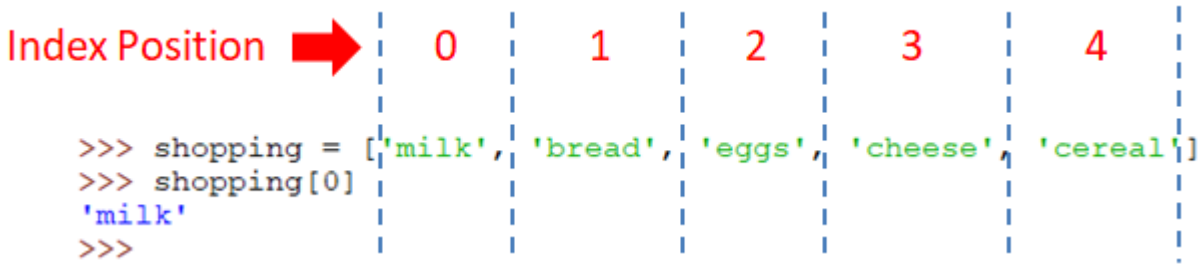
8. LISTS (ARRAYS)

An array is a data structure that allows us to store multiple items using just one variable name. An array in Python is known as a LIST and there are a number of different methods we can use to manipulate and access data inside the list.

We can think about the data items in our list being contained in separate cells in computer memory; this means we can directly access each element in the list using its INDEX POSITION.



The index position starts counting at 0 in Python.

Other ways to access the data in a list:

8.1 SLICE LISTS

Selects the items from the start of the list to INDEX POSITION 3

```
>>> shopping = ['milk', 'bread', 'eggs', 'cheese', 'cereal']
>>> shopping[:3]
['milk', 'bread', 'eggs']
```

Selects the items from INDEX POSITION 2 to the end

```
>>> shopping = ['milk', 'bread', 'eggs', 'cheese', 'cereal']
>>> shopping[2:]
['eggs', 'cheese', 'cereal']
```

Selects the items in the list from INDEX POSITION 1 and up to (but not including) INDEX POSITION 4

```
>>> shopping = ['milk', 'bread', 'eggs', 'cheese', 'cereal']
>>> shopping[1:4]
['bread', 'eggs', 'cheese']
```


EXERCISE 13: SLICES

1. Open the Python shell and create a list of the months of the year; you can use abbreviations like 'Jan', 'Feb', etc.
2. SLICE the list to create these results:

```
>>>
['Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug']
['Jan', 'Feb', 'Mar', 'Apr']
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov']
['Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
```

3. What happens if you type:

   months[:-6]

   months[-3:]

   (or whatever you have called your list of months)?

4. What have you learned from this?

An array or list in Python can also hold integers, and we can use a range of list operations:

- Concatenate (add two or more lists together)
- Multiply lists
- Insert items into a list (using the index position)
- Add items to a list
- Check whether items are in a list
- Find the length of a list
- Delete an item from a list
- Sort a list
- Reverse a list

8.2 CONCATENATE LISTS

In this example, the lists contain INTEGERS:

```
>>> a = [3, 7, 45, 98]
>>> b = [13, 17, 2, 1, 9]
>>> c = a + b
>>> c
[3, 7, 45, 98, 13, 17, 2, 1, 9]
>>> c.sort()
>>> c
[1, 2, 3, 7, 9, 13, 17, 45, 98]
```

We can also use another BUILT-IN FUNCTION to sort the list into numerical order.

This also works for real numbers, numbers with a fractional part, usually called 'floats' in Python.

```
>>> d = [2.5, 3.7, 2.9, 34.5]
>>> e = [5.3, 7.8, 1.97, 12.1]
>>> f = d + e
>>> f
[2.5, 3.7, 2.9, 34.5, 5.3, 7.8, 1.97, 12.1]
>>> f.sort()
>>> f
[1.97, 2.5, 2.9, 3.7, 5.3, 7.8, 12.1, 34.5]
```


It also works for strings:

```
>>> friends_1 = ['Sam', 'Billy', 'Jamie', 'Danny', 'Emily']
>>> friends_2 = ['Claire', 'Jenny', 'Charlie', 'Tom', 'Aaron']
>>> all_friends = friends_1 + friends_2
>>> all_friends
['Sam', 'Billy', 'Jamie', 'Danny', 'Emily', 'Claire', 'Jenny', 'Charlie',
'Tom', 'Aaron']
>>> all_friends.sort()
>>> all_friends
['Aaron', 'Billy', 'Charlie', 'Claire', 'Danny', 'Emily', 'Jamie', 'Jenny
', 'Sam', 'Tom']
```

EXERCISE 14: PRACTISE IN THE SHELL

1. Open the Python Shell and try each of the examples above; you do not have to save anything.
2. Make sure that each example works as shown.

