TÓPICO 02 - CONVENÇÕES E NOMENCLATURAS

Clean Code - Professor Ramon Venson - SATC 2025.2

Nomenclaturas

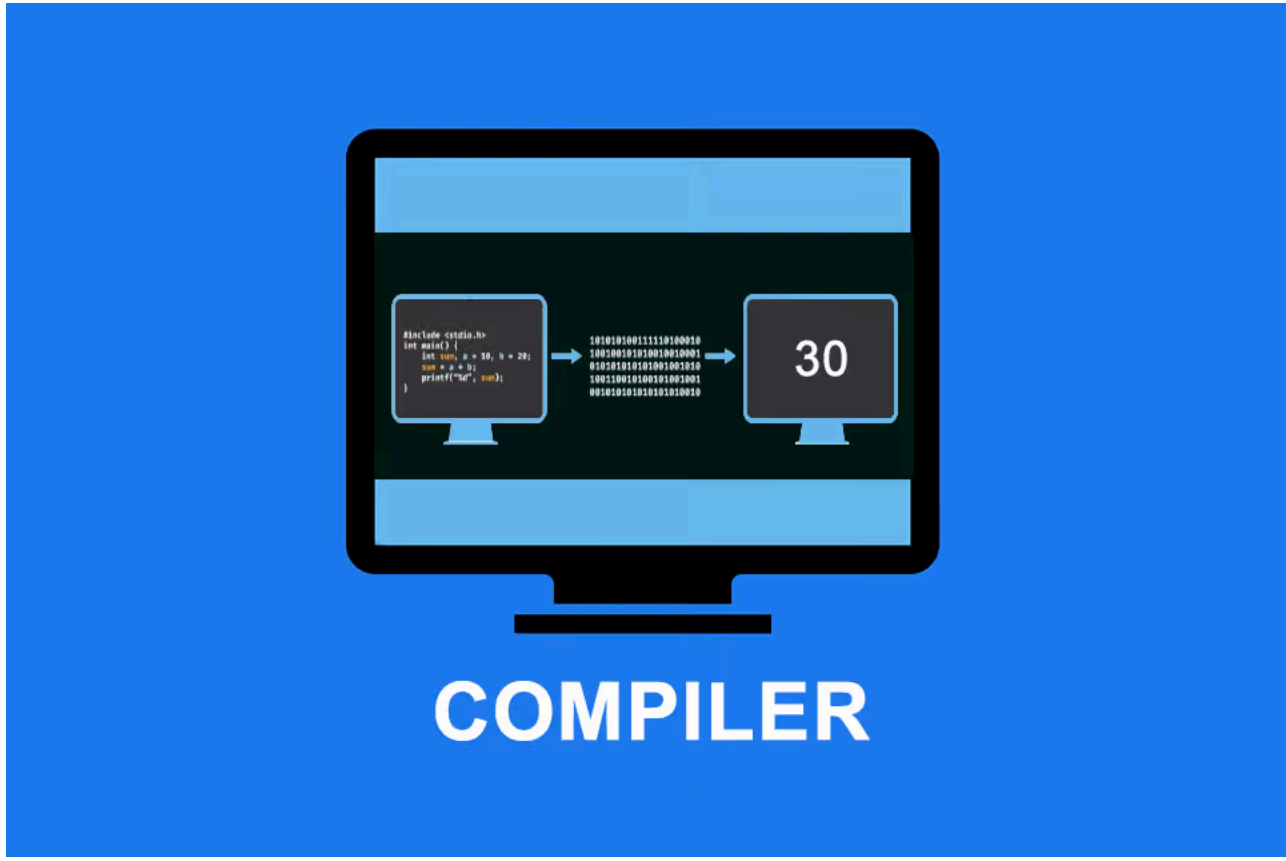
> Existem apenas duas coisas difíceis em ciências da computação: Invalidação de cache e dar nome as coisas

Phil Karlton, Netscape developer

70% do código-fonte do Eclipse é composto por nomes

Pra um computador `Sgu9Asd1M` pode ser a mesma coisa do que `blue`.

Qual a vantagem de construir programas com nomes claros e significativos?



