Introduction

In this video we're going to learn about Python Lists and how to work with items within the lists we create. Lists do exactly what they say, they store information in one place. A list can be two pieces of information or millions of pieces. You'll come to see and value how powerful lists are as you start to use them. As your Python experience grows you are going to be using lists a lot, and you'll learn how they are used across the Python programming world in areas such as data science and analysis and web development. Let's get started.

What makes a list?

Think of a shopping list, it's made up of items, usually one line at a time. The items on your shopping list are created using letters, numbers, names of products to buy, and maybe the odd exclamation mark. Basically, you can put anything you want on your shopping list and they can be completley unrelated. You could have a shopping list for different shops with different items to buy in each shop on your list. The items on your list don't need to be in any particular order either.

Because lists usually contain more than one item in them it's good practice to name your lists in the plural:

- planes
- trains
- automobiles
- names
- staff
- groceries

Creating lists in Pyton

In Pyton, lists are created using square brackets ([ ]). Items or what we are now going to call elements within a list are seperated by a comma (,). Let's take a look:

In [29]:
```
# Python list example with theree elements
cars = ['ford', 'tesla', 'toyota']
print(cars)
```

```
['ford', 'tesla', 'toyota']
```

If you're an end user, square brackets and quoatation marks, is not the output that you want to see on screen.

How to access individual elements of a list

Every item or element in a list has a position, this position is called the index. You can access each element in a list by telling Python the index that you wish to access. Let's take a look at how this is done:

In [30]:
```
# How to access individual elements of a list
cars = ['ford', 'tesla', 'toyota']
print(cars[0])
```

```
ford
```

As you can see, to access an element of a list, you write the name of the list, followed by the index of the element to access in square brackets. To output the index to screen I have used a print statement. Also note, the element was returned without quotation marks or square brackets.

In a previous video we looked at string methods. You'll be glad to hear that they can be used with lists.

In [31]:
```
# Python lists with string methods
cars = ['ford', 'tesla', 'toyota']
print(cars[1].title())
```

```
Tesla
```

List index

When using and accessing lists in Python, the first item in a list is at position 0, not position 1. In the list of cars above, this means:

Item in list	Position in list
ford	0
tesla	1
toyota	2

You'll find that this ordering of lists is the same in most programming languages.

To access items in our cars list at positions 0 and 2 we write:

In [32]:
```
# Accessing more then one element in a Pyton list
cars = ['ford', 'tesla', 'toyota']
print(cars[0])
print(cars[2])
```

```
ford
toyota
```

Imagine you have a list that contains hundreds or thousands of items and you want to access the last item on the list. Without counting every item and determing the last position in the list how would you do that? Python has a specific syntax for this task. Let's take a look:

In [33]:
```
# Accessing the last element in a Pyton list
cars = ['ford', 'tesla', 'toyota', 'dodge', 'bmw', 'honda', 'jeep']
print(cars[-1])
```

```
jeep
```

Now we can access the last item on the list without knowing how long the list is. As you move into the areas of data science and data analysis you will find this syntax very helpful as you are sometimes given lists that contain millions of items.

We can also use the same negative index syntax to find other items at the end of a list:

In [34]:
```
# Using negative syntax to access elements in a Pyton list
cars = ['ford', 'tesla', 'toyota', 'dodge', 'bmw', 'honda', 'jeep']
print(cars[-2])
print(cars[-3])
```

```
honda
bmw
```

Something important to point out here is that when accessing the last item on a list it is not at position -0. The last item on a list is only accessed with -1.

Using items from a list

Now that you know how to access the indivial items in a list it's time to start learning how to make use of them. Individual items from a list can be used just as would use other variables. We can use f-strings to create a sentence based on items in our list. Let's see how:

In [35]:
```
# Using elements from a list
cars = ['ford', 'tesla', 'toyota', 'dodge', 'bmw', 'honda', 'jeep']
message = f"Welcome to the car showroom, here we sell, {cars[1].title()}, an
print(message)
```

```
Welcome to the car showroom, here we sell, Tesla, and Dodge.
```

In this example we have created a simple welcome message in which we pulled items from positions 1 and 3 of our car lists, assigned this message to a variable and printed it out.

Exercises

1. Create your own list of days of the week and print out each day of the week by individually accessing each item in the list.
2. From your list of days of the week, print out the weekdays and weekend days.
3. From your list of days of the week print a message containing something relevant to you, for example, "On Monday I am going on my holidays.

