

```
function randomPonyInRace() {
  const pony = { name: 'Rainbow Dash' };
  const position = 2;
  // ...
  return { pony, position };
}

const { pony } = randomPonyInRace();
```

And you will only have the pony!

## 3.6. Default parameters and values

One of the characteristics of JavaScript is that it allows developers to call a function with any number of arguments:

- if you pass more arguments than the number of the parameters, the extra arguments are ignored (well, you can still use them with the special `arguments` variable, to be accurate).
- if you pass fewer arguments than the number of the parameters, the missing parameter will be set to `undefined`.

The last case is the one that is the most relevant to us. Usually, we pass fewer arguments when the parameters are optional, like in the following example:

```
function getPonies(size, page) {
  size = size || 10;
  page = page || 1;
  // ...
  server.get(size, page);
}
```

The optional parameters usually have a default value. The OR operator will return the right operand if the left one is `undefined`, as will be the case if the parameter was not provided (to be completely accurate, if it is *falsy*, i.e `0`, `false`, `""`, etc.). Using this trick, the function `getPonies` can then be called:

```
getPonies(20, 2);
getPonies(); // same as getPonies(10, 1);
getPonies(15); // same as getPonies(15, 1);
```

This worked alright, but it was not really obvious that the parameters were optional ones with default values, without reading the function body. ES6 introduces a more precise way to have default parameters, directly in the function definition: