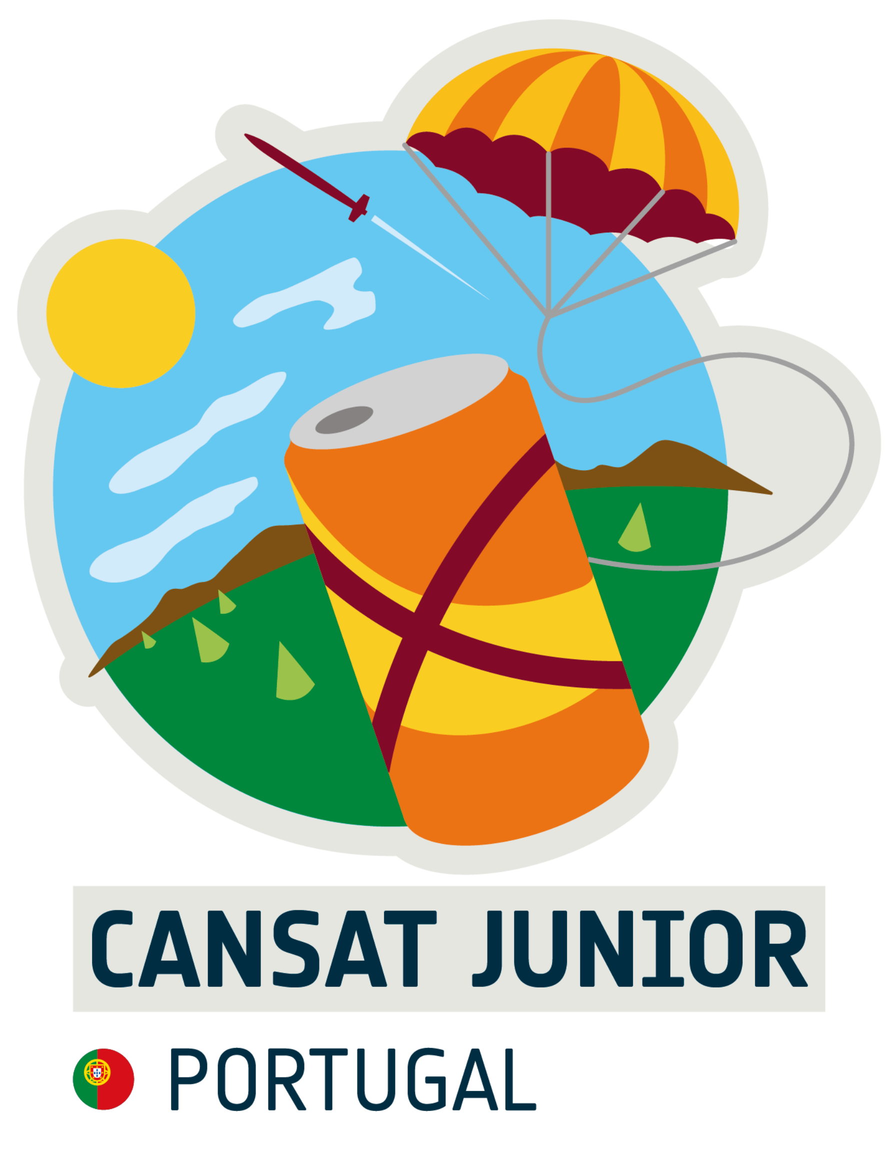


Manual do Kit CanSat Júnior

v3.3 (2024-05-11)

Índice

Introdução **1**

Componentes do Kit 1

Montagem do Kit **3**

Paraquedas 5

Dimensionamento do Paraquedas 5

Antena Yagi-Uda 5

PCB “Arduino” 7

PCB Rádio 7

PCB Sensores 7

Programação do CanSat **8**

Limitações do Scratch 11

Conselhos 11

Pré-requisitos 12

Procedimentos 12

Exportar Programa em Scratch para C/C++ 12

Programar Arduino a partir de Código C/C++ 12

Ver saída do Arduino 15

Exemplos de Programas 16

Estação Base **17**

Anexo A: Introdução a Programação com Scratch **18**

Expressões e Tipos 19

Constantes 19

Aritmética 20

Lógica 20

Comparação 20

Operadores Lógicos Binários 20

Negação 21

Outros tipos de expressões 21

Variáveis 21

Estruturas de Controlo 22

Se-Faça-Senão (if-then-else) 22

Ciclos 22
Ciclos “repita n vezes” (repeat-n) 22
Ciclos “repita enquanto” (while) 22
Ciclos “contar com” (for) 22
Escape e Continuação Abrupta de Ciclos (break e continue) 23
Funções e Procedimentos 23
Anexo B: Entrada/Saída do Kit no Scratch **25**
Escreve no ecrã (Serial) 26
Rádio 26
digitalWrite 28
LED do Arduino 28
Tempo ativo 28
Sensor de Temperatura DS18B20 28
Sensor de Pressão BMP180 29
Sensor de Temperatura e Humidade DHT11 30
Receptor GPS 30
Glossário **31**

Introdução

O Kit do CanSat Júnior contém todo o material necessário para construir um micro-satélite, um paraquedas e uma estação base (fora o computador). Foi desenhado para poder ser usado sem qualquer experiência prévia em informática e eletrónica, e deixar alunos e professores confortáveis com sensores, microcontroladores e comunicação por rádio.

Este manual usa terminologia técnica que pode não ser familiar para todos. Aconselha-se a consulta do Glossário.

Componentes do Kit

Para a construção do paraquedas, são fornecidos:

- 40x40 (cm) de tecido
- 1 gancho com desenrolador
- 3m de cordelete

Para a construção da estação base:

- 1x PCB Arduino
    - 1x Arduino Nano
- 1x PCB Rádio (inclui *transceiver* RFM69HCW já soldado), com cabo terminado em ficha SMA fêmea (1m)
    - 1x *Level Shifter* (TXB0104)
- 2x Calha 0.5m
- 1x Fita métrica (1.5m)
- 1x Cabo terminado em ficha SMA macho (1m)
- 3x barras de 2x6 pinos longas

Para a construção do CanSat:

- 1x PCB "Arduino"
    - 1x Arduino Nano
- 1x PCB Rádio (inclui *transceiver* RFM69HCW já soldado), com antena fio monopolo
    - 1x *Level Shifter* (TXB0104)
- 1x PCB Sensores
    - 1x Sensor de temperatura (DS18B20)
    - 1x Sensor de pressão (BMP180)

    - 1x *Buzzer* ativo
- 6x barras de 2x6 pinos longos
- 1x Conector de pilha 9V com master switch
- 2x Pilhas 9V
- 1x Placa de topo
- 1x Placa de fundo
- 4x Varões roscados M3
- 50x Porcas M3
- 50x Anilhas M3
- 1x Parafuso com Anel M6
- 1x Porca M6
- 2x Anilhas M6
- 1x Invólucro de CanSat dividido em duas peças impressas em 3D

Montagem do Kit

O CanSat é constituído pelas placas de Arduino, rádio e sensores, em conjunto com a sua estrutura (varões roscados e placas de topo/base), paraquedas, e invólucro. É importante usar a placa rádio com a antena fio monopolo, não a que tem um cabo comprido (1 m) com conector SMA.

A estação base é constituída pelas placas de Arduino e rádio. É importante usar a placa rádio com cabo terminado em conector SMA fêmea (1m), não a que tem um pequeno fio soldado (antena monopolo). A placa rádio deverá ser ligada à antena pelas fichas SMA presentes nas extremidades dos cabos das duas, e a placa Arduino deverá ser ligada a um computador com recurso a um cabo USB. Este cabo USB é usado normalmente para ligar telemóveis ao carregador ou ao computador e é fornecido no kit.

A ligação entre as PCBs “Arduino”, rádio e sensores não tem ordem. Basta alinhar os 3 conectores pretos da que escolherem ficar em baixo com os 3 conjuntos de pinos metálicos da que escolherem ficar em cima e juntar. Para garantir **separação suficiente entre as placas**, deverá **usar as barras de pinos extras** fornecidas no kit.

Sugestão: As placas precisam de alguma força para encaixar/desencaixar. Não as encaixe completamente antes de testar que ficou tudo corretamente montado.

CUIDADO: Não esmagar o sensor de temperatura da placa de sensores. Podem dobrar com cuidado os pinos do sensor para mudar a sua orientação, ou escolher ter a placa de sensores em cima

CUIDADO: Ligar componentes/placas na orientação correta. Quando ligado de forma incorreta, um componente pode aquecer e eventualmente queimar. Caso seja detetado que um componente seja montado na orientação errada, deverá desligar o CanSat ou a estação base e aguardar entre 2 a 5 minutos para garantir que o componente teve tempo para arrefecer.

CUIDADO: Não encaixar duas PCBs do mesmo tipo. O circuito não foi desenhado para isso.

CUIDADO: A alimentação do CanSat deve ser feita através do cabo USB que ligue o Arduino a um computador/fonte ou de uma pilha de 9V (garantir que colocação correta do conector da pilha).

CUIDADO: Alguns cabos USB só transportam energia. Estes não funcionarão para programar o Arduino nem para obter dados do Arduino da estação base. Teste o seu cabo verificando que, usando-o para ligar o seu telemóvel ou o Arduino ao computador, o computador deteta o dispositivo.

Paraquedas

O material fornecido destina-se à criação de um paraquedas flat-shape, que deverá conferir uma velocidade terminal ao CanSat entre 8 e 11 m/s.

O paraquedas é preso ao parafuso na parte superior do CanSat pelo gancho com destorcedor fornecido no kit. A este destorcedor devem ser presos fios (de comprimento igual) que o prendem ao tecido do paraquedas (pelas pontas).

Dimensionamento do Paraquedas

Para o dimensionar, deverá ser analisada a situação de equilíbrio quando o satélite adquire a velocidade terminal. Aí, a força de atrito e da gravidade anulam-se: $F_{atrito} = F_G$

$$F_{atrito} = \frac{1}{2} * \rho * k * A * v^2$$

$$F_G = m * g$$

onde:

ρ - densidade do ar

k - coeficiente de atrito

A - área do paraquedas (exclui chaminé ou outros orifícios)

v - velocidade (terminal)

m - massa do satélite

De notar que o coeficiente de atrito é obtido experimentalmente, e depende da forma do paraquedas, do tecido utilizado e dos fios utilizados (tanto do fio, como da sua disposição). Para um paraquedas circular, é habitual observar valores entre 0.75 e 0.80 para este coeficiente.

Antena Yagi-Uda

Uma antena Yagi-Uda é constituída por n elementos perpendiculares a um eixo central. Os vários elementos têm objetivos diferentes:

- O dipolo transmite o sinal. É constituído por dois condutores (isolados um do outro), ligados ao sinal e massa (*ground*) do conector da antena. Existe sempre pelo menos um.
- O refletor, que se coloca atrás do dipolo, age como uma parede que reflete o sinal de volta para o dipolo (e aumenta a sua intensidade). Está isolado dos restantes elementos.
- Os diretores, que se colocam depois do dipolo, funcionam como um funil para o sinal, também contribuindo para o ganho da antena. Estão também isolados dos restantes elementos.

Os refletores e diretores são também chamados de elementos parasitas por não estarem ligados ao sinal.

