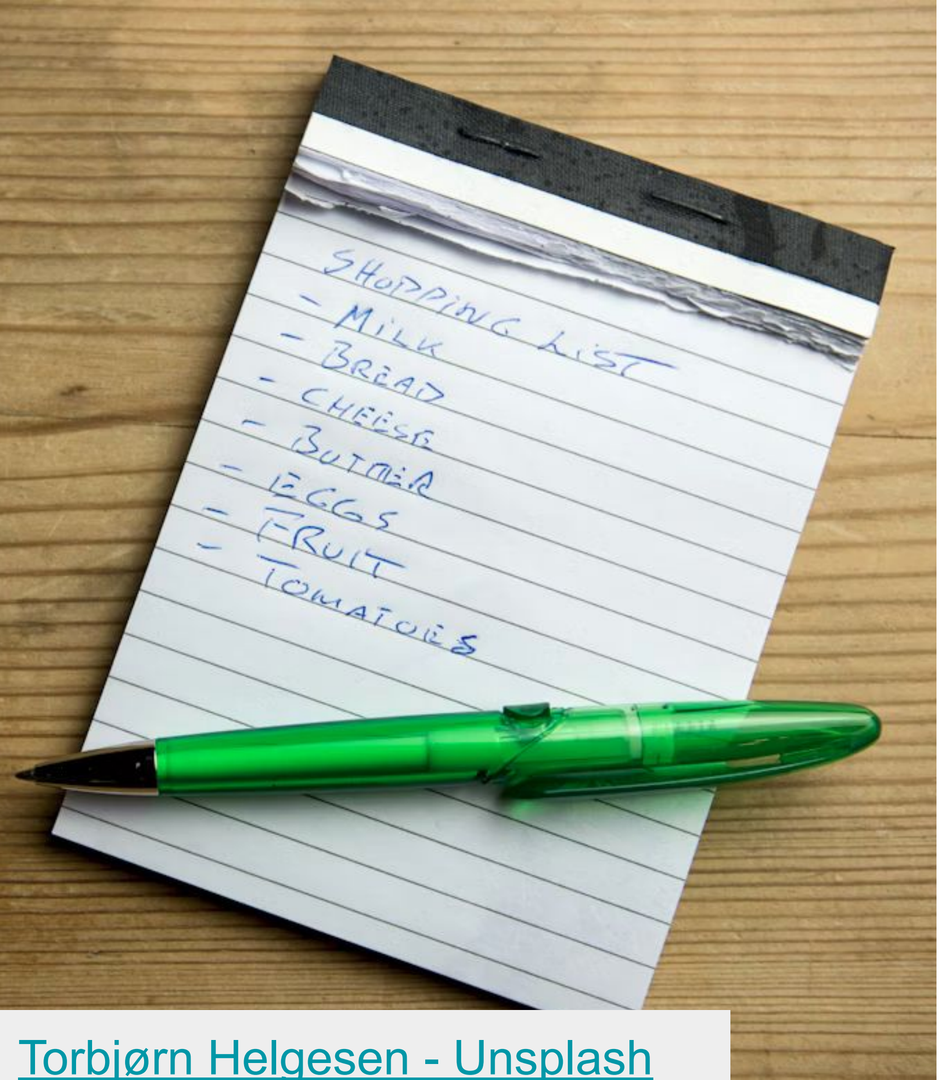

Torbjørn Helgesen - Unsplash

More on lists

Sorting lists

Sorting lists

Imagine we have a list:

```
my_list = [9, 1, 8, 8, 5, 10, 6, 9, 10, 8]
```

Sorting lists

Imagine we have a list:

```
my_list = [9, 1, 8, 8, 5, 10, 6, 9, 10, 8]
```

We can run `my_list.sort()`

Sorting lists

Imagine we have a list:

`my_list = [9, 1, 8, 8, 5, 10, 6, 9, 10, 8]`

We can run `my_list.sort()`

Now, `my_list` is `[1, 5, 6, 8, 8, 8, 9, 9, 10, 10]`

Sorting lists

Imagine we have a list:

`my_list = [9, 1, 8, 8, 5, 10, 6, 9, 10, 8]`

We can run `my_list.sort()`

Now, `my_list` is `[1, 5, 6, 8, 8, 8, 9, 9, 10, 10]`

We have sorted my_list in place!

