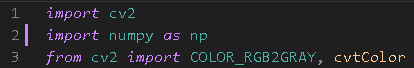
Explicação do código

**Image.py**

Basicamente, o arquivo image.py é quem está responsável pelos tratamentos ou alguma manipulação na imagem que podem ser necessários, como o desenho dos círculos na imagem para se identificar a área do furo e da “coroa”.

**Importações:**



cv2: Biblioteca OpenCV para processamento de imagem.

numpy: Biblioteca para operações numéricas, especialmente com arrays (imagens são tratadas como arrays).

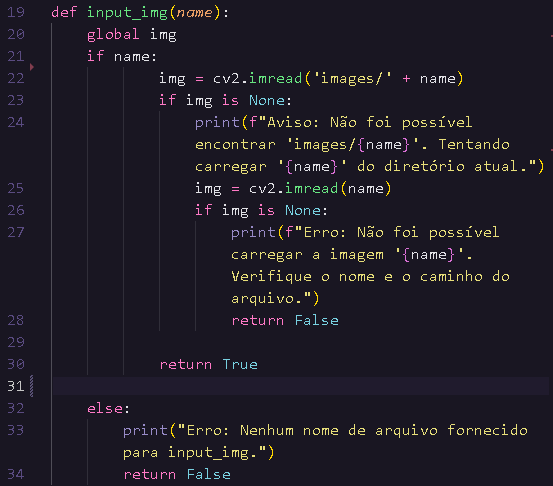
COLOR\_RGB2GRAY, cvtColor: Funções específicas do OpenCV para conversão de espaço de cores.

**Variáveis Globais:**

img = None  
center\_x = 0  
center\_y = 0  
lines = 0  
cols = 0  
diam\_drill = 6  
diam\_delamina = 6.5  
ray\_drill = diam\_drill/2  
ray\_delamina = diam\_delamina/2  
wResi = 0  
hResi = 0

* img: Armazena a imagem carregada (inicialmente None).
* center\_x, center\_y: Coordenadas do centro da imagem (calculadas na função measures()).
* lines, cols: Dimensões da imagem em pixels (altura/linhas, largura/colunas) (calculadas na função measures()).
* diam\_drill, diam\_delamina: Diâmetros inicializados com valores em mm.
* ray\_drill, ray\_delamina: Raios correspondentes aos diâmetros (calculados na função measures()).
* wResi, hResi: Largura e altura da janela de exibição (calculadas em measures).

**Função “input\_img(name)”:**



A função “input\_img(name)” receberá a imagem que o usuário quer verificar.

Declara img como global para modificar a variável global.

•Verifica se “name” foi fornecido.

•Tenta carregar a imagem do diretório images/ usando cv2.imread(). imread retorna um array NumPy representando a imagem ou None se falhar.

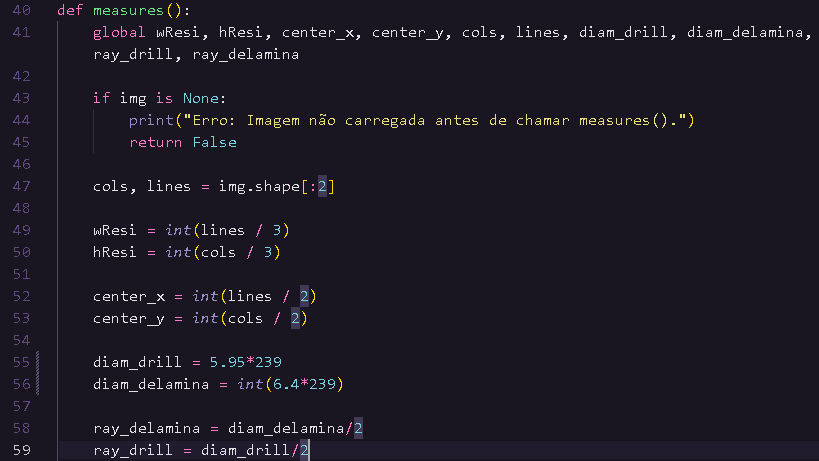
•Se falhar ao carregar de images/, tenta carregar do diretório atual.

•Se falhar novamente, imprime um erro e retorna False.

•Se carregar com sucesso, retorna True.

