# TUPLES AND SETS

# Objectives

- Describe, create and access tuples and sets
- Use built in methods to modify sets and access values in tuples
- Iterate over sets using loops and set comprehensions
- Compare and contrast sets & tuples with lists & dictionaries

# What is a Tuple?

Pronounced *too-pul* or *tupple*. You decide.

An ordered collection or grouping of items!

```
numbers = (1, 2, 3, 4)
```

But it is immutable!

# Immutable?

Can NEVER be changed!

```
x = (1,2,3)
3 in x # True
x[0] = "change me!" # TypeError: 'tuple' object does not support item assignment
```

# Why use a Tuple?

Tuples are faster than lists

It makes your code safer

Valid keys in a dictionary

Some methods return them to you - like .items() when working with dictionaries!

# Creating / Accessing

Create using **()** or the **tuple** function

Accessing is just like a list!

```
first_tuple = (1, 2, 3, 3, 3)

first_tuple[1] // 2
first_tuple[2] // 3
first_tuple[-1] // 3

second_tuple = tuple(5, 1, 2)

second_tuple[0] # 5
second_tuple[-1] # 2
```

# Looping

We can use a *for* loop to iterate over a tuple just like a list!

```
names = ('Colt', 'Blue', 'Rusty', 'Lassie')

for name in names:
    print(name)

# Colt
# Blue
# Rusty
# Lassie
```

# Tuple Methods

There are only two!

# count

Returns the number of times a value appears in a tuple:

```
x = (1,2,3,3,3)
x.count(1) # 1
x.count(3) # 3
```

# index

Returns the index at which a value is found in a tuple.

```
t = (1,2,3,3,3)
t.index(1) # 0
t.index(5) # ValueError: tuple.index(x): x not in tuple
t.index(3) # 2 - only the first matching index is returned
```

YOUR
TURN

# Sets

- Sets are like formal mathematical sets.
- Sets do not have duplicate values
- Elements in sets aren't ordered.
- You cannot access items in a set by index.
- Sets can be useful if you need to keep track of a collection of elements, but don't care about ordering, keys or values and duplicates

# Creating / Accessing

```
# Sets cannot have duplictes
s = set({1, 2, 3, 4, 5, 5, 5}) # {1, 2, 3, 4, 5}

# Creating a new set
s = set({1, 4, 5})

# Creates a set with the same values as above
s = {1, 4, 5}

4 in s
# True

8 in s
# False
```

# Accessing All Values in a Set

A good old for loop!

```
numbers = {1,2,3,4}

for number in numbers:
    print(number)

# 1
# 2
# 3
# 4
```

# Set Methods

Working with sets is very common - there are quite a few things we can do!

# add

Adds an element to a set. If the element is already in the set, the set doesn't change:

```
s = set([1, 2, 3])

s.add(4)

s # {1, 2, 3, 4}

s.add(4)

s # {1, 2, 3, 4}
```

# remove

removes a value from the set - returns a KeyError if the value is not found

```
set1 = {1,2,3,4,5,6}

set1.remove(3)

print(set1) # {1, 2, 4, 5, 6}
```

if you need to avoid KeyErrors use .discard()

# copy

Creates a copy of the set

```
s = set([1,2,3])
another_s = s.copy()
another_s # {1, 2, 3}
another_s is s # False
```

# clear

Removes all the contents of the set

```
s = set([1, 2, 3])

s.clear()

s # set()
```

# Set Math

Sets have quite a few other mathematical methods

Including:

`intersection`

`symmetric_difference`

`union`

# Set Comprehension

```
{x**2 for x in range(10)}

# {0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
```

```
def are_all_vowels_in_string(string):
    return len({char for char in string if char in 'aeiou'}) == 5
```

# Recap

- tuples are ordered collections of elements, they are immutable!
- tuples are faster than lists and useful for protecting data
- sets are unordered collections of unique values
- sets and tuples can be created with {} and () or the set() or tuple() function
- set comprehension is useful when converting other data types to a set

# YOUR TURN