Sorting lists

Imagine we have a list:

`my_list = [9, 1, 8, 8, 5, 10, 6, 9, 10, 8]`

We can instead run `sorted(my_list)`

Sorting lists

Imagine we have a list:

`my_list = [9, 1, 8, 8, 5, 10, 6, 9, 10, 8]`

We can instead run `sorted(my_list)`

Now, `my_list` is `[9, 1, 8, 8, 5, 10, 6, 9, 10, 8]`

Sorting lists

Imagine we have a list:

```
my_list = [9, 1, 8, 8, 5, 10, 6, 9, 10, 8]
```

We can instead run `sorted(my_list)`

Now, `my_list` is `[9, 1, 8, 8, 5, 10, 6, 9, 10, 8]`

But sorted(my_list) returns a sorted *copy*

```
[1, 5, 6, 8, 8, 8, 9, 9, 10, 10]
```

Sorting lists

Both `list.sort()` and `sorted()` sort to *increasing* order by default:

```
    [1, 5, 6, 8, 8, 8, 9, 9, 10, 10]
```

Sorting lists

Both `list.sort()` and `sorted()` sort to *increasing* order by default:

```
[1, 5, 6, 8, 8, 8, 9, 9, 10, 10]
```

Both functions allow use to flip the order:

Sorting lists

Both `list.sort()` and `sorted()` sort to *increasing* order by default:

```
[1, 5, 6, 8, 8, 8, 9, 9, 10, 10]
```

Both functions allow use to flip the order:

```
my_list.sort(reverse=True)
sorted(my_list, reverse=True)
```

Sorting lists

Both `list.sort()` and `sorted()` sort to *increasing* order by default:

```
[1, 5, 6, 8, 8, 8, 9, 9, 10, 10]
```

Both functions allow use to flip the order:

```
my_list.sort(reverse=True)
sorted(my_list, reverse=True)
```

Our sorted lists will be:

```
[10, 10, 9, 9, 8, 8, 8, 6, 5, 1]
```

Sorting lists

What will this output?

```
letters = ['b', 'C', 'A', 'a', 'B', 'c']
print(sorted(letters))
```

Sorting lists

What will this output?

```
    letters = ['b', 'C', 'A', 'a', 'B', 'c']
    print(sorted(letters))
```

Output: `['A', 'B', 'C', 'a', 'b', 'c']`

Sorting lists

What will this output?

```
letters = ['b', 'C', 'A', 'a', 'B', 'c']
print(sorted(letters))
```

Output: `['A', 'B', 'C', 'a', 'b', 'c']`

What if we want to ignore case?

Sorting lists

What will this output?

```
letters = ['b', 'C', 'A', 'a', 'B', 'c']
print(sorted(letters))
```

Output: `['A', 'B', 'C', 'a', 'b', 'c']`

What if we want to ignore case?

```
print(sorted(letters, key = str.lower))
```

Sorting lists

What will this output?

```
letters = ['b', 'C', 'A', 'a', 'B', 'c']
print(sorted(letters))
```

Output: `['A', 'B', 'C', 'a', 'b', 'c']`

What if we want to ignore case?

```
print(sorted(letters, key = str.lower))
```

Output: `['A', 'a', 'b', 'B', 'C', 'c']`

A more complex key example

```
def my_func(in_str):
    return in_str[-1]
```

A more complex key example

```
def my_func(in_str):
    return in_str[-1]

my_list = ['dog', 'cat', 'doe', 'ant']
```

A more complex key example

```
def my_func(in_str):
    return in_str[-1]

my_list = ['dog', 'cat', 'doe', 'ant']

print(sorted(my_list, key=my_func))
```

A more complex key example

```
def my_func(in_str):
   return in_str[-1]

my_list = ['dog', 'cat', 'doe', 'ant']

print(sorted(my_list, key=my_func))
```

Output: `['doe', 'dog', 'cat', 'ant']`

A more complex key example

```
def my_func(in_str):
    return in_str[-1]

my_list = ['dog', 'cat', 'doe', 'ant']

print(sorted(my_list, key=my_func))
```

Output: `['doe', 'dog', 'cat', 'ant']`

This sorts by last character!

Tuples

Tuples

Tuples are a new *type*

Tuples

Tuples are a new *type*

They look like this:

```
my_tuple = (5, 0, 'dog')
```

Tuples

Tuples are a new *type*

They look like this:

```
my_tuple = (5, 0, 'dog')
```

Tuples are also <u>sequences of values</u>

Tuples

Tuples are a new *type*

They look like this:

```
my_tuple = (5, 0, 'dog')
```

Tuples are also sequences of values

The one difference: **tuples are immutable** (can't be changed)

Converting between sequences

Converting between sequences

Tuple to list: `list(my_tuple)`

Converting between sequences

Tuple to list: `list(my_tuple)`

List to tuple: `tuple(my_list)`

Converting between sequences

Tuple to list: `list(my_tuple)`

List to tuple: `tuple(my_list)`

String to list (one char = one element): `list(my_str)`

Converting between sequences

Tuple to list: `list(my_tuple)`

List to tuple: `tuple(my_list)`

String to list (one char = one element): `list(my_str)`

List to string: well...

