Hello World

Welcome to the Ruby tutorial.

Computer programs consist of many machine instructions. Nobody has time to write machine instructions, so programming languages have been designed. One of these languages is Ruby.

Quick Start

Create the hello world app

We create a program that outputs “Hello, World” to the screen.

Copy the code below and save the file as hello.rb

```
1
2 #!/usr/bin/ruby -w
3 puts "Hello, World!";
```

Then run the program with:

```
1 ruby hello.rb
```

If the program runs fine, you’ll see this output:

Exercises

1. Create a program that outputs your name
2. Create a program that outputs your address (multiple lines).

Strings

Ruby strings could be understood as a group of characters. A string could be of length 1 (one character), but its usually longer. A string is always defined in double qoutes.

This means a *string* variable in ruby could hold *text*: words, sentences, books

Programming languages have variables. These variables have a *data type*
A variable could be of the data type string.

String example

String variable

In the example below we use ruby to print text.
First define a string variable, then print the variable.

```
#!/usr/bin/ruby
str = String.new("This is a string example")
puts "#{str}"
```

Save as strings-example.rb. Then run with

```
ruby strings-example.rb
```

Multiple lines

Generally there are 2 ways to print multiple lines.

Method 1. call display function x times,
Method 2. use the newline character inside the string

Both of these are common in programming.

```
#!/usr/bin/ruby
# method 1: multiple calls
puts "Hello"
puts "World"
# method 2: newline characters
puts "Hello World"
```

Exercises

1. Create a program with multiple string variables
2. Create a program that holds your name in a string.

Keyboard input

Ruby can read keyboard input from the console. In this section you will learn how to do that..

To get keyboard input, first open a console to run your program in. Then it will ask for keyboard input and display whatever you've typed.

Keyboard input in ruby

Example

The ruby program below gets keyboard input and saves it into a string variable. The string variable is then shown to the screen.

```
#!/usr/bin/ruby
print "Enter city name: "
city = gets
puts "You live in #{city}"
```

The data that we read from the console (keyboard), is stored in a variable and printed to the screen.

```
ruby keyboard-example.rb
```

Sample output of program.

```
Enter city name: Sydney
You live in Sydney
```

This line will get keyboard input and store it in a variable:

```
city = gets
```

You can get as many input variables as you want, simply by duplicating this line and changing the variable names.

Exercises

1. Make a program that lets the user input a name
2. Get a number from the console and check if it's between 1 and 10.

Variables

Variables often hold text or numeric data. In ruby there are several types of variables, including strings and numeric variables.

Variables can be reused in your code. Arithmetic operations can be used on numeric variables.

String variables can also be changed (sub-strings, concatenation).

Variables in ruby

Numeric variables

Lets start with numeric variables. We create a program that calculates
the VAT for a given price.

Define a series of products, sum the price ex. VAT, then calculate the VAT and add it to the price.

Copy the code below and save the file as variables.rb

```
1
2
3  #!/usr/bin/ruby
4  pear = 3
5  coffee = 1
6  potato = 2
7  puts ""
8  total = pear + coffee + potato
9  puts "Price:            #{total}"
10 vat = total * 0.15
11 puts "VAT:              #{vat}"
12 price = total + vat
13 puts "Price (+VAT):     #{price}"
14 puts ""
15
```

All arithmetic operations can be run on variables: division (/), substraction (-), addition (+) and multiplication (*)

Run with:

```
1 ruby variables.rb
```

Exercises

1. Calculate the year given the date of birth and age
2. Create a program that calculates the average weight of 5 people.

Comments

ruby supports comments. A comment is text that is ignored on execution.
but may be useful to the programmer. You can add any kind of text inside
your code.

ruby ignores comments. Comments can be single line or multi line.

Quick Start

Comments in ruby

Write some code. In this example we output some text.
Then add single and multi line comments.

Copy the code below and save the file as hello.rb

```
1
2  #!/usr/bin/ruby -w
3  # This is a single line comment.
4  puts "Ruby code!"
5  =begin
6  This is a multi line comment
7  Yes, you can type multiple
8  lines of comment.
9  =end
10 puts "Ruby code!"
11
```

Execute with:

```
1 ruby example5.rb
```

Exercises

1. Create a program and add a comment with your name
2. When would you use multiline comments over single line?
3. Can you write code and comments on the same line?

