For Dad,
Who always taught me to question how things work
but wouldn’t understand a word of this book.

Contents

About the author

Jez Whitworth: The warranty voiding inventortainer, slowly taking apart the world whilst encouraging creativity in the people he encounters.

With qualifications in art, design and computing and with a background in software development, his IT skills have helped him create an innovative and exciting approach to deliver creative and fun lessons for all ages and abilities. His website is packed full of creative ideas and project build that he hopes will inspire people to get making: www.rustyrocket.co.uk

Jez set up **First Coding (**www.firstcoding.co.uk) to offer comprehensive coding lessons for children. Covering the important concepts of programming in a fun and creative environment.

In the evenings he can be found in his shed (of which he was a Shed of the Year finalist in 2006) and goes by the name of **@RocketJez** on Twitter (www.twitter.com/RocketJez) whilst also writing about his many backyard engineering adventures. He is currently attempting to break a world record, building the largest working Gameboy. He dislikes writing about himself in the third person.

Jez has also spoken about the benefits of creative coding at many events. You can view his latest TEDx talk here: https://www.ted.com/talks/jez_whitworth_creative_coding

Top 5 Coding Tips

Complete your projects – This is the hardest of all the points in this series to stick to. It is really important to see a project idea through to the end. Not only will you feel satisfied that you got an idea across the finishing line but you will also have a better understanding of the process of developing an idea and following it all the way to completion.

Never be afraid to reach out for help – There is a wealth of information online to help you if you become stuck, but sometimes it can be a little overwhelming. First Coding is always contactable to help whether it's via email (hello@firstcoding.co.uk) or via social media, we're here to help troubleshoot any coding problems you may have. If you need an answer to a question to help you move on with your project, then please just drop us a line and ask it!

Enjoy yourself – Learning to code should be fun as well as challenging. Once you've mastered the basics, try to develop one of your own ideas for a game or an app. If you're stuck for inspiration, try and write something that already exists like a game of Space Invaders or Tetris. This will help you develop your skills of breaking down a task and planning the steps required to achieve your goal.

Always start by learning the basics – You've heard the term "Learn to walk before you can run", this can be said for learning to code too. Start off learning the basics, variables, loops, functions etc, writing little programs to try out and discover how these elements work and interact with one another. You will then have a good foundation in which to build on. Take your time.

Don't be afraid to make mistakes – I see this a great deal with pupils, there is always a worry that your code may break if you try to incorporate a new idea into your program. My advice would be to always back up your work then should things not behave as you intended, you can always revert the code back to how it was. Always remain curious with your code, try out new ideas and concepts, it's the only way to learn and develop.

Game Design Using Scratch

I think it's important to not exclude the younger generation when it comes to game development. As a teacher I am always amazed at the thoughts and ideas that come out of the heads of children and it's this level of creativity that so many independent game developers crave. Sure, we can master the countless software solutions out there that will allow us to produce truly incredible things but without an idea, without that spark of inspiration we are never going to fly.

Scratch (https://scratch.mit.edu/) is a popular platform used in schools to teach the basic fundamentals of coding. Scratch projects can be optimised for mobile devices as well as incorporated into websites, instantly opening up a wealth of possibilities and proof that Scratch can hold its own in the creative world of game development.

We'll be looking at how we plan and build a simple platform game in Scratch. Using popular game design techniques to improve the user experience as well as exploring ways to showcase and even monitise your Scratch creations.

For those who wish to skip the process of making the Taco Cat game and visit the game's page to read its feedback and to play it yourself, then click here: https://rustyrocket.itch.io/taco-cat

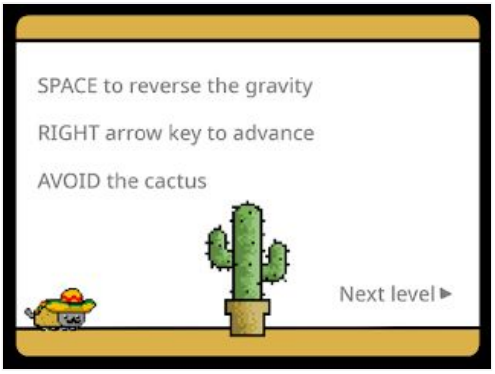
Character Design

There is a lot of truth behind the saying, "Keep It Simple Stupid" and when developing 2d pixelated games I like to create characters that contain bold colours and subtle animations. The Taco Cat character was based on a gif of the same name, drawn using the free Pixelart (https://www.pixilart.com/draw) online and animated to give the appearance of walking.

Controls

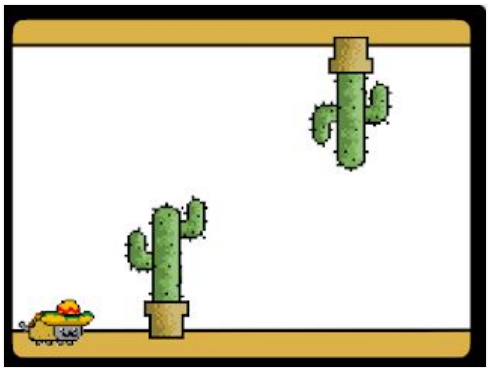
I wanted to add something a little different to the game, so not only can you move the character through the level from left to right, you can toggle the gravity to vertically flip Taco Cat adding a new dimension of play to the level.

With any game, it's important that the player knows the rules and controls from the very beginning and Taco Cat is no exception - Simple controls for a simple game. I've found that with the majority of my games it's of huge benefit if the game opens with the controls and rules of the game. In this case the controls are displayed in the background of the first introductory level.

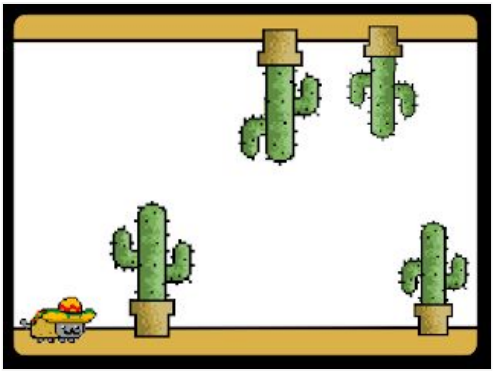


Level Design

This is an interesting one and a topic that has countless books written about it. It's important that the player understands the rules of the game by completing the opening levels and ensuring that the player levels up quickly and easily at the start.



The levels in Taco Cat start with the easiest possible obstacle of all, a single cactus before increasing in number and complexity. It's important not to make the game too difficult too soon, the skill is finding that sweet spot that makes the game not too difficult but still poses a challenge.



Hopefully I've managed to highlight that if you have a great idea there will always be a tool out there to help you build on it. Just look at Taco Cat, built using Scratch and entered into a competition where it was pitched against some truly fantastic designers and their games. Never underestimate the impact that Scratch can make!

Make Money from your Scratch Games

It’s always nice to showcase a well made Scratch project and it’s even nicer to receive a little reward for all your hard work. This chapter explores two free methods of how you can adopt to display your Scratch games whilst accepting donations and revenue from ads.

Method One

Some of my students are giving their pocket money a boost by building an online arcade with ads to showcase their Scratch games. Whether it's desktop, tablet or smartphone, anyone can play. You can play the games in the virtual arcade here: https://therocketarcade.000webhostapp.com Here's how it was made…

There are a number of ways to embed Scratch projects into HTML pages. One service I've come across that works really well is HTMLifier (https://sheeptester.github.io/htmlifier/) and this will allow your Scratch projects to be played full screen in the browser.

An arcade page is then built that contains an iframe to display the HTMLifier file. A free web hosting site called 000 Web Host (000webhostapp.com) was used.

We used a couple of different ad services and you can see the two different banners in the screenshot provided. The first banner located at the top is generated using the eBay Partner network: https://partnernetwork.ebay.co.uk/ and is free to subscribe to as long as you have an eBay account.

The second banner originates from the Amazon Affiliates Program: https://affiliate-program.amazon.co.uk/ and again it's free to sign up with an existing Amazon account.

Both services provide you with options to cherry pick products to display in the banners but also offer an automatic selection if you wish. Both offer a certain percentage of any sales made as a result of the banners being clicked on. Personally I prefer the eBay service as it allows you to select bespoke items that could potentially attract more clicks.

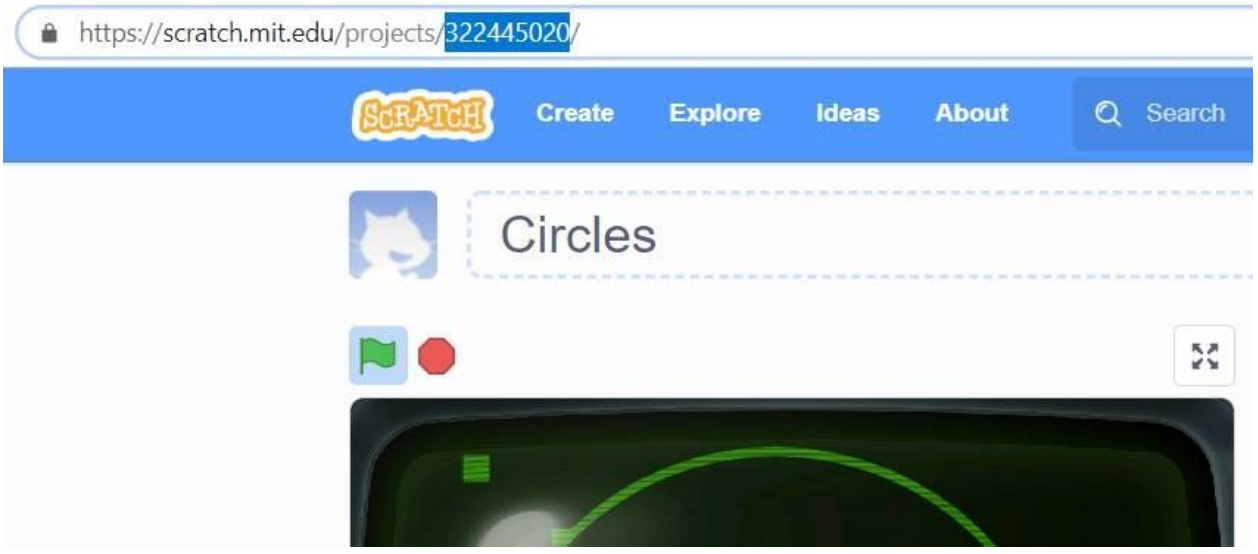
Method Two

In addition to the Scratch gallery of shared projects, there are also other platforms that will allow you to showcase your work and allow you to sell or accept donations in recognition of your hard work.

