#

# Looping

# Running Ruby

Just like the other topics, we will be running our Ruby here

<http://repl.it/languages/Ruby>.

We’ll be using some of the stuff we learnt before, so unless you have a really, really good memory, it’s a good idea to keep the old bits of paper handy.

# Paper Scissors Stone

# I’m sure you’ve played Paper, Scissors, Stone before. There are two players. You make your hand into the shape of either paper, scissors or stone. Both players do it at exactly the same time so there’s no cheating. To decide the winner you have to remember



**Stone blunts scissors**

**Paper wraps stone**

**Scissors cut paper**

We can write this game so you can play against the computer. Type this in

puts "Press Enter to play"

enter = gets.chomp

number = rand(3)

if (number == 0)

 puts "Paper"

elsif (number == 1)

 puts "Scissors"

elsif (number == 2)

 puts "Stone"

end

Now run the program. Make your hand into the shape of paper, scissors or stone then quickly press Enter to see what Ruby chooses. Who won?

How does the program work? The clever bit is how Ruby decides between paper, scissors and stone. See this line?

number = rand(3)

This tells Ruby to make a random choice between 3 things. It will put the number 0, 1 or 2 into our **number** variable. The rest of the code uses an **if** to print Paper for number 0, Scissors for number 1 or Stone for number 2.

Paper, Scissors, Stone is more fun if you play it lots of times, which is where looping comes in. We can make Ruby play the game 10 times. Change your program so it looks like this:

1.upto(10) do

 puts "Press Enter to play"

 enter = gets

 number = rand(3)

 if (number == 0)

 puts "Paper"

 elsif (number == 1)

 puts "Scissors"

 elsif (number == 2)

 puts "Stone"

 end

end

puts "Game over"

The line

1.upto(10) do

tells Ruby to run the code 10 times. You need an

end

as well, so Ruby knows exactly what code to run 10 times. The stuff between the **upto** and the **end** should be shifted in a bit to make the code easier to understand.

After 10 goes the loop will finish and Ruby prints

Game over

## *Challenge – Get Ruby to tell you who won*

Instead of pressing enter to see Ruby’s choice, change the program so you type in p for paper, s for scissors or t for stone (you can’t use s twice). When you’ve done that you can get Ruby to figure out who wins. Don’t forget that you can draw if you both players choose the same thing.

When you run your program it should look something like this.

Paper, scissors or stone (p/s/t)?

p

Ruby chooses stone.

Paper wraps stone, you win.

Paper, scissors or stone (p/s/t)?

s

Ruby chooses paper.

Scissors cut paper, you win.

Paper, scissors or stone (p/s/t)?

t

Ruby chooses paper.

Paper wraps stone, Ruby wins.

Paper, scissors or stone (p/s/t)?

p

Ruby chooses paper.

It’s a draw.

## *Challenge – Keep score*

Make your program keep track of how many games you win and how many Ruby wins. At the end, just after the Game Over message, print out who won the most.

Game over

You won 3

Ruby won 4

Ruby wins!

# Guess the number

Let’s write another game. Ruby will think up a number from 1 to 5, then you have to guess it.

Type this in and run it a few times.

number = rand(5) + 1

puts "What is your guess (1-5)?"

guess = gets.to\_i

if (guess == number)

 puts "Right!"

else

 puts "Wrong"

end

This is how Ruby thinks up a number

number = rand(5) + 1

When Ruby sees **rand(5)** it will think up a number between 0 and 4. To make it between 1 and 5 rather than 0 and 4, we add 1 to it. The rest of the code is stuff we’ve done before – there’s a variable called **guess** and an **if** to see if we’ve guessed right.

