# technocamps

UNDEB EWROPEAIDD
EUROPEAN UNION

Llywodraeth Cymru
Welsh Government

**Cronfa Gymdeithasol Ewrop**
**European Social Fund**

Prifysgol Abertawe
Swansea University

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

PRIFYSGOL BANGOR UNIVERSITY

Cardiff Metropolitan University
Prifysgol Metropolitan Caerdydd

i.t.wales

PRIFYSGOL ABERYSTWYTH UNIVERSITY

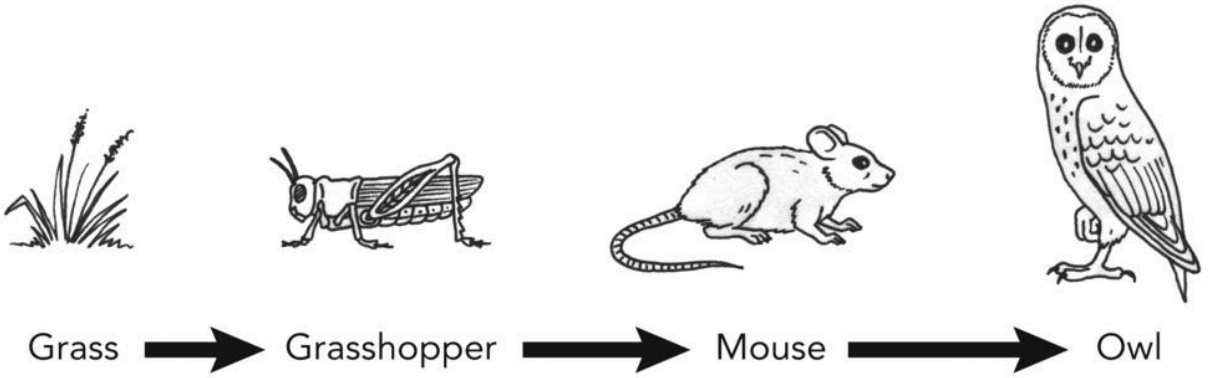
PRIFYSGOL glyndŵr Wrecsam
Wrexham glyndŵr UNIVERSITY

University of South Wales
Prifysgol De Cymru

technocamps

# Greenfoot Ecosystems

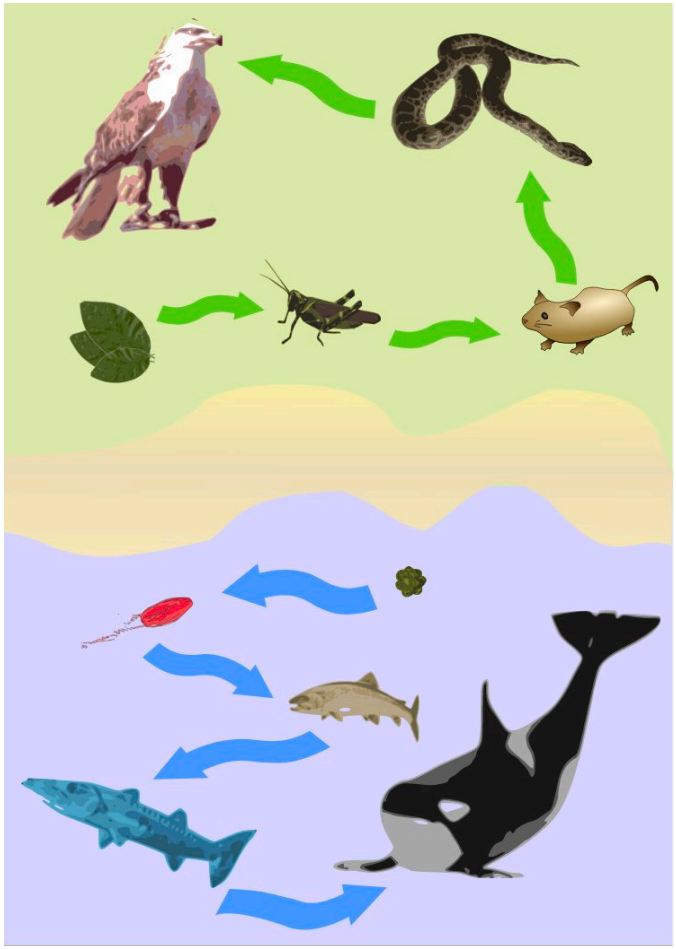
# Activity: What are Food Chains?



# Food Chains

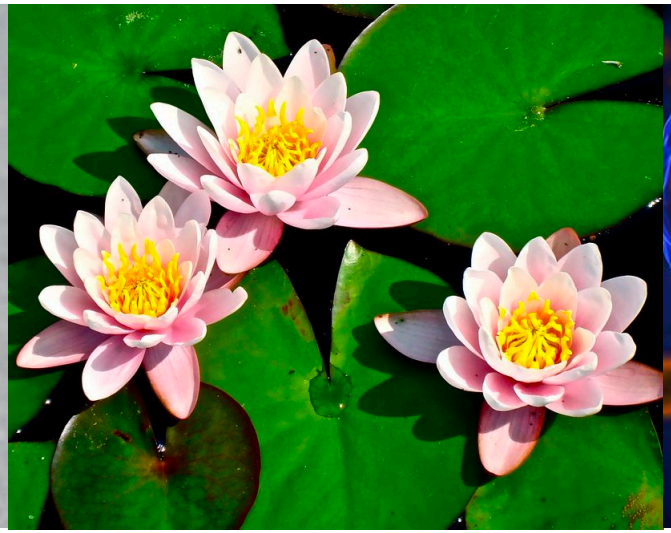
A **food chain** shows the flow of energy from one living thing to another.

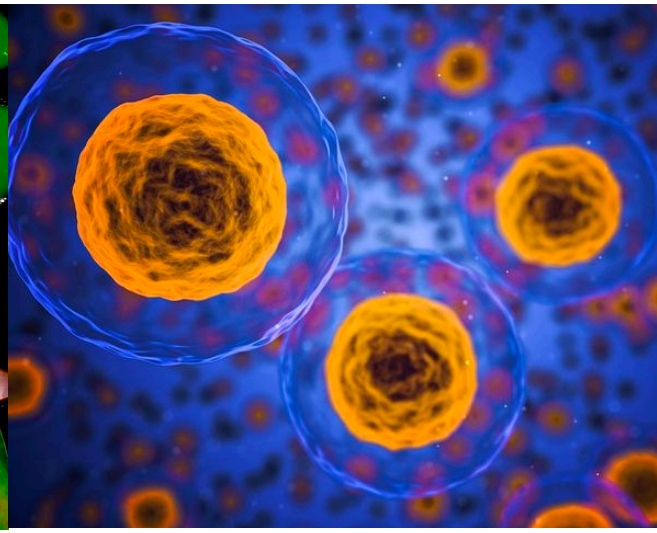
A **habitat** is the natural home or environment of an animal, plant, or another organism.



# What is an Organism?

An **organism** is an individual animal, plant, or single-celled life form.



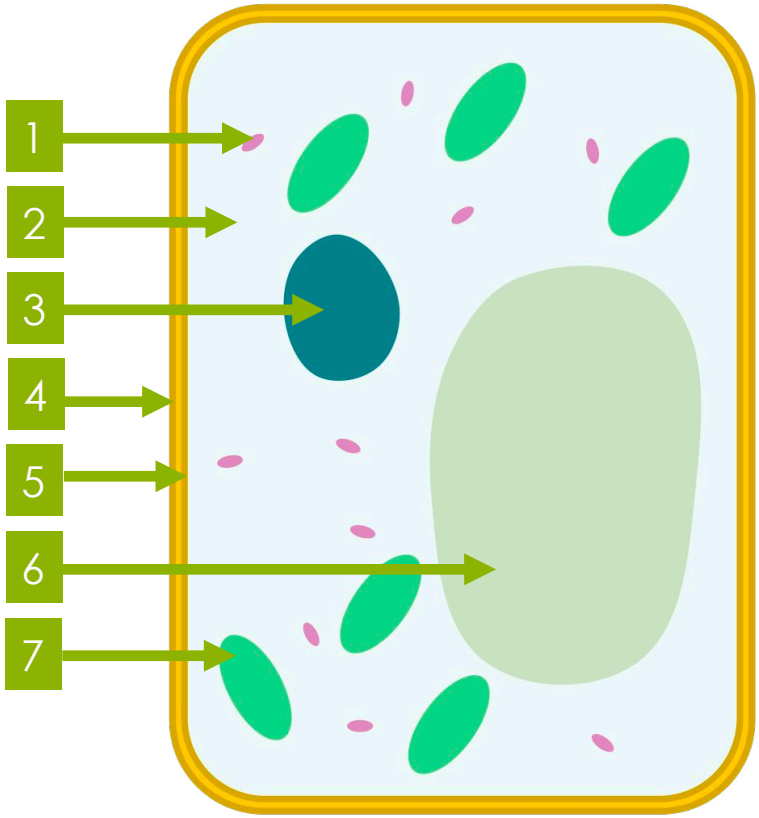


# Activity: Energy Source

What is the ultimate source of energy for most living things?

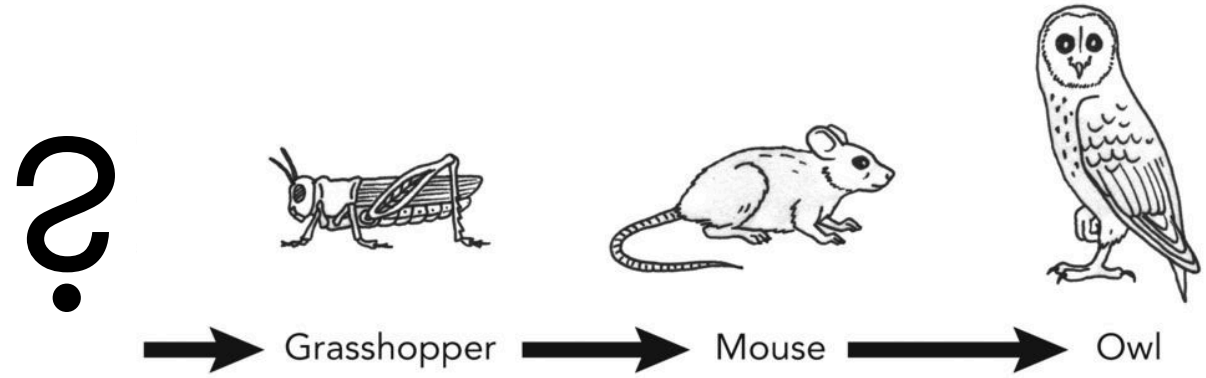
Does anyone know what kind of cell this is?

Extension: Do you know what 1, 2, 3, 4, 5, 6 and 7 are?



# Activity: What is a Producer?

A food chain begins with a **producer**.




In your workbook, write down what you think a **producer** is?

Can you think of an example of a producer?

