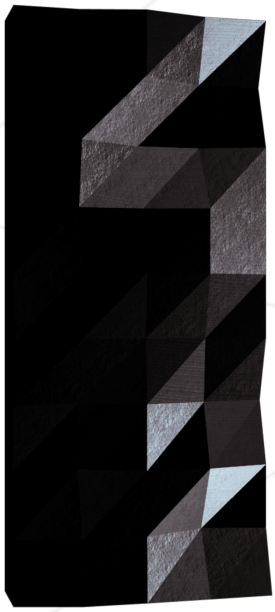

Introduction to Programming 42

Day 02

Kai kai@42.us.org
Gaetan gaetan@42.us.org

Summary: This document is the subject of the day 02 of the introduction to programming piscine.

Contents

I	Guidelines	2
II	Preamble	3
III	Exercise 00 : puts each	5
IV	Exercise 01 : display rev params	6
V	Exercise 02 : Did you get that	7
VI	Exercise 03 : Strings Search	8
VII	Exercise 04 : Array Play	9
VIII	Exercise 05 : Array++	10
IX	Exercise 06 : Array+=2	11
X	Exercise 07 : Count It	12
XI	Exercise 08 : Pig Latin	13

Chapter I

Guidelines

- Corrections will take place in the last hour of the day. Each person will correct another person according to the peer-corrections model.
- Questions? Ask the neighbor on your right. Next, ask the neighbor on your left.
- Read the examples carefully. The exercises might require things that are not specified in the subject...
- Your reference manual is called Google / "Read the Manual!" / the Internet / ...

Chapter II

Preamble

From the Wikipedia article for Whitespace (Programming Language):

Whitespace is an esoteric programming language developed by Edwin Brady and Chris Morris at the University of Durham (also developers of the Kaya and Idris programming languages). It was released on 1 April 2003 (April Fool's Day). Its name is a reference to whitespace characters. Unlike most programming languages, which ignore or assign little meaning to most whitespace characters, the Whitespace interpreter ignores any non-whitespace characters. Only spaces, tabs and linefeeds have meaning. An interesting consequence of this property is that a Whitespace program can easily be contained within the whitespace characters of a program written in another language, except possibly in languages which depend on spaces for syntax validity such as Python, making the text a polyglot.

The language itself is an imperative stack-based language. The virtual machine on which the programs run has a stack and a heap. The programmer is free to push arbitrary-width integers onto the stack (currently there is no implementation of floating point numbers) and can also access the heap as a permanent store for variables and data structures. Commands are composed of sequences of spaces, tab stops and linefeeds. For example, tab-space-space-space performs arithmetic addition of the top two elements on the stack. Data is represented in binary using spaces (0) and tabs (1), followed by a linefeed; thus, space-space-space-tab-space-tab-tab-linefeed is the binary number 0001011, which is 11 in decimal. All other characters are ignored and thus can be used for comments.

Code is written as an Instruction Modification Parameter (IMP) followed by the operation. The table below shows a list of all the IMPs in Whitespace.

IMP Meaning
[Space] Stack Manipulation
[Tab][Space] Arithmetic
[Tab][Tab] Heap Access
[LineFeed] Flow Control
[Tab][LineFeed] I/O

The following is a commented Whitespace program that simply prints "Hello, world!",

where each Space, Tab, or Linefeed character is preceded by the identifying comment "S", "T", or "L", respectively:

```
S S S T S S T  S S S L
T  L
S S S S S T  T  S S T  S T  L
T  L
S S S S S T  T  S T  T  S S L
T  L
S S S S S T  T  S T  T  S S L
T  L
S S S S S T  T  S T  T  T  T  L
T  L
S S S S S T  S T  T  S S L
T  L
S S S S S T  S S S S S L
T  L
S S S S S T  T  T  S T  T  T  L
T  L
S S S S S T  T  S T  T  T  T  L
T  L
S S S S S T  T  T  S S T  S L
T  L
S S S S S T  T  S T  T  S S L
T  L
S S S S S T  T  S S T  S S L
T  L
S S S S S T  S S S S T  L
T  L
S S L
L
L
```

...It's a minimalist aesthetic.

