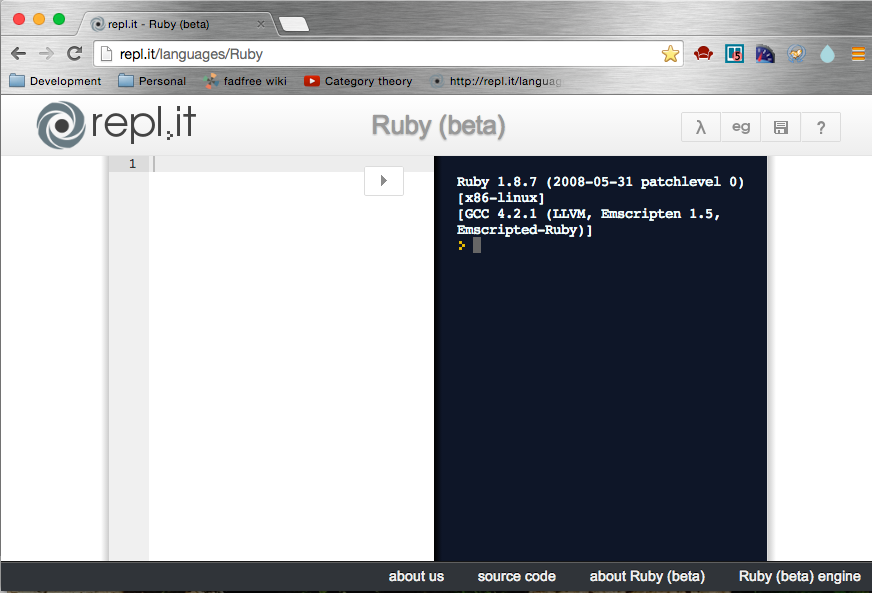
# Variables

# Running Ruby

Open <http://repl.it/languages/Ruby>. You will see this…



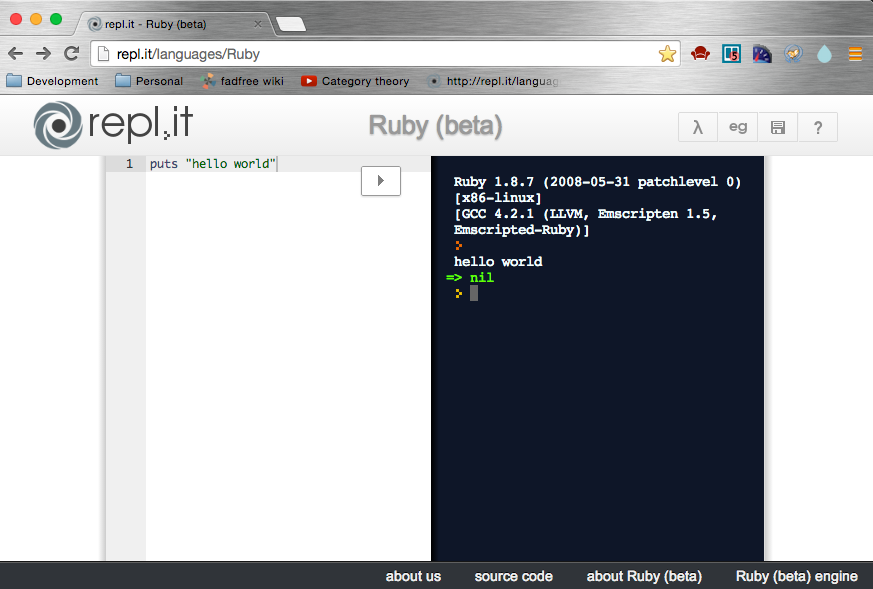
You can type your Ruby code in the left hand side and then click the  button to run it.



Try it now. Type

puts “hello world”

in the left hand side then click . You should see this…



See how **hello world** appears on the right? That’s your program running. So what is the green => nil all about? There is an answer, but it’s complicated, confusing and I’m not going to tell you. You can ask one of the class tutors if you like, but you will regret asking!

# Using a variable

Modify your program to this

greeting = “hello world”

puts greeting

You see the same result

hello world

So what’s the point of a variable? Well it has lots of uses. You can use it to read stuff that people type in.

puts "What is your name?"

name = gets.chomp

puts "hello " + name

Try running this program. It will ask your name then say hello back to you. We use the variable **name** to store your name, then print it back out after the word **hello**. Notice the **space** after the word **hello**. Without this it would print out **helloWilliam**, which is rubbish. The space turns it into **hello William**, or whatever your name is.

# Variable names

Ruby doesn’t care what you call your variables. Try changing your variable name and it will do exactly the same thing. Don’t forget to change it in both places!

puts "What is your name?"

silly = gets.chomp

puts "hello " + silly

# 

# When it goes wrong

What happens if you make a mistake and only change the variable name in one place? Try it.

puts "What is your name?"

name = gets.chomp

puts "hello " + silly

What is your name?

William

(eval):2: undefined local variable or method `silly' for main:Object (NameError)

Error messages in Ruby can be difficult to understand. You’ll see them a lot though, so you will have to read them to figure out what’s gone wrong. In this one we can see it’s telling us we have an undefined variable called **silly**, which is enough to make us look at our code, spot the stupid mistake and fix it. Once the variable names match, our program will work and all is well with the world once again.

# Saying Hello

Variables are called variables because they can ‘vary’ - you can change their values. Try this

puts "What is your first name?"

name = gets.chomp

puts "hello " + name

puts "What is your last name?"

name = name + gets.chomp

puts "hello " + name

What is your first name?

William

hello William

What is your last name?

