Australian Computing Academy

DT Challenge Blockly

Year 5 Biology

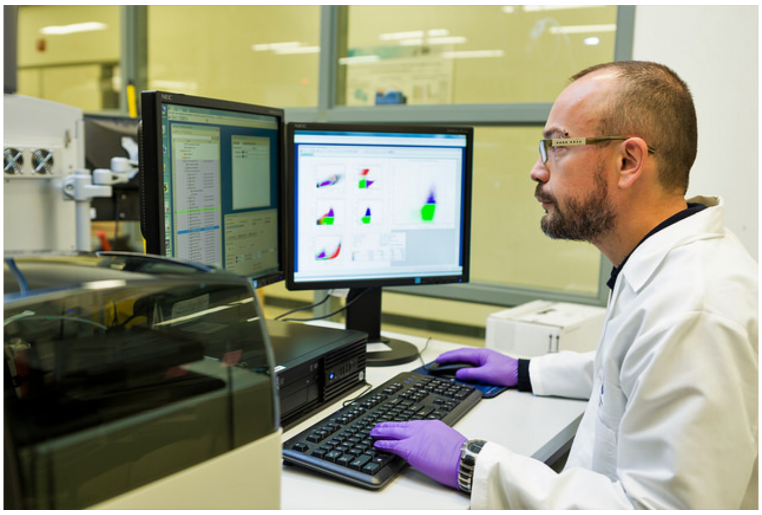
1 OUTPUT

1.1. Getting started

1.1.1. Why biology and Programming?

In this course you'll learn two things:

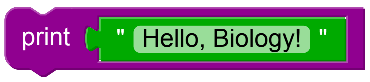
- how animals have different features and adapt to their environment; and
- writing computer programs with Blockly


Biologists use computers to analyse data. Daniel Soñé Photography, LLC (https://www.flickr.com/photos/64860478@N05/38997905444/) CC BY 2.0

Scientists rely on computers to do lots of data processing and analysis for them. This data processing is all performed with computer programs!

1.1.2. Hello, Biology!

Normally, the first program you write when learning a new programming language is Hello, World! (https://en.wikipedia.org/wiki/Hello_world_program), but we're going to write a slightly different one instead:



```
print " Hello, Biology! "
```

```
Hello, Biology!
```

You can run it by clicking the ▶ button (above).

When you run the program, you can see that it writes a message. That's what the print block does, it prints messages.

Writing Blockly or Python?

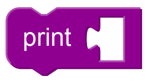
Blockly uses *visual programming*, like Scratch (https://scratch.mit.edu/). The blocks are code, just like any real-world language, such as Python (https://python.org).

You can see the Python version by clicking the button.

If you want to learn Python, try the Python (https://groklearning.com/course/aca-dt-7-py-biology/) version instead.

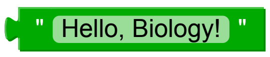
1.1.3. How to write programs

What do the coloured blocks mean?



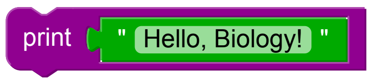
The print block is an *instruction* for the computer to follow. The hole in the block means it needs extra information to do its job.

You need to tell the computer what to print. Here is a message:



This message block doesn't do anything by itself. It's just a message, not an *instruction*. So if you press play, nothing happens!

Put them together and you have an instruction that the computer can understand and run:



```
Hello, Biology!
```

1.1.4. Problem: Hello, Biology!

Time to solve your first problem! Write a program that prints out:

```
Hello, Biology!
```

What, again? **Yes, now it's your turn to write it from scratch.**

You need to put your blocks in the editor (the big area on the right).

The print block is in the **Output** menu. Drag it into the editor.

Add an empty " " string block from the **Strings** menu. Drag it into the hole in the print block, and change the message.

How do I submit?

1. Write your program in the editor (large panel on the right);
2. Run your program by clicking ▶ Run in the top right-hand menu bar. The output will appear below your code. **Check the program works correctly!**
3. Mark your program by clicking ★ Mark and we will automatically check if your program is correct, and if not, give you some hints to fix it up.

Testing

☐ Testing that the words are correct.

☐ Testing that the whitespace is correct.

☐ Testing that the punctuation is correct.

☐ Testing that the capitalisation is correct.

☐ **Hurrah, you got *everything* right!**

1.1.5. Problem: Do you know Mr DNA?

In Jurassic Park (http://www.imdb.com/title/tt0107290/), the park scientists were able to re-create extinct animals using their DNA (https://en.wikipedia.org/wiki/DNA).

Scientists don't believe this would actually be possible (http://www.iflscience.com/technology/could-jurassic-park-ever-come-true/), but DNA does tell us how living things have changed over time.

DNA has a double-helix structure and contains your unique genetic code

In the movie there is an educational video to explain how scientists used DNA to bring dinosaurs back to life. The video introduces Mr. DNA with the following phrase:

```
Oh, Mr. DNA! Where did you come from?
```

Write a program that prints this same welcome to the screen.

Remember that the marker is really picky about punctuation and spelling.

You don't have to type this out!

Remember what we said on the last slide about making sure your question meets the requirements exactly?

Testing

- ☐ Testing that the words are correct.
- ☐ Testing that the whitespace is correct.
- ☐ Testing that the punctuation is correct.
- ☐ Testing that the capitalisation is correct.
- ☐ **Nice work! You got *everything* right!**

1.2. Printing multiple times

1.2.1. Adaptation

In order to survive in their environments, plants and animals develop both *behavioural* and *structural* adaptations. This adaptation occurs over many generations.

Migrating waders in Roebuck Bay, Western Australia
By Mdk572 (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons (https://commons.wikimedia.org/wiki/File%3AWaders_in_flight_Roebuck_Bay.jpg)

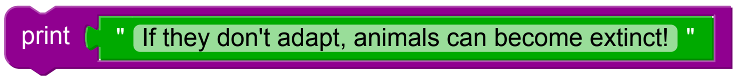
For example, the migration of birds across countries is a *behavioural* adaptation.

A camel storing fat in its hump allows it to survive for days without eating in the desert, and is an example of a *structural* adaptation.

1.2.2. A string of characters

When we want to print things out on the screen, we need to let the computer know how to understand it. We do this by using a *string*.

The green string block can contain any letters, numbers, punctuation and spaces that you want to use in a message:



print " If they don't adapt, animals can become extinct! "

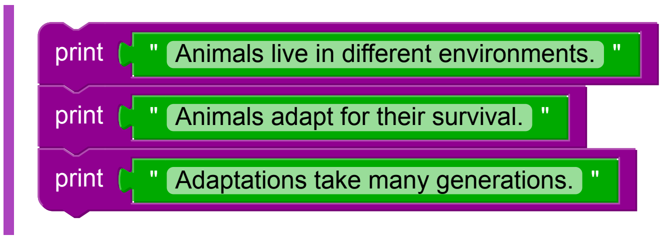
```
If they don't adapt, animals can become extinct!
```

The individual letters, digits, symbols and spaces are called *characters* and the word string is short for *string of characters*.

1.2.3. Printing multiple times

Our programs would be pretty boring if they only printed one thing!

We can add multiple print statements to print multiple lines! Blockly will run the statements in order, so to print multiple lines you can use:



```
print " Animals live in different environments. "
print " Animals adapt for their survival. "
print " Adaptations take many generations. "
```

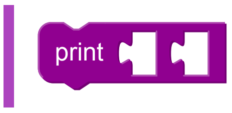
```
Animals live in different environments.
Animals adapt for their survival.
Adaptations take many generations.
```

Each message will be printed on it's own line, **run the example to check**!

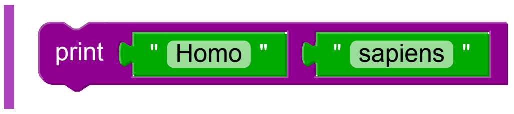
1.2.4. More print blocks

What if you want to print more than one message on a line?

Blockly has print blocks with more than one hole, e.g.:



You can use these with multiple strings, like this:



```
print " Homo " " sapiens "
```

```
Homo sapiens
```

When you run it, notice that the output is not `Homosapiens`.

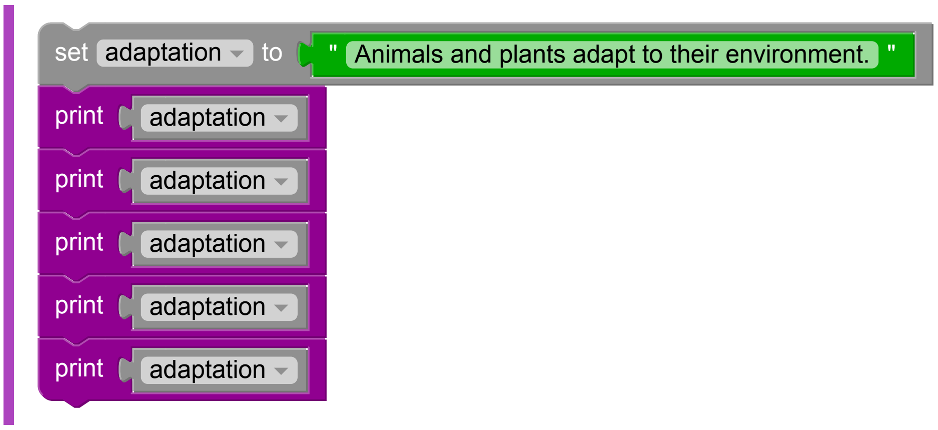
The print automatically adds a space between the two strings.

1.2.5. Storing things in variables

Writing out a long message many times is a pain. It would be great if we could just store the message somewhere and reuse it.

A *variable* is that place for storing a value so we can use it later.

Each variable has a *name* which we use to set and get its value. We create a new variable using a set block:


```
set adaptation to "Animals and plants adapt to their environment."
print adaptation
print adaptation
print adaptation
print adaptation
print adaptation
```

```
Animals and plants adapt to their environment.
Animals and plants adapt to their environment.
Animals and plants adapt to their environment.
Animals and plants adapt to their environment.
Animals and plants adapt to their environment.
```

The `set adaptation` block creates a new variable called **`adaptation`**. It holds the message `"Animals and plants adapt to their environment."`. We can then use the `adaptation` variable to print that message as often as we want.

Creating a variable

To create a new variable, click the down arrow next to the variable name and select **New Variable...**

1.2.6. Problem: Organisms

Sometimes, it's hard to remember a fact or statement. One way to do it is to repeat it multiple times so that you can remember!

We've written a program in the editor that repeats a statement.

Click ▶ Run to see what it does.

The statement is stored in a variable called `remember`.

Update this program so that it works for a different statement: `'Structural adaptations are physical features.'` Your updated program should print the message:

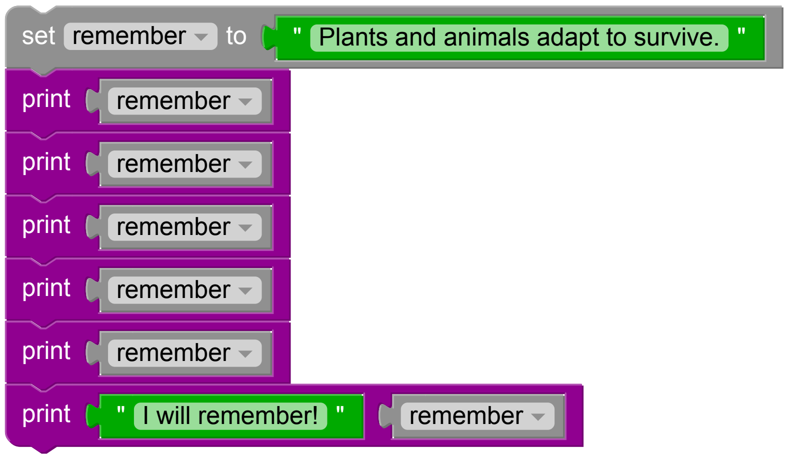
```
Structural adaptations are physical features.
Structural adaptations are physical features.
Structural adaptations are physical features.
Structural adaptations are physical features.
Structural adaptations are physical features.
I will remember! Structural adaptations are physical features.
```

Only change the remember variable!

You just need to change the value of remember to be `'Structural adaptations are physical features.'` instead of `'Plants and animals adapt to survive.'`, run it to check it works, and then mark it.

You'll need

program.blockly



```
set remember to " Plants and animals adapt to survive. "
print remember
print remember
print remember
print remember
print remember
print " I will remember! " remember
```

Testing

- ☐ Testing that the words are correct.
- ☐ Testing that the whitespace is correct.
- ☐ Testing that the punctuation is correct.
- ☐ Testing that the capitalisation is correct.
- ☐ **Great work! You'll be a master of Biology in no time!**

1.2.7. Problem: Non-living things

In Biology, there are three categories we can use to classify all things

- living - alive
- non-living - was never alive
- dead - no longer alive.

There are lots of important non-living things, like sunlight, water and air. These are things that enable life to exist, but are not living things themselves.