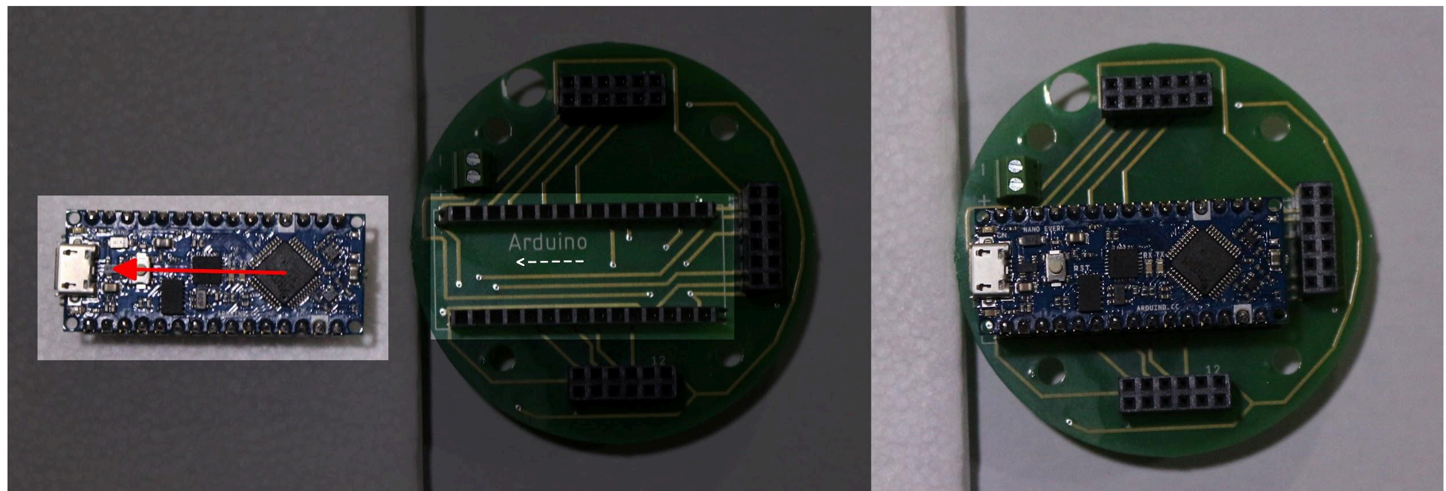
A teoria por trás destas antenas permite muitas configurações diferentes. A que recomendamos usar tem 1 refletor, 1 dipolo e vários diretores (pelo menos 1, quantos mais melhor). Para saber as dimensões de cada elemento e entre elementos recomenda-se o uso de uma calculadora como a disponível em: https://www.changpuak.ch/electronics/yagi_uda_antenna_DL6WU.php

De notar que o tamanho total do dipolo deve ter entre 0,498 e 0,5 vezes o comprimento de onda.

A antena pode ser construída com a calha, fita métrica, e o cabo com terminação SMA macho fornecidos. Caso tenham oportunidade, podem fazer os elementos ligeiramente maiores do que o necessário e afinar a vossa antena com a ajuda de alguém com equipamento especializado.

PCB “Arduino”

1. Caso esteja a montar a PCB Arduino do CanSat, garanta que usa a que tem o bloco de terminais para ligar os conectores da bateria. Caso esteja a montar a da estação base, garanta que o bloco de terminais **não** está presente.
2. Encaixar Arduino Nano na PCB “Arduino” orientando-a de modo a que o conector USB fique acessível do exterior.



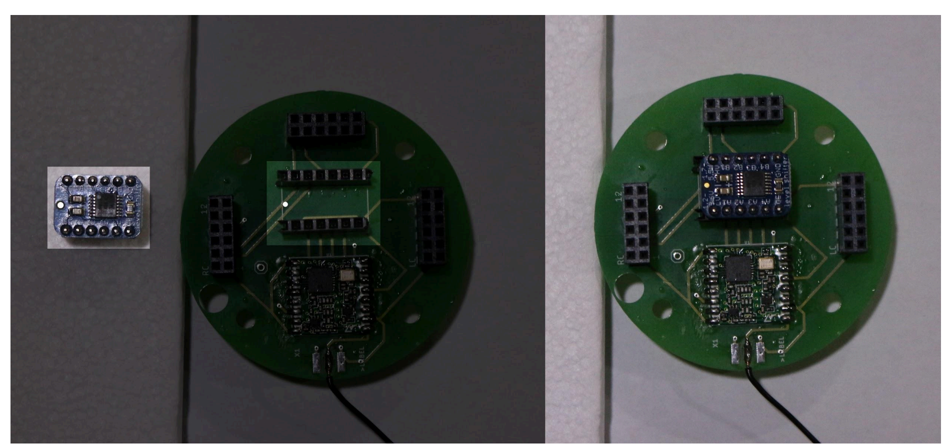
3. **Caso se trate da placa para o CanSat em si,** o conector da bateria deverá ser ligado ao bloco de terminais verdes. O fio vermelho corresponde ao pólo positivo (+), e o preto ao negativo (-). Os pólos estão indicados à frente do conector em tinta branca.

PCB Rádio

O rádio (RFM69HCW) já vem soldado a esta placa.

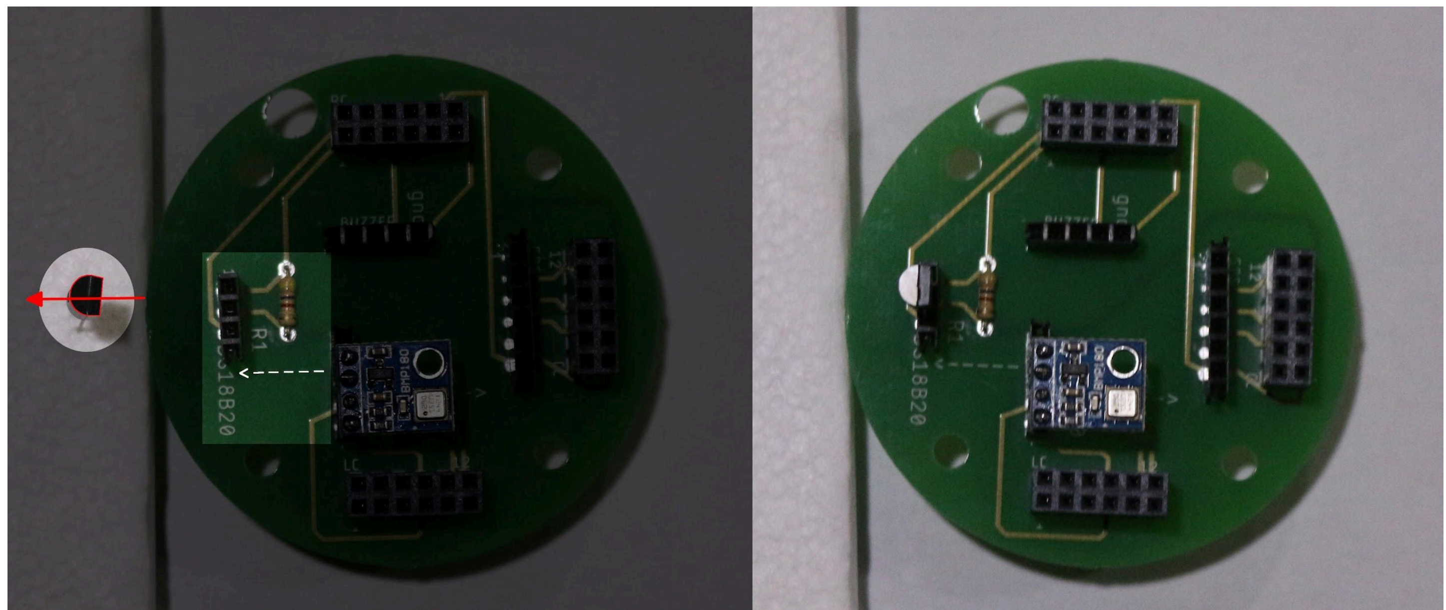
IMPORTANTE: Os kits da primeira edição do CanSat Júnior (2020) são incompatíveis com as seguintes. Confirme que a PCB Arduino que está a usar tem a inscrição v2 ou que a PCB rádio tem um fio vermelho ou preto soldado na parte de trás entre dois dos pinos de ligação entre placas. <u>Pode ignorar este aviso caso não use materiais fornecidos na primeira edição do CanSat Júnior (2020).</u>

1. Encaixar *level shifter* (TXB0104) na placa, alinhando o ponto dourado do componente com o marcado na PCB.

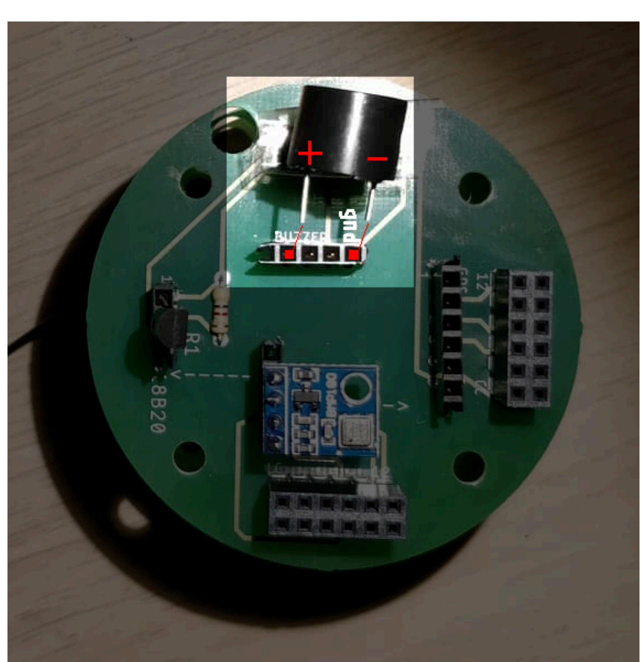


PCB Sensores

1. Encaixar sensor de pressão (BMP180) na placa, seguindo a orientação da seta.
2. Encaixar sensor de temperatura (DS18B20+) na placa, seguindo a orientação da seta. Consideramos a parte redonda do sensor como a sua frente.



3. Encaixar *buzzer* no *pin header* marcado com “BUZZER”. O pino mais curto (-) deverá ir para a ponta do *header* marcada com “gnd”, o mais longo (+) pode ser encaixado em qualquer um dos restantes buracos. Dependendo do modelo do buzzer o pino mais longo (+) poderá ser acompanhado por uma marca no topo ou na lateral.



Nota: O buzzer está internamente ligado ao pino 7.

4. O *pin header* marcado com GPS deverá ser deixado vazio.

Programação do CanSat

O CanSat (e a estação base) usam um Arduino como microcontrolador, que é habitualmente programado em C/C++. Para evitar lidar com a complexidade desta linguagem, recomenda-se a utilização de uma linguagem baseada em Scratch com editor disponível em https://cj.breda.pt/scratch. Tem 3 “tabs”:

- **Blocos** - Vista principal com editor visual por blocos;
- **Arduino** - Pré-visualização do equivalente do programa em blocos em código para o Arduino;
- **XML** - Representação XML (editável) do programa em blocos.

Este editor produz um ficheiro *.ino* (código C/C++) que pode ser aberto pelo programa “Arduino IDE”, que programa o Arduino.

Caso nunca tenha programado ou usado uma linguagem Scratch, aconselha-se a leitura do Anexo A: Introdução a Programação com Scratch.

Aconselha-se a leitura do Anexo B: Entrada/Saída do Kit no Scratch, que funciona como referência para todos os blocos específicos ao kit, e alguns que consideramos úteis específicos ao Arduino.

Caso já tenha programado um Arduino com C/C++, aconselhamos a não usar o Scratch e aproveitar para aplicar o seu conhecimento.Teste o seu cabo verificando que, usando-o para ligar o seu telemóvel ou o Arduino ao computador, o computador deteta o dispositivo.

