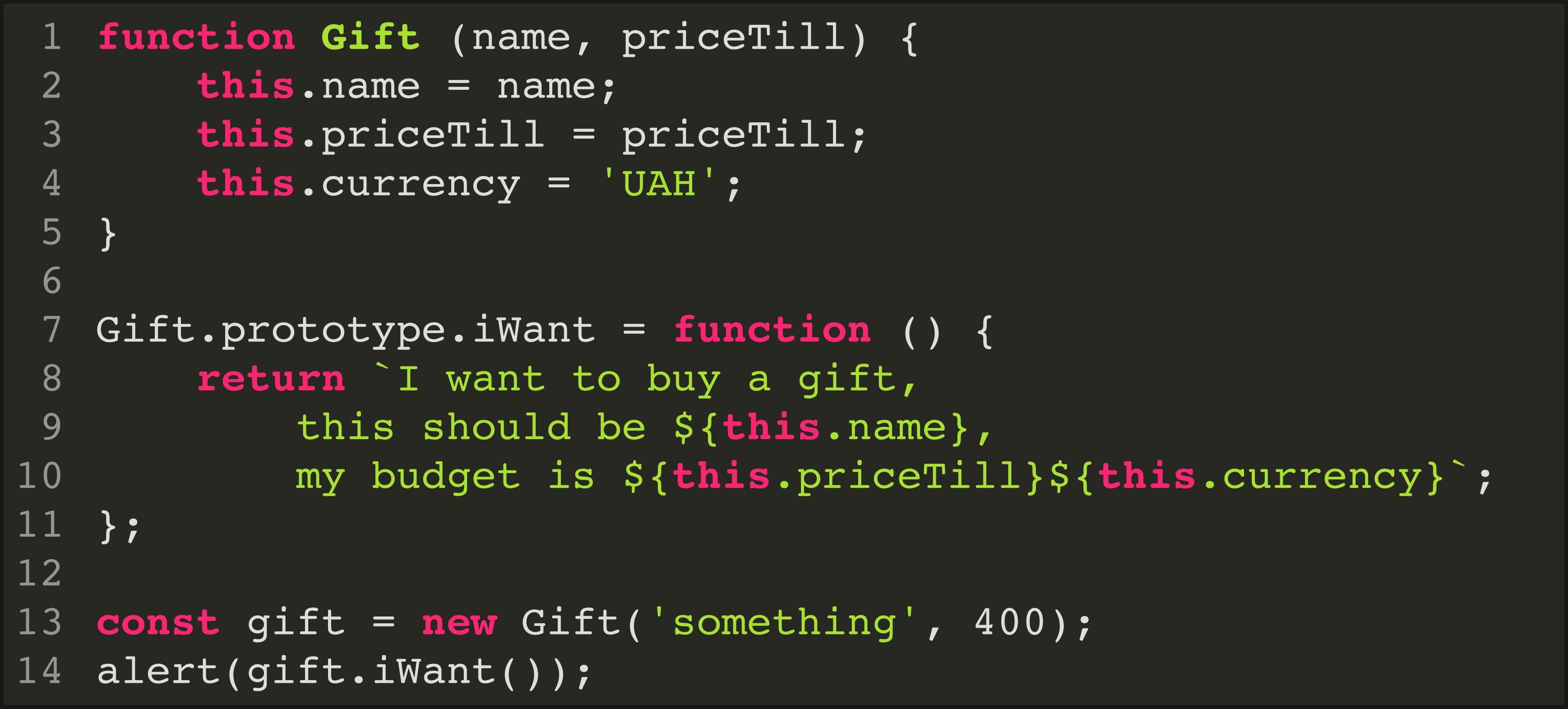
CONSTRUCTOR FUNCTIONS

WHAT IS IT?

Simple function, but it creates `new` objects

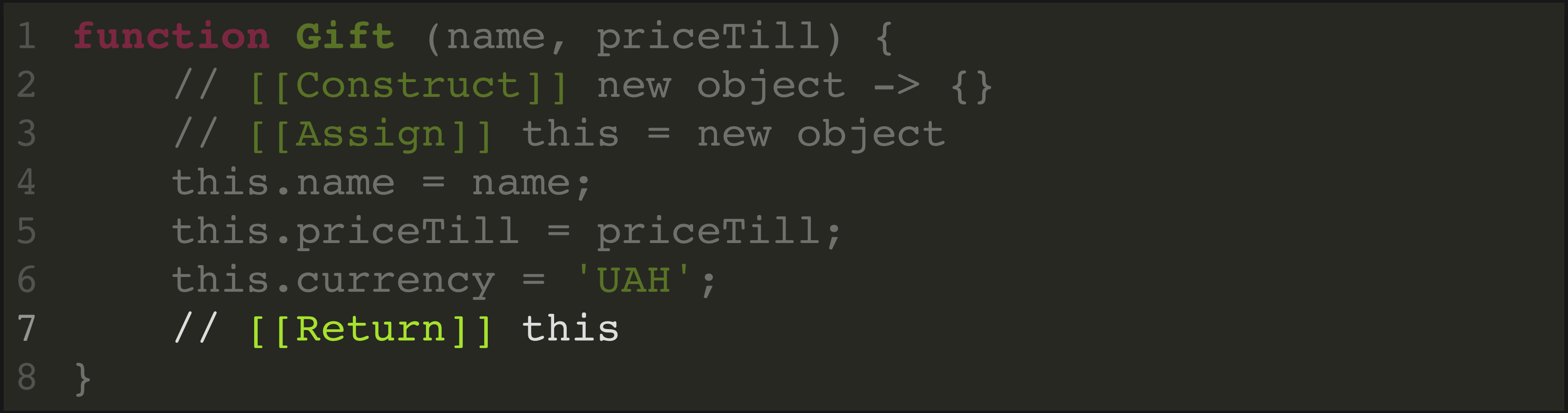


```javascript
function Gift (name, priceTill) {
    this.name = name;
    this.priceTill = priceTill;
    this.currency = 'UAH';
}

Gift.prototype.iWant = function () {
    return `I want to buy a gift,
        this should be ${this.name},
        my budget is ${this.priceTill}${this.currency}`;
};

const gift = new Gift('something', 400);
alert(gift.iWant());
```

```javascript
function Gift (name, priceTill) {
    // [[Construct]] new object -> {}
    // [[Assign]] this = new object
    this.name = name;
    this.priceTill = priceTill;
    this.currency = 'UAH';
    // [[Return]] this
}
```

```
function Gift (name, priceTill) {
    // [[Construct]] new object -> {}
    // [[Assign]] this = new object
    this.name = name;
    this.priceTill = priceTill;
    this.currency = 'UAH';
    // [[Return]] this
}
```

```javascript
function Gift (name, priceTill) {
    // [[Construct]] new object -> {}
    // [[Assign]] this = new object
    this.name = name;
    this.priceTill = priceTill;
    this.currency = 'UAH';
    // [[Return]] this
}
```



```
function Gift (name, priceTill) {
    // [[Construct]] new object -> {}
    // [[Assign]] this = new object
    this.name = name;
    this.priceTill = priceTill;
    this.currency = 'UAH';
    // [[Return]] this
}
```

INSTANCE

Name of a created entity from the constructor

instanceof

```
function Gift () {}
const gift = new Gift();
gift instanceof Gift // true
gift instanceof Object // true
gift instanceof Function // false
```

проверяет, присутствует ли объект Gift.prototype в цепочке прототипов gift

“ The instanceof operator tests to see if the prototype property of a constructor appears anywhere in the prototype chain of an object ”

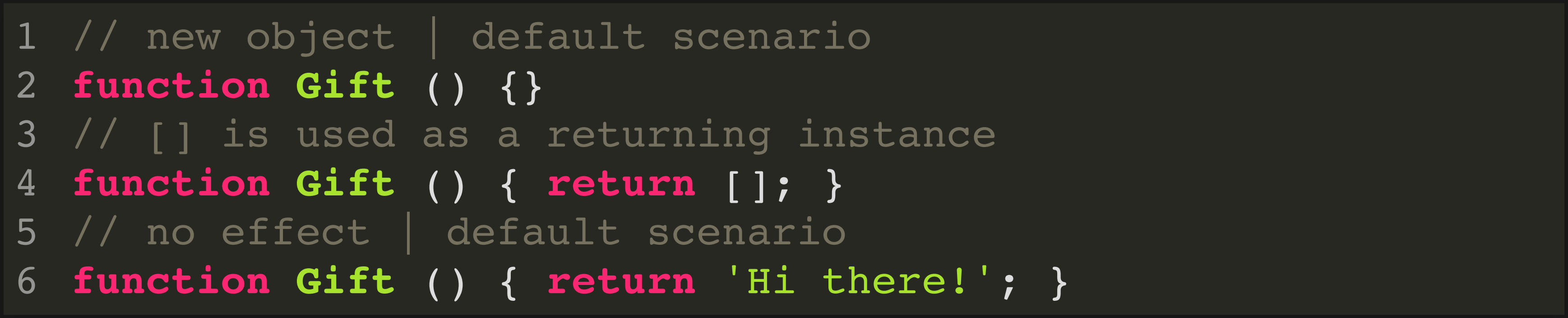
RETURNING VALUE

Return nothing if you ~~are human~~ want an instance

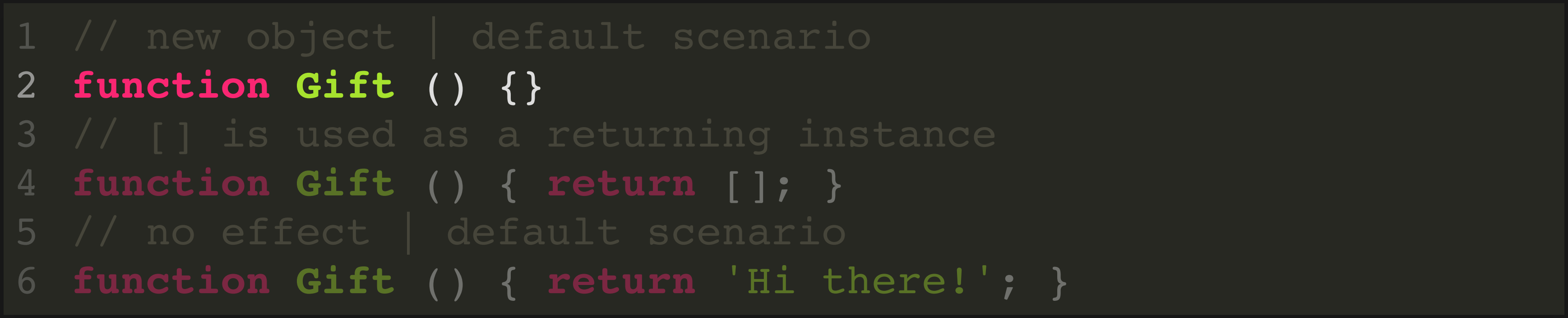
This is default scenario

Return object if you want to alter the [[Construct]]

Returning primitive will have no effect

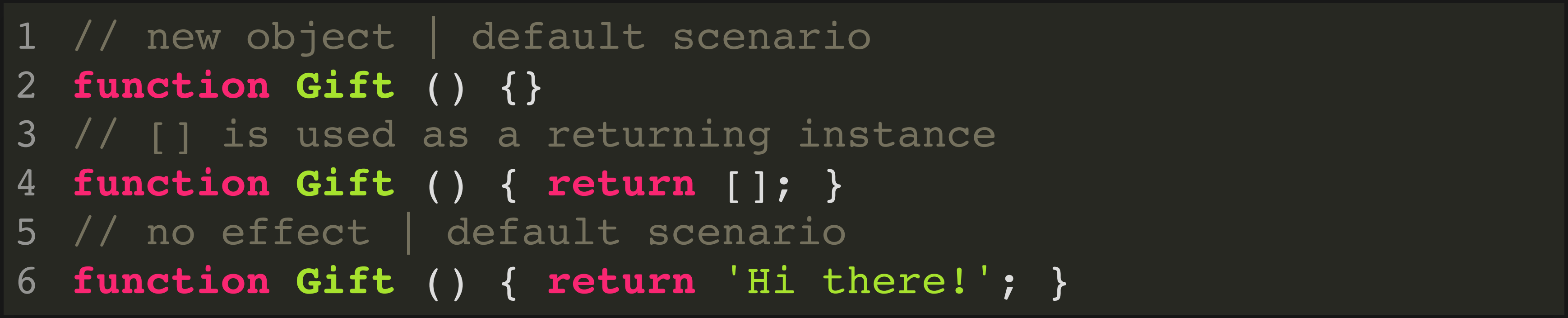


```
// new object | default scenario
function Gift () {}
// [] is used as a returning instance
function Gift () { return []; }
// no effect | default scenario
function Gift () { return 'Hi there!'; }
```



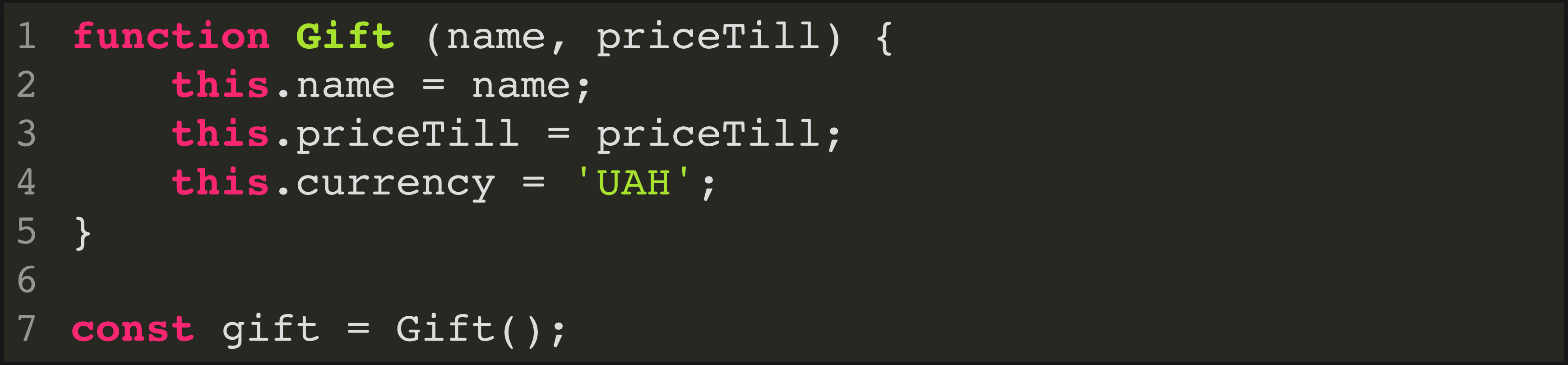
```
// new object | default scenario
function Gift () {}
// [] is used as a returning instance
function Gift () { return []; }
// no effect | default scenario
function Gift () { return 'Hi there!'; }
```

```
// new object | default scenario
function Gift () {}
// [] is used as a returning instance
function Gift () { return []; }
// no effect | default scenario
function Gift () { return 'Hi there!'; }
```

