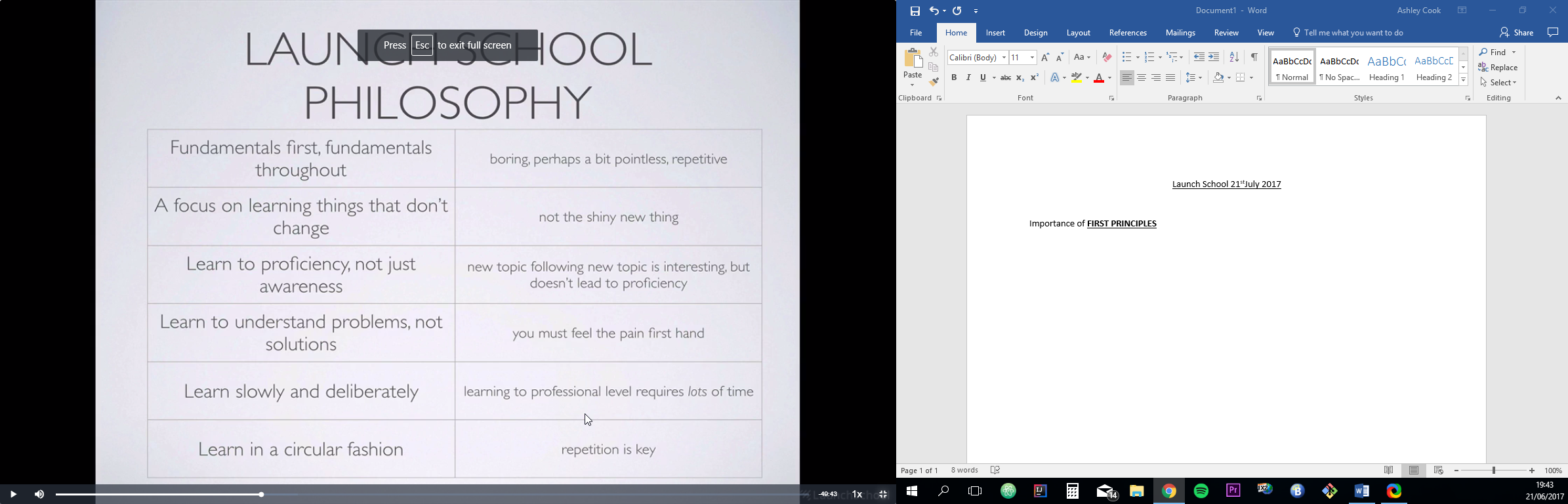
Launch School 21stJuly 2017

Importance of **FIRST PRINCIPLES**



Growth Mindset.

Sacrifice.

Repetition.

Communication.

Work ethic.

Understand first principles.

Ability to ship quality code.

Study Habits

TAKE NOTES!!!

READ OVER NOTES!!!

FLASH CARDS!!

Mastery learning (find your dissertation maybe?), book by George Leonard will be useful. Can be read in a morning! Make notes on the book!

Build a mental model of how everything works, ask WHY over HOW when asking questions.

22nd July 2017

--What is code article—

Smalltalk’s history is often described as slightly tragic, because many of its best ideas never permeated the culture of code. But it’s still around, still has users, and anyone can use Squeak or Pharo. Also—

1. Java is an object-oriented language, influenced by C++, that runs on a virtual machine (just like Smalltalk).
2. Objective-C, per its name, jammed C and Smalltalk together with no apologies.
3. C# (pronounced “C sharp”) is based on C and influenced by Java, but it was created by Microsoft for use in its .NET framework.
4. C++ is an object-oriented version of C, although its roots are more in Simula.

Python- “Object-oriented,” “imperative,” and “functional” are paradigms; a language like Python is referred to as “multiparadigm.”

**Nutrition: Soylent**

Old languages still in use that were developed in the 50s, Fortran and LISP.

Lisp is a language that programmers venerate because of its rich history and its simplicity—and the fact that such simplicity makes it possible for Lisp to program itself; it’s described as a “programmable programming language.” If you ever have bright programmers who need a challenge, send them off to learn Lisp. Some may return as insufferable evangelists, but more likely they will come back smarter and more flexible.