Limitações do Scratch

O Scratch está desenhado para ser usado com linguagens mais expressivas que C/C++ como Python ou Javascript, que são mais relaxadas em vários aspetos (como tipos de dados e gestão de memória). Assim, há algumas operações que não podem ser expressadas neste Scratch:

1. Em C/C++ números inteiros e com casas decimais são representados com tipos diferentes (`int`/`long`/… e `float`/`double`). Neste Scratch, não há essa distinção.
2. Uma operação que misture números inteiros e decimais causa uma conversão do operando inteiro para decimal (`double`).
3. As variáveis só guardam inteiros (tipo `long int` em C) que conseguem guardar valores entre -2^{31} e 2^{31}-1.
   a. Se tentar guardar um valor com casas decimais numa variável, será implicitamente convertido para inteiro e arredondado para baixo (parte decimal truncada).
   b. Se tentar guardar um valor menor que -2^{31} ou maior que 2^{31}-1, o número vai “dar a volta”. -2^{31}-1 será guardado como 2^{31}-1 e 2^{31} será guardado como -2^{31}.
   c. Valores lógicos (verdadeiro ou falso) e níveis de tensão (HIGH e LOW) podem ser guardados em variáveis sem perda de informação (serão convertidos automaticamente para o tipo certo).
   d. Texto (*strings*) não pode ser guardado em variáveis.
   e. O número dado pelo bloco “Tempo ativo” está entre 0 e 2^{32} milissegundos. Isto significa que após ~24.9 dias de uso, se tentar guardar este valor vai obter um tempo ativo negativo (começando em -2^{31}, e vai subir até 0).

    f. Os argumentos e o valor de retorno de uma função são também variáveis, e estão por isso sujeitos a todas estas restrições.

4. O Scratch permite usar o valor de uma variável como valor lógico (por exemplo na estrutura “se-faça”). Caso tenha sido atribuído um valor numérico ou outro valor não-lógico à variável aplicam-se as seguintes regras de conversão:
    a. números diferentes de 0 são considerados “verdadeiros”;
    b. número 0 é “falso”.
5. A comparação de ordem ($<$, $>$, $\leq$, $\geq$) só é suportada para expressões numéricas. Outros valores poderão ser implicitamente convertidos para um número inteiro.

6. O programa dado vai correr em ciclo infinito. Caso o seu programa inclua (termine) num ciclo infinito, isto é inconsequente (se nunca sai do seu ciclo, nunca vai voltar ao início do seu programa).

Estas limitações não impedem a programação do CanSat (p. ex. não é expectável um CanSat estar 25 dias ligado), nem obrigam a ter um programa mais complicado do que o necessário. Note que o CanSat deve fazer o mínimo possível de processamento de dados, tanto para poupar tempo como energia.

Conselhos

- **Planeie** o que quer que o CanSat faça, da forma mais específica possível (“primeiro A, depois B, depois se C, então D, senão E”) antes de começar a usar o Scratch.
    - Pode ajudar fazer vários planos, cada vez mais específicos, sobre o que se pretende fazer.
    - À medida que ganha experiência, passa a ser só necessário um plano breve sobre o que se pretende fazer.
- Faça **pequenas experiências** com o Scratch antes de tentar programar o CanSat inteiro. Corra os exemplos, faça alterações, veja o que acontece.
- **Comente** o programa. Os comentários são pequenos bocados de texto que descrevem o que cada parte do programa está a fazer. São ignorados pelo compilador (e consequentemente pelo Arduino), por isso podem mesmo ter qualquer coisa.
    - Como sugerido no Anexo A, blocos de texto soltos funcionam bem como comentários.

Pré-requisitos

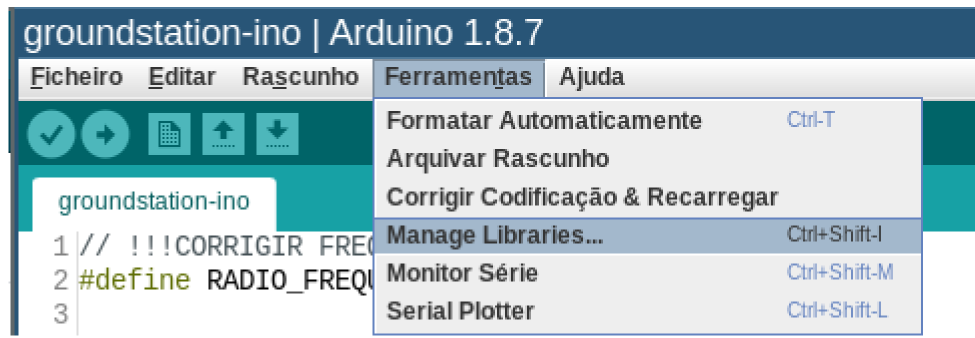
Para começar deve transferir e instalar a Arduino IDE, disponível em https://www.arduino.cc/en/Main/Software.

Qualquer engenheiro/programador é preguiçoso, e por isso não vai repetir o trabalho de escrever código para interagir com um sensor/atuador. Ou usa uma biblioteca já feita, ou escreve a sua e depois usa-a sempre que precisa. Para programar o CanSat, precisamos das seguintes bibliotecas:

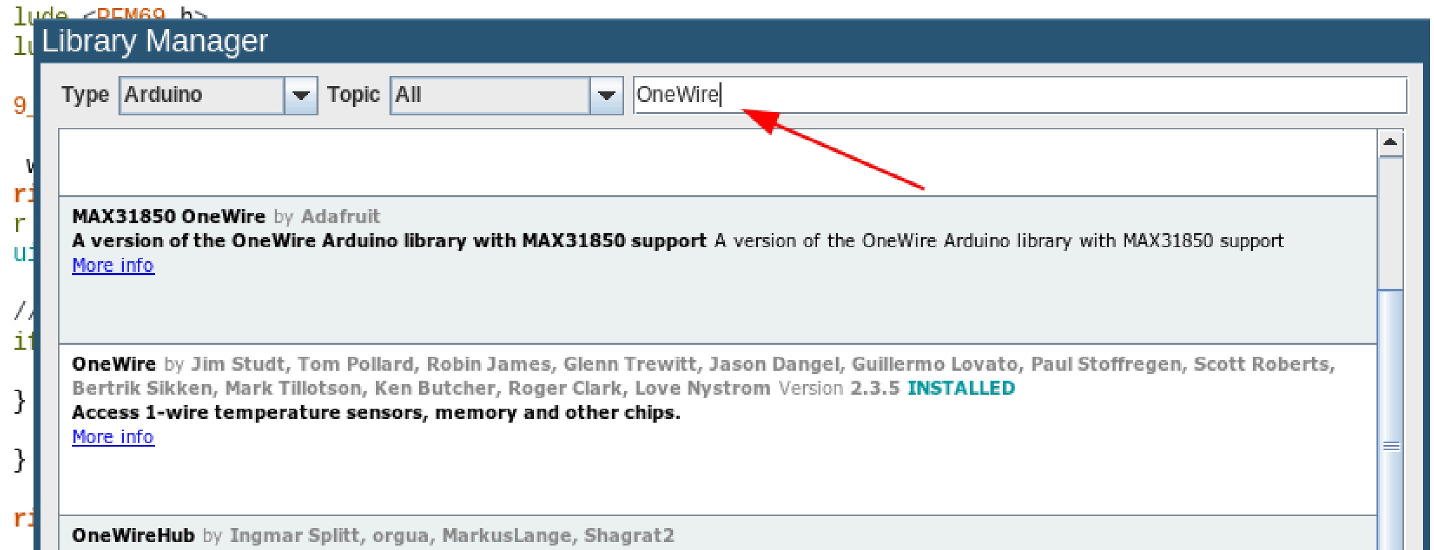
- Adafruit BMP085 Library
- DallasTemperature (aceite a instalação das dependências)
- RFM69_LowPowerLab (aceite a instalação das dependências)

- **Caso esteja a usar o sensor DHT11:** DHT Sensor Library (aceite a instalação das dependências)
- **Caso esteja a usar o módulo GPS:** TinyGPSPlus (tem de ser transferida de https://github.com/mikalhart/TinyGPSPlus/archive/refs/tags/v1.0.2b.zip e instalada manualmente através de Sketch > Incluir Biblioteca > Adicionar biblioteca .ZIP)

Para instalar uma biblioteca, use o *Library Manager* em Ferramentas > Manage Libraries…



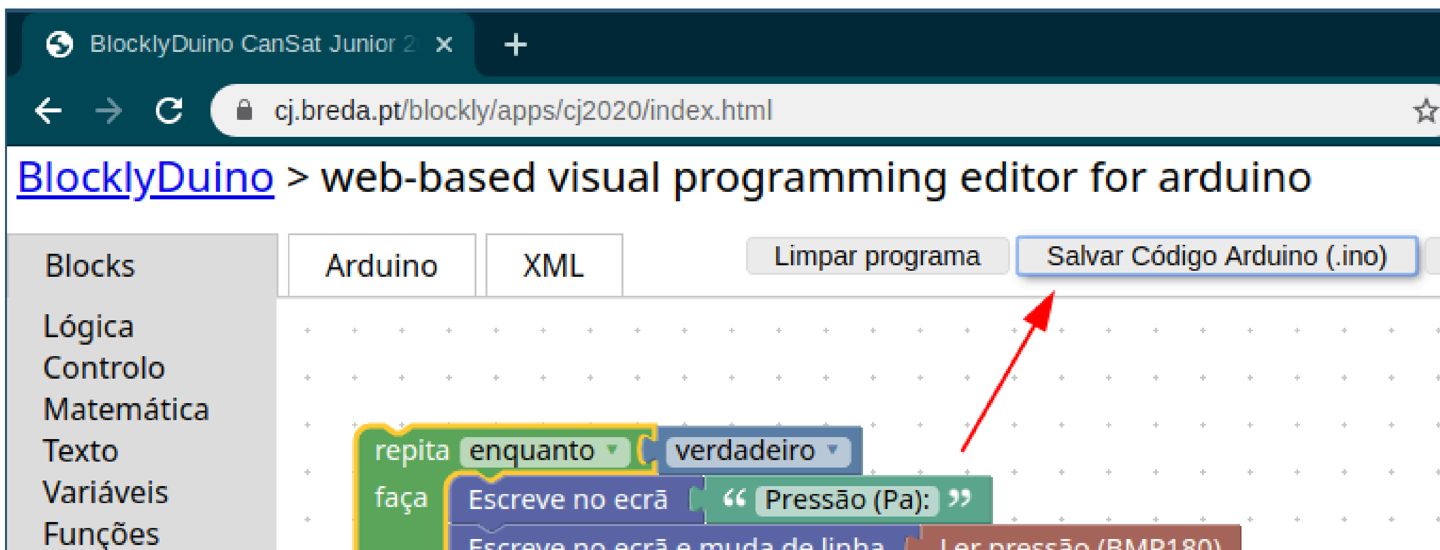
A sua operação é idêntica ao *Boards Manager* usado anteriormente. Como existem muitas bibliotecas disponíveis, aconselha-se o uso da caixa de texto no topo da janela para filtrar os resultados:



Procedimentos

Exportar Programa em Scratch para C/C++

Para exportar o seu programa Scratch para um formato compatível com a Arduino IDE, use o botão “Salvar Código Arduino (.ino)”:

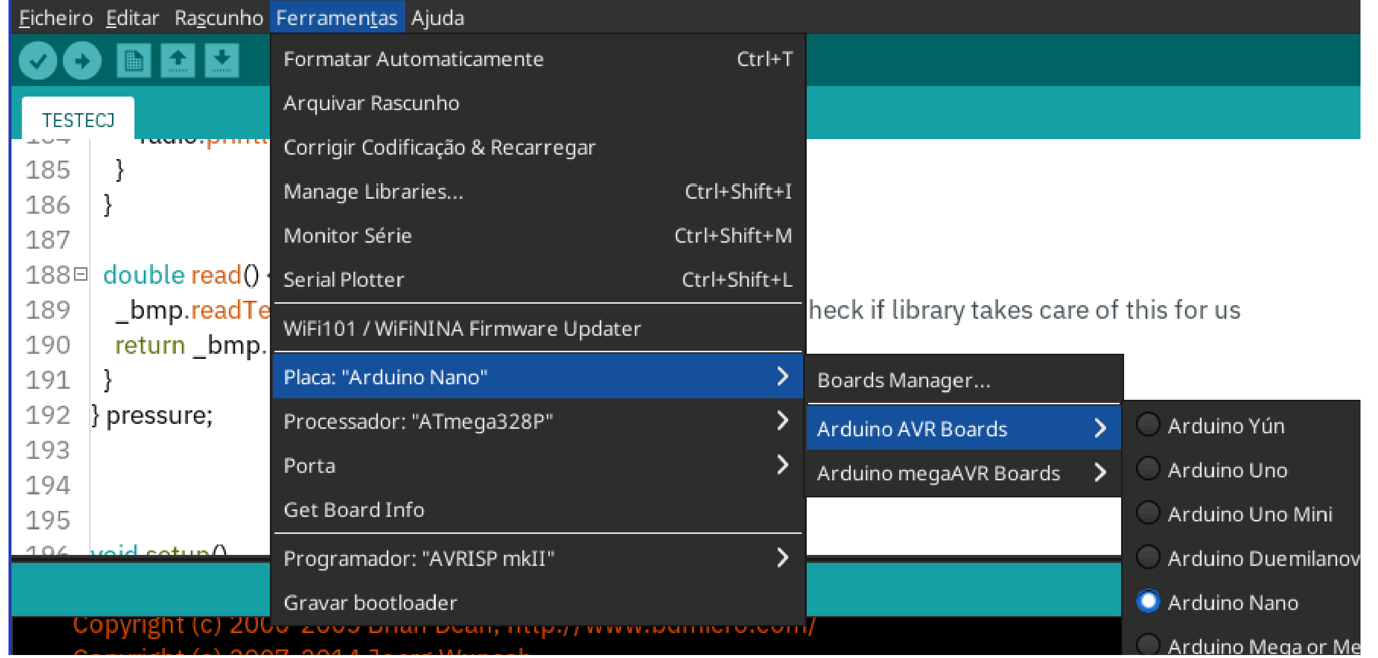


Será perguntado o nome que quer dar ao ficheiro, e depois ele será transferido para o seu computador.

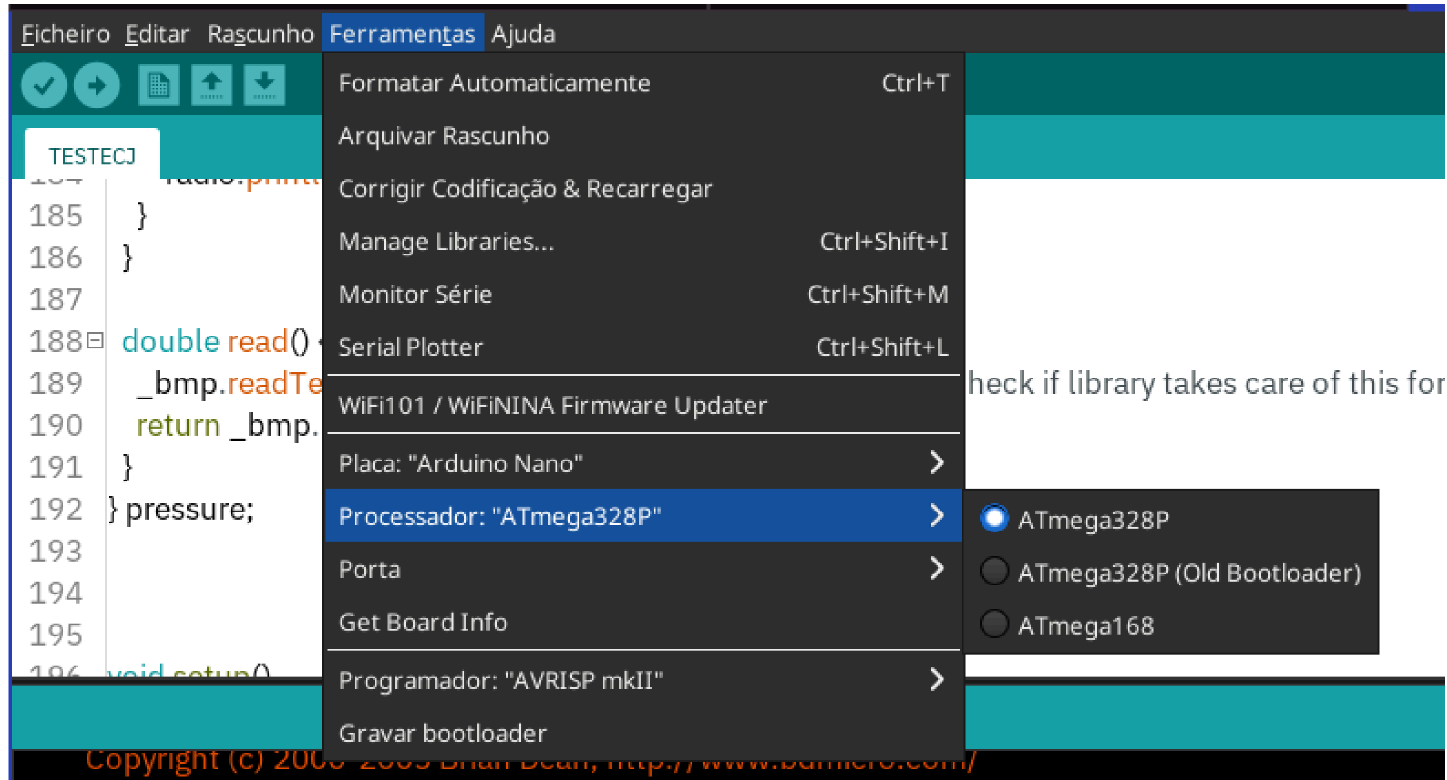
Programar Arduino a partir de Código C/C++

Abra a “Arduino IDE” que instalou na secção de Pré-requisitos. Use o menu Ficheiro > Abrir… para abrir o programa (ficheiro .ino) que quer usar no Arduino.

Garanta que selecionou “Arduino Nano” em Ferramentas > Placa:

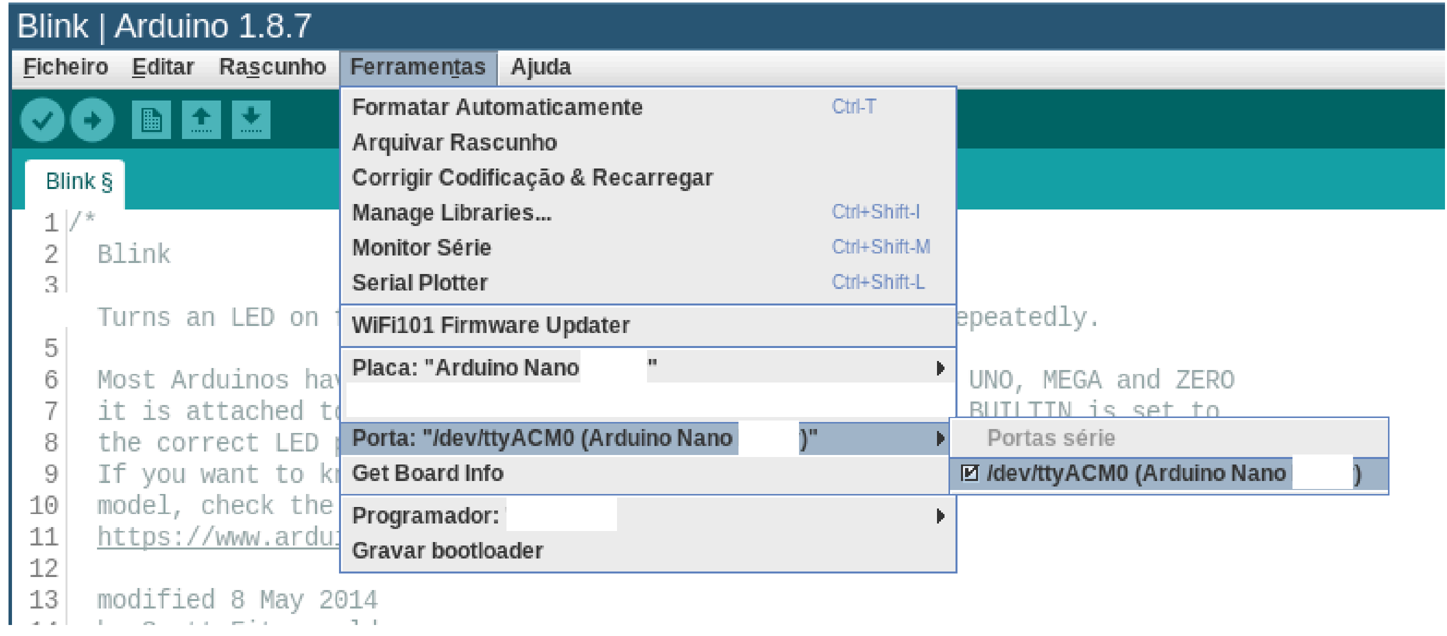


Confirme que a opção escolhida em Ferramentas > Processador é "ATmega**328P**" e **não** "ATmega 328P **(Old Bootloader)**" ou "ATmega**168**", como é visível na imagem abaixo.



Se a opção incorreta estiver selecionada a programação do Arduino vai falhar com o erro *"avrdude: stk500v2_ReceiveMessage(): timeout"*. Basta selecionar a opção correta e tentar novamente.

Garanta que o seu Arduino está selecionado em Ferramentas > Porta (a nomenclatura de nomes de porta apresentada - /dev/tty… - é a de macOS e Linux, em Windows aparece algo como COM1):



As definições de tipo de placa e porta são mostradas no canto inferior esquerdo da Arduino IDE. O tipo de placa é relembrado em execuções futuras.

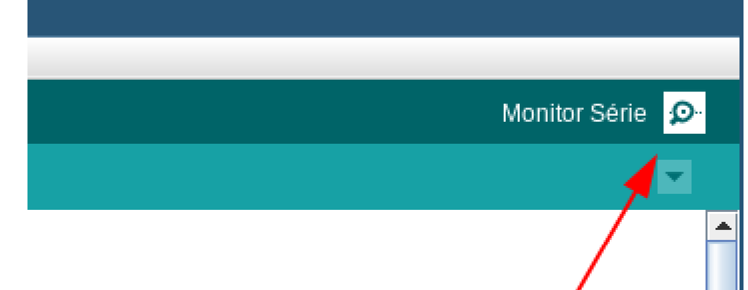
Está agora pronto para fazer *upload* do programa para o Arduino. Clique na pequena seta redonda:

Quando o carregamento do programa terminar, aparecerá uma mensagem semelhante a “Envio completo” no fundo da janela por cima da zona preta com mensagens do compilador/programador.

