

# Projeto de programas

---

Marco A L Barbosa  
malbarbo.pro.br

Departamento de Informática  
Universidade Estadual de Maringá



Este trabalho está licenciado com uma Licença Creative Commons - Atribuição-Compartilhável 4.0 Internacional.

<http://github.com/malbarbo/na-programacao>

O André viaja muito. Sempre antes de fazer uma viagem ele calcula o quanto ele irá gastar com combustível. Ele determina a distância que ele irá percorrer na viagem, o preço do litro do combustível e consulta as suas anotações para ver o consumo do carro, isto é, a quantidade de quilômetros que o carro anda com um litro de combustível e então faz o cálculo do custo. O André acha um pouco chato fazer os cálculos na mão, então ele pediu para você escrever um programa que faça os cálculos para ele.

Como projetar um programa que atenda a necessidade do André?

Seguindo um processo, uma sequência de etapas.

Projetar programas que funcionem corretamente e sejam bem escritos é um desafio, seguir um processo é uma ferramenta indispensável nesse processo.

No início, para problemas simples, o processo poderá parecer muito custoso, mas vamos apreciar a sua utilidade conforme progredimos.

O processo que vamos seguir está dividido em 6 etapas:

- Análise
- Definição dos tipos de dados
- Especificação
- Implementação
- Verificação
- Revisão

Cada etapa tem um objetivo

- Análise: identificar o problema a ser resolvido
- Definição dos tipos de dados: identificar e definir como as informações serão representadas
- Especificação: especificar com precisão o que a função deve fazer
- Implementação: implementar a função de acordo com a especificação
- Verificação: verificar se a implementação está de acordo com a especificação
- Revisão: identificar e fazer melhorias

Ao final de cada etapa produzimos resultados que são utilizados nas etapas posteriores, então devemos tentar seguir as etapas em ordem.

No entanto, em algumas situações, pode ser necessário mudar a ordem.

- Estamos na implementação e encontramos uma nova condição e devemos voltar e alterar a especificação.
- Não estamos conseguindo entender o problema (análise) e então fazemos alguns exemplos (especificação) para nos ajudar.

Mas devemos evitar fazer a implementação diretamente!

Mas esse processo serve para projetar funções, como projetamos programas?

Um programa é composto de várias funções, então temos que decompor o programa em funções e aplicar o processo para projetar cada função.

Vamos treinar com problemas simples, de uma função, depois vamos utilizar o processo em problemas mais elaborados.

Vamos iniciar resolvendo o problema do André!

O André viaja muito. Sempre antes de fazer uma viagem ele calcula o quanto ele irá gastar com combustível. Ele determina a distância que ele irá percorrer na viagem, o preço do litro do combustível e consulta as suas anotações para ver o consumo do carro, isto é, a quantidade de quilômetros que o carro anda com um litro de combustível e então faz o cálculo do custo. O André acha um pouco chato fazer os cálculos na mão, então ele pediu para você escrever um programa que faça os cálculos para ele.

**Objetivo:** identificar o problema a ser resolvido.

- Quais informações são relevantes e quais podem ser descartadas?
- Existe alguma omissão?
- Existe alguma ambiguidade?
- Quais conhecimentos do domínio do problema são necessários?

## Resultado

Calcular o custo em reais para percorrer uma determinada distância levando em consideração o desempenho do carro e o preço do litro do combustível.

### Análise

Calcular o custo em reais para percorrer uma determinada distância levando em consideração o desempenho do carro e o preço do litro do combustível.

**Objetivo:** identificar e definir como as informações serão representadas.

- Quais são as informações envolvidas no problema?
- Como as informações serão representadas?

### Resultado

As informações são a distância em Km, rendimento em Km/l, preço em R\$/l e o custo da viagem em R\$.

Todos os valores serão representados por números positivos.

## Análise

Calcular o custo em reais para percorrer uma determinada distância levando em consideração o desempenho do carro e o preço do litro do combustível.

## Tipos de dados

As informações são a distância em Km, rendimento em Km/l, preço em R\$/l e o custo da viagem em R\$.

Todos os valores serão representados por números positivos.