8.3 MULTIPLY LISTS

This works by duplicating the items in the list. We can only use INTEGERS to multiply a list.

```
>>> x = [3, 7, 45, 98]
>>> print(x * 2)
[3, 7, 45, 98, 3, 7, 45, 98]
>>> print(x * 1.5)
Traceback (most recent call last):
  File "<pyshell#38>", line 1, in <module>
    print(x * 1.5)
TypeError: can't multiply sequence by non-int of type 'float'
>>> print(x * 3)
[3, 7, 45, 98, 3, 7, 45, 98, 3, 7, 45, 98]
>>> print(d * 3)
[2.5, 3.7, 2.9, 34.5, 2.5, 3.7, 2.9, 34.5, 2.5, 3.7, 2.9, 34.5]
>>> print(friends_1 * 2)
['Sam', 'Billy', 'Jamie', 'Danny', 'Emily', 'Sam', 'Billy', 'Jamie',
'Danny', 'Emily']
```

8.4 INSERT INTO A LIST

This uses the insert() built-in function. All you need to know is the index position:

```
>>> all_friends.insert(8,'Keiran')
>>> all_friends
['Aaron', 'Billy', 'Charlie', 'Claire', 'Danny', 'Emily', 'Jamie', 'Jenny',
 'Keiran', 'Sam', 'Tom']
```

8.5 ADD ITEMS TO A LIST

We use the append() built-in function, which simply adds the item to the end of the list:

```
>>> all_friends.append('Leah')
>>> all_friends
['Aaron', 'Billy', 'Charlie', 'Claire', 'Danny', 'Emily', 'Jamie', 'Jenny',
 'Keiran', 'Sam', 'Tom', 'Leah']
```


8.6 CHECK WHETHER AN ITEM IS IN THE LIST

This uses the 'in' operator to find out whether the value is in the list, BUT make sure you match the case:

```
>>> all_friends
['Aaron', 'Billy', 'Charlie', 'Claire', 'Danny', 'Emily', 'Jamie', 'Jenny',
 'Keiran', 'Sam', 'Tom', 'Leah']
>>>
>>> 'Charlie' in all_friends
True
>>> 'CHARLIE' in all_friends
False
```

8.7 FIND THE LENGTH OF A LIST

This is a very simple process of using the len() built-in function. Remember that although the INDEX POSITION will start counting from 0, the number of items in the length of the list will start counting at 1.

```
>>> students = ['Tom','Leah','Jamie','Holly','Charlie','Zoe','Aaron']
>>> len(students)
7
```

8.8 DELETE AN ITEM FROM A LIST

This will delete the first occurrence of the item from your list:

```
>>> students = ['Tom','Leah','Jamie','Holly','Charlie','Zoe','Aaron']
>>> students.remove('Zoe')
>>> students
['Tom', 'Leah', 'Jamie', 'Holly', 'Charlie', 'Aaron']
```

8.9 SORT A LIST

We can use the same built-in function to sort lists with strings and integers or floats:

```
>>> students = ['Tom', 'Leah', 'Jamie', 'Holly', 'Charlie', 'Aaron']
>>> students.sort()
>>> students
['Aaron', 'Charlie', 'Holly', 'Jamie', 'Leah', 'Tom']
```

```
>>> numbers = [3, 7, 45, 98, 13, 17, 2, 1, 9]
>>> numbers.sort()
>>> numbers
[1, 2, 3, 7, 9, 13, 17, 45, 98]
```

This also works with a list of mixed integers and floats:

```
>>> mix = [5, 6.3, 65, 12.45, 87, 1.32]
>>> mix.sort()
>>> mix
[1.32, 5, 6.3, 12.45, 65, 87]
```


8.10 REVERSE A LIST

```
>>> students = ['Tom', 'Leah', 'Jamie', 'Holly', 'Charlie', 'Aaron']
>>> students.reverse()
>>> students
['Aaron', 'Charlie', 'Holly', 'Jamie', 'Leah', 'Tom']
```

EXERCISE 15: PRACTICE IN THE PYTHON SHELL

1. Open the Python Shell and try each of the examples above; you do not have to save anything.
2. Make sure that each example works as shown.