# Producers

**Producers** are organisms that make their own foods.



# Object-Oriented Programming

# Object-Oriented Programming

Object oriented programming is a way of programming which is slightly different to (how we usually use) Python.

It is structured differently to Python's sequential way of coding.

Java uses Classes and Objects.

Does anyone know what these are? Has anyone used them before?

# Objects and Classes

Class:

A sweetened frozen food, usually made from milk, typically eaten as a dessert.

Objects:

Vanilla Ice Cream
Chocolate Ice Cream
Mint Choc Chip Ice Cream
Salted Caramel Ice Cream

# Class: Student

Imagine I had a class called Student, what properties does a student have i.e. what makes a student a student?

# Class: Student

A class is best understood as the blueprints/template for any objects of that class. A class describes the behaviour/state that the object of its type supports.

For example:

Take a class called Student. A specific student would be an object of the class Student. An example would be a girl named Jenifer, who is 12 years old, whose favourite subject is Chemistry. So each of these features would be defined in the class Student.

| Object | Gender | Name | Age | Favourite Subject |
|---|---|---|---|---|
| student1 | Female | Jenifer | 12 | Chemistry |
| student2 | Male | Tomos | 13 | History |

# Class: Student

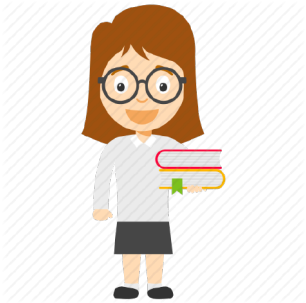
Tomos
Male
13
History

Class

Student:

Name
Gender
Age
Favourite Subject

Objects



Jenifer
Female
12
Chemistry

Luke
Male
15
D&T

# Activity: Actors and World


The classes we are given in Greenfoot are Actors and World. If we look at Super Mario for example, which parts are the Actors and which parts are the World?

Discuss with a partner which objects are part of the Actor class and which objects are part of the World class.

# Greenfoot

# What is Greenfoot?

Greenfoot is an introductory visual programming environment using the "Java" programming language, a highly valued language in the Computer Science Industry.

http://www.greenfoot.org/download

# Greenfoot Version 2.4.2

The version we will be using is Greenfoot Version 2.4.2 in order to make sure we're all on the same page and have the same methods available to us.

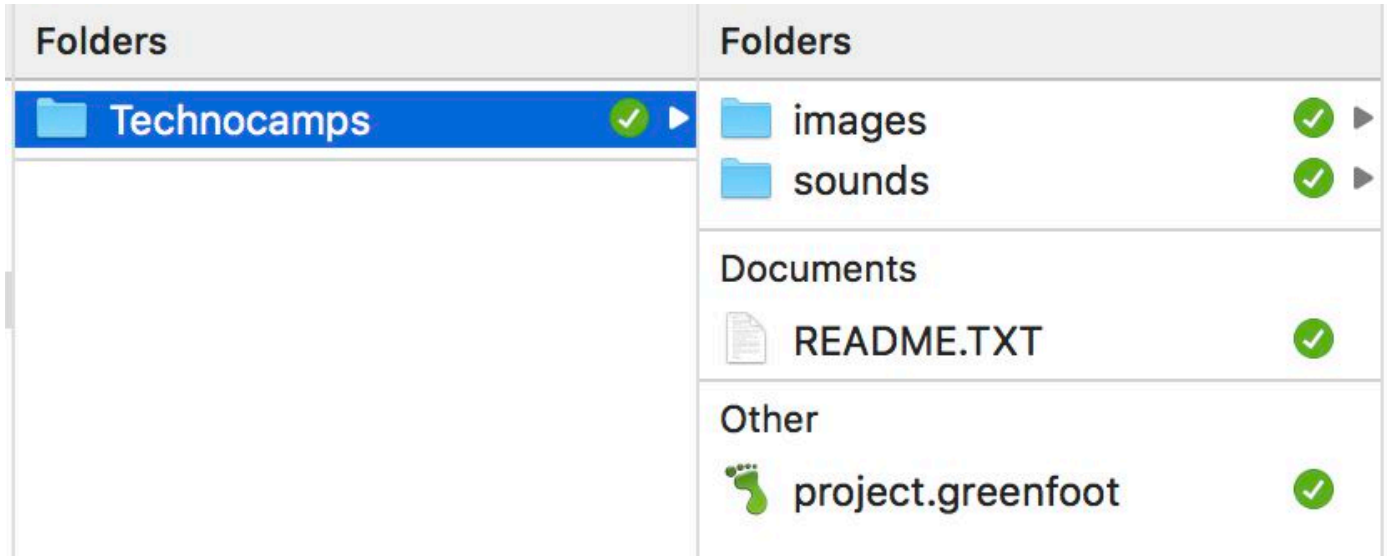
# Activity: Greenfoot

First create a scenario:

1. Click on ‘Scenario’ on the top left and then ‘New Scenario’.
2. Name Your Program. Do not call your Scenario “Greenfoot”!

# Starting Greenfoot

This creates a folder containing the project file and anything we create will also be saved automatically in this folder.

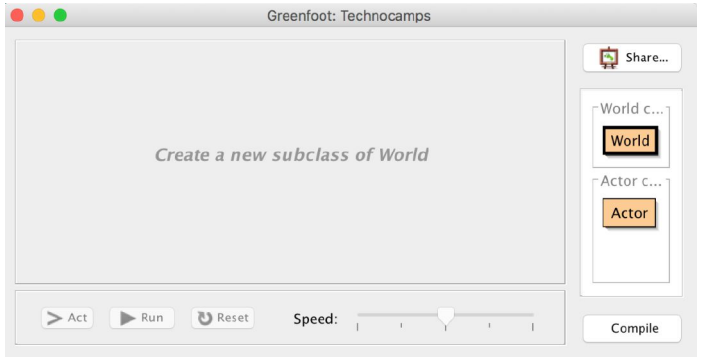


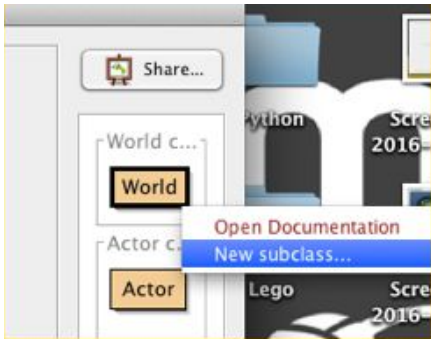
# Creating a New World

When the Greenfoot window opens it should look like the image below. To create our world (the environment where the game is happening):

- Right-click on '**World**',
- Select the option '**New subclass**' and name that subclass '**Habitat**'.

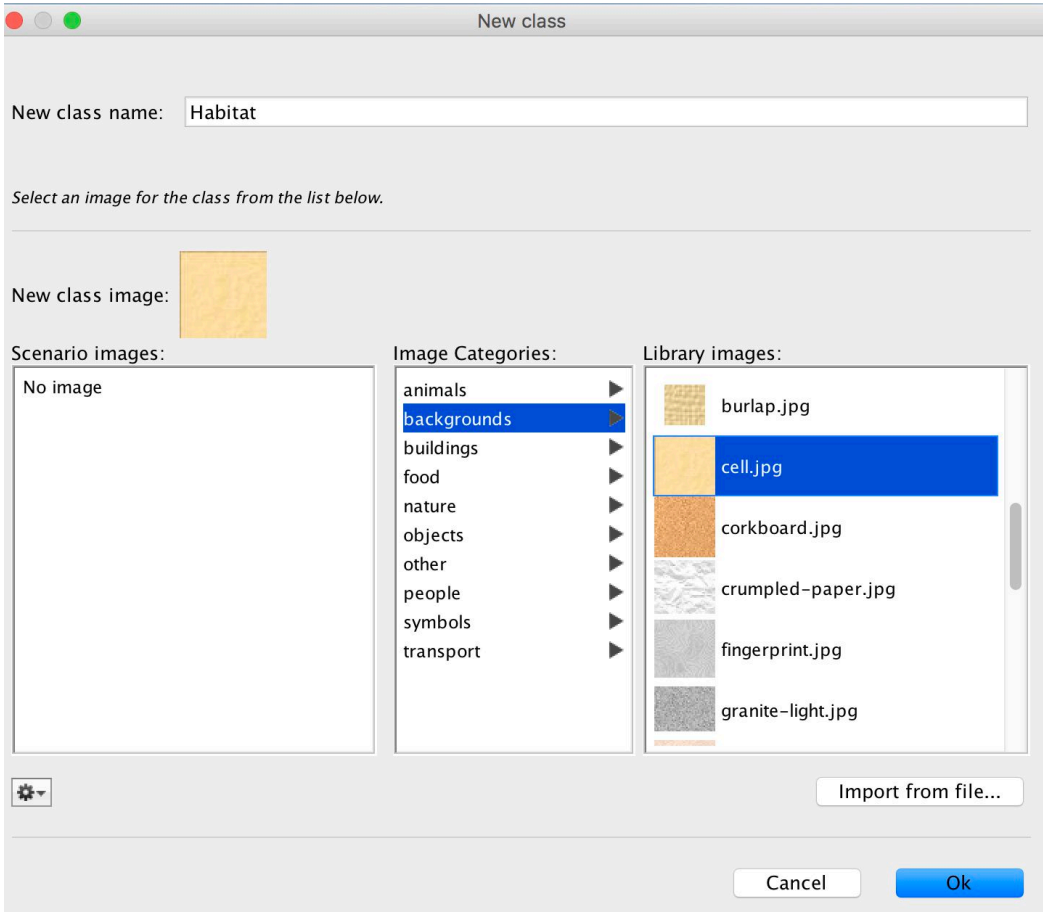
When naming classes in Java we always start with a capital letter and capitalise the first letter of each new word, never with spaces.



# Creating a New World

Choose the ‘Cell’ image

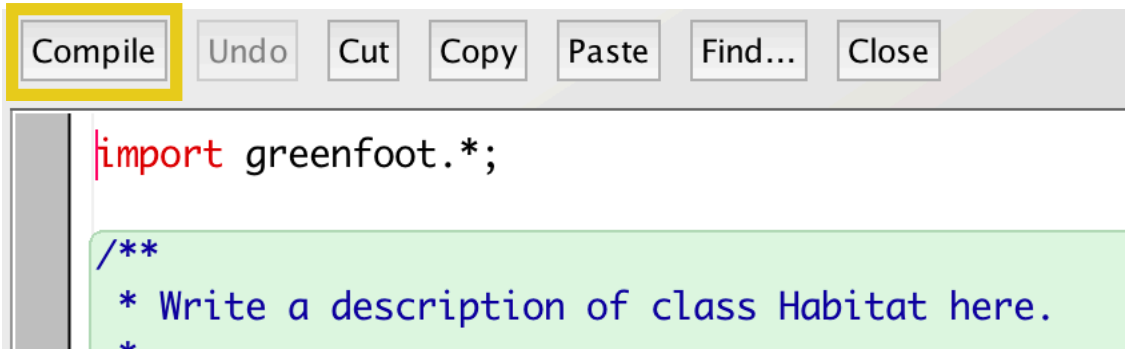


# Changing Your World

Now change your 'World' by changing the number of 'cells' or 'grids' in our program. To do this:

1. Right-click on your 'World' subclass i.e. 'Habitat'
2. Click 'Open editor'

Note: Whenever you change the code in your subclass, do not forget to click on the 'compile' button to update the changes.



