Parenthesis Vs. Square Brackets

Stéphane Ducasse and Damien Cassou

http://stephane.ducasse.free.fr/
stephane.ducasse@inria.fr

( ) vs. [ ]

- `( )` just changes the priority of an execution but the program is executed.
- `[ ]` blocks program execution: the program is NOT executed.

- Therefore use `[ ]` when you do not know if your program will be executed.
- When the message can change the execution of your program (if, while, ...) use a block.

```
n timesRepeat: [ self doSomething ]
```

- `timesRepeat:` executes a number of times its argument, therefore the argument is a block

( ) vs. [ ] Example

```
x includes: 3 ifTrue: [...]
```

- The message is read as `includes:ifTrue:` and does not exist

```
(x includes: 33) ifTrue: [ self doSomething ]
```

- We use `( )` and not `[ ]` for the receiver (`x includes: 33`) because this expression should be executed only once.
- we use `( )` because we should make sure that the compiler identifies two messages: `includes:` and `ifTrue:` and not just one `includes:ifTrue:`
- We should use parentheses because we want `includes:` to be executed first.

( ) vs. [ ] Example

```
x isNil ifTrue: [ self doSomething ]
```

- `ifTrue:` may execute or not its argument, therefore the argument is a block

( ) vs. [ ] Example

```
[ self start ] whileTrue: [ self doSomething ]
```

- `whileTrue:` may execute both its receiver and argument multiple times, therefore they are both a block.

Quiz

```
1 to: 100 do: ... self doSomething ...
```

```
x ifEmpty: ... self doSomething ...
```

Quiz

```
1 to: 100 do: [:i | self doSomething ]
```

```
x ifEmpty: [ self doSomething ]
```

Summary

- `( )` is about changing the order of a computation.
- `[ ]` is freezing the computation and controlling it.