Arrays

ruby arrays can hold multiple values. The minimum elements of an array is zero, but they usually have two or more. Each element in an array has a unique index.

The index starts at zero (0). That means to access the first element of an array, you need to use the zeroth index.

Arrays in ruby

Example

The program below is an example of an array loop in ruby.
The array we define has several elements. We print the first item with the zeroth index.

```
1
2 #!/usr/bin/ruby
3 a = Array[1, 2, 3, 4,5]
4 puts "First element: #{a[0]}"
5 puts "Second element: #{a[1]}"
```

Upon running this program it will output the first (1) and second (2) element of the array. They are referenced by the zeroth and first index. In short: computers start counting from 0.

```
1 ruby example6.rb
```

This program will output:

```
1 First element: 1
2 Second element: 2
```

The index should not be larger than the array, that could throw an error or unexpected results.

Exercises

1. Create an array with the number 0 to 10
2. Create an array of strings with names

For loops

ruby can repeat a code block with a *for loop*. All for loops have a condition, this can be the amount of times or a list.

You need loops to repeat code: instead of repeating the instructions over and over, simply tell ruby to do it n times.

For loops in ruby

Example

The program below is an example of a for loop in ruby. The for loop is used to repeat the code block.

Ruby will jump out of the code block once the condition is true, but it won't end the program.

The code block can contain anything, from statements to function calls.

```
1
2 #!/usr/bin/ruby
3 for i in 0..4
4   puts "Iteration #{i}"
5 end
```

The code block can be as many lines as you want, in this example its just one line of code that gets repeated.

ruby runs the code block only n times. The number of repetitions is called *iterations* and every round is called an iteration.

To start save it as forloops.rb, then type this command:

```
1 ruby forloops.rb
```

Exercises

1. Can for loops exist inside for loops?
2. Make a program that counts from 1 to 10.

If statements

Ruby can make choose based on data. This is very useful, as programs shouldn’t always do the same. The concept is known as *if statements.*

This data (variables) are used with a condition: if statements start with a condition. A condition may be (x > 3), (y < 4), (weather = rain).

What do you need these conditions for? Only if a *condition* is true, code is executed.

If statements are present in your everyday life, some examples:

- **if (elevator door is closed), move up or down.**
- **if (press tv button), next channel**

If statements in ruby

Example

The program below is an example of an if statement.

```
1
2 #!/usr/bin/ruby
3 x = 3
4 if x > 2
5   puts "x greater than 2"
6   puts "executing this code block (puts)"
7 end
```

ruby runs the code block only if the condition (x >2) is true. If you change variable x to any number lower than two, its codeblock is not executed.

Save as example.rb then run with:

```
1 ruby example.rb
```

Else

You can execute a codeblock if a condition is not true

```
1
2  #!/usr/bin/ruby
3  x = 1
4  if x > 2
5    puts "x greater than 2"
6    puts "executing lines of code (puts)"
7  else
8    puts "other codeblock now"
9    puts "condition (x > 2) is not true"
10 end
```

Save as example2.rb, run with

```
1 ruby example2.rb
```

Change the variable x and run it several times.

Exercises

1. Make a program that divides x by 2 if it’s greater than 0
2. Find out if if-statements can be used inside if-statements.

While loops

ruby can repeat a code block with a *while loop.* The while loop repeats code until a condition is true.

While loops are used when you are not sure how long code should be repeated.

A practical example: Think of a tv that should continue its function until a user presses the off button. Video game that should keep running until the user stops.

While loops in ruby

Example

The program below is an example of a while loop in ruby.
It will repeat until a condition is true, which could be forever.

The code block can contain anything, from statements to function calls.

```
1
2 #!/usr/bin/ruby
3 $i = 1
4 $max = 5
5 while $i < $max  do
6   puts("Loop iteration #$i" )
7   $i +=1
8 end
9
```

In the example it repeats the code block until variable i is greater than max.
You must always increment the iterator (i), otherwise the while loop repeats forever.

The code block can be as many lines as you want, in this example its just one line of code that gets repeated.

Save as example8.rb. Then start with:

```
1 ruby example8.rb
```

Exercises

1. How does a while loop differ from a for loop?

File exists

There is a ruby function that checks if a file exists. This function returns true if the file exists.