DATABASES

When people talk about databases, they often use the acronym CRUD, for create, read, update, and delete.

24th July 2017

|  |  |  |
| --- | --- | --- |
| Quality Assurance (QA) | Verify features are built according to specification. Often the gate-keepers to deploying to production. | Well rounded technical knowledge, and attention to detail. Many QA positions also require some programming ability in order to write test scripts. |
| System/Network Administrator (sometimes called Devops) | Manages servers and server infrastructure. Manages user accounts, permissions, and access to servers. Sometimes also manages the hardware within a company. | Two main areas: Unix and Windows. Expert knowledge of either Unix or Windows servers and networks. Basic programming of scripts is usually required. |
| Database Administrator | A subset of System Administrator role that focuses solely on databases and database architecture (only for large deployments). | Expert knowledge of relational databases, and other data stores. |
| Platform/Infrastructure Engineer | More engineering concerns, like data processing, algorithmic complexity, concurrency, etc. | Usually requires bachelor's or advanced degree in Computer Science or related field. General purpose expert programmer/scientist, and not necessarily related to web application development |

Do not use course projects are portfolio, simply because you will have had help and guidance throughout. MUST BE YOUR OWN PROJECTS, to act as a portfolio.

Keep code examples small and simple, not large projects.

Make sure to include test, even for simple methods, this shows how thoughtful you are!

Make sure to use branches or pull requests so you show you know how to use those things.

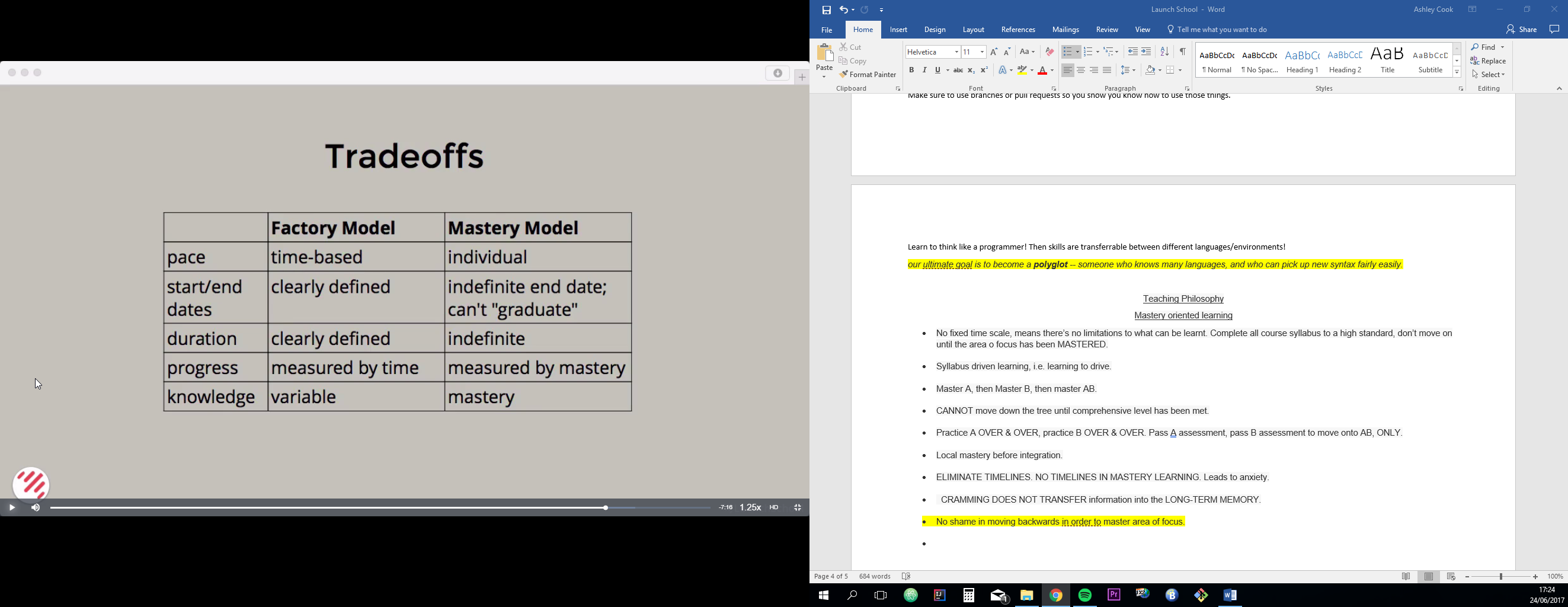
Learn to think like a programmer! Then skills are transferrable between different languages/environments!