Ver saída do Arduino

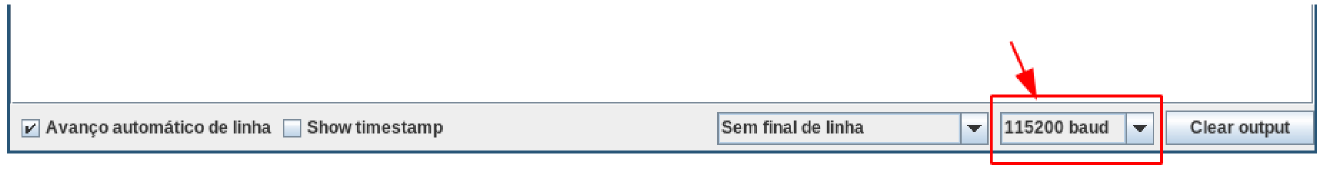
O Arduino que usamos tem uma única UART: com pinos expostos, e ligada à porta USB, que nos permite enviar dados para o computador.

A UART funciona como um tubo, despejamos dados de um lado, e eles aparecem no outro pela mesma ordem que foram enviados.



Para escrever na UART podem ser usados os blocos da família “Escreve no ecrã”. Para ver a saída da UART do Arduino pode usar o “Monitor Série” da Arduino IDE (garanta que a porta do Arduino se encontra selecionada).

Garanta que a taxa de transmissão de dados no fundo da janela do monitor está definida para “115200 baud”:



Caso a taxa de transmissão seja diferente do Arduino poderá receber dados errados ou corrompidos

Exemplos de Programas

Em conjunto com este manual são fornecidos vários ficheiros .xml, que podem ser abertos pelo editor scratch.

Os exemplos relativos a sensores (temperatura-ds18b20.xml e pressao.xml), limitam-se a enviar a leitura do sensor para o computador (pela UART) e para durante 1 segundo, em ciclo infinito.

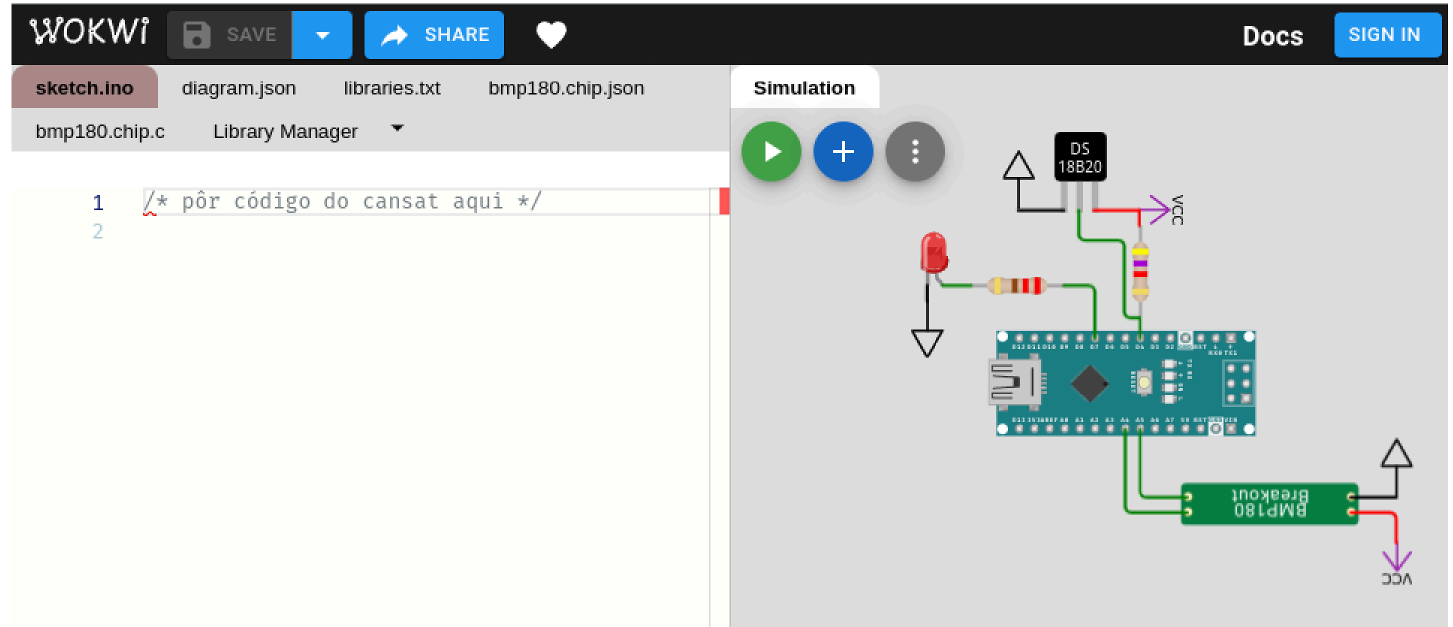
O exemplo do buzzer (buzzer.xml) é um pouco mais complexo (já usa uma variável e uma estrutura se-então-senão): liga/desliga o buzzer a cada segundo, em ciclo infinito.

O exemplo do rádio, envia o tempo que passou desde que o Arduino foi ligado (*uptime*), reporta quanto tempo passou durante o envio através da UART, e espera 1 segundo, em ciclo infinito.

Caso opte por usar C/C++ em vez de Scratch: Estes exemplos continuam a ser relevantes (depois de exportados para C/C++)! O código gerado é inteligível o suficiente para ser reutilizado. Recomenda-se em particular usar a classe *StreamedRF* do exemplo do rádio, de modo a garantir compatibilidade com o código fornecido para a estação base. Esta classe oferece uma interface idêntica aos objetos *Serial* (funções *write*, *print*, *println* e *flush*) tornando-a um método simples para interagir com o rádio.

Simulador

É possível simular o comportamento de um programa no CanSat Júnior recorrendo ao simulador disponível em https://cj.breda.pt/simulador/. Na zona esquerda estão os ficheiros usados pela simulação (só deve alterar o “sketch.ino” que aparece por omissão), e na zona direita o circuito a ser simulado.



Limitações do Simulador

De momento, não é possível simular todos os componentes de um CanSat Júnior. O sensores de pressão (BMP180) e as capacidades do Arduino são totalmente suportados. O buzzer ativo é simulado por um LED.

Relativamente ao sensor de temperatura (DS18B20), é parcialmente suportado: reporta a temperatura correta, mas o tempo gasto no processo de medição não corresponde ao real. Adicionalmente, mudar a resolução do sensor não tem efeito.

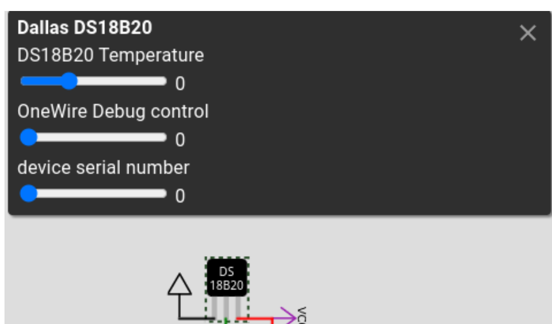
Por fim, o rádio (RFM69HCW) e componentes não incluídos no kit não são suportados de todo. Como o envio de mensagens via rádio congela a simulação, as bibliotecas necessárias ao funcionamento do rádio não estão presentes no ambiente de simulação por omissão. Assim, tentar usar o rádio no ambiente de simulação vai dar um erro relativo a bibliotecas não encontradas.

Usar o Simulador

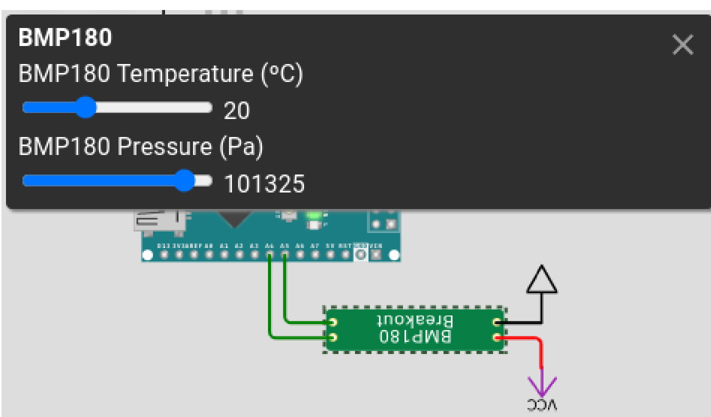
Para usar o simulador, o seu programa deverá ser um único ficheiro (ou ter origem no editor scratch). Basta colocar o programa na zona branca correspondente ao ficheiro “sketch.ino” (pode copiar e colar o texto do Scratch ou da IDE do Arduino), e clicar no botão .

A simulação começa imediatamente. Durante a simulação, pode-se observar a saída do Arduino (ver “Serial Monitor” que aparece no canto inferior esquerdo) e o estado do buzzer, representado como um LED vermelho (aceso quando o buzzer está ligado, apagado caso contrário). É possível também alterar os valores reportados pelos sensores: basta clicar no sensor a mudar e usar as barras deslizantes para alterar os parâmetros associados a cada sensor.

O sensor de temperatura DS18B20 simulado (retângulo preto no topo do circuito) permite alterar diversos parâmetros, mas para o CanSat Júnior só deverá interessar o primeiro - a temperatura:



O sensor de pressão BMP180 simulado (retângulo estreito verde no fundo do circuito) permite manipular a temperatura e pressão reportadas. Note que, apesar do editor Scratch só permitir obter a medição de pressão final deste sensor, o BMP180 afina a sua medição de pressão com uma de temperatura.



Para parar ou pausar a simulação use os botões ■ ou ⏸, que aparecem à direita do botão que dá início à simulação.

Estação Base

O programa do Arduino da estação base e o programa de captura de dados estão disponíveis em https://github.com/abread/groundstation/releases. Existem 3 ficheiros “zip” com a aplicação para Windows, macOS e Linux, e um último ficheiro “zip” com o código do Arduino.

IMPORTANTE: O conteúdo do ficheiro “zip” tem de ser extraído na íntegra para o programa de captura de dados funcionar.

IMPORTANTE: A versão compilada para macOS não foi testada. Se abre e funciona, não há razão para deixar de funcionar, mas também não podemos dar garantias de que vai funcionar.

IMPORTANTE: Garanta que o cabo USB que usa consegue transmitir dados (um bom teste é programar o Arduino com ele).

O Arduino da estação base deverá ser programado da mesma maneira que o do CanSat, mas com o programa fornecido para a estação base. Devem **antes de fazer “Upload”**, **alterar a frequência do rádio** (em Hz) para a desejada (na 2ª linha do ficheiro, que tem por defeito 433000000 Hz).

O rádio e Arduino da estação base são alimentados pelo cabo USB ligado ao Arduino (por isso a PCB não tem um bloco de terminais para ligar a bateria).

O programa para o computador da estação base tem 2 fases de funcionamento:

1. Configuração, em que seleciona a porta de série do Arduino (tal como na Arduino IDE) e o ficheiro em que os dados serão guardados (o nome pré-definido será guardado na pasta em que executam o programa).
2. Operação, em que são mostrados os dados a chegar e o RSSI atual, que indica a intensidade do sinal recebido

Os dados são gravados continuamente à medida que são recebidos. Quando a missão termina basta fechar o programa.

Anexo A: Introdução a Programação com Scratch

O computador é uma máquina simples: dada uma lista de instruções (programa), executa-as uma a uma, sequencialmente. Para conseguir dinamismo, existem estruturas de controlo que podem fazer o computador saltar para sítios diferentes do programa (em vez de continuar para a instrução seguinte), e variáveis que permitem guardar valores diferentes ao longo do tempo.

