

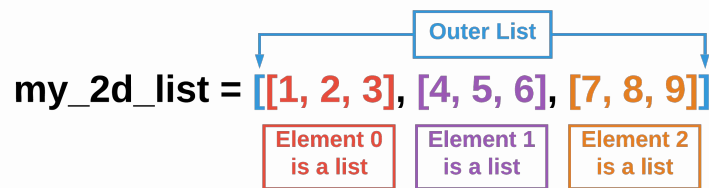
Learning Objectives - 2D Lists

- **Define a 2D list**
- **Demonstrate how to add an element to a 2D list**
- **Describe how to access elements in a 2D list**
- **Demonstrate how to iterate over a 2D list**
- **Demonstrate how to print a 2D list in a human readable way**

Declaring 2D Lists

Declaring a 2D List

A 2D list is a list whose individual elements are other lists; sometimes referred to as a list of lists.



2D List

To declare a 2D list, start as normal with a pair of square brackets ([]). For each element, declare another list with more square brackets. Be sure to use commas when separating each internal list.

```
my_2d_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(my_2d_list)
```

challenge

Try this variation:

- Create a 2D list comprised of empty lists. Print this 2D list.

▼ Solution

```
my_empty_2d_list = [[], [], []]
print(my_empty_2d_list)
```

Declaring 2D Lists in a Human Readable Way

2D lists are useful for when you want a group of data where each group is itself a group of data. Imagine that you are going to list the three most populous cities for America, France, China, and India. Declaring the list as

described above, would make for a long line of code. While Python will not have trouble understanding this line of code, some humans might. It is a good idea to write your code in a way that is easy for humans to read. This will make debugging your code easier.

```
populous_cities = [
    ["USA", "New York City", "Los Angeles",
     "Chicago"],
    ["France", "Paris", "Marseille", "Lyon"],
    ["China", "Shanghai", "Beijing",
     "Chongqing"],
    ["India", "Mumbai", "Delhi", "Bangalore"]
]
print(populous_cities)
```

▼ Printing a 2D list

The standard print statement does not print a 2D list in a human readable way. However, it is possible to print a 2D list in a more readable way. This will be covered on a later page.

challenge

Try this variation:

- Create a 2D list in a human readable way based on the information in the table below. Print the list when done.

Popular Breeds

Pet | Popular Breeds |

:-:-----:-|

Dogs | Labrador Retriever, German Shepherd, Golden Retriever |

Cats | Siamese, Persian, Maine Coon |

▼ Solution


```
pets = [
    ['dogs', 'labrador retriever', 'german shepherd',
     'golden retriever'],
    ['cats', 'siamese', 'persian', 'maine coon']
]
print(pets)
```

Accessing Elements in 2D Lists

Accessing Elements in a 2D List

In a traditional list, square brackets and a number are used to denote a particular element. For example, `my_list[2]`. Python would return the third element in the list. The 2D list below has three color palettes (warm, cool, and neutral) in hexadecimal code. A visual representation of the colors can be found to the right of the 2D list.

```
colors = [
    ["#de0c1c", "#fe2d2d", "#fb7830", "#fecf02", "#ffdd46"],
    ["#1db4d4", "#246fb4", "#100ab6", "#130976", "#110240"],
    ["#ebe9e7", "#c4bdb7", "#8a7b70", "#4f3928", "#3c2411"]
]
```



Color Palettes

In a 2D list, `colors[2]` would return the third element in `colors` which is another list (the neutral color palette).

```
colors = [
    ["#de0c1c", "#fe2d2d", "#fb7830", "#fecf02",
    "#ffdd46"],
    ["#1db4d4", "#246fb4", "#100ab6", "#130976",
    "#110240"],
    ["#ebe9e7", "#c4bdb7", "#8a7b70", "#4f3928",
    "#3c2411"]
]
print(colors[2])
```

▼ What is hexadecimal?

Hexadecimal is a way of representing numbers using a base of 16. The symbols “0 - 9” represent the values zero to nine, and the symbols “a - f” represent the values ten to fifteen. Hexadecimal is often used to denote colors on the web.

challenge

Try this variation:

- Print the first element (the warm color palette).



Solution

```
print(colors[0])
```

- Print the second element (the cool color palette).



Solution

```
print(colors[1])
```

Accessing a Color in a Specific Palette

To print a particular hexadecimal color from a particular color palette, you need to use two sets of square brackets. The first set references the color palette and the second set references the hexadecimal color. In the example below, you will see how `colors[0][1]` is resolved.

```
colors = [
    colors[0][1]
    ["#de0c1c", "#fe2d2d", "#fb7830", "#fecf02", "#ffdd46"],
    ["#1db4d4", "#246fb4", "#100ab6", "#130976", "#110240"],
    ["#ebe9e7", "#c4bdb7", "#8a7b70", "#4f3928", "#3c2411"]
]
```

Two Indexes

```
print(colors[0][1])
```

challenge

Try this variation:

- Print the hexadecimal color #100ab6.