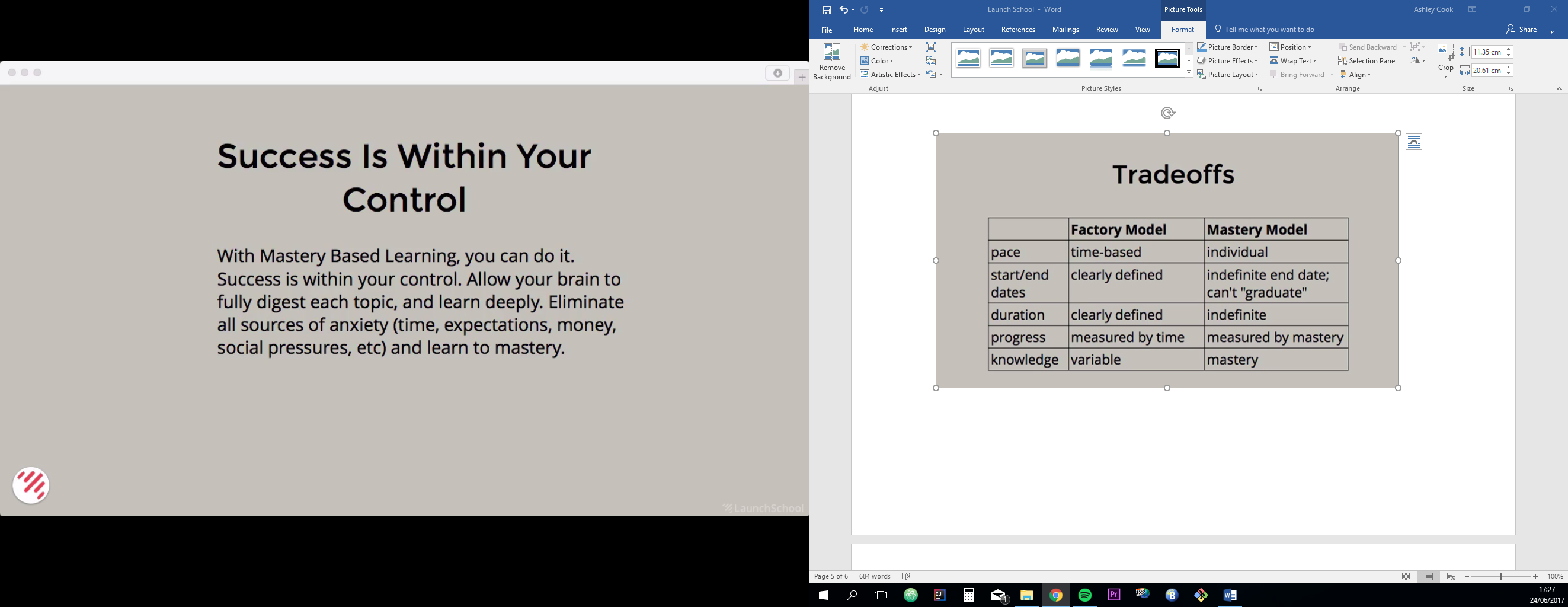
*our ultimate goal is to become a****polyglot****-- someone who knows many languages, and who can pick up new syntax fairly easily.*

Teaching Philosophy

Mastery oriented learning

* No fixed time scale, means there’s no limitations to what can be learnt. Complete all course syllabus to a high standard, don’t move on until the area o focus has been MASTERED.
* Syllabus driven learning, i.e. learning to drive.
* Master A, then Master B, then master AB.
* CANNOT move down the tree until comprehensive level has been met.
* Practice A OVER & OVER, practice B OVER & OVER. Pass A assessment, pass B assessment to move onto AB, ONLY.
* Local mastery before integration.
* ELIMINATE TIMELINES. NO TIMELINES IN MASTERY LEARNING. Leads to anxiety.
* CRAMMING DOES NOT TRANSFER information into the LONG-TERM MEMORY.
* No shame in moving backwards in order to master area of focus.





