COMPUTATIONAL THINKING AND PROGRAMMING SKILLS

FOR ADVANCED

primary school - 5th to 9th grades

VERSÃO EM PORTUGUÊS

CodeInnova

CAKOVEC

UNIVERSITY OF DYSNAKOVÁ

Co-funded by the Erasmus+ Programme of the European Union

Índice

5º ANO

1. Jogos lógicos *1*

2.IoT A Internet das coisas *3*

3. Microbit – output Smiley *4*

4. Vamos conhecer input/output *6*

5. O gato do Scratch quer conhecer-te: micro:bit *9*

6. O que é que o meu smartwatch pode fazer? *10*

6º ANO

1. Encontrar e corrigir erros e fazer o teu programa funcionar *12*

2. Jogo do esconde-esconde *14*

3. Jogo da adição *17*

4. A magia de um número de três dígitos ao contrário *19*

5. Desenhar um quadrado no Scratch e no Python *20*

6. O meu jogo – trabalho de projeto *26*

7º Ano

1. Quem apanha primeiro o balão? Jogo *28*

2. Quem é o mais velho? Jogo *31*

3. Estruturas e tipos de dados no Python *37*

4. Strings de dados – listas *41*

5. Funções de construção e métodos de trabalho com listas *44*

6. Funções no Python *49*

8º Ano

1. O meu robô resolve o problema *56*

2. Micro:bit comunicação de rádio *57*

3. Conetar LEDs ao micro:bit *58*

4. Controlo analógico de LEDs *59*

5. Ligando os sensores *60*

6. Interface interativo – RELP *60*

9º Ano

1. Telemóvel – como surgiu e qual é o seu futuro?..... 62

2. Vou desenvolver a minha própria aplicação mobile..... 63

3. O que é o blockchain? 65

4. Como posso usar o brainstorming para os meus projetos?..... 66

5. Estou a promover a minha aplicação mobile 68

5º ANO

1. Jogos lógicos

Os objetos, os eventos e os seres têm múltiplas características que os definem. Algumas dessas características interagem com outras coisas com as quais formam uma unidade coesa.

No dia a dia, muitos obstáculos e problemas são resolvidos reconhecendo e eliminando intrusos. Por exemplo, quando vamos para a escola e verificamos o nosso horário, colocamos na mochila só as coisas de que vamos necessitar naquele dia. Se não fizermos essa seleção, podemos não colocar tudo o que precisamos na mochila e/ou a mochila ficará muito pesada com tudo o que lá colocarmos.

Os smartphones têm um ecrã sensível ao toque. Os peixes vivem na água. A queda de neve é feita de flocos de neve. Algumas características dos objetos, dos eventos e dos seres fazem com que se distingam dos outros e se tornem especiais.

Um objeto, um evento ou um ser que tenha características diferentes é considerado um intruso. Para uma melhor compreensão do exposto, precisamos de olhar atentamente para as coisas que nos rodeiam e ver as suas características.

1. Elimina o intruso.
- **a)** um carro
- **b)** um barco
- **c)** uma carrinha
- **d)** um autocarro

2. Elimina o intruso.
- **a)** um avião
- **b)** um helicóptero
- **c)** uma nave espacial
- **d)** um autocarro

3. Elimina o intruso.
- **a)** um leão
- **b)** um tigre
- **c)** uma girafa
- **d)** um puma

4. Elimina o intruso.

a) 16
b) 65
c) 36
d) 81
e) 49

5. Elimina o intruso.

a) NE
b) N
c) NO
d) SO

6. Elimina o intruso.

7. Elimina o intruso.

8. Um gato gosta do número 25, mas não gosta do número 24. Ele gosta do número 400, mas não do número 300. Ele gosta do 144, mas não do 145. De qual destes números gosta o gato?

a) 64
b) 124
c) 200
d) 169

2.IoT A Internet das coisas

Que aparelhos eletrónicos usamos?

Além de computadores, tablets, smartphones e dispositivos semelhantes – haverá outros nas casas modernas que estejam conectados à Internet?

A "Internet das Coisas" (IoT) é um conceito que se refere à conexão de dispositivos de uso quotidiano em rede. O número destes objetos digitais está a crescer anualmente e estima-se que, no ano 2025, cada um de nós tenha à sua volta mais de uma dúzia deste tipo de aparelhos.

Estes dispositivos facilitam as nossas vidas, pois, além de conectados à Internet, combinam o mais recente software com sensores sem fios.

Comparando com os objetos já em uso, os aparelhos IoT proporcionam novas opções através da Internet– isto significa que podem ser constantemente atualizados.

Alguns dos benefícios destes dispositivos são:

- Os aparelhos eletrónicos estão ligados em rede.
- Tudo na nossa casa passa a ser automático – poupa-nos tempo.
- Mais tempo significa ter mais tempo para fazer outras coisas – portanto economizamos dinheiro.
- Os dispositivos podem ser atualizados, logo duram mais.

Algumas desvantagens são:

- Quanto mais complexos os dispositivos, maior a probabilidade de falhar.
- Estes aparelhos estão mais sujeitos a ameaças de segurança.
- Não existe um padrão IoT internacional – cada fabricante faz os aparelhos segundo os seus próprios padrões.
- A poupança de tempo deixa de ser uma vantagem a partir do momento em que temos de cuidar dos próprios aparelhos.

Depois de teres aprendido o que são os dispositivos IoT, podes nomear alguns dos que tens em casa ou que tenhas visto em algum outro lado?

Alguns dos mais populares são:

Coluna inteligente Amazon Echo

Detetor de fumo com alarme

Sistema de iluminação Philips

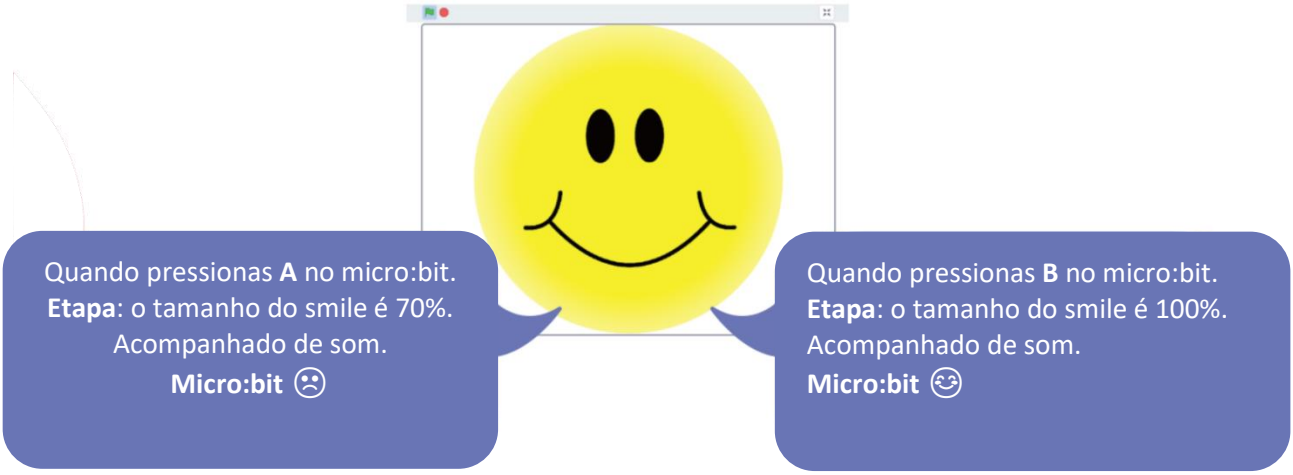
Aspirador robô Roomba

Tarefa:

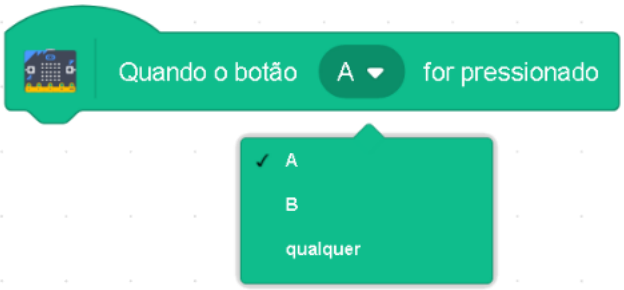
Desenha a tua casa ideal. Que dispositivos IoT usarias?

3. Microbit – output Smiley

Pressionando o botão A, o micro:bit, através de um código de computação, mostra o smile triste (output), e pressionando o botão B, mostra o smile alegre (output).



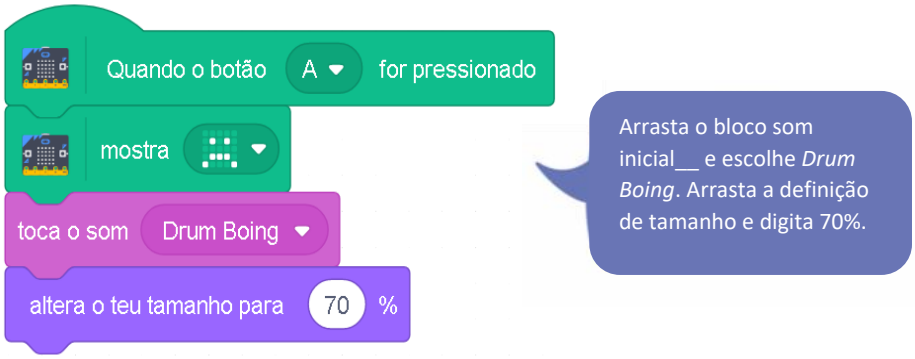
1. Grava um objeto gráfico no teu computador (cuidado com o *copyright*).
2. Arrasta o botão 'Quando __' nas categorias da toolbox, pressionando o bloco e escolhe A.



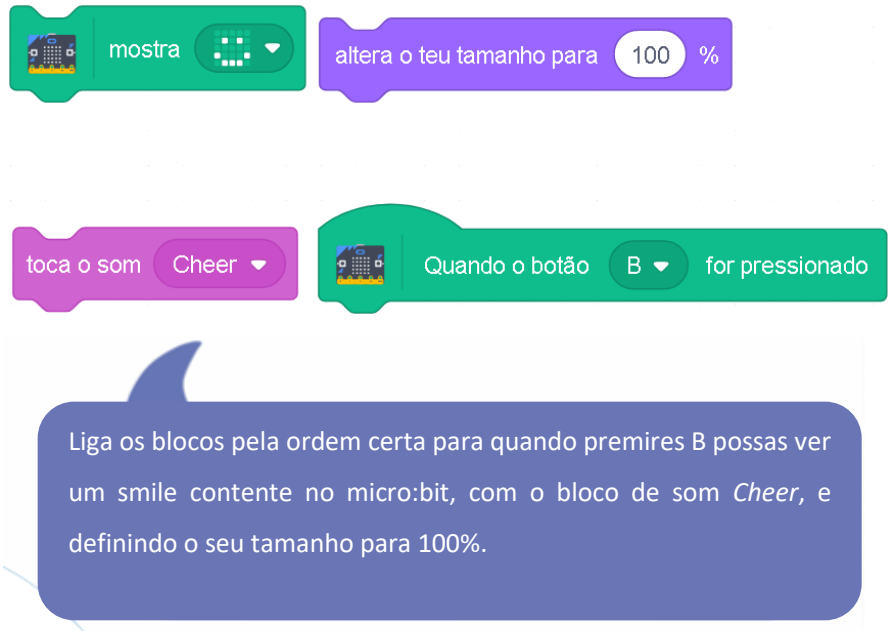
3.

Quando o botão A for pressionado
mostra
Arrasta o bloco mostra__ no ecrã e clica nas áreas do menu suspenso para mostrar o smile triste.

4.

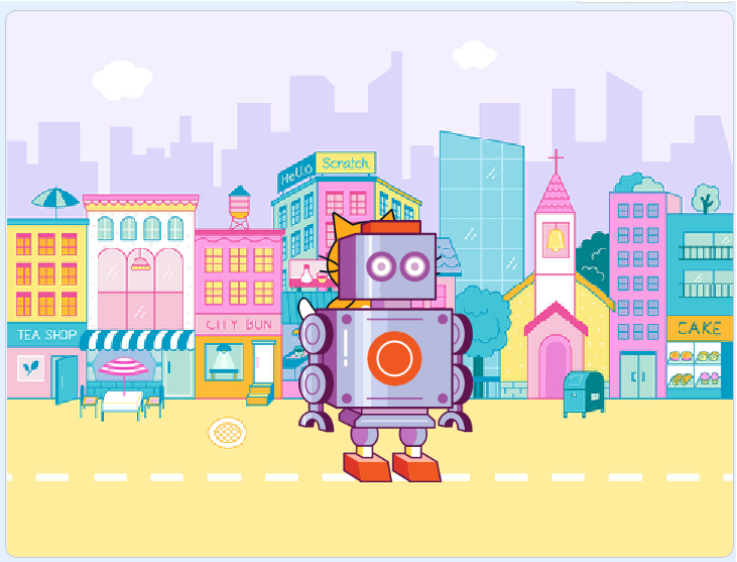

Quando o botão A for pressionado
mostra
toca o som Drum Boing
altera o teu tamanho para 70 %
Arrasta o bloco som inicial__ e escolhe *Drum Boing*. Arrasta a definição de tamanho e digita 70%.

5.

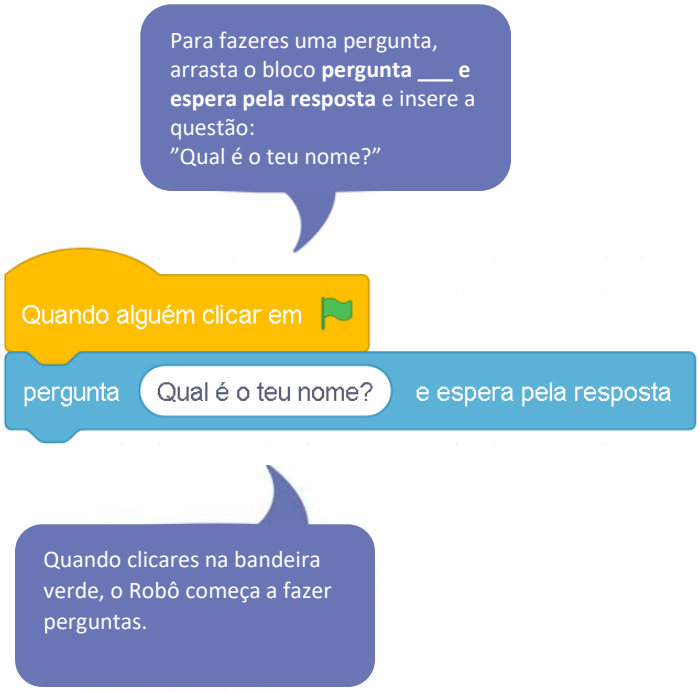

mostra
altera o teu tamanho para 100 %
toca o som Cheer
Quando o botão B for pressionado
Liga os blocos pela ordem certa para quando premires B possas ver um smile contente no micro:bit, com o bloco de som *Cheer*, e definindo o seu tamanho para 100%.

4. Vamos conhecer input/output

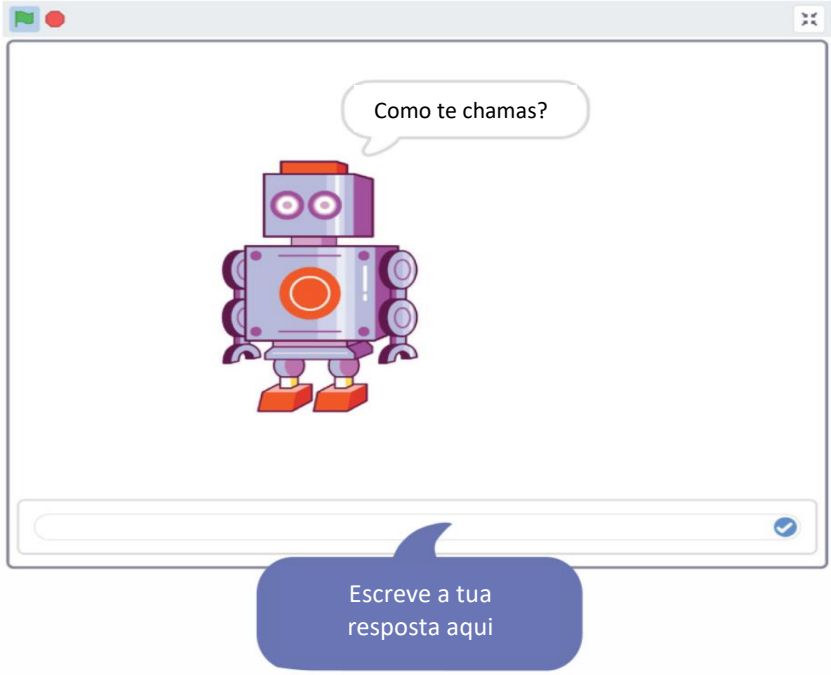
O Robô faz perguntas para te conhecer.



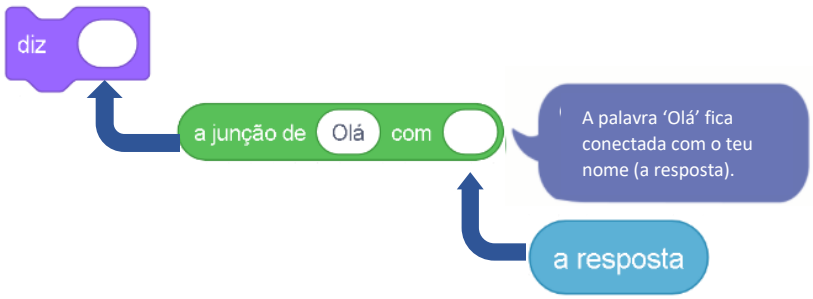
1. Abre o programa Scratch.
2. Carrega o sprite Robô e seleciona como fundo a cidade.
3. Quando clicares na bandeira verde, o Robô começa a fazer perguntas. Digita a tua resposta na caixa de texto.



O palco agora aparece como na imagem:



4. Quando digitares a tua resposta, o Robô cumprimenta-te com a palavra "Olá", juntando o nome que introduziste na resposta. Para veres isto, precisas de blocos das categorias 'Looks' (Aparência), 'Sensing' (Sensores) e 'Operators' (Operadores), conforme mostra a imagem:



5. Espera 3 segundos antes de fazeres a próxima pergunta.

EXERCÍCIO 1

Segue o exemplo abaixo e repete o procedimento para as seguintes questões e respostas:

1. **Em que ano estás?**

O Robô responde: **A tua resposta + fantástico!**

2. **Qual é o teu desporto preferido?**

O Robô responde: **A tua resposta + é também o meu preferido!**

3. **Qual é o teu jogo preferido?**

O Robô responde: **A tua resposta.**

4. **Quantos anos tens?**

O Robô responde: **Prazer em conhecer-te! Adeus!**

Solução:

EXERCÍCIO 2

Através da extensão 'Texto para fala', do Scratch, reproduz as tuas perguntas com voz sintetizada.

