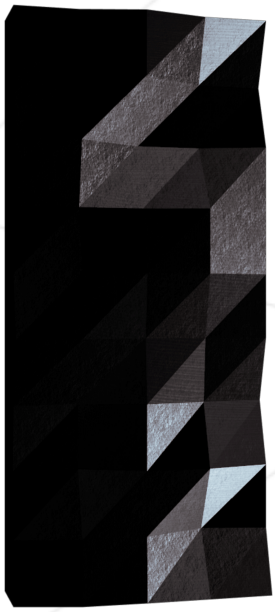

Atlantis D00

Dawn of the First Day

Gaetan gaetan@42.us.org
Meo meo@42.us.org

Summary: Atlantis will rise again.

Contents

Chapter I

Foreword

Blah blah blah

Chapter II

Introduction

Welcome to 42 Silicon Valley - we are the sister school to Ecole 42 in Paris.

We have 1024 computers in this lab, and they are open 24-7 for students to work on the curriculum and side projects on a self-determined schedule.

In Atlantis, we will follow the same traditions as we use for year-round students. Two of the fundamentals are:

1) Learn how to learn

You will work through a set of problems or a project each day, and for beginners, it can sometimes be frustrating that this is not a tutorial. In the piscine, the guiding principles are:

- Did you Google it?
  Error messages that look intimidating can often be resolved just by searching for the error phrase that the program gives.

- Did you ask your neighbor?
  Programming is sometimes seen as a solitary art, but group collaboration is essential for working on complex projects. You will find a balance between sharing ideas vs putting headphones on to problem solve by yourself.

- Did you read the manual?
  In the end, learning to read official documentation is the most direct way to get your understanding from the source. Documentation for the Ruby language is hosted at ruby-doc.org.

2) Peer Correction

For the core curriculum at 42, our projects are evaluated by a computer script - which is VERY strict! - as well as graded by peers. Here, we are going to do brief peer-corrections each day without the computer script. During correction, the person being corrected needs to defend what they wrote. Think of it as trying to sell your work to a paying customer! But because the person correcting you has also done (or will do) the project, stay open to helpful criticism. They might have some tips on your coding style that can help you improve in the long run.

Do not be cruel to each other during corrections, but do not be a pushover either. We are all here to get better. Grade each other honestly and take any feedback gracefully.

Chapter III

The Terminal

- The terminal is your best friend. But also your new working environment.
- The terminal allows you to work on your computer in a fast and textual way.
- Under MacOS, the terminal used is called iTerm. You can find it by clicking on the magnifying glass at the top right of your screen and typing its name ("iTerm"). Then click on the program to open it.

`Better yet, try to access iTerm without using your mouse. What kind of cool keyboard shortcuts are there?`

Chapter IV

The Only Tutorial

- To create a directory, the command is named `mkdir`. The `mkdir` command followed by the name of your choice creates a new directory with that name.

```
$> mkdir example
$>
```

- To list the contents of the current directory, use the `ls` command.

```
$> ls
 example
```

  Note the example directory that you created before.

- There are many options you can use to modify `ls`. You can find them by reading the manual.

```
$> man ls
$>
```

  You can escape by pressing the "q" key!

- Options are added when yo call the command with a hypen and some letters. Thus, `ls -la` adds the options -a and -l to the command `ls`.

```
$> ls -la
total 2896
drwxr-xr-x 11 Elendar staff 374 Jun 14 16:34 .
drwxr-xr-x 5 Elendar staff 170 Jun 14 15:33 ..
-rw-r--r--@ 1 Elendar staff 6148 Jun 14 15:34 .DS_Store
drwxr-xr-x 2 Elendar staff 68 Jun 14 15:59 example
$>
```

- Your directory 'example' is created. It's time to move in. The equivalent of a click on the directory is replaced by the command: `cd` [name of directory].

```
$> cd example
$>
```

  You have just moved into your 'example' directory.

- To check where you are and view the file tree, see the `pwd` command.

```
$> pwd
/Users/Elendar/bootcamp-42/example
$>
```

Do not expect the exact same path on your machine as on the one in the example.

`cd` which, actually means 'change directory', allows you to easily move from directory to directory. Small useful details:

- `cd ..` moves you up one level in the tree. For example, if you are in the 'example' subdirectory, this places you in the directory that contains 'example'.
- `cd` puts you back in your 'Home Directory', the place where you start when you open a new terminal. It's the root of the file tree.

- `touch` followed by the name of a file will create an empty file, which you can then fill using various text editors.

```
$> ls -la
total 0
drwxr-xr-x 2 Elendar staff 68 Jun 14 18:29 .
drwxr-xr-x 7 Elendar staff 238 Jun 14 17:24 ..
$> touch test
$> ls -la
total 0
drwxr-xr-x 3 Elendar staff 102 Jun 14 18:29 .
drwxr-xr-x 7 Elendar staff 238 Jun 14 17:24 ..
-rw-r--r-- 1 Elendar staff 0 Jun 14 18:29 test
$>
```

- `rm` followed by the name of a file will delete the file. `rm -rf` means "remove recursively, with force" and will delete a directory.

Be careful! Files deleted with rm do not go to the Trash. You cannot undelete them.

```
$> ls -la
total 0
drwxr-xr-x 3 Elendar staff 102 Jun 14 18:29 .
drwxr-xr-x 7 Elendar staff 238 Jun 14 17:24 ..
-rw-r--r-- 1 Elendar staff 0 Jun 14 18:29 test
$> rm test
$> ls -la
total 0
drwxr-xr-x 2 Elendar staff 68 Jun 14 18:29 .
drwxr-xr-x 7 Elendar staff 238 Jun 14 17:24 ..
$>
$> ls -la
total 408
drwxr-xr-x 7 Elendar staff 238 Jun 14 18:36 .
drwxr-xr-x 5 Elendar staff 170 Jun 14 15:33 ..
drwxr-xr-x 2 Elendar staff 68 Jun 14 18:30 example
```

```
$> rm -rf example
$> ls -la
total 374
drwxr-xr-x 6 Elendar staff 204 Jun 14 18:36 .
drwxr-xr-x 5 Elendar staff 170 Jun 14 15:33 ..
$>
```

- `chmod` allows you to give "read" or "write" or "run" rights to a file. It must be followed by arguments specifying which rights you want to give to who. Give it three numbers in a row between 0 and 7. 0 means no rights and 7 means that person can do anything to it. The first digit is for the user who owns that file (you if you created it); the second digit is for a category (group) of users; and the third digit represents the permissions for everyone (all users).

```
$> ls
42.rb
$> ./42.rb
zsh: permission denied: ./42.rb
$> chmod 700 42.rb
$> ./42.rb
The answer is 42 !
$> ls -l
total 8
-rwx------ 1 Elendar staff 14 Jun 25 23:06 42.rb
$>
```

Note that `ls -l` gives you details about the permissions applied to each file or directory.

- `cat` is a command that allows you to view the contents of a file. Of course, if there is nothing in the file, you do not display anything.

```
$> cat bonjour.rb
#!/usr/bin/ruby
print "bonjour !"
$>
```

- `open` is a MacOS feature that allows you to run a certain program. Thus, `open` followed by the name of the pdf will open the pdf. Open. (The file name is important) will open the current directory in the Finder (the file exporter of your mac).

```
$> open dontpanic.pdf
$>
$> open .
$>
```

- Finally, pressing the `Ctrl` and `C` keys simultaneously will stop the programs or commands while driving. This may be useful for some unfortunate commands or infinite loops of your programs.

```
$> yes "ctrl + c"
ctrl + c
ctrl + c
ctrl + c ^C
$>
```

You are now ready for anything!

