Sequences

Warm-up

Last class, we looked at data types, built-in functions and errors. Test your knowledge by answering the following questions:

Questions (10 min) Start time:

1. Write the corresponding output for each statement assuming that they are executed in order. Also, indicate the data type next to your answer. If an error occurs, write what type of error. Place an asterisk (*) next to any output for which you are unsure.

Python code	Output
`print(x)`	
`x = abs(-3.14)`	
`print(x)`	
`y = float("3.761")`	
`print(y)`	
`z = int(y)`	
`print(z)`	
`y = int("1.14")`	
`print(round(3.76))`	
`print(round(3.76,1))`	
`print(round(3.76,2))`	
`print(max(3,4,-5))`	
`print(max(-4.3,2.1))`	
`print(min(3,4,-5))`	
`print(len("3.14"))`	
`print(len(3.14))`	

2. What is the difference between the `int` function and the `round` function?

Model 1 Lists

So far we learned about variable data types that can only hold one value, namely, `int` and `float`. A variable can hold multiple values in the form of a ***list***. The values are separated by commas and wrapped in square brackets. For example:

```
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

Each ***element*** of the list can be referenced by an ***index***, which is the sequential position starting at 0. For example, `primes[4]` is 11.

index	0	1	2	3	4	5	6	7	8	9
value	2	3	5	7	11	13	17	19	23	29

Questions (10 min) Start time:

3. Evaluate each line of code and write down its corresponding output. If an error occurs, write what type of error. Place an asterisk (*) next to any output for which you are not sure.

Python code	Output
`odd = [1, 3, 5, 7]`	
`odd`	
`odd[2]`	
`odd[4]`	
`len(odd)`	
`number = odd[1]`	
`number`	
`odd[1] = 2`	
`odd`	
`number`	

4. What is the index of the second element of `primes`? What is the value at that index?

5. How does the index number compare to the position of the element?

6. How did you reference the value of the 3rd element of `odd`?

7. What did the output of the len() function tell you about the list?

8. One of the lines displays an error. Explain the reason for the error.

9. Write a statement that assigns a list of three integers to the variable `run`.

10. Write a statement that assigns the value 100 to the last element of `run`.

11. Write a statement that assigns the first value of `run` to a variable named `first`.

Model 2 Sequences

Strings and lists are examples of ***sequence*** types. The word "sequence" is a generic term for an ordered set of data.

Questions (10 min) Start time:

12. Complete the table below to explore how sequences work:

Python code	Output
`seq1 = "one two"`	
`type(seq1)`	
`len(seq1)`	
`seq1[1]`	
`seq1[1] = '1'`	
`seq3 = ["one", "two"]`	
`type(seq3)`	
`seq3[1]`	
`seq3[1] = 1`	
`print(seq3)`	
`number = 12345`	
`number[3]`	

13. How does a sequence type differ from a number? (See the last row of the table.)

14. Which sequence type allows their elements to be changed? Which does not?

15. Is it possible to store values of different types in a sequence? If yes, give an example from the table; if no, explain why not.

Model 3 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`) support indexing and slicing.

Questions (15 min) Start time:

16. Complete the table below to further explore how strings work:

Python code	Output
`dna = 'CTGACGACTT'`	
`dna[5]`	
`dna[10]`	
`len(dna)`	
`dna[:5]`	
`dna[5:]`	
`dna[5:10]`	
`triplet = dna[2:5]`	
`print(triplet)`	
`dna[-5]`	
`dna[-10]`	
`dna[:-5]`	
`dna[-5:]`	
`triplet = dna[-4:-1]`	
`print(triplet)`	

17. 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:										

18. 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:										

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

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

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

a) Is the value at the start of the resulting string the same as the value at index `m` (i.e., `dna[m]`)? If not, describe what it is.

b) Is the value at the end of the resulting string the same as the value at index `n` (i.e., `dna[n]`)? If not, describe what it is.

c) Explain what it means when only a single number is referenced when creating a slice, such as `[m:]` or `[:n]`.

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

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

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

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

Model 4 Working with Lists