We've put a program in the editor that has a variable, non living, that contains information to be printed.

Click ▶ Run to see what it does.

Mistake! The program is about a `zombie`, which is a fictional character that doesn't exist, and wouldn't be biologically correct even if it did.

Update the program to bring it back to reality to work for a different object, a `rock`.

Your updated program should print the message:

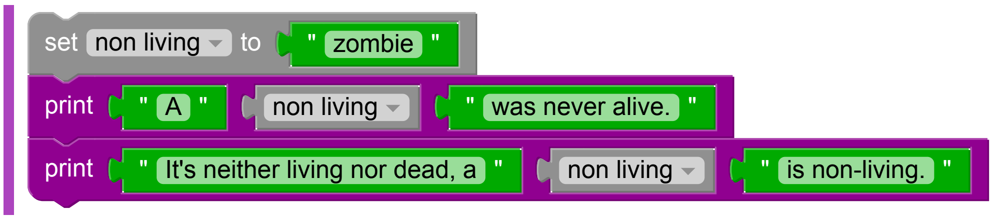
```
A rock was never alive.
It's neither living nor dead, a rock is non-living.
```

Only change the non living variable!

You just need to change the value of non living to be `rock` instead of `zombie`, run it to check it works, and then mark it.

You'll need

program.blockly



```
set non living to " zombie "
print " A " non living " was never alive. "
print " It's neither living nor dead, a " non living " is non-living. "
```

Testing

☐ Testing that the words are correct.

☐ Testing that the whitespace is correct.

☐ Testing that the punctuation is correct.

☐ Testing that the capitalisation is correct.

☐ **Great work, you rock!**

1.3. Adaptation

1.3.1. Problem: Adaptations

When an animal adapts its *actions* to survive in its environment, this is known as...

- ◯ Variables
- ◯ Taxonomy
- ◯ Behavioural adaptation
- ◯ Structural adaptation

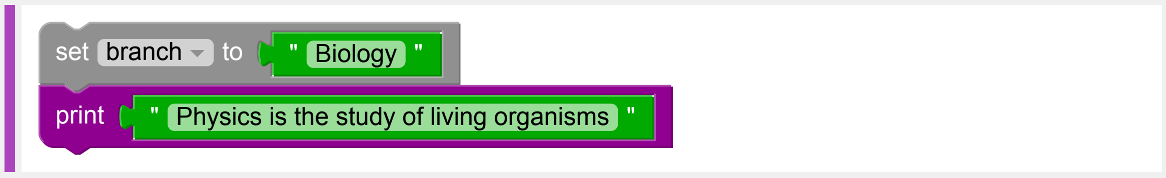
Testing

☐ That's right!

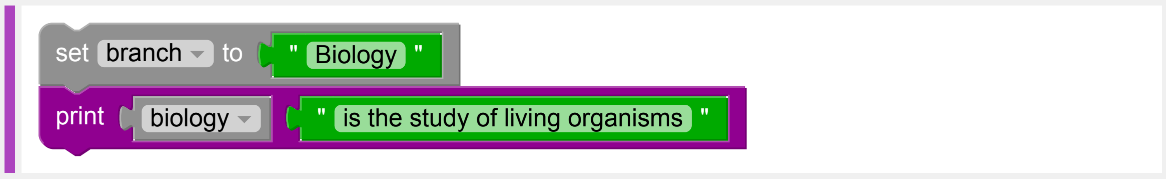
1.3.2. Problem: The study of living organisms

Which of the following code snippets will correctly print the message below on the screen?

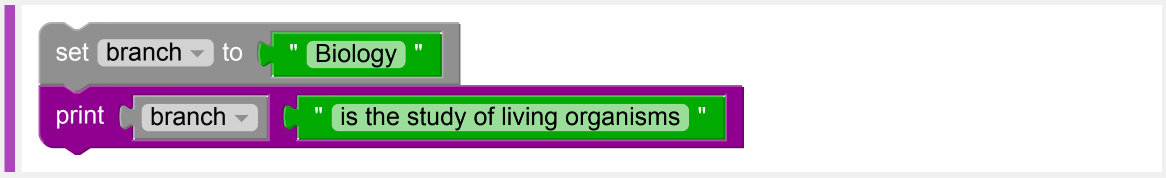
```
Biology is the study of living organisms
```

- [ ] 

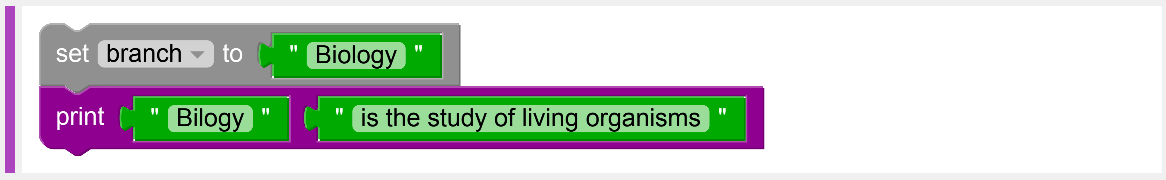
```
set branch to "Biology"
print "Physics is the study of living organisms"
```

- [ ] 

```
set branch to "Biology"
print biology "is the study of living organisms"
```

- [ ] 

```
set branch to "Biology"
print branch "is the study of living organisms"
```

- [ ] 

```
set branch to "Biology"
print "Bilogy" "is the study of living organisms"
```

Testing

☐ That's right!

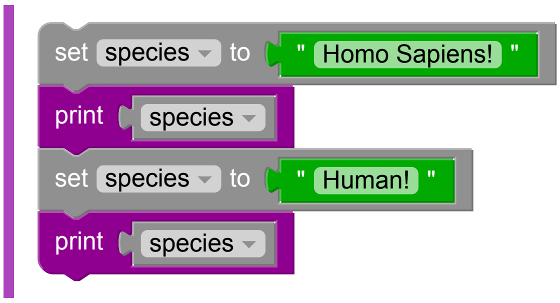
2 INPUT

2.1. Reading user input

2.1.1. Assigning to a variable

Setting the contents of a variable is called *assigning* a value to the variable. Python creates variables by assignment.

When you assign a new value to an existing variable, it replaces the old contents of the variable:



```
set species to "Homo Sapiens!"
print species
set species to "Human!"
print species
```

The old value in `species` - `'Homo Sapiens'` - is replaced with the new value - `'Human!'` - before the second `print` statement, producing:

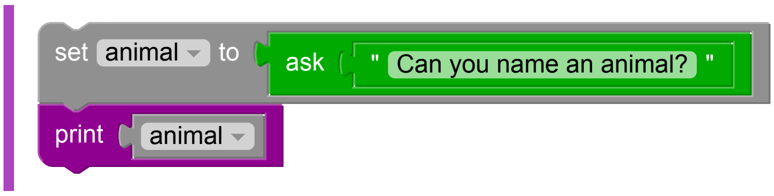
```
Homo Sapiens!
Human!
```

Variables are like files

Variables are like files on your computer: they have a name and you can store data in them. You can look at the contents or overwrite the contents with new data.

2.1.2. Asking the user a question

Let's write a program that asks the user to name an animal:



```
set animal to ask "Can you name an animal?"
print animal
```

Run this program. Even if you haven't run any so far, run this one!

You will need to type a name and press `Enter`:

```
Can you name an animal? Duck
Duck
```

The ask block needs a question string to *ask the user*. It returns the user's answer to our program as a new string. Our program stores the answer in the animal variable so we can print it later.

Run it again with a different animal. Then try changing the question.

2.1.3. Problem: Lyrebird mimic

The Lyrebird (https://en.wikipedia.org/wiki/Lyrebird) is an Australian bird that has the ability to mimic sounds it hears in its environment. It can mimic not only natural sounds but artifical ones too! It's also on the 10c coin.

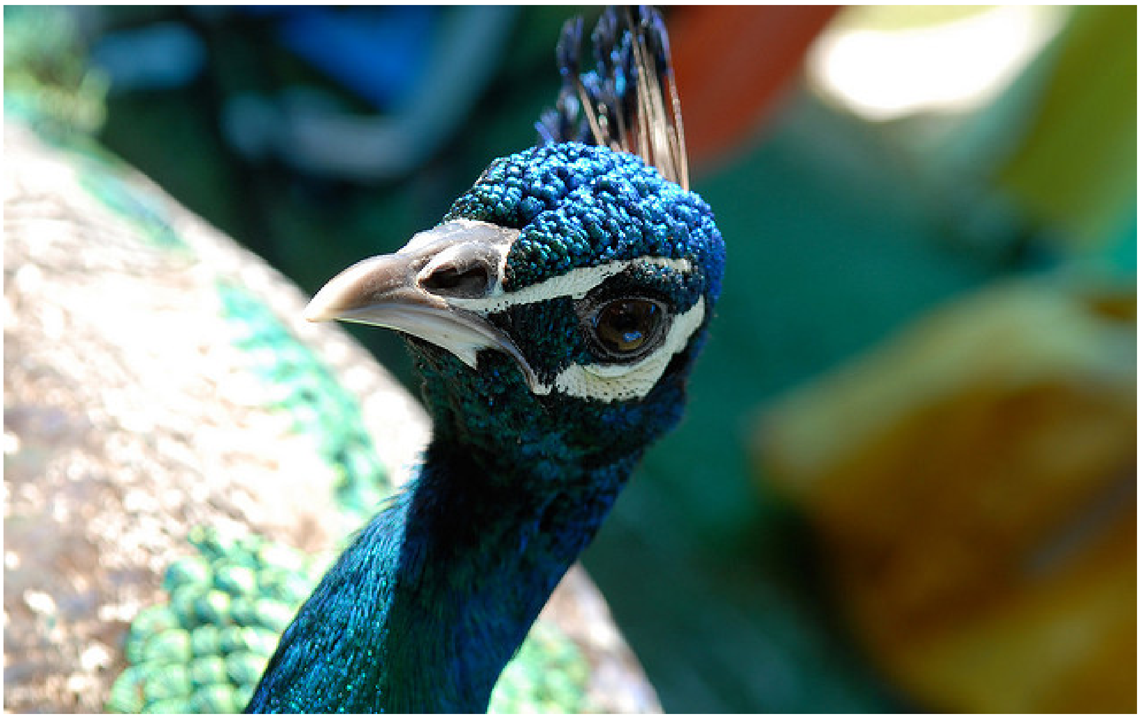


Photo by Leon Wilson CC-BY-2.0 (https://creativecommons.org/licenses/by/2.0/)

Write a program that simulates the lyrebird's mimic ability. You need to get input from the user and print back exactly what the user entered.

You are going to need a variable to store the words, an ask block to get the words from the user and a print block to make it appear on the screen.

```
What sound do you want to make? Squawk!
Squawk!
```

Your program should work with anything the user types:

```
What sound do you want to make? Blurrgh!
Blurrgh!
```

Hint

Make sure you give ask the same prompt question that is used in the example above.

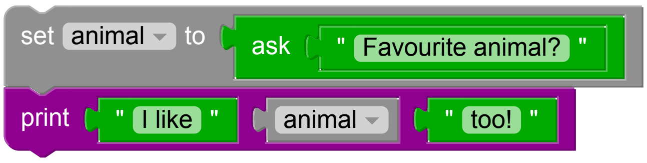
Testing

- ☐ Testing that the words in the prompt are correct.
- ☐ Testing that the punctuation in the prompt is correct.
- ☐ Testing that the capitalisation in the prompt is correct.
- ☐ Testing that the white space in the prompt is correct.
- ☐ Testing with the word `Squawk!`

- ☐ Testing with the word `Blurrgh!`
- ☐ Testing a hidden test case (to make sure your program works in general).

2.1.4. Variable variables!

Now we see why variables are called *variables*! When you run the program and ask the user for input, they could type anything:



```
set animal to ask "Favourite animal?"
print "I like" animal "too!"
```

```
Favourite animal? tigers
I like tigers too!
```

Here, the `animal` variable contains `'tigers'`, but if the user types in something else, it will contain something else:

```
Favourite animal? pineapples
I like pineapples too!
```

This time, the `animal` variable contains `'pineapples'`.

Variables are *variable* because you may not know their value when you write the program, it could be *anything*!

2.1.5. Problem: The animal is...

Write a program to take an input species, and say it back to us:

```
Animal: human
The animal you wrote was: human
```

Your program should work with any animal name:

```
Animal: pig
The animal you wrote was: pig
```

When you run your program, it should wait for you to type in the animal name, using ask, then use the same name that the user typed in when printing the message.

Testing

- ☐ Testing that the words in the prompt are correct.
- ☐ Testing that the punctuation in the prompt is correct.
- ☐ Testing that the capitalisation of the prompt is correct.
- ☐ Testing that the whitespace in the prompt is correct.
- ☐ Testing that the words in your output are correct.
- ☐ Testing that the punctuation in your output is correct.
- ☐ Testing that the capitalisation in your output is correct.
- ☐ Testing that the whitespace in your output is correct.
- ☐ **Yay, you've got the first example right!**
- ☐ Testing the second example in the question (when the user enters `pig`).
- ☐ Testing with the animal `giraffe`.
- ☐ Testing a two word animal (`saltwater crocodile`).
- ☐ Testing a hidden test case (to make sure your program works in general).
- ☐ Testing another hidden test case.