One such site is called **Itch.io** (https://itch.io/) that nicely allows you to embed your Scratch game to make it playable within their site but they also make it possible for you to ask for money for people to play your game. The Scratch license will permit this commercial use.

Preparing the Game

Once you have completed your Scratch game and have tested it to ensure it is working as intended, the next step is to share your project, making sure you complete all the details on its project page. Make a note of the project's reference number that can be found in the address bar.



In order to submit the game to a platform that allows payment, we first need to create a simple webpage with your Scratch project embedded using HTML. This is easily achieved using the snippet of code below. Be sure to replace the reference number of 1234567890 with your game's reference number you noted down from the previous step.

```
<center><div style="overflow: hidden; height: 480px; width: 640px;" >
```

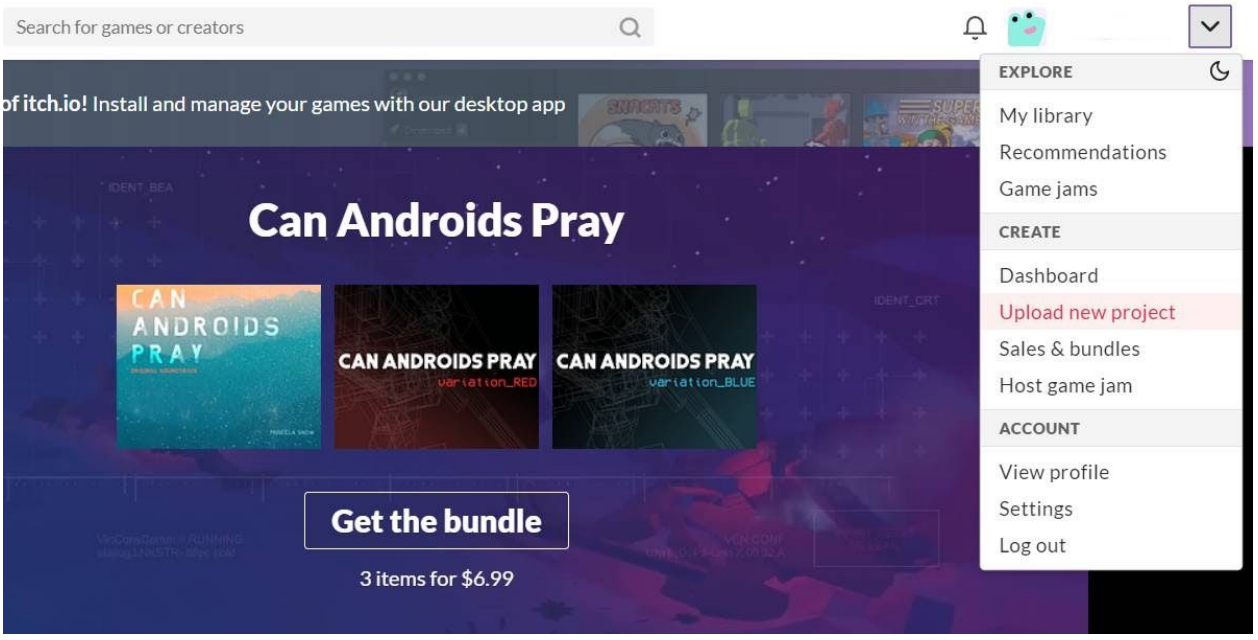
<iframe style="margin-top:-50px; margin-left:-11px;" scrolling="no" allowtransparency="false" width="658" height="536" bgcolor=#220000 src="https://scratch.mit.edu/projects/1234567890/embed/" allowfullscreen></iframe>

All you need is Notepad to complete this step and once done save as index.html.

Uploading to Itch.io

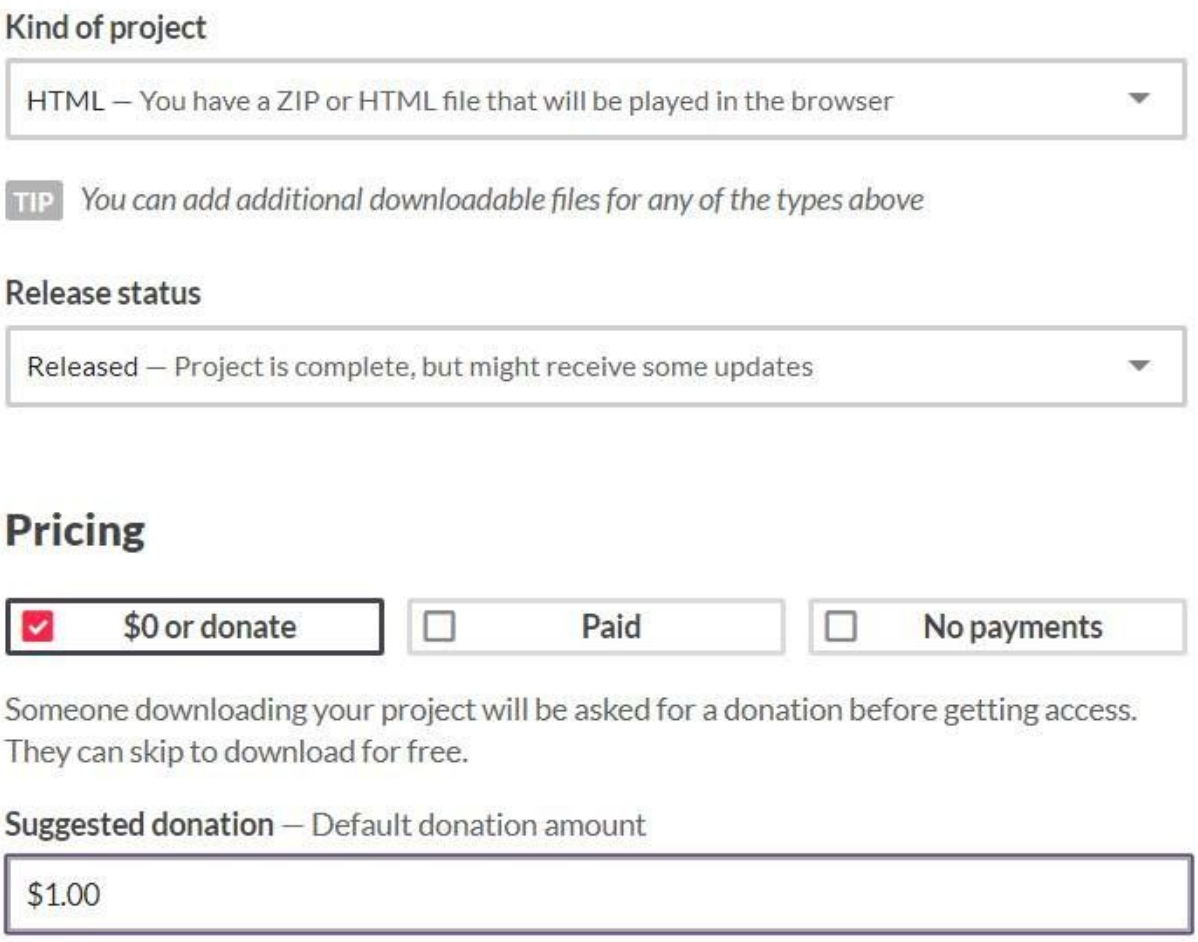
Itch.io is a popular platform to discover, download and distribute indie games.

In order to showcase your game in the Itch.io platform you will first need to register with them for a free account.



When completing the online form to submit your game you will need to select HTML from the **Kind of Project** dropdown field.