Changing, adding and removing elements from a list

As you become a more proficient Python programmer the majority of lists that you work with will not be simple static lists as above. You will need to know how to dynamically work with lists. For example, imagine you own a company and have two lists, a product list and a staff list. Both of these lists will change regularly as you add new products and staff members.

Modifying items in a list

To modify elements in a list we will be using similar code to that of accessing items in a list. Let's take look:

```
In [36]: # Modifying an item in a list
fruit_products = ['aple', 'orange', 'lemon', 'lime', 'banana']
print(fruit_products)
```

```
['aple', 'orange', 'lemon', 'lime', 'banana']
```

As you can see in the code above, we have a spelling mistake at index 0. Apple is incorrectly spelt as 'aple'. Let's fix that:

```
In [37]: # Modifying an item in a list
fruit_products = ['aple', 'orange', 'lemon', 'lime', 'banana']
print(fruit_products)

fruit_products[0] = 'apple'
print(fruit_products)
```

```
['aple', 'orange', 'lemon', 'lime', 'banana']
['apple', 'orange', 'lemon', 'lime', 'banana']
```

In the code above, to change an element we use the name of the list followed by the index of the item we want to change, we then assign to it the new value, in this example 'apple'. You can change any element of a list with this approach, not just the first one.

```
In [38]: # Modifying an item in a list
fruit_products = ['aple', 'orange', 'lemon', 'lime', 'banana']
print(fruit_products)

fruit_products[0] = 'tomatoe'
print(fruit_products)
```

```
['aple', 'orange', 'lemon', 'lime', 'banana']
['tomatoe', 'orange', 'lemon', 'lime', 'banana']
```

Adding items to a list

Rather than correct or replace an item in a list as we did in the previous code examples you may want to add a new item to a list. Perhaps a new staff member has joined your company. In Python there are several ways to achieve this, let's take a look.

Appending elements to a list

Using the append() method is one of the easiest ways to add to a list in Python. When you append to a list, the item that you append gets added to the end of the list.

```
In [39]: # Appending an element to a list
team_members = ['jane', 'tony', 'mark']
print(team_members)

team_members.append('mary')
print(team_members)
```

```
['jane', 'tony', 'mark']
['jane', 'tony', 'mark', 'mary']
```

As you can see, we've added the name 'mary' to the list **team_members** without impacting any other items in the list.

Imagine that you have an email list, and it's empty. You ask people to subscribe to your newsletter with their email address. Where do these email addresses go?

In [40]:
```
# Using the append() method on an empty list

#Initialize an empty list
emails = []

# Append email addresses to emails list
emails.append('tony@example.com')
emails.append('jane@test.com')
emails.append('mary@sample.com')

# Print emails list
print(emails)
```

```
['tony@example.com', 'jane@test.com', 'mary@sample.com']
```

Inserting items into a list

We can add new items to a list at any position by using the **insert()** method. Let's take a look:

In [41]:
```
# Insert an element into a list
team_members = ['jane', 'tony', 'mark']
print(team_members)

# Insert a new team member
team_members.insert(0, 'alice')
print(team_members)
```

```
['jane', 'tony', 'mark']
['alice', 'jane', 'tony', 'mark']
```

As mentioned in a previous video, methods act on the code that they are called upon. In this example the **insert()** method has acted on the **team_members** list by moving all items in the list one position to the right and inserting the value passed to the method, 'alice'.

Removing items from a list

How do we remove an item from a list? Let's say a member of our team, 'jane' has left the company. We now need to remove her from our list of team members.

del statement

If you know the position of the item within the list that you would like to remove, you can use the **del** statement.

In [42]:
```
# Removing an item from a list
team_members = ['jane', 'tony', 'mark']
print(team_members)
```

```
del team_members[0]
print(team_members)
```

```
['jane', 'tony', 'mark']
['tony', 'mark']
```

With the **del** statement you can remove an item from any position in a list, so long as you know its postion or index.

In [43]:
```
# Removing an item from a list
team_members = ['jane', 'tony', 'mark']
print(team_members)

del team_members[2]
print(team_members)
```

```
['jane', 'tony', 'mark']
['jane', 'tony']
```

Once you have used the **del** statement in a list you will no longer have access to that element.

Removing an item using the pop() method

In the last code example above, we removed 'mark' from our team members list and by using the **del** statement we no longer have access to that element. But what if we need access to the element? We have removed 'mark' but we may still need to do something with his name such as passing it our HR system or to payroll with a notice of his employment coming to an end.

