Programação

10.º ano

Ministério da Educação

Recursos digitais acessíveis por QR Code no manual. Versão digital em www.escolavirtual.cv

escola virtual
um serviço Porto Editora

Explora o manual digital do teu livro

Exercícios Interativos
Para resolução com *feedback* imediato.

Vídeos e interatividades
Explicam a matéria de forma motivadora.

Jogos
Exploram os conceitos curriculares de forma lúdica.

Áudios
Dão vida aos textos e ajudam a reforçar as competências linguísticas.

QuizEV
Desafiam-te a mostrares o que sabes. Podes, também, jogar com os teus amigos.

www.escolavirtual.cv

Programação

10.º ano

Explora o teu manual digital

https://escolavirtual.cv

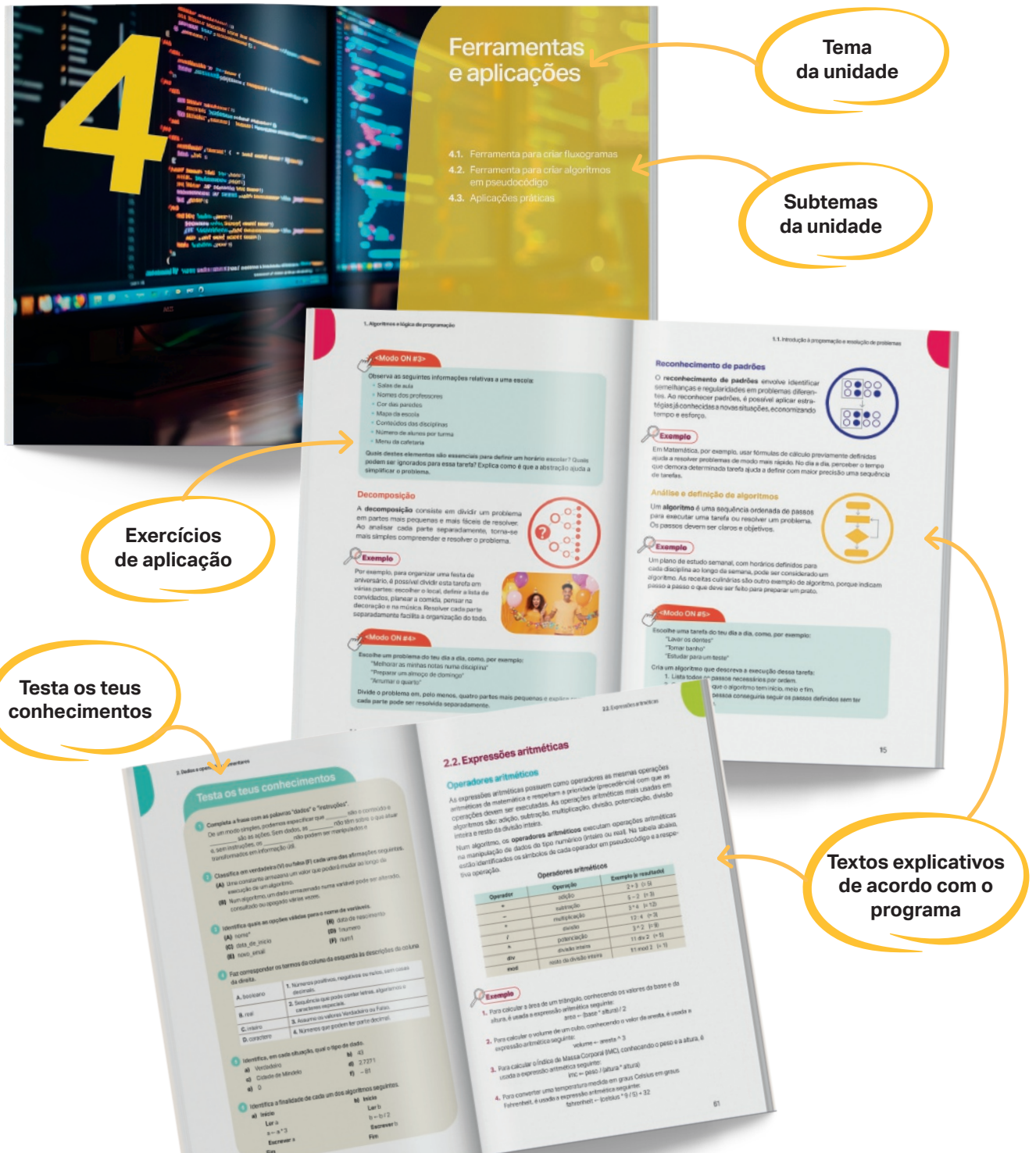
Acesso e condições de utilização em
www.escolavirtual.cv

Ministério da Educação

Conhece o teu caderno

Este manual tem como objetivo ajudar-te na iniciação à programação e está estruturado em quatro temas, cada um subdividido em subtemas, de acordo com o plano curricular do ensino secundário. Os três primeiros temas procuram orientar a tua aprendizagem em algoritmia. O último tema incorpora guias de utilização de duas ferramentas computacionais, que poderás usar a qualquer momento do ano letivo.

Cada tema e subtema são compostos por…



Índice


1

Algoritmos e lógica de programação

1.1. Introdução à programação e resolução de problemas

1.2. Conceitos básicos sobre algoritmos

1.3. Representação de algoritmos

1.4. Testar e depurar

1 Algoritmos e lógica de programação

No final deste capítulo, deverás ser capaz de:

- Reconhecer e aplicar as etapas do processo de resolução de problemas.
- Compreender a lógica de programação.
- Compreender o conceito, as características e a finalidade de um algoritmo.
- Identificar entradas, processamento e saídas em diferentes problemas.
- Distinguir e identificar linguagens naturais e linguagens formais.
- Elaborar algoritmos simples através de linguagem natural, fluxogramas e pseudocódigo.
- Reconhecer a correção, a diferença e a eficácia de diferentes estratégias da resolução de um problema.
- Desenvolver atitudes de persistência, colaboração e reflexão crítica perante os erros.

1.1. Introdução à programação e resolução de problemas

O que é a programação?

A **programação** é o processo de criar um conjunto de instruções que indicam a um computador, *tablet*, telemóvel ou outro dispositivo eletrónico o que fazer. Implica escrever instruções claras, detalhadas passo a passo, numa linguagem de programação que uma máquina consiga entender e executar para, assim, realizar tarefas.

A programação está presente em quase tudo no nosso dia a dia, como, por exemplo:

Em computadores e telemóveis:

- sistemas operativos (Windows®, iOS®, Android®, …);
- aplicações (jogos, redes sociais, mensagens,…);
- *websites* e lojas *online*;
- motores de busca;
- plataformas de vídeo e música.

Nos transportes:

- semáforos;
- GPS e aplicações de navegação;
- sistemas de controlo nos automóveis.

Em casa:

- televisões;
- eletrodomésticos;
- consolas de jogos.

No comércio e serviços:

- bancos e caixas de levantamento de dinheiro;
- caixas registadoras e pagamentos;
- equipamentos médicos.

Na escola:

- plataformas de aprendizagem *online*;
- projetores e quadros interativos;
- sistemas de gestão.

Porque é necessário programar?

Os computadores são extremamente rápidos e capazes de fazer milhões de operações por segundo, mas é necessário explicitar o que têm de fazer ao detalhe. Essa explicação é feita através de **linguagens de programação**, que os computadores conseguem interpretar.

Quando programamos, estamos a transformar uma ideia num conjunto de instruções que o computador pode executar.

Exemplo

1. Quando pressionas a tecla espaço para uma personagem saltar, existe um conjunto de instruções que diz algo como: "Se a tecla espaço for pressionada → saltar".
2. Quando uma *app* recomenda vídeos parecidos com o que viste antes, ela está a seguir instruções programadas para analisar os teus interesses.

Porque vale a pena aprender a programar?

Aprender programação não serve apenas para criar apps e jogos. A programação ajuda-te a desenvolver várias capacidades importantes:

- **raciocínio lógico** – aprender a organizar ideias e passos;
- **criatividade** – gerar ideias novas e originais;
- **resolução de problemas** – descobrir soluções eficientes e práticas;
- **autonomia tecnológica** – compreender melhor o mundo digital que te rodeia.

O que é um problema computacional?

Um **problema computacional** é uma questão que pode ser resolvida através de uma sequência ordenada de instruções precisas que um computador consegue executar. Pode ser uma tarefa muito simples, como calcular a média de um conjunto de números, ou envolver estruturas de instruções mais complexas, como reconhecer padrões em imagens.

A palavra computacional resulta do facto de a solução poder ser automatizada, ou seja, um computador conseguir seguir os passos sem intervenção humana.

Manual Interativo

Interatividades

Modelo Big6

Metodologia de Resolução de Problemas

Para escrever as instruções necessárias à resolução de um problema ou executar uma determinada tarefa é essencial:

- identificar qual deve ser a **saída** – o resultado desejado;
- identificar qual deve ser a **entrada** – os dados necessários;
- determinar como **processar** a entrada para obter a saída desejada.

Exemplo

No dia a dia há muitas tarefas em que podemos identificar etapas de entrada, processamento e saída. Repara nestes exemplos:

1. Fazer um bolo de banana

   Entrada: ingredientes e respetivas quantidades.

   Processamento: a receita que indica como proceder para combinar os ingredientes.

   Saída: bolo de banana.

2. Calcular o preço de um produto com desconto

   Entrada: preço original do produto e valor do desconto.

   Processamento: cálculo que permite determinar o preço final (preço final = preço original – preço original × desconto %).

   Saída: preço final do produto com desconto.

3. Montar uma mesa (embalada em peças):
   Entrada: peças e parafusos.
   Processamento: as instruções que explicam como montar as peças passo a passo.
   Saída: mesa montada.

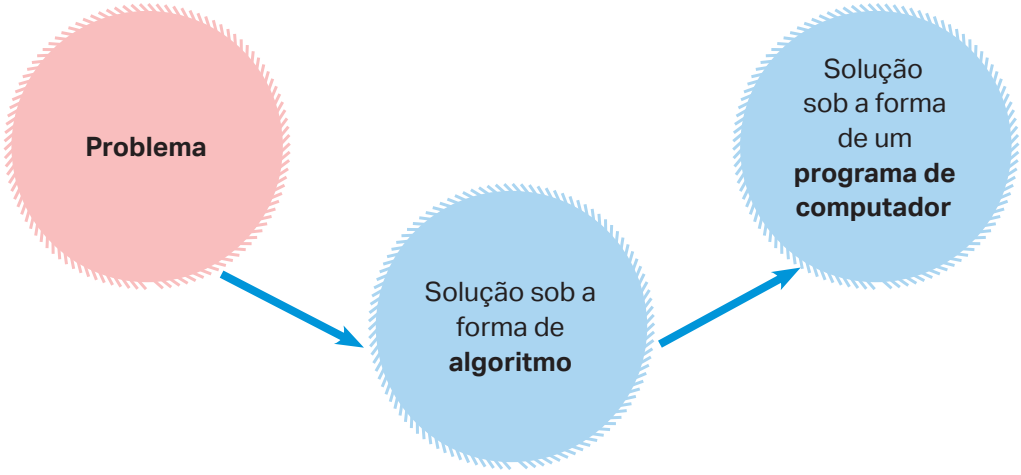
<Modo ON #1>

Identifica três tarefas do teu dia a dia em que consigas identificar as etapas de entrada, processamento e saída de um resultado. Pensa, por exemplo, em tarefas que fazes rotineiramente em casa e nos "dados de entrada" que a tua mãe te dá.

Programas e algoritmos

Um **programa** é uma sequência detalhada de instruções, escritas numa linguagem de programação compreensível para o computador, que descreve passo a passo a resolução de um problema.

Contudo, para chegar a uma solução sob a forma de um programa, é necessário que primeiro exista uma fase de interpretação e resolução do problema.



Um **algoritmo** é uma sequência de instruções que descrevem um modo de resolver um problema. Estas instruções devem estar apresentadas de forma lógica, ordenada e precisa, de modo que não haja qualquer ambiguidade na resolução do problema.

Por isso, é necessária uma análise sistemática do problema: que dados são necessários à resolução?, que resultados devem ser produzidos?, que tipo de situações anómalas podem ocorrer?, que ações desencadear neste caso?, etc.

Exemplo

1. Para construir um algoritmo que calcule a média das classificações de dois exames, é necessário analisar o problema:

- dados de entrada: classificação de cada exame;
- dados de saída: média das classificações.

Depois, escreve-se o algoritmo:

1. Pedir ao utilizador para introduzir as duas classificações.
2. Calcular a média.
3. Apresentar valor da média no ecrã.

2. Para construir um algoritmo que efetue a divisão de dois números é necessário em primeiro lugar analisar o problema:

- dados de entrada: dividendo e divisor;
- dados de saída: quociente e resto;
- situações anómalas: divisão por zero é impossível (o divisor não pode ser zero);
- ações a desencadear: mensagem de erro e pedir ao utilizador novo divisor.

Após essa análise, o algoritmo poderá ser:

1. Pedir ao utilizador para introduzir o dividendo.
2. Pedir ao utilizador para introduzir o divisor.
3. Se o divisor for zero, dar uma mensagem de erro e voltar ao passo 2. Se não for zero, passar ao passo 4.
4. Efetuar a divisão inteira.
5. Apresentar ao utilizador o quociente e o resto.

<Modo ON #2>

Escreve informalmente um algoritmo que calcule a área de um retângulo.

Diferenças entre um algoritmo e um programa de computador

	Algoritmo	Programa
Definição	Um conjunto de passos para resolver um problema.	Implementação de um algoritmo em código.
Forma	Escrito em linguagem natural, fluxograma ou pseudocódigo.	Escrito numa linguagem de programação.
Execução	Não pode ser executado diretamente por um computador.	Pode ser executado por um computador.
Foco	Focado na lógica e no contexto da resolução de problemas.	Focado na funcionalidade e na implementação.

Ciclo de desenvolvimento de um programa de computador

1

Analisar o problema
Compreender o que o programa deve fazer, qual deve ser a saída. Ter uma ideia clara de que dados (entrada) são fornecidos. Perceber qual a relação entre a entrada e a saída desejada.

2

Planear a solução
Encontrar uma sequência lógica e precisa de passos para resolver o problema – um algoritmo. O algoritmo deve incluir todos os passos, mesmo aqueles que parecem óbvios. Esta fase de planeamento também deve envolver testes e validação do algoritmo, usando dados representativos.

3

Escolher a interface e codificar
Determinar em que ferramenta se pretende construir o programa e traduzir o algoritmo para a linguagem de programação adequada. Introduzir o programa na ferramenta.

4

Testar e corrigir erros (depurar)
Localizar e remover eventuais erros do programa. Os erros podem resultar do facto de o programa não estar escrito de acordo com as regras da linguagem de programação utilizada (erros de sintaxe) ou de não estar correta a sequência de ações a ser executada (erros de semântica). Estes erros têm de ser detetados através de testes exaustivos com dados variados para os quais a saída deve ser conhecida.

5

Completar a documentação
A documentação serve para que qualquer pessoa (ou, mais tarde, o próprio programador) entenda o programa.

Pensamento computacional

O **pensamento computacional** é a capacidade de organizar o raciocínio para resolver problemas de forma lógica, eficiente e estruturada.

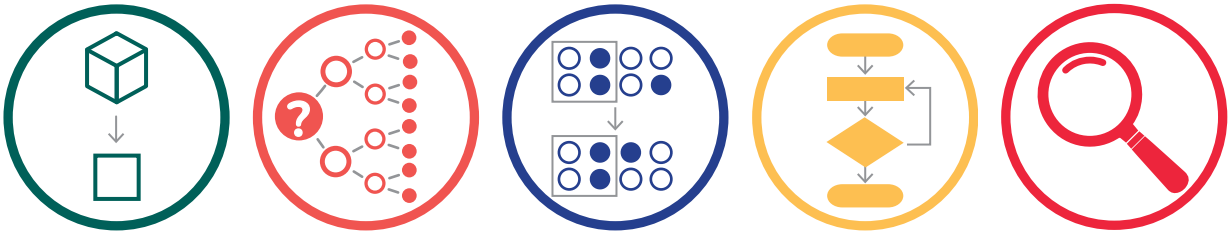
É muito mais do que simplesmente saber programar: envolve o desenvolvimento de capacidades que permitem não só saber desenhar programas e aplicações, mas também saber explicar e interpretar o mundo como um sistema complexo de processos de informação.

Manual Interativo

Jogo

Mistérios do *micro:bit* e do *Arduino*

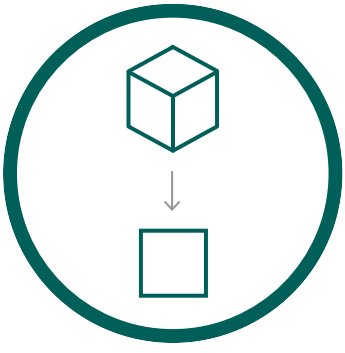
O pensamento computacional pressupõe o desenvolvimento, de forma integrada, de práticas como:



Abstração | **Decomposição** | **Reconhecimento de padrões** | **Análise e definição de algoritmos** | **Depuração**

Abstração

A **abstração** visa reduzir a complexidade de uma situação. Implica focar nos elementos importantes de um problema e ignorar os detalhes que podem não ser relevantes. A abstração permite criar modelos simplificados da realidade, facilitando a análise e a resolução de problemas.



Exemplo

Tendo em vista a exploração da ilha de Santiago por um grupo de turistas, recomenda-se a utilização de um mapa das principais vias, com a respetiva orientação, para suportar a tomada de decisões.

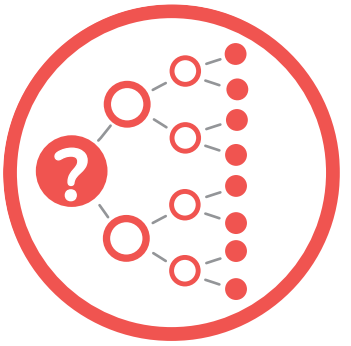
<Modo ON #3>

Observa as seguintes informações relativas a uma escola:

- Salas de aula
- Nomes dos professores
- Cor das paredes
- Mapa da escola
- Conteúdos das disciplinas
- Número de alunos por turma
- Menu da cafetaria

Quais destes elementos são essenciais para definir um horário escolar? Quais podem ser ignorados para essa tarefa? Explica como é que a abstração ajuda a simplificar o problema.

Decomposição



A **decomposição** consiste em dividir um problema em partes mais pequenas e mais fáceis de resolver. Ao analisar cada parte separadamente, torna-se mais simples compreender e resolver o problema.

Exemplo

Por exemplo, para organizar uma festa de aniversário, é possível dividir esta tarefa em várias partes: escolher o local, definir a lista de convidados, planear a comida, pensar na decoração e na música. Resolver cada parte separadamente facilita a organização do todo.

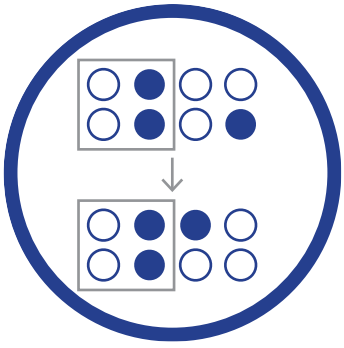
<Modo ON #4>

Escolhe um problema do teu dia a dia, como, por exemplo:

"Melhorar as minhas notas numa disciplina"
"Preparar um almoço de domingo"
"Arrumar o quarto"

Divide o problema em, pelo menos, quatro partes mais pequenas e explica como cada parte pode ser resolvida separadamente.

Reconhecimento de padrões

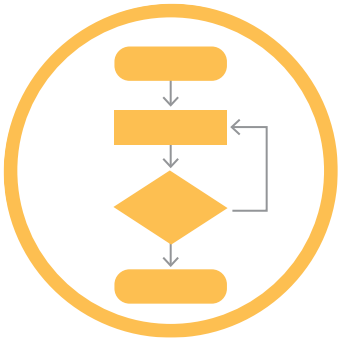


O **reconhecimento de padrões** envolve identificar semelhanças e regularidades em problemas diferentes. Ao reconhecer padrões, é possível aplicar estratégias já conhecidas a novas situações, economizando tempo e esforço.

Exemplo

Em Matemática, por exemplo, usar fórmulas de cálculo previamente definidas ajuda a resolver problemas de modo mais rápido. No dia a dia, perceber o tempo que demora determinada tarefa ajuda a definir com maior precisão uma sequência de tarefas.

Análise e definição de algoritmos



Um **algoritmo** é uma sequência ordenada de passos para executar uma tarefa ou resolver um problema. Os passos devem ser claros e objetivos.

Exemplo

Um plano de estudo semanal, com horários definidos para cada disciplina ao longo da semana, pode ser considerado um algoritmo. As receitas culinárias são outro exemplo de algoritmo, porque indicam passo a passo o que deve ser feito para preparar um prato.

<Modo ON #5>

Escolhe uma tarefa do teu dia a dia, como, por exemplo:

"Lavar os dentes"
"Tomar banho"
"Estudar para um teste"

Cria um algoritmo que descreva a execução dessa tarefa:

1. Lista todos os passos necessários por ordem.
2. Certifica-te de que o algoritmo tem início, meio e fim.
3. Avalia se outra pessoa conseguiria seguir os passos definidos sem ter qualquer dúvida.

Depuração

Testar e **depurar** é procurar e corrigir erros que possam existir nos passos definidos para a resolução de um problema e otimizar e refinar o processo.
Implica responder a questões como, por exemplo:

- Como podemos garantir que a nossa solução funciona?
- O resultado corresponde ao que era esperado?
- Como é que sabemos se conseguimos corrigir o erro?
- Como posso otimizar a estratégia tornando o processo mais eficiente?

Exemplo

A seguinte sequência de passos não está correta porque não respeita a ordem com que os passos devem ser executados:

1. Agarrar num pão.
2. Abrir o pão com uma faca.
3. Comer o pão.
4. Colocar queijo dentro do pão.

<Modo ON #6>

Observa a seguinte sequência de passos que descreve como lavar as mãos e descobre qual o erro existente:

1. Abrir a torneira.
2. Molhar as mãos.
3. Fechar a torneira.
4. Colocar sabonete nas mãos.
5. Esfregar as mãos.
6. Abrir a torneira.
7. Molhar as mãos e esfregar até sair o sabonete todo.
8. Limpar as mãos à toalha.

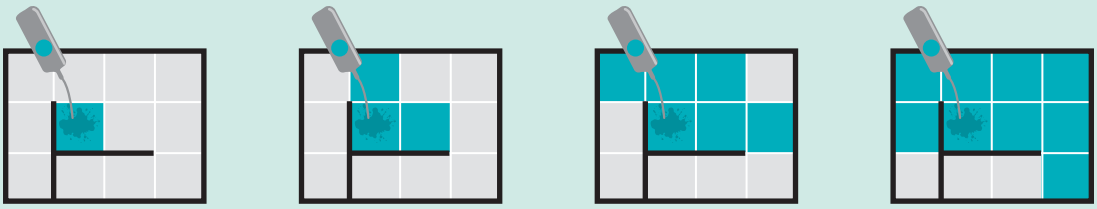
Sabias que...

O pensamento computacional é uma competência essencial no século XXI. Ao desenvolveres o teu pensamento computacional, aprendes a abordar problemas de forma sistemática e lógica, melhorando a tua capacidade de resolução de problemas em todas as áreas do conhecimento. Adicionalmente, esta competência prepara-te para um mundo cada vez mais digitalizado, no qual a capacidade de compreender e interagir com a tecnologia é crucial.
O **Bebras** é uma iniciativa internacional que pretende promover o pensamento computacional. Acede a https://bebras.pt/ para procurares algumas tarefas desafiantes!

Nestas páginas estão algumas tarefas adaptadas das provas do Bebras.

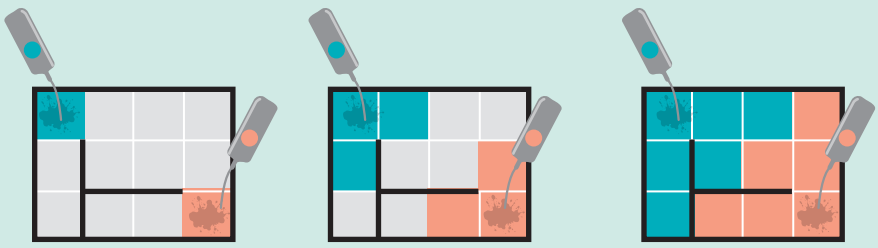
<Modo ON #7>

Num labirinto digital, quando é colocada tinta num quadrado, essa tinta espalha-se para os quadrados vizinhos a cada segundo. A tinta não se espalha através das paredes, como se pode ver na figura abaixo.



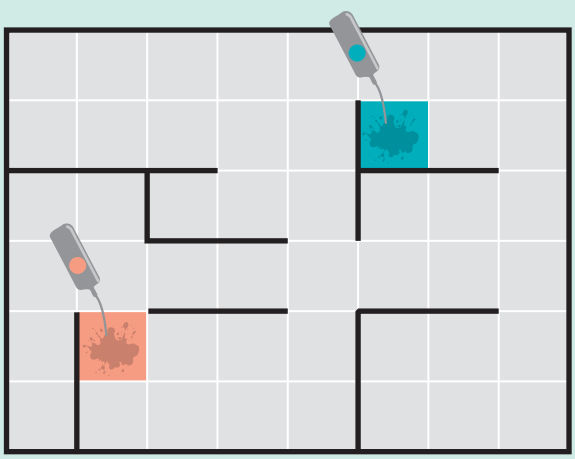
O segundos | 1 segundo | 2 segundos | 3 segundos

Se for colocada mais do que uma cor no labirinto, a primeira cor a chegar a cada quadrado vai preenchê-lo na totalidade. Se as cores chegarem ao mesmo tempo a um quadrado, o quadrado fica com a tinta da cor mais escura.



O segundos | 1 segundo | 2 segundos

Imagina que foram colocadas duas cores num labirinto, tal como na imagem abaixo.

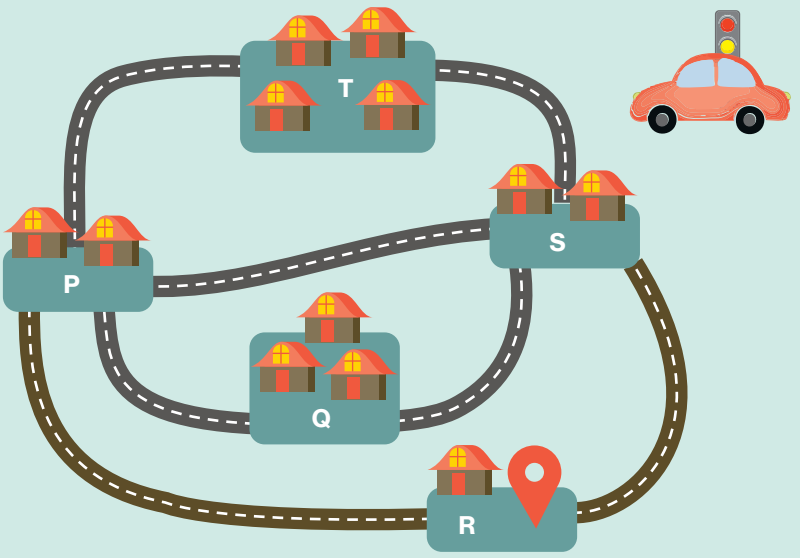


Qual será o aspeto do labirinto quando todos os quadrados estiverem preenchidos? Quantos segundos serão necessários para preencher o labirinto?

<Modo ON #8>

O CarroMaps está a recolher imagens de todas as estradas que ligam as aldeias representadas no mapa abaixo.

Devido a limitações de tempo, dispõe de apenas sete horas para captar estas imagens. Pode escolher qualquer rota, tendo em conta que é preciso exatamente uma hora para captar imagens de cada estrada que liga duas aldeias.



Partindo da aldeia R, quantos percursos diferentes poderia percorrer para captar as imagens de todas as estradas em, exatamente, sete horas?

<Modo ON #9>

O Jefferson tem uma mangueira mágica perto da sua casa:

- sempre que um pássaro poisa na árvore, nascem duas mangas;
- sempre que um lagarto trepa a árvore, a mangueira deixa cair uma manga (se existirem mangas);
- sempre que uma cobra toca na árvore, todas as mangas desaparecem.

Numa manhã, o Jefferson verificou que a mangueira mágica tinha 25 mangas e passou o resto do dia a observá-la, registando todos os animais que foram à árvore. Os desenhos que fez, por ordem, foram:

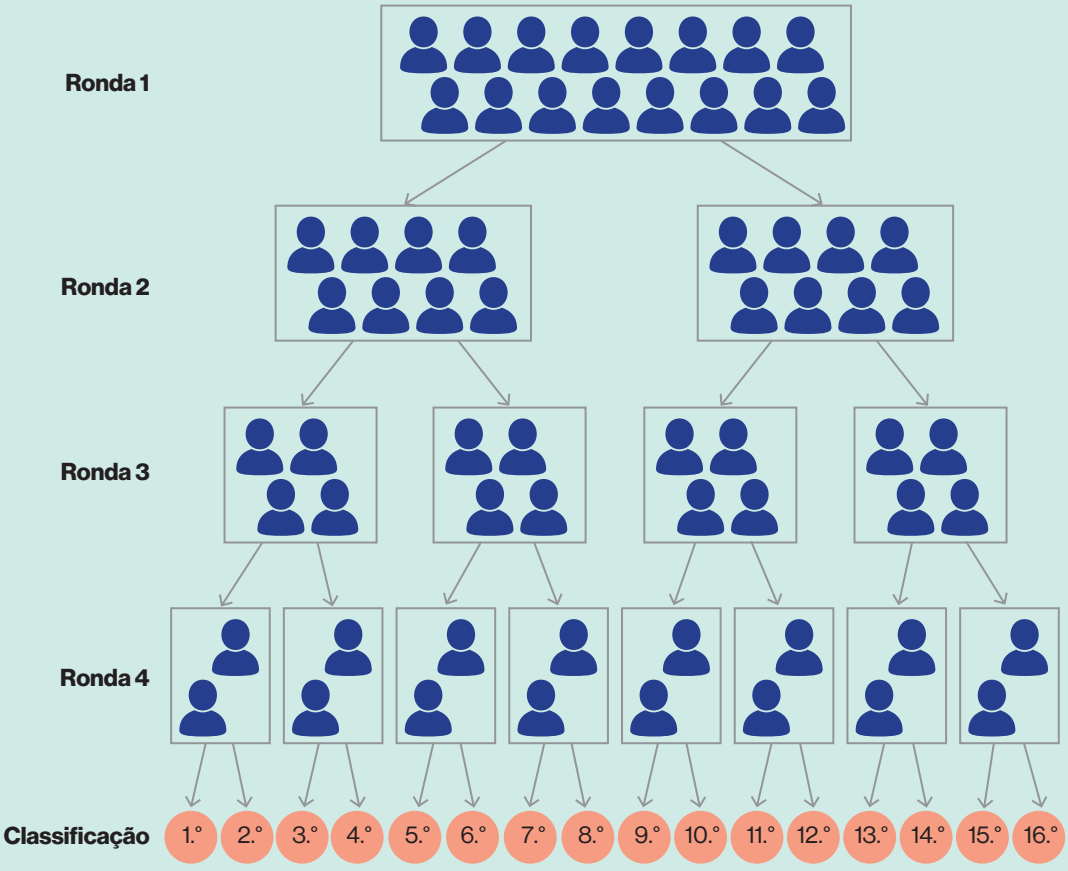
Início do dia — Final do dia

Quantas mangas tinha a árvore no final do dia?

<Modo ON #10>

Num torneio de uril, 16 jogadores competiram em 4 rondas, estabelecendo a sua classificação geral do 1.º ao 16.º lugar.

Todos os jogadores jogam juntos na ronda 1, mas depois de cada ronda os jogadores separam-se. Os jogadores vencedores seguem as setas para a esquerda para a ronda seguinte. Os jogadores derrotados seguem a seta para a direita para a ronda seguinte.



Por exemplo, um jogador que ganhe nas rondas 1 e 2, mas perca nas rondas 3 e 4, fica classificado em 4.º lugar.

O Wilson era um dos jogadores do torneio. Se ele perdeu exatamente uma única ronda (e venceu três), em qual dos seguintes lugares ele não pode ter ficado?

(A) 2.º (B) 3.º (C) 5.º (D) 7.º (E) 9.º

Testa os teus conhecimentos

1 **Classifica em verdadeira (V) ou falsa (F) cada uma das afirmações seguintes.**

(A) Um algoritmo é uma sequência de passos lógicos para resolver um problema.

(B) Um programa de computador é um algoritmo escrito numa linguagem de programação.

(C) Um algoritmo pode ser executado diretamente por um computador.

(D) O pensamento computacional é considerado uma capacidade transversal porque só é aplicável em programação.

2 **Classifica em Entrada (E), Processamento (P) ou Saída (S) cada uma das etapas seguintes.**

(A) Ler a idade do utilizador.

(B) Adicionar dois números.

(C) Apresentar o resultado no ecrã.

(D) Ler o valor do salário.

(E) Calcular o desconto.

3 **Considera um programa que lê o preço por hora definido no contrato de um trabalhador e a quantidade de horas trabalhadas num mês, calcula o salário e apresenta o resultado.**

a) Quais são os dados de entrada?

b) Qual é o processamento realizado?

c) Quais são os dados de saída?

4 **O que é a abstração no contexto do pensamento computacional? Seleciona a opção correta.**

(A) Aumentar a complexidade de um problema.

(B) Simplificar um problema, destacando os aspetos mais importantes.

(C) Focar nos detalhes.

(D) Ignorar padrões.

5 **Qual é o benefício da decomposição de problemas? Seleciona a opção correta.**

(A) Ignorar os aspetos essenciais do problema.

(B) Ignorar padrões.

(C) Aumentar a complexidade de um problema.

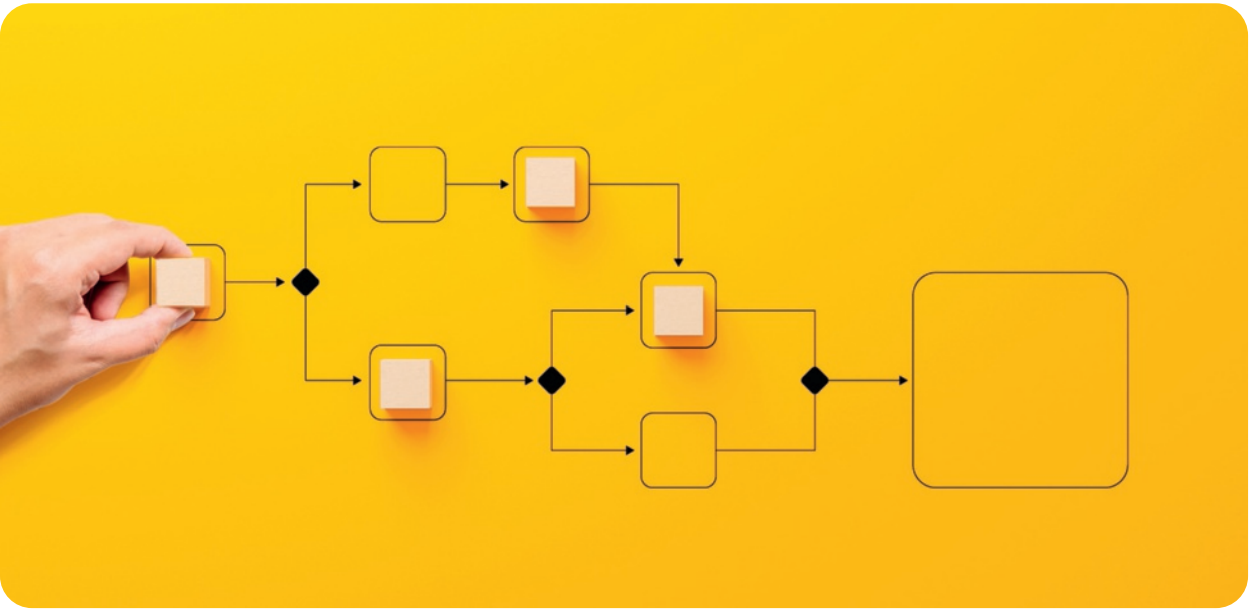
(D) Facilitar a resolução ao dividir o problema em partes menores.

1.2. Conceitos básicos sobre algoritmos

Definição de algoritmo

Tal como referido anteriormente, um algoritmo é uma sequência de passos lógicos que permite solucionar problemas. Usamos algoritmos diariamente sempre que estabelecemos um plano mental para realizar uma determinada tarefa, a qual exija executar um conjunto de passos até atingir o objetivo desejado.

Um **algoritmo** consiste num conjunto finito e bem definido de instruções que descrevem os passos lógicos necessários à realização de uma tarefa ou resolução de um problema.
Dada uma entrada, a execução do algoritmo conduz a uma saída.



Exemplo

Um dos vários exemplos do uso de algoritmos no dia a dia são as receitas culinárias, pois estas possuem um conjunto de passos que devem ser seguidos para obter o resultado esperado.