Converting between sequences

Tuple to list: `list(my_tuple)`

List to tuple: `tuple(my_list)`

String to list (one char = one element): `list(my_str)`

List to string: well…

```
my_list = ['a', 'b', 'c']
print(str(my_list))
```

Converting between sequences

Tuple to list: `list(my_tuple)`
List to tuple: `tuple(my_list)`
String to list (one char = one element): `list(my_str)`
List to string: well…

```
my_list = ['a', 'b', 'c']
print(str(my_list)) -> ['a', 'b', 'c']
```

Converting between sequences

Tuple to list: `list(my_tuple)`

List to tuple: `tuple(my_list)`

String to list (one char = one element): `list(my_str)`

List to string: well…

```
my_list = ['a', 'b', 'c']
print(str(my_list)) -> ['a', 'b', 'c']
''.join(my_list)
```

Converting between sequences

Tuple to list: `list(my_tuple)`

List to tuple: `tuple(my_list)`

String to list (one char = one element): `list(my_str)`

List to string: well…

```
my_list = ['a', 'b', 'c']
print(str(my_list)) -> ['a', 'b', 'c']
''.join(my_list) -> 'abc'
```

Converting between sequences

Tuple to list: `list(my_tuple)`

List to tuple: `tuple(my_list)`

String to list (one char = one element): `list(my_str)`

List to string: well…

```
my_list = ['a', 'b', 'c']
print(str(my_list)) -> ['a', 'b', 'c']
''.join(my_list) -> 'abc'
```

(only works if list is all strings!)

Converting between sequences

Tuple to list: `list(my_tuple)`

List to tuple: `tuple(my_list)`

String to list (one char = one element): `list(my_str)`

List to string: well…

```
my_list = ['a', 'b', 'c']
print(str(my_list)) -> ['a', 'b', 'c']
''.join(my_list) -> 'abc'  (only works if list is all strings!)
```

If you want to turn another list to a string, use a loop!

Refresher

What does this do?

```
list_1 = [1, 2, 3]
list_2 = [4, 5, 6]
print(list_1 + list_2)
```

Refresher

What does this do?

```
list_1 = [1, 2, 3]
list_2 = [4, 5, 6]
print(list_1 + list_2)
```

Concatenates and outputs: `[1, 2, 3, 4, 5, 6]`

Refresher

What does this do?

```
list_1 = [1, 2, 3]
list_2 = [4, 5, 6]
list_1.append(list_2)
print(list_1)
```

Refresher

What does this do?

```
list_1 = [1, 2, 3]
list_2 = [4, 5, 6]
list_1.append(list_2)
print(list_1)
```

We now have a list in a list: `[1, 2, 3, [4, 5, 6]]`

Nested lists

Assume I have lists that store information about species:

```
species_1 = ["capybara", "mammal", 4, True]
species_2 = ["monitor lizard", "reptile", 4, True]
species_3 = ["chimpanzee", "mammal", 2, True]
species_4 = ["velociraptor", "reptile", 2, False]
```

Nested lists

Assume I have lists that store information about species:

```
species_1 = ["capybara", "mammal", 4, True]
species_2 = ["monitor lizard", "reptile", 4, True]
species_3 = ["chimpanzee", "mammal", 2, True]
species_4 = ["velociraptor", "reptile", 2, False]
```

How do you get the name of species 3?

How do you get the number of legs for species 2?

Nested lists

Assume I have lists that store information about species:

```
species_1 = ["capybara", "mammal", 4, True]
species_2 = ["monitor lizard", "reptile", 4, True]
species_3 = ["chimpanzee", "mammal", 2, True]
species_4 = ["velociraptor", "reptile", 2, False]
```

How do you get the name of species 3? `species_3[0]`

How do you get the number of legs for species 2?

Nested lists

Assume I have lists that store information about species:

```
species_1 = ["capybara", "mammal", 4, True]
species_2 = ["monitor lizard", "reptile", 4, True]
species_3 = ["chimpanzee", "mammal", 2, True]
species_4 = ["velociraptor", "reptile", 2, False]
```

How do you get the name of species 3? `species_3[0]`

How do you get the number of legs for species 2? `species_2[2]`

Nested lists

```
species_1 = ["capybara", "mammal", 4, True]
species_2 = ["monitor lizard", "reptile", 4, True]
species_3 = ["chimpanzee", "mammal", 2, True]
species_4 = ["velociraptor", "reptile", 2, False]
```

Now imagine we combined this lists into one giant list!