For when we need access to an element removed from a list we can use the **pop()** method. This method removes the **last** item from a list but allows you to keep working with it. Why is it called pop? Think if a list in Python as a stack, with the pop() method you are popping one item from the end of the stack.

In [44]:
```
# Using Python's pop() method
team_members = ['jane', 'tony', 'mark']
print(team_members)

# Create a new variable
popped_team_members = team_members.pop()

# Print list with last item removed
print(team_members)

# Print variable containing popped item
print(popped_team_members)
```

```
['jane', 'tony', 'mark']
['jane', 'tony']
mark
```

Now that we have access to the popped item, how might we use it. As discussed above, if 'mark' is leaving our company then there could be a number of things that need to be done, such as removing him from payroll.|

In [45]:
```
# Create a new list
team_members = ['jane', 'tony', 'mark']

team_leaver = team_members.pop()
print(f"{team_leaver.title()} has left the company, please remove him from p
```

```
Mark has left the company, please remove him from payroll.
```

Popping items from other positions on a list

So the pop() method is great, if we want to remove the last item from a list. But what about when we want to remove an item that is not the last one in a list, such as the third of forthy first?

As we've already seen, each method in Python ends with parentheses so that we can pass a value to the method before it acts on the code it is called upon. We can include the index of the item that we want to remove in the parentheses of the pop() method.

In [46]:
```
# Create a new list
team_members = ['jane', 'tony', 'mark']

team_leaver = team_members.pop(1)
print(f"{team_leaver.title()} has left the company, please remove him from p
```

```
Tony has left the company, please remove him from payroll.
```

When considering to use the **del** statement or the **pop()** method to remove an item from a list, ask yourself this question, 'will I need access to this element again in the future?' If the answer is no then use the del statement, if the answer is yes then user the pop() method.

Removing items from a list by value

There will be times when you are working with lists so big you won't know what position items are in but you will know the names of the items. Think again of a list of employees which contains 5,000 items. You won't know what position the employees are in but you will know their names, becasue someone will email you to tell you they are leaving.

In [47]:
```
# Create a new list
team_members = ['jane', 'tony', 'mark']
print(team_members)

team_members.remove('jane')
print(team_members)
```

```
['jane', 'tony', 'mark']
['tony', 'mark']
```

When using the remove() method you can continue to work with the value that has been removed.

In [48]:
```
# Create a new list
team_members = ['jane', 'tony', 'mark']
print(team_members)

resigning = 'jane'
team_members.remove(resigning)
```

```
print(team_members)
print(f"{resigning.title()} has decided to resign, please remove her from pa
```

```
['jane', 'tony', 'mark']
['tony', 'mark']
Jane has decided to resign, please remove her from payroll.
```

This code example has a little more going on than previous examples, so let's take a moment to step through it.

- At lines two and three we are simply defining a list of team members and printing that list out
- At line five we have created a new variable called 'resigning' and assigned to it an element from the team_members list
- At line six we have passed the variable **resigning** to the **remove()** method and then used this method on the team_members list
- At line seven we print out the updated team_members list with 'jane' removed
- Finally at line eight we demonstrate that even though the element 'jane' has been removed we can still work with it

The remove() method only removes the first instance of the value passed to it. If our team_members list contained more than one 'jane' then we would have to take a different coding approach to removing all 'jane' elements. We'll look at how to do this later.

Exercises

1. Create a list of friends or family. Print out each person from your list.
2. You've just made a new friend, insert this person at the start of your list using the insert() method.
3. Use the append() method to add another friend to your list.
4. You've misspelled a name on your list, update your list with the correct spelling.
5. Use the del statement to remove a friend from your list
6. Use the pop() method to remove friends from your list at different positions.

How to organize a list in Python

In the real world when working with lists in Python they won't be your lists. Someone is going to send you a list of data that you have to perform some action on. More often than not, after you receive a list, you are going to want to reorder it. In Python there are several ways to order lists, let's take a look.

