CMSC330 NOTES

I. Intro

- Semantics: the meaning of sentences/languages
- Syntax: the structures of the language
- **Why so many languages?**

=> There are many programming languages because different languages have been designed to solve specific problems and cater to various requirements

II. Ruby

1. Comments: Use #

2. Print: Use puts

```
Example:    puts "hello world!"     #output: hello world!
            puts "abc" "ABC"        #output: abcABC
            puts 3 + 4              #output: 7
            puts 3 + "abc"          #output: TypeError
            puts "abc"*3            #output: abcabcabc
            puts 3*"abc"            #output: TypeError
```

3. Typing
   - Type Checking: The process of determining a variable's type
     + Dynamic typing: Type checking is performed at runtime
     + Static typing: Type checking is performed at compile time
   - Explicit/Implicit Typing:
     + Manifest (explicit): explicitly telling the compiler the type of new variables
       => Types are associated with variables
     + Latent (implicit): not needing to give a type to a variable
       => Types are associated with values

   >>> Ruby uses **dynamic** and **latent** typing

4. "Primitive" Data Types

1. Integer

- Arithmetic Operations: +, -, *, /, % (modulus), ** (exponentiation)
- Convert to other data types:

```
+ Float:          to_f            3.to_f      #3.0
+ String:         to_s            3.to_s      #"3"
+ Binary String:  to_s(2)         3.to_s(2)   #"11"
```

- Bitwise Operations: AND (&), OR (|), XOR (^), NOT (~), left shift (<<), and right shift (>>)
- Hexadecimal and binary representations: 0x and 0b
- Notes: 1_000_000 is also Integer

2. Float (Similar to Integer)

Notes:

2. and .0 are not valid for floats
2.0/2 = 1.0 where 2/2 = 1
Instead of doing Math.sqrt(3), we can do 3 ** 0.5

3. String

- Create strings: Use either single quotes or double quotes
- Concatenation and repetition: str + str and str * int (int * str doesn't work)
- String indexing and slicing: str = "Hello world"

```
+ Access individual character:      str[4]          # "o"
                                    str[20]         # nil
+ Extract substrings:               str[6..]        # "world"
                                    str[0,5]        # "Hello"
                                    str[3..7]       # "lo wo"
```

- Find substrings inside a string: str = "Hello world"

```
                                    str["Hello"]    # "Hello"
                                    str["hello"]    # nil
```

- Escaping characters: quotes(\") and newline(\n) and others
- String methods: Some helpful methods are length, reverse, upcase, downcase, capitalize, strip,...
- Regular expressions
- Interpolation: using #{expression} "I'm #{2023-2001} years old"
  Note: Strings created by single quotes doesn't allow interpolation
- Convert to numbers: "101".to_i => 101 "101".to_i(2) => 5
- Compare strings: == (https://medium.com/@khalidh64/difference-between-eql-equal-in-ruby-2ffa7f073532)

4. Symbol

- Creation: Using a colon followed by the identifier, such as :hello
- Immutable: Their value cannot be changed
- Unique: Two symbols with the same name refer to the same object

5. Array

- Creation: arr = [] or arr = [1,2,3] or arr = Array.new or arr = Array.new(10,1)
- Indexing: using arr[index], starting at 0
- Slicing: Subarrays can be extracted (similar to String)
- Modification: Arrays are mutable, elements can be added, removed, modified
- Iteration: By using for loops or using each (https://mixandgo.com/learn/ruby/each)

```
for i in arr      | for i in 0..arr.length-1     | arr.each{|x|
    puts i        |    puts arr[i]               |     puts x
end               | end                          | x}
```

- Adding and Removing methods: arr = [1,2,3,4] /Examples below are separately/
  - + **push**(element1, element2,...): add elements to the end of an array
    ```
    arr.push(5,6)          # => arr = [1,2,3,4,5,6]
    ```
  - + **pop** or **pop**(n): remove the (n) last element of an array and return it
    ```
    arr.pop                # => 4
    arr.pop(2)             # => [3,4]
    ```
  - + **unshift**(element1, element2,...): add elements to the beginning of an array
    ```
    arr.unshift(0,1,2)     # => arr = [0,1,2,1,2,3,4]
    ```
  - + **shift** or **shift**(n): remove the (n) first element of an array and return it
    ```
    arr.shift              # => 1      (arr = [2,3,4])
    arr.shift(3)           # => [1,2,3](arr = [4])
    ```
  - + **delete**(value): remove an element from an array based on its value
    ```
    arr.delete(3)          # => 3      (arr = [1,2,4])
    arr.delete(5)          # => nil    (arr = [1,2,3,4])
    ```
  - + **delete_at**(index): remove an element from an array based on its index
    ```
    arr.delete_at(1)       # => 2      (arr = [1,3,4])
    ```
- Dynamic Sizing:
  ```
  arr = []
  arr[5] = 5 # => arr = [nil,nil,nil,nil,nil,5]
  ```
- Array Operations: A = [1,2,3,4,5] B = [4,5,6,7,8]
  - + add: A + B # => [1,2,3,4,5,4,5,6,7,8]
  - + difference: A - B # => [1,2,3] B - A # => [6,7,8]
  - + union: A | B # => [1,2,3,4,5,6,7,8] B | A # => [4,5,6,7,8,1,2,3]
  - + intersect: A & B # => [4,5]
- Some other helpful methods in Array:
  - + first: return the first element of an array (arr[0])
  - + last: return the last element of an array (arr[-1])
  - + **length** or size: return the number of elements in an array
  - + empty?: return true if the array is empty, false otherwise
  - + **include?(element)**: return true if the array includes the element, false
  - + **index(element)**: return the index of the first occurrence, nil if not found
  - + **sort** or sort!: sort an array
  - + **reverse** or reverse!: reverse an array
  - + **map** or map!:
    ```
    arr = [1,2,3,4,5]
    arr.map!{|x| x**2} # arr = [1,4,9,16,25]
    ```
  - + **select** or select!:
    ```
    arr = [1,2,3,4,5]
    arr.select!{|x| x % 2 == 1}   # arr = [1,3,5]
    ```
  - + reduce:
    ```
    arr = [1,2,3,4,5]
    arr.reduce(:*)    # => 120 (1*2*3*4*5)
    ```
  - + **each**:
    ```
    arr = [1,2,3,4,5]; sum = 0
    arr.each{|x| sum += x}; puts sum # output: 15
    ```
  - + **join**(Array => String):
    ```
    arr = [1,2,3,4,5]
    str = arr.join(",") # => str = "1,2,3,4,5"
    ```

Note: Methods with ! will change the original array instead of creating new one

6. Hash

7. Boolean

n. Object Oriented Programming

- Everything is a class

```
Example:    a = "Hello"
            a.class          #String
            3.class          #Integer
            3.14.class       #Float
            true.class       #TrueClass
            nil.class        #NilClass
```

- Objects have methods

```
Example:    3.methods
```