Categorias de Nomes

Para variáveis, constantes e classes, usamos `substantivos` (as vezes em conjunto com um `adjetivo`).

Ex.: `usuario`, `analise_agenda`, `device_storage`, `user_profile`.

Para funções ou métodos, usamos `verbos`.

Ex.: `save_user_profile`, `get_user_info`, `get_user_data`.

Formas de Nomes para Funções

Caprile e Tonella identificaram seis categorias de nomes:

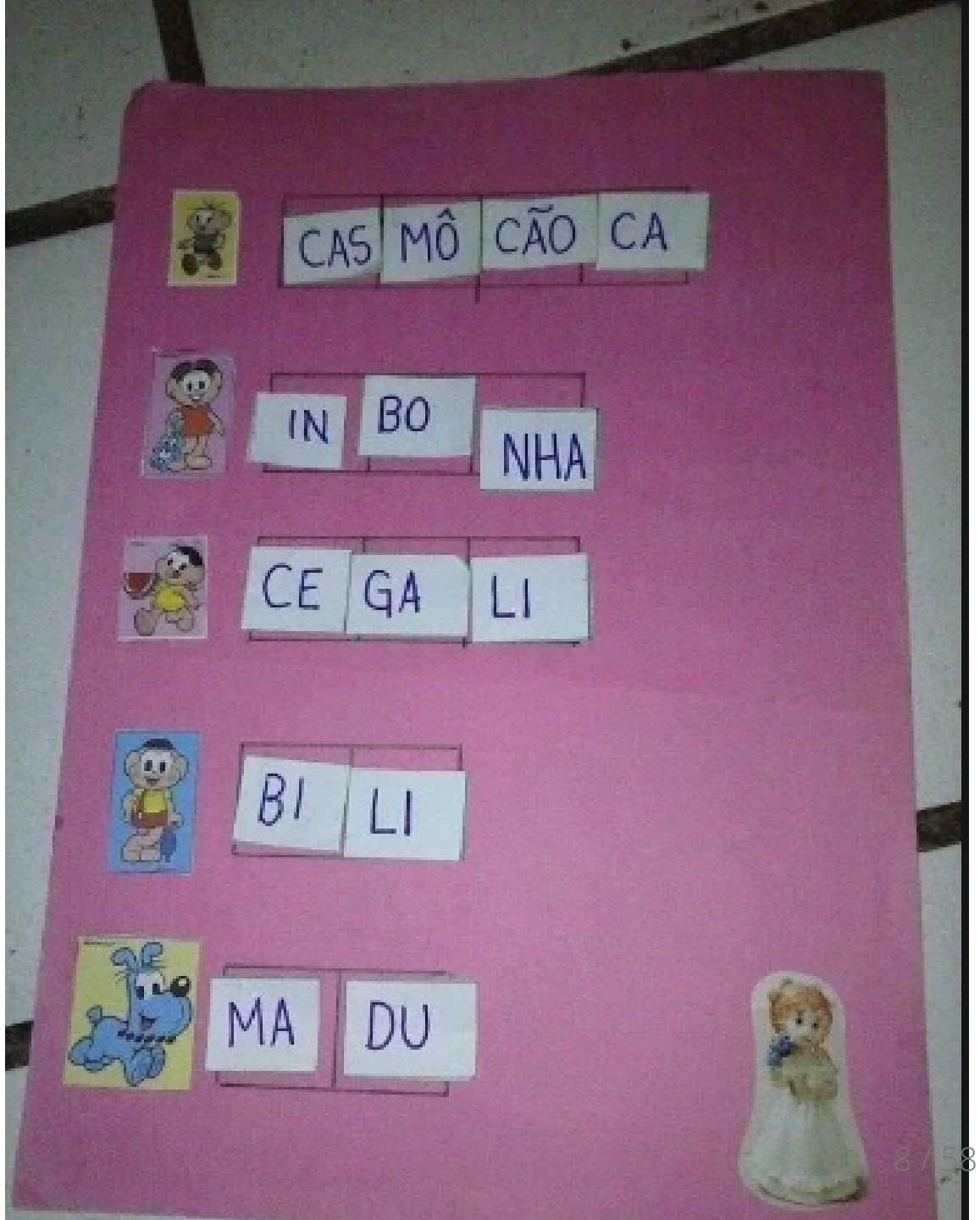
- Ação indireta (`error` , `on_error` , `on_error_callback`).
- Ação direta (`open` , `close` , `kill`).
- Ação sobre objeto (`read_line` , `write_line`)
- Ação dupla (`search_and_replace_text` , `confirma_e_salva`) [Não recomendado]
- Checagem (`is_valid` , `is_empty` , `is_valid_email`)
- Transformação (`convert_to_hex` , `conterte_para_binario`)

Os verbos mais utilizados no código fonte do `Bash` são: `get`, `set` e `make`.