8.11 USING LISTS IN PROJECTS: STORING MENU CHOICES

You may be asked to create a menu that is displayed when the program is run and provides the user with options.

Here is a simple example:

```
# use print to display the menu

print("\t\tGame Menu\n")
print("\t\tA - Enter Name\n\t\tB - Play Game\n\t\tC - Quit")

selection = input("Please enter your choice: ")
```

Questions:

1. *What does \n do?*
2. *What does \t do?*

The current menu will allow me to enter any value but I want to restrict this to A, B or C only.
I can use a list to check whether the user input is a valid menu option by using the 'in' operator:

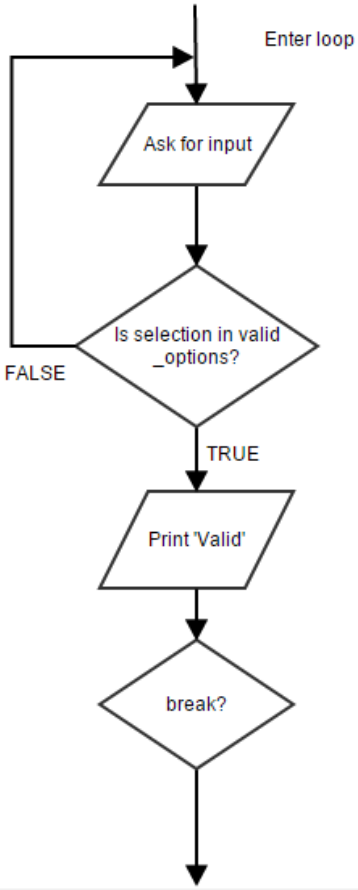
```
valid_option = ['A','B','C']

if selection in valid_option:
    print("That is a valid choice")
else:
    print("That is not a valid choice")
```

EXERCISE 16: CHECKING ITEMS IN A LIST

1. Open the Python shell and choose File >> New File (or New Window).
2. Save your file as 'simple_menu.py'.
3. Copy the code (both parts) and press F5 to run it.
    a. Enter 'g'; it should print 'That is not a valid choice'.
    b. What should the code be able to do now?

Hopefully, you discovered that the menu only works once, so we cannot make another choice. How could the code be amended so that can be achieved?

Look at this example flow chart of the problem:



The solution is to use ITERATION. Look back at the section on Structured Programming about for loops and while loops if you are unsure about where to use these.

```
1   # use print to display the menu
2
3   print("\t\tGame Menu\n")
4   print("\t\tA - Enter Name\n\t\tB - Play Game\n\t\tC - Quit")
5
6   valid_option = ['A', 'B', 'C']
7
8   while True:
9       selection = input("Please enter your choice: ").upper()
10      if selection in valid_option:
11          print("That is a valid choice")
12          break
13      else:
14          print("That is not a valid choice")
```

** The code is displayed in the PyCharm IDE to show line numbers.*

The code will continue to loop and ask for a choice UNTIL one of the three items in the **valid_option** list has been entered. When either A, B or C is entered, the print statement on line 11 is executed, followed by the break statement on line 12 which stops the while loop.

Why is a while loop used here and not a for loop? Check back to the LOOPS section if you are unsure.

READ OR WRITE ARRAY VALUE WITH LOOPS

Iteration can also be used to add values to an array or read values from an array. In your exam, any questions involving arrays may start indexing at number 1 rather than 0, but this will be made clear in the question.

```
1  party_invites ← []
2
3  FOR index ← 1 TO 5
4      party_invites(index)← INPUT
5      NEXT index
6  PRINT party_invites
```

This example uses a FOR loop to add five friends to your party invitation list. The empty array is declared in Line 1. The FOR loop starts on Line 3 and specifies how many times the loop will repeat, with the variable 'index' being used to control how many iterations have been completed.

