Storing many values until now...

If you want to store many values at the moment you can use many variables. For many tasks this is suboptimal.

In [ ]:

```
animal_1 = 'cat'
animal_2 = 'bat'
animal_3 = 'rat'
animal_4 = 'elephant'
```

Lists - what, are they?

A list is a collection of items in a particular order (also called a sequence). You can make a list that includes the letters of the alphabet, the digits from 0–9, or the names of all the people in your family. You can put anything you want into a list, and the items in your list do not have to be related in any particular way.

Hint: name your lists in plural, such as `letters` , `digits` , or `names`

Square brackets (`[]`) indicate a list, and individual elements in the list are separated by commas.

In [3]:

```
numbers = [1, 2, 3, 1, 2]
print(numbers)
```

```
[1, 2, 3, 1, 2]
```

In [6]:

```
print(type(numbers))
```

```
<class 'list'>
```

In [7]:

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

```
['cat', 'bat', 'rat', 'elephant']
```

In [8]:

```
empty_list = []
print(empty_list)
```

```
[]
```

Mixed values in a list

In some programming language, lists can only contain one type of things. For instance only strings or only numbers, but not a mixture of strings and numbers. In Python, lists can contain anything.

In [9]:

```
animals_with_numbers = ['cat', 'bat', 'rat', 3, 5.0]
print(animals_with_numbers)
```

```
['cat', 'bat', 'rat', 3, 5.0]
```

In [10]:

```
animals_with_weights = [['cat', 'bat', 'rat'],
                        [3, 5.0, 10.2],
                        ['yellow', 'black', 'black']]
print(animals_with_weights)
```

```
[['cat', 'bat', 'rat'], [3, 5.0, 10.2], ['yellow', 'black', 'black']
]
```

Accessing Elements

When you want to look at a single element inside the list. You can do that with square brackets and a number, like `[1]` . Remember that lists are ordered. What do you think this gives us?

In [15]:

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

```
['cat', 'bat', 'rat', 'elephant']
```

Here you learned that 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 [16]:

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

```
catwoman and batman
```

In [17]:

```
animals[100]
```

```
---------------------------------------------------------------------
-------
IndexError                                Traceback (most recent cal
l last)
<ipython-input-17-1341fdb18487> in <module>
----> 1 animals[100]

IndexError: list index out of range
```

Indexing elements in lists containing lists

In [23]:

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

```
bat 5.0
```

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

In [24]:

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

```
elephant
```

Negative indexes

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

In [25]:

```
1 animals = ['cat', 'bat', 'rat', 'elephant']
2 print(animals[-5])
```

```
---------------------------------------------------------------------
-------
IndexError                                Traceback (most recent cal
l last)
<ipython-input-25-32b7ecff3aa2> in <module>
      1 animals = ['cat', 'bat', 'rat', 'elephant']
----> 2 print(animals[-5])

IndexError: list index out of range
```

In [26]:

```
1 print(animals[-3])
```

```
bat
```

In [ ]:

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

Slices

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 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 [28]:

```
animals = ['cat', 'bat', 'rat', 'elephant']
print(animals[1:3])
```

```
['bat', 'rat']
```

In [29]:

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

```
['rat']
```

In [ ]:

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

List deconstruction

Imagine you want to fetch all the elements from a list:

In [30]:

```
fst_sentence = ['Call', 'me', 'Ishmael']
verb = fst_sentence[0]
pronoun = fst_sentence[1]
name = fst_sentence[2]
print(verb)
print(pronoun)
print(name)
```

```
Call
me
Ishmael
```

That is a lot of code to not do very much. Let's fix that. We can deconstruct a list into variables by assigning the values to *multiple* variables:

In [31]:

```
fst_sentence = ['Call', 'me', 'Ishmael']
verb, pronoun, name = fst_sentence

print(verb)
print(pronoun)
print(name)
```

```
Call
me
Ishmael
```

This is also called multiple assignment (because we are assining multiple variable).

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 [36]:

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

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

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

In [37]:

```
print(fst_sentence * 3)
```

```
['Call', 'me', 'Ishmael', 'Call', 'me', 'Ishmael', 'Call', 'me', 'Is
hmael']
```

Removing Values from Lists with del Statements

The `del` statement will delete values at an index in a list. All of the values in the list after the deleted value will be moved up one index.