Solution

```
print(colors[1][2])
```

- Print the hexadecimal color #c4bdb7.



Solution

```
print(colors[2][1])
```

- Print the hexadecimal color #ffdd46.



Solution

```
print(colors[0][4])
```

Adding Elements to 2D Lists

Adding Elements to a 2D List

A common task for lists is to add an element to a list. You can use the + operator to concatenate 2D lists.

```
list_1 = [[1, 2, 3], [4, 5, 6]]  
list_2 = [[7, 8, 9], [10, 11, 12]]  
  
print(list_1 + list_2)
```

challenge

Try this variation:

- Change the print statement to `print(list_2 + list_1)`
- Change the print statement to `print(list_2 + [[13, 14, 15]])`
- Change the print statement to `print(list_2 + [13, 14, 15])`
- Change the print statement to `print(list_2 + 13)`

▼ Concatenating `[[13, 14, 15]]`, `[13, 14, 15]`, and `13`

The output from these list concatenations might be a little confusing. Here is what Python is doing:

- **`[[13]]`** - This is a 2D list with one element that also only has one element. So the final result is a list of two lists. The first has three elements, while the second only has one.
- **`[13]`** - This is a traditional list with one element. So the final result is a list with two elements. The first element is a list with three elements, while the second element is the integer 13. In Python, a 2D list can have elements that are lists as well as elements that are not lists.
- **`13`** - This causes an error because the concatenation operator expects to join two lists together. It can join two 2D lists or a 2D list and a traditional list. It cannot join a list and a non-list.

You can think of the concatenation process as removing a set of square brackets from the data being concatenated. So a 2D list becomes a traditional list (`[[13, 14, 15]]` becomes `[13, 14, 15]`), and a traditional list becomes a sequence of new elements (`[13, 14, 15]` becomes `13, 14, 15`). However, there are no square brackets to remove when trying to concatenate a non-list, which is why there is an error.

The append Method

The append method can be used with 2D lists to add another element to a list.

```
my_list = [{"a", "b", "c"}, {"d", "e", "f"}]
another_list = ["g", "h", "i"]

my_list.append(another_list)
print(my_list)
```


challenge

Try this variation:

- Change the append statement to `my_list.append("g")`
- Change the append statement to `my_list.append(["g", "h", "i"])`
- Change the append statement to `my_list.append([["g", "h", "i"]])`

▼ Appending "g", ["g", "h", "i"], and [["g", "h", "i"]]

The output from the append method might be a little confusing.

Here is what Python is doing:

- **"g"** - This creates a 2D list with three elements. The first two elements are lists of strings, while the third element is the string "g".
- **["g", "h", "i"]** - This creates a 2D list with three elements. The first two elements are lists of strings, while the third element is a list with "g", "h", and "i" as its elements.
- **[["g", "h", "i"]]** - This creates a 2D list with three elements. The first two elements are lists of strings, while the third element is another 2D list. This list has one element, another list with "g", "h", and "i" as its elements.

You can think of the append method as adding its parameter to the list. Passing append a string ("g") will add the string as a new element. Passing append a list (["g", "h", "i"]) will add the list as a new element. Passing append a 2D list ([["g", "h", "i"]]) will add the 2D list as a new element.

Iterating Over 2D Lists

Iterating with Indices

Using a traditional for loop to iterate over a 2D list works, but it does not allow access to all of the information stored in the 2D list. Below is a list of the tallest mountains in Asia, North America, and Africa. Assume you want to print its name if the mountain has six or fewer characters. Those mountains would be K2 and Denali.

```
mountains = [  
    ["Mount Everest", "K2", "Kangchenjunga"],  
    ["Denali", "Mount Logan", "Pico de Orizaba"],  
    ["Mount Kilimanjaro", "Mount Kenya", "Mount  
    Ngaliema"]  
]  
  
for mountain in mountains:  
    if len(mountain) <= 6:  
        print(mountain)
```

The code above did not produce the desired result. That is because the for loop only looked at each of the three lists. It **did not** look inside each list. So the line of code `len(mountain)` was not calculating the length of each individual mountain. Instead it was calculating the length of each list of mountains. Since each list has three elements, all of the lists were printed.

To access each of the elements inside the inner lists, you need to use a nested loop.

```
mountains = [  
    ["Mount Everest", "K2", "Kangchenjunga"],  
    ["Denali", "Mount Logan", "Pico de Orizaba"],  
    ["Mount Kilimanjaro", "Mount Kenya", "Mount  
    Ngaliema"]  
]  
  
for row in range(3):  
    for column in range(3):  
        if len(mountains[row][column]) <= 6:  
            print(mountains[row][column])
```

The results of the program match our expectations. This is because the outer loop (row) indicates each list of mountains, while the inner loop (column) indicates each mountain in its list.

