Software Explained Simply

Episode 2 - Variables, Types, and Data Structures

Outline

- HTML/CSS vs “programming languages”
- Variables
- Types
- Simple data structures

“Programming languages”

- Can perform logic or instructions
- Deals with data (numbers, text, dates, etc.)
- Controls the flow and output of the program

Example languages

A programming language:

- Stores and retrieves data
- Performs calculations or transformations on data
- Do repetitive tasks easily

(These examples are in Ruby)

Types

50 (integer, number)

2.15 (float, decimal)

"John" (string, text)

true / false (boolean)

Types

```
125 - the number one hundred twenty-five

"125" - the string with the characters 1, 2, and 5
```

Variables (assignment)

```
x = 25

name = "John"

is_hungry = true
```

Variables (assignment)

```
x = 25

x + 1 => 26

x - 5 => 20

x * 4 => 100

x / 5 => 5

x % 2 => 1  (modulus operator or “remainder”)
```

Integer vs Float Division

```
6 / 4 => 1

6 / 4.0 => 1.5

3 / 0.34 => 8.823529411764705 3
```

Variables (assignment)

```
x = 25

y = x + 5

z = x + y
```

Variables (assignment)

```
x = 25

y = x + 5

z = x + y
```

```
x = 25

y = 30

z = 55
```

Variables (assignment)

```
x = 25        # x is 25

x = 5         # x is 5

x = x - 1     # x is 4
```

Variables (assignment)

```
x = 25

y = x + 5

x = 1
```

Variables (assignment)

```
x = 25

y = x + 5

x = 1
```

```
_

x = 1

y = 30
```

Variable Styles

Ruby

```
age = 65

is_hungry = true
```

JavaScript

```
var age = 65;

var isHungry = true;
```

Data Structures

- Give options on how to store the data in a way that is optimized for its use.

- Possible considerations:
  - Is the data in a specific order?
  - Does the data need to stay grouped together?
  - How much or what types of data are there?

Common Data Structure: Arrays

- **Ordered** list of elements
- Usually represented by **[ ]**
- Can access elements at a particular index (“give me the first element, the 3rd element, etc.”)

Using Arrays

```
x = [1, 2, 3, 4]
```

Using Arrays

```
x = [1, 2, 3, 4]
```

```
x = [1, true, 3, "4"]
```

Using Arrays

```
x = [1, 2, 3, 4]
```

```
x = [1, true, 3, “4"]
```

```
x = [1, ["hi", "bye"], 3, "4"]
```

Accessing Elements Using [ ]

```
x = [1, 2, 3, 4]
```

Accessing Elements Using [ ]

```
x = [1, 2, 3, 4]

x[0] => 1
```

Accessing Elements Using [ ]

```
x = [1, 2, 3, 4]

x[0] => 1

x[1] => 2
```

Accessing Elements Using [ ]

```
x = [1, 2, 3, 4]
```

x[0] => 1

x[1] => 2

x[2] => 3

x[3] => 4

Accessing Elements Using [ ]

```
x = [1, 2, 3, 4]

x.length => 4
```

Accessing Elements Using [ ]

```
x = [1, 2, 3, 4]

x.length => 4

x[4] => nil
```

Changing Arrays

```
x = [1, 2, 3, 4]
```

```
x[4] = 5
```

Changing Arrays

```
x = [1, 2, 3, 4]
```

```
x[4] = 5
```

```
=> [1, 2, 3, 4, 5]
```

Changing Arrays (concatenation)

```
x = [1, 2, 3, 4]
```

```
x + [5]
```

```
=> [1, 2, 3, 4, 5]
```

Changing Arrays

```
x = [1, 2, 3, 4]

x.push(5)
```

```
=> [1, 2, 3, 4, 5]
```

Changing Arrays

```
x = [1, 2, 3, 4]

x.push(5)

=> [1, 2, 3, 4, 5]

x.pop

=> [1, 2, 3, 4]
```

Array Summary

- An **ordered** list of elements
- Access elements by using **[ ]**
- Useful for storing data in order, or accessing particular elements (“give me the second item”)
- Zero-indexed — the first item is at index `0`, the last item is at index `length - 1`

Common Data Structure: Hashes

- Used for **associated** data
- Usually represented by **{ }**
- Stores data under a specific key
- Sometimes called Maps, Dictionaries, or Keyword Lists

Using Hashes

```
x = {

   "first_name" => "John",

   "last_name" => "Smith"

}
```

Using Hashes

```
x = {"first_name" => "John", "last_name" => "Smith"}
```

Using Hashes

```
x = {"first_name" => "John", "last_name" => "Smith"}

x[0] => nil
```

Using Hashes

```
x = {"first_name" => "John", "last_name" => "Smith"}

x[0] => nil

x["first_name"] => "John"
```

Changing Hashes

```
x = {}

x["first_name"] = "John"

=> {"first_name" => “John”}
```

Changing Hashes

```
x = {}

x["first_name"] = "John"

=> {"first_name" => “John”}

x.delete("first_name")

=> {}
```

Side-note about Ruby hashes

```
x = {"name" => "John Doe", "age" => 25}

_
```

```
x = {name: "John Doe", age: 25}

x[:name] => "John Doe"
```

Hash Summary

- An **associated** set of keys and elements
- Access elements by using **[<the-key>]**
- Useful for natural groupings of data about the same thing (ex: all of the attributes of a person)

Arrays vs Hashes

```
x = [1, 2, 3]

x = {"first" => 1, "second" => 2, "third" => 3}
```

Arrays vs Hashes

```
x = [1, 2, 3]

x = {"first" => 1, "second" => 2, "third" => 3}

x = ["John Doe", 25]

x = {"name" => "John Doe", "age" => 25}
```

Arrays vs Hashes

```
people = [

  {"name" => "John Doe", "age" => 25},

  {"name" => "Jane Doe", "age" => 22}

]
```

Homework

- In the language of your choice become comfortable with:
  - Variables
  - Numbers
  - Strings
  - Arrays
  - Hashes

Homework

- Also do the following:
    - Describe yourself using a Hash
    - Describe your family using an Array of Hashes
    - Test out using variables inside of data structures

Thanks!

@johnmosesman