Lists have ***methods*** (like 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. See https://docs.python.org/3/tutorial/datastructures.html#more-on-lists for more details. The back of the handout also has a list of select functions.

Python code	Output
`rolls = [4, 6, 6, 2, 6]`	
`len(rolls)`	
`print(rolls[5])`	
`rolls.append(1)`	
`print(rolls)`	
`print(rolls[5])`	
`lucky.append(1)`	
`lucky = []`	
`print(lucky[0])`	
`lucky.append(5)`	
`print(lucky)`	
`print(lucky[0])`	
`rolls.count(6)`	
`rolls.remove(6)`	
`print(rolls)`	
`help(rolls.remove)`	
`help(rolls)`	

Questions (15 min) Start time:

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

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

28. Explain why two lines of code caused an `IndexError`.

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

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

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

Model 5 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. The back of the handout also has a list of select functions.

Python code	Output
`dna = 'CTGACGACTT'`	
`dna.lower()`	
`print(dna)`	
`lowercase = dna.lower()`	
`print(lowercase)`	
`dnalist = list(dna)`	
`print(dnalist)`	
`dnalist.reverse()`	
`print(dnalist)`	
`type(dna)`	
`dna = dna.split('A')`	
`print(dna)`	
`type(dna)`	
`dna.replace('C', 'g')`	
`print(dna[0])`	
`type(dna[0])`	
`dna[0].replace('C', 'g')`	
`print(dna)`	

Questions (15 min) Start time:

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

33. Describe the `list` function—what does `list(dna)` return?

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

35. Name several other string methods not shown in the provided code. (Read the documentation.)

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

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

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

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

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

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

List methods

- `append(item)` — Adds a new item to the end of a list
- `insert(position, item)` — Inserts a new item at the position given
- `extend(lst)` — Extend the list by appending all the items from lst
- `pop()` — Removes and returns the last item
- `pop(position)` — Removes and returns the item at position
- `sort()` — Modifies a list to be sorted
- `reverse()` — Modifies a list to be in reverse order
- `index(item)` — Returns the position of first occurrence of item
- `count(item)` — Returns the number of occurrences of item
- `remove(item)` — Removes the first occurrence of item
- `copy()` — Return a clone of the list
- `clear()` — Remove all items from the list

String methods

- `upper()` — Returns a string in all uppercase
- `lower()` — Returns a string in all lowercase
- `capitalize()` — Returns a string with first character capitalized, the rest lower
- `strip()` — Returns a string with the leading and trailing whitespace removed
- `lstrip()` — Returns a string with the leading whitespace removed
- `rstrip()` — Returns a string with the trailing whitespace removed
- `count(item)` — Returns the number of occurrences of item
- `replace(old, new)` — Replaces all occurrences of old substring with new

- `center(width)` — Returns a string centered in a field of width spaces
- `ljust(width)` — Returns a string left justified in a field of width spaces
- `rjust(width)` — Returns a string right justified in a field of width spaces
- `find(item)` — Returns the leftmost index where the substring item is found, or -1 if not found
- `rfind(item)` — Returns the rightmost index where the substring item is found, or -1 if not found
- `index(item)` — Like find except causes a runtime error if item is not found
- `rindex(item)` — Like rfind except causes a runtime error if item is not found
- `split(`*`separator`*`)` — Return a list of the words in the string, using (optional) separator as the delimiter string
- `join(lst)` — Return a string which is the concatenation of the strings in lst
- `isalpha()` — Return True if all characters in the string are alphabetic and there is at least one character
- `isdigit()` — Return True if all characters in the string are decimal characters and there is at least one character
- `islower()` — Return True if all cased characters in the string are lowercase and there is at least one cased character
- `isspace()` — Return True if there are only whitespace characters in the string and there is at least one character
- `isupper()` — Return True if all cased characters in the string are uppercase and there is at least one cased character