More about lists

Lists are **zero-indexed**. It means two things:

1. Elements in lists are ordered by a number, starting from left to right (indexed)
2. That index starts with 0 and *not* 1

Indexes are incredibly useful because they unambiguously identify elements.

```
# index      0       1       2       3
animals = ['cat', 'bat', 'rat', 'elephant']
```

In [ ]:

```
animals = ['cat', 'bat', 'rat', 'elephant']
print(animals[2])
```

In [ ]:

```
print(animals[0] + 'woman and ' + animals[1] + 'man')
```

In [ ]:

```
animals[100]
```

Indexing elements in lists containing lists

In [ ]:

```
animals_with_numbers = [['cat', 'bat', 'rat'], [3, 5.0, ['ui', 9]]]
print(animals_with_numbers[1][2][0])
```

If positive indexes finds elements starting from the left what are negative indexes doing?

In [ ]:

```
animals = ['cat', 'bat', 'rat', 'elephant']
print(animals[-1])
```

Negative indexes

Searches the list from right to left instead of from left to right.

In [ ]:

```
animals = ['cat', 'bat', 'rat', 'elephant']
print(animals[-2])
```

In [ ]:

```
animals = ['cat', 'bat', 'rat', 'elephant']
print(animals[-3])
```

In [ ]:

```
print(animals[-1000])
```

A slice of life

Parts of a list (*sublists*) can be split into *slices*. They are called slices because they are *slices* of the actual list, like slices of a piece of bread.

Slices of the list `['cat', 'bat', 'rat', 'elephant']` could be:

- `['cat']`
- `['cat', 'bat']`
- `[]`

Getting Sublists with Slices

- `animals[2]` is a list with an index (one integer)
- `animals[1:4]` is a list with a slice (two integers)

In a slice, the first integer is the index where the slice starts. The second integer is the index where the slice ends. A slice goes up to, but will not include, the value at the second index.

A slice evaluates to a new list value.

In [ ]:

```
animals = ['cat', 'bat', 'rat', 'elephant']
print(animals[1:4])
```

In [ ]:

```
print(animals[0:-1])
```

In [ ]:

```
print(animals[-2:-1])
```

In [ ]:

```
print(animals[:3])
```

Stepping on lists

Slices take every element between the two indexes you give them. But what if you want to step over some of them? You can do that by adding yet another colon (`:`)!

```
>>> [1, 2, 3, 4][::1]
[1, 2, 3, 4]
```

In [5]:

```
['Alex', 'Ina', 'Bob', 'Susan'][::3]
```

Out[5]:

```
['Ina']
```

In [ ]:

```
[1, 2, 3, 4][::2]
```

In [ ]:

```
[1, 2, 3, 4][::3]
```

In [ ]:

```
[1, 2, 3, 4][::-1]
```

In [ ]:

```
[1, 2, 3, 4][::-2]
```

In [ ]:

```
[1, 2, 3, 4][0:3:2]
```

In [ ]:

```
[1, 2, 3, 4][-1:0:-1]
```

In [ ]:

```
[1, 2, 3, 4][-1:2:-1]
```

In [ ]:

```
animals = ['cat', 'bat', 'rat', 'elephant']
print(animals[::2])
```

Getting a List's Length with `len()`

Lists have a length which is the number of elements in the list.

In [6]:

```
fst_sentence = ['Call', 'me', 'Ishmael']
print(len(fst_sentence))
```

```
3
```

In [ ]:

```
print(len([1, 2, 3, 4]))
```

How to convert a string to a list of words?

Earlier, you saw the list of strings

```
fst_sentence = ['Call', 'me', 'Ishmael']
```

But, how do I get that programatically when I only have a string like `'Call me Ishmael'` ?

In [ ]:

```
new_sentence = 'Call me Ishmael'.split()

fst_sentence = ['Call', 'me', 'Ishmael']
new_sentence == fst_sentence
print(new_sentence)
```

Changing Values in a List with Indexes

We saw how we can *get* the first element with `[0]` . Turns out that we can also *set* it! It works just like we would set a variable with `=` :

In [7]:

```
fst_sentence = ['Call', 'me', 'Ishmael']

fst_sentence[1] = 'him'
print(fst_sentence)
fst_sentence[0] = fst_sentence[1]
print(fst_sentence)
```

```
['Call', 'him', 'Ishmael']
['him', 'him', 'Ishmael']
```

In [ ]:

```
fst_sentence[0] = fst_sentence[1]
print(fst_sentence)
```

In [ ]:

```
fst_sentence[-1] = 'what?'
print(fst_sentence)
```

List Concatenation and List Replication

- `+` operator combines two lists to create a new list value
- `*` operator can also be used with a list and an integer value to replicate the list

In [ ]:

```
fst_sentence = ['Call', 'me', 'Ishmael']
numbers = [1, 2, 3, 4]

concat = fst_sentence + numbers
print(concat)
```

In [ ]:

```
fst_sentence * 3
```

Déjà vu?

Where have you seen this type of replication with `*` before?

With strings! Wait.. Does that mean that strings are lists?

In [ ]:

```
'Am I a list?' * 3
```

In [ ]:

```
'Am I a list?'[0:4]
```

In [ ]:

```
'What is the meaning of life'[0:19] + '?'
```

Copying a List

If you change something in a list you destroy it for good. So sometimes you need to copy entire lists to avoid that. In Python you can do this with `[:]`. Think about it like something along: 'give me everything from the beginning to the end, but in a new list'. That actually makes sense because there is no number before the colon (meaning `0`) and no number after the colon (meaning `len(the_list)`).

In [ ]:

```
fst_sentence = ['Call', 'me', 'Ishmael']
' '.join(fst_sentence)
```

In [ ]:

```
fst_sentence = ['Call', 'me', 'Ishmael']

new_sentence = fst_sentence[:]
new_sentence[1] = 'him'

print(fst_sentence)
print(new_sentence)
```