**Objetivo:** especificar com mais precisão e com exemplos o que o programa deve fazer.

- Assinatura da função (nome, tipo das entradas e saídas)
- Propósito da função
- Exemplos de entrada e saída

## Assinatura

```
def custo_viagem(distancia: float, rendimento: float, preco: float) -> float:  
    return 0.0
```

Note que colocamos o `return` com um valor padrão para que a função fique bem formada.

## Propósito da função

O propósito descreve **o quê** a função deve fazer (faz, depois de implementada). Devemos usar o nome dos parâmetros na descrição do propósito para que a relação da entrada e da saída fique clara.

```
def custo_viagem(distancia: float, rendimento: float, preco: float) -> float:  
    '''  
    Calcula o custo em reais para percorrer a *distancia* especificada  
    considerando o *rendimento* do carro e o *preco* do litro do combustível.  
    '''  
    return 0.0
```

No propósito da função descrevemos **o quê** a função faz, e não **como** ela faz (que é a implementação - as vezes precisamos dizer como é feito, mas isso é raro).

Número par

- O quê: verifica se um número é par
- Como: faz o resto da divisão do número por 2 e compara com 0; ou; faz a divisão inteira do número e multiplica por 2 e compara com o número

## Exemplos

Ilustrar com exemplos de entrada e saída o funcionamento da função. O primeiro objetivo dos exemplos é ajudar o projetista a entender melhor como a função deve funcionar e como ela pode ser implementada.

Como escolher bons exemplos?

- Usar valores de casos práticos para o problema
- Considerar diversas situações, incluindo casos extremos

```
>>> # (120.0 / 10.0) * 5.0
>>> custo_viagem(120.0, 10.0, 5.0)
60.0
```

```
>>> # (300.0 / 15.0) * 6.0
>>> custo_viagem(300.0, 15.0, 6.0)
120.0
```

Note que podemos deixar como comentário a expressão utilizada para calcular a resposta.

Para saber se a especificação está boa, faça a seguinte pergunta:

Outro desenvolvedor, que não tem acesso ao problema original e nem a análise, tem as informações necessárias na especificação para fazer uma implementação e verificação inicial?

Se a resposta for sim, então a especificação está boa, senão ela está incompleta.

**Objetivo:** escrever o corpo da função para que ela faça o que está na especificação.

```
def custo_viagem(distancia: float, rendimento: float, preco: float) -> float:  
    ...  
  
    Calcula o custo em reais para percorrer a *distancia* especificada  
    considerando o *rendimento* do carro e o *preco* do litro do combustível.
```

Exemplos

```
>>> # (120.0 / 10.0) * 5.0  
>>> custo_viagem(120.0, 10.0, 5.0)  
60.0  
...
```

Observando a especificação, em particular os **exemplos**, generalizamos a forma de calcular a resposta. Nesse problema, só temos uma forma de resposta, então a generalização é direta.

```
return (distancia / rendimento) * preco
```

**Objetivo:** verificar se a implementação está de acordo com a especificação.

Usamos os exemplos para fazer a **verificação**.

No modo interativo, digitamos cada exemplo e conferimos se a resposta é a esperada:

```
>>> custo_viagem(120.0, 10.0, 5.0)
60.0
```

```
>>> custo_viagem(300.0, 15.0, 6.0)
120.0
```

Ok, as respostas são as esperadas.

Se na verificação um exemplo produzir uma resposta diferente da esperada, onde está o erro?

- No exemplo
- No código da função
- Em ambos

Primeiro conferimos os exemplos, se algum estiver errado, corrigimos o exemplo e fazemos a **verificação novamente**.

Se os exemplos estiverem corretos, então analisamos o corpo da função para tentar identificar e corrigir o erro. Após a alteração do código, fazemos a **verificação novamente**.

**Objetivo:** alterar a organização do programa para que fique mais fácil de ser lido, entendido e alterado.

Se modificarmos o código, precisamos fazer a **verificação novamente!**

Alguma parte desse processo parece repetitiva?

Sim, a verificação dos exemplos.

O podemos utilizar para executar tarefas repetitivas?

Um programa de computador.

Usaremos o próprio Spython para isso.

Quando clicamos em **Run** no Spython, ele formata o código, faz a verificação estática de tipos, executa os exemplos e por fim, se não identificou nenhuma falha, executa o programa.