•Se nenhum nome for fornecido, imprime um erro e retorna False.

**Função “measures()”:**



Declara as variáveis globais que serão modificadas.

•Verifica se a imagem (img) foi carregada; se não, imprime erro e retorna False.

•Obtém as dimensões da imagem: cols (largura) e lines (altura) a partir de img.shape[:2].

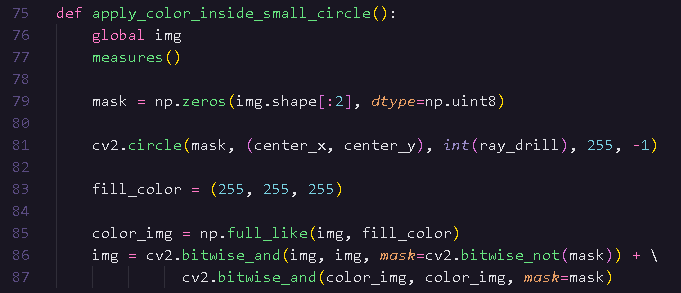
•Calcula as dimensões da janela de exibição (wResi, hResi) como um terço das dimensões originais.

•Calcula as coordenadas do centro (center\_x, center\_y). Atenção: Parece haver uma troca entre lines (altura) e cols (largura) no cálculo do centro (usa lines para center\_x e cols para center\_y, o que pode estar invertido dependendo da convenção x,y).

•Recalcula diam\_drill e diam\_delamina em pixels, usando valores (6 mm e 6.6 mm) que devem ser alterados conforme necessário e uma escala fixa (239 px/mm). Isso sobrescreve os valores iniciais das variáveis globais. Serão utilizados para calcular o raio.

•Calcula os raios (ray\_drill, ray\_delamina) em pixels a partir dos diâmetros em pixels. Serão utilizados para realizar as circunferências sobre as imagens

**Função “apply\_color\_inside\_small\_circle()”:**



Essa função insere uma cor sólida no interior do círculo menor, isolando quaisquer pixels a mais que possam tirar parte da precisão na quantidade de pixels que representam a delaminação (que serão utilizados para calcular a área da delaminação).

Declara img como global.

•Chama measures().

•Cria uma máscara (mask) preta do mesmo tamanho da imagem.

•Desenha um círculo branco preenchido (255, thickness=-1) na máscara, usando o centro e ray\_drill (raio do furo em pixels).

•Define a cor de preenchimento (fill\_color) como branco (255, 255, 255).

•Cria uma imagem (color\_img) totalmente preenchida com fill\_color.

•Usa cv2.bitwise\_and para combinar as imagens:

•cv2.bitwise\_and(img, img, mask=cv2.bitwise\_not(mask)): Mantém os pixels da imagem original (img) onde a máscara invertida é branca (ou seja, fora do círculo pequeno).

•cv2.bitwise\_and(color\_img, color\_img, mask=mask): Mantém os pixels da imagem de cor sólida (color\_img - branca) onde a máscara original é branca (ou seja, dentro do círculo pequeno).

•A soma dessas duas partes resulta na imagem original com a área dentro do círculo pequeno substituída pela cor branca.

•Atualiza a variável global img com o resultado.

•Imprime os raios.

**Explicação parâmetros da função “bitwise\_and”:**

Respectivamente: src1, src2, mask, dst. Src1 e 2 são obrigatórios, os demais, opcionais.

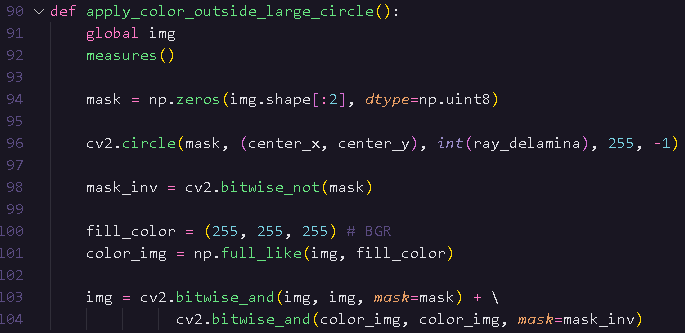
Src1: 1° imagem de entrada (pode ser usada com ela mesma).

