Australian **Computing** Academy

DT Challenge Python

Year 7 Biology Extension

 (https://creativecommons.org/licenses/by/4.0/)

The Australian Digital Technologies Challenges is an initiative of, and funded by the Australian Government Department of Education and Training (https://www.education.gov.au/).

© Australian Government Department of Education and Training.

LISTS

1.1. Biology Extension

1.1.1. Before we begin...

Welcome to the **Biology Extension course**!

This course assumes all knowledge from the ACA Biology DT Challenge (https://groklearning.com/course/aca-dt-7-py-biology/), which covered the following programming concepts:

- printing
- variables
- user input
- f-strings
- if statements

as well as the topics in Biology:

- taxonomy
- common and scientific names

If you haven't done the previous course, then please have a look at it first.

If you've just finished the previous course, great — dive on in!

1.2. Lists of values

1.2.1. Storing multiple values

Imagine writing a program to store a group of reptiles in alphbetical order. Given what we have done so far, you would use a separate variable for each replile name and use each variable individually to `print` them:

```
reptile1 = 'crocodile'
reptile2 = 'lizard'
reptile3 = 'snake'
print(reptile1)
print(reptile2)
print(reptile3)
```

```
crocodile
lizard
snake
```

But what if you had 100 favourite reptiles!? When scientists store information about different animals, they don't want to update every variable each time they change something.

This repetition is a big problem. What if you wanted to add another reptile in the middle? You would have to change each `reptile` variable individually. Argh, the pain!

In the rest of this module, we'll solve these problems (and more) using a Python `list`.

> **Data structures**
>
> Data structures store and organise data so it can be accessed and modified efficiently. Lists are an ordered and changeable collection of related data, and the simplest collection data structures in Python.

1.2.2. Accessing list elements

Let's use a `list` to simplify our program. We can store all of the reptiles in a single `reptiles` variable using a list.

In Python, you can `print` a list to see what its contents are. This is a simple way to check what it contains:

```
reptiles = ['crocodile', 'lizard', 'snake']
print(reptiles)
```

```
['crocodile', 'lizard', 'snake']
```

We can also access each *element* in a list separately:

```
reptiles = ['crocodile', 'lizard', 'snake']
print(reptiles[0])
print(reptiles[1])
print(reptiles[2])
```

```
crocodile
lizard
snake
```

reptiles[0]

Each element in the list is accessed using its *index*, a number that shows where in the list the element is.

Notice how the *first* element is accessed with index **zero**. In Python, we start counting from 0 instead of 1.

1.2.3. Don't go too far!

Watch out! If you go out of bounds and try to check an element that doesn't exist, Python will give you an error.

```
reptiles = ['crocodile', 'lizard', 'snake']
print(reptiles[0])
print(reptiles[1])
print(reptiles[2])
print(reptiles[3])
```

reptiles[0]


```
crocodile
lizard
snake
Traceback (most recent call last):
  File "program.py", line 2, in <module>
    print(reptiles[3])
IndexError: list index out of range
```

1.2.4. Jump right in

The great thing about accessing list elements is that it's really efficient! You can jump straight to the element you're looking for, without having to check them all one at a time.

```
reptiles[1]
```

This means your program runs faster!

1.2.5. Negative indices

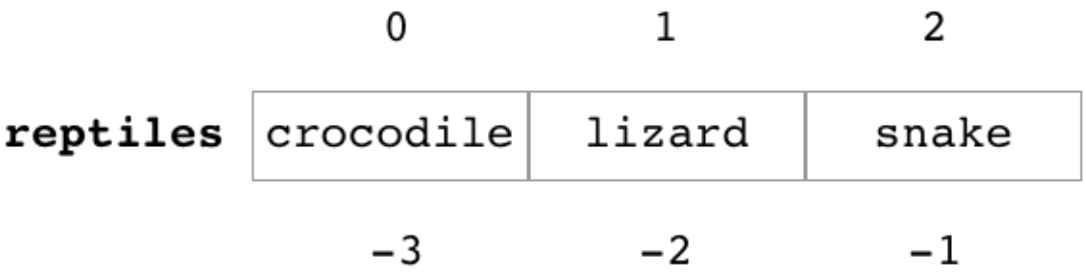
In Python, we can use an *negative index* to count *backwards from the end*.

The *last* item will be `-1`, then *second last* is `-2`, and so on...

```
reptiles = ['crocodile', 'lizard', 'snake']
print(reptiles[-1])
print(reptiles[-2])
print(reptiles[-3])
```

```
snake
lizard
crocodile
```

We can summarise list indexing using the following image:



Positive and negative indices of the `reptiles` list

Indexing

Remember *negative indices* count from the **end**.

1.2.6. Importing data

Scientists often need to import information from other files so they can use it — for example, to process data that is organised in a data file.

To import *everything* from a file we can use the * operator. Here's an example importing from `data.py`:

```
from data import *

print(ants[-1])
```

The first line makes everything in the file `data.py` available for the program to use.

Let's say the contents of the `data.py` file is:

data.py

```
ants = ['fire ant', 'bull ant', 'carpenter ant']
```

Then our program produces the following output:

```
carpenter ant
```

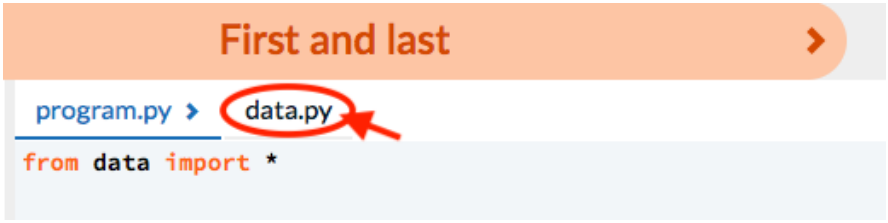
Try changing the contents of the **ants** list and running the example again.

1.2.7. Problem: First and last

The file `data.py` contains a list called `animals`. Your job is to import this list from the file and print out the *first* and *last* items in the list.

The line to do the importing `from data import *` is in your workspace already.

> **Editing the data file**
>
> You can edit the data file to test your program with different inputs. We'll be changing the list when we test your program!



Because the list can change, you'll need to use your knowledge of **indexing** to *always* get the first and last animals in the list.

For the default data file:

data.py
```
animals = ['gecko', 'bull ant', 'crocodile']
```

the output should be:

```
The first animal is: gecko
The last animal is: crocodile
```

Here's another example where the file `data.py` has the following contents:

data.py
```
animals = ['quokka', 'tardigrade']
```

```
The first animal is: quokka
The last animal is: tardigrade
```

You'll need

program.py
```
from data import *
```

data.py

data.py
```
animals = ['gecko', 'bull ant', 'crocodile']
```

Testing

- [ ] Testing the first example from the question.
- [ ] Testing the second example from the question.
- [ ] Testing with a really long list of animals.
- [ ] Testing with a list with only one entry.

- [ ] Testing a hidden case.
- [ ] **Well done! You passed your first Biology Extension question!**

1.2.8. Problem: Kangaroo Kiosk

You've been asked by the local museum to make a Kangaroo Kiosk, where people can find out information about kangaroos! Your program should ask the user what particular information they would like, and then give that information to them.

The common name and Linnaean classification (https://en.wikipedia.org/wiki/Linnaean_taxonomy) information about the kangaroo has been stored in the `kangaroo` list in the file `data.py`. The first line to import this data is given to you.

If a user wants the `'common name'`, they should type that query *exactly*, and the common name is *always* stored in the **first element** of the list. So if our file contains:

data.py
```
kangaroo = ['Red kangaroo', 'Animalia', 'Chordata', 'Mammalia', 'Marsupialia', 'Dipro
```

the output should be:

```
Welcome to the Kangaroo Kiosk
What do you want to know? common name
Red kangaroo
```

If the user asks for the `'scientific name'` instead, you should generate it from the **last two** elements of the list (the second last element is the *genus* and the last is the *species*):

```
Welcome to the Kangaroo Kiosk
What do you want to know? scientific name
Macropus rufus
```

The information stored in the list may vary for different animals, but *the common name will **always** be the **first** element*, and the *genus and species will **always** be the **last two** elements in the list*.

If anything else is typed by the user, you should print `I do not understand.`

```
Welcome to the Kangaroo Kiosk
What do you want to know? scientific
I do not understand.
```

> **You must use indexing!**
>
> **Just like the last question, the data will change** so you will need to use *indexing* to solve this problem in all cases.