Mastery by George Leonard Notes-

Mastery is a journey.

Mastery is not reserved for those “born with exceptional abilities”

It is for anybody WILLING to get on the path and stay on it, regardless of uncontrollable factors and previous exp.

No goal, no destination but simply a process, a journey.

Animal sports event example.

The mastery curve features many states at the plateau stage with small incremental upcurves along the way.

The dabbler, the obsessive, the hacker.

The path of endless climax, i.e. adverts with alcohol etc

Loving the plateau!

Goals exist in the future and past, only practice exists in the present.

CHAPTER TWO- The five mastery keys

* Instructions- importance of teacher/other learning materials

“When you learn too easy you’re easily tempted not to work hard, not to penetrate to the marrow of a practice”

* Practice- There’s a distinction between practice (doing it for a living) and a master’s practice. Mastery is practice, mastery is staying on the path.
* Surrender- A willingness to surrender to the teacher or the demands of your discipline. This also includes surrendering your hard-won proficiency from time to time to reach a higher level of proficiency.
* Intentionality- The importance of the vision. Every master is a master of the vision.
* The Edge- A balance of knowing when you’re pushing yourself. Exceed your limits. Sacrifice is required, finish at all costs, attain the unattainable.

Tools for Mastery

We gain energy by using energy. 30mins of aerobic exercise is the best remedy for weariness.

Liberation comes through the acceptance of limits. You cannot do everything, but you can do thing, and then another and another.

Consistency of practice is the mark of a master.

Being overly self-critical is destructive to creativity. Mastery is not about perfection, it’s about a process, a journey.

--------Ruby codecademy prerequisite------

.chomp method prevents get data from being put on new lines each time

.capaitalize! <---- The '!' adjusts the variable to keep the changes for when it is called again.

--control flow--

keyword 'unless' used, unless condition true runs code, if condition =true then run the else code.

--boolean/logical operators--

&& || and !(not)

--comparitive/relational operators--

== != < > etc

.include method—

We can do that using Ruby's .include?method, which evaluates to true if it finds what it's looking for and falseotherwise.

(As a general rule, Ruby methods that end with ? evaluate to the boolean values true or false.)

.gsub method aka global substitution

.gsub(/s/, “replacement”)

#{my\_string} to print string.

While Loop

While condition

Condition += 1

end

Until Loop

Var = 1

Until var > 10

Var += 1

End

For Loop

for num in 1...10

puts num

end

P.s. the three dots tell Ruby to exclude the final number i.e 10. Two dots means include the highest number in the range!

The Iterator

An iterator is just a Ruby method that repeatedly invokes a block of code.

The Loop Method

Loop do

-------- Example code------------

Break {condition}

End

“next” Keyword

Next if {condititon}, --skips the loop then—

Using the .each iterator

Object.each {|item| #code}

Using the.times iterator

10.times{puts “print something”}

Redact information

The .split method

Takes in a string and returns an array. Object.split(“ “) would split everytime Ruby saw a space.

Data Structures

My\_array = [1,2,3]

multi\_d\_array = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]] these are arrays within arrays

Hashes

Essentially like a Java hashmap.

Hash = {

Key1 => value1,

}

Array = Hash.new(“TAKES ARGUMENT HERE FOR DEFAULT VALUE”)

Array[key] = value

My\_array.each do |key,value|

Puts “#{key}:#{value}”

.sort\_by method

returns an array of an array

e.g. frequencies = frequencies.sort\_by do |key,value|

    value

end

.reverse! method

reverses a string or array

Methods

Arguments are the pieces of code that are passed to the method and usually reside in the parentheses.

Parameters are the placeholders that sit in the method waiting for arguments to be put in their place.

Splat Arguments

\* is used if not sure how many arguments will be passed to method

Blocks

Methods without a name. i.e. anonymous functions

1.times do

Puts “hello”

End

The combined comparison operator

<=> Used to compare two Ruby objects.

Returns 0 if the first operand equals the second operator

1 if first operand is greater than second

-1 if less than second operand

By “greater” it could mean that the object would be after the other object is sorted with the .sort method.

