**Welcome to Battleship!**

In this project you will build a simplified, one-player version of the classic board game Battleship! In this version of the game, there will be a single ship hidden in a random location on a 5x5 grid. The player will have 10 guesses to try to sink the ship.

To build this game we will use our knowledge of lists, conditionals and functions in Python. When you're ready to get started, click Next to continue.

**Getting Our Feet Wet**

The first thing we need to do is to set up the game board.

**Instructions**

**1.**

Create a variable board and set it equal to an empty list.

# Make a List

Good! Now we'll use a built-in Python function to generate our board, which we'll make into a 5 x 5 grid of all "O"s, for "ocean."

print ["O"] \* 5

will print out ['O', 'O', 'O', 'O', 'O'], which is the basis for a row of our board.

We'll do this five times to make five rows. (Since we have to do this five times, it sounds like a loop might be in order.)

**Instructions**

**1.**

Create a 5 x 5 grid initialized to all 'O's and store it in board.

* Use range() to loop 5 times.
* Inside the loop, .append() a list containing 5 "O"s to board, just like in the example above.

Note that these are capital letter "O" and not zeros.

Hint

What if you use a for loop that does the appending in the range (0,5)?

|  |
| --- |
| board = []  for i in range(5):  board.append(["O"]\*5) |

# Check it Twice

Great job! Now that we've built our board, let's show it off.

Throughout our game, we'll want to print the game board so that the player can see which locations they have already guessed. Regularly printing the board will also help us debug our program.

The easiest way to print the board would be to have Python display it for us using the print command. Let's give that a try and see what the results look like—is this a useful way to print our board for Battleship?

**Instructions**

**1.**

Use the print command to display the contents of the board list.

Hint

The syntax for printing a list is print list\_name.

# Custom Print

Now we can see the contents of our list, but clearly it would be easier to play the game if we could print the board out like a grid with each row on its own line.

We can use the fact that our board is a list of lists to help us do this. Let's set up a forloop to go through each of the elements in the outer list (each of which is a row of our board) and print them.

**1.**

First, delete your existing printstatement.

Then, define a function named print\_board with a single argument, board\_in.

Inside the function, write a for loop to iterates through each row in board and print it to the screen.

Call your function with board to make sure it works.

Hint

Remember that the syntax to declare a function is def fun\_name(parameters):. The syntax for iterating through a list is for x in list\_name:

|  |
| --- |
| board = []  for i in range(5):  board.append(["O"]\*5)  def print\_board(board\_in):  for item in board\_in:  print item  print\_board(board) |

**Printing Pretty**

We're getting pretty close to a playable board, but wouldn't it be nice to get rid of those quote marks and commas? We're storing our data as a list, but the player doesn't need to know that!

letters = ['a', 'b', 'c', 'd'] print " ".join(letters) print "---".join(letters)

1. In the example above, we create a list called letters.
2. Then, we print a b c d. The .joinmethod uses the string to combine the items in the list.
3. Finally, we print a---b---c---d. We are calling the .join function on the "---"string.

We want to turn each row into "O O O O O".

**Instructions**

**1.**

Inside your function, inside your forloop, use " " as the separator to .join the elements of each row.

Hint

Your function should contain a forloop that iterates over each row in the board. For each row, it should

print " ".join(row)

|  |
| --- |
| board = []  for i in range(5):  board.append(["O"]\*5)  def print\_board(board\_in):  for item in board\_in:  print " ".join(item)  print\_board(board) |

**Hide...**

Excellent! Now, let's hide our battleship in a random location on the board.

Since we have a 2-dimensional list, we'll use two variables to store the ship's location, ship\_row and ship\_col.

from random import randint coin = randint(0, 1) dice = randint(1, 6)

1. In the above example, we first import the randint(low, high) function from the random module.
2. Then, we generate either zero or one and store it in coin.
3. Finally, we generate a number from one to six inclusive.

Let's generate a random\_row and random\_colfrom zero to four!

**Instructions**

**1.**

Define two new functions, random\_row and random\_col, that each take board\_in as input.

These functions should return a random row index and a random column index from your board, respectively. Use randint(0, len(board\_in) - 1).

Call each function on board.

Hint

While we could just call randint(0, 4), we use len(board) - 1 in case we want to change the board size later.

|  |
| --- |
| def random\_row(board\_in):  return randint(0, len(board\_in)-1)  def random\_col(board\_in):  return randint(0, len(board\_in)-1) |

# ...and Seek!

Good job! For now, let's store coordinates for the ship in the variables ship\_row and ship\_col. Now you have a hidden battleship in your ocean! Let's write the code to allow the player to guess where it is.

number = raw\_input("Enter a number: ") if int(number) == 0: print "You entered 0"

raw\_input asks the user for input and returns it as a string. But we're going to want to use integers for our guesses! To do this, we'll wrap the raw\_inputs with int()to convert the string to an integer.