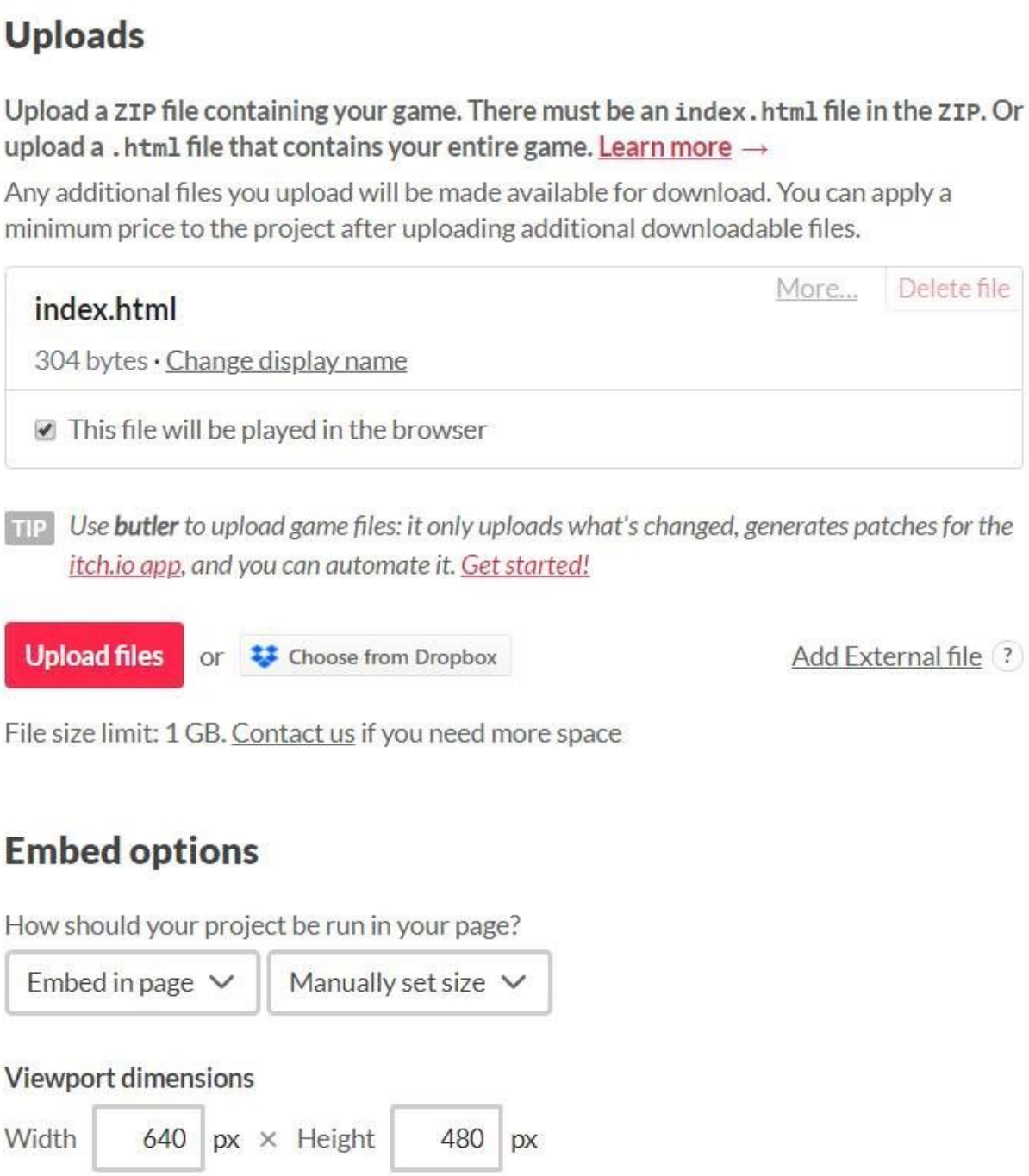
The **Pricing** section is further down the form and this is where you can set a price for your game. As most games on Itch.io are free, you would be better off selecting the **$0 or Donate** option in this case. You can then set a suggested donation amount.



Uploading the Game

When you come to the **Upload** section of the form, you will need to ensure that the **This file will be played in the browser** option box is ticked after the index.html file has been uploaded.

It is also recommended that the viewport dimensions be amended to match that of the dimensions stated earlier in your HTML file. Amend them to a width of 640px and a height of 480px.



The rest of the online form is simple to complete. Make sure you give a great deal of thought to the description of your game. Notice that you can format the writing too to make it eye catching and really stand out.

Finally, don't forget to make your game's page public so everyone can view it. This setting can be found under the **Visibility & Access** section of the form.

Game Design Using Python

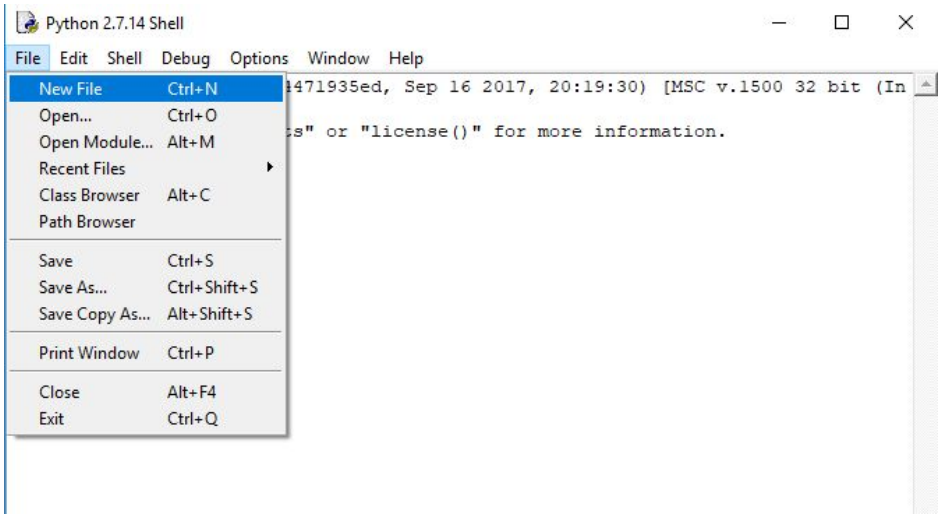
The natural progression after Scratch is Python. Python is a friendly text-based programming language that actually shares a number of similarities with Scratch and because of this, it's easy to learn.

Installation

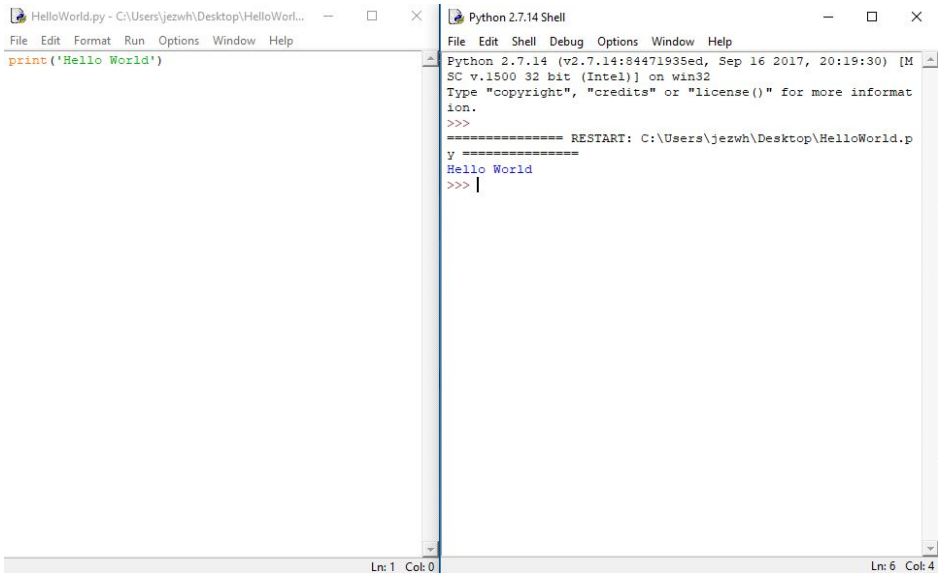
If you would like to install Python on your computer, you can do this by simply visiting the official Python website (https://www.python.org) and click on the Download option. Follow the simple instructions on how to install it on either your Windows machine or Mac.

Once installed, open up the Python program. You will be greeted by the Python IDLE (Integrated Development Environment) which is an area to allow you to write and run your Python code.

Selecting "New File" from the File menu within the IDLE will open up a new window. This is where you will write your Python code.

