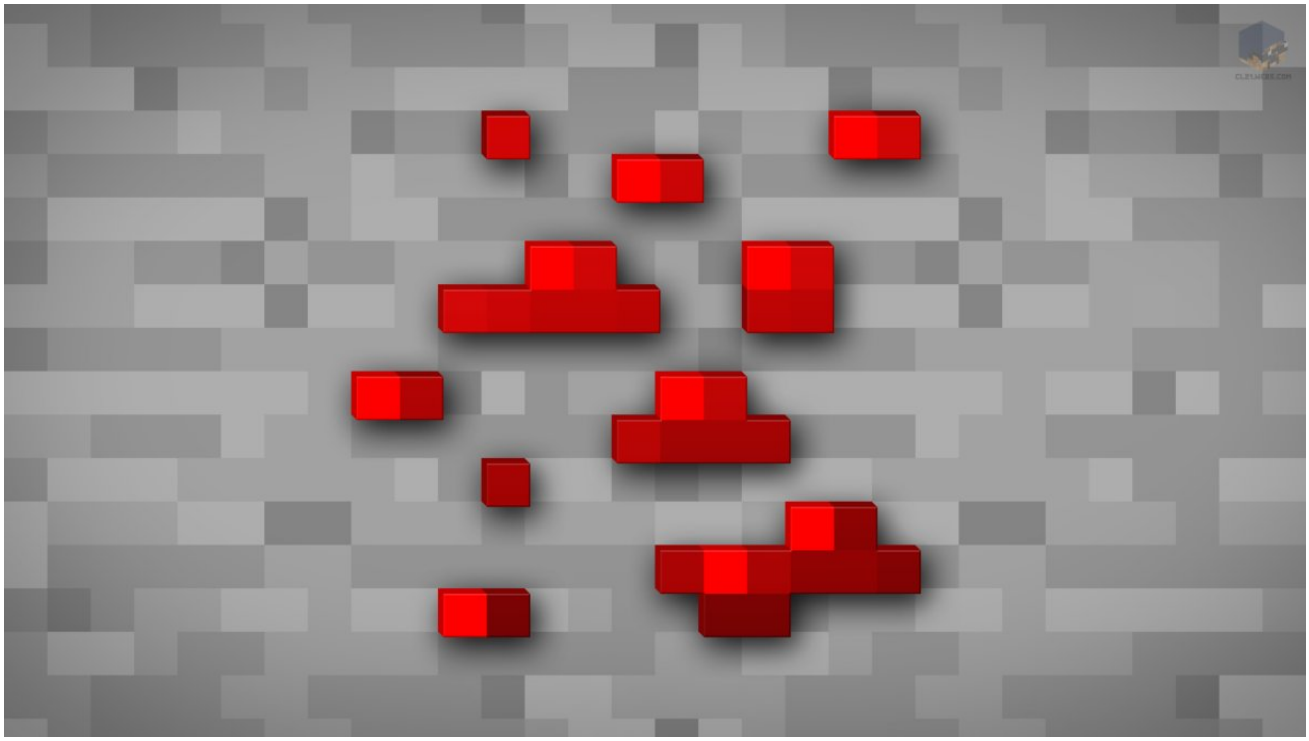


Minecraft U Sequence 3: ComputerCraft

Use the Minecraft mod ComputerCraft to learn the basics of programming and then apply those skills via your own in-game robots. These robots can do anything, from build you a house to find and mine diamonds for you. The only limit is your ability to direct them. At the advanced level, we'll discuss the limitations of computers and how to work around them.

Table of Contents

Section 1: ComputerCraft Basics

Introduction

> ComputerCraft is a modification for Minecraft that's all about computer programming. It allows you to build in-game Computers and Turtles, and write programs for them using the Lua programming language. The addition of programming to Minecraft opens up a wide variety of new possibilities for automation and creativity. If you've never programmed before, it also serves as excellent way to learn a real world skill in a fun, familiar environment.

This will be your first taste of the ComputerCraft mod, and for some of you, your first time using a command line interface. With this knowledge, you can start using other command line software, and you'll get a solid foundation in simple programming skills.

The Command Line

- Open the ComputerCraft world
- Open your inventory and search for `computer`
- Place a computer on the ground and right-click on it

When you place a computer and right click on it, the first thing you see is this.

The beginning of your command line life.

This is a command line. Here, we can type words to run programs, and we can look inside folders just like in a graphical interface. Let's do that now.

Some simple commands to run.