Why? If you try to open a file that doesn't exist, at best it will return an empty string and at worst it will crash your program. That would lead to unexpected results and thus you want to check if the file exists.

File exists in ruby

Example

The following ruby code will check if the specified file exists or not.

```
1
2 #!/usr/bin/ruby
3 #Ruby function to check if file existensts
4 if(File.exist?('example.txt'))
5   puts 'File exists'
6 else
7   puts 'File does not exist'
8 end
```

If no file root is specified, it will look for the file in the same directory as the code.
If the file exists, it will return true. If not, it will return false.

Error checking

Sometimes you want to check if a file exists before continuing the program. This leads to clean code: first check for errors, if no errors continue.

```
1
2 #!/usr/bin/ruby
3 #Ruby function to check if file existensts
4 if(!File.exist?('example.txt'))
5   puts 'File does not exist'
6 end
7 puts 'Ruby'
8
```

Exercises

1. Check if a file exists on your local disk
2. Can you check if a file exists on an external disk?

Read file

ruby can be used to read files. You can either read a file directly into a string variable or read a file line by line.

These functionality ruby provides out of the box is for read files on the hard disk, not on the cloud.

Read files in ruby

Read file

The ruby program below reads a file from the disk. ruby will read the file from the same directory as your program. If the file is in another directory, specify its path.

If you want to read a file at once, you can use:

```
1
2 #!/usr/bin/ruby
3 content = File.read("read.rb")
4 puts content
```

This reads the entire file into a ruby string.

Line by line

If you want to read a file *line by line*, into an array, you can use this code:

```
1
2 #!/usr/bin/ruby
3 lines = File.readlines('read2.rb')
4 puts lines
5 # It's an array so we can access elements
6 puts lines[0]
7 puts lines[1]
8
```

Exercises

1. Think of when you'd read a file 'line by line' vs 'at once'?
2. Create a new file containing names and read it into an array

Write file

ruby can open a file for reading and writing (r+) or writing (w+). This is for files on the hard disk, not on the cloud.

In this article we will cover how to write files in ruby. If you are interested in reading files, see tthe read-file article instead.

Write files in ruby

Write file

The ruby program below writes a file from the disk. If the file exists, it will be overwritten (w+). If you want to add to end of the file instead, use (a+).

```
#!/usr/bin/ruby
myFile = File.new("file.txt", "w+")
if myFile
   myFile.syswrite("Ruby example of file writing.
")
else
   puts "Unable to open file!"
end
```

This writes the ruby string into the newly created file. If you get an error, it means you don’t have the right user permissions to write a file (no write access) or that the disk is full.

Otherwise the new file has been created (file.txt). This file contains the string contents which you can see with any text editor.

Flags

If you use the w+ flag the file will be created if it doesn’t exist. The w+ flag makes ruby overwrite the file if it already exists.

The r+ does the same, but ruby then allows you to read files. Reading files is not allowed with the w+ flag.

If you want to append to a file (add) you can use the a+ flag. This will not overwrite the file, only append the end of the file.

Exercises

1. Write a list of cities to a new file.

rename file

Rename files with ruby. Once you have a file in a directory you can simply rename it from your code.

The file will be renamed in the same directory. If you want to rename and move it to a new directory, change the variable to_file.

Rename file in ruby

Example

The program below renames an existing file. Make sure the file exists before running the file. You can simply create an empty file.

```
1
2  #!/usr/bin/ruby -w
3  # get current directory
4  folder_path = Dir.pwd
5  filename = "hello.mp3"
6  # source and destination file
7  from_file = folder_path + "/" + filename
8  new_file = from_file + ".txt"
9  # rename file
10 File.rename(from_file, new_file)
11
12
```

The file defined by the variable from_file will be renamed to the filename new_file.

Run program with the command:

```
1 ruby rename.rb
```

The file will be renamed to a new file.

Rename in shell

Now there are other ways to do this, for example on a Linux or Mac OS X system you can run the command

```
1 mv source.txt destination.txt
```

But this may or may not work on other platforms. Thats why you should always use the modules provides by the programming language.

Struct

A struct can bundle attributes together. If you create a struct, you can set a couple of variables for that struct. Those variables can be of any datatype.

A key difference from an array is that elements of an array are all of the same datatype. That is not the case with a struct.

