Tuples

Quest

We have reached our final container of the Containers section: **Tuples**! If your first thought when you see that word is "What in the world is a tuple?" don't worry. Tuples are pretty simple containers. You can think of a tuple as a list that you can't change. This means that whatever we assign to the tuple at its declaration is the tuple's final value.

We declare a tuple using parentheses `()`...

```
names = ('Sabeen', 'Justin', 'Mukesh', 'Chen', 'Lucia', 'Devon', 'Yousaf', 'Aisha')
```

...and access values using square brackets `[]` and indexes just like with lists:

```
names[0]
```

If you remember our brief discussions on mutability from the last sections, tuples are another example of an immutable type. If you try to change the value of an element in a tuple, you will get an error 🔴:

```
def main():
    names = ('Sabeen', 'Justin', 'Mukesh', 'Chen',
             'Lucia', 'Devon', 'Yousaf', 'Aisha')

    names[4] = 'Edwin'      # this line will error
    print(names[4])


if __name__ == '__main__':
    main()
```

```
Traceback (most recent call last):
  File "/lib/python3.9/site-packages/_pyodide/_base.py", line 415, in eval_code
    CodeRunner(
  File "/lib/python3.9/site-packages/_pyodide/_base.py", line 296, in run
    coroutine = eval(self.code, globals, locals)
  File "<exec>", line 3, in <module>
  File "unthrow.pyx", line 53, in unthrow.Resumer.run_once
  File "<exec>", line 15, in mainApp
  File "<exec>", line 10, in main
```

▶ Run >_ Hide

You might be wondering why we would want to use tuples. Why use a list that has less functionality? It turns out there are several things that make tuples a useful container in Python.

1. Safety

If we know the values that we want to store in a list ahead of time, it might be safer to store them in a tuple. That way, we know that the values will never be altered accidentally (like by a helper function).

2. Tuples can be dictionary keys

In the last section, we mentioned that dictionary keys must be immutable. This means that lists and dictionaries can't be keys, but what if we want to make a group of values the key? We can just use tuples.

3. Returning multiple things

Tuples also have another special use. Remember back in Multiple Returns when we said that there was a way to return multiple things at once? We meant tuples! Check out this example below:

This is a program to find the equation of a linear line based on coordinates from two points. Don't worry if you are not familiar with the algebra that goes into this. All you need to know is that the equation of a line requires two things: the slope and the intercept. We give you the equations for both in the program below.

```
def get_equation(x1, y1, x2, y2):
    slope = (y2 - y1) / (x2 - x1)
    intercept = y1 - (slope * x1)

    return slope, intercept


def main():
    slope, intercept = get_equation(1, 1, 2, 4)
    print('slope:', slope)
    print('intercept:', intercept)


if __name__ == '__main__':
    main()
```

```
slope: 3.0
intercept: -2.0
```

▶ Run >_ Hide

As you can see above, all that is required to return multiple things is a comma separating each term. Both things are outputted by the function. It is important to note that when setting variables equal to the result of the function call, the number of variables on the left side of the equal sign should match the number of things returned by the function. If you have too many variables, an error will occur 🔴.

```
def get_equation(x1, y1, x2, y2):
    slope = (y2 - y1) / (x2 - x1)
    intercept = y1 - (slope * x1)

    return slope, intercept


def main():
```

```
    slope, intercept, distance = get_equation(1, 1, 2, 4)
    print('slope:', slope)
    print('intercept:', intercept)
    print('distance:', distance)


if __name__ == '__main__':
    main()
```

```
Traceback (most recent call last):
  File "/lib/python3.9/site-packages/_pyodide/_base.py", line 415, in eval_code
    CodeRunner(
  File "/lib/python3.9/site-packages/_pyodide/_base.py", line 296, in run
    coroutine = eval(self.code, globals, locals)
  File "<exec>", line 3, in <module>
  File "unthrow.pyx", line 53, in unthrow.Resumer.run_once
  File "<exec>", line 21, in mainApp
  File "<exec>", line 14, in main
```

▶ Run >_ Hide

Ok, at this point, you may be wondering what exactly all of this has to do with tuples. Well, let's see what happens if we only put one variable on the left side of the assignment operator.

```
def get_equation(x1, y1, x2, y2):
    slope = (y2 - y1) / (x2 - x1)
    intercept = y1 - (slope * x1)

    return slope, intercept


def main():
    equation_vars = get_equation(1, 1, 2, 4)
    print('slope and intercept:', equation_vars)


if __name__ == '__main__':
    main()
```

```
slope and intercept: (3.0, -2.0)
```

▶ Run >_ Hide

As you can see, when you return multiple things you are just returning a tuple! It is almost like there are invisible parentheses around `slope, intercept` in the return statement of `get_equation`.

So, tuples provide safety for storing values we don't want to change and allow us to return multiple values from functions. As of this moment, you have officially learned all three of the containers we are going to teach you! You rock!