UNIT 2A
An Introduction to Programming

Announcements

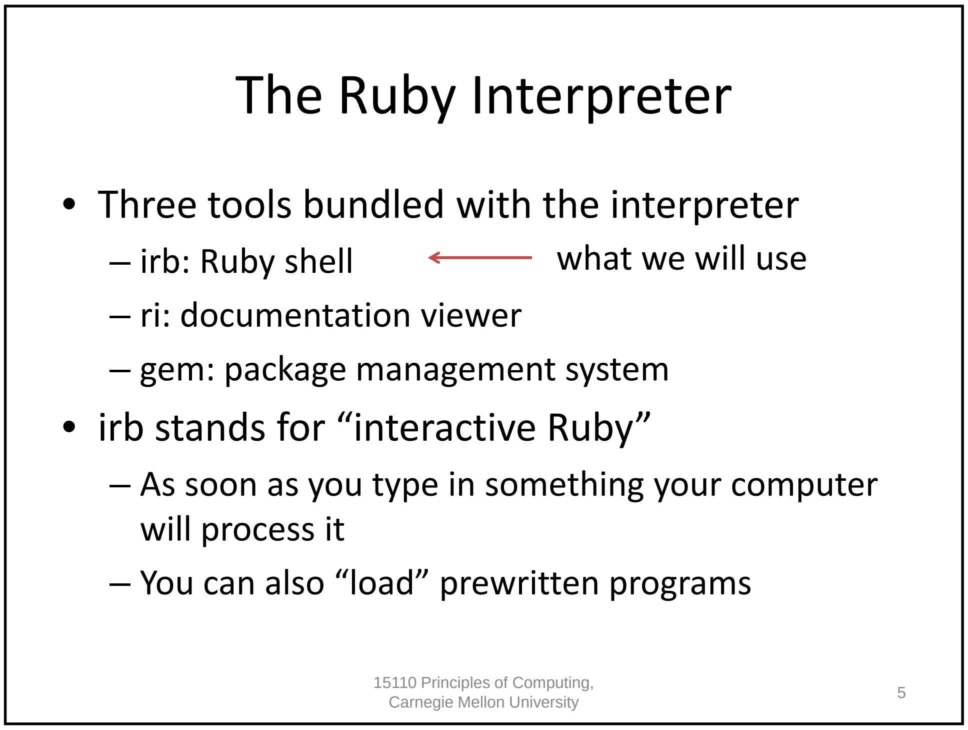
- Office hour locations about to be finalized. Check the office hours link on the Web page:
  - Wednesday GHC 5222 (6-7:50)
  - Thursday GHC 4215 (7-9:30)
  - Others to be announced soon
- If you had a problem submitting your lab and Autolab shows a grade less than 3, it is an error that we will fix.
- Please give special accommodation requests to the instructors

Last Week

- History Unit
  - Key enabling technologies: transistor, integrated circuit, microprocessor
  - Move from big to small, costly to cheap
  - Moore's law
  - Revolutionary ideas: stored program concept, internetworking, graphical user interfaces
  - People to remember: Babbage, Turing

This Week

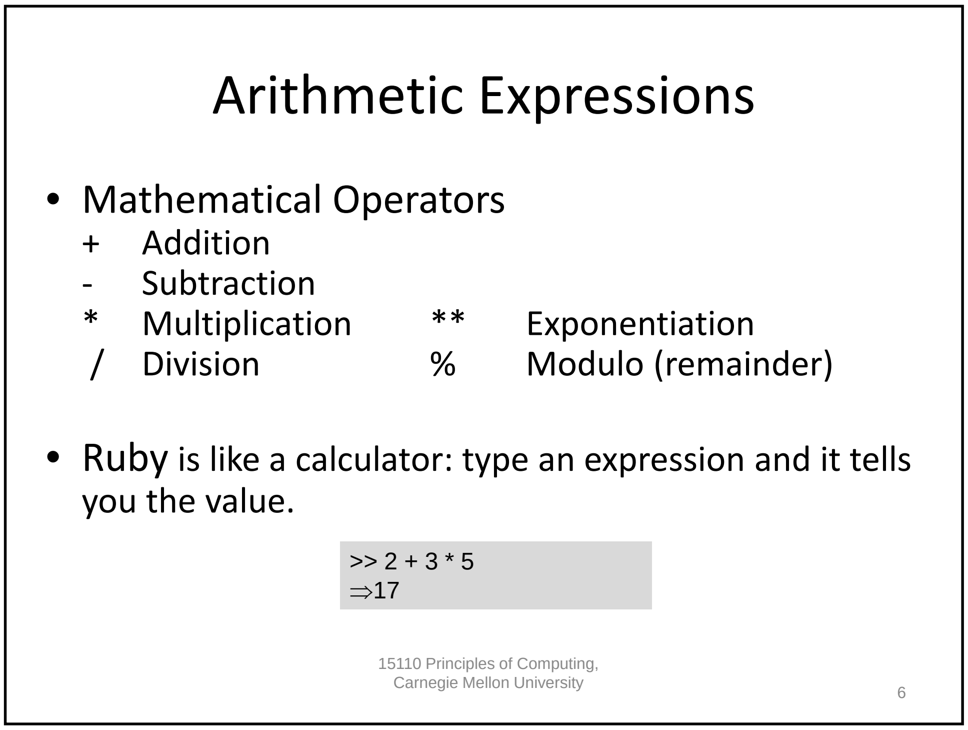
- Introduction to programming with Ruby
  - Basic data types: integer, float, string
  - Variables
  - Expressions
  - Function (method) definitions
  - Basic control structures
  - Predefined modules