In [38]:

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

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

```
['Call', 'me']
```

Can we do that with strings as well?

In [39]:

```
book_title = 'Moby Dick'
del book_title[3]
print(book_title)
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-39-bbd5a433287f> in <module>
      1 book_title = 'Moby Dick'
----> 2 del book_title[3]
      3 print(book_title)

TypeError: 'str' object doesn't support item deletion
```

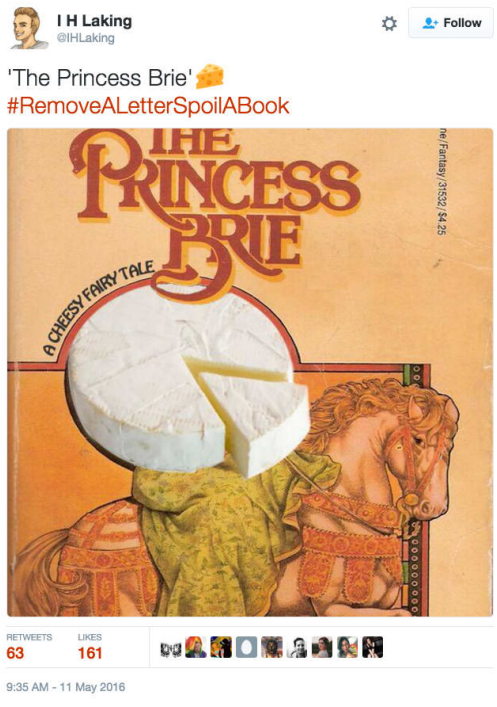
Unfortunately not because strings are *immutable*: they cannot be changed. We'll get back to that later.

We can do something else though:

In [40]:

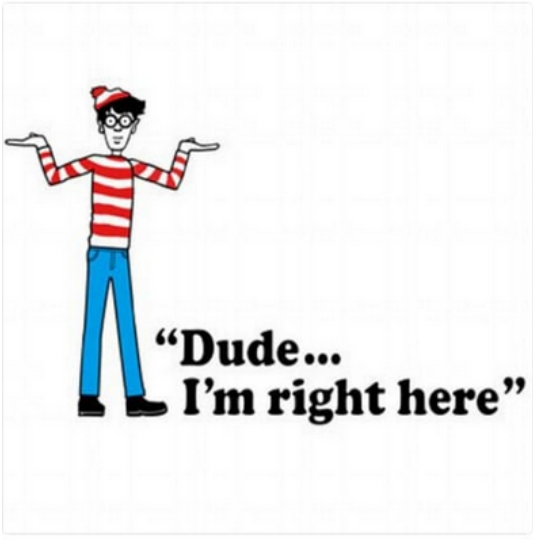
```
book_title = 'The Princess Bride'
book_title = book_title[:16] + book_title[17:]
print(book_title)
```

```
The Princess Brie
```



Your turn: Can you ruin this book title by removing a letter?

```
book_title = 'Where is Waldo?'
```



What is the result of this?

```
book_title = 'Fantastic Beasts and Where to Find them'
book_title = book_title[:13] + book_title[14:]
print(book_title)
```

Amanda Hopkins
@amandaampersand

Follow

Fantastic Beats and Where to Find Them
#RemoveALetterSpoilABook

RETWEETS 248 LIKES 401

11:39 AM - 11 May 2016

In [ ]:

Lists - what, are they?

A list is a collection of items in a particular order (also called a sequence). You can make a list that includes the letters of the alphabet, the digits from 0–9, or the names of all the people in your family. You can put anything you want into a list, and the items in your list do not have to be related in any particular way.

Hint: name your lists in plural, such as `letters` , `digits` , or `names`

Square brackets (`[]`) indicate a list, and individual elements in the list are separated by commas.

In [3]:

```
[1, 2, 3, 1, 2]
```

In [6]:

```
<class 'list'>
```

In [7]:

```
['cat', 'bat', 'rat', 'elephant']
```

In [8]:

```
[]
```

Mixed values in a list

In some programming language, lists can only contain one type of things. For instance only strings or only numbers, but not a mixture of strings and numbers. In Python, lists can contain anything.