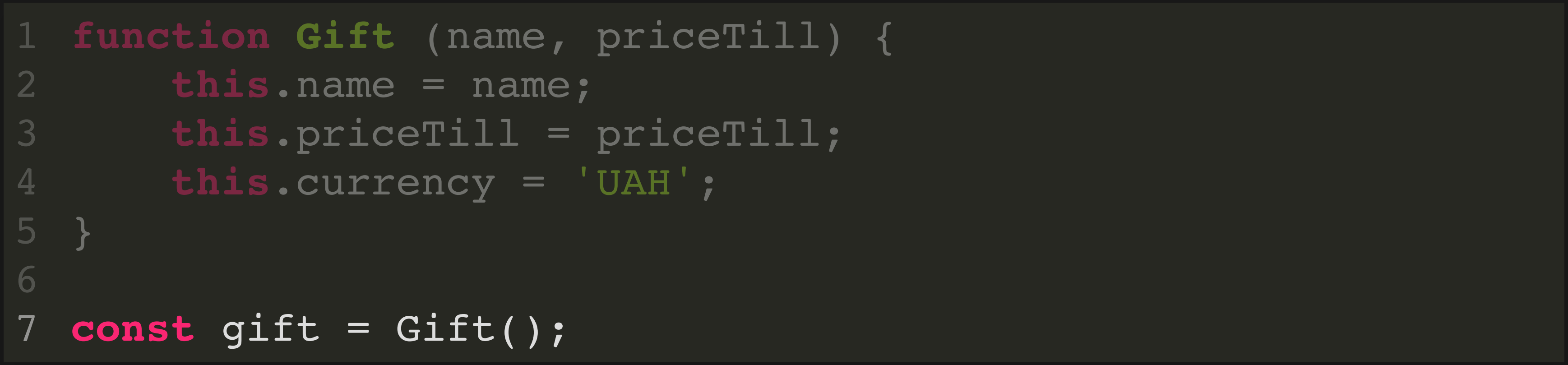


```
// new object | default scenario
function Gift () {}
// [] is used as a returning instance
function Gift () { return []; }
// no effect | default scenario
function Gift () { return 'Hi there!'; }
```