Receita de cachupa

1. Separar os ingredientes:
   - ½ kg de milho para cachupa;
   - 1 copo de feijão branco;
   - 1 copo de feijão americana;
   - ½ kg de feijão verde;
   - 1 frango inteiro;
   - 1 kg de porco ou costeleta de carne;
   - 1 kg de repolho grosseiramente cortado em cubos;
   - 1 kg de batata-doce fresca descascada e cortada em pedaços pequenos;
   - 1 kg de abóbora de inverno;
   - 1 kg de mandioca;
   - ½ kg de cenouras;
   - 1 kg de inhame;
   - 1 kg de banana verde;
   - 1 kg de tomates maduros;
   - 2 cebolas grandes;
   - 3 dentes de alho;
   - 150 dl de azeite;
   - 150 dl de vinho branco;
   - 1 colher de chá de sal;
   - 2 folhas de louro.

2. Na véspera, colocam-se o milho e os feijões de molho em água fria.
3. No próprio dia, levam-se a cozer estes ingredientes, num tacho, com água (2h30m).
4. Depois de cozidos o milho e os feijões, adicionam-se a couve, a mandioca, a banana, o inhame, a cenoura e outros legumes.
5. Numa caçarola, colocam-se a cebola, alho e azeite e leva-se a refogar o tomate, as carnes e o chouriço; junta-se ao refogado vinho branco e adiciona-se a este preparado os ingredientes já cozidos.
6. Tempera-se com sal e deixa-se apurar 30 minutos em lume brando.
7. Serve-se numa travessa com rodelas de chouriço.

<Modo ON #11>

1. **Escreve um algoritmo que descreva os passos necessários à preparação de uma sande de queijo.**
2. **Escreve um algoritmo que verifique se uma senha (*password*) é forte.**
   Regras sugeridas:
   – ter, no mínimo, 8 caracteres;
   – ter, pelo menos, uma letra maiúscula;
   – ter, pelo menos, um algarismo.

☑ Not@ que:

Podem existir muitos caminhos diferentes para resolver o mesmo problema. Ou seja, podem ser criados vários algoritmos diferentes para resolver o mesmo problema. Na maioria das vezes, não é possível decidir qual o melhor algoritmo, mas existem alguns critérios que podem ser usados para comparar:

- Qual cumpriu a tarefa mais rapidamente?
- Qual cumpriu a tarefa mais eficientemente?
- Qual é mais fácil de validar?

Exemplo

Exemplo de três algoritmos diferentes que permitem determinar se um número é par ou ímpar:

Algoritmo 1:

– Pedir um número ao utilizador.
– Se o resto da divisão do número por 2 for 0, então escrever "O número é par".
– Senão, escrever "O número é ímpar".

Algoritmo 2:

– Pedir um número ao utilizador.
– Se o algarismo das unidades for 0, 2, 4, 6 ou 8, então escrever "O número é par".
– Senão, escrever "O número é ímpar".

Algoritmo 3:

– Pedir um número ao utilizador.
– Se o resto da divisão do algarismo das unidades por 2 for 0, então escrever "O número é par".
– Senão, escrever "O número é ímpar".

O primeiro algoritmo é muito mais lento do que os outros, pois, por exemplo, para saber se o número 34123443134 é par, seria necessário uma divisão enorme!

Propriedades de um algoritmo

Na elaboração de um algoritmo devem ser especificadas instruções claras e precisas que resultem na solução do problema proposto. Para que um algoritmo seja útil e eficaz, deve respeitar um conjunto de propriedades fundamentais. Estas propriedades garantem que qualquer pessoa (ou computador) consegue compreender e executar os passos corretamente.

Precisão	Um algoritmo deve ser **preciso** e não conter ambiguidades. Cada instrução deve dizer exatamente o que fazer e só pode ter uma única forma de ser interpretada. Por exemplo, em vez da instrução "misturar durante algum tempo", deve ser dada a instrução "misturar durante 30 segundos".
Ordem	As instruções devem aparecer numa **sequência lógica e coerente**, onde cada passo depende ou complementa o anterior. Por exemplo, não é possível ter a instrução "imprimir o resultado" antes desse resultado ter sido calculado.
Completude	Um algoritmo deve ser **completo**. Todas as ações devem ser descritas através de instruções. Se as instruções forem executadas, o resultado esperado será sempre atingido. Por exemplo, não é possível calcular a área de um retângulo sem que primeiro tenham sido dadas as medidas do comprimento e da largura do retângulo (entrada).
Finitude	Um algoritmo deve ser **finito** e terminar ao fim de um número limitado de passos. Por exemplo, a instrução "substituí n por $n \times 2$, enquanto n for positivo" pode originar uma sequência infinita de passos (se o número n for positivo, o seu dobro vai continuar sempre a ser um número positivo).

<Modo ON #12>

1 **Dos exemplos abaixo, qual não pode ser considerado um algoritmo? Justifica.**

(A) Receita de doce de coco.

(B) Guia de instalação de um *software*.

(C) Lista de preços de uma padaria.

2 **Qual é o valor lógico da afirmação: "Um algoritmo é uma sequência de passos lógicos infinitos e não ambíguos que permitem solucionar problemas."? Justifica.**

3 **Ordena corretamente os passos do algoritmo.**

– Escrever a soma obtida.
– Calcular a soma de **a** com **b**.
– Início.
– Ler **a**, **b**.
– Fim.

4 **Qual é o erro do algoritmo abaixo?**

– Início.
– Ler **x**.
– Calcular o resultado da soma de **x** com **y**.
– Escrever o resultado obtido.
– Fim.

Manual Interativo

Exercícios

Algoritmos de ordenação

Algoritmo de ordenação por seleção

Algoritmo de ordenação por bolha

Algoritmos de pesquisa

Algoritmo de pesquisa binária

Sabias que...

A **tríade fundamental** de um algoritmo é composta por três tipos essenciais de instruções que permitem construir qualquer programa:

- **Estruturas sequenciais** – São instruções simples, que são executadas pela ordem em que aparecem. Por exemplo:
  Escrever no ecrã "O computador vai encerrar".
- **Estruturas condicionais** – São instruções de decisão, ou seleção, que permitem alternar entre um ou outro conjunto de instruções após a avaliação lógica de uma condição. Por exemplo:
  Se a temperatura for menor que 15 ºC então vestir casaco.
- **Estruturas de repetição** – São instruções de repetição que permitem executar de forma repetitiva um conjunto de instruções. Esta execução depende do valor lógico de uma condição que é testada em cada iteração para decidir se a execução do ciclo continua ou termina. Por exemplo:
  Enquanto houver pão, continuar a fazer sandes.

Testa os teus conhecimentos

1 **Classifica em verdadeira (V) ou falsa (F) cada uma das afirmações seguintes.**

(A) Um algoritmo consiste num conjunto infinito e bem definido de instruções que descrevem os passos lógicos necessários à realização de uma tarefa.

(B) Podem ser criados vários algoritmos diferentes para resolver o mesmo problema.

(C) Um problema computacional é sempre uma tarefa muito simples.

2 **Faz corresponder os termos da primeira coluna, identificados com letras, às descrições da segunda coluna, identificadas com números.**

A. Finitude	**1.** Dados lidos pelo algoritmo.
B. Entrada	**2.** As instruções são precisas, sem ambiguidades.
C. Completude	**3**. As instruções devem aparecer numa sequência lógica e coerente.
D. Precisão	**4.** Dados fornecidos pelo algoritmo.
E. Ordem	**5.** Número limitado de passos.
F. Saída	**6.** Se as instruções forem executadas, o resultado esperado será sempre atingido.

3 **Considera o algoritmo seguinte:**

1. Ler um número inteiro.
2. Adicionar 10 ao número lido.
3. Duplicar o resultado anterior.
4. Apresentar o resultado final no ecrã.

a) O algoritmo possui entrada? Se sim, qual?

b) Qual é a saída do algoritmo?

c) O que irá aparecer no ecrã se como entrada for lido o número 7?

4 **Elabora um algoritmo que resolva o seguinte problema.**

Calcular o dobro de um número introduzido.

1.3. Representação de algoritmos

Existem diversas formas de representação de algoritmos, podendo ser usadas linguagens naturais ou linguagens formais. Cada uma das formas de representação possui vantagens e desvantagens e compete ao programador escolher qual utilizar.

Formas mais comuns de representação de um algoritmo

Linguagem natural | **Fluxogramas** | **Pseudocódigo**

Linguagem natural

Um algoritmo pode ser representado numa linguagem natural, ou seja, a sequência de passos a executar para a resolução de um problema pode ser descrita na nossa língua nativa (português).

Exemplo

Exemplo de um algoritmo representado em linguagem natural:

1. Ler as classificações obtidas em dois testes.
2. Calcular a média aritmética das classificações.
3. Se a média for maior ou igual a 10, então escrever "O aluno é aprovado".
4. Senão, escrever "O aluno é reprovado".

<Modo ON #13>

Cria um algoritmo em linguagem natural que apresente as instruções necessárias para executar as seguintes tarefas:

a) mudar um pneu;

b) fazer um doce de coco.

☑ Not@ que:

Um **aspeto positivo** do uso da linguagem natural na representação de algoritmos é não ser necessário aprender novos conceitos.

Um **aspeto negativo** do uso da linguagem natural é poderem existir diversas interpretações e algumas ambiguidades, dificultando a transcrição desse algoritmo para um programa computacional.

Manual Interativo

Vídeo

Como se constrói um fluxograma

Contrariamente a uma linguagem natural, uma linguagem formal é projetada com um determinado fim e delineada para evitar ambiguidades e equívocos. Numa linguagem formal, os termos e as regras são estabelecidos previamente, antes da linguagem ser usada para representar um algoritmo.

Vais aprender a utilizar fluxogramas e pseudocódigo.

Fluxogramas

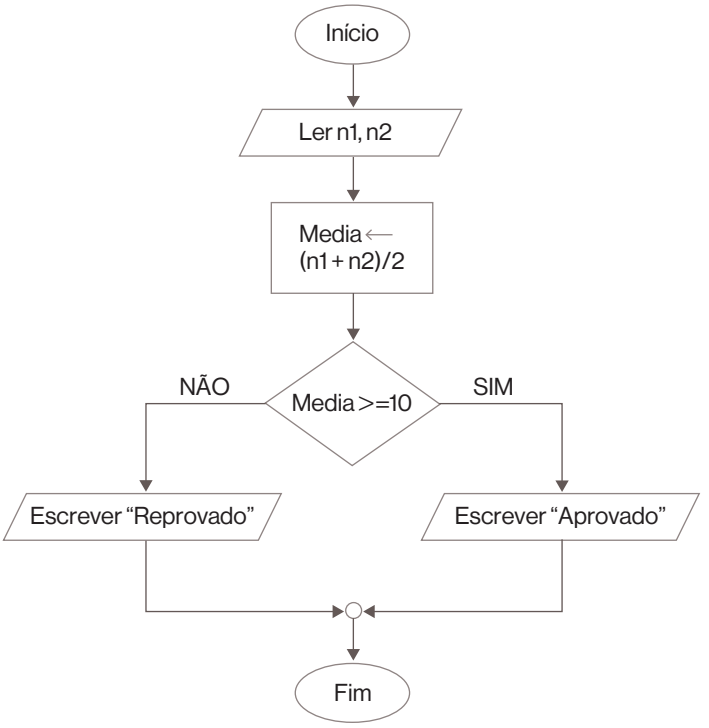
O que é um fluxograma?

Um fluxograma é uma representação gráfica de um algoritmo que descreve graficamente a sequência de passos a executar para resolver um determinado problema. Utiliza símbolos gráficos padronizados, em que formas geométricas diferentes representam instruções diferentes.

Num fluxograma, a representação visual de um algoritmo facilita a sua análise, compreensão e comunicação.

Exemplo

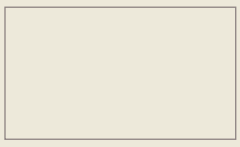
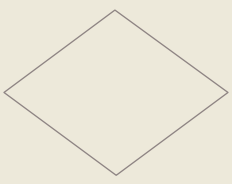
Exemplo de um algoritmo representado em fluxograma:



Componentes de um fluxograma

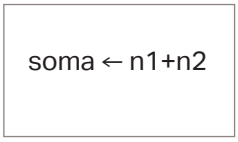
Num fluxograma, são usados símbolos padronizados para representar as instruções, em que formas geométricas diferentes representam instruções diferentes, e as linhas de fluxo são usadas para mostrar o modo como essas instruções estão interligadas.

Os símbolos mais comuns de que irás precisar para escrever os teus algoritmos são:

Oval		É usado para representar o **início** e o **fim** do algoritmo.
Paralelogramo		É usado para operações de **entrada** e **saída** (a informação a ler ou a escrever é indicada no interior).
Retângulo		É usado para representar **processos** (operações de manipulação dos dados, como, por exemplo, "Calcular a soma").
Retângulo (traçado)		É usado para representar **subprocessos** (permite executar outro algoritmo e depois regressar ao algoritmo atual).
Losango		É usado para indicar um ponto de **decisão** (tem uma entrada e duas saídas correspondentes ao resultado do teste ser verdadeiro ou falso, por exemplo, "O número é positivo?".
Linha de fluxo (seta)		É usado para ligar os outros símbolos, indicando a **sequência de operações**.
Conector		É usado para **ligar** diferentes linhas de fluxo.

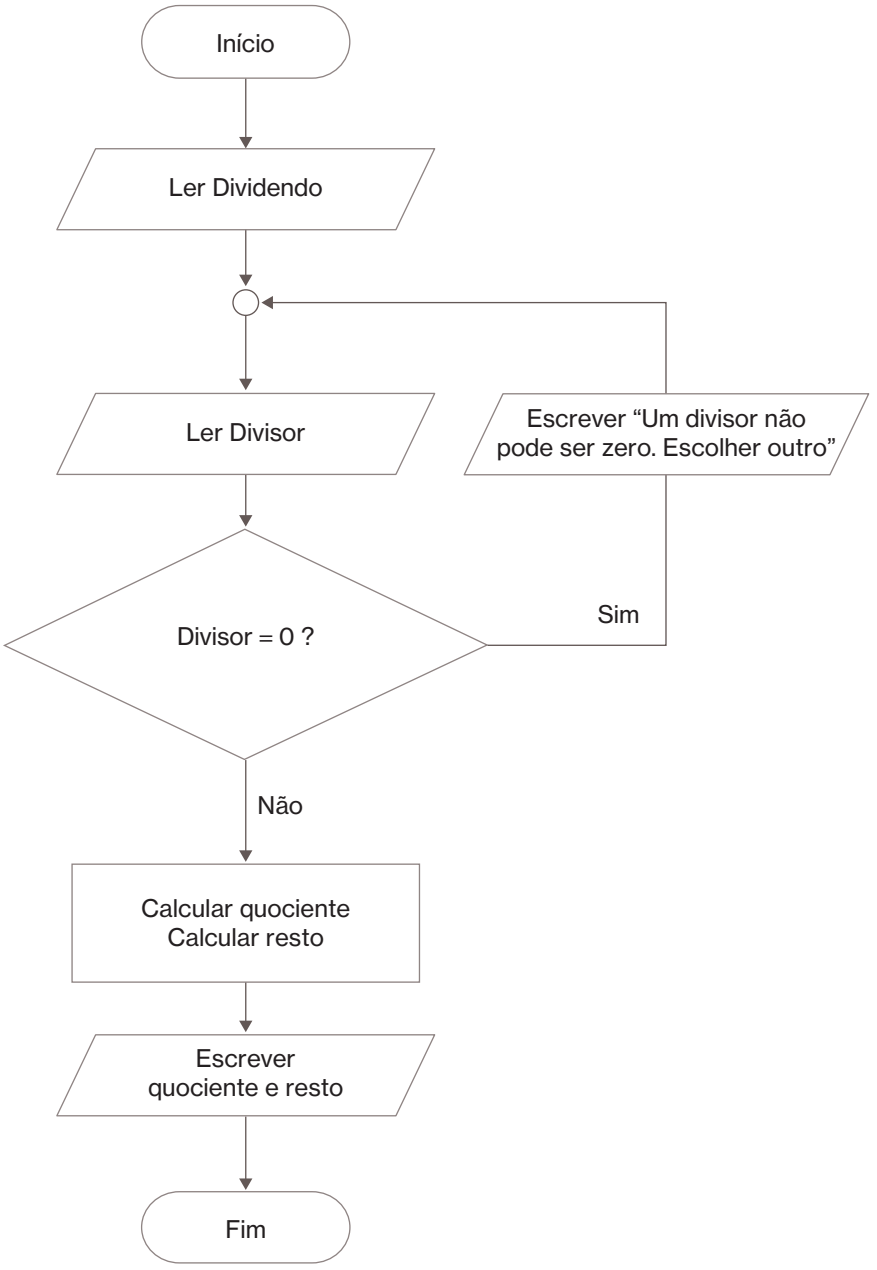
A instrução a executar em cada passo deve ser descrita com maior detalhe com texto no interior do símbolo.

Por exemplo:



Exemplo

Exemplo de um algoritmo em fluxograma que calcula a divisão inteira entre dois números inteiros, depois de garantir que o divisor introduzido é um número diferente de zero.



<Modo ON #14>

Apresenta sob a forma de fluxograma o algoritmo seguinte:

- Início
- Ler num1
- Ler num2
- Calcular o produto (produto = num1 × num2)
- Calcular a divisão do produto por 3 (quociente = produto / 3)
- Escrever o produto
- Escrever o quociente
- Fim

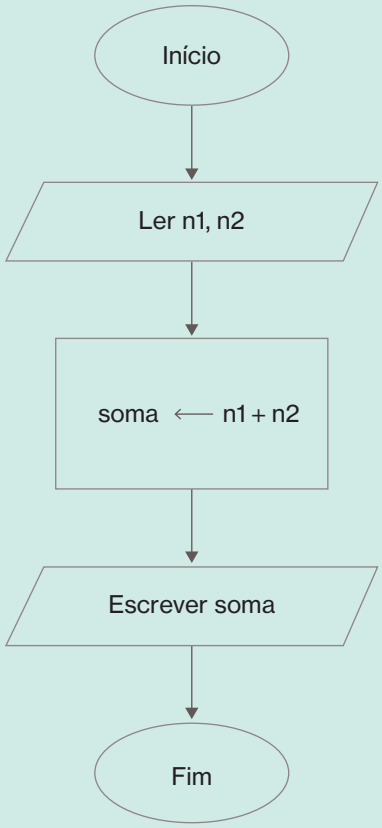
☑ Not@ que:

Um **aspeto positivo** do uso de fluxogramas é que usualmente a compreensão de elementos gráficos é mais simples do que a compreensão de texto. É mais intuitivo ler o algoritmo e entender a tarefa que é executada.

Um **aspeto negativo** do uso de fluxogramas é o facto de o algoritmo poder não estar suficientemente detalhado para facilitar a sua transcrição para uma linguagem de programação que possa ser lida por um computador.

<Modo ON #15>

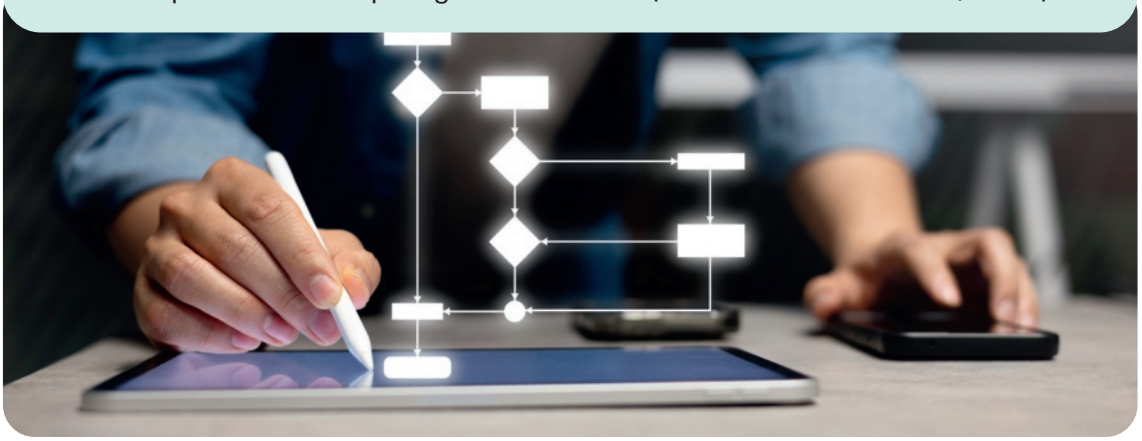
1 Observa o seguinte fluxograma e determina qual a tarefa que é executada.



2 Cria em fluxograma:

a) um algoritmo que calcule a média de três números introduzidos pelo utilizador;

b) um algoritmo que leia uma temperatura introduzida em graus Celsius e depois a converta para graus Fahrenheit (Fahrenheit = Celsius × 1,8 + 32).



Pseudocódigo

O que é o pseudocódigo?

O **pseudocódigo** é uma linguagem formal de representação que descreve a sequência de instruções de um algoritmo usando uma linguagem parecida com a linguagem comum (inglês, português, ...), através de frases com construções próximas das que são usadas em muitas linguagens de programação.

O pseudocódigo segue uma estrutura semelhante a muitas linguagens de programação, com comandos similares e requisitos de que estas linguagens necessitam, facilitando a codificação para qualquer linguagem que se deseje.

Estrutura de um algoritmo em pseudocódigo

Algoritmo < nome_do_algoritmo >

Var
  < declaração_de_variáveis >

Início
  < corpo_do_algoritmo >

Fim

Exemplo

O algoritmo seguinte permite calcular a média de dois números dados pelo utilizador:

Algoritmo Média_dois_numeros

Var
  n1, n2, media : real

Início
  Ler n1, n2
  media ← (n1 + n2)/2
  Escrever media

Fim

☑ **Not@ que:**

1. Em pseudocódigo, cada linha tem uma instrução.
2. As palavras **Algoritmo**, **Var**, **Início** e **Fim** estão a negrito porque correspondem a palavras reservadas que expressam uma instrução ou identificador específico e não podem ser usadas para outra função.

Instruções em pseudocódigo

Declaração de variáveis

As variáveis a utilizar no algoritmo devem ser identificadas no início do algoritmo, usando-se a instrução **Var**. Tal como na maioria das linguagens de programação, deve também ser referido o tipo de variável (inteiro, real, *caractere**, lógico, etc.).

Exemplo:

```
Var
x, y : real
a, b : inteiro
```

Entrada e saída de dados

A entrada e saída de dados faz-se através das instruções Ler e Escrever.

Exemplos:

```
Ler x, y                          (entrada)
Escrever "A média é", media       (saída)
```

Atribuição

Durante um processamento da informação, a atribuição de um valor a uma variável (ou alteração do valor) é feita usando o símbolo ←.

Exemplos:

```
x ← 5
media ← (n1 + n2 )/2
```

Comentário

Para se introduzir comentários entre as linhas de instruções (que expliquem partes do algoritmo) deve ser usado o símbolo **//** no início da frase.

Exemplos:

```
// este algoritmo serve para calcular uma média
// a variável count serve de contador de ocorrências
```

* Cáracter assume a grafia *caractere* no contexto de programação.

Estruturas

Para representar instruções relativas a estruturas condicionais e de repetição, são usadas palavras específicas (palavras reservadas) que irás aprender mais tarde neste manual.
Exemplo: **Se**, **Então**, **Senão**, **Enquanto**, **Repetir**, **Até**, entre outras.

Manual Interativo
Exercício
Algoritmos e pseudocódigo

Ao escreveres algoritmos em pseudocódigo deves tentar seguir algumas boas práticas que ajudam à leitura e compreensão:

- incluir comentários que expliquem as instruções;
- utilizar nomes significativos para as variáveis e constantes (que ajudem a identificar o conteúdo);
- usar indentação das linhas para facilitar a leitura.

Exemplo

O algoritmo seguinte permite calcular o perímetro de um retângulo, depois de as medidas dos lados do retângulo serem dadas pelo utilizador:

```
Algoritmo Perímetro_retângulo

Var
  c: real
  l: real
  p: real

Início
// leitura das medidas do retângulo
  Escrever "Qual é o comprimento do retângulo?"
  Ler c
  Escrever "Qual é a largura do retângulo?"
  Ler l
  // cálculo do perímetro
  p ← c + c + l + l
// apresentação do valor do perímetro no ecrã
  Escrever "O perímetro do retângulo é", p

Fim
```

<Modo ON #16>

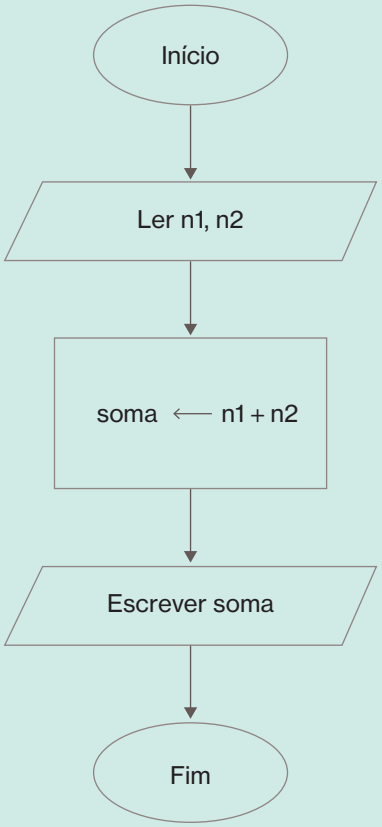
Observa o seguinte algoritmo em pseudocódigo e determina qual a tarefa que é executada:

```
Algoritmo Nome
Var
  a: real
  b: real

Início
  Escrever "Quais são os números?"
  Ler a,b
  a ← b-a
  Escrever a

Fim
```

Propõe um nome adequado para o algoritmo.

<Modo ON #17>

Escreve em pseudocódigo o seguinte algoritmo:



☑ Not@ que:

Um **aspeto positivo** do uso de pseudocódigo é o facto de permitir uma representação precisa e detalhada de um algoritmo sem exigir o conhecimento específico de uma linguagem de programação.

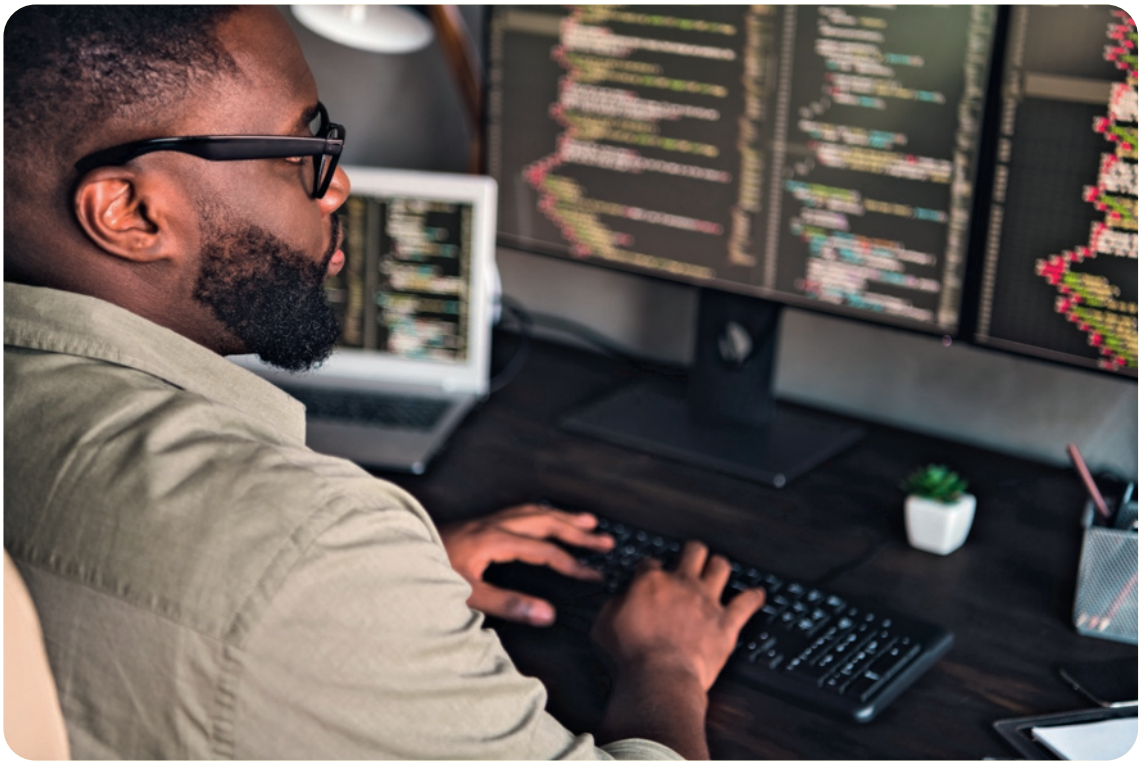
Um **aspeto negativo** do uso de pseudocódigo é o facto de exigir uma aprendizagem prévia da sintaxe e regras do pseudocódigo, não sendo inicialmente tão intuitivo.

<Modo ON #18>

Cria um algoritmo em pseudocódigo que:

a) calcule a média de três números introduzidos pelo utilizador;

b) leia uma temperatura introduzida em graus Celsius e depois a converta para graus Fahrenheit (fórmula: F = C × 1,8 + 32).



Testa os teus conhecimentos

1 **Completa as seguintes afirmações.**

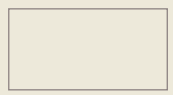
a) Um _____ pode ser escrito em linguagem natural, fluxograma ou pseudocódigo. Um _____ é escrito numa linguagem de programação.

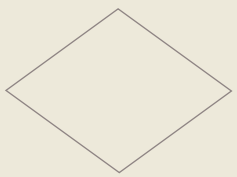
b) Contrariamente a uma linguagem _____, uma linguagem _____ é projetada previamente com um determinado fim e delineada para evitar ambiguidades e equívocos.

2 **Identifica a função de cada um dos seguintes símbolos utilizados no desenho de fluxogramas:**

a) 

b) 

c) 

d) 

e)

f) 

3 **Classifica em verdadeira (V) ou falsa (F) cada uma das seguintes afirmações relativas à representação de algoritmos em pseudocódigo.**

(A) Uma linha pode ter várias instruções.

(B) As palavras reservadas que expressam uma instrução ou identificador específico não podem ser usadas para outra função.

(C) A entrada de dados faz-se através da instrução Ler.

(D) A saída de dados faz-se através da instrução Escrever.

(E) Não se deve incluir comentários que expliquem as instruções.

4 **Elabora um algoritmo, em fluxograma, que tenha os seguintes dados de entrada e de saída:**

Entrada: Distância percorrida por um automóvel (em km)
Quantidade de combustível consumida (em litros)

Saída: Consumo médio do automóvel na viagem

Relembra que o consumo médio de um automóvel é o número de litros de gasolina gastos a cada 100 km.

Ou seja, consumo = quantidade / distância × 100.

5 **Cria em pseudocódigo um algoritmo que pergunte o nome e a idade ao utilizador e que depois escreva uma frase semelhante a "Olá (nome)! O triplo da tua idade é (idade)".**

1.4. Testar e depurar

O teste de um algoritmo permite avaliar se as instruções dadas permitem executar a tarefa pretendida. Para isso, faz-se o ***tracing*** do algoritmo, que consiste em escolher valores de entrada e ir seguindo por ordem as instruções, simulando cálculos e verificando os resultados obtidos. O *tracing* permite detetar e corrigir os possíveis erros existentes.

Exemplo

O algoritmo seguinte pretende calcular o perímetro de um triângulo.

```
Algoritmo Perímetro_triângulo
Var
  n1, n2, n3, p: real
Início
  Escrever "Quais as medidas dos lados do triângulo?"
  Ler n1, n2, n3
  p ← n1 + n2
  Escrever "O perímetro do triângulo é", p
Fim
```

O ***tracing*** do algoritmo para um triângulo com medidas dos lados 3, 4 e 5 conduz aos seguintes resultados:

n1 = 3
n2 = 4
n3 = 5
p = 3 + 4 = 7

Ao analisar o cálculo de p, verificamos que existe um erro num passo do algoritmo pois estão apenas a ser adicionados dois dos lados do triângulo. Para corrigir o algoritmo, devemos substituir a instrução do cálculo do perímetro *p* por:

p ← n1 + n2 + n3

Fazendo novamente o *tracing*, verificamos que o cálculo do perímetro está correto:

n1 = 3
n2 = 4
n3 = 5
p = 3 + 4 + 5 = 12

Sabias que...

Depurar, em inglês, diz-se **debug**.
Esta palavra, muito comum em programação, significa procurar, isolar e corrigir erros (*bugs*) num programa.

<Modo ON #19>

Considera o algoritmo que permite trocar os valores de duas variáveis a e b, entre si.

```
Algoritmo Trocar_valores
Var
  a, b, c: inteiro
Início
  a ← 3
  b ← 5
  c ← a
  a ← b
  b ← c
  Escrever a, b
Fim
```

Faz o *tracing* deste algoritmo.

Exemplo

Considera o seguinte algoritmo:

```
Algoritmo Soma
Var
  n1, n2, s: inteiro
Início
  Escrever "Digite um número inteiro"
  Ler n1
  Escrever "Digite outro número inteiro"
  Ler n2
  s ← n1 + n2
  Escrever "Soma =", s
Fim
```

Por vezes, para se fazer o *tracing* de um algoritmo, usa-se uma tabela para registar os valores passo a passo. Observa o registo para um teste do algoritmo com os números 11 e 13:

	n1	n2	s	Saída
1.º passo	11			
2.º passo	11	13		
3.º passo	11	13	24	
4.º passo	11	13	24	24

<Modo ON #20>

Considera o algoritmo seguinte que permite calcular o perímetro de um triângulo.

Algoritmo Perímetro_triângulo

Var

a, b, c, p: real

Início

Escrever "Quais as medidas dos lados do triângulo?"

Ler a, b, c

p ← a + b + c

Escrever "O perímetro do triângulo é", p

Fim

Usa as tabelas abaixo para fazeres o *tracing* dos testes ao algoritmo para a = 7, b = 3, c = 6

	a	b	c	p	Saída
1.º passo					
2.º passo					
3.º passo					

para a = 8, b = 8, c = 5:

	a	b	c	p	Saída
1.º passo					
2.º passo					
3.º passo					

para a = 9, b = 4, c = 11:

	a	b	c	p	Saída
1.º passo					
2.º passo					
3.º passo					

Relembra...

Em algoritmia e programação, usa-se o termo **sintaxe** para referir a forma como os comandos/instruções devem ser escritos, para que possam ser entendidos e executados pelos programas de computador. Uma palavra mal escrita impede que um programa seja executado.

O termo **semântica** é usado para referir o significado de uma instrução ou conjunto de instruções. Uma violação das regras semânticas é similar a uma frase em linguagem natural que, embora use palavras escritas corretamente, não faça sentido e seja incompreensível.

Testa os teus conhecimentos

1 **Faz corresponder os termos da primeira coluna, identificados com letras, às descrições da segunda coluna, identificadas com números.**

A. Linguagem de programação	**1.** Forma como os comandos e instruções devem ser escritos.
B. Sintaxe	**2.** Significado de um conjunto de instruções.
C. Semântica	**3.** Linguagem projetada previamente para um determinado fim.
D. Linguagem natural	**4.** Algoritmo escrito numa linguagem de programação.
E. Linguagem formal	**5.** Linguagem com instruções que os computadores conseguem interpretar.
F. Programa	**6.** Linguagem não projetada previamente para um determinado fim.

2 **Considera os algoritmos seguintes:**

Algoritmo Calculo1
Var
  n: real
Início
  Ler n
  n ← n + 10
  n ← n × 2
  Escrever n
Fim

Algoritmo Calculo2
Var
  n: real
Início
  Ler n
  n ← n × 2
  n ← n + 10
  Escrever n
Fim

a) Faz o *tracing* de cada um dos algoritmos para n = 5.

b) Qual é a diferença entre os dois algoritmos?

3 **Considera um algoritmo que lê dois números, calcula o dobro da soma dos dois números e apresenta o resultado.**

a) Representa o algoritmo em pseudocódigo e em fluxograma.

b) Faz o *tracing* do algoritmo com os números:

(i) 6 e 8;

(ii) 50 e 20.


2

Dados e operações elementares

2 Dados e operações elementares

No final deste capítulo, deverás ser capaz de:

- Identificar e utilizar diferentes tipos de dados em programas.
- Compreender o conceito de variável e o seu papel no armazenamento e manipulação de informação.
- Reconhecer e aplicar diferentes operadores aritméticos, relacionais e lógicos, respeitando as suas regras de prioridade.
- Utilizar expressões e operações simples para resolver problemas práticos.
- Desenvolver atitudes de raciocínio lógico, precisão e rigor na escrita de algoritmos.

2.1. Variáveis e tipos de dados

Informações: dados e instruções

Um computador é uma máquina eletrónica capaz de receber, processar, armazenar e transmitir informações.

