

Bloco de programação

No Cronapp, os blocos de programação são estruturas visuais que permitem arrastar e acoplar (*drag-and-drop*) de modo a gerar uma estrutura lógica. Existem centenas de blocos, desde os mais simples, como [SE](#) (if) ou [Imprime](#) (print ou console.log) até blocos complexos, onde o desenvolvedor só precisa passar os parâmetros e o código embutido no bloco se encarrega de fazer todo o processo, exemplo do [Enviar e-mail](#).

Os blocos são gerados a partir do [Blockly](#), tecnologia criada em 2012 pelo Google e tinha como objetivo inicial facilitar o estudo de programação por crianças. O Cronapp incorporou o *blockly* de modo a reduzir a curva de aprendizado ao mesmo tempo em que consegue aumentar a produtividade de usuários experientes. Dessa forma, o desenvolvedor praticamente não precisa se preocupar com a sintaxe das linguagens e foca no que é realmente importante: a lógica e regra de negócio do seu projeto.

Esse recurso está disponível tanto no código fonte servidor, quanto no código fonte cliente web e mobile. A Figura 1 apresenta uma função de bloco de programação Servidor e o código Java gerado, já a figura 1.1 mostra uma função bloco de programação cliente (web ou mobile) e o código Javascript gerado.

A subaba **Código** só fica disponível quando a opção **Modo Avançado** está **habilitada** ou ao habilitar a opção **Sempre mostrar código fonte** na janela de preferências do projeto, acessível a partir do menu do sistema **Espaço de Trabalho > Preferências**. Para mais informações dessas funcionalidades, acesse o tópico "Espaço de Trabalho" na documentação [Tour pelo Cronapp](#).

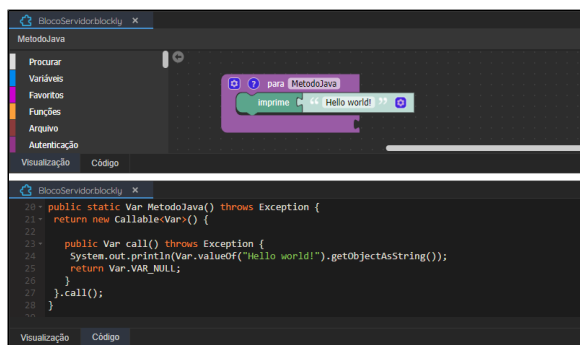


Figura 1 - Bloco servidor e o que foi gerado de código Java

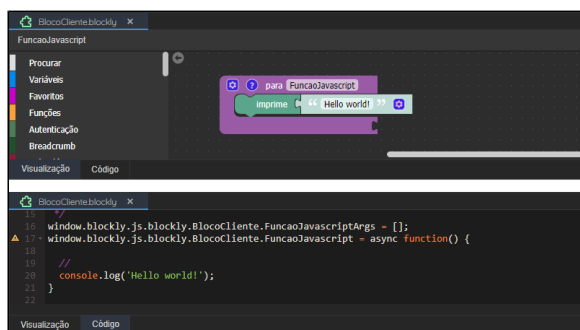


Figura 1.1 - Bloco cliente e o que foi gerado de código Javascript

De modo resumido, citamos as principais vantagens em utilizar os blocos de programação no Cronapp:

- Em geral, as funções se tornam mais legíveis;
- O processo de codificação é parecido tanto para back-end quanto front-end;
- Menor preocupação com a sintaxe das linguagens;
- Você pode criar e compartilhar, caso queira, seus próprios blocos de programação;
- É possível desenvolver parte do projeto usando o blockly e na outra parte codificar de forma convencional;
- Visualize o código gerado em tempo real (apenas quando o [Modo Avançado](#) ou a opção **Sempre mostrar código fonte** estiver habilitada);
- Altere o código-fonte gerado.

Nesta página

- [Arquivos de desenvolvimento](#)
 - [Desativando o Modo Avançado](#)
- [Criar arquivo Blockly](#)
- [Janela de edição](#)
 - [Funções](#)
 - [Estrutura das funções](#)
 - [Parâmetros](#)
 - [Propriedades do Blockly](#)
 - [Cliente](#)
 - [Servidor](#)
 - [Ordem de execução](#)
 - [Debug Visual](#)
 - [Breakpoints](#)
 - [Arquivo Java](#)
 - [Arquivo map](#)
 - [Nomenclatura](#)
 - [Menu de contexto](#)
 - [Favoritos](#)
 - [Comentários](#)
 - [Ponto de interrupção](#)
 - [Colapsar e expandir](#)
 - [Criar bloco](#)
 - [Deletar](#)
 - [Habilitar e desabilitar](#)
 - [Destacar](#)
 - [Duplicar](#)
 - [Adicionar ou Remover retorno](#)
 - [Entradas externas x incorporada](#)
 - [Retorno](#)
 - [Ajuda](#)
 - [Desfazer e Refazer](#)
 - [Limpar blocos](#)
- [Informações](#)
 - [Ajuda](#)
 - [Ícone engrenagem](#)
 - [Desacoplamento de blocos](#)
 - [Outros parâmetros](#)
 - [Comandos](#)
 - [Caixa de seleção](#)
 - [Gerador de documentação](#)
- [Blocos assíncronos](#)
 - [Como atualizar](#)
 - [Verificar atualização](#)
- [Criar blocos low-code customizados](#)

Conteúdo complementar

Além das centenas de blocos de programação (servidor e clientes) existentes no Cronapp, também é possível criar seus próprios blocos customizados, acesse o tópico e documentações abaixo para mais detalhes:

Arquivos de desenvolvimento

O editor visual dos blocos de programação costuma gerar outros arquivos além do código fonte. Porém, durante o desenvolvimento só é necessário nos preocuparmos com os arquivos que possuem as extensões ***.blockly** e ***.blockly.js**, pois são eles que abrem o editor visual. Os demais arquivos são mantidos e atualizados automaticamente quando salvamos o conteúdo no editor visual (Figura 2.1). Para visualizar as funções contidas em um arquivo de bloco de programação, basta expandir seu conteúdo clicando no símbolo ">" localizado ao lado dos arquivos blockly. Ao fazer isso, as funções serão exibidas em ordem alfabética (destaques 1 da figura 2.1) e com um duplo clique sobre uma dessas funções, é possível abri-la no editor de blocos de programação.

Importante

Nas pastas **blockly** (Localização: Lógicas/) deve conter apenas diretórios e arquivos relacionados aos blocos, sejam eles servidores (arquivos *.blockly, *.java e *.map) ou clientes (arquivos *.blockly e *.blockly.js).

- Tópico [Criar blocos low-code customizados](#)
- [Criando blocos cliente customizados \(mobile e web\)](#)
- [Criando blocos servidor customizados](#)

Atualização para a versão 2.6

A partir da versão 2.6, os blocos de programação clientes (web e mobile) funcionarão de forma assíncrona. Assim, caso você tenha criado um projeto em uma versão anterior e migrado para a nova versão, será necessário atualizar os blocos de programação clientes. Não é necessário realizar nenhuma ação para projetos criados após essa versão.

Acesse o tópico [Blocos assíncronos](#) para mais detalhes.

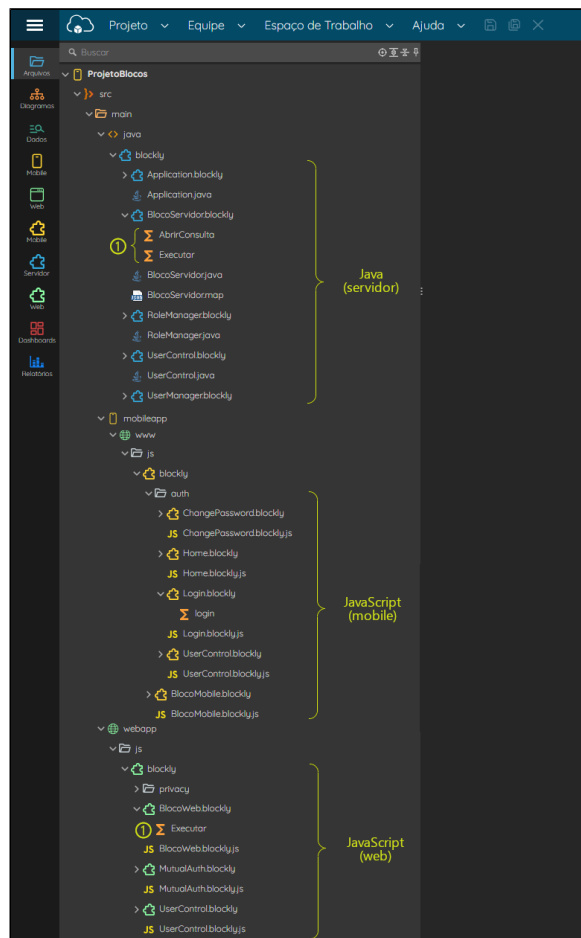


Figura 2.1 - Imagem editada para exibir apenas os arquivos gerados a partir do editor blockly

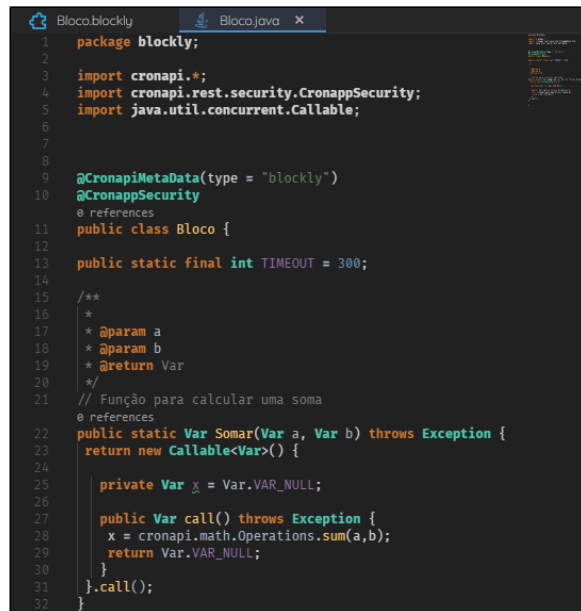
- **Arquivos Servidor**
 - **.blockly**: abre o editor do bloco servidor e é responsável por armazenar os blocos e gerar o código fonte Java. Ícones do editor de bloco servidor são exibidos sempre em azul.
 - **.java**: arquivo fonte Java gerado a partir do arquivo *.blockly.
 - **.map**: arquivo JSON gerado a partir do editor, é responsável por fazer o controle dos blocos utilizados e permitir o debug visual nos blocos servidor.
- **Arquivos Mobile**
 - **.blockly.js**: abre o editor do bloco mobile e é responsável por armazenar os blocos e gerar o código fonte JavaScript. Ícones do editor de bloco cliente são exibidos sempre em verde.
 - **.js**: arquivo fonte JavaScript gerado a partir do arquivo *.blockly.js.