You'll need

program.py
```
from data import *
```

data.py

data.py
```
kangaroo = ['Red kangaroo', 'Animalia', 'Chordata', 'Mammalia', 'Marsupialia', 'Dipro
```

Testing

- [ ] Testing the first example from the question.
- [ ] Testing the second example from the question.

- [ ] Testing the third example from the question.
- [ ] Testing with an Eastern grey kangaroo.
- [ ] Testing with the scientific name of the Eastern grey kangaroo.
- [ ] Testing the common name with a longer list (Red kangaroo).
- [ ] Testing the common name with a longer list (Red kangaroo).
- [ ] Testing the common name with a longer list.
- [ ] Testing the scientific name with a longer list.
- [ ] Testing the common name with a shorter list (Red kangaroo).
- [ ] Testing the common name with a shorter list (Red kangaroo).
- [ ] Testing a hidden case #1.
- [ ] Testing a hidden case #2.
- [ ] Testing a hidden case #3.
- [ ] Testing a hidden case #4.
- [ ] **Well done! You're an indexing master!**

1.3. Numbers

1.3.1. Variables as indices

So far we've accessed elements in a list by using a number for their index:

```
reptiles = ['crocodile', 'lizard', 'snake']
print(reptiles[1])
```

```
lizard
```

You can also use a *variable* as a list index — as long as the variable contains an **integer**, Python will substitute that value in as the index:

```
reptiles = ['crocodile', 'lizard', 'snake']
an_index = 0
another_index = -1

print(reptiles[an_index])
print(reptiles[another_index])
```

```
crocodile
snake
```

This allows us to write code that can use different elements in a list depending on the integer that is stored in the variable.

> **Integers**
>
> You may not have heard the term integer before. Integers are **whole numbers**, like 0, 10, -4 and 372. They can be positive or negative, but never contain any fractions or decimal points.

1.3.2. Numbers and input

When you get `input` from the user and store it in a variable, that data is always stored as a string. This means that if the user types in a value like the number 1, it gets stored as the string `'1'`, not the integer `1`.

This is important because list indices are **always** integers, so trying to use a string as an index for a list will not work:

```
reptiles = ['crocodile', 'lizard', 'snake']
index = input('Enter a number: ')
print(reptiles[index])
```

```
Enter a number: 1
Traceback (most recent call last):
  File "program.py", line 3, in <module>
    print(reptiles[index])
TypeError: list indices must be integers or slices, not str
```

What a mess! But it's easy to fix: we just need to convert our `input` from a string (`str` as referred to in the error message) into an integer. The function `int` allows us to do this:

```
reptiles = ['crocodile', 'lizard', 'snake']
index = input('Enter a number: ')
index = int(index)
print(reptiles[index])
```

```
Enter a number: 1
lizard
```

Computers interpret integers and strings in different ways — so if you're getting input from the user and that input needs to be be a *number*, you'll want to make sure you always do the conversion before you use it!

1.3.3. Simple maths on indices

Once we have converted `input` to a number, we can perform mathematical operations on it — like addition:

```
reptiles = ['crocodile', 'lizard', 'snake']
text = input('Type in a number: ')
number = int(text)
print(reptiles[number+1])
```

```
Type in a number: 1
snake
```

or subtraction:

```
reptiles = ['crocodile', 'lizard', 'snake']
text = input('Type in a number: ')
number = int(text)
print(reptiles[number-1])
```

```
Type in a number: 1
crocodile
```

A common situation where you might do this is if you ask people to select from a list of choices, but then you need to convert that choice to an index in a list:

```
reptiles = ['crocodile', 'lizard', 'snake']
choice = input('Choose an animal (1-3): ')
index = int(choice) - 1
print(reptiles[index])
```

(Because, *as we all remember*, in Python the index starts at 0, not 1!)

1.3.4. Problem: List indices

The file `data.py` contains a list called `animals`. Write a program to print out a user's selection from the list.

Your program should take an input number from the user, and print off which element in the list that is — so for an input of **1** your program should print off the **first** element in the list, and so on.

For example, if the file is:

data.py
```
animals = ['dog', 'cat', 'mouse', 'rat']
```

then the program output would look like:

```
Which animal? 1
That animal is: dog
```

Here's another example:

data.py
```
animals = ['kangaroo', 'emu', 'koala']
```

```
Which animal? 3
That animal is: koala
```

> **Use the index**
>
> Remember that lists start at **index 0**, so you will need to convert the input number to the corresponding index of this list.

You'll need

program.py
```
from data import *
```

data.py

data.py
```
animals = ['dog', 'cat', 'mouse', 'rat']
```

Testing

- [ ] Testing the first example in the question.
- [ ] Testing the second example in the question.
- [ ] Testing another input number on the default example.
- [ ] Testing on a long list of animals.
- [ ] Testing a hidden case.
- [ ] **Well done! You found the right index!**

1.3.5. Problem: Fact checking

It's important for scientists to verify the data that they collect to make sure it's correct. Your Biology teacher has asked you to write a program to check the data in a file.

The data file **`data.py`** contains a list called **`animals`**. Your program should ask the user to input a number for the list element that they want to check (so typing '1' means the first item in the list, and so on).

Then the program needs to ask the user to type in what animal they *expect* that list item should be. Finally, the program compares the element in the list with the user's expected data.

If the element in the list matches the expected animal, then the program should say `Correct!`, otherwise the program's output should be `Invalid data.`

For example, if the file contains the list:

data.py
```
animals = ['tree kangaroo', 'quokka', 'eastern grey']
```

then an example program output would look like:

```
Number: 1
Expected output: tree kangaroo
Correct!
```

When the list item *doesn't* match the user's expected animal:

data.py
```
animals = ['tree kangaroo', 'quokka', 'eastern grey']
```

```
Number: 2
Expected output: eastern grey
Invalid data.
```

Your program should work for different contents of the **`animals`** list:

data.py
```
animals = ['kookaburra', 'pigeon', 'ibis']
```

```
Number: 3
Expected output: ibis
Correct!
```

You'll need

program.py
```
from data import *
```

data.py

data.py
```
animals = ['tree kangaroo', 'quokka', 'eastern grey']
```

Testing

- [ ] Testing the first example in the question.
- [ ] Testing the second example in the question.
- [ ] Testing the third example in the question.
- [ ] Testing on some birds.

- [ ] Testing on some other birds.
- [ ] Testing on some reptiles.
- [ ] Testing another reptile.
- [ ] Testing an incorrect reptile.
- [ ] Testing a hidden case.
- [ ] Testing another hidden case.
- [ ] Testing a third hidden case.
- [ ] **Well done, you validated the data!**

1.4. Looking inside lists

1.4.1. Checking if something is in a list

Once we have a list, how can we tell if a particular bit of data is in that list?

Easy! We can check if something is *in* our list using the `in` operator:

```
fish = ['barramundi', 'snapper', 'trout', 'murray cod']
print('snapper' in fish)
```

```
True
```

The `in` operator can be `True` or `False` depending on whether the item is in the list or not — which means we can use it for an `if`:

```
fish = ['barramundi', 'snapper', 'trout', 'murray cod']
if 'snapper' in fish:
  print('"snapper" is in the list!')
```

```
"snapper" is in the list!
```

Remember when we use `if` statements, our condition needs to match what we are testing *exactly*:

```
fish = ['barramundi', 'snapper', 'trout', 'murray cod']
if 'Snapper' in fish:
  print('"snapper" is in the list!')
else:
  print('I could not find "Snapper".')
```

```
I could not find "Snapper".
```

Because `'Snapper'` has a capital letter, it did not match `'snapper'` *exactly*.

1.4.2. Part of an element

When checking if a string is in a list, it will only match if the **whole** exact string is in the list — if only *part* of the string is in the list, it won't match.

For example:

```
fish = ['barramundi', 'snapper', 'trout', 'murray cod']
if 'cod' in fish:
  print('"cod" is in the list!')
else:
  print('"cod" is NOT in the list.')
```

```
"cod" is NOT in the list.
```

1.4.3. Problem: Island animals

You're a biologist managing a list of all the different species you see on the islands that you visit. If you spot an animal, it's important to check whether it has been catalogued in your list already.

We've given you a list of animals in the file `data.py`. Your program needs to import this data, and it has to work properly if it is given *any* list of `animals`.