Como o Spython identifica os exemplos que devem ser executados?

Através da biblioteca **doctest**, ele procura trechos de comentários semelhantes a uma seção do modo interativo (por isso escrevemos os exemplos com `>>>`), extrai os códigos de entrada e as saídas, executa cada código de entrada e verifica se a saída produzida é a esperada.

# Verificação automatizada

```
↑ ↓ ⚙️ 🗑️ ☰
1 # Análise
2 #
3 # Calcular o custo em reais para percorrer uma determinada distância levando em
4 # consideração o desempenho do carro e o preço do litro do combustível.
5
6 # Tipos de Dados
7 #
8 # Distância, rendimento, preço do litro e custo são representado por números
9 # positivos.
10
11
12 def custo_viagem(distancia: float, rendimento: float, preco: float) -> float:
13     """
14     Calcula o custo em reais para percorrer a *distancia* especificada
15     considerando o *rendimento* do carro e o *preco* do litro do combustível.
16
17     Exemplos
18     >>> custo_viagem(120.0, 10.0, 5.0)
19     60.0
20     >>> custo_viagem(300.0, 15.0, 6.0)
21     120.0
22     """
23     return (distancia / rendimento) * preco
24
```

Running tests...  
2 tests, 2 successes, 0 failures and 0 errors.  
>>>

Vamos fazer outro exemplo.

Um construtor precisa calcular a quantidade de azulejos necessários para azulejar uma determinada parede. Cada azulejo é quadrado e tem 20cm de lado. Ajude o construtor e defina uma função que receba como entrada o comprimento e a altura em metros de uma parede e calcule a quantidade de azulejos inteiros necessários para azulejar a parede. Considere que o construtor nunca perde um azulejo e que recortes de azulejos não são reaproveitados.

**Objetivo:** identificar o problema a ser resolvido.

- Quais informações são relevantes e quais podem ser descartadas?
- Existe alguma omissão?
- Existe alguma ambiguidade?
- Quais conhecimentos do domínio do problema são necessários?

## Resultado

Calcular o número de azulejos necessários para azulejar uma parede com determinado comprimento e altura. Cada azulejo mede 0,2m x 0,2m. Nenhum azulejo é perdido e recortes são descartados.

## Análise

Calcular o número de azulejos necessários para azulejar uma parede com determinado comprimento e altura. Cada azulejo mede 0,2m x 0,2m. Nenhum azulejo é perdido e recortes são descartados.

**Objetivo:** identificar e definir como as informações serão representadas.

- Quais são as informações envolvidas no problema?
- Como as informações serão representadas?

## Resultado

O comprimento e a altura da parede são dados em metros e representados com números positivos.

## Análise

Calcular o número de azulejos necessários para azulejar uma parede com determinado comprimento e altura. Cada azulejo mede 0,2m x 0,2m. Nenhum azulejo é perdido e recortes são descartados.

## Tipos de dados

O comprimento e a altura da parede são dados em metros e representados com números positivos.

**Objetivo:** especificar com mais precisão e com exemplos o que o programa deve fazer.

- Assinatura da função (nome, tipo das entradas e saídas)
- Propósito da função
- Exemplos de entrada e saída

```
def numero_azulejos(comprimento: float, altura: float) -> int:  
    ...  
    Calcula o número de azulejos de 0,2mx0,2m necessários para azulejar uma  
    parede de tamanho *comprimento* x *altura* (em metros) considerando que  
    nenhum azulejo é perdido e que recortes são descartados.  
    ...  
    return 0
```

Qual deve ser o resultado para `numero_azulejos(1.5, 2.3)`? 96 (discutido em sala).

```
>>> # math.ceil(1.5 / 0.2) * math.ceil(2.3 / 0.2)
```

```
>>> numero_azulejos(1.5, 2.3)
```

```
96
```

```
>>> # math.ceil(2.0 / 0.2) * math.ceil(2.4 / 0.2)
```

```
>>> numero_azulejos(2.0, 2.4)
```

```
120
```

```
>>> numero_azulejos(0.2, 0.2)
```

```
1
```

```
>>> numero_azulejos(0.3, 0.2)
```

```
2
```

```
>>> numero_azulejos(0.3, 0.3)
```

```
4
```

```
>>> numero_azulejos(0.4, 0.4)
```

```
4
```

Implementação

```
import math
```

```
def numero_azulejos(comprimento: float, altura: float) -> int:  
    return math.ceil(comprimento / 0.2) * math.ceil(altura / 0.2)
```

Verificação

6 passed and 0 failed.

