Ex 22

PUTS:
puts "put string" prints a string, number, variable with a new line (can also carriage-return blank).

print does the same thing except there's no carriage-return.

"" double quotes, used when there's string interpolation.
'' single quotes only when there's plain text (no interpolation).
hash/octothorpe for creating a single line of commented out text (won't show in code). can also temporarily disable code; can also appear on same line of code at its end for instruction or advisories.
=begin (press return)
=end same thing for blocks of commented-out text.
.to_s converts to a string.

MATHS:
+ will add integers (whole numbers) and floating point 2.00 numbers, can also add two separate strings (concatenate).
- subtracts integers and floats
***** multiplies them.
/ divides them.
[1, 2, 3] is an array or list of items in sequential order starting with 0 then 1, 2, 3... can be strings, numbers and combinations of same including variables.
More MATHS symbols: > is greater than. < is lesser than. >= greater than or equal to. <= less than or equal to.
% is modulo. What's 'left over' after a calc. 2 % 5 = 1.

VARIABLES:
variable points to something, like maths, strings, arrays, hashes.

.length counts the length of a thing (usually strings) in characters.
'this won\'t run, or will it?' backslash is escape key so the single quote won\'t end the string early and cause an error.
"this won't run" but it will as the double quotes tell ruby there\'s a thing to be read and acted upon within the string.

"what is 21 - 5 #{21 - 5}?" #{} tells ruby there is MATHS or a calculation to be acted on within a string (interpolation) and the results returned within the string.

Single quotes will cause an error as they only print a string with no calculations performed.
Variables 'point' to a thing using '='. Known as an assignment operator.

Variables can also be **PEMDAS'd** with each other once they point to a number or an object containing a number (arrays, hashes, other variables, symbols).
Variables can also point to strings, be joined with other strings using 'concatenation'.

% is also used in a format string where % determines the values, which are supplied in the same sequence as the formatting codes %{} in the format string (ex 8).

\ escape character, performs actions on a string without printing or outputting to the screen.
\n = newline
\t = tab
\\ = single backslash

% q behaves like a single quote string (no character escaping or interpolation) %Q behaves like its double quote cousin.

""" on first line and
on the last enables writing a block of text """

INPUT:
gets.chomp asks for input from the user before proceeding '.to_s' converts to a string, '.to_i' to an integer and '.to_f' to a floating point number.

ARGV is the 'argument variable' found in many languages that 'holds' whatever you pass into it and then allows one or more variables to point to it. Kinda like parameters in method definitions.

$stdin is 'standard input' used with kernel#gets (.gets) as ARGV and #gets don't work well together so $stdin is more explicit. ARGV can take in files and text where $stdin takes in just text.. Use ARGV on the command line.

.first when applied to ARGV takes in only the very first item passed into it, even f there's a 100 typed in. A character '> ' can be used as a prompt.

open() is a method that has two main functions: 1- opens a file and 2- assigns an object to that filename i.e. File.open and File.new. Another difference is File.open can be associated with a block unlike File.new. In the parentheses, a 'w' tells open() you're going to write to the file. variable.write() with strings and str interpolation #{} places the physical text into the object. then variable.close closes the still-open file.

.read is a method called on any variable pointing to an open()'ed file. It returns the contents of the opened file as a string.

File.exist?() checks if a file exists; returns true if yes, false if no.

.truncate() truncates a given text after a given length if text is longer than length.

***args** creates a list of arguments, allows for multiple args at once.

def..end defines a function; usually referred to as a method as it defines or 'instantiates' an instance method.

.seek(0) goes to a given position (as integer) in a stream or file.

+= is an incrementer operator. Short way of adding a number to something so's you are always increasing the number by that amount. You would also set a limit like >= 'greater than or equal to'

return is a keyword that says: 'must add or subtract or whatnot AND return or show the answer/result', "puts" just prints what you type, it doesn't figure anything out.

.reverse reverses the text or numbers to the left of it.

.capitalize upper-cases the very first letter of the very first word it sees. .upcase makes capital letters of EVERY single letter and .downcase does the exact opposite.

%w() or %w[] creates an array (a list) or strings (letters, words, sentences) without commas or quotes. %W is almost the same, it provides double-quotes "" so you can add things like: #{ruby_code} into the string which is like adding more text or numbers from anywhere else in the application.

.slice(0..5) works a little like truncate() in that it chops off letters/numbers at a specific point; in this case, it's after the 6th character (it counts the '0' too)

.inspect takes a look at the things you've just typed, it prints a line describing what they are; string, integer, variable, symbol etc. Gives it a unique identifier number all in an array.

variable = { name: "Julian", age: "don't ask" } hash, variable pointing to curly braces containing key/value pairs vs. an array which is an indexed list of things. When called, the key goes from this: to :this and the value can be absolutely ANYTHING (string, number, variable any object)

.values is a method that can convert a hash into an array (or list).