EXERCÍCIO 3

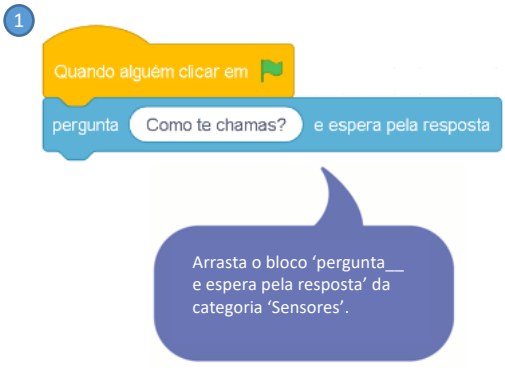
Traduz o projeto para um outro idioma, usando os blocos da extensão 'Traduzir'.

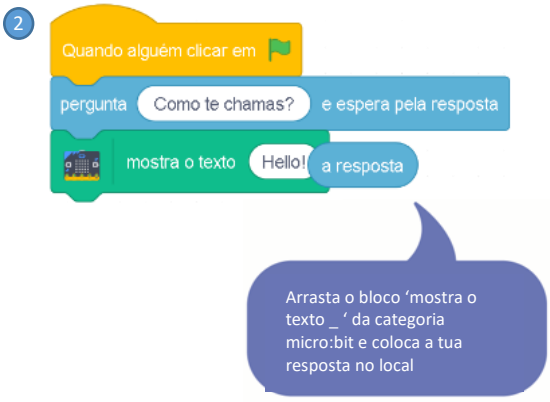
Grava o projeto no teu computador como 'Este sou eu'.

Podemos inserir blocos em caixas de outro bloco. Para pausa, usamos o bloco 'espera _ segundos'.

5. O gato do Scratch quer conhecer-te: micro:bit

Através do micro:bit, a resposta a uma questão do gato aparece no ecrã.





EXERCÍCIO 1

Faz as alterações necessárias nos blocos para que o gato pergunte várias e diferentes questões e mostre as respostas no micro:bit. Grava o projeto no teu computador.

6. O que é que o meu smartwatch pode fazer?

Durante as últimas décadas, os computadores tornaram-se cada vez mais pequenos. Apesar de usarem as mesmas componentes básicas, como o CPU, a RAM ou os discos de armazenamento – o desenvolvimento tecnológico vai disponibilizando computadores cada vez mais rápidos e de pequenas dimensões.

Pergunta: quais são os computadores que usamos hoje?

Desde os grandes computadores de secretária até aos pequenos laptops, os pequenos tablets e os ainda mais pequenos smartphones – hoje os computadores mais pequenos que usamos são os smartwatches.

Um smartwatch é um computador com as mesmas componentes de um tablet ou de um smartphone, mas foi reduzido a um tamanho pequeno o suficiente para caber num relógio.

Os smartwatches mais populares usam dois sistemas operativos: o WatchOS (Apple Watch) e o Wear OS (baseado no Android da Google, usado por muitos fabricantes). Série 6 da Apple Watch:

Um smartwach disponibiliza novas funções. As mais úteis são:

- Rastrear as tuas funções vitais.

A maior parte dos smartwatches podem rastrear os teus batimentos cardíacos, enquanto alguns, como os da Apple Watch, podem dar-te uma imagem ECG ou rastrear os níveis de oxigénio no sangue.

No caso do usuário desmaiar, o smartwatch pode automaticamente ligar a um contacto previamente definido, bem como a uma ambulância.

- Rastrear a condição física.

A par dos batimentos cardíacos, os smartwatches podem também rastrear o exercício físico diário, quanto tempo estiveste em movimento, quanto tempo estiveste parado, contar os passos, ou a distância que nadaste (a maior parte dos smartwatches é à prova de água).

- Encontrar o telefone perdido.

Quantas vezes aconteceu deixares o smartphone algures e não o conseguires encontrar? Se tiveres um smartwatch, basta clicar em „Find my phone“ (Encontra o meu telemóvel) e o smartphone começará a tocar imediatamente.

- Verificar mais discretamente as suas mensagens e notificações.

Quando estás com outras pessoas ou em grupo, olhares para o teu telemóvel pode ser considerado indelicado, porque distrai a tua atenção das pessoas com quem estás a falar. Se mesmo assim quiseres saber quem enviou uma mensagem, por exemplo, é muito fácil lançares um olhar ao teu smartwatch.

- Falar.

Se tens um smartwatch, não precisas de parar uma atividade, como cozinhar, escrever ou andar de bicicleta para pegares no telemóvel e atenderes uma chamada. Podes simplesmente responder, uma vez que o microfone é bastante potente para que possas confortavelmente falar sem teres de colocar o smartwatch perto da tua cara.

- Fazer pagamentos numa loja.

Um smartwatch pode ser facilmente conectado a uma aplicação para fazer pagamentos. Quando tens de pagar numa loja, não precisas de encontrar a carteira – basta fazeres um único clique no teu smartwatch para ativar a opção de pagamento e depois aproximá-lo do dispositivo de pagamento da loja.

Este procedimento é muito mais simples, seguro e rápido do que andar com a carteira cheia de cartões ou de dinheiro.

- Informação sobre o tempo e outras.

Sim, podes usar o smartwatch para verificar as horas, como se fosse um relógio. Mas também podes consultar a previsão meteorológica, a temperatura do ar, o volume do ambiente, o teu próximo evento no calendário e o tempo que falta até o teu almoço estar pronto.

EXERCÍCIO 1

Pesquisa os mais modernos e populares smartwatches no mercado.

Existe a opção da “melhor compra”?

Precisarias de um smartwatch? Ajudar-te-ia no dia a dia?

6º ANO

1. Encontrar e corrigir erros e fazer o teu programa funcionar

Criar os nossos primeiros programas e colocá-los a fazer aquilo que nós queremos é muito divertido, mas também pode ser frustrante se tiver erros que impedem o nosso programa de trabalhar.

E isto é especialmente frustrante se não conseguimos rapidamente encontrar o erro e assim ocupamos o tempo que poderia ser utilizado numa outra atividade.

A maior parte dos erros de um programa resume-se à sintaxe da programação – normalmente, esquecemos um parêntesis ou um apóstrofo ou então um bloco que não está corretamente agrupado.

Há formas de evitar erros na escrita de programas e alguns deles são:

- Certifica-te de que o teu código é legível.

Não é importante a linguagem de programação que estás a usar – sejam blocos através do Scratch ou código com o Python – é muito importante que o teu código seja legível para as outras pessoas.

E sim, isto também é válido para programas que não vão ser vistos por mais ninguém. Um código facilmente legível significa que os blocos estão ordenados e essa ordem tem uma lógica, ou o texto está escrito com espaçamentos claros entre as linhas, quando isso é a coisa lógica a fazer.

Se a tua programação tem partes múltiplas, deve ficar claro o que cada parte faz.

- Usa comentários sempre que possível.

Imagina que és a pessoa que vê o teu código pela primeira vez. Se houver uma parte do código que não é automaticamente entendida por outra pessoa (assumindo que ela conhece a linguagem que estás a usar) – deves optar por escrever comentários, que irão explicar o que determinada parte do código está a fazer.

- Lê programações de outras pessoas.

Se te queres tornar num melhor fotógrafo, deves olhar para as fotografias que os profissionais fazem, e se queres ser um melhor jogador de basquetebol, deves ver os jogos da NBA.

É sempre um bom princípio aprender com os profissionais para se ser melhor em qualquer atividade e a programação não é exceção.

Mantém-te online e vê como os outros estão a escrever o seu código na mesma linguagem de programação que estás a usar.

Se seguires os conselhos acima referidos e, mesmo assim, ainda tiveres uma mensagem de erro, tenta uma das seguintes sugestões:

- Vê o que o teu programa diz sobre o erro.

Independentemente da programação que estejas a usar, o programa vai muito provavelmente dizer-te onde está o erro. Isto significa que irá marcar graficamente essa parte do programa, ou dizer-te em que linha do código aconteceu o erro, ou referir qual o comando ou variável com problemas. Antes de mudares o que quer que seja, concentra-te em alterar aquilo que o teu programa te disse para mudares.

- Usa o Google.

Quando não consegues resolver o erro por ti próprio, é muito provável que seja um problema comum e que não sejas a primeira pessoa a tê-lo. Com uma rápida pesquisa no Google sobre esse erro, poderás encontrar um texto onde alguém explica como também o encontrou e, mais importante, como o resolveu.

- Pede a alguém que veja a tua programação.

Frequentemente, os programadores estão tão envolvidos com os detalhes da sua programação que não conseguem vê-la como um todo. Às vezes encontramos um erro que facilmente identificaríamos, mas tal é difícil de conseguir depois de se passar horas a programar. Nestes casos, será útil pedir a alguém que veja a nossa programação.

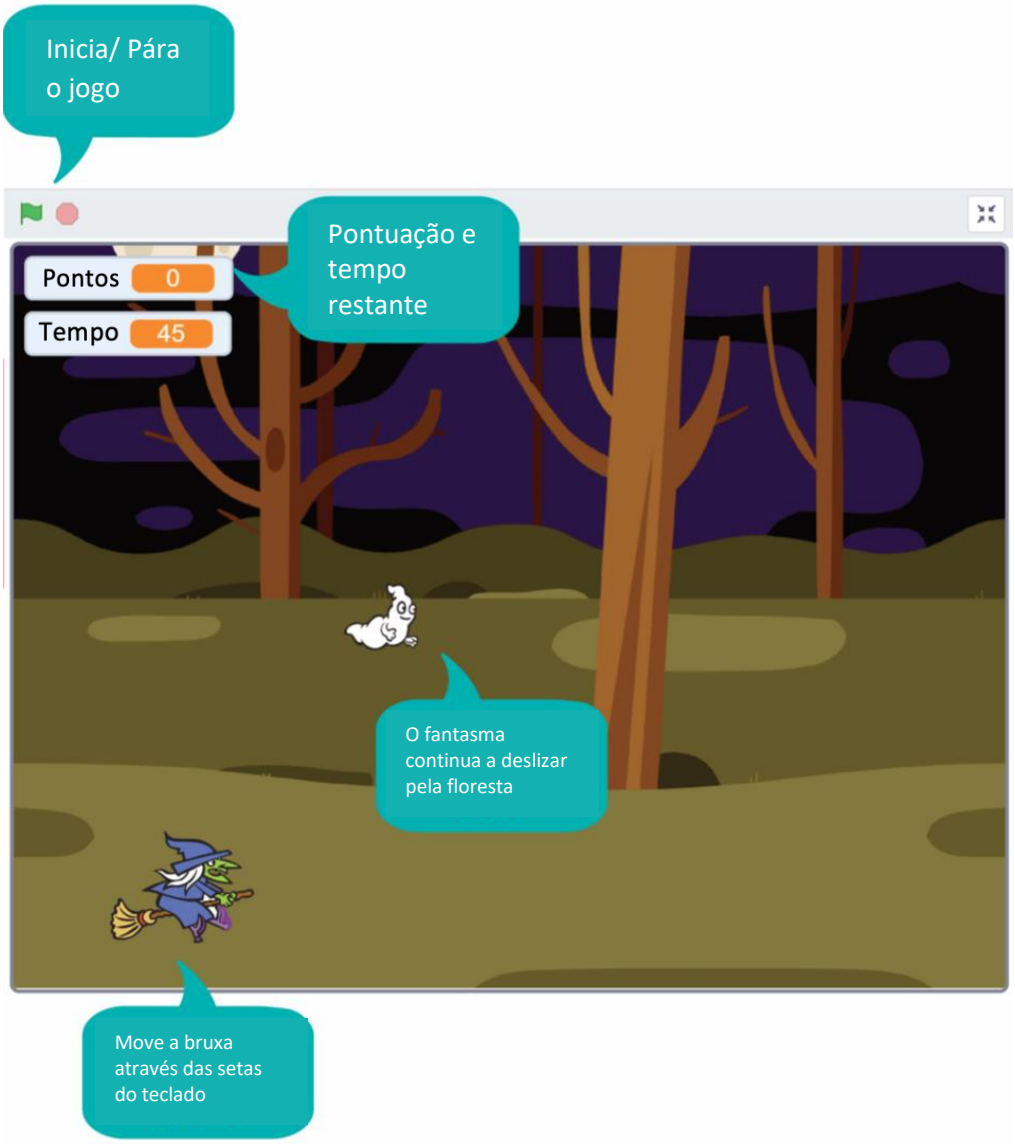
- Faz uma pausa.

Se nada disto resultar – desiste da programação, mas só para uma pausa.

Afasta-te do computador, come qualquer coisa, vê um episódio do teu programa ou série preferidos, vai dar uma volta ou faz uma sesta. O erro ainda lá estará quando regressares, mas o teu cérebro estará à procura de uma nova solução e é bem possível que surja algo que ainda não tenhas tentado.

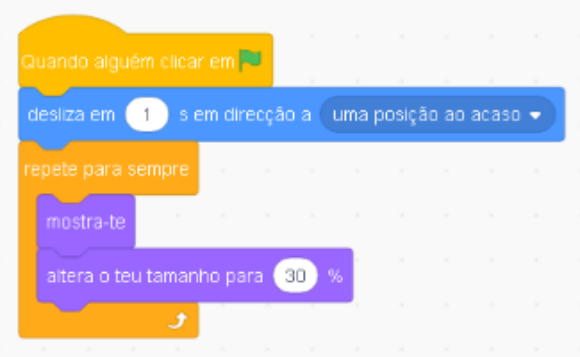
2. Jogo do esconde-esconde

Este jogo consiste em mover a bruxa, usando as setas do teclado, para apanhar o fantasma. Cada vez que a bruxa toca no fantasma, o jogador ganha um ponto. O fantasma irá deslizar em posições aleatórias. Quando a bruxa tocar no fantasma, ouve-se o som 'pop' e o fantasma esconde-se. Reaparece ao fim de dois segundos numa nova posição aleatória. Define o tempo do jogo, 45 segundos, e um tamanho apropriado para o objeto gráfico. Para criares um ambiente mais aprazível, adiciona uma melodia "Dance slow mo" (suave).



1. Quando clicares na bandeira verde, a bruxa move-se sempre na direção da seta pressionada no teclado. Ajusta o tamanho da bruxa. Liga os seguintes blocos:

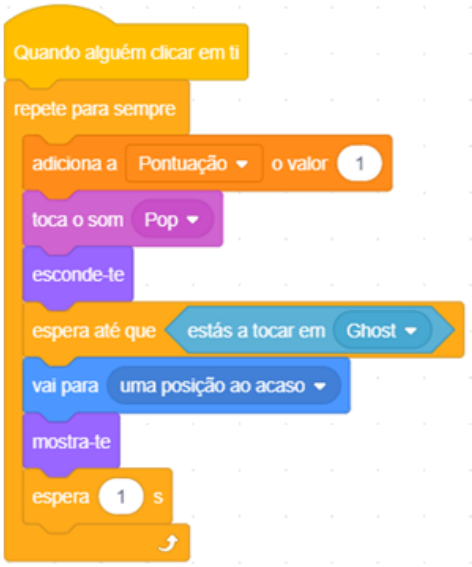
2. Quando clicas na bandeira verde, o fantasma aparece e desliza por um segundo numa 'random position' (posição aleatória). Ajusta o tamanho do fantasma (set size to_). Corrige o bloco do script!



3. Quando clicas na bandeira verde, os pontos estão no 0. Dará início a uma contagem decrescente de 45 segundos. Liga os blocos para a bruxa.

4. A música Dance slow Mo deve estar a tocar a 70% do volume. Corrige os blocos e adiciona-os à bruxa!

5. Sempre que a bruxa tocar no fantasma, ouve-se o som 'pop' e o jogador ganha um ponto. O fantasma desaparece por um segundo e reaparece numa posição aleatória. Corrige os blocos!



6. No início do jogo, coloca a bruxa no canto inferior direito do ecrã. Bom trabalho! Diverte-te!

EXERCÍCIO 1

Apimenta o jogo juntando um fantasma mau. Cada vez que toca na bruxa, ela perde um ponto!

EXERCÍCIO 2

Adiciona um feiticeiro e deixa-o disputar com a bruxa quem irá ganhar mais pontos.

3. Jogo da adição

Neste jogo, o esquilo inteligente quer saber quantas adições consegues resolver corretamente em 45 segundos. O esquilo pede-te para adicionares dois números aleatórios do 1 ao 99. Cada vez que responderes corretamente, ouvirás o som Big Boing e obterás um ponto. Se a tua resposta estiver incorreta, ouvirás o som de uma trombeta e perdes um ponto. Não desiludas o esquilo!

1. Quando clicas na bandeira verde, o cronómetro fica definido para 45 segundos e os pontos no 0. O esquilo diz: 'Mostra-me quão rápido consegues adicionar números durante 3 segundos' e depois "Três, dois, um!", durando um segundo.
2. A tarefa é repetida até o cronómetro chegar a 0 segundos.
3. Os números de 1 a 99 são selecionados aleatoriamente para a primeira e segunda adição. A soma dos dois números serão guardados para serem comparados com a tua resposta.

4. Se a tua resposta estiver incorreta, a mesma tarefa ser-te-á dada até responderes corretamente.
5. Se a tua resposta estiver correta, ganhas um ponto e ouves o som Big Boing; caso contrário, ouves o som da trombeta e perdes um ponto.

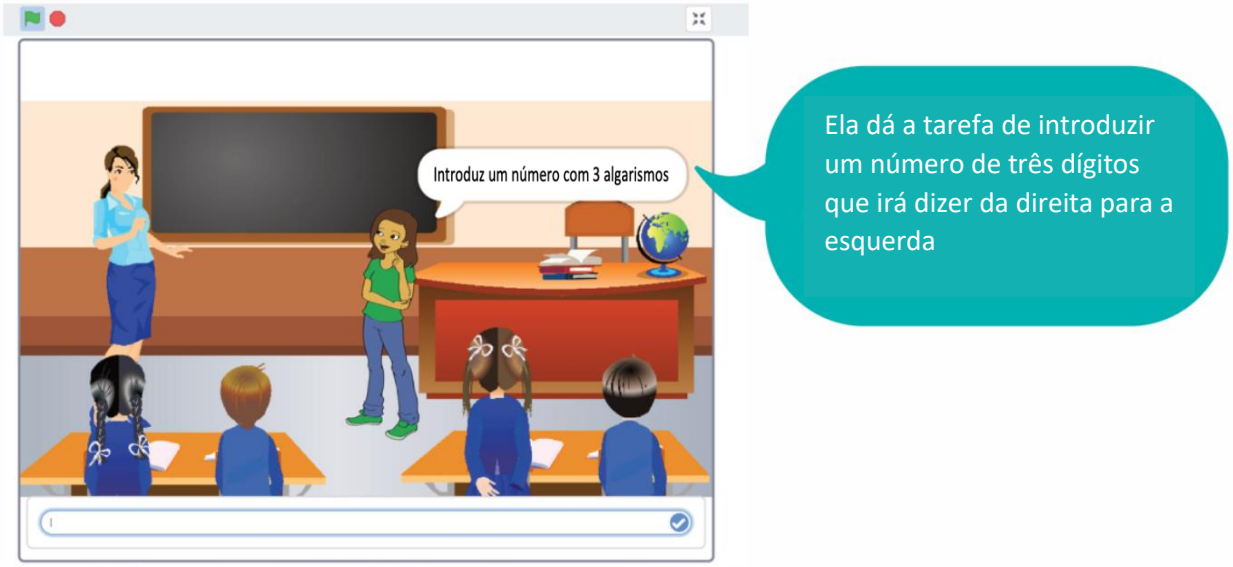
   Neste jogo, precisas dos seguintes blocos:

EXERCÍCIO 1

Altera o jogo para, em vez da adição, verificares a tabuada!

