Lists and Strings

Many interesting problems involve manipulating sequences of data. You've learned about lists and strings before, but this activity provides a more in-depth look at what they can do.

Content Learning Objectives

After completing this activity, students should be able to:

- Name four methods that lists provide, and describe what each method does.
- Explain the syntax and meaning of slice operations, with and without indexes.
- Name four methods that strings provide, and describe what each method does.

Process Skill Goals

During the activity, students should make progress toward:

- Gaining insight about data structures from many examples. (Information Processing)

Facilitation Notes

This activity is slightly longer than usual, so it's important that the managers keep track of time and make sure teams don't fall behind. Remind students not to give too much thought to the shell output when completing the tables in each model.

Questions #5 and #6 in Model 1 are good for reporting out. Call on at least 2–3 teams to share different perspectives on these answers. Ask teams to explain what else they learned about these functions, if time permits.

When reporting out Model 2, make a connection between the slice `[m:n]` and the `range(m, n)`. In both cases, the range includes `m` but not `n`. Briefly explain that slices may have a third argument, just like the `range` function. Just for fun, show an example of reversing a list or string using the slice `[::-1]`.

On Model 3, make sure students notice the type of `dna` vs the type of `dna[0]`. When discussing question #20, challenge students to describe (hypothetically) what a `replace` method might look like for lists, and how it would be different than `replace` for strings. Report out for other questions as time permits.

Model 1 Working with Lists

Recall that a variable can hold multiple values in the form of a list. The values are separated by commas and wrapped in square brackets.

Lists have ***methods*** (built-in functions) that can be called using dot notation. For example, to add a new element to the end of a list, we can use the `append` method.

Python code	Shell output
`rolls = [4, 6, 6, 2, 6]`	
`len(rolls)`	5
`print(rolls[5])`	IndexError: list index out of range
`rolls.append(1)`	
`print(rolls)`	[4, 6, 6, 2, 6, 1]
`print(rolls[5])`	1
`lucky.append(1)`	NameError: name 'lucky' is not defined
`lucky = []`	
`print(lucky[0])`	IndexError: list index out of range
`lucky.append(5)`	
`print(lucky)`	[5]
`print(lucky[0])`	5
`rolls.count(6)`	3
`rolls.remove(6)`	
`print(rolls)`	[4, 6, 2, 6, 1]
`help(rolls.remove)`	remove first occurrence of value
`help(rolls)`	Help on list object (multiple pages)

Questions (15 min) Start time: _____

1. What is the result of calling the `append` method on a list?

The value gets added to the end of the list. Nothing is returned.

2. What must be defined prior to using a method like `append`?

The list itself; `lucky.append(5)` is an error if `lucky` is not defined.

3. Explain why two lines in Model 1 caused an `IndexError`.

In both cases, we asked for an index that was out of range. If the length of an index is n, the highest index is $n - 1$.

4. What is the result of calling the `remove` method on a list?

It removes the first occurrence of a value. The list changes as a result of this method.

5. Based on the `help` output, name several list methods not shown in Model 1. Do not include methods that begin and end with two underscores (e.g., __add__).

Answers may include: clear, copy, extend, index, insert, pop, reverse, sort.

6. Give one example of a list method that requires an argument and one that does not.

Methods that require arguments: append, count, extend, index, insert, remove. Methods that do not: clear, copy, pop, reverse, sort.

7. Describe the similarities and differences between using a list method like `append` and Python built-in functions like `print`.

Both use parentheses and take arguments. The list methods come after the dot operator, and the built-it functions surround the list itself.

8. Complete the function below (two lines are missing). It should prompt the user for numbers and build a list by adding one number at a time to the end of the list. The loop terminates when the user inputs the number 0.

```
def input_numbers():
    x = 1
    numbers = []
    _____________________
    while x != 0:
        x = int(input("Enter the next number: "))
        numbers.append(x)
        ___________________________________
    return numbers
```

Model 2 Indexing and Slicing

A string is a sequence of characters in single quotes (') or double quotes ("). Depending on the application, we can treat a string as a single value (e.g., `dna`), or we can access individual characters using square brackets (e.g., `dna[0]`). We can also use ***slice notation*** (e.g., `dna[4:8]`) to refer to a range of characters. In fact, all types of sequences (including `list` and `tuple`) support indexing and slicing.