Gates

hello WilliamGates

You see here we are greeting William twice, both times with the same line

puts "hello " + name

The first time **name** is just William, and the second time **name** is WilliamGates, so exactly the same line of Ruby prints two different things! The magic happened on this line, which changes the value of **name**

name = name + gets.chomp

This says, change the value of **name** to what it was before, plus something extra on the end, in this case his last name. But there is a mistake here…

## *Challenge – Fix the mistake*

Instead of

hello WilliamGates

it should be

hello William Gates

Can you fix it? Have a go. For a clue, look at the experiments below. Show your answer to one of the tutors.

Try some more experiments with variables. Here are some examples. Type them in and see what they do. Try some of your own.

day = "Saturday"

puts "Today is " + day

puts day + " comes once a week"

puts "Every " + day + " is fun"

puts "This " + day + " is just like last " + day

puts "What is your favourite place?"

place = gets.chomp

puts "On " + day + " we will go to the " + place

## *Challenge – Write a story maker*

**Using variables and **gets.chomp**, write a Ruby program that asks some questions then writes a story containing the answers. Your story can be about anything at all - shopping, school, aliens, anything! Grab one of the tutors and show them, they like this kind of stuff.

# Numbers

Up to now we have been writing Ruby that reads in and prints out words (or in Ruby-speak, **Strings**), but Ruby knows about numbers too, and Ruby can do maths. Type in these commands and see what Ruby does.

puts 2 + 2

puts 10 – 6

puts 3 \* 7

puts 100 / 5

puts 2 \* 5 + 7 - 1

You can read numbers that people type in too

puts "Enter a number"

number = gets.to\_i

puts "Twice that number is "

puts number \* 2

Enter a number

3

Twice that number is

6

## *Challenge – How old will you be in 5 years?*

Write some Ruby that asks your age, then tells you how old you will be in 5 year’s time. When run, the program should look something like this

How old are you?

9

In 5 years you will be

14

## 

## *Bonus Challenge – How old will you be in the year 2050?*

This program will go something like this

How old are you?

9

In 2050 you will be

45

## *Super Special Bonus Challenge – How old will you be in any year?*

This program will ask you to type in two things – your age and a year. Then it will do some maths to work out how old you will be in that year. It should look something like this

How old are you?

9

Enter a year

2090

By then you will be

85

**

Which is pretty old, but you’ll probably still be around in 2090, which is impressive.



# Strings and numbers don’t play nicely together

We’re going to write three simple Ruby programs, two of which make sense, and one of which will worry you. Here goes…

puts "hello " + "Paul"

hello Paul

No problem. Second program…

puts 2 + 2

4

Easy. Third program…

puts "Paul is " + 8

(eval):27: (eval):27:in `+': can't convert Fixnum into String (TypeError)

from (eval):27

What on earth just happened there? Well, you can add strings to strings and numbers to numbers, but if you try to add a string and a number, bad things happen. Strings and numbers just don’t play nicely together.

You can fix it though. The way to do it is to ask the number to turn itself into a string, then it will play nicely with the other string. How do you tell a number to turn itself into a string? You put **.to\_s** on the end of it (**to\_s** means turn into a string).

Type in this Ruby and run it.

puts "Paul is " + 8.to\_s

Paul is 8

But why bother? All this .to\_s stuff seems like a lot of unnecessary effort. Why not just make the 8 a string and type in

puts “Paul is “ + “8”

Or better still

puts “Paul is 8”

Well, often you can indeed just use 8, or any other number, as a string. Put it in quotation marks and it becomes a string, it will play nicely with other strings. But sometimes you want to do some maths, and maths only works with numbers, so when we do some maths and want to put the result in a string, we have to use **.to\_s**.

## *Challenge – Guess what happens*

What do you think will be the result of this program?

puts 2 + 2

How about this one?

puts "2" + "2"

Run them and find out if you were right. Was it what you expected?

In the second program Ruby sees the quotation marks and decides that these “2”s are strings, and it knows that adding strings together means putting one after the other.

## 

## *Challenge Revisited – How old will you be in 5 years time?*

In the first version of this challenge your Ruby program had to write something like this…

How old are you?

9

In 5 years you will be

14

But now I want it to look a little bit different…

How old are you?

9

In 5 years you will be 14 years old

Can you do it? Show a tutor if you can. They’ll be impressed.

As well as converting numbers into strings, there’s another trick Ruby can do – it can go the other way and convert strings into numbers. Why would you want to do that? Well, it’s because when we read in something from the keyboard with **gets** it always comes in as a string. If you want to do some maths, you have to turn it into a number first. Remember this bit of Ruby we typed in earlier?

puts "Enter a number"

number = gets.to\_i

puts "Twice that number is "

puts number \* 2

Enter a number

3

Twice that number is

6

See this line?

number = gets.to\_i

That tells Ruby to read a string that the user has typed in, turn it into a number then put it into the variable called “number” because we want to do some maths with it later.

## Final Challenge – Write another story maker that does some maths

To finish off, write another story maker that asks a bunch of questions and writes out a story. Some of the questions you ask should have number answers, and you should do some maths with them. Here’s an example story to get you going…

Ask the person’s name, how many brothers and sisters they have, where they live, the name of a teacher at school and the lesson they give. Then write out a story that looks like this.

*Sarah and her 2 brothers and sisters left their house in Camden and walked to the junk yard where they found a space ship. All 3 of them got inside and flew to Mars, where the aliens all looked a bit like Mr Heathwood. They caught one of the aliens and took him back to Earth, where he now gives Chemistry lessons to 2nd years who don’t understand why he is green and has 4 arms.*

Notice the maths here is the 3, which is the number of brothers and sisters you have + 1 (the one being you).