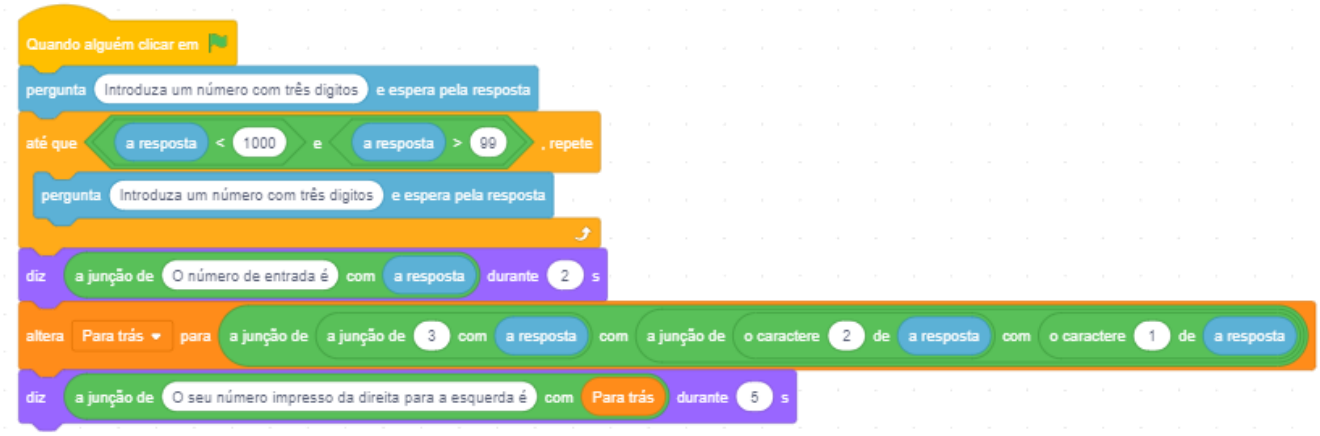
4. A magia de um número de três dígitos ao contrário

A Dee tem um supertalento. Tu digitas um número de três dígitos e ela di-lo ao contrário. Descobre o seu segredo! Ela dá-te uma tarefa para introduzires um número de três dígitos, que ela depois registará da direita para a esquerda.



1. A Dee dá-te uma tarefa para introduzires um número de três dígitos, e repete-a até introduzires corretam Introduza um número com três dígitos ero dos dígitos.
2. A Dee diz: 'O número introduzido é _____' por 4 segundos.
3. A Dee fala por dez segundos: 'O teu número, da direita para a esquerda, é _____'.

Corrige os blocos no script:



EXERCÍCIO 1

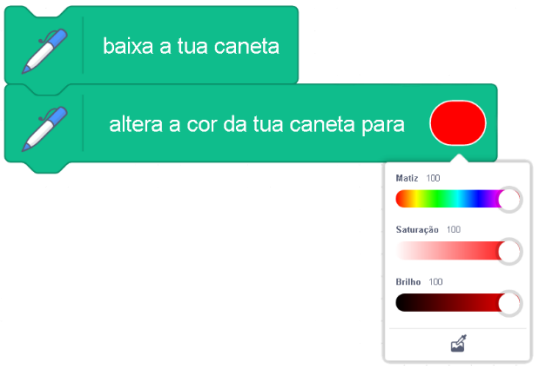
Muda os blocos no script para que a Dee possa fazer o mesmo com um número de cinco dígitos!

5. Desenhar um quadrado no Scratch e no Python

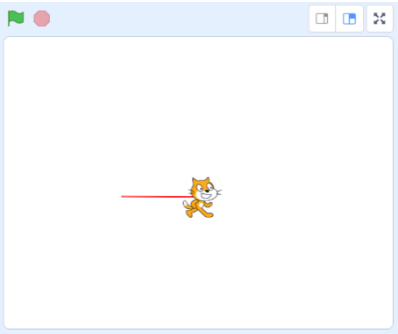
a) Scratch

Usando o bloco da categoria Pen (caneta), das extensões do Scratch, aprendes a desenhar quadrados.

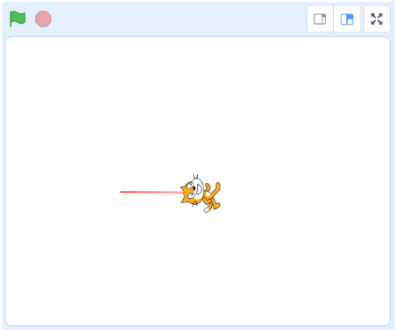
1. Abre um novo projeto.
2. Mostra o bloco da categoria Pen (caneta), das extensões do Scratch.
3. Para que a caneta desenhe, deves agrupar o bloco Pen down.
4. Define a cor vermelha para a caneta.



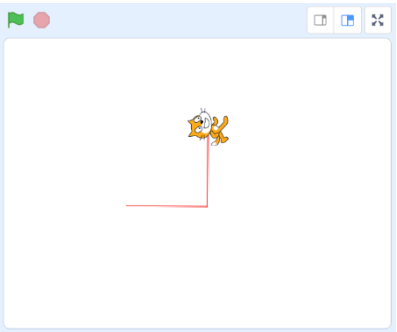
5. O gato desenha o primeiro lado, como se vê na figura.



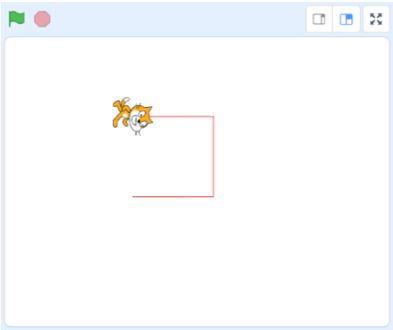
6. O gato gira para a esquerda 90 graus.



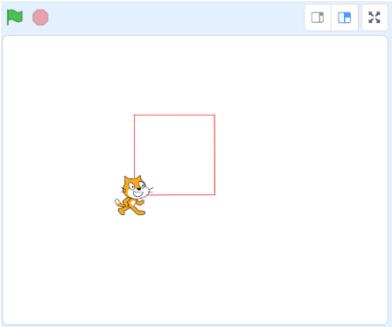
7. Adiciona blocos para desenhar um quadrado.



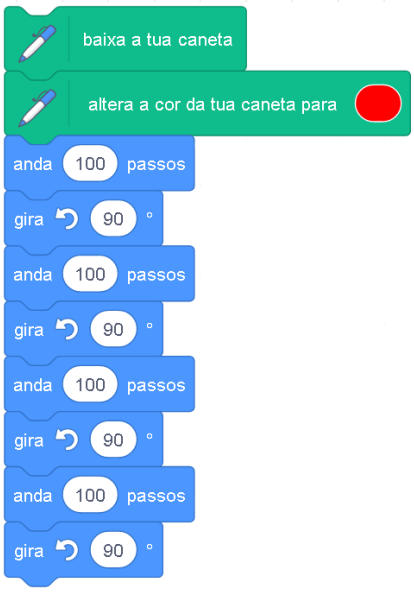
8.



9.

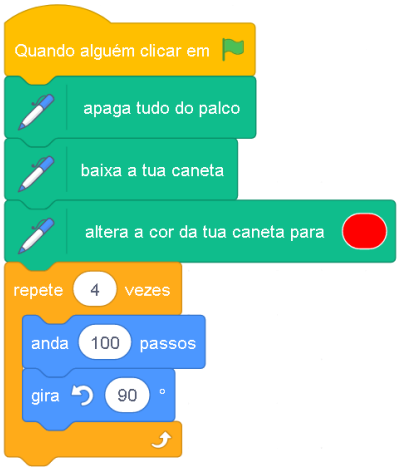


10. Olha para o script dos blocos. O que vês?



11. Utiliza o bloco 'repeat___' para que o desenho da linha e a rotatividade sejam repetidas.

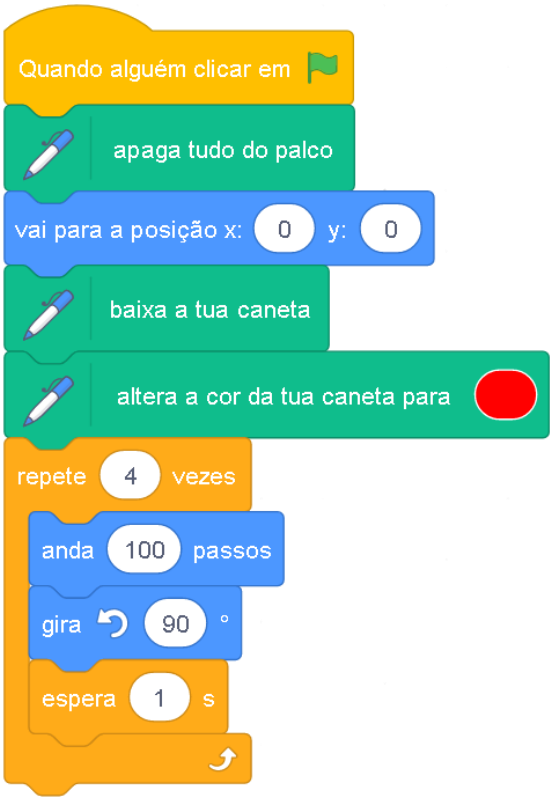
12. O projeto deve começar quando clicas na bandeira verde. No início, tudo é apagado da tela.



13. O objeto digital deve sempre começar a desenhar no mesmo ponto. Define-o no ponto inicial (0,0). Para veres claramente o que acontece, tens de adicionar o bloco ‘wait_ secs’ (espera ___ segundos) depois de rodar para a esquerda.

Grava o projeto no computador com o nome “Quadrado”.

Solução:



Usando o bloco ‘repeat _’ (repetir __), neste caso quatro vezes, o desenho do quadrado é feito em pouco tempo.

EXERCÍCIO 1

Muda o guião para que cada linha no quadrado tenha uma cor diferente.

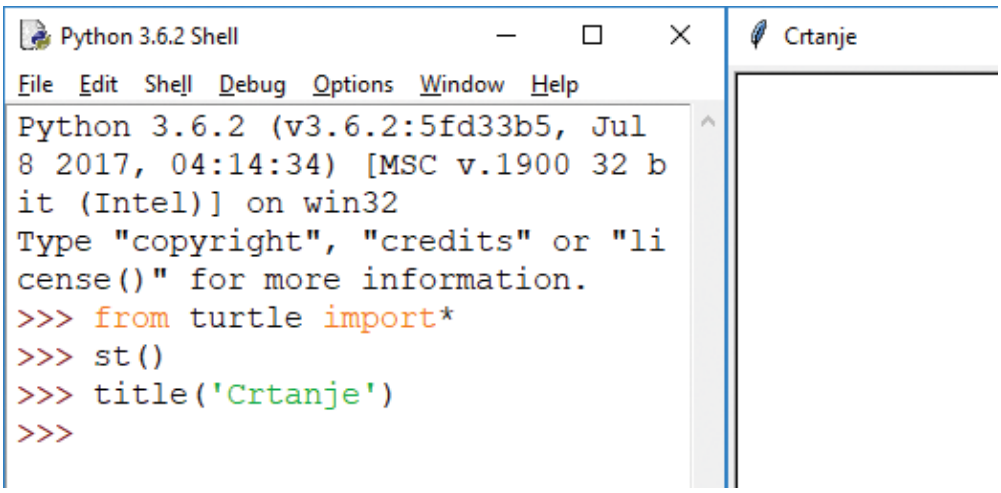
EXERCÍCIO 2

Altera o guião para que seja desenhado um quadrado com o comprimento lateral de 65, depois com 50 e, finalmente, um quadrado com o comprimento lateral de 120.

b) Python

O Python, bem como outras linguagens de programação, como o Logo, torna possível desenhar com gráficos tartaruga. A ideia básica destes gráficos é desenhar usando comandos que fazem com que um objeto como uma tartaruga ou uma seta se movam pelo ecrã. A "turtle" é um módulo da Python para desenhar utilizando a lógica da programação e que torna possível a utilização de gráficos. Para se utilizarem todas as suas funções, o módulo é ativado através do comando from turtle import*.

A janela do desenho será mostrada através da inserção no teclado de qualquer comando gráfico e de uma interface de utilizador interativa. Vamos começar por mostrar a caneta com o comando **st()**, e depois daremos um nome à janela com o comando **title()**.



Comandos básicos para mover a caneta

Os comandos básicos na Python utilizam palavras da língua inglesa e podem ser escritos de forma abreviada. A maior parte dos comandos contêm um valor numérico, que serão escritos entre parêntesis. Usando os comandos de movimento, a caneta pode ser:

• movida num plano para um valor numérico específico (em dots ou pixels).

• rodada num determinado valor em graus (para a esquerda ou para a direita).

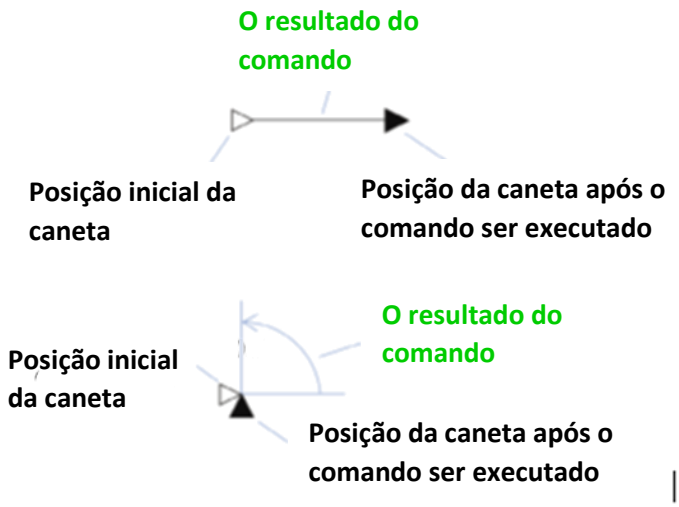
A caneta é sempre movida da sua posição inicial no ecrã. Quando se move, deixa um rasto e assim temos as linhas.

Os comandos básicos para desenho estão descritos na tabela abaixo.

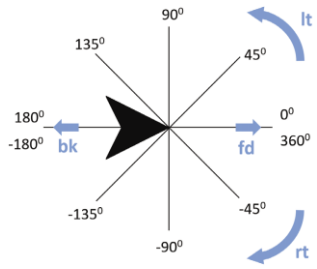
Comando	Abreviatura	Descrição
forward(a)	fd(a)	Move a caneta um passo para a frente
backward(a)	bk(a)	Move a caneta um passo para trás
right(angle)	rt(angle)	Roda a caneta num determinado valor em graus para a direita
left(angle)	lt(angle)	Roda a caneta num determinado valor em graus para a esquerda
penup()	pu()	Levanta a caneta e para de deixar rasto (linha)
pendown()	pd()	Baixa a caneta e continua a deixar rasto (linha)
undo()		Reverte o comando anterior, pode ser usado várias vezes
showturtle()	st()	A caneta fica visível na interface gráfica
hideturtle()	ht()	A caneta fica invisível na interface gráfica
home()		Coloca a caneta na posição inicial
reset()		Coloca a caneta na posição inicial e apaga tudo
clear()		Apaga tudo, mas a caneta não se move

Depois de teclar o comando na interface, e confirmando-o clicando Enter, o comando é executado na janela. Como desenhamos algo? Vejamos um exemplo simples usando o comando fd() e o comando lt().

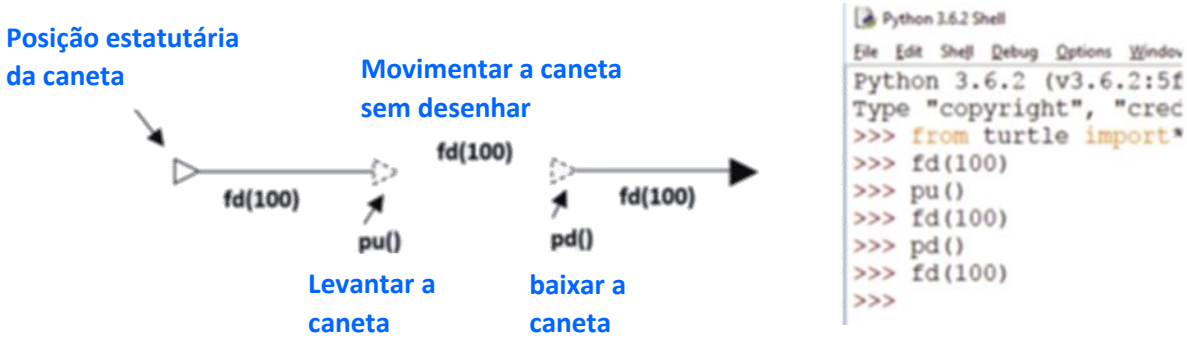
Comando fd(100) + Comando lt(90)



Enquanto roda para a direita ou para a esquerda, a tartaruga (com a forma de seta) não se move para a frente nem para trás. As formas como a podemos rodar está visível na imagem.

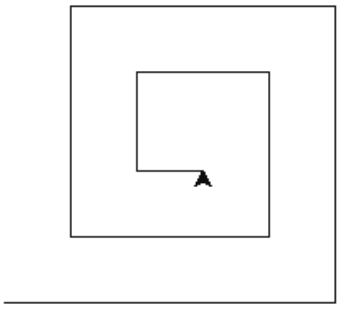


Às vezes, enquanto desenhamos, não queremos que a tartaruga deixe um rasto. Nesses casos, a tartaruga deve ser levantada usando para tal o comando pu(). Depois de levantar a tartaruga, temos de voltar a pousá-la se quisermos voltar a desenhar, usando para tal o comando pd().



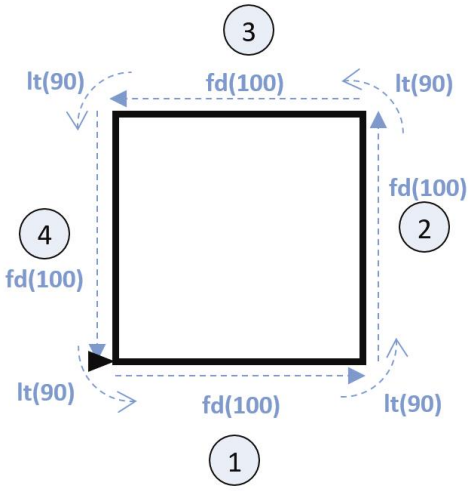
Os comandos podem ser escritos numa única linha, usando o ponto e vírgula ; . Ao escrever o código desta forma, fica mais claro e fácil de ler.