2.1.6. Problem: What we saw at the zoo

There are lots of different things to see at the zoo and you want to remember all the things you saw! Write a program that prints out a sentence you can copy and paste into your diary for any of the things you saw.

```
What did we see? elephants
We saw elephants at the zoo!
```

Your program should work with anything!

```
What did we see? wombats
We saw wombats at the zoo!
```

Testing

- ☐ Testing that the words in the prompt are correct.
- ☐ Testing that the punctuation in the prompt is correct.
- ☐ Testing that the capitalisation of the prompt is correct.
- ☐ Testing that the whitespace in the prompt is correct.
- ☐ Testing that the words in the output sentence are correct.
- ☐ Testing that the punctuation in the output sentence is correct.
- ☐ Testing that the capitalisation in the output sentence is correct.
- ☐ Testing that the whitespace in the output sentence is correct.
- ☐ **Yay, you've got the first example right!**
- ☐ Testing the second example in the question (when the user enters `wombats`).
- ☐ Testing when you saw `people`.
- ☐ Testing something with two words (`tasmanian devils`).
- ☐ Testing a hidden test case (to make sure your program works in general).
- ☐ Testing another hidden test case.

2.2. More with variables

2.2.1. Structural Features

Animals have structural features that help them survive in their environment.

These features can come in many different forms - it could be to help them regulate temperature in hot environments, or features that make it easier to get food, there are almost infinite possibilities!

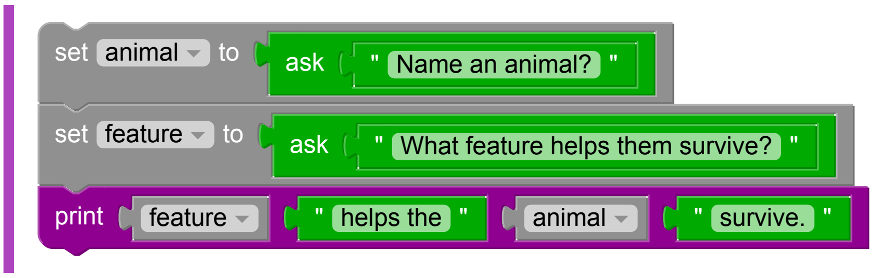
For example, a koala has thick, waterproof fur that helps it keep warm and dry!

Photo by DAVID ILIFF. License: CC-BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)

2.2.2. Using multiple variables

When scientists use computers to store information about animals, they often need to store lots of different bits of information.

To do this they can create more variables! You can have as many as you like as long as each has a different name.



```
set animal to ask "Name an animal?"
set feature to ask "What feature helps them survive?"
print feature "helps the" animal "survive."
```

The variable names make it very clear what is stored in each!

Here we can tell what is stored in both the animal and feature variables!

```
Name an animal? koala
What feature helps them survive? Thick fur
Thick fur helps the koala survive.
```

Good variable names help explain what the code does!

2.2.3. Problem: Desert survival

In harsh environments, plants and animals have developed lots of different structural features to help them survive.

Part of your assignment is to list what types of animals are in the **desert** and what features have helped them adapt.

Write a program that asks for the *animal*, and the *feature* prints out how it helps them survive.

Here is an example:

```
What is the living thing? cactus
What feature helps it survive? Storing water
The cactus lives in the desert.
Storing water helps it survive.
```

Here's another example, the thorny devil has impervious skin to help it survive in the desert (that means it doesn't lose any water via its skin):

```
What is the living thing? thorny devil
What feature helps it survive? Impervious skin
The thorny devil lives in the desert.
Impervious skin helps it survive.
```

Testing

- ☐ Testing the words in the first prompt message.
- ☐ Testing the words in the second prompt message.
- ☐ Testing the capitalisation of the prompts.
- ☐ Testing the punctuation in the prompts.
- ☐ Testing the spaces in the prompts.
- ☐ Testing the words in the match line.
- ☐ Testing the punctuation, spaces and capitals in the match line.
- ☐ **Testing the first example in the question.**
- ☐ Testing the second example in the question.
- ☐ Testing a red kangaroo.
- ☐ Testing a bilby.
- ☐ Testing a hidden case.

2.2.4. Problem: Surviving in different habitats

There are many different habitats that plants and animals live in. Some survive underwater, others live in rainforests, and everywhere in between!

Write a program to take in a habitat and *two* common features, from a user, that plants and animals use to survive in that habitat.

Here is an example, where having gills and fins help creatures survive in the sea:

```
Habitat? sea
What is feature 1? Gills
What is feature 2? fins
Gills and fins helps species survive in the sea
```

Here's another example of Camoflage and Nocturnality (creatures sleeping during the day and being awake at night) being features that are often used to help animals survive in the rainforest.

```
Habitat? rainforest
What is feature 1? Camoflage
What is feature 2? nocturnality
Camoflage and nocturnality helps species survive in the rainforest
```

Testing

- ☐ Testing the words in the first prompt message.
- ☐ Testing the words in the second prompt message.
- ☐ Testing the words in the third prompt message.
- ☐ Testing the capitalisation of the prompts.
- ☐ Testing the punctuation in the prompts.
- ☐ Testing the spaces in the prompts.
- ☐ Testing the words in the first match line.
- ☐ Testing the words in the second match line.
- ☐ Testing the punctuation, spaces and capitals in the match lines.
- ☐ **Testing the first example in the question.**
- ☐ Testing the second example in the question.
- ☐ Testing the desert.
- ☐ Testing wetlands.
- ☐ Testing a hidden case.

2.3. Structural features

2.3.1. Problem: Structural features

A *structural feature* of an animal is...

- ○ The way it behaves
- ○ A physical feature
- ○ A characteristic
- ○ The way an animal survives

Testing

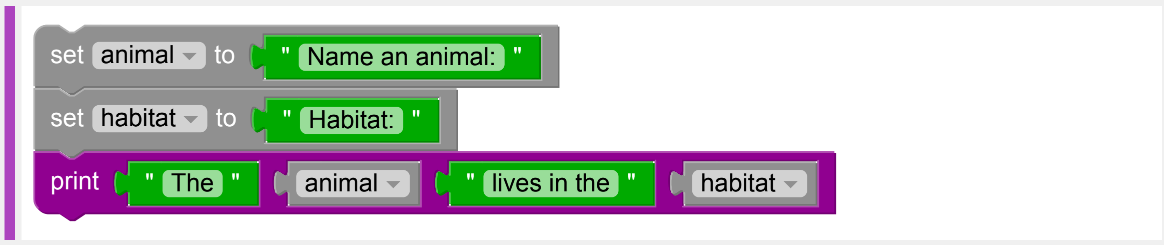
☐ That's right!

2.3.2. Problem: Animal and habitat

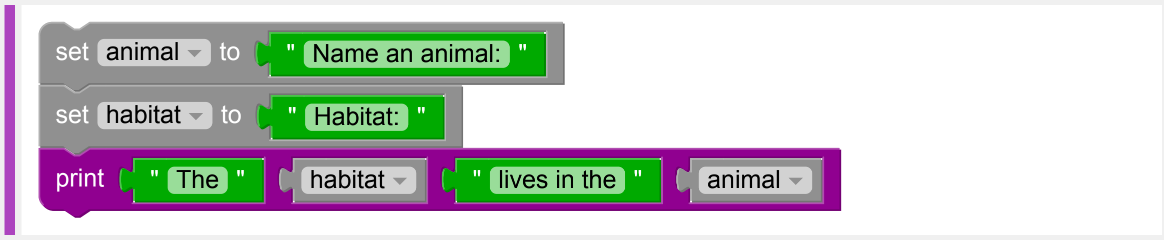
Which of the following programs would correctly generate both outputs shown below?

```
Name an animal: cassowary
Habitat: tropics
The cassowary lives in the tropics
```

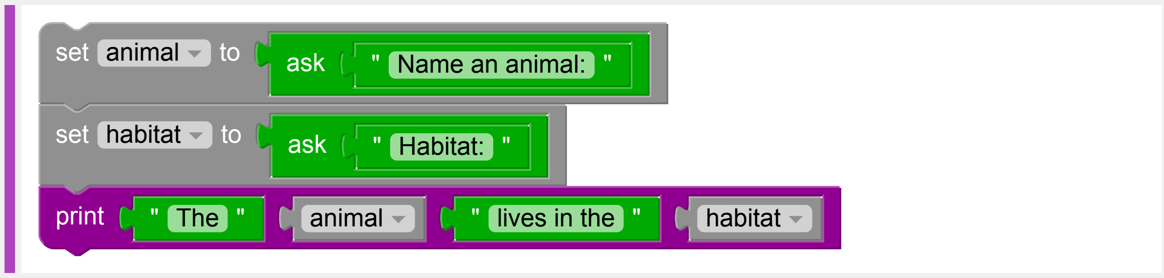
```
Name an animal: emu
Habitat: savannah
The emu lives in the savannah
```

○ 

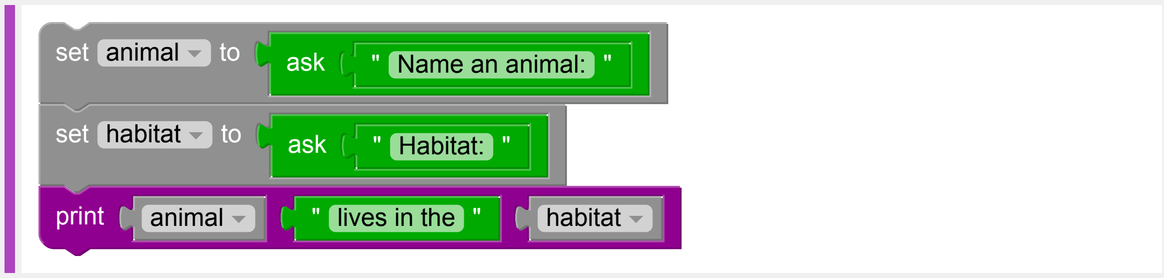
```
set animal to " Name an animal: "
set habitat to " Habitat: "
print " The " animal " lives in the " habitat
```

○ 

```
set animal to " Name an animal: "
set habitat to " Habitat: "
print " The " habitat " lives in the " animal
```

○ 

```
set animal to ask " Name an animal: "
set habitat to ask " Habitat: "
print " The " animal " lives in the " habitat
```

○ 

```
set animal to ask " Name an animal: "
set habitat to ask " Habitat: "
print animal " lives in the " habitat
```

Testing

☐ That's right!

3

DECISIONS

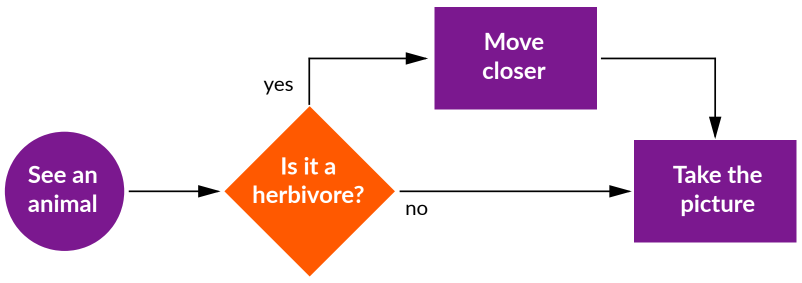
3.1. Making decisions

3.1.1. Why do we need decisions?

So far our programs have been a simple sequence of steps. The interpreter executes the statements from top to bottom, and so the program *runs the same way every time*.

In the real world, we **decide** to **take different steps** based on our situation. For example, if we were on a safari and wanted to take a photo of an animal - we might want to know if the animal not going to eat us (a *herbivore*), before moving closer.

This *flowchart* describes this process (or *algorithm*):



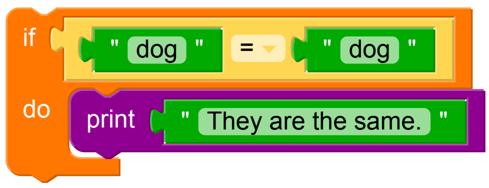
The diamond requires a `yes` or `no` decision. The answer determines which line we follow. If the answer is `yes`, we do the extra step of moving closer. If the answer is no `no`, we skip it.

We can write this using an if block.

3.1.2. Are two strings the same?

The computer decides what steps to run with an if block.

If the two strings **are the same** then the message is printed:

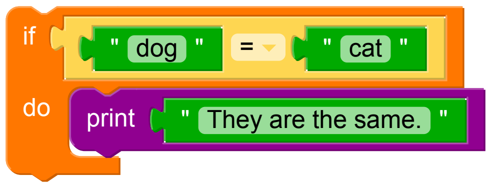


```
if " dog " = " dog "
do print " They are the same. "
```

```
They are the same.
```

" dog " is *equal to* (the same as) " dog ", so the print is run.