If you want to combine variables, structs are the way to go. Unlike the concept of object oriented programming, they are just data holders.

Struct in ruby

Struct example

The ruby example creates a new struct. Then it sets the variables.

```
1
2  Person = Struct.new(:name, :job) do
3    def show
4      puts "My name is #{name} and I'm a #{job}!"
5    end
6  end
7  p1 = Person.new("Albert", "Professor")
8  puts p1.name
9  puts p1.job
10 p1.show
11
```

The program bundles the variables (job, name) into a struct named Person.Then that structure can be used to set variables.

In this example the struct has only two variables, but a struct can have as many as you need.

You can create multiple items with the same struct, all with different values. Elements of a struct can be accessed immediately.

Exercises

1. Create a struct house with variables noRooms, price and city
2. How does a struct differ from a class?

Random numbers

ruby can generate random numbers. A random number is unknown before running: it's like telling the computer, give me any number.

Random numbers in computing are not truely random, they are often based on a pseudo random number generator algorithm. Eitherway for most program that degree of randomness is enough.
In this article you will learn how to generate random numbers.

Random number in ruby

Example

The ruby program below generates a number between 0 and 10. The starting number (0) is not given and thus 0 is assumed as lowest number.

```
1
2 #!/usr/bin/ruby
3 r = Random.new
4 x = r.rand(10)
5 print("Random number: #{x}\n")
6
```

To generate a number between 20 and 40 you can use the code below:

```
1
2 #!/usr/bin/ruby
3 r = Random.new
4 x = 20 + r.rand(20)
5 print("Random number: #{x}\n")
6
```

Exercises

1. Make a program that rolls a dice (1 to 6)
2. Can you generate negative numbers?

Methods

ruby methods are reusable code blocks. By calling a ruby method, all of the code in the method will be executed.

Methods should start with a lowercase character and only contain alphabetic characters.
A method can take one or more parameters which can be used in the code block.

Methods in ruby

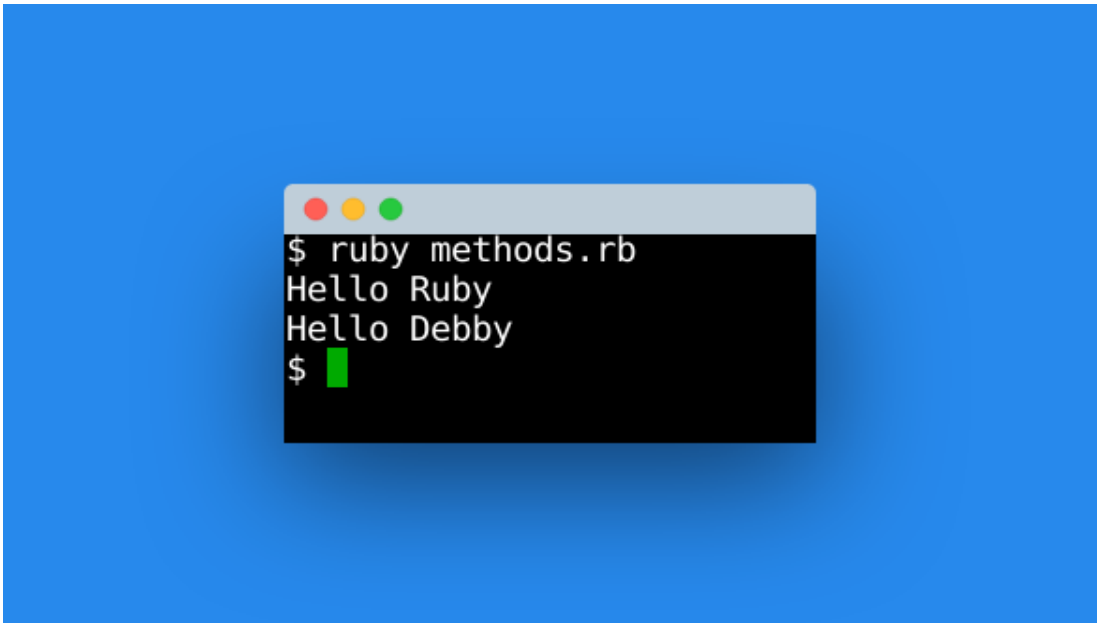
Example

The method below can be called as many times as you want: a method is reusable code.
Methods can also return output, this output can then be used in the program.

```
1
2 #!/usr/bin/ruby
3 def hello(x1 = "Debby")
4   puts "Hello #{x1}"
5 end
6 hello "Ruby"
7 hello
```

The method hello above is called with a parameter and without.
Sometimes parameters are necessary for your codeblock, but at times they are not.