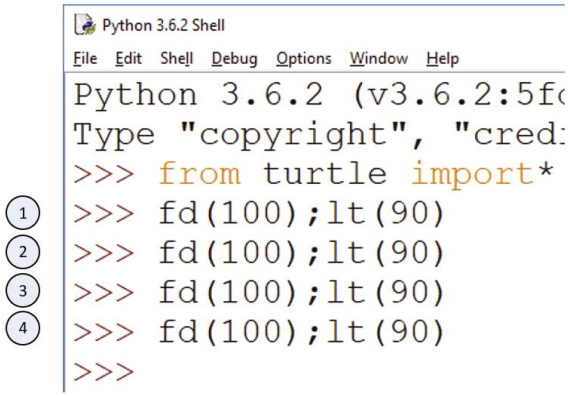
```
>>> from turtle import*
>>> title('Crtanje')
>>> fd(200);lt(90)
>>> fd(180);lt(90)
>>> fd(160);lt(90)
>>> fd(140);lt(90)
>>> fd(120);lt(90)
>>> fd(100);lt(90)
>>> fd(80);lt(90)
>>> fd(60);lt(90)
>>> fd(40);lt(90)
```



Desenhar um quadrado

Ao desenhar formas geométricas, como um retângulo ou um quadrado, a caneta roda 360°. Tem de terminar onde começou a desenhar. Um quadrado é uma forma geométrica em que todos os lados são iguais, bem como todos os ângulos. Como todos os ângulos são de 90°, as rotações da nossa tartaruga serão também de 90°. Para desenhar um quadrado, é melhor começar colocando a tartaruga apontada para a frente.





6. O meu jogo – trabalho de projeto

Se queremos programar um jogo – temos primeiro de compreender o que é um jogo. Um jogo que estamos a programar num computador ou em qualquer outro dispositivo eletrónico é um programa como qualquer outro, mas tem de atender a duas condições específicas:

- Tem de haver um jogador que consegue controlar parte do programa (jogo).
- O jogador tem de ter um objetivo que queira atingir com o programa (jogo).

Qualquer programação que responda a estas duas condições pode chamar-se jogo, como também qualquer jogo que exista certamente atende a estas duas condições.

Não interessa se estamos a falar de Fortnite ou Apex Legends ou CS:GO ou ainda Gran Turismo.

Qualquer jogo tem uma programação como qualquer outro, mas também acolhe as duas condições mencionadas acima.

Como podemos fazer o nosso próprio jogo? Facilmente, seguindo alguns passos:

1. Faz um plano.

Enquanto iniciantes, temos de decidir como queremos o nosso jogo. Para que se torne mais fácil, podemos usar estas sugestões:

- Escolhe um tema para o teu jogo.
- Decide o que o teu jogador deve fazer no jogo.
- Escolhe o que o jogador tem de fazer para ganhar ou avançar no jogo.

Podes escolher o teu tema definindo o tipo do jogo (i.e. se é um jogo de corrida, de estratégia, de tabuleiro, etc.), e também precisas de escolher uma configuração.

O jogo decorrerá durante o dia ou durante a noite? Num espaço fechado ou num aberto? Na terra ou no espaço?

Depois de definires tudo isto, podes passar para o próximo passo:

2. Desenha o teu jogador.

Se, por exemplo, estás a usar o Scratch, tens de desenhar um objeto gráfico para cada um dos teus jogadores. Estes objetos são imagens que podem aparecer na parte principal do programa e também podem ser movidas, dependendo do código.

No canto direito do Scratch há um botão chamado „Choose a Sprite“ (escolhe o objeto), e depois de escolheres o teu jogador podes configurá-lo com detalhe.

3. Escolhe um fundo para o teu jogo e – código.

Depois de usares o botão „New Sprite“, para escolheres um fundo para o teu jogo, tens de definir o que o teu jogador fará, e depois programá-lo.

Os jogos típicos de Scratch incluem aqueles em que o jogador tem de responder a perguntas, recolher objetos, ou fugir dos outros objetos.

Para começares o código, escolhe o bloco „when green flag is clicked“ (quando a bandeira verde é acionada), que significa que o jogo vai começar quando o jogador clicar na bandeira verde.

Depois disto, podes usar todo o conhecimento que adquiriste no Scratch – e a tua imaginação, para criar qualquer jogo que desejes.

Para poderes desenvolver a tua criatividade, dá uma olhadela no que outros criaram, como por exemplo alguns jogos simples feitos no Scratch:

https://scratch.mit.edu/projects/22469560/editor/

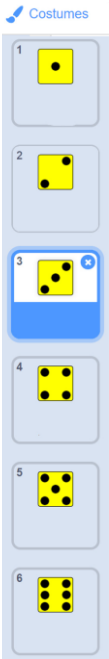
7º Ano

1. Quem apanha primeiro o balão? Jogo

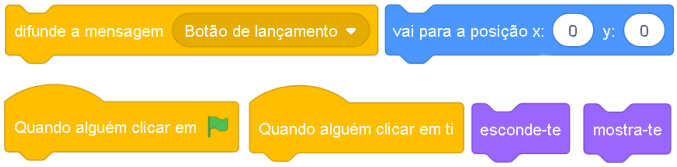
Os Sprite são objetos gráficos utilizados comummente em jogos, que podem ser criados ou importados. Possuem características que descrevem a sua localização, movimento e visual.

Neste jogo, os sprites do 1 ao 6 competem para ver quem apanha primeiro o balão. O primeiro a chegar ao balão é o vencedor. O balão está sempre em movimento, deslizando aleatoriamente com o valor de X. Quando clicas em lançar o dado, este roda e mostra um número. O sprite com esse número move-se aleatoriamente por várias etapas.

1. Grava o fundo (cenário) e as personagens sprites. Dá o nome aos sprites: sprite1, sprite2, sprite3, sprite4, sprite5 e sprite6. Desenha um sprite ‘Lança o dado’.
2. Desenha um dado com seis faces. Cada face mostra um número do 1 ao 6.



3. Desenha um objeto (sprite) para clicares quando iniciares o jogo. Esse objeto dirá que precisas de clicar no sprite ‘Lança o dado’, que aparecerá no próprio cenário depois de clicares na bandeira verde. Quando clicas neste sprite, ele reenvia uma mensagem ao sprite ‘Lança o dado' para que apareça no cenário e não seja mais mostrado. Esta mensagem chama-se ’Botão de lançamento’. Liga os blocos:



4. O sprite ‘Lança o dado’ aparece em (200, -150) depois de receber a mensagem do sprite ‘Botão de lançamento’ no passo 3. Quando clicamos neste botão, ele define no dado um número aleatório de 1 a 6 e envia uma mensagem ao dado. Chamamos a mensagem „Lançando”. Liga os blocos:

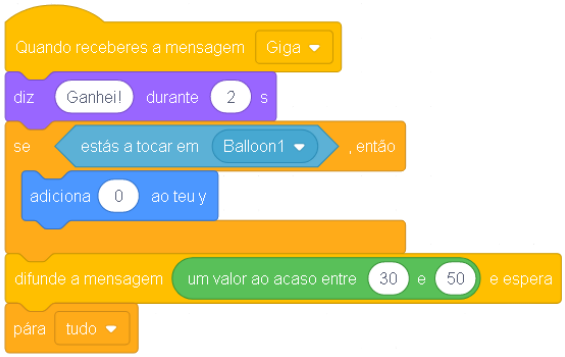
5. Quando recebe a mensagem do “Botão de lançamento”, o dado aparece em (200, -110) e aponta na direção 90°. Liga os blocos.
6. Quando recebe a mensagem, o dado é mostrado em (200, -110) e aponta na direção 90°. O dado repete a volta entre cinco a 15 vezes entre -45° and 45° e muda as faces do 1 ao 6. Depois espera 0.1 segundos. Seguidamente, o dado define a face correspondente ao valor variável. Se o número variável no dado for igual a 1, a mensagem segue para o sprite1 e espera. O mesmo acontece com os sprites de 2 a 6.

   Corrige o script e junta os blocos que faltam:

7. Dispõe os sprites, do sprite1 ao sprite6, no cenário de fundo. Quando clicamos na bandeira verde, eles devem estar colocados nessa posição. Por exemplo, definimos o sprite1 em (-200, -140). Certifica-te que o valor Y é igual em todos os sprites.

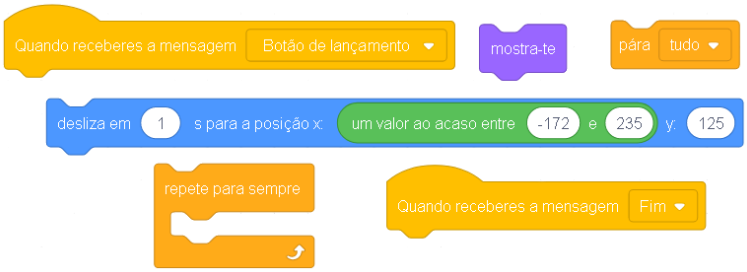
8. Quando o sprite1 recebe a mensagem ‘sprite1’, muda o valor Y pelo número de passos entre 30 e 50. Se ele toca no balão, ele diz “Eu sou o vencedor” durante dois segundos, manda a mensagem „fim”, e tudo para. Faz o mesmo para os sprites 2 a 6.

Corrige os blocos:



9. Insere o objeto balão. Quando ele recebe a mensagem „Botão de lançamento”, o balão é mostrado e desliza segundo um valor aleatório de X desde -172 to 235, e de Y de 125. Quando recebe a mensagem „fim”, tudo para.

Liga os blocos:



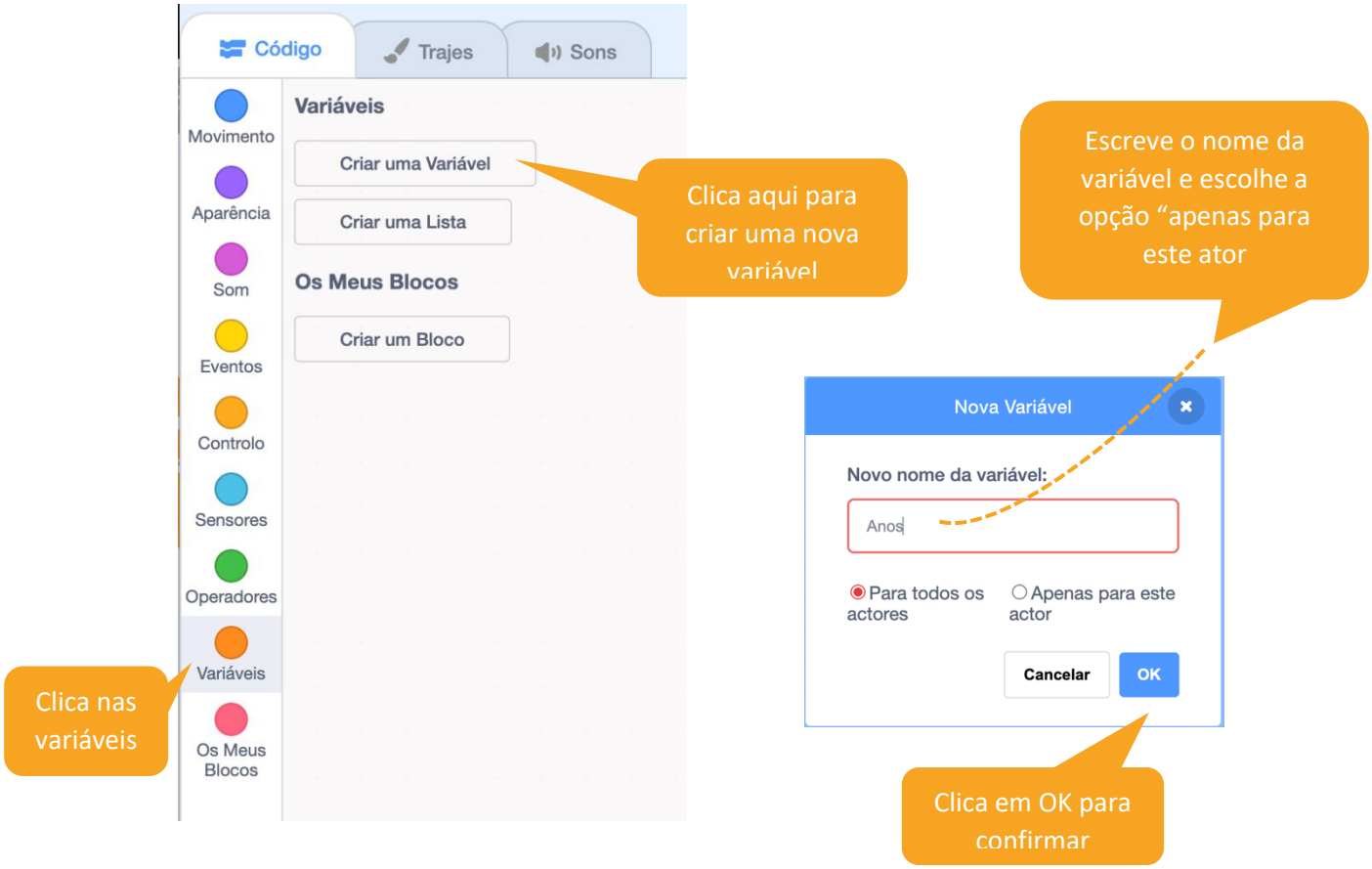
2. Quem é o mais velho? Jogo

O Robô agora sabe que idade tens e quer comparar a sua idade com a tua. Ele irá guardar a sua idade numa variável para a comparar com a tua idade e verificar quem é mais velho ou mais novo.

1. Carrega o projeto “This is me” do teu computador.

A variável é um espaço onde se armazena informação. É usada para recordar informação como pontos, nomes ou velocidades.

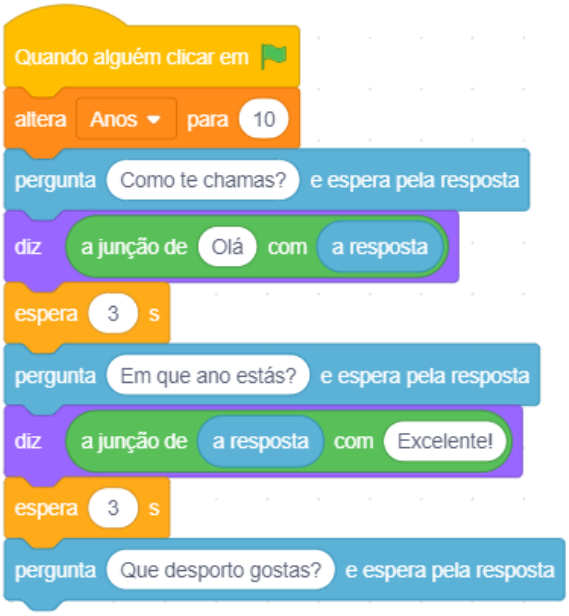
2. Deves lembrar-te da idade do Robô, uma vez que irás fazer uma variável de Anos e definir o seu valor para 10, porque o Robô tem 10 anos de idade.



Nota: desliga o mostrador das variáveis no ecrã.

Define o valor da variável para 10 e junta o bloco ao projeto como é mostrado na imagem.

Digita o nome da variável e define que só está válida para este sprite.



Compara os seguintes números e introduz um caractere em cada caixa que mostre uma relação entre cada par de números.

7 ☐ 10 12 ☐ 10 10 ☐ 10

Da mesma maneira, o Robô irá verificar se tens menos <, mais > ou os mesmos = anos. A sua resposta a essa questão dependerá disso. Isto significa que o programa deve DECIDIR quando ele é mais novo, mais velho ou se têm a mesma idade.

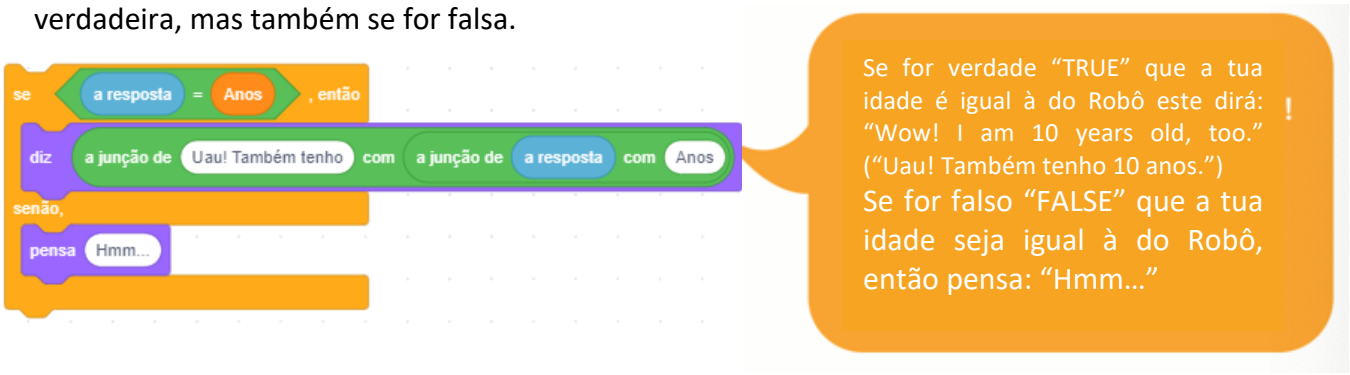
Para comparar os **números,** usa blocos da categoria Operadores:

O bloco

permite-nos decidir o que acontecerá a seguir. Quando usamos esse bloco, inserimos nele outros blocos. Os blocos que inserimos no bloco ”**if __ then**” (“Se__então) só serão executados se a resposta da operação dada no bloco estiver correta.

O bloco

decide o que acontecerá a seguir se a resposta à operação for verdadeira, mas também se for falsa.



Vamos agora comparar a nossa resposta com a idade do Robô e decidir o que vai acontecer no programa.

Se a resposta = 10 **então**

o Robô responde: “Fixe, eu tenho **resposta** anos, também!”

ou

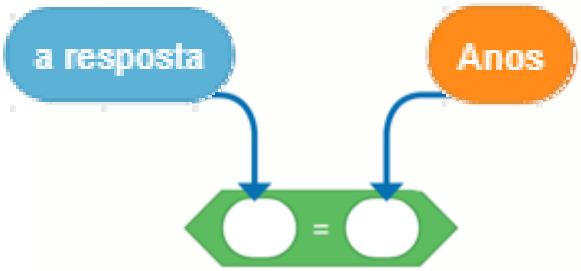
Se a resposta < 10 **então**

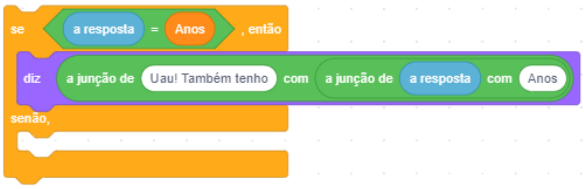
o Robô responde: “Sou mais velho que tu. Eu tenho **resposta** anos.”

ou

o Robô responde: “Sou mais novo do que tu. Eu tenho **resposta** anos.”

Agora junta a resposta à pergunta “Quantos anos tens?”. Usa blocos das categorias Control (controlo), Operators (operadores) e Sensing (sensoriamento).





Comparamos se a resposta é igual ao valor da variável Anos, que é definido como 10.

Se a resposta = 10 **então**

o Robô responde: “Fixe, eu também tenho **resposta** anos!”

Se a tua idade for igual à do Robô, então diz:

“Uau! Eu tenho 10 anos, também.”

Se a tua resposta não for 10, então deves comparar se é menor que aquele número (menos de 10). Mais uma vez, usa o bloco da categoria Sensing (sensoriamento) para comparar números e juntar a **resposta** e a variável dos **Anos**.

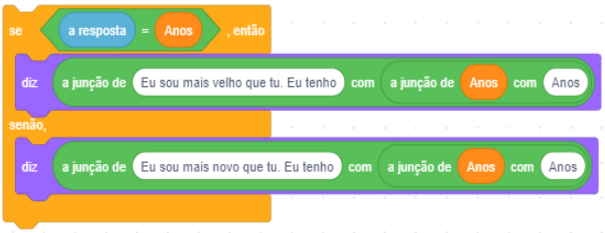
Compara os números e decide o que fazer se for verdade que o número é menor, e se for falso.

