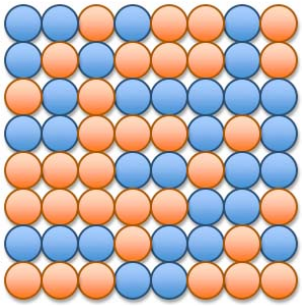


UNIT 2A
An Introduction to Programming

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
```

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  -0.80333333333333333`

  `7.34e+014`

- Strings

  `"hello"  "A"  " "  ""  "7up!"`

- Booleans

  `true  false`

George Boole,
1815-1864

Integer Division

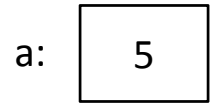
In Ruby:

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

Variables

- A variable is n*ot* an “unknown” as in algebra.
- In computer programming, a variable is simply a place where you can store a value.

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



Variables

- A variable is n*ot* 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: 5

b: 10

Variables

- A variable is n*ot* 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”
```

a: “Woof”

b: 10

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`.

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)
```


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)

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.
- 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)
```

What Could Possibly Go Wrong?

```
alpha=5
2 + alhpa

3/0
sqrt(-1)
sqrt(2, 3)

start = 35
end = 37
```