- **Arquivos Web**

- **.blockly.js**: abre o editor do bloco web e é responsável por armazenar os blocos e gerar o código fonte JavaScript. Ícones do editor de bloco cliente são exibidos sempre em verde.
- **.js**: arquivo fonte JavaScript gerado a partir do arquivo *.blockly.js.

É possível alterar diretamente o código fonte Java ou JavaScript gerado pelo editor *blockly*, porém essas alterações não serão refletidas nos blocos de programação. Assim, se voltar a utilizar o editor *blockly*, as alterações feitas diretamente no código fonte serão perdidas.

Para editar o código de forma **High-Code** é necessário acessar o arquivo **.java** ou **.js** gerado a partir do **.blockly** ou **.blockly.js**.

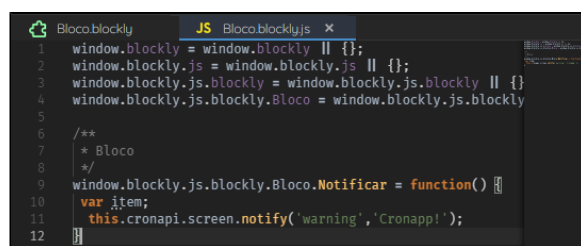


```

1 package blockly;
2
3 import cronapi.*;
4 import cronapi.rest.security.CronappSecurity;
5 import java.util.concurrent.Callable;
6
7
8
9 @CronapiMetadata(type = "blockly")
10 @CronappSecurity
11 // references
12 public class Bloco {
13
14     public static final int TIMEOUT = 300;
15
16     /**
17      * @param a
18      * @param b
19      * @return Var
20      */
21     // Função para calcular uma soma
22     // references
23     public static Var Somar(Var a, Var b) throws Exception {
24         return new Callable<Var>() {
25
26             private Var x = Var.VAR_NULL;
27
28             public Var call() throws Exception {
29                 x = cronapi.math.Operations.sum(a,b);
30                 return Var.VAR_NULL;
31             }
32         }.call();
33     }
34 }

```

Figura 2.2 - Código Java gerado a partir do Blockly servidor



```

1 window.blockly = window.blockly || {};
2 window.blockly.js = window.blockly.js || {};
3 window.blockly.js.blockly = window.blockly.js.blockly || {};
4 window.blockly.js.blockly.Bloco = window.blockly.js.blockly
5
6 /**
7  * Bloco
8  */
9 window.blockly.js.blockly.Bloco.Notificar = function() {}
10 var item;
11 this.cronapi.screen.notify('warning', 'Cronapp!');
12

```

Figura 2.3 - Código JavaScript gerado a partir do *Blockly mobile* ou *web*

Para mais detalhes sobre a organização dos arquivos dos projetos Cronapp, acesse a documentação [Estrutura de arquivos](#).

Desativando o Modo Avançado

Essa opção, acessível a partir do botão do menu do sistema (destaque 1 da figura 2.4), tem o objetivo de facilitar o desenvolvimento no Cronapp, ocultando recursos com foco no desenvolvimento *high-code*. Ao desabilitar o **Modo Avançado**, o conteúdo da árvore de arquivos é exibido de forma simplificada, os arquivos *.blockly, *.blockly.js e *.map ficam ocultos.

Devido a forma como o conteúdo é organizado e os arquivos ocultos, acessar os blocos de programação e suas funções com o Modo Avançado desabilitado (figura 2.4) é muito mais simples que na figura 2.1. De modo similar, para visualizar as funções contidas em um arquivo de bloco de programação, basta expandir seu conteúdo clicando no símbolo ">" localizado ao lado dos arquivos blockly. Ao fazer isso, as funções serão exibidas em ordem alfabética e, com um duplo clique sobre uma dessas funções, é possível abri-la no editor de blocos de programação.

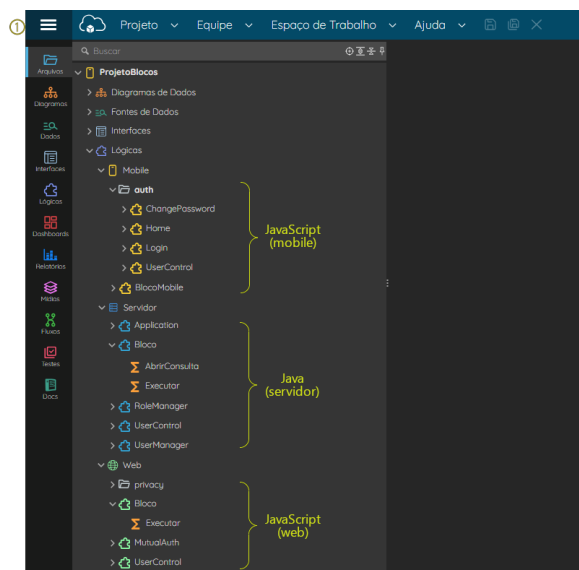


Figura 2.4 - Arquivos blockly sendo exibidos com a opção Modo Avançado desabilitada

Criar arquivo Blockly

Os projetos Cronapp possuem diretórios específicos para armazenar seus blocos de programação, recomendamos criar subdiretórios de forma a melhorar a organização dos seus arquivos fonte. Alguns diretórios podem ser facilmente acessados a partir dos atalhos na árvore de arquivos.

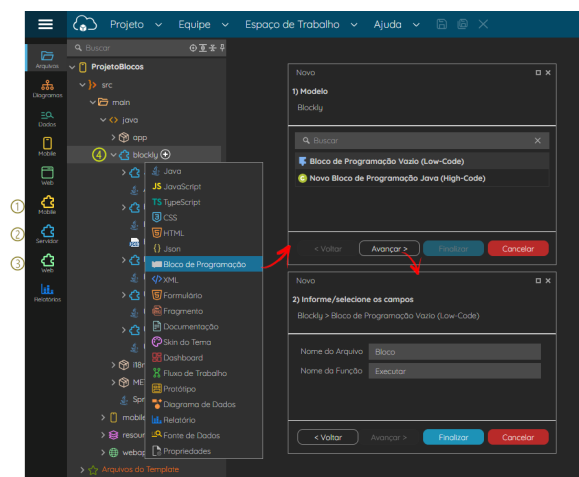


Figura 3.1 - Acesse o menu de contexto para criar blocos de programação

1. **Atalho bloco mobile:** clique para ser direcionado para a pasta do bloco de programação mobile.
2. **Atalho bloco servidor:** clique para ser direcionado para a pasta do bloco de programação servidor.
3. **Atalho bloco web:** clique para ser direcionado para a pasta do bloco de programação web.
4. **Bloco de programação:** diretório dos blocos de programação servidor.

Para criar um arquivo da sua regra de negócio, clique com no botão "+" ao lado do diretório **blockly/** (B1) oco de programação/ servidor, *mobile* ou *web*) e selecione a opção **Bloco de programação** para abrir a janela de configurações (Figura 3.1). Essa opção também é acessível a partir do menu de contexto do diretório.

Na primeira janela (Modelo) é possível definir como irá trabalhar:

- **Bloco de Programação Vazio (Low-Code):** cria um arquivo blockly que abre o editor visual e permite gerar o arquivo fonte Java ou JavaScript.
- **Novo Bloco de Programação Java/JavaScript (High-Code):** cria um arquivo Java ou JavaScript, permitindo codificar sem o editor visual. Essa opção só está disponível no [Modo Avançado](#).

Após avançar, a segunda janela exibe os campos abaixo:

- **Nome do Arquivo / Classe:** nome dado a classe e a todos os arquivos gerados a partir do editor visual:
 - **low-code:** *.blockly, *.java e *.map (servidor); *.blockly e *.blockly.js (cliente);
 - **high-code:** *.java (servidor), *.blockly.js (cliente).
- **Nome da Função:** nome da primeira função / método.

Janela de edição

Após finalizar o passo da Figura 3.1 ou dar um duplo clique nos arquivos com extensão *.blockly ou *.blockly.js, será aberta a janela de edição dos blocos de programação. Na Figura 4.1 descrevemos suas partes.

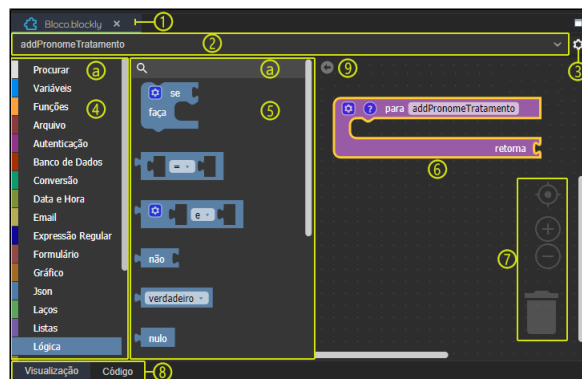


Figura 4.1 - Detalhes da janela do Bloco de programação

1. **Aba do editor visual:** a cor do ícone identifica o tipo de bloco - azul para servidor e verde para cliente. A extensão do arquivo será exibida apenas no [Modo Avançado](#).
2. **Índice de função:** alterna entre os métodos ou funções dessa classe. ao clicar na caixa de seleção, as funções desse arquivo serão listadas em ordem alfabética.
3. **Configurações do arquivo blockly:** define algumas configurações para todas as funções desse arquivo, veja mais detalhes [abaixo](#).
4. **Categorias:** lista das categorias de blocos de programação;
 - a. **Categoria Procurar:** essa categoria possui um campo de busca que filtra todos os blocos de todas as categorias presentes no editor.
5. **Seleção de blocos:** exibe todos os blocos de uma categoria;
 - a. **Campo de busca:** esse campo está presente na maioria das categorias e filtra por todos os blocos contidos na categoria selecionada.
6. **Função:** função (vazia) que espera receber os blocos.
7. **Ações (de cima para baixo):** centralizar a função, *zoom in*, *zoom out* e excluir um bloco de programação ao arrastá-lo para cima da lixeira (equivalente a apertar o botão delete).
8. **Exibe função:** exibe a função em modo visual ou o código fonte gerado. A subaba **Código** só está disponível no [Modo Avançado](#) ou ao habilitar a opção "Sempre mostrar código fonte" na janela de preferências do projeto, acessível a partir do menu do sistema Espaço de Trabalho > Preferências (veja o tópico "Espaço de Trabalho" na documentação [Tour pelo Cronapp](#)).
9. **Ocultar/Exibir Categorias:** por padrão, o ícone é exibido com a seta apontando para a esquerda, sinalizando que ao clicar fará com que as categorias de blocos de programação

fiquem ocultas. Quando estão ocultas, o sentido da seta muda, indicando que ao clicar, as categorias serão exibidas novamente.