Os vários blocos disponíveis estão organizados em categorias:

- Lógica: estrutura de controlo se-faça-senão, comparações, operadores lógicos, valores lógicos
- Controlo: restantes estruturas de controlo, pausa, tempo ativo
- Matemática: valor numérico, aritmética
- Texto: valor textual (*string*)
- Variáveis: atribuição de valores a variáveis e leitura de variáveis
- Funções: definição e uso de funções/procedimentos, estrutura se-retorna
- Entrada/Saída: blocos que interagem com o mundo (ler sensores/pinos, enviar mensagens, etc.)

As categorias estão dispostas numa barra cinzenta do lado esquerdo do ecrã. Para ver os blocos disponíveis numa categoria, clique no nome da mesma. Para usar um bloco, arraste-o para fora da lista de blocos da categoria para o espaço de trabalho (zona branca com grelha de pontos).

Alguns blocos podem trazer outros agarrados por conveniência (isto acontece quando um bloco é muito usado com outro).

As saliências dos blocos indicam onde outros blocos podem ser encaixados: instruções podem ser encaixadas umas nas outras na vertical, expressões encaixam noutros blocos na horizontal.

Todos os blocos no espaço de trabalho são parte do seu programa e aparecem no código exportado.

O programa é lido como um texto, da esquerda para a direita, de cima para baixo.

Nota: Este anexo foi escrito para ser útil na maioria das implementações do Scratch, e os conceitos apresentados são relevantes para a maioria das linguagens de programação. Por esta razão, **não** serão aqui abordados blocos de Entrada/Saída, que são diferentes para cada tipo de computador.

Antes de avançar, convém ter um método para experimentação. O bloco “Escreve no ecrã” da categoria Entrada/Saída aceita qualquer expressão e mostra-a no ecrã.

No Anexo B apresentaremos mais blocos de Entrada/Saída, que dizem respeito a um Arduino em geral, e aos sensores/rádio do kit.

Expressões e Tipos

Uma expressão é um valor. Constantes, variáveis, aplicação de funções, operações são todas expressões válidas (“1”, “f(x, 2)”. “2*(1+1)”). Expressões iguais têm o mesmo valor (1+1 e 2 são a mesma coisa).

Quando escrevo 1+1, eventualmente o computador vai ter de **avaliar a expressão**, isto é, calcular o seu valor na forma mais simples (constante). Esta avaliação pode ser feita de muitas maneiras: assim que chegamos à expressão, só quando a tentamos imprimir/usar, pelo compilador antes do programa sequer executar (caso o

valor possa ser determinado em tempo de compilação), etc. O que interessa é que a avaliação é sempre feita a tempo - quando precisamos, o valor foi calculado.

Como nem todos os dados são representados da mesma maneira por um computador e nem todas as operações fazem sentido (o que é “Olá” * “Adeus”?), todas as expressões têm um tipo.

Porquê falar de expressões? É útil saber que em todos os sítios que podemos dar um bloco com um valor constante (expressão atómica), também podemos dar um bloco com uma expressão mais complexa, com um número arbitrário de operações envolvidas.

Os tipos disponíveis no Scratch são:

- Numérico
- Textual (*string*)
- Lógico - verdadeiro ou falso

Muitas linguagens subdividem o tipo numérico em tipos inteiros (com vários limites de tamanho do número) e que números com casas decimais (normalmente de vírgula flutuante, com vários limites de precisão).

Constantes

O tipo de expressões mais simples que existe são as constantes.

Exemplos de constantes:

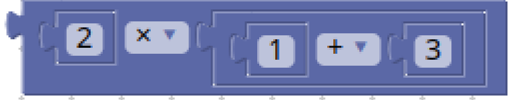
Da esquerda para a direita: o número 42, o texto “A resposta” e o valor lógico verdadeiro.

Dica: O bloco de texto, quando não está ligado a nada, pode ser usado como comentário. Vai aparecer no código exportado, mas será eliminado pelo compilador (porque não faz nada).

Aritmética

Os operadores aritméticos podem ser aplicados a qualquer par de expressões numéricas, e são uma expressão numérica. O bloco de aritmética pode somar, subtrair, multiplicar, dividir e exponenciar (^); para alterar a operação do bloco, clique na operação atual (que tem uma pequena seta ao lado a indicar que é um menu).

Exemplo: $2 * (1 + 3)$
equivalente ao número 8



Lógica

Temos dois três blocos diferentes a representar expressões lógicas: comparação, operadores lógicos binários e negação.

Comparação

Um operador lógico de comparação compara dois valores e representa um valor lógico.

Se a linguagem subjacente converter tipos automaticamente, essa conversão será feita antes da comparação, caso contrário valores de tipos diferentes ou não são comparáveis, ou são sempre diferentes.

Exemplo:
1+3=4, 2x2=4,
4 não é diferente de 4,
$4 \neq 4$ é falso,
logo $(1 + 3) \neq (2 * 2)$ é falso,
logo a expressão $((1 + 3) \neq (2 * 2)) = falso$ é verdadeira.

Este foi um exemplo mais complexo que pretendeu demonstrar a infinidade de operações possíveis com expressões.

Operadores Lógicos Binários

Permitem expressar a conjunção (“e”) e a disjunção (“ou”). São binários apenas porque aceitam dois operandos.

Assumindo que *coisa-1* e *coisa-2* são expressões lógicas (verdadeiras ou falsas):

- “*coisa-1* e *coisa-2*” é verdade se e só se *coisa-1* é verdade e *coisa-2* é verdade
- “*coisa-1* ou *coisa-2*” é verdade se pelo menos uma das coisas forem verdade, e falso se nenhuma for.

Por exemplo, se eu disser “Gosto de carne e peixe”, mas só gostar de carne, estou a mentir; mas se disser “Gosto de carne **ou** peixe” estou tecnicamente a dizer a verdade (embora de uma forma estranha).

Exemplo:
Esta expressão é verdadeira.
falso **ou** 1=1 (que é verdadeiro) é verdadeiro,
logo (falso ou 1=1), que é verdadeiro, **e** verdadeiro é verdadeiro.

Negação

O operador “não” inverte um valor lógico. não verdadeiro é falso.

Outros tipos de expressões

São abordados nas secções seguintes, mas podemos saber desde já que variáveis e aplicação de funções ("usar" uma função) também são expressões.

Curiosidade: Há linguagens de programação em que tudo (ou quase tudo) são expressões.

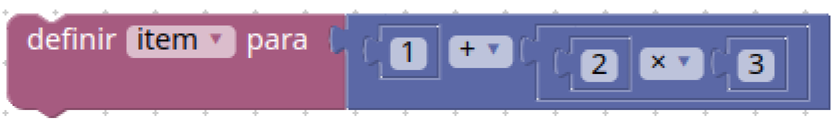
Variáveis

O computador tem memória que lhe permite guardar valores. Na verdade, tem vários tipos de memória, numa hierarquia organizada pela velocidade de acesso (e quando o valor que quer aceder não está presente na 1ª, tenta a seguinte que é mais lenta). Damos destaque à memória RAM, que guarda todos os dados do programa em execução (incluindo uma cópia do próprio programa, dependendo da arquitetura do computador).

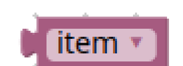
Podemos pensar nas variáveis como baldes com etiqueta (nome) que guardam um valor a dado momento. Podemos a qualquer momento ver o que está dentro do balde procurando-o pelo nome na etiqueta, e podemos também substituir o conteúdo do balde (usando mais uma vez o nome).

Em Scratch, as variáveis são declaradas implicitamente, isto é, são criadas automaticamente pelo Scratch à medida que pedimos variáveis diferentes.

Para atribuir um valor diferente a uma variável, usamos o bloco "definir-para", que pode atribuir qualquer expressão a uma variável:



Para usar o valor de uma variável como expressão:

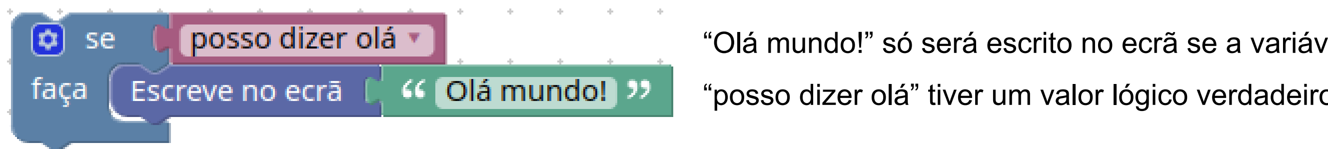


Embora as expressões abordadas na secção anterior sejam capazes de representar muitas computações, não trazem nada de interessante ao programa sem variáveis. Aliás, como otimização, o compilador pré-calcula expressões que só envolvem constantes e o programa compilado apenas contém o resultado. Isto significa também que podem ter cálculos no vosso programa para não ter números "caídos do céu", sem ocupar mais espaço por isso.

Estruturas de Controlo

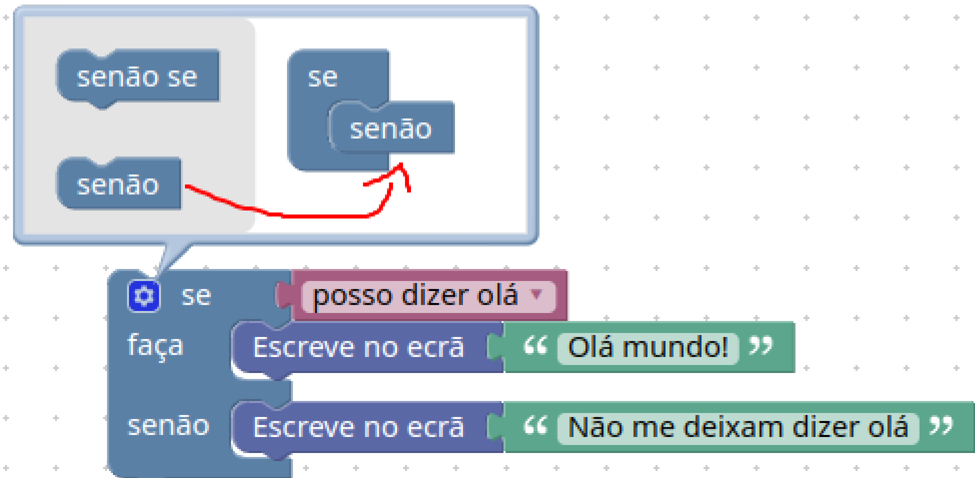
Se-Faça-Senão (*if-then-else*)

A estrutura se-faça-senão permite executar partes do programa condicionalmente.



"Olá mundo!" só será escrito no ecrã se a variável "posso dizer olá" tiver um valor lógico verdadeiro.

Para executar outra ação quando a condição é falsa, podemos estender este bloco com o bocado “senão”. Para isso clique na roda dentada azul à esquerda de “se” e arraste o bloco “senão” para o mini-bloco “se”.



Pode ainda acrescentar os bocados “senão-se” antes do “senão” final. Os blocos correspondentes à primeira condição verdadeira serão executados (e mesmo que outras seguintes sejam verdadeiras, serão ignoradas). Caso nenhuma seja verdadeira, os blocos associados ao bocado “senão” são executados.

Ciclos

Os ciclos permitem executar os mesmos blocos repetidamente.

Ciclos “repita n vezes” (*repeat-n*)

Um ciclo “repita n vezes” repete os blocos associados a ele o nº de vezes especificado.

repita 42 vezes
faça Escreve no ecrã “ Olá ”

Este exemplo escreve “Olá” 42 vezes no ecrã.

Ciclos “repita enquanto” (*while*)

Um ciclo “repita enquanto” executa os blocos associados a ele repetidamente, enquanto a sua condição seja verdadeira.

repita enquanto verdadeiro
faça Escreve no ecrã “ Olá ”

Este exemplo escreve “Olá” no ecrã infinitamente, em ciclo.