Line 4 adds or appends each input to the array, and the array is then printed on Line 6.

Here is the example shown in Python. In many programming languages, including Python, indexing starts at 0, so the range counts from 0 up to, but not including, 5.

```
Choose a friend for the party: Ann
Choose a friend for the party: Billie
Choose a friend for the party: Chloe
Choose a friend for the party: Dean
Choose a friend for the party: Elisha
['Ann', 'Billie', 'Chloe', 'Dean', 'Elisha']
```

```
4   party_invites = []
5
6   for index in range(5):
7       friend = input("Choose a friend for the party: ")
8       party_invites.append(friend)
9
10  print(party_invites)
```

EXERCISE 17: SIMPLE MENU

1. Amend your 'simple_menu.py' file to run the menu inside a function and return the selection variable.
2. Use the main() function to run the simple program:
    a. When A is selected, the program should ask for your name and print a welcome message including the name. The menu should then be displayed again.
    b. When B is selected, the program should simply print 'Game is starting'. The menu should then be displayed again.
    c. When C is selected, the program should print 'Thank you for playing' and finish.

Note: Check back to the FUNCTIONS topic if you are unsure about how to start this exercise.

EXERCISE 18: PUTTING IT ALL TOGETHER

Role-play games are very popular and usually involve moving around a location solving puzzles and collecting items in your inventory. These items can then be stored to use later to help you in the game.

1. Write a simple program to store items in an inventory for the game.
    a. Do an Internet search if you are not sure what an inventory is.

2. Your program must:
    i. Be able to add items to your inventory by choosing that option from the menu.
    ii. Be able to display all the items you have in the inventory by choosing that option from the menu.
    iii. Allow you to 'get' an item from the inventory (this means it is no longer in the inventory) by choosing that option from the menu.
    iv. Display the menu again after each choice until you enter 'q'.

TIP: Ideally you will use separate functions for each option.

Example:

```
****** Options Menu******
Enter 'p' to print inventory
Enter 'a' to add item to inventory
Enter 'g' to get item out of inventory
Enter 'q' to quit
Please enter choice: >>
```

Hint: remember to pass the updated inventory between the functions in your program.

8.12 2D LISTS/ARRAYS

So far we have just looked at a one-dimensional list; however we can also create a two-dimensional array, i.e. a list of lists.

I can also mix data types within a 2D list, making them even more versatile. Look at this example:

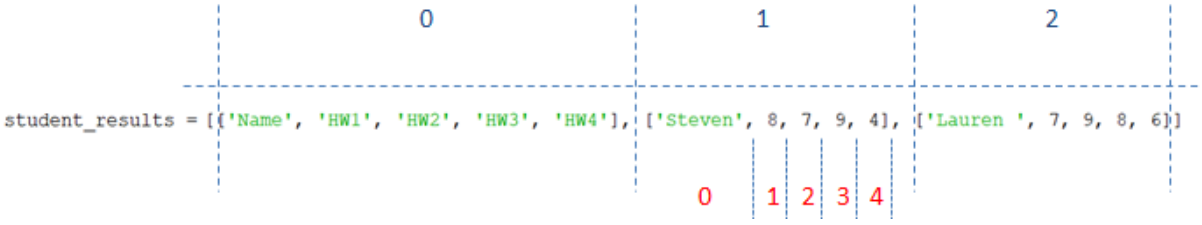
File Edit Format Run Options Window Help

```
studentResults = [['Name', 'HW1', 'HW2', 'HW3', 'HW4'],
                  ['Steven', 8, 7, 9, 4], ['Lauren ', 7, 9, 8, 6]]
```


ACCESSING ITEMS IN A 2D LIST

We can think of a 2D list as a table with rows and columns. If I want to access a particular element in my 2D list, I can use the index position of the list inside the outer list and then the location of the data item inside the nested list. In this example, the row number is 1 and the column number is 0.