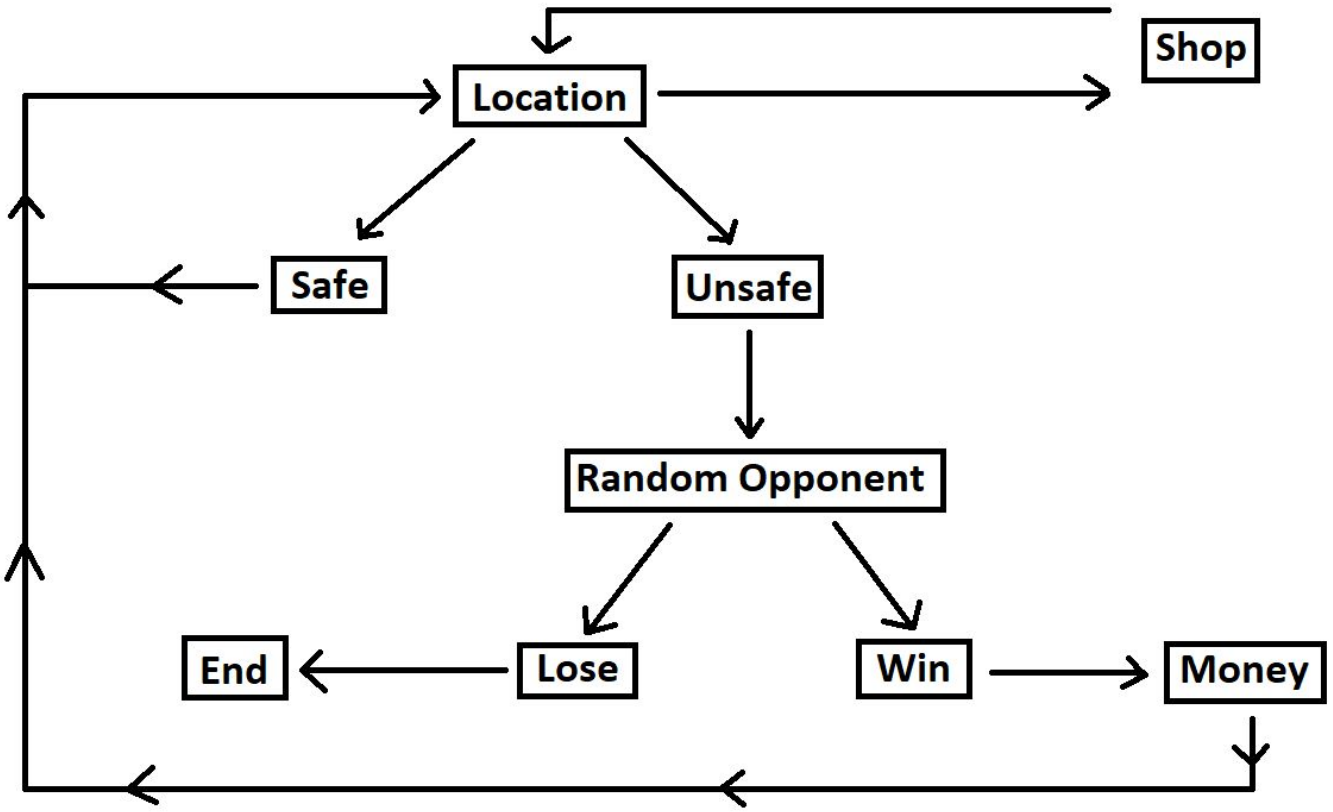


Always remember to save you work as you go along. Once you are ready to test you code, press F5 to run it in the IDLE.



We'll look at how we can use some of the basic principles of programming to produce a challenging and engaging Choose Your Own Adventure game including: randomisation, lists, loops and decision making. You can see a live demo of a game built using this framework here: http://numberstation.co.uk/

As always, the first thing to do is start to plan it out on paper. Here is the basic flow of the game:



The game will be written in a loop, meaning that an infinite world can easily be generated, allowing any lucky player to continue through the game for any length of time. There are options to choose your character, affecting what money and items you start with, as well as visiting shops at certain points in the game to purchase additional items to improve your chances when fighting opponents.

Fighting

When we face an opponent during the game, we must fight them. Fighting is done by dice rolling. The person with the highest roll wins the round and health points are deducted from the loser. The fight continues until one person's health has fully been depleted and the game ends. If a player wins a fight, they are rewarded with a small amount of money before play continues with their current health value.

The Shop

There is a shop available in the game that only becomes available when a player is visiting a particular location. It is here a player can spend money they've collected to purchase additional weaponry. Checks are carried out to ensure there are sufficient funds available before any purchase takes place. Weapons are designed to give the player an advantage during a fight, increasing their chances of achieving a higher dice roll.

This is a basic framework designed to give people a good starting point for when they design and make their own games.

```
import random

print("")
print("********")
print("Adventure Game Workshop")
print("********")

money = 10
OurHealth = 50
Advantage = 0
Inventory = []

Name = input("Enter your character's name: ")
print("")
print("Choose your character type:")
print("Option 1: Warrior")
print("Option 2: Wizard")
print("Option 3: Tank")
print("")
OptionChoice = input("Make your choice: ")
print("")

if OptionChoice == "1":
  print("You have selected the Warrior.")
  Inventory.append(" Spear")
  print("You have a spear with an advantage of 1.")
  Advantage = Advantage + 1

elif OptionChoice == "2":
  print("You have chosen the Wizard.")
  OurHealth = OurHealth + 10
  print("You start the game with an additional 10 health points.")

elif OptionChoice == "3":
  print("You have chosen the Tank.")
  OurHealth = OurHealth + 20
  print("You start the game with an additional 20 health points.")
  Advantage = Advantage - 1

while True:

  EnemyHealth = 50

  print("")
  location = ["wood", "street", "castle"]
  enemy = ["fighting robot", "green goblin", "mad scientist"]
  LocationChoice = random.choice(location)
  print("You are in a "+LocationChoice)
  print("Health: "+str(OurHealth))
  print("Money: "+str(money))
  print("Inventory:")
  print(Inventory)

  print("There are two routes to take.")

  choice = input("Do you go left or right? Enter L or R.")

  if choice == "s":
```

```
  print("Welcome to the shop")
  print("")
  print("Option 1: Sword 100")
  print("Option 2: Dagger 50")
  print("Option 4: Exit the shop")
  print("")
  ShopChoice = input("Please make your selection: ")
  if ShopChoice == "1":
    if money > 100:
      print("You've bought the sword")
      Advantage = Advantage + 2
      money = money-100
      Inventory.append(" Sword")
    else:
      print("You don't have enough money and are thrown out of the shop")
  elif ShopChoice == "2":
    if money > 50:
      print("You've bought the dagger")
      Advantage = Advantage + 1
      money = money-50
      Inventory.append(" Dagger")
    else:
            print("You don't have enough money and are thrown out of the shop")
  elif ShopChoice == "3":
      print("You exit the shop and continue on your way")
  else:
          print("Not a valid option. You exit the shop")

unsafe = random.randint(1,2)

if unsafe == 1:

  print("")
  print("You have chosen the safe route. You continue on your way.")
else:
  print("")
  print("You have chosen the unsafe path")

  print("You must battle a "+random.choice(enemy))

  while OurHealth > 0 and EnemyHealth > 0:
    pause = input("Press ENTER to fight")
    OurRoll = random.randint(1,6 + Advantage)
    EnemyRoll = random.randint(1,6)

    print("You rolled: "+str(OurRoll))
    print("They rolled:"+str(EnemyRoll))

    if OurRoll > EnemyRoll:
      print("You won that round")
      EnemyHealth = EnemyHealth - OurRoll

    if EnemyRoll > OurRoll:
      print("You lost that round")
      OurHealth = OurHealth - EnemyRoll

    if EnemyRoll == OurRoll:
      print("You blocked the attack")
```

```
    print("Your Health: "+str(OurHealth))
    print("Enemy Health: "+str(EnemyHealth))

  if EnemyHealth <= 0:
    print("You have won the fight.")
    MoneyWon = random.randint(10,20)
    print("You have won "+str(MoneyWon)+" gold coins.")
    money = money + MoneyWon
    print("Gold coins collected = "+str(money))

  if OurHealth <=0:
    print("You have lost the fight. Game over")
    exit()
```

Errors

There will be times when your code doesn't run properly and results in an error being displayed in the IDLE. The key is not to panic as most of the time errors will appear worse than the actual problem and all problems can be corrected.

More errors will appear in the SHELL window, the window that will display any outputs from your code. If an error displays "XXXXX is not defined" then it's likely that there's an incorrect variable. There might be a line number displayed too that will direct you to where the mistake is located.

If a "syntax error" pop-up appears when you attempt to run your code there is a good chance that there's a spelling or typing mistake somewhere in your code.

If the error reads "Unexpected indent" then there is most likely additional spacing that is causing a line of code to become indented unnecessarily.

Common issues

Try not to mix up single and double quotes. If you open with a double quote, you must close with a double quote.

Python is case sensitive so be sure to use the correct case in all cases. Print must be written as lowercase **print** and all variables that have been set up with capital letters must be referenced correctly throughout your code.

Be sure to understand how the different brackets are used so as not to get them mixed up. (), [] and {} all have different uses.

Troubleshooting Checklist

☐ If you are copying a sample of code, have you copied it correctly?
☐ Are there any spelling mistakes?
☐ Do the starting quotations and the finishing quotations match in all cases?
☐ Are there extra spaces at the start of your lines that are indenting the code incorrectly?
☐ Have you asked someone else to check your code?

Distributing Python Games

In order to share your Python games, it'll need to be hosted and run in the cloud as this will make it easier for users to play your game as there isn't the need to download and install the Python IDLE. Whilst there are many websites out there that will allow you to run code via a web browser, the most convenient site that's also free to register and use is called repl.it (https://repl.it/).