Array.sort! do |firstItem,secondItem| secondItem ⬄ firstItem ---------- Would sort them reverse alphabetically

Ruby Symbols

Symbols ARE NOT strings, they’re a *sort of* name.

There can only be one copy of a symbol at any given time, whereas you could have the same string multiple times but it would be independent and unique from the last one.

Syntax = “:” e.g. :my\_symbol

Uses of symbols

Usually used as hash keys or for referencing methods

.to\_sym method converts strings to symbols

EXAMPLE

strings = ["HTML", "CSS", "JavaScript", "Python", "Ruby"] ----An array of strings

# Add your code below!

symbols = [] ----Create an empty array

strings.each do |s| ----iterates through the string array, s as a placeholder for each element

symbols.push(s.to\_sym) –s element gets converted and pushed to symbol array .intern can also be used!

End

The Case Statement

Case language

When “JS”

Puts “Websites”

When ”Ruby”

Puts “Web Apps”

Else

Puts ”I don’t know!”

End

CRUD- CREATE, READ, UPDATE & DELETE.

The Zen of Ruby

Simpler ‘If’ Statement

Expression if Boolean

Ternary Conditional Expression

Takes 3 arguments, a Boolean, expression to evaluate if true and an expression to evaluate if false.

Syntax- boolean ? Do this if true: Do this if false

puts 1 < 2 ? "One is less than two!" : "One is not less than two."

Case statement

case test

when “test” then puts “this is test”

end

Conditional assignment operator

||= assigns value to variable IF it is not already being used

Implicit Return

No need to use return keyword as Ruby returns result of last evaluated expression.

For loop alternatives

Can use the .upto and .downto methods instead of expecting for loop to meet conditions

OR “3.times do”

Use the .respond\_to? Symbol to check if the object can receive that method

Array.respond\_to?(:push) *would return true.*

<< is the concatenation operator

String interpolation “I love #{lamp}”

**Collect method**

Takes code block and applies it to every element in the array.

**Yielding**

Aka ‘Closures’ in other languages.

Allows methods to yield and run blocks of code outside of the method, then continue to run the rest of the method.

def yield\_name(name)

puts "In the method! Let's yield."

yield("Kim")

puts "In between the yields!"

yield(name)

puts "Block complete! Back in the method."

end

yield\_name("Eric") { |n| puts "My name is #{n}." }

# Now call the method with your name!

yield\_name("Ash") do |n| puts "My name is #{n}."

end

**Objects (Procs)**

Blocks are not objects in Ruby. Blocks cannot be saved as variables because of this.

Procs are essentially “saved” blocks.

Can be passed back using the collect method.

floats = [1.2, 3.45, 0.91, 7.727, 11.42, 482.911]

round\_down = Proc.new do |number| number.floor

end

ints = floats.collect(&round\_down)

**Lambda**

Lambda’s are also objects. The syntax differs slightly.

symbolize = lambda do |string| string.intern ----------*–BLOCK OF CODE NOW A LAMBDA*

end

symbols = strings.collect(&symbolize) -----------*string array collected and symbolize with the lambda*

**Lambdas vs. Procs**

First, a lambda checks the number of arguments passed to it, while a proc does not.

Second, when a lambda returns, it passes control back to the calling method; when a proc returns, it does so immediately, without going back to the calling method.

Blocks- Bit of code with the do keyword, CAN be passed to methods like .each and .select.

Proc- Saved block we can reuse over and over again.

Lambda- Just like a proc but cares about arguments and returns to calling method rather than returning straight away.

crew = {

captain: "Picard",

first\_officer: "Riker",

lt\_cdr: "Data", <<<ARRAY

lt: "Worf",

ensign: "Ro",

counselor: "Troi",

chief\_engineer: "LaForge",

doctor: "Crusher"

}

# Add your code below!

first\_half = lambda do |key, value| value < "M"

end

a\_to\_m = crew.select(&first\_half) <<<<PASSED TO A LAMBDA METHOD

**O-O Programming – Classes**

A class is just a way of organizing and producing objects with similar attributes and methods.

Use @before a variable to state that it is an *INSTANCE VARIABLE*.