Src2: Deve ter o tamanho e tipo igual que “Src1”.

Mask: Mácara de aplicação, imagem em escala de cinza.

Dst: Saída da operação, onde será alocada as alterações nas imagens.

**Função “apply\_color\_outside\_circle()”:**

****

Essa função insere uma cor sólida no que corresponde ao que contorna o círculo maior, isolando quaisquer pixels a mais que possam tirar parte da precisão na quantidade de pixels que representam a delaminação (que serão utilizados para calcular a área da delaminação).

Declara img como global.

•Chama measures().

•Cria uma máscara (mask) preta do mesmo tamanho da imagem.

•Desenha um círculo branco preenchido (255, thickness=-1) na máscara, usando o centro e ray\_delamina (raio da delaminação em pixels).

•Inverte a máscara (mask\_inv = cv2.bitwise\_not(mask)). Agora, a área fora do círculo é branca na mask\_inv.

•Define a cor de preenchimento (fill\_color) como branco.

•Cria uma imagem (color\_img) totalmente preenchida com fill\_color.

•Usa cv2.bitwise\_and para combinar as imagens:

•cv2.bitwise\_and(img, img, mask=mask): Mantém os pixels da imagem original (img) onde a máscara original é branca (ou seja, dentro do círculo grande).

•cv2.bitwise\_and(color\_img, color\_img, mask=mask\_inv): Mantém os pixels da imagem de cor sólida (color\_img - branca) onde a máscara invertida é branca (ou seja, fora do círculo grande).

•A soma dessas duas partes resulta na imagem original com a área fora do círculo grande substituída pela cor branca.

•Atualiza a variável global img com o resultado.

**Explicação parâmetros da função “bitwise\_and”:**

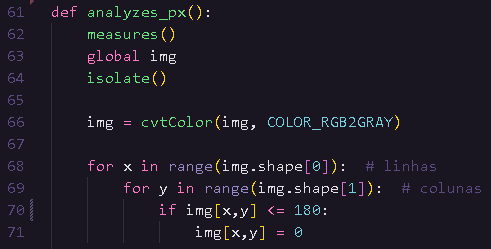
Respectivamente: src1, src2, mask, dst. Src1 e 2 são obrigatórios, os demais, opcionais.

Src1: 1° imagem de entrada (pode ser usada com ela mesma).

Src2: Deve ter o tamanho e tipo igual que “Src1”.

Mask: Mácara de aplicação, imagem em escala de cinza.

Dst: Saída da operação, onde será alocada as alterações nas imagens.

**Função “analyzes\_px()”:**  


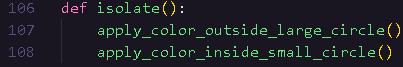
Faz uma verificação do valor de cada pixel, arredondando-o para preto ou branco, tornando os valores da imagem padronizados, apenas branco ou preto, deixando de existir a escala de cinza.

Chama measures() para garantir que as dimensões e raios estejam calculados/atualizados.

•Declara img como global.

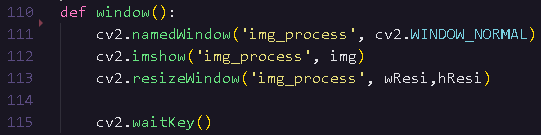
•Chama isolate() para aplicar as máscaras branca nas áreas fora do círculo maior e dentro do círculo menor.

•Converte a imagem img (já modificada por isolate) para escala de cinza usando cvtColor.

**Função “isolate()”:  
**

Simplesmente chama as duas funções em sequência: primeiro preenche a área externa ao círculo maior, depois preenche a área interna ao círculo menor. O resultado é que a imagem global img fica com a "coroa" (área entre os dois círculos) preservada e o resto preenchido de branco.

**Função “window()”:**



A função “window()” está responsável por retornar uma janela que exibe o resultado da imagem com o círculo que demarca as bordas da delaminação(alcance máximo) e do furo.

Cria uma janela nomeada ('img\_process') que permite redimensionamento (cv2.WINDOW\_NORMAL).

•Exibe a imagem global img na janela criada. Importante: A imagem exibida é o resultado final do processamento realizado pelas chamadas anteriores em run.py (ou seja, a imagem binarizada e isolada).