Here's a sample list of animals:

data.py
```
animals = ['quokka', 'manta ray', 'wallaby', 'dingo',  'dugong', 'koala', 'cuttlefish
```

Write a program to ask for the animal the user has spotted, and then to check the list of species and tell whether that species is already in the list. Your program should work like this:

```
What animal did you see? quokka
Yes! I found a match!
```

If the animal is *not* in the list, then your program should print that it couldn't find it:

```
What animal did you see? emu
No, I couldn't find that animal.
```

The name of the animal must match the name in the list *exactly* — here is an example, where `'mantis shrimp'` is in the list:

```
What animal did you see? shrimp
No, I couldn't find that animal.
```

You'll need

program.py
```
from data import *
```

data.py

data.py
```
animals = ['quokka', 'manta ray', 'wallaby', 'dingo',  'dugong', 'koala', 'cuttlefish
```

Testing

- [ ] Testing the first example from the question.
- [ ] Testing the second example from the question.
- [ ] Testing the third example from the question.
- [ ] Testing when you spot a manta ray.
- [ ] Testing when you spot a wallaby.
- [ ] Testing when you spot a cassowary.
- [ ] Testing when you spot a cat.
- [ ] Testing when you spot a dugong.
- [ ] Testing when you spot a koala.

- [ ] Testing on a different file.
- [ ] Testing another value on the different file.
- [ ] Testing a hidden case.
- [ ] Testing a hidden case.
- [ ] **Well done! You found everything in there!**

1.4.4. Problem: Is it a reptile?

As a biology student, sometimes it is hard to remember if a species belongs to a particular *family* — for example, is a komodo dragon (https://en.wikipedia.org/wiki/Komodo_dragon) a reptile, or not?

For your next exam, you know you will be quizzed on reptiles. Write a program to check if a species you type in is one of the list of reptiles you'll need to remember.

We've given you a list of reptiles in the file **`data.py`** (but your program should work on any **`reptiles`** list):

data.py
```
reptiles = ['frilled-neck lizard', 'crocodile', 'chameleon', 'gecko', 'iguana', 'komo
```

```
Enter an animal: frilled-neck Lizard
Yes, it is a reptile. You should remember it!
```

If the animal is not in the list, then you tell yourself to ignore it by saying `That will not be in the test.`

```
Enter an animal: raptor
That will not be in the test.
```

The name of the reptile must match the name in the list *exactly* — here is an example, where `'komodo dragon'` is in the list:

```
Enter an animal: dragon
That will not be in the test.
```

You'll need

program.py
```
from data import *
```

data.py

data.py
```
reptiles = ['frilled-neck lizard', 'crocodile', 'chameleon', 'gecko', 'iguana', 'komo
```

Testing

- [ ] Testing the first example from the question.
- [ ] Testing the second example from the question.
- [ ] Testing the third example from the question.
- [ ] Testing the crocodile.
- [ ] Testing the chameleon.
- [ ] Testing the velociraptor.
- [ ] Testing the Australian green tree frog.
- [ ] Testing the komodo.
- [ ] Testing the gecko.
- [ ] Testing the iguana.

- [ ] Testing with a different file.
- [ ] Testing a hidden case.
- [ ] **You remembered all the reptiles!**

1.5. Parts of lists

1.5.1. Slicing up lists

Let's say we have a list of animals:

```
animals = ['bat', 'cat', 'dog', 'frog', 'horse', 'quokka']
```

And from this, we want to create a shorter *sub*list — one that contains just the first 3 elements of the original list.

To do this we need to create a *slice* of the `animals` list:

```
animals = ['bat', 'cat', 'dog', 'frog', 'horse', 'quokka']
first_animals = animals[0:3]
print(first_animals)
```

```
['bat', 'cat', 'dog']
```

The slice includes elements starting from the first number (index) and ***up to but not including*** the second index — notice how the slice in the example above, `animals[0:3]`, includes `'dog'` (which is at index 2) but doesn't include `'frog'` (which is at index 3)?

Remember that phrasing — **up to but not including** — and you should avoid making an [off-by-one error (https://en.wikipedia.org/wiki/Off-by-one_error)](https://en.wikipedia.org/wiki/Off-by-one_error) with your second index.

Here are a few more examples of how different values will create different slices:

```
animals = ['bat', 'cat', 'dog', 'frog', 'horse', 'quokka']
print(animals[2:4])
print(animals[0:1])
print(animals[3:-1])
```

```
['dog', 'frog']
['bat']
['frog', 'horse']
```

Experiment with different values (including negative indices) until you're confident with how list slicing works.

1.5.2. Slicing to the end

If we want to create a sublist that goes right to the end of a list, we can do that with a slice if we know how many items are in the list. We can find out the *length* of the list using the `len` function:

```
animals = ['bat', 'cat', 'dog', 'frog', 'horse', 'quokka']
print(animals[2:len(animals)])
print(animals[4:len(animals)])
```

```
['dog', 'frog', 'horse', 'quokka']
['horse', 'quokka']
```

But there's another way!

If we leave out the second index in a slice (but keeping the `:`!) then it will give us the sublist up until the end of the list, just like it did with `len`:

```
animals = ['bat', 'cat', 'dog', 'frog', 'horse', 'quokka']
print(animals[2:])
print(animals[4:])
```

```
['dog', 'frog', 'horse', 'quokka']
['horse', 'quokka']
```

But what happens if we try to find a sublist that doesn't exist?

```
animals = ['bat', 'cat', 'dog', 'frog', 'horse', 'quokka']
new_animals = animals[10:20]
print(new_animals)
```

```
[]
```

The list `animals` contains fewer than 10 elements, so the sublist stored in `new_animals` (i.e. `animals[10:20]`) doesn't exist. So instead, we end up with an empty list.

1.5.3. Sublists are lists too

Since sublists are just lists created from the contents of other lists, you can do all of the same things with the sublists you create that you can do with any other list.

So, you can look inside them for specific contents:

```
animals = ['bat', 'cat', 'dog', 'frog', 'horse', 'quokka']
first_animals = animals[0:3]
print('dog' in first_animals)
print('frog' in first_animals)
```

```
True
False
```

Or you can slice them up even more if you want to:

```
animals = ['bat', 'cat', 'dog', 'frog', 'horse', 'quokka']
some_animals = animals[2:5]
last_animals = some_animals[2:]
print(last_animals)
```

```
['horse']
```

All lists — whether they are made by typing in the list values, or created from other lists — are treated the same way in your programs.

Drop the zero!

If the list slice starts from the beginning, then we don't need to include the `0`:

```
animals = ['frog', 'dog', 'cat', 'mouse']
print(animals[:2])
```

```
['frog', 'dog']
```

1.5.4. Problem: Wingtags

The Royal Botanic Gardens in Sydney have asked you to analyse data that members of the public have collected in the Wingtags (https://www.rbgsyd.nsw.gov.au/science/the-wingtags-project) project. They need you to print out the most recent bird sightings.

009 - Kenickie, the Australian white ibis - Photo: Owen Brasier

Your program should take in a number, and print out that number of items from the beginning of the list.

The data is stored in the `birds` list in the file `data.py`

If the data is:

data.py
```
birds = ['058 - Bozo the Clown', '001 - Cath & Kim', '014 - Popeye']
```

then an example output could be:

```
Number of birds: 2
['058 - Bozo the Clown', '001 - Cath & Kim']
```

If the input number was only 1, the output would look like:

```
Number of birds: 1
['058 - Bozo the clown']
```

On a different data file:

data.py
```
birds = ['017 - Predictable Lad', '039 - Wazza', '081 - Seano', '004 - Cake eater']
```

the program would run like below:

```
Number of birds: 3
['017 - Predictable Lad', '039 - Wazza', '081 - Seano']
```

Printing a list

You can print a whole list by calling `print` on it:

```
abc = ['1', '2', '3']
print(abc)
```

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

You'll need

program.py

```
from data import *
```

data.py

data.py

```
birds = ['058 - Bozo the clown', '001 - Cath & Kim', '014 - Popeye']
```

Testing

- [ ] Testing the first example from the question.
- [ ] Testing the second example from the question.
- [ ] Testing the third example from the question.
- [ ] Testing some Sulphur-crested cockatoos.
- [ ] Testing some Brush turkeys.
- [ ] Testing the third example from the question.
- [ ] **Congratulations! You sliced up the list!**

1.6. Review

1.6.1. Problem: Lists

There are quite a few different situations where we might use a list instead of a variable to store data. Which of the following would be best suited to a list?

