Como nomear variáveis?

Nomear as coisas é difícil. Então, vamos tentar tornar isso mais fácil.

Embora essas sugestões possam ser aplicadas a qualquer linguagem de programação, usarei JavaScript para ilustrá-las na prática.

Língua Inglesa

Use o idioma inglês ao nomear suas variáveis e funções.

```
/* Ruim */
const primerNombre = 'Gustavo'
const amigos = ['Kate', 'John']

/* Bom */
const firstName = 'Gustavo'
const friends = ['Kate', 'John']
```

> *Goste ou não, o inglês é a linguagem dominante na programação: a sintaxe de todas as linguagens de programação é escrita em inglês, assim como inúmeras documentações e materiais educacionais. Ao escrever seu código em inglês, você aumenta drasticamente sua coesão.*

Convenção de nomes

Escolha **uma** convenção de nomenclatura e siga-a. Pode ser `camelCase`, ou `snake_case`, ou de qualquer outra forma, não importa. O que importa é que permaneça consistente (não misture mais de uma).

```
/* Ruim */
const page_count = 5
const shouldUpdate = true

/* Bom */
const pageCount = 5
const shouldUpdate = true

/* Good as well */
const page_count = 5
const should_update = true
```

S-I-D

A noemação deve ser *short* (pequeno), *intuitive* (intuitivo) e *descriptive* (descritivo):

- **Short**. Um nome não deve demorar para ser digitado e, portanto, lembre-se;
- **Intuitivo**. Um nome deve ser lido naturalmente, o mais próximo possível da fala comum;
- **Descritivo**. Um nome deve refletir o que ele faz / possui da maneira mais eficiente;

```
/* Ruim */
const a = 5 // "a" pode significar qualquer coisa
const isPaginatable = a > 10 // "Paginável" soa extremamente artificial
const shouldPaginatize = a > 10 // Verbos inventados são muito divertidos!

/* Bom */
const postCount = 5
const hasPagination = postCount > 10
const shouldDisplayPagination = postCount > 10 // alternativamente
```

Evite contrações

Não use contrações. Elas não contribuem para nada além da diminuição da legibilidade do código. Encontrar um nome curto e descritivo pode ser difícil, mas a contração não é uma desculpa para não fazê-lo.

`js /* Ruim */ const onItmClk = () => {}

/* Bom */ const onItemClick = () => {} `

Evite a duplicação de contexto

Um nome não deve duplicar o contexto no qual está definido. Sempre remova o contexto de um nome, se isso não diminuir sua legibilidade.

```
class MenuItem {
  /* O nome do método duplica o contexto (que é "MenuItem") */
  handleMenuItemClick = (event) => { ... }

  /* Se lê muito melhor assim como `MenuItem.handleClick ()` */
  handleClick = (event) => { ... }
}
```

Refletir o resultado esperado

Um nome deve refletir o resultado esperado.

```
/* Ruim */
const isEnabled = itemCount > 3
return <Button disabled={!isEnabled} />

/* Bom */
const isDisabled = itemCount <= 3
return <Button disabled={isDisabled} />
```

Funções de nomenclatura

A/HC/LC padronizar

Há um padrão útil a seguir ao nomear funções:

```
prefix? + action (A) + high context (HC) + low context? (LC)
```

```
prefixo? + ação (A) + contexto alto (HC) + contexto baixo? (LC)
```

Dê uma olhada em como esse padrão pode ser aplicado na tabela abaixo.

Name	Prefix	Action (A)	High context (HC)	Low context (LC)
`getPost`		`get`	`Post`	
`getPostData`		`get`	`Post`	`Data`
`handleClickOutside`		`handle`	`Click`	`Outside`
`shouldDisplayMessage`	`should`	`Display`	`Message`	

> ***Note:*** *A ordem do contexto afeta o significado de uma variável. Por exemplo,* `shouldUpdateComponent` *significa que você está prestes a atualizar um componente, enquanto* `shouldComponentUpdate` *diz que o component irá atualizar por si mesmo, e você está controlando quando ele deve ser atualizado. Em outras palavras,* ***contexto alto enfatiza o significado de uma variável****.*

Actions

A parte do verbo do nome da função.A parte mais importante responsável por descrever o que a função *faz*.