**Instructions**

**1.**

Create a new variable called guess\_row and set it to int(raw\_input("Guess Row: ")).

Create a new variable called guess\_col and set it to int(raw\_input("Guess Col: ")).

Click **Run** and then answer the prompts by typing in a number and pressing Enter (or Return on some computers).

|  |
| --- |
| guess\_row = int(raw\_input("Guess Row: "))  guess\_col = int(raw\_input("Guess Col: ")) |

# It's Not Cheating—It's Debugging!

Awesome! Now we have a hidden battleship and a guess from our player. In the next few steps, we'll check the user's guess to see if they are correct.

While we're writing and debugging this part of the program, it will be helpful to know where that battleship is hidden. Let's add a print statement that displays the location of the hidden ship.

Of course, we'll remove this output when we're finished debugging since if we left it in, our game wouldn't be very challenging. :)

**Instructions**

**1.**

Before the lines prompting the user for input:

Print the value of ship\_row.

Print the value of ship\_col.

# You win!

Okay—now for the fun! We have the actual location of the ship and the player's guess so we can check to see if the player guessed right.

For a guess to be right, guess\_col should be equal to ship\_col and guess\_row should be equal to ship\_row.

if guess\_col == 0 and guess\_row == 0: print "Top-left corner."

The example above is just a reminder about if statements.

**Instructions**

**1.**

On line 29, add an if to check if guess\_row equals ship\_row and guess\_col equals ship\_col.

If that is the case, please print out "Congratulations! You sank my battleship!"

When you run this code, be sure to enter integer guesses in the panel where it asks for "Guess Row" and then "Guess Col".

|  |
| --- |
| from random import randint  board = []  for x in range(0, 5):  board.append(["O"] \* 5)  def print\_board(board):  for row in board:  print " ".join(row)  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)  print ship\_row  print ship\_col  guess\_row = int(raw\_input("Guess Row: "))  guess\_col = int(raw\_input("Guess Col: "))  # Write your code below!  if guess\_row == ship\_row and guess\_col == ship\_col:  print "Congratulations! You sank my battleship!" |

# Danger, Will Robinson!!

Great! Of course, the player isn't going to guess right all the time, so we also need to handle the case where the guess is wrong.

print board[2][3]

The example above prints out "O", the element in the 3rd row and 4th column.

**1.**

Add an else under the if we wrote in the previous step.

Print out "You missed my battleship!"

Set the list element at guess\_row, guess\_col to "X".

As the last line in your elsestatement, call print\_board(board)again so you can see the "X".Make sure to enter a col and row that is on the board!

Hint

To specify a single element in a 2-dimensional list, you need to give two indices. The syntax is: list\_name[i][j], where i is an index in the outer list and j is an index in the inner list.

|  |
| --- |
| if guess\_row == ship\_row and guess\_col == ship\_col:  print "Congratulations! You sank my battleship!"  else:  print "You missed my battleship!"  board[guess\_row-1][guess\_col-1] = "X"  print\_board(board) |

**Bad Aim**

Great job! Now we can handle both correct and incorrect guesses from the user. But now let’s think a little bit more about the "miss" condition.

1. They can enter a guess that's off the board.
2. They can guess a spot they’ve already guessed.
3. They can just miss the ship.

We'll add these tests inside our elsecondition. Let's build the first case now!

if x not in range(8) or \ y not in range(3): print "Outside the range"

The example above checks if either x or yare outside those ranges. The \ character just continues the if statement onto the next line.

**Instructions**

**1.**

Add a new if statement that is nested under the else.

Like the example above, it should check if guess\_row is not in range(5)or guess\_col is not in range(5).

If that is the case, print out "Oops, that's not even in the ocean."

After your new if statement, add an else that contains your existing handler for an incorrect guess. Don't forget to indent the code!

Hint

The valid values for guess\_row are 0 to rows - 1 and the valid values for guess\_col are 0 to cols - 1. You will need to build up a complex Boolean expression of the form:

if guess\_row not in range(your\_desired\_range) or guess\_col not in range(your\_desired\_range): # do something

|  |
| --- |
| if guess\_row == ship\_row and guess\_col == ship\_col:  print "Congratulations! You sank my battleship!"  else:  print len(board)  if guess\_row not in range(len(board)+1) or guess\_col not in range(len(board)+1):  print "Oops, that's not even in the ocean."  else:  print "You missed my battleship!"  board[guess\_row-1][guess\_col-1] = "X"  print\_board(board) |

# Not Again!

Great! Now let's handle the second type of incorrect guess: the player guesses a location that was already guessed. How will we know that a location was previously guessed?

print board[guess\_row][guess\_col]

The example above will print an 'X' if already guessed or an 'O' otherwise.