If the strings are **not equal**, then the message is not printed:



```
if " dog " = " cat "
do print " They are the same. "
```

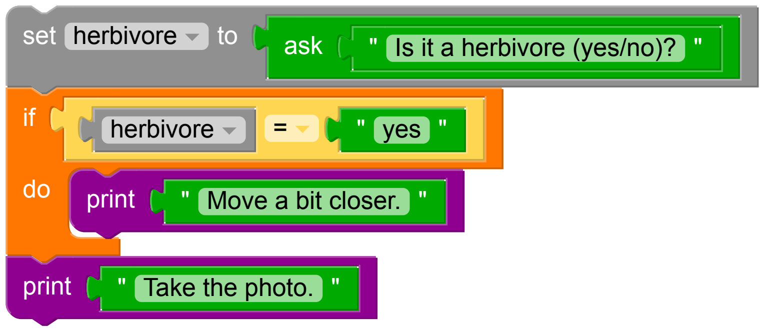
The print is not run because " dog " is not *equal to* " cat ".

Huh, these run the same way each time!?

Yes, but we don't normally compare two string blocks. These examples just show you how if blocks work. **We normally use if blocks with variables.** The if runs the do part only if the value in the variable matches the string.

3.1.3. Making decisions

Let's write our flowchart as a Python program:



```
set herbivore to ask " Is it a herbivore (yes/no)? "
if herbivore = " yes "
do print " Move a bit closer. "
print " Take the photo. "
```

Try it! What happens when you say `yes`, `no`, or any other answer?

Notice that the first print is *inside* the if block. It only runs when the condition is `True`.

If the value stored in herbivore *is equal to* " yes " (because the user entered `yes`), then the body is run. Otherwise, it is skipped.

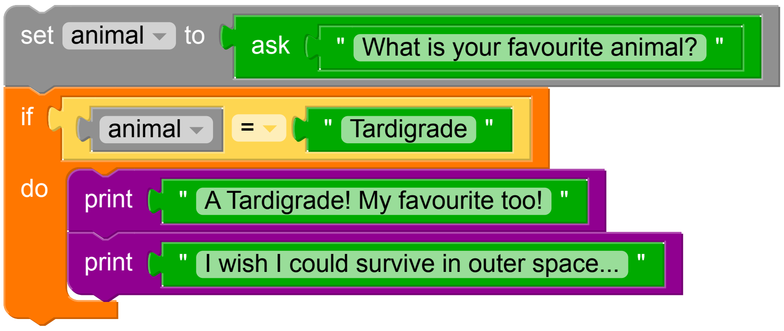
The second print always runs, because it is not indented, and isn't controlled by the if.

An if block is a *control structure*

The if block *controls* how the program runs by deciding if the body is run or not.

3.1.4. Controlling a block of code

You can put as many blocks inside the if block as you want. The code *controlled* by the if (in the do part) is called its *body*:



```
set animal to ask " What is your favourite animal? "
if animal = " Tardigrade "
do  print " A Tardigrade! My favourite too! "
    print " I wish I could survive in outer space... "
```

Here, we ask the user what their favourite animal is. If they answer `Tardigrade`, the value in animal will be equal to " Tardigrade ", and *both* print blocks in the body will run.

If they answer anything else, both the print blocks will be skipped.

Move a print outside of the if body to see the difference.

3.1.5. Problem: I love marsupials!

You're in science class and the teacher is teaching you different features of animals.

Marsupials (https://en.wikipedia.org/wiki/Marsupial) are animals that have pouches, like a kangaroo!

A kangaroo carries its young, or *joey*, in its pouch

Write a program to check the feature the teacher says. Since all animal features are pretty cool, no matter what feature it is your program still should print out `That's a cool feature.`

```
What is the feature? feathers
That's a cool feature.
```

If it's a `pouch`, the program should also print `I love marsupials!` For example:

```
What is the feature? pouch
That's a cool feature.
I love marsupials!
```

Here is another example, with a different feature:

```
What is the feature? gills
That's a cool feature.
```

Remember that you should always print out `That's a cool feature.`, whether that feature is a `pouch` or not.

Testing

- ☐ Testing the words in the input prompt.
- ☐ Testing the capital, punctuation and spaces in the input prompt.
- ☐ Testing the first example in the question.
- ☐ Testing the punctuation, spaces and capital letters in the first example.
- ☐ Testing the second example in the question.
- ☐ Testing the third example in the question.
- ☐ Testing a different feature (`claws`).
- ☐ Testing a different feature (`trunk`).
- ☐ Testing a hidden case.
- ☐ Testing another hidden case.

3.1.6. Problem: The "bin chicken"

Lots of animals native to Australia are now forced to live in cities because humans have altered their natural environments. This can make survival difficult for these species.

Ask the user to type in an animal, and whatever they type your program should print `I hope the <animal> can survive out there...`, **substituting the animal's name into the output**. For example:

```
What animal is it? bat
I hope the bat can survive out there...
```

There's a good chance in large cities you'll see an ibis (https://en.wikipedia.org/wiki/Australian_white_ibis), which local residents refer to informally as the Bin Chicken (http://www.nationalgeographic.com.au/australia/magpie-beats-bin-chicken-for-aus-bird-of-the-year.aspx).

The Australian white ibis

If the animal is an `ibis`, exclaim that you've seen `A Bin Chicken!`.

```
What animal is it? ibis
A Bin Chicken!
I hope the ibis can survive out there...
```

Any answer other than `ibis` should work the same way:

```
What animal did you see? quokka
I hope the quokka can survive out there...
```

The sad story of the Ibis

The *Australian White Ibis* has been forced to live off rubbish in urban environments (https://www.gizmodo.com.au/2017/11/in-defence-of-the-bin-chicken/) due to human development gradually forcing it out of its natural habitat. Eating rubbish isn't good for them, and restoring their wetlands would go a long way to improving their long-term survival.

Testing

- ☐ Testing the words in the input prompt.
- ☐ Testing the capital, punctuation and spaces in the input prompt.

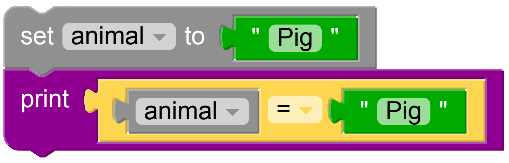
- ☐ Testing the first example in the question.
- ☐ Testing the punctuation, spaces and capital letters in the first example.
- ☐ Testing the second example in the question.
- ☐ Testing the third example in the question.
- ☐ Testing a different animal (`kangaroo`).
- ☐ Testing a two-word animal (`alley cat`).
- ☐ Testing a hidden case.
- ☐ Testing another hidden case.

3.1.7. True or False?

An if block allows your program to make *yes or no* decisions. In programming, yes is called `True`, and no is called `False`.

The body of the if block runs when the comparison (the = block) is `True`. It doesn't run if the comparison is `False`.

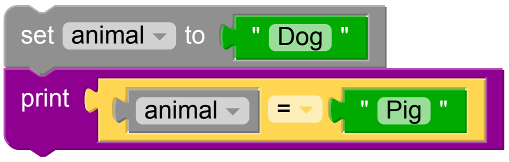
Calculations that result in `True` or `False` values are called *Boolean expressions*. We can print their value directly:



```
set animal to "Pig"
print animal = "Pig"
```

```
True
```

If we change `animal`, the conditional expression will be `False`:



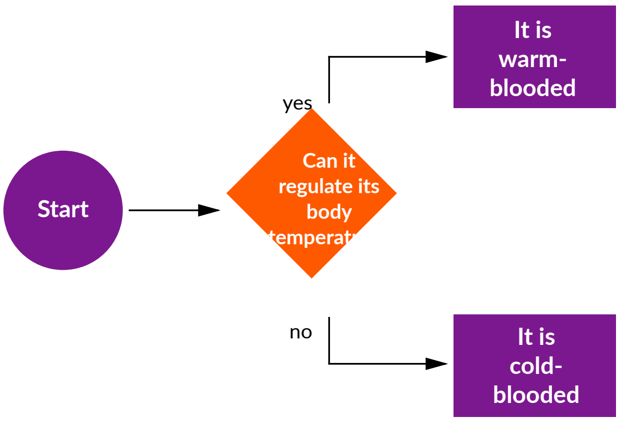
```
set animal to "Dog"
print animal = "Pig"
```

```
False
```

3.2. Decisions with two options

3.2.1. Two options

It's often useful to make decisions that have *two* options. Let's take a look at an example.



A characteristic of all warm-blooded animals is their ability to regulate their own body temperature. This allows them to keep their body temperature the same even when the weather changes.

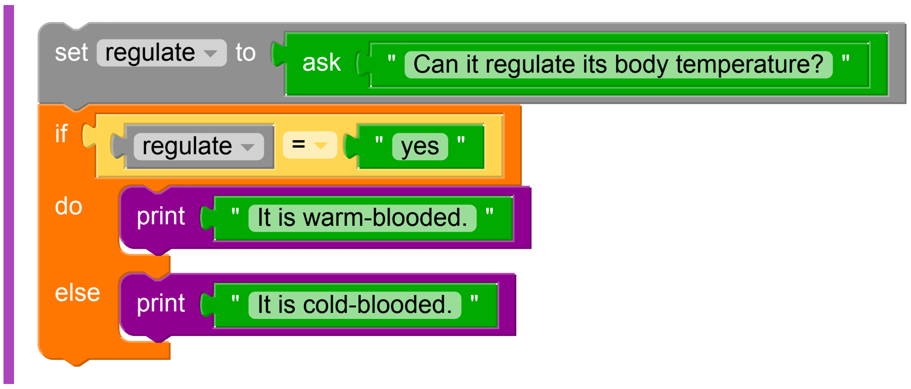
Cold-blooded animals become hotter and colder when the temperature outside changes. At night their bodies get cooler, and during the day when they are in the sun they warm up.

We could write a program that asks if the animal can regulate its body temperature. If the user says `'yes'` it can, then it tells the user it is warm-blooded. Otherwise it says it is cold-blooded.

What we want is an extra part to the if block which is only run when the condition is `False`.

3.2.2. Warm- or cold-blooded?

Let's implement the two options in the new flowchart using a if block with an else part:



```
set regulate to ask " Can it regulate its body temperature? "
if regulate = " yes "
do   print " It is warm-blooded. "
else print " It is cold-blooded. "
```

Here, either the first or second print block will be run, depending on the answer, **but not both.**

3.2.3. Problem: Plant or animal?

Scientifically, the difference between a plant and animal is that plants have a cell wall, while animals do not.

Write to program to check if the organism is a plant or an animal, if the user types in `yes`, then it should say `Then it's a plant!`.

Here's an example:

```
Does it have a cell wall? yes
Then it's a plant!
```

If they say *anything* else, you should say, `It's probably an animal.`

```
Does it have a cell wall? no
It's probably an animal.
```

If the answer is not *exactly* `yes`, you should still print out the other message. For example, your programs does not understand slang:

```
Does it have a cell wall? yessss
It's probably an animal.
```

Testing

- ☐ Testing the words in the input prompt.
- ☐ Testing the capital, punctuation and spaces in the input prompt.
- ☐ Testing the words in first example in the question.
- ☐ Testing the punctuation, spaces and capital letters in the first example.
- ☐ Testing the words in the second example in the question.
- ☐ Testing the punctuation, spaces and capital letters in the second example.
- ☐ Testing the third example.
- ☐ Testing a typo.
- ☐ Testing a hidden case.

3.2.4. Problem: Predator or prey

Often, the *features* of an animal help us determine what type of animal it is. *Predators* are animals that hunt others, and *prey* are the animals they hunt.

For example, most *predators* don't have the ability to *camoflage*, but *prey* do.

Write a program to find out if the animal is predator or prey by asking for a feature. If the feature is `'claws'` then it is probably a predator, otherwise it is most likely prey.

Here's an example:

```
What is the feature? claws
It is a predator.
```

If they say *anything* else, you should say, `It is prey.`

```
What is the feature? pouch
It is prey.
```

If the answer is not *exactly* `claws`, you should still print out the other message. For example:

```
What is the feature? Claws
It is prey.
```

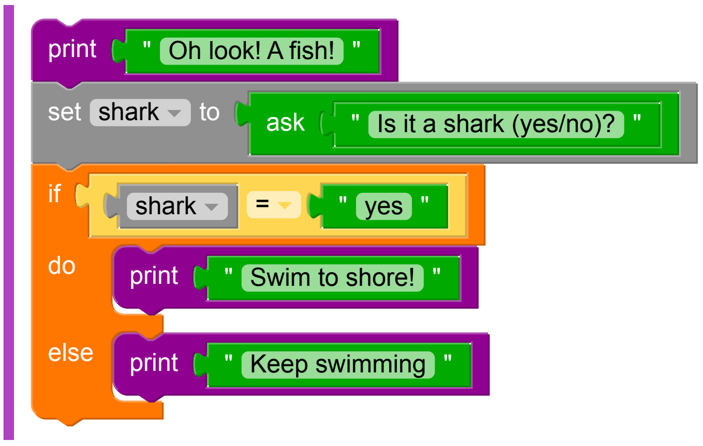
Testing

- ☐ Testing the words in the input prompt.
- ☐ Testing the capital, punctuation and spaces in the input prompt.
- ☐ Testing the words in first example in the question.
- ☐ Testing the punctuation, spaces and capital letters in the first example.
- ☐ Testing the words in the second example in the question.
- ☐ Testing the punctuation, spaces and capital letters in the second example.
- ☐ Testing the third example.
- ☐ Testing a typo.
- ☐ Testing a hidden case.