In [49]:
```
# Using Python's sort() method (permanent)
current_staff = ['tony', 'jane', 'mike', 'phil', 'lisa', 'steve', 'mary']
current_staff.sort()
print(current_staff)
```

```
['jane', 'lisa', 'mary', 'mike', 'phil', 'steve', 'tony']
```

In the code above, we have used Python's **sort()** method to sort our list, current_staff, in aplabetical order. This is one of the easiest ways to sort a list in Python. The same original list can also be sorted in reverse alphabetical oder:

In [50]:
```
# Using Python's sort method in reverse
current_staff = ['tony', 'jane', 'mike', 'phil', 'lisa', 'steve', 'mary']
current_staff.sort(reverse=True)
print(current_staff)
```

```
['tony', 'steve', 'phil', 'mike', 'mary', 'lisa', 'jane']
```

Using Python's sorted() function

There will be times when working with lists that you want to keep the original order but present it to the user in a new sorted order. To achieve this action you can used Python's **sorted()** function. This function allows you to display your list to the user in a particular order, but the order of the list doesnt change. You are only presenting it to the user in a different order. As always, this is easier to understand with an example:

In [51]:
```
# Using Python's sorted() function (temporary)
current_staff = ['tony', 'jane', 'mike', 'phil', 'lisa', 'steve', 'mary']

print("This is a list of current staff in our company:")
print(current_staff)

print("\nHere is a list of current staff in alphabetical order:")
print(sorted(current_staff))

print("\nHere is the original staff list:")
print(current_staff)
```

```
This is a list of current staff in our company:
['tony', 'jane', 'mike', 'phil', 'lisa', 'steve', 'mary']

Here is a list of current staff in alphabetical order:
['jane', 'lisa', 'mary', 'mike', 'phil', 'steve', 'tony']

Here is the original staff list:
['tony', 'jane', 'mike', 'phil', 'lisa', 'steve', 'mary']
```

In [52]:
```
# Sorting a list in reverse alphabetical order
current_staff = ['tony', 'jane', 'mike', 'phil', 'lisa', 'steve', 'mary']
print(sorted(current_staff, reverse=True))
```

```
['tony', 'steve', 'phil', 'mike', 'mary', 'lisa', 'jane']
```

So far our lists have all contained elements in lowercase, it gets a bit more complicated when the elements are uppercase or a mix of both. We'll be taking a look at this in a later lesson.

Reversing a list

To reverse the original order of a list we can use Python's **reverse()** method. Let's take a quick look:

In [53]:
```
# Python's reverse method()
current_staff = ['tony', 'jane', 'mike', 'phil', 'lisa', 'steve', 'mary']
print(current_staff)
```

```
current_staff.reverse()
print(current_staff)
```

```
['tony', 'jane', 'mike', 'phil', 'lisa', 'steve', 'mary']
['mary', 'steve', 'lisa', 'phil', 'mike', 'jane', 'tony']
```

The **reverse()** method doesnt sort backwards alphabetically as in previous examples. It just reverses the order of the list as it is. When you use the reverse() method it alters the list that it is called on permanently. Can you think of how to undo this?

In [54]:
```
# Python's reverse method()
current_staff = ['tony', 'jane', 'mike', 'phil', 'lisa', 'steve', 'mary']
print(current_staff)

current_staff.reverse()
print(current_staff)

current_staff.reverse()
print(current_staff)
```

```
['tony', 'jane', 'mike', 'phil', 'lisa', 'steve', 'mary']
['mary', 'steve', 'lisa', 'phil', 'mike', 'jane', 'tony']
['tony', 'jane', 'mike', 'phil', 'lisa', 'steve', 'mary']
```

Finding the length of a list

As I've mentioned previoulsy, you won't always be the author of the lists that you work with. Most of the time the lists will be given to you. When this happens, and the list massive how can you quickly find its length?

In [55]:
```
# Finding the length of a list
days_of_week = ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'sat
print(len(days_of_week))
```

```
7
```

Exercises

1. Create a new list of anything you want, food, furniture, programming languages, just don't put it in alphabetical order
2. Print your new list out
3. Use Python's **sorted()** function to print your list in alphabetical order
4. Show that your original list list maintains its order compared to your **sorted()** list. Do this by print both lists together.
5. Use Python's **reverse()** method to print your list in reverse. Print out the results
6. Print out the length of your list

Errors

No matter how much you want to avoid them, as you begin to learn more Python and progam more, errors will start to creep into your code. When working with Python a comman error is called the **Index Error**. Below is a list of days of the week:

```
In [56]: # Demonstrating Index Errors
         days_of_week = ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'sat
         print(days_of_week[7])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [56], in <cell line: 3>()
      1 # Demonstrating Index Errors
      2 days_of_week = ['monday', 'tuesday', 'wednesday', 'thursday', 'frid
ay', 'saturday', 'sunday']
----> 3 print(days_of_week[7])

IndexError: list index out of range
```

We've just asked Python to give use the item at index 7, but there is nothing there. The index error means that Python can't find an item at the index location you asked for.

- Remember that lists in Python start at position or index 0
- Don't forget to retrieve the last element from a list you can use [-1]