Python code	Shell output
`dna = 'CTGACGACTT'`	
`dna[5]`	'G'
`dna[10]`	IndexError: string index out of range
`len(dna)`	10
`dna[:5]`	'CTGAC'
`dna[5:]`	'GACTT'
`dna[5:10]`	'GACTT'
`triplet = dna[2:5]`	
`print(triplet)`	GAC
`dna[-5]`	'G'
`dna[-10]`	'C'
`dna[:-5]`	'CTGAC'
`dna[-5:]`	'GACTT'
`triplet = dna[-4:-1]`	
`print(triplet)`	'ACT'

Questions (15 min) Start time: _____

9. What is the *positive* index of each character in the `dna` string? Check your answers above.

Character:	C	T	G	A	C	G	A	C	T	T
Index:	0	1	2	3	4	5	6	7	8	9

10. What is the *negative* index of each character in the `dna` string? Check your answers above.

Character:	C	T	G	A	C	G	A	C	T	T
Index:	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

11. Based on the previous questions, what are `dna[2]` and `dna[-2]`? Explain your answers.

They are G and T, respectively. Index 2 means to the third from the left, and index -2 means the second from the right.

12. Explain the IndexError you observed. What is the range of indexes for the `dna` string?

Because the length of the string is 10, the indexes range from 0 to 9. Therefore, `dna[10]` is out of range.

13. Consider the notation of the operator `[m:n]` for slicing the string.

a) Is the value at `m` the same as the corresponding index value (i.e., `dna[m]`)? If not, describe what it means. Yes; `m` is the first character in the slice.

b) Is the value at `n` the same as the corresponding index value (i.e., `dna[n]`)? If not, describe what it means. No; `n` is the index after the last character.

c) Explain what it means when only a single number is referenced when creating a slice, such as `[m:]` or `[:n]`. The slice `[m:]` means "from the index `m` to the end". The slice `[:n]` means "from the beginning to the index just before `n`" (i.e., the first `n` characters).

14. What is the simplest way to get the first three characters of `dna`? What is the simplest way to get the last three characters?

Based on the previous question, we know that `dna[:3]` gets the first three characters. To get the last three, we use `dna[-3:]`.

15. Write a Python expression that slices `'GACT'` from `dna` using positive indexes. Then write another expression that slices the same string using negative indexes.

```
dna[5:9]      dna[-5:-1]
```

16. Write a Python assignment statement that uses the `len` function to assign the last letter of `dna` to the variable `last`.

```
last = dna[len(dna) - 1]
```

17. Write a Python assignment statement that uses a negative index to assign the last letter of `dna` to the variable `last`.

```
last = dna[-1]
```

Model 3 Common String Methods

Like lists, strings have ***methods*** (built-in functions) that can be called using dot notation. See https://docs.python.org/3/library/stdtypes.html#string-methods for more details.

Python code	Shell output
`dna = 'CTGACGACTT'`	
`dna.lower()`	'ctgacgactt'
`print(dna)`	CTGACGACTT
`lowercase = dna.lower()`	
`print(lowercase)`	ctgacgactt
`dnalist = list(dna)`	
`print(dnalist)`	['C', 'T', 'G', 'A', 'C', 'G', 'A', 'C', 'T', 'T']
`dnalist.reverse()`	
`print(dnalist)`	['T', 'T', 'C', 'A', 'G', 'C', 'A', 'G', 'T', 'C']
`type(dna)`	<class 'str'>
`dna = dna.split('A')`	
`print(dna)`	['CTG', 'CG', 'CTT']
`type(dna)`	<class 'list'>
`dna.replace('C', 'g')`	AttributeError: 'list' object has no attribute 'replace'
`print(dna[0])`	CTG
`type(dna[0])`	<class 'str'>
`dna[0].replace('C', 'g')`	'gTG'
`print(dna)`	['CTG', 'CG', 'CTT']

Questions (15 min) Start time: _____

18. Does the `lower` method change the contents of the `dna` string? Justify your answer.

No, it does not. The next line of code prints `dna`, which is unchanged.

19. Describe the `list` function—what does `list(dna)` return in Model 3?

It returns a list of the individual characters. Each element of the list is a string of length 1. (Note that Python does not have a character data type.)

20. Why is it possible to call the `replace` method on `dna[0]` but not `dna`?

The `list` data type does not include a replace method. However, strings allow you to "find and replace" any text.

21. Name several other string methods not shown in Model 3. (Read the documentation.)

There are dozens of string methods; the model only uses lower, split, and replace.

22. Consider the application of a method on a variable:

a) Does a string variable change after applying a method? Provide justification.

No it doesn't; neither `lower` nor `replace` modify the string.

b) Does a list variable change after applying a method? Provide justification.

It might; for example, the `reverse` method changes the list.

c) Identify the data type that is ***immutable*** (i.e., the value never changes).

String

23. Write a single statement to change the final contents of `dna` to `['CTG', 'cc', 'CTT']`. Confirm that your code works in a Python Shell.

```
dna[1] = 'cc'
```

24. Why do you think Python has a `replace` method for strings but not for lists?

Answers may vary. One reason might be that lists are more complex than strings: they can store any type of data, not just characters. Another reason might be that there are fewer applications of replacing data in lists than patterns in text.