Esses verbos são utilizados com sinonomos para uma melhor compreensão do contexto (`get_info`).

Nomes com Significado

Utilize nomes que revelem o conteúdo ou propósito.



```
d = 50 # tempo decorrido em dias
print(d)
```

```
# en
days_since_modification = 50
print(days_since_modification)

# pt-br
dias_desde_modificacao = 50
print(dias_desde_modificacao)
```

Mesmo com um nome maior, o código fica mais claro pra quem deseja saber em qual contexto o número está sendo utilizado.

```
#en
def user(user_id):
    return db.get_user(user_id)

#pt-br
def usuario(id_usuario):
    return db.get_user(id_usuario)
```

```
#en
def get_user_by_id(user_id):
    return db.get_user(user_id)

#pt-br
def retorna_usuario_por_id(id_usuario):
    return db.get_user(id_usuario)
```

A entrada e a saída serão automaticamente inferidas pelo programador. A função retorna um `user` a partir de um `id`.

```python
def c(x, y):
    return x * y + 10
a = 5
b = 3
print(c(a, b))
```

```python
def calcular_preco_total(preco_unitario, quantidade):
    taxa_fixa = 10
    return preco_unitario * quantidade + taxa_fixa

preco = 5
quantidade = 3
print(calcular_preco_total(preco, quantidade))
```

É muito mais fácil entender o que essa função faz e como ela deve ser utilizada se os nomes descrevem exatamente os dados com o qual ela trabalha.

Nomes Enganosos

Se uma variável ou função é chamada de `max`, não é uma boa ideia retornar o valor mínimo.

Não use nomes que induzem o programador a pensar que algo é verdadeiro quando não é.

```
#en
accountList = Account()

#pt-br
lista_de_contas = Conta()
```

```
#en
account = Account()

#pt-br
conta = Conta()
```

Uma lista deve conter um conjunto de itens, e não uma única coisa.

```
#en
students = Student()

#pt-br
alunos = Aluno()
```

```
#en
student = Student()

#pt-br
aluno = Aluno()
```

Variáveis que possuem apenas um único elemento não devem estar no plural.

```python
#en
def get_first_number(numbers):
    return min(numbers)

#pt-br
def get_primeiro_numero(numeros):
    return max(numeros)
```

```
#en
def get_lower_number(numbers):
    return min(numbers)

#pt-br
def get_menor_numero(numeros):
    return min(numeros)
```

A palavra `first` ou `primeiro` pode ser ambígua e não deve ser utilizada, como no exemplo, como sinonimo de menor número deve ser retornado.

Faça distinções significativas

Crie distinções significativas entre termos, especialmente aqueles que possuem funções opostas.

```
#en
def delete_user(user_id):
    # ...

def remove_user(user_id):
    # ...

# pt-br
def deletar_usuario(id_usuario):
    # ...

def remover_usuario(id_usuario):
    # ...
```

```
#en
def delete_user(user_id):
    # ...

def disable_user(user_id):
    # ...

# pt-br
def deletar_usuario(id_usuario):
    # ...

def desabilitar_usuario(id_usuario):
    # ...
```

Evite o uso de palavras que são fortes sinônimos entre si para funções diferentes.

Seja consistente com o uso dessas palavras no código.

```
#en
def login(user_id, password):
    # ...

def cancel(user_id):
    # ...

#pt-br
def logar(id_usuario, senha):
    # ...

def cancelar(id_usuario):
    # ...
```

```
#en
def login(user_id, password):
    # ...

def logout(user_id):
    # ...

# pt-br
def logar(id_usuario, senha):
    # ...

def deslogar(id_usuario):
    # ...
```

Use nomes que façam distinção entre funções que possuem funções opostas.