3.3. Decisions review

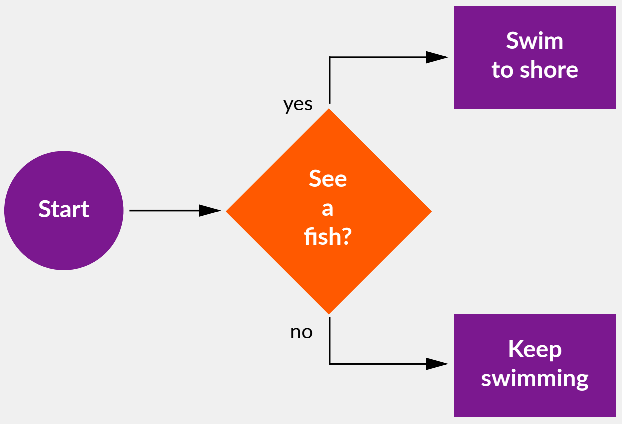
3.3.1. Problem: Is it a shark?

Choose the flowchart below that represents the following code snippet.

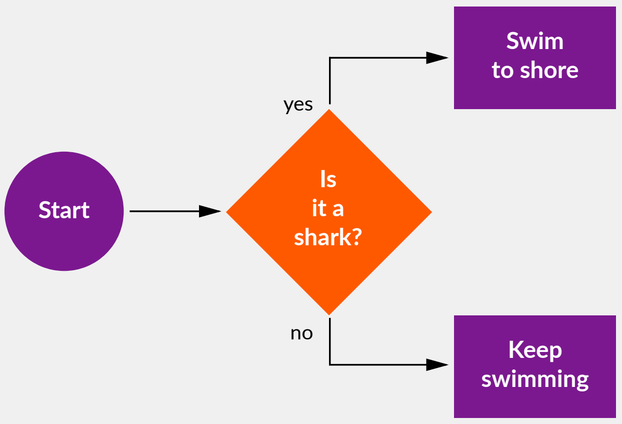


```
print "Oh look! A fish!"
set shark to ask "Is it a shark (yes/no)?"
if shark = "yes"
do print "Swim to shore!"
else print "Keep swimming"
```

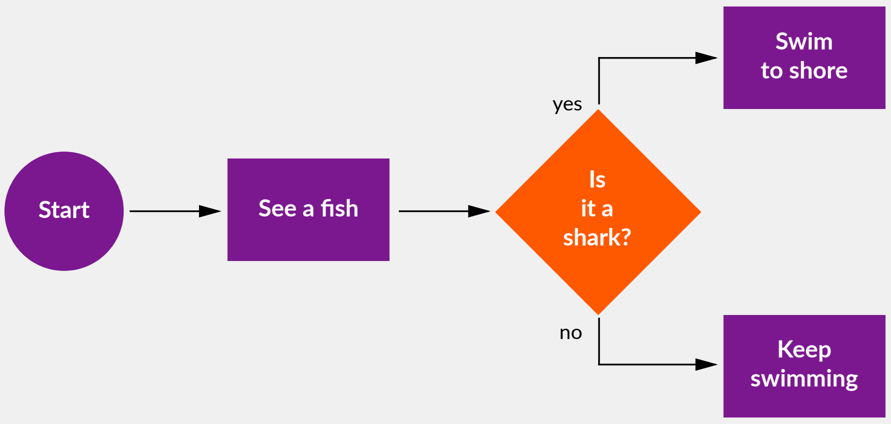
○


Start
See a fish?
yes
Swim to shore
no
Keep swimming

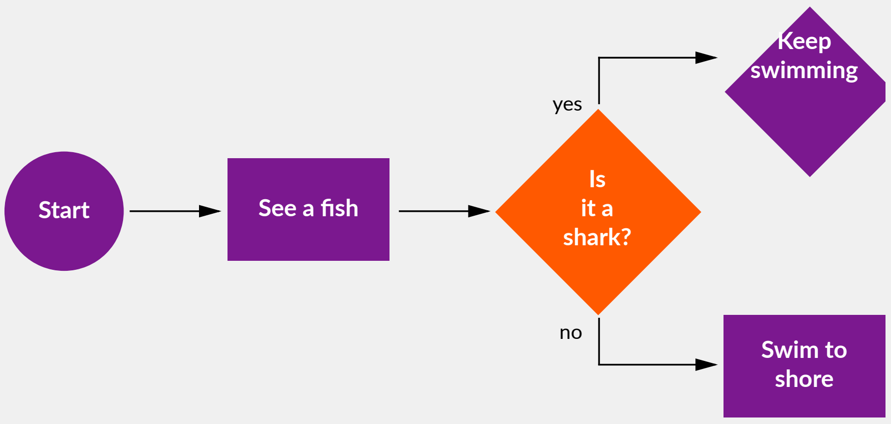
○


Start
Is it a shark?
yes
Swim to shore
no
Keep swimming

○


Start
See a fish
Is it a shark?
yes
Swim to shore
no
Keep swimming

○


Start
See a fish
Is it a shark?
yes
Keep swimming
no
Swim to shore

Testing

☐ That's right!

3.3.2. Problem: Australia's favourite bird

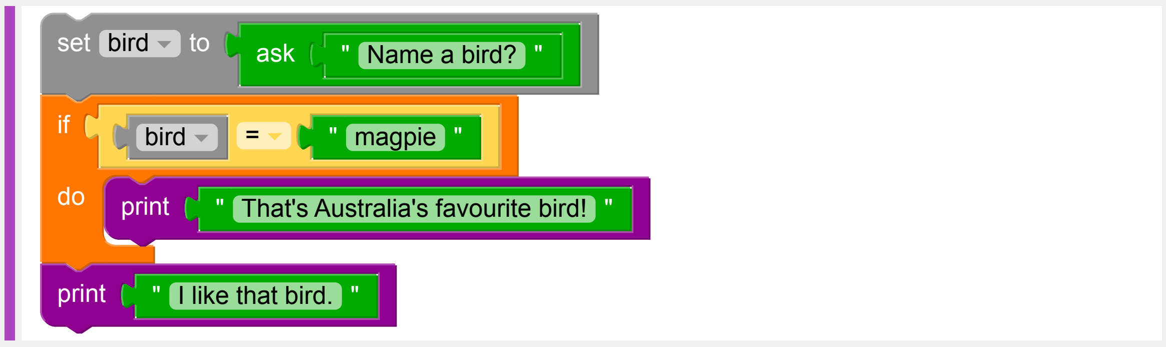
Which of the following code snippets will produce the output?

```
Name a bird? magpie
That's Australia's favourite bird!
```

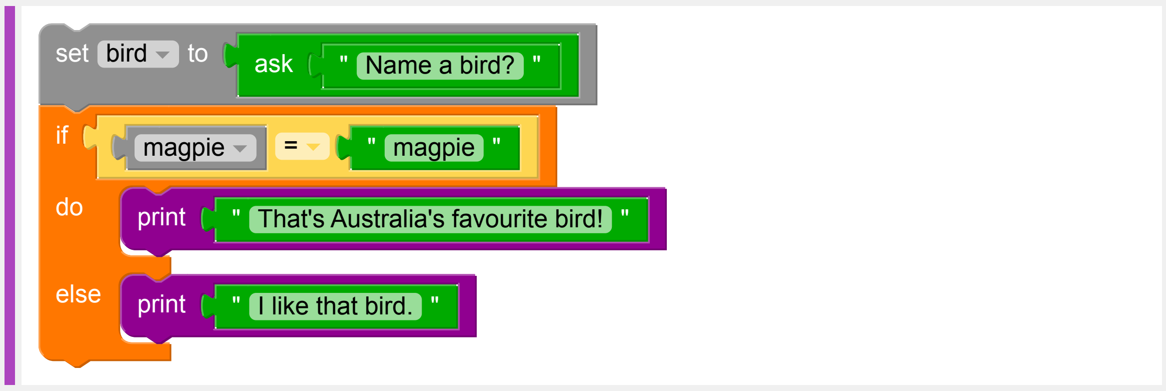
```
Name a bird? swan
I like that bird.
```

```
Name a bird? tiger
I like that bird.
```

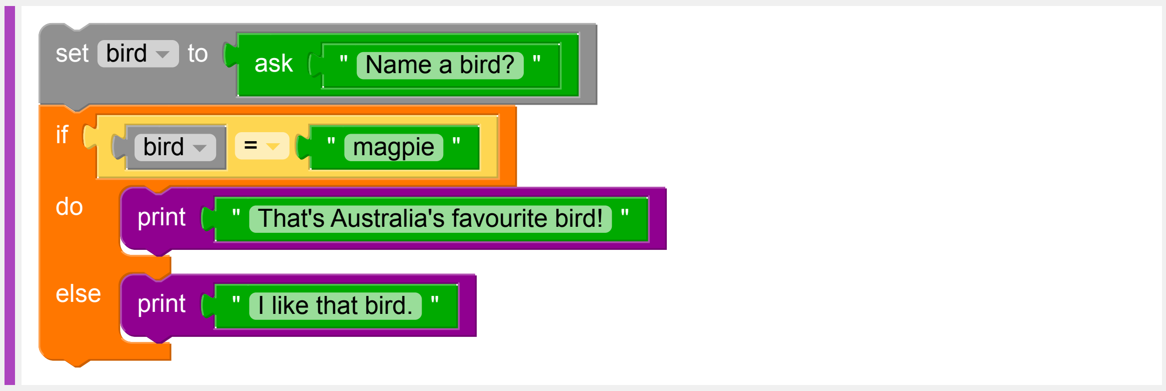
* according to The Guardian (https://www.theguardian.com/environment/live/2017/dec/11/bird-of-the-year-150000-votes-counted-as-ibis-fans-anxiously-await-results) poll, anyway.

○ 

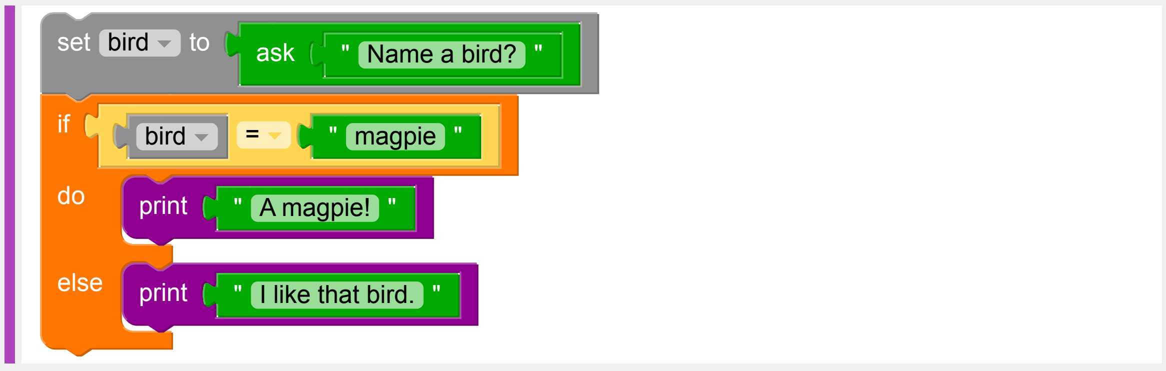
```
set bird to ask "Name a bird?"
if bird = "magpie"
do  print "That's Australia's favourite bird!"
print "I like that bird."
```

○ 

```
set bird to ask "Name a bird?"
if magpie = "magpie"
do   print "That's Australia's favourite bird!"
else print "I like that bird."
```

○ 

```
set bird to ask "Name a bird?"
if bird = "magpie"
do   print "That's Australia's favourite bird!"
else print "I like that bird."
```

○ 

```
set bird to ask "Name a bird?"
if bird = "magpie"
do   print "A magpie!"
else print "I like that bird."
```

Testing

☐ That's right!