# Lots of guesses

Now we’re going to change the game so it keeps asking until you get it right. Change your code so it looks like this.

number = rand(5)

while (true)

 puts "What is your guess (0-4)?"

 guess = gets.to\_i

 if (guess == number)

 puts "Right!"

 break

 else

 puts "Wrong"

 end

end

Do you see this line?

while (true)

This tells Ruby to keep doing the same thing again and again. Just like with **upto**, you need an **end** so Ruby knows what it should be doing again and again. But **while** is not the same as **upto**. **Upto** will loop exactly as many times as you tell it to, but **while(true)** will keep going forever. You can get it to stop looping though, using the **break** command

break

Break tells Ruby to stop doing things again and again. In our game we break once you’ve guessed right, since that’s the end of the game.



## *Challenge – Higher or Lower?*

Change the game so after you guess the program gives you a clue. If you guess too low it says “Higher”, and if you guess too high it says “Lower”. This will make the game too easy, so make Ruby think up a number between 1 and 100 instead.

## *Challenge – How many guesses*

## Make your game count how many guesses it takes you to get the right number, then print that out at the end. Your program might print something like this.

What is your guess (1-100)?

40

Lower

20

Higher

30

Higher

33

Right!

It took you 4 guesses.

# Card Games

Now we know how to ask Ruby to think up a number, we can write card games. We can ask Ruby to think up a number from 1 to 10. 1 is the Ace, then you have 2 all the way up to 10. We’ll ignore the Jack, Queen and King for now as they’re a bit more difficult.

## *Challenge – Pick a card*

Can you write a program where Ruby thinks up a card from Ace to 10? It should print the card to the screen. 1 counts as an Ace. Write your program so the card it picks is always a heart. When you run your program it should look something like this.

Your card is the 4 of hearts

Of course it might not be the four of hearts when you run it. Run it lots of times and make sure the card is always between 1 and 10.

# Twist or Bust

Have you played Twist or Bust? Some people call it Pontoon. Here are the rules:

There are two players. On each turn the players can choose to Twist (which means you take another card) or Stick (which means you don’t). The aim is to get your cards to add up to something close to 21, but not go over it. If you go over you shout “Bust!” and the other player wins.



Here is a pretty good hand. It adds up to 19. The only hands better than this are ones adding up to 20 or 21. 21 is the best hand you can have.

I’m going to show you how to write the program for one player, then ask you to make it work for two.

player1hand = 0

while (true)

 puts "Player 1 - twist or stick (t/s)?"

 choice = gets.chomp

 if (choice == "s")

 puts "You stick on " + player1hand.to\_s

 break

 else

 card = rand(10) + 1

 puts "Your card is the " + card.to\_s + " of hearts"

 player1hand = player1hand + card

 puts "Your hand now adds up to " + player1hand.to\_s

 if (player1hand > 21)

 puts "Bust!"

 break

 end

 end

end

Can you see how this works? Read it through to make sure you do.

## *Challenge – Now make it work for two players*

Add a second player to the program. The players should take it in turns to twist or stick. Your program should tell you who wins.

Remember, the game will end when one of the players busts, or when both have stuck. If both have stuck you need to work out who’s closest to 21 to see who wins. There might even be a draw.

# What other games can you write?

You’ve learnt a lot of Ruby now, enough to write lots of games. With **loops** and **random** numbers and **variables** and **ifs** you can do loads of exciting stuff.



You can write dice games. Get Ruby to think of a number between 1 and 6 for the dice throws. Google **simple dice games** to get some ideas.



You can write spinning wheel games, like Twister. Ruby can think of a number for as many slots as you have on the wheel. I wrote one of these when I was a kid and I had friends round to stay. We didn’t have enough beds, so the wheel would tell us who could sleep in the bed, who slept on the sofa, and who was on the floor!



You can write a conker game (the random number would be how much damage your conker takes, then when it takes too much damage your conker smashes to pieces and you lose).

You can make fruit machines, coin games, anything at all. When I was ten I got a computer for Christmas and I started writing games. By the time I left school I had written hundreds and hundreds of them. The more you practice the better your games will be. You might even be able to do it as a job when you grow up, then you’ll get paid for writing games!