Remember: we start counting elements in data structures from 0.




```
print(student_results[1][0])
print(student_results[2][4])
```

```
>>>
Steven
6
>>>
```

PRINTING OUT A 2D LIST

Using the command **print(student_results)** produces this result:

```
[['Name', 'HW1', 'HW2', 'HW3', 'HW4'], ['Steven', 8, 7, 9, 4], ['Lauren ', 7, 9,
 8, 6]]
>>>
```

IMPROVING THE PRINTED RESULTS OF 2D LISTS

The printed output is not easy to read; ideally we want this to print in some form of tabular layout so that we can read down each column of data:

Student	HW1	HW2	HW3	HW4
Steven	8	7	9	4
Lauren	7	9	8	6

Option 1

Loop through the list to print out the five items:

```
for a, b, c, d, e in student_results:
    print(a, b, c, d, e)
```

```
>>>
Name HW1 HW2 HW3 HW4
Steven 8 7 9 4
Lauren  7 9 8 6
>>>
```


This is better, but the data is still not easy to read. Look back at the strings formatting section in ***Chapter 3 – Data Types***.

```
for a, b, c, d, e in student_results:
    print('{0:10}{1:<10}{2:<10}{3:<10}{4:<10}'.format(a, b, c, d, e))
```

produces this:

```
>>>
Name       HW1       HW2       HW3       HW4
Steven     8         7         9         4
Lauren     7         9         8         6
>>>
```

Option 2

We can split the data and print out separately to try to get a better layout:

```
header = student_results[0]

for each in header:
    print('{0:8}'.format(each),end = '')

data = student_results[1:]
print('\n')

for a, b, c, d, e in data:
    print('{0:<8}{1:<8d}{2:<8d}{3:<8d}{4:<8d}'.format(a, b, c, d, e))
```

This example first prints out the headings on one row; the data is then sliced from the original 2D list and a new line is printed.

Each of the fields is then right-aligned and an integer format option is added to the numerical fields to produce this:

```
>>>
Name     HW1      HW2      HW3      HW4

Steven   8        7        9        4
Lauren   7        9        8        6
>>>
```

Option 3

We could use the escape sequence '\t' that we have used in the menu exercise above and in the STRINGS section.

```
header = student_results[0]

for each in header:
    print('{0}\t'.format(each),end = '')

data = student_results[1:]
print('\n')

for a, b, c, d, e in data:
    print('{0:<5}\t{1:<5d}\t{2:<5d}\t{3:<5d}\t{4:<5d}'.format(a, b, c, d, e))
```

displays as:

```
Name     HW1      HW2      HW3      HW4

Steven   8        7        9        4
Lauren   7        9        8        6
```


8.13 ADDING USER INPUT TO A 2D LIST

The first stage is to create a list and add some header details:

```
File  Edit  Format  Run  Options  Window  Help
favMeal = []
header = ['Starter','Main','Dessert','Drink']
favMeal.append(header)

print(favMeal)
```

```
>>>
[['Starter', 'Main', 'Dessert', 'Drink']]
>>>
```

All the inputs to the list will be strings, so we can create a 'helper' function to repeat the request for data inputs.


```
def get_string_inputs(p):
    """requests the string data and returns"""
    str_data = input("Please enter {0}: ".format(p))
    return str_data


def add_data_rows():
    """ adds data to the row array"""
    meal_details = [ ]
    # the helper function is used to get the dish for each course
    starter_c = get_string_inputs("your starter dish ")
    meal_details.append(starter_c)
    main_c = get_string_inputs("your main course ")
    meal_details.append(main_c)
    dessert_c = get_string_inputs("your dessert course ")
    meal_details.append(dessert_c)
    drink = get_string_inputs("your drink  ")
    meal_details.append(drink)
    return meal_details
```

The helper function can be repeated using different prompts and different names for the return value

In this example, the data is put into a new 1D list which we can add as a row of data:

It is good practice to use a main() function to control the order in which your code is executed.

```
def main():
    """runs all functions"""

    fav_meal= [ ]
    header = ['Starter', 'Main', 'Dessert', 'Drink']
    fav_meal.append(header)

    meal_details = add_data_rows()
    fav_meal.append(meal_details)
    for a, b, c, d in fav_meal:
        print('{0:16}{1:<16}{2:<16}{3:<16}'.format(a, b, c, d))

main()
```

```
>>>
Please enter your starter dish : Hummus
Please enter your main course : Tuna Salad
Please enter your dessert course : Cheesecake
Please enter your drink  : Fizzy Water
Starter         Main            Dessert         Drink
Hummus          Tuna Salad      Cheesecake      Fizzy Water
```

EXERCISE 19: DEVELOP THE FAVOURITE FOODS PROGRAM

You will use the program to find out the favourite meals of students in your class.