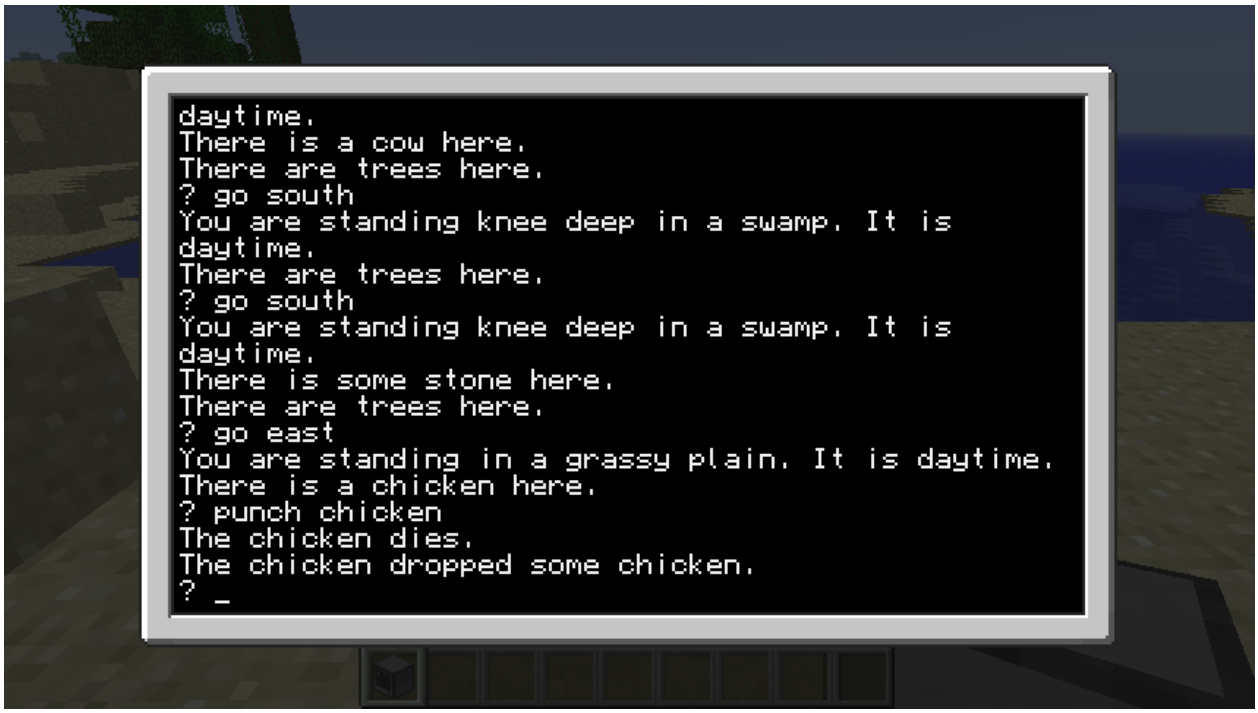
- The `ls` command lists the contents of the current directory
- The `cd` command changes directories, e.g. `cd rom`

Up next we'll run the edit program and see what it looks like.
Type `edit test`.

This is the program you use to edit your programs. Looks pretty simple huh? This program is all you need to start writing you own code in ComputerCraft.

Play text adventure Minecraft inside a ComputerCraft computer

- Type `adventure`
- Some of the commands available in the Adventure program:
  - `punch`
  - `take` or `grab`
  - `craft` or `make`
  - `go`
  - `eat`
  - `inventory`



Adventure

In case you haven't noticed by now, Adventure is really just text-based Minecraft. You're playing Minecraft on a computer inside Minecraft.

Peripherals

- Open your inventory and search for `disk drive`
- Place the disk drive next to the computer

- Right click the disk drive to open it.
- Try putting a music disk in the disk drive.
- Play the music by right clicking your computer and running the `dj` program.