The parameter in this example is x1, which is given a value outside the method.



Return value

A value inside a ruby method only exists there (local scope). It can be given to the program with the return statement, return x1. That then needs to be saved in an output variable.

```
1
2 #!/usr/bin/ruby
3 def multiply(x1 = 0, x2 = 0)
4   x3 = x1 * x2
5   return x3
6 end
7 val = multiply 2, 3
8 puts val
9
```

Exercises

1. Create a method that sums two numbers
2. Create a method that calls another method.

Slices

ruby slices are subsets. A slice can be a subset of an array, list or string. You can take multiple slices out of a string, each as a new variable.

A slice is never longer than then the original variable. This makes sense, if you take a slice of a pizza you don't suddenly have two pizzas. In programming it's similar.

Example

The ruby code will take a slice out of a list of numbers.

Start by creating a list of numbers (myset). Then take a slice by specifying the lower and upper bounds. In programming languages the lower bounds is zero (arrays start at 0).

```
1
2
3
4
5
6
7
8
9
10
```

```
#!/usr/bin/ruby
# define set
myset = Array[0,1,2,3,4,5,6,7,8,9]
# take slice
s = myset[0..3]
# output
puts s.join(' ')
```

The above program takes a slice and outputs it.

```
1 [0 1 2 3]
```

String slicing

Ruby strings can be sliced too. The resulting slice will then also be a string, but of smaller size. When slicing, remember that the index 0 is the first character of the string. The last character is the numbers of characters minus 1.

```
#!/usr/bin/ruby
# create string
myStr = String.new("Ruby programming language")
# take slice
s = myStr[0..4]
# output
puts s
```

Exercises

1. Take the string 'hello world' and slice it in two.
2. Can you take a slice of a slice?

Date and time

ruby can display date and time. In this article you will learn how to deal with date and time in ruby.

Date is default, but time cannot go back earlier than 1970. Why? that's when the logic was added to computers.

Date and time in ruby

Example

The program below is an example of date and time in ruby. The formatting is explicitly defined (%y for year, %m for month etc).

```
1
2
3  #!/usr/bin/ruby -w
4  time = Time.new
5  print "Date : "
6  puts time.strftime("%Y-%m-%d")
7  print "Time : "
8  puts time.strftime("%H:%M:%S")
9  # others:
10 # %A   full weekday name (Monday)
11 # %a   short weekday name (Mon)
12 # %b   short month name (Jan)
13 # %B   full month name (January)
14 # %j   day of the jear (001.. 366)
15 # %U   week number, from sun
16 # %W   week number of current year, from mon
17 # %w   day of the week
18
19
```

The code above displays the date and time. You can use an alternative formatting if you want..
The output will be similar to this:

```
1 Date : 2018-06-07
2 Time : 10:38:01
```

Exercises

1. Display date in DD/MM/YYYY format

Class

ruby class can be used to create objects. Classes are the cornerstone of object orientated programming (oop).

A ruby program can have multiple objects. An object is essentially a collection of methods and variables. Each object has its own values, but the methods and variable names derrive from the class definition.

Classes in ruby

Class example

This concept can be challenging to grasp, and is easier to understand with an example.
The ruby program below creates an object from the defined class.

```
1
2
3  #!/usr/bin/ruby
4  class Cube
5    def initialize(length)
6      @length = length
7    end
8
9    def showLength
10     puts "Size is #@length x #@length x #@length"
11   end
12 end
13 # create object
14 object1 = Cube.new(3)
15 object1.showLength
16 # create another object
17 object2 = Cube.new(4)
18 object2.showLength
19
20
```

The above program defines a class (cube) which has a property and a method. Then two objects are created.

The attributes of the objects are set, and shown using the class method.
The output should be similar to:

```
1 Cube is 3x3x3
2 Cube is 4x4x4
```

Exercises

1. Create two objects from a class.
2. Can an object create a class?

Ruby Open Classes

An existing Ruby class is never closed. You can constantly add methods to an existing class. This applies to all classes.