Para compreender o seu funcionamento, é importante perceber o que se entende por informações processadas por um computador. Essas informações podem ser classificadas em dados e instruções, que permitem ao computador executar tarefas e produzir resultados.

Dados

Os **dados** são os elementos sobre os quais o computador trabalha. São valores que podem ser introduzidos pelo utilizador ou obtidos a partir de outros sistemas.

Exemplos de dados:
– números: 25, 3,14, 100;
– letras e palavras: "escola", "Cabo Verde";
– resultados de cálculos intermédios.

Instruções

As **instruções** (ou comandos) definem as operações a realizar com os dados. São responsáveis pelo funcionamento do computador, dizendo como processar, modificar ou apresentar os dados.

Exemplos de instruções:
– somar dois números;
– comparar valores;
– guardar um ficheiro;
– mostrar um texto no ecrã.

Exemplo

Quando se levanta dinheiro num ATM, os dados são as informações introduzidas ou utilizadas durante a operação:
– cartão bancário;
– código pin;
– valor a levantar (por exemplo, 1000$);
– saldo disponível.

As instruções são as ações que o sistema executa com base nos dados:

1. ler o cartão;
2. pedir o pin;
3. verificar se o pin está correto;
4. verificar se existe saldo suficiente;
5. entregar o dinheiro;
6. atualizar o saldo;
7. devolver o cartão.

Sabias que...

Num computador, toda a informação é representada de forma digital através do sistema binário, que utiliza apenas dois valores: 0 e 1. Estes valores correspondem a estados elétricos (ligado/desligado) e permitem ao computador representar qualquer tipo de informação.
Também são utilizadas outras formas de representação auxiliares, como, por exemplo, o sistema hexadecimal, que utiliza 16 dígitos (0 1 2 3 4 5 6 7 8 9 A B C D E F) ou o sistema octal, que utiliza 8 dígitos (0 1 2 3 4 5 6 7).
De seguida, observa como as quantidades 0 a 16 são representadas nos diferentes sistemas:

decimal	binário	hexadecimal	octal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17
16	10000	10	20

<Modo ON #21>

Os caracteres alfanuméricos são armazenados internamente num computador na forma binária utilizando o padrão ASCII (American Standard Code for Information Interchange).

Observa a tabela de conversão ASCII seguinte, em que se faz a correspondência entre os caracteres e o sistema binário.

0	0011 0000	I	0100 1001	b	0110 0010	v	0111 0110
1	0011 0001	J	0100 1010	c	0110 0011	w	0111 0111
2	0011 0010	K	0100 1011	d	0110 0100	x	0111 1000
3	0011 0011	L	0100 1100	e	0110 0101	y	0111 1001
4	0011 0100	M	0100 1101	f	0110 0110	z	0111 1010
5	0011 0101	N	0100 1110	g	0110 0110		
6	0011 0110	O	0100 1111	h	0110 1000	:	0011 1010
7	0011 0110	P	0101 0000	i	0110 1001	;	0011 1011
8	0011 1000	Q	0101 0001	j	0110 1010	?	0011 1111
9	0011 1001	R	0101 0010	k	0110 1011	·	0010 1110
		S	0101 0011	l	0110 1100	'	0010 1111
		T	0101 0100	m	0110 1101	!	0010 0001
A	0100 0001	U	0101 0101	n	0110 1110	'	0010 1100
B	0100 0010	V	0101 0110	o	0110 1111	"	0010 0010
C	0100 0011	W	0101 0111	p	0111 0000	(	0010 1000
D	0100 0100	X	0101 1000	q	0111 0001	)	0010 1001
E	0100 0101	Y	0101 1001	r	0111 0010	space	0010 0000
F	0100 0110	Z	0101 1010	s	0111 0011		
G	0100 0111			t	0111 0100		
H	0100 1000	a	0110 0001	u	0111 0101		

1 Usa a tabela para descodificares a seguinte mensagem:

0100 0101 0111 0101 0010 0000
0110 1101 0110 1111 0111 0010 0110 1111 0010 0000
0110 0101 0110 1101 0010 0000
0100 0011 0110 0001 0110 0010 0110 1111 0010 0000
0101 0110 0110 0101 0111 0010 0110 0100 0110 0101

2 Usa a tabela para codificares a seguinte mensagem:

Tud dret

Tipos de dados

Os dados são armazenados de acordo com o tipo de informação que se deseja representar e com o tipo de operação que será realizada com eles. Os dados mais comuns encontrados na maioria das linguagens de programação, denominados por **dados simples** ou **dados primitivos**, são os dados numéricos, alfanuméricos e lógicos.

Dados numéricos

Os dados numéricos incluem dois tipos de dados: **inteiro** e **real**.

Um dado do tipo **inteiro** não possui parte decimal.

Um dado do tipo **real** ou, em inglês, **float**, pode ter uma parte decimal.

Inteiro

−4 763 0
2 -33

Real

−2.11 3.14159
$\sqrt{2}$

Dados alfanuméricos

Os dados alfanuméricos classificam-se como sendo do tipo ***caractere*** ou em inglês **string**. Estes dados são constituídos por uma sequência de caracteres que pode conter letras (a...z, A...Z), algarismos (0...9) e caracteres especiais (! @ # $ % & *). Usualmente, este tipo de dados são representados nos algoritmos com o uso de aspas (“) ou plicas (’):

Caractere

“escola” ‘Ilha de São Vicente’
“nano23@gmail.com” “x + y = 3”

Dados lógicos

Os dados de tipo **lógico** ou **booleano** têm apenas dois valores possíveis: Verdadeiro/Falso. Em algumas linguagens podem existir outras formas de representar estes dois valores lógicos, como, por exemplo, Sim/Não ou 1/0 ou True/False.

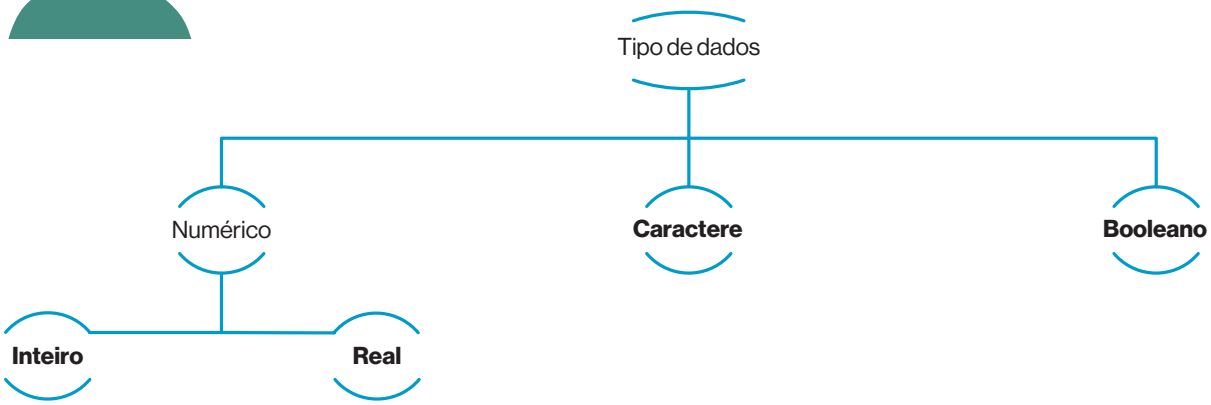
Booleano

Verdadeiro/Falso 1/0
Sim/ Não True/False

Exemplo

1. Os dados numéricos representam números inteiros e reais:
   – idade de uma pessoa: 14;
   – temperatura: 22,5;
   – quantidade de alunos numa turma: 28.
2. Os dados alfanuméricos são formados por letras, números e símbolos:
   – nome de um aluno: "Luana Soares";
   – morada: "Rua das Flores, 25";
   – endereço de e-mail: "aluno@email.cv".
3. Os dados lógicos só podem assumir dois valores:
   – O aluno está presente? → Verdadeiro.
   – A luz está ligada? → Falso.

Em pseudocódigo, classificamos cada tipo de dados usando as palavras: **inteiro**, **real**, ***caractere*** e **booleano**.



<Modo ON #22>

Indica o tipo de dado (inteiro, real, *caractere* ou booleano) em cada uma das situações seguintes.

Dado	Tipo de dado	Dado	Tipo de dado
23		–4	
3.45		0	
"abc"		–0.23	
Verdadeiro		"2+4"	
"Amarelo"		Falso	
5		"27.2"	

Variáveis e constantes

Para que seja possível armazenar e manipular dados num computador, é necessário representá-los internamente através de constantes ou de variáveis.

Um dado deve ser representado por uma **constante** quando o seu valor não se altera ao longo da execução de um algoritmo, ou seja, tem o mesmo valor desde o início até ao final.

Num algoritmo, deve ser usada uma constante quando se pretende armazenar uma informação ou um valor fixo que não será alterado durante a execução.

Um dado deve ser representado por uma **variável** quando o seu valor pode ser alterado ao longo da execução de um algoritmo.

O conteúdo de uma variável pode ser alterado, consultado ou apagado ao longo da execução de um algoritmo. Ao alterar o conteúdo de uma variável, a informação anterior é perdida, ou seja, a variável apenas armazena a última informação recebida.

Os **nomes** das constantes e variáveis obedecem a algumas regras:

- devem iniciar sempre com uma letra;
- não podem ter caracteres especiais (exceto o subtraço "_");
- não podem conter espaços em branco nem hífenes;
- não é permitido utilizar palavras reservadas.

Ao criar nomes, é conveniente usar palavras simbólicas que ajudem a entender e relembrar o conteúdo armazenado na variável ou constante.

Exemplo

1. Exemplo de nomes de variáveis válidos: num1, *x*, idade, A1, NumAlunos.
2. Exemplo de nomes de variáveis inválidos: 1num, (*x*), A#1, Num Alunos.

<Modo ON #23>

Identifica as opções válidas para o nome de uma variável. Justifica a tua resposta.

(A) soma	**(B)** nome*	**(C)** num_clientes
(D) 88num	**(E)** media	**(F)** média
(G) num_1	**(H)** nova:soma	

Declaração de variáveis e constantes

Ao longo de um algoritmo, os dados são manipulados através do nome das variáveis onde estão armazenados. Assim, o primeiro passo num algoritmo deve ser a **declaração** de variáveis e constantes.

Na declaração de variáveis, deve ser definido o nome e o tipo de dado a armazenar:

<nome variável> : <tipo de dado>

Na declaração de constantes, deve ser definido o nome e o valor da constante:

<nome constante> = <valor da constante>

Exemplo

1. Exemplo de declaração de variáveis:
   n1, n2 : inteiro
   x : real
   nome1 : caractere
2. Exemplo de declaração de constantes:
   pi = 3.14159
   cambio = 1.23
   maximo = 1000

Atribuição de valores a variáveis

Após a declaração de variáveis, é possível iniciar a manipulação de dados através de instruções de **atribuição** de valores. A atribuição de valores a uma variável é feita utilizando o símbolo ←.

<nome variável> ← <valor>

<nome variável> ← <expressão>

Exemplo

Exemplos de atribuição de valores a uma variável:

n1 ← 23

x ← (5 * n1 + 7) / 2

nome1 ← "William"

morada ← "Rua das Flores, 34"

☑ Not@ que:

1. Uma variável só pode armazenar valores e resultados de expressões que sejam do mesmo tipo para o qual foi definida. Por exemplo, considerando duas variáveis *n1* e *n2* do tipo inteiro, na atribuição:

   media ← (n1+n2)/2

   a variável media deverá ser do tipo real, uma vez que a média de dois números inteiros pode não ser um número inteiro.

2. Uma variável só armazena um único valor de cada vez, ou seja, sempre que um novo valor é atribuído à variável, o valor anterior armazenado é perdido.

Exemplo

No algoritmo seguinte, a variável contador é aumentada duas vezes com um incremento de +2. Inicialmente, tem o valor 1 e no final do algoritmo terá o valor 5.

Início

contador ← 1

contador ← contador +2

contador ← contador +2

Fim

<Modo ON #24>

1. Qual o valor da variável soma no final da execução do algoritmo?

```
Início
    n1 ← 5
    n2 ← 8
    soma ← n1 + n2
Fim
```

2. Qual o valor da variável *n*1 no final da execução do algoritmo?

```
Início
    n1 ← 6
    n2 ← 9
    n1 ← n1 + 5
    n1 ← n1 + n2
Fim
```

3. Qual o valor da variável *z* no final da execução do algoritmo?

```
Início
    x ← 5
    y ← 7
    z ← (x + y) / 2 + (x + y) / 3 + (x + y) / 4
Fim
```

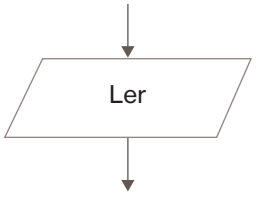
Entrada e saída de dados

A entrada e a saída de dados permitem a interação entre o utilizador e o computador. Estas operações acontecem constantemente quando usamos um computador, um telemóvel ou qualquer outro dispositivo digital. Geralmente as instruções de entrada permitem fazer a leitura de dados a partir de um teclado e as instruções de saída apresentam os dados num ecrã.

Instruções de entrada

Num algoritmo, para a entrada de dados é utilizada a instrução **Ler**.

Fluxograma



Pseudocódigo

```
Início
...
Ler <lista de variáveis>
...
Fim
```

Antes da utilização da instrução **Ler**, é necessário que esteja definida a variável que irá armazenar o valor que será lido.

Exemplo

1. Exemplo de um algoritmo em pseudocódigo que permite ler três números:

```
Algoritmo Ler_3_numeros
Var
  n1, n2, n3: inteiro
Início
  Ler n1, n2, n3
Fim
```

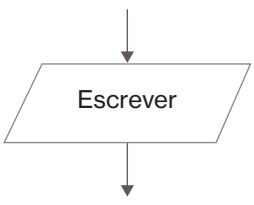
2. Exemplo de um algoritmo em pseudocódigo que permite ler o nome e a idade de um utilizador:

```
Algoritmo dados_pessoais
Var
  nome: caractere
  idade: inteiro
Início
  Ler nome
  Ler idade
Fim
```

Instruções de saída

Num algoritmo, para a saída de dados é utilizada a instrução **Escrever**.

Fluxograma



Pseudocódigo

```
Início
...
Escrever  <textos variáveis>
...
Fim
```

As instruções de escrita permitem realizar a saída de dados (geralmente para o ecrã), podendo estes ser valores de variáveis ou textos (que devem ser escritos dentro de aspas).

Exemplo

1. Exemplo de um algoritmo em pseudocódigo que, após ler o nome e a idade de um utilizador, mostra uma mensagem de cumprimento:

```
Algoritmo Cumprimento
Var
  nome: caractere
  idade: inteiro
Início
  Escrever "Qual é o teu nome?"
  Ler nome
  Escrever "Qual é a tua idade?"
  Ler idade
  Escrever "Olá", nome, "!"
  Escrever "Tens ", idade, " anos."
Fim
```

2. Exemplo de um algoritmo em pseudocódigo que apresenta o resultado de uma soma:

```
Algoritmo Somas
Início
  Escrever "4 + 3 =", 4 + 3
  Escrever "5 + 6 =", 5 + 6
Fim
```

No ecrã irão aparecer as informações 4 + 3 = 7 e 5 + 6 = 11.

<Modo ON #25>

1 **Analisa as seguintes instruções e indica o que irá aparecer no ecrã ao serem executadas.**

a)
```
a ← 10
b ← 20
Escrever b
b ← 5
Escrever a
Escrever b
c ← a + b
Escrever c
```

b)
```
a ← 30
b ← 20
c ← a + b
Escrever c
b ←10
Escrever b
```

c)
```
a ←10
b ← 20
c ← a
b ← c
a ← b
Escrever a
Escrever b
Escrever c
```

d)
```
a ←10
b ←  a + 1
a ← b + 1
b ← a + 1
Escrever a
a ← b + 1
Escrever a
Escrever b
```

2 **O algoritmo seguinte permite cambiar escudos em euros, considerando uma taxa de conversão. Escreve-o em fluxograma.**

```
Algoritmo Cambio
Var
  taxa: real
  valor: real
Início
  Escrever "Qual é a taxa de câmbio atual de escudos para euros?"
  Ler taxa
  Escrever "Qual é o valor em escudos?"
  Ler valor
  valor ← taxa*valor
  Escrever "O valor introduzido corresponde a ", valor, " euros."
Fim
```

3 **Escreve em pseudocódigo um algoritmo que leia um valor numérico inteiro (introduzido no teclado pelo utilizador) e que escreva (no ecrã) o seu antecessor e o seu sucessor.**

☑ Not@ que:

É possível escrever mensagens detalhadas que expliquem ao utilizador o tipo de dados que deve introduzir. Para isso, podemos utilizar uma instrução de saída de dados imediatamente antes da instrução de entrada. Por exemplo:

Escrever "Introduz a tua idade (em anos)"

Ler idade

<Modo ON #26>

1 **Completa o algoritmo com a introdução de mensagens, através de instruções de saída, que expliquem ao utilizador o tipo de dados que deve introduzir.**

```
Algoritmo Media_testes
Var
  teste1, teste2, teste3: inteiro
  media: real
Início
  Ler teste1
  Ler teste2
  Ler teste3
  media ← (teste1 + teste2 + teste3) / 3
  Escrever "A tua média nos três testes é", media
Fim
```

1 **Escreve um algoritmo que armazene o valor 10 numa variável *x* e o valor 20 numa variável *y*.**
A seguir deve trocar os seus conteúdos fazendo com que o valor que está em *x* passe para *y* e vice-versa (analisa quantas variáveis vai ser necessário utilizar no algoritmo).
No final, deve escrever os valores que ficaram armazenados nas variáveis.

3 **Constrói um algoritmo que leia a idade de uma pessoa expressa em anos, meses e dias e escreve a idade dessa pessoa expressa apenas em dias.**
O algoritmo deve pedir ao utilizador que introduza, separadamente, quantos anos, meses e dias tem a sua idade e, depois, efetuar os cálculos, considerando que o ano tem 365 dias e o mês tem 30 dias.

Testa os teus conhecimentos

1 **Completa a frase com as palavras "dados" e "instruções".**

De um modo simples, podemos especificar que _____ são o conteúdo e _____ são as ações. Sem dados, as _____ não têm sobre o que atuar e, sem instruções, os _____ não podem ser manipulados e transformados em informação útil.

2 **Classifica em verdadeira (V) ou falsa (F) cada uma das afirmações seguintes.**

(A) Uma constante armazena um valor que poderá mudar ao longo da execução de um algoritmo.

(B) Num algoritmo, um dado armazenado numa variável pode ser alterado, consultado ou apagado várias vezes.

3 **Identifica quais as opções válidas para o nome de variáveis.**

(A) nome*

(B) data de nascimento

(C) data_de_inicio

(D) 1numero

(E) novo_email

(F) num1

4 **Faz corresponder os termos da coluna da esquerda às descrições da coluna da direita.**

A. booleano	**1.** Números positivos, negativos ou nulos, sem casas decimais.
B. real	**2.** Sequência que pode conter letras, algarismos e caracteres especiais.
C. inteiro	**3.** Assume os valores Verdadeiro ou Falso.
D. caractere	**4.** Números que podem ter parte decimal.

5 **Identifica, em cada situação, qual o tipo de dado.**

a) Verdadeiro

b) 43

c) Cidade de Mindelo

d) 2.7271

e) 0

f) – 81

6 **Identifica a finalidade de cada um dos algoritmos seguintes.**

a)
```
Início
    Ler a
    a ← a * 3
    Escrever a
Fim
```

b)
```
Início
    Ler b
    b ← b / 2
    Escrever b
Fim
```

2.2. Expressões aritméticas

Operadores aritméticos

Manual Interativo
Exercício
Tipos de dados e operadores 1

As expressões aritméticas possuem como operadores as mesmas operações aritméticas da matemática e respeitam a prioridade (precedência) com que as operações devem ser executadas. As operações aritméticas mais usadas em algoritmos são: adição, subtração, multiplicação, divisão, potenciação, divisão inteira e resto da divisão inteira.

Num algoritmo, os **operadores aritméticos** executam operações aritméticas na manipulação de dados do tipo numérico (inteiro ou real). Na tabela abaixo, estão identificados os símbolos de cada operador em pseudocódigo e a respetiva operação.

Operadores aritméticos

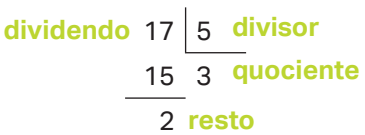
Operador	Operação	Exemplo (e resultado)
+	adição	2 + 3 (= 5)
–	subtração	5 – 2 (= 3)
*****	multiplicação	3 * 4 (= 12)
/	divisão	12 : 4 (= 3)
^	potenciação	3 ^ 2 (= 9)
div	divisão inteira	11 div 2 (= 5)
mod	resto da divisão inteira	11 mod 2 (= 1)

Exemplo

1. Para calcular a área de um triângulo, conhecendo os valores da base e da altura, é usada a expressão aritmética seguinte:
   area ← (base * altura) / 2
2. Para calcular o volume de um cubo, conhecendo o valor da aresta, é usada a expressão aritmética seguinte:
   volume ← aresta ^ 3
3. Para calcular o Índice de Massa Corporal (IMC), conhecendo o peso e a altura, é usada a expressão aritmética seguinte:
   imc ← peso / (altura * altura)
4. Para converter uma temperatura medida em graus Celsius em graus Fahrenheit, é usada a expressão aritmética seguinte:
   fahrenheit ← (celsius * 9 / 5) + 32

☑ Not@ que:

1. De uma operação entre valores inteiros, resulta sempre um valor inteiro, exceto para a operação divisão. Também não é possível dividir um valor por zero.
2. Numa divisão inteira, todos os números obtidos são inteiros, por isso, se o dividendo não for um múltiplo do divisor, vai existir um quociente e um resto.
   Por exemplo, o resultado da divisão inteira de 17 por 5 é dado por:
   $17 = 3 \times 5 + 2$



3. Numa linguagem algorítmica, é possível obter estes valores através dos operadores:

   17 div 5 = 3 (quociente inteiro)

   17 mod 5 = 2 (resto da divisão inteira)

Exemplo

1. Para determinar se um número *N* é par, verifica-se se o resto da divisão por 2 é zero usando-se a expressão aritmética *N* mod 2.
2. Um ano é bissexto se for múltiplo de 4, por isso, para determinar se um ano é bissexto, usa-se a expressão aritmética ano mod 4.

<Modo ON #27>

1. **Determina o resultado de cada operação.**
   - **a)** 26 / 4
   - **b)** 4 * 2
   - **c)** 4 ^ 2
   - **d)** 23 div 4
   - **e)** 23 mod 4
   - **f)** 12 mod 4
   - **g)** 10 mod 4
   - **h)** 10 / 4
2. **Classifica em verdadeiras (V) ou falsas (F) as igualdades entre as expressões seguintes.**
   - **(A)** 4 + 2 = 32 div 5
   - **(B)** 17 mod 5 = 8 – 6
   - **(C)** 3 ^ 2 = 10 – 1

Regras de prioridade em expressões aritméticas

Quando existem vários operadores numa mesma expressão, um computador segue uma ordem de prioridade, tal como em matemática. O uso de parênteses permite definir uma ordem específica de prioridade:

Operador	Operação	Exemplo (e resultado)
1.ª	Parênteses	( )
2.ª	Potenciação	^
3.ª	Multiplicação, divisão, resto e divisão inteira	*, /, mod, div
4.ª	Adição e subtração	+, –

Exemplo

1. Observa os vários exemplos de prioridade das expressões:

Expressão	Resultado	Explicação
1 + 3 ^ 2	10	A potência tem prioridade sobre a adição.
10 * 2 + 5 * 6	50	As multiplicações têm prioridade sobre a adição.
7 * (4 – 1)	21	Os parênteses dão prioridade à subtração.
6 / 3 * 2	4	A multiplicação e a divisão têm a mesma prioridade, pelo que são executadas da esquerda para a direita.
16 div 5 + 2	5	A divisão inteira tem prioridade sobre a adição.
16 mod 5 + 2	3	O resto da divisão inteira tem prioridade sobre a adição.
16 mod (5 + 2)	2	Os parênteses dão prioridade à adição.

2. Considera a expressão aritmética:

(5 + 3) ^ 2 * (5 – 2) + 8

A expressão será executada na seguinte ordem:

Passo 1	8 ^ 2 * 3 + 8
Passo 2	64 * 3 + 8
Passo 3	192 + 8
Passo 4	200

3. Considera a expressão:

$$(a + b) + (2 * b + (c - a))$$

A expressão será executada na seguinte ordem:

Passo 1. (a + b)

Passo 2. (c – a)

Passo 3. 2 * b

Passo 4. Adição do resultado do passo 2 com o resultado do passo 3.

Passo 5. Adição do resultado do passo 1 com o resultado do passo 4.

1 **Determina o resultado de cada expressão.**

a) 3 * 6 + 4 / 2

b) 5 + 4 ^ 2

c) 15 / (7 – 2)

d) 20 / 5 – 3

e) (7 ^ 2 – 1) / 3 + 2 * 5

f) 26 mod 5 + 23 mod 4

g) 5 ^ 2 – 10 + (2 ^ 3 – 5)

h) 15 – (2 + 1) ^ 2 + 2 ^ 3

2 **Os seguintes pares de instruções, produzem o mesmo resultado?**

a) (4 / 2) + (2 / 4) e 4 / 2 + 2 / 4

b) 4 / (2 + 2) / 4 e 4 / 2 + 2 / 4

c) (4 + 2) * 2 – 4 e 4 + 2 * 2 – 4

<Modo ON #29>

1 **Analisa as instruções seguintes e indica o que irá aparecer no ecrã ao serem executadas.**

a)
```
a ← 10
b ← 20
c ← (a + b) mod 4
Escrever c
```

b)
```
a ← 3
b ← 2
c ← a * 10 div (a + b)
Escrever c
```

c)
```
a ← 7
b ← 3
c ← 10
d ← (a + b) / c
Escrever d
```

d)
```
a ← 20
b ← 30
c ← a * 3 + b * 2
d ← c mod (a + b)
e ← c div (a + b)
Escrever d
Escrever e
```

2 **Escreve as seguintes expressões de modo que possam ser executadas por um computador.**

a) $\frac{4*3-7}{9*2-4*2}$

b) $\frac{2+3*5}{3^2-4} - \frac{3+4}{5*3}$

3 **Identifica as expressões aritméticas em que existem parênteses que não são necessários e reescreve-as com o mínimo de parênteses possível.**

a) 6 * (3 + 2)

b) (6 / 3) + (8 / 2)

c) 2 + (6 * (3 + 2))

d) ((3 + (8 / 2)) * 4) + (3 * 2)

e) 2 + (3 * 6) / (2 + 4)

f) (6 * (3 * 3) + 6) – 10

g) 2 * (8 / (3 + 1))

h) (((10 * 8) + 3) * 9)

i) 3 + (16 – 2) / (2 * (9 – 2))

j) ((–12) * (–4)) + (3 * (–4))

Boas práticas na escrita de expressões aritméticas

Num algoritmo, cada expressão aritmética deve ser escrita apenas numa única linha. Por esse motivo, muitas vezes é necessário utilizar parênteses para garantir que todas as operações são executadas na ordem adequada.

Por exemplo, a expressão:

$$\frac{9 + \sqrt{23+2}}{1+3} + 23$$

deve ser escrita na seguinte forma:

((9 + (23 + 2) ^ (1 / 2)) / (4 + 3)) + 23

permitindo que seja executada de modo adequado:

((9 + 25 ^ (0.5)) / 7) + 23
((9 + 5) / 7) + 23
(14 / 7) + 23
2 + 23
25

A escrita correta de expressões aritméticas nos algoritmos é essencial para garantir que os cálculos sejam claros, corretos e fáceis de interpretar tanto por pessoas como por computadores. Seguir boas práticas evita erros e ambiguidades.

Seguem-se algumas práticas que deves tentar seguir.

1. Respeitar a ordem das operações e usar parênteses para clareza

Deve ter-se sempre em conta a prioridade dos operadores. Em caso de dúvida, é preferível usar parênteses. Mesmo quando não são obrigatórios, os parênteses podem ajudar a que a expressão seja mais fácil de compreender.

Por exemplo, escrever 5 + (3 * 2) pode facilitar a compreensão.

2. Utilizar espaços para facilitar a leitura

Adicionar espaços entre números e operadores melhora a legibilidade da expressão.

Por exemplo, em vez de escrever 3+5*2−4, é mais fácil ler 3 + 5 * 2 − 4.

3. Evitar expressões longas e complexas

Expressões muito extensas aumentam a probabilidade de erro. É uma boa prática dividir cálculos complexos em partes mais simples.

Por exemplo, em vez de:

```
resultado ← (a + b + c + d + e) * f / g
```

Usar:

```
soma ← a + b + c + d + e
resultado ← soma * f / g
```

4. Ter atenção aos números negativos

Os números negativos devem ser escritos de forma clara, usando parênteses quando necessário. Por exemplo:

```
3 * (– 4)

7 + (– 2)

(– 8) / 4
```

5. Confirmar os parênteses utilizados

Um erro bastante comum ao escrever um algoritmo é deixar parênteses sem par nas expressões aritméticas. Um teste prático para evitar esse tipo de erro consiste em contar quantos parênteses esquerdos e direitos existem na expressão, e conferir se estão em igual número.

Uma boa prática, seguida por muitos programadores, é escrever logo ambos os parênteses e só depois completar a expressão no seu interior. Por exemplo, começar logo por escrever 7 * () e só depois escrever dentro do parenteses 7 * (3 + 5).

6. Testar com exemplos

Depois de escrever a expressão, testa com valores simples para confirmar se o resultado é o esperado.

Nem todas as expressões aritméticas possuem um significado matemático. Por exemplo, não é possível dividir um número por zero nem calcular raízes quadradas de números negativos. A avaliação desse tipo de expressão deve ser sempre evitada a partir da verificação dos valores que farão parte das mesmas.

Exemplo

O algoritmo abaixo usa uma expressão aritmética demasiado longa, difícil de ler e compreender, que poderá originar erros.

```
Início
  Ler a, b, c, d, e
  resultado ← (a + b * c – d) / e + c + a * b – c + d * (a + b + c / e)
  Escrever resultado
Fim
```

O mesmo algoritmo pode ser escrito usando partes mais simples e legíveis, de modo que seja mais fácil testar e corrigir erros.

```
Início
  Ler a, b, c, d, e
  parte1 ← a + b * c – d
  parte2 ← c + a * b – c
  parte3 ← a + b + c / e
  resultado ← parte1 / e + parte2 + d * parte3
  Escrever resultado
Fim
```

<Modo ON #30>

1 **Cria expressões aritméticas claras e corretas para as situações seguintes.**

a) Calcular o quadrado da soma de dois valores.

b) Calcular o salário mensal a partir do número de horas trabalhadas e do valor por hora.

c) Converter um valor em metros para centímetros.

2 **Escreve um algoritmo que leia as classificações de três testes de um aluno e calcule e escreva a média ponderada final deste aluno. Considera que o peso das notas de cada teste é 2, 3 e 5, respetivamente, e que a fórmula para o cálculo da média ponderada final é:**

$$\text{mediafinal} = \frac{n1 * 2 + n2 * 3 + n3 * 5}{10}$$

3 **Escreve um algoritmo que leia o número total de eleitores de um município e leia o número de votos válidos, nulos e brancos. O algoritmo deve calcular e escrever a percentagem que cada tipo de voto representa relativamente ao total de eleitores.**

Testa os teus conhecimentos

1 **Analisa os algoritmos seguintes e indica o que irá aparecer no ecrã ao serem executados.**

a)
Início
a ← 10
b ← 5
c ← a + b * 3
a ← 10
Escrever a, b, c
Fim

b)
Início
x ← 5
y ← 6
z ← y ^ 2 – x
x ← x + 7
y ← x + z
Escrever x, y, z
Fim

2 **Classifica em verdadeiras (V) ou falsas (F) as igualdades entre as expressões seguintes.**

(A) 8 mod 2 = 32 mod 8
(B) 17 div 5 = 22 div 5
(C) (8 – 2) * 4 = 8 – 2 * 4
(D) 5 ^ 2 = 100 / (2 * 2)

3 **Identifica, para cada situação, a expressão aritmética correta.**

a) Calcular a média de quatro valores:
(1) media ← (a + b + c + d) / 4
(2) media ← a + b + c + d / 4

b) Calcular o dobro da soma de *x* com *y*:
(1) resultado ← x + y * 2
(2) resultado ← (x + y) * 2

c) Calcular o valor final com desconto:
(1) valor ← preco - preco * desconto
(2) valor ← preco - (preco * desconto)

4 **Cria expressões aritméticas claras e corretas para as situações seguintes.**

a) Calcular a média de três números.

b) Calcular o valor final de um produto após aplicar um aumento de 20%.

c) Calcular a idade de uma pessoa sabendo o ano atual e o ano de nascimento.

5 **O preço de um carro novo para o consumidor é a soma do custo de fábrica com a percentagem do distribuidor e dos impostos (ambos aplicados ao custo de fábrica). Supondo que a percentagem do distribuidor é de 28% e a dos impostos é de 45%, constrói um algoritmo que leia o custo de fábrica de um carro e que calcule e escreva no ecrã o preço final para o consumidor.**

2.3. Expressões lógicas

As expressões lógicas são expressões com valor lógico de verdadeiro ou falso. São compostas por operadores relacionais e/ou operadores lógicos e podem ter variáveis, constantes e expressões aritméticas.

Operadores relacionais

Os **operadores relacionais** são usados para comparar dois valores ou duas expressões do mesmo tipo, resultando sempre um valor lógico: verdadeiro ou falso. Por exemplo, a expressão 2 < 3 é uma expressão lógica cujo valor é verdadeiro. Em contrapartida, a expressão 2 = 8 é uma expressão lógica cujo valor é falso.

Operadores relacionais

Operador	Significado	Exemplo (e resultado)
==	igual	2 == 3 (falso)
>	maior que	5 > 2 (verdadeiro)
<	menor que	3 < 4 (verdadeiro)
>=	maior ou igual que	4 >= 7 (falso)
<=	menor ou igual que	4 <= 4 (verdadeiro)
<>	diferente	3 <> 4 (verdadeiro)

Exemplo

Na tabela estão registados os resultados lógicos, para a = 7 e b = 12, de várias expressões compostas por operadores relacionais.

Expressão lógica	Resultado (para a = 7 e b = 12)
a == b	Falso
a <= b	Verdadeiro
a > b	Falso
a <> b	Verdadeiro
a < b	Verdadeiro

<Modo ON #31>

Constrói uma tabela idêntica à do exemplo anterior, na qual deves colocar as mesmas expressões lógicas e determinar o seu resultado lógico para a = 11 e b = 5.

☑ Not@ que:

Quando uma expressão lógica contém expressões aritméticas, os operadores aritméticos têm prioridade sobre os operadores relacionais.

Exemplo

Na tabela estão registados os resultados de várias expressões lógicas que contêm expressões aritméticas.

Expressão lógica	Resultado
7 + 3 == 16 – 6	Verdadeiro
5 + 6 <= 9	Falso
3 * 5 >= 15	Verdadeiro
3 ^ 2 < 4 + 3 * 2	Verdadeiro
5 ^ 2 <> 5 * 5	Falso

<Modo ON #32>

1 Para a = 20, b = 10 e c = 30, determina o resultado das expressões lógicas seguintes.

a) a + b > c

b) c – b <= a

c) a * b >= c

d) a == c – b

e) a – c <> b – a

f) (c – a) * b < a + b + c

g) a * b >= a * c

2 Para n1 = 10, n2 = 15 e n3 = 20, determina o resultado das expressões lógicas seguintes.

a) n1 + n2 > n3

b) n2 – n1 < 5

c) n1 + 25 <> n3 + 5

d) n1 * 3 >= n2 * 2

e) (n1 + n2) * 4 > 100

f) n1 + 30 <> n3 * 2

g) n1 + 30 == n3 * 2

Operadores lógicos

Os operadores lógicos são usados para combinar ou alterar expressões lógicas. Assim, é essencial começar por compreender o que é uma **expressão lógica**.

Uma **expressão lógica** ou **afirmação** é uma frase que pode ser verdadeira ou falsa.

Exemplo

1. As expressões lógicas têm sempre um valor lógico de verdadeiro ou falso:
   - "2 é maior que 1" é uma expressão lógica com valor lógico verdadeiro;
   - "5 é menor que 3" é uma expressão lógica com valor lógico falso;
   - "A Terra gira em torno do Sol" é uma expressão lógica com valor lógico verdadeiro.
2. Nem todas as frases são expressões lógicas, por exemplo:
   - "Está a chover?" não é uma expressão lógica, é uma pergunta;
   - "Vai dormir!" não é uma expressão lógica, é uma ordem;
   - "x + 5 = 10" não é uma expressão lógica porque o seu valor lógico vai depender do valor da variável x.

<Modo ON #33>

1 **Quais das seguintes frases são proposições lógicas?**

(A) "O número 15 é par." **(B)** "Que horas são?"