**Instructions**

**1.**

Add an elif to see if the guessed location already has an 'X' in it.

If it has, print "You guessed that one already."

Hint

Remember to use two equal signs (==) when you're checking for equality.

|  |
| --- |
| elif board[guess\_row-1][guess\_col-1] == "X":  print "You guessed that one already." |

# Play It, Sam

You can successfully make one guess in Battleship! But we’d like our game to allow the player to make up to 4 guesses before they lose.

for turn in range(4): # Make a guess # Test that guess

We can use a for loop to iterate through a range. Each iteration will be a turn.

**Instructions**

**1.**

Add a for loop that repeats the guessing and checking part of your game for 4 turns, like the example above.

At the beginning of each iteration, print "Turn", turn + 1 to let the player know what turn they are on.

Indent everything that should be repeated.

|  |
| --- |
| from random import randint  board = []  for x in range(0, 5):  board.append(["O"] \* 5)  def print\_board(board):  for row in board:  print " ".join(row)  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)  print ship\_row  print ship\_col  # Everything from here on should be in your for loop  # don't forget to properly indent!  for turn in range(4):  print "Turn", turn + 1  guess\_row = int(raw\_input("Guess Row: "))  guess\_col = int(raw\_input("Guess Col: "))  if guess\_row == ship\_row and guess\_col == ship\_col:  print "Congratulations! You sank my battleship!"  else:  if guess\_row not in range(5) or \  guess\_col not in range(5):  print "Oops, that's not even in the ocean."  elif board[guess\_row-1][guess\_col-1] == "X":  print( "You guessed that one already." )  else:  print "You missed my battleship!"  board[guess\_row-1][guess\_col-1] = "X"  print\_board(board) |

**Game Over**

If someone runs out of guesses without winning right now, the game just exits. It would be nice to let them know why.

Since we only want this message to display if the user guesses wrong on their last turn, we need to think carefully about where to put it.

1. We’ll want to put it under the else that accounts for misses
2. We’ll want to print the message no matter what the cause of the miss
3. Since our turn variable starts at 0 and goes to 3, we will want to end the game when turn equals 3.

**1.**

Add an if statement that checks to see if the user is out of guesses.

* Put it under the else that accounts for misses.
* Put it after the if/elif/elsestatements that check for the reason the player missed. We want "Game Over" to print regardless of the reason.

If turn equals 3, print "Game Over".

Hint

Your new if should go under the else that handles all of the incorrect guesses, but outside of any of the nested conditions. This is because it doesn't matter why the guess is wrong; after 4 wrong guesses, the game is over.

|  |
| --- |
| if turn == 3:  print "Game Over" |

# A Real Win

Almost there! We can play Battleship!, but you’ll notice that when you win, if you haven’t already guessed 4 times, the program asks you to enter another guess. What we’d rather have happen is for the program to end—it’s no fun guessing if you know you’ve already sunk the Battleship!

We can use the break command to get out of a for loop.

**Instructions**

**1.**

Add a break under the win condition to end the loop after a win.

Hint

Your break should go inside your outer if statement, right after your "Congratulations!" message.

|  |
| --- |
| from random import randint  board = []  for x in range(0, 5):  board.append(["O"] \* 5)  def print\_board(board):  for row in board:  print " ".join(row)  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)  print ship\_row  print ship\_col  # Everything from here on should be in your for loop  # don't forget to properly indent!  for turn in range(4):  print "Turn", turn + 1  guess\_row = int(raw\_input("Guess Row: "))  guess\_col = int(raw\_input("Guess Col: "))  if guess\_row == ship\_row and guess\_col == ship\_col:  print "Congratulations! You sank my battleship!"  break  else:  if guess\_row not in range(6) or \  guess\_col not in range(6):  print "Oops, that's not even in the ocean."  elif board[guess\_row-1][guess\_col-1] == "X":  print( "You guessed that one already." )  else:  print "You missed my battleship!"  board[guess\_row-1][guess\_col-1] = "X"  print\_board(board)  if turn == 3:  print "Game Over" |

# Extra Credit

You can also add on to your Battleship! program to make it more complex and fun to play. Here are some ideas for enhancements—maybe you can think of some more!

1. Make multiple battleships: you'll need to be careful because you need to make sure that you don’t place battleships on top of each other on the game board. You'll also want to make sure that you balance the size of the board with the number of ships so the game is still challenging and fun to play.
2. Make battleships of different sizes: this is trickier than it sounds. All the parts of the battleship need to be vertically or horizontally touching and you’ll need to make sure you don’t accidentally place part of a ship off the side of the board.
3. Make your game a two-player game.
4. Use functions to allow your game to have more features like rematches, statistics and more!

Some of these options will be easier after we cover loops in the next lesson. Think about coming back to Battleship! after a few more lessons and see what other changes you can make!