Check all that apply

- [ ] A user's name
- [ ] Animals that have been observed in a particular area
- [ ] Observed temperatures for your local city over a monthly period
- [ ] The complete taxonomy of a particular species
- [ ] Observed populations of different cities from around the world in different years

Testing

- [ ] That's right!

1.6.2. Problem: List slicing

Given the following list:

```
birds = ['Ibis','Crane','Flamingo','Eagle','Hawk','Emu','Cassowary','Pigeon','Kookabu
```

which program would generate a final list `animals` with the following contents?

```
birds = ['Eagle','Hawk','Emu','Cassowary']
```

```
animals = birds[:7]
birds = animals[3:]
```

```
birds = birds[:7]
birds = birds[4:]
```

```
animals = birds[:7]
animals = animals[4:]
```

```
birds = birds[:6]
birds = birds[3:len(birds)-1]
```

Testing

That's right!

2

WORKING WITH LISTS

2.1. Adding things to lists

2.1.1. Creating an empty list

Sometimes we don't know how long our list is going to be — in these situations we can create an *empty* list, and add items to it as needed.

```
new_list = []
```

Empty lists have the square brackets notation `[]`, just with no items inside them!

2.1.2. Putting things in lists

Once we've created a list, we can add items to it using `.append` (which adds it to the end).

Here's an example:

```
insects = []
insects.append('fly')
print(insects)
insects.append('mosquito')
print(insects)
```

```
['fly']
['fly', 'mosquito']
```

It's also possible to add variables, instead of the string directly:

```
insects = []
insect_spotted = 'ant'
insects.append(insect_spotted)
print(insects)
```

```
['ant']
```

2.1.3. Adding input to a list

Now that we know how to add variables to a list, we can use that with input from a user to add anything the user types in to a list:

```
insects = []
new_insect = input('What insect did you see? ')
insects.append(new_insect)
print(insects)
```

```
What insect did you see? butterfly
['butterfly']
```

2.2. Separating strings into lists

2.2.1. Constructing a list from input

So far our programs have always asked the user for a single piece of input, either a number or a string, which we've stored in a variable. If we want to get multiple pieces of input from the user, we could store them in a *list*.

To do that, we need a way to *split* the user's input into pieces — we can do this using the `.split` method:

```
data = input('Enter some marsupials: ')
marsupials = data.split()
print(marsupials)
```

```
Enter some marsupials: possum wombat kangaroo
['possum', 'wombat', 'kangaroo']
```

This program asks the user for a list of marsupials, and stores these in a variable `data`. It then uses the `split` method to convert this to a list.

> **The `split` method**
>
> `split` takes an input string and *splits* it between any whitespace characters (which include *spaces*, *tabs* and *newline* characters).

2.2.2. Printing out an element

Because `split` turns a string into a list, you can do all the same things to it as you can with any other list.

For example, if we were only interested in the *second* item typed in, we could get it like this:

```
data = input('Enter some marsupials: ')
marsupials = data.split()
print(marsupials[1])
```

```
Enter your subjects: possum wombat kangaroo
wombat
```

2.2.3. Splitting with anything

By default, the `split` method will use *the space character* (`' '`) to decide where to break a string into a list (known as the *delimiter*).

But we can choose a different delimiter if our data is separated by something else, like a comma (`','`). The comma delimiter can be chosen by using `split(',')` — for example:

```
data = input('Enter some marsupials: ')
marsupials = data.split(',')
print(marsupials)
```

```
Enter your subjects: possum,wombat,kangaroo
['possum', 'wombat', 'kangaroo']
```

You can split a string into a list using *any string* to separate the elements. For example if we have a number of animals separated by the word *and*:

```
text = 'kangaroo and possum and echidna and koala and emu'
```

We can extract just the animals in the following way, where the *delimiter* is `' and '`:

```
text = 'kangaroo and possum and echidna and koala and emu'
animals = text.split(' and ')
print(animals)
```

```
['kangaroo', 'possum', 'echidna', 'koala', 'emu']
```

Delimiters

Delimiter is a very useful word to remember. It just means *the separator between each element*.

2.2.4. Problem: Most awesome animal

It's time to decide which is the most awesome animal!

Write a program that takes a string containing a space (`' '`) seperated list of animals and then **`split`**s the string into a list. Then asks for a number indicating which animal won. The winning number starts from 1, just like a human counting.

Your program should then `print` out the winning animal.

Here's an example of how your program should work:

```
Animals: quokka tardigrade honeybadger octopus
Most awesome: 2
The tardigrade is the most awesome animal!
```

Here's another example:

```
Animals: cuttlefish okapi rhino elephant
Most awesome: 1
The cuttlefish is the most awesome animal!
```

> **Hint!**
>
> The contestant numbers start at 1, but Python list indexes count from 0. You can subtract a number with the – symbol.

Testing

- [ ] Testing the first example from the question.
- [ ] Testing the second example from the question.
- [ ] Testing when there's only one animal.
- [ ] Testing with lots and lots of animals.
- [ ] Testing a hidden case.
- [ ] **Nice job! You're awesome! :D**

2.2.5. Problem: Extracting the scientific name

The scientific name of an animal is the genus and species, which are the **last two** ranks of the taxanomic hierarchy (https://en.wikipedia.org/wiki/Taxonomic_rank).

Write a program that takes a list of taxonomic ranks, and prints out the scientific name. We don't know exactly how many ranks will be listed, but in all cases there will be at least two ranks.

The input will be **separated by a comma and a space**.

Here's an example of how your program should work:

```
Taxonimic ranks: Marsupialia, Diprotodontia, Macropodidae, Macropus, rufus
The scientific name is Macropus rufus.
```

Here's another example:

```
Taxonomic ranks: Macropodidae, Setonix, brachyurus
The scientific name is Setonix brachyurus.
```

> **List indexing**
>
> Remember to use *negative indexes* to count from the end of the list.

Testing

- [ ] Testing the first example from the question.
- [ ] Testing the second example from the question.
- [ ] Testing when there are only two items.
- [ ] Testing with the whole taxonomic tree of a echidna.
- [ ] Testing a hidden case.
- [ ] **Well done! You split off the scientific name!**

2.3. Strings from lists

2.3.1. Joining them back together

So far we have taken a string and `split` it into a list.

We can also go the other way and `join` lists together to make a string!

Let's look at an example!

```
dangerous = ['box jellyfish', 'saltwater crodocile', 'brown snake']
print(','.join(dangerous))
```

```
box jellyfish,saltwater crodocile,brown snake
```

The example above *joins* the strings back together using the comma (`,`) as the delimiter. If we wanted to do it with a *space*...

```
dangerous = ['box jellyfish', 'saltwater crodocile', 'brown snake']
print(' '.join(dangerous))
```

```
box jellyfish saltwater crodocile brown snake
```

If there is only one item in the list, `join` will only print out that item.

```
dangerous = ['box jellyfish']
print(' '.join(dangerous))
```

```
box jellyfish
```

2.3.2. More joining

The result of using the `join` function is a string. So we can change output so it makes a little more sense.

```
dangerous = ['box jellyfish', 'saltwater crodocile', 'brown snake']
danger_string = ', '.join(dangerous)
print(f'These are dangerous animals: {danger_string}!')
```

```
These are dangerous animals: box jellyfish, saltwater crodocile, brown snake!
```

Another neat trick is to join a list with an empty string, or a string of more than one character:

```
values = ['b', 'i', 'o', 'l', 'o', 'g', 'y']
print(''.join(values))
print(' - '.join(values))
```

```
biology
b - i - o - l - o - g - y
```

2.3.3. Problem: Found an ibis!

You're a council worker in Sydney, and you need to print out all the species that you find while walking through the parks. Write a program to print out the animals you see.