`get`

Acessa os dados imediatamente (ou seja, abreviatura de dados internos).

```
function getFruitCount() {
  return this.fruits.length
}
```

> *See also compose.*

set

Define uma variável de forma declarativa, com o valor `A` para o valor`B`.

```
let fruits = 0

function setFruits(nextFruits) {
  fruits = nextFruits
}

setFruits(5)
console.log(fruits) // 5
```

reset

Define uma variável de volta ao seu valor ou estado inicial.

```
const initialFruits = 5
let fruits = initialFruits
setFruits(10)
console.log(fruits) // 10

function resetFruits() {
  fruits = initialFruits
}

resetFruits()
console.log(fruits) // 5
```

fetch

Solicitação de alguns dados, que leva algum tempo indeterminado (ou seja, solicitação assíncrona).

```
function fetchPosts(postCount) {
  return fetch('https://api.dev/posts', {...})
}
```

`remove`

Removes something *from* somewhere.

For example, if you have a collection of selected filters on a search page, removing one of them from the collection is `removeFilter`, **not** `deleteFilter` (and this is how you would naturally say it in English as well):

```
function removeFilter(filterName, filters) {
  return filters.filter((name) => name !== filterName)
}

const selectedFilters = ['price', 'availability', 'size']
removeFilter('price', selectedFilters)
```

> *Veja também a seção delete.*

`delete`

Apaga COMPLETAMENTE a existência de algo.

Imagine que você é um editor de conteúdo e há aquele post notório do qual deseja se livrar. Depois de clicar em um botão VERMELHO BRILHANTE “Excluir postagem”, o CMS executou uma ação `deletePost`, **e não** `removePost`.

```
function deletePost(id) {
  return database.find({ id }).delete()
}
```

> *Veja também a seção remove.*

compose

Cria novos dados a partir do existente. Principalmente aplicável a strings, objetos ou funções.

```
function composePageUrl(pageName, pageId) {
  return `${pageName.toLowerCase()}-${pageId}`
}
```

> *See also get.*

handle

Lida com uma ação. Frequentemente usado ao nomear um método de retorno de chamada.

```
function handleLinkClick() {
  console.log('Clicked a link!')
}

link.addEventListener('click', handleLinkClick)
```

Contexto

Um domínio no qual uma função opera.

Uma função geralmente é uma ação em *algo*. É importante indicar qual é o seu domínio operável, ou pelo menos um tipo de dados esperado.

```
/* A pure function operating with primitives */
function filter(predicate, list) {
  return list.filter(predicate)
}

/* Function operating exactly on posts */
function getRecentPosts(posts) {
  return filter(posts, (post) => post.date === Date.now())
}
```

> *Algumas suposições específicas do idioma podem permitir a omissão do contexto. Por exemplo, em JavaScript, é comum que `filter` opere em Array.Adicionar `filterArray` explícito seria desnecessário.*

—

Prefixos

Prefixo realça o significado de uma variável. Raramente é usado em nomes de funções.

`is`

Descreve uma característica ou estado do contexto atual (geralmente `booleano`).

```
const color = 'blue'
const isBlue = color === 'blue' // characteristic
const isPresent = true // state

if (isBlue && isPresent) {
  console.log('Blue is present!')
}
```

`has`

Descreve se o contexto atual possui um certo valor ou estado (geralmente `booleano`).

```
/* Ruim */
const isProductsExist = productsCount > 0
const areProductsPresent = productsCount > 0

/* Bom */
const hasProducts = productsCount > 0
```

`should`

Reflete uma declaração condicional positiva (geralmente `booleana`) associada a uma determinada ação.

```
function shouldUpdateUrl(url, expectedUrl) {
  return url !== expectedUrl
}
```

`min`/`max`

Representa um valor mínimo ou máximo. Usado ao descrever limites.

```
/**
 * Renders a random amount of posts within
 * the given min/max boundaries.
 */
function renderPosts(posts, minPosts, maxPosts) {
  return posts.slice(0, randomBetween(minPosts, maxPosts))
}
```

`prev`/`next`

Indica o estado anterior ou seguinte de uma variável no contexto atual.Usado ao descrever transições de estado.

```
function fetchPosts() {
  const prevPosts = this.state.posts

  const fetchedPosts = fetch('...')
  const nextPosts = concat(prevPosts, fetchedPosts)

  this.setState({ posts: nextPosts })
}
```

Singular e Plurals

Como um prefixo, os nomes de variáveis podem ser transformados no singular ou no plural, dependendo se eles contêm um único valor ou vários valores.

```
/* Ruim */
const friends = 'Bob'
const friend = ['Bob', 'Tony', 'Tanya']

/* Bom */
const friend = 'Bob'
const friends = ['Bob', 'Tony', 'Tanya']
```