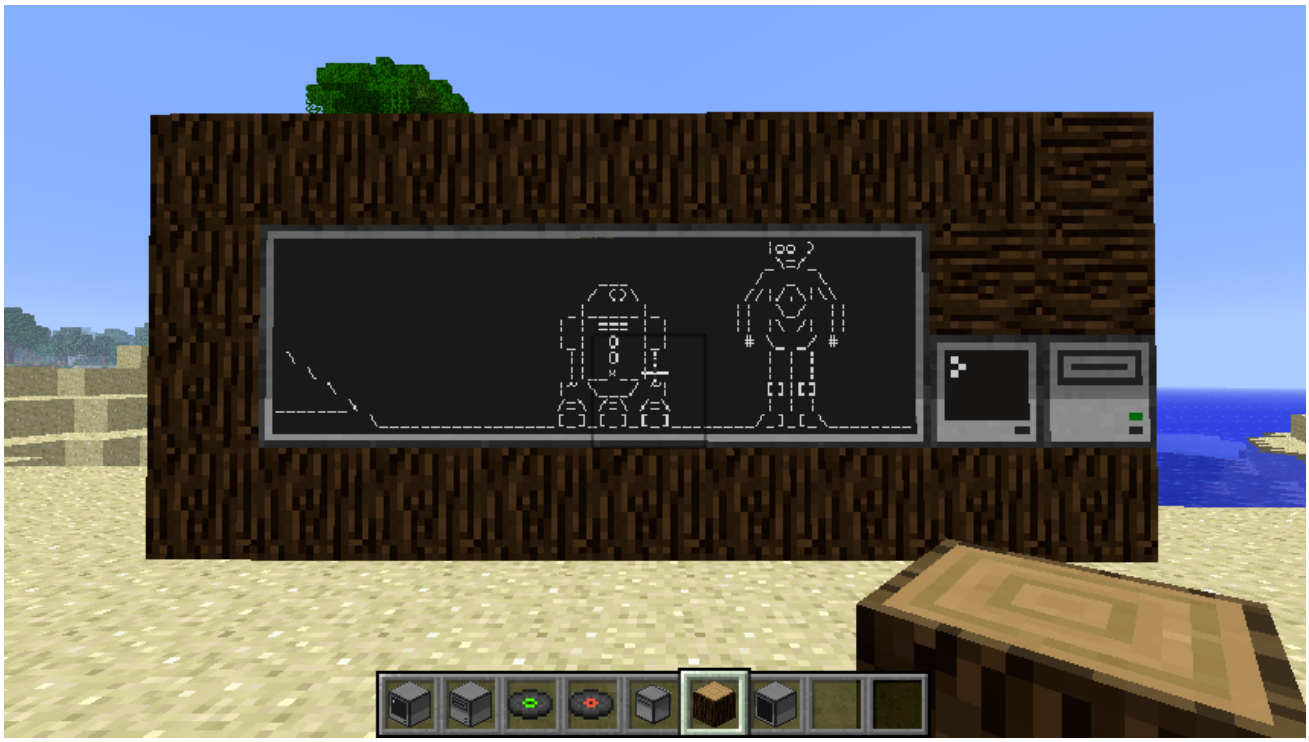
Create a monitor

- Open your inventory and search for `monitor`
- Place 12 monitors in a 6 wide by 2 high pattern to create a giant widescreen monitor
- Place a computer next to the monitor
- Place a disk drive next to the computer

Watch a movie

- Open your inventory and search for `disk`
- Find a disk labeled `alongtimeago` and place it into the disk drive

- Right click on the computer and run the `alongtimeago` program
- To run a program on the disk, specify the full path to the program like this: `disk/alongtimeago`
- To run the program on the monitor, specify the monitor first, like this: `monitor [top|bottom|left|right|front|back] disk/alongtimeago`
- The syntax `[top|bottom|left|right|front|back]` means pick which side your monitor is on and only type that direction, of those six shown. So your command would be something like `monitor left disk/alongtimeago`.



Watching on the big screen

- To quit any running program, hold down `ctrl + t`
- To restart the computer, hold down `ctrl + r`

Turtles

Intro

Turtles are programmable robots that you can use to collect resources, clear terrain, and other such tasks. They run an OS called turtleOS and the programs they run can be stored on internal memory or floppy disks. There are farming, mining, crafting, and melee turtles. They are categorized based on the Diamond tool* you equip them with.

*Note: Tools equipped to turtles will not wear out and turtles themselves are indestructible (unless you break them yourself). This makes them one of the safest ways to utilize diamond tools, not to mention the time they will save you.

Like any robot, turtles require fuel. They can get energy from anything that would work in a furnace as well as other more advanced options we'll get to later. Different types of fuel will yield different *fuel counts* which is the number of blocks the turtle can move with that amount of fuel. For example, coal will give the turtle 80 fuel, so the turtle can now move 80 blocks.

1. Add some dancing turtles
2. Open your inventory and search for `turtle`
3. Place a turtle or two on the ground
4. Right-click on the turtle
5. Run the `dance` program

Make it move!

Turtles have several default programs including the “go” program.

1. Select a turtle and put a *coal* in its inventory.
2. type `refuel`
3. Notice it says *Fuel level is 80*
4. type `go forward 10` and watch it go!
5. type `refuel` and notice that the fuel level is now 70.
6. Whenever there is no fuel source in the turtle’s inventory, you can type `refuel` to check its fuel level.