Alternativamente, pode trocar o ciclo para um “repita até”, que executa os blocos associados em ciclo até que a condição seja verdadeira, isto é, enquanto ela for falsa.

Ciclos “contar com” (*for*)

Um ciclo “contar com” é usado para repetir uma ação até uma variável contador atingir determinado valor. Pode ser especificado também o valor inicial da variável e quanto é que é somado a ela após cada iteração (incremento).

Este exemplo imprime todos os números de 1 a 42, de 2 em 2.

Escape e Continuação Abrupta de Ciclos (*break* e *continue*)

Por vezes, é mais fácil parar um ciclo a meio ou saltar uma iteração à frente sem complicar mais a condição do ciclo ou introduzir estruturas de controlo adicionais.

O bloco "sair do ciclo" sai do ciclo e continua a execução do programa no bloco imediatamente a seguir ao ciclo.

O bloco "continuar com a próxima iteração do ciclo" salta o resto da iteração atual à frente, e avança para a seguinte.

Podem ser usados em qualquer tipo de ciclo, e apenas dentro de um ciclo. O aviso (triângulo azul) indica quando são usados fora de um ciclo - um erro.

Caso tenha vários ciclos uns dentro dos outros, este bloco atua sobre o ciclo interior:

No exemplo anterior, o bloco “sair do ciclo” atua sobre o ciclo “contar com”, não sobre o “repita 10 vezes”.

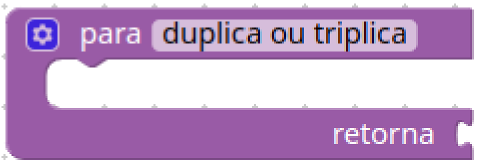
Funções e Procedimentos

A arte da programação passa por evitar repetição, por isso um bom programador não escreve o mesmo código mais do que uma vez: agrupa-o numa função/procedimento e depois usa essa função/procedimento.

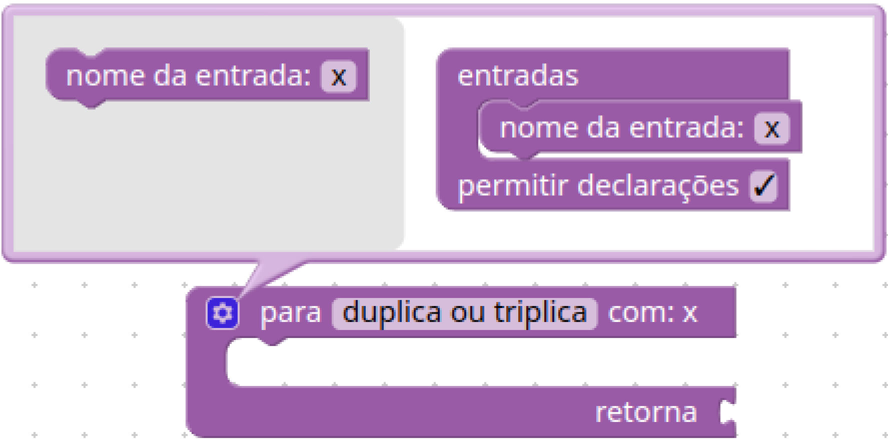
A diferença entre uma função e um procedimento, é que um procedimento apenas executa ações, e uma função retorna um valor.

Como caso de estudo, vamos implementar a função “duplica ou triplica”, que aceita um número “x”, duplica-o se for ímpar, triplica-o caso contrário.

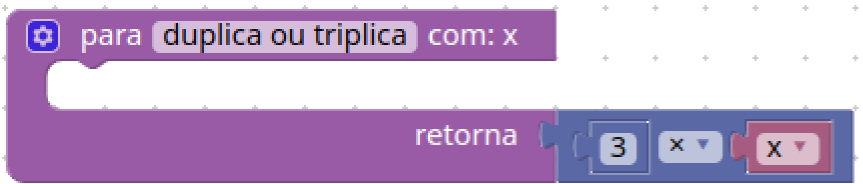
Começamos com o bloco de função vazio, e damos um nome à função:



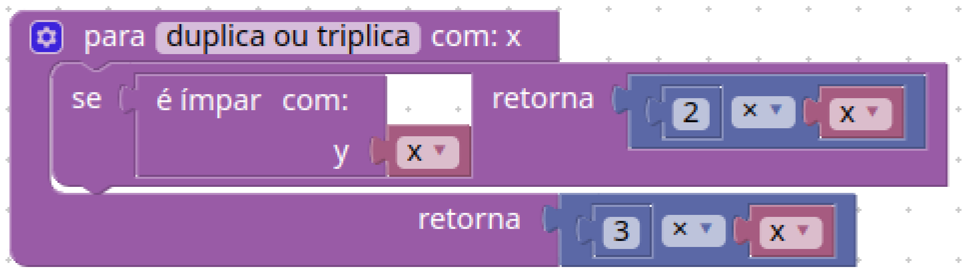
Depois, vamos adicionar o parâmetro/entrada “x”, arrastando o bloco “nome de entrada” para dentro do mini-bloco “entradas”. O nome do parâmetro pode ser mudado.



No que toca ao corpo da função, tratemos primeiro de triplicar x:



Temos um problema: como é que podemos retornar o dobro de x quando é ímpar? Não há nenhum bloco “retorna” isolado! Podíamos calcular o resultado no corpo da função, guardá-lo numa variável e retornar o valor dessa variável, mas vamos optar por usar o bloco especial “se-retorna”:



Note que o bloco “é ímpar” não existe no Scratch! É uma função criada previamente (a definição não é apresentada aqui) que retorna verdadeiro se o número y fornecido é ímpar, falso caso contrário. Para cada função definida, o Scratch mostra automaticamente o bloco para a chamar na secção “Funções”.
Note também que dentro da função “é ímpar” o parâmetro chama-se y, não x.

Quando uma função é executada, os valores passados aos seus argumentos são atribuídos aos nomes usados internamente pela função. As variáveis correspondentes aos argumentos de uma função (no caso de duplica ou triplica, a variável x) são temporárias: só existem durante a execução da função.

Por último, note que o bloco “se-retorna” não permite adicionar cláusulas “senão-se” ou “senão”. Não são necessárias, porque quando uma função retorna um valor, volta para o sítio onde foi chamada. Ter vários blocos “se-retorna” seguidos e o “retorna” final é equivalente a um bloco “se” com uma série de cláusulas “senão se” e uma cláusula “senão” final.

Nota: Há uma distinção subtil entre parâmetro e argumento. Um parâmetro de uma função é aquilo que a função aceita (o buraco onde temos de encaixar um valor), o argumento é o valor/expressão que efetivamente lhe damos.

Curiosidade: Na maioria das linguagens de programação atuais todos os procedimentos são funções que retornam um tipo “vazio” (*void* em C).

Anexo B: Entrada/Saída do Kit no Scratch

“Se uma árvore cai na floresta e ninguém está perto para a ouvir, será que faz um som?” - George Berkeley

Um computador que não é capaz de atuar sobre o mundo real (seja mostrando informação, seja mexendo um braço robótico) é um aquecedor caro. E um computador que não consegue percecionar o que se passa no exterior, mas que atua sobre ele, é um temporizador de rega caro.

Para que os recursos do computador não sejam desperdiçados, tem mecanismos de entrada e saída de informação. No caso do Arduino destacamos os seguintes:

- Pinos, cuja tensão pode ser lida
  - No caso dos “pinos digitais”, pode ser lida como HIGH (>2.5V) ou LOW (<2.5V)
  - No caso dos “pinos analógicos”, pode ser lida como um número entre 0 e 1024, que corresponde a uma tensão entre 0 e 5V
- Pinos, cuja tensão pode ser “escrita”
  - Um pino pode ser ligado ou desligado “escrevendo” nele HIGH ou LOW
- Uma UART
  - Ligada à porta USB que permite mostrar texto no computador ligado ao Arduino
  - Também exposta nos pinos RX/TX

Também suporta nativamente os protocolos de comunicação I^2C e SPI, e o esquema de saída PWM, que não serão aqui abordados diretamente.

Os sensores e rádio incluídos no kit usam estes protocolos de comunicação, mas estes pormenores de uso foram abstraídos, tanto pelas bibliotecas usadas no código, como pelos blocos do Scratch.

Nota: Quando escrevemos numa UART do Arduino, ele não espera que todos os dados sejam transmitidos antes de continuar o programa. Copia os dados para um *buffer* e um circuito independente do processador trata da transmissão. Só espera por transmissão se o *buffer* não tiver espaço suficiente para todos os dados. Isto significa que escrever no ecrã não demora quase tempo nenhum no nosso programa, fora casos em que se escreva demais.

Escreve no ecrã (*Serial*)

Para escrever no ecrã do computador (através da UART), podemos usar os blocos "Escreve no ecrã", que aceitam qualquer expressão (e convertem-na para texto antes de enviar):

Escreve no ecrã "Olá"
Escreve no ecrã e muda de linha "mundo!"
Escreve no ecrã e muda de linha "Olá."

Neste exemplo, aparece no ecrã:

Olá mundo!

Olá.

Rádio

Para definir a frequência base de operação do rádio, existe o bloco "Rádio: definir frequência". **Use a frequência atribuída à sua equipa!**

AVISO: Nem todas as frequências suportadas pelo rádio podem ser usadas, por questões legais. Consulte a

Rádio: definir frequência para 433000000 Hz

legislação local para saber que partes do espectro são de uso livre sem licença (frequências na faixa 430-435 MHz são de uso livre sem licença em Portugal à data de escrita, sujeitas a restrições de potência máxima de transmissão).

Para enviar informação para a estação base (escrever no rádio), temos blocos muito semelhantes aos "Escreve no ecrã".

Rádio: escreve "Olá"
Rádio: escreve e muda de linha "mundo!"
Rádio: escreve e muda de linha "Olá."
Rádio: forçar envio

Neste exemplo, aparece na estação base:

Olá mundo!
Olá.

Os blocos "Rádio: escreve" aceitam também qualquer expressão, e convertem o seu valor para texto antes de enviar.

O rádio faz *buffering*, isto é, agrupa os dados em blocos (de 61 caracteres) antes de começar a tentar enviar por questões de eficiência.

Cada vez que enviamos dados, é calculado e enviado um código de deteção de erros. Se em vez de enviar 10 caracteres no mesmo bloco enviarmos um de cada vez, acabamos por ter de enviar o triplo dos dados (por

causa do código de deteção de erros, entre outros), o que corresponde a gastar ~3x mais tempo e ~3x mais energia.

O rádio não tem maneira de distinguir entre dados "presos" no *buffer* à espera que chegue o ciclo seguinte e comecemos a escrever novamente (como na frase "Olá.") e dados que simplesmente vão ser escritos com várias instruções (como na frase "Olá mundo!"). Para evitar a situação de dados presos, temos o bloco "Rádio: forçar envio", que obriga o rádio a enviar os dados que tem à espera, mesmo que não sejam do tamanho de um bloco inteiro. Caso não existam dados pendentes, o bloco "Forçar envio" não tem efeito.

É possível que a transmissão de dados falhe, pelo que se recomenda escolher um formato de envio de dados resiliente a perdas. A perda de um bloco de 60 caracteres não deve tornar a informação irrecuperável.