Revisão

O código está ok.

O Jorge precisa saber a massa de diversos pequenos tubos de ferro mas está sem uma balança. No entanto, ele possui um paquímetro e pode medir com precisão o diâmetro interno e externo e a altura dos tubos, agora ele só precisa de um programa para fazer os cálculos. Algum voluntário?

Alguma coisa parece complicada nesse exercício?

Nesse exercício precisamos de conhecimento de um domínio (área de conhecimento), que talvez ainda não tenhamos, isso pode fazer o problema parecer mais difícil do que realmente é. Mas então, como proceder nesses casos?

Precisamos de uma pessoa (ou livros) que possam nos instruir sobre o conhecimento do domínio, geralmente os interessados no software podem indicar tais pessoas.

O importante é entender que o desenvolvedor de software geralmente resolve o problema de outras pessoas, e esses problemas podem envolver conhecimentos que não temos e por isso precisamos estar dispostos a estudar e aprender o conhecimento de outras áreas.

Vamos resolver esse problema, por onde começamos?

### Análise

- Calcular a massa de um tubo de ferro a partir das suas dimensões. Como as dimensões de um tubo de ferro estão relacionadas com a massa do tubo?
- Dimensões  $\rightarrow$  Volume  $\rightarrow$  Massa
- Como determinamos o volume de um tubo de ferro a partir das suas dimensões?

$$\pi \times \left( \left( \frac{\text{diámetro\_externo}}{2} \right)^2 - \left( \frac{\text{diámetro\_interno}}{2} \right)^2 \right) \times \text{altura}$$

- Como obtemos a massa a partir do volume?  $\text{volume} \times \text{densidade}$ .
- Qual é a densidade do ferro?  $7874 \text{ kg/m}^3$ .

### Definição de tipos de dados

- Comprimento é um número positivo dado em metros.
- Massa é um número positivo dado em quilogramas.

### Especificação

```
def massa_tubo_ferro(diametro_externo: float, diametro_interno: float, altura: float) -> float
```

```
    '''
```

```
    Calcula a massa de um tubo de ferro a partir das suas dimensões.
```

```
    Requer diametro_externo > diametro_interno.
```

### Exemplos

```
>>> # 3.14 * ((0.05 / 2) ** 2 - (0.03 / 2) ** 2) * 0.1 * 7874
```

```
>>> massa_tubo_ferro(0.05, 0.03, 0.1)
```

```
0.9889744
```

```
    '''
```

### Implementação

Direto a partir da especificação (do exemplo).

```
def massa_tubo_ferro(diametro_externo: float, diametro_interno: float, altura: float) -> float:  
    return 3.14 * ((diametro_externo / 2) ** 2 - (diametro_interno / 2) ** 2) * altura * 7874
```

### Verificação

Failed example:

```
    massa_tubo_ferro(0.05, 0.03, 0.1)
```

Expected:

```
    0.9889744
```

Got:

```
    0.9889744000000004
```

Comparação de igualdade de números de ponto flutuante quase não dá certo! Nesses casos, podemos arredondar o resultado.

```
>>> round(massa_tubo_ferro(0.05, 0.03, 0.1), 7)
0.9889744
```

### Revisão

```
def massa_tubo_ferro(diametro_externo: float, diametro_interno: float, altura: float) -> float:  
    return 3.14 * ((diametro_externo / 2) ** 2 - (diametro_interno / 2) ** 2) * altura * 7874
```

O que podemos melhorar?

- Definir constantes para os números “mágicos”
- Separar o cálculo em etapas

```
PI: float = 3.14
```

```
DENSIDADE_FERRO: float = 7874
```

```
def massa_tubo_ferro(diametro_externo: float, diametro_interno: float, altura: float) -> float:  
    area_externa = PI * (diametro_externo / 2) ** 2  
    area_interna = PI * (diametro_interno / 2) ** 2  
    volume = (area_externa - area_interna) * altura  
    return volume * DENSIDADE_FERRO
```

Constantes são geralmente definidas fora das funções (escopo global) e nomeadas com letras maiúsculas.