(C) "Mindelo é uma cidade." **(D)** "Lava a loiça agora!"

2 **Classifica em verdadeiras (V) ou falsas (F) as expressões lógicas seguintes.**

(A) Quatro é maior do que três. **(B)** 2 + 3 = 5.

(C) 10 é um número ímpar. **(D)** Praia é a capital de Cabo Verde.

(E) 7 é maior do que 12.

3 **Escreve a negação de cada frase.**

a) O número 8 é par.

b) Todos os alunos gostam de Matemática.

c) Hoje está a chover.

Os **operadores lógicos** são usados para combinar ou negar expressões lógicas, resultando sempre um valor lógico: verdadeiro ou falso. Por exemplo, a expressão "Vou tomar banho e vou lavar os dentes" é uma expressão lógica verdadeira se tanto "Vou tomar banho" como "Vou lavar os dentes" forem expressões verdadeiras.

Manual Interativo

Exercício
Tipos de dados e operadores 2

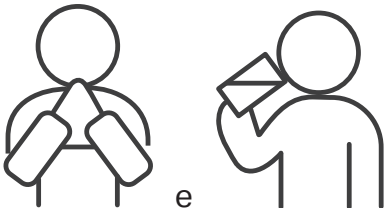
Operadores lógicos

Operador lógico	Símbolo lógico	Operação lógica	Valor lógico
e	∧	conjunção	É verdadeiro quando as duas expressões forem verdadeiras.
ou	∨	disjunção	É verdadeiro quando pelo menos uma das expressões for verdadeira.
não	~	negação	É falso quando a expressão for verdadeira e vice-versa.

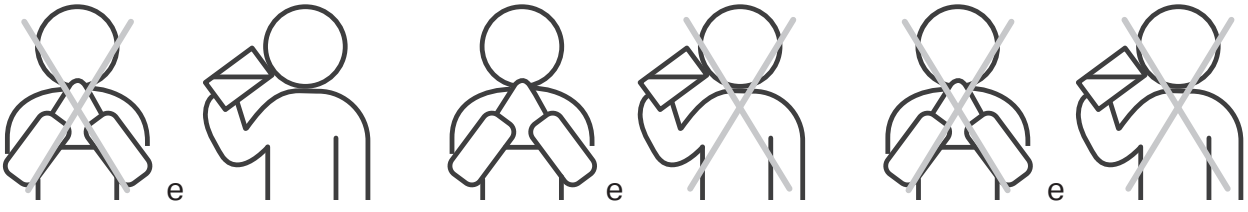
Operador "e"

O operador **e** é um operador binário de conjunção, que também pode ser representado por **∧** ou **and**. Quando duas expressões são unidas por este operador, a expressão resultante só é verdadeira se ambas as expressões também forem verdadeiras.

Por exemplo, a expressão "Estou a comer e beber" só é verdadeira se as duas expressões forem verdadeiras.



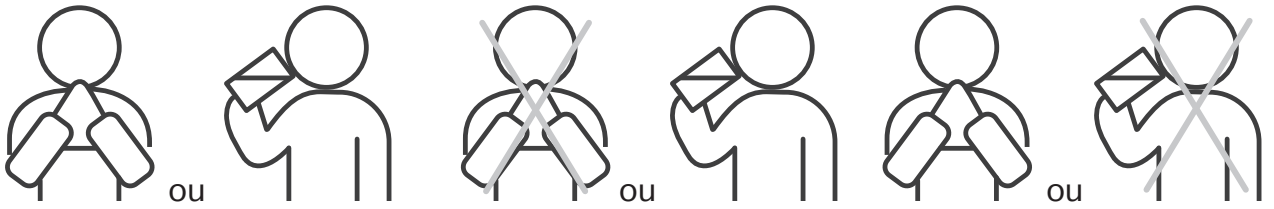
Se pelo menos uma das ações não ocorrer, a frase como um todo é falsa.



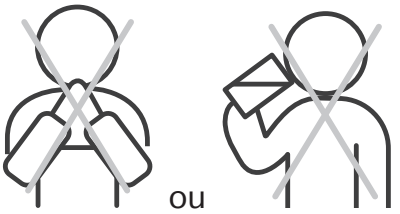
Operador "ou"

O operador **ou** é um operador binário de disjunção, que também pode ser representado por **v** ou **or**. Quando duas expressões são unidas por este operador, a expressão resultante é verdadeira se pelo menos uma das expressões for verdadeira.

Por exemplo, a expressão "Estou a comer ou beber" é verdadeira se pelo menos uma das duas expressões for verdadeira.



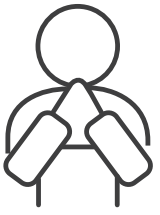
A frase só será falsa se ambas as expressões forem falsas.



Operador "não"

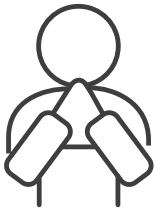
O operador **não** é um operador binário de negação, que também pode ser representado por **~** ou **not**. Este operador transforma uma expressão no seu contrário, ou seja, inverte o valor lógico da expressão à qual se aplica.

Por exemplo, a frase "Estou a comer" é transformada na frase "Não estou a comer". Se "Estou a comer" for verdadeiro, a frase "Não estou a comer" é falsa.


verdadeiro


falso

Se "Estou a comer" for falso, a frase "Não estou a comer" é verdadeira.


falso


verdadeiro

Exemplo

1. A negação da expressão "Vou estudar" é verdadeira apenas se a expressão for falsa:

Vou estudar	**Não vou estudar**
Verdadeiro	Falso
Falso	Verdadeiro

2. A frase "Vou estudar e vou à praia" é verdadeira apenas se ambas as expressões forem verdadeiras:

Vou estudar	**Vou à praia**	**Vou estudar e vou à praia**
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Falso

3. A frase "Vou estudar ou vou à praia" é verdadeira se pelo menos uma das expressões for verdadeira:

Vou estudar	**Vou à praia**	**Vou estudar ou vou à praia**
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso

<Modo ON #34>

1 **Classifica em verdadeiras (V) ou falsas (F) as afirmações seguintes.**

(A) Quatro é maior do que três e quatro não é um número par.

(B) Seis é um número par e seis é maior do que quatro.

(C) Nove é maior do que cinco e nove não é um número primo.

2 **Considera as expressões seguintes.**

P: "O número é par" Q: "O número é divisível por 3"

Escreve uma expressão lógica usando P e Q para representar cada uma das seguintes afirmações.

a) O número é par e não é divisível por 3.

b) O número não é par e não é divisível por 3.

c) O número é par ou é divisível por 3.

Operadores condicionais

Os **operadores condicionais** são usados para estabelecer uma relação entre expressões lógicas.

Operadores condicionais

Operador lógico	Símbolo lógico	Operação lógica	Valor lógico
se...então	$\longrightarrow$	condicional	É falso quando a expressão antecedente for verdadeira e a consequente for falsa.
...se somente se...	$\longleftrightarrow$	bicondicional	É verdadeiro quando ambas as expressões forem verdadeiras ou quando ambas forem falsas.

Operador "se... então"

O operador **se... então** é um operador condicional que estabelece uma condição entre duas expressões lógicas. A expressão "Se A então B" indica que B acontece se A acontecer. Esta implicação apenas é falsa se A for verdadeiro e B for falso.

Exemplo

A frase "Se vou estudar então vou à praia" apenas é falsa se "Vou estudar" for verdadeira e "Vou à praia" for falsa. Se a expressão "Vou estudar" for falsa, a frase não é contrariada independentemente de "Vou à praia" ser verdadeira ou falsa.

Vou estudar	Vou à praia	Se vou estudar então vou à praia
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Verdadeiro

<Modo ON #35>

Em qual das frases seguintes está presente um operador condicional?

(A) O Edson não está com febre e está desidratado.

(B) A Luana tem febre.

(C) Se o Dwayne tem febre, então deve ficar de cama 48 horas.

(D) Algum paciente tem febre.

Manual Interativo

Exercício
Operadores e expressões (linguagem Pascal)

<Modo ON #36>

Considera a frase:

"Se o Amílcar é carteiro e a Bernardina não é professora, então o Celestino não é biólogo."

Assinala a opção que apresenta, através de símbolos lógicos, a frase anterior.

(1) $(A \wedge \sim B) \longrightarrow C$

(2) $(A \vee \sim B) \longrightarrow \sim C$

(3) $(A \wedge \sim B) \longrightarrow \sim C$

(4) $(\sim A \wedge B) \longrightarrow \sim C$

Operador "... se somente se..."

O operador **... se somente se...** é um operador bicondicional que estabelece uma dupla condição entre duas expressões lógicas. A expressão "A se somente se B" indica que A só acontece se B também acontece. Esta implicação é verdadeira se ambas as expressões forem verdadeiras ou se ambas as expressões forem falsas.

Exemplo

A frase "Vou estudar se e somente se vou à praia" é verdadeira se as expressões "Vou estudar" e "Vou à praia" forem simultaneamente verdadeiras ou simultaneamente falsas.

Vou estudar	Vou à praia	Vou estudar se somente se vou à praia
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Verdadeiro

<Modo ON #37>

Considera os valores lógicos das expressões seguintes.

A = Verdadeiro B = Verdadeiro C = Falso

Determina o resultado lógico das expressões.

a) $A \longleftrightarrow B$

b) $A \longleftrightarrow C$

c) $B \longrightarrow C$

d) $C \longrightarrow B$

e) $A \longrightarrow B$

<Modo ON #38>

1 **Assinala a opção que apresenta, por esta ordem, uma disjunção, uma conjunção e um operador condicional.**

(A) $P \wedge Q, P \vee Q, P \longrightarrow Q$

(B) $P \vee Q, P \longrightarrow Q, P \wedge Q$

(C) $P \vee Q, P \wedge Q, P \longrightarrow Q$

2 **Considera as expressões lógicas seguintes.**

P: "Eu gosto de programação." Q: "Eu gosto de Matemática."

Escreve, usando linguagem natural, as seguintes expressões:

a) $P \wedge Q$

b) $P \vee Q$

c) $\sim P$

d) $P \longrightarrow Q$

e) $P \longleftrightarrow Q$

3 **Qual é o valor lógico de cada uma das expressões seguintes?**

a) Se o dobro de dois é quatro então o triplo de cinco é trinta.

b) O quadrado de cinco é vinte se e somente se o dobro de seis é dez.

c) O quadrado de dez é cem ou o quadrado de 9 é oitenta.

4 **Considera as expressões lógicas seguintes.**

A: "A Emily está a comer manga." **B:** "A Emily está a lanchar.""

Escreve, usando símbolos lógicos, as seguintes frases.

a) Se a Emily está a comer manga então está a lanchar.

b) A Emily está a comer manga e está a lanchar.

c) A Emily não está a lanchar.

d) A Emily está a comer manga se e somente se está a lanchar.

5 **Supõe que a frase "Se Alcides é pai de Nuemy, então Marcelo é pai de Kiara." é verdadeira. Nesse caso, sabendo que Kiara não é filha de Marcelo, qual é a expressão verdadeira?**

(A) Alcides é pai de Nuemy.

(B) Alcides não é pai de Nuemy.

Avaliação de expressões compostas

Regras de prioridade

Quando existem vários operadores, aritméticos e lógicos, numa mesma expressão, o computador segue uma ordem de prioridade específica que importa respeitar para obter resultados corretos. É, por isso, importante conhecer as regras de prioridade entre operadores.

Ordem	Operador	Descrição
1.ª	( )	Parênteses
2.ª	^	Operadores aritméticos
3.ª	*, /, mod, div	
4.ª	+, –	
5.ª	==, <, >, <=, >=, <>	Operadores relacionais
6.ª	~ (não)	Operadores lógicos
7.ª	∧ (e)	
8.ª	∨ (ou)	

Exemplo

Para a = 4, b = 2 e c = 5, a expressão:

(a + b) > c ∧ b * c <= a ^ 2

é verdadeira porque, ao respeitarmos a ordem de prioridade dos operadores, obtemos:

(4 + 2) > 5 ∧ 2 * 5 <= 4 ^ 2
6 > 5 ∧ 10 <= 16
verdadeiro ∧ verdadeiro
verdadeiro

<Modo ON #39>

1 **Para a = 3, b = 7 e c = 4, determina o resultado lógico das expressões seguintes.**

a) (a + c) > b ∧ a < 5

b) b >= (a + 2) ∨ c == (b – a)

c) (b + a) <= c ∧ (c + a) > b

2 **Para x = 2 e y = 3, determina o resultado lógico das expressões seguintes.**

a) x > 2 ∧ y > 2

b) ~(x > 2) ∧ y > 2

c) ~(x + 7 > 8) ∨ y > 2

Tabelas de verdade

Uma **tabela de verdade** é uma ferramenta utilizada no estudo da lógica para determinar se uma proposição é verdadeira ou falsa. É especialmente útil para analisar expressões compostas, formadas por combinações de expressões mais simples. Numa tabela de verdade são colocadas todas as combinações possíveis dos valores lógicos (verdadeiro ou falso) das expressões simples, sendo depois determinado, para cada caso, o valor lógico da expressão composta.

Exemplo

Para construir a tabela de verdade da expressão:

$$P \rightarrow Q$$

devemos começar por colocar na tabela todas as combinações possíveis dos valores lógicos das expressões P e Q (serão necessárias $2 \times 2 = 4$ linhas).

P	**Q**	**P → Q**
Verdadeiro	Verdadeiro	
Verdadeiro	Falso	
Falso	Verdadeiro	
Falso	Falso	

Determinamos, de seguida, o valor lógico da expressão composta para cada caso.

P	**Q**	**P → Q**
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Verdadeiro

Exemplo

Para construir a tabela de verdade da expressão:

$$P \rightarrow (Q \vee R)$$

devemos começar por colocar na tabela todas as combinações possíveis dos valores lógicos das expressões P, Q e R (serão necessárias $2 \times 2 \times 2 = 8$ linhas).

P	**Q**	**R**	**Q v R**	**P → (Q v R)**
Verdadeiro	Verdadeiro	Verdadeiro		
Verdadeiro	Verdadeiro	Falso		
Verdadeiro	Falso	Verdadeiro		
Verdadeiro	Falso	Falso		
Falso	Verdadeiro	Verdadeiro		
Falso	Verdadeiro	Falso		
Falso	Falso	Verdadeiro		
Falso	Falso	Falso		

Determinamos, de seguida, o valor lógico da expressão composta Q v R para cada caso.

P	**Q**	**R**	**Q v R**	**P → (Q v R)**
Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro	
Verdadeiro	Verdadeiro	Falso	Verdadeiro	
Verdadeiro	Falso	Verdadeiro	Verdadeiro	
Verdadeiro	Falso	Falso	Falso	
Falso	Verdadeiro	Verdadeiro	Verdadeiro	
Falso	Verdadeiro	Falso	Verdadeiro	
Falso	Falso	Verdadeiro	Verdadeiro	
Falso	Falso	Falso	Falso	

E, por último, determinamos o valor lógico da expressão composta P → (Q v R).

P	**Q**	**R**	**Q v R**	**P → (Q v R)**
Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Verdadeiro	Falso	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso	Falso	Falso
Falso	Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro
Falso	Verdadeiro	Falso	Verdadeiro	Verdadeiro
Falso	Falso	Verdadeiro	Verdadeiro	Verdadeiro
Falso	Falso	Falso	Falso	Verdadeiro

Através da tabela de verdade, conseguimos verificar que a expressão P → (Q v R) apenas é falsa quando P for verdadeira e Q e R forem simultaneamente falsas.

<Modo ON #40>

Completa as tabelas de verdade de cada uma das expressões compostas:

a) $P \wedge Q \rightarrow P \vee Q$

P	Q	P ∧ Q	P ∨ Q	P ∧ Q → P ∨ Q
Verdadeiro				
Verdadeiro				
Falso				
Falso				

b) $P \rightarrow (Q \wedge R)$

P	Q	R	Q ∧ R	P → (Q ∧ R)
Verdadeiro	Verdadeiro	Verdadeiro		
Verdadeiro	Verdadeiro	Falso		
Verdadeiro	Falso	Verdadeiro		
Verdadeiro	Falso	Falso		
Falso	Verdadeiro	Verdadeiro		
Falso	Verdadeiro	Falso		
Falso	Falso	Verdadeiro		
Falso	Falso	Falso		

c) $(P \vee Q) \rightarrow R$

P	Q	R	P ∨ Q	(P ∨ Q) → R
Verdadeiro				
Verdadeiro				
Verdadeiro				
Verdadeiro				
Falso				
Falso				
Falso				
Falso				

d) $(P \wedge Q) \rightarrow R$

P	Q	R	P ∧ Q	(P ∧ Q) → R

Exemplo

1. A tabela de verdade da expressão ~(P ∧ Q) é dada por:

P	Q	P ∧ Q	~(P ∧ Q)
Verdadeiro	Verdadeiro	Verdadeiro	Falso
Verdadeiro	Falso	Falso	Verdadeiro
Falso	Verdadeiro	Falso	Verdadeiro
Falso	Falso	Falso	Verdadeiro

Com base na tabela, é possível perceber que a negação da expressão P ∧ Q tem um resultado verdadeiro quando P é verdadeiro e Q é falso ou quando P é falso e Q é verdadeiro ou quando P é falso e Q é falso.

Portanto:

$$\sim(P \wedge Q) \longleftrightarrow \sim P \vee \sim Q$$

2. A tabela de verdade da expressão ~(P v Q) é dada por:

P	Q	P v Q	~(P v Q)
Verdadeiro	Verdadeiro	Verdadeiro	Falso
Verdadeiro	Falso	Verdadeiro	Falso
Falso	Verdadeiro	Verdadeiro	Falso
Falso	Falso	Falso	Verdadeiro

Com base na tabela é possível perceber que a negação da expressão P v Q tem um resultado verdadeiro quando P é falso e Q é falso.

Portanto:

$$\sim(P \vee Q) \longleftrightarrow \sim P \wedge \sim Q$$

Sabias que...

As **Leis de Morgan** são regras da lógica que descrevem como negar expressões com os operadores lógicos de conjunção (e) e disjunção (ou).

Negação da conjunção (e):

$\sim(P \wedge Q) \longleftrightarrow \sim P \vee \sim Q$

Negar que "A Daniela tem olhos verdes **e** a Bruna tem olhos azuis" é o mesmo que dizer que "A Daniela **não** tem olhos verdes **ou** a Bruna **não** tem olhos azuis".

Negação da disjunção (ou):

$\sim(P \vee Q) \longleftrightarrow \sim P \wedge \sim Q$

Negar que "O Lucas tem olhos verdes **ou** o Rafael tem olhos azuis" é o mesmo que dizer que "O Lucas **não** tem olhos verdes **e** o Rafael **não** tem olhos azuis".

<Modo ON #41>

1 **Indica qual das opções é a negação da expressão "O Gabriel é informático e a Alícia é *designer.*"**

(A) "O Gabriel é informático ou a Alícia é *designer.*"

(B) "O Gabriel não é informático e a Alícia é *designer.*"

(C) "O Gabriel não é informático ou a Alícia não é *designer.*"

(D) "O Gabriel não é informático ou a Alícia é *designer.*"

2 **Qual a negação da expressão "A Juceila gosta de viajar de barco ou o Denilson gosta de viajar de avião."?**

3 **Constrói a tabela de verdade de cada uma das expressões indicadas.**

a) $\sim(P \vee \sim Q)$

b) $\sim(P \rightarrow \sim Q)$

c) $Q \leftrightarrow \sim Q \wedge P$

4 **Uma empresa quer avaliar quais dos seus trabalhadores estão em condições de solicitar a reforma. Para um trabalhador se poder reformar, tem de satisfazer um dos seguintes requisitos:**

– Ter no mínimo 65 anos de idade.

– Ter trabalhado no mínimo 30 anos.

– Ter no mínimo 60 anos e ter trabalhado no mínimo 25 anos.

Foi construído um algoritmo que, para cada trabalhador, lê o ano do seu nascimento e o ano do seu ingresso na empresa, calcula a sua idade e o seu tempo de trabalho, e escreve a mensagem 'Pode requerer reforma' ou 'Não pode requerer reforma', de acordo com a análise da situação.

Preenche a tabela com os resultados do algoritmo para os diferentes valores introduzidos.

Ano de nascimento	Ano de ingresso na empresa	Idade	Anos de trabalho	Mensagem
1958	1995			
1963	1998			
1968	1998			
1962	2002			
1959	2005			

Testa os teus conhecimentos

1 **Completa a tabela com a identificação dos diferentes tipos de operadores.**

Operadores	Tipo de operadores
+ , – , * , /, ^ , mod, div	
==, <, >, <=, >=, <>	
∧ (e), ∨ (ou), ~ (não)	

2 **Determina o resultado de cada uma das expressões seguintes.**

a) 7 > 3 ∨ 5 < 2

b) 2.23 <> 2.33

c) 5 <> 2 ∧ 4 > 2

d) 8 >= 4 ⟶ 8 <= 2

3 **Considera as expressões lógicas seguintes.**

P: "4 elevado ao quadrado é igual a 8."

Q: "Metade da raiz quadrada de 64 é igual a 4."

Determina o resultado lógico das expressões compostas seguintes.

a) P ∨ Q

b) P → Q

c) Q → P

d) ~P ∧ Q

4 **Considera as expressões lógicas seguintes.**

A: "A Melissa vai comprar um carro." B: "O Enzo vai comprar uma mota.""

Escreve, usando símbolos lógicos, as frases seguintes.

a) A Melissa vai comprar um carro e o Enzo não vai comprar uma moto.

b) Se a Melissa vai comprar um carro então o Enzo vai comprar uma mota.

c) A Melissa não vai comprar um carro se e somente se o Enzo comprar uma mota.

d) A Melissa vai comprar um carro ou o Enzo vai comprar uma moto.

Testa os teus conhecimentos

5 **Considera as expressões lógicas seguintes.**

P: "Eu visitei a ilha Brava." Q: "Eu visitei a Boa Vista."

Escreve, usando linguagem natural, as expressões lógicas seguintes.

a) $P \wedge Q$

b) $P \longrightarrow Q$

c) $\sim P$

d) $P \vee Q$

6 **Para a = 2, b = 3 e c = 4, determina o valor lógico da expressão:**

(a + b) <> c * 3 ∧ c ^ a <= 10 * b

7 **Constrói a tabela de verdade de cada uma das expressões lógicas seguintes.**

a) $\sim P \wedge Q$

P	Q	~P	~P ∧ Q

b) $(P \vee Q) \wedge (\sim R)$

P	Q	R	P ∨ Q	~R	(P ∨ Q) ∧ (~R)

c) $\sim(P \wedge Q) \rightarrow R$

P	Q	R	$P \wedge Q$	$\sim(P \wedge Q)$	$\sim(P \wedge Q) \rightarrow R$

d) $(P \vee Q) \wedge (Q \vee R)$

P	Q	R	$P \vee Q$	$Q \vee R$	$(P \vee Q) \wedge (Q \vee R)$


3

Algoritmia e programação estruturada

3 Algoritmia e programação estruturada

Manual Interativo

Vídeos
Introdução ao *MakeCode*

Programar o micro:bit

EVStory
Vamos aprender juntos sobre micro:bit

Vídeos
Micro:bit – O teu primeiro projeto

Projetos micro:bit – Cronómetro com som e bússola

Micro:bit – Luz de presença

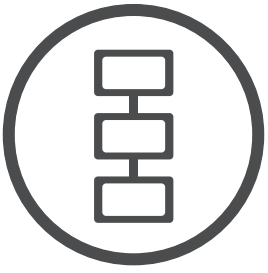
No final deste capítulo, deverás ser capaz de:

- Compreender e aplicar estruturas de controlo de fluxo (sequência, decisão e repetição) na resolução de problemas.
- Desenvolver programas simples e funcionais que integrem diferentes tipos de estruturas de forma coerente.
- Adotar boas práticas de programação, nomeadamente a clareza do código, a indentação e a estruturação adequada.
- Demonstrar autonomia, responsabilidade e espírito colaborativo no desenvolvimento de projetos.

3.1. Estruturas sequenciais

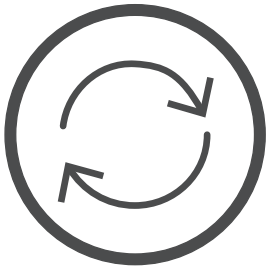
A **tríade fundamental** de um algoritmo é composta por três tipos essenciais de instruções que permitem construir qualquer programa:

Estruturas sequenciais



Estruturas condicionais

Estruturas de repetição

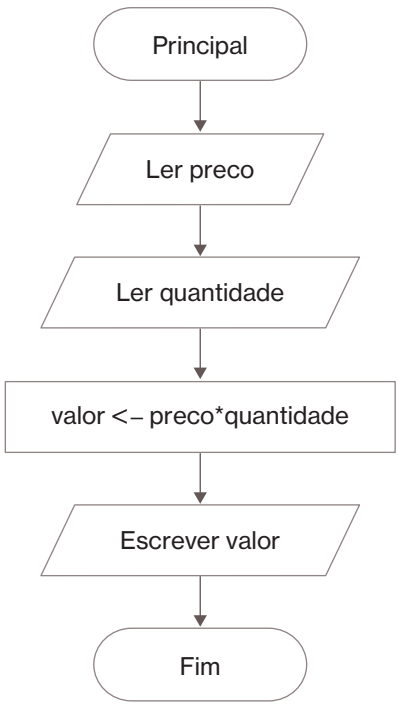


Estruturas sequenciais

Uma **estrutura sequencial** é a mais simples estrutura de controlo num algoritmo, sendo formada por uma sequência fixa de instruções que são executadas pela ordem em que aparecem. Numa estrutura sequencial é seguido um fluxo único, sem desvios, repetições ou decisões.

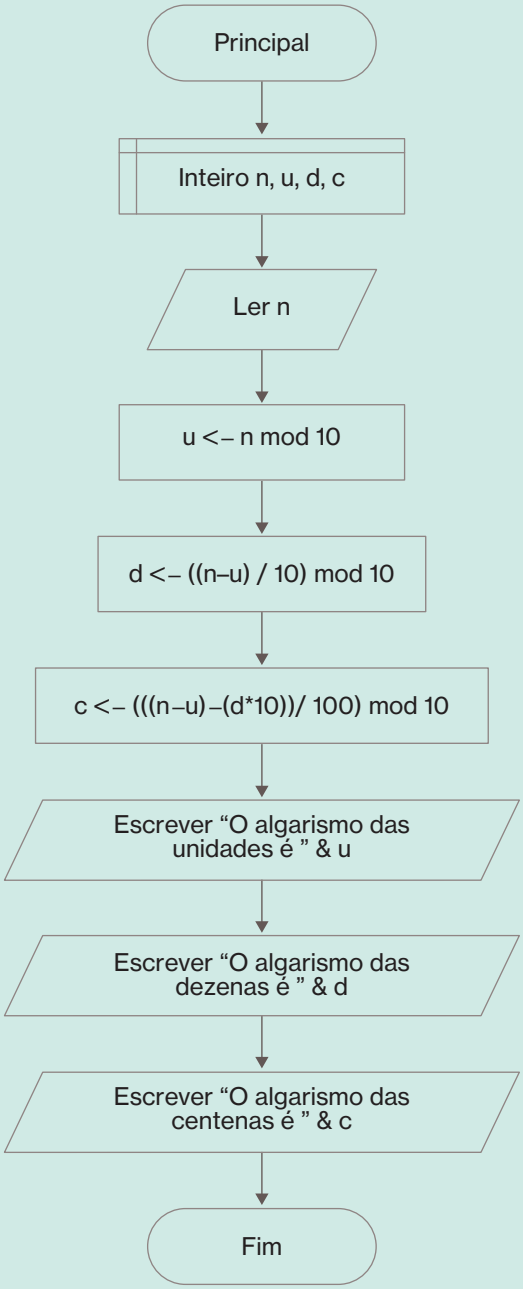
Exemplo

1. No seguinte algoritmo é seguida uma estrutura sequencial que sucessivamente lê o nome, o preço e a quantidade de um produto, calcula o valor a pagar e apresenta esse valor no ecrã (na ferramenta Flowgorithm, no início do algoritmo, é usada a palavra **Principal**).



<Modo ON #42>

1 **Considera o algoritmo seguinte.**



a) Testa o algoritmo com diferentes números (por exemplo, 423, 340, 78, 6).

b) Escreve o algoritmo em pseudocódigo.

2 **Escreve um algoritmo que leia uma data no formato DDMMAA e escreva:**

Dia: 17

Mês: 05

Ano: 26 (quando o número introduzido for 170526)

Testa os teus conhecimentos

1 **Classifica em verdadeiras (V) ou falsas (F) as afirmações seguintes.**

(A) Numa estrutura sequencial, as instruções são executadas pela ordem em que aparecem no algoritmo.

(B) Um algoritmo só pode ter estruturas sequenciais.

(C) Nas estruturas sequenciais, todas as instruções são sempre executadas.

(D) Nas estruturas sequenciais, podem existir desvios no fluxo do algoritmo.

(E) Ler dois números e calcular a sua média é um exemplo de uma estrutura sequencial.

2 **Escreve um algoritmo que auxilie no cálculo dos descontos aplicados numa loja de artesanato.**

O algoritmo deve:

– Ler o nome de um produto.

– Ler o preço.

– Ler o desconto.

– Calcular o valor final a pagar após aplicar o desconto.

– Apresentar o nome do produto e o preço a pagar.

3 **Escreve um algoritmo que simule uma minicalculadora.**

O algoritmo deve:

– Ler dois números.

– Calcular a soma dos dois números.

– Calcular o resultado da subtração (1.º número lido – 2.º número lido).

– Calcular o produto dos dois números.

– Apresentar no ecrã os resultados na forma:

A soma dos números é…

A diferença entre os números é…

O produto dos números é…

Testa os teus conhecimentos

4 **Numa loja de roupa, os funcionários recebem um salário fixo por mês, acrescido de uma comissão de 5% sobre o valor total das vendas da loja.**

Escreve um algoritmo que:

– Leia o valor do salário fixo de um funcionário.

– Leia o valor total das vendas da loja.

– Calcule o valor final que o funcionário deverá receber.

– Apresente o valor final a receber.

5 **Constrói um algoritmo que, no momento de consultar o saldo bancário, determine o saldo atual tendo por base o último saldo calculado, o débito e o crédito.**

O algoritmo deve:

– Ler o número da conta do cliente.

– Ler o saldo anterior.

– Ler o valor do débito.

– Ler o valor do crédito.

– Calcular o valor do saldo atual (saldo atual = saldo anterior – débito + crédito).

– Apresentar o valor do saldo atual.

3.2. Estruturas condicionais

Como já sabes, um algoritmo segue e executa as instruções exatamente pela ordem com que foram escritas. Para que um algoritmo possa seguir diferentes sequências de instruções são usadas as **estruturas condicionais**.

As **estruturas condicionais** permitem que um algoritmo tome decisões. Com base numa condição, o algoritmo pode escolher o que fazer e seguir diferentes caminhos.

As **estruturas condicionais** num algoritmo são instruções de decisão, ou seleção, que permitem alternar entre um ou outro conjunto de instruções após a avaliação lógica de uma condição.

Numa estrutura condicional, o algoritmo:

– Lê uma condição.

– Avalia o resultado (verdadeiro ou falso).

– Decide qual o bloco de instruções a executar.

– Executa apenas as instruções selecionadas.

As **estruturas condicionais** subdividem-se em estruturas de decisão simples, estruturas de decisão composta, estruturas de decisão encadeadas e estruturas de decisão múltipla.

Tipos de estruturas condicionais

Estrutura de decisão simples	**SE ... ENTÃO ...**
Estrutura de decisão composta	**SE ... ENTÃO ...** **SENÃO ...**
Estrutura de decisão encadeada	**SE ... ENTÃO ...** **SE ... ENTÃO ...** **SE ... ENTÃO ....**
Estrutura de decisão múltipla	**SELECIONAR** **CASO ...** **CASO ...** **CASO ...** **CASO ...**

Manual Interativo

Vídeos

Instalação do *Arduino*

Arduino – Usando botões e LED

Arduino – O meu primeiro circuito

Usando componentes associados ao *Arduino*

Estrutura de decisão simples: SE ... ENTÃO

Uma **estrutura de decisão simples** permite bifurcar a execução de um algoritmo em dois fluxos distintos. Se a condição for verdadeira é executado o conjunto de instruções da estrutura, caso contrário, nada acontece e o algoritmo continua com a execução das instruções que se seguem a esta estrutura.

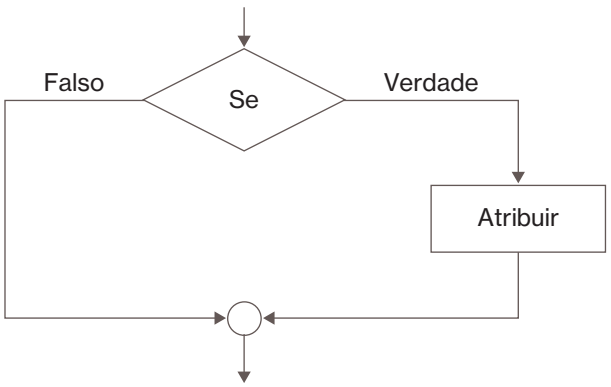
Exemplo

Exemplo de uma estrutura de decisão sobre quando se come um gelado.

SE temperatura > 30 ºC **ENTÃO** comer um gelado

Em fluxograma, numa estrutura de decisão simples é usado o símbolo losango e são definidas instruções apenas para quando a condição for verdadeira:

Fluxograma



Em pseudocódigo, é utilizada a instrução **Se ... Então ... Fimse**.

Pseudocódigo

```
Início
...
Se <condição> Então
  <instruções>
Fimse
...
Fim
```

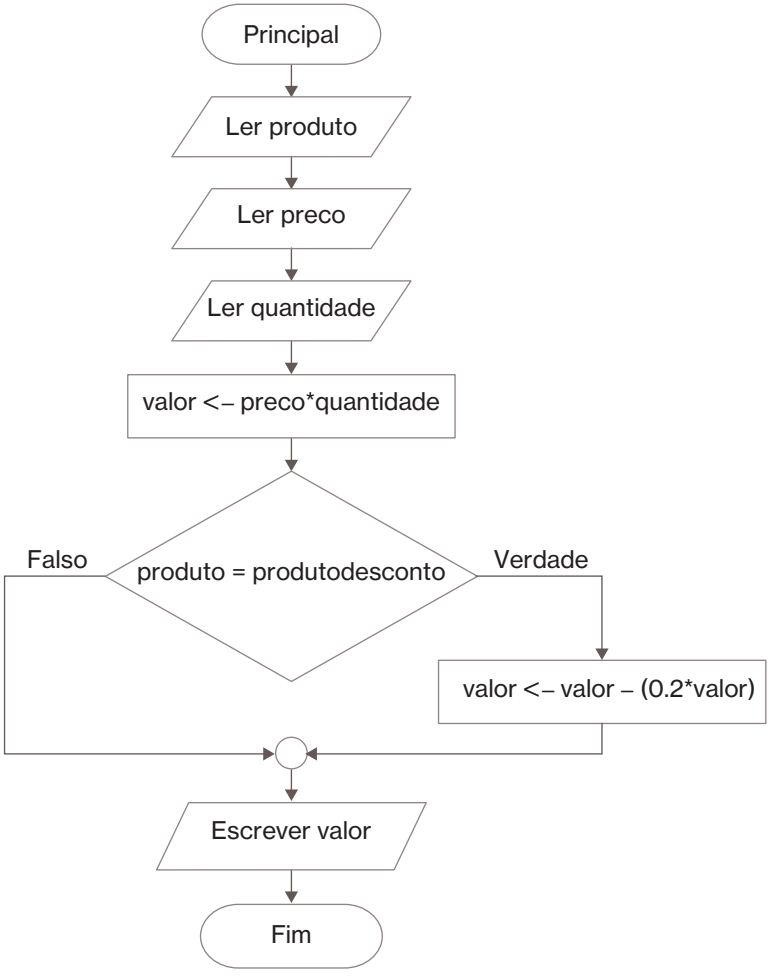
Nas **estruturas de decisão simples**, as instruções apenas são executadas quando a condição de seleção for verdadeira.