The Ruby Interpreter

- Three tools bundled with the interpreter
  - irb: Ruby shell $\longleftarrow$ what we will use
  - ri: documentation viewer
  - gem: package management system
- irb stands for “interactive Ruby”
  - As soon as you type in something your computer will process it
  - You can also “load” prewritten programs

15110 Principles of Computing,
Carnegie Mellon University 5



Arithmetic Expressions

- Mathematical Operators

+	Addition		
-	Subtraction		
*	Multiplication	**	Exponentiation
/	Division	%	Modulo (remainder)

- Ruby is like a calculator: type an expression and it tells you the value.

```
>> 2 + 3 * 5
⇒17
```

15110 Principles of Computing,
Carnegie Mellon University 6

Expressions: Technical Points

Order of operator precedence:

`**` → `* / %` → `+ -`

Use parentheses to force alternate precedence

5 * 6 + 7 ≠ 5 * (6 + 7)

Left associativity except for **

2 + 3 + 4 = (2 + 3) + 4

2 ** 3 ** 4 = 2 **(3 ** 4)

Data Types

- Integers

```
4       15110    -53      0
```

- Floating Point Numbers

```
4.0     -0.8 03333333333333333
7.34e+014
```

- Strings

```
"hello" "A"   " "   "" "15110-s13"
```

- Booleans

```
true   false
```

George Boole, 1815-1864

Integer Division

In Ruby:

- 7 / 2 equals **3**
- 7.0 / 2.0 equals 3.5
- 7 / 2.0 equals …
- 7.0 / 2 equals …

Variables

- A variable is *not* an “unknown” as in algebra.
- In computer programming, a variable is simply a place where you can store a value.

```
>> a=5
=> 5
```

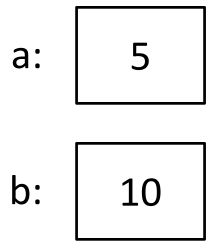
a: $\boxed{5}$

Variables

- A variable is *not* an “unknown” as in algebra.
- In computer programming, a variable is simply a place where you can store a value.

```
>> a=5
⇒5

>> b=2*a
⇒10
```



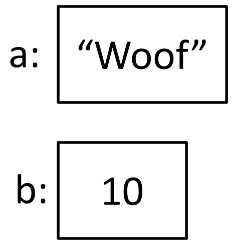
Variables

- A variable is *not* an “unknown” as in algebra.
- In computer programming, a variable is simply a place where you can store a value.

```
>> a=5
⇒5

>> b=2*a
⇒10

>> a=“Woof”
⇒“Woof”
```



Variable Names

- All variable names must start with a lowercase letter.
- The remainder of the variable name (if any) can consist of any combination of uppercase letters, lowercase letters, digits and underscores (_).
- Identifiers in Ruby are case sensitive. Example: `Value` is not the same as `value`.

Write Your Own Methods

```
def tip (total)
    return total * 0.18
end


>> tip(20)
⇒3.6
>> tip(135.72)
⇒24.4296
```

Method Syntax

```
def  methodname ( parameterlist )
   instructions
end
```

- `def` and `end` are reserved words and cannot be used as variable names.