No período de 2015 a 2016 todos os números de telefones celulares no Brasil passaram a ter nove dígitos. Na época, os números de telefones que tinham apenas oito dígitos foram alterados adicionando-se o 9 na frente do número. Embora oficialmente todos os números de celulares tenham nove dígitos, na agenda de muitas pessoas ainda é comum encontrar números registrados com apenas oito dígitos. Projete uma função que adicione o nono dígito em um dado número de telefone celular caso ele ainda não tenha o nono dígito. Considere que os números de entrada são dados com o DDD entre parênteses e com um hífen separando os últimos quatro dígitos. Exemplos de entradas: (44) 9787-1241, (51) 95872-9989, (41) 8876-1562. A saída deve ter o mesmo formato, mas garantindo que o número do telefone tenha 9 dígitos.

### Análise

Ajustar o número de um telefone adicionando 9 como o nono dígito se necessário.

### Definição de tipo de dados

O número de telefone é uma string no formato (XX) XXXX-XXXX ou (XX) XXXXX-XXXX, onde X pode ser qualquer dígito.

### Especificação

A seguir.

## Exemplo - Ajuste número telefone

```
def ajusta_numero(numero: str) -> str:
    """
    Ajusta *numero* adicionando o 9 como nono dígito se necessário, ou seja, se
    *numero* tem apenas 8 dígitos (sem contar o DDD).

    Requer que numero esteja no formato (XX) XXXX-XXXX ou (XX) XXXXX-XXXX, onde
    X pode ser qualquer dígito.

    Exemplos
    >>>
    >>> ajusta_numero('(51) 95872-9989')
    '(51) 95872-9989'
    >>>
    >>> ajusta_numero('(44) 9787-1241')
    '(44) 99787-1241'
    """
    return numero
```

## Exemplo - Ajuste número telefone

```
def ajusta_numero(numero: str) -> str:
    '''
    Ajusta *numero* adicionando o 9 como nono dígito se necessário, ou seja, se
    *numero* tem apenas 8 dígitos (sem contar o DDD).

    Requer que numero esteja no formato (XX) XXXX-XXXX ou (XX) XXXXX-XXXX, onde
    X pode ser qualquer dígito.

    Exemplos
    >>> # não precisa de ajuste, a saída é a própria entrada
    >>> ajusta_numero('(51) 95872-9989')
    '(51) 95872-9989'
    >>> # '(44) 9787-1241'[:5] + '9' + '(44) 9787-1241'[5:]
    >>> ajusta_numero('(44) 9787-1241')
    '(44) 99787-1241'
    '''
    return numero
```

O que mudou na forma que calculamos a resposta dos exemplos desse projeto em relação aos exemplos dos projetos anteriores?

Nos projetos anteriores a resposta tinha apenas uma forma. Nesse projeto existem duas formas de resposta: ou a resposta é a própria entrada ou fazemos algumas operações específicas.

Como escolher entre uma forma de resposta e outra?

Usando instrução de seleção! Vamos continuar na próxima aula.

Quais são as etapas do processo de projeto de funções?

- Análise, definição dos tipos de dados, especificação, implementação, verificação e revisão.

Qual é o objetivo da análise?

- Identificar o problema a ser resolvido, descartando informações irrelevantes e esclarecendo ambiguidades.

Qual é o objetivo da definição dos tipos de dados?

- Identificar as informações e como elas serão representadas.

O que compõe a especificação de uma função?

- A assinatura (nome, parâmetros com tipos e tipo de retorno), o propósito e os exemplos.

Qual é a principal propriedade de uma boa especificação?

- Ser precisa o suficiente para que um desenvolvedor consiga fazer a implementação apenas a partir dela.

Como o Spython ajuda na verificação?

- O **Run** do Spython verifica os tipos e executa os exemplos automaticamente, conferindo se as saídas estão corretas.

Qual é o objetivo da revisão?

- Melhorar a organização do código para que fique mais fácil de ler e entender. Após qualquer modificação, a verificação deve ser feita novamente.