Variables

- Can hold persistent data inside our objects
- Can be used to represent the state of an object, or simply a temporary value for computation
- `+ variables:`
  `name` `(`*`type`*`).`

  Example:

```
+ variables:
    counter (int).
    angle, distance (float).
    toFood (vector).
```

Types

- `int`
- `float`
- `list`
- `object`
- `vector`
- `matrix`

Assigning Variables

- `myLight = new BraitenbergLight.`
- `myInt = 4.`
- `myInt *= 7.`
- `myInt++.`

Using variables for planning

- Use a variable to specify a current state or goal

Example:

```
currentTarget (object).

if currentTarget: {
     self pursue target currentTarget.
} else {
     currentTarget = (self pick-target).
}
```

`int` and `float`

- `int`: a whole number (1, 4, -3, etc)
- `float`: a real number (1.2, -4.3, 3.14, etc)
- `floats` are also sometimes called `doubles`
- Mathematical operators: +, -, /, *, %
- ints and float can be converted, but converting from float to int loses precision

Vectors

- points or vectors in 3D space
- `vectorVariable = (x, y, z)`
- `vectorVariable::x`, `::y` and `::z` give access to individual vector components

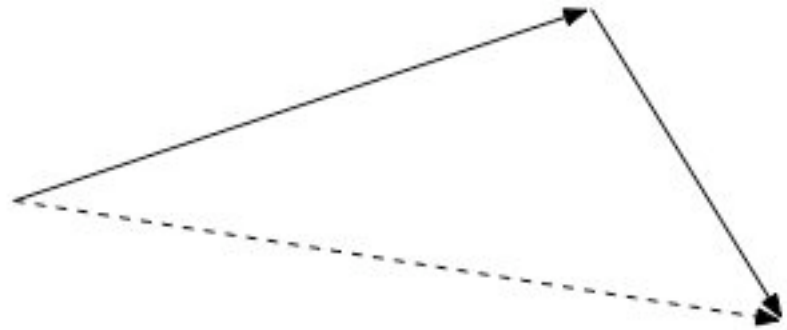
Examples:

```
myVector = (1.0, 2.0, 3.0).

myVector::x = 9.0.

print myVector::y.
```

Vector arithmetic



- `vector + vector`, vector addition
- `vector - vector`, vector subtraction
- `vector * float`, vector scaling
- `| vector |`, vector length

Useful vector examples:

- The vector pointing from one agent to another:

```
(object2 get-location) - (object1 get-location).
```

- "Normalize" a vector:

```
vector / | vector |.
```

- Random vector:

```
random[(10, 20, 30)].
```

Random values

- `random[`*`maxValue`*`].`
- Works with ints, floats and vectors.

Examples:

```
x = random[10].

self set-color to random[(1.0, 1.0, 1.0)].

randomLocation = random[(10, 10, 10)] - (5, 5, 5).

self set-speed to random[1.0].
```

Lists

- Hold groups of variables (of any type)
- `listVariable{n}`, the Nth item in the list
- `{ x, y, z }`, a list containing 3 items

Examples:

```
myList = { 1, 2, 3 }.

myList{0} = 5.

print myList{0}
```

List operators

- `push` *`value`* `onto` *`list`*`.`
  (adds *value* to the end of *list*).
- `pop` *`list`*`.`
  (removes and returns the last item in *list*).
- `|` *`list`* `|.`
  (the length of the *list*–the number of items it contains)

`foreach`-loop

- iterates through a list
- `foreach` *`item`* `in` *`list`*`: ...`

Example:

```
foreach myObject in myList: {
     print (myObject get-location).
}
```

`for`-loop

- iterates through a series of numbers
- `for` *`initializer, test, iterator`*`:`
  `...`

  Example:

```
for n=0, n<5, n = n + 1: {
     print “the value of n = $n”.
}
```

while-loop

- Repeats an action while a statement is true
- `while` *`test`*`: ...`

```
Example:

while x < 10: {
     print “x = $x”.
     x++.
}
```

True or False?

- Compare values with "==","<=","<",">", ">=" and "!="
- Numbers are "true" if they do not equal zero, otherwise they are "false"
- Objects are "true" if they hold a valid instance (created with `new`), otherwise they are "false"
- Vectors are "true" if their length is *not* zero, otherwise they are "false"
- Combine tests with "and" (`&&`), "or" (`||`)
- Negate a test with "!"

More about conditional statements

- Loop actions can be single statements, which require no braces:
  - `if x == 1: print “yes!”.`
  - `foreach i in agents: print i.`
- Loop actions with multiple statement must be wrapped in braces:
  - ```
    if x == 1: {
        print “yes!”.
        print “I really love the variable x!”.
    }
    ```
    ```

Defining methods

- Defines a behavior that your agent can execute
- Can be called internally, like from an agent's iterate method
- Can be called externally by other agents

Defining methods

- `to` *`methodName`*`:`
  `...`
- `to` *`methodName [ argument definitions ]`*`:`
  `...`
- An argument definition consists of:
  `keyword name` `(`*`type`*`)`

Examples:

```
+ to print-hello:
      print “hello!”.

+ to print-message with-text message (string) with-number num (int):
      print “the message is $message, the number is $num”.
```

Overriding methods

- Classes inherit behaviors from superclasses
- We can override these methods to customize our agent's behaviors
- We call the superclass method if we want the original behavior in addition to our own

Examples:

```
+ to eat food theFood (object):
      print "yummy!".
      super eat food theFood.

+ to eat food theFood (object):
      print "I'm not hungry!".
```

Local Variables

- Variables used by a method for computation
- Always initialized to zero (or analogous value)
- Not saved between invokations

Example:

```
+ to count to total (int):
      counter (int).

      for counter=0, counter<total, counter++: {
          print “counter = $counter”.
      }
```

“Return” statements

- Stops the execution of a method
- “Returns” a value to the calling method

Example:

```
+ to get-closest-food:
    bestDistance (double).
    best, item (object).

    bestDistance = 200.

    foreach item in all Food: {

        if |(self get-location) - (item get-location)| < bestDistance: {
            best = item.
            bestDistance = |(self get-location) - (item get-location)|.
        }
    }

    return best.
```

Things to try...

- Continue to develop simple agent behaviors
- Use class variables for planning and maintaining an agent's "state"
- Define your own methods and begin to build a repertoire of agent behaviors
- Make a "plan" using a list (plan to eat the food in a certain order)