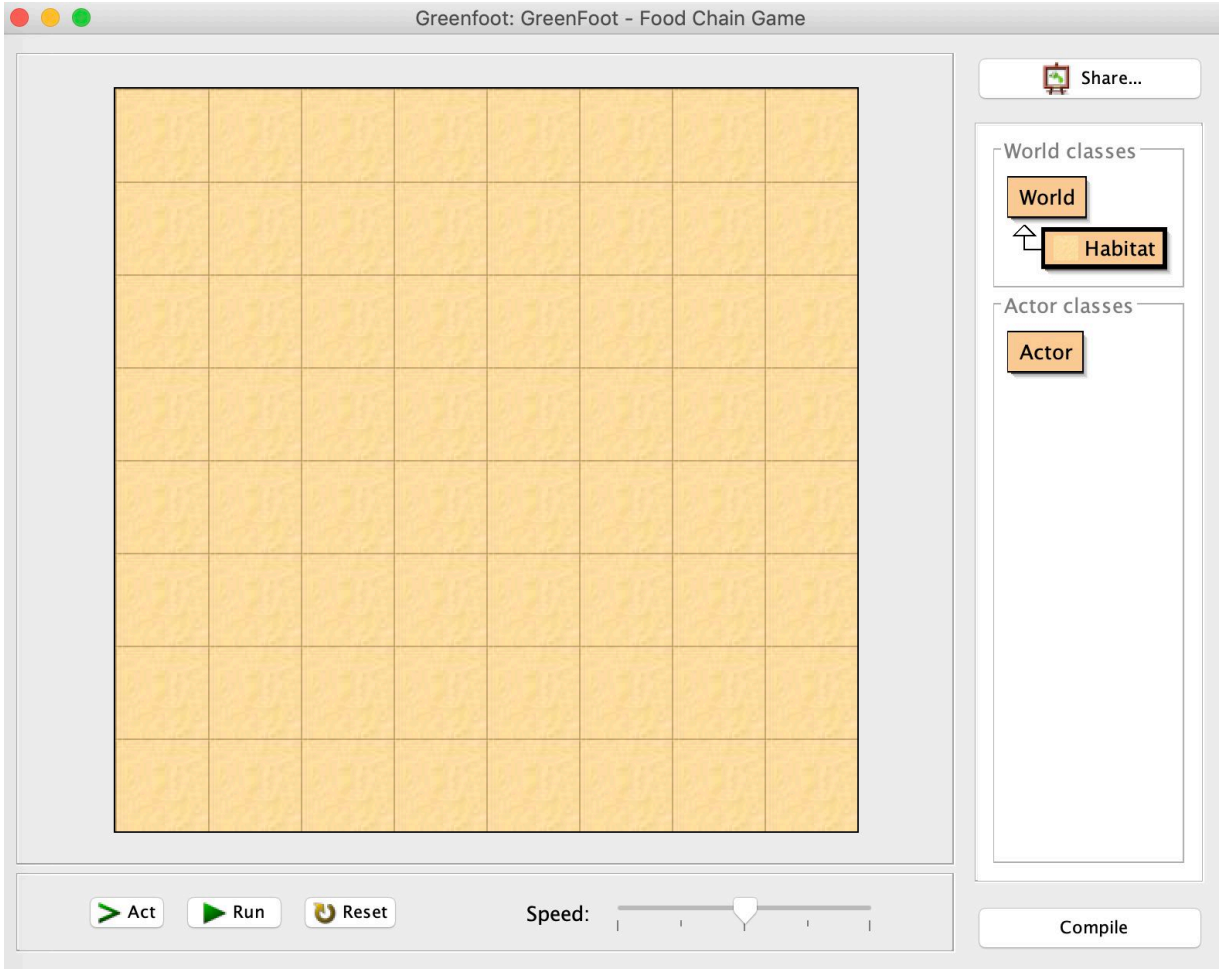
# Changing the Grid

Currently as default the 'World' contains a 'grid' of 600x400 with a size of 1x1 (which translates to a lot of cells, but with a very small size).

Change it so that it is 8x8 cells, that is to say it should contain 8 rows and 8 columns, and each cell has the size of 60x60 pixels.

```
public class Habitat extends World
{

    /**
     * Constructor for objects of class Habitat.
     *
     */
    public Habitat()
    {
        // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
        super(600, 400, 1);

    }
}
```

# Your Program should look like this:



# Greenfoot Coordinate System

X

Y

| X,Y | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|---|
| **0** | 0,0 | 1,0 | 2,0 | .. | .. | .. | .. | 7,0 |
| **1** | 0,1 | | | | | | | .. |
| **2** | 0,2 | | | | | | | .. |
| **3** | .. | | | | | | | .. |
| **4** | .. | | | | | | | .. |
| **5** | .. | | | | | | | .. |
| **6** | .. | | | | | | | .. |
| **7** | 0,7 | .. | .. | .. | .. | .. | .. | 7,7 |

# Actors

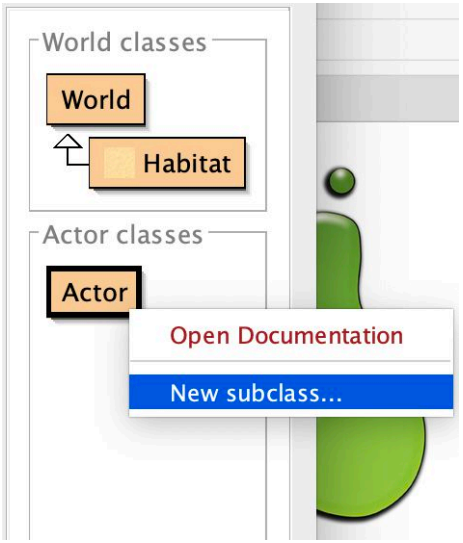
Below the 'World' class, there is an '**Actor**' class. This class is used to create objects to be placed into your 'World'. Objects like:

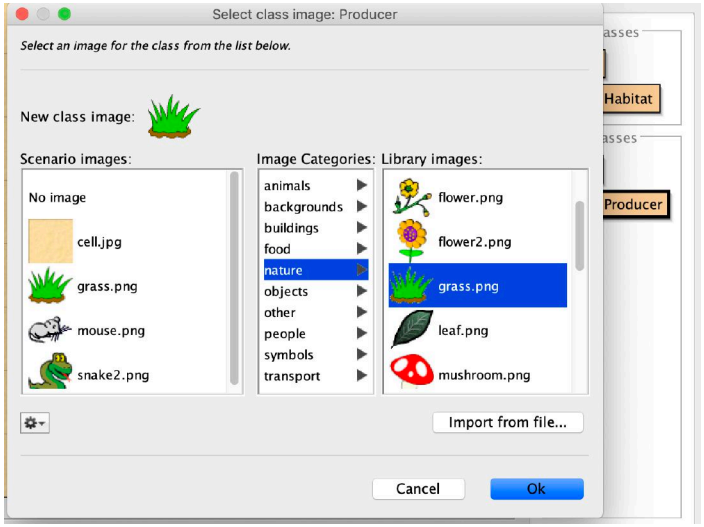
- Main characters: hero, man, woman...
- Animals: rabbit, wombats...
- Collectables: flowers, ball...

# Actors

To create a **subclass:**

- Right-click on the '**Actor**'
- Select the option '**new subclass**'. We will name our first 'Actor' **subclass** as '**Producer**'.
- Select the grass image.



# Adding Our Actor (Producer) into the World (Habitat)

Now we need to program inside our 'World' subclass '**Habitat**'.

- Right-click on your '**Habitat**' subclass
- Click '**Open editor**'. This will open up the programming window for the '**Habitat**', subclass.
- Use the following line of code:

```
Producer grass1 = new Producer();
```

Question: What is this line of code saying?

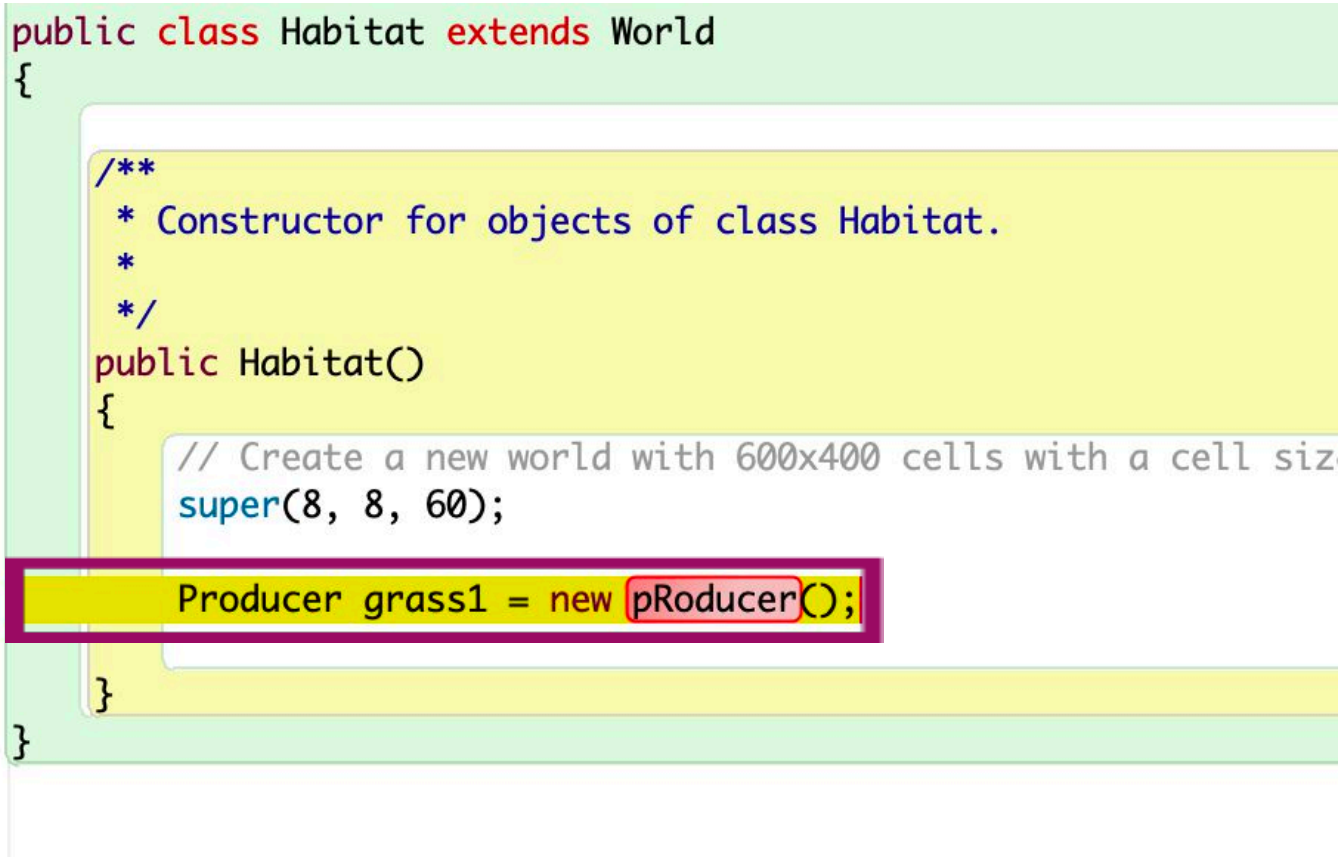
# Java Syntax

Syntax is what we call the programming rules of a computer language. In Java we have to finish **most but not all** lines of code with a semi colon ";". Without the semi-colon the programming file cannot be compiled (i.e. translated for the computer to understand).

```
 * @version (a version number or a date)
 */
public class Habitat extends World
{

    /**
     * Constructor for objects of class Habitat.
     *
     */
    public Habitat()
    {
        // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
        super(8, 8, 60);

        Producer grass1 = new Producer();

    }
}
```