Your program should read in the names of the animals, separated by a space, and print out the list.

```
What animals were in the park? seagull
seagull
```

If there wwas more than one animal, it should print out all of the animals separated by a *comma and a space*.

```
What animals were in the park? bat possum
bat, possum
```

Here is another example:

```
What animals were in the park? duck lizard frog seagull
duck, lizard, frog, seagull
```

However, if an `'ibis'` is one of the animals, you will forget about all the other animals that you see, and instead print `Spotted a bin chicken!`

```
What animals were in the park? owl parrot pigeon magpie fox ibis rosella
Spotted a bin chicken!
```

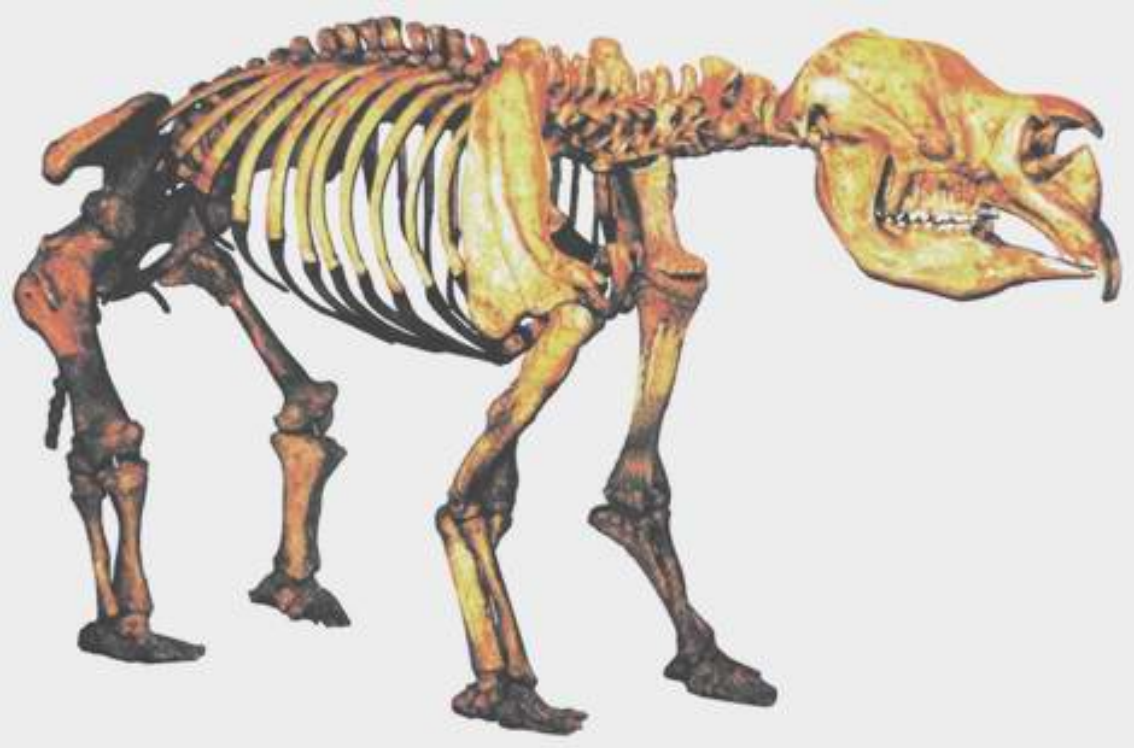
> **`in` a list**
>
> Use the `in` keyword to determine whether an item is in a list.

Testing

- [ ] Testing the first example in the question.
- [ ] Testing the second example in the question.
- [ ] Testing the third example in the question.
- [ ] Testing the fourth example in the question.
- [ ] Testing with multiple ibis inputs.
- [ ] Testing the same input.
- [ ] Testing a hidden case #1.
- [ ] Testing a hidden case #2.
- [ ] Testing a hidden case #3.

2.3.4. Problem: Diprotodon

An archeologist is looking over bone data from different regions trying to find extinct Australian Megafauna, specifically the Diprotodon (https://en.wikipedia.org/wiki/Diprotodon), an animal similar to a wombat but weighing up to 2700kg!



Diprotodon - Image: Wikipedia

The archeologist will paste in some animal names from a database in the following format:

```
<State>:<species one>:<species two>:...
```

Each species will be separated by a colon (`':'`), and the *first* element will always be the location of the data.

If there is a `'Diprotodon'` in the list, you will forget about all the other animals, because we found what we're looking for, and instead print `Diprotodon found in <State>.`

```
Paste in data: QLD:Sarcophilus:Phascolonus:Thylacoleo carnifex:Diprotodon:Obdurodon
Diprotodon found in QLD.
```

If there is only one species and isn't a Diprotodon, it should simply print out the species.

```
Paste in data: WA:Palorchestes
Palorchestes
```

All other entries should be printed with a *comma and a space* separating them.

```
Paste in data: NSW:Zaglossus hacketti:Macropus rufus
Zaglossus hacketti, Macropus rufus
```

Here is another example:

```
Paste in data: VIC:Zygomaturus:Euowenia:Macropus thor:Simosthenurus
Zygomaturus, Euowenia, Macropus thor, Simosthenurus
```

> **Use slicing!**
>
> You will need to use list slicing to ignore the first element, and get the data from all remaining elements.

Testing

- [ ] Testing the first example in the question.
- [ ] Testing the second example in the question.
- [ ] Testing the third example in the question.
- [ ] Testing the fourth example in the question.
- [ ] Testing with multiple Diprotodons.
- [ ] Testing with lots of the same name.
- [ ] Testing a state with no animals.
- [ ] Testing a hidden case #1.
- [ ] Testing a hidden case #2.
- [ ] Testing a hidden case #3.
- [ ] **Well done! You joined them all!**

2.4. Review

2.4.1. Problem: Appending

Which program could produce the following output?

```
Animal 1: ibis
Animal 2: quokka
['ibis', 'quokka']
```

```
animals = []
animal1 = input('Animal 1: ')
animal2 = input('Animal 2: ')
animals.append(animal1)
animals.append(animal2)
print(animals)
```

```
animals = []
animal1 = input('Animal 1: ')
animal2 = input('Animal 2: ')
animals.append(animal2)
animals.append(animal1)
print(animals)
```

```
animal1 = input('Animal 1: ')
animal2 = input('Animal 2: ')
animals.append(animal1)
animals.append(animal2)
print(animals)
```

```
animal1 = input('Animal 1: ')
animal2 = input('Animal 2: ')
print(animal1, animal2)
```

Testing

- [ ] That's right!

2.4.2. Problem: Delimiter

If we wanted to split the following string into a list

```
reptiles = 'crocodie, gecko, frilled-neck lizard'
```

which line of code would create a list without any additional characters other than the animal name?

- ```
  reptiles.split(',')
  ```
- ```
  reptiles.split()
  ```
- ```
  reptiles.split(', ')
  ```
- ```
  reptiles.split(' ')
  ```

Testing

- [ ] That's right!

LOOPING

3.1. For loops

3.1.1. Don't repeat yourself

Computers are not very smart, but they are extremely fast — at doing *exactly* what you tell them to do over, and over, and over again.

Until now, the only way you can repeatedly run code is to duplicate it on a new line.

Using what we've done so far, to output each animal in a list we need a `print` call for each one:

```
animals = input('Enter some animals? ')
animals = animals.split()
print(animals[0])
print(animals[1])
print(animals[2])
print(animals[3])
```

but there are 4 `print`'s here, what happens if we only entered 3 animals?

```
Enter your animals? lion tiger elephant
lion
tiger
elephant
Traceback (most recent call last):
 File "program.py", line 5, in &lt;module&gt;
 print(animals[3])
IndexError: list index out of range
```

The program either stops with an `IndexError` (on fewer animals) or doesn't print out all of the animals (with many animals). How can you solve the problem for animals of any length?

The answer is *loops*!

3.1.2. Using for loops

We can get the Python interpreter to repeat things using `for` loops:

```
animals = input('Enter some animals? ')
animals = animals.split()
for animal in animals:
 print(animal)
```

```
Enter some animals? lizard echidna platypus
lizard
echidna
platypus
```

The `for` loop runs the indented code for each element in a sequence, in this case each animal in the **`animals`** list entered by the user.

If there are three animals, then the `for` loop will run `print(animal)` three times.

Try entering some animals above and seeing what prints out! Each time you enter a space, it will be a different element and be printed on a new line.

3.1.3. Problem: A whole lotta lizards

Write a program to ask for `'Reptiles: '`, which will be a string containing reptiles separated by commas.

Split the reptiles string using the commas to create a reptiles list.

Then, *for* each reptile in the list of reptiles, print the reptile!

You should end up with one reptile per line.

Here's an example:

```
Reptiles: frilled-neck lizard,iguana,Komodo dragon
frilled-neck lizard
iguana
Komodo dragon
```

and another:

```
Reptiles: turtle,snake,chameleon,gecko
turtle
snake
chameleon
gecko
```

Testing

- [ ] Testing the first example in the question.
- [ ] Testing the second example in the question.
- [ ] Testing just a few reptiles.
- [ ] Testing lots of reptiles.
- [ ] Testing a hidden case.
- [ ] **Well done! You'll be loopy in no time!**

3.1.4. Conditions inside loops

Once we are comfortable with looping over a list, we can do more than just print it out!