The “go” program has the following format:
`go <direction> <distance>`

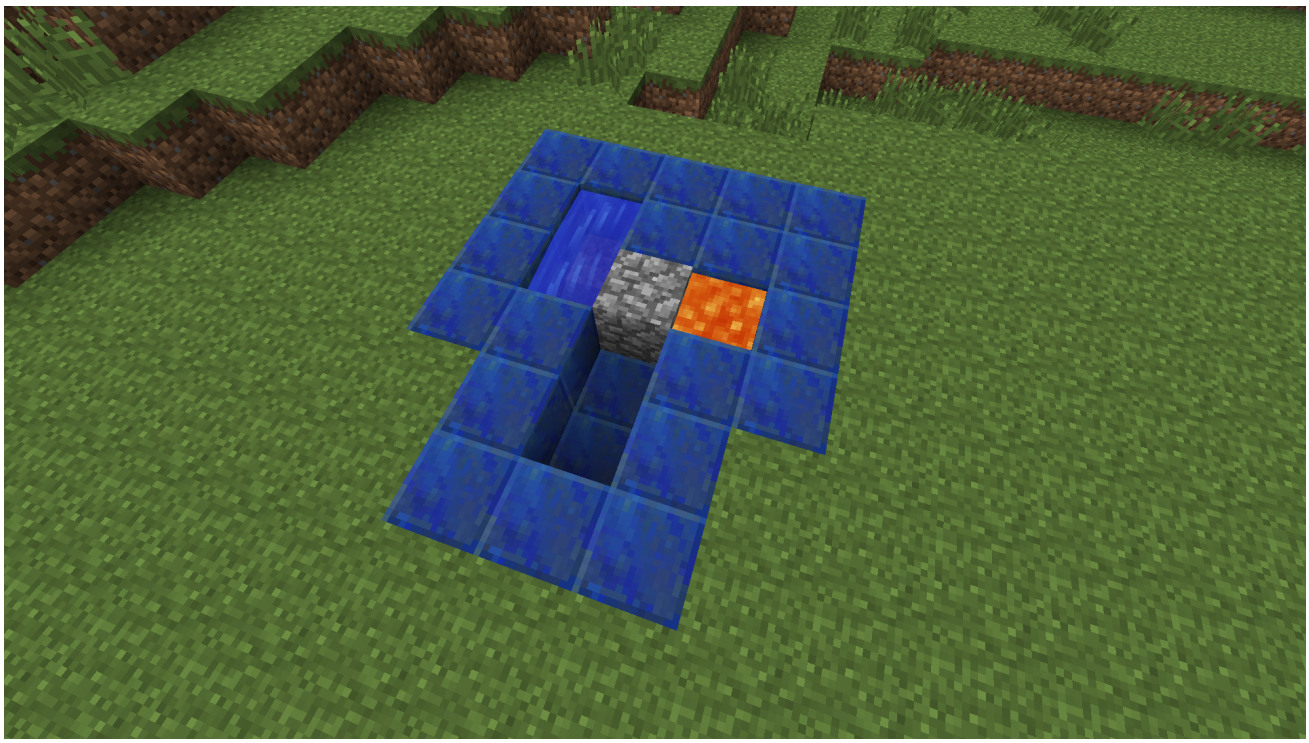
Note: For fast/mass refueling, type `refuel all`

