CSCI 145 · Quiz 5 Reference Sheet (Lectures 9–10)

LISTS (SLICING, +, AND IN WORK ON STRINGS TOO)

`seq[i]`	Item at index i; first is 0, last is `-1`
`seq[a:b]`	New sequence from index a up to, not including, b
`seq[1:]`	Everything but the first item
`seq[:-1]`	Everything but the last item
`a + b`	New sequence: items of a, then items of b
`x in seq`	`True` if x is in seq. Otherwise, `False`

LIST METHODS (CHANGE THE LIST)

Call	What it does	Returns
`items.append(x)`	Adds x to the end	None
`items.pop()`	Removes the last item	the item
`items.pop(i)`	Removes item at index i	the item
`items.sort()`	Sorts, smallest first	None
`items.reverse()`	Reverses the order	None

`sorted(seq)`	A **new** list of the items, smallest first; seq doesn't change

TWO NAMES, ONE LIST

- `b = a` does **not** copy the list: a and b are two names (**aliases**) for one list
- A change made to the list through either name shows up through both
- Assigning a new list, `b = [...]`, makes b name a new list; a still names the old one
- A list passed to a function is another name for the same list: if the function changes it, the caller sees the change

Copies: `a[:]`, `a.copy()`, and `list(a)` each make a new list with the same items

MAP, FILTER, AND REDUCE

Pattern	What it does	Result
Map	Changes every item	a new list
Filter	Keeps some of the items	a new list
Reduce	Combines the items into one value	one value

LIST COMPREHENSIONS

```
[expression for variable in sequence]
[expression for variable in sequence if condition]
```

The second form builds the same list as this loop:

```
result = []
for variable in sequence:
    if condition:
        result.append(expression)
```

- `if` keeps only the items that pass (filter)
- The expression gives each new item (map)
- A comprehension can map and filter, but not reduce

RECURSION

A **recursive** function calls itself on a smaller version of the problem.

- **Base case:** handled directly, with no recursive call. A function can have more than one
- **Recursive case:** calls the function on an input closer to the base case
- Each call gets its own frame on the **call stack**, with its own variables

WHAT "SMALLER" MEANS

Input	Smaller input	Typical base case
`a number n`	`n - 1`	`n == 0`
`a string word`	`word[1:]`	`word == ""` or `len(word) < 2`
`a list nums`	`nums[1:]`	`len(nums) == 0`

RECURSION THAT RETURNS AN ANSWER

```
def f(nums):
    if len(nums) == 0:
        return answer for an empty list
    elif nums[0] settles it:
        return that answer
    else:
        return f(nums[1:])
```

- The recursive case **returns** whatever the smaller call returns
- Without that `return`, the answer is lost and the function returns None
- **Writing one:** decide the base cases first, then trust the recursive call

WHEN THE CALLS NEVER STOP

- With no base case, or a step that skips past it (e.g. `n - 2` from an odd n with base case `n == 0`), the calls never stop
- After about 1,000 frames, Python stops with a **RecursionError**

THE TURTLE (import turtle as t)

`t.forward(distance)`	Move forward
`t.backward(distance)`	Move backward
`t.left(angle)`	Turn left by angle degrees
`t.right(angle)`	Turn right by angle degrees
`t.up() / t.down()`	Lift the pen / put it down