Create an instance of the class using keyword .New

$= global variable

@=instance variable i.e. stays with the instance that gets created

@@=class variable i.e. can see how many instances of that class created as that variable stays with the class

**Inheritance**

A class taking on the attributes and methods of another using the IS-A relationship.

class DerivedClass < BaseClass

*# Some stuff!*

end

You can read "<" as "inherits from."

**Override**

Simply call the method and put in the new code and it will run!

Keyword super will run the superclass version of that method.

Ruby DOES NOT ALLOW multiple inheritance.

**Classes**

Example Syntax—

Class Dog

Def initialize(name, breed)

@name = name

@breed = breed

End

End

Another way to say this is that the method cannot be called with an explicit receiver. You've been using receivers all along—these are the objects on which methods are called! Whenever you call object.method, object is the receiver of the method.

**Attr\_reader, attr\_writer**

Can use attr\_reader and write to read and write the attributes/variables in a class.

Attr\_accessor acts as both the reader and writer.

Pass our instance variables over as symbols. Attr\_accessor :name

This is used to replace initialize method variables!!

**Modules**

Essentially toolboxes which contain methods. Cannot create new instances of these “classes”.

Constant variables are used, ALL CAPS.

:: 🡨------ Is called the scope resolution operator. E.g. Math::PI , tells where Ruby where to look.

Include or Require used to use modules.

Extend a class so it can use the methods in the module.

EXAMPLE. Create a module called moddy with a method called mod

Class test

Extend moddy

End

Test.mod

**CMD LINE USERFACES**

Pwd= present working directory

Tar command- archival. -c to create, -z to zip and -f stands for file.

| **Command** | **Description** |
| --- | --- |
| cd | Change directory. |
| ls | List files and directories in current directory. |
| pwd | Display the path of the current directory. |
| touch | Create a file. |
| mkdir | Create a directory. |
| rm | Remove a file or directory. Warning: deleting a file or directory with this command is permanent! |
| cp | Copy a file or directory. |
| mv | Move or rename a file or directory. |
| echo | Print text to STDOUT. |
| cat | Display contents of a file. |
| more | Display contents of a file, starting at the top and letting the user scroll down. |
| less | Display contents of a file in an even more interactive way. |
| head | Display the first part of a file. |
| tail | Display the last part of a file. |
| man | Display documentation about a command. |

* / - The root directory or a separator when listing directories
* . - The current directory (also ./) or the same level
* .. - The directory one level up (also ../)
* ../.. - Two levels up
* ~ - Your "home" directory, or the directory you are placed in when you log in.
* \* - The "splat" or "glob" operator. This is the wildcard of the command line and represents "any characters."

| **Command** | **What it does** |
| --- | --- |
| cp | Copy one or more files to a new location |
| mv | Move or rename a file or directory |
| mkdir | Make a new directory |
| touch | Create a new file or update modification time if a file with that name exists |
| rm | Remove one or more files or directories |

**Permissions**

In Unix and Linux file systems, permissions are divided into two parts: ownership and access types. There are three levels of ownership: **user**, **group**, and **other**; and three access types: **read**, **write**, and **execute**.

Rwx -read,write,execute permissions

Chmod used to edit permissions.

**Root user and Sudo**

Sudo borrows the root’s privileges to complete tasks.

 "Read-Eval-Print-Loop"

-evaluates the input then outputs

**Git and GitHub**

Source control management or version control management.

Git is a distributed system that allows developers to push commits from local repos while other commits are being made by other devs at the same time from their local repos.

**Branching**

**Using Ruby**

Ruby v2.4 installed, gems package managed also bundled in.

Use binding.pry to troubleshoot, ctrl +D to continue with code execution.

**Methods**

It is possible for some methods to **mutate** the caller, an example would be using pop on array, which would permanently mutate the data in that array versus using last which would not mutate the array.

Need to be aware of using data that gets returned ie. “hello”.length.to\_s, as puts returns nil and return would return 5!

**Flow Control**

Conditionals are fork in which your data reaches these points and the conditionals determine which path the data will take.

Order of precedence- Comparison, Equality, Logical And, Logical Or

**# Ternary operator example**

**irb :001 > true ? "this is true" : "this is not true"**

**=> "this is true"**

**Case statement**

Case,when,else and end used.

Define the case first, e.g. case a.