Each Python project (called Repls) has its own unique URL allowing anyone with the project's web address to instantly play your game in their browser. Further steps can be taken to build a website for your project, embedding the game for all to play by following the steps detailed in the **Making Money from your Scratch Games** section. A good example of this is the Number Station game that has been embedded with a simple HTML framework built around it. You can view it here: http://numberstation.co.uk/

The Repl project is embedded into the Number Station website using the tag <iframe> and a copy of the HTML code, complete with links to other sites and ad placement is below:

```
<!DOCTYPE html>
<html>
<head>

<style>

body {background-color: black;}
h1   {color: blue;}
p    {color: red;}

</style>
</head>
<body>
<center>
<img src="NumberStationIcon.png"><p>

<a href="https://en.wikipedia.org/wiki/Numbers_station" target="new"><img src="learn.png"></a>
<a href="http://priyom.org/" target="new"><img src="listen.png"></a>
<a href="https://www.instructables.com/id/Broadcast-Your-Own-Number-Station-on-the-Raspberry/" target="new"><img
src="build.png"></a><p>

<iframe src="https://NumberStation.rustyrocket.repl.run" frameborder="0" width=1000 height="500"
style="overflow:hidden;" seamless="seamless" scrolling="no"></iframe>
<p>
</center>
</body>
</html>
```

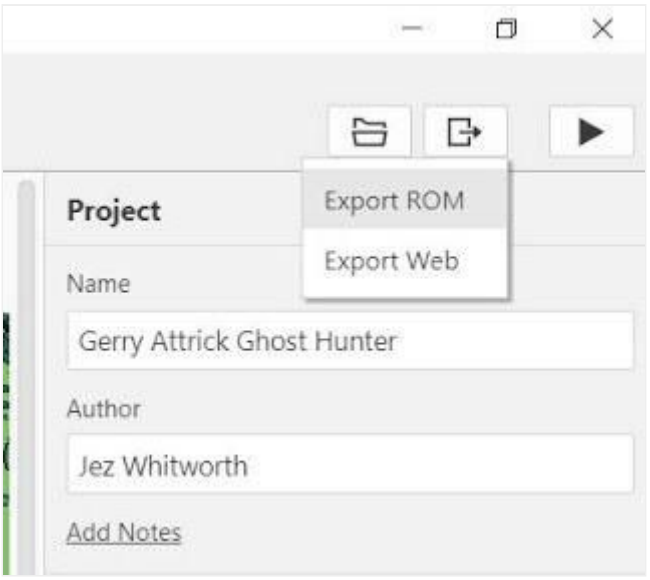
Game Design for the Gameboy

It's always great to be able to build games that run on original hardware. Sure, the same task can be achieved more easily through various emulation platforms such as RetroPie for the Raspberry Pi but there's nothing quite like the feeling of coding something that plays on the console it was originally intended for. For years I thought that coding for old, obsolete consoles was very much a dark art until I found a really nice drag and drop game creator called GB Studio (https://www.gbstudio.dev/) that popped up last year and has been well met within the community.

This is all backed up with some documentation and although relatively young, support for GB Studio is growing, especially on YouTube and I would recommend checking out Pixel Pete's channel (https://www.youtube.com/user/MilkoDaily/videos) for some really useful tutorials on how to get started.

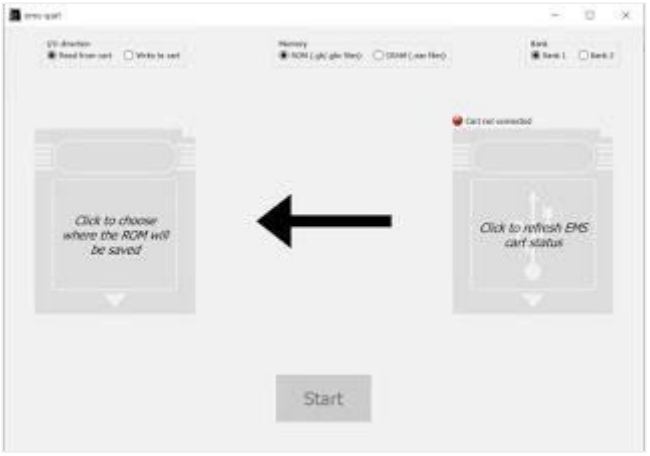
Pixel Pete's channel is also proving a good resource for people who are wanting to develop their pixel art skills further.

What I really like about GB Studio is the ability to not only export as a ROM file but also as a web page with your game embedded within it. This is such a nice feature and one that I would like to explore further when I have the time.



Once you have exported your ROM, you're going to need need to transfer it onto a flash cart of some kind. The one I went for was being sold by Retro Towers (https://www.retrotowers.co.uk/gb-gameboy-usb-smart-card-64m).

Now for some reason I had trouble using the software that came with the cartridge, I found it too clunky. However, I managed to find a suitable alternative called ems-qart and you can download the latest version from here: https://github.com/rbino/ems-qart/releases. This made transferring files so much easier and I recommend that you use this software from the start, to avoid the same frustrations that I encountered.



Here is the game running on my original Gameboy.

It even runs and looks slightly nicer due to the additional colours on my Gameboy Advanced SP.

I did have trouble running it on my much favoured Gameboy Pocket, as the game failed to load and kept rebooting itself. After a quick search in the forums, apparently the issue is due to the power that the flash cart requires in order to successfully run and the Gameboy Pocket just can't deliver.

Game Jams

I would definitely recommend making and entering a game in one of the many game jams that take place on a weekly basis. They're a great way to test your working process and they're such a great way of getting friendly and constructive feedback from the community. Here are a few game jam tips to get you started.

- The most important point is to always participate. This may sound silly but if nobody took part in games jams there would be no game jams. Even with little experience, people can benefit from the game jam experience on so many levels, so just get stuck in.
- It’s a mistake made by many, people start working on a game before the theme of the game jam is announced and they then try to shoehorn their game idea to fit the theme. Always wait for the theme of the jam before brainstorming ideas.
- Always keep ideas simple as quality will always triumph over quantity. People will always favour playing a really good short game over a lengthy one. Keeping ideas short and simple will also make it easier to experiment with new concepts and at the end of the day, that is the main point of game jams.
- Create multiple ideas at the pen and paper planning stage, then condense these down to one. Take time to do this, giving each idea careful consideration as this will pay dividends further down the line. Expect this final idea to keep evolving over time as development progresses.
- Manage your time and write your development time on your calendar, plan your work in two hour blocks and put everything in your calendar. Remember to always factor in time for polishing and bug fixing as these stages are often overlooked. You don’t have to work the full allotted time of the jam.
- Never overlook the impact that sound has in a game. If your skills lie elsewhere, hunt out all the free resources that will help you out when source sound effects and backing tracks.
- Write a good, clear explanation for your game and pick a great title of it.
- Most Importantly, have fun!

A good place to start when looking for a game jam to enter would be on the itch.io website (https://itch.io/jams).

Helpful Tips to Becoming an Indie Games Developer

You’re passionate about games and have a creative flare for game design, so taking the bold step into game development and the industry seems like a great way of expressing your creativity and innovation, right? The games industry is an extremely competitive business but getting started the right way will give you the best fighting chance.

Developing games takes on many forms but it mainly comes down to problem solving. Being able to break down a problem into easy to manage parts to find a solution is a great skill to have when making games and one that is hugely transferrable into many other areas. To be able to do this however you need to be a logic-thinking tinkerer who is able to take apart a digital system, analyse it and put it together again.

So is a qualification needed for game development? Absolutely not. Speaking for myself, my game development days started with a laptop and a handful of good, fun ideas scribbled on the pages of an old notebook. If you do choose computing at school then you’ll most likely study Javascript and Python which isn’t a bad thing. You may also go down the root of studying HTML and CSS which aren’t really programming languages, more methods of scripting but again this isn’t a bad thing at all. It’s a really good thing that schools offer these but as they start off basic, they’ll probably finish basic too. Even if you have worked hard for a qualification at school, I would still recommend that you learn more in your own time, repeating what you’ve learnt but using it as a foundation to learn more advanced techniques. At school or college, you're just going to be taught the general skills and usage and any work or projects you do will also be fairly general. Just stopping at the basic level will prevent you from standing out when applying for jobs or go freelance.

Just to repeat, a degree qualification is NOT necessary. Developers want to see skill, they don't care about a degree. Don’t get me wrong, it doesn't hurt at all to have one, but a degree means nothing if you can't back it up, and if someone else has a portfolio that contains far superior work without a degree, then they're going to get hired over you. A degree simply shows general competency, not that you are skilled in your field.

The best tip I can give you is to work, work, work. Never stop. The best programmers are the ones with an inquisitive mind, always exploring, learning new skills and techniques on their own. I know from first hand that this can be extremely difficult at times, especially when you're the only one to hold yourself accountable.

Back to Basics

Most people quit before they even finish making their first game because the experience wasn't what they expected. Let’s be honest, you're not going to make a game that rivals those hugely popular games, so it's important to think reasonably. Think of your first game as a learning exercise and aim to build something that you can actually play. Be proud of the single level game that involves a couple of obstacles that took you weeks - Be proud of this achievement.

Start planning your first game around what skills you have. Existing skills can serve as a good foundation that can always be built upon through the use of tutorials and online videos. If you're an artist but not a coder, maybe focus more on your artistic skills with a little coding tuition. Don't be afraid of the actual coding side of things though, if you have a well planned out game you'll be surprised at how little coding is required.

If your skills lie more with the coding side of things rather than the artistic, then check out the wealth of asset stores where you can pick up great artwork for a very small fee. Don’t forget also that you'll find that many people are out there on the forums and sites who will gladly offer help and advice if you become stuck. I guess it all stems back to having an explorer attitude, never be afraid to ask questions.

Never give up either, motivation is the key to success. Sometimes life gets in the way of projects and that's fine Once again, being able to break problems down into manageable chunks will help you achieve your overall goals, though it may take you longer to get there, so don't get impatient.

Planning

For your first game, plan a project that won't take longer than a month. If it takes longer than this however, don't worry. Surprises always have a habit of appearing when you least expect them but if your project ends up passing the three month milestone then maybe your expectations are a little too high. Aim to build something that works quickly as it's the experience that is more important than the end result at this stage.

When you have your plan you are in a great position to set yourself key goals. Once you've laid them out, take those goals and break them down further into mini-goals. Having a granular tasklist will be easier to accomplish. By keeping things small, you'll be able to make games that let you experiment with exciting game concepts without increasing the risk of failure or abandonment.

A key tip is to review your game at least once a week, even if it's for just for half an hour a week. Weeks can pass so quickly, often without realising and without regular reviews how will you know that you're achieving your goals?