This can be done on classes you defined, but also on standard classes. Meaning you can add functionality to existing Ruby classes also.

Example

Ruby open class

First define a class and create an object.

```
1
2 class Dog
3   def bark
4     puts 'Woof! Woof!'
5   end
6 end
7 # make an object
8 d = Dog.new()
9 d.bark
```

How? Open up a class definition for an existing class, add the new content.

```
1
2  class Dog
3    def bark
4      puts 'Woof! Woof!'
5    end
6  end
7  class Dog
8     def spin
9       puts 'spins'
10    end
11 end
12 # make an object
13 d = Dog.new()
14 d.bark
15 d.spin
16
```

The class is not redefined here. Instead it’s reopened, and the method is attached to it.

Overwrite methods

You can also overwrite an existing method. The latest method will always be the one used.

Using Ruby open classes can create strange bugs. In general you should always update the existing classes.

Ruby Include File

Incude Ruby files instead of all of your code in a one file. Single file hold all of your code just won't do, except for small programs. Split file or files.

To prevent chaos, you can include ruby files and organize. If your program is object orientated, you can create a file for every class.

Example

Load

The load method includes a Ruby file, each time a method is executed:

```
1 load 'filename.rb'
```

To be clear: the load method add a ruby file into your code, every time a script is executed. It does not work like a module or library.

Require

The require methods loads the given file only once.

```
1 require 'filename'
```

If you want to load a module, use **require()**.
To execute code, use **load()**.

Require doesn't use the rb extension (filename.rb), because not all extensions end with .rb.

Example

Create two files in the same directory, one named example.py

```
1
2 require './hello'
3 hello()
```

and another file named hello.rb

```
1 def hello
2   puts "Hello"
3 end
```

When executing, it will load hello.rb and call its method.

Overloading Methods

A ruby class can have only one method with the same name. Ruby overrule method if they are defined again.

The number of arguments in a method can be variable. How?

Example

Method overloading

Create a new object, then you can call a method with 2 or 3 arguments.

```
1 a = Example.new
2 a.hello('Thomas')
3 a.hello('Thomas','Edison')
```

This works because internally it checks the number of arguments with an if statement.

Method overloading is defined inside the class definition.

In the method hello, pass the arguments (*args). Then validate with an if statement.

```
1
2  class Example
3    def hello(*args)
4      if args.size < 1 || args.size > 2
5        raise 'Method takes 1 or 2 arguments'
6      else
7        if args.size == 1
8          puts "Hello #{args[0]}"
9        else
10         puts "Hello #{args[0]} #{args[1]}"
11       end
12     end
13   end
14 end
15 a = Example.new
16 a.hello('Thomas')
17 a.hello('Thomas','Edison')
```

Initializer overloading

The initializer is called when you create a new object.
Method overloading also works for the initializer:

```
1 a = Shape.new(2,4)
2 b = Shape.new(1,2,3)
```

To achive that you use an if statement inside the method:

```
1
2  class Shape
     def initialize(*args)
```

```
3
4       if args.size < 2 || args.size > 3
5         puts 'Method takes 2 or 3 arguments'
6       else
7         if args.size == 2
8           puts '2 arguments'
9         else
10          puts '3 arguments'
11        end
12      end
13    end
14  end
15  a = Shape.new(2,4)
16  b = Shape.new(1,2,3)
```

Method Overriding

Method overriding is a language feature. Its part of the object-oriented programming concept.

If you have two classes, the implementation of a sub class replaces (overrides) the implementation of the super class. Simplified, you can call it “method replacement”.

Method overriding in ruby

Overriding example

The ruby program below creates defines two classes with the same method. Class SUB inherits from the SUPER class.

```
1
2
3  #!/usr/bin/ruby
4  class SUPER
5    def hello
6        puts 'In class SUPER'
7    end
8  end
9  class SUB < SUPER
10   def hello
11       puts 'In class SUB'
12   end
13 end
14 b = SUB.new()
15 b.hello()
16
```

Class SUB replaces the implementation of the method hello. Once a new object is created, that implementation is used. If the method is not defined in the sub class, it will use the implementation of the super class.