Section 2: Cobblestone Miner

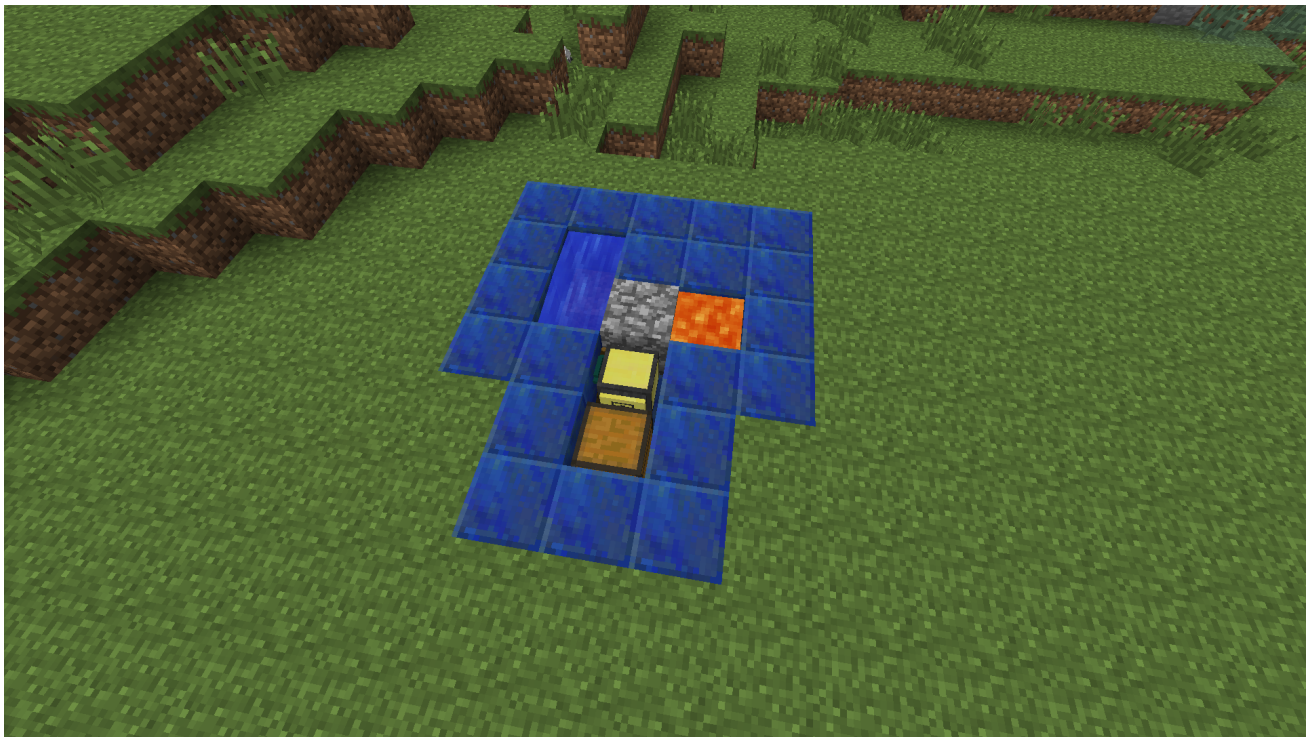
> So far all we’ve done is run commands that are already built into the turtles and computers. Starting in this section, we’ll write some of our own programs, to make the turtle do whatever we want it to.

Let’s start off with a simple program, one that will make sure you never run out of cobblestone again. If you’ve used a cobblestone generator before, this is similar to that, except that it’s a turtle doing the harvesting, not you.

First build a simple cobblestone generator, seen here surrounded by lapis blocks for decoration.



If you mine out that cobblestone, more will replace it after it breaks. Now we need to place our turtle so it can mine the cobblestone for us. Place your turtle facing the cobblestone, and put a chest behind it. Make sure you use a `mining turtle`, and to make your code easier to write, use an `advanced mining turtle`.



Now we need to set our turtle's label, so that if we ever break him, he'll keep the programs we write. Type `label set cobblefarmer` and press `enter`.

Congrats! You've done all the actual block placing you need to. Now we can start writing code. Type `edit cobblefarm` and press `enter`. This will open the editing program and let you start writing your own code.

The editor program looks like this. Since you typed `edit cobblefarmer`, your program will be called `cobblefarmer` when you save it. To open the `SAVE/EXIT/PRINT` menu, press `control` or `ctrl` on your keyboard. The text in the corner that says `Ln 1` is the line counter. When you write code, it's split up into lines, just like regular writing. Whenever there's an error in your code, it will tell you which line the error is on, and that's when this line counter is really useful.