Exemplo

1. Exemplo de um algoritmo, em pseudocódigo, que informa quando um aluno é aprovado num exame.

```
Algoritmo Nota_exame
Var
  nota: real
Início
  Ler nota
  Se nota >= 10 Então
    Escrever "O aluno está aprovado."
  Fimse
Fim
```

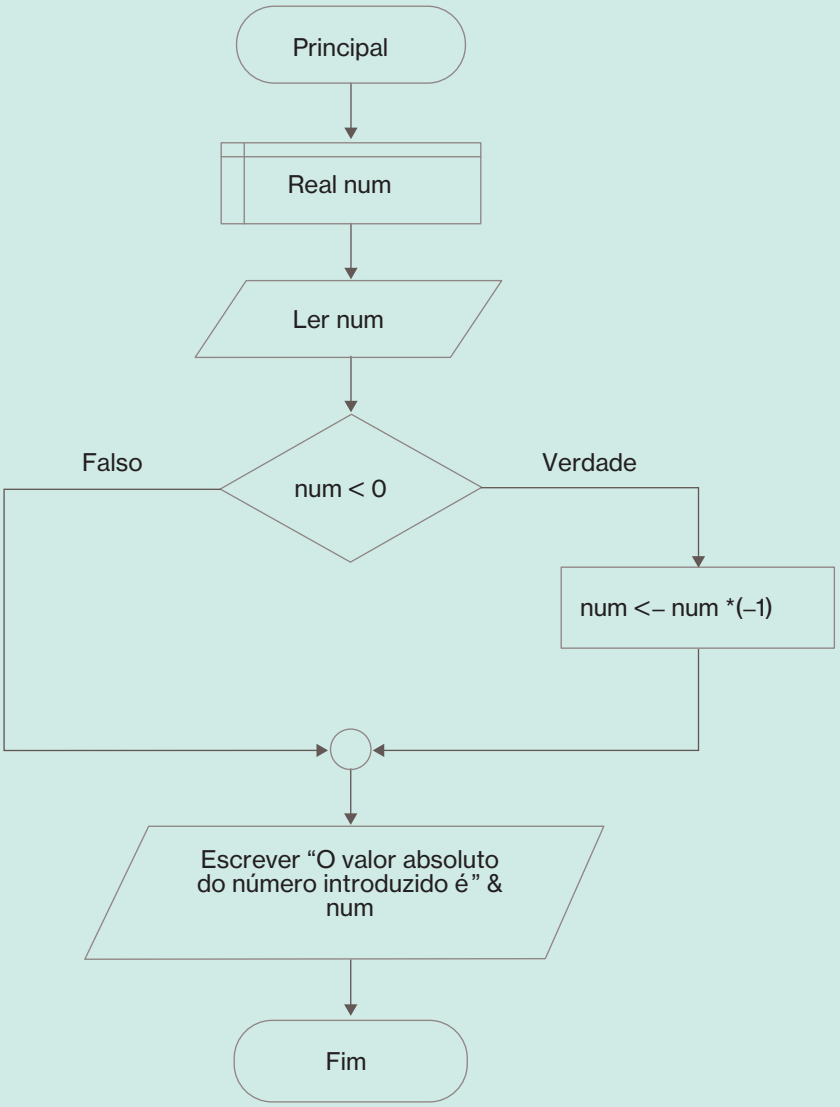
2. No algoritmo seguinte, após uma estrutura sequencial que lê o nome, o preço e a quantidade de um produto e calcula o valor a pagar, existe uma estrutura de decisão que, caso o produto tenha desconto (é consultada uma lista interna dos produtos com desconto), altera para o valor com desconto.



<Modo ON #43>

1 **Escreve, em pseudocódigo, um algoritmo que solicite a um utilizador que introduza as duas classificações obtidas por um aluno em dois testes e calcule a média das mesmas.**
O algoritmo deve apresentar no ecrã o valor da média e, caso esta seja igual a ou maior que 10, deve também aparecer no ecrã "Está aprovado!".

2 **Considera o algoritmo seguinte:**



a) Testa o algoritmo com a introdução de diferentes números positivos e negativos (por exemplo, 4, 12, –3, –7).

b) Explica qual a tarefa desempenhada pelo algoritmo.

c) Escreve o algoritmo em pseudocódigo.

Estrutura de decisão composta: SE ... ENTÃO ... SENÃO

Uma **estrutura de decisão composta** permite bifurcar a execução de um algoritmo em dois fluxos distintos, existindo dois conjuntos distintos de instruções a executar.

Se a condição for verdadeira, é executado o primeiro conjunto de instruções da estrutura, caso contrário, se for falsa, é executado o segundo conjunto.

Após executar um dos dois conjuntos de instruções, o algoritmo continua com a execução das instruções fora desta estrutura.

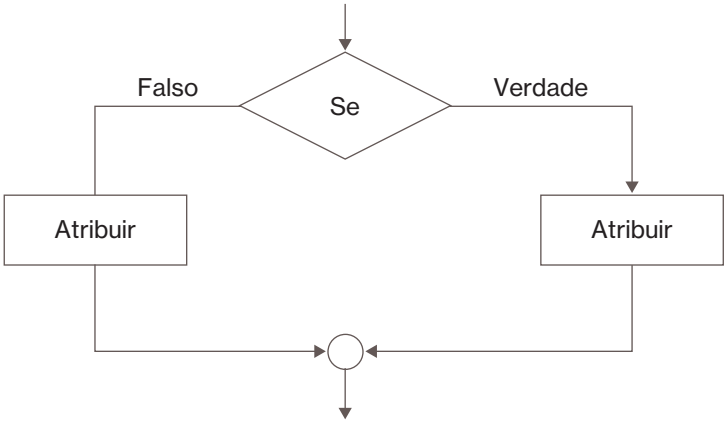
Exemplo

Exemplo de uma estrutura de decisão composta sobre beber água.

SE temperatura > 20 ºC **ENTÃO** "beber água fresca"
SENÃO "beber água natural"

Em fluxograma, numa estrutura de decisão composta, é usado o símbolo losango e são definidos dois conjuntos distintos de instruções.

Fluxograma



Em pseudocódigo, é utilizada a instrução **Se...Então...Senão...Fimse**.

Pseudocódigo

```
Início
...
Se <condição> Então
  <instruções>
Senão
  <instruções>
Fimse
...
Fim
```

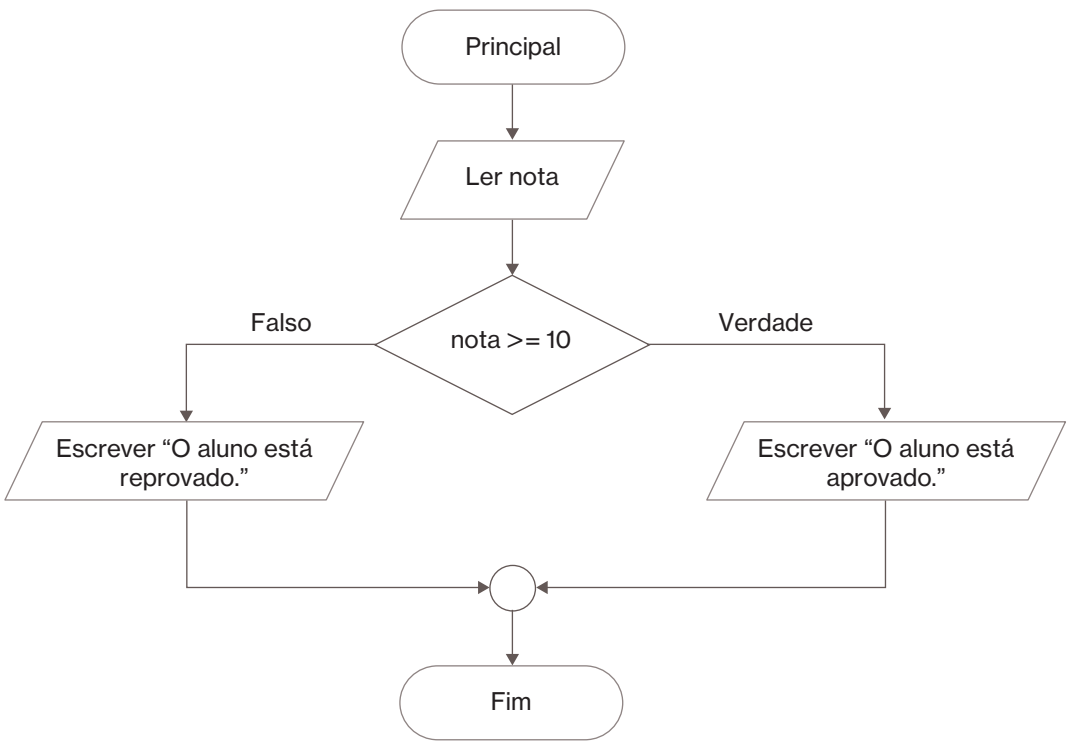
Nas **estruturas de decisão composta** existem dois conjuntos de instruções em que apenas um deles é executado. A seleção de qual o conjunto a executar depende do valor lógico da condição.

Exemplo

1. Exemplo de uma estrutura de decisão num algoritmo, escrito em pseudocódigo, que informa se um aluno está aprovado ou reprovado.

```
Algoritmo Resultado_exame
Var
    nota: real
Início
    Ler nota
    Se nota >= 10 Então
      Escrever "O aluno está aprovado."
    Senão
      Escrever "O aluno está reprovado."
    Fimse
    ...
Fim
```

2. Exemplo de um algoritmo em fluxograma com a mesma estrutura de decisão.



<Modo ON #44>

1. Constrói um algoritmo que leia um número e que escreva no ecrã a mensagem "É maior do que 5!", se o número lido for maior que 5. Caso contrário, deve escrever "Não é maior que 5!".
2. Constrói um algoritmo que leia o ano atual e o ano de nascimento de uma determinada pessoa e calcule a sua idade. O algoritmo deverá depois escrever uma mensagem que diga se a pessoa pode ou não votar este ano (não é necessário considerar o mês em que a pessoa nasceu).
3. Constrói um algoritmo que leia um número inteiro e determine se o número é par ou ímpar. No ecrã deverá depois surgir uma das seguintes mensagens: "O número inserido é par" ou "O número inserido é ímpar".
4. Constrói um algoritmo que leia dois números (considera que nunca serão lidos dois números iguais) e escreva no ecrã qual é o maior dos dois.

Exemplo

Neste algoritmo são lidas as medidas dos lados de uma figura retangular e é calculada a sua área. Se as medidas dos dois lados forem iguais, trata-se de um quadrado. Caso contrário, trata-se de um retângulo. Para cada um dos casos, é apresentada a mensagem adequada.

```
Algoritmo Area
Var
  lado1, lado2, area: real
Início
  Escrever "Introduz as medidas dos lados"
  Ler lado1, lado2
  area ← lado1 * lado2
  Se lado1 == lado2 Então
     Escrever "A área do quadrado é ", area
  Senão
     Escrever "A área do retângulo é ", area
  Fimse
Fim
```

<Modo ON #45>

1 **A D. Eneida vende produtos no mercado e os seus preços dependem das quantidades que são compradas. As cebolas custam 10$ cada se forem compradas menos de uma dúzia e custam 8$ cada se forem compradas pelo menos 12 cebolas.**

Constrói um algoritmo que informe qual o valor a pagar numa compra de *N* cebolas, tendo em conta que o número *N* de cebolas pode ser menor que 12 ou maior ou igual (usa o símbolo >= 12).

2 **Constrói um algoritmo que converta graus Celsius em graus Fahrenheit, ou vice-versa, conforme a opção do utilizador. O utilizador deve introduzir uma temperatura e identificar a conversão pretendida.**

Se introduzir a temperatura em graus Celsius, para converter para graus Fahrenheit deve ser usada a fórmula seguinte:

$$F = (C \times 1.8) + 32$$

Se introduzir a temperatura em graus Fahrenheit, deve ser usada a fórmula seguinte:

$$C = (F - 32) / 1.8$$

Estruturas de decisão encadeadas

As **estruturas de decisão encadeadas** são usadas quando é necessário avaliar várias condições diferentes, uma após a outra, para decidir qual o conjunto de instruções a executar. Dependendo do valor lógico de cada uma das condições, o algoritmo decide qual o caminho a seguir e quais as instruções a executar.

Exemplo

1. No dia a dia, podem ser usadas estruturas de decisão encadeadas para destacar um desempenho excelente entre os alunos aprovados numa prova.

```
SE  nota >= 10  ENTÃO
    SE  nota >= 18  ENTÃO  escrever "Excelente"
    SENÃO  escrever "Aprovado"
SENÃO  escrever "Reprovado"
```

2. Este exemplo de estruturas de decisão encadeadas pode ser representado através de um algoritmo escrito em pseudocódigo:

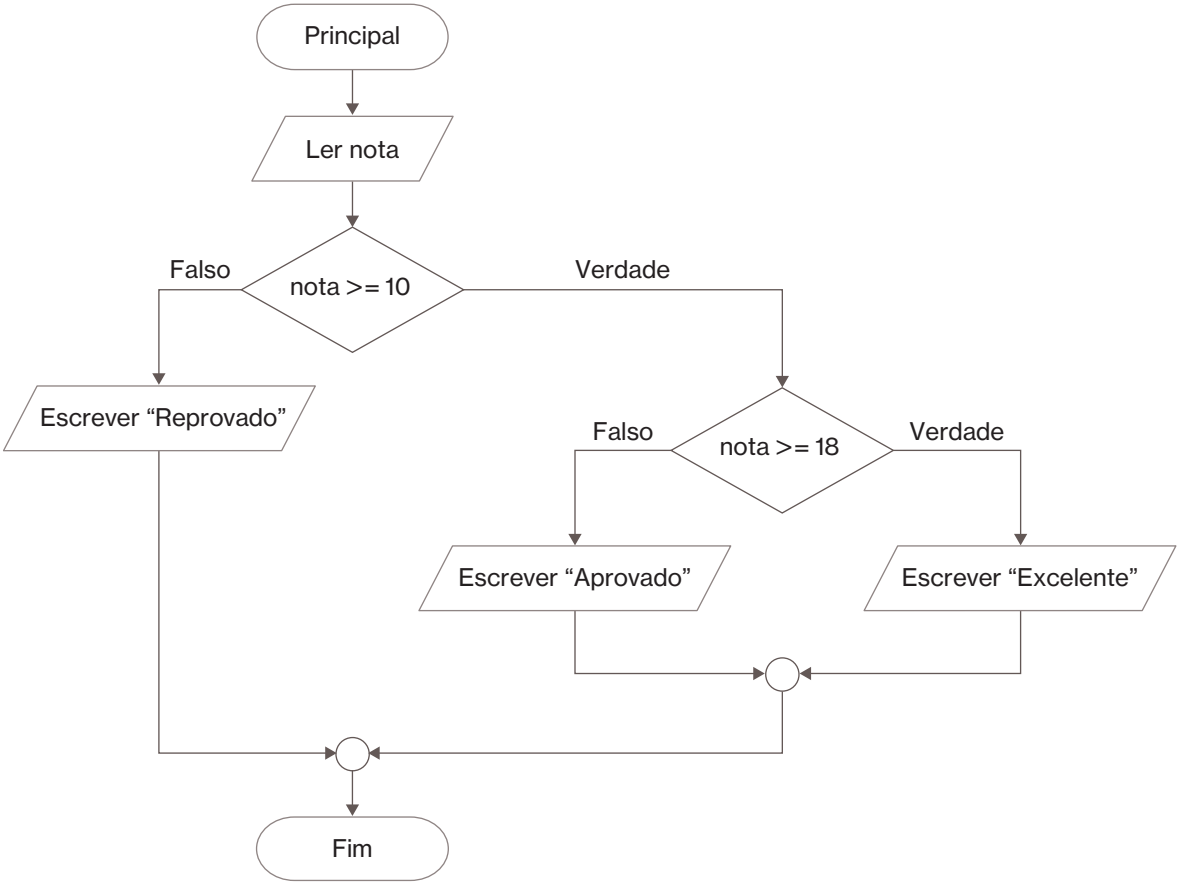
```
Algoritmo Resultado_prova
Var
        nota: real
Início
        Ler nota
        Se nota >= 10 Então
            Se nota >= 18 Então
              Escrever "Excelente."
            Senão
              Escrever "Aprovado."
            Fimse
        Senão
            Escrever "Reprovado."
        Fimse
        ...
Fim
```

☑ Not@ que:

1. Quando escreveres em pseudocódigo estruturas de decisões encadeadas é importante que uses indentação nas instruções. Esse cuidado facilita a leitura e compreensão do algoritmo e ajuda a evitar erros.
2. Após introduzires várias estruturas condicionais no teu algoritmo, deves sempre verificar se cada uma delas é iniciada e terminada. Ou seja, para cada "**Se**" deve aparecer sempre um "**Fimse**".

Exemplo

O exemplo da página anterior relativo ao desempenho numa prova pode também ser escrito em fluxograma usando uma estrutura de decisão encadeada.



<Modo ON #46>

1. **Constrói um algoritmo que leia um número e escreva no ecrã se ele é positivo, negativo ou nulo.**
2. **Constrói um algoritmo que leia dois números e escreva no ecrã se os números são iguais ou, no caso de não serem, escreva qual é o maior.**
3. **Constrói um algoritmo que leia três números e escreva no ecrã qual é o maior (considera que são sempre introduzidos três números diferentes).**
4. **Constrói um algoritmo que leia três números e escreva no ecrã a soma dos dois maiores (considera que são sempre introduzidos três números diferentes).**
5. **Constrói um algoritmo que leia três números e que os escreva no ecrã por ordem crescente (considera que são sempre introduzidos três números diferentes).**

<Modo ON #47>

1. **Constrói um algoritmo que leia o nome de duas equipas de futebol e o número de golos marcados por cada equipa.**
   O algoritmo deve depois escrever o nome da equipa vencedora e caso não haja vencedor deve escrever "Empate".

2. **A D. Eneida, no mercado, vende as batatas a três preços distintos:**
   – cada batata custa 8$, se forem compradas menos de 10 batatas;
   – cada batata custa 7$, se forem compradas entre 10 e 20 batatas;
   – cada batata custa 5$, se forem compradas mais de 20 batatas.
   Constrói um algoritmo que informe qual o valor a pagar numa compra de *N* batatas.

<Modo ON #47>

3 **Considera o algoritmo seguinte.**

```
Algoritmo Triangulo
Var
    a, b, c: real
    mensagem: caractere
Inicio
  Ler a, b, c
  Se (a == b) e (b == c) Então
    mensagem ← "Triângulo equilátero"
  Senão
    Se (a == b) ou (b == c) ou (a == c) Então
            mensagem ← "Triângulo isósceles"
    Senão
            mensagem ← "Triângulo escaleno"
    Fimse
  Fimse
  Escrever mensagem
Fim
```

Testa o algoritmo para diferentes valores de *a*, *b* e *c*. Preenche a tabela com as mensagens correspondentes.

a	b	c	mensagem
2	2	3	
2	3	4	
5	5	5	
4	3	3	
5	4	5	

Estrutura de decisão múltipla: ESCOLHA … CASO

Uma **estrutura de decisão múltipla** é usada quando existem várias opções possíveis que dependem da mesma variável ou expressão. Para cada valor dessa variável, cada **caso**, existe um conjunto de instruções a seguir.

É uma alternativa mais simples do que ter de usar sucessivamente várias estruturas de decisão encadeadas **'Se … Então … Senão'**.

Exemplo

Considera uma máquina de café que tem apenas quatro funções: ligar, desligar, café e água. Sempre que se carrega num dos botões "L", "D", "C" ou "A" aparece escrito no monitor da máquina:

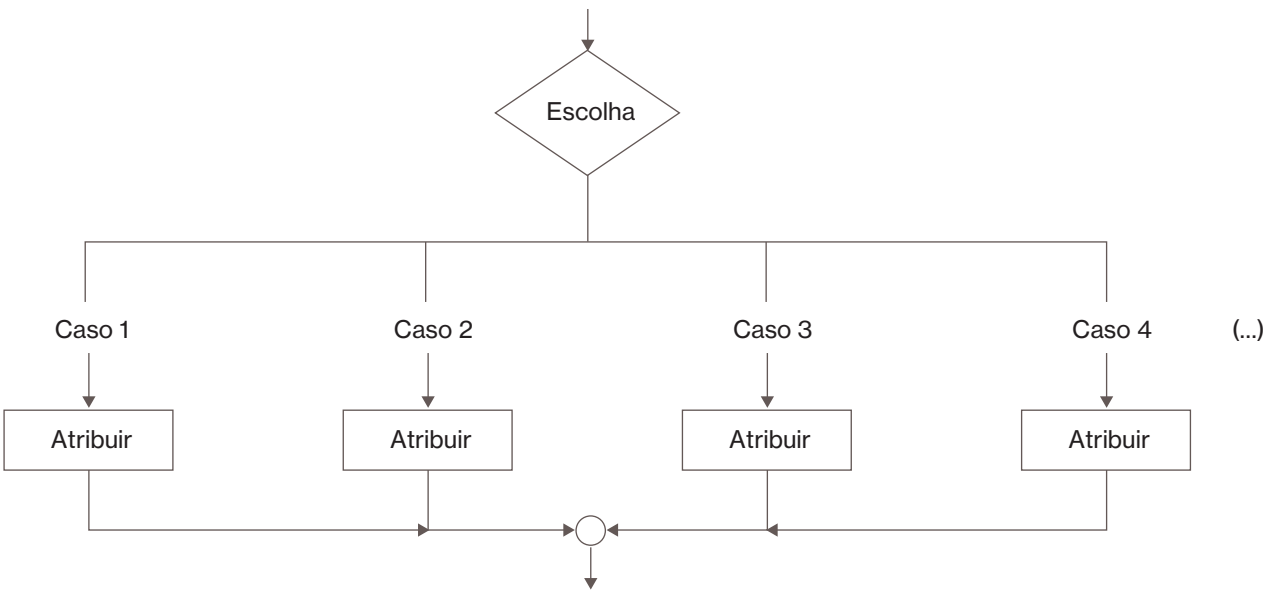
L – "Ligar"
D – "Desligar"
C – "Café"
A – "Água"

Para decidir o que escrever no ecrã, o algoritmo apenas tem de verificar se uma variável "botao" tem o valor L, D, C ou A.

Uma **estrutura de decisão múltipla** permite que o algoritmo selecione um de entre vários conjuntos de instruções, dependendo do valor de uma variável ou expressão.

Em fluxograma, numa estrutura de decisão múltipla, é usado o símbolo losango com múltiplas opções de saída, uma para cada caso possível, e são definidas instruções para cada opção.

Fluxograma



Em pseudocódigo, numa estrutura de decisão múltipla é utilizada a instrução **Escolha... Caso... Caso ... Caso... (...) Fimescolha**.

Pseudocódigo

```
Início
...
Escolha <variável>
  Caso <valor1>:
        <instruções>
  Caso <valor2>:
        <instruções>
  Caso <valor3>:
        <instruções>
  (...)
Fimescolha
...
Fim
```

Exemplo

Considerando a máquina de café referida anteriormente, o algoritmo que decide a mensagem a aparecer no monitor da máquina, em função do botão que foi premido, pode ser escrito em pseudocódigo por:

```
Algoritmo Monitor_maquina
Var
    botao: caractere
Início
Ler botao
Escolha botao
    Caso L:
      Escrever "Ligar"
    Caso D:
      Escrever "Desligar"
    Caso C:
      Escrever "Café"
    Caso A:
      Escrever "Água"
Fimescolha
Fim
```

<Modo ON #48>

Constrói um algoritmo que leia uma variável dia_semana, que pode assumir os valores 1, 2, 3, 4, 5, 6 e 7, e que, em função do valor dessa variável, escreva no ecrã a mensagem respetiva:

1: "Domingo"
2: "Segunda"
3: "Terça"
4: "Quarta"
5: "Quinta"
6: "Sexta"
7: "Sábado"

☑ Not@ que:

Só podes utilizar uma estrutura de decisão múltipla quando as várias opções dependem do valor de uma só variável ou expressão.

Exemplo

É possível escrever em pseudocódigo um algoritmo que simule uma calculadora simples que apresenta o resultado do cálculo da soma, diferença, produto ou divisão de dois números, em função da variável **operador** assumir os valores "+", "–", "×" ou ":".

```
Algoritmo Calculadora
Var
    n1, n2: real
    operador: caractere
Início
  Ler n1
  Ler operador
  Ler n2
  Escolha operador
    Caso "+" :
      Escrever n1 + n2
    Caso "–" :
      Escrever n1 – n2
    Caso "×" :
      Escrever n1 * n2
    Caso ":" :
      Escrever n1 / n2
Fimescolha
Fim
```

<Modo ON #49>

1 **Constrói um algoritmo que leia três variáveis numéricas do dia, mês e ano, e escreva no ecrã a data num formato em que o mês é escrito com uma palavra. Por exemplo, quando lidos os três valores 20, 03, 2026, irá aparecer escrito:**

20 março 2026

Repara que a decisão quanto ao que escrever apenas depende da variável mês que pode assumir os valores 01, 02, 03, 04, 05, 05, 07, 08, 09, 10, 11 e 12. Por exemplo, para o valor 01 podes usar a instrução:

Caso 01:
  Escrever dia, " Janeiro ", ano

2 **Constrói um algoritmo que leia uma letra A, B, C, D ou F e escreva no ecrã a mensagem correspondente:**

A : "Excelente"
B : "Muito Bom"
C : "Bom"
D : "Suficiente"
F : "Insuficiente"

3 **Constrói um algoritmo que leia o número do mês e escreva no ecrã a estação correspondente:**

12, 1, 2 : "Inverno"
3, 4, 5 : "Primavera"
6, 7, 8 : "Verão"
9, 10, 11 : "Outono"

4 **Constrói um algoritmo que leia um número entre 1 e 5, que representa o nível de dificuldade de um jogo, e escreva no ecrã a informação correspondente:**

1 : "Muito fácil"
2 : "Fácil"
3 : "Normal"
4 : "Difícil"
5 : "Muito difícil"

Se o nível de dificuldade for maior ou igual a 4 deve apresentar também a seguinte mensagem:

"Pontuação bónus ativa"

Testa os teus conhecimentos

1 **Constrói um algoritmo que simule uma consulta numa caixa ATM. O algoritmo deve ler:**

- o saldo anterior;
- o débito;
- o crédito.

Para além de calcular e escrever o valor do saldo atual, saldo atual = saldo anterior – débito + crédito, o algoritmo deve escreve:

– "Saldo Positivo", se o valor do saldo atual for maior do que zero;

– "Saldo Negativo", caso contrário.

2 **Considera o seguinte algoritmo:**

```
Algoritmo Troca
Var
   x, y, z: real
   resposta: caractere
Início
   Ler x
   Ler y
   z ← (x * y) + 5
   Se  z <= 0  Então
      resposta ← "A"
   Senão
      Se z <= 100 Então
         resposta ← "B"
      Senão
         resposta ←  "C"
      Fimse
   Fimse
   Escrever  z, resposta
Fim
```

Faz o *tracing* do algoritmo para diferentes valores de entrada e completa a tabela seguinte.

x	y	z	resposta
4	3		
120	2		
–1	8		
5	–2		
4	30		

Testa os teus conhecimentos

3 **A jornada de trabalho semanal de um trabalhador do município é de 40 horas. Um trabalhador que trabalhe mais de 40 horas numa semana recebe por cada hora extra de trabalho 1,5 vezes o valor regular pago por hora.**

Constrói um algoritmo que, para um determinado trabalhador, leia:

– o número de horas trabalhadas numa semana;

– o valor regular pago por hora.

O algoritmo deve depois calcular o salário semanal do trabalhador (40 horas são pagas a um valor regular e as restantes são pagas a 1,5 vezes esse valor).

Testa o teu algoritmo com diferentes valores (por exemplo, 41 horas, 45 horas e 50 horas).

4 **Constrói um algoritmo que leia:**

- a quantidade atual de leite armazenado numa escola;
- a quantidade mínima de leite necessário;
- a quantidade máxima que é possível armazenar.

Deve, depois, calcular o valor médio entre a quantidade máxima e a quantidade mínima e escrever:

– "Comprar leite", se a quantidade atual for menor do que o valor médio;

– "Não comprar leite", caso contrário.

5 **Uma frutaria está a vender frutas com a seguinte tabela de preços:**

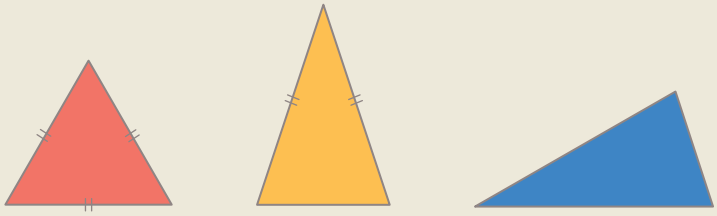
	Até 5 kg	Mais de 5 kg
Banana (kg)	250$	220$
Maçã (kg)	200$	180$

Se o cliente comprar mais de 8 kg em frutas ou o valor total da compra ultrapassar 1800$, receberá ainda um desconto de 10% sobre este total.

Constrói um algoritmo que leia a quantidade (em kg) de banana e a quantidade (em kg) de maçãs escolhidas pelo cliente e que escreva o valor total a ser pago.

6 **Considera o seguinte algoritmo:**

```
Algoritmo Triângulos
Var
    a, b, c, p: inteiro
Início
    Escrever "Introduza as medidas dos 3 lados do triângulo"
    Ler a, b, c
    p ← a + b + c
    Se  a < (b + c)  e  b < (a + c)  e  (c < a + b)  Então
        Se  a == b  e  b == c  Então
            Escrever "O perímetro do triângulo equilátero é ", p
        Senão
            Se  a == b  ou  b == c  ou  a == c  Então
                Escrever "O perímetro do triângulo isósceles é ", p
            Senão
                Escrever "O perímetro do triângulo escaleno é ", p
            Fimse
        Fimse
    Senão
        Escrever "Não é possível formar um triângulo com essas medidas"
    Fimse
Fim
```



Faz o *tracing* do algoritmo e preenche a tabela para os seguintes valores das variáveis:

a	b	c	p	Mensagem que aparece no ecrã
2	3	4		
2	3	5		
5	5	5		
4	3	3		
2	6	3		

Testa os teus conhecimentos

7 **Um posto de combustíveis tem afixada uma tabela com descontos:**

	Até 20 litros	Mais de 20 litros
Gasóleo	desconto de 3% por litro	desconto de 5% por litro
Gasolina	desconto de 4% por litro	desconto de 6% por litro

Escreve um algoritmo que leia o número de litros vendidos e o tipo de combustível (gasóleo ou gasolina) e escreva o valor a ser pago pelo cliente.

valor real = número de litros × preço por litro

valor a pagar = valor real – (valor real × desconto)

Considera que o preço da gasolina é 122$ por litro e o preço do gasóleo é 104$ por litro.

8 **Constrói um algoritmo que possibilite o acesso a um programa de computador.**

O algoritmo deve solicitar e ler o ID de utilizador.

– Se este ID for diferente do código armazenado internamente (igual a 1234), deve ser apresentada a mensagem "Utilizador inválido".

– Se o ID estiver correto, o algoritmo deve solicitar e ler a *password*.

– Se a *password* estiver errada (a certa é 9999), deve ser apresentada a mensagem "*Password* incorreta".

– Se a *password* estiver correta, deve ser apresentada a mensagem "Acesso permitido'.

3.3. Estruturas de repetição

Um dos principais fatores que contribuíram para o sucesso da utilização de computadores na resolução de problemas é a sua capacidade de repetir o processamento de um conjunto de operações para grandes quantidades de dados.

As **estruturas de repetição** permitem que um algoritmo execute as mesmas instruções várias vezes sem ter de as escrever repetidamente.

Por outro lado, muitas vezes não se sabe previamente quantas vezes vai ser necessário repetir a execução da mesma instrução e pode ser usada uma condição que defina quanto deve durar um ciclo de repetição.

As **estruturas de repetição**, ou **ciclos**, permitem a execução de forma repetitiva de um conjunto de instruções. Esta execução pode depender do valor lógico de uma condição que é testada em cada iteração para decidir se a execução do ciclo continua ou termina.

As estruturas de repetição subdividem-se em três tipos.

Tipos de estruturas de repetição

Estrutura PARA	Quando se conhece o número de vezes que é necessário repetir a instrução. **PARA ...** **FAZER ...**
Estrutura ENQUANTO	Quando o número de vezes que é necessário repetir a instrução depende de uma condição (condição testada no início). **ENQUANTO ...** **FAZER ...**
Estrutura REPETIR... ATÉ	Quando o número de vezes que é necessário repetir a instrução depende de uma condição (condição testada no fim). **REPETIR ...** **ATÉ ...**

Manual Interativo

EVStory
Vamos aprender juntos sobre programação em Scratch

Vídeos
Programar em Scratch

Um jogo com labirinto no Scratch

Um jogo de caça o Gobo no Scratch

Estrutura: PARA

Uma estrutura de repetição **PARA** permite repetir a execução de um conjunto de instruções durante um número predefinido de vezes.

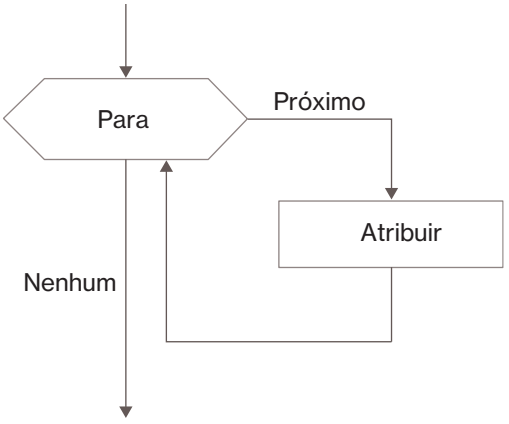
Exemplo

No dia a dia usas um ciclo de repetição **PARA** quando, por exemplo, calculas os primeiros 20 múltiplos de 30. O que fazes é calcular $30 \times i$ para todos os números i de 1 até 20. Ou seja,

PARA i de 1 até 20 **FAZER** calcular $30 \times i$

Em fluxograma, numa estrutura de repetição **PARA**, é usado o símbolo hexágono, onde é definido o número de vezes que o ciclo de instruções deve ser repetido.

Fluxograma



Em pseudocódigo, é utilizada a instrução **Para... Fazer... Fimpara**.

Pseudocódigo

```
Início
...
Para <variável> ← <início> até <fim> Fazer
  <instruções>
Fimpara
...
Fim
```

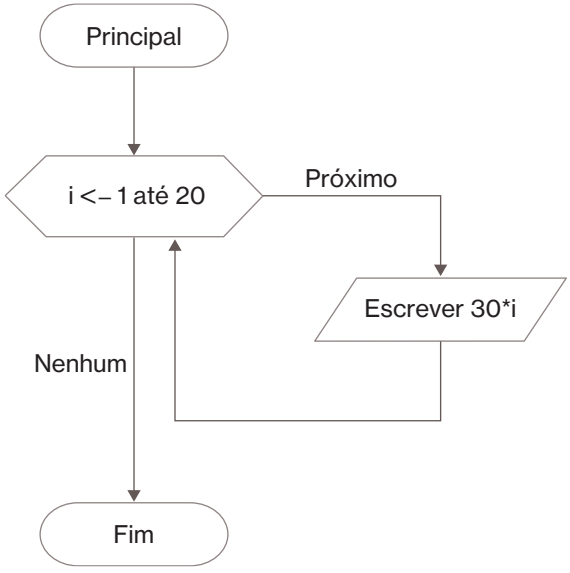
Nas **estruturas de repetição PARA** as instruções são executadas repetidamente um número preciso de vezes.

Exemplo

1. Exemplo de um algoritmo, em pseudocódigo, que calcula os primeiros 20 múltiplos de 30.

```
Algoritmo Múltiplos_30
Var
  i: inteiro
Início
  Para i ← 1 até 20 Fazer
      Escrever 30 * i
  Fimpara
Fim
```

2. O mesmo algoritmo pode ser representado em fluxograma.



<Modo ON #50>

Faz o *tracing* do algoritmo seguinte, de modo a descobrires o que vai aparecer no ecrã.

```
Algoritmo Múltiplos_7
Var
  i: inteiro
Início
  Para i ← 1 até 10 Fazer
      Escrever 7 * i
  Fimpara
Fim
```

O número de vezes que o ciclo é repetido pode ser **fixo** no algoritmo ou pode ser um **valor dado por uma variável**.

Exemplo

1. Exemplo de um algoritmo, em pseudocódigo, em que o utilizador introduz quantos múltiplos de 30 pretende que apareçam escritos no ecrã.

```
Algoritmo Múltiplos_30
Var
  n, i: inteiro
Início
  Ler n
  Para i ← 1 até n Fazer
      Escrever 30 * i
  Fimpara
Fim
```

2. Exemplo de um algoritmo que calcula a soma das pontuações obtidas nas perguntas de um teste. O utilizador tem de começar por introduzir o número de perguntas do teste e depois a pontuação obtida em cada pergunta.

```
Algoritmo Classificação_teste
Var
  n, i: inteiro
  soma, p: real
Início
  soma ← 0
  Escrever "Quantas perguntas tem o teste?"
  Ler n
  Para i ← 1 até n Fazer
      Escrever "Introduz a pontuação obtida "
      Ler p
      soma ← soma + p
  Fimpara
  Escrever "Obtiveste no total ", soma, " pontos"
Fim
```

<Modo ON #51>

1. **Constrói um algoritmo que leia 10 números e que depois escreva a soma dos mesmos.**
2. **Constrói um algoritmo que pergunte ao utilizador quantos números quer introduzir e que depois leia os números e escreva a soma de todos os números lidos.**
3. **Num armazém está a ser feito o levantamento do valor total de todas as mercadorias armazenadas. Escreve um algoritmo que peça que sejam introduzidas as seguintes informações:**
   – o número total de mercadorias no armazém;
   – o valor de cada mercadoria.