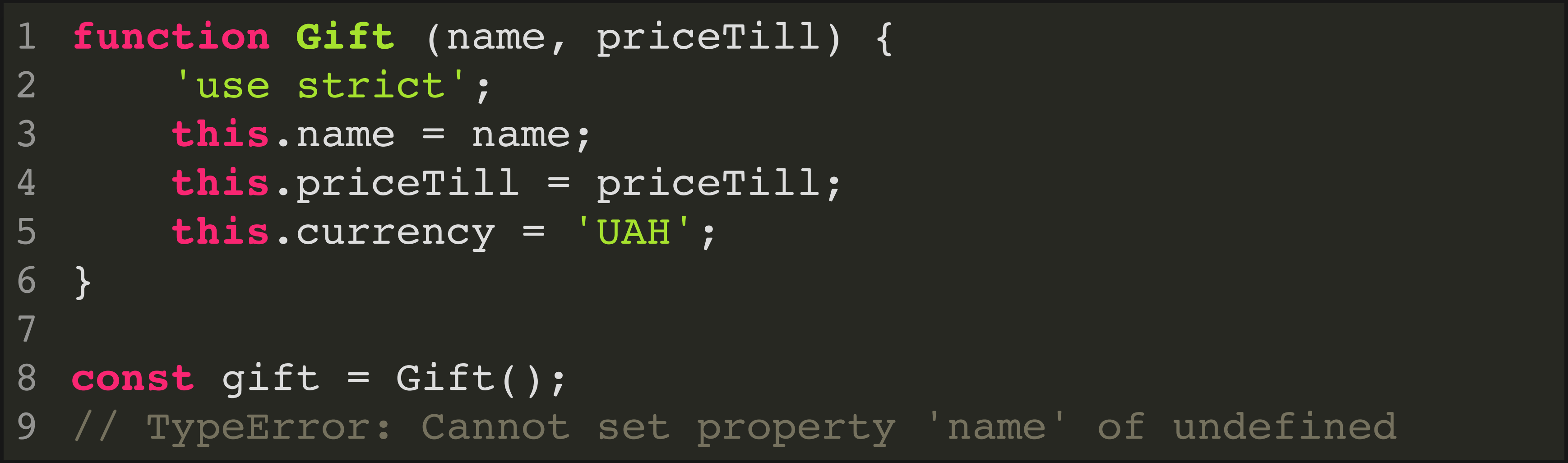
ALIEN USAGE



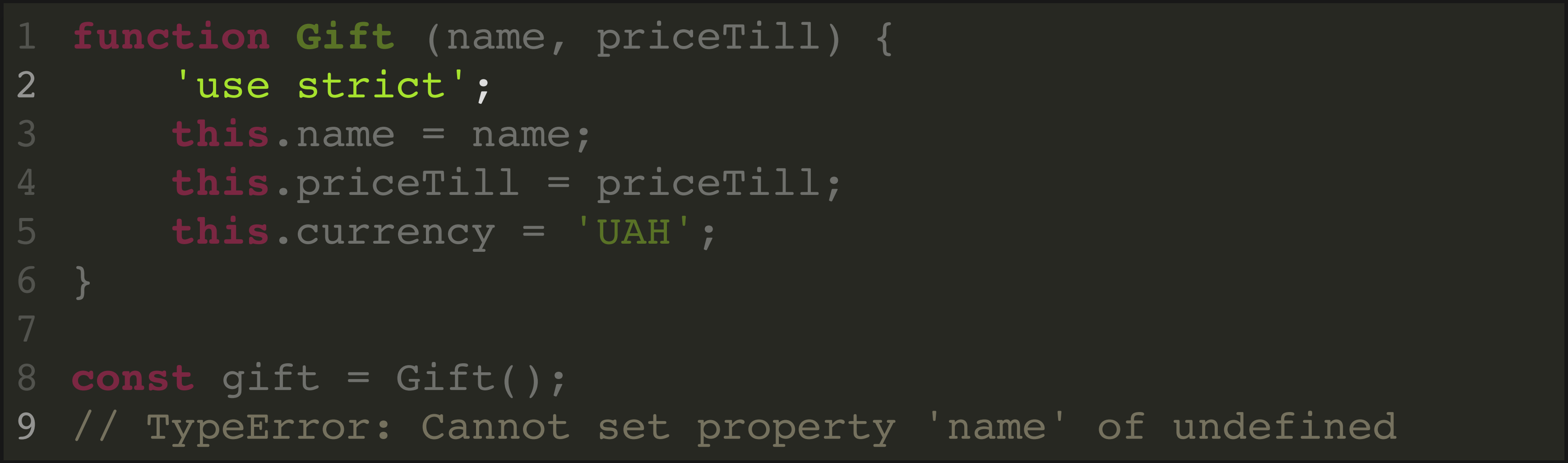
```javascript
function Gift (name, priceTill) {
    this.name = name;
    this.priceTill = priceTill;
    this.currency = 'UAH';
}

const gift = Gift();
```