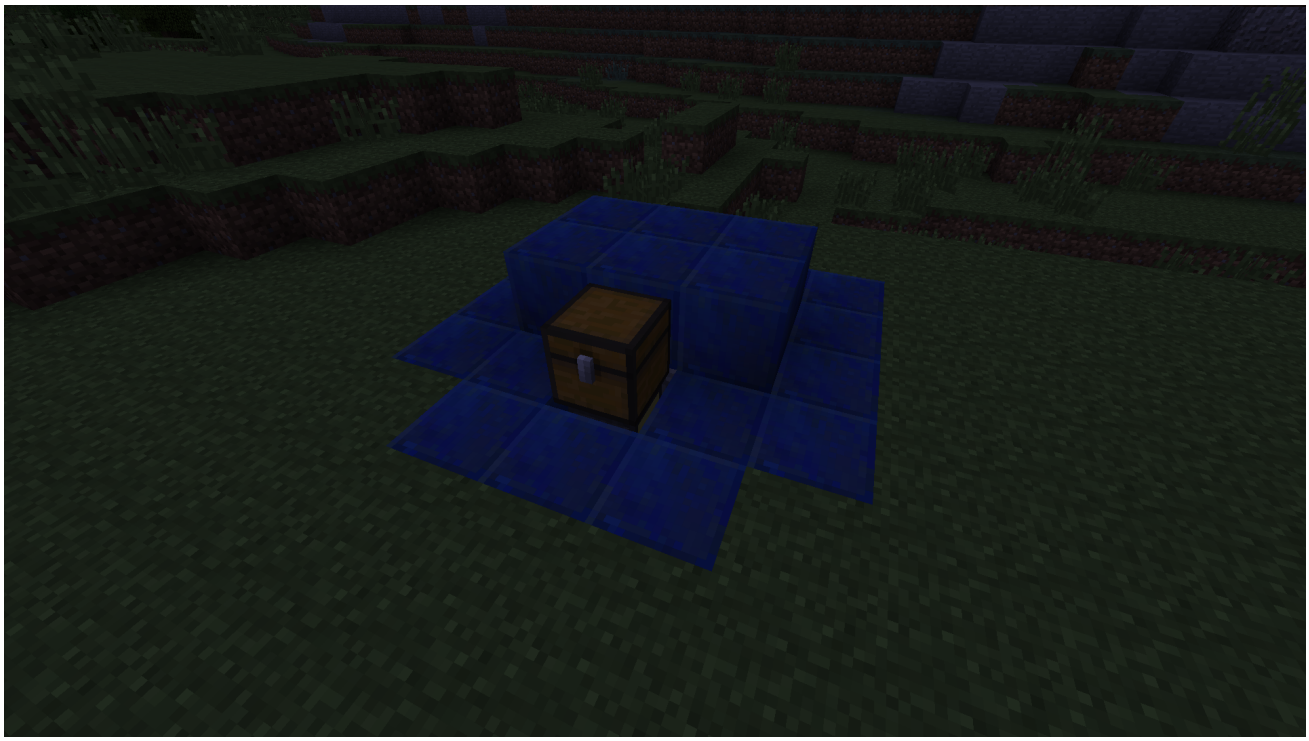
Let's start writing some code! Start off by copying down the code written here. This is called a while loop. The `while` command checks if something is true or false and then keeps running the code if the thing is true. The `do` part is what tells the computer that you're done defining the loop and you want it to start running code. At the end of every loop you write, you have to have the code `end`. This tells the program to end the looped section of code.

Since we wrote `while true do`, the code will run forever, because the statement `true`, by itself, will always be `true` and not `false`.

Next we'll fill in our loop with some code. All we need for this program is two commands. Copy down the code as shown below. The command `turtle.dig()` tells the turtle to mine the block in front of it. Once it mines the block, it will have it in it's inventory. The second command is to put the cobblestone into a chest, and it looks like we made a mistake! There are command for `turtle.drop()`, `turtle.dropUp()`, and `turtle.dropDown()`, but nothing for `turtle.dropBack()`. Instead we'll just use `turtle.dropUp()` and move our chest in a moment.

Once you've written you two lines of code, press `control` or `ctrl` on your keyboard to open the menu, and then press `enter` to save your program. Now press `control` again, press the `right arrow key` to move to `exit`, and then press `enter` to close the editor program.

Before we start our program, let's move our chest. We should also cover up the lava and water so it's safer to keep around, and the chest doesn't light on fire.



Now look at your turtle behind the chest and right click it. Type in `cobblefarm` and press `enter`. This will start your program! If you press `escape` and then look in the chest, you'll see that your turtle is gathering cobblestone for you.

Once the chest is full, you can open your turtle again and hold down `control` and `t` at the same time to terminate your program.

Feel good about yourself! You just wrote your first simple program in ComputerCraft. In the next few sections, we'll write more useful programs, and more complex ones.

Section 3: Chopping Trees

> From now on, all the code you see in this book will be text only, not a screenshot from the game. It's still the same code, and it will work just like a screenshot would.

For this section, we should pick another repetitive task that we have to do often and give it to the turtle. Let's use tree chopping! That takes awhile, and you can always use more wood.

For this program, you need:

- An `Advanced Felling Turtle`.
- A sapling, **birch or spruce only**.
- Some dirt.
- The Sun.
- Your brain.

Let's kick it off by putting our items in the correct places. Plant the sapling on the dirt, make sure it has plenty of room on all sides to grow, and then put the turtle down facing the sapling.

> Remember to label your turtle whenever you make a new program. Use `label set <name>`

Now let's start writing our program. You'll want to put your code in a while loop again, since we want it to always run.

```
while true do

end
```

Now think about what you need to do to chop down a tree. First you have to break the wood, then move up (breaking any leaves in your way), then break again. Once there are no more blocks to chop, you have to go back to where you came from and put your spoils in a chest.

> Thinking through programs like this before you write them will make it easier to organize them, and will make it more likely for your program to work on the first try.