   No final, o algoritmo deve escrever qual o valor total das mercadorias armazenadas.

Numa estrutura de repetição, as instruções que são repetidas no ciclo podem incluir vários tipos de instruções e de estruturas.

Exemplo

Exemplo de um algoritmo que lê 20 números inteiros e calcula a soma de todos os números lidos que têm um valor inferior a 50.

```
Algoritmo Soma_inferior_50
Var
  i: inteiro
  soma, n: inteiro
Início
  soma ← 0
  Para  i ← 1 até 20  Fazer
      Ler n
      Se  n < 50  Então
        soma ← soma + n
      Fimse
  Fimpara
  Escrever "A soma dos números inferiores a 50 é ", soma
Fim
```

<Modo ON #52>

1. **Constrói um algoritmo que leia 500 números e que calcule a soma de todos os números lidos que com valor inferior a 10.**
2. **Constrói um algoritmo que leia 100 números e que, no final, escreva o maior número lido.**
   Sugestão: Usa uma variável **max** que te permita ir comparando os vários valores lidos.
   Podes começar por ler o primeiro número e ser esse o valor inicial da variável **max**.
   Depois, usa uma estrutura de repetição **PARA** que possibilite a leitura dos 99 números seguintes e compara cada valor lido com o valor da variável **max**.
   Sempre que o novo número lido seja maior do que o valor da variável **max**, o algoritmo deve substituir o valor da variável pelo novo número através de uma estrutura condicional:

   Se n > max **Então** max ← n **Fimse**

☑ Not@ que:

1. Podes utilizar uma estrutura de repetição **PARA** sempre que for possível saber exatamente quantas vezes o ciclo deve ser repetido.
2. Uma estrutura de repetição **PARA** é muito fácil de ler, pois aparece na mesma linha a variável de controlo, o valor inicial e o valor final. Utilizar este tipo de estrutura facilita a leitura e a compreensão de um algoritmo.

<Modo ON #53>

1. **Constrói um algoritmo que calcule e escreva os primeiros 20 múltiplos de 9.**
2. **Constrói um algoritmo que calcule e escreva a média aritmética dos números inteiros entre 15 (inclusive) e 50 (inclusive).**
3. **Constrói um algoritmo que leia 100 números e que calcule a média aritmética de todos os números lidos.**
4. **Constrói um algoritmo que pergunte e leia quantos alunos realizaram um teste e que depois leia as classificações dos alunos. No final, deve apresentar a média aritmética de todas as classificações.**

Estrutura: ENQUANTO

Uma **estrutura de repetição ENQUANTO** repete a execução de um conjunto de instruções enquanto uma determinada condição lógica for verdadeira. Esta estrutura é adequada quando não é possível conhecer previamente o número de vezes que o ciclo deve ser repetido.

Manual Interativo

Vídeos
MIT App Inventor – Screen1

MIT App Inventor – Screen2

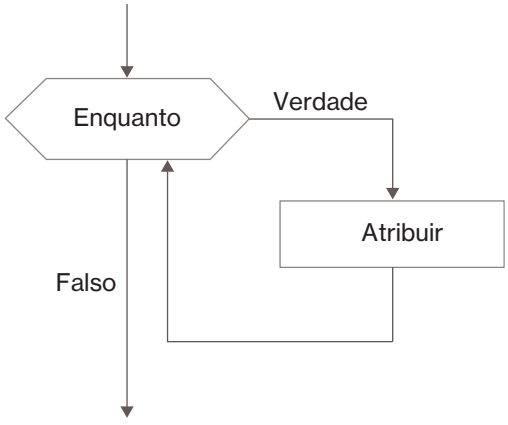
Exemplo

No dia a dia usas um ciclo de repetição **ENQUANTO** quando, por exemplo, bebes um gole de sumo sempre que verificas que ainda há sumo no copo. Ou seja,

ENQUANTO o copo tem sumo **FAZER** beber um gole

Em fluxograma, numa estrutura de repetição **ENQUANTO**, é usado o símbolo hexágono para definir a condição que determina se o ciclo de instruções deve ser repetido.

Fluxograma



Em pseudocódigo, é utilizada a instrução **Enquanto... Fazer... Fimenquanto**.

Pseudocódigo

```
Início
...
Enquanto <condição> Fazer
  <instruções>
Fimenquanto
...
Fim
```

Nas **estruturas de repetição ENQUANTO** as instruções são executadas repetidamente enquanto a condição inicial continuar a ser verdadeira.

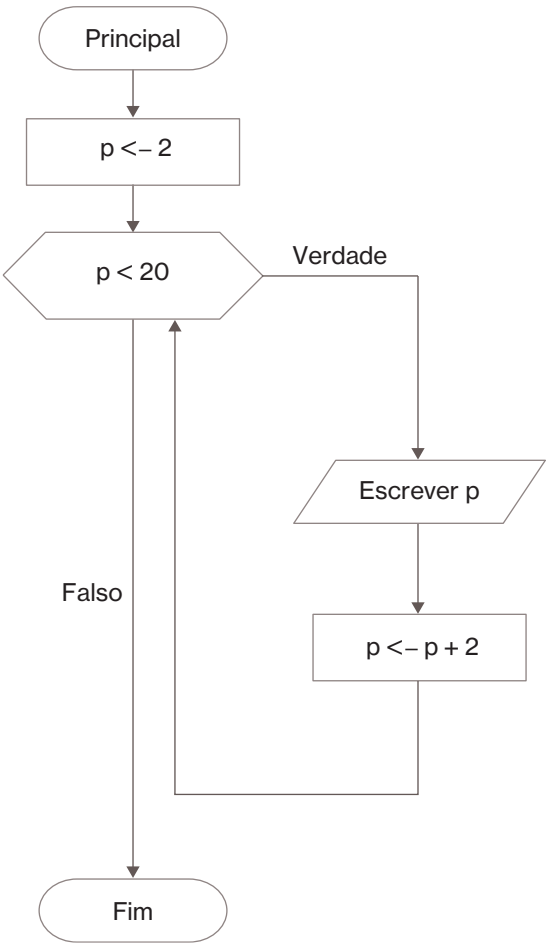
Exemplo

1. Exemplo de um algoritmo, em pseudocódigo, que permite escrever sucessivamente os números pares inferiores a 20.

```
Algoritmo Números_pares
Var
  p: inteiro
Início
  p ← 2
  Enquanto p < 20 Fazer
     Escrever p
     p ← p + 2
  Fimenquanto
Fim
```

Ao fazer o *tracing*, verificamos que o algoritmo escreve os números 2, 4, 6, 8, 10, 12, 14, 16 e 18. Quando a variável *p* passa a ter o valor 20, o ciclo de repetição já não é iniciado e o número 20 não é escrito.

2. O mesmo algoritmo pode ser representado em fluxograma.



<Modo ON #54>

1 **Faz o *tracing* do algoritmo seguinte para descobrires quantas potências de 2 são escritas.**

```
Algoritmo Potencia_2
Var
  p: inteiro
Início
  p ← 1
  Enquanto p < 500 Fazer
      Escrever p
      p ← p * 2
  Fimenquanto
Fim
```

2 **Constrói um algoritmo que escreva os múltiplos de 7 inferiores a 100 (usa a estrutura de repetição ENQUANTO).**

Exemplo

O algoritmo seguinte permite calcular o resto da divisão inteira entre dois números a e b.

```
Algoritmo divisão_inteira
Var
  a, b, r : inteiro
Início
  Ler a, b
  r ← a
  Enquanto r >= b Fazer
      r ← r - b
  Fimenquanto
  Escrever r
Fim
```

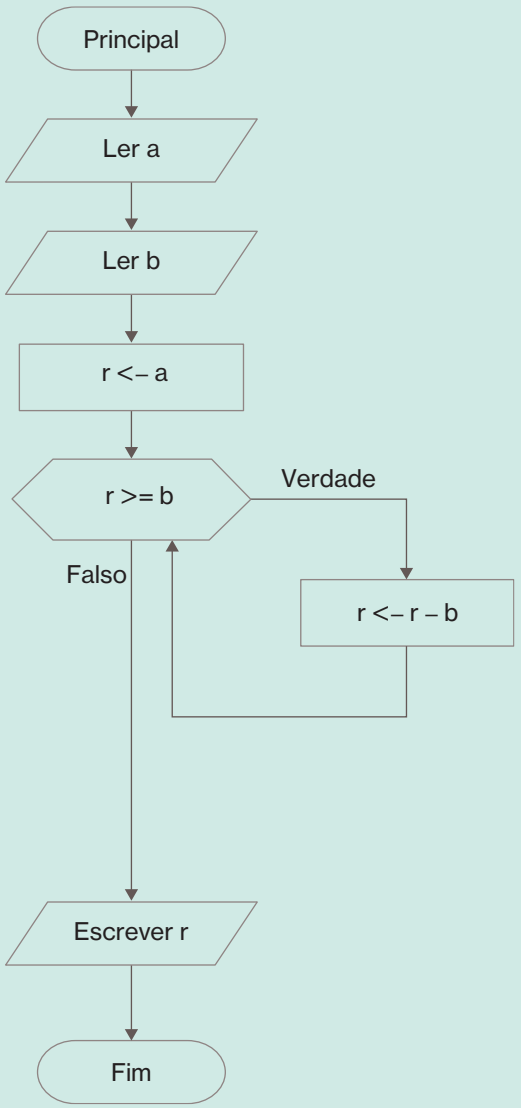
Introduzindo, por exemplo, a = 17 e b = 5, o algoritmo calcula sucessivamente:

r = 17
r = 17 – 5 = 12
r = 12 – 5 = 7
r = 7 – 5 = 2

O resto da divisão de 17 por 5 é 2.

<Modo ON #55>

Considera a representação em fluxograma de um algoritmo que calcula o resto da divisão inteira entre dois números a e b.



Faz o *tracing* do algoritmo para diferentes valores de a e b e completa a tabela abaixo.

a	b	r	a = b × q + r
17	5	2	17 = 5 × 3 + 2
24	7		
34	6		
41	6		

Validação de valores introduzidos

Num algoritmo, a estrutura de repetição **ENQUANTO** é muito adequada para **validar os valores introduzidos** pelo utilizador.

Exemplo

1. Na validação de uma *password*, a estrutura **ENQUANTO** garante que o algoritmo só avança quando a *password* introduzida for a correta.

```
Ler password
Enquanto password <> "1234" Fazer
    Escrever "Password incorreta. Introduza nova password."
    Ler password
Fimenquanto
```

2. Na validação de que um número introduzido é diferente de 0, a estrutura **ENQUANTO** é apenas executada se o número lido for 0, pedindo-se nesse caso um novo número e repetindo sucessivamente até que seja introduzido um número diferente de 0.

```
Ler n
Enquanto n == 0 Fazer
    Escrever "Introduza um número diferente de 0."
    Ler n
Fimenquanto
```

<Modo ON #56>

1. Constrói um algoritmo que peça ao utilizador para adivinhar um número secreto. Escolhe um número e usa a estrutura **ENQUANTO** para que o algoritmo peça novos números ao utilizador até acertar.
2. Constrói um algoritmo que solicite ao utilizador que introduza dois números a e b e calcule o resultado da divisão de a por b. Antes do cálculo, deve verificar se b é diferente de 0 (caso não seja, deve solicitar um novo valor b).
3. Constrói um algoritmo que solicite ao utilizador que introduza a medida da largura e do comprimento de um retângulo e se calcule a sua área. Antes do cálculo da área, o algoritmo deve verificar se ambos os valores são positivos (caso não sejam, o algoritmo deve solicitar um novo valor).

Contadores

Um **contador** é uma variável usada para registar quantas vezes algo acontece. Uma variável contador deve ser atualizada ao longo da execução das instruções e pode ser utilizada para:

– contar repetições;
– contar valores;
– controlar ciclos.

1. A estrutura de repetição **PARA** tem uma variável contador incorporada, usualmente designada por "i".

```
...
 Para i ← 1 até 100 Fazer
     Escrever i
 Fimpara
...
```

2. Pode ser usada uma variável contador "c" para contar quantos números positivos foram lidos.

```
...
c ← 0
Para i ← 1 até 100 Fazer
     Ler n
     Se n > 0 Então
        c ← c + 1
     Fimse
Fimpara
...
```

3. Numa estrutura de repetição **ENQUANTO**, pode ser usada uma variável contador para controlar o número de vezes que o ciclo repete.

```
...
c ← 1
Enquanto c <= 100 Fazer
     Escrever c
     c ← c + 1
Fimenquanto
...
```

Comparação entre as estruturas de repetição PARA e ENQUANTO

Há muitas situações em que tanto pode ser usada uma estrutura de repetição **PARA** como uma estrutura **ENQUANTO**. Ambas permitem repetir instruções e parar num determinado momento. A diferença está no modo como é controlado o ciclo de repetição.

Exemplo

O algoritmo seguinte permite escrever a tabuada de um número lido usando a estrutura de repetição **PARA**.

```
Algoritmo Tabuada
Var
  n, i: inteiro
Início
  Ler n
  Para  i ← 1 até 10 Fazer
      Escrever  i, " × ", n, "=", i*n
  Fimpara
Fim
```

Alternativamente, a mesma tarefa pode ser executada através de um algoritmo que use a estrutura de repetição **ENQUANTO** e uma variável contador.

```
Algoritmo Tabuada
Var
  n, c: inteiro
Início
  Ler n
  c ← 1
  Enquanto  c  <= 10 Fazer
      Escrever  c, " × ", n, "=", c*n
      c ← c + 1
  Fimenquanto
Fim
```

<Modo ON #57>

Constrói um algoritmo que escreva a tabuada do número 5 usando:

a) uma estrutura de repetição **PARA**;

b) uma estrutura de repetição **ENQUANTO**.

Estrutura: REPETIR... ATÉ

Uma **estrutura de repetição REPETIR... ATÉ** repete a execução de um conjunto de instruções até que uma determinada condição lógica seja verdadeira. Esta estrutura deve ser utilizada sempre que se pretende que as instruções sejam executadas pelo menos uma vez.

Exemplo

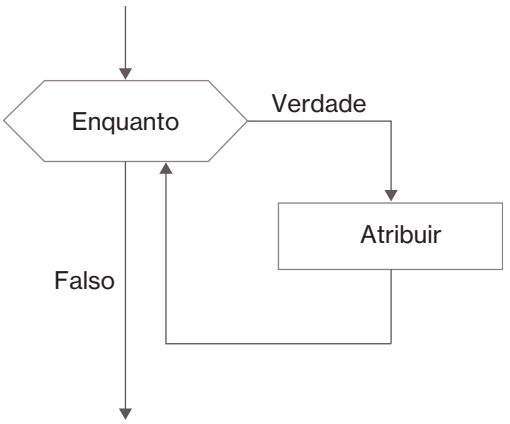
No dia a dia usas um ciclo de repetição **REPETIR** quando, por exemplo, estás a tentar adivinhar um número escolhido por um colega. Ou seja,

REPETIR dizer um número **ATÉ** acertar no número correto

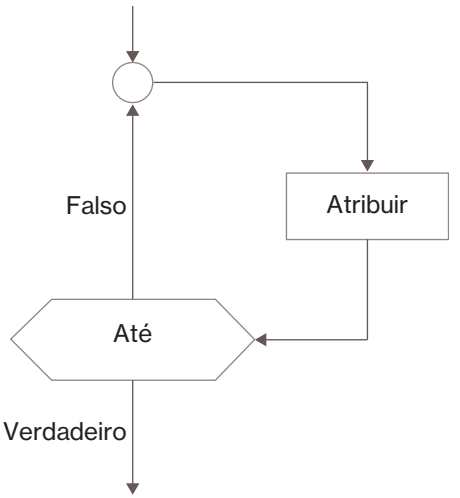
Em fluxograma, numa estrutura de repetição **REPETIR... ATÉ**, é usado o símbolo hexágono para definir a condição que determina se o ciclo de instruções deve continuar a ser repetido:

Fluxograma

ENQUANTO



REPETIR... ATÉ



A estrutura **REPETIR... ATÉ** é muito semelhante à estrutura **ENQUANTO** (ambas dependem de uma condição), mas neste caso a condição é verificada no final de cada ciclo de instruções.

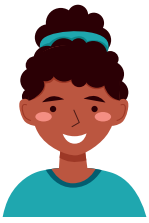
Diferenças entre as estruturas ENQUANTO e REPETIR... ATÉ

ENQUANTO	REPETIR... ATÉ
Testa antes	Testa depois
Repete enquanto verdadeiro	Repete até verdadeiro
Pode não executar	Executa sempre uma vez

Em pseudocódigo, é utilizada a instrução **Repetir... Até...**.

Pseudocódigo

```
Início
...
Repetir
  <instruções>
Até <condição>
...
Fim
```



Nas **estruturas de repetição REPETIR... ATÉ** as instruções são executadas repetidamente até que uma condição final passe a ser verdadeira.

Exemplo

1. Exemplo de estrutura que permite ler sucessivamente números até que seja introduzido o número 0.

```
...
Repetir
    Ler n
Até n == 0
...
```

2. Exemplo de estrutura que permite escrever sucessivamente números pares inferiores a 20.

```
...
n ← 2
Repetir
    Escrever n
    n ← n + 2
Até n >= 20
...
```

Ao fazer o *tracing*, verificamos que o algoritmo escreve os números 2, 4, 6, 8, 10, 12, 14, 16 e 18. Quando a variável *n* passa a ter o valor 20, o ciclo de repetição é terminado e o número 20 não é escrito.

<Modo ON #58>

1. **Constrói um algoritmo que escreva os múltiplos de 10 inferiores a 200 (usa a estrutura de repetição REPETIR... ATÉ).**

2. **Considera o algoritmo seguinte:**

```
Algoritmo Soma
Var
  soma, num: inteiro
Início
  soma ← 0
  Repetir
      Ler num
      soma ← soma + num
  Até num == 0
Fim
```

   a) Faz o *tracing* do algoritmo, supondo que são introduzidos sucessivamente os seguintes números:
   3, 4, 5, 2, 7, 1, 1, 3, 6, 4, 2, 0, 2, 2, ...

   b) Em que momento o algoritmo foi terminado? Qual o valor final da variável soma?

3. **Constrói um algoritmo que escreva os números de 1 (inclusive) a 10 (inclusive) em ordem crescente, utilizando a estrutura REPITA... ATÉ e uma variável contador.**

A estrutura de repetição **REPETIR... ATÉ** também pode ser usada para validação dos valores introduzidos num algoritmo.

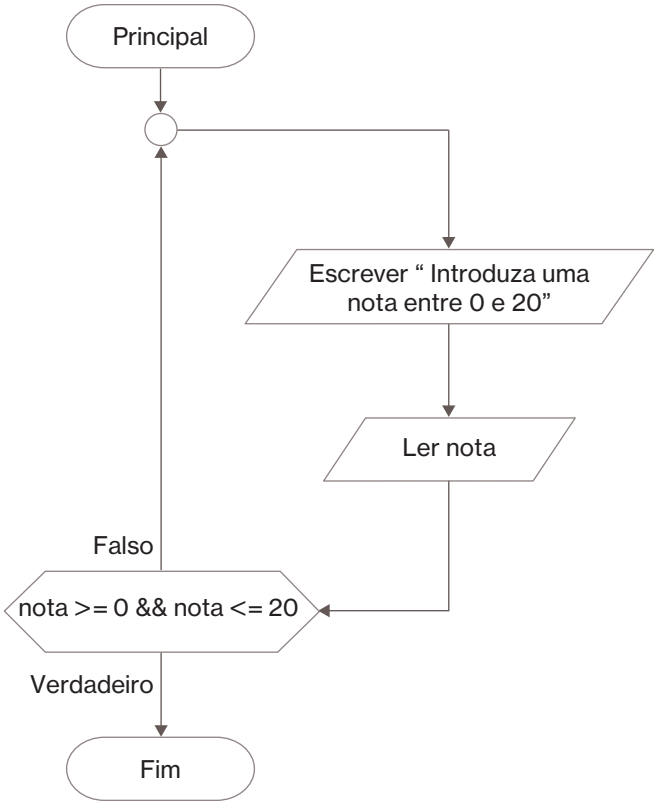
Exemplo

O algoritmo seguinte valida se um número introduzido é positivo. Se não for positivo, é novamente solicitado um número.

```
Algoritmo numero_positivo
Var
  n: inteiro
Inicio
  Repetir
     Escrever "Introduza um número positivo."
     Ler n
  Até n > 0
  Escrever "Número valido!"
Fim
```

Exemplo

O algoritmo seguinte usa a estrutura **REPETIR... ATÉ** para garantir que o valor de uma nota introduzida está situado entre 0 e 20.



<Modo ON #59>

1. **Constrói um algoritmo que leia um número *N* e escreva todos os números de 1 até *N*. O algoritmo deve começar por garantir que o número *N* introduzido é um número positivo.**

2. **Constrói um algoritmo que solicite que seja introduzido um número secreto.**
   – O número secreto é 7744.
   – Quando o utilizador acertar no número secreto, deve aparecer a mensagem seguinte:
   "Número secreto correto!"

3. **Constrói um algoritmo que leia números até que seja introduzido o número 999. No final, deve apresentar no ecrã o valor da soma total de todos os números lidos.**
   Sugestão: Utiliza uma variável soma para ires acumulando a soma dos valores que são lidos.

☑ Not@ que:

As estruturas **ENQUANTO** e **REPETIR... ATÉ** são adequadas quando não sabemos quantas repetições serão necessárias.

Contudo, se a condição não for definida de um modo correto, corremos o risco de criar um ciclo infinito – o algoritmo nunca termina a repetição.

Exemplo

A estrutura de repetição seguinte tem um erro: como a variável 'num' não altera o seu valor, a condição que permite terminar o ciclo nunca é verdadeira e o ciclo repete-se infinitamente.

```
...
num ← 1
Repetir
    Escrever num
Até  num > 5
...
```

Para corrigir este erro, é necessário garantir que o valor da variável 'num' é alterado e que em algum momento a condição que permite terminar o ciclo é verdadeira:

```
...
num ← 1
Repetir
    Escrever num
   num ← num + 1
Até  num > 5
...
```

<Modo ON #60>

1. **Constrói um algoritmo que escreva os 10 primeiros números inteiros maiores do que 100, usando:**
   a) a estrutura de repetição **ENQUANTO**;
   b) a estrutura de repetição **REPETIR... ATÉ**.
2. **Constrói um algoritmo que leia um número *N* e escreva os 10 primeiros números inteiros maiores que *N*.**

Estruturas de repetição encadeadas

Por vezes, na construção de um algoritmo, para tornar os procedimentos mais eficientes, é necessário utilizar uma estrutura de repetição dentro de outra.

As **estruturas de decisão encadeadas** são usadas quando é necessário executar dentro de um ciclo de repetição um conjunto de instruções que exigem o recurso a um outro ciclo de repetição.

Exemplo

No teu dia a dia, imagina que tens uma app que te avisa "Vai para a aula!".
A semana tem cinco dias e considera que em cada dia tens seis disciplinas. Assim, a tua app teria um funcionamento do tipo:

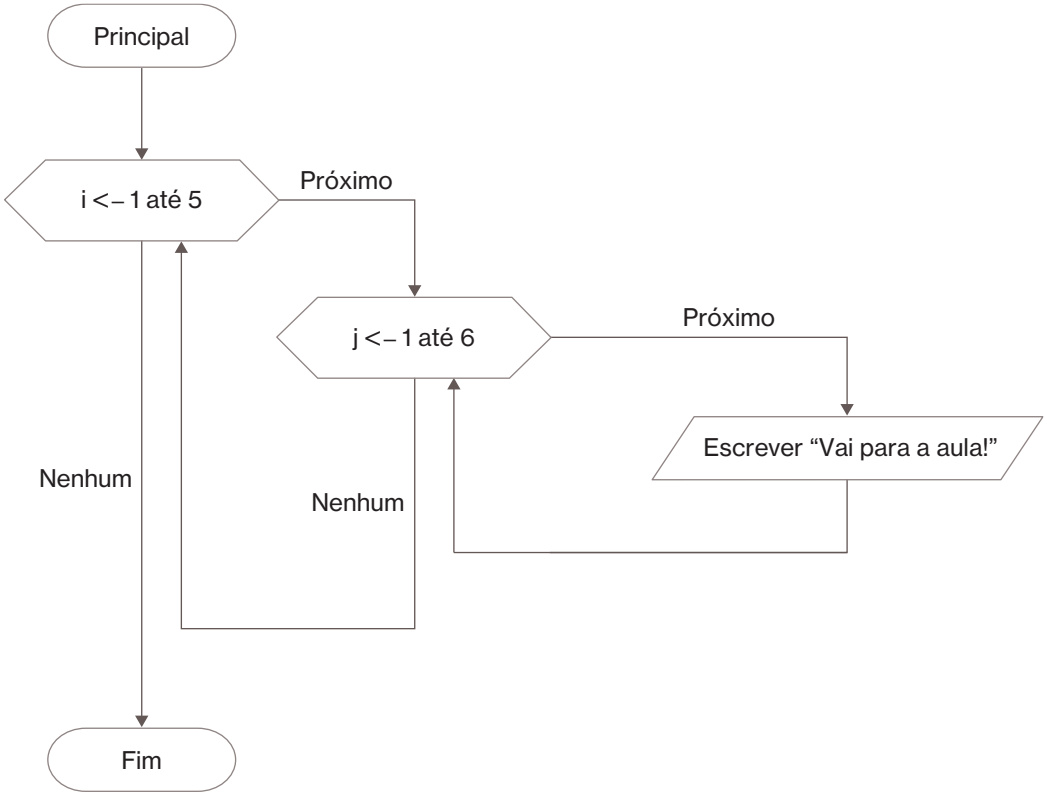
PARA i de 1 até 5 **FAZER**
    PARA j de 1 até 6 **FAZER** escrever "Vai para a aula!"

A primeira estrutura de repetição é executada cinco vezes, uma para cada dia da semana, e para cada uma dessas vezes a segunda estrutura de repetição é executada seis vezes.

5 dias
× 6 disciplinas

No final dos cinco dias da semana, a app escreveu 30 vezes a frase "Vai para a aula!".

A representação em fluxograma do algoritmo da app é:



☑ Not@ que:

1. Quando escreveres em pseudocódigo estruturas de repetições encadeadas é importante que uses indentação nas instruções. Esse cuidado facilita a leitura e compreensão do algoritmo e ajuda a evitar erros.
2. Tal como nas estruturas condicionais, após introduzires várias estruturas de repetição no teu algoritmo, deves sempre verificar se cada uma delas é iniciada e terminada.

Exemplo

O algoritmo seguinte permite escrever a tabuada dos números de 1 a 10 usando duas estruturas de repetição **PARA** encadeadas.

```
Algoritmo Tabuada_numeros_1_a_10
Var
   i, j: inteiro
Início
   Para  i ← 1 até 10   Fazer
         Escrever  "Tabuada do ",  i
         Para  j ← 1 até 10  Fazer
               Escrever  j, " x ", i, " = ", j*i
         Fimpara
   Fimpara
Fim
```

A execução do algoritmo irá apresentar no ecrã:

```
Tabuada do 1
1 × 1 = 1
2 × 1 = 2
3 × 1 = 3
...
```

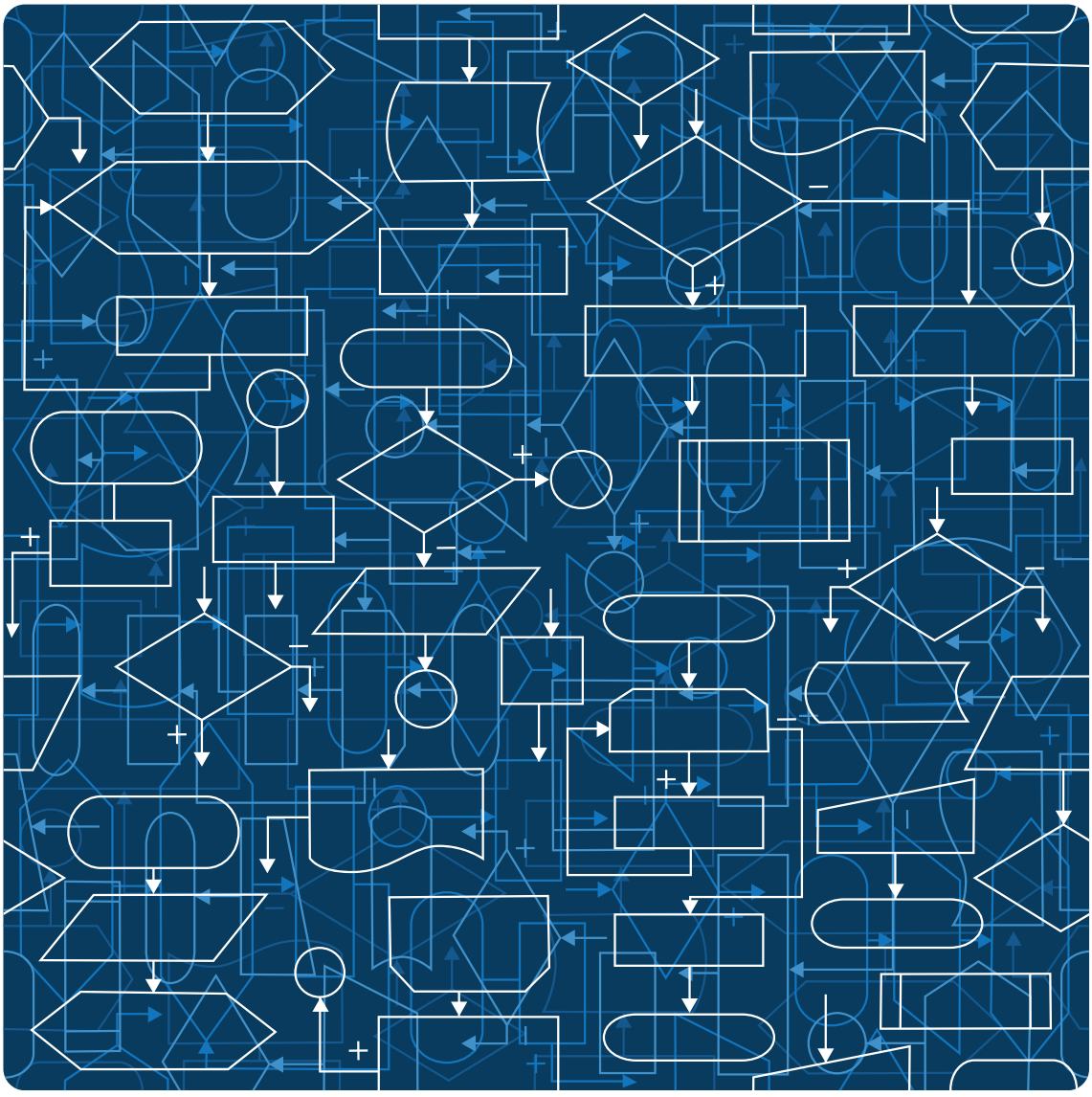
<Modo ON #61>

Constrói um algoritmo que escreva as sequências de números seguintes:

1, 1 2 3 4 5 6 7 8 9 10
2, 1 2 3 4 5 6 7 8 9 10
3, 1 2 3 4 5 6 7 8 9 10

e assim sucessivamente, até:

10, 1 2 3 4 5 6 7 8 9 10



Testa os teus conhecimentos

1 **Qual é a estrutura de repetição que não depende da avaliação de uma condição?**

(A) REPETIR... ATÉ.

(B) ENQUANTO.

(C) PARA.

2 **Constrói um algoritmo que leia 10 números e informe quantos deles são negativos.**

3 **Constrói os seguintes algoritmos:**

a) Um algoritmo que escreva os números de 1 (inclusive) a 10 (inclusive) por ordem crescente.

b) Um algoritmo que escreva os números de 1 (inclusive) a 10 (inclusive) por ordem decrescente.

4 **Um engenheiro pretende determinar o peso total de todos os contentores que estão a ser colocados num navio.**

a) Escreve um algoritmo que peça a entrada das seguintes informações:
– o número total de contentores a carregar no navio;
– o peso de cada contentor.

No final, o algoritmo deve dar o peso total de todos os contentores.

b) Considera a mesma situação, mas supondo que não é possível saber previamente o número total de contentores a carregar no navio. Constrói um algoritmo que leia o valor do peso de cada contentor e pergunte sucessivamente "Mais contentores (S/N)?".

No final, o algoritmo deve indicar o número e o peso total de todos os contentores lidos.

5 **Constrói um algoritmo que peça ao utilizador para adivinhar um número secreto.**

O número secreto é 724.

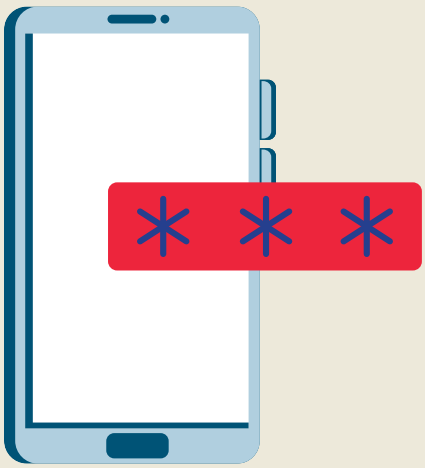
Sempre que o utilizador introduz um novo valor sem acertar, o algoritmo deve:

– escrever "Maior", se o número for inferior a 724;

– escrever "Menor", se o número for superior a 724;

– contar o número de tentativas.

Quando o utilizar introduzir o número correto, o algoritmo deve escrever "Acertaste" e informar quantas tentativas foram necessárias.



6 **Constrói um algoritmo que leia números até que seja introduzido o número 999. No final, o algoritmo deve indicar:**

– o total de números lidos;

– quantos números eram positivos;

– a percentagem de números positivos introduzidos.

7 **Constrói um algoritmo que leia números até que seja introduzido o número zero. No final, o algoritmo deve indicar:**

– a soma dos números pares lidos;

– a quantidade de números ímpares lidos;

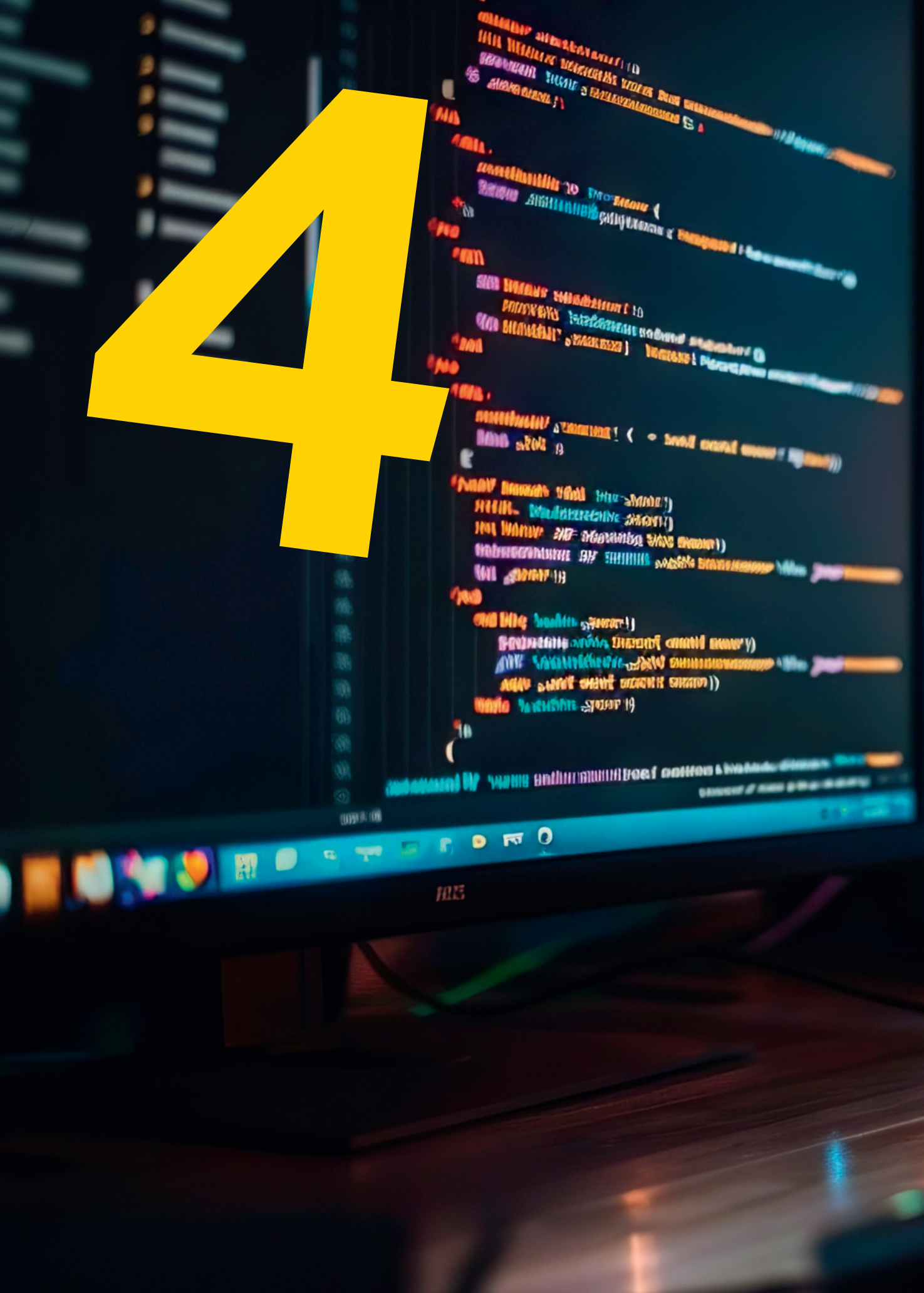
– o maior número lido;

– a média de todos os números lidos.