To check if the item is in a list, you can use an `if` statement:

```
mammals = input('Mammals: ')
mammals = mammals.split()
for animal in mammals:
  if animal == 'quokka':
    print('Saw a quokka!')
```

```
Mammals: quokka kangaroo quokka
Saw a quokka!
Saw a quokka!
```

Note!

This is slightly different from:

```
mammals = input('Mammals: ')
mammals = mammals.split()
if 'quokka' in mammals:
  print('Saw a quokka!')
```

This is because the first example will run the condition *each time* an item is in the list, not just once like in the second example.

3.1.5. Problem: Bird watching

Biologists often have to go bird watching to do studies on how many birds there are in an area.

Write a program to take in a list of animals that you spot, and `print` each one on a new line.

However if you see `'bird'`, you should shout `BIRD!` instead so your colleague can keep count of them.

Here is an example:

```
Animals: rodent rodent bird insect
rodent
rodent
BIRD!
insect
```

and another example:

```
Animals: insect insect insect bird rodent ungulate insect ungulate
insect
insect
insect
BIRD!
rodent
ungulate
insect
ungulate
```

Testing

- [ ] Testing the first example in the question.
- [ ] Testing the second example in the question.
- [ ] Testing not many animals.
- [ ] Testing lots of animals.
- [ ] Testing a hidden case.
- [ ] Testing another hidden case.
- [ ] **Congrats! You can shout with the best of them!**

3.2. Appending in loops

3.2.1. Appending

Often we want to create a list of all the items, but instead of writing many **append** statements, we can put it inside a for loop!

Here's an example of appending each new item to a new list.

```
items = []
data = input('Input some animals: ')
animals = data.split(' ')
for animal in animals:
  items.append(animal)
print(items)
```

This might not seem useful, because we've just replicated the **animals** list, but what if we wanted to make a new list that has an animal *removed*, we could do this:

```
new_animals = []
data = input('Input some animals: ')
remove = input('Item to remove: ')
animals = data.split(' ')
for animal in animals:
  if animal != remove:
    new_animals.append(animal)
print(new_animals)
```

```
Input some animals: bat dog dog pig dog
Item to remove: dog
['bat', 'pig']
```

3.2.2. More appending

Here's another example of appending, if we wanted to only get *unique* items from the animals list!

```
unique = []
data = input('Input some animals: ')
animals = data.split(' ')
for animal in animals:
  if animal not in unique:
    unique.append(animal)
print(' '.join(unique))
```

```
Input some animals: bat quokka dog quokka pig
bat quokka dog pig
```

Notice how `quokka` is only printed once?

3.2.3. Problem: The foreign cuscus

A cuscus is a type of possum found in parts of Indonesia, Papua New Guinea and Northern Australia. They have nice faces but it's not a good idead to get too close, because they can bite.

Ground Cuscus - image: Wikipedia

The Australian Museum needs to print out all of the cuscuses separated by a space, dash and space (`" - "`). You can tell if it is a cuscus by looking for the genus **Phalangar**.

Your program should take in a group of scientific names separated by a comma (`,`), and test each one. If the scientific name contains **Phalanger** then it should be added to the list to be printed.

Here's an example of how it should work:

```
Scientific names: Phalanger vestitus,Phalanger matanim,Phalanger mimicus
Phalanger vestitus - Phalanger matanim - Phalanger mimicus
```

And here's another example

```
Scientific names: Phalanger orientalis,Spilocuscus maculatus,Phalanger ornatus,Phalan
Phalanger orientalis - Phalanger ornatus - Phalanger rothschildi
```

> **Hint!**
>
> Use `in` to check if a string is part of another string.

Testing

- [ ] Testing the first example in the question.
- [ ] Testing the second example in the question.

- [ ] Testing with one Phalanger.
- [ ] Testing a hidden case.
- [ ] **Nice work! You found all the cuscuses!**

3.2.4. Problem: Find the genus

After your success at sorting cuscuses the Australian Musuem wants you to organise all of the unsorted animal specimens in their large collection.

In order to sort out the collection you need to write a program that takes in the scientific names of the animals in your collection (separated by a comma), then takes in a genus and prints all the animals with that specific genus separated by a *comma and a space*.

Here's an example:

```
Scientific names: Macropus rufus,Phascolarctos cinereus,Macropus giganteus
Genus: Macropus
Macropus rufus, Macropus giganteus
```

If there is only one match, it should just print out the scientific name of the algorithm with that genus:

```
Scientific names: Macrotis lagotis,Isoodon macrourus,Macrotis leucura
Genus: Isoodon
Isoodon macrourus
```

No tests will be run without any matches.

> **Hint!**
>
> Use `in` to check if a string is part of another string.

Testing

- [ ] Testing the first example in the question.
- [ ] Testing the second example in the question.
- [ ] Testing different types of bilbies.
- [ ] Testing some pythons.
- [ ] Testing a hidden case.
- [ ] **Well done, you found the common genus!**

3.3. Review

3.3.1. Problem: Looping

Which program could produce the following output?

```
Enter some animals: kangaroo koala echidna gecko
kangaroo
koala
echidna
gecko
```

```
data = input('Enter some animals: ')
animals = data.split()
for a in animals:
  print(a)
```

```
animals = input('Enter some animals: ')
for a in animals:
  print(a)
```

```
animals = input('Enter some animals: ')
data = animals.split()
for a in animals:
  print(a)
```

```
animals = input('Enter some animals: ')
print(animals)
```

Testing

☐ That's right!

4

READING FROM FILES

4.1. Files

4.1.1. Files

So far we've mostly been getting information into our programs using the `input` function, to get a user to type it in.

Scientists often need to get *lots* of information to process, usually some a spreadsheet or data from an equipment, which comes in a text or csv file.

Python makes it really easy to read from files, and for this course we are going to assume all files are in the same *directory*.

4.1.2. Opening files

Just like on a computer, to get data from a file we need to *open* it, and this is done using the `open` function. For example, if a scientist haD the following in a file called `fish.txt`:

fish.txt

```
cod
pufferfish
goldfish
```

We can open this file in Python using the following code:

program.py

```
f = open('fish.txt')
print(f)
```

When run, it produces some weird text:

```
<_io.TextIOWrapper name='test.txt' mode='r' encoding='UTF-8'>
```

Which doesn't actually give us the *contents* of the file, but it does tell us the file is now *open*. We'll look at how to read the contents on the next slide.

If you try and open a file that doesn't exist...

```
f = open('missing.txt')
```

```
Traceback (most recent call last):
  File "program.py", line 1, in &lt;module&gt;
    f = open('missing.txt')
FileNotFoundError: [Errno 2] No such file or directory: 'missing.txt'
```

You get a `FileNotFoundError` exception. More importantly you also get the message `No such file or directory` which tells you what the actual problem was.

4.1.3. Looping over files

Let's learn how to actually open a file! You can treat it just like a list in a `for` loop, it's important to *strip* each line, to remove special characters (if we don't, it will print an extra new-line):

```
f = open('fish.txt')
for line in f:
  print(line.strip())
```

The code above is opening up the following file, `fish.txt`:

fish.txt

```
cod
pufferfish
goldfish
```

You can see the output is exactly the contents of the file! Try changing the file and seeing what happens!

```
cod
pufferfish
goldfish
```

That's all there is to reading in a file!

Now we can make this shorter because we don't need to put the file in a separate variable (`f`), so we can substitute it into the `for` loop itself:

```
for line in open('fish.txt'):
  print(line.strip())
```

4.1.4. The strip function

In the last slide, you may have noticed the `strip` function when we printed each line.

Let's take a look to see exactly what it does

```
for line in open('fish.txt'):
  print(line)
```

fish.txt

```
cod
pufferfish
goldfish
```

Without the `strip` function, the output adds an extra line between each line in the file.

```
cod

pufferfish

goldfish
```

Try changing it to `print(line.strip())` to see the difference.

4.1.5. Problem: Print a file

You've taken some notes on Biology and stored them all in the file **notes.txt**. Eventually you'd like to do some processing on the notes, but first make sure you can open it correctly and print the contents of the file.

Write a program to read the file, **notes.txt** and print out each line, and print the heading `Content of notes.txt:` at the top.