4

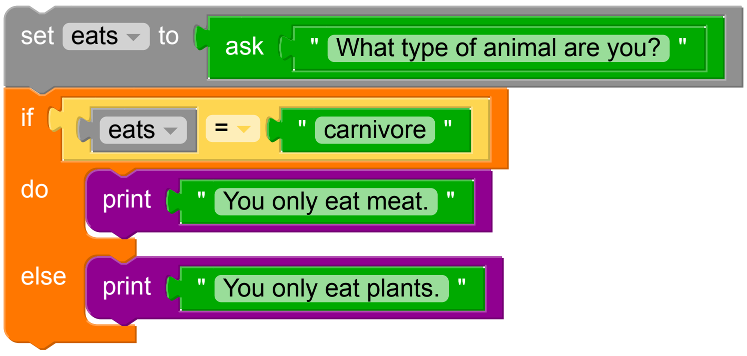
COMPLEX DECISIONS

4.1. Decisions with more options

4.1.1. More difficult questions

The if / else block is great for making decisions with only two options (warm or cold blooded, plant or animal, day or night). When there are more than two possibilities, it can get confusing.

This program tries to decide if you are a `carnivore` or `herbivore`:



```
set eats to ask " What type of animal are you? "
if eats = " carnivore "
do   print " You only eat meat. "
else print " You only eat plants. "
```

If the user enters `carnivore` or `herbivore` the response is correct. But what if it is an `omnivore`?

This isn't right! It's possible that the animal eats both meat and plants, which would make it an `'omnivore'`. We need a way for our program to deal with that!

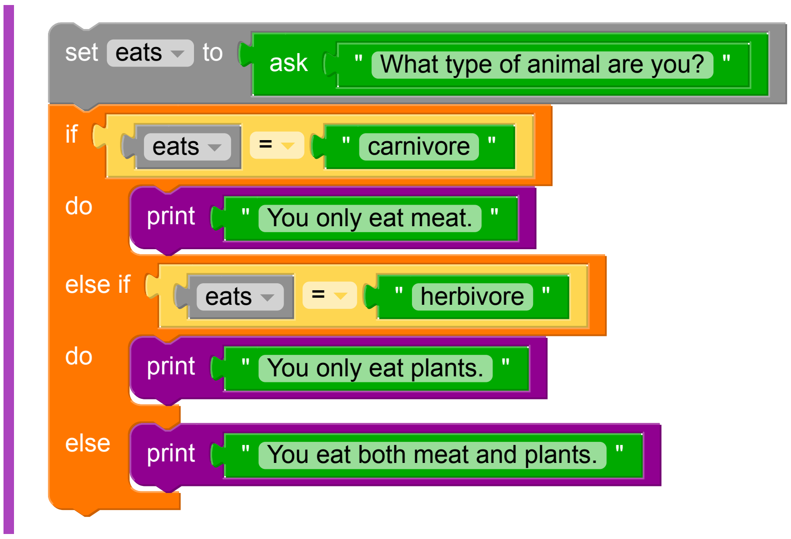
else catches *everything* else!

Remember: the else runs whenever the condition is `False`. Here, we were expecting `carnivore` or `herbivore`, but anything that is not `carnivore` will also return `False`.

4.1.2. More than two options

If we want to test if an animal is a *carnivore*, *herbivore* or *omnivore*, we need to add something extra to our if statement. Computers can only check one thing at a time and those statements always need to evaluate to `True` or `False`.

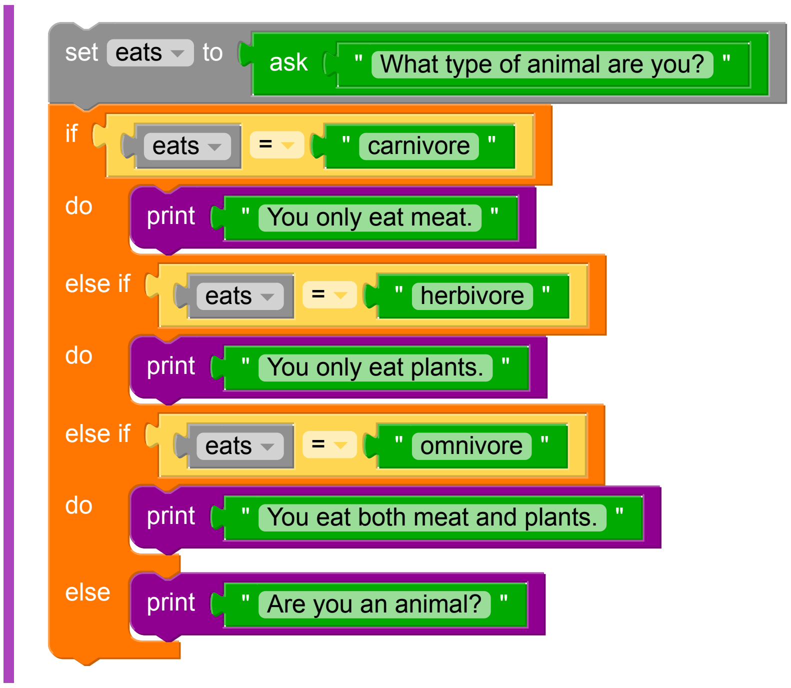
This means we need to check one condition first and, if that one is `False`, have the computer then check a different one. We use an else if to do this.



```
set eats to ask " What type of animal are you? "
if eats = " carnivore "
do print " You only eat meat. "
else if eats = " herbivore "
do print " You only eat plants. "
else print " You eat both meat and plants. "
```

4.1.3. Even more options

Our previous example assumed that if you weren't a `'carnivore'` or `'herbivore'` you had to be an `'omnivore'`, but it's possible the user might type something else in that isn't correct. To fix this, we want to also check if `'omnivore'` is typed in by the user, and we can do this with another `else if`:



```
set eats to ask " What type of animal are you? "
if eats = " carnivore "
do print " You only eat meat. "
else if eats = " herbivore "
do print " You only eat plants. "
else if eats = " omnivore "
do print " You eat both meat and plants. "
else print " Are you an animal? "
```

Remember that an `else` can always be used to catch anything that isn't matched by **any** of the `if` or `else if` conditions - it's a good way to make sure that you only execute the right code for correct values.

4.1.4. Problem: Feature to class

The **class** of an animal describes what general type of animal it is, such as *mammal*, *reptile* or *fish*.

Write a program to tell the difference between three *classes* of animal in the table below, based on the feature from the table:

Class	Feature
Mammal	Fur
Fish	Gills
Bird	Wings

Osteichthyes are fish, and Aves are birds.

Your program should ask for the feature then print out the type of animal that feature corresponds to. For example:

```
What is the feature? Fur
Mammal
```

Here is another example:

```
What is the feature? Gills
Fish
```

If the feature isn't one of the three above, your program should print `I'm not sure.` For example:

```
What is the feature? Scales
I'm not sure.
```

Testing

- ☐ Testing the first example in the question.
- ☐ Testing the second example in the question.
- ☐ Testing the third example in the question.
- ☐ Testing the class for the feature `Flying`.
- ☐ Testing a different feature (`Claws`).
- ☐ Testing a hidden case.
- ☐ Testing another hidden case.

4.1.5. Problem: Find the animal!

In Biology, we can use the *characteristics* of an animal to tell which one is feach.

Your teacher has given you a list of Australian animals with a characteristic of each. Write a program to accept an `input` characteristic, and print out the correct animal from the table.

Animal	**Characteristic**
Kangaroo	Pouch
Cassowary	Casque
Goanna	Claws

Your program should ask for the characteristic then print out the corresponding animal. For example:

```
Characteristic? Pouch
Kangaroo!
```

Notice that there is an exclamation mark at the end of the output!

Here is another example:

```
Characteristic? Casque
Cassowary!
```

If the feature isn't one of the three above, your program should print `I do not know.` For example:

```
Characteristic? Hooves
I do not know.
```

Testing

- ☐ Testing the first example in the question.
- ☐ Testing the second example in the question.
- ☐ Testing the third example in the question.
- ☐ Testing the animal for the characteristic `Claws`.
- ☐ Testing a different characteristic - `Wings`.
- ☐ Testing a hidden case.
- ☐ Testing another hidden case.

4.2. Decisions and algorithms

4.2.1. Making complex decisions

We can use the structural and behavioural characteristics of animals to identify their likely habitat.

Let's say we wanted to separate a Fish (https://en.wikipedia.org/wiki/Fish), Lizard (https://en.wikipedia.org/wiki/Lizard) and a Bird (https://en.wikipedia.org/wiki/Bird). We can simplify things a bit and say that the only things we have information about are whether the animal can fly, and if it has gills.

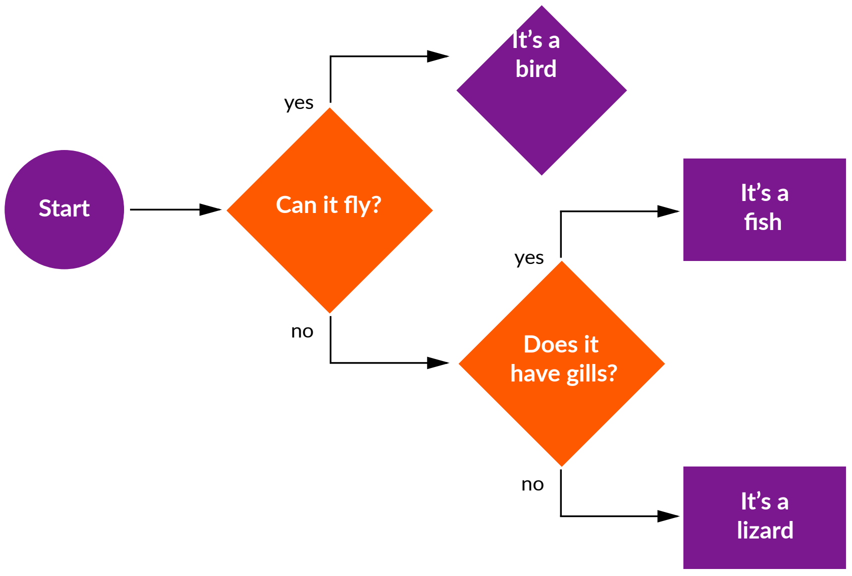
This table summarises some simple facts about these three types:

	Fish	**Lizard**	**Bird**
Can fly?	No	No	**Yes**
Has gills?	**Yes**	No	No

So Birds can fly, Fish have gills, and lizards don't fly or have gills.

4.2.2. Asking the right questions

The flowchart below demonstrates how we can use the properties of *can it fly?* and *does it have gills?* to determine which animal it is:



Because only birds can fly, we can confidently say that a yes to our first question means we have a bird.

Whether we ask the second question at all *depends* on the answer to the first!

4.2.3. Decisions within decisions

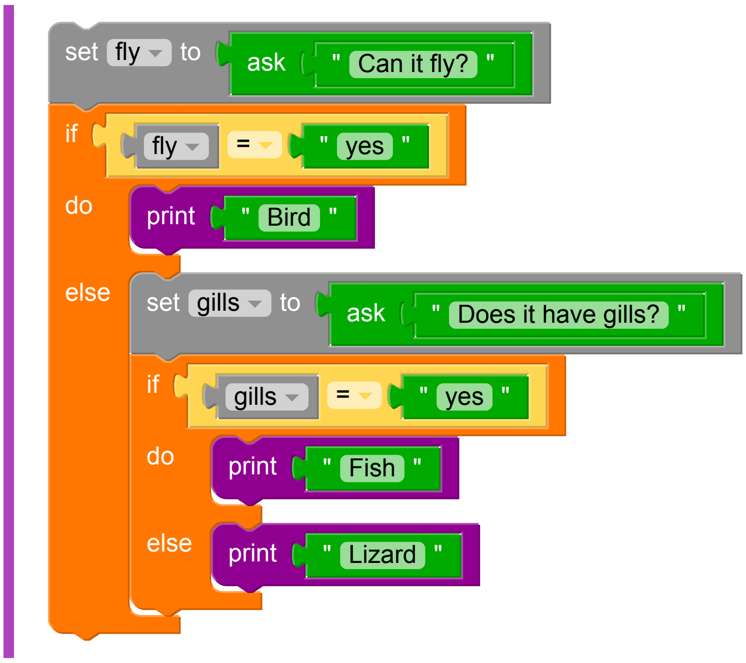
The body of an `if`, `else if` or `else` blocks may contain another `if` block.

This is called *nesting*, and is how we represent our algorithm of a decision within a decision, in code.

When using nesting **asking more questions *depends* on the previous answer!** If we have eliminated all possibilities, there is no need to ask any more questions.

This is the advantage of using nesting.