In [9]:

```
['cat', 'bat', 'rat', 3, 5.0]
```

In [10]:

```
[['cat', 'bat', 'rat'], [3, 5.0, 10.2], ['yellow', 'black', 'black']
]
```

Accessing Elements

When you want to look at a single element inside the list. You can do that with square brackets and a number, like `[1]` . Remember that lists are ordered. What do you think this gives us?

In [15]:

```
['cat', 'bat', 'rat', 'elephant']
```

Here you learned that 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 [ ]:

In [16]:

```
catwoman and batman
```

In [17]:

```
---------------------------------------------------------------------
-------
IndexError                                Traceback (most recent cal
l last)
<ipython-input-17-1341fdb18487> in <module>
----> 1 animals[100]

IndexError: list index out of range
```

Indexing elements in lists containing lists

In [23]:

```
bat 5.0
```

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

In [24]:

```
elephant
```

Negative indexes

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

In [25]:

```
---------------------------------------------------------------------
-------
IndexError                                Traceback (most recent cal
l last)
<ipython-input-25-32b7ecff3aa2> in <module>
      1 animals = ['cat', 'bat', 'rat', 'elephant']
----> 2 print(animals[-5])

IndexError: list index out of range
```

In [26]:

```
bat
```

In [ ]:

Slices

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 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 [28]:

```
['bat', 'rat']
```

In [29]:

```
['rat']
```

In [ ]:

List deconstruction

Imagine you want to fetch all the elements from a list:

In [30]:

```
Call
me
Ishmael
```

That is a lot of code to not do very much. Let's fix that. We can deconstruct a list into variables by assigning the values to *multiple* variables:

In [31]:

```
Call
me
Ishmael
```

This is also called multiple assignment (because we are assining multiple variable).

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 [36]:

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

In [37]:

```
['Call', 'me', 'Ishmael', 'Call', 'me', 'Ishmael', 'Call', 'me', 'Is
hmael']
```

Removing Values from Lists with del Statements

The `del` statement will delete values at an index in a list. All of the values in the list after the deleted value will be moved up one index.

In [38]:

```
['Call', 'me']
```

Can we do that with strings as well?

In [39]:

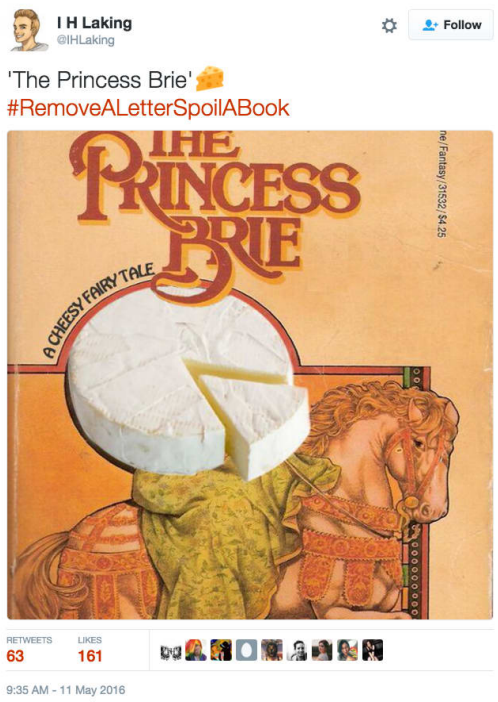
```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-39-bbd5a433287f> in <module>
      1 book_title = 'Moby Dick'
----> 2 del book_title[3]
      3 print(book_title)

TypeError: 'str' object doesn't support item deletion
```

Unfortunately not because strings are *immutable*: they cannot be changed. We'll get back to that later.

We can do something else though:

In [40]:

```
The Princess Brie
```



Your turn: Can you ruin this book title by removing a letter?

```
book_title = 'Where is Waldo?'
```

What is the result of this?

```
book_title = 'Fantastic Beasts and Where to Find them'
book_title = book_title[:13] + book_title[14:]
print(book_title)
```

Amanda Hopkins
@amandaampersand

Follow

Fantastic Beats and Where to Find Them
#RemoveALetterSpoilABook

RETWEETS 248 LIKES 401

11:39 AM - 11 May 2016