# Python Track | Codecademy

## Modifying a List

**n.append(item)** adds item at end of list

**n.pop(index)** removes item at index and returns it to you

**n.remove(item)** removes item if found

**del(n[index])** will remove item at given index, but won't return it

**" ".join(n)** combines items in a list into a string, and whatever precedes the period is inserted between the items e.g. spaces in this case

## Iterating through List

**for item in n:** is simple but can cause problems when modifying items

**for item in range()** function defaults are 0 start and 1 step, and stop means up to but not including stop

* **range(stop)** e.g. range(len(n))
* **range(start, stop)**
* **range(start, stop, step)** e.g. range(0, len(n), -1) returns list backwards

Note to self: Do notice whether you are referring to list or list[item]

## Nesting Loops

Example: you have a list containing two lists

n = [[1, 2, 3], [4, 5, 6, 7, 8, 9]]  
  
def flatten(lists):  
    results = []  
    for lst in lists:  
        for numbers in lst:  
    results.append(numbers)  
    return results  
  
print flatten(n)  
  
==> [1, 2, 3, 4, 5, 6, 7, 8, 9]

## Battleship

**randint(low, high)** function from the random module can be imported to generate random numbers given an inclusive range.

from random import randint  
   
board = []  
   
for x in range(5):  
    board.append(["O"] \* 5)  
   
def print\_board(board):  
    for row in board:  
        print " ".join(row)  
   
print "Let's play Battleship!"  
print\_board(board)  
   
def random\_row(board):  
    return randint(0, len(board) - 1)  
   
def random\_col(board):  
    return randint(0, len(board[0]) - 1)  
   
ship\_row = random\_row(board)  
ship\_col = random\_col(board)  
# Only for debugging purposes:  
# print ship\_row  
# print ship\_col  
   
# Everything from here on should go in your for loop!  
# Be sure to indent four spaces!  
for turn in range(4):  
   
    guess\_row = int(raw\_input("Guess Row:"))  
    guess\_col = int(raw\_input("Guess Col:"))  
    # If player guesses the right location  
    if guess\_row == ship\_row and guess\_col == ship\_col:  
        print "Congratulations! You sunk my battleship!"  
        break  
    else:  
        # If player is wrong on last turn  
        if turn == 3:  
            print "Game Over"  
        # If player guesses out of range  
        elif (guess\_row < 0 or guess\_row > 4) or (guess\_col < 0 or guess\_col > 4):  
            print "Oops, that's not even in the ocean."  
        # If player guesses that already  
        elif(board[guess\_row][guess\_col] == "X"):  
            print "You guessed that one already."  
        else:  
            print "You missed my battleship!"  
            board[guess\_row][guess\_col] = "X" # Be careful about indentation here, kept running into IndexError: list index out of range because I was missing an indent  
   
    print "Turn ", turn + 1  
    print\_board(board)

## Loops

**while** loop continues to execute while a condition is true instead of if it is true  
be careful - make sure the while condition can be reached, otherwise your loop could go on forever and become an infinite loop i.e. a loop which never exits, which could crash your computer / browser!  
**break** means "exit the current loop," prior to it you can write define the stopping condition using an if statement  
**while/else** the else block will execute when the loop condition is evaluated to False, unless the loop exits as a result of a break  
for i in range(10): This kind of loop is useful when you want to do something a certain number of times (not inclusive thoguh, this example goes from 0-9)

**for c in thing:** Using a for loop, you can loop through each individual character in a string, which is useful for string manipulation  
**,** keeps print statement in the same line, similar to concatenation using + except it adds a space  
**for animals in zoo:** Perhaps the most useful (and most common) use of for loops is to go through a list.  
**for key in d:** Using a for loop, you can also loop through a dictionary to get the keys and or the values  
 **print key, d[key]** In this example, key is the key and d[key] is the associated value

**enumerate(sequence, start=0)** Built-in enumerate function supplies a corresponding index to each list element  
**for index, season in enumerate(seasons):** During the loop, index increases as item moves to next in sequence  
 **print index+1, season** Index usually starts at 0.   
**list(enumerate(seasons, start=1))** Output: [(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]

Equivalent to:

def enumerate(sequence, start=0):  
    n = start for elem in sequence:  
        yield n, elem  
        n += 1

**zip** will create pairs of elements when passed two (or more) lists and will stop at the end of the shorter list  
**print zip(list\_a, list\_b)** Output: [(3, 2), (9, 4), (17, 8), (15, 10), (19, 30)]  
**for a, b in zip(list\_a, list\_b):** iterates over two lists at once  
**for/else** the else statement is executed after the for, but only if the for ends normally—that is, not with a break

Example:

for f in fruits:  
    if f == 'tomato':  
        print 'A tomato is not a fruit!' # (It actually is.)  
        # break here would make the else not run  
    print 'A', f  
    else:  
        print 'A fine selection of fruits!'