Matchmaking

Matchmaking is a method used in games to allow players to be pitched against fair opponents, where both sides have a reasonable chance of winning. It is important as it maintains a good balance in games, ensuring that the games aren't too easy or too hard. Opponent matchmaking is usually achieved through algorithms that will utilise the use of variables to keep track of character statistics, useful when carrying out opponent selection.

The methods used to achieve fair matchmaking is a topic still very much debated today amongst game developers as there is no perfect method, but luckily the approaches used fall into two camps:

1. **Tournament**, where character variables are used to select opponents in single, self contained events. This is probably the oldest and simplest method of matchmaking and is a great way of finding an absolute winner to your game. The simplest tournament structure would be where all players get to play each other once and adopting a simple points structure would then allow you to slim down the player list before progression to the next round occurs.

   One issue with this approach to matchmaking is that there is a predetermined number of matches set.

2. **Historic,** this is currently the most popular way of matchmaking where opponents are matched based on players' previous performances. It uses a similar method used to grade chess players called Elo, and was a popular way of predicting chess match outcomes.

   The nice thing about Elo is that the winning player takes points from the losing player based on how far apart they were matched. This would mean a low rated player that beats a higher rated opponent would gain more points if they won a match against another low points opponent.

When you next play the games you enjoy, take a moment to work out how the game is matchmaking you to the opponents.

MVP

Refers to the smallest thing that could be released. If it's 80% complete then ship it. This should be your goal and is a great way of discovering the key parts of your games.

Find the absolute minimum set of features that doesn't compromise the core development. If you can cut a feature and still ship your game then it's probably not a core feature.

Many of the platform games I've produced can be distilled down to walking, jumping, obstacle avoidance and a level reset if the player dies. When building your first game, you can usually strip all its features to get down to its foundations. Too much content ini yout MVP will just add clutter and will distract from the gameplay. You don't want to give your player a negative experience.

Genres ranked in order of difficulty to produce a MVP:

Racing game
Top down shooter
2d platformer
Combat game
Action adventure

These games can be distilled down into simple blocks with functioning mechanics such as acceleration and collision detection for the racing game genre. Remember most of the popular games are simple block-based games that consist of nothing but moving shapes.

Launching

Once you've made your game, what are you going to do with it? The first thing is to let people know about it. Understand your own message, why should people play your game? Once you know what's special, let's look at how you can get that message out.

Much can be achieved with little or no marketing budget. Make a trailer using the many free software packages and cap it under 3 minutes. Make a website too, I would recommend starting with a free website first. Then get sociable on social media, start following other people in the games industry and engage with other indie developers similar to yourself. Try to get a mention from other developers as well as indie dev websites, especially ones that specialise in your chosen genre.

If you don’t currently have a website there are many free solutions out there that will help you start to build a presence online. I would recommend starting off with one of these free solutions before considering buying your own domain and hosting.

Wordpress
URL: https://wordpress.com

Wordpress is one of the largest free solutions out there that allows you to build dynamic websites that look nice. It is open source and doesn’t restrict functionality through trial versions. There is a huge and helpful community which is the reason behind it’s popularity and robustness. They have a phone app for both iOS and Android to allow you to add content on the go.

Blogger
URL: https://www.blogger.com

A popular alternative to Wordpress is Blogger which is Google's free blogging platform. The interface is simpler to use in comparison to Wordpress but it still manages to retain all the major features, including a nice selection of free templates that will suit your needs. As it is owned by Google, I find the indexing of Blogger sites slightly quicker than sites built on other platforms. Blogger also has a phone app for both iOS and Android to allow you to manage and update your site on the go.

As your launch date gets closer, have a basic plan in place to generate interest leading up to the big day. Reddit is a good starting point.

Where are you going to launch your game? Digital distributors make their money from the distribution of indie games, whether it's 100 or 1 million units so don't be afraid to reach out to the big hitters such as Steam but you will have to put in a huge amount of effort with publicity and marketing. Mobile stores actually have a wider audience when compared to the likes of Steam and have fewer requirements in comparison.

Even Facebook can be viewed as a distribution possibility. Porting a game across a variety of platforms has never been easier, so when you're thinking of where to distribute your game - the answer should be everywhere.

Wherever you are distributing your game, chances are your store listing could be improved. Review the content regularly to ensure that your keywords attract the biggest audience. Monitor your store statistics weekly and keep a record of the number of store listing views and purchases. See if making changes to your listing content and keywords affects your visitor stats.

Getting Noticed

Attending events are the key though chances are you're not going to be able to afford a stand or booth at these events - Showcasing on the go is going to be the next best approach when it comes to getting your game in front of as many people as possible.

There are a few things to consider when adopting this tactic approaching people on the fly to play your game. Firstly, think about what you want to achieve from this exercise. Most of the time you need people to play your game for the feedback. If you have areas of your game that you're not sure about then why not bring different versions of your game for players to test and provide comment? Remember though that players want to be entertained so make sure your builds are the best they can be at the time but remember if you're presenting different build versions, try and avoid confusing the player - Make each build as self-explanatory as possible.

When you meet people, especially the press make sure you have something to hand out so that they can remember you by - Maybe make up a little press kit to leave them with? Also try not to forget journalists, bloggers and influencers are all interested in stories, so make sure you have a good tale to tell about your development journey and the hurdles you overcame.

Most importantly, make friends - get people interested in you and your game. You're not only selling your game but all selling yourself, so be friend and polite at all times. Make an effort to keep in touch with as many contacts you make after the event.

Preparation, Preparation, Preparation

Make sure you know what is fun and unique about your game so that you're not caught off guard when speaking with people.

Make sure you have your handouts ready. As mentioned previously, giving something out is a great way of getting people to remember you after the event. I've found that to ensure a greater rate of success, avoid flyers. Flyers do not work as they are often too text based and cheap looking, instead focus more on the artwork rather than the text. Make something that's more high quality than the usual flyer: postcards, small comics, that sort of thing - Really stand out from the crowd by making it look fantastic and something that you yourself would enjoy receiving.

The final preparation point would be to ensure your game runs on a mobile device, even if it's not intended to be a mobile game. Even on a tablet, this will allow you to easily pass it around and showcase.

Approach

It may sound odd but learn and get confident talking to strangers at events. It's not as hard as it sounds, even for an introvert like myself, they are all people like yourself united by a common interest. Seriously though, the key is to be friendly, show an enthusiasm for your game and be excited for the games made by others in the room. If other games you're talking about are similar to yours then it all helps the player's understanding of how your game fits in.

Supporting

The feedback you will get can hurt at times but you will learn a huge amount about your game, more than you ever would from just your friends and family. Just remember that your games will get better as you build on your experiences from releasing your first game.

Back your game up with great support. This included bug fixing, game adjustments and keeping in touch with your fanbase.

Adding the ability to sign up to a mailing list in your game will allow you to improve your chances of repeat business as they have been proven to be the most effective way of keeping in touch with your fans. However your users aren't going to give you their email address for nothing, offer an incentive. I've seen additional game features and rewards unlocked when users sign up to the newsletter. Keep patching, keep adding and keep telling people!

The key point to note with your first game is that it's more important to build a fanbase than actual sales.

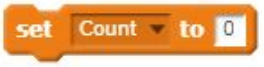
Additional Notes

There are a great deal of Python elements that are used in Scratch. Although they may look a little different, they pretty much do the same thing. The help chart below will show you how some of the most common Scratch commands are translated into Python.

Scratch to Python Help Chart

print(‘Hello’)

Count = 0



Word = ‘Winner’

Number = input(‘Pick a number’)
Print (‘You picked ‘ + Number)

Count = Count + 1

Count + 5

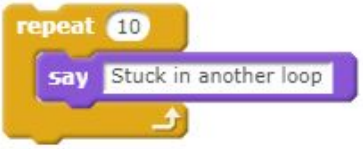
Count - 5

Count * 5

Count / 5


```
while True:
      print (‘Stuck in a loop’)
```


```
for i in range (10):
     print (‘Stuck in another loop’)
```



Count == 2

Count > 2

Count < 2


```
from random import randint
Count = randint(1, 10)
```

and

or

not

print (Word + Count)

len(Word)

Word(0)

ShoppingList = list()

ShoppingList.append(Word)

len(ShoppingList)


```
while Count != 6:
    print('Counting')
```


```
if 'Soap' in ShoppingList:
    print('Smelly')
```

Python Notes

Modules

These are used to import and make available specific functions in your program, for example:

```
import random
import sys
import os
```

Hello World Exercise

This simple Hello World exercise is just one line of code. The print() is used to display the text on the screen;

```
print("Hello World")
```

Variables

A variable allows you to store data throughout your program. It is always best to name a variable to something appropriate so that it is clear what the variable's role is. A variable have to start with a letter but afterwards can contain more letters, numbers or underscores. It cannot start with a number.

```
name = "Jez"
```

```
print(name)
```

There are five different variable types that can be used to store data. The data types are: Numbers, Strings, List, Tuple, Dictionary

```
Number = 15
print(number)
```

Number Variables

The operators that are available to us are: +, -, *, /, %, **, // with ** being and exponential calculation and // being floor division. Some example are below:

```
print("5 + 2 =", 5+2)
print("5 - 2 =", 5-2)
print("5 * 2 =", 5*2)
print("5 / 2 =", 5/2)
print("5 % 2 =", 5%2)
print("5 ** 2 =", 5**2)
print("5 // 2 =", 5//2)
```

It is important to remember that when making calculation in your program to remember the order of operation. For example: print("1 + 2 - 3 * 2 =", 1 + 2 - 3 * 2) will produce a different result than print("(1 + 2 - 3) * 2 =", (1 + 2 - 3) * 2)

String Variables

A string is another name a collection of characters or words. Strings can be joined together as shown below:

```
Weather = “raining”
print(“Today the weather is ” + weather)
```

Wrapping the string in three single quotes will allow you do display a sentence across multiple lines:

```
MultipleLines = ‘’’ This is over
two lines’’’
print(MultipleLines)
```

Strings can also be printed out multiple times:

```
print (“How many times?” * 5)
```

List Variables

A list allows you to create variables with multiple values that can be manipulated and changed. Each list is indexed, with the first value having a value of 0, the second entry having an index value of 1 and so on. This allows us to find our way around the list if we ever need to retrieve or change particular values in a list.

```
shopping_list = [“Milk”, “Bread”, “Bananas”, “Cereal”]
print('The first item is', grocery_list[0])
```

Don’t forget the first item in the list has an index value of 0.