There are a few fancy commands that let turtles look at what kind of block is in front of them. One of them is `turtle.inspect()`. We'll use this and a **variable** to look at the block in front of the turtle. A variable is just a store for some data. We can set variables to `true`, `false`, `elbow`, `1231241`, or anything else we can think of.

```
while true do
  local success, data = turtle.inspect()
  print(data.name)
end
```

This code will set `success` to `true` or `false` depending on if it sees a block or not. It also sets `data` to the the block it finds. You can then use `data.name` to get the name of the block as a **string**.

> A **string** is a type of data in a program. You can have strings, numbers, and booleans (which are just true or false) in a program.

Here we use the `print()` function to print out the name. If you run this code while the turtle is facing a sapling, it prints out `minecraft:sapling` forever. Once the tree grows, it will print out `minecraft:log` instead.

Now that we can check what block we're looking at, we can start cutting down trees.

```
while true do
  local success, data = turtle.inspect()
  if data.name == "minecraft:log" then
    while turtle.detect() do

    end
  end
end
```

Now, even though our program checks all the time for an item, it will only start doing things if that item is a `minecraft:log`. We did this by using an **if/then statement**.

> An if/then statement is similar to a while loop because it checks if something is true. Here we check if the item is a log or not. If it is, the code inside the statement runs, if not, then the code is skipped. The code will only run once if it does run, and then the program will continue.

We also put another while loop inside the if/then statement. This will keep doing the same thing over and over until there are no blocks in front of the turtle.

Now let's fill in the code to cut down the tree.

```
while true do
  local success, data = turtle.inspect()
  if data.name == "minecraft:log" then
    while turtle.detect() do
      turtle.dig()
      turtle.digUp()
      turtle.up()
    end
  end
end
```

This code will chop the tree and break leaves until it gets to the top, and then stop. After that we have to go back down. We can do that using another while loop.

```
while true do
  local success, data = turtle.inspect()
  if data.name == "minecraft:log" then
    while turtle.detect() do
      turtle.dig()
      turtle.digUp()
      turtle.up()
    end
    while not turtle.detectDown() do
      turtle.down()
    end
  end
end
```

Now we're back where we started, and we have a tree in our inventory. We should place a chest below the turtle's starting place and have the turtle put the wood into it.

```
while true do
  local success, data = turtle.inspect()
  if data.name == "minecraft:log" then
    while turtle.detect() do
      turtle.dig()
      turtle.digUp()
      turtle.up()
    end
    while not turtle.detectDown() do
      turtle.down()
    end
    turtle.dropDown()
  end
end
```

And now try running your program! If you get an error, look for the line number in the error and double check that line. If it works, you should now have an endless source of wood. All you need to do is plant the saplings.

> It's possible to have the turtle plant saplings also, but that takes a lot more code, and it's more fun to figure that out for yourself. Try looking at the ComputerCraft command reference for the commands to check the turtle's inventory for items.

Yay! That was your second program, and this one was much more complex. Now you've taken two boring and time consuming things and made turtles that can do them for you!

Section 4: Diggy Diggy Hole

Another useful program! You can point this one at a wall and it will dig a two block tall and one block wide tunnel. It will also place torches every 8 blocks.

> We're going to be building this program with **functions**. Functions are what you use in more complex programs so that you don't have to type all of your code in the same block over and over again. Think of a function as something that does one thing really well, and if you put them together, they do a complex task.

The first function we write will check if the turtle is on solid ground, and if not, it will place a block below itself. You define functions like this.

```
function placeFloorBlock()
  if not turtle.detectDown() then
    turtle.select(3)
    if turtle.placeDown() then
      return true
    end
    print("Placing Floor Failed")
    return false
  end
end
```

Now you can use `placeFloorBlock()` anywhere else in this program and it will run this code.

This next function is a basic check for fuel, and if it sees that the turtle is low on fuel, it will try to refuel from inventory slot 1.

```
function fuel()
  if turtle.getFuelLevel() < 25 then
    turtle.select(1)
    if turtle.refuel(1) then
      return true
    else
      print("Refuelling Failed")
      return false
    end
  end
end
```

You can see that this function returns true if the refueling was successful, and false if not.