Funções

Cada arquivo **blockly** pode conter diversas funções. Para isso, acesse o menu de categorias lateral **Funções** e arraste para o centro da janela uma das duas opções do bloco **Para** (Figura 4.2): **sem retorno** (primeira opção) ou **com retorno** (segunda opção).

Perceba, no destaque 1 da figura 4.2, que no menu lateral **Funções** é possível encontrar e arrastar os blocos das funções criadas nesse arquivo. Para alternar entre as funções do bloco, use a caixa de seleção no topo da janela (destaque 2). Ao clicar nessa caixa, as funções desse arquivo serão listadas em ordem alfabética, facilitando a navegação e seleção da função desejada.

Para chamar funções de outras classes, utilize os blocos **Chamar bloco servidor assíncrono**, **Chamar Bloco (com retorno)** e **Chamar Bloco (sem retorno)** no lado cliente ou **Chamar Bloco (com retorno)** e **Chamar bloco (sem retorno)** no lado servidor.

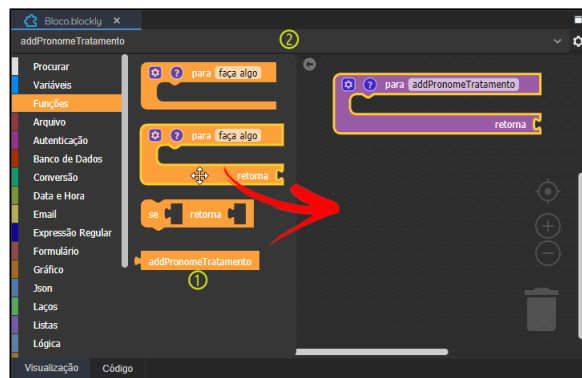


Figura 4.2 - Criando outra função dentro do bloco de programação

Estrutura das funções

Na Figura 4.3 vemos um pequeno exemplo de função (**AddPronomeTratamento**) que possui o seguinte algoritmo:

- Recebe o nome do usuário por parâmetro (nomeUsuario),
- Concatena em uma variável de nome **var** o termo "Sr.(a) " mais o nome do usuário recebido via parâmetro,
- Imprime no terminal o valor da variável **var**,
- O conteúdo da variável **var** alimenta o retorno da função.

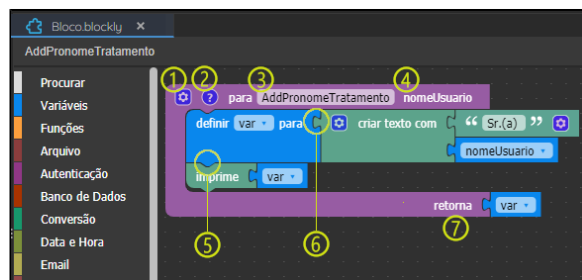


Figura 4.3 - Exemplo de função

1. **Parâmetros**: abre uma janela para adicionar ou remover parâmetros de entrada da função.
2. **Comentário**: adiciona um comentário para descrever a função. Veja mais detalhes no tópico [Comentários](#).
3. **Nome**: altera o nome da função de bloco de programação. É possível, porém, não recomendamos inserir nomes com espaços ou acentos. Acesse o tópico "Padrões para nomes" do nosso [Manual de Boas Práticas](#).
4. **Parâmetros**: lista de parâmetros dessa função de bloco de programação.

5. **Saliência abaixo ou interna em um bloco**: equivale a uma nova linha em uma programação textual.
6. **Abertura lateral**: recebe parâmetros, retorno de uma função ou atribuição através do encaixe de um bloco.
Encaixe lateral: passa parâmetros, retorno para uma função ou uma atribuição através do encaixe de um bloco.
7. **retorna**: área do retorno dessa função do bloco de programação.

Parâmetros

O ícone engrenagem (destaque 1 da figura 4.4) nos blocos de funções (**Para**) abre uma janela onde é possível arrastar para adicionar ou remover os parâmetros de entrada. Após adicionar os parâmetros à função, você os encontrará na categoria **Variáveis** do menu lateral (destaque 3).

Os blocos **nome de entrada** também possuem ícones de configuração (destaque 2), clique para abrir a janela que configura o tipo do parâmetro. Para mais detalhes, acesse [Variável](#).

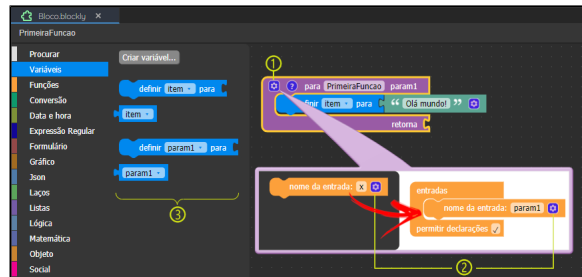


Figura 4.4 - Passando Parâmetro para uma função

Parâmetros e entidades

Ao passar uma entidade (objeto) para um bloco como parâmetro, ela é passada por referência e não por valor. O que significa que caso o parâmetro (objeto) seja alterado na função chamada, o objeto será modificado em todas as funções.

No exemplo abaixo, a função **ParametroReferencia** (Figura 4.6) alimenta uma entidade User na variável "user", imprime no console o campo *e-mail* desse objeto, em seguida, chama função **AlterarEmail** passando por parâmetro a variável "user" e, ao final, imprime novamente o mesmo campo *e-mail*, porém agora o conteúdo do campo está diferente pois foi alterado na função **AlterarEmail**.

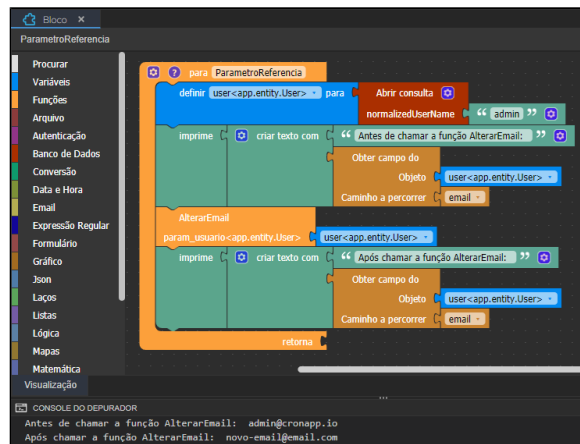


Figura 4.4.1 - Utilizando parâmetro com entidade para alterar um campo na entidade

Observação

Esse exemplo funciona apenas com entidades (objetos). Se for passado um texto ou número, será passado como valor e não como referência.

Propriedades do Blockly

Cada arquivo *.blockly ou *.blockly.js possui uma janela de propriedades que pode ser acessada através do ícone "engrenagem" no lado direito/superior da aba (seta das figuras 4.5 e 4.6). As configurações dessa janela irão afetar todas as funções contidas nesse arquivo.

Uma das propriedades mais importante dessas janelas é o campo **Formulário de Referência**, que permite que os blocos reconheçam os componentes visuais da página referenciada.

Veremos a diferença entre as propriedades dos blocos cliente e servidor.

Cliente

Janela com as propriedades do editor de bloco de programação cliente.

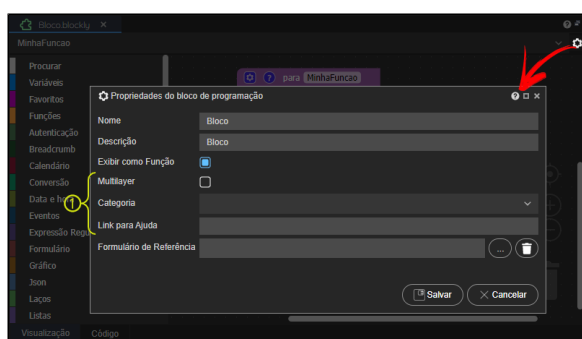


Figura 4.5 - Propriedades do bloco de programação cliente

- **Nome:** nome do arquivo *.blockly.js.
- **Descrição:** descrição do arquivo *.blockly.js.
- **Exibir como Função:** ao selecionar esta opção, todas as funções contidas neste arquivo *.blockly.js serão apresentadas na lista de categorias do editor de blocos desse projeto, juntamente com os blocos padrões do Cronapp. Os campos com destaque 1 da figura 4.5 só serão exibidos quando essa opção estiver marcada. Veja mais detalhes no tópico [Criar blocos low-code customizados](#).
- **Multilayer:** (disponível após habilitar a opção "Exibir como Função") ao selecionar esta opção, todas as funções contidas neste arquivo estarão disponíveis no editor de blocos, tanto no cliente (web e mobile) quanto no servidor. Desmarque-a para exibir apenas no editor de blocos do cliente (web e mobile).
- **Categoria:** (disponível após habilitar a opção "Exibir como Função") selecione a categoria em que os blocos serão exibidos ou digite o nome de uma nova categoria.
- **Link de Ajuda:** (disponível após habilitar a opção "Exibir como Função") informe um endereço com a documentação das funções de bloco.
- **Formulário de Referência:** vincula o *.blockly.js a uma página web ou tela mobile (view). O caminho da tela/página exibido nesse campo varia a depender de como a opção [Modo Avançado](#) esteja configurado.