In the example from the previous slide, the second decision only happens inside the `'no'` case (or the else of our first question). Let's turn the flowchart into code!



```
set fly to ask " Can it fly? "
if fly = " yes "
do  print " Bird "
else set gills to ask " Does it have gills? "
     if gills = " yes "
     do  print " Fish "
     else print " Lizard "
```

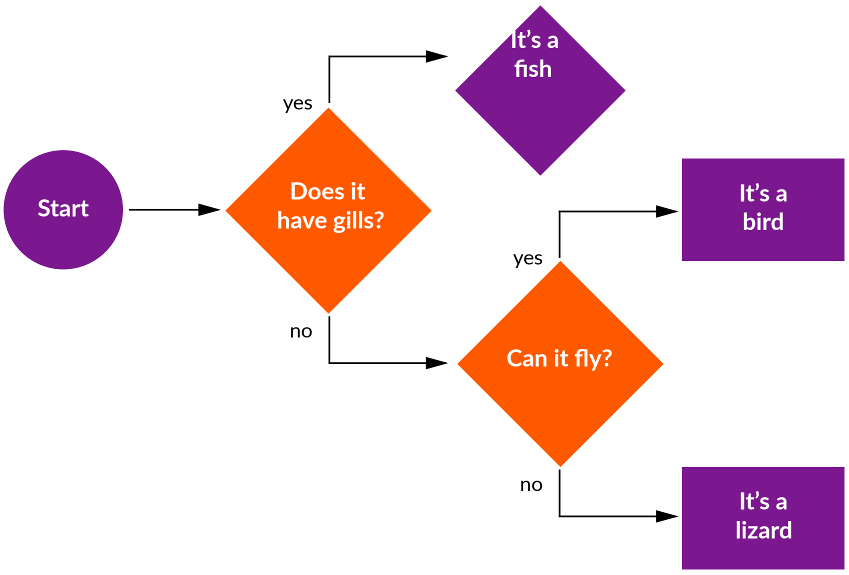
```
Can it fly? no
Does it have gills? no
Lizard
```

Notice how set gills is *inside* the else part of our if else block like the indentation of the rectangles in our flowchart?

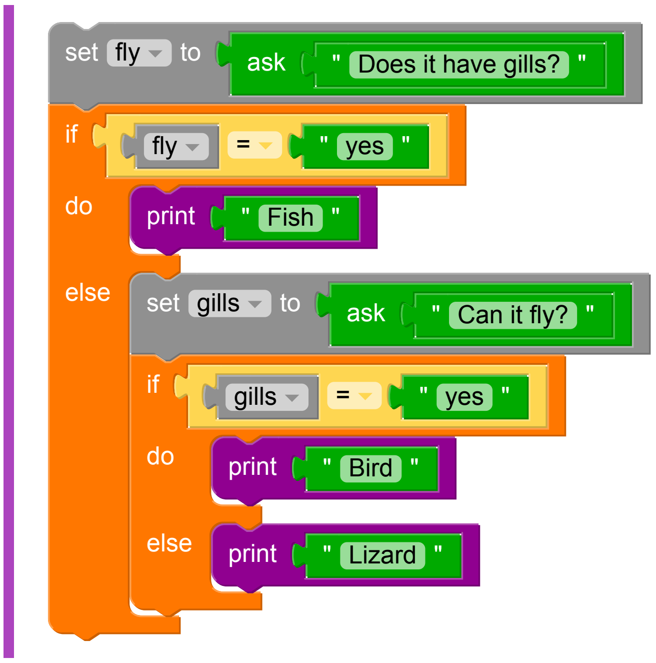
Test the example by running the code above. Try different answers to all of the questions. You'll also notice that if you answer `'yes'` to the first question, the second question won't get asked at all!

4.2.4. Order matters!

Changing the order that you ask your questions can change your algorithm. Look what happens to the flowchart if we ask if the animal lays eggs first:



See how we only need to ask one question to identify the fish this time? This changes how we nest our if statements, and changes the code to:



```
set fly to ask " Does it have gills? "
if fly = " yes "
do
    print " Fish "
else
    set gills to ask " Can it fly? "
    if gills = " yes "
    do
        print " Bird "
    else
        print " Lizard "
```

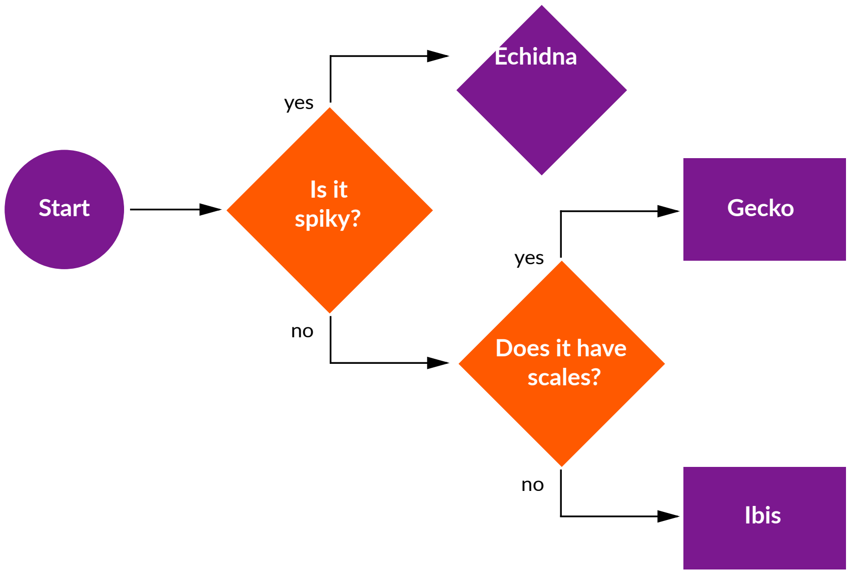
Run this program to see how it works, and try different answers to the questions.

Both algorithms are correct even though the programs are different!

4.2.5. Problem: Which animal?

We want to tell apart the **echidna**, **ibis** and **gecko**.

Write a program that will identify each animal according to the algorithm shown below:



The first question is `'Is it spiky? '`. if it is, it is from genus `'echidna'`:

```
Is it spiky? yes
It is an echidna.
```

If the answer to the first question is `'no'`, ask `'Does it have scales? '` to see if it is a `'gecko'`:

```
Is it spiky? no
Does it have scales? yes
It is a gecko.
```

If the answer to the spur question is `'no'`:

```
Is it spiky? no
Does it have scales? no
It is an ibis.
```

Hint!

Remember that the algorithm in the flowchart provides hints about how to nest your `if` statements.

Testing

- ☐ Testing your input prompt for the first question.
- ☐ Testing an echidna.
- ☐ Testing your input prompt for the second question.
- ☐ Testing a gecko.
- ☐ Testing the genus Macropus.
- ☐ Testing an answer that is not yes or no.
- ☐ Testing a hidden case.

4.2.6. Problem: What's my diet?

In our earlier notes we saw how we could use an `elif` statement to print different messages for animals that were `'carnivore'`, `'herbivore'` or `'omnivore'`. We'll use what we just learned about nesting to do this a little bit differently.

Write a program that asks the user if the animal eats meat or plants and depending on their answer identifies the diet of the animal. It should ask about meat first.

You must ask each question separately and in the correct order, and you should only ask the second question if it is required. Here is how the program behaves when you answer anything *other than* `yes` to `'Does it eat meat?'`:

```
Does it eat meat? no
It's a herbivore!
```

If you answer `'yes'` to the first question, ask `'Does it eat plants? '`:

```
Does it eat meat? yes
Does it eat plants? no
It's a carnivore!
```

And `'yes'` to both questions:

```
Does it eat meat? yes
Does it eat plants? yes
It's an omnivore!
```

Hint!

If you're having trouble with the nesting, a flowchart might help you work out how to nest your `if` statements.

Testing

- ☐ Testing your input prompt for the first question.
- ☐ Testing a herbivore.
- ☐ Testing your input prompt for the second question.
- ☐ Testing a carnivore.
- ☐ Testing an omnivore.
- ☐ Testing an answer that is not yes or no.
- ☐ Testing a hidden case.

4.3. Decision review

4.3.1. Problem: Nesting or `else if`?

When is it better to use nesting instead of an `else if`?

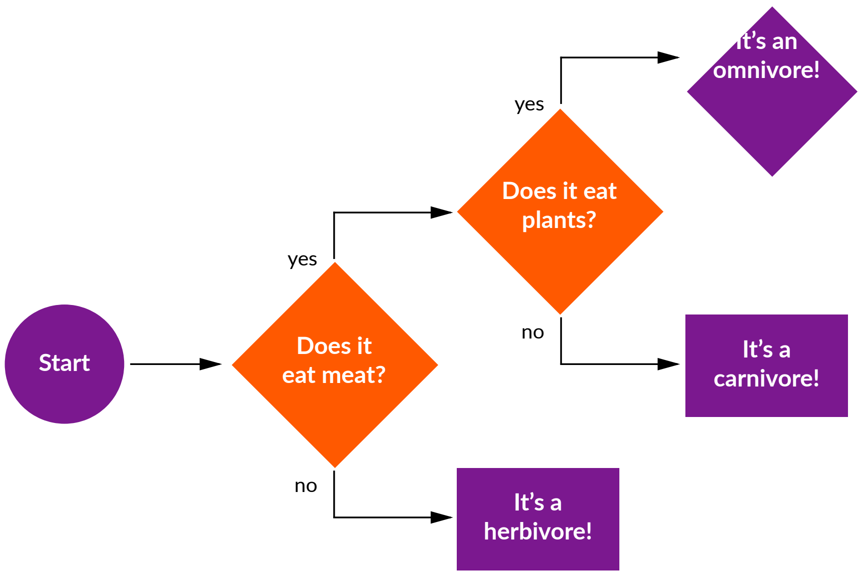
- ○ When there are more than two possible answers to a single question.
- ○ Using `else if` is always the best option.
- ○ When you want to check something that depends on a previous condition.
- ○ Using nesting is always the best option.

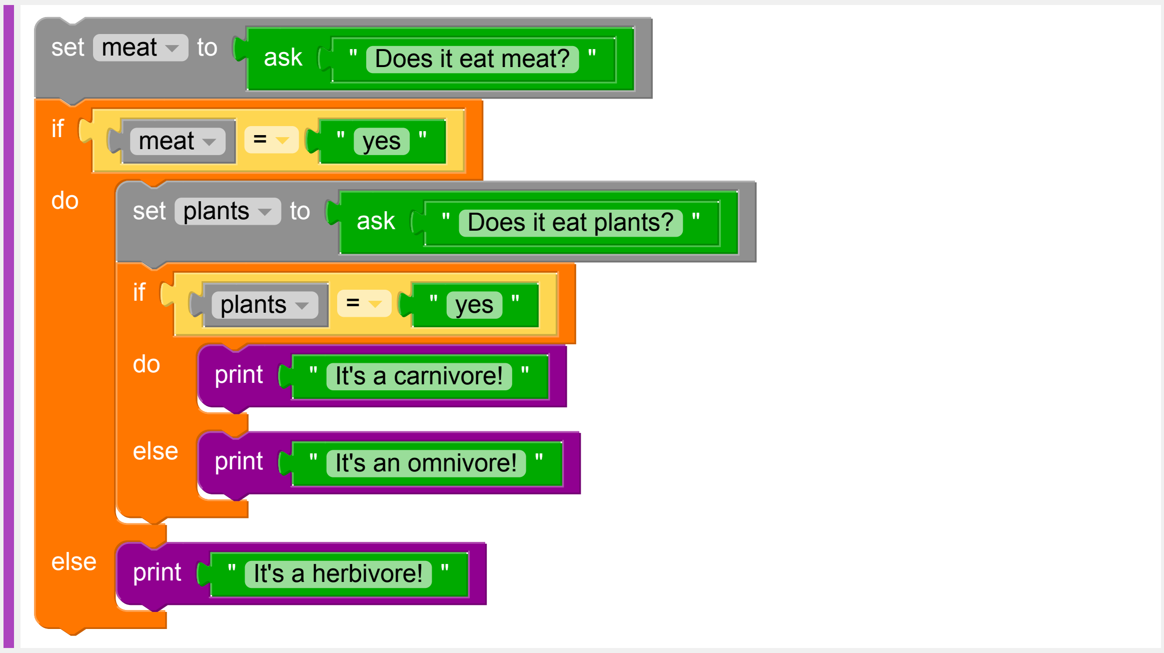
Testing

☐ That's right!