Se a resposta < 10 **então**

o Robô responde: “Sou mais velho que tu. Eu tenho **resposta** anos.”

ou

o Robô responde: “Sou mais novo do que tu. Eu tenho **resposta** anos.”



Se a tua idade é MENOR DO QUE A DE Robô, ele então diz:

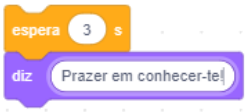
“Eu sou mais velho do que tu. Eu tenho 10 anos.”

Se a tua idade é MAIOR DO QUE A DE Robô, ele então diz:

“Eu sou mais novo do que tu. Eu tenho 10 anos.”

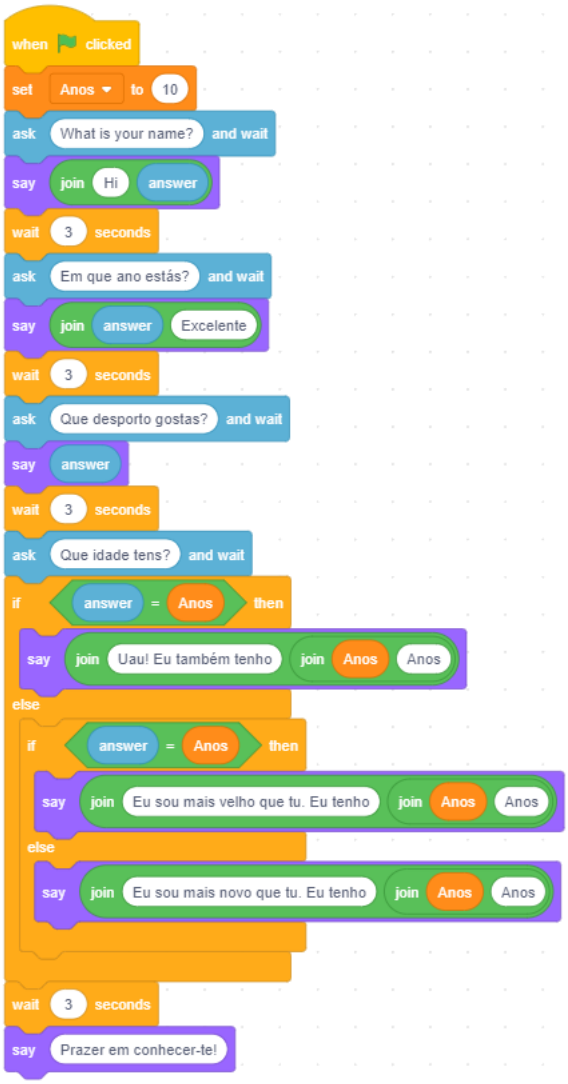
Agrupa os blocos do passo anterior ao bloco **if__then__else**, que vai verificar se a tua idade é igual à do Robô.

Liga os blocos para perguntar “How old are you?” (que idade tens) e espera. Depois junta o bloco “wait 3 seconds” (espera 3 segundos) e diz: “Nice meeting you!” (prazer em conhecer-te).



Bom trabalho! Verifica se o teu script tem a seguinte sequência:

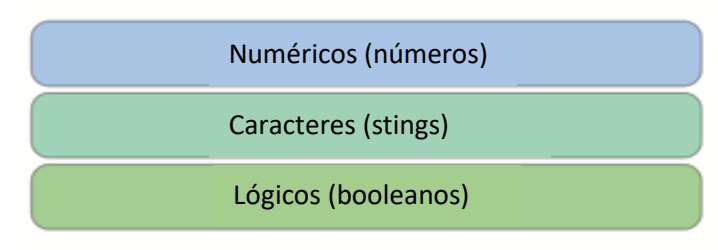
Grava o projeto com o nome “Years”.



3. Estruturas e tipos de dados no Python

Enquanto resolvemos problemas no nosso computador, criamos programas que usam diferentes variáveis de entrada. Entende-se por variável uma área de memória onde armazenamos um valor qualquer. Portanto, enquanto se projeta uma solução, um dos primeiros passos é rever essas mesmas variáveis. Vamos conhecer diferentes tipos de dados usados no Python.

Nos nossos primeiros programas de computador no Python, usamos dados numéricos e de texto. Os dados de texto foram escritos usando apóstrofos ('), que delimitam os "strings" (palavras ou cadeias de caracteres). Os tipos de dados usados nas soluções do computador são importantes na aplicação de comandos adequados de entrada e saída de dados, bem como nas diferentes funções que adicionalmente ajustam os dados. No Python há três tipos diferentes de dados simples:



Dados de tipo numérico

Ao começar a aprender a codificar e a arranjar novas soluções no computador com a linguagem Python, ficamos a conhecer diferentes tipos de dados como um número inteiro, caracteres e valores lógicos. Também ficamos a conhecer comandos para a entrada e saída desses valores usando um programa de computador. No entanto, os números que utilizamos no dia a dia não têm de ser inteiros – por exemplo podem ser decimais. Vamos então ver como podemos usar números decimais no Python. Os números decimais são escritos como em matemática, como dois números inteiros separados por um ponto, por exemplo 4.56.

```
>>> 10
10
>>> - 10
-10
>>>
```

```
>>> 4.56
4.56
>>> .25
0.25
>>>
```

Quais as operações que podem ser usadas com números decimais?

A capacidade de separar números decimais e os inteiros de que fazem parte números decimais é muito interessante na programação e, por essa razão, muitas linguagens de programação têm uma função incorporada.

Como se converte um número decimal num número inteiro?

Vamos ver alguns exemplos.

```
>>> int(4.2)
4
>>> int(4.8)
4
>>>
```

Função **int()** converte um número decimal num número inteiro ignorando a parte decimal do número e deixando apenas o número inteiro.

```
>>> float(5)
5.0
>>>
```

Função **float()** converte um número inteiro num decimal juntando um zero depois do ponto a que se chama **ponto flutuante**.

Tipo de dados lógicos

O tipo de dados lógicos é comummente usado em comandos quando procuramos um valor com condições lógicas. Uma condição lógica é uma construção lógica em que se verifica o valor de algo. Normalmente, só tem dois valores: True (verdadeiro) ou False (falso). Em comandos de ramificação (if, if-else, elif-else) e repetição condicional (while) verificamos que, dependendo da condição lógica, o programa toma a decisão de continuar a funcionar. Se queremos armazenar um valor lógico numa variável, podemos atribuir-lhe um valor lógico.

```
>>> a=4<5
>>> print(a)
True
>>>
```

```
>>> b=5<2
>>> print(b)
False
>>>
```

```
>>> b=False
>>> print(b)
False
>>>
```

Strings

Os dados strings são um tipo básico de dados para armazenar valores textuais. Eles representam palavras ou frases dentro de aspas (“) ou semi-aspas (‘).

```
>>> a='python'
>>> print(a)
python
>>>
```

```
>>> b="dobar dan"
>>> print(b)
dobar dan
>>>
```

Também aprendemos como aplicar operações matemáticas numa string, por exemplo adição ou multiplicação.

```
>>> x='abc'
>>> y='def'
>>> x + y
'abcdef'
>>>
```

```
>>> x * 4
'abcabcabcabc'
>>>
```

Adicionando dois caracteres, a string cria uma nova string que contém os caracteres originais.

Multiplicando uma string com um número vai multiplicar a string. Uma string não pode ser multiplicada por outra string – temos de usar um número inteiro.

Uma string pode ainda ser definida como vazia, quando não se escrevem os caracteres entre as semi-aspas.

```
>>> a        (1)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    a
NameError: name 'a' is not defined
>>> a=''     (2)
>>> a
''
>>>
```

Definição ineficaz de string a

Definição eficaz de string a

Indexando uma string

A string pode ser vista como vários caracteres dispostos numa certa ordem. A atribuição de números às posições dos caracteres chama-se indexação. Neste procedimento, a cada caractere é atribuído um número do índice ou, noutras palavras, um número que descreve onde o caractere está na string.

Na Python o primeiro caractere numa string está sempre associado ao número zero (0), e depois ao caractere seguinte é atribuído o número 1, depois o 2, 3, 4…

r =	P	y	t	h	o	n
index	0	1	2	3	4	5

Posição do caractere na string (índice)

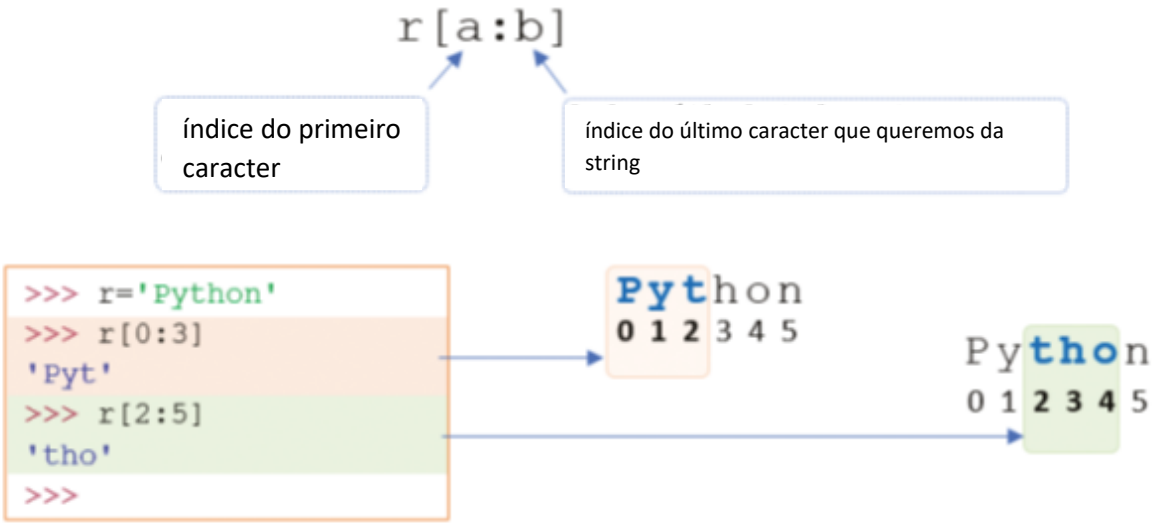
EXCERCÍCIO 1

Usando a interface de usuário interativo, junta o primeiro e o último caractere na string „Dubrovnik“.

```
>>> r='Dubrovnik'
>>> r[0], r[8]
('D','k')
```

Junta vários caracteres ao mesmo tempo – uma parte da string

Se queremos juntar múltiplas partes da string, escrevemos os respetivos índices da primeira e da última parte da string que queremos usar e separamo-las com dois pontos (:) dentro dos parêntesis da string.



Há várias formas de extrair partes específicas da string dependendo se queremos essas partes para incluir o primeiro e o último caractere na string. Uma parte da string que inclua o primeiro caractere pode ser extraído sem o índice 0, por exemplo, em vez de r[0:3] podemos escrever r[:3]. O mesmo pode acontecer com o último caractere da string, por exemplo podemos escrever r[2:].

Se não definirmos os pontos de partida e de chegada de uma string, Python apenas levará a string inteira.

```
>>> r[:]
'Python'
>>>
```

4. Strings de dados – listas

Nos exemplos que vimos até agora, conhecemos diferentes tipos de dados input, com os quais voltaremos a trabalhar posteriormente. Se trabalhamos com múltiplos dados input, armazenamo-los em diferentes variáveis. Até mesmo quando os usamos para realizar a mesma operação, tivemos de os escrever segundo um comando especial, o que tornou as nossas soluções mais complexas. Podemos criar soluções mais simples?

Se precisarmos de escrever uma grande quantidade de dados, todos com uma característica comum, é aconselhável armazená-los numa estrutura única de dados. Assim, podemos usar os dados input através de nomes comuns e indexados, que nos dirão a posição de cada parte dos dados numa estrutura de dados maior.

Alguns exemplos mais usuais são os nomes dos alunos, as temperaturas diárias do ar, e outros do mesmo género.

A lista é, pois, uma estrutura de dados no Python que pode ser usada para armazenar vários ítens numa única variável. Os ítens de uma lista podem ser de qualquer tipo de dados, a que damos o nome de elementos da lista.

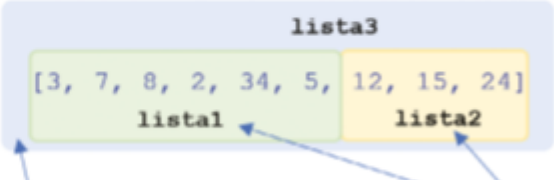
Exemplos de listas	
[8,5,3,7,6,2]	Lista de números com 6 elementos
['Ana','Ivo','Marko','Lucija','Nikola']	Lista de caracteres string com 5 elementos
[7,8,'Leona','Darko',3.14,1.4]	Lista de 6 elementos – números inteiros e decimais, e caracteres string
[12,[2,5],[ ],['a','b','c']]	Lista com um número inteiro, lista com números, lista com caracteres, e uma lista vazia

Repare-se então que uma única lista pode conter números inteiros ou decimais, caracteres, e até mesmo outras listas.

Vamos usar alguns exemplos no nosso interface para mostrar diferentes formas de definir listas. As listas podem ser acrescentadas a uma variável, e também podemos fazer uma nova lista aplicando a adição (+) e a multiplicação (*).

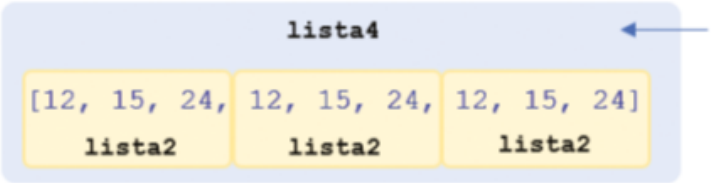
```
>>> lista1=[3,7,8,2,34,5]
>>> lista1
[3, 7, 8, 2, 34, 5]
>>> lista2=[12,15,24]
>>> lista2
[12, 15, 24]
```

```
>>> lista3=lista1+lista2
>>> lista3
[3, 7, 8, 2, 34, 5, 12, 15, 24]
>>>
```



Lista 4 é o resultado da adição de duas listas (lista 1 e lista 2)

```
>>> lista4=lista2*3
>>> lista4
[12, 15, 24, 12, 15, 24, 12, 15, 24]
>>>
```



Lista 4 é o resultado da multiplicação da lista 2 com o número inteiro 3

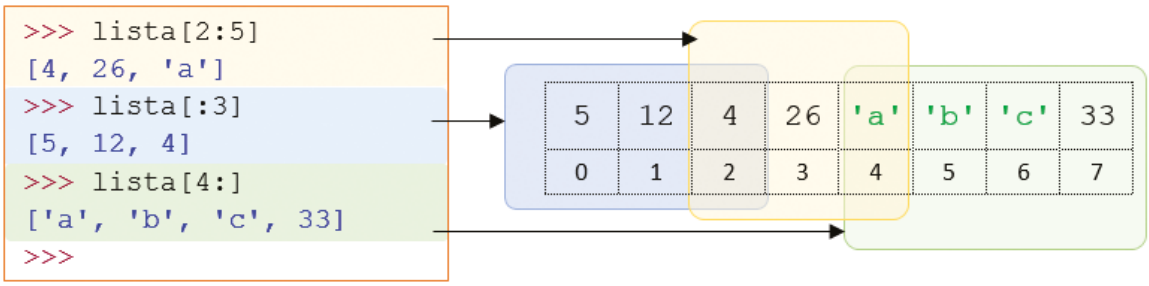
Extraindo numa parte da lista com Índice

```
>>> lista=[5, 12, 4, 26, 'a', 'b', 'c', 33]
>>> print(lista)
[5, 12, 4, 26, 'a', 'b', 'c', 33]
>>> lista[3]
26
>>> lista[5]
'b'
>>>
```

Como podemos extrair diferentes elementos de uma lista? Para tal, usamos índices, da mesma forma que apanhámos elementos de uma string (cadeia, corda).

Uma parte da lista

Tal como com as strings, uma parte de uma lista é definida digitando o primeiro e o último elemento que queremos, separados como dois pontos (:).



EXERCÍCIO 1

Coloque todos os elementos de uma lista de dados única, um a seguir ao outro.

Como podemos saber o número de elementos de uma lista? Tal como com as strings, usamos a função **len()**, que dá o resultado do número de elementos numa lista.

```
>>> imena=['Ivan','Marko','Ana','Leona']
>>> for i in range (len(imena)):
        print(imena[i])

Ivan
Marko
Ana
Leona
>>>
```

1. definimos uma lista de nomes
2. utilizámos len(imena) para determinar o comprimento da lista
3. utilizámos um laço para imprimir todos os elementos

Número da repetição	Valor do índice no loop	Comando de execução	Itens publicados
1. repetition	i=0	Print (imena[0])	Ivan
2. repetition	i=1	Print (imena[1])	Marko
3. repetition	i=2	Print (imena[2])	Ana
4. repetition	i=3	Print (imena[3])	Leona

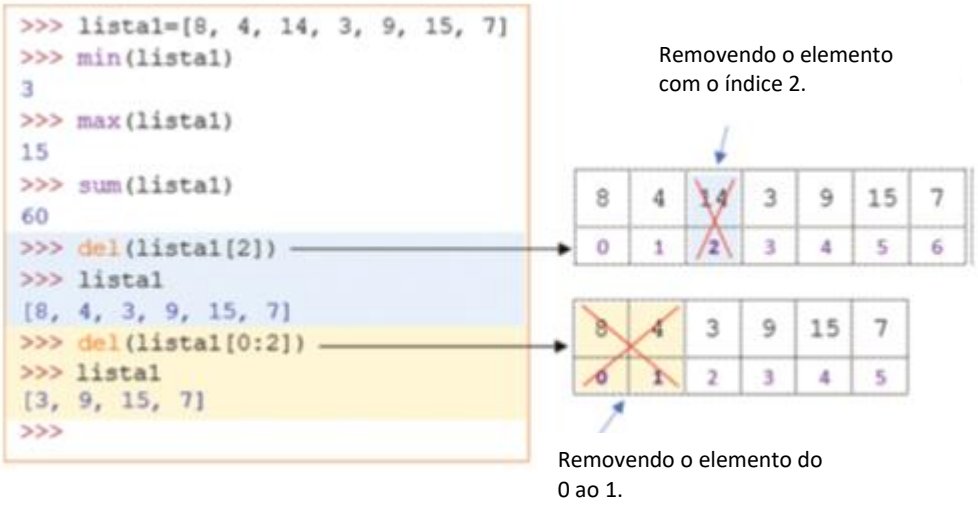
5. Funções de construção e métodos de trabalho com listas

Aprendemos como armazenar dados com características comuns (strings) com a mesma variável e como ter acesso a cada um dos elementos dessa variável. Vamos agora conhecer algumas funções e métodos que podem tornar o nosso trabalho com as listas ainda mais simples.

Ao usar as strings para armazenar diferentes tipos de dados com atributos comuns, tornámos mais simples aceder e ver dados input. Muitas vezes precisamos de determinar o tamanho (número de elementos) na string, de encontrar o menor ou o maior valor, ou até apagar um elemento. Muitas linguagens de programação oferecem diferentes funções que fazem isso mesmo. Apresentamos na tabela abaixo algumas das funções disponibilizadas no Python.