# Syntax Error

Greenfoot helpfully highlights any errors in your code.



```
public class Habitat extends World
{

    /**
     * Constructor for objects of class Habitat.
     *
     */
    public Habitat()
    {
        // Create a new world with 600x400 cells with a cell siz
        super(8, 8, 60);

        Producer grass1 = new pRoducer();

    }
}
```

# Java Syntax

{ } Braces (or Curly Braces) are very important in Java. They are used to group statements and declarations.

[ ] Brackets (or Square Brackets) are used for indexing a list.

( ) Parentheses are used to control the order of operations in an expression and to supply parameters to a method or function.

# Java Syntax

// Double slash is used for single line comments in Java.

/*
* Multiple line comments in Java.
*/

; Every statement in Java ends with a semi-colon.

# Naming Conventions

| | |
|---|---|
| Class: | The class name should always start with **uppercase.** E.g. MainCharacter |
| Object: | The object name should always start with **lowercase.** E.g frogConsumer |
| Variable: | The variable name should start with **lowercase** and should have **camel case.** E.g. scoreCounter |
| Constant: | The constants should be named using **upper case.** E.g. MAX_LIFE |
| Filename: | The Java/Scenario file name should start with **uppercase** and continue with **camel case.** E.g. GameOfLife.java |

# Difference between Actor and Actor's object

```
Producer grass1 = new Producer();
```

Question: what is the difference between the 'Producer' and 'grass1'?

- 'Producer' is an Actor.
- 'grass1' is an object of type Producer.

Breakdown:

- the Actor 'Producer' is a blueprint.
- 'grass1' is an object made from that blueprint.

One contains the plans for creating the object. The other is the object made by those plans/blueprint.

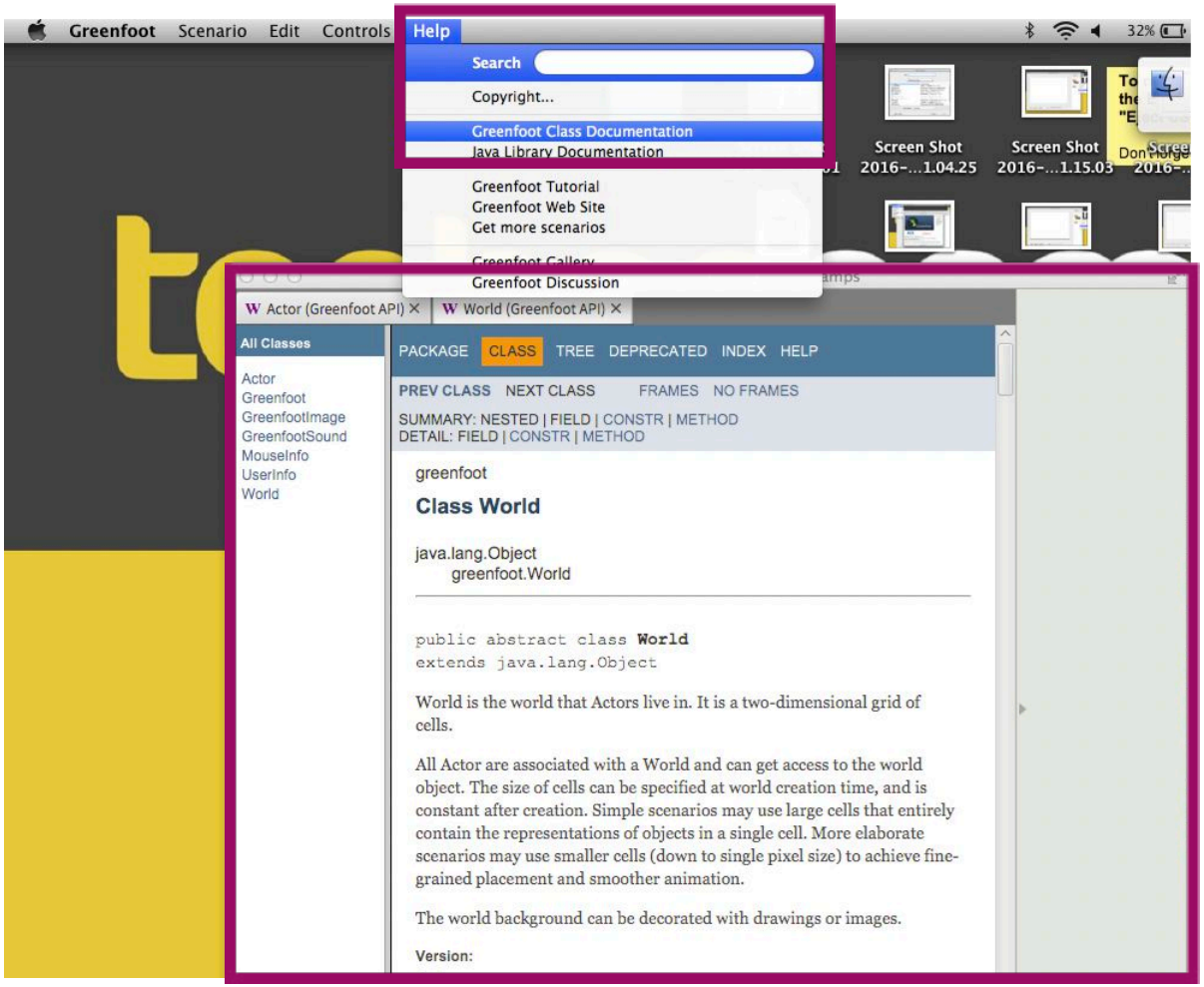
# Greenfoot Documentation

Documentation is what we call the material that provides official information or evidence about the language i.e. Greenfoot.

In Greenfoot there is a whole website that covers all of the pre-defined methods that are available to the user. These methods are things like:

- move()
- turn()
- removeTouching()
- setRotation()

# To Find The Documentation



# Displaying our Actor Objects

There is an object called ‘grass1’ of type ‘Producer’ which is created in the ‘Habitat’. Note we have just created the object but it is not automatically displayed.

To display ‘grass1’, use the Greenfoot function ‘addObject();’.

This function is already pre-defined into the Greenfoot program and can be found in the ‘Greenfoot Class Documentation’ page. **All we have to do is use it.**

# Using the addObject() Method

To find out how to use this method, utilise the documentation.

```
void                    addObject(Actor object, int x, int y)
                        Add an Actor to the world.
```

It requires three arguments (values) inside the brackets, what do you think these are?

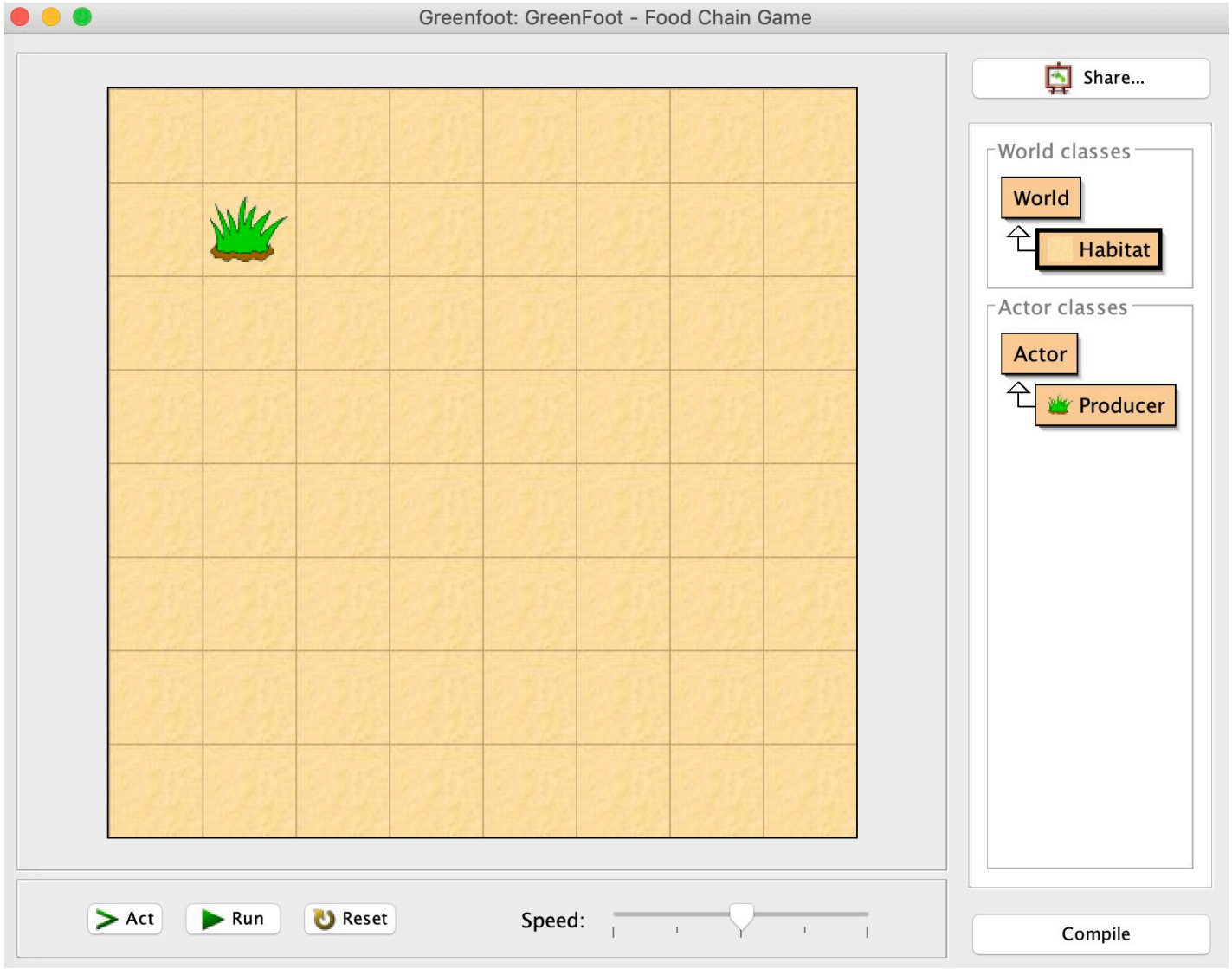
# addObject()

Discuss with the person next to you about how to use the addObject method. Then try and use the addObject(..., ..., ...) method to add your Actor object into your 'Habitat' world.