So if the file contrains the lines:

notes.txt
```
Taxonomy - classification of animals
Scientific name - Genus + species
Adaptions - Animals have features that help them survive
```

Then the program should print out the contents of the file exactly

```
Content of notes.txt:
Taxonomy - classification of animals
Scientific name - Genus + species
Adaptions - Animals have features that help them survive
```

Here's another example:

notes.txt
```
Common name
Red kangaroo
Scientific name
Macropus rufus
```

```
Content of notes.txt:
Common name
Red kangaroo
Scientific name
Macropus rufus
```

> **Hint!**
>
> You will need to call **strip** on each line to remove the extra characters.

You'll need

notes.txt

notes.txt
```
Taxonomy - classification of animals
Scientific name - Genus + species
Adaptions - Animals have features that help them survive
```

Testing

- [ ] Testing that the first line is `Content of notes.txt:`.
- [ ] Testing the first example in the question.
- [ ] Testing the second example in the question.
- [ ] Testing a different file.

- [ ] Testing a hidden case.
- [ ] **Congratulations! You opened the file!**

4.1.6. Testing files

Not only can we print out a file, but once we've loaded each line into python, we can run tests on the contents of the file! Let's write our own query system to test if there is the item `kangaroo` in the file `animals.txt`.

```
for line in open('animals.txt'):
  line = line.strip()
  if line == 'kangaroo':
    print('Another roo!')
```

```
Another roo!
```

Here is the contents of the file:

animals.txt

```
koala
lorikeet
kangaroo
wallaby
```

Try running the code window up the top of the page to see the output!

4.1.7. Problem: Under the sea

Some oceanographers have taken some data of different types of fish that they spotted while doing surveys in the Pacific Ocean. One of their data sets is stored in the file **`fish.txt`**.

Write a program to query whether the given type of fish is in one of the scientists data files. All of the files with have the name **`fish.txt`**.

Here's an example, with the file **`fish.txt`** which has *two* instances of **`ocellaris clownfish`** in the file:

```
What fish are we looking up? ocellaris clownfish
Found a ocellaris clownfish!
Found a ocellaris clownfish!
```

Here is an example where **`blue tang`** appears only once in the first data file.

```
What fish are we looking up? blue tang
Found a blue tang!
```

If there is no match, your program should not produce any output.

```
What fish are we looking up? sting ray
```

You'll need

fish.txt

fish.txt

```
ocellaris clownfish
blue tang
puffer fish
manta ray
ocellaris clownfish
snapper
dugong
```

Testing

- [ ] Testing the first example in the question.
- [ ] Testing the second example in the question.
- [ ] Testing the third example in the question.
- [ ] Testing a manta ray.
- [ ] Testing ocellaris clownfish in a different file.
- [ ] Testing the dugong in a different file.
- [ ] Testing a hidden case.
- [ ] Testing another hidden case.
- [ ] **Congratulations! You found all the fish!**

4.2. Comma-separated values

4.2.1. CSV

A CSV file stands for *comma-separated value*, and is a really common way for scientists to store raw scientific data for processing.

If we read the file in Python, we can *split* each line into a *list*, using the comma as a *delimiter*.

```
for line in open('animals.csv'):
  line = line.strip()
  animal = line.split(',')
  common_name = animal[0]
  feature = animal[3]
  print(f'{common_name} has {feature}.')
```

Looking at the data below, it can look a little messy. But if we wanted to print the *common name* and the *feature* of each animal, we just need to know the *index* of each.

In our file the *common name* is index 0, and the *feature* is index 3 in our list.

animals.csv

```
koala,Phascolarctos,cinereus,claws
laughing kookaburra,Dacelo,novaeguineae,feathers
frilled-neck lizard,Chlamydosaurus,kingii,scales
murray cod,Maccullochella,peelii,gills
```

Running the program above gives the following output! **Try chaning the file and running it again!**

```
koala has claws.
laughing kookaburra has feathers.
frilled-neck lizard has scales.
murray cod has gills.
```

Chaining functions

It's possible to chain function calls together. For example:

```
line = line.strip()
animal = line.split(',')
```

could be re-written as

```
animal = line.strip().split(',')
```

4.2.2. Another example

Let's take a look at another example!

If we wanted to print the *scientific name*, it corresponds to the *genus* and *species* of our animals, which in the file `reptiles.csv`, is the *first* and *second* column.

```
for line in open('reptiles.csv'):
  line = line.strip()
  animal = line.split(',')
  genus = animal[0]
  species = animal[1]
  print(f'The scientific name is {genus} {species}.')
```

Because our list starts at index **0**, the values we want are at index 0 and 1.

reptiles.csv

```
Chelonia,mydas,Green sea-turtle
Underwoodisaurus,milii,Thick-tailed gecko
Morelia,viridis,Green tree python
Crocodylus,johnstoni,Freshwater crocodile
Pogona,barbata,Eastern bearded dragon
Tiliqua,occipitalis,Western blue-tongued skink
Chlamydosaurus,kingii,Frilled-neck lizard
```

The output of our program is the following:

```
The scientific name is Chelonia mydas.
The scientific name is Underwoodisaurus milii.
The scientific name is Morelia viridis.
The scientific name is Crocodylus johnstoni.
The scientific name is Pogona barbata.
The scientific name is Tiliqua occipitalis.
The scientific name is Chlamydosaurus kingii.
```

The power of loops over a csv is that it works for files of *any length*.

4.2.3. Problem: Snaaake!

The CSV file format is one of the widely used ways to store scientific data. It's a very convenient format because it's easy to export to programs like Excel, or to do more sophisticated analysis using programming languages like Python.

Write a program which reads from the `snakes.csv` CSV file and asks which column you would like to print out.

The file comes in the format:

```
Common name,Genus,Species,alternate name
```

Here's an example:

```
Which index? 0
Eastern brown snake
Mainland tiger snake
Inland taipan
Mulga snake
```

Your program should work for any index in the file, here is another example:

```
Which index? 3
common brown snake
common tiger snake
small-scaled snake
king brown snake
```

> **Hint!**
>
> Remember to convert the number to an *integer* so you can access the index.
>
> ```
> index = int(text)
> ```
>
> Will do this for you.

You'll need

snakes.csv

```
Eastern brown snake,Pseudonaja,textilis,common brown snake
Mainland tiger snake,Notechis,scutatus,common tiger snake
Inland taipan,Oxyuranus,microlepidotus,small-scaled snake
Mulga snake,Pseudechis,australis,king brown snake
```

Testing

- [ ] Testing the first example in the question.
- [ ] Testing the second example in the question.
- [ ] Testing another index.
- [ ] Testing on a different file.
- [ ] Testing a hidden case.
- [ ] **Congratulations! You turned the csv into lists!**

4.3. Review

4.3.1. Problem: Opening a file

Which is an example of opening the file `file.txt`?

- `f = open('file.txt')`
- `f = open(file.txt)`
- `f = open('file')`
- `f = open()`

Testing

- [ ] That's right!

4.3.2. Problem: CSV files

Given a CSV file `animals.csv` with the contents of:

animals.csv

```
kangaroo,Macropus,rufus
koala,Phascolarctos,cinereus
echidna,Tachyglossus,aculeatus
```

Which code would produce an output of:

```
Index: 0
kangaroo
koala
echidna
```

```
index = input('Index: ')
index = int(index)
for line in open('animals.csv'):
  line = line.strip()
  animal = line.split(',')
  print(animal[index])
```

```
index = input('Index: ')
for line in open('animals.csv'):
  line = line.strip()
  animal = line.split(',')
  print(animal[index])
```

```
index = input('Index: ')
index = int(index)
for line in open('animals.csv'):
  line = line.strip()
  print(line[index])
```

```
index = input('Index: ')
index = int(index)
for line in open('animals.csv'):
  animal = line.split(',')
  print(animal[index])
```

Testing

- [ ] That's right!

5

PROJECT: ANIMAL CLASSIFIER

5.1. Building a classifier

5.1.1. Project overview

In this extension module we've looked at much greater detail as to how Biologists process data.

- reading from CSV files
- accessing items from a list
- slicing a list to make a new list
- looping
- checking if something is `in` a list or a string

In this project we're coming to combine these skills to make a much more sophisticated classifier than what we did in the previous project.