Função	Descrição
len(a)	Retorna a extensão da lista
min(a)	Retorna o menor elemento da lista
max(a)	Retorna o maior elemento da lista
sum(a)	Retorna a soma de todos os elementos
del(a[i])	Remove o elemento com o índice **i**
del(a[i:j])	Remove todos os elementos de **i** a **j-1**

Aplicar estas funções depende do tipo de dados dos elementos na string.



EXERCÍCIO 1

Cria uma lista com a altura dos alunos de uma turma. Usando funções, encontra e visualiza a altura mais baixa e a mais alta.

```
>>> lista=[156,160,145,147,160,157]
>>> print('The height of the shortest student is',min(lista))
The height of the shortest student is 145
>>> print('The height of the highest student is',max(lista))
The height of the highest student is 160
>>>
```

EXERCÍCIO 2

Cria uma lista com as notas dos alunos num teste. Usando funções, publica a média das classificações.

```
>>> lista=[1,2,3,4,5,1,2,3,4,5]
>>> print(sum(lista)/len(lista))
3.0
>>>
```

Tal como com as strings, uma lista pode estar vazia.

Podemos criar uma lista vazia atribuindo parêntesis retos vazios a uma variável, por exemplo: a=[].

```
>>> empty=[]
>>> print(empty)
[]
>>>
```

Métodos de construção para trabalho com as listas

Além de operações e de funções, quando trabalhamos com as listas, podemos usar métodos. Estes são sempre atribuídos a um determinado objeto (uma variável ou uma lista) e aparecem sempre depois do objeto a que estão atribuídos. Alguns dos métodos que usamos no Python são:

Nome do método	Forma de o usar	Descrição
append()	a.append(b)	Adiciona o elemento b no fim da lista a
insert()	a.insert(i,b)	Insere o elemento b antes do elemento i da lista a
remove()	a.remove(b)	Remove o elemento b da lista. Se houver múltiplos b-s, o que tiver o índice mais baixo é apagado.
reverse()	a.reverse()	Faz uma inversão na ordem dos elementos da lista a.
sort()	a.sort()	Classifica a lista do menor para o maior valor.
count()	a.count()	Retorna o número de repetições do objeto b na lista a
index()	a.index(b)	Encontra o índice do elemento b na lista.

Vamos ver estes métodos em alguns exemplos.

```
>>> a=[9,4,6,12,16,3]
>>> a.append(13)
>>> a
[9, 4, 6, 12, 16, 3, 13]
>>>
```

acrescenta um novo elemento com valor 13 e coloca-o no fim da lista

```
>>> a.insert(3,17)
>>> a
[9, 4, 6, 17, 12, 16, 3, 13]
>>>
```

acrescenta um novo elemento com o valor 17 e coloca-o no índice 3

```
 0  1  2  3  4  5  6  7
>>> a.remove(16)
>>> a
[9, 4, 6, 17, 12, 3, 13]
>>>
```

apaga o elemento com o valor 16 de uma lista a. A lista é mais curta por 1 elemento.

```
>>> a.reverse()
>>> a
[13, 3, 12, 17, 6, 4, 9]
>>>
```

Os elementos da lista a são revertidos.

```
>>> a.sort()
>>> a
[3, 4, 6, 9, 12, 13, 17]
>>>
```

Os elementos da lista a são ordenados a partir dos mais pequenos para os mais altos.

```
>>> a.append(6)
>>> a
[3, 4, 6, 9, 12, 13, 17, 6]
>>> a.count(6)
2
>>>
```

acrescentamos um elemento com os valores 6.
contamos quantas repetições o valor 6 tem na lista a.

```
>>> a.index(12)
4
>>> a
[3, 4, 6, 9, 12, 13, 17, 6]
```

imprimimos o índice de um elemento com o valor 12.

EXERCÍCIO 1

Escreve uma programação que tenha uma lista de **n** números naturais como input. Imprime a lista dos números numa única linha.

```
n=int(input('Enter the number of elements in the list:'))
a=[]
for i in range(n):
    m=int(input('Enter a number: '))
    a.append(m)
print(a)
```

Explicação: na tarefa, podemos ver que o comprimento da lista está especificado – tem n elementos. Escrevemos os elementos usando a função para a qual escreverá um valor **n** vezes. Com o método **append()**, acrescentamos o elemento no final da lista a. Tivemos de inicializar a lista no início da programação.

No final, visualizamos a lista usando simplesmente o comando print.

Fazer uma lista a partir de valores input

Na última atividade, mostrámos como criar uma lista a partir de valores dados – caracteres ou números. No entanto, muitas vezes não sabemos o número de elementos na lista. Neste caso, não podemos usar o **for loop** com um número atribuído de funções. Neste caso, temos de usar o método **split()**, com o qual se divide uma sequência numa lista, ou seja, podemos separar palavras com um espaço em branco, dentro da string, e assim criar uma nova lista de palavras. Os elementos de uma lista feita com o método **split()** são sempre caracteres.

```
>>> names=input('Enter the names of the students: ').split()
Enter the names of the students: Steve, John, Katie, Sarah
>>> print(names)
['Steve,', 'John,', 'Katie,', 'Sarah']
>>> |
```

EXERCÍCIO 2

Escreve um programa no qual daremos entrada a várias palavras e depois criamos uma nova lista que contém apenas a primeira letra de cada palavra.

```
a=int(input('Enter a list of words:').split()
b=[]
for i in range(len(b)):
    x=a[i]
    b.append(x(0))
print(b)
```

Explicação

a=input('Enter a list of words: ').split()	**Dá entrada às palavras da lista a**
b=[]	Cria uma lista **b** vazia, na qual serão armazenadas mais tarde as letras iniciais da lista **a**.
for i in range(len(b)):	Define a repetição dependendo do comprimento da lista **a**.
x=a[i]	Cada palavra da lista a está temporariamente armazenada a variante **x**.
b.append(x(0))	Adiciona novos elementos à lista, a primeira letra da palavra armazenada em x.
print(b)	Publica a nova lista b.

EXERCÍCIO 3

Escreve um programa que dará entrada a uma lista de letras numa única linha e depois inverte a ordem dos elementos na lista. Publica a nova lista formada.

Exemplo de teste

```
a=input('Input a list of letters:').split()
a.reverse()
print(a)
```

Programa de computador

```
Input a list of letters:a b c d e f
['f', 'e', 'd', 'c', 'b', 'a']
>>>
```

Podemos usar o último programa para reverter a lista dos números? Se os valores numéricos forem inseridos como caracteres (**sem int()**), então precisamos de os converter em números, usando essa função (**int()**).

Lista de números armazenados como uma lista de caracteres.

```
>>> list=input('Input a list of numbers: ').split()
Input a list of numbers: 7 6 3 12 4 15 8
>>> print(list)
['7', '6', '3', '12', '4', '15', '8']
>>>
```

Usando a função int() converte os caracteres da lista original em números.

```
>>> list=input('Input a list of numbers: ').split()
Input a list of numbers: 7 6 3 12 4 15 8
>>> new=[]
>>> for i in range(len(list[i])):
        new.append(int(list[i]))
>>> new.reverse()
>>> print(new)
[8, 15, 4, 12, 3, 6, 7]
>>>
```

EXERCÍCIO 4

Depois de cada teste, os professores corrigem as respostas e analisam o trabalho dos alunos. Escreve um programa que introduza todas as classificações que os alunos obtiveram no teste. Usando as funções que aprendeste, imprime o número de alunos que teve A (ou 100) como nota, e o número de alunos que não passou no teste.

Exemplo de teste

```
Input grades: 1 2 3 5 4 3 2 5 3 5 1
3 students have an A grade.
2 students have not passed the test.
>>>
```

Programa de computador

```
list=input('Input grades: ').split()
new=[]
for i in range(len(list)):
        new.append(int(list[i]))
print(new.count(5),'students have an A grade.')
print(new.count(1),'students have not passed the test.')
```

EXERCÍCIO 5

Escreve um programa que insira uma lista arbitrária de números. O programa tem de criar e escrever uma nova lista de números contendo apenas os números pares e que foram inseridos em primeiro lugar.

```
list=input('input the numbers: ').split()
new=[]
for in in range(len(list)):
    if int(list[i])%2==0:
        new.append(int(list[i]))
print(new)
```

Explicação

Tal como na atividade anterior, temos de criar uma nova lista, mas não a partir de todos os valores inseridos. Estamos só a usar aqueles que respondem à condição de serem números pares. Enquanto resolvemos este exercício, devemos lembrar-nos do uso do operador % (remainder/restante) para testar se o número é par. O comando crucial na resolução desta atividade é:

if int(list[i])%2==0:

list[i]	muda os índices e leva-nos através da lista
int	torna o elemento da lista num número
%2==0	verifica se o número é par
new.append	insere o número par no final da nova lista

6. Funções no Python

Ao arranjar soluções num computador, normalmente começamos por analisar o problema e depois tentamos dividi-lo em partes mais pequenas. Desta forma, passamos a olhar para o problema baseando-nos em soluções para cada parte. Como podemos escrever soluções de pequenos programas dentro de um programa com o qual podemos resolver partes do problema, e a que podemos recorrer várias vezes?

Nos programas de computador, cada série de comandos tem um papel a desempenhar. Alguns lidam com os dados input, outros com output, algumas partes formam soluções, outras lidam com os dados de acordo com as instruções.

Vários comandos que fazem sentido como um todo podem ser separados como uma parte independente do programa, a que se chama função.

Até agora, vimos várias funções integradas, como int(), input(), print(), len()…

Estas funções já fazem parte do Python e só as usamos quando precisamos delas.

Além do uso das funções integradas, também podemos fazer as nossas próprias funções para tarefas específicas.

Geralmente, precisamos de fazer funções para separar partes independentes do programa que tendem a repetir-se. Quando fazemos isso, estamos a aperfeiçoar o programa e este trabalha mais rapidamente.

No Python, é necessário definir uma nova função, o que significa que temos de usar comandos que descrevam o que fazem. Depois, temos de lhe dar um nome e uma lista de argumentos, que irá usar. Por exemplo:

```
def funcion_name (paramethers):
    block_of_commands
    return value
```

Após definirmos a função, ativamo-la no programa bastando para tal utilizar o seu nome. Considerando que as funções usam muitas vezes dados input, temos de dar esse tipo de dados à função se queremos que funcione corretamente. As funções podem ter diferentes parâmetros de entrada que o utilizador usa quando e se forem necessários. Há quatro tipos de funções diferentes no Python:

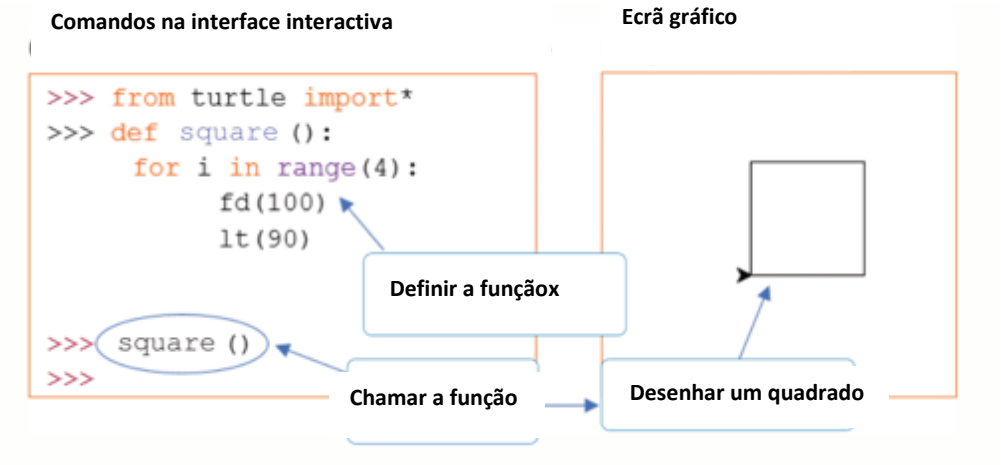
Funções que não retornam um valor e:	Funções que retornam um valor e:
Não têm parâmetros de entrada	Não têm parâmetros de entrada
Têm parâmetros de entrada	Têm parâmetros de entrada

Uma função que não tem valores de entrada e não retorna valores depois de execução

Esta forma de função não retorna quaisquer valores pelo que não precisamos de usar o comando de retorno. Esta função é executada utilizando o seu nome sem quaisquer parâmetros dentro dos parêntesis: function_name().

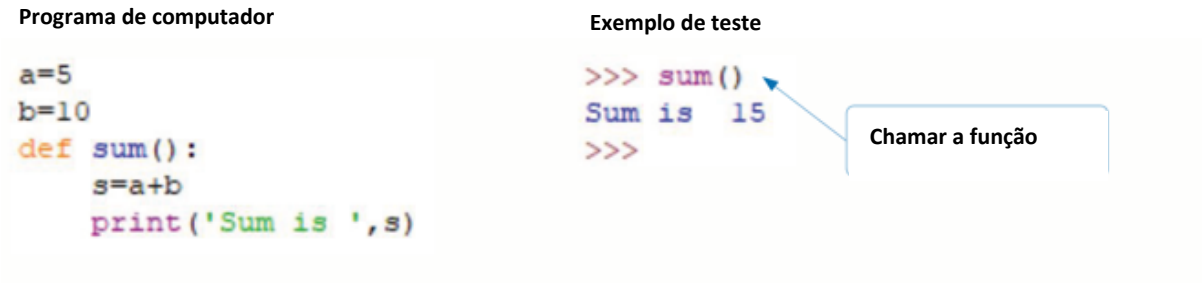
EXERCÍCIO 1

Vamos aplicar a função para desenhar um quadrado com 100 pixéis de lado.

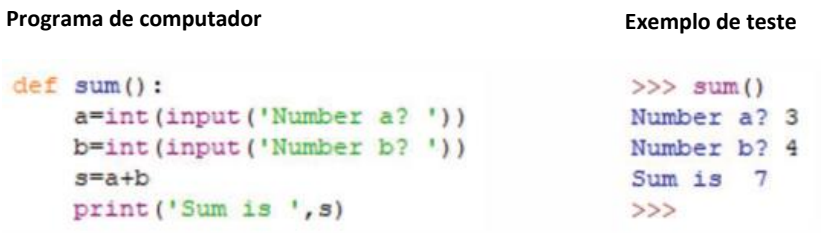


EXERCÍCIO 2

Escreve um programa no qual uses uma função para adicionar dois números dados. Os tamanhos dos números (variáveis a e b) serão definidos fora da função, no programa.



Neste exercício, escrevemos um programa de computador que adiciona dois valores previamente definidos, e isso é bastante limitador. Uma melhor forma de o fazer poderá ser integrar as variáveis input que queremos adicionar.



Se quisermos usar a mesma função de adição várias vezes, podemos usar a **for loop**, para repetir, por exemplo, uma sequência.

Programa de computador

```
def sum():
    a=int(input('Number a? '))
    b=int(input('Number b? '))
    s=a+b
    print('Sum is ',s)
for i in range(3):
    sum()
    print()
```

Exemplo de teste

```
Number a? 3
Number b? 4
Sum is  7

Number a? 5
Number b? 6
Sum is  11

Number a? 7
Number b? 8
Sum is  15

>>>
```

O comando print() tem a tarefa de criar uma linha vazia entre os valores impressos

Uma função que tem valores de entrada e não retorna um valor após a execução

A função é executada registando o seu nome seguido de parêntesis. E o parâmetro é a variável listada dentro dos parêntesis na definição da função. Este tipo de função tem um ou múltiplos parâmetros de entrada, function_name(lista de parâmetros), e não retorna qualquer valor após ser executado.

EXERCÍCIO 3

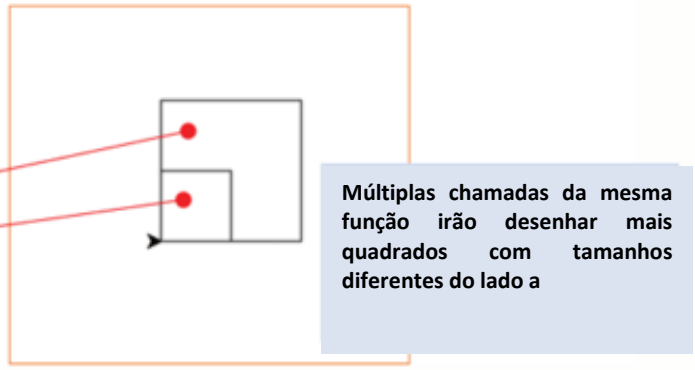
Vamos utilizar uma função que desenhe um quadrado com um pixel de lado. O comprimento do lado "a" é definido ao "chamar" a função como um parâmetro de entrada.

Comandos na interface

```
>>> from turtle import*
>>> def square (a):
      for i in range(4):
            fd(a)
            lt(90)

>>> square (100)
>>> square (50)
>>>
```

Ecrã gráfico



Múltiplas chamadas da mesma função irão desenhar mais quadrados com tamanhos diferentes do lado a

A função **square()** do exercício anterior usa um parâmetro input escrito entre parêntesis depois do nome da função. Se queremos usar parâmetros input para "chamar" a função, então esses parâmetros têm de ser definidos: **def square(a)**. No Python, uma função é definida usando a palavra-chave **def.** Uma função pode ter vários parâmetros definidos, que serão mostrados no próximo exercício.

EXERCÍCIO 4

Vamos escrever um programa que use uma função para adicionar dois números.

Programa de computador

```
def sum(a,b):
    s=a+b
    print('Sum is ',s)
```

Exemplo de teste

```
>>> sum(5,7)
Sum is  12
>>>
```

Chamar a função

Uma função que não tem parâmetros input e não retorna valores depois de executada

Os parâmetros input são escritos quando mudamos a função incorporada **input()**, e o comando "return" dá o resultado do cálculo. O valor que será devolvido ao programa é escrito após o comando de retorno.

EXERCÍCIO 5

Vamos escrever um programa que calcule e imprima uma multiplicação de dois números. Para tal, teremos de usar uma função especial. Os números são escritos para o programa principal como valores input.

```
def multi():
    z*a*b
    return z
a=int(input('Enter the first number: '))
b=int(input('Enter the second number: '))
print('The result is',multi())
```

Explicação

Usámos vários valores **input** para escrever os valores **a** e **b** e a função **multi()**, que multiplica estes dois números. Depois de executar a função **multi()**, o resultado é armazenado na variável **z**, como resultado de **multi()**, no programa principal.

EXERCÍCIO 6

Escreve um programa que dê entrada a dois números de três dígitos e imprima um número com dígitos menores do que 1. A função escrita irá comparar os dígitos e produzir o número desejado.

```
a=int(input('Write the first number'))
b=int(input('Write the second number'))
a1 = a%0
b1 = b%10
def smaller():
    if a1<b1:
        return a
    else:
        return b
print('Number',smaller(),'has a smaller digit.')
```

Explicação

Ao usar o operador **%** (restante) separámos os dígitos dos elementos escritos em **a** e **b**. Depois armazenámos em **a1** e **b1**. Através da função **smaller()** dentro de **print()** comparámos os valores das variáveis **a1** e **b1** e devolve o valor correspondente ao programa. Como resultado, esse valor será visualizado.