Servidor

Janela com as propriedades do editor de bloco de programação servidor.

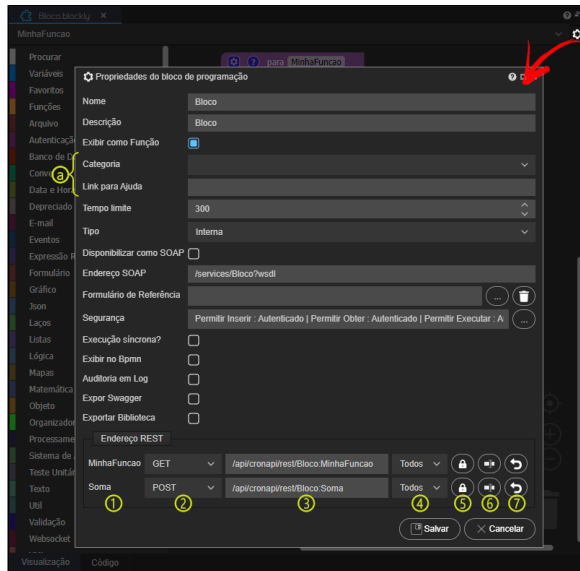


Figura 4.6 - Propriedades do bloco de programação Servidor

- **Nome:** nome do bloco.
- **Descrição:** descrição sobre o bloco.
- **Exibir como Função:** ao selecionar esta opção, todas as funções contidas neste arquivo *.blockly serão apresentadas na lista de categorias do editor de blocos servidor desse projeto, juntamente com os blocos padrões do Cronapp. Os campos com destaque "a" da figura 4.6 só serão exibidos quando essa opção estiver marcada. Veja mais detalhes no tópico [Criar blocos low-code customizados](#).
- **Categoria:** (disponível após habilitar a opção "Exibir como Função") selecione a categoria em que os blocos serão exibidos ou digite o nome de uma nova categoria.
- **Link de Ajuda:** (disponível após habilitar a opção "Exibir como Função") informe um endereço com a documentação das funções de bloco.
- **Tempo limite:** este parâmetro representa o limite, em segundos, para a conclusão da execução. Uma vez atingido esse limite, a ação é encerrada.

É importante observar que esse período também é influenciado pelo campo "Tempo limite para o balanceador de carga" nas configurações da aplicação dos [Serviços de Cloud](#). Assim, se os 2 campos estiverem configurados com tempos diferentes, a execução irá parar a partir do campo que possuir o menor tempo.

- **Tipo:** define como a estrutura de requisições REST será montada:
 - **Interna:** permite a personalização dos campos da área "Endereço REST"
 - **Externa:** mantém compatibilidade com projetos antigos e não permite alterações nos campos da área "Endereço REST".
- **Disponibilizar como SOAP:** gera uma API do tipo SOAP. Para saber mais, acesse [Disponibilizar Web Service SOAP](#).
- **Endereço SOAP:** endereço do WSDL do serviço SOAP.
- **Formulário de Referência:** vincula o *.blockly a uma página web ou tela mobile (view). O caminho da tela/página exibido nesse campo varia a depender de como a opção [Modo Avançado](#) esteja configurado.
- **Segurança:** abre outra janela que nos permite dar autorização de CRUD e Execução a diferentes [permissionáveis](#).
- **Execução síncrona?:** define se a execução das funções será de forma [síncrona](#) ou [assíncrona](#).
- **Exibir no Bpmn:** torna o bloco disponível para o plugin [Cronapp BPM](#). Essa opção é restrita para alguns planos.
- **Auditoria em Log:** gera histórico de chamada das funções desse bloco. Para saber mais, acesse [Log de Auditoria](#). Essa opção é restrita para alguns planos.
- **Expor Swagger:** exibe os endpoints gerados a partir das funções contidas no bloco de programação (arquivo blockly) na página do Swagger, veja mais detalhes em [Swagger - OpenAPI](#).
- **Exportar Biblioteca:** ao selecionar essa opção, será possível exportar todas as funções contidas no blockly [via biblioteca](#).
- **Endereço REST:** exibe e permite configurar as URIs do serviço REST de cada função dentro do blockly.

Colunas:

1. **Nome da função:** nomes das funções contidas no blockly.
2. **Verbs HTTP:** especifica os tipos de operações que podem ser realizadas em uma comunicação cliente-servidor usando o protocolo HTTP. Consulte a [documentação oficial](#) para mais detalhes de cada tipo. São eles:
 - GET.
 - HEAD.

- POST.
 - PUT.
 - PATCH.
 - DELETE.
 - OPTIONS.
 - TRACE.
3. **Endereço REST:** exibe as URIs do serviço REST de cada função dentro do blockly. Esse endereço será concatenado com o domínio do sistema e assim será obtido o endereço completo. Para saber mais, acesse [Disponibilizando Web Service REST](#).
4. **Media type:** especifica o [formato](#) dos dados enviados e recebidos em uma requisição. São baseados nos tipos disponíveis no Spring Framework, consulte a [documentação oficial](#) para mais informações de cada tipo. São eles:
- Atom.
 - CBOR.
 - Event Stream.
 - Form (URL Encoded).
 - HTML.
 - Imagem GIF.
 - Imagem JPEG.
 - Imagem PNG.
 - Json.
 - Markdown.
 - Multipart (Form).
 - Multipart (Mixed).
 - Multipart (Related).
 - NDJSON.
 - Octet-Stream.
 - Pdf.
 - Problem (JSON).
 - Problem (XML).
 - RSS (XML).
 - Stream (JSON).
 - Texto.
 - Todos.
 - XHTML.
 - XML.
5. **Segurança:** permite dar autorização de CRUD e execução aos permissionáveis do sistema. A segurança definida no arquivo blockly tem prioridade sobre a segurança de cada função.
6. **Configurar endereço REST:** permite personalizar o endereço REST da função. É necessário manter a rota inicial ao mudar o endereço, ficando `/api/cronapi/rest/<novo-valor>`. Exemplo: `/api/cronapi/rest/calculadora`.
- Sua estrutura deve seguir algumas convenções da tecnologia Spring, veja mais detalhes nas documentações oficiais ([documentação1](#) e [documentação2](#)).
- Caso a função necessite de parâmetros de entrada, é necessário configurá-los. Acesse o tópico "Parâmetros da função" da [Variável](#).
7. **Redefinir endereço REST:** redefine o endereço alterado pelo usuário no campo "Configurar Endereço REST".

Ordem de execução

Da mesma forma que ocorre com a programação textual, nas funções de bloco de programação a execução ocorre chamando os blocos em uma pilha de execução da esquerda para direita, retorna a pilha (destaque 1 da figura 4.7) e em seguida passa para a próxima linha (2 da figura 4.7).

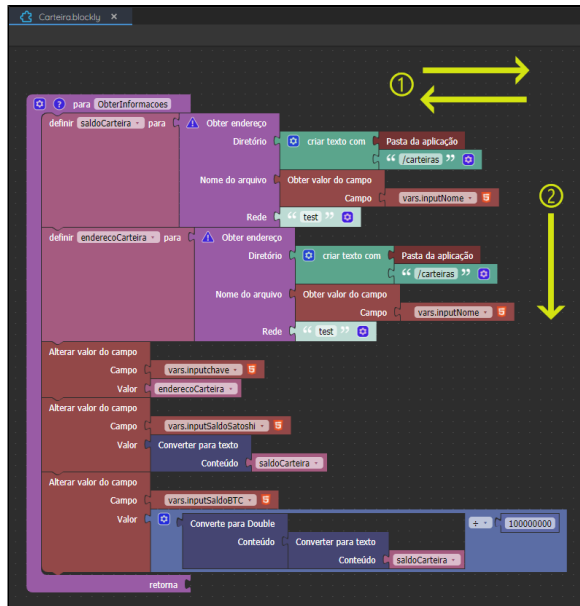


Figura 4.7 - Exemplo de função e como ocorre sua execução

Debug Visual

O processo de depuração (*debug* ou *debugging*) permite encontrar erros de sintaxe ou lógica no código fonte durante a execução da aplicação. Esse procedimento pode ser feito de forma tradicional (ver [documentação](#)) ou através do *Debug* visual, onde é possível definir os *breakpoints* diretamente no bloco que deseja analisar. Atualmente essa funcionalidade só está disponível para os blocos do tipo servidor.

Para que a depuração funcione, é necessário executar os passos abaixo:

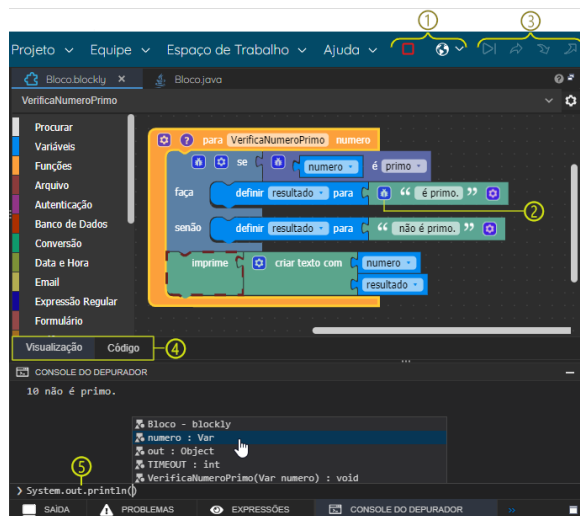


Figura 4.8 - Debug Visual em execução com a aba Console do depurador aberta

1. Clique no botão verde **Play** (não exibido na imagem) para executar o projeto em modo *Debug* e aguarde até exibir os botões **Stop** e **Abrir Navegador**.
2. Adicione uma *breakpoint* em algum bloco;
3. Assim que a aplicação executar a função que possui o *breakpoint*, utilize os botões de controle para executar bloco a bloco:
 - **Prosseguir execução**: executa a aplicação normalmente, ignorando os *breakpoints*;
 - **Próximo passo**: segue para a próxima linha do código, mas não entra no método se houver;
 - **Entrar**: segue para a próxima linha do código e entra no método;

- **Sair:** caso esteja dentro de método, clicando em sair, você segue para a primeira linha de código que fica fora do método.
4. Utilize as abas Visualização / Código para alterar entre a programação visual e o código;
 5. Use a aba inferior **Console do Depurador** para imprimir conteúdo, erros, exceções e inserir comandos:
 - Veremos mais abaixo como as demais abas também são úteis.

Breakpoints

Breakpoints são pontos de parada para verificar o estado atual da aplicação ou acompanhar o conteúdo de uma determinada variável.

Para adicionar um ponto de interrupção em um bloco, clique com o botão direito do mouse em cima do bloco e selecione a opção **Adicionar Ponto de Interrupção** (Figura 4.9), após isso você verá um ícone em formato de inseto (*bug*) representando um ponto de interrupção no bloco selecionado (destaque 2 da figura 4.8).

Para retirar o *breakpoint*, abra novamente o menu de contexto do bloco com o ícone *bug* e selecione a opção **Remover ponto de interrupção**.

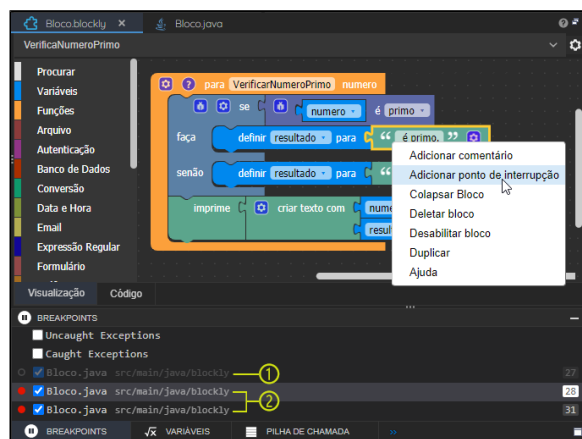


Figura 4.9 - Adicionando ponto de interrupção ao bloco

Selecione a aba inferior "**Breakpoint**" para exibir as funções e as linhas onde estão inseridos os pontos de parada:

1. Alguns blocos não permitem o *breakpoint*, nesse caso eles serão exibidos em uma cor escura e o círculo na sua frente não estará em vermelho.
2. Já os blocos com a tonalidade mais clara e que possuem um círculo vermelho na frente serão pausados, clique sobre ele para destacar o bloco que o representa.

Arquivo Java

Da mesma forma que o bloco de programação fica em destaque quando o código está pausado, no arquivo Java a linha analisada fica em amarelo. À medida que usamos os botões de controle (1 da figura 4.10), os próximos blocos ou linhas serão destacados.

Os pontos vermelhos antes do número da linha representam os *breakpoints*, assim como na aba inferior de mesmo nome (Figura 4.9).

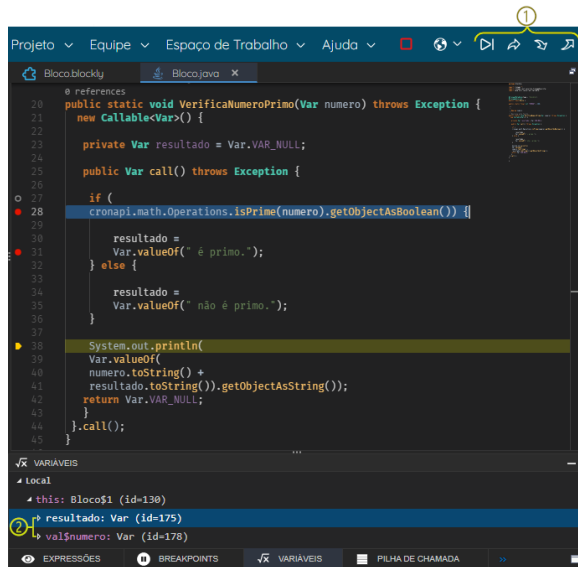


Figura 4.10 - O destaque em amarelo representa a linha atualmente em depuração

A figura 4.10 mostra a aba inferior **Variáveis** habilitada, nela são exibidas todas as variáveis da função em execução. Clique no ícone "triângulo" (destaque 2) antes do nome da variável para expandir suas informações.

Arquivo map

Ao alterar da aba **Visualização** para **Código** (destaque 1 da figura 4.11) será possível visualizar o código Java gerado pelo editor de blocos. Note que na maioria das linhas existe um comentário (em roxo) informando `sourceMappingStart` + código hash, ele representa o identificador do bloco, necessário para o mapeamento dos blocos durante o Debug visual.

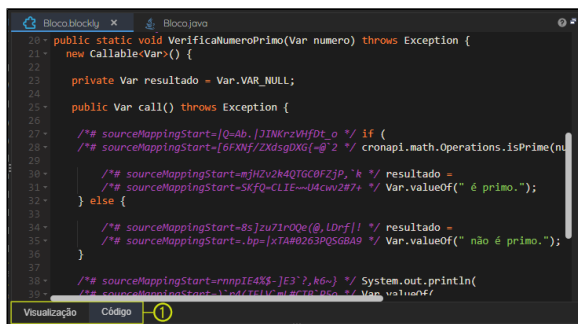


Figura 4.11 - Na aba Código (inferior) é possível visualizar o código Java e o ID dos blocos

Os identificadores dos blocos de programação são utilizados apenas para controle interno no Cronapp e não são adicionados no arquivo fonte **Java**, como podemos ver na figura 4.10, são exibidos apenas na aba **Código** (1 da figura 4.11) do editor de blocos. Os identificadores são referenciados no arquivo *.map, arquivo que é gerado automaticamente ao criar um arquivo *.blockly e possui uma lista de objetos que mapeia o ID do bloco com sua respectiva linha do código Java (Figura 4.12).

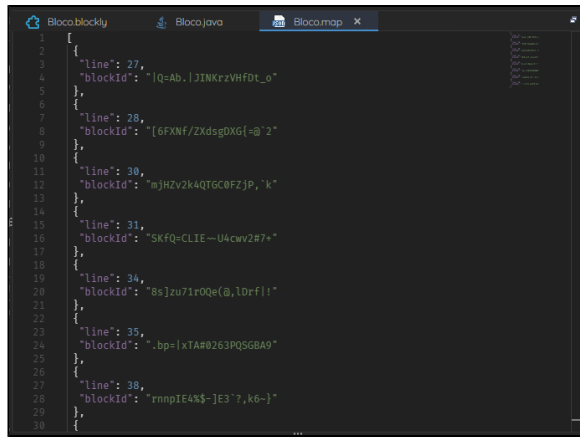


Figura 4.12 - Arquivo .map mapeia o Id dos blocos com o número da linha no código Java

Nomenclatura

Se você ainda estiver um pouco confuso com os termos "blocos" ou "função de blocos", tentaremos sanar nesse tópico.

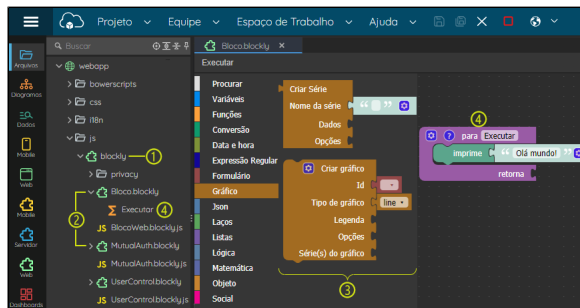


Figura 4.13 - Definição dos termos usados nos blocos de programação

1. **Diretório dos blocos de programação**: diretório onde são armazenados os blocos de programação;
2. **Arquivos de bloco de programação**: também chamado de "arquivos de bloco", "bloco" ou "blockly";
3. **Blocos**: cada uma das estruturas do menu lateral representa uma função, e como qualquer função, pode possuir parâmetros de entrada e conter ou não retorno. Quando usamos essas estruturas para montar funções de blocos de programação no editor de blockly, costumamos chamar apenas de "bloco";
4. **Função de bloco de programação**: bloco responsável por criar funções de bloco de programação. Essas funções também são listadas na árvore de arquivos (destaque 4 da figura 4.13) ao expandir os arquivos *.blockly (destaque 2).

Menu de contexto

A quantidade de itens exibido no menu de contexto irá variar com o tipo e o status atual do bloco selecionado. Mostraremos abaixo todas as opções disponíveis.

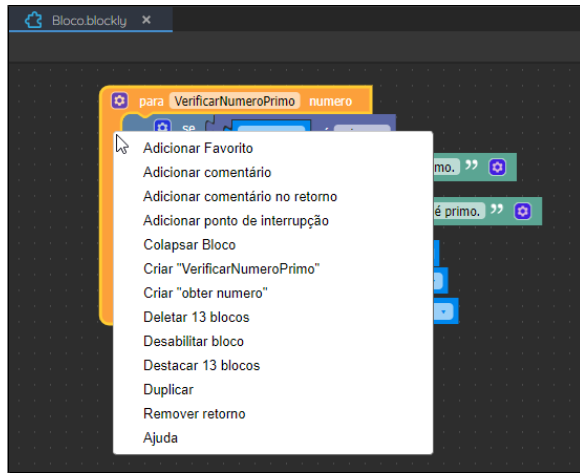


Figura 4.14 - Menu de contexto do bloco de programação

Favoritos

Quando um bloco é adicionado aos favoritos, ele aparecerá na categoria **Favoritos** (destaque 1 da figura abaixo) com uma estrela amarela ao lado, gerando uma lista com os blocos mais utilizados durante o desenvolvimento. Para remover o bloco dessa categoria, basta acessar o menu de contexto do bloco (dentro ou fora da categoria **Favoritos**) e selecionar a opção "Remover Favoritos" (figura 4.15). Os blocos favoritados são vinculados ao *workspace* do usuário, logo, a lista de blocos favoritos será igual para todos os projetos do mesmo usuário.