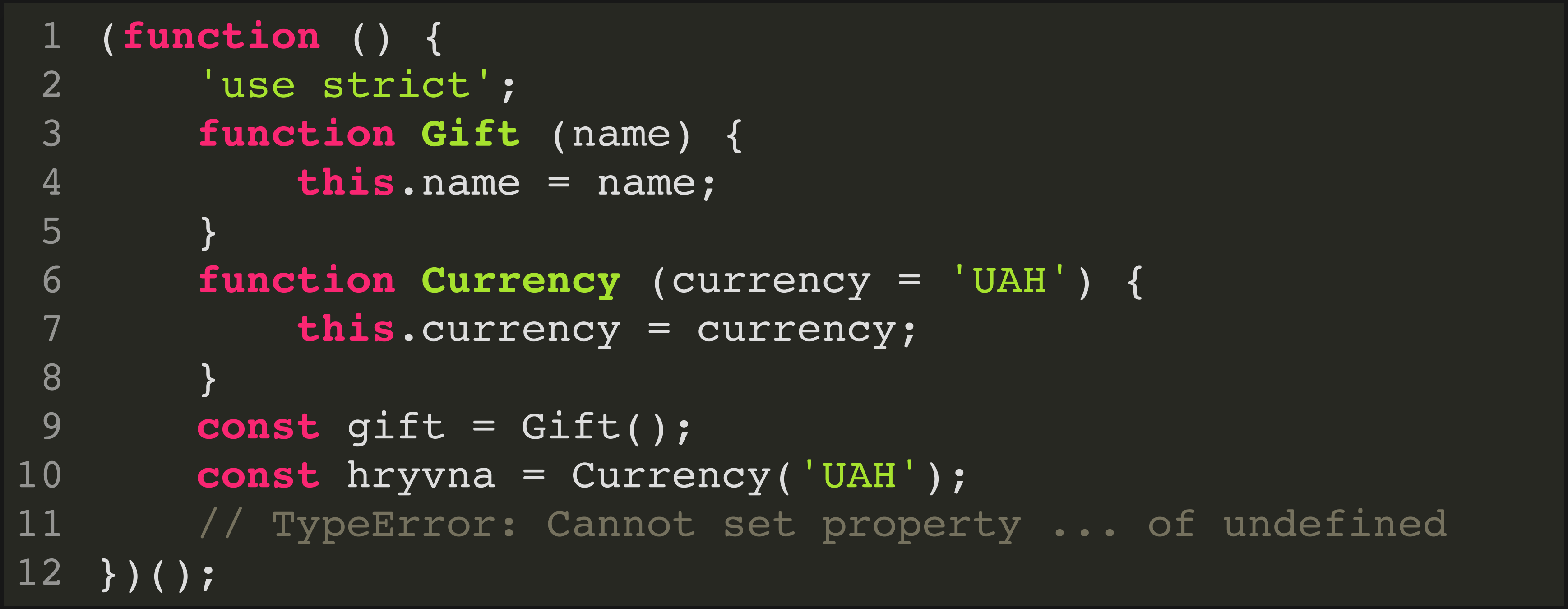
```
function Gift (name, priceTill) {
    this.name = name;
    this.priceTill = priceTill;
    this.currency = 'UAH';
}

const gift = Gift();
```