1. Use the example here as a starting point; your program also needs to ask for each student's name and add this to each row of data in the 2D list.
2. Your program should ask whether more data is to be added after each student's details are entered.
3. Your program should print the full 2D list of data collected in a tabular format when all data has been added.

EXERCISE 20: FUNDRAISER PART 1

It can be very useful to use a 2D list to store data for your program so that data can be passed easily between functions, making your code more efficient. For example:

Activity	Charge per hour
Bag Pack	£2.50
Car Wash	£3.50

Your school is organising a charity fundraising day and wants students to volunteer to take part in an activity. As part of this, you think it will be useful to try to work out how much money students in your form could raise by deciding on a charge for the activities. The three activities your form has chosen are:

- Bag packing at a local supermarket
- Washing cars at school
- Dog walking

Tasks

1. Write a program that will display a table of activities with the charge per hour, similar to the one shown on the previous page.

2. The program should:

    a. Ask for the name of the activity.
        i. For extra credit, the maximum length should be 10 characters or fewer.
        ii. You should display a suitable message if the text length is longer.
        iii. Your code should not allow the user to continue until a suitable length of characters has been entered.

    b. Ask for the amount to be charged per hour for the activity.
        i. For extra credit, the acceptable charges are between 99p and £5.99.
        ii. You should display a suitable message if the charge is not within these values.
        iii. Your code should not allow the user to continue until a value between those amounts has been entered.

3. When all data has been entered, the program should print out the information in a tabular format with suitable column headings.

Your solution should use:

- Functions and a main() function to run the program.
- The functions should have the correct structure and use docstrings to say what they do.
- You should pass the data structure between functions in order to print it.
- You could consider using helper functions for more efficient code (see Add User Input to a 2D List).

EXERCISE 21: FUNDRAISER PART 2

In this part of the exercise, you need to enter the student name, ask for the activity they completed, ask how many hours they spent on the activity, and calculate the totals for each student.

1. Develop the program to ask for the number of students you want to enter.
2. Using helper functions, the program should:
    a. Ask for the student name.
    b. Ask for the activity completed (using the list created in the previous exercise).
    c. Ask for the number of hours spent on the activity (as a whole number).
3. Your program should then use the values entered for each activity (from the list created in the previous exercise) to calculate a total collected.
4. Your program should then print out the information about student name, activity completed, hours spent and total funds each student has raised.

8.14 CREATING A GAME BOARD

Many games use the idea of a grid, where players move around the grid to play the game; these are common programming project tasks and can easily be completed in Python.

We already know how to create a list of lists (a 2D list):

```
File  Edit  Format  Run  Options  Window  Help
# create a simple game board

row1 = [0, 0, 0, 0, 0]
row2 = [0, 0, 0, 0, 0]
row3 = [0, 0, 0, 0, 0]
row4 = [0, 0, 0, 0, 0]
row5 = [0, 0, 0, 0, 0]

game_board = [row1, row2, row3, row4, row5]

for row in game_board:
    print(" ")
    for each in row:
        print(each,end = "")
```

```
00000
00000
00000
00000
00000
>>>
```

How do I show where my player is on the board?

Remember that we can think of the first index as the row number and the second index as the column number. Use the index position in each list; for example, if I want to show my player piece is in the third column in the third row, I can insert the variable using the index values of the nested list and the index value INSIDE that list.

```
player_location = "X"

game_board[2][2]    = player_location
```

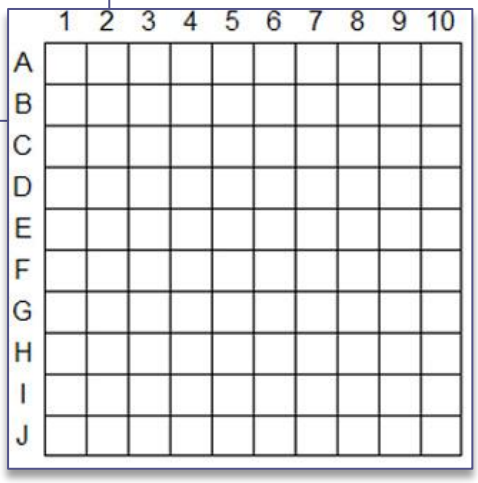
```
00000
00000
00X00
00000
00000
>>>
```

Why have I said the third row and column and then used [2][2] as my index positions?