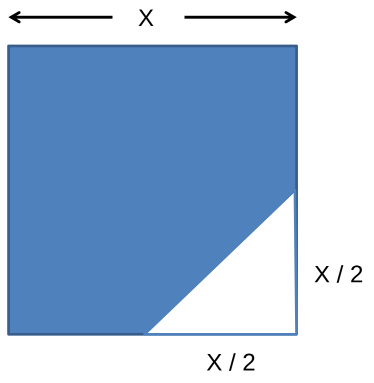
Methods (cont’d)

- The name of a method follows the same rules as names for variables: start with a lowercase letter.
- The parameter list can contain 1 or more variables that represent data to be used in the method’s computation.
- A method can also have no parameters.

```
def hello_world()
   print "Hello World!\n"
end
```

(`\n` is a newline character)

Example: Countertop



countertop.rb

parameter

```
def compute_area(side)
  square = side * side
  triangle = 0.5 * side / 2 * side / 2
  area = square - triangle
  return area
end
```

To run the function in `irb`:

```
 load "countertop.rb"
 compute_area(109)
```

argument
(run function with side = 109)

Methods (cont’d)

- To run a method, we say we “call” the method.
- A method can return either one answer or no answer to its “caller”.
- The `hello_world` function does not return anything to its caller. It simply prints something on the screen.

different

- The `compute_area` function does return its result to its caller so it can use the value in another computation:
  `compute_area(109) + compute_area(78)`

Methods (cont’d)

- Suppose we write `compute_area` this way:

```
def compute_area(side)
   square = side * side
   triangle = 0.5 * side/2 * side/2
   area = square - triangle
   print area
end
```

- Now this computation does not work since each function call prints but returns nothing:
  `compute_area(109) + compute_area(78)`

escape.rb

(a function with two parameters)

```
def compute_ev(mass, radius)
  # computes escape velocity
  univ_grav = 6.67e-011
  return sqrt(2*univ_grav*mass/radius)
end
```

Comments begin with #

To run the function for Earth in `irb`:

```
load "escape.rb"
compute_ev(5.9742e+024, 6378.1)
```

Built-In Functions (Methods)

- Lots of math stuff, e.g., sqrt, log, sin, cos

```
r = 5 + Math.sqrt(2)

alpha = Math.sin(Math::PI/3)
```

Using predefined modules

- `Math` is a predefined module of methods that we can use without writing their implementations.

```
Math.sqrt(16)
Math::PI
Math.sin(Math::PI / 2)
```

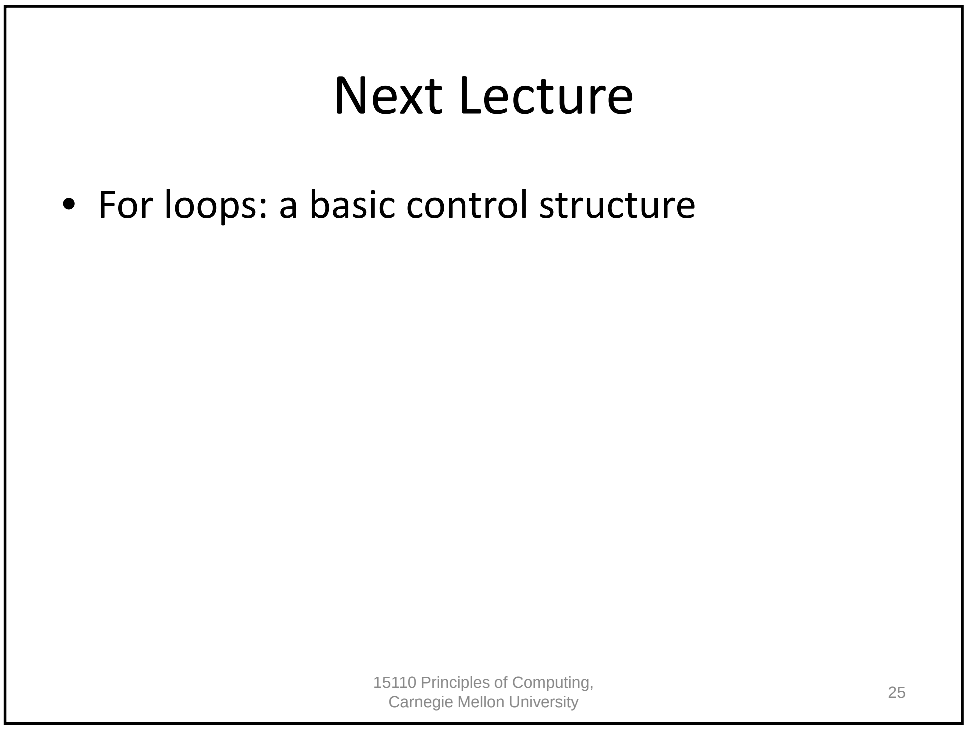
- If we are going to use this module a lot, we can include it first and then leave off the module name when we call a function.

```
include Math
sqrt(16)
sin(PI / 2)
```

What Could Possibly Go Wrong?

alpha=5
2 + alhpa $\longleftarrow$ syntax error

3/0 $\longleftarrow$ semantic errors,
sqrt(-1) e.g. calling a function with
sqrt(2, 3) a wrong argument type



Next Lecture

- For loops: a basic control structure