Os blocos Cliente (web e mobile) e Servidor são diferentes, ou seja, um bloco adicionado aos favoritos do tipo Cliente, não irá aparecer no Servidor e vice-versa.

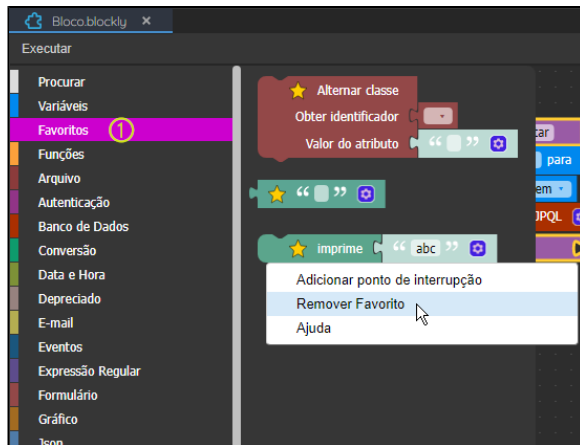


Figura 4.15 - Remoção de um bloco da categoria Favoritos

Comentários

Adiciona o ícone "?" no bloco de programação, ao clicar, um balão é exibido para inserir um comentário sobre o bloco no código fonte.

Para adicionar comentários visíveis a partir da função de bloco, utilize o bloco **Comentário** (Servidor e Cliente).

A figura 4.16 apresenta os locais onde é possível adicionar comentários, já a figura 4.17 aponta onde esses comentários são refletidos no código gerado pela função. Esse recurso funciona de forma similar para os blocos **clientes** (JavaScript) e **servidor** (Java).

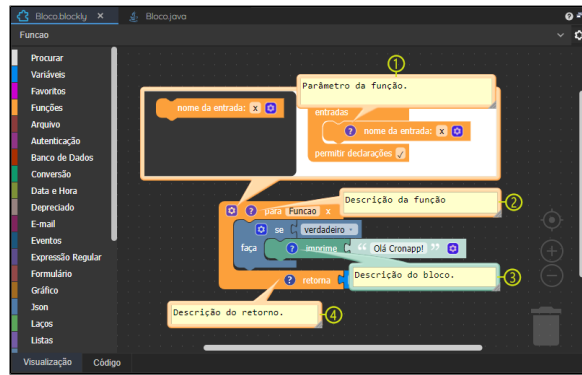


Figura 4.16 - Caixas de comentários dos blocos de programação

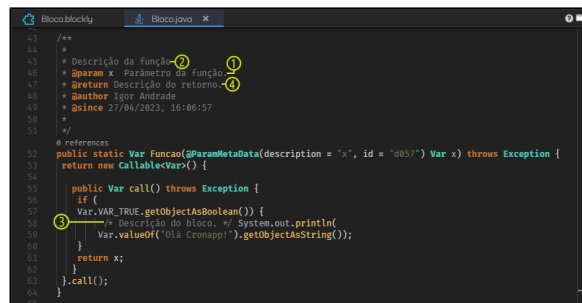


Figura 4.17 - Locais no código onde são exibidos os comentários

1. **Parâmetro da função:** permite descrever o que será passado como argumento em cada parâmetro da função. Caso a função esteja com a propriedade **Exibir como Função** habilitada, esse comentário será usado para renomear e descrever o parâmetro. Utilize o caractere dois pontos ":" para separar o nome e a descrição.
Ex: "Novo nome do param: Descrição do parâmetro."
2. **Descrição da função:** comentário para descrever o objetivo da função. Caso a função esteja com a propriedade **Exibir como Função** habilitada, esse comentário será usado para renomear o bloco e descrever a função. Utilize o caractere dois pontos ":" para separar o nome e a descrição.
Ex: "Novo nome do bloco: Descrição do bloco."
3. **Blocos dentro da função:** permite comentar sobre um bloco em específico dentro da função ou um grupo de blocos.
4. **Retorno da função:** descrição sobre o que será retornado pela função. Esse recurso só estará disponível para funções com retorno.

Ponto de interrupção

As opções **Adicionar** e **Remover ponto de interrupção** permite incluir ou retirar **breakpoints** para utilizar o **Debug visual**, o ícone **bug** será exibido sobre o bloco selecionado.

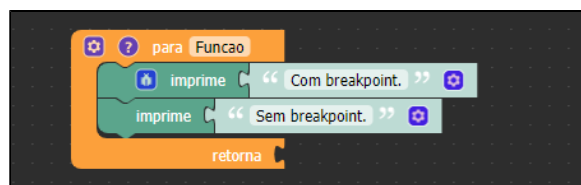


Figura 4.18 - Blocos com e sem breakpoint

Colapsar e expandir

Permite comprimir o bloco selecionado e todos os blocos acoplados a ele. Após colapsar, acesse novamente o menu de contexto do bloco comprimido para exibir a opção **Expandir bloco**. Essa opção também é acionada ao dar um duplo clique sobre bloco.

O colapsar de blocos não interfere em nada no algoritmo, apenas reduz o tamanho dos blocos na função, melhorando a sua legibilidade.

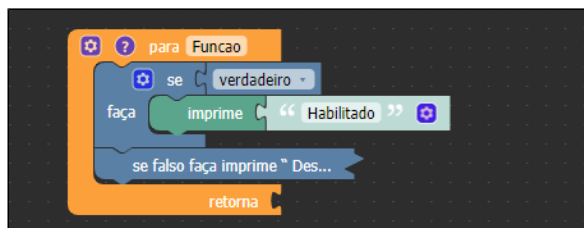


Figura 4.19 - A função possui 2 blocos "se", um colapsado e o outro não

Criar bloco

Essa opção está disponível nos menus de contextos dos blocos de **função** ou blocos de **variáveis** (acesse o tópico "Obter bloco complementar" em [Variável](#)).

Nos blocos de função é possível criar um bloco que executa o próprio bloco ou criar um bloco de variável com o parâmetro da função. A chamada da função também pode ser obtida na categoria **Funções**, já as variáveis de parâmetros também podem ser obtidas a partir da categoria **Variáveis** (destaque 4 da figura 4.1).

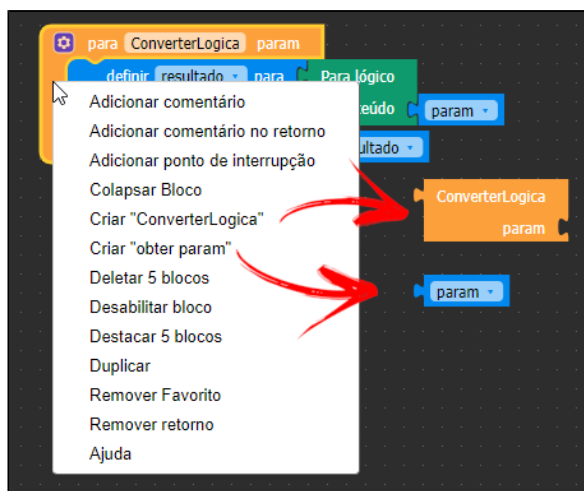


Figura 4.20 - Gerando blocos de chamada da função e dos parâmetros da função

Deletar

Deleta o bloco selecionado e todos os blocos acoplados a ele. Se existir blocos acoplados, o menu de contexto exibirá a quantidade de blocos que serão excluídos.

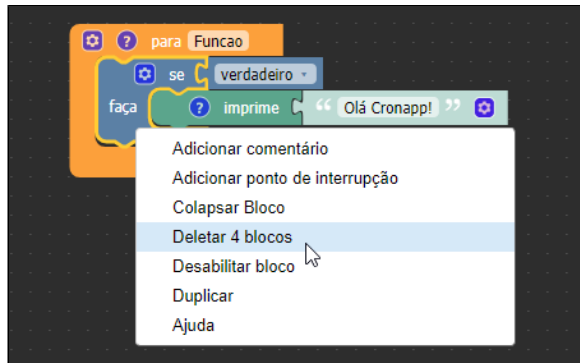


Figura 4.21 - A opção Deletar apaga todos os blocos acoplados a ele

Habilitar e desabilitar

Blocos desabilitados não são executados ao rodar a aplicação. Ficam disponíveis, porém desabilitados no arquivo .blockly e não são gerados no código fonte.

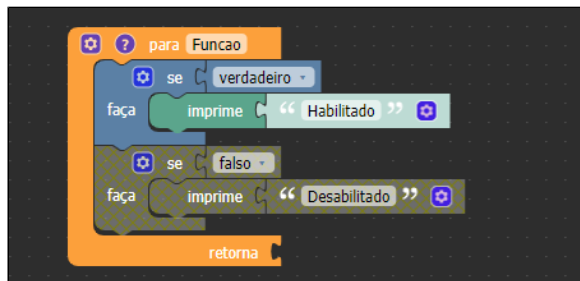


Figura 4.22 - O segundo bloco "se" não será visualizado no código fonte

Destacar

Essa opção desenha o bloco selecionado e todos os blocos acoplados a ele. Equivalente a pressionar a tecla CTRL, clicar sobre um bloco e arrastar.

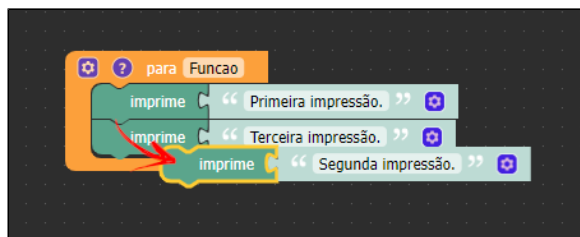


Figura 4.23 - Retira o bloco selecionado da sequência

Duplicar

Faz uma cópia do bloco selecionado e todos os blocos acoplados a ele.

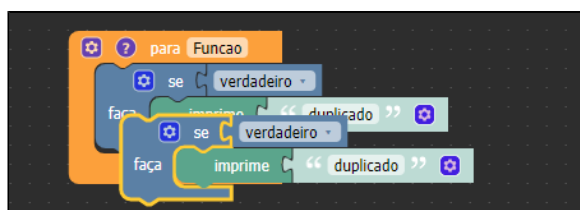


Figura 4.24 - Duplicação do bloco "se"

Adicionar ou Remover retorno

Essa opção está disponível para os blocos de função, com ou sem retorno. Ela permite que, após a criação de uma função, seja possível adicionar ou remover o encaixe de retorno da função. Na figura 4.25, a função **Executar** foi criada com retorno e, após clicar na opção **Remover retorno**, do menu de contexto, o retorno da função é removido (Figura 4.26) e o bloco de chamada na categoria **Funções** também é alterado (destaque 1 da Figura 4.26).