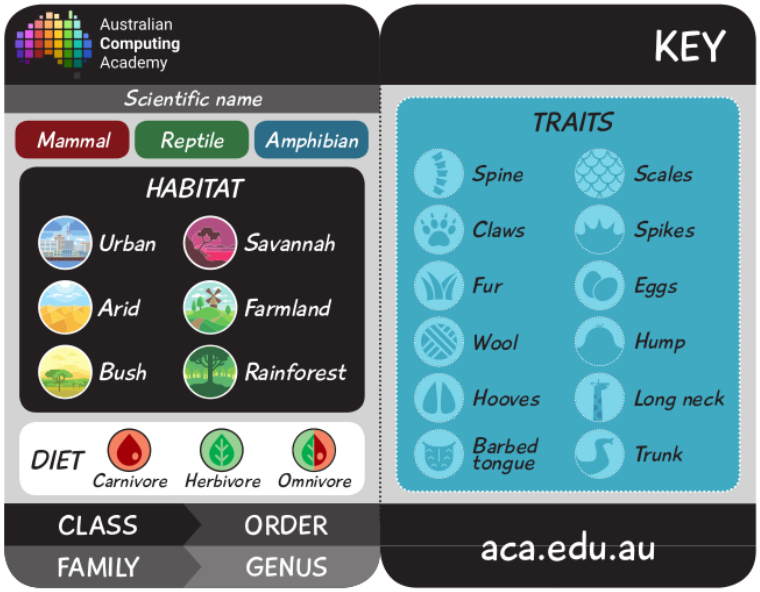
5.1.2. Our data

The data of the animals we need to classify comes in the form of trading cards. The information required about each card can be found from the icons on the card, and the meaning for each icon can be found on the key card.

Download the animal cards (https://groklearning-cdn.com/modules/Ap3LThdG39oPvRq6Tj3hdY/extension_cards.pdf)

Let's take a look at an example, the Australian green tree frog and the key card look like this:

(https://groklearning-cdn.com/modules/Ap3LThdG39oPvRq6Tj3hdY/extension_cards.pdf)



(https://groklearning-cdn.com/modules/Ap3LThdG39oPvRq6Tj3hdY/extension_cards.pdf)

By looking at the key card we can find out all the information required about the frog.

- Common name: Australian green tree frog
- Scientific name: Litoria caerulea
- Class: Amphibia
- Diet: carnivore
- Habitat: rainforest
- Traits: spine, lays eggs

Use the cards to fill in the information on the next question!

5.1.3. Problem: The classification matrix

Fill in the table by researching all of the features of the animals from the animal cards (https://groklearning-cdn.com/modules/Ap3LThdG39oPvRq6Tj3hdY/extension_cards.pdf).

	Spine	Fur	Eggs	Long Neck	Wool	Claws	Spikes	Barbed Tongue	Hoof	Scales	Hump	Trunk
tree frog												
blue-tongue												
cat												
echidna												
sheep												
camel												
elephant												
giraffe												

Testing

- [ ] Checking the "tree frog" row.
- [ ] Checking the "blue-tongue" row.
- [ ] Checking the "cat" row.
- [ ] Checking the "echidna" row.
- [ ] Checking the "sheep" row.
- [ ] Checking the "camel" row.
- [ ] Checking the "elephant" row.
- [ ] Checking the "giraffe" row.

5.1.4. Problem: Biology classifier

In the first biology course you built a classifier that was able to distinguish between four different animals. This time you will use your new found knowledge of lists and files, combined with the knowledge of **dichotomous keys** (https://groklearning.com/learn/aca-dt-7-py-biology/decisions/7/) from the last course, to make a more advanced version that is not only able to classify eight animals, but reads from external data to present more details about the animal you identify.

The program will use the same features and questions you used in the previous question and on our new set of trading cards (https://groklearning-cdn.com/modules/Ap3LThdG39oPvRq6Tj3hdY/extension_cards.pdf).

The program will operate according to the following rules:

The input prompts specify just the feature being checked, followed by a colon, e.g. `Spine:` or `Long Neck:` for two word features. **All feature names possible can be found from the first line of the `species.csv` file, from `Diet` onwards**

- Answers will be `"y"` or `"n"` for all features except:
  - `Diet` will accept `"carnivore"`, `"omnivore"` or `"herbivore"`
  - `Habitat` will accept any string from any of the animal trading cards, all in lower case e.g. `"woodland"`
- Only features from the trading cards are allowed
- The order of the questions does not matter as long as the correct answer is found
- You don't have to use every feature if it is not required
- The same program will work for every animal in the data

Once the animal is identified, you will then print out the following information:

- Common name;
- Scientific name;
- Taxonomic class;
- Diet;
- Habitat;
- The features used to identify it (when the users answer matched the question) *separated by a comma and a space*
  - If a feature doesn't match the user input, then you should negate the feature by putting the word `not` in front of it.

The information will be formatted as shown in the examples below:

```
Diet: herbivore
Habitat: farmland
Common name: sheep
Scientific name: Ovis aries
Taxonomic class: Mammalia
Diet: herbivore
Habitat: farmland
Identified by: herbivore, farmland
```

Here is another correct example:

```
Eggs: y
Diet: carnivorous
Spikes: n
Common name: Australian green tree frog
Scientific name: Litoria caerulea
Taxonomic class: Amphibia
Diet: carnivore
Habitat: rainforest
Identified by: Eggs, carnivorous, not Spikes
```

It does not matter how many questions or features you use to identify each animal as long as you identify each one *uniquely* and *correctly*.

You must format the output of your program as shown in the examples above. You can assume your program will always find a match.

Once the animal is identified, you should save the animal's `Scientific Name` in a variable, and use this value to extract the relevant data from the file `species.csv` into a list. You can then use the data in the list to generate your correctly formatted output.

You can view or download the contents of this file by clicking on its tab in the editor.

> **Plan your approach**
>
> There are quite a few things you need to tackle in this problem. You may find it easier to work on each part separately before putting it all together. For example, you might want to extract the information for a single animal and format it correctly, and **pass the first test-case**, then see if it works when you change the name. From there, write your classification questions to work out which name to use to extract the data.

You'll need

species.csv

```
Scientific Name,Common Name,Kingdom,Phylum,Class,Order,Family,Genus,Species,Diet,Habita
Litoria caerulea,Australian green tree frog,Animalia,Chordata,Amphibia,Anura,Hylidae,Li
Tiliqua scincoides,eastern blue-tongue,Animalia,Chordata,Reptilia,Squamata,Scincidae,Ti
Felis catus,domestic cat,Animalia,Chordata,Mammalia,Carnivora,Felidae,Felis,catus,carni
Tachyglossus aculeatus,short-beaked echidna,Animalia,Chordata,Mammalia,Monotremata,Tach
Ovis aries,sheep,Animalia,Chordata,Mammalia,Artiodactyla,Bovidae,Ovis,aries,herbivorous
Camelus dromedarius,one-humped camel,Animalia,Chordata,Mammalia,Artiodactyla,Camelidae,
Loxodonta africana,African bush elephant,Animalia,Chordata,Mammalia,Proboscidea,Elephar
Giraffa camelopardalis,northern giraffe,Animalia,Chordata,Mammalia,Artiodactyla,Giraffi
```

Testing

- [ ] Testing the example from the question (sheep).
- [ ] Testing the second example in the question - Litoria caerulea (Australian green tree frog).
- [ ] Testing Felis catus (domestic cat).
- [ ] Testing Camelus dromedarius (one-humped camel).
- [ ] Testing Tiliqua scincoides (eastern blue-tongue).
- [ ] Testing Tachyglossus aculeatus (short-beaked echidna).
- [ ] Testing Loxodonta africana (African bush elephant).
- [ ] Testing Giraffa camelopardalis (northern giraffe).
- [ ] Testing whether the common name is being looked up from the file data.

- [ ] Testing whether the common name is being looked up from the file data.
- [ ] **Amazing! You made an advanced Biology Classifier that reads from files! High five!**

5.2. Congratulations!

5.2.1. Congratulations!

Well done on completing the extension challenge!

After completing this course, you should have a much better idea of how scientists read and process information from data files.

If you want to look further, investigate the python CSV module (https://docs.python.org/3/library/csv.html), and some more complex data structures (https://docs.python.org/3/tutorial/datastructures.html) that help process that information.

To learn more python, try our Python Chatbot course (https://groklearning.com/course/aca-dt-78-py-chatbot/)!

5.2.2. Problem: Classifier Playground!

What's this?! You thought you were finished?

Why not have a go at writing some code to classify other animals? You can write whatever code you like in this question. Consider it your personal classifier playground!

> **Save or submit your code!**
>
> There are no points to be earned for this question, so you can submit whatever code you like. Make sure you save programs that you want to keep!

Testing

- [ ] This question is a playground question! There's no right or wrong.