Nomes Pronunciáveis

Se você não pode pronunciar um nome, como vai explicar para alguém como utilizar o código?

```
#en
ymdhms = datetime.datetime.now()

#pt-br
amdhms = datetime.datetime.now()
```

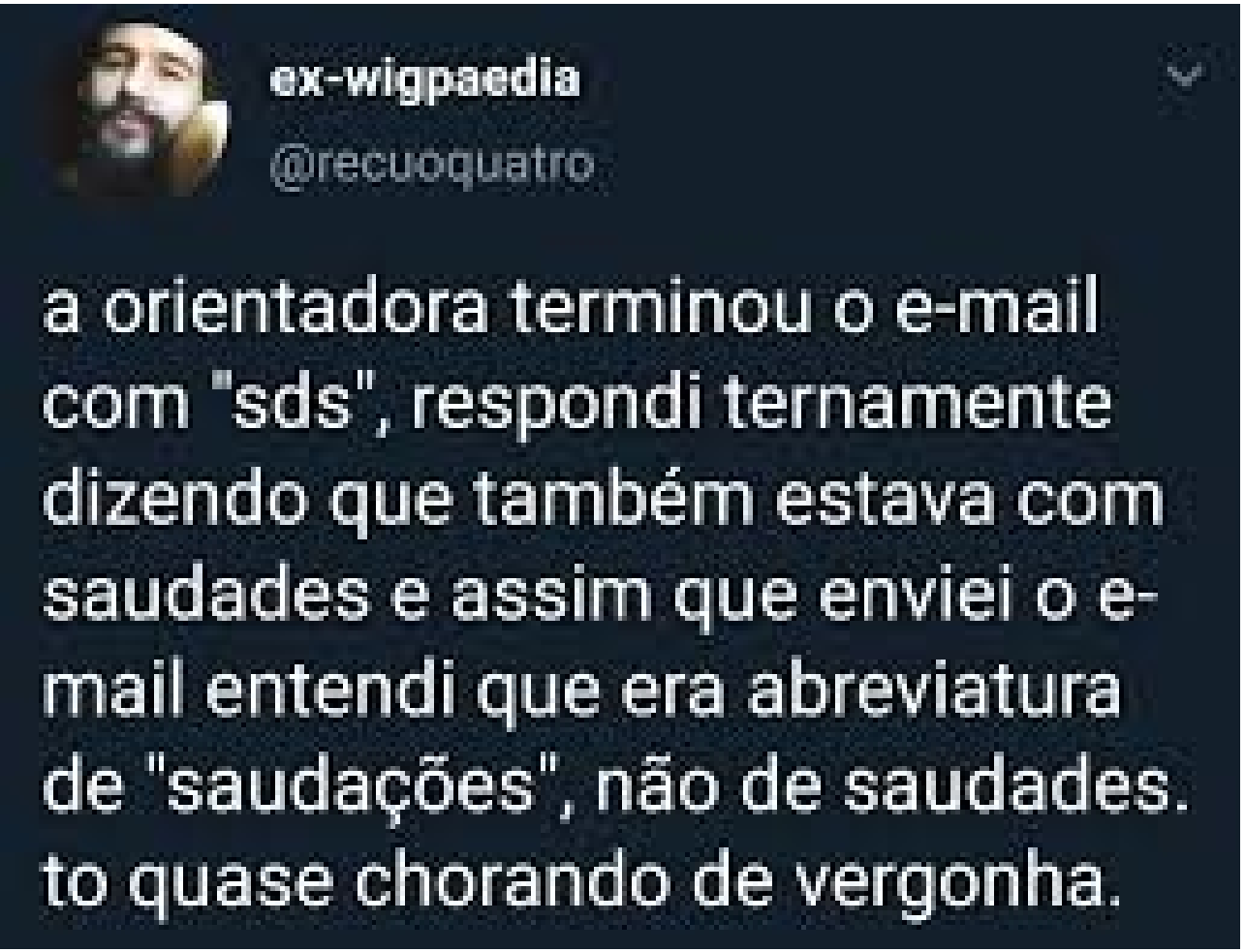
```
#en
timestamp = datetime.datetime.now() # YYYY-MM-DD HH:MM:SS.mmmmmm

#pt-br
timestamp = datetime.datetime.now() # AAAA-MM-DD HH:MM:SS.mmmmmm
```

O formato de data e hora é padronizado, portanto, não é necessário utilizar nomes que descrevam o formato. Opcionalmente, use comentários para padrões de formato.

```
temp_55834 = 55834
```

```
population = 55834
```

Não use nomes que descrevem o formato, mas que não descrevem o que o dado representa.