Nested lists

```
species_1 = ["capybara", "mammal", 4, True]
species_2 = ["monitor lizard", "reptile", 4, True
species_3 = ["chimpanzee", "mammal", 2, True]
species_4 = ["velociraptor", "reptile", 2, False]
```

Now imagine we combined this lists into one giant list!

```
species_data = [species_1, species_2, species_3, species_4]
```

Nested lists

```
species_1 = ["capybara", "mammal", 4, True]
species_2 = ["monitor lizard", "reptile", 4, True]
species_3 = ["chimpanzee", "mammal", 2, True]
species_4 = ["velociraptor", "reptile", 2, False]
```

Now imagine we combined this lists into one giant list!

```
species_data = [species_1, species_2, species_3, species_4]
```

or

```
species_data = []
species_data.append(species_1)
species_data.append(species_2)
species_data.append(species_3)
species_data.append(species_4)
```

Nested lists

```
species_1 = ["capybara", "mammal", 4, True]
species_2 = ["monitor lizard", "reptile", 4, True
species_3 = ["chimpanzee", "mammal", 2, True]
species_4 = ["velociraptor", "reptile", 2, False]
```

Now imagine we combined this lists into one giant list!

```
species_data = [species_1, species_2, species_3, species_4]
```

What is `species_data[0]`?

What is `species_data[-1]`?

Nested lists

```
species_1 = ["capybara", "mammal", 4, True]
species_2 = ["monitor lizard", "reptile", 4, True]
species_3 = ["chimpanzee", "mammal", 2, True]
species_4 = ["velociraptor", "reptile", 2, False]
```

Now imagine we combined this lists into one giant list!

```
species_data = [species_1, species_2, species_3, species_4]
```

What is `species_data[0]? The capybara info list!`

What is `species_data[-1]? The velociraptor list!`

Nested lists

```
species_1 = ["capybara", "mammal", 4, True]
species_2 = ["monitor lizard", "reptile", 4, True
species_3 = ["chimpanzee", "mammal", 2, True]
species_4 = ["velociraptor", "reptile", 2, False]
```

Now imagine we combined this lists into one giant list!

```
species_data = [species_1, species_2, species_3, species_4]
```

What is `species_data[0]? The capybara info list!`

What is `species_data[-1]? The velociraptor list!`

How do you access the name of the third species?

Nested lists

```
species_1 = ["capybara", "mammal", 4, True]
species_2 = ["monitor lizard", "reptile", 4, True]
species_3 = ["chimpanzee", "mammal", 2, True]
species_4 = ["velociraptor", "reptile", 2, False]
```

Now imagine we combined this lists into one giant list!

```
species_data = [species_1, species_2, species_3, species_4]
```

What is `species_data[0]? The capybara info list!`
What is `species_data[-1]? The velociraptor list!`

How do you access the name of the third species?

```
    species_data[2][0]
```

Nested lists

```
species_1 = ["capybara", "mammal", 4, True]
species_2 = ["monitor lizard", "reptile", 4, True]
species_3 = ["chimpanzee", "mammal", 2, True]
species_4 = ["velociraptor", "reptile", 2, False]
```

Now imagine we combined this lists into one giant list!

```
species_data = [species_1, species_2, species_3, species_4]
```

What is `species_data[0]? The capybara info list!`
What is `species_data[-1]? The velociraptor list!`

How do you access the name of the third species?

```
    species_data[2][0]
```

What would it look like to loop through and print info for all species?

One more list problem

One more list problem

Imagine I have this list:

```
temps = [60, 65, 50, 52]
```

One more list problem

Imagine I have this list:

```
temps = [60, 65, 50, 52]
```

I skipped a day! I need to add 61 here!
How can I do that?

One more list problem

Imagine I have this list:

```
temps = [60, 65, 50, 52]
```

I skipped a day! I need to add 61 here!
How can I do that?

Complicated way:

```
    temps = temps[:2] + [61] + temps[2:]
```

One more list problem

Imagine I have this list:

```
temps = [60, 65, 50, 52]
```

I skipped a day! I need to add 61 here!
How can I do that?

Complicated way:

```
    temps = temps[:2] + [61] + temps[2:]
```

Simpler way:

```
    temps.insert(2, 61)
```