•Redimensiona a janela usando as dimensões wResi e hResi calculadas em measures().

•cv2.waitKey(): Pausa a execução e espera que o usuário pressione qualquer tecla. Sem isso, a janela poderia fechar imediatamente. O valor 0 significa esperar indefinidamente.

**Explicação dos parâmetros de “cv2.namedWindow()”:**

1°parâmetro (‘img\_process’): Nome atribuído a janela.

2°parâmetro (cv2.WINDOW\_NORMAL): Representa se o tamanho da janela é ajustado automaticamente ou ajustável. Como o parâmetro passado é “WINDOW\_NORMAL”, o tamanho da janela se torna ajustável. “WINDOW\_NORMAL” deve ser importado na classe cv2(openCv).

**Explicação dos parâmetros de “cv2.imshow()”:**

1°parâmetro(‘img\_process’): Deve ser colocado entre aspas simples (‘’) o nome da janela (mesmo nome definido na função “cv2.namedWindow()”) que o programa exibirá a imagem.

2°parâmetro(img\_process\_rgb): Imagem que será mostrada na janela, deve ser inserida por meio da variável que armazena a imagem.

**Explicação dos parâmetros de “cv2.resizeWindow()”:**

1°parâmetro(‘img\_process’): Nome da janela que terá o tamanho ajustado, deve ser colocado entre aspas simples (‘’).

2°parâmetro(wResi): Comprimento (em pixeis) que a janela deverá ter, pode ser inserido por meio de uma variável (como foi feito no código) ou colocar um valor diretamente. Exemplo:cv2.resizeWindow(‘img\_process’, **1000**...).

3°parâmetro(hResi): Altura (em pixeis) que a janela deverá ter, pode ser inserido por meio de uma variável (como foi feito no código) ou colocar um valor diretamente. Exemplo:cv2.resizeWindow(‘img\_process’, wResi**, 1000**).

**calculo.py**

Este módulo está responsável por realizar cálculos que serão necessários para se extrair o fator de delaminação(*F*) das imagens.

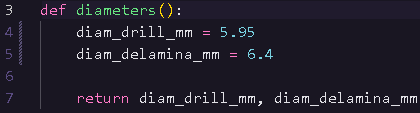
**Importações:**

****

image: Importação do módulo “image” (image.py).

as img: Forma como o módulo será chamado para acessar seus métodos.

**Função “diameters()”:**



Essa função está responsável por aramzenar os diâmetros dos círculos que foram desenhados para demarcar a circunferência do furo e do alcance máximo da delaminação.

“cols”: Armazena o comprimento da imagem inserida pelo usuário.

“lines”: Armazena a altura da imagem inserida pelo usuário.

“diam\_drill”: Calcula o diâmetro do furo presente na imagem (é o mesmo valor de “diam\_drill” presente em “image.py”).

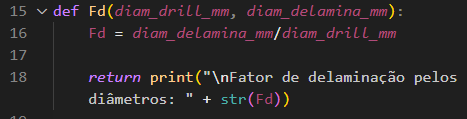
“diam\_delamina”: Calcula o diâmetro do alcance máximo presente na imagem (é o mesmo valor de “diam\_delamina” presente em “image.py”).

“diam\_drill\_mm” e “diam\_delamina\_mm”: Conversão dos diâmetros do furo e do alcance máximo da delaminação (respectivamente) de pixels para milímetros (pois é a escala determinada com base nas imagens que foram utilizadas durante o desenvolvimento).

“return diam\_drill\_mm, diam\_delamina\_mm”: Retorna os diâmetros convertidos para sejam utilizados em outros métodos.

“238.9869”: Corresponde a quantidade de pixels um milímetro possui baseado na escala das imagens utilizadas como base para o desenvolvimento.

**Função “Fd(diam\_drill\_mm, diam\_delamina\_mm)”:**

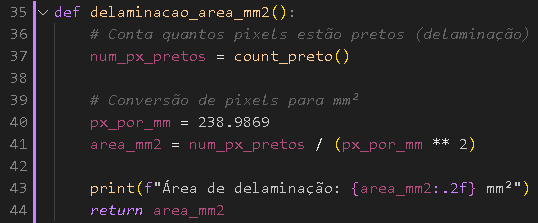
****

A função acima calcula o fator de delaminação comum (*Fd*) que é extraído a partir da divisão do diâmetro de maior alcance da delaminação pelo diâmetro do furo.