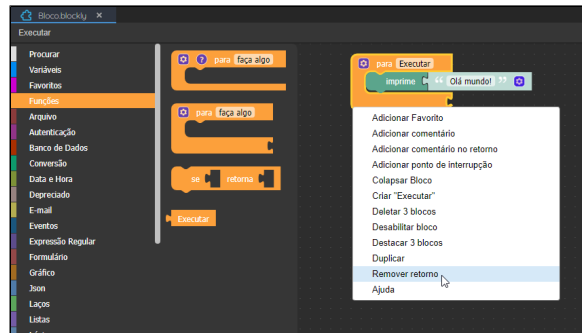


Figura 4.25 - Bloco função com retorno

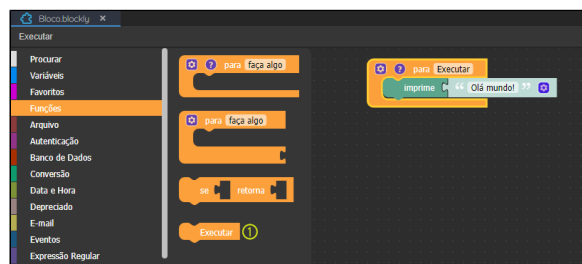


Figura 4.26 - Bloco função sem retorno

Entradas externas x incorporada

Essa opção permite alterar o formato do bloco de programação, possibilitando melhorar a legibilidade da função. A opção **Entrada externa** exibe seus parâmetros na vertical, enquanto a opção **Incorporada** exibe os parâmetros na horizontal.

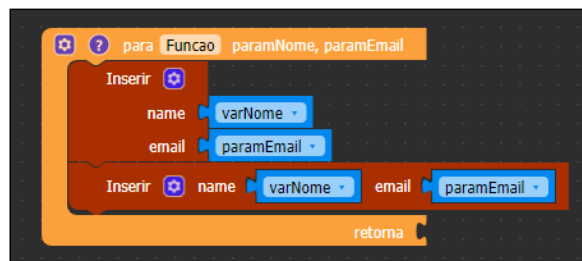


Figura 4.27 - Os 2 blocos "inserir" possuem o mesmo comportamento

Retorno

Essa opção só está disponível para alguns blocos que possuem retorno. Ela permite executar a função e ignorar o seu retorno. Na figura abaixo, os 2 blocos **Imprimir** realizam a mesma ação, porém o segundo não retorna nada.

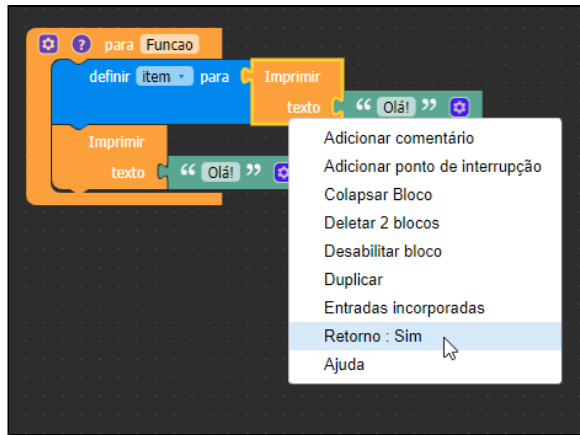


Figura 4.28 - O menu de contexto informa se o bloco exibe ou não o retorno

Ajuda

Abre uma nova guia no seu navegador com a documentação do bloco selecionado.

Desfazer e Refazer

Essas opções somente serão exibidas ao clicar na área de trabalho do editor de bloco (fundo preto pontilhado). Elas equivalem a utilizar os botões CTRL + Z e CTRL + Y, desfazendo uma ação e refazendo em seguida.

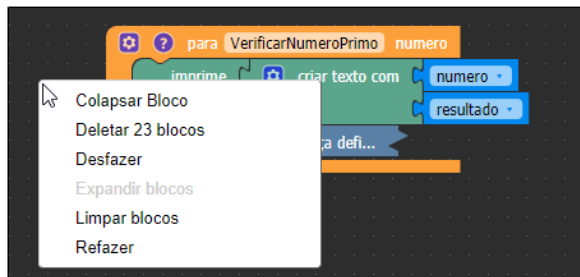


Figura 4.29 - Opções Desfazer e Refazer

Limpar blocos

Essa opção só será exibida ao clicar na área de trabalho do editor de bloco (fundo preto pontilhado). Ao selecionar, todos os blocos que estiverem desacoplados fora da função serão alinhados.

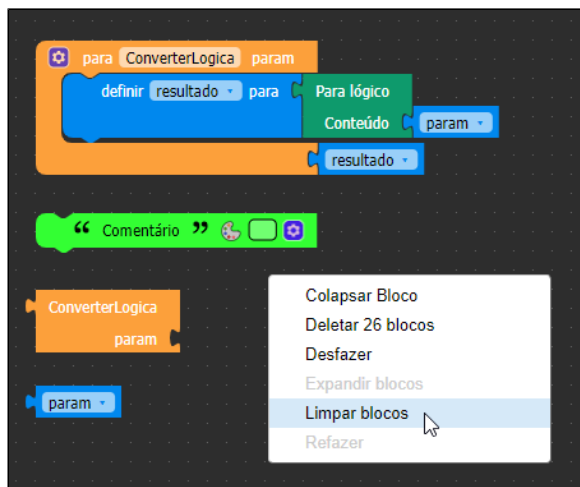


Figura 4.30 - Opção para organizar blocos espalhados

Informações

Neste tópico mostraremos algumas dicas e informações para ajudar em sua produtividade enquanto trabalha no editor de blocos.

Ajuda

Para obter uma breve descrição do bloco antes de selecioná-lo, repouse o cursor do mouse sobre ele para estender uma aba com as seguintes informações:

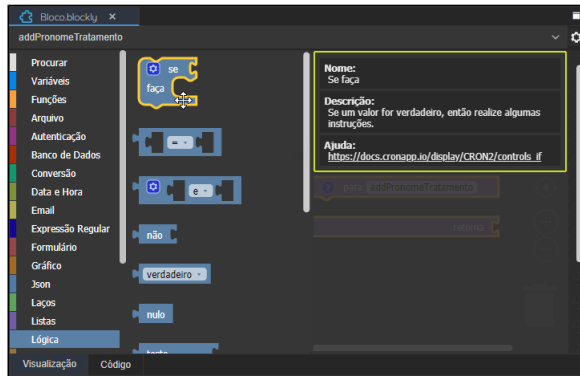


Figura 5.1 - Menu estendido com a descrição do bloco selecionado

- **Nome:**
- **Descrição:** breve descrição do bloco selecionado;
- **Ajuda:** link para a documentação do bloco.

Deixe o cursor do mouse parado sobre um bloco na área de edição para exibir sua descrição em um *tooltip*.

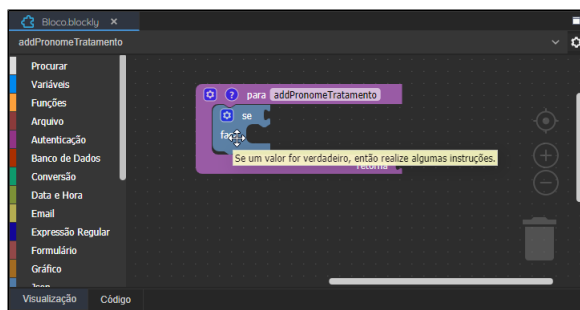


Figura 5.2 - Tooltip de descrição do bloco

Ícone engrenagem

Diversos blocos possuem esse ícone, ao clicar, uma janela é aberta para permitir realizar alguma configuração no bloco selecionado. O conteúdo da janela varia com o tipo de bloco selecionado.

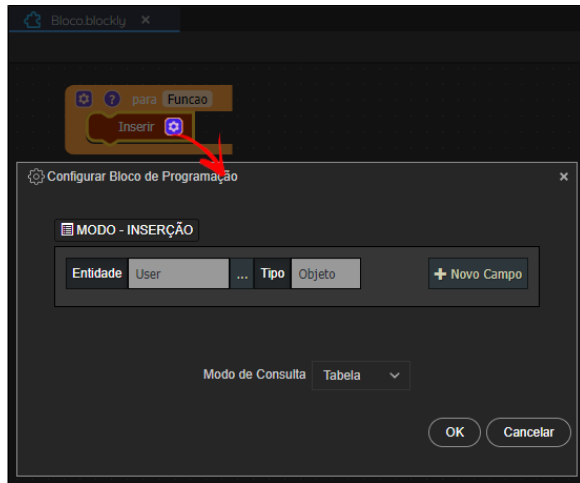


Figura 5.3 - Ícone engrenagem do bloco Inserir (Banco de dados)

Desacoplamento de blocos

Se uma função possui uma sequência de blocos empilhados e você arrasta um dos blocos do meio, todos os blocos abaixo do selecionado serão arrastados juntos. Porém, ao realizar a mesma ação com o botão CTRL ou ALT pressionado, somente o bloco selecionado será desacoplado.

Na primeira parte da figura abaixo (direita), o bloco **inserir** foi arrastado e o bloco **Imprimir** foi removido junto, já no lado esquerdo da mesma figura, o bloco **Inserir** foi arrastado com o botão CTRL pressionado, e somente ele foi desacoplado.

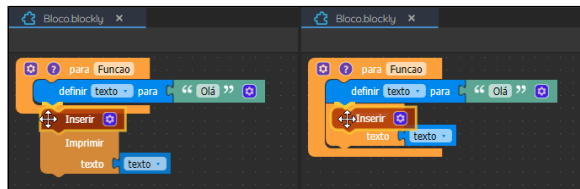


Figura 5.4 - Desacoplamento de blocos intermediários

Outros parâmetros

Alguns blocos possuem parâmetros com características diferentes da maioria dos blocos, veremos aqui algumas delas.