You can also get a range of entries between two index values.

print(shopping_list[1:3])

You can also have lists contained within lists.

todo_list = [“Go shopping”, “Call office”, “Walk dog”, “feed cat”]
list_list = [tot_list, shopping_list]

print(list_list)

If we wanted to get the second item from the second list:

print(list_list[1][1])

The first index is the index value of the list, the second value is the index value of the item within that particular list.

To insert into a list:
shopping_list.insert(1, "Cat food")

To remove an item from the list:
shopping_list.remove("Pickle")

We can also sort the item within a list:
shopping_list.sort()

Let’s sort them the other way:
shopping_list.reverse()

To delete and item from a list:
del shopping_list[4]
print(shopping_list)

We can also combine lists together:
list_list = todo_list + shopping_list
print(list_list)

To get a length of a list:
print(len(list_list))

To get the maximum item in a list:
print(max(list_list))

To get the minimum item in a list:
print(min(list_list))

Tuple Variables

Tuples are similar to lists but their values cannot be changed.

```
tuple = (3, 1, 4, 1, 5, 9)
```

Just like with lists you can get the length, min and max values of a tuple using:

```
print(len(tuple))
print(min(tuple))
print(max(tuple))
```

Dictionary Variables

These are values that have a unique key for each entry.

```
super_villains = {'Fiddler' : 'Isaac Bowin',
                  'Captain Cold' : 'Leonard Snart',
                  'Weather Wizard' : 'Mark Mardon',
                  'Mirror Master' : 'Sam Scudder',
                  'Pied Piper' : 'Thomas Peterson'}
```

So if we wanted to know the real name of one of the villains:

```
print(super_villains['Captain Cold'])
```

To delete an entry:

```
del super_villains['Fiddler']
print(super_villains)
```

To replace a value:

```
super_villains['Pied Piper'] = 'Hartley Rathaway'
```

To print the number of entries in the dictionary:

```
print(len(super_villains))
```

Conditions

Similar to other visual programming languages such as Scratch, the conditional statements that are used in python are: if, else, elif with comparison operators being ==, !=, >, <, >=, <= .

IF statements will only run if the condition they are testing for is true.

```
age = 30
if age > 17 :
    print('You are old enough to drive')
```

Notice how the action that will be carried out if the condition is met is indented. This is how Python determines what is contained within the IF statement.

The ELSE works in a very similar way.

```
if age > 17 :
    print('You are old enough to drive')
else :
```

```
    print('You are not old enough to drive')
```

The ELIF is used to check for multiple conditions:

```
if age >= 21 :
    print('You are old enough to drive a tractor')
elif age >= 17:
    print('You are old enough to drive a car')
else :
    print('You are not old enough to drive')
```

Loops

Loops allows you to perform the same task multiple times.

FOR Loops

You can use loops to cycle through a list:

```
shopping_list = ['Juice', 'Tomatoes', 'Potatoes', 'Bananas']
for y in shopping_list:
    print(y)
```

You can also cycle through a list of values using a range:

```
for x in [2,4,6,8,10]:
    print(x)
```

WHILE Loops

The While loop is useful in situations where you don't know the number of times you need to repeat the task.

```
random_number = random.randrange(0,100)

while (random_number != 15):
    print(random_number)
    random_number = random.randrange(0,100)
```

Functions

Functions allow you to write and reuse the same piece of code. To set up a function you will need to type def, followed by the function's name, followed by the parameters it is to receive.

```
def addNumbers(fNum, sNum):
    sumNum = fNum + sNum
    return sumNum

print(addNumbers(1, 4))
```

You can also include any variables that have been declared outside of the function:

```
newNum = 0;
```

```
def subNumbers(fNum, sNum):
    newNum = fNum - sNum
    return newNum

print(subNumbers(1, 4))
```

User Input

```
print('What is your name?')
name = sys.stdin.readline()

print('Hello', name)
```

Files

You can create and overwrite files with Python.

```
test_file = open("test.txt", "wb")
```

To get the file's name:

```
print(test_file.name)
```

To write a new line to the file:

```
test_file.write(bytes("Write me to the file\n", 'UTF-8'))
```

To close the file:

```
test_file.close()
```

To open a file for reading and writing:

```
test_file = open("test.txt", "r+")
```

To read text within a file:

```
text_in_file = test_file.read()
print(text_in_file)
```

To delete a file:

```
os.remove("test.txt")
```

Classes and Objects

These allow you to create objects with multiple attributes and these are called object variables. You can also create object abilities and these are called object functions.

```
class Animal:
    __name = 0
    __height = 0
    __weight = 0
    __sound = 0
```

To set up an object we use something called a constructor.

```
    def __init__(self, name, height, weight, sound):
```

```
        self.__name = name
        self.__height = height
        self.__weight = weight
        self.__sound = sound
```

The self allows the function to refer to itself:

```
    def set_name(self, name):
        self.__name = name

    def set_height(self, height):
        self.__height = height

    def set_weight(self, height):
        self.__height = height

    def set_sound(self, sound):
        self.__sound = sound

    def get_name(self):
        return self.__name

    def get_height(self):
        return str(self.__height)

    def get_weight(self):
        return str(self.__weight)

    def get_sound(self):
        return self.__sound

    def get_type(self):
        print("Animal")
```

To print out all the information on the screen:

```
    def toString(self):
        return "{} is {} cm tall and {} kilograms and says {}".format(self.__name, self.__height, self.__weight,
self.__sound)
```

To add a new object:

```
cat = Animal('Whiskers', 33, 10, 'Meow')

print(cat.toString())
```