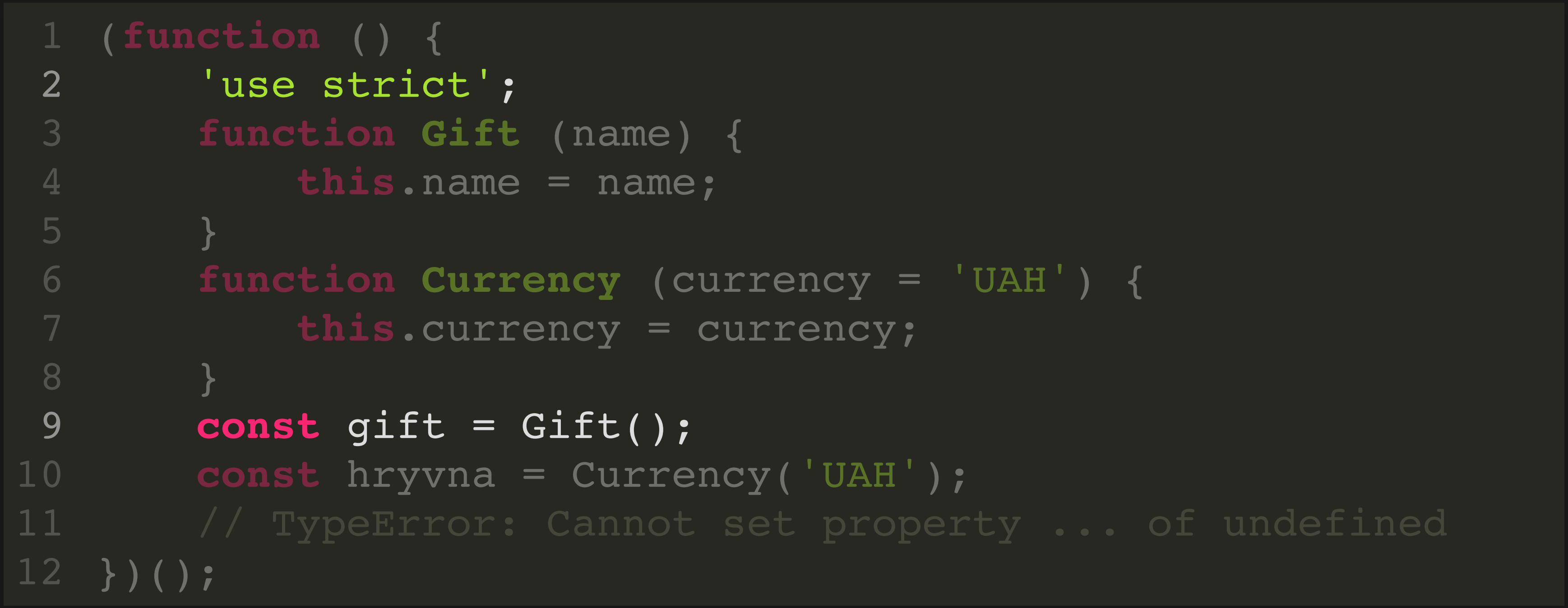
```
function Gift (name, priceTill) {
    'use strict';
    this.name = name;
    this.priceTill = priceTill;
    this.currency = 'UAH';
}

const gift = Gift();
// TypeError: Cannot set property 'name' of undefined
```



```javascript
function Gift (name, priceTill) {
    'use strict';
    this.name = name;
    this.priceTill = priceTill;
    this.currency = 'UAH';
}

const gift = Gift();
// TypeError: Cannot set property 'name' of undefined
```



```
(function () {
    'use strict';
    function Gift (name) {
        this.name = name;
    }
    function Currency (currency = 'UAH') {
        this.currency = currency;
    }
    const gift = Gift();
    const hryvna = Currency('UAH');
    // TypeError: Cannot set property ... of undefined
})();
```



```
(function () {
    'use strict';
    function Gift (name) {
        this.name = name;
    }
    function Currency (currency = 'UAH') {
        this.currency = currency;
    }
    const gift = Gift();
    const hryvna = Currency('UAH');
    // TypeError: Cannot set property ... of undefined
})();
```

Ripley style

```
function Gift (name) {
    if ((this instanceof Gift) === false) {
        return new Gift(name);
    }
    this.name = name;
}

const gift = Gift('How is that even?');
gift.name // 'How is that even?'
```

Ripley style

```
function Gift (name) {
    if ((this instanceof Gift) === false) {
        return new Gift(name);
    }
    this.name = name;
}

const gift = Gift('How is that even?');
gift.name // 'How is that even?'
```

DETAILS

Create method within `constructor` or in `prototype`?

```javascript
function Gift (name) {
    this.say = function () { alert('So...') };
}

Gift.prototype.say = function () { alert('So...') };
```

- Methods are the same but different
- Memory allocation question
- Hard to update

In some proprietary derivatives of JavaScript like CoffeeScript or TypeScript constructor properties assignment implemented in more clever way:

- CoffeeScript - @name
- TypeScript - constructor parameter properties