# addObject()

```
public Habitat()
{
    // Create a new world with 600x400 cells with a cell
    super(8, 8, 60);

    Producer grass1 = new Producer();
    addObject(grass1, 1, 1);

}
```

# Does it look like this?



# Object Position

The grass is now visible. It is unlikely to have appeared in the very top left square of the grid.

What values would you have to put into the addObject() function to achieve this?

What about the bottom right corner?

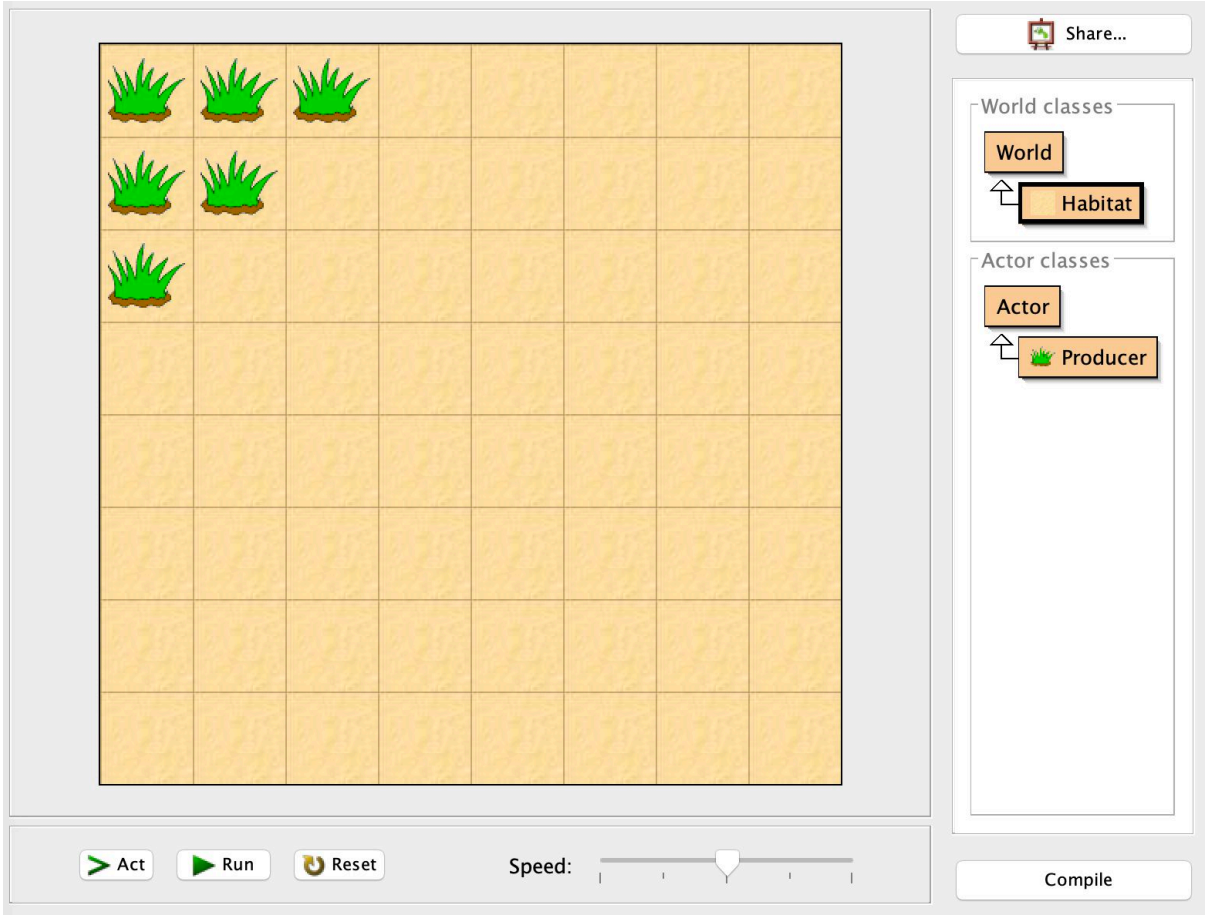
And the other two?

# Add multiple 'Producer' objects into the world

**Create** and **display** multiple '**Producer**' objects into your '**Habitat**'. Speak to the person next to you and talk about how you think this is done.

Note: you do **not** need to make a new Actor subclass!

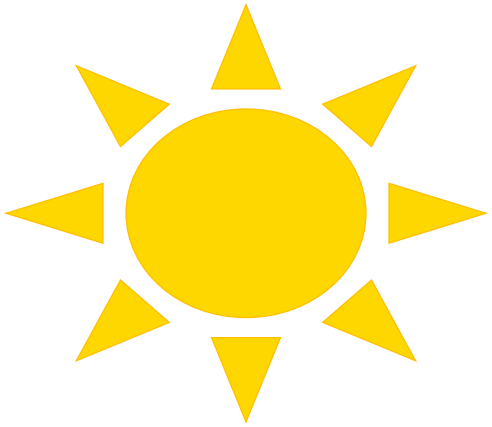
# Extension: Make Your Program Look Like This



# Activity: Name the Process

What is the name of the process in which a **producer** makes food?

Extension: Discuss how this process works.

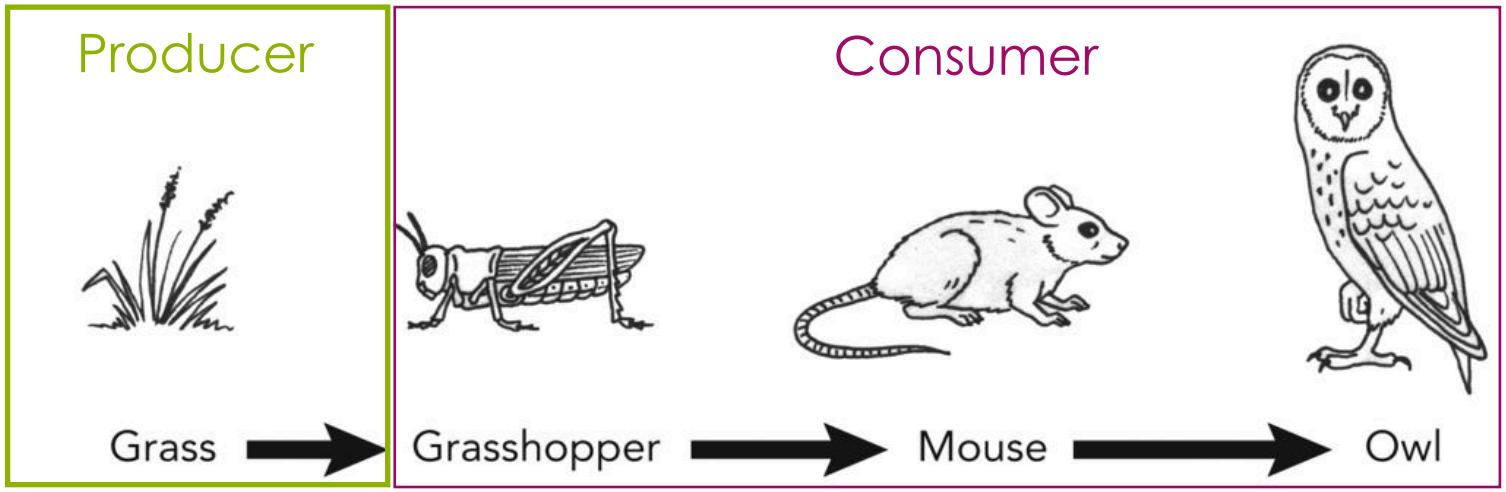


# Photosynthesis

Producers are organisms that make their own organic nutrients or food - usually using energy from sunlight. This process is called, **photosynthesis**.

**Photosynthesis** is the chemical process where plants make glucose (or carbohydrates) and oxygen, from carbon dioxide and water, using light energy.

# What is a Consumer?




The other organisms in a food chain are **consumers**, animals that eat plants or other animals.

They all get their energy by consuming other organisms.

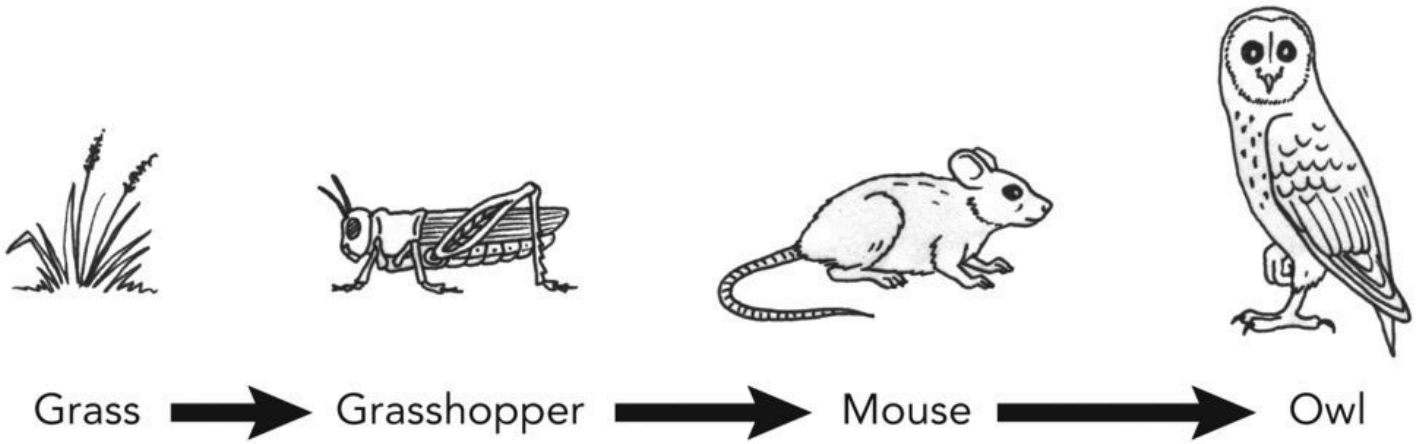
# Consumer Positions

| Organism | How it gets its energy |
|---|---|
| Consumer | Feeding on other organisms |
| Primary consumer | Eating plants |
| Secondary consumer | Eating primary consumers |
| Tertiary consumer | Eating secondary consumers |
| Herbivore | Eating plants |
| Carnivore | Eating other animals |
| Decomposer | Feeding on dead and decaying organisms, and on the undigested parts of plant and animal matter in faeces |

