brown fox jumps over the lazy dog'
```

Write Python statements that will produce the following slices.   Again, make sure you type the above statement and your responses to check your work.

'quick'

print(text[4:9])

'jump'

print(text[20:24])

'ox'

print(text[17:19])

'he'

print(text[1:3])

'row'

print(text[11:14])

'own'

print(text[12:15])

'lazy dog'

print(text[35:])

'jumps over'

print(text[20:30])

'hazy'

print(text[1] + text[36:40])

7. The split function takes a string and splits it into a list of separate strings based on some delimiting character or string. For example, if we call the split function without an argument, it splits the string based on the space character. Issue the following statement in your Python shell:

```
print(  text.split()  )
```

What is the output of the above statement?

```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
```

8. What happens if we specify a different delimiting character? Issue the following statement:

```
print( text.split('e') )
```

What is the output of the above statement?

['Th', ' quick brown fox jumps ov', 'r th', ' lazy dog']

Why?
The split command created a space wherever there was an "e".

Input
Recall that we can read from the keyboard (stdin) with the ***input*** statement. At the Python shell prompt, issue the statement:

```
stuff = input('Type a word: ')
```

Note that the string inside the input function call (the argument) is ***prompt***. The *prompt* is printed out to the monitor by the interpreter in order to instruct the user what to input. The interpreter does not return to the normal Python shell (>>>), but now waits for the user to type in something.

9. At the prompt from the above statement, type in some text and hit the return (enter) key. Python will store this text in the variable *stuff* for further processing. What did you type?

i typed "word"

10. At the Python shell, issue the statement:

```
print( stuff )
```

What is returned by the Python shell?
word

11. The input statement always reads in data from the keyboard as a string. Issue the statement:

```
type(stuff)
```

What type is returned?
i did print(type(stuff))
<class 'str'> was my output

12. Note that this is true even if you type in numeric values. Again, issue the statement:

```
stuff = input('Type a word: ')
```

When prompted this time, enter the value 55 and press enter. Then issue the statement:

```
type(stuff)
```

What type is returned by the Python shell now?

still <class 'str'>

13. In order to get a numeric value, we must typecast the value as numeric. Issue the statement:

```
num = int( stuff )
```

Then issue the statement:

```
print( num )
```

What is returned by the print statement?
55

Now issue the statement:

```
type( num )
```

What data type is num?

an int instead of a str.
<class 'int'>

Encoding

14. Do some research on the internet regarding different encoding for Python. Make sure you visit the site:

https://docs.python.org/3/howto/unicode.html

Write a short report on encodings, why they are necessary, and what some of the options are and attach it to this lab sheet.

They are necessary because it gives a universal meaning and numeric value for all languages characters and alphabet along with emojis, etc.
UTC-8 (most commonly used)
UTC-16
UTC-32

Lists

Index	Value
0	sparrow
1	penguin
2	stork
3	canary
4	hawk

15. A ***List*** is a sequence of elements, each of which is identified by a numeric index. To create a list, we simply enumerate the list elements and enclose them in square brackets. For example, issue the following statement at the Python shell:

```
birds  =  [ ‘sparrow’, ‘penguin’, ‘stork’, ‘canary’, ‘hawk’ ]
```

Then issue the statement:

```
print( birds )
```

What does the Python interpreter return?

['sparrow', 'penguin', 'stork', 'canary', 'hawk']

16. To select a particular value from the list, we must know its position in the list, also known as its ***index***. For example, the value ‘sparrow’, in the first position on the list, lies at index 0 (recall that the computer begins counting at zero). To access a single element like ‘sparrow’, we use the following syntax:

list_name[*index*]

Thus to access the element ‘sparrow’, we would use the syntax:

birds[0]   # access the 0th element of the list birds

At the Python interpreter, issue the statement:

```
print( birds[0] )
```

What does the Python interpreter return?
sparrow

17. Below, write a Python statement that will print the value ‘stork’ from the list *birds*.
print(birds[2])

18. Using the list *birds* and string concatenation, create a string variable that contains the following text:

hawks eat sparrows

birds = ['sparrow', 'penguin', 'stork', 'canary','hawks eat sparrows', 'hawk']

19. To find out how many elements are in a list, we can use the *len* function. For example, issue the statement:

```
len(birds)
```

What does the Python interpreter return? Why?

i did print(len(birds))
i got 6

20. Issue the statement:

```
len( birds[1] )
```

What does the Python interpreter return now? Why? How does this statement differ from that of # 19 above?
I got 7, which is how many characters are in penguin which is in index 1.
21. There are two ways to add values to a list. One is to place new values at the end of the list. In Python, we call this ***appending*** values. At the Python shell prompt, issue the statement:

```
birds.append( 'ostrich' )
```

then issue a command to print the list. What command did you issue:
print(birds)
What is the output of the command?
['sparrow', 'penguin', 'stork', 'canary', 'hawks eat sparrows', 'hawk', 'ostrich']
Where was the new value placed?
At the end.
22. Issue a command to add the value 'hummingbird' to the list. Write your command below:
i did birds.append('hummingbird')

23. Below, write a command to return the new length of the list:
print(len(birds))
24. What is the current list length?
8
25. What is the index of the value 'hummingbird' that you just placed in the list?
7
26. Write a Python statement to print out just the value 'ostrich' from the list:
print(birds[6])

27. We can also add values to the middle of the list. However, we must then specify the exact location (or index) at which we wish to place the new element. In Python, the command to add values in a particular location in a list is ***insert***. The generic syntax of the ***insert*** statement is:

list_name.**insert**(*index*, *value*)

At the Python prompt, issue the statement:

```
birds.insert(3, 'condor')
```

Then issue the statement:

```
print( birds )
```

Below, write the values that are returned. In addition, above each printed element, write the index of that element.
['sparrow', 'penguin', 'stork', 'condor', 'canary', 'hawks eat sparrows', 'hawk', 'ostrich', 'hummingbird']
What is the index of the new element 'condor'?
3
What happened to the elements after the new element 'condor'? Have their indices changed in any way?
They each moved down one, so each of their indexes changed by 1
28. Write and execute a statement to add the value 'eagle' after the element 'canary'. Write your statement below:
birds.insert(5, 'eagle')
29. What is the length of the list now? How do you know?
10, i did print(len(birds))
30. Again, write a statement to print out the element 'ostrich' from the list:
print(birds[8])
31. There is also a command to remove elements from a list. The syntax of the command is:

del *list_name*[*index*]

For example, at the Python prompt, issue the command:

```
del  birds[2]
```

and then print the list. What are the contents of the list now:
['sparrow', 'penguin', 'condor', 'canary', 'eagle', 'hawks eat sparrows', 'hawk', 'ostrich', 'hummingbird']
32 Which element of the list was deleted? What happened to the indices of all the subsequent elements?
stork
33. What is the length of the list now? How do you know?
9, i did print(len(birds))

34. Again, issue a statement to print out the element 'ostrich' from the list:
print(birds[7])
35. Write a statement that will delete the element 'hawk' from the list:
del birds[7]

36. We can take ***slices*** from lists in the same manner that we used for strings. To take a ***slice*** from a list, use the following syntax:

list_name[*starting_index* : *ending_index*]

Recall that the ending_index is exclusive – meaning that we stop at the index before that index. Print out the elements of the list birds below and write the index of each element above it:

