CHAPTER 3:
CORE PROGRAMMING ELEMENTS

Introduction to Computer Science Using Ruby

Variables

- A **variable** is a single datum or an accumulation of data attached to a name
  - The datum is (or data are) stored in memory
  - The name is mostly **arbitrary** but should be chosen wisely
    - Variables can have almost any name
    - Names should improve the readability of the code

Variables in Ruby

- Use the format **variable_name = value**
- This format also **initializes** variable data

```
irb(main):001:0> a = 4
     => 4
irb(main):002:0> b = 3
     => 3
```

- The **equal sign (=)** assigns the right-hand side to the variables in the left hand side

Common Standards for Variable Names

- Cannot start with an integer
  - Ex: *bank1*, not *1bank*
- Should avoid having special characters
  - Ex: *money_spent*, not *$_spent*
  - Special characters have specific uses in many languages, including Ruby
- Should explain the data they stand for
  - Ex: *balance*, not *b*
- Should complement the programming language style
  - Ex: *check_balance*, not *checkBalance* or *checkbalance*
    - Names with **underscores** match Ruby's style
    - Last two names are different because names are **case sensitive**

Variables

- Most programming languages assign the variable's data to **an address in memory**
  - The programmer does not need to decide the location

Memory – Figure 3.2

Variables

- Constants are "**variables**" that are assigned a value that "cannot" be changed
  - Constant names contain only capital letters
  - Ex: PI or PAI for 3.14159286 (π) ; C for speed of light constant

Data Classes

- Variables can represent words, numbers, and other entities depending on their **data classes**
- A **data class** indicates the properties of the data stored in a variable
  - The nomenclature "**Data Type**" is used in non-object oriented languages
  - The notion of "**Class**" has far more reaching meaning than "**Type**"

Data Classes

Data classes can:

- Specify the domain of valid values of variables
- Determine the amount of memory allocated
- Determine the operations allowed for (or on) it

Data classes in **Ruby** include:

- Integers (Fixnum and Bignum)
- Floats
- Strings
- Boolean
- Many more… Stay tuned

Data Classes in Ruby: Fixnum

- Natural numbers in **integer range** and their negatives
- Integer values range from **-2,147,483,648** to **2,147,483,647** in a **32-bit system**
  - Standard in almost all languages
  - **Note:** asymmetry between the positive and negative numbers

Data Classes in Ruby: Integers

Fixnum

- Stores values **within** the 32-bit range

```
irb(main):01:0> x = 5
  => 5
```

Bignum

- Stores values **outside** the 32-bit range

```
irb(main):02:0>
  x=1_000_000_000_000_
  000
=>1000000000000000
```

  - **Note:** No use of commas with the numbers

Data Classes in Ruby: Float

- A decimal number that includes positive and negative values
- Can be defined using **decimal places** or **scientific notation**
  - *3.5e2* indicates *3.5 x* 10^2 in scientific notation

Float Examples:

```
irb(main):001:0> x = 5.0
=> 5.0
irb(main):002:0> x = -3.1415
=> -3.1415
irb(main):003:0> x = 3.5e2
=> 350.0
```


Data Classes in Ruby: Strings

- Character sequence surrounded by **quotes**
  - Both **double** (“) and **single** (‘) quotes can be used, but double quotes must be used if a single quote is inside a string

```
irb(main):001:0> x = ‘hello world’
  => hello world
irb(main):002:0> y = “hello,‘world’”
  => hello ‘world’
```

Basic Arithmetic Operators

- Used to perform **mathematical operations**
- Most are **binary operators** and require **two** operands

Symbol	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Power

Table 3.1

Basic Arithmetic Operators

- The **modulus operator,** %, is used to find the remainder when diving two integers

```
irb(main):001:0> x = 5%2
  => 1
```


Advanced Mathematical Functions

- Ruby's Math Module provides advanced mathematical functions, referred to as **Methods** (Table 3.2)

Method	Operation
sqrt()	Square Root
sin()	Sine
cos()	Cosine
tan()	Tangent
log()	Natural Log (ln)
log10()	Log (Base 10)

- Math Module methods are used in the following format:

Math.Function_name(Value)

```
irb(main):001:0> x = Math.sqrt(16)
  => 4
```


Use of Methods

- Ruby's Math Module provides advanced mathematical functions, referred to as **Methods**
- *There is a way to include a whole module (like Math), without the need to specify it with every use*

Input & Output: Direct Output

- The **puts** instruction displays text on the screen (i.e., standard out)

```
irb(main):001:0> puts "Hello World"
```

- Variables are displayed on the screen using **puts**
  - To use **puts** for a variable, enter the variable name without quotations

```
irb(main):002:0> text = "Hello World"
  => "Hello World"
irb(main):003:0> puts text
  => Hello World
```


Input & Output: Input Using Variables

- The **gets** instruction stores values that are entered from the keyboard (i.e., standard input device)
- Its format is very similar to **puts**

```
irb(main):001:0> age_input = gets
```

- **gets** stops the program and waits for the user to type
  - Type the **input**, then press enter

Input & Output: Input Using Variables

- **gets** will store values as character strings
- To change the data from one class to another (i.e., a string into an integer), you need to explicitly perform a **type (class) conversion**, usually creating a **new variable** of the appropriate class

Input & Output: Conversion

- **gets** will store character strings

```
irb(main):001:0> age_input = gets
```

  - If you typed 19, `age_input` will be the **string** "19", NOT the number 19
  - To convert "19" to 19, perform the following:

```
irb(main):002:0> age =
age_input.to_i
```

    - `.to_i` converts the contents of a variable to an integer

Common Programming Errors

- **Syntax errors** refer to code that Ruby cannot execute

```
irb(main):001:0> x = 1 + "hello"
Type Error: String can't be coerced into Fixnum
  from (irb):1:in '+'
  from (irb):1
```

- Ruby stops execution and tells the location where it had to stop

Common Programming Errors

- Error messages can seem **unrelated** to the problem

```
irb(main):002:0> x = hello
NameError: undefined local
  variable or method 'hello' for
main: Object
from (irb):2
```

- Ruby assumed that `hello` was a **variable** since strings have quotes

Common Programming Errors

- Ruby cannot catch **logic errors**
  - The program runs, but the results are incorrect
- Logic errors are often harder to find because the error's **location** is not given
  - A common logic error involves **integer division**
  - Ruby performs integer division correctly, but many casual users expect a different result

```
irb(main):003:0> 5/2
=> 2
```

    - A result of 2.5 may be expected, but it would not be an integer

Mixing Data Classes

- Ruby always tries to keep the **same data class** for all of its operands
  - Ruby will convert data classes when it has different ones in the **same arithmetic operation**
- To get a decimal from the previous example, add a **float** or perform an **explicit conversion**

```
irb(main):003:0>
  1.0*5/2
=> 2.5
```

- However, some data classes cannot be converted
  - Ruby will either create an **error condition**, or worse, produce an **incorrect result**

```
irb(main):002:0> x
= "hello".to_i
=> 0
```

NOTE possible version dependency!!!

Summary

- A **variable** is data attached to a name
- There are **common guidelines** to follow when creating variable names
- **Constants** are "variables" (really **values**) that never change
- **Programs** use various **methods** (operators and functions) available in each of the data classes to perform operations
- Ruby has many classes of operators and methods to perform math and other operations

Summary

puts

- The **puts** command is used to generate output on the screen (i.e., standard out)

gets

- The **gets** command is used to obtain information from the keyboard (i.e., standard in)

Three types of programming errors are **syntax errors, logic errors,** and **type errors**