8 **Constrói um algoritmo que ajude no controlo das despesas numa ida ao mercado.**

O algoritmo deve permitir que o utilizador vá registando o valor de cada compra e ir calculando o total gasto.

Quando o valor das compras ultrapassar 1000$, o algoritmo deve emitir uma mensagem de alerta e terminar.


4

Ferramentas e aplicações

4 Ferramentas e aplicações

No final deste capítulo, deverás ser capaz de:

- Desenvolver programas simples e funcionais que integrem diferentes tipos de estruturas de forma coerente.
- Adotar boas práticas de programação, nomeadamente a clareza do código, a indentação e a estruturação adequada.
- Demonstrar autonomia, responsabilidade e espírito colaborativo no desenvolvimento de projetos.

4.1. Ferramenta para criar fluxogramas

Ferramenta Flowgorithm

Há várias ferramentas que poderás usar para criares os teus algoritmos num computador. As ferramentas de programação visual utilizam símbolos ou ícones na construção de programas, ao invés de utilizarem exclusivamente texto, facilitando a construção de programas computacionais de modo intuitivo (por exemplo, Dia, Draw.io, ToonTalk, Scratch, Flowgorithm, VisuAlg, entre outras).

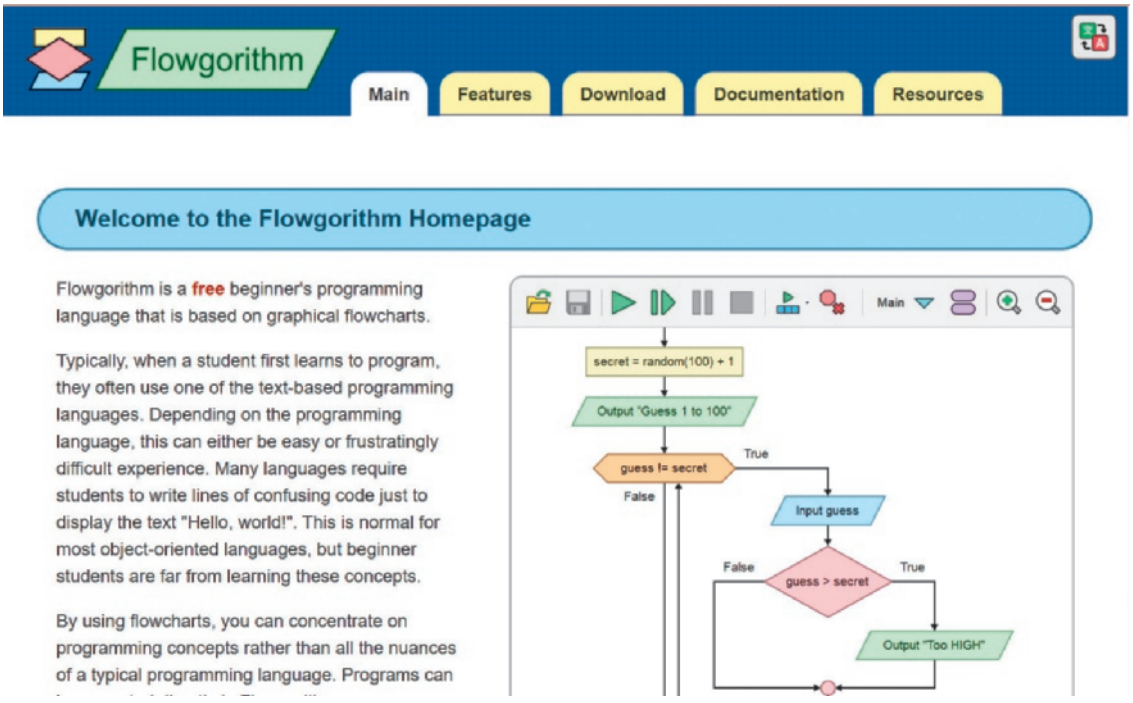
O **Flowgorithm** é uma ferramenta muito simples para construir algoritmos em fluxograma. É ideal para programadores principiantes.

O nome Flowgorithm deriva da combinação das palavras:

- *Flowchart* (fluxograma): refere-se à representação gráfica de algoritmos utilizando símbolos-padrão de fluxogramas;
- *Algorithm* (algoritmo): um procedimento passo a passo para resolver um problema ou realizar uma tarefa.

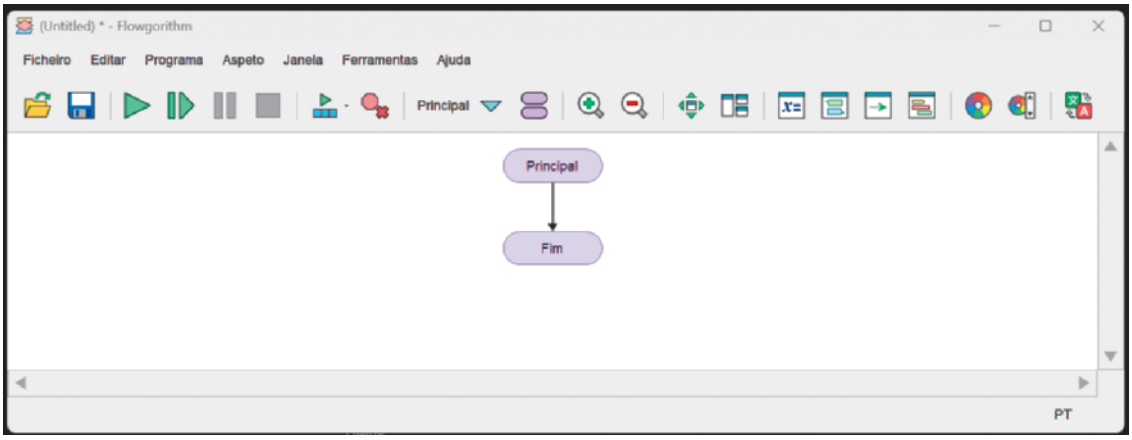
Instalar o Flowgorithm

- Acede ao *site* oficial: http://www.flowgorithm.org/
- Faz o *download* da versão mais simples do programa (*executable only*) e instala como habitualmente, seguindo as instruções apresentadas.

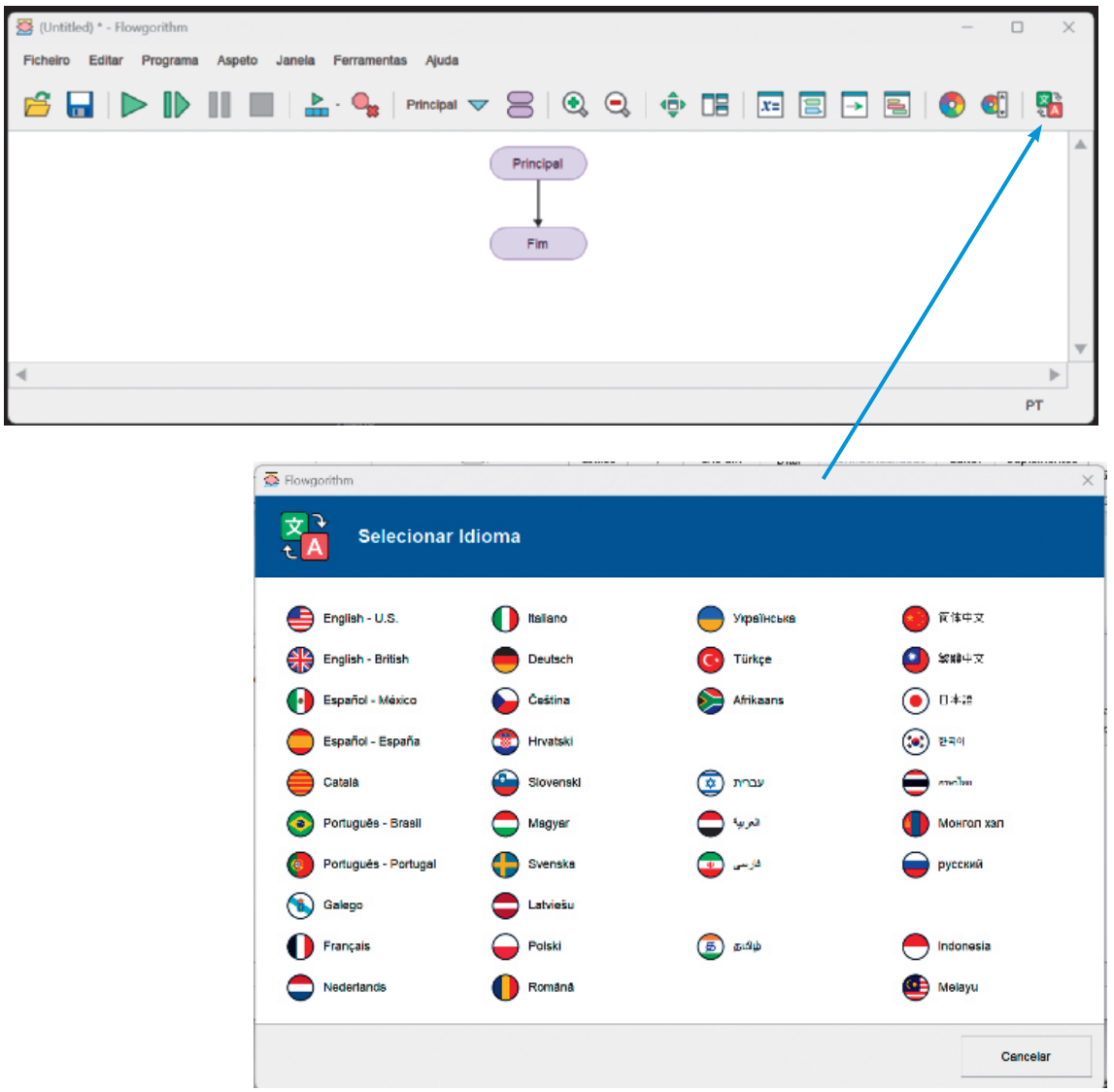


Explorar o ambiente de trabalho do Flowgorithm

Depois da instalação, quando acedes ao Flowgorithm, visualizas esta janela:



Deves começar por selecionar o idioma "Português", para que seja mais fácil interagir com o programa.

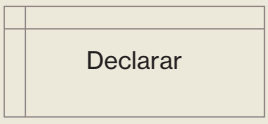
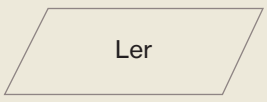
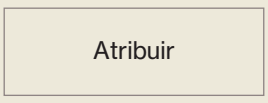
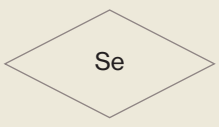
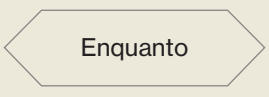
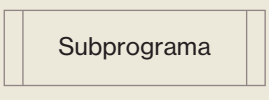


O ambiente de trabalho inclui vários menus e funcionalidades:

- **Ficheiro (File):** Permite gerir ficheiros (criar novos algoritmos, abrir algoritmos existentes, gravar alterações e imprimir).
- **Editar (Edit):** Contém as funções-padrão (cortar, copiar, colar, apagar, desfazer e refazer).
- **Programa (Program):** Permite executar um algoritmo (executar e executar passo a passo para verificar erros) e permite adicionar subprogramas.
- **Aspeto (Apperance):** Permite configurar as definições de visualização, incluindo a alteração do estilo, cores dos blocos e ajuste do *zoom*.

- **Janela (Window):** Permite adicionar janelas que apoiam a monitorização do algoritmo, nomeadamente, mostrar o algoritmo escrito em pseudocódigo ou em linguagens como Python, Java ou C#.
- **Ferramentas (Tools):** Permite editar estilos, temas de cores e exportar imagens.
- **Ajuda (Help):** Permite aceder a documentação e *sites* com informação adicional.

A construção de um algoritmo é feita de um modo muito simples e intuitivo. Necessitas de conhecer a função de cada um dos símbolos gráficos e respeitar a sintaxe nas instruções.

	Início (é usada a palavra Principal) e fim do algoritmo
	Declaração de variáveis
	Leitura de dados
	Escrita de dados
	Atribuição de valores
	Estrutura de controlo
	Ciclos de repetição
	Subprograma ou sub-rotina

Operadores no Flowgorithm

Para introduzir expressões aritméticas e expressões lógicas, o Flowgorithm permite usar os símbolos de diferentes tipos de linguagens.

Operador	C	BASIC	Matemática (Unicode)
Igualdade	==	=	=
Desigualdade	!=	<>	≠
Menor ou igual	<=	<=	≤
Maior ou igual	>=	>=	≥
Não (lógico)	!	not	¬
E (lógico)	&&	and	∧
Ou (lógico)	||	or	∨
Multiplicação	*	*	×
Divisão	/	/	÷
Potenciação		^	
Módulo	%	mod	

Para unir duas ou mais cadeias de *caracteres* é usado o símbolo **&**.

Guia rápido para começares a usar o Flowgorithm

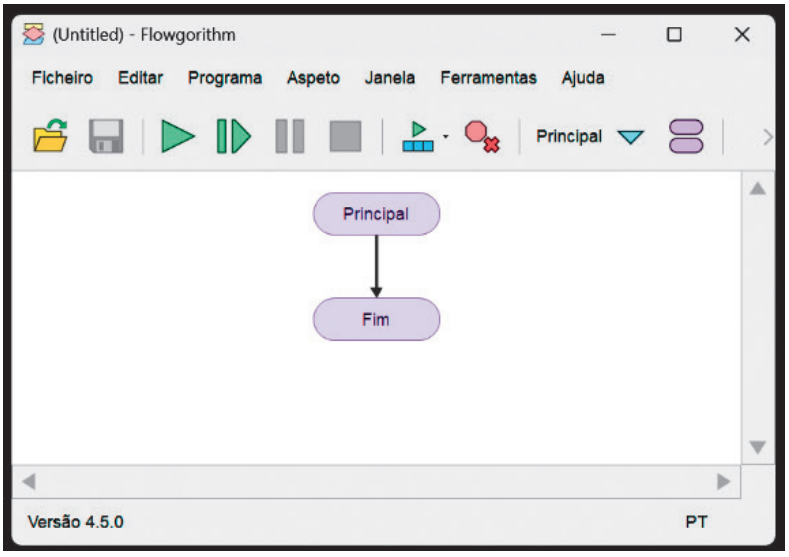
O modo mais rápido de aprenderes a usar o Flowgorithm é seguires os passos indicados para criares um algoritmo em fluxograma que resolva a tarefa abaixo. No final, poderás testar o teu programa.

> TAREFA: Criar um algoritmo que solicite o ano de nascimento de uma pessoa e o ano atual e que depois apresente a idade dessa pessoa.

Passo 1. Criação e início do fluxograma

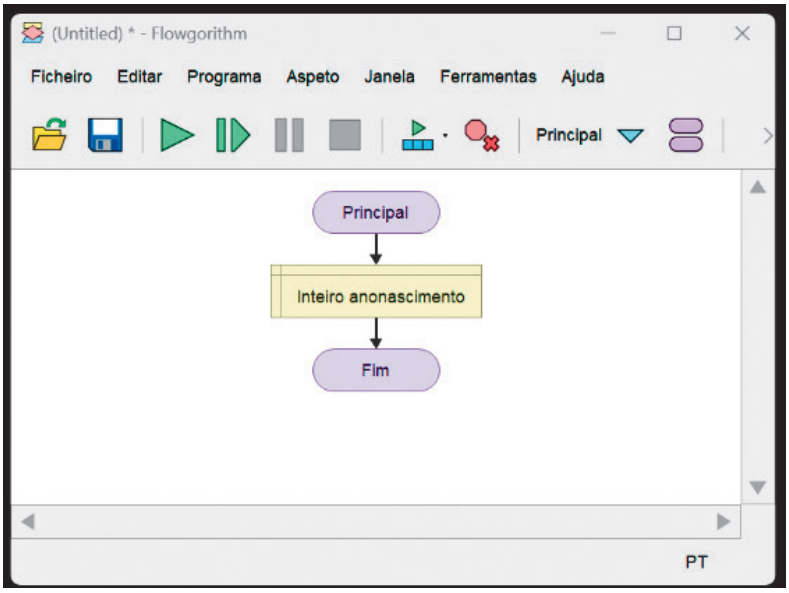
Ao abrires o programa **Flowgorithm.exe** é criado um novo documento onde estão dispostos os símbolos de início e de fim do fluxograma.

Para adicionar novos símbolos é sempre necessário clicar sobre uma seta existente.

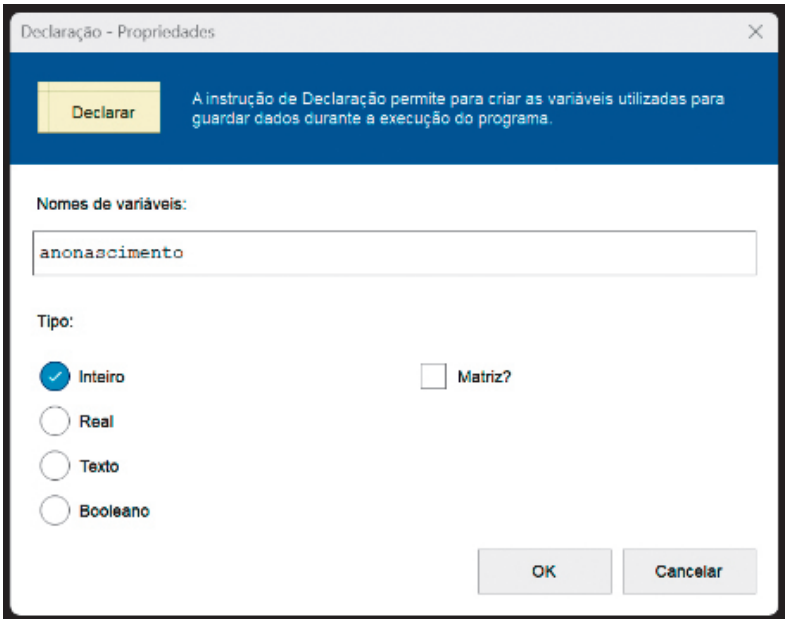


Passo 2. Criar uma variável para o ano de nascimento

Clica sobre a seta e escolhe o símbolo **Declarar** para criares a tua variável.

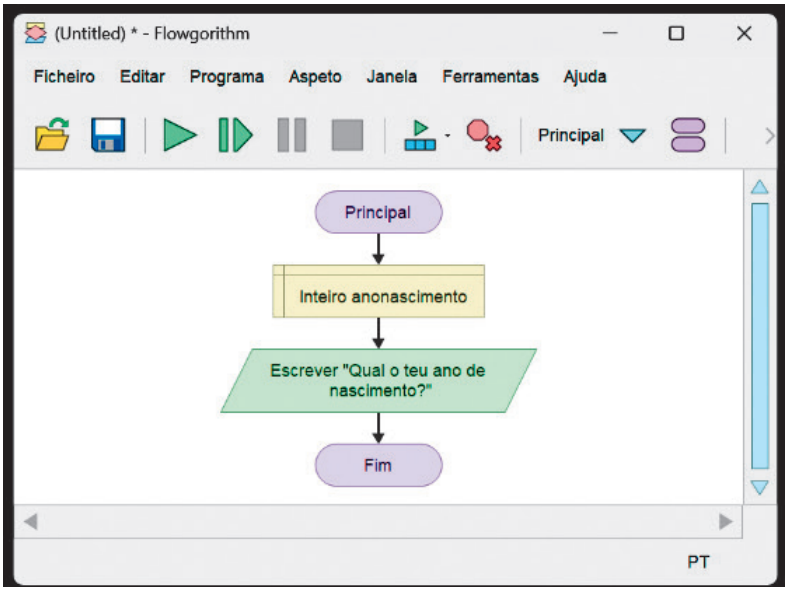


Escolhe o nome da variável e o tipo (esta variável irá armazenar o ano de nascimento e deve ser um valor inteiro).

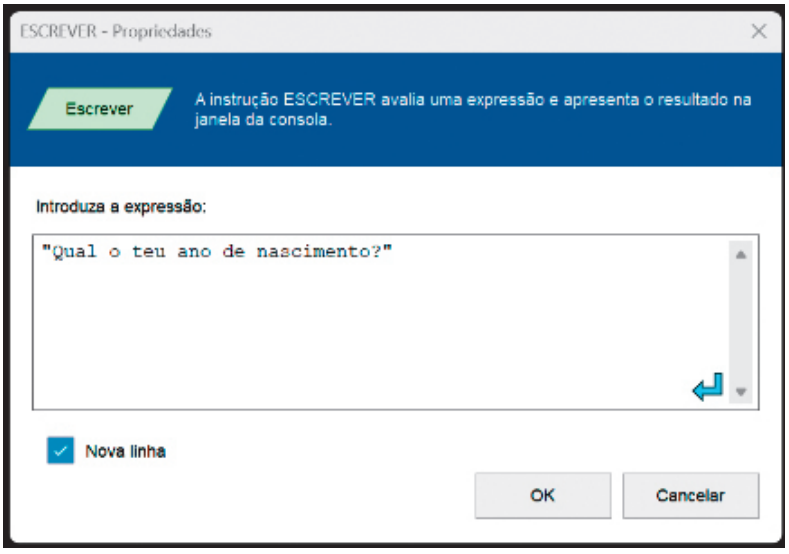


Passo 3. Criar uma mensagem a solicitar o ano de nascimento

Clica sobre a seta abaixo da declaração de variável e escolhe o símbolo **Escrever** para introduzires a mensagem.

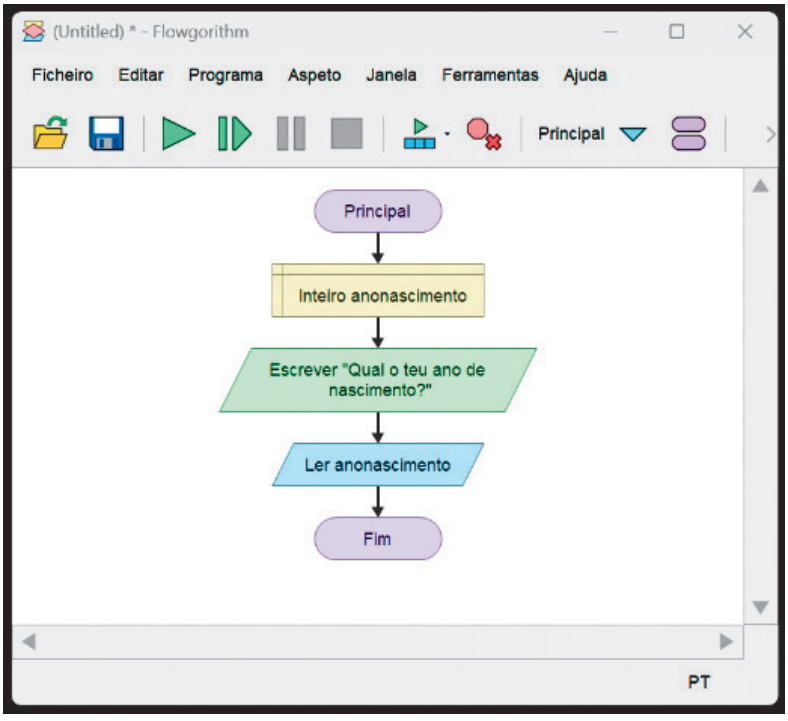


A mensagem que irá aparecer no ecrã deve ser escrita entre aspas.



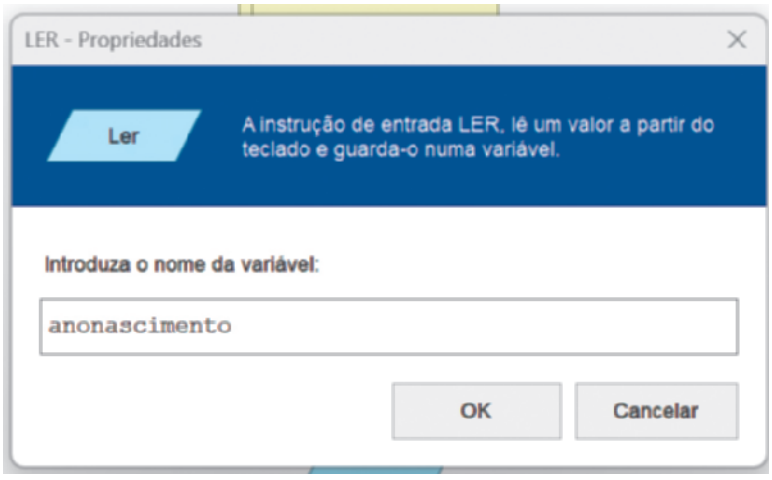
Passo 4. Ler e armazenar o ano de nascimento

Clica sobre a seta abaixo da mensagem e escolhe o símbolo **Ler** para introduzires a instrução que vai permitir ler o valor introduzido pelo utilizador e guardá-lo na variável.

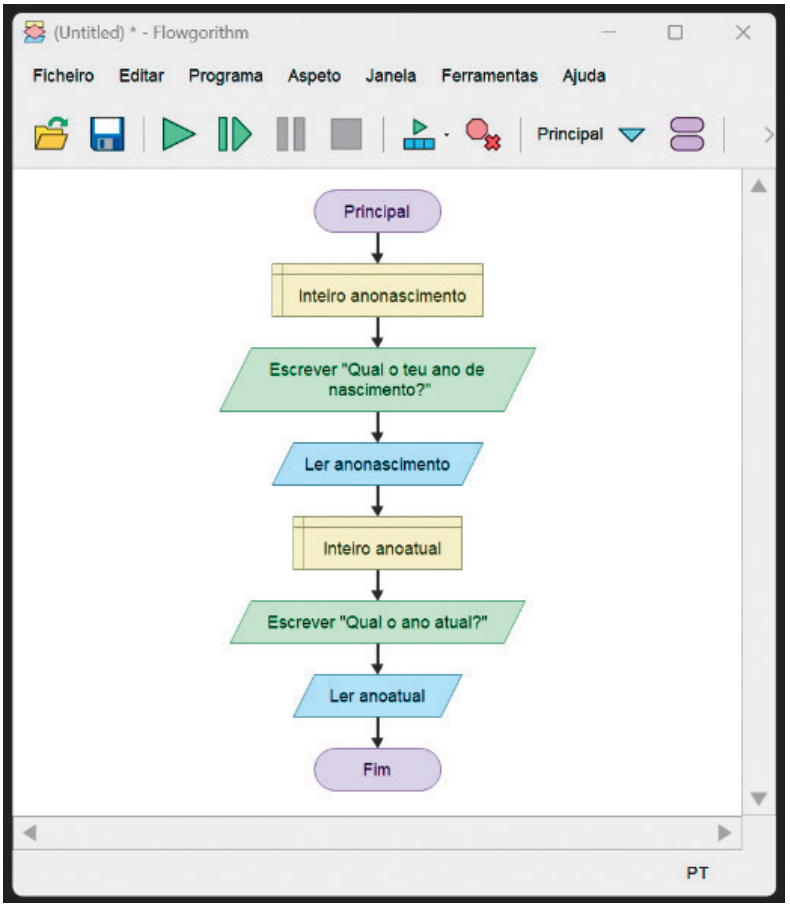


Introduz o nome da variável onde queres guardar o dado inserido pelo utilizador.

Repara que antes de uma instrução **Ler** ou **Atribuir** é necessário declarar a variável a utilizar (fica atento para escreveres corretamente os nomes que definiste).

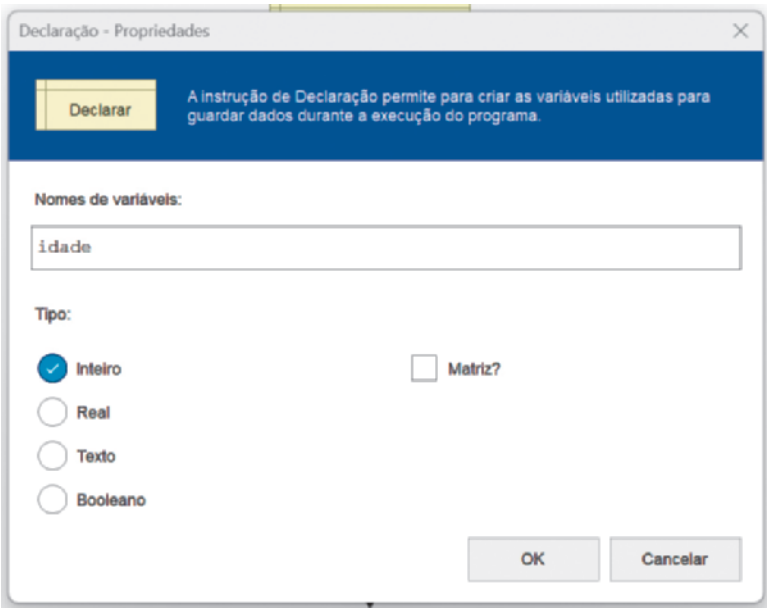


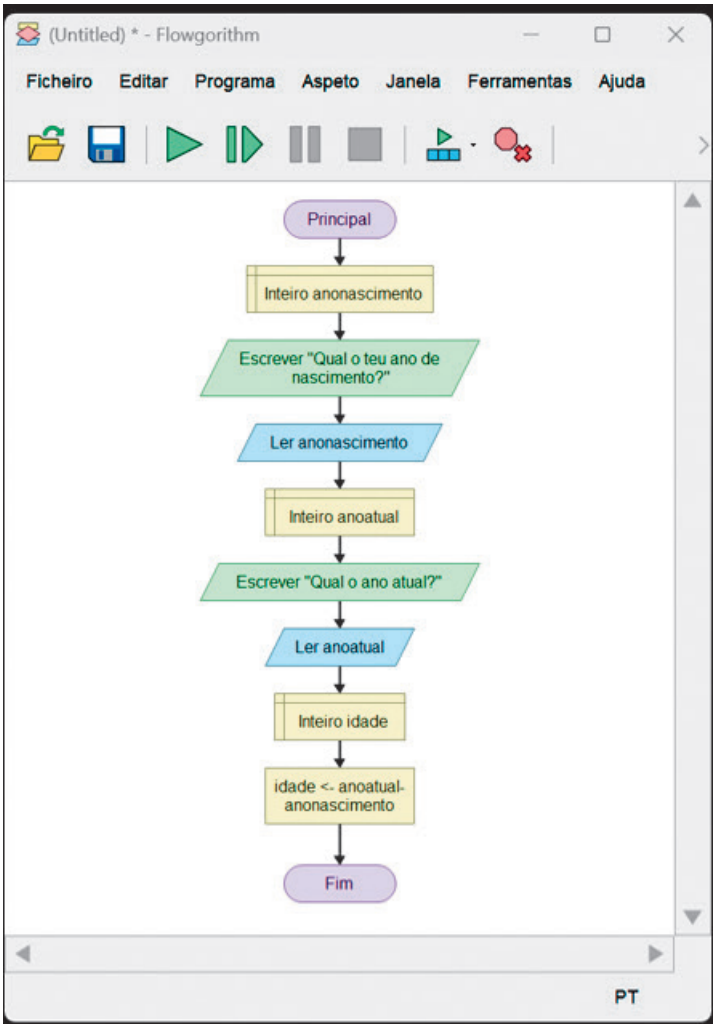
Passo 5. Repetir os passos 2, 3 e 4 para criar, solicitar, ler e armazenar o ano atual



Passo 6. Criar uma variável para a idade

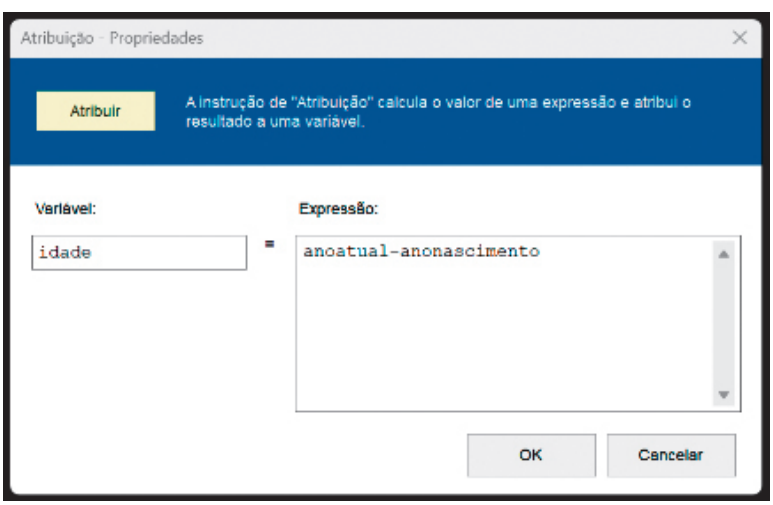
Cria uma terceira variável do mesmo modo que criaste as duas primeiras. Esta variável será usada para calcular a idade.





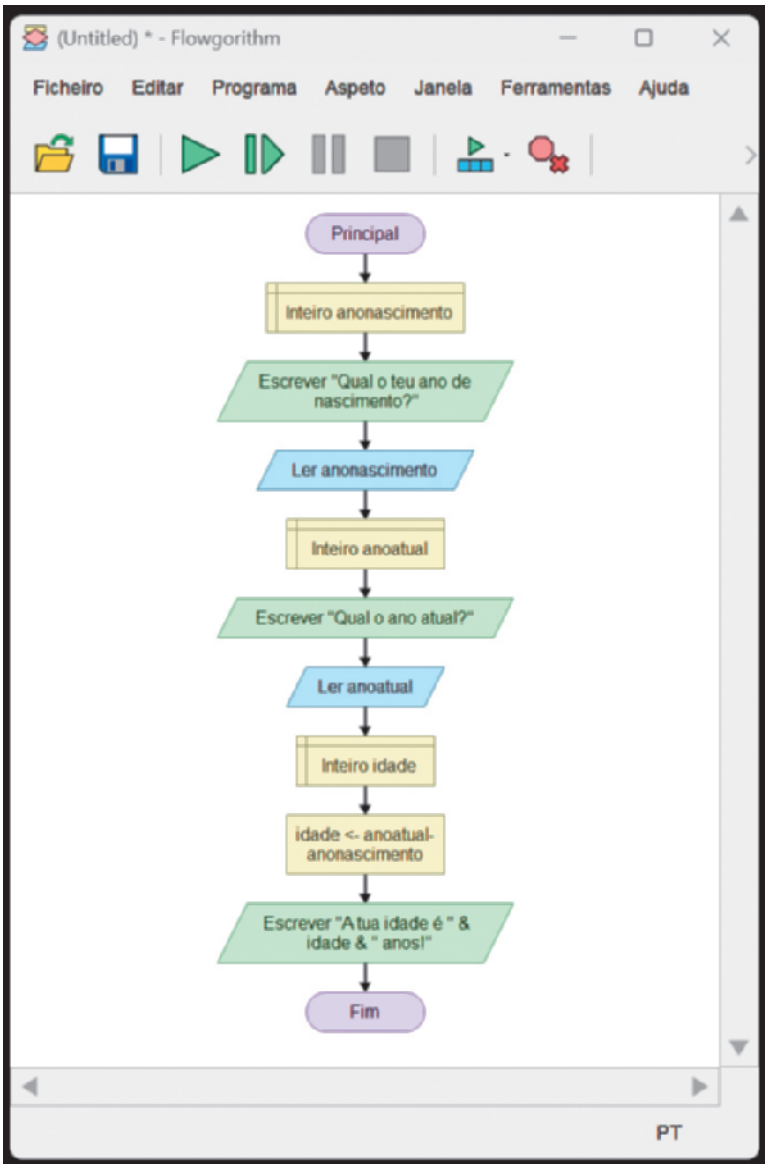
Passo 7. Cálculo da idade

Clica sobre a seta seguinte e escolhe o símbolo **Atribuir** para definires como é calculado o valor da variável idade.