“Fd”: Variável que calcula o fator de delaminação.

“return print(...)”: Retorna um texto informando o fator de delaminação comum (a partir dos diâmetros).

**Função “delaminação\_area\_mm2”:**



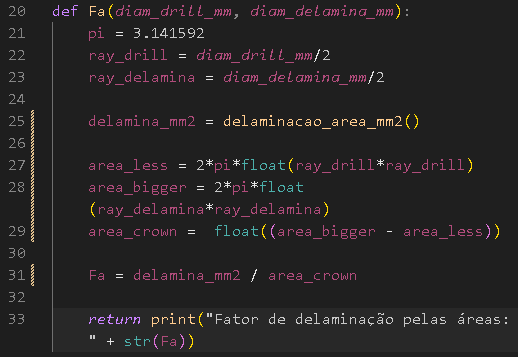
Faz a conversão da área da delaminação de pixels para milímetros².

“num\_px\_pretos”: Armazena a quantidade de pixels que correspondem a área da delaminação. Se a área da delaminação estiver em preto chame “count\_preto()”, mas se representado por branco, então “count\_branco()”.

“px\_por\_mm”: Quantidade de pixels por milímetros.

“área\_mm2”: Área da delaminação em mm².

**Função “Fa(diam\_drill\_mm, diam\_delamina\_mm)”:**



Calcula o fator de delaminação por meio da razão entre a área da delaminação e pela área da coroa (área que abrange toda delaminação – área do furo).

“pi”: Armazena o valor de pi para calcular a área das circunferências.

“ray\_drill”: Raio do círculo do furo.

“ray\_delamina”: Raio do círculo que engloba toda a delaminação.

“delamina\_mm2” = Armazena a área em mm² da delaminação.

“area\_less”: Calcula a área do círculo do furo.

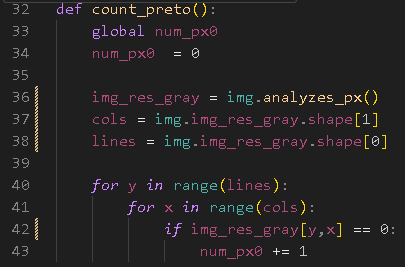
“area\_bigger”: Calcula a área que engloba toda a delaminação.

“área\_crown”: Calcula a área da coroa.

“Fa”: Calcula o fator de delaminação utilizando a área da delaminação e a coroa que engloba toda a delaminação.

“return print(...)”: Retorna um texto que exibe o fator de delaminação pelas áreas.

**Função “count\_preto()”:**



Essa função realiza a contagem de pixels pretos para se calcular a delaminação a partir da quantidade de pixels. Porém ela só deve ser usada caso a delaminação esteja demarcada, representada por pixels pretos.

“num\_px0”: Variável que armazenará a quantidade de pixels pretos que constituem a delaminação.

“img\_res\_gray”: Importação da imagem com todos pixels “arredondados” para completamente preto ou branco.

“cols”: Obter o comprimento da imagem.

“lines”: Obter a altura da imagem.

“range()”: Retorna a quantidade de itens presentes em uma matriz, independentemente de ser unidimensional ou não, por isso foi utilizado

“for y in range(lines)”: A variável “y” percorrerá pela matriz “lines”. “lines” foi utilizada como base por possuir mais itens que “cols”.

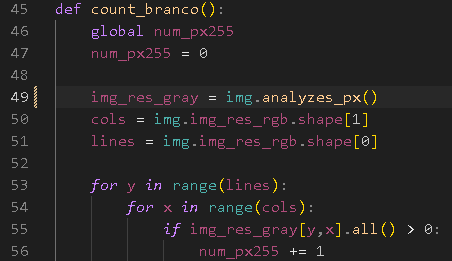
“for x in range(cols): A variável x corresponderá a todos pixels da imagem (um por vez), será por meio dela que será feito a verificação da tonalidade de cinza do pixel (valor = 0, pixel preto).

“if img\_res\_gray[y,x].all() == 0”: Verifica os valores do pixel da imagem. Leitura prática: “Se o pixel da posição[y,x] for = 0 (preto), acrescente mais um na variável que armazena a quantidade de pixels pretos (num\_px0)”.