To access the value directly:

```
print(cat.__name)

# INHERITANCE ----------
# You can inherit all of the variables and methods from another class

class Dog(Animal):
    __owner = None
```

```
    def __init__(self, name, height, weight, sound, owner):
        self.__owner = owner
        self.__animal_type = None

        # How to call the super class constructor
        super(Dog, self).__init__(name, height, weight, sound)

    def set_owner(self, owner):
        self.__owner = owner

    def get_owner(self):
        return self.__owner

    def get_type(self):
        print ("Dog")

    # We can overwrite functions in the super class
    def toString(self):
        return "{} is {} cm tall and {} kilograms and says {}. His owner is {}".format(self.get_name(),
self.get_height(), self.get_weight(), self.get_sound(), self.__owner)

    # You don't have to require attributes to be sent
    # This allows for method overloading
    def multiple_sounds(self, how_many=None):
        if how_many is None:
            print(self.get_sound)
        else:
            print(self.get_sound() * how_many)

spot = Dog("Spot", 53, 27, "Ruff", "Derek")

print(spot.toString())

# Polymorphism allows use to refer to objects as their super class
# and the correct functions are called automatically

class AnimalTesting:
    def get_type(self, animal):
        animal.get_type()

test_animals = AnimalTesting()

test_animals.get_type(cat)
test_animals.get_type(spot)

spot.multiple_sounds(4)
'''
This is a multi-line comment
'''

# A variable is a place to store values
# Its name is like a label for that value
```

```
name = "Derek"
print(name)

# A variable name can contain letters, numbers, or _
# but can't start with a number

# There are 5 data types Numbers, Strings, List, Tuple, Dictionary
# You can store any of them in the same variable

name = 15
print(name)

# The arithmetic operators +, -, *, /, %, **, //
# ** Exponential calculation
# // Floor Division
print("5 + 2 =", 5+2)
print("5 - 2 =", 5-2)
print("5 * 2 =", 5*2)
print("5 / 2 =", 5/2)
print("5 % 2 =", 5%2)
print("5 ** 2 =", 5**2)
print("5 // 2 =", 5//2)

# Order of Operation states * and / is performed before + and -

print("1 + 2 - 3 * 2 =", 1 + 2 - 3 * 2)
print("(1 + 2 - 3) * 2 =", (1 + 2 - 3) * 2)

# A string is a string of characters surrounded by " or '
# If you must use a " or ' between the same quote escape it with \
quote = "\"Always remember your unique,"

# A multi-line quote
multi_line_quote = ''' just
like everyone else" '''

print(quote + multi_line_quote)

# To embed a string in output use %s
print("%s %s %s" % ('I like the quote', quote, multi_line_quote))

# To keep from printing newlines use end=""
print("I don't like ",end="")
print("newlines")

# You can print a string multiple times with *
print('\n' * 5)

# LISTS ----------
# A list allows you to create a list of values and manipulate them
# Each value has an index with the first one starting at 0
```

```
grocery_list = ['Juice', 'Tomatoes', 'Potatoes', 'Bananas']
print('The first item is', grocery_list[1])

# You can change the value stored in a list box
grocery_list[0] = "Green Juice"
print(grocery_list)

# You can get a subset of the list with [min:up to but not including max]

print(grocery_list[1:3])

# You can put any data type in a a list including a list
other_events = ['Wash Car', 'Pick up Kids', 'Cash Check']
to_do_list = [other_events, grocery_list]

print(to_do_list)

# Get the second item in the second list (Boxes inside of boxes)
print(to_do_list[1][1])

# You add values using append
grocery_list.append('onions')
print(to_do_list)

# Insert item at given index
grocery_list.insert(1, "Pickle")

# Remove item from list
grocery_list.remove("Pickle")

# Sorts items in list
grocery_list.sort()

# Reverse sort items in list
grocery_list.reverse()

# del deletes an item at specified index
del grocery_list[4]
print(to_do_list)

# We can combine lists with a +
to_do_list = other_events + grocery_list
print(to_do_list)

# Get length of list
print(len(to_do_list))

# Get the max item in list
print(max(to_do_list))

# Get the minimum item in list
print(min(to_do_list))
```

```
# TUPLES ----------
# Values in a tuple can't change like lists

pi_tuple = (3, 1, 4, 1, 5, 9)

# Convert tuple into a list
new_tuple = list(pi_tuple)

# Convert a list into a tuple
# new_list = tuple(grocery_list)

# tuples also have len(tuple), min(tuple) and max(tuple)

# DICTIONARY or MAP ----------
# Made up of values with a unique key for each value
# Similar to lists, but you can't join dicts with a +

super_villains = {'Fiddler' : 'Isaac Bowin',
                  'Captain Cold' : 'Leonard Snart',
                  'Weather Wizard' : 'Mark Mardon',
                  'Mirror Master' : 'Sam Scudder',
                  'Pied Piper' : 'Thomas Peterson'}

print(super_villains['Captain Cold'])

# Delete an entry
del super_villains['Fiddler']
print(super_villains)

# Replace a value
super_villains['Pied Piper'] = 'Hartley Rathaway'

# Print the number of items in the dictionary
print(len(super_villains))

# Get the value for the passed key
print(super_villains.get("Pied Piper"))

# Get a list of dictionary keys
print(super_villains.keys())

# Get a list of dictionary values
print(super_villains.values())

# CONDITIONALS ----------
# The if, else and elif statements are used to perform different
# actions based off of conditions
# Comparison Operators : ==, !=, >, <, >=, <=

# The if statement will execute code if a condition is met
# White space is used to group blocks of code in Python
```

```
# Use the same number of proceeding spaces for blocks of code

age = 30
if age > 16 :
    print('You are old enough to drive')

# Use an if statement if you want to execute different code regardless
# of whether the condition ws met or not

if age > 16 :
    print('You are old enough to drive')
else :
    print('You are not old enough to drive')

# If you want to check for multiple conditions use elif
# If the first matches it won't check other conditions that follow

if age >= 21 :
    print('You are old enough to drive a tractor trailer')
elif age >= 16:
    print('You are old enough to drive a car')
else :
    print('You are not old enough to drive')

# You can combine conditions with logical operators
# Logical Operators : and, or, not

if ((age >= 1) and (age <= 18)):
    print("You get a birthday party")
elif (age == 21) or (age >= 65):
    print("You get a birthday party")
elif not(age == 30):
    print("You don't get a birthday party")
else:
    print("You get a birthday party yeah")

# FOR LOOPS ----------
# Allows you to perform an action a set number of times
# Range performs the action 10 times 0 - 9
for x in range(0, 10):
    print(x , ' ', end="")

print('\n')

# You can use for loops to cycle through a list
grocery_list = ['Juice', 'Tomatoes', 'Potatoes', 'Bananas']

for y in grocery_list:
    print(y)

# You can also define a list of numbers to cycle through
for x in [2,4,6,8,10]:
```

```
    print(x)

# You can double up for loops to cycle through lists
num_list =[[1,2,3],[10,20,30],[100,200,300]];

for x in range(0,3):
    for y in range(0,3):
        print(num_list[x][y])

# WHILE LOOPS ----------
# While loops are used when you don't know ahead of time how many
# times you'll have to loop
random_num = random.randrange(0,100)

while (random_num != 15):
    print(random_num)
    random_num = random.randrange(0,100)

# An iterator for a while loop is defined before the loop
i = 0;
while (i <= 20):
    if(i%2 == 0):
        print(i)
    elif(i == 9):
        # Forces the loop to end all together
        break
    else:
        # Shorthand for i = i + 1
        i += 1
        # Skips to the next iteration of the loop
        continue

    i += 1

# FUNCTIONS ----------
# Functions allow you to reuse and write readable code
# Type def (define), function name and parameters it receives
# return is used to return something to the caller of the function
def addNumbers(fNum, sNum):
    sumNum = fNum + sNum
    return sumNum

print(addNumbers(1, 4))

# Can't get the value of rNum because it was created in a function
# It is said to be out of scope
# print(sumNum)

# If you define a variable outside of the function it works every place
newNum = 0;
def subNumbers(fNum, sNum):
    newNum = fNum - sNum
```

```
    return newNum

print(subNumbers(1, 4))

# USER INPUT ----------
print('What is your name?')

# Stores everything typed up until ENTER
name = sys.stdin.readline()

print('Hello', name)

# STRINGS ----------
# A string is a series of characters surrounded by ' or "
long_string = "I'll catch you if you fall - The Floor"

# Retrieve the first 4 characters
print(long_string[0:4])

# Get the last 5 characters
print(long_string[-5:])

# Everything up to the last 5 characters
print(long_string[:-5])

# Concatenate part of a string to another
print(long_string[:4] + " be there")

# String formatting
print("%c is my %s letter and my number %d number is %.5f" % ('X', 'favorite', 1, .14))

# Capitalizes the first letter
print(long_string.capitalize())

# Returns the index of the start of the string
# case sensitive
print(long_string.find("Floor"))

# Returns true if all characters are letters ' isn't a letter
print(long_string.isalpha())

# Returns true if all characters are numbers
print(long_string.isalnum())

# Returns the string length
print(len(long_string))

# Replace the first word with the second (Add a number to replace more)
print(long_string.replace("Floor", "Ground"))

# Remove white space from front and end
print(long_string.strip())
```

```
# Split a string into a list based on the delimiter you provide
quote_list = long_string.split(" ")
print(quote_list)

# FILE I/O ----------

# Overwrite or create a file for writing
test_file = open("test.txt", "wb")

# Get the file mode used
print(test_file.mode)

# Get the files name
print(test_file.name)

# Write text to a file with a newline
test_file.write(bytes("Write me to the file\n", 'UTF-8'))

# Close the file
test_file.close()

# Opens a file for reading and writing
test_file = open("test.txt", "r+")

# Read text from the file
text_in_file = test_file.read()

print(text_in_file)

# Delete the file
os.remove("test.txt")

# CLASSES AND OBJECTS ----------
# The concept of OOP allows us to model real world things using code
# Every object has attributes (color, height, weight) which are object variables
# Every object has abilities (walk, talk, eat) which are object functions

class Animal:
    # None signifies the lack of a value
    # You can make a variable private by starting it with __
    __name = None
    __height = None
    __weight = None
    __sound = None

    # The constructor is called to set up or initialize an object
    # self allows an object to refer to itself inside of the class
    def __init__(self, name, height, weight, sound):
        self.__name = name
        self.__height = height
        self.__weight = weight
```

```
        self.__sound = sound

    def set_name(self, name):
        self.__name = name

    def set_height(self, height):
        self.__height = height

    def set_weight(self, height):
        self.__height = height

    def set_sound(self, sound):
        self.__sound = sound

    def get_name(self):
        return self.__name

    def get_height(self):
        return str(self.__height)

    def get_weight(self):
        return str(self.__weight)

    def get_sound(self):
        return self.__sound

    def get_type(self):
        print("Animal")

    def toString(self):
        return "{} is {} cm tall and {} kilograms and says {}".format(self.__name, self.__height, self.__weight,
self.__sound)

# How to create a Animal object
cat = Animal('Whiskers', 33, 10, 'Meow')

print(cat.toString())

# You can't access this value directly because it is private
#print(cat.__name)

# INHERITANCE ----------
# You can inherit all of the variables and methods from another class

class Dog(Animal):
    __owner = None

    def __init__(self, name, height, weight, sound, owner):
        self.__owner = owner
        self.__animal_type = None

        # How to call the super class constructor
```

```
        super(Dog, self).__init__(name, height, weight, sound)

    def set_owner(self, owner):
        self.__owner = owner

    def get_owner(self):
        return self.__owner

    def get_type(self):
        print ("Dog")

    # We can overwrite functions in the super class
    def toString(self):
        return "{} is {} cm tall and {} kilograms and says {}. His owner is {}".format(self.get_name(),
self.get_height(), self.get_weight(), self.get_sound(), self.__owner)

    # You don't have to require attributes to be sent
    # This allows for method overloading
    def multiple_sounds(self, how_many=None):
        if how_many is None:
            print(self.get_sound)
        else:
            print(self.get_sound() * how_many)

spot = Dog("Spot", 53, 27, "Ruff", "Derek")

print(spot.toString())

# Polymorphism allows use to refer to objects as their super class
# and the correct functions are called automatically

class AnimalTesting:
    def get_type(self, animal):
        animal.get_type()

test_animals = AnimalTesting()

test_animals.get_type(cat)
test_animals.get_type(spot)

spot.multiple_sounds(4)
```