Remember: we start counting elements in data structures from 0.

This is not a very efficient way to create a game board; using ITERATION will make our code much more efficient.


```
File  Edit  Format  Run  Options  Window  Help
# create a simple game board

player_location = "X"
game_board = []

for x in range(0, 5):
    row = ['0'] * 5
    game_board.append(row)

game_board[2][2] = player_location


for row in game_board:
    print(" ")
    for each in row:
        print(each,end = "")
```



If I want to enter instructions into my game about where the player should move, I may want to show the game board like the figure above using row and column headings.

EXERCISE 22: DEVELOP THE SIMPLE GAME BOARD

1. Write a function to create a game board grid, measuring 8 × 8.
2. Add a row of numbers, from 1 to 8, across the top of the grid.
    a. Hint: this will be the first nested list in your game board list.
3. Add your player location in the bottom left-hand corner of the grid.
4. Create a function to print out the grid.
5. Call your two functions from within a main function.

How can the letters be added to the start of each row?

There are two methods we could use:

1. When the grid has been created, we can loop through and add our letter using the index position [0] in each row. In order to keep the same number of locations in the grid, the rows would need to be increased in length by one.

2. When the grid is printed, we can use the string method .join() to add the letters at the start.

Method 1:

```
4   def create_game_board():
5       """creates the game board"""
6
7       new_board = []
8       top_row = ['  1 ', ' 2 ', ' 3 ', ' 4 ', ' 5 ']
9       # spaces add to improve layout when printed
10      new_board.append(top_row)
11      # append as first item in the 2D array.
12
13      for x in range(0, 5):
14          row = [' 0 '] * 6
15          # this value needs to be 6 as the letter will be added at position[0]
16          new_board.append(row)
17
18      # add the letters to each row
19      row_letters = ['A', 'B', 'C' 'D', 'E']
20
21      for x in range(len(row_letters)):
22          new_board[x + 1][0] = row_letters[x]
23      return new_board
```

*Note: On line 13 the rows are set at 6 columns wide. On lines 21 and 22 each letter is inserted at index position [0] in each row. The variable 'x' is being used to loop through both the **new_board** 2D list AND the **row_letters** list.*

Method 2:

```
4   def create_game_board():
5       """creates the game board"""
6
7       new_board = []
8       top_row = [' 1 ', ' 2 ', ' 3 ', ' 4 ', ' 5 ']
9       # spaces add to improve layout when printed
10      new_board.append(top_row)
11      # append as first item in the 2D array.
12
13      for x in range(0, 5):
14          row = [' 0 '] * 5
15          # the value this time is 5 as the letters are added
16          # when the board is printed
17          new_board.append(row)
18
19      return new_board
20
21
22  def print_board(b):
23      """ prints the board layout to screen"""
24      # add the letters to each row
25      row_letters = [' ', 'A', 'B', 'C', 'D', 'E']
26
27      i = 0   # variable used to iterate through row_letters
28      for row in b:
29          print(row_letters[i], "".join(row))
30          i += 1
```

*Note: In this method the **new_board** 2D list is returned and passed into a print function. In order to get the same layout an extra 'blank' is added at the start of row_letters. The variable **i** is used to loop through each element in **row_letters** and join to the start of each row as it is printed.*

Both of these methods produce the same printed output:

```
  1  2  3  4  5
A 0  0  0  0  0
B 0  X  0  0  0
C 0  0  0  0  0
D 0  0  0  0  0
E 0  0  0  0  0
>>>
```

Remember: when we design a function we can simply use 'placeholders', which we call the parameters, and then supply the arguments, the actual values, when we call the function.

If you look at this function which prints the game board, the actual arguments are supplied when the function is called from inside the main function used to control the program.

```
33  def main():
34      """ runs all functions"""
35      player_location = " X "
36      new_board = create_game_board()
37      new_board[1][2] = player_location
38      print_board(new_board)
39
40
41  main()
```


8.15 MOVING A PLAYER ON THE GAME BOARD

How do I get my player to move around the game? There are a number of ways this could be achieved. For example:

- Import the random module and create a dice to randomly choose the next location.
- Ask the player where they want to move.