Evite Abreviações

Abreviações são difíceis de serem pronunciadas e podem ser confundidas com outras abreviações.

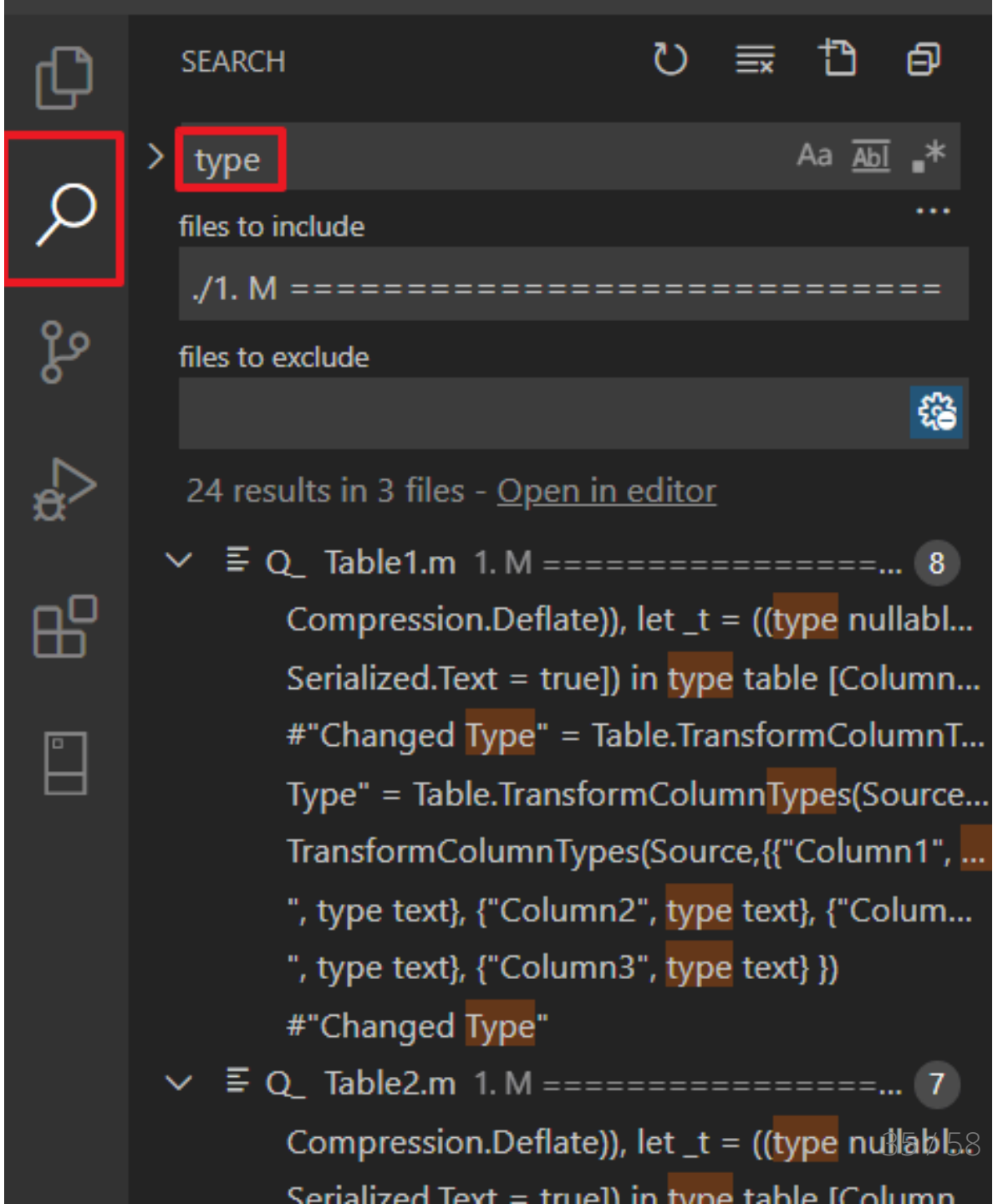
```
#en
def get_df_user()

#pt-br
def recebe_usuario_pd()
```

```
#en
def get_default_user()

#pt-br
def recebe_usuario_padrao()
```