**https://www.bbc.com/bitesize/guides/z2m39j6/revision/1**

# Activity: Consumers

Producer




| | |
|---|---|
| Tertiary consumer | Eating secondary consumers |
| Secondary consumer | Eating primary consumers |
| Primary consumer | Eating plants |

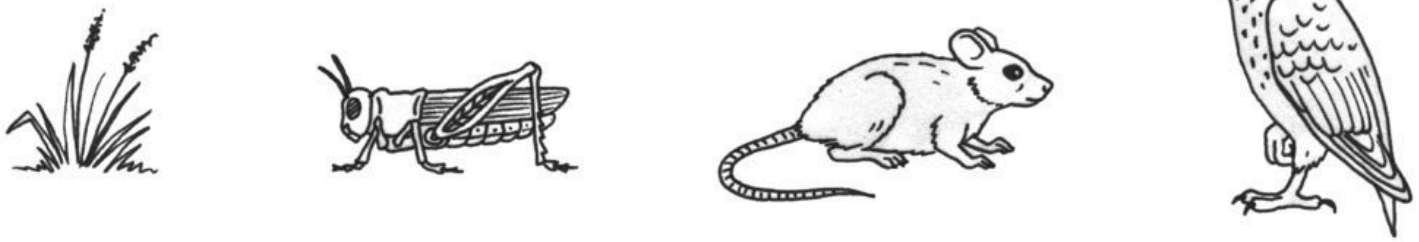
# Consumer Positions

Producer

Primary Consumer

Secondary Consumer

Tertiary Consumer

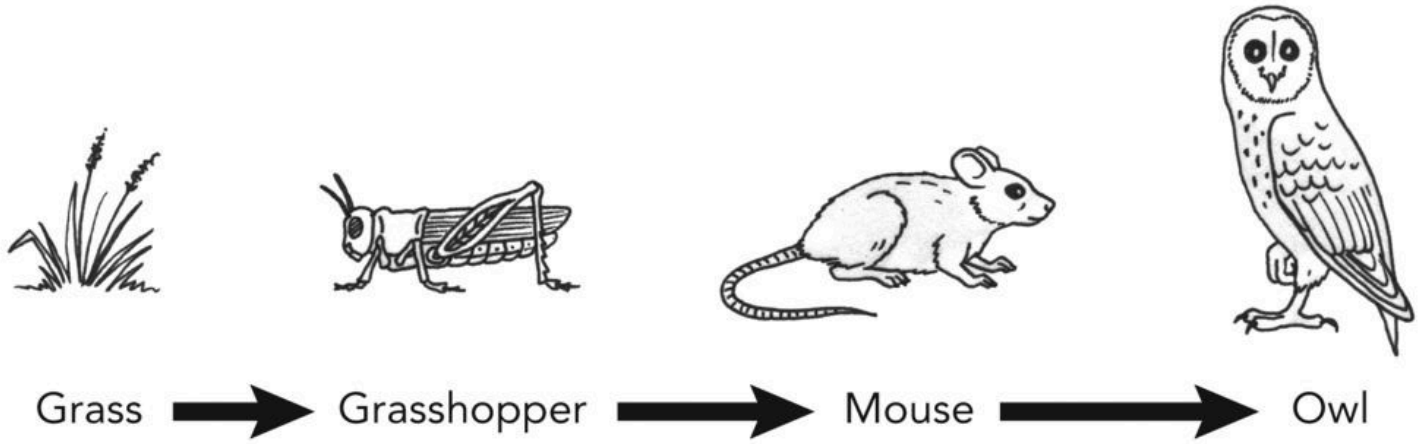


Grass → Grasshopper → Mouse → Owl

# Example Food Chain

In this example, grass is eaten by grasshoppers, and grasshoppers are eaten by mice, and mice are eaten by owls.

The arrows between each **organism** in the chain always point in the **direction of energy flow** from the food to the feeder.



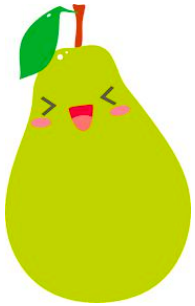
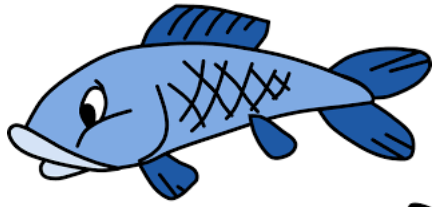
# Activity: Food Chain

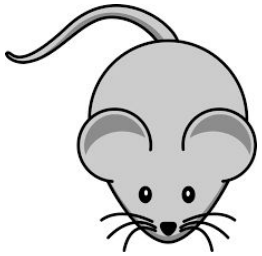
Group organisms on the following slide into **producers**, **consumers**, **herbivores** and **carnivores**.

Form some food chains from the provided organisms.

Group organisms on the following slide into **primary consumers**, **secondary consumers**, **tertiary consumers** (answers depend on chosen food chain).