```
0                      1       2                3            4            5                           6            7
['sparrow', 'penguin', 'condor', 'canary', 'eagle', 'hawks eat sparrows', 'ostrich', 'hummingbird']
```

37. At the Python shell, issue the statement:

birds[2: 5]

What is the output of this command?

['condor', 'canary', 'eagle']

38. Write a Python statement that takes a slice of the *birds* list containing the elements:

[‘penguin’, ‘condor’]

Give your statement below:

print(birds[1:3])

39. At the Python shell, issue the statement:

birds[4:]

What is returned by the shell? Why (that is, what happens when we leave off the ending index)?

['eagle', 'hawks eat sparrows', 'ostrich', 'hummingbird']

It went from the 4th index until the end.

40. At the Python shell, issue the statement:

birds[: 3]

What is returned by the shell? Why (that is, what happens when we leave out the starting index)?

['sparrow', 'penguin', 'condor']

it went from the beginning and ended at the 3rd index (did not include it)

41. The ***sorted*** function returns a copy of the list in lexicographical order. At the Python shell, issue the statement:

sorted(*birds*)

What is returned by the interpreter?

['canary', 'condor', 'eagle', 'hawks eat sparrows', 'hummingbird', 'ostrich', 'penguin', 'sparrow']

42. Again issue the statement:

print(birds)

Is the original list in sorted order?

No

43. We can sort a list in place using the ***sort*** function. At the Python shell prompt, issue the statement:

birds.sort()

Print your list and write the elements below:

['canary', 'condor', 'eagle', 'hawks eat sparrows', 'hummingbird', 'ostrich', 'penguin', 'sparrow']

Is the original list in sorted order?

Yes

44. Add the element ‘Owl’ to the end of your list (note that the first letter is capitalized). What statement did you use to do so?

birds.append(‘Owl’)

45. Again issue the statement:

birds.sort()

and print your list. Where was the new ‘Owl’ element placed in the sorted list? Why?

['Owl', 'canary', 'condor', 'eagle', 'hawks eat sparrows', 'hummingbird', 'ostrich', 'penguin', 'sparrow']

In the beginning because it is capitalized.

46. We can also ***concatenate*** lists. At the Python prompt, create the following lists:

shapes = [‘square’, ‘triangle’, ‘circle’]
fruits = [‘orange’, ‘pear’, ‘apple’]

Then issue the statement:

stuff = shapes + fruits

Print out the list *stuff* and write its contents below:

['square', 'triangle', 'circle', 'orange', 'pear', 'apple']

47. What is the length of new list *stuff*? How do you know?

6 because theres 6 “items” aswell as i did print(len(stuff))

48. Write Python statements that print out the element ‘pear’ from the *fruits* list as well as the *stuff* list.

print(stuff[4])
49. Write a Python statement that will create the slice [‘circle’, ‘orange’, ‘pear’] from the list *stuff*.
print(stuff[2:5])
50. Write a statement that will print the elements of the list stuff in sorted order:
print(sorted(stuff))
51. We can check whether or not an element is present in the list with the *in* operator. For example, at the Python shell prompt, issue the statement:

```
‘circle’  in  stuff
```

What does the interpreter return?
true
52. At the Python prompt, issue the statement:

```
‘circle’ in fruits
```

What does the interpreter return?
false
Numeric Lists
53. We can also create numeric lists with the ***range*** function. For example, at the Python shell prompt, issue the statement:

```
list( range( 1,5 ) )
```

What is returned by the interpreter?
[1, 2, 3, 4]
54. The generic syntax for the range command is:
range(*start*, *end*)
Recall that the *end* is *exclusive* in Python. This means that the function will generate a set of numbers up to, but not including, the end value. For example, consider the following statement:

```
list( range( 3, 7 ) )
```

What values will the list created by this statement contain?
Numeric
55. Type in and execute the statement in your Python shell. What list does it return?
[3, 4, 5, 6]
56. Below, write a Python statement that creates a numeric list that contains the numbers 6 through 11:
print(list(range(6, 12)))
57. Python has several numeric functions that take a list of values as input. At the Python shell prompt, create the following list:

```
vals  =  [ 2, 4, 6, 8, 10 ]
```

Then issue the statement:

```
min( vals )
```

What is returned by the interpreter?
2
58. At the Python shell prompt, issue the command:

```
max( vals )
```

What is returned by the interpreter?
10
59. There is also a function to take the sum of the values in the list. At the Python shell prompt, issue the statement:

```
sum( vals )
```

What is returned by the interpreter?
30
60. Using the above functions, write a Python statement that will take the average of the numbers in the list named *val*. Type in your statement below and execute it:

```
avg = sum(vals)/6
print(avg)
```

What value is returned?
5.0
61. Do you think these functions will work with lists of strings? Why or why not?
to an extent. maybe adding but not dividing ect.
62. At the Python shell prompt, issue the command:

```
sum( shapes )
```

What does the shell return?
TypeError: unsupported operand type(s) for +: 'int' and 'str'
63. At the Python shell prompt issue the command:

```
min( shapes )
```

Does this function work on strings? Why do you think this is the case?
Yes, because its using its indexes.
64. Write a Python statement that creates a list with strings naming at least 6 different types of pie:

```
pies = ['apple', 'cherry', 'blueberry', 'strawberry', 'grape', 'pumpkin' ]
print(pies)
```

65. Write a Python statement that puts an additional element at the end of the list:
```
pies.append(‘cranberry’)
```

66. Write a Python statement that inserts another element in the third position in the list.
```
pies.insert(3, ‘banana’)
```
67. Write a Python statement that deletes the element at index 2.
```
del pies[2]
```
68. Write a Python statement that takes the length of your list:
```
print(len(pies))
```
69. Write a Python statement that takes a slice containing the elements at indexes 2, 3, and 4.
```
print(pies[2:5])
```
70. Write a Python statement that will sort your list:
```
print(sorted(pies))
```