Redefining methods

Methods can be redefined. It will always use the latest method. It’s recommended that you only use one method definition.

```
1  class EXAMPLE
2    def hello
3        puts 'In class Example'
4    end
5    def hello
6        puts 'hello'
7    end
8  end
9  b = EXAMPLE.new()
10 b.hello()
```

Inheritance

A class can have an inheritance relation with a super class. The sub class inherits all the attributes and methods from the super class.

If an object is created using a sub class, it has the methods and attributes of both the sub class and the super class. You can have multiple sub classes that inherit from the same super class.

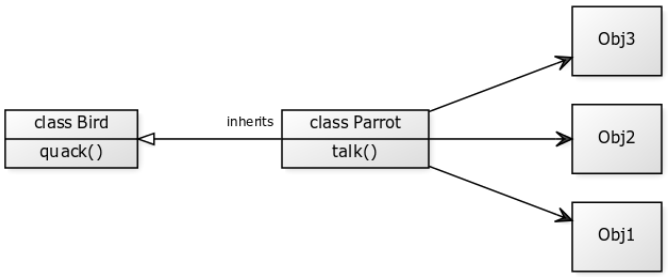
Inheritance in ruby

Inheritance example

The ruby program below creates an object with the defined sub class.

```
1
2
3  #!/usr/bin/ruby
4  class Bird
5    def quack
6      puts "Quack"
7    end
8  end
9  class Parrot < Bird
10   def talk
11     puts "I like ruby"
12   end
13 end
14 obj = Parrot.new()
15 obj.talk()
16 obj.quack()
17
18
```

The above program defines a super class: Bird. Then a sub class is defined (parrot). The sub class inherits all attributes and methods from the super class. Finally we create an object. The object has both the methods and attributes of the super class and the sub class.



Ruby constants

Constants (should) have the same value during execution of the program.
Its like a variable, except a Ruby constant should not change its value.

This is very useful for certain types of data that never change. Think of a value like Pi or the version of the program.

If you try to change the value of a constant, the Ruby interpreter will throw an warning. Ruby is however not strict, it will change the value anyway.

Constants in ruby

Example

You can create a constant by adding _CONST to the name. In Ruby, the value of a constant can be changed but it will show a warning.

```
1
2 #!/usr/bin/ruby
3 A_CONST = 3
4 A_CONST = A_CONST + 1
5 print(A_CONST)
```

This wil show the warning:

```
1 test.rb:3: warning: already initialized constant A_CONST
2 test.rb:2: warning: previous definition of A_CONST was here
```

But the actual value is changed. In other programming languages constants are more strict, its value cannot be changed.

Unless a value is actually assigned to them, constants do not exist.

Constants cannot be defined inside methods. But they can be defined throughout the program or inside classes.

Constants inside classes

To access constants inside classes, you can use the :: operator.

The simple example below deomnstrates how to use a constant defined in a class.

```
1 class Example
2   A_CONST = 3
3 end
4 puts Example::A_CONST
```

This will output the value of the constant.

Ruby exceptions

How to indicate that something has gone wrong?

An exception object is raised (or thrown).

Exception can occur during execution of code. By default, ruby excution ends when an exception occurs. But, you can declare exception handlers.

Exceptions in ruby

Example

The program below raises an exception whenever the method raise_exception is called.

```
#!/usr/bin/ruby
def raise_exception
  puts 'Before execution.'
  raise 'An error has occured'
  puts 'Does Ruby get here?'
end
raise_exception
```

This wil show:

```
Before execution.
Traceback (most recent call last):
        1: from g.rb:6:in `<main>'
g.rb:3:in `raise_exception': An error has occured (RuntimeError)
```

This raises an exception using the RuntimeError class.

Practical use

Exceptions can be handled with the rescue keyword.
In the example below we raise an exception and handle it:

```
def example
  begin
    raise StandardError
  rescue StandardError => e
    puts 'Handling exception'
  end
end
example
```

Lets create a situation in which Ruby raises an exception, we’ll divide by zero:.

```
def example
  x = 1 / 0
  rescue ZeroDivisionError => e
```

```
    puts 'Handling exception'
end
example
```

Instead of crashing, Ruby handles the exception and continues execution.

Custom Exceptions

You can use other classes to generate errors, like a custom class:

```
class CustomError < StandardError
end
def raise_exception
  puts 'Before execution.'
  raise CustomError, 'An error has occured'
end
raise_exception
```