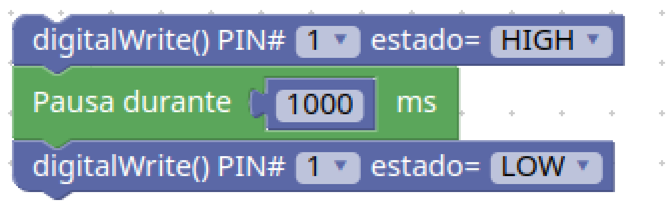
Sugestão: usar apenas 59 caracteres para dados, e começar e terminar pacote com mudança de linha ou outro carácter separador, e forçar o envio antes e depois da escrita do pacote. Caso exista muita informação para transmitir, separá-la em vários pacotes de tipos diferentes que serão processados de forma independente em terra

Nota: Ao contrário do que acontece com o "Escrever no ecrã"/UART, escrever no rádio vai parar o programa até que os dados sejam todos enviados quando o *buffer* encher/chamarem "forçar envio". Não há circuito independente no Arduino para copiar dados para o rádio.

digitalWrite

Para ligar/desligar um pino digital, usamos o bloco digitalWrite(). O estado HIGH representa ligado, LOW representa desligado.

Por exemplo, para fazer barulho com um *buzzer* ligado ao pino 1 durante 1 segundo:



Curiosidade: Este bloco chama-se *digitalWrite* porque é esse o nome da função em C/C++ para fazer esta operação.

LED do Arduino

O Arduino tem um LED embutido que pode ser controlado pelo programa. Controlar o LED significa controlar o pino que lhe dá energia. No caso do Scratch, não está na lista de pinos do bloco *digitalWrite*, mas temos um bloco que o deixar desligar e ligar:

Tempo ativo

Um computador tem tipicamente um relógio sempre ligado (quando o computador está desligado é alimentado ou pela bateria, caso exista, ou por uma pilha dedicada ao relógio), para quando o voltarem a ligar ele saber a hora atual. O Arduino não tem um relógio assim, mas consegue saber quanto tempo passou desde que foi ligado. O bloco "Tempo ativo" dá-nos o quantos milissegundos passaram desde que o Arduino foi ligado (volta a 0 a cada 2^{32} milissegundos).

Está na categoria "Controlo" porque no caso do Arduino não é tecnicamente um dispositivo de entrada/saída, mas um contador no programa (a cada 1ms a execução é interrompida para aumentar o contador).

Tempo ativo

Curiosidade: Isto existe porque um processador funciona à base de um relógio. Cada vez que o relógio avança (e o tempo entre cada avanço é na ordem dos 10^{-6}s para o arduino e 10^{-9}s para o computador), o processador executa mais uma instrução. O tempo ativo funciona contando quantos ciclos de relógio passaram e multiplicando pela duração de um ciclo.

Sensor de Temperatura DS18B20

Este sensor suporta medições assíncronas, isto é, o Arduino pede uma medição, pode continuar a trabalhar noutra coisa, e mais tarde terá uma leitura pronta. Além disso, é possível ligar vários sensores em paralelo no mesmo circuito.

Para ler a temperatura atual (em ºC) do(s) sensor(es) DS18B20, comece por pedir a todos os DS18B20 ligados para começar a medição:

DS18B20: Pedir medição de temperatura

O tempo que o sensor demora a fazer a medição varia consoante a resolução configurada, que por omissão é o valor mínimo - 9 bits - que precisa do menor tempo de espera. Pode ser alterada com o bloco "DS18B20: Definir resolução para __ bits". Consulte a *data sheet* do sensor para mais informação sobre as resoluções possíveis e tempos de espera associados (*Temperature Conversion Time*).

Quando quiser aceder aos resultados da última medição, o bloco "Obter temperatura" irá bloquear (esperar) até que a última medição pedida acabe, caso ainda não tenha passado tempo suficiente, e retornar a última temperatura observada pelo sensor pedido:

DS18B20: Obter temperatura do sensor # 0

Caso tenha apenas um sensor, este será identificado pelo número 0 (zero), como mostrado na figura acima. Para configurações com múltiplos sensores, serão todos detectados automaticamente e a ordem de deteção é determinística (a ordem relativa entre números atribuídos aos sensores não muda).

Nota: Ao mover mais trabalho para entre o pedido e obtenção da medição de temperatura, é possível aumentar a resolução do sensor.

AVISO: CUIDADO AO GUARDAR TEMPERATURAS EM VARIÁVEIS. Recordando as Limitações do Scratch, as variáveis só são capazes de guardar números inteiros. Pode multiplicar o valor por uma potência de 10 (ou, ainda melhor, por uma potência de 2, que é mais eficiente) no CanSat, e voltar a dividir do lado da estação base, para evitar perder casas decimais.

Curiosidade: Este bloco abstrai toda a comunicação necessária para pedir uma leitura ao sensor e obtê-la. Concretamente, abstrai a implementação do protocolo OneWire da Maxim (fabricante do DS18B20), e a sua utilização para ler o sensor (que fala este protocolo).

Sensor de Pressão BMP180

Para ler a pressão atual (em Pa) do sensor BMP180, podemos usar o bloco “BMP180: Ler pressão”:

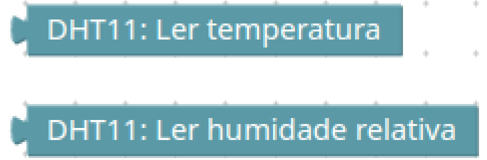
AVISO: CUIDADO AO GUARDAR PRESSÃO EM VARIÁVEIS. Recordando as Limitações do Scratch, as variáveis só são capazes de guardar números inteiros. Pode multiplicar o valor por uma potência de 10 (ou, ainda melhor, por uma potência de 2, que é mais eficiente) no CanSat, e voltar a dividir do lado da estação base, para evitar perder casas decimais.

Curiosidade: Este bloco abstrai toda a comunicação necessária para pedir uma leitura ao sensor e obtê-la. Concretamente, abstrai o uso do protocolo I^2C (já implementado no Arduino) para pedir leituras de temperatura e pressão ao sensor. A leitura de temperatura é usada pelo sensor para compensar a variação de pressão com a temperatura.

Sensor de Temperatura e Humidade DHT11

NOTA: O kit não inclui este sensor. Não é necessário à missão primária.

Para ler a temperatura atual (em ºC) ou a humidade relativa (%, número decimal entre 0 e 1), podemos usar os seguintes blocos:



AVISO: CUIDADO AO GUARDAR TEMPERATURA/HUMIDADE RELATIVA EM VARIÁVEIS. Recordando as Limitações do Scratch, as variáveis só são capazes de guardar números inteiros. Pode multiplicar o valor por uma potência de 10 (ou, ainda melhor, por uma potência de 2, que é mais eficiente) no CanSat, e voltar a dividir do lado da estação base, para evitar perder casas decimais.

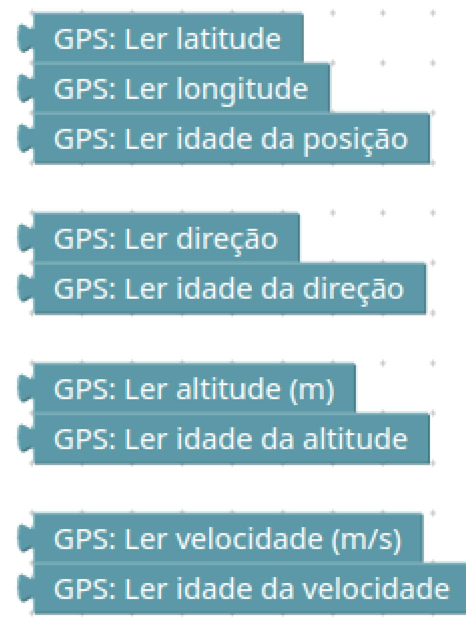
Receptor GPS

NOTA: O kit não inclui receptor GPS. Não é necessário à missão primária.

Para bom funcionamento do receptor GPS, o primeiro bloco presente no programa tem de ser o “GPS: Processar dados pendentes”:

GPS: Processar dados pendentes

O receptor GPS funciona de forma independente do Arduino: comunica com vários satélites para determinar latitude, longitude, altitude, direção, velocidade, etc. e reporta esta informação ao Arduino 1 vez por segundo. O bloco “processar dados pendentes”, lê estes relatórios do GPS e interpreta-os, disponibilizando a informação através dos blocos que se seguem:



Latitude e longitude expressas em graus.
Idade da posição corresponde a quantos milissegundos passaram desde a última informação de latitude/longitude válida recebida do GPS.

Direção expressa em graus.
Idade da direção corresponde a quantos milissegundos passaram desde a última informação de direção válida recebida do GPS.

Altitude expressa em metros.
Idade da altitude corresponde a quantos milissegundos passaram desde a última informação de altitude válida recebida do GPS.

Velocidade expressa em metros por segundo.
Idade da velocidade corresponde a quantos milissegundos passaram desde a última informação de velocidade válida recebida do GPS.

AVISO: CUIDADO AO GUARDAR VALORES DO GPS EM VARIÁVEIS. Recordando as Limitações do Scratch, as variáveis só são capazes de guardar números inteiros. Pode multiplicar o valor por uma potência de 10 (ou, ainda melhor, por uma potência de 2, que é mais eficiente) no CanSat, e voltar a dividir do lado da estação base, para evitar perder casas decimais. As idades são sempre números inteiros, pelo que não precisam de tratamento especial.

O bloco “Pause durante ___ ms” processa dados pendentes do GPS antes de pausar o Arduino quando o GPS está em uso. Por esta razão, a pausa pode ser superior ao tempo pedido

Glossário

Arduino	Microcontrolador desenhado para ser de baixo custo e de fácil uso/aprendizagem.
Atrito	Força que se opõe ao movimento. É a força que nos trava quando paramos subitamente de correr, impedindo-nos de deslizar infinitamente.
Buffer	Bocado de memória para armazenamento temporário de dados. Pode ser usado para enviar dados em blocos (por razões de eficiência), ou como fila de espera para um dispositivo de entrada/saída mais lento (UART).
C/C++	C e C++ são duas linguagens de programação. C++ foi desenhada a partir de C, e a maioria da sintaxe de C é válida em C++
Compilador	Tradutor. Normalmente de uma linguagem de programação de "alto nível" (mais abstrata, mais legível), para uma de "baixo nível" (mais próxima da linguagem máquina). No caso do Arduino, o compilador (gcc) compila C/C++ para linguagem máquina.
Level shifter	Circuito que faz tradução entre dois níveis de lógica (p. ex. entre 5V e 3.3V).
Linguagem máquina	A linguagem que o computador executa ("percebe").
Microcontrolador (μC, MCU)	Pequeno computador usado em sistemas pequenos. Integra um microprocessador, memória, e dispositivos de entrada e saída.
Operador binário	Operador/Operação com dois operados. Exemplo: a divisão tem dois operandos (o dividendo e o divisor), logo é uma operação binária.
Operador unário	Operador/Operação com um operando. Exemplo: a negação tem um operando (verdadeiro ou falso), logo é uma operação unária.
PCB	Placa de circuito impresso (do inglês *Printed Circuit Board*).
Pin header **ou barra de pinos**	Fichas onde pinos podem ser inseridos para contacto elétrico e encaixe.
RAM (memória)	Do inglês: *Random Access Memory* Memória habitualmente usada para armazenar dados usados em execução (programa e variáveis).
RSSI	Do inglês: *Received Signal Strength Indicator* Indica quão forte é o sinal recebido. No caso do RFM69HCW é medido em dBm e varia entre -115 e 0 (quanto maior, mais forte é o sinal).
SMA (ficha)	Conector/Ficha para cabos coaxiais, de uso comum em antenas WiFi.
Transceiver	Rádio. Combina a funcionalidade de um transmissor (***trans****mitter*) e receptor (*re****ceiver***) de dados por radiofrequência.
UART	Do inglês: *Universal Asynchronous Receiver-Transmitter* Hardware com formato de transmissão e velocidade de transmissão configuráveis.