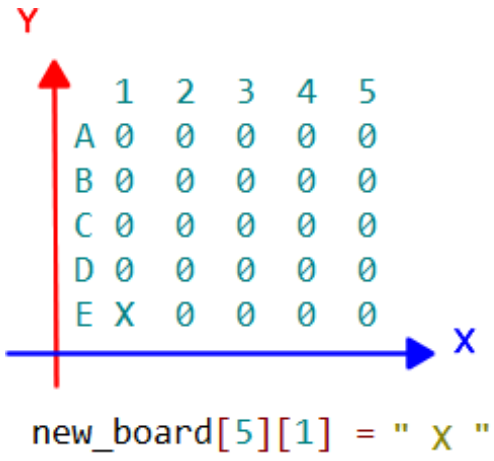
In either of these cases, you will need to be able to check that the proposed move is within the boundaries of the game board to ensure no moves are allowed that will take a player outside the range of the 2D list.

EXERCISE 23: MOVE A PLAYER PART 1

Using the file you have created for the Simple Board Game:

1. Write a function to ask if the player wants to move up, move down or stay on the row. Make sure you store their response for use later.
2. Write a function to ask whether the player wants to move left, move right or stay on the same column. Make sure you store their response for use later.
3. Write a function to ask how many 'squares' the player wants to move. Again, you should store this response in a suitable variable for use later.

Keeping track of where your player is on the board is important. Look at this diagram of the board game with an 'X' in the starting position:



The Y position is [5] and the X position is [1]; using this information we can keep track of where the current position of the player is and whether the new position is within the boundaries of the board, i.e. inside the 2D list.

Example: You have chosen to move 3 squares up the game board and 4 squares to the right. This means that the new position for Y will be [2] and for X will be [5].

Your code will need to check whether the move is possible. If it is, make the move; if not, display an error message or do something else. Don't forget that you also need to change the old position for 'X' back to a '0' at the same time.

EXERCISE 24: MOVE A PLAYER PART 2

You should now know enough to be able to ask which direction to move and how many squares to move, and make the game board show this move.

1. Write a function to make a move, inside which you should:
    a. Call the function for vertical movement
    b. Call the function for the number of squares to move
    c. Check whether this is possible by using the current player location for axis 'y'
    d. If this is not a valid move: the game is over
    e. If the move is possible
    f. Call the function for horizontal movement
    g. Call the function for the number of squares to move
    h. Check the move is possible by using the current player location for axis 'x'
    i. If this is not a valid move: the game is over
    j. If the move is possible
    k. Return the new x and y values
    l. Update the previous player location to show '0' not 'X'
    m. Print the updated grid with the new player location

2. Add a while loop to the main function to keep asking/moving until five moves have been made.

TIP: In order to update the board and change the 'X' to a '0' when a move is made, you should store the current x and y values in two new variables.

8.16 WINNING THE GAME

The game is won when the player lands on a 'magic square'. You could set this by creating a variable and 'hard-coding' the location, or make this change each time the player moves by using a random function.

Remember that you need to import the random function before you can use it; look back at chapter ***6 – Importing Modules*** if you are unsure about how to do this.

```
97      def random_magic_square(b):
98          """ randomly choose location of magic square"""
99          board_size = (len(b) - 1)
100         magic_y = random.randrange(1, board_size)
101         magic_x = random.randrange(1, board_size)
102         return magic_y, magic_x
```

In this example, the 2D list game board is passed into the function so that the size of the board can be calculated on line 99. I could just insert the size of the board by hard-coding the value; my example is a 5 × 5 board, but using this method allows my code to be developed for any board size.

The 'magic square' location can now be checked each time the game loop is run. The game can be developed so that you continue to play until EITHER you find the 'magic square' OR you choose to exit the game.

EXERCISE 25: CREATE A MAGIC SQUARE

An easy version of this game will be to create the magic square location just once and then check whether the player has moved to that location each time the game loop runs.

The hard version of the game involves creating a new random magic square each time the game loop runs.

1. Write your function to create the location of the magic square.

2. Write a function that will ask the player whether they want to continue playing the game; ensure that you write robust code to check their answer can only be Y or N.

3. Develop the while loop that runs your game so that the game continues until the answer is N.