`mountains[row][column]` where `row = 1` and `column = 2`

```
mountains = [  
    ["Mount Everest", "K2", "Kangchenjunga"],  
    ["Denali", "Mount Logan", "Pico de Orizaba"],  
    ["Mount Kilimanjaro", "Mount Kenya", "Mount Ngaliema"]  
]
```

Iterating Over a 2D List

challenge

Try this variation:

Use the following 2D list to help you solve the coding problem below.

```
numbers = [  
    [11, 13, 22, 76, 54],  
    [2, 65, 107, 112, 8],  
    [33, 90, 34, 7, 804]  
]
```

Iterate over the list `numbers` and for every element print even if it is an even number or print odd if it is an odd number.

▼ Solution

```
numbers = [  
    [11, 13, 22, 76, 54],  
    [2, 65, 107, 112, 8],  
    [33, 90, 34, 7, 804]  
]  
  
for row in range(3):  
    for column in range(5):  
        if numbers[row][column] % 2 == 0:  
            print('even')  
        else:  
            print('odd')
```

Iterating without Indexes

The examples above show how to use a nested loop with indexes to iterate over a 2D list. We saw in an earlier section how to iterate over a traditional list in a more pythonic way. The following code snippet iterates over the list `numbers` and prints each element. No indexes (`number[1]`) were used.

```
numbers = [1, 2, 3, 4, 5]

for number in numbers:
    print(number)
```

▼ What does pythonic mean?

Pythonic means using the features of the Python language to make your code simple, concise, and easy to read. In this case, use the pattern “for element in sequence” instead of the list name and an index.

This same idea can be used on a 2D list. The code below uses the mountain example but without any indexes. This code is a bit easier to read since `mountain` replaces `mountains[row][column]`.

```
mountains = [
    ["Mount Everest", "K2", "Kangchenjunga"],
    ["Denali", "Mount Logan", "Pico de Orizaba"],
    ["Mount Kilimanjaro", "Mount Kenya", "Mount
    Ngaliema"]
]

for row in mountains:
    for mountain in row:
        if len(mountain) <= 6:
            print(mountain)
```

challenge

Try this variation:

Use the following 2D list to help you solve the coding problem below.

```
numbers = [  
    [11, 13, 22, 76, 54],  
    [2, 65, 107, 112, 8],  
    [33, 90, 34, 7, 804]  
]
```

Iterate over the list `numbers` and for every element print even if it is an even number or print odd if it is an odd number. Write your loop such that no indices are needed.

▼ Solution

```
numbers = [  
    [11, 13, 22, 76, 54],  
    [2, 65, 107, 112, 8],  
    [33, 90, 34, 7, 804]  
]  
  
for row in numbers:  
    for number in row:  
        if number % 2 == 0:  
            print('even')  
        else:  
            print('odd')
```

Printing 2D Lists

Printing a 2D List

You previously saw how a single print statement can print a 2D list. For smaller lists, this works alright. As the 2D list grows in length and complexity, printing a 2D list on a single line becomes harder to read.

```
import random

numbers = [[random.randint(1, 101) for columns in range(10)] for
            rows in range(10)]
print(numbers)
```

▼ What is going on with the variable numbers?

The variable `numbers` is declared using something called a list comprehension. A list comprehension is a very concise (some would say difficult to understand) way of creating a list. `numbers` is a 2D list with ten columns and ten rows. Each element is a random integer between 1 and 100. Every time you run the code, the 2D list will be populated with different numbers.

One way to increase readability is to use a single for loop and print each inner list.

```
import random

numbers = [[random.randint(1, 101) for columns in range(10)] for
            rows in range(10)]
for row in numbers:
    print(row)
```

challenge

Try this variation:

Rewrite the following code such that each inner list is printed on its own line.

```
symbols = [{"*" for columns in range(5)] for rows in range(7)]  
print(symbols)
```

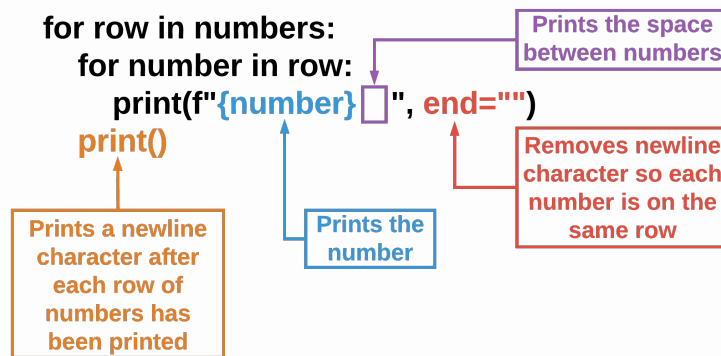
▼ Solution

```
symbols = [{"*" for columns in range(5)] for rows in range(7)]  
  
for row in symbols:  
    print(row)
```