Busca por nomes

Nomes muito curtos podem ser mais difíceis de serem encontrados em buscas.

Isso não é um problema muito sério para IDEs com um *intellisense* mais avançado, mas é uma boa prática.



```
x = 3
y = 20
```

```
x_axis = 3
y_axis = 20
```

Nem sempre esse é o caso, mas é uma boa prática quando for possível. Para classes que trabalham com coordenadas, por exemplo, é comum utilizar nomes como `x` e `y`.

Prefixos e Sufixos

Antigamente, havia uma tendência de utilizar nomes curtos e prefixos para indicar o tipo de dado.

Com a evolução das IDEs, isso não é mais uma prática recomendada.

Data Type	Prefix	Example
BOOL	x	xCmd_OpenValve
BYTE	by	byStatusByte
WORD	w	wAlarmTrigger
DWORD	dw	dwErrorCode
SINT	si	siCfg_ModuleNo
UINT	ui	uiStepNo
DINT	di	diIndex
UDINT	udi	udiTotalCount
REAL	r	rPressure_Scaled
STRING	si	sUserName
TIME	tim	timDelay_Start
DATE_AND_TIME	dt	dtActualDate
POINTER	p	pArrayElement
ARRAY	a	adiArrayOfDInt

```
#en
s_name = "John"

#pt-br
s_nome = "John"
```

```
#en
name = "John"

#pt-br
nome = "John"
```

Exceções

Para interfaces e implementações, ainda é comum utilizar prefixos para indicar o tipo de estrutura:

```
public interface IUserRepository {
    void save(User user);
}

public class UserRepositoryImpl implements IUserRepository {}
```

Mapeamento Mental

Guarde sua memória para o essencial.

Nomes de variáveis excessivamente complexos e difíceis de lembrar são ineficientes, especialmente em contextos mais complexos.

```
for i in range(10):
    print(i)
```

Não há problema em usar nomes curtos em contextos simples, como em loops.

```python
class Usuario:
    def __init__(self, nome, idade, email, x, y, z, a, b):
        self.nome = nome
        self.idade = idade
        self.email = email
        self.x = x
        self.y = y
        self.z = z
        self.a = a
        self.b = b
```

Atributos com nomes como `x`, `y`, `z`, `a` e `b` não são claros e podem ser difíceis de lembrar em contextos mais complexos.

```
#en
def findArticlesWithoutTitlesThenApplyDefaultStyles ()

#pt-br
def encontrar_artigos_sem_titulos_então_aplicar_estilos_padrão ()
```

Nomes muito longos também podem ser difíceis de lembrar na memória a curto prazo.

```
#en
def find_articles_without_titles()
def apply_default_styles()

#pt-br
def encontrar_artigos_sem_titulos()
def aplicar_estilos_padrão()
```

Quebre funções e variáveis com nomes muito extensos em partes menores. Isso é um exemplo de *code smell* e ajuda a separar responsabilidades (*Single Responsibility Principle*).