Comandos

Blocos com esse tipo de parâmetro permitem adicionar trechos de blocos (códigos) para serem executados junto com a função chamada.

Na figura abaixo vemos que primeiro parâmetro (destaque 1 da figura 5.5) do bloco **Agendar Execução** permite adicionar uma sequência de outros blocos.

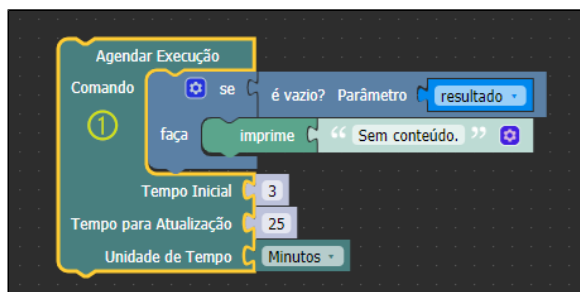


Figura 5.5 - Parâmetro de comandos

Caixa de seleção

Alguns blocos possuem opções limitadas para um parâmetro, nesses casos é exibido uma caixa de seleção com as opções possíveis.

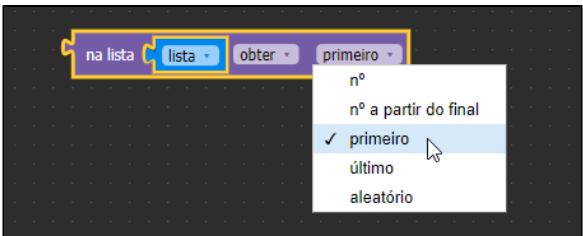


Figura 5.6 - Parâmetro com limite de opções

Gerador de documentação

O Cronapp possui uma funcionalidade para [Gerar documentação](#) dos recursos desenvolvidos na plataforma, dentre elas, os blocos de programações Servidor e Cliente. Essas documentações dos blocos são gerados a partir de *tags* nos comentários adicionadas no código fonte dos blocos de programação. Por padrão, você não precisa se preocupar com as *tags* se estiver desenvolvendo em **Low code**, pois são geradas automaticamente, mas caso precise, ao ativar o [Modo Avançado](#), poderá também incluir as [tags do javadoc](#) (para Servidor) e [jsdoc](#) (para cliente) no modo **High code**. Abaixo (Destaque da figura 5.7), segue um exemplo da inserção de uma *tag* no código de um bloco servidor.

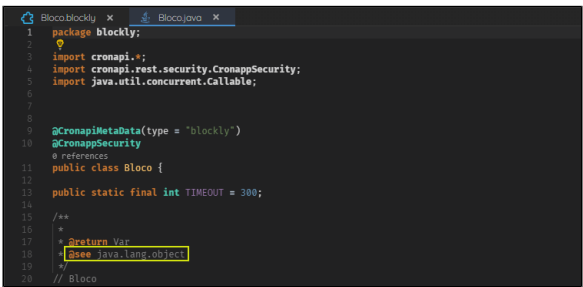


Figura 5.7- Inserindo tag javadoc na doc do bloco Servidor

Exemplo [tags Servidor](#):

Exemplo Tags Servidor
@see @throws

Exemplo [tags Cliente](#):

Exemplo Tags Servidor
@see @global @implements

Blocos assíncronos

A partir da **versão 2.6** do Cronapp, os blocos de programação clientes (*web* e *mobile*) funcionam de forma **assíncrona**. Se o seu projeto foi criado após essa versão, não é necessário realizar nenhuma ação. Porém, projetos criados antes dessa versão e migrados posteriormente, precisam atualizar os arquivos de blocos de programação.

Como atualizar

Para atualizar, basta abrir os arquivos de blocos cliente, realizar qualquer modificação (arrastar o bloco função, por exemplo) e salvar. Após isso, o Cronapp converterá todas as funções contidas nessa classe em assíncrona.

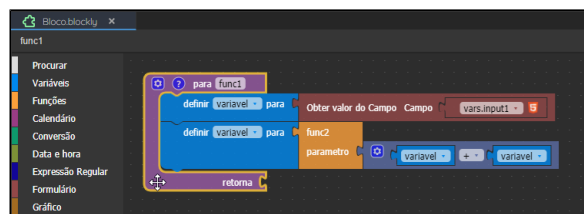


Figura 6.1 - Arraste um bloco e salve para atualizar todas as funções do arquivo

Esse procedimento desse ser realizado nos blocos cliente (*web* e *mobile*).

Verificar atualização

As alterações só serão percebidas através do código JavaScript (necessário habilitar o **Modo Avançado**). Por isso, acesse a aba inferior **Código** (destaque 1 da figura abaixo) e verifique que antes de cada função criada teremos agora a palavra-chave **async** (destaque 2), já a palavra-chave **await** (3) será usada na chamada de outras funções.

```
1  /**  
2   * Função 1  
3   */  
4  window.blockly.js.blockly.Bloco.func1 = async function() {  
5    var item, parametro, variavel;  
6    variavel = this.cronapi.screen.getValueOffield("vars.input1");  
7    variavel = await this.blockly.js.blockly.Bloco.func2(variavel + variavel);  
8  }  
9  /**  
10   * Função 2  
11   */  
12  window.blockly.js.blockly.Bloco.func2 = async function(parametro) {  
13    var item, variavel;  
14    this.cronapi.screen.changeValueOffield("vars.input2", parametro);  
15  }  
16  Visualização  
17  Código
```

Figura 6.2 - Atualizações das funções assíncronas

Criar blocos low-code customizados

O editor de blocos permite criar funções para gerar blocos de programação acessíveis na lista de categorias em qualquer arquivo blockly dentro do projeto, juntamente com os blocos padrões do Cronapp. Esse recurso pode substituir o uso do bloco **Chamar bloco**, com a vantagem de permitir criar uma categoria personalizada com todas as suas funções convertidas em blocos, apresentando de forma amigável os nomes dos blocos, parâmetros, descrições e a possibilidade de linkar com uma documentação própria.

Esse recurso é útil para a criação de blocos de programação simples, para blocos mais complexos, que necessitam de *callback* ou parâmetros personalizados, ainda será necessário seu desenvolvimento de forma high-code. Veja mais detalhes nas documentações abaixo:

- [Criando blocos servidor customizados](#)
- [Criando blocos cliente customizados \(mobile e web\)](#)

O exemplo da figura 7 apresenta uma função cliente (Javascript) de bloco com o nome "Div" que possui 2 parâmetros: "num1" e "num2".

Utilizamos os comentários dos parâmetros (destaque 1 da figura 7) e da função (2) para renomear e descrever esses elementos. Isso pode ser feito utilizando o caractere dois pontos ":" para separar o nome do bloco ou parâmetro e a descrição do bloco ou parâmetro. Com isso, é possível criar nomes mais legíveis com acentos e espaços. Caso não utilize os comentários, o bloco será criado com os nomes originais.

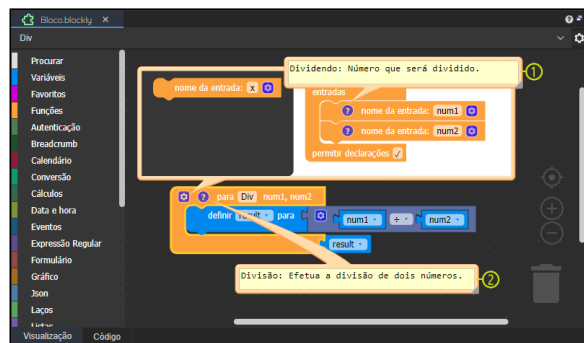


Figura 7 - Função e parâmetros com seus comentários

Acesse a janela de **Propriedades do bloco de programação** (seta da figura 7.1) para habilitar a opção que irá exibir todas as funções criadas nesse arquivo na lista de categorias do editor Blockly.

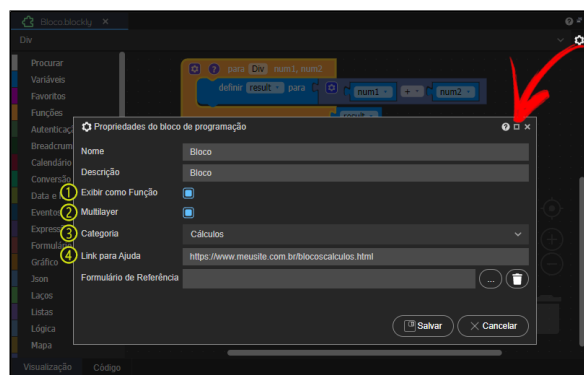


Figura 7.1 - Habilitando a opção de exibir as funções como blocos

1. **Exibir como Função**: marque essa opção para exibir os campos com destaque 2, 3 e 4 da figura 7.1.
2. **Multilayer**: (exclusivo das propriedades dos blocos de programação cliente) ao marcar, todas as funções contidas neste arquivo estarão disponíveis no editor de blocos cliente (web e mobile) e servidor.
3. **Categoria**: selecione uma das categorias de blocos já existentes ou digite o nome de uma nova categoria.
4. **Link de Ajuda**: informe um endereço com a documentação de suas funções de bloco. O endereço será o mesmo para todos os blocos de programação desse arquivo.

A figura 7.2 apresenta um editor de bloco de programação servidor listando uma categoria contendo um bloco criado a partir do editor de bloco cliente.

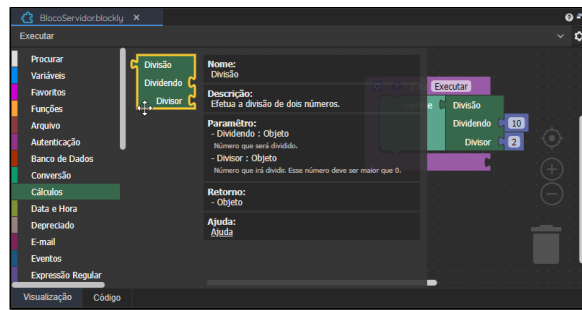


Figura 7.2 - Nova categoria e blocos de programação