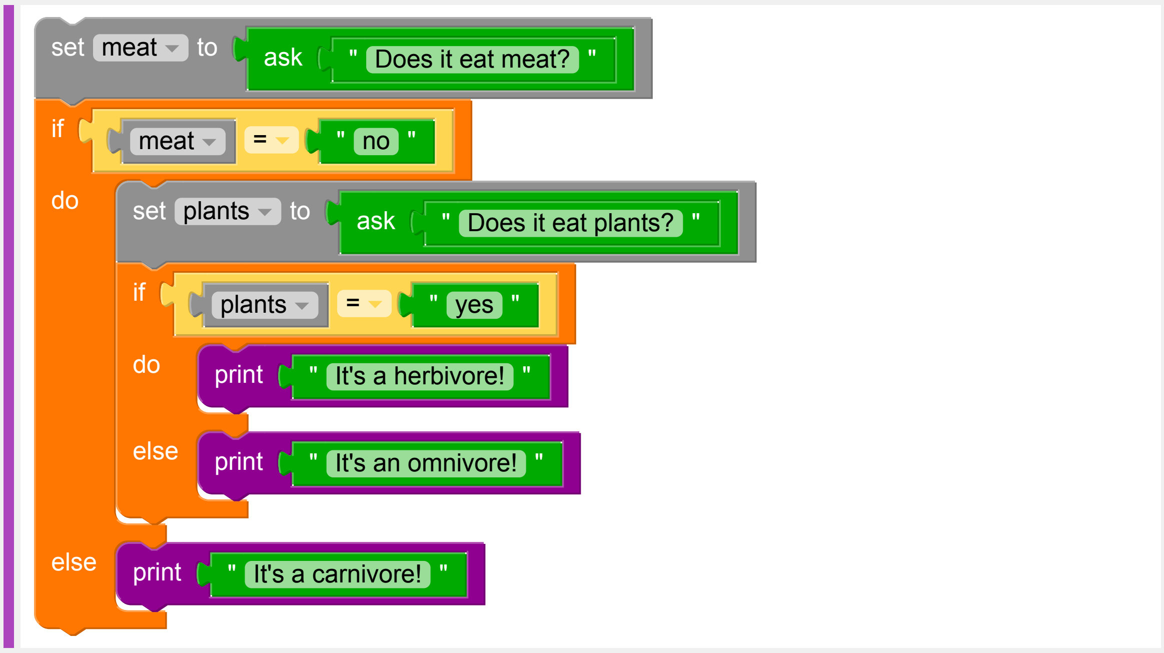
4.3.2. Problem: Which is it?

Which of the programs is the correct representation of the algorithm shown in the flowchart below?



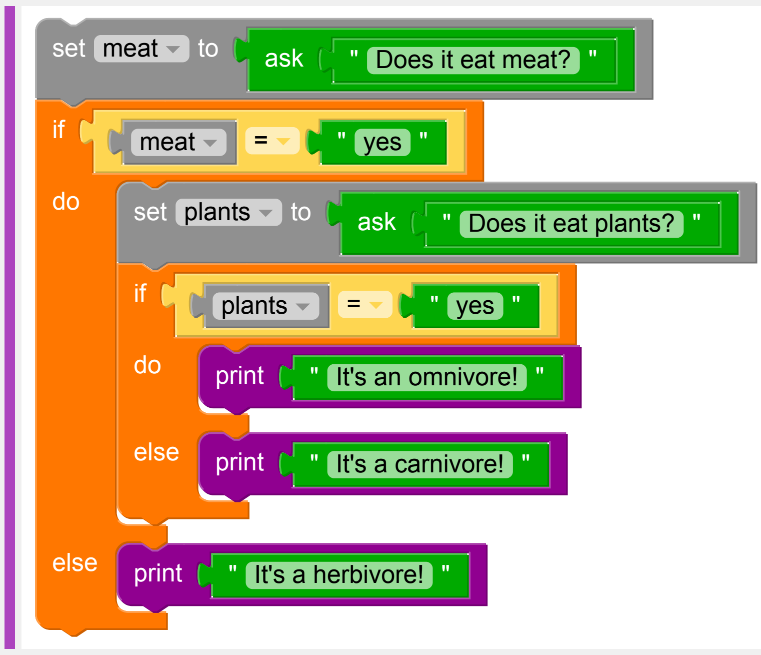
○


```
set meat to ask " Does it eat meat? "
if meat = " yes "
do  set plants to ask " Does it eat plants? "
    if plants = " yes "
    do   print " It's a carnivore! "
    else print " It's an omnivore! "
else print " It's a herbivore! "
```

○


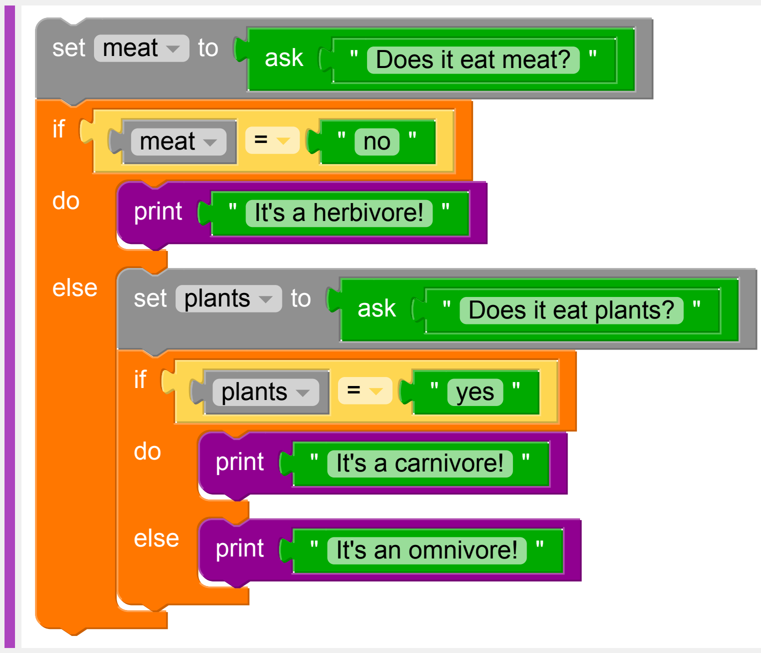
```
set meat to ask " Does it eat meat? "
if meat = " no "
do  set plants to ask " Does it eat plants? "
    if plants = " yes "
    do   print " It's a herbivore! "
    else print " It's an omnivore! "
else print " It's a carnivore! "
```

○



```
set meat to ask " Does it eat meat? "
if meat = " yes "
do  set plants to ask " Does it eat plants? "
    if plants = " yes "
    do   print " It's an omnivore! "
    else print " It's a carnivore! "
else print " It's a herbivore! "
```

○



```
set meat to ask " Does it eat meat? "
if meat = " no "
do   print " It's a herbivore! "
else set plants to ask " Does it eat plants? "
     if plants = " yes "
     do   print " It's a carnivore! "
     else print " It's an omnivore! "
```

Testing

☐ That's right!

5

MINI PROJECT: SIMPLE CLASSIFIER

5.1. Project description

5.1.1. What's the project?

So far in the challenge you've learned quite a lot about both Biology and Digital Technologies:

Biology

- Animals can be identified by looking at their physical features
- Animal features help them survive in their habitat

Digital Technologies

- Computers perform their functions through code *written by people*
- We can *display text* on the screen in Blockly using the `print` block
- The user can *enter text* into our program via the `input` block
- *Variables* are used to store data in a computer program
- An `if` block is used for *branching* in Blockly. It runs instructions when a condition is `True`.
- A block with an `elif` or `else` allows you to perform different instructions under different conditions.

This project guides you through the process of building your own animal identifier based on its features.

5.1.2. Classifying animals

The first step in constructing our classifier is to develop an algorithm that will allow us to correctly identify each of our animals using their features. The animals we will be classifying are:

- Koala
- Short-beaked echidna
- Eastern blue-tongue
- Laughing kookaburra

You can download a set of trading cards (https://aca.edu.au/public/resources/classifier-blockly.pdf) for these animals that gives you the information you need about their features to identify each animal uniquely.

5.1.3. Problem: Animal features

Using the information provided in the set of trading cards (https://aca.edu.au/public/resources/classifier-blockly.pdf), identify the features of each animal that will be used for identification.

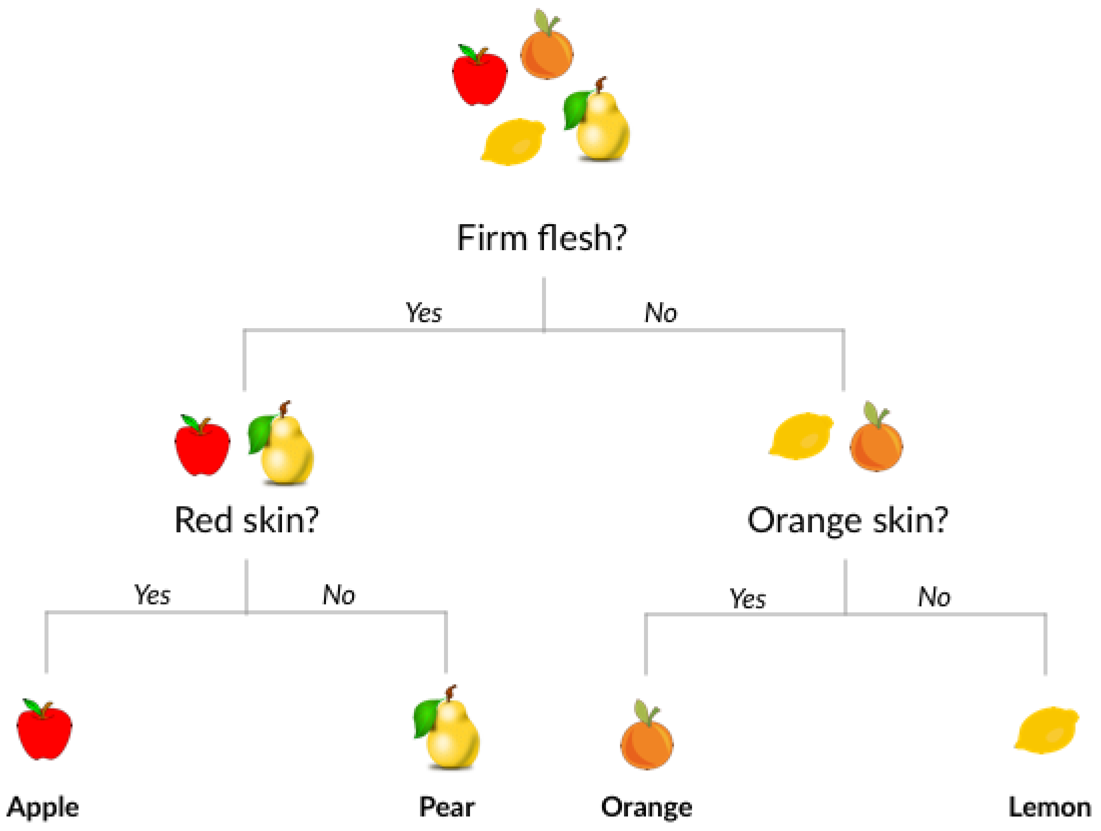
	Fur	Eggs	Claws	Scales	Spikes
Koala					
Short-beaked echidna					
Eastern blue-tongue					
Laughing kookaburra					

Testing

- ☐ Checking the "Koala" row.
- ☐ Checking the "Short-beaked echidna" row.
- ☐ Checking the "Eastern blue-tongue" row.
- ☐ Checking the "Laughing kookaburra" row.

5.1.4. Dichotomous tree

Now that you know which features belong to which animal, your next step is to work out an appropriate algorithm to identify them in your program. We used flowcharts to do this in our previous modules, but another way you can represent this is shown below:



Classifying fruit using their features

This example classifies the four different fruits based on their features using just two questions for each one. You'll be doing a similar thing with our four animals.

[This template (https://aca.edu.au/public/resources/classifier-template.pdf)](https://aca.edu.au/public/resources/classifier-template.pdf) will help you work out which questions you need to ask using the trading cards from the previous slides.

There are a few different solutions to this problem, but the best one will only require asking ***two questions*** to identify each animal uniquely.

You can check your solution with your teacher first, or progress to the next slide and write your program!

5.1.5. Problem: Biology classifier

Now that you know a little bit about the features of our four animals, the final step is to write the code that identifies the animals for us. The program will use features and questions you worked out in your algorithm to print out the name and habitat of the selected animal.

The program will operate according to the following rules:

- The input prompts specify just the feature being checked, followed by a colon, e.g. `Fur:` or `Eggs:`
- Answers will be `"y"` or `"n"` for all features
- Only features from the trading cards (https://aca.edu.au/public/resources/classifier-blockly.pdf) 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

One example that correctly identifies a koala might be:

```
Scales: n
Claws: y
Fur: y
Spikes: n
The species is koala.
It lives in bushland.
```

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*. **Challenge yourself and see if you can write a program that only asks two questions!**

Once the animal is identified, you should immediately print its common name and habitat to the screen. You must format the output of your program as shown in the example above.

Use your algorithm

If you've worked through the activities in each of the slides up to this point, you should be able to follow the same process we did in earlier modules to convert your algorithm into code.

Testing

☐ Testing the example animal from the question (koala).

☐ Testing the laughing kookaburra.

☐ Testing the short-beaked echidna.

☐ Testing the eastern blue-tongue.

☐ **Amazing! You made your own classifier! Well done!**

5.2. Project complete

5.2.1. Congratulations!

Excellent work on finishing the project! You've learnt a little bit about how biological organisms can be identified, as well as how to solve some interesting problems with code!

If you want to diver deeper into the programming concepts, check out our Pirate Chatbot course (https://groklearning.com/course/aca-dt-56-bk-chatbot/)!

(https://groklearning.com/course/aca-dt-56-bk-chatbot/)

Pirate Chatbot
(https://groklearning.com/course/aca-dt-56-bk-chatbot/)

(https://groklearning.com/course/aca-dt-56-bk-chatbot/)