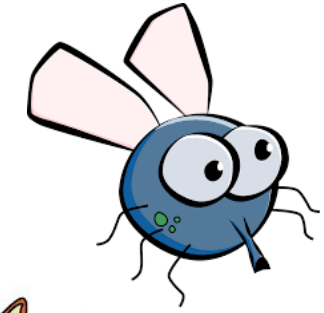


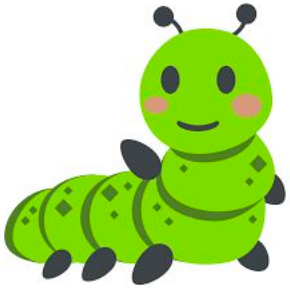







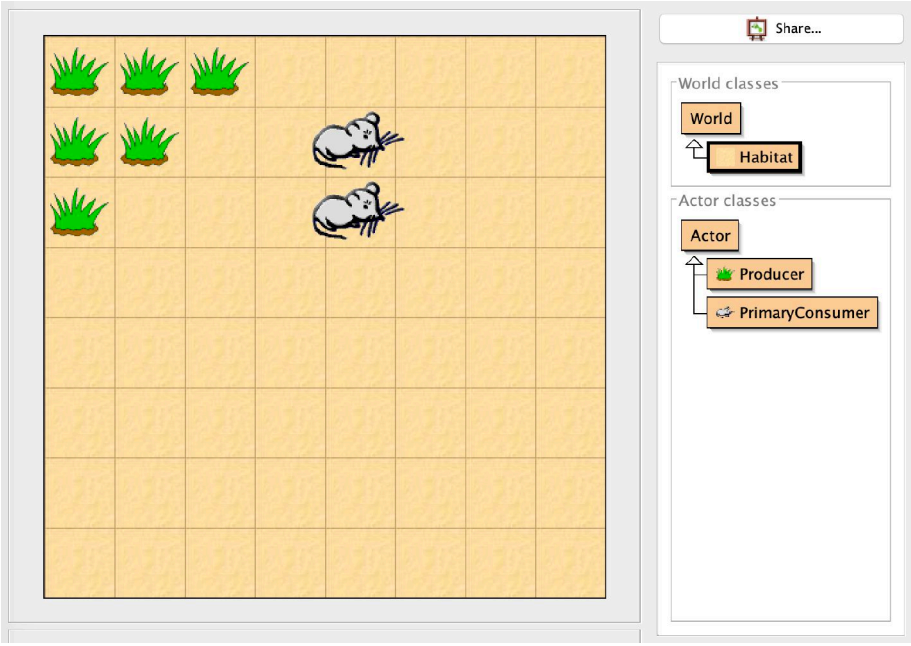




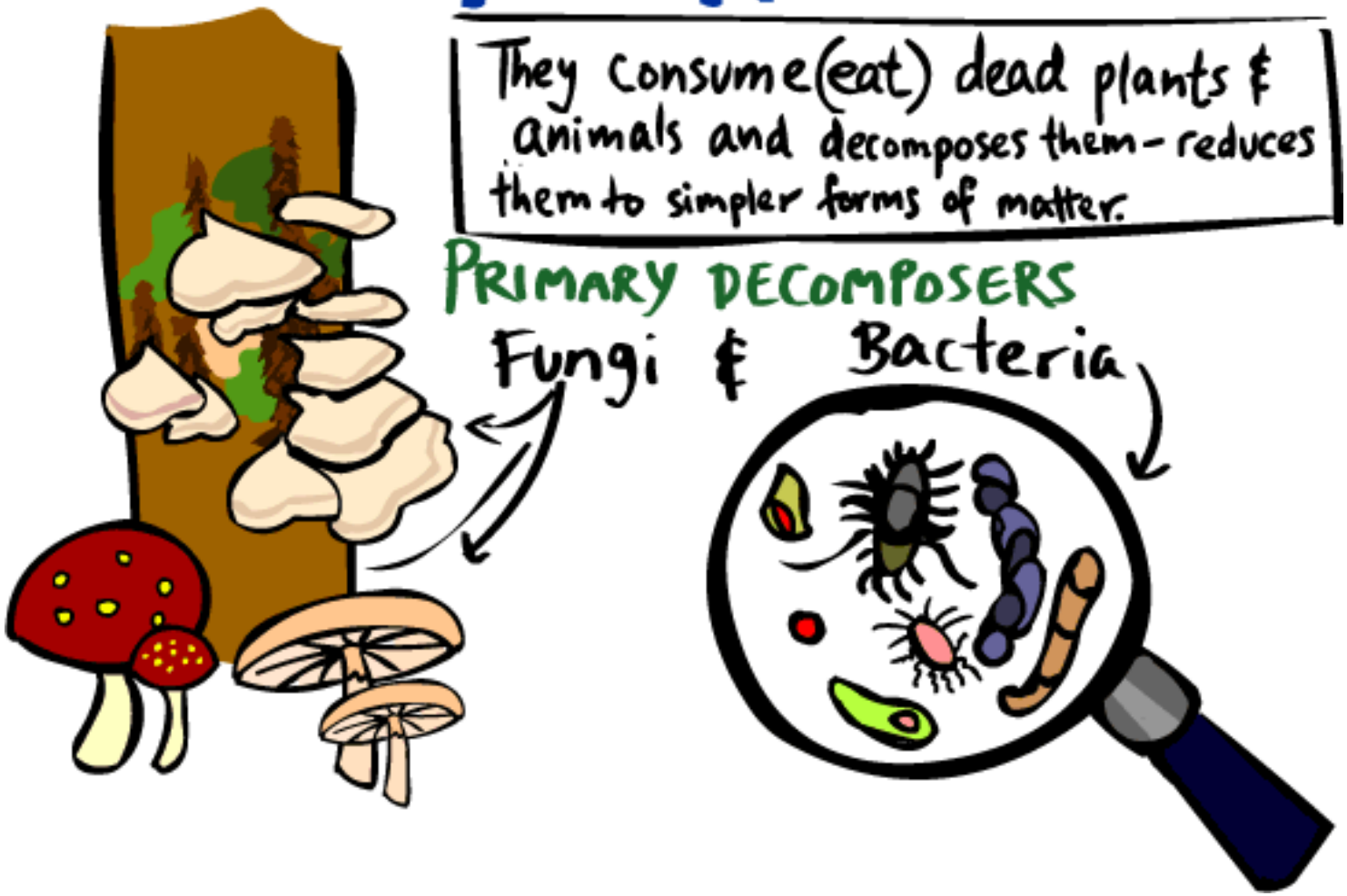




# Activity: PrimaryConsumer

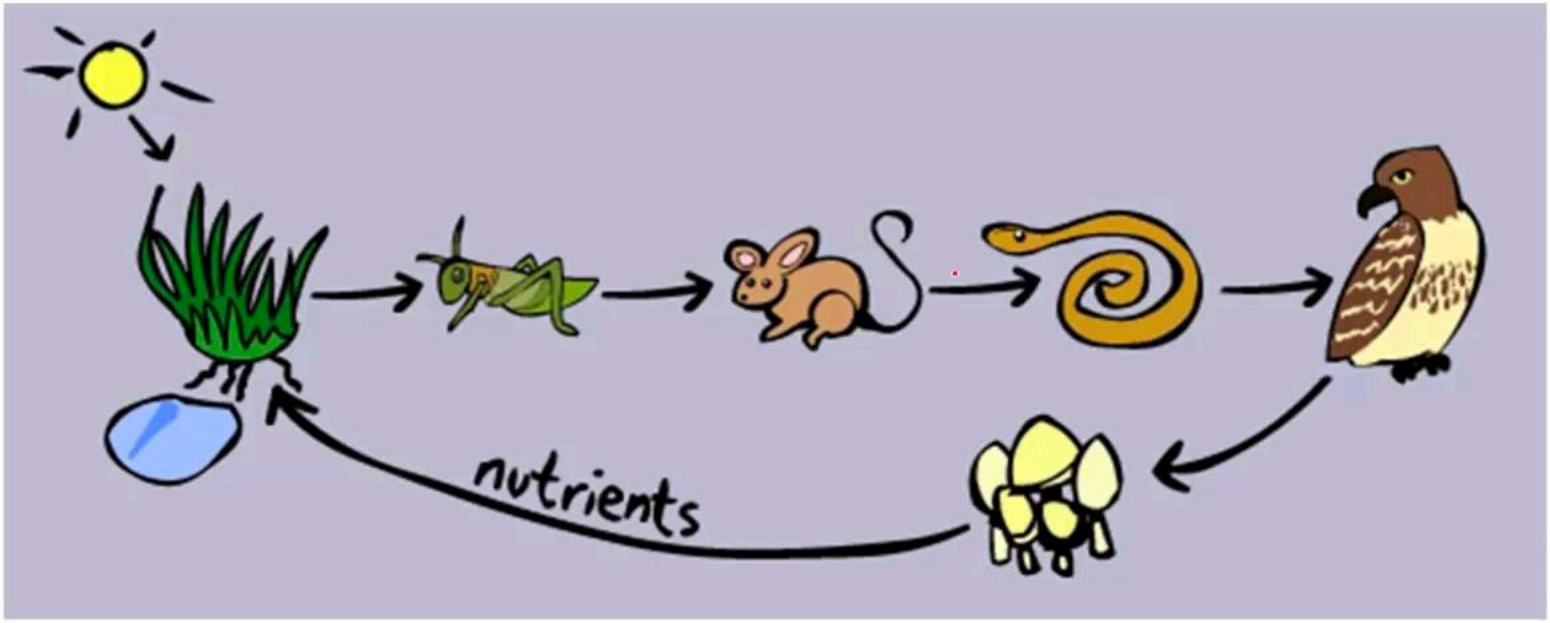
- Create a second **Actor** subclass called **PrimaryConsumer**.
- Set image to a **mouse**.
- Create **two** objects of the **PrimaryConsumer** and get it to display in the **middle** of your '**Habitat**'.



# DECOMPOSERS



# Decomposers' Role in the Food Chain

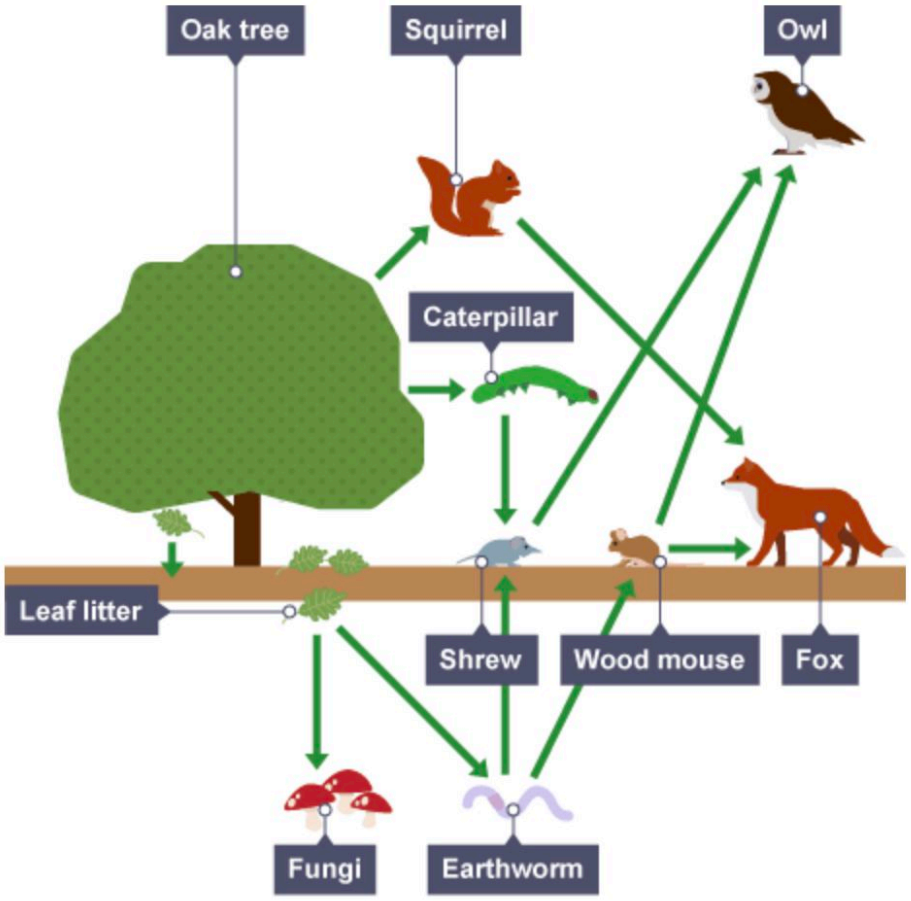


# What is a Food Web?

A **food web** is a network of interconnected food chains. It shows the energy flow through part of an ecosystem.

An **ecosystem** is a community of animals, plants and microorganisms, together with the habitat where they live.

# Food Web




A woodland food web

https://www.bbc.com/bitesize/guides/z2m39j6/revision/2

# Food Web

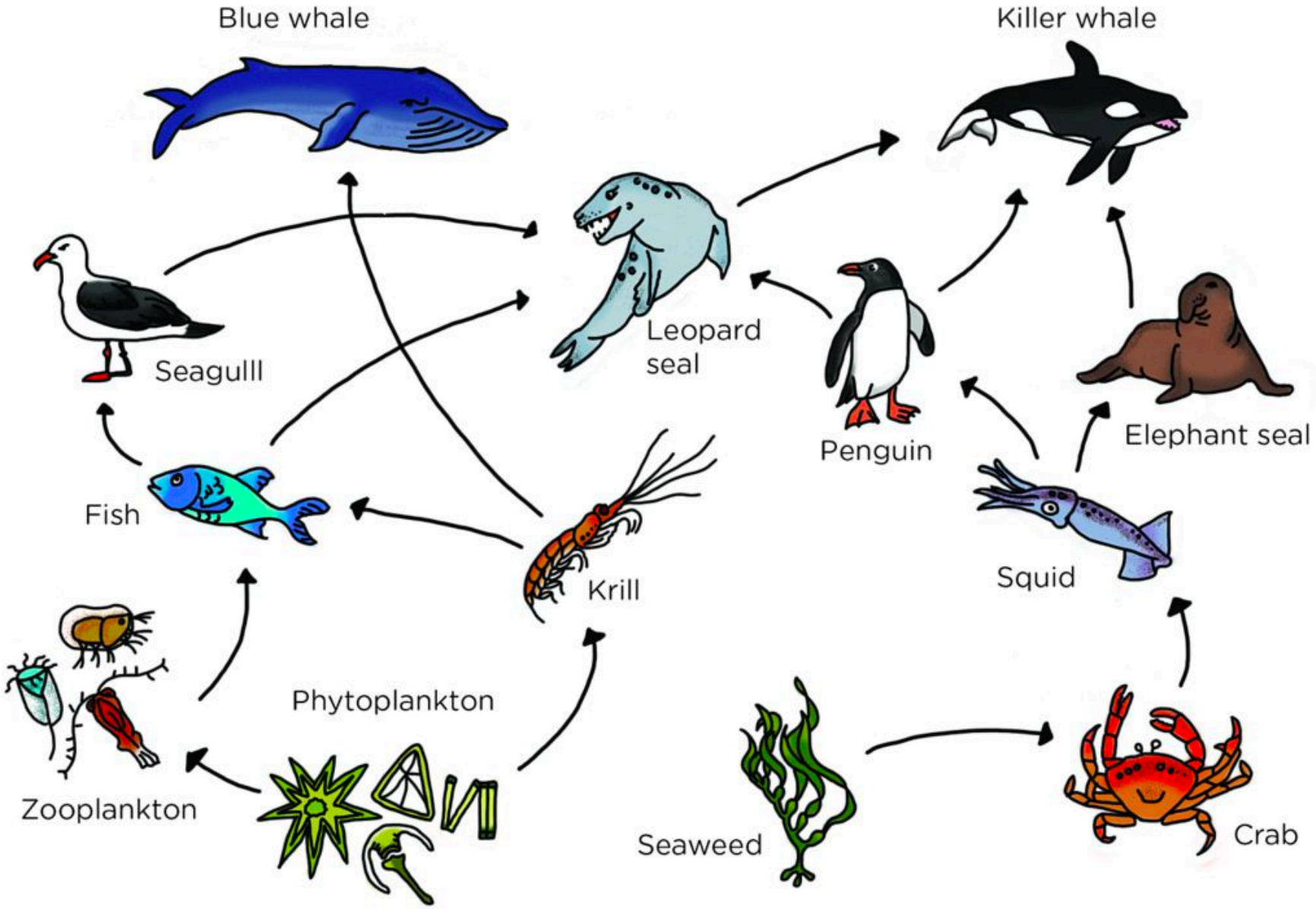
Here are three food chains from the food web on the previous slide:

- Oak tree → Squirrel → Fox
- Oak tree → Earthworm → Wood mouse → Fox
- Oak tree → Earthworm → Wood mouse → Owl

# Activity: Food Web

Using the food web on the following slide:

- Create a food chain with a length of three
- Create a different food chain with the length of four
- Create the longest food chain possible in this food web



# Energy Transfer

Energy is transferred along food chains from one level to the next. However, the amount of available energy decreases from one level to the next.

