Episode 3
Lists and Strings

Lists and strings are two of the most important types in Python. For these, one variable name stands for several items, so we call them *compound types*.

Lists

Lists are much like lists in everyday life; they are an ordered sequence where each item can be identified by a number.

Groceries
1. Coffee
2. Sugar
3. Bread
4. Mustard
5. Lettuce

In Python the name of the variable could be `groceries`, and its content would be the items represented as strings:

```
groceries=["coffee","sugar","bread","mustard","lettuce"]
```

To refer to any of the items in our list we write brackets around their number:
`groceries[1]`

One important difference is that in Python we number our items from zero, not one. So `groceries[1]` is `"sugar"`.

Each item is called an *element* and its number is called an *index*. Any element of a list may be of any legal type.

```
breakfast=[3,"spam",2,"eggs",4,"sausage "]
```

List elements can also be lists.

```
L2D=[[1,2],[3,4],[5,6],[7,8]]
```

`L2D[1]` is `[3,4]`
`LD2[1][1]` is `4`

```
We can index lists with variables, but the type of those
variables must be integer.
```

```
In  [1]: a=1.
In  [2]: groceries[a]
TypeError: list indices must be integers, not float
In  [3]: i=1
In  [4]: groceries[i]
```

We can also extract elements with a slice. The colon is called the *range operator*.

```
numbers=[1,2,3,4,5,6,7,8,9,10]
```
`numbers[0]` is `1`
`numbers[:3]` is another list `[1,2,3]`
`numbers[8:]` is `[9,10]`
`numbers[2:5]` is `[3,4,5]`

It is very important to notice that when the upper bound is specified, it is *not included*. This is generally true for Python ranges.

We can use negative indices as shorthand for len(list)-index:
```
numbers[-1] is 10
numbers[-2] is 9
```
In a range, the negative index is like any other second bound and is not included.
```
numbers[3:-1] is [4,5,6,7,8,9]
```

We can provide the range with an increment
```
numbers[1:6:2] is [2,4,6]
```
If the increment is present, the first bound must be present even if it is zero.
```
numbers[0:9:2] is [1,3,5,7,9]
```
The increment can even be negative:
```
numbers[9:5:-1] is [10,9,8,7]
Notice the second bound is not included.
```

A handy way to reverse a list is
```
rnumbers=numbers[::-1]
```

We can add an element to the end of a list with `append`. We prefix it with the name of the list variable, then we put the item to be appended into parentheses.

```
numbers.append(11)
```

We can `extend` a list with another list
```
numbers.extend([12,13,14])
```

We can find the length of the list
```
len(numbers)
```

We can insert an element before a specified index with `insert(index,element)`
`numbers.insert(0,-1)`

Deleting is `del`
`del numbers[12]`

We can join (concatenate) two lists with +
`L2=L0+L1`

We can create a list by starting from an empty list
`new_list=[]`
We'll then have to append to it.

If we know how many elements we want, we can also use replication to set up the list.
`data=[0.]*num_obs`
This creates a list of length `num_obs`, whose value must be known by the time this statement is invoked; all elements of the list are zero and of type floating point.

A useful operator on lists that we did not cover in the videos is the `in` operator. It tests whether some quantity is an element of the list. We use this mostly with the conditionals we will study in the next episode.

```
In  [1]: print 2 in numbers
Out [1]: True
```

There are other list operations, but these are the most commonly used. Any book or reference on Python can give you a complete enumeration.

One important characteristic of lists is that we can change them in place. That is, we can replace an element by something different.

```
numbers[4]=22
```

In Python, a type we can change in place is said to be *mutable*. The main primitive types (numerical types and characters) are *immutable*. We can overwrite them but we do not replace them. When we type
`x+=11`
we are actually deleting the old `x` entirely and creating a new `x`.

Strings

The next type we will study is a *string*. A string is an ordered sequence of characters.

```
"This is a string."
```

We can assign strings to a variable just like any other type, but unlike lists we cannot change any of the characters in that variable. Strings are *immutable*.

```
message="Please test your input values."
message[17:22]="output"    #this is illegal
```

Like lists, strings can be indexed and they count from zero. We can extract slices just like we can for lists; we just can't change them within the original string.

```
in_out=message[17:22]

letter=message[0]
```

We can go backwards in a string:
`message[7::-2]`

As we saw with lists, we can reverse a string with
`message[::-1]`

Strings can also be concatenated using the + sign

```
out_message=message[:17]+"output"+message[22:]
```

You see here that we can extract parts of a string, we can rearrange it, and we can add new strings, but we cannot assign those back to the same string variable; we must use a new variable.

There are a few special characters represented with a backslash \ followed by a letter. The most important of these is `\n`, which represents a newline. Each line in a file is terminated by one or more special characters. Different operating systems use different characters, but `\n` will indicate the end-of-character marker in all operating systems, so your Python scripts can be run on Windows, Mac OSX, or Linux. Another example of these *nonprinting* characters is `\t` for tab.

At the iPython console type

```
string3=”Your\t”+ string1[2:] +”\n”
print string3
string3
```

Now we are ready to start to learn to do interesting things with our variables.

Further References

Google has an introductory tutorial on lists (https://developers.google.com/edu/python/lists), but you may want to wait till after you study the next episode to go through this tutorial. A very complete discussion is at https://www.programiz.com/python-programming/list but it also uses some concepts that we will cover in the next episode.

Strings are a versatile data type and we barely scratched the surface of the string-handling functionality built into Python. Because of this, Python is popular as a text-processing language. A good basic tutorial on strings is at
https://www.tutorialspoint.com/python/python_strings.htm