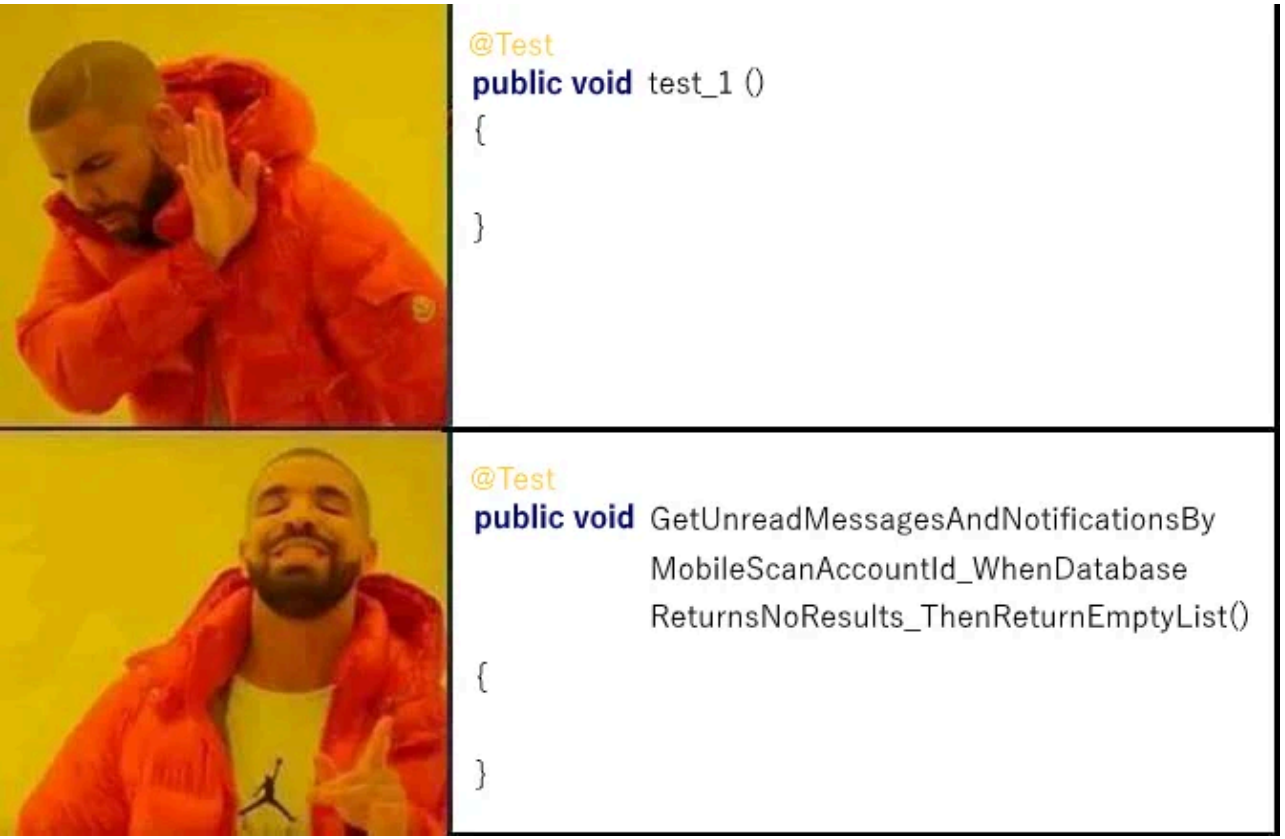
Nomes de Classes

Nomes de classes, especialmente aquelas responsáveis por dados, são geralmente substantivos ou frases substantivas, como: `User`, `Product`, `Account`, `AddressAnalysis`.

Evite nomes de classe como: `Manager`, `Processor`, `Data`, `Info`.

Fish.class

AbstractAnimalThatLivesInWaterAnd HasGillsAndFins.class



Nomes de Métodos

Nomes de métodos são geralmente verbos ou frases verbais, como: `post_tweet`, `delete_user`, `calculate_total`.

Não seja espertinho

Não use palavras ou gírias que outros programadores podem não reconhecer.

`jogar_granada_de_mao` é um nome engraçado para uma função que deleta usuários do sistema, mas pode não muito clara para todos os programadores.

Domínios da Solução

Em alguns casos, pode não ser prudente pensar em nomes a partir do domínio do problema.