**Energy Loss:** In a food chain, only around **10%** of the energy is passed on to the next organism. The rest of the energy passes out of the food chain in a number of ways:

- It is used as heat energy.
- It is used for life processes such as movement.
- Faeces and remains are passed to decomposers.

# Activity: Energy Transfer

Currently, we know that in a food chain only around **10%** of the energy is passed onto the next level.

Let us say we have a producer (mushrooms):

Its total energy intake is 5000kJ (kilojoules).

What amount of energy would be passed on to the next consumer if it were eaten? Work in pairs to figure out the answer.

Kilojoules (kJ) – is the measure of energy values in foods.

# Activity: Greenfoot Ecosystem

# Changing the Behaviour of the Consumers

In order to define what an object can do, or how it behaves, we can use methods.

Methods are sets of instructions that we can use in our program, for example we can use methods such as **turn()** and **move()** to turn and move the objects in our world.

# Activity: Use Greenfoot Documentation

Use Greenfoot documentation to research the following methods:

- turn();
- move();
- setRotation();

Speak to the person next to you and talk about these methods, what you think they do and how you would use them.

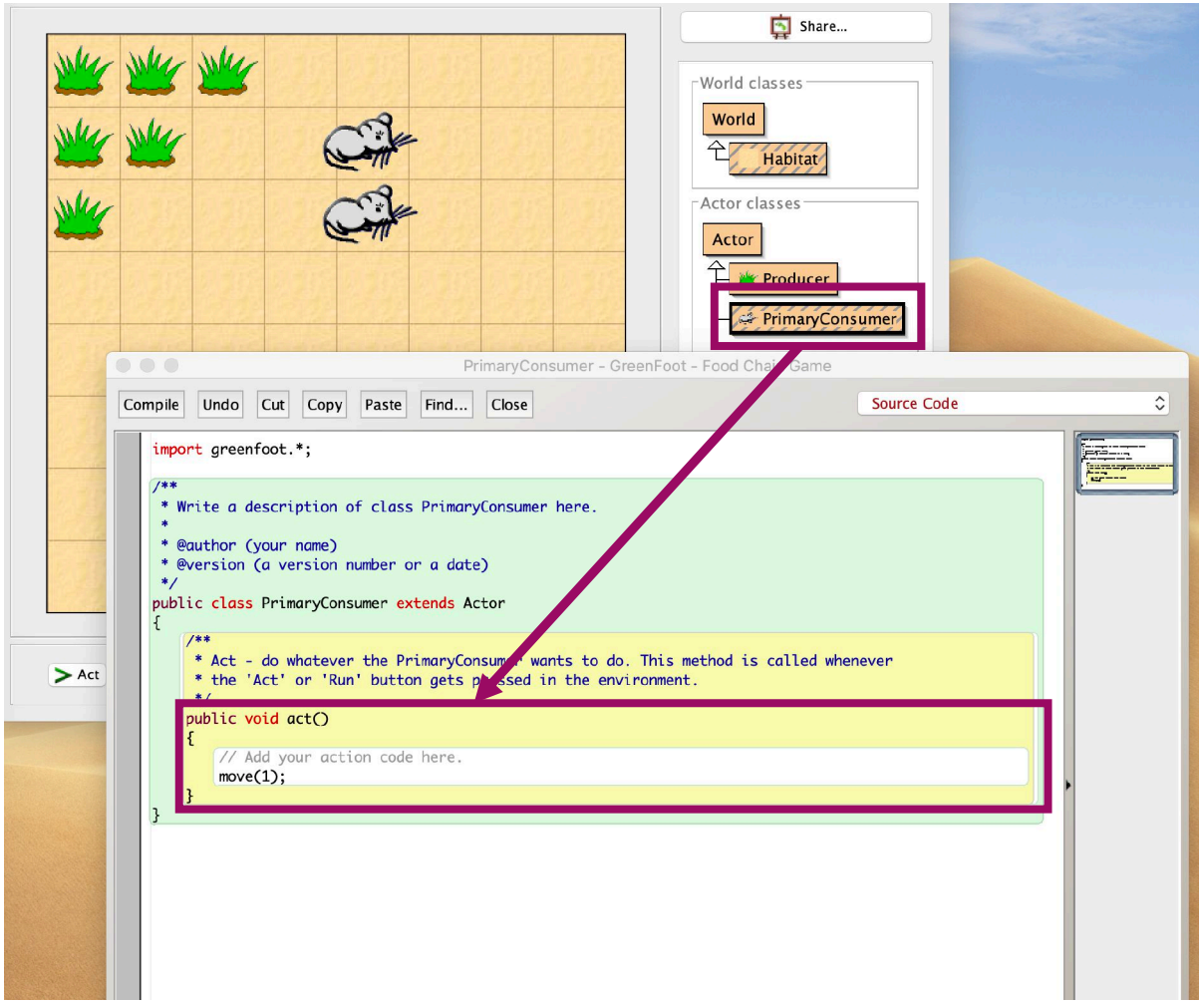
Extension: look up methods, isTouching(); and removeTouching():

# Activity: Program your Consumer Objects to Move

Use what you found in the Greenfoot documentation to program your PrimaryConsumer objects to move by 1 step.

Can you also get your objects to turn or rotate?

# Activity: Program your Consumer Objects to Move



# Activity: Random Movement

One way to make our game more fun is to have them moving around randomly. One way to do this is to use random numbers.

Generate a random number in Greenfoot using the method Greenfoot.getRandomNumber(___).

For example, Greenfoot.getRandomNumber(4) will return a random number between 0 and 3.

Using the code above and the methods turn() and move(), get your PrimaryConsumer objects to move and turn randomly.

# If – Else Statements

If statements are used in programming in order to make decisions. If something is true you can allow a piece of code to run, if it is not true then it will not run.

General example:

If you are wearing a jumper, raise your hand.

We can extend this to include an 'Else' as well:

If you are wearing a jumper, raise your hand. Else, stand up.

# If Statement in Greenfoot

If you are wearing a jumper, raise your hand.

In Greenfoot:

```
if(clothes == "Jumper")
{
    raiseYourHand();
}
```

# Checking for Collisions Between Objects in the World

When the **PrimaryConsumer** objects, in this case the '**mice**', touch the **Producer** objects, in this case the '**grass**', we want to remove the '**grass**' objects from the world.

To do this, combine an **if statement** with the **isTouching()** method and the using the **removeTouching()** method remove the object.

If you have not looked at the Greenfoot documentation on **isTouching()** and **removeTouching()** do it now.

# .class

To tell Greenfoot that we want to check for an Actor object belonging to a subclass we use the **.class** method.

For example, to check if an object belongs to the **Producer** subclass we would write '**Producer.class**'. For **PrimaryConsumer** what would it be?

In pairs discuss how do you think you can combine isTouching()' and a **.class** method?

# .class

- isTouching(Producer.class)
- isTouching(PrimaryConsumer.class)

Extension: How would you use the removeTouching(); method to remove a **PrimaryConsumer** object?

How would you use the removeTouching(); method to remove a **Producer** object?

# Activity: Combining If Statements and the Methods

Using if statement implement the program so that when the **PrimaryConsumer** objects, the '**mice**', touches the **Producer** objects in this case the '**grass**', we remove the '**grass**' objects from the world.

Note: you want to write this code inside the **PrimaryConsumer** subclass editor.

# Current Program State

```
/**
 * Write a description of class PrimaryConsumer here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class PrimaryConsumer extends Actor
{
    /**
     * Act - do whatever the PrimaryConsumer wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
        turn(Greenfoot.getRandomNumber(4)*90);
        move(Greenfoot.getRandomNumber(2));
        if (isTouching(Producer.class))
        {
            removeTouching(Producer.class);
        }
    }
}
```

# Activity: Implement the SecondaryConsumer Subclass

Using all that we have learned extend your game to do the following:

- Create a third **Actor** subclass called **SecondaryConsumer**.
- Set image to a **snake**.
- Create an object of the **SecondaryConsumer** and get it to display in your '**Habitat**'.
- The **SecondaryConsumer** should also move randomly like the **PrimaryConsumer** but when the it touches a **PrimaryConsumer** object it removes it from the world i.e. if the **snake** touches the **mouse**, the mouse disappears.

# Game Demo

# Extension: Object Interaction

Now alter the game objects to interact with each another. For example, if two primary consumers touch, get them to produce an offspring.

To do this, in your 'Habitat' add the following code:

```
public void createPrimaryConsumer()
{
    PrimaryConsumer newMouse1 = new PrimaryConsumer();
    addObject(newMouse1, getRandomNumber(3,5), getRandomNumber(3,5));
}
```

This will create a method in our Habitat subclass that will allow us to create new PrimaryConsumer objects.

# Extension: Object Interaction

Using what we have learnt so far, in the PrimaryConsumer, if two primary consumers touch, we can use the createPrimaryConsumer() method we just added into the Habitat. to create another primaryConsumer and add it to our world.

To use this method, we can use the following lines of code:

```
Habitat habitat = (Habitat)getWorld();
habitat.createPrimaryConsumer();
```

# Extension: Object Interaction

```
import greenfoot.*;

/**
 * Write a description of class PrimaryConsumer here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class PrimaryConsumer extends Actor
{
    /**
     * Act - do whatever the PrimaryConsumer wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */

    private int REPRODUCE_DELAY = 7;
    private int reproduceTimer = 0;

    public void act()
    {
        // Add your action code here
        turn(Greenfoot.getRandomNumber(4*90));
        move(Greenfoot.getRandomNumber(2));
        if (isTouching(Producer.class))
        {
            removeTouching(Producer.class);
        }

        // Delay
        if (reproduceTimer > REPRODUCE_DELAY && isTouching(PrimaryConsumer.class))
        {
            Habitat habitat = (Habitat)getWorld();
            habitat.createPrimaryConsumer();
            reproduceTimer = 0;
        }

        reproduceTimer = reproduceTimer + 1;
    }
}
```

# Extension: Object Interaction

The highlighted code in the previous slide you do not need to understand fully. Just know that when two PrimaryConsumer objects touch they will call a method called 'createPrimaryConsumer()' and will automatically make and add a new PrimaryConsumer object into the world.

The variables 'REPRODUCE_DELAY' and 'reproduceTimer' delays the amount of offsprings made at one time.

# Game Demo With Object Interaction

# More Ideas: Big Game Demo