Sequences II

Warm-up

Last class, we looked at `for`-loops. Test your knowledge by answering the following questions:

1. Write a function called `get_int` that takes in a list of digits, calculates and returns the number corresponding to the digits in the list. For example, `get_int([3,7,1])` must return `371`.

```
def get_int(digits: list) -> int:
    number=0
    for digit in digits:
      number = number * 10 + digit
    return number
```

2. Provide testcases that you would use to test the `get_int` function.

```
get_int([]) must return 0 and get_int([1]) must return 1
get_int([1,2]) must return 12 and get_int([2,1]) must return 21
get_int([0,0,1]) must return 1 and get_int([1,0,0]) must return 100
```

3. Write a function called `draw_pyramid` that takes in an integer and draws a pyramid of the size equal to the integer parameter. For example, for `draw_pyramid(1)` the function must print:

```
 *
* *
```

For `draw_pyramid(2)` the function must print:

```
  *
 * *
*   *
```

For `draw_pyramid(3)` the function must print:

```
   *
  * *
 *   *
*     *
```

```
def draw_pyramid(size: int) -> None:
    print(' '*size + '*')
    for i in range(1,size+1):
        print(' '*(size-i) + '*' + ' '*(2*i-1) + '*')
```

Model 1 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.

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

Python code	Output
`dna = 'CTGACGACTT'`	
`dna[5]`	'G'
`dna[10]`	IndexError: 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'

5. 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

6. 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

7. 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.

8. 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.

9. 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. Yes; `m` is the first character in the slice.

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. 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).

10. 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:]`.

11. 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]
```

12. 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]
```

13. 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 2 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 with a novice friendly documentation.

Python code	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)

14. 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.

15. 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.

16. Explain why two lines of code 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 $n - 1$.

17. 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.

18. 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.

19. 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.

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

Python code	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 'replace'
`print(dna[0])`	CTG
`type(dna[0])`	<class 'str'>
`dna[0].replace('C', 'g')`	'gTG'
`print(dna)`	['CTG', 'CG', 'CTT']

20. 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.

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

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.)

22. 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.

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

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

24. 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

25. 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'
```

26. 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.

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