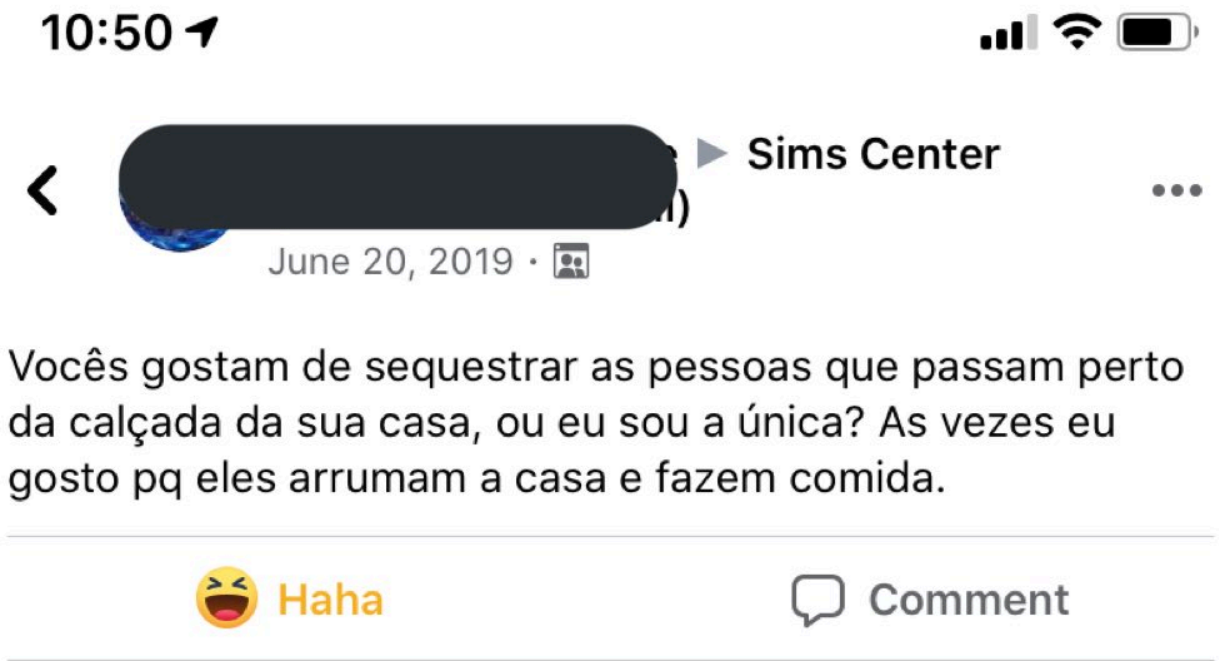
Uma classe ou atributo chamado `sala_de_espera` pode ser convertida para `fila_de_espera` ou apenas `fila`.

Isso reflete melhor o propósito do elemento.

Domínios do Problema

Quando não for possível utilizar o domínio da solução, utilize o domínio do problema.

Por exemplo, ao invés de nomear uma classe como `IndividuoPreterido`, nomeie-a como `JogadorReserva`.



Contextos desnecessários

Não é necessário que toda a classe de uma aplicação chamada Gerenciador de Estudantes possua o sufixo `GE`, como em: `ProfessorGE`.

Resumo dos problemas

Em geral, um nome pode ser:

1. Muito específico:
    - Faltam informações (`set`)
    - É muito longo ou com muitas palavras (`get_all_users_with_permission_to_edit_article`)
2. Muito ambíguo
    - Expressa um conceito muito genérico (`inteiro` , `string`)
    - Tem múltiplos significados (`file` ou `arquivo`)
    - Não é óbvio (`blowfish`)

3. Incorreto
    - Não condiz com o que a variável representa
4. Inconsistente
    - Não é consistente com o resto do código

Mais Dicas

- Use dicionário/tesauros para encontrar nomes significativos:
  - thesaurus.com
  - sinonimos.com.br
- Procure reutilizar nomes e verbos já usados na aplicação
- Padronize tudo! Não seja muito original

Conclusão

- Nomes claros e significativos
- Evite abreviações e nomes enganosos
- Consistência é essencial
- Prefira nomes descritivos
- Refatore sempre que necessário

> "Escreva código para humanos, não para máquinas."

Material de Apoio

- Naming conventions in programming – a review of scientific literature

O que aprendemos hoje?

- Importância das nomenclaturas no código
- Práticas recomendadas para nomeação
- Como evitar nomes enganosos
- Como aplicar nomenclaturas em diferentes contextos
- Como utilizar nomes descritivos e significativos
- Como refatorar nomes para melhorar a legibilidade