Chapter III

Exercise 00 : puts each

	Exercise 00
Letś traverse the array	
Turn-in directory : $ex00/$	
Files to turn in : `puts_each.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `puts_each.rb` which declares a number array [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] and uses each method to iterate over the array and display each value.

```
?> ./puts_each.rb | cat -e
1$
2$
3$
4$
5$
6$
7$
8$
9$
10$
?>
```

`Google array, each`

Chapter IV

Exercise 01 : display rev params

	Exercise 01
Display a table in reverse	
Turn-in directory : *ex*01/	
Files to turn in : `display_rev_params.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `display_rev_params.rb`

```
?> ./display_rev_params.rb | cat -e
none$
?> ./display_rev_params.rb "yolo" | cat -e
none$
?> ./display_rev_params.rb "Garkbit" "God" "Genghis Khan" | cat -e
Genghis Khan$
God$
Garkbit$
?>
```

`Google ARGV, array, reverse`

Chapter V

Exercise 02 : Did you get that

	Exercise 02
description	
Turn-in directory : *ex*02/	
Files to turn in : `i_got_that.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `i_got_that.rb`. This script must contain a while loop that accepts a user input, write a return phrase, and stops only when the user has entered "STOP!". Each round of loops must accept input from the user.

```
?> ./i_got_that.rb
What you gotta say?: Hello
I got that! Anything else?: I like ponies
I got that! Anything else?: stop...
I got that! Anything else?: STOP!
?>
```

`Google while, break`

Chapter VI

Exercise 03 : Strings Search

	Exercise 03
Getting to know strings	
Turn-in directory : *ex*03/	
Files to turn in : `string_search.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `string_search.rb` that takes a character string as a parameter. When executed, the script displays "z" for each character "z" in the string passed as a parameter, followed by a newline. If the number of parameters is different that 1, or there is no "z" character in the string, display "none" followed by a newline.

```
?> ./strings_search.rb "The target character is not in this string" | cat -e
none$
?> ./strings_search.rb "z" | cat -e
z$
?> ./strings_search.rb "Zealously zany zapping zebras zigzag zested Zoology zone" | cat -e
zzzzzzz$
?>
```

`Scan Each string carefully!`

Chapter VII

Exercise 04 : Array Play

	Exercise 04
Manipulating arrays	
Turn-in directory : *ex*04/	
Files to turn in : `play_with_arrays.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `play_with_arrays.rb` which takes an array of numbers (that you define) and builds a new array that is the result of adding the value 2 to each value of the original array. You must have two arrays at the end of the program, the original one and the new one you created. Display the two arrays on the screen using the p method rather than puts.

- For example, if your original array is [2, 8, 9, 48, 8, 22, -12, 2] you will get the following output:

```
?> ./play_with_arrays.rb | cat -e
[2, 8, 9, 48, 8, 22, -12, 2]$
[4, 10, 11, 50, 10, 24, -10, 4]$
?>
```

`Google p method in ruby, array map`

Chapter VIII

Exercise 05 : Array++

	Exercise 05
Manipulating arrays, encore	
Turn-in directory : *ex*05/	
Files to turn in : `play_with_arrays_2.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Repeat the previous script, but this time you will only process values greater than 5 from the original array.
- For example, if your original array is [2, 8, 9, 48, 8, 22, -12, 2], you will get the following output:

```
?> ./play_with_arrays_2.rb | cat -e
[2, 8, 9, 48, 8, 22, -12, 2]$
[10, 11, 50, 10, 24]$
?>
```

`Google array range`

Chapter IX

Exercise 06 : Array+=2

	Exercise 06
Manipulating arrays, forever!	
Turn-in directory : *ex*06/	
Files to turn in : `play_with_arrays_3.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Repeat the previous script, but this time you will no longer have duplicate output. Be careful, you do not have to explicitly remove values from your arrays.
- For example, if your original array is [2, 8, 9, 48, 8, 22, -12, 2], you will get the following output:

```
?> ./play_with_arrays_3.rb | cat -e
[2, 8, 9, 48, 8, 22, -12, 2]$
[10, 11, 50, 24]$
?>
```

`Google array, uniq`

Chapter X

Exercise 07 : Count It

	Exercise 07
Counting and measuring parameters	
Turn-in directory : $ex07/$	
Files to turn in : `count_it.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `count_it.rb`, which, when executed, has "parameters:" and then the number of parameters, followed by a newline, then each parameter and its size followed by a newline. If there is no parameter, output `none` followed by a newline.

```
?> ./count_it.rb "Life" "Universe" "Everything" | cat -e
parameters: 3$
Life: 4$
Universe: 8$
Everything: 10$
?>
```

`Google size`

`Strictly adhere to the format indicated in the example.`

Chapter XI

Exercise 08 : Pig Latin

	Exercise 08
Modifying strings	
Turn-in directory : *ex*08/	
Files to turn in : `atinlay.rb`	
Allowed functions : `All`	
Notes : `n/a`	

- Create a script `atinlay.rb` that translates its arguments into pig latin.
- If the word starts with one or more consonants, remove the group of consonants and add them to the end of the string with a hypen. Then, add the letters "ay".
- If the first letter is a vowel, just add a "way" to the end.
- If there is no parameter, display `none` followed by a newline.
- You should probably downcase them, too

```
?> ./atinlay.rb "paranoid" "android" "Marvin" "so" "stoic" | cat -e
aranoidpay$
androidway$
arvinmay$
osay$
oicstay$
?>
```

oink, oink