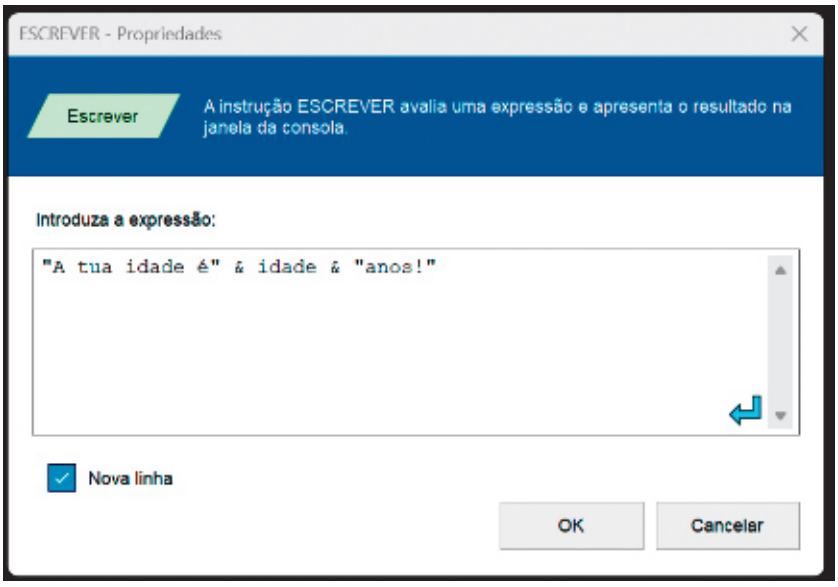


Passo 8. Apresentar a idade calculada

Clica sobre a seta seguinte e escolhe o símbolo **Escrever** para definires a mensagem final.



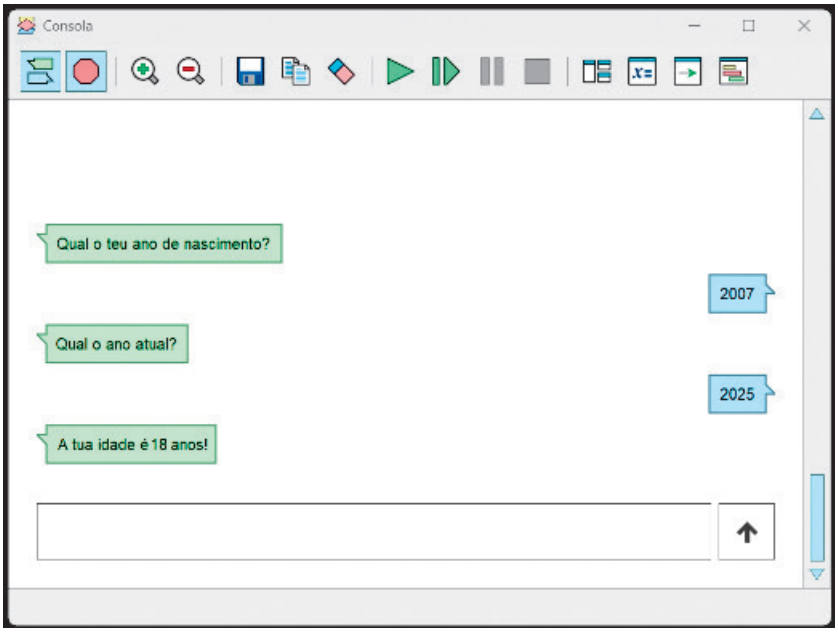
Usa o carácter **&** para separares o texto da variável.



Passo 9. Testar o programa

No menu **Programa**, escolhe a opção **Executar** e testa o teu programa.

É aberta uma consola onde o programa irá interagir com o utilizador. Testa o teu algoritmo com diferentes valores (ano de nascimento e ano atual).



Testa os teus conhecimentos

1 **Para cada uma das seguintes tarefas, constrói no Flowgorith um algoritmo que as permita executar (estruturas sequenciais).**

Tarefa A: Escrever no ecrã: "Cabo Verde é um país lindo!".

Tarefa B: Perguntar o nome do utilizador e escrever "Olá [nome]!", (lembra-te que para unir várias sequências de *caracteres* tens de usar o símbolo **&**).

Tarefa C: Ler dois números e mostrar a sua soma no ecrã.

Tarefa D: Ler dois números e mostrar a sua média no ecrã.

2 **Para cada uma das seguintes tarefas, constrói no Flowgorith um algoritmo que as permita executar (estruturas condicionais).**

Tarefa A: Ler um número e escrever no ecrã se o número é par ou ímpar.

Tarefa B: Ler dois números e escrever no ecrã o maior dos dois.

Tarefa C: Ler uma classificação (0 a 20) e escrever no ecrã se o aluno está ou não aprovado, de acordo com:

Se nota ≥ 10 "Aprovado"

Senão "Reprovado"

3 **Para cada uma das seguintes tarefas, constrói no Flowgorith um algoritmo que as permita executar (estruturas de repetição).**

Tarefa A: Escrever no ecrã os números de 1 até 10.

Tarefa B: Ler um número *N* e apresentar no ecrã o resultado da soma de todos os números de 1 até *N*.

Tarefa C: Ler um número *N* e apresentar no ecrã a sua tabuada de 1 a 10.

4 **Constrói no Flowgorith um algoritmo que permita resolver cada um dos seguintes desafios.**

Desafio A: Ler cinco números e escrever no ecrã o maior.

O algoritmo tem de comparar os cinco números lidos para identificar qual o maior.

Desafio B: Ler as classificações de cinco alunos e escrever no ecrã a média das suas classificações e qual o aluno que teve a classificação mais alta.

O algoritmo tem de solicitar que cada um dos alunos introduza o seu nome e a sua classificação.

4.2. Ferramenta para criar algoritmos em pseudocódigo

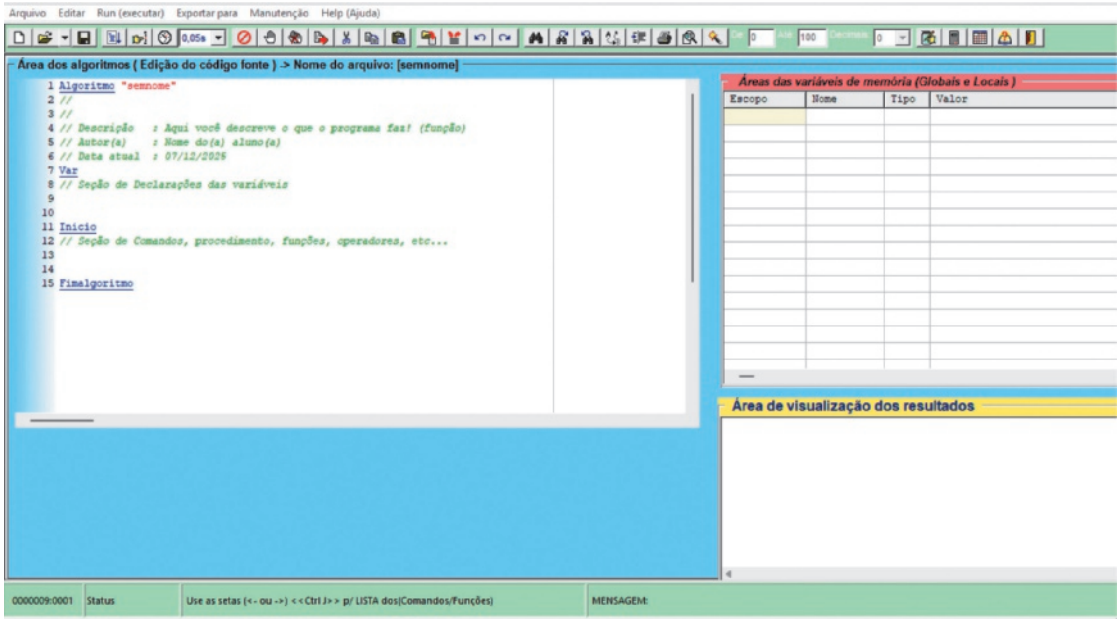
Ferramenta VisuAlg

O **VisuAlg** é um *software* que permite criar, editar e executar algoritmos escritos em português (Portugal). Podes instalar o VisuAlg no teu computador ou usar a versão *online* para construíres os teus algoritmos.

Instalar o VisuAlg

- Acede ao *site*: https://sourceforge.net/projects/visualg30/
- Faz o *download* da versão **VisuAlg 3.0**
- Instala como habitualmente, seguindo as instruções apresentadas.

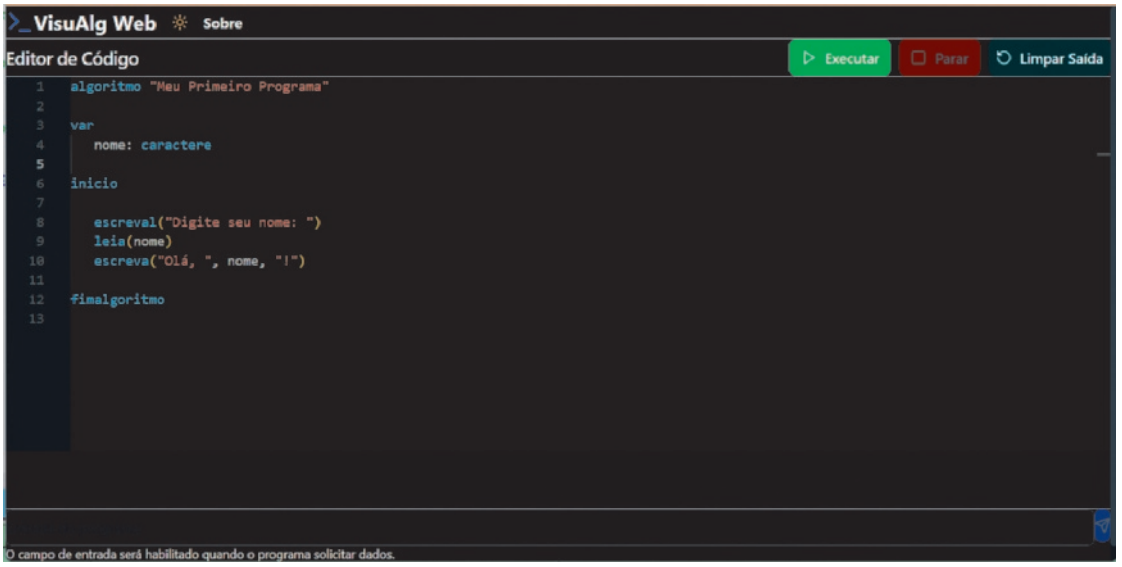
VisuAlg – Versão instalada no computador (*offline*)



Usar *online* o VisuAlg

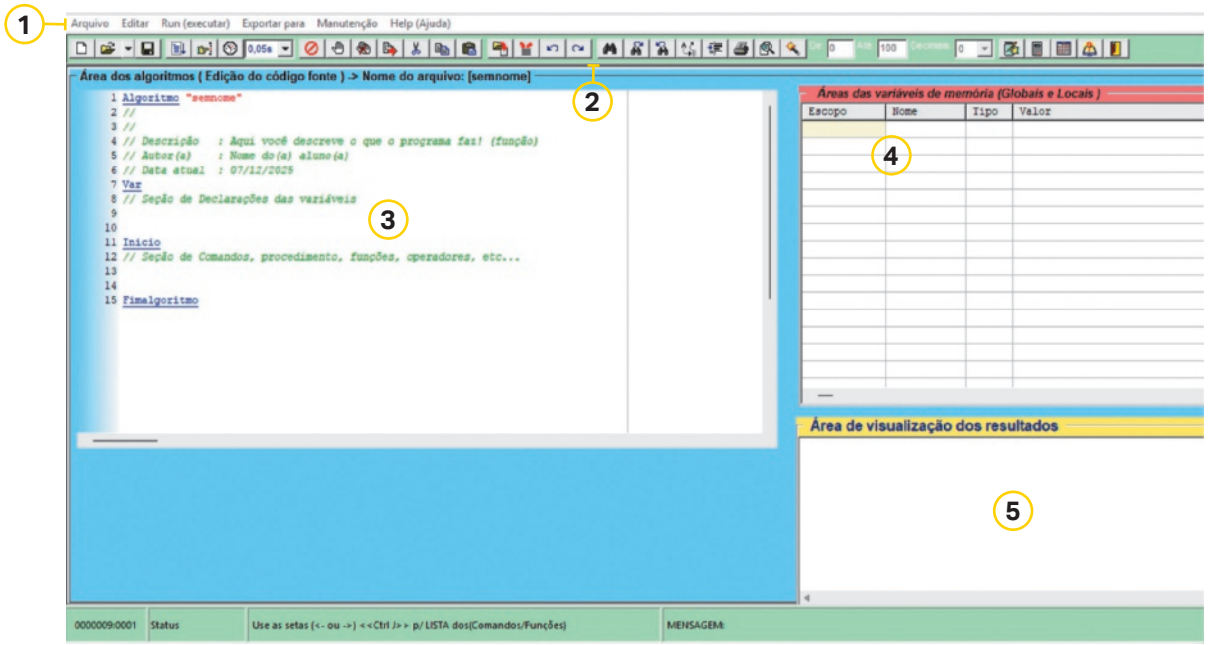
- Acede ao *site* da ferramenta em https://visualg.com.br/ e podes iniciar a construção dos teus algoritmos.

VisuAlg (*online*)



Explorar o ambiente de trabalho do VisuAlg

O ambiente de trabalho do VisuAlg é simples e intuitivo.



1. **Barra de menus** – Inclui os vários menus e funcionalidades:
   - **Arquivo:** Permite gerir ficheiros (criar, abrir, gravar e imprimir).
   - **Editar:** Contém funções-padrão (cortar, copiar, colar, apagar, desfazer, refazer e localizar).
   - **Run (executar):** Permite executar um algoritmo (executar e executar passo a passo).
   - **Exportar para:** Converte o algoritmo escrito noutras linguagens: Pascal, C/C++ e Java.
   - **Manutenção:** Permite configurar o ambiente de trabalho e aceder à calculadora e ao calendário.
   - **Help (Ajuda):** Permite aceder a informação adicional sobre o *software*.
2. **Barra de ferramentas** – Contém botões de acesso rápido às funções mais usadas dos menus.
3. **Área de edição** – É a zona principal da ferramenta, na qual é escrito o algoritmo.
4. **Área das variáveis de memória** – Janela onde vai aparecer a listagem das variáveis criadas.
5. **Área da visualização dos resultados** – Nesta janela aparecem as mensagens, os pedidos de dados e os resultados da execução do algoritmo.

Estrutura de um algoritmo no VisuAlg

Ao abrires o VisuAlg, na área de edição podes construir um novo algoritmo. Qualquer algoritmo no VisuAlg tem este formato:

```
1 Algoritmo "semnome"
2 //
3 //
4 // Descrição   : Aqui você descreve o que o programa faz! (função)
5 // Autor(a)    : Nome do(a) aluno(a)
6 // Data atual  : 01/02/2026
7 Var
8 // Seção de Declarações das variáveis
9
0
1 Inicio
2 // Seção de Comandos, procedimento, funções, operadores, etc...
3
4
5 Fimalgoritmo
```

As palavras reservadas estão claramente identificadas (neste caso a azul e sublinhadas).

A palavra **Algoritmo** identifica o programa e deves editar o texto **"semnome"** para introduzires o nome do teu algoritmo.

As linhas iniciadas por "//" são ignoradas pelo VisuAlg e permitem introduzir comentários e explicações que ajudem qualquer pessoa a interpretar o algoritmo. O próprio *software* incentiva a que te identifiques como autor do algoritmo e que insiras uma breve descrição da sua função.

A seção **Var** é usada para a declaração das variáveis que vão ser usadas no algoritmo. Deves indicar o nome e o tipo de cada variável. Os nomes das variáveis devem começar por uma letra e têm até um limite de 30 *caracteres*. O VisuAlg prevê quatro tipos de dados: inteiro, real, *caractere* e lógico.

Deves construir o teu algoritmo entre as linhas **Inicio** e **Fimalgoritmo**.

☑ Not@ que:

1. O VisuAlg apenas permite um comando por linha.
2. É permitida a inclusão de comentários: qualquer texto precedido de "**//**" é ignorado até se atingir o final da sua linha.
3. Todas as palavras-chave do VisuAlg foram implementadas sem acentos, cedilhas e caracteres especiais.
4. O VisuAlg não distingue maiúsculas e minúsculas no reconhecimento de palavras-chave e nomes de variáveis.

Instruções no VisuAlg

- **Declarar variáveis**

Para criares variáveis deves escrever o nome da variável seguido por **':'** e depois pelo tipo de variável.

```
num1: inteiro
x: real
nome: caractere
numero: real
```

- **Atribuir valores a variáveis**

Para atribuíres valores a uma variável deves usar os símbolos **'<-'** (menor e traço).

```
num1 <– num1 + 5
nome <– "Mindelo"
```

- **Ler e escrever dados**

Para as instruções de leitura e escrita deves usar as palavras '**leia**' e '**escreva**'.

```
leia (num1)
escreva (num1)
escreva ("Olá!")
```

Para várias expressões podes usar uma vírgula ou o símbolo '**+**'. Por exemplo, a instrução:

```
escreva ("Cabo " + "Verde")
```

irá aparecer no ecrã: Cabo Verde

- **Estruturas condicionais**

Para introduzires estruturas condicionais deves usar as palavras: '**se**', '**entao**', '**senao**' e '**fimse**'.

```
se media >= 10 entao
  escreva ("Aprovado")
senao
  escreva ("Reprovado")
fimse
```

Para introduzires estruturas de decisão múltipla deves usar as palavras: '**escolha**', '**caso**' e '**fimescolha**'.

```
leia (num)
escolha num
caso 1
   escreva ("Janeiro")
caso 2
   escreva ("Fevereiro")
caso 3
   escreva ("Março")
caso 4
   escreva ("Abril")
caso 5
   escreva ("Maio")
fimescolha
```

- **Estruturas de repetição**

Para introduzires estruturas de repetição do tipo **PARA**, deves usar as palavras: '**para**', '**ate**', '**faca**' e '**fimpara**'.

```
para i de 1 ate 10 faca
  escreva (i)
fimpara
```

Para introduzires estruturas de repetição do tipo **ENQUANTO**, deves usar as palavras: '**enquanto**', '**faca**' e '**fimenquanto**'.

```
enquanto x < 10 faca
  x <– x + 1
  escreva (x)
fimenquanto
```

Para introduzires estruturas de repetição do tipo **REPETIR...ATÉ**, deves usar as palavras: '**repita**' e '**ate**'.

```
repita
  ler (n)
ate n > 0
```

- **Operadores aritméticos e relacionais**

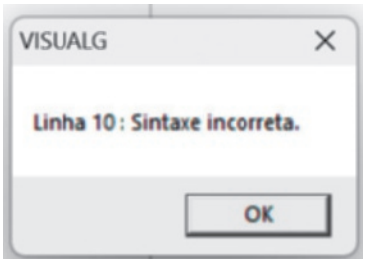
Para introduzir expressões aritméticas e expressões lógicas deves usar os seguintes símbolos:

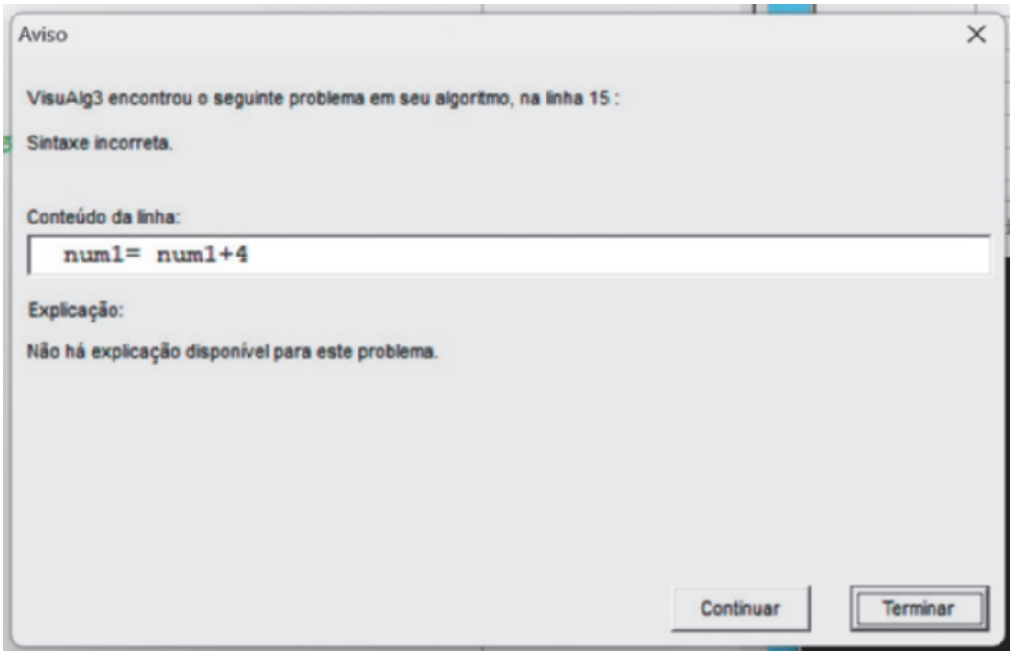
Operador	Símbolo a usar no VisuAlg
Igualdade	=
Desigualdade	<>
Menor ou igual	<=
Maior ou igual	>=
Não (lógico)	nao
E (lógico)	e
Ou (lógico)	ou
Multiplicação	*
Potenciação	^
Divisão	/
Divisão inteira	\
Módulo	MOD ou %

☑ Not@ que:

Podes ir testando o teu algoritmo à medida que vai sendo construído. Basta carregar no botão executar e fazer correr o algoritmo, mesmo que ainda esteja incompleto.

Sempre que alguma instrução não está escrita de um modo correto, o VisuAlg interrompe a execução do algoritmo e emite um alerta de erro, identificando a linha onde existe um erro na sintaxe.





Guia rápido para começar a usar o VisuAlg

O modo mais rápido de começares a aprender a usar o VisuAlg é tentares construir e testar um algoritmo.

Constrói o algoritmo "**Soma**", que permita calcular a soma de dois números.

```
Algoritmo "Soma"
// soma de dois numeros
Var
n1: inteiro
n2: inteiro
n3: inteiro
//
Inicio
Escreva ("Introduz um número. ")
Leia (n1)
Escreva ("Introduz outro número. ")
Leia (n2)
n3 <– n1 + n2
Escreva ("A soma dos números que introduziste é ", n3)
Fimalgoritmo
```

O primeiro passo é definir o nome do algoritmo a construir e guardar, para que não percas o teu trabalho.

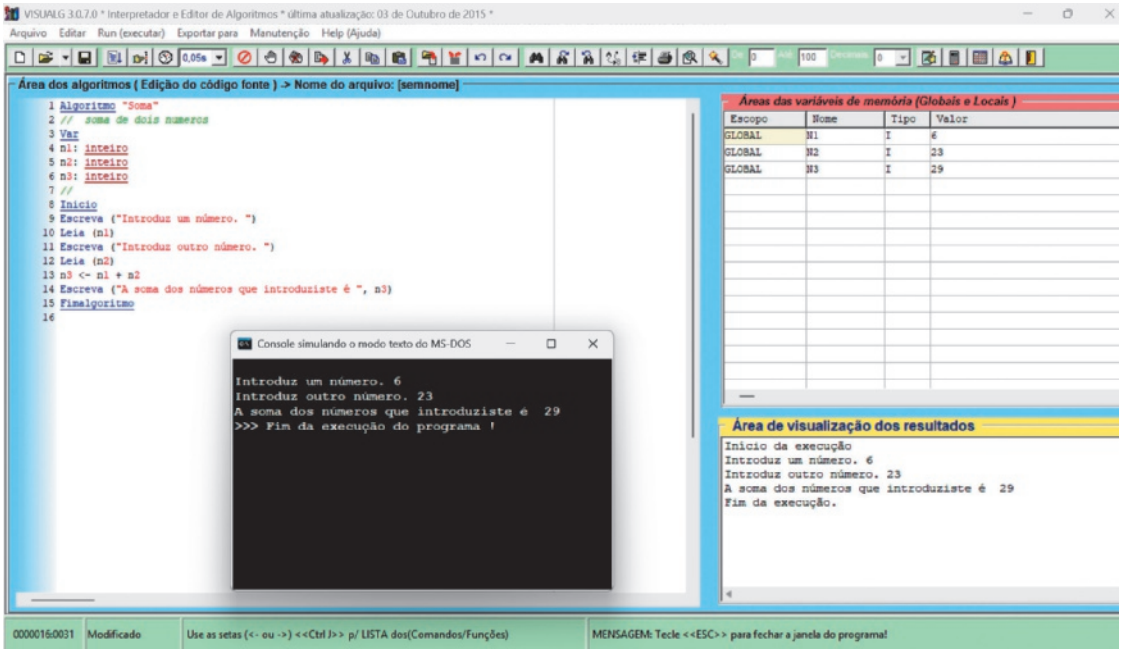
De seguida, introduz as instruções dos vários passos do algoritmo.

Deves obter algo semelhante a:

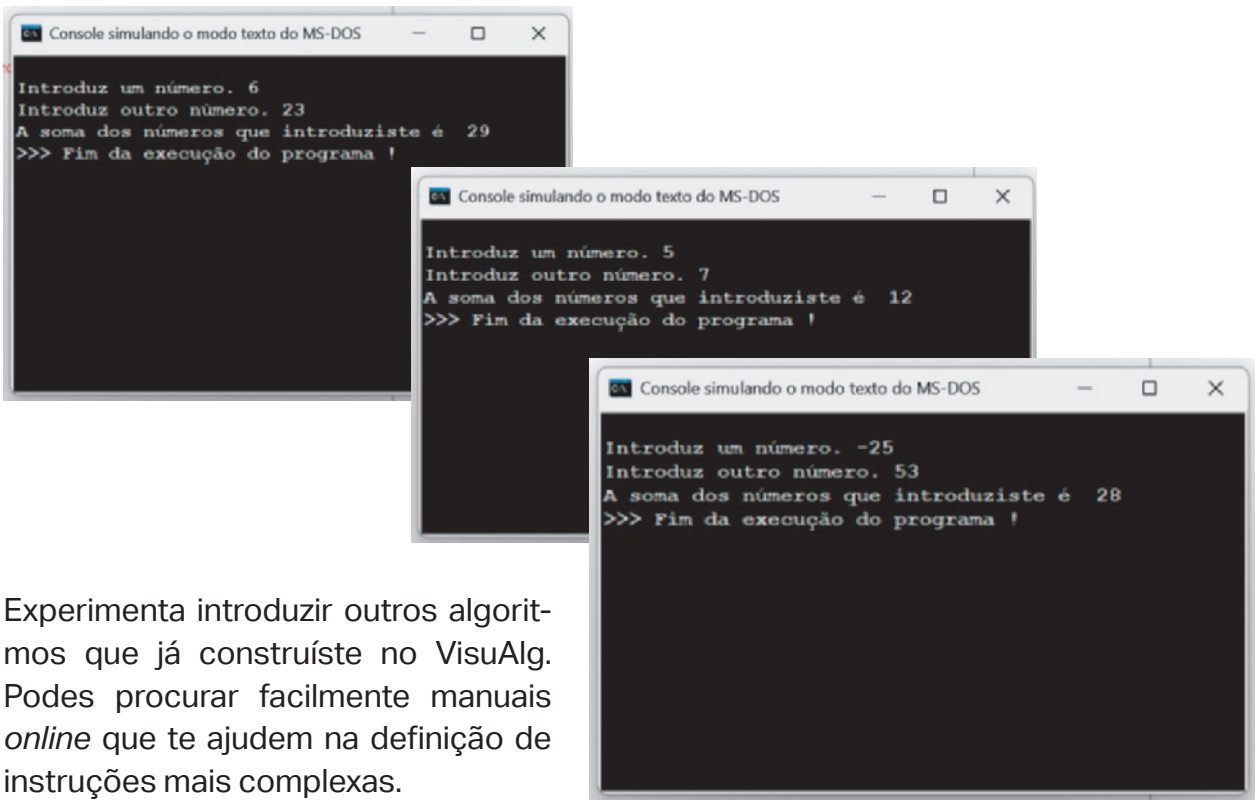
```
1 Algoritmo "Soma"
2 //  soma de dois numeros
3 Var
4 n1: inteiro
5 n2: inteiro
6 n3: inteiro
7 //
8 Inicio
9 Escreva ("Introduz um número. ")
10 Leia (n1)
11 Escreva ("Introduz outro número. ")
12 Leia (n2)
13 n3 <- n1 + n2
14 Escreva ("A soma dos números que introduziste é ", n3)
15 Fimalgoritmo
16
```

O próximo passo é testares se o algoritmo está correto.

Executa o algoritmo clicando no botão '**Executar**' e introduz os valores que são solicitados pelo algoritmo.



Deves realizar vários testes com valores diferentes.



Experimenta introduzir outros algoritmos que já construíste no VisuAlg. Podes procurar facilmente manuais *online* que te ajudem na definição de instruções mais complexas.

Testa os teus conhecimentos

1 **Para cada uma das seguintes tarefas, constrói no VisuAlg um algoritmo que as permita executar estruturas sequenciais.**

Tarefa A: Escrever no ecrã: "Cabo Verde tem 10 ilhas e 8 ilhéus.".

Tarefa B: Perguntar o nome do utilizador e escrever: 'O nome [nome] é muito bonito!'.

Tarefa C: Ler dois números e mostrar o seu produto no ecrã.

Tarefa D: Ler três números e mostrar a sua média no ecrã.

2 **Para cada uma das seguintes tarefas, constrói no VisuAlg um algoritmo que as permita executar estruturas condicionais.**

Tarefa A: Ler um número e escrever no ecrã se o número é par ou ímpar.

Tarefa B: Ler dois números e escrever no ecrã o maior dos dois.

Tarefa C: Ler uma classificação (0 a 20) e escrever no ecrã se o aluno está ou não aprovado, de acordo com:

Se nota $\geqslant$ 10 "Aprovado"

Senão "Reprovado"

3 **Para cada uma das seguintes tarefas, constrói no VisuAlg um algoritmo que as permita executar estruturas de repetição.**

Tarefa A: Escrever no ecrã os números de 1 até 20.

Tarefa B: Ler um número *N* e apresentar no ecrã o resultado da soma de todos os números de 1 até *N*.

Tarefa C: Ler um número *N* e apresentar no ecrã a sua tabuada de 1 a 10.

4 **Constrói no VisuAlg um algoritmo que permita resolver o seguinte desafio.**

Desafio: Ler um número não negativo e escrever no ecrã o seu fatorial.

O fatorial de um número *N* é definido pelo produto de todos os números de 1 até *N*:

$$N! = 1 \times 2 \times 3 \times 4 \times 5 \times \ldots \times (N - 1) \times N$$

O fatorial do número zero é 1, por definição:

$$0! = 1$$

Caso seja introduzido um número negativo, o algoritmo deve apresentar uma mensagem de erro e solicitar um novo número.

4.3. Aplicações práticas

Desenvolvimento de um programa de computador

Participar em projetos que envolvam a construção de um programa de computador é fundamental para desenvolver o teu pensamento computacional e a tua capacidade de resolução de problemas.

Ao trabalhar em projetos que envolvam situações do dia a dia, é mais fácil estabelecer uma ligação direta entre conceitos abstratos e aplicações práticas.

Este tipo de projetos promove competências como o raciocínio lógico, a organização de ideias, o trabalho colaborativo, a comunicação e a capacidade de analisar e melhorar soluções.

Relembra as várias etapas de desenvolvimento de um programa de computador.

Manual Interativo

Vídeos

Criar uma história no Scratch

Criar um jogo no Scratch, controlado pelo micro:bit

Criar o jogo das pedras preciosas no Scratch

Ciclo de desenvolvimento de um programa de computador

1

Analisar o problema
Compreender o que o programa deve fazer e qual deve ser a saída. Ter uma ideia clara de que dados (entrada) são fornecidos. Perceber qual a relação entre a entrada e a saída desejada.

A primeira etapa no desenvolvimento de um projeto é a escolha e definição clara do problema. Um bom problema deve ser relevante e compreensível. É essencial compreender profundamente o problema antes de tentares resolvê-lo. Isso implica que respondas a questões como:

- Qual é exatamente o problema a resolver?
- Quem são os utilizadores?
- Em que contextos vai ser aplicado?
- Que dificuldades existem atualmente?

Uma definição imprecisa do problema pode conduzir a soluções inadequadas ou incompletas.

Depois de identificares o problema, é necessário que o analises de forma sistemática. A análise consiste em decompor o problema em partes mais pequenas e identificar os seus elementos essenciais. Nesta fase, deves definir:

- **entradas**: os dados e informações que o algoritmo vai receber;
- **processos**: operações e decisões a realizar;
- **saídas**: resultados que o algoritmo deve produzir.

Este processo de modelação do problema permite reduzir a complexidade e facilita a criação de uma solução lógica. É também neste momento que deves definir regras, restrições e possíveis exceções que o algoritmo deve considerar.

2

Planear a solução
Encontrar uma sequência lógica e precisa de passos para resolver o problema – um algoritmo. O algoritmo deve incluir todos os passos, mesmo aqueles que parecem óbvios. Esta fase de planeamento também deve envolver testes e validação do algoritmo usando dados representativos.

O desenvolvimento do algoritmo deve ser feito passo a passo, desde a definição das variáveis e das várias etapas de instruções até à construção de uma solução inicial, que pode ser sucessivamente refinada e melhorada.

Podes utilizar pseudocódigo, fluxogramas ou descrições textuais estruturadas. Independentemente da forma que escolhas, é fundamental que o algoritmo seja coerente, não ambíguo e eficiente. Lembra-te que é também importante que seja fácil de compreender. Deves valorizar a clareza da solução, garantindo que qualquer pessoa consiga compreender o funcionamento do algoritmo apenas lendo a sua descrição.

Relembra algumas boas práticas na escrita de algoritmos: usar nomes significativos para as variáveis, usar indentação nas instruções e colocar comentários breves.

3

Escolher a interface e codificar
Determinar em que ferramenta se pretende introduzir o algoritmo e traduzir o algoritmo para a linguagem de programação adequada.

Podes usar o Flowgorithm ou VisuAlg para introduzires o teu algoritmo num computador ou qualquer outra ferramenta com que te sintas à vontade.

4

Testar e corrigir erros (depurar)
Localizar e remover eventuais erros do programa. Os erros podem resultar do facto de o programa não estar escrito de acordo com as regras da linguagem de programação utilizada (erros de sintaxe) ou de não estar correta a sequência de ações a ser executada (erros de semântica). Estes erros têm de ser detetados através de testes exaustivos com dados variados para os quais a saída deve ser conhecida.

Após o desenvolvimento do programa, é fundamental testá-lo, para verificar se cumpre corretamente os objetivos definidos. Os testes devem ser realizados com vários utilizadores, para identificar erros, ambiguidades ou limitações da tua solução.

O *feedback* dos utilizadores é uma fonte valiosa de informação, pois permite avaliar a clareza, a utilidade e a eficácia do algoritmo num contexto real. Com base nos resultados dos testes, deves analisar o desempenho da solução e encontrar possíveis melhorias.

5

Completar a documentação

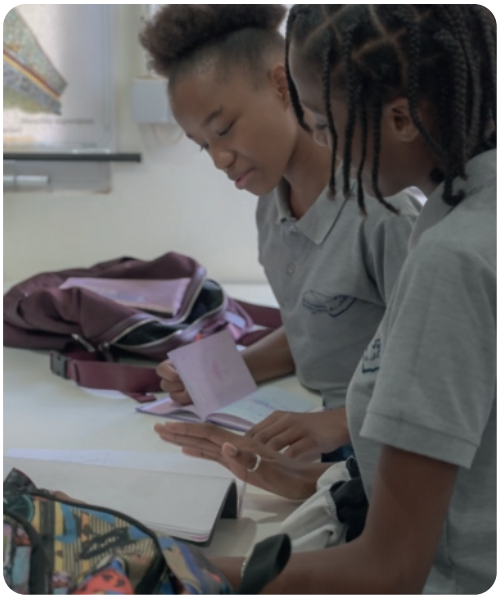
A documentação serve para que qualquer pessoa (ou, mais tarde, o próprio programador) entenda o programa.

Documentar um projeto significa registar todas as etapas do processo, desde a definição do problema até à solução final. Este registo irá permitir justificar as tuas decisões e explicar a qualquer pessoa o funcionamento do algoritmo.

Uma boa documentação inclui a descrição do problema, a análise realizada e a explicação do algoritmo passo a passo. Também é importante que registes as dificuldades encontradas e as opções que tomaste.

Trabalho colaborativo

O trabalho colaborativo é fundamental no desenvolvimento de projetos. Trabalhar em grupo permite a **partilha de ideias**, a **divisão de tarefas** e o **desenvolvimento de métodos** de trabalho mais autónomos e estruturados.



Num projeto colaborativo, é importante que o grupo defina claramente os papéis de cada elemento. Esta **distribuição de responsabilidades** ajuda a organizar o trabalho, promove o sentido de compromisso e garante que todos contribuem de forma ativa.

A **comunicação** eficaz entre os elementos do grupo é essencial para o sucesso do projeto. Os alunos devem discutir ideias, justificar decisões e resolver divergências de forma construtiva. O **diálogo** e a **argumentação** permitem melhorar a qualidade da solução final, uma vez que diferentes perspetivas ajudam a identificar erros, lacunas ou encontrar opções mais eficientes.

O trabalho colaborativo