Chapter V

Exercise 00 : name

	Exercise 00
Display a name	
Turn-in directory : $ex00/$	
Files to turn in : `name.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `name.rb` in which you define a first_name variable and a last_name variable, initialized with your first and last names respectively, and then display them followed by a newline.

```
$> ./name.rb | cat -e
Arthur Dent$
$>
```

Chapter VI

Exercise 01 : what's your name

	Exercise 01
What's your name?	
Turn-in directory : $ex01/$	
Files to turn in : `whatsyourname.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `whatsyourname.rb` that first asks the user to enter their first name, then their last name, and finally displays both. Both first name and last name should be concatenated into a third variable.

```
$> ./whatsyourname.rb
Hey, what's your first name ? : Arthur
And your last name ? : Dent
Well, pleased to meet you Arthur Dent.
$>
```

Chapter VII

Exercise 02 : age

	Exercise 02
Receive and modify a number	
Turn-in directory : *ex*02/	
Files to turn in : `age.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `age.rb` that asks the user to enter their age, and then displays how old the user will be in 10 years, 20 years, and 30 years.

```
$> ./age.rb
Please tell me your age : 15
You are currently 15 years old.
In 10 years, you'll be 25 years old.
In 20 years, you'll be 35 years old.
In 30 years, you'll be 45 years old.
$>
```

`Google string to_i.`

Chapter VIII

Exercise 03 : disp_first_param

	Exercise 03
Display a parameter	
Turn-in directory : *ex*03/	
Files to turn in : `disp_first_param.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `disp_first_param.rb` which, when executed, displays the first string passed as a parameter, followed by a newline. If there are no parameters, display `none` followed by a newline.

```
$> ./disp_first_param.rb | cat -e
none$
$> ./disp_first_param.rb "Beeblebrox" "Improbability" "Slartibartfast" | cat -e
Beeblebrox$
$>
```

`Google ARGV, array`

Chapter IX

Exercise 04 : UPCASE_IT

	Exercise 04
Show in all caps	
Turn-in directory : *ex*04/	
Files to turn in : `upcase_it.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `upcase_it.rb` which takes a character string as a parameter. When executed the script displays the string in all caps followed by a newline. If the number of parameters is different from 1, display `none` followed by a newline.

```
$> ./upcase_it.rb | cat -e
none$
$> ./upcase_it.rb "don't panic" | cat -e
DON'T PANIC$
$> ./upcase_it.rb 'tHiS iS sO eAsY! - rUbY iS bAe' | cat -e
THIS IS SO EASY! - RUBY IS BAE$
$>
```

Chapter X

Exercise 05 : downcase_it

	Exercise 05
Show in lowercase	
Turn-in directory : $ex05/$	
Files to turn in : `downcase_it.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `downcase_it.rb` which takes a character string as a parameter. When executed, the script displays the string in lowercase followed by a newline. If the number of parameters is different from 1, display `none` followed by a newline.

```
$> ./downcase_it.rb | cat -e
none$
$> ./downcase_it.rb "TRILLIAN" | cat -e
trillian$
$> ./downcase_it.rb 'tHiS iS sO eAsY! - rUbY iS bAe' | cat -e
this is so easy! - ruby is bae$
$>
```

Chapter XI

Exercise 06 : scan_it

	Exercise 06
Scan the text	
Turn-in directory : *ex*06/	
Files to turn in : `scan_it.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `scan_it.rb` that takes two parameters. The first is a keyword to look for in a string. The second is the string to search. When executed, the program displays the number of occurrences of the keyword in the string. If the number of parameters is different from 2 or the first string does not appear in the second string, then display `none` followed by a newline.

```
$> ./scan_it.rb | cat -e
none$
$> ./scan_it.rb "the" | cat -e
none$
$> ./scan_it.rb "the" "these exercises in day01 are not the hardest ones we'll see \=P" | cat -e
2$
$>
```

Chapter XII

Bonus part

When a student invests time in a project and the goals are met, it's innate to will to go further ! The bonus section is here to satisfy such ambition. Of course, the bonus part is exclusively available if and only if the mandatory part is complete and perfect.

Chapter XIII

Turn-in and peer-evaluation

This part describes the conditions and instructions regarding the turn-in and the peer-evaluation of the project. If your project does not require odd turn-in or peer-evaluation instructions, feel free to use the following paragraph as it is:

Turn your work in using your `GiT` repository, as usual. Only work present on your repository will be graded in defense.