**EVERY** expression evaluates to true when in **flow control** except for false and nil. i.e. a = 5 would return true.

**Loops and Iterators**

While loops, do/while loops and for loops.

Iterators like each are useful.

**Arrays**

Need to be aware of methods that mutate the caller!

.uniq deletes any duplicate, BUT this does not mutate the original array. If used on it’s own it returns a new array with duplicates removed. IF you use the BANG operator (!) then the array gets mutated. .uniq! is a separate method, you CANNOT just add ! onto the end of methods to make them destructive.

.pop removes last element in array, mutates.

.unshift(1) add new element to start of the array

.include?(1) checks for element and returns Boolean.

.flatten turns a multi-dimensional array into a one dimensional array. Use bang operator to mutate.

.product method combines array into combination of all elements.

Use .each to iterate through an array, or map! to mutate that array.

**Hashes**

Fetch method allows user to look for key/value AND return something specific if that key cannot be found.

To\_A changes it to an Array. Does not mutate.

**Files**

IO Open Mode determines permissions for that file. I.e. myfile = File.new(“test.txt”, “w+”)

On the second line, we are closing the file. We want to **always close files.** Open files will continue to occupy space in memory.

* r: read-only (starts at beginning of file)
* w: write-only (if the file exists, **overwrites** everything in the file)
* w+: read and write (if the file exists, **overwrites** everything in the file)
* a+: read-write (if file exists, starts at end of file. Otherwise creates a new file). Suitable for updating files.

If file is written to with a block it is then automatically closed.

\*gem install axlsx

\*gem install nokogiri

**Extras**

**Regex**

Search for patterns. i.e. if the expression “ss” appears in “Mississippi” etc.