Uma função que tem parâmetros input e retorna valores depois de executada

A mais complicada forma de função é aquela que tem parâmetros input e retorna um valor depois de executada. Podemos ver que essa função tem de ter uma lista de parâmetros expectáveis quando a definimos e também tem de ter um valor depois do comando de retorno.

EXERCÍCIO 7

Vamos escrever um programa que irá calcular a soma dos primeiros números naturais. Dentro do programa, vamos aplicar a função para calcular a adição dos números dados.

Programa de computador

```
def sum(n):
    s=0
    for i in range(1,n+1):
        s=s+i
    return s
n=int(input('Write a number n: '))
print('Sum of the first',n,'natural numbers is',sum(n))
```

Exemplo de teste

```
Write a number n: 5
Sum of the first 5 natural numbers is 15
>>>
Write a number n: 7
Sum of the first 7 natural numbers is 28
>>>
```

EXERCÍCIO 8

Muitas vezes vemos números primos nos exercícios de matemática. Por exemplo, um estudante tem de verificar se um número é primo ou não. Vamos ajudá-lo a escrever um programa que introduzirá qualquer número como input para verificar se é um número primo. No nosso programa usaremos a função que retornará a mensagem dizendo se um número é primo ou não.

```
def prime(n):
    x='The number is a prime'
    for i in range(2,n):
        if n%i==0:
            x='The number is not a prime'
    return x
n=int(input('Write a number: '))
print(prime(n))
```

Explicação

Sabemos que os números primos não podem ser divididos exceto pelo um (1) ou pelos próprios. Então só precisamos de verificar quando o **n** pode ser dividido por outros números de 2 a n-1. Se não houver esse número, **n** é um número primo.

A função **for loop** garante que todos os números são verificados. Se encontramos esse número, o valor de **x** será alterado.

EXERCÍCIO 9

Vamos atualizar o nosso último programa para que imprima números até chegar ao zero. Com cada número, também irá imprimir uma mensagem dizendo se o número é primo.

Programa de. computador

```
def prime(n):
    x='Number is a prime.'
    for i in range(2,n):
        if n%i==0:
            x='Number is not a prime.'
        return x
n=int(input('Write a number: '))
while n!=0:
    print(prime(n))
    print()
    n=int(input('Write a number: '))
```

Exemplo de teste

```
Write a number: 6
Number is not a prime.

Write a number: 7
Number is a prime.

Write a number: 8
Number is not a prime.

Write a number: 0
>>>
```

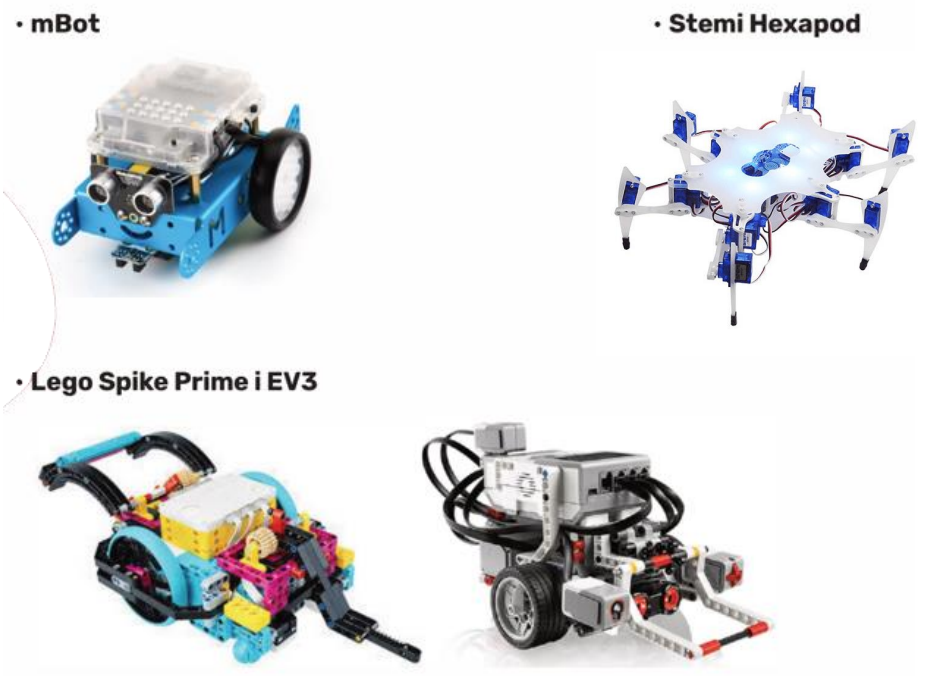
Parar o programa

8º Ano

1. O meu robô resolve o problema

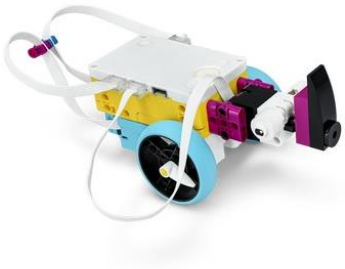
Atualmente, contamos com os robôs para resolver tarefas no mundo à nossa volta. Desde o uso dos robôs na construção de edifícios até à intervenção em cérebros humanos, abelhas robô que ajudam a resolver a fome no mundo, até algoritmos de IA (Inteligência Artificial) que rastreiam criminosos reconhecendo os seus rostos.

Se queremos ser capazes de programar e construir grandes robôs, temos de começar pelos mais pequenos. Felizmente, hoje em dia, já há muitos robôs que foram feitos para aprender. Alguns dos mais populares são:

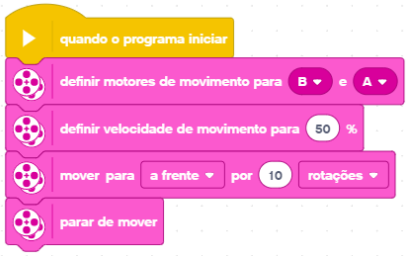


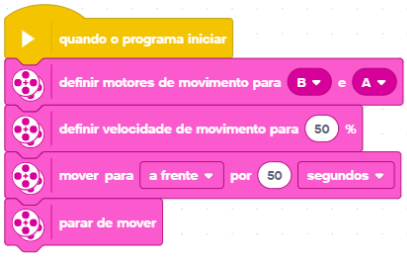
Cada um destes robôs pode ser programado simplesmente por blocos – por exemplo, o mais recente Lego Spike Prime pode ser programado no Scratch.

Um simples exemplo da programação de um robô será mostrado no Spike Prime. Primeiro, precisamos de um módulo Spike Prime, sob o qual construiremos dois grandes motores ligados às rodas, e na parte da frente montaremos um sensor com uma peça lego que mais parece uma cabeça de rinossauro:

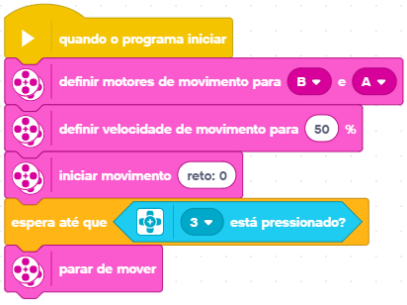


O objetivo é que o robô se movimente em frente, mas que pare antes de atingir a parede feita de peças Lego. Podemos medir a distância à mão e depois escrever um programa simples que faça com que o nosso robô avance numa certa distância ou num determinado número de rotações das rodas:





Uma melhor solução é usar um sensor ao toque incorporado. Se dissermos ao robô para parar quando o sensor é pressionado, não temos de calcular o valor da distância. O robô irá detetar a parede por ele próprio:



2. Micro:bit comunicação de rádio

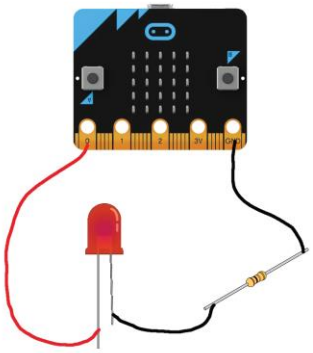
O micro:bit tem a capacidade de comunicar com outros aparelhos através de uma conexão de rádio. O próximo programa deve ser lido em, pelo menos, dois micro:bits. Dessa forma, irás reparar que, se pressionares o botão A em qualquer micro:bit, todos os outros irão mostrar a letra A no seu ecrã. O mesmo é válido para o botão B.

```
1 from microbit import *
2 import radio
3
4 radio.on()
5 radio.config(channel=1)
6 radio.config(power=6)
7
8 while True:
9
10     if button_a.was_pressed():
11         radio.send('A')
12         display.show("A")
13
14     if button_b.was_pressed():
15         radio.send('B')
16         display.show("B")
17
18     primljeno = radio.receive()
19
20     if primljeno == 'A':
21         display.show("A")
22
23     if primljeno == 'B':
24         display.show("B")
```

Explicação do código

import radio	importa a biblioteca para rádio
radio.on()	liga o rádio
radio.vonfig(channel=1)	define qual o canal será usado pelo micro:bit. O número pode ser do 0 ao 100.
radio.config(power=6)	define a força do sinal do output. A força máxima é 7.
primljeno = radio.recieve[]	grava tudo o que for recebido numa variável chamada „primljeno“.

3. Conetar LEDs ao micro:bit



O micro:bit tem pinos de saída que permite ser conectado a diferentes componentes ou sensores. Se tiveres um diodo LED, um resistor de 100 ohms e três fios, podes ligar o LED ao teu micro:bit. Um fio deve ser ligado à saída do micro:bit marcado como GND e a um lado do resistor. Depois o segundo fio vai do outro lado do resistor ao (-) no LED. O (+) no LED fica ligado diretamente ao zero (0) no micro:bit.

A voltagem do micro:bit é de 3 Volts, enquanto a voltagem do LED é mais baixa, e é por isso que usamos um resistor. Neste exemplo, é melhor usar uma bateria para alimentar o micro:bit.

O próximo programa fará ligar e desligar o LED com intervalos de 500 milisegundos.

```
from microbit import *

while True:

    pin0.write_digital(1)
    sleep(500)
    pin0.write_digital(0)
    sleep(500)
```

Se tiveres um LED múltiplo podes tentar ligar mais do que um ao micro:bit. Usa os pinos 0, 1, e 2. Se tiveres um LED verde, amarelo e vermelho – podes tentar fazer um semáforo.

4. Controlo analógico de LEDs

Além de se conseguir ligar ou desligar, também podemos controlar o quanto os LEDs irão brilhar. O próximo programa aumentará lentamente a intensidade da luz no LED, e depois também a diminuirá.

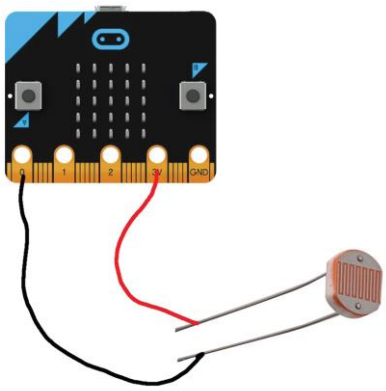
```
from microbit import *

while True:
    for i in range(0,400,10):
        pin0.write_analog(i)
        sleep(40)

    for i in range(400,0,-10):
        pin0.write_analog(i)
        sleep(40)
```

for i inrange(0 , 400, 10):	***For* loop** aumenta o valor do **i** de 0 até 100 de 10 em 10. Então é 0, 10, 20, 30, … , 390, 400. Todos os comandos escritos em baixo serão repetidos até o contador chegar aos 400
for i inrange(400 , 0, -10):	conta ao contrário de -10 em -10
pin0.write_analog(i) Estabelecemos o valor do **i** no pino zero.	Neste exemplo, o valor muda de 0 a 400. O máximo valor que podemos usar é 1023.
O Micro:bit não regula o brilho do LED regulando a voltagem. Em vez disso, ele rapidamente liga e desliga o LED. *pin0.write_analog(0) – LED desligado, pino 0 está inativo.* *pin0.write_analog(1023) – LED brilha o máximo, pino 0 está ativo.* *pin0.write_analog(511) – pino 0 está ativo por um curto período de tempo, e depois inativo pelo mesmo período de tempo.*	

5. Ligando os sensores



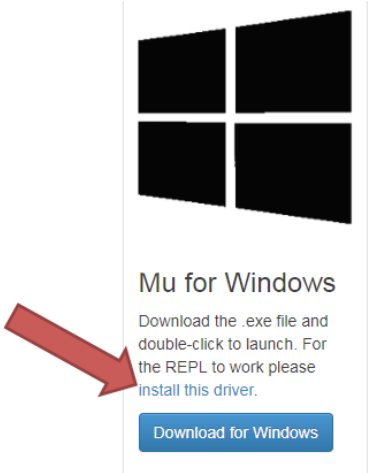
Podemos adicionar vários sensores ao micro:bit. Um dos sensores que é facilmente incorporado é o foto sensor GL5528. Trata-se basicamente de um resistor que muda a sua resistência dependendo da quantidade de luz que incidir sobre ele. Tem dois pinos e não precisamos de nos preocupar com a polaridade quando ligados. Um dos pinos está ligado ao pino 0 no micro:bit, enquanto o outro está ligado ao pino 3 V. (Também podes ligá-lo ao pino GND em vez do pino 3 V).

O próximo programa lê o valor do foto sensor e mostra-o no ecrã do micro:bit. O comando **pin0.read_analog()** retorna um valor entre o 0 e o 1023. Dependendo do tipo do foto sensor que estamos a usar, de como está ligado e da quantidade de luz que incidir sobre ele, os valores vão variar entre os números 0 e 1023. Se colocares a mão no sensor verás como o valor no ecrã está a mudar.

```
from microbit import *

while True:
    ocitanje = pin0.read_analog()
    display.scroll(str(ocitanje))
```

6. Interface interativo – RELP



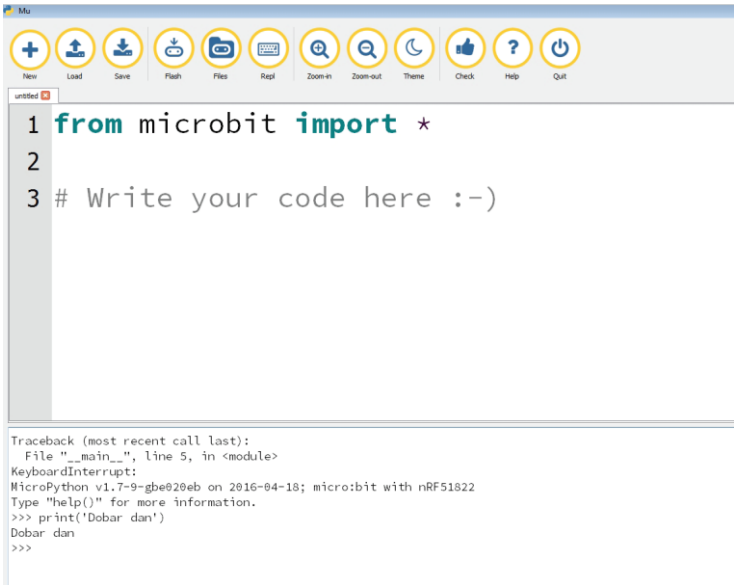
Se queremos poder usar o interface interativo do **micro:bit interactive interface** no Windows, temos primeiro de instalar drives para a comunicação em série. Podemos fazer o download aqui: https://codewith.mu/

Também podes digitar „download the embeded windows serial port driver“ no google e seguir o primeiro resultado.

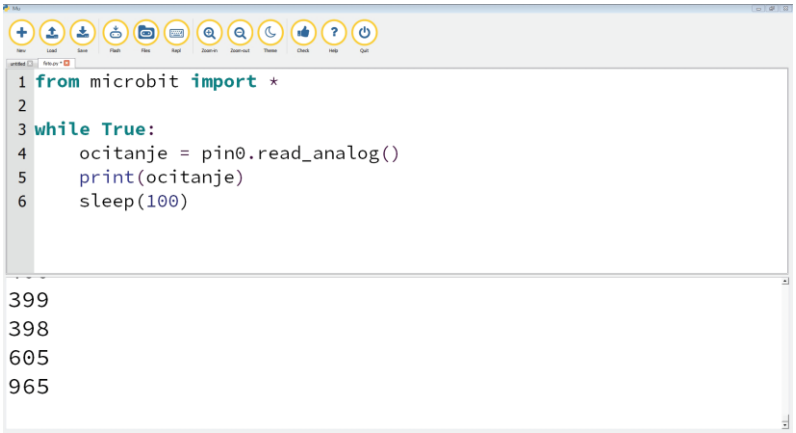
O interface interativo inicia com o botão Relp e, na parte de baixo da janela, veremos uma consola na qual podemos escrever os comandos MicroPython. Se tiveres alguma experiência com o Python “básico”, tenta escrever um programa e verás esses comandos básicos

Python também a funcionar no MicroPython. É importante perceber que o programa é executado no micro:bit, e não no nosso computador.

Se em vez de usares o comando **print(„Hello world“)** escreveres **display.scroll(„Hello world!“)**, verás que a mensagem „Hello world!“ é mostrada não no interface, mas no ecrã do micro:bit.



Nós usamos o interface interativo para experimentar comandos e imprimir dados se queremos lê-los através do ecrã do nosso computador. No exemplo com os valores do foto sensor, podes ver os resultados do sensor no ecrã do computador.



Apenas precisamos do comando **display.scroll(str(ocitanje))** e substituí-lo por **print(ocitanje)**. Antes de clicares no botão Flash, o interface já precisa de estar a correr.

Agora podes ver os valores da foto sensor no teu écrã, e podes ver como eles mudam dependendo da quantidade de luz que atingir o sensor. Põe a tua mão sobre o sensor e vê o valor a mudar, ou tenta apontar o sensor para uma fonte de luz. Tenta escrever um programa que mostre uma cara feliz no micro:bit se houver luz suficiente a atingir o sensor, e uma cara triste se o sensor estiver no escuro (podes tu mesmo definir os valores de fronteira).

9º Ano

1. Telemóvel – como surgiu e qual é o seu futuro?

EXERCÍCIO 1

Faz uma pesquisa sobre a origem dos telemóveis. Os seus fabricantes ainda existem hoje em dia?

Como eram os primeiros sistemas operativos (OS-es)?

Na tua pesquisa, talvez encontres um telemóvel como este Benefon:

Hoje em dia nem conseguimos imaginar ter um aparelho como este. O Benefon já não existe e o seu sistema operativo era igual ao de um telefone fixo.

EXERCÍCIO 2

Pesquisa como eram os primeiros smartphones. Os seus fabricantes ainda existem hoje em dia? Como eram os primeiros sistemas operativos?

Irás provavelmente encontrar um aparelho como o Palm Pocket PC e o sistema operativo Palm:

Este aparelho é um pouco mais parecido com os smartphones de hoje, mas ainda está longe dos que usamos e o sistema Palm já não existe.

O primeiro smartphone "real", depois do qual os outros foram modelados, foi feito pela Apple e, claro, chama-se iPhone. O iPhone foi o primeiro telefone a apresentar um ecrã com capacidade de toque, que é hoje utilizado em todos os smartphones e todos os smartphones, atualmente, também se parecem com este iPhone:

EXERCÍCIO 3

Na tua opinião, que tipo de smartphones serão usados no futuro?

Consideras possível, daqui a uns anos, usarmos óculos smart?