Printing Just the Data

Using a single for loop to print each inner list on its own line helps with readability. However, you still see the square brackets and commas. Using a nested loop can remove these extra symbols.

```
import random  
  
numbers = [[random.randint(1, 101) for columns in range(10)] for rows in range(10)]  
for row in numbers:  
    for number in row:  
        print(f"{number} ", end="")  
    print()
```



Just the Data

challenge

Try this variation:

Rewrite the following code such that each inner list is printed on its own line and there are no square brackets or commas.

```
symbols = [{"*" for columns in range(5)] for rows in range(7)]
print(symbols)
```

▼ Solution

```
symbols = [{"*" for columns in range(5)] for rows in range(7)]

for row in symbols:
    for symbol in row:
        print(f'{symbol}', end='')
    print()
```

Formatting the Data

The output can be refined a further step by making the spacing between the numbers more consistent. There is a space between each number, but when a number can be one, two, or three digits the overall alignment does not look that good. Instead of printing a space between each number, print a tab (this requires the escape character `(\t)`).


```
import random

numbers = [[random.randint(1, 101) for columns in range(10)] for
            rows in range(10)]

for row in numbers:
    for number in row:
        print(f"{number}\t", end=" ")
    print()
```

Manipulating 2D Lists

Manipulating 2D Lists with Methods

This page is not about one particular method used with a 2D list. Instead, this page is about how 2D list **do not** respond to methods the same way traditional lists do. Look at the 2D list `numbers` below. It contains all numbers from 1 to 20. There are four inner lists with five numbers each. Pay attention to how Python sorts `numbers`.

```
numbers = [  
    [6, 7, 10, 9, 8],  
    [12, 13, 14, 11, 15],  
    [20, 18, 17, 19, 16],  
    [1, 2, 3, 4, 5]  
]  
numbers.sort()  
# loop to print the 2D list  
for row in numbers:  
    print(row)
```

Python sorted `numbers` based on the inner lists, but it did not sort the inner lists themselves. In order to use list methods or function on the inner lists, use a for loop.

```
numbers = [  
    [6, 7, 10, 9, 8],  
    [12, 13, 14, 11, 15],  
    [20, 18, 17, 19, 16],  
    [1, 2, 3, 4, 5]  
]  
numbers.sort()  
# loop to sort and print the 2D list  
for row in numbers:  
    row.sort()  
    print(row)
```

challenge

Try this variation:

- Comment out the first `numbers.sort()` and run the program

```
# numbers.sort()  
# loop to sort and print the 2D list  
for row in numbers:  
    row.sort()  
    print(row)
```

▼ Why are the lists out of order?

The first sort method sorts the four inner lists, while the second sort method sorts the inner lists themselves. Since the first sort method has been commented out, the elements of each inner list are in order, the but lists themselves are not in order.

Manipulating a 2D Lists of Numbers

The 2D list `numbers` is a list of lists of numbers. Which means `sum(numbers)` would return an error because the `sum` function expects a list of numbers. To use `sum`, `max` or `min` you need to use a for loop.

```
numbers = [  
    [6, 7, 10, 9, 8],  
    [12, 13, 14, 11, 15],  
    [20, 18, 17, 19, 16],  
    [1, 2, 3, 4, 5]  
]  
  
# loop to print the sum of each inner list  
for row in numbers:  
    print(sum(row))
```

Notice, however, that this does not print the sum of all four inner lists. Instead it prints the sum of each individual list. To find the total of all of the lists, declare the `total` with an initial value of 0. Add the sum of each inner list to `total`. Then print `total`

```
numbers = [  
    [6, 7, 10, 9, 8],  
    [12, 13, 14, 11, 15],  
    [20, 18, 17, 19, 16],  
    [1, 2, 3, 4, 5]  
]  
  
total = 0  
  
# loop to add the sum of each inner list to total  
for row in numbers:  
    total += sum(row)  
print(total)
```

challenge

Try this variation:

- Use the variable `numbers` and write a program that finds the largest number in the 2D list. **Hint:** use the `max` function.

```
numbers = [  
    [6, 7, 10, 9, 8],  
    [12, 13, 14, 11, 15],  
    [20, 18, 17, 19, 16],  
    [1, 2, 3, 4, 5]  
]
```

▼ Solution

```
numbers = [  
    [6, 7, 10, 9, 8],  
    [12, 13, 14, 11, 15],  
    [20, 18, 17, 19, 16],  
    [1, 2, 3, 4, 5]  
]  
  
largest = 0  
for row in numbers:  
    if max(row) > largest:  
        largest = max(row)  
print(largest)
```

Formative Assessment 1

Formative Assessment 2
