Pegue essa sua resposta e crie um arquivo txt com ela

Boa, ótima conexão com o que a gente acabou de discutir.

Vamos usar exatamente o caso do dado `"a"` na chamada `mmc → mdc` para fixar C[d], PC[d] e N[d] na sua cabeça.

Código em foco:

```
def mdc(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def mmc(a, b):
    return abs(a * b) // mdc(a, b)
```

Estamos analisando o dado `d = "a"` na chamada:

- source (quem depende): `mmc`
- target (quem é chamado): `mdc`
- conexão: `mdc(a, b)` dentro de `mmc`

Pela fórmula do Briand:

- EC[a] = C[a] × PC[a] × N[a]

Agora, um por um:

1. N[a] – número de conexões via “a”

- O que é: quantas vezes o dado “a” aparece como dado de conexão entre funções (argumento em chamadas).
- No código todo, “a” entra em chamada de função uma única vez: `mdc(a, b)` dentro de `mmc`.
- Então:
  - N[a] = 1

Interpretação: “a” estabelece 1 conexão entre `mmc` e `mdc`.

2. C[a] – complexidade do dado “a”

- Ideia no artigo: C[d] mede quão “complexo” é o dado:
  - escalar simples (int, float, bool) → complexidade mínima (tipicamente 1);

- estruturas compostas (registros, objetos com vários campos) → valor maior.

- No nosso caso, “a” é um parâmetro escalar, simples:
  - não é struct,
  - não é array complexo,
  - não é um objeto com muitos campos.

Logo, na convenção usual:

- C[a] = 1

Interpretação: “a” é um dado simples; ele não aumenta o acoplamento por ser estruturalmente complexo.

3. PC[a] – uso do dado para controle (Program Control)

- Ideia no artigo: PC[d] sobe quando o dado influencia o controle de fluxo (ifs, whiles, switches).
- No trecho da chamada:

```
return abs(a * b) // mdc(a, b)
```

  - “a” está sendo usado só em uma expressão aritmética,
  - não decide qual ramo de código é executado, nem controla laços.

Então:

- PC[a] = 1 (não é dado de controle)

Se “a” fosse, por exemplo, um flag que definisse se uma parte do código roda ou não em outro módulo, aí você poderia ter PC[a] > 1.

4. Juntando: EC[a] para a conexão mmc → mdc

- C[a] = 1 (simples)
- PC[a] = 1 (não controla fluxo)
- N[a] = 1 (uma conexão)

Então:

- EC[a] = C[a] × PC[a] × N[a]
- EC[a] = 1 × 1 × 1 = 1

Interpretação final:

- Esse dado “a” contribui com 1 unidade de elementary coupling para o acoplamento da função `mmc` em relação a `mdc`.
- Ele é um acoplamento de dados fraco, exatamente como você comentou: é só um parâmetro local sendo passado.

Se você quiser, no próximo passo a gente pega um exemplo com variável global, define juntos C[d], PC[d], N[d] para ela, e compara com esse caso do “a” para ver como o EC cresce.

⁂

1. https://www.coolutils.com/pt/online/PY-to-TXT
2. https://www.reddit.com/r/learnpython/comments/uyqgwx/how_do_i_turn_my_python_code_into_txt_file/
3. https://www.youtube.com/watch?v=6fIUTm44Siw
4. https://products.aspose.com/words/pt/python-net/conversion/doc-to-txt/
5. https://pt.stackoverflow.com/questions/589538/como-transformar-um-texto-que-peguei-de-um-site-com-web-scraping-em-um-arquivo-t
6. https://www.youtube.com/watch?v=ZfBWkaQ4ni4
7. https://www.datacamp.com/pt/tutorial/two-simple-methods-to-convert-a-python-file-to-an-exe-file
8. https://rocketseat.com.br/blog/artigos/post/python-como-ler-informacoes-de-arquivos
9. https://www.reddit.com/r/learnpython/comments/fyjo5q/how_to_save_output_as_txt_file_in_python/