Our next function digs our tunnel. The basic idea of this while and if loop is that if the turtle is not moving forward, it will dig, and if it can't dig, it attacks. It also detects blocks above it and digs those as well.

```
function DigAndMove()
  while not turtle.forward() do
    if not turtle.dig() then
      turtle.attack()
    end
  end
  while turtle.detectUp() do
    turtle.digUp()
    sleep(0.5)
  end
  placeFloorBlock()
end
```

This function turns the turtle around. Should be obvious.

```
function turnAround()
  turtle.turnLeft()
  turtle.turnLeft()
end
```

This is a function that turns the turtle to the right, digs out a block, checks make sure that the block didn't get filled in, and then places a torch in the space. If it fails to place a torch, it returns false, otherwise it returns true.

```
function placeTorch()
  turtle.turnRight()
  turtle.dig()
  if not turtle.detect() then
    turtle.select(2)
    if turtle.place() then
      turtle.turnLeft()
      return true
    end
  end
  print("Place Torch Failed")
  turtle.turnLeft()
  return false
end
```

Here we set two variables, one for the length you give as an argument, and one to use in the `moveBack` section of code.

```
local length = tonumber(tArgs[1])
local moveBack = 0
```

> The `tArgs[1]` bit of the code above is an **argument**. And argument is something you give a program when you run it. If you run a program like `go forward 10`, then your first argument (`tArgs[1]`) is `forward` and your second argument (`tArgs[2]`) is `10`.

Now we put it all together.

- First we fuel the turtle by calling `fuel()`.
- Then we call `DigAndMove()` to move the turtle forward one tunnel section.
- We increment the `blocksMovedForward` variable to keep track of where we are.
- Next we check if the block we are on is a multiple of 8.
- If it is, we call the `placeTorch()` function.
- At the end, we turn the turtle around and retrace our steps back to the start of the tunnel.

```
local blocksMovedForward = 0
while blocksMovedForward < length do
  fuel()
  DigAndMove()
  blocksMovedForward = blocksMovedForward + 1

  -- Add torch every 8 blocks
  if (blocksMovedForward % 8) == 0 then
    placeTorch()
  end

  if blocksMovedForward == length then
    turnAround()
    while moveBack < length do
      turtle.forward()
      sleep(1)
      print("Taking step: ")
      print(moveBack)
      moveBack = moveBack + 1
    end
  end
end
```

Woop! That was your third program. This one was long, but it let's you find diamonds much easier. Try digging down to level 12 or 13 and putting six turtles running this program down, with a 2 block space between them. You'll be drowning in diamonds in no time.

Section 5: The Room

Let's write a program to build a simple room: `edit room`

Step one is to check your arguments to make sure they are what you need them to be.

For that we use the code `local tArgs = {...}`, which loads the arguments into your program.

```
local tArgs = {...}

if #tArgs ~= 3 then
  print ("Usage: room <l> <w> <h>")
end
for i=1,3 do
  if tonumber(tArgs[i]) < 1 then
    print("Usage: room <l> <w> <h>")
  end
end
```

This block of code takes arguments and checks to make sure there are 3 of them, and that all are numbers greater than 1.
If they aren't, it prints a line that tells the user how to run the program.

Now, we take the three arguments and assign them to the correct variables.

```
length = tArgs[1]
width = tArgs[2]
height = tArgs[3]
```

And we refuel the turtle.

```
turtle.select(1)
turtle.refuel()

if turtle.getFuelLevel() < 150 then
  print("Fuel level too low.")
  return false
end
```

Here, we select the first slot, and load the turtle with fuel from that slot.

This program is going to have a few functions as well. This is our first function. It searches for items starting in slot 2, the slot right after the fuel slot. Note that it doesn't check to make sure they are placable items, and will not stop you from trying to build a house made of carrots.

```
function findItems()
  local slot = 2
  while slot < 16 do
    if turtle.getItemCount(slot) < 1  then
      slot = slot + 1
    else
      turtle.select(slot)
      return true
    end
  end
  return false
end
```

This function builds one single row of blocks, using a for loop to stop it when it reaches the desired length.

```
function buildRow(rowLen)
  for i = 1, rowLen, 1 do
    findItems()
    turtle.placeDown()
    turtle.forward()
  end
end
```

This function calls the `buildRow()` function using the variables we set earlier, to make a whole layer of the house. Note that it turns right every time. This means that you must start the house at the bottom left corner.

```
function buildLayer()
  turtle.up()
  buildRow(length)
  turtle.turnRight()
  buildRow(width)
  turtle.turnRight()
  buildRow(length)
  turtle.turnRight()
  buildRow(width)
  turtle.turnRight()
end
```

Now that we have useful functions to call, we can write the part of the program that does things!

```
for i=1, height, 1 do
  buildLayer()
end
```

All this does is call buildLayer() for the number of layers you specified with `height`

And we're done! Feed your turtle coal and a building material and run the program with `room <length> <width> <height>`

> Now you're finished with the main parts of Sequence 3! Hopefully you have a good idea of how to write your own programs in the future. If you have any questions feel free to ask us, and remember, *the wiki is your friend*.