Como é que as nossas vidas podem ser melhoradas com uma combinação de óculos inteligentes e tecnologia AR (realidade aumentada)?

2. Vou desenvolver a minha própria aplicação mobile

Hoje em dia, todas as pessoas têm um smartphone (ou dois), e cada um desses smartphones tem mais de uma dúzia de aplicações.

Se tens um iPhone, então podes usar a App Store para procurar novas aplicações, existem mais de 2,2 milhões aplicações na loja.

Se tens um smartphone com o sistema operativo Android, podes fazer o mesmo na Play Store, e escolher entre mais de 2,8 milhões de aplicações.

Mas, como podes desenvolver a tua própria aplicação?

Para tal, não precisas só de saber codificar, mas também necessitas de pensar numa série de outros passos. Alguns deles são:

1. Esboça uma ideia para o teu aplicativo

O primeiro passo é decidir o que a tua app deve fazer e anotar essas ideias. Tenta manter a tua ideia tão simples quanto possível, para que qualquer pessoa que o leia consiga instantaneamente perceber para que serve a tua aplicação.

Pensa nas características que a tua app tem de ter e que características seria bom ter. Geralmente, é uma boa ideia fazer uma primeira versão tão simples quanto possível e, mais tarde, melhorá-la.

Pensa se haverá alguma aplicação como a tua já disponível no mercado e por que razão as pessoas haveriam de a querer.

2. Explora o mercado

Talvez tenhas uma grande ideia, mas será que ainda não existe no mercado uma app como a que tu queres desenvolver?

O que precisas de saber é, antes de mais, se os usuários a vão querer, quanto é que pagariam por ela e se já existe alguma app igual ou com as mesmas funções/características no mercado.

Mesmo que uma app como a tua já exista, talvez possas desenvolver uma nova app que seja melhor do que as que já estão disponíveis no mercado?

Talvez a tua app venha a ter uma característica nova que nenhuma das existentes tenha?

3. Desenha um esboço da interface do utilizador

Podes fazer isto num programa como o MS Paint, ou podes usar um programa mais especializado como o Balsamiq.

A ideia é fazeres um esboço de como irá parecer a interface da tua app.

Pensa em como será mais fácil utilizá-la, bem como que diferentes partes da app fazem sentido juntas.

Também podes pensar sobre as cores que farão com que a tua app seja mais apelativa e fácil de usar.

4. Desenha os elementos gráficos

Nesta fase, seria importante se soubesses trabalhar com um programa como o Photoshop, mas não é obrigatório.

Também podes desenhar os elementos gráficos usando um template que já exista. Alguns dos mais populares são: NOW, TETHR, Start UI kit, iOS iPhone GUI, Stitch, Phoenix, etc.

Podes utilizá-los para desenhar os elementos gráficos, que, mais tarde, utilizarás no esboço que foi feito anteriormente.

5. Faz um website para a tua app

Qualquer app precisa de ter uma boa ação de marketing. Podes construir um website apelativo para promover a tua app usando uma das mais populares ferramentas de acesso livre como: Wordpress, Site Builder, ou Wix.

Cada uma delas tem dezenas de templates para escolher entre aqueles que foram feitos especificamente para promover as apps.

Só tens de escolher a de que gostas mais e começar a construir o website.

6. Programa

Agora chegamos à parte mais difícil. Para criar uma app, tens de aprender a programar numa das interfaces comummente usadas para conceber apps. Também tens de definir duas partes da tua app:

- Front end: a parte da app que os utilizadores irão ver, ou seja, a interface completa, com todos os elementos gráficos e as conexões entre eles.

- Back end: a parte da app que os utilizadores não veem, onde irás armazenar a tua database, ficheiros, etc.

Para seres capaz de construir uma app, deves ir ao Youtube e assistir a tutoriais de programação de interfaces para iniciantes.

Se queres criar uma app para um telemóvel Android, tens de aprender a trabalhar com o Android Studio. E se quiseres conceber uma app para iOS, terás de aprender a codificar em Swift.

Depois de todo este processo, só tens de publicar a tua app na App Store ou na Play Store e, claro, promovê-la de forma a que seja conhecida.

3. O que é o blockchain?

O blockchain e o bitcon são um enorme sucesso no mundo da tecnologia há vários anos, mas muitos de nós ainda não sabem do que se trata afinal.

O que é o blockchain, realmente?

O **Blockchain** é basicamente uma nova forma de transferência digital de dinheiro, mas também pode ser usada para transferir contratos ou outros dados "sensíveis".

Originalmente, foi inventado para a primeira moeda digital – o Bitcoin, mas hoje já é usada com muitos outros propósitos.

O aplicativo Blockchain decorre mais ou menos assim:

- Alguém solicita uma transação, como por exemplo trocar uma moeda digital por dinheiro.
- Essa transação é publicada em toda a rede de computadores, e essa network cria um algoritmo que permite que a transação aconteça.
- Uma vez que a transação (block) seja aprovada, é incorporada numa cadeia de transações semelhantes.
- Quando a transação chega a esta cadeia, não pode ser alterada de forma alguma.
- A transação é aprovada e, então, será realizado o que foi requerido.

Uma vantagem desta forma de transferência de dinheiro é que não existe um banco central, que poderia vir a ser pirateado, dado que a transação não poderá ser alterada por um estranho ao processo.

Todo o sistema é descentralizado e é construído em milhares de computadores de usuários simples. Outra característica específica é que não existem despesas de transação.

Por exemplo, quando usas o cartão de crédito ou débito para pagar numa loja – a segurança da tua compra é garantida pela companhia ou banco do cartão de crédito, que se certificam que a

loja só retira da tua conta a quantia pela qual adquiriste o bem ou serviço. Mas também garantem que a loja receberá a quantia acordada. O que acontece é que a operadora do cartão de crédito cobra uma pequena taxa à loja e, desta forma, financia o seu trabalho pela transação. O Blockchain não tem nada sequer parecido. O sistema é descentralizado, então não há taxas.

EXERCÍCIO 1

Faz uma pesquisa sobre que moedas digitais existem e quais são as diferenças entre elas. Pesquisa também como podes comprar dinheiro digital e como podes usá-lo para pagar um produto ou serviço.

EXERCÍCIO 2

Faz uma pesquisa sobre quem foi o fundador do Bitcoin e o inventor do Blockchain. O que conseguirás encontrar sobre eles na Internet?

4. Como posso usar o brainstorming para os meus projetos?

À medida que vamos progredindo na vida, os problemas com que nos deparamos e os projetos em que nos envolvemos tornam-se cada vez mais complexos.

Nesta sessão, vamos explorar o **brainstorming** – uma técnica muito interessante que pode ajudar-nos a resolver problemas mais complicados.

O Brainstorming é uma técnica que nos permite criar novas ideias para resolver problemas. Por exemplo, se queremos desenvolver uma app, podemos usar o brainstorming para reunir ideias sobre como a nossa app pode fazer dinheiro.

O Brainstorming é uma técnica muito eficaz em situações nas quais existe mais do que uma situação possível.

Há diferentes técnicas de brainstorming e as mais populares são:

- Fazer uma lista

Esta técnica é muito simples. Só precisas de um papel e uma caneta e começa a escrever tudo o que achas que pode resolver um certo problema.

Se nada vier à tua mente, existem maneiras de “hakear” o cérebro humano. Por exemplo, procura imaginar que és uma outra pessoa. Como é que essa pessoa resolveria o teu problema?

Ou tu podes imaginar que és um superherói da Marvel Multiverse. Como é que alguém com as capacidades do Iron Man resolveria o teu problema?

Também podes usar o alfabeto – tenta registar palavras que começam com cada uma das letras do alfabeto, que, de alguma forma, possam ajudar a resolver o teu problema.

- Fazer um mapa mental

O mapa mental é uma interessante forma de desenhares as tuas ideias.

Pega num pedaço grande de papel e começa a escrever as tuas ideias, mas não as coloques numa lista.

Usa cores diferentes dependendo de como vês as tuas ideias, diferentes tamanhos de letra dependendo de quão importante são, desenha-as longe ou perto do centro dependendo de quão cruciais são para a resolução do problema.

Depois, ligas as tuas ideias com setas ou coloca-as em forma de árvore – para ver como interagem umas com as outras. A mais importante característica do mapa mental é que tem de ser somente teu e completamente único. Não vejas como os outros estão a fazer os seus mapas mentais – em vez disso, procura fazer um que funcione para ti.

- Escrita livre

Esta técnica significa extamente o que diz – tu começas por escrever o que vem à tua mente. Pensa sobre o problema que queres solucionar, mas não escrevas apenas soluções para esse problema.

Concentra-te no problema, mas procura estar tão relaxado quanto possível e escreve o que quer que apareça na tua mente. Nunca se sabe que tipo de ideias estão no teu subconsciente e não sabes que ideias poderão advir daí.

- Brainstorming round-robin

Para experimentar o brainstorming round-robin precisas de um grupo de pessoas que estão a tentar resolver o mesmo problema.

Uma pessoa tem de escrever todas as ideias num único documento.

Formem um círculo: a primeira pessoa a surgir com uma ideia, essa ideia é registada no documento. Depois dessa primeira pessoa vem a segunda, e por aí adiante.

As ideias não podem ser criticadas, mas, depois de todos no círculo terem dado a sua contribuição, as ideias poderão ser discutidas.

Se da discussão não resultou nenhuma nova forma de resolver o problema – façam outro círculo.

EXERCÍCIO 1

Escolhe um dos grandes problemas que a humanidade está a enfrentar no momento, como a fome mundial, o aquecimento global, ou as pandemias.

Tenta aplicar cada um dos métodos para resolver o problema escolhido.

Qual é o método que funciona melhor para ti e porquê?

5. Estou a promover a minha aplicação mobile

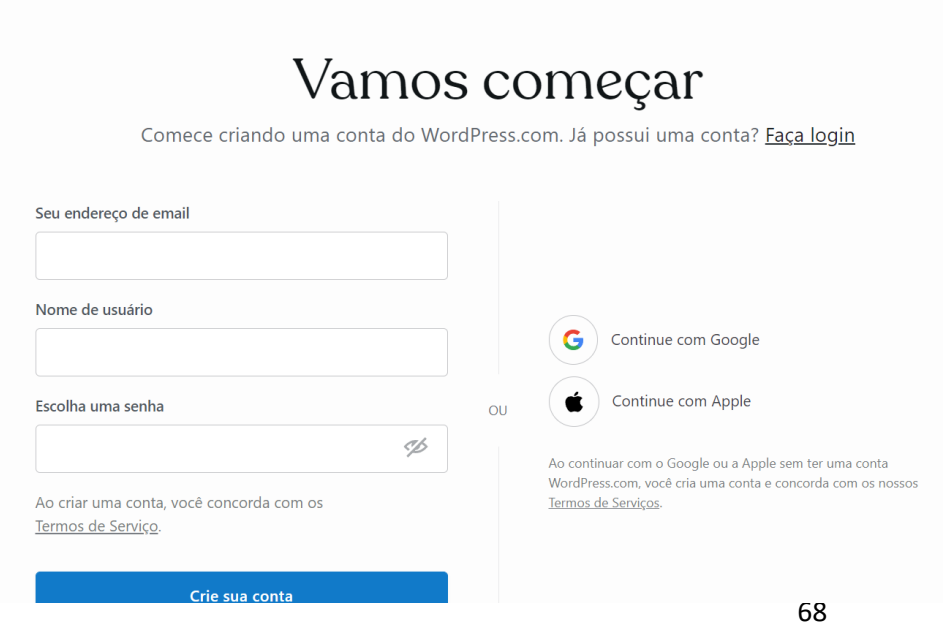
Se estás a construir uma app para o telemóvel ou qualquer outro projeto – precisas de o promover convenientemente.

O objetivo é fazer com que as pessoas se interessem pelo teu projeto, mas também para mostrar às pessoas porque é que o teu projeto é merecedor de investimento. Como podemos promovê-lo?

EXERCÍCIO 1

Faz um website para o teu projeto.

Para construirmos o nosso website, usaremos a ferramenta mais popular para construir websites – a Wordpress. O primeiro passo é ires a wordpress.com, clicares em "Start your website", e criares a tua conta.



Depois de criares a tua conta, tens de escolher o nome e o endereço (URL) para o teu website.

Os URLs que terminem em „wordpress.com“, são grátis, então usaremos um como: “nameofourwebsite.wordpress.com“.

Se esse URL já estiver a ser usado, isso significa que o nome da tua app ou projeto não é original. Tens de dar um passo atrás e escolher um novo nome.

Quando tiveres o URL, só precisas de escolher um tema para o website diferente dos templates já existentes, e confirmar tudo clicando num link na ativação de e-mail que tens da Wordpress.

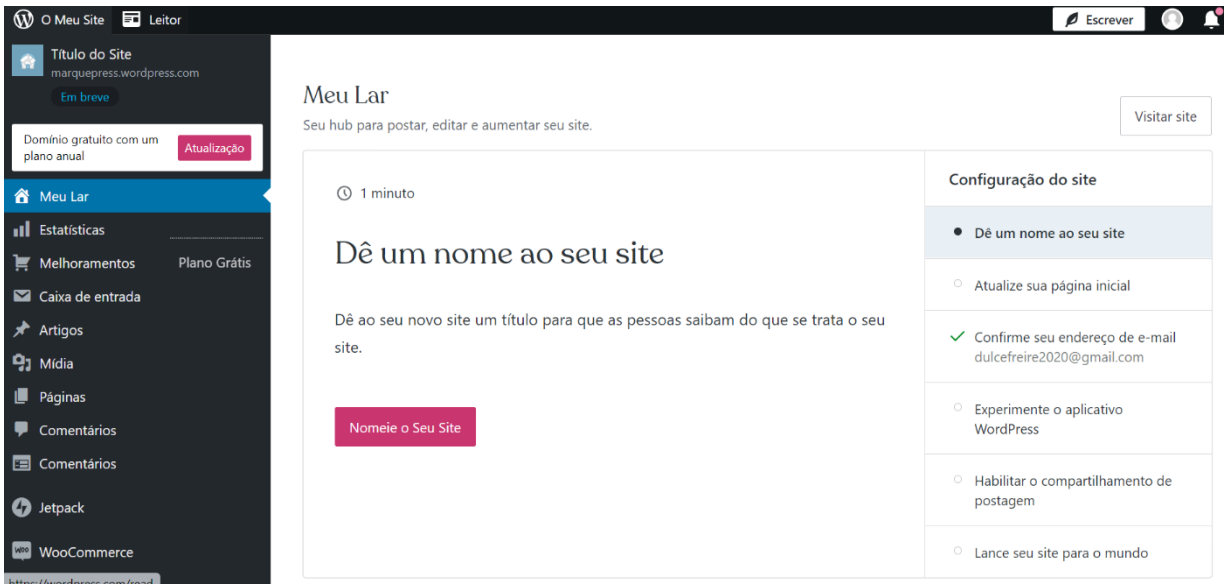
Editar o teu website na Wordpress é muito simples.

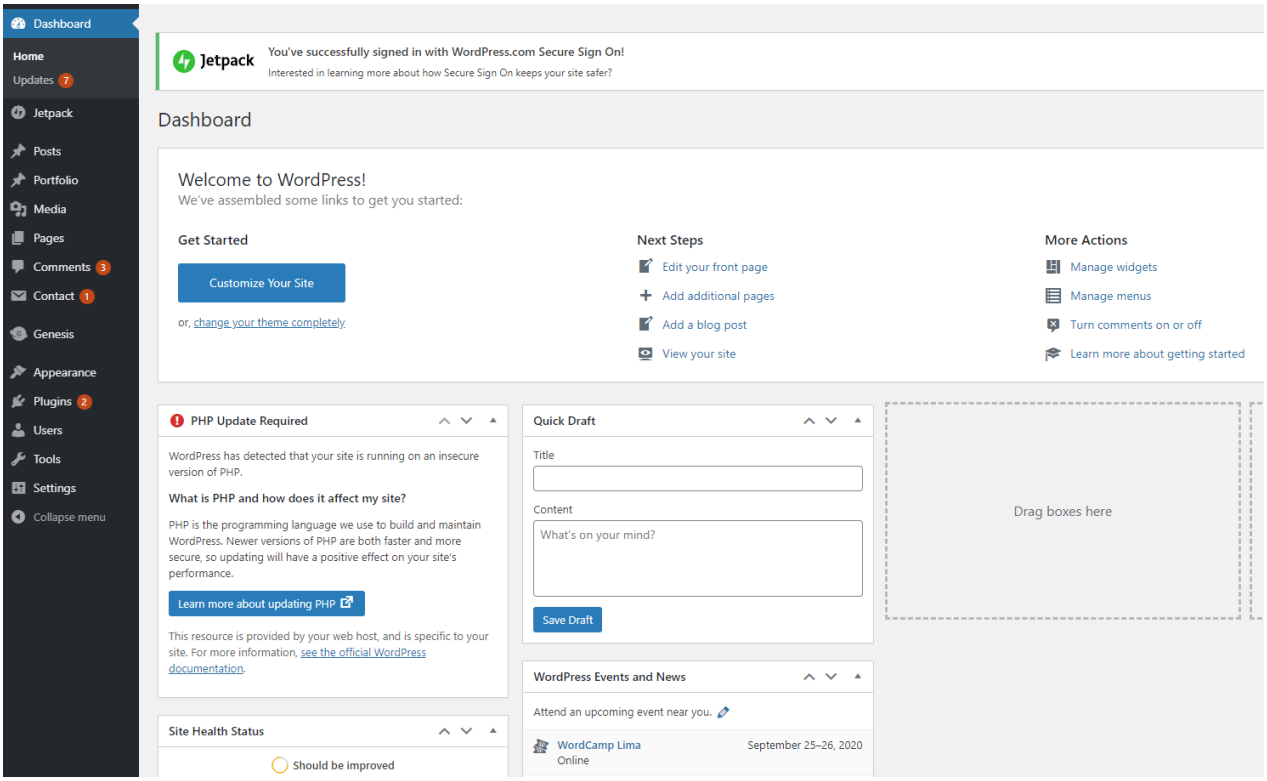
Só precisas de abrir duas janelas num browser como o Google Chrome.

Primeiro, terás o back-end do teu website aberto e podes acedê-lo indo ao URL “nameofourwebsite.wordpress.com/wp-admin“.

Na outra janela, tens o modo privado do Chrome para veres como o teu website aparecerá às outras pessoas.

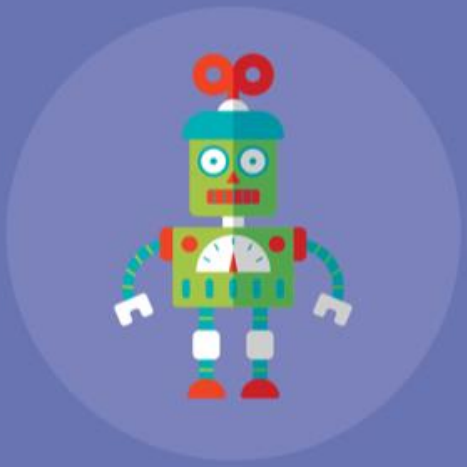
Depois de mudares qualquer coisa no back-end, irás para a janela privada e verás como realmente está o teu website.





EXERCÍCIO 2

A par com o website Wordpress, procura criar para o teu projeto uma conta no Instagram, bem como uma página do Facebook.



P95267