**irb :001 > "powerball" =~ /b/**

=> 5

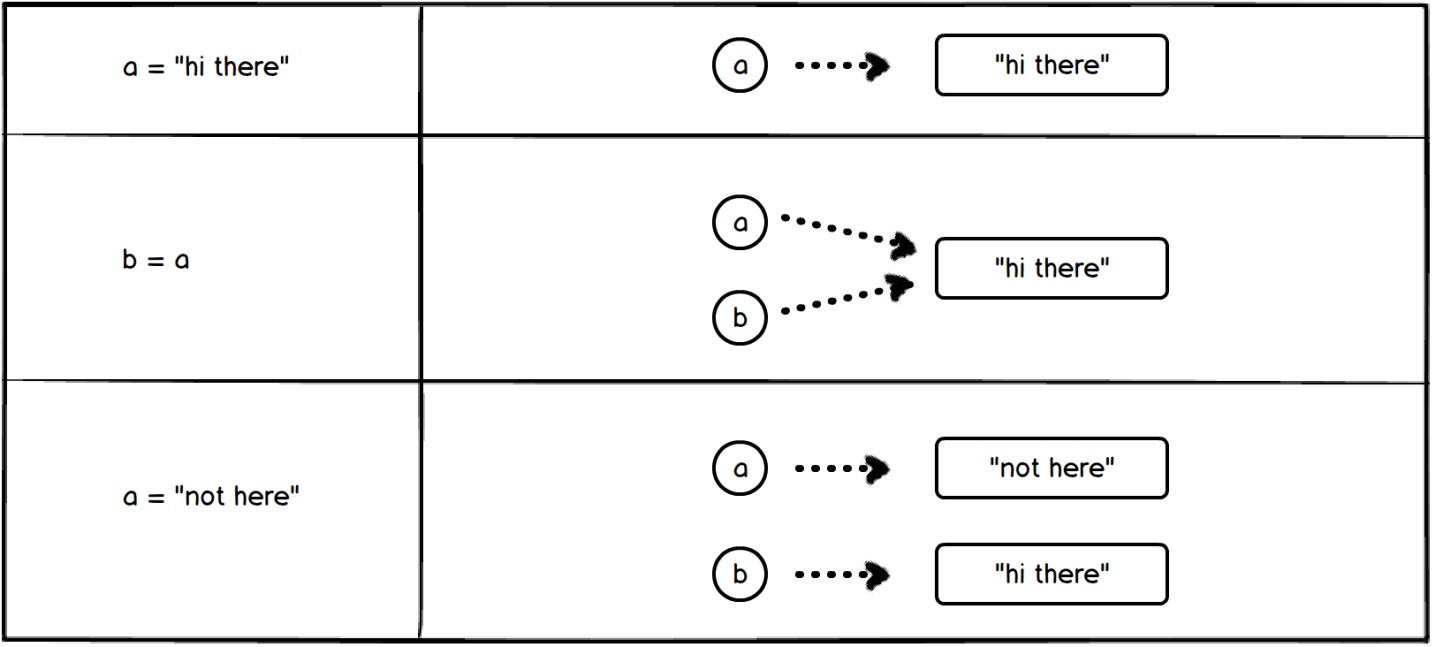
**irb :001 > /b/.match("powerball")**

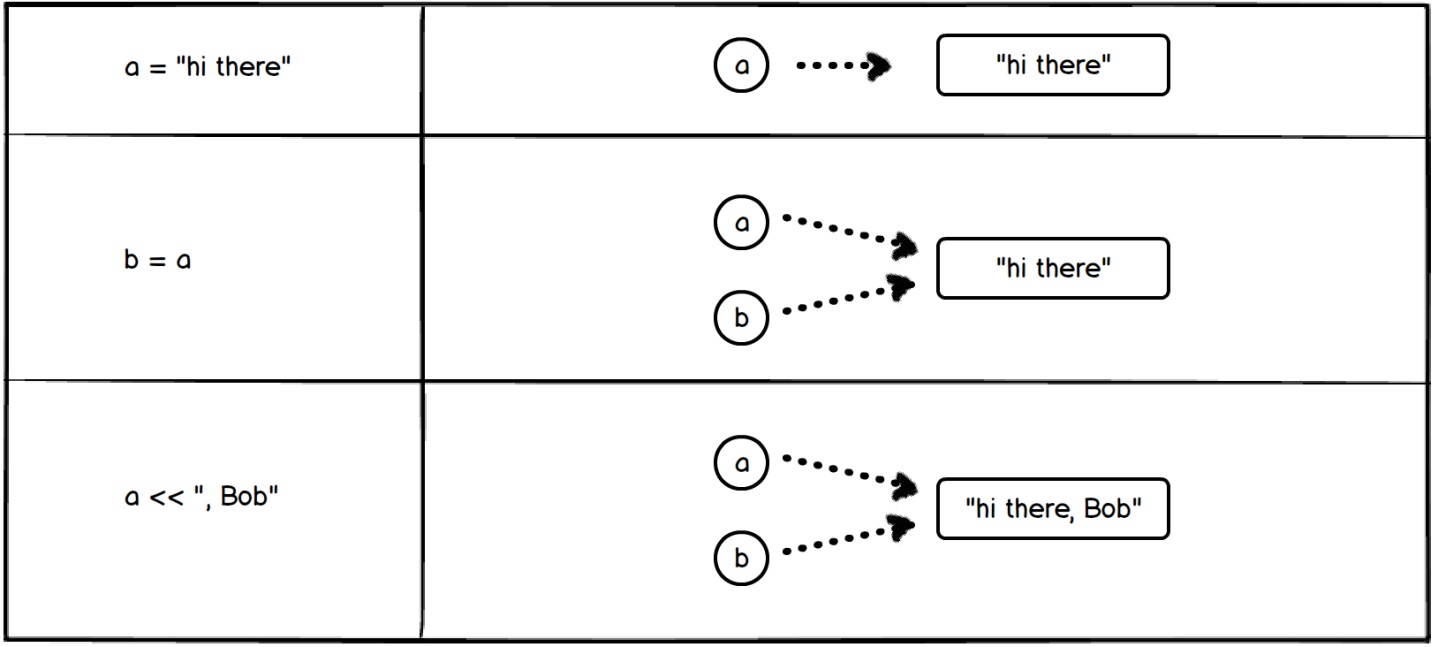
=> #<MatchData "b">

**Standard Classes**

Use the likes of Math class to complete Math orientated tasks.

**Variables as Pointers**





**Blocs and Procs**

def take\_block(&block)

block.call

end

take\_block do

puts "Block being called in the method!"

end

talk = Proc.new do |name|

puts "I am talking to #{name}"

end

talk.call "Bob"

**Exception Handling**