“img\_res\_gray”: é utilizado por conseguir destacar melhor os pixels, devido a escala de cinza.

“img\_res\_gray[y,x]”: Corresponde a posição do pixel que será verificado.

“all()”: Leva em consideração todos os valores de cada item da matriz, embora a imagem esteja em uma escala de cinza, seus pixels ainda são constituídos por 3 valores, por isso, os 3 valores precisam ser analisados simultaneamente, precisam atender a condição.

**Função “count\_branco()”:**

Essa função realiza a contagem de pixels brancos para se calcular a delaminação a partir da quantidade de pixels. Porém ela só deve ser usada caso a delaminação esteja demarcada, representada por pixels brancos.

“num\_px255”: Variável que armazenará a quantidade de pixels brancos que constituem a delaminação.

“img\_res\_gray”: Importação da imagem com todos pixels “arredondados” para completamente preto ou branco.

“cols”: Obter o comprimento da imagem.

“lines”: Obter a altura da imagem.

“range()”: Retorna a quantidade de itens presentes em uma matriz, independentemente de ser unidimensional ou não, por isso foi utilizado

“for y in range(lines)”: A variável “y” percorrerá pela matriz “lines”. “lines” foi utilizada como base por possuir mais itens que “cols”.

“for x in range(cols): A variável x corresponderá a todos pixels da imagem (um por vez), será por meio dela que será feito a verificação da tonalidade de cinza do pixel (valor = 255, pixel != preto).

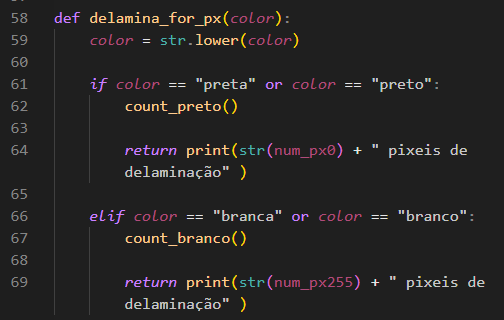
“if img\_res\_gray[y,x].all() == 0”: Verifica os valores do pixel da imagem. Leitura prática: “Se o pixel da posição[y,x] for = 255 (branco), acrescente mais um na variável que armazena a quantidade de pixels brancos (num\_px255)”.

“img\_res\_gray” é utilizado por conseguir destacar melhor os pixels, devido a escala de cinza.

“img\_res\_gray[y,x]”: Corresponde a posição do pixel que será verificado.

“all()”: Leva em consideração todos os valores de cada item da matriz, embora a imagem esteja em uma escala de cinza, seus pixels ainda são constituídos por 3 valores, por isso, os 3 valores precisam ser analisados simultaneamente, precisam atender a condição.

**Função “delamina\_for\_px(color)”:**

****

Realiza o cálculo da delaminação por meio da contagem dos pixels que constituem a delaminação.

“color(parâmetro)”: Parâmetro passado que possuirá seu valor inserido pelo usuário. Corresponde a cor que a delaminação está marcada.

“color(variável)”: Recebe o valor passado pelo parâmetro da função (um texto) e deixa-o todo em minúsculo.

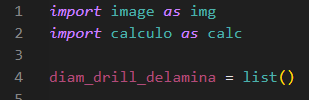
1° “if”: Se a delaminação estiver representada por pixels pretos, é chamado a função “count\_preto()” para se obter a quantidade de pixels pretos (num\_px0). “return print(...)” exibe por meio de um texto a quantidade de pixels da delaminação.

2° “if”: Se a delaminação estiver representada por pixels brancos, é chamado a função “count\_branco()” para se obter a quantidade de pixels brancos (num\_px255). “return print(...)” exibe por meio de um texto a quantidade de pixels da delaminação.

**run.py**

“run.py” é responsável pela “interação” com o usuário, ele fará as chamadas dos métodos necessários para realizar os cálculos e passar o resultado para o usuário.

**Importações**

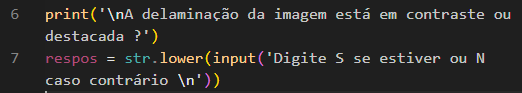
****

“*import* image *as* img”: Importa o módulo “image” (image.py) e o torna possível ser chamado por apenas “img”.

“*import* calculo *as* calc”: Importa o módulo “calculo” (calculo.py) e o torna possível ser chamado por apenas “calc”.

“diam\_drill\_delamina”: Variável do tipo lista que armazenará os valores de delaminação.

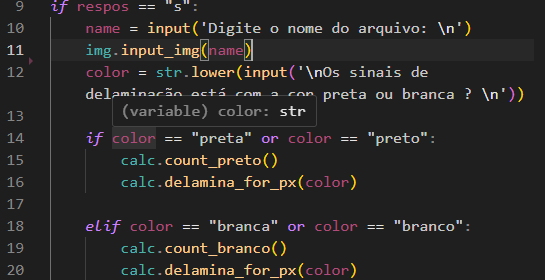
**1° “print(...)”**



Responsável por decidir se será necessário a utilização de algum método do módulo “image”, que é utilizado para manipulação da imagem.

“respos”: Variável que armazena (em minúsculo – str.lower()) a resposta do usuário com relação a delaminação da imagem.

**Verificação da resposta do usuário (parte 1):**



Identificará os métodos que deverão ser chamados caso a delaminação da imagem do usuário esteja contrastada ou destacada.

“name”: Variável que armazena o nome do arquivo que contém a imagem.

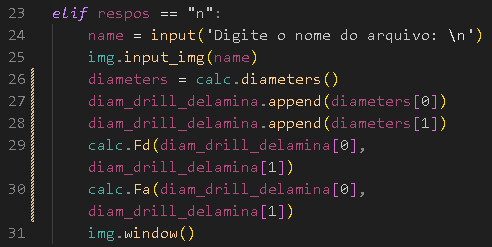
“img.input\_img(name)”: Passa para a função “input\_img()” como parâmetro a variável “name”.

“color”: Variável que receberá a cor pela qual a delaminação está em destaque.

*“if* color == "preta" or color == "preto"”: Verifica se a cor da delaminação está destacada em preto. Caso esteja, chamará a função responsável por contar a quantidade de pixels pretos (calc.count\_preto()) e a que determina a delaminação por pixels (calc.delamina\_for\_px(color) – Color é passado como parâmetro para função).

*“elif* color == "branca" or color == "branco"”: Verifica se a cor da delaminação está destacada em branco. Caso esteja, chamará a função responsável por contar a quantidade de pixels brancos (calc.count\_branco()) e a que determina a delaminação por pixels (calc.delamina\_for\_px(color) – Color é passado como parâmetro para função).

**Verificação da resposta do usuário (parte 2):**

****

Responsável por chamar as funções responsáveis pela manipulação da imagem para a análise da delaminação.

“name”: Variável que armazena o nome do arquivo que contém a imagem.

“img.input\_img(name)”: Passa para a função “input\_img()” como parâmetro a variável “name”.

“diameters”: Armazena os valores dos diâmetros em mm².

“diam\_drill\_delamina.append(diameters[0])”: Adiciona na variável “diam\_drill\_delamina” por meio do método “append()” o valor do diâmetro em mm² do furo.

“ diam\_drill\_delamina.append(diameters[1])”: Adiciona na variável “diam\_drill\_delamina” por meio do método “append()” o valor do diâmetro em mm² da circunferência que engloba toda a delaminação.

“calc.Fd(diam\_drill\_delamina[0], diam\_drill\_delamina[1])”: Chama a função responsável pelo cálculo da delaminação por meio dos diâmetros (calc.Fd()) que recebe como parâmetro os valores de obtidos na conversão dos diâmetros de pixels para milímetros.

“calc.Fa(diam\_drill\_delamina[0], diam\_drill\_delamina[1])”: Chama a função responsável pelo cálculo da delaminação por meio das áreas (calc.Fa()) que recebe como parâmetro os valores de obtidos na conversão dos diâmetros de pixels para milímetros.

“img.window()”: Exibi a imagem com os círculos desenhados. Sua função é mais visual, para que o usuário saiba se será necessário manipular o tamanho do diâmetro em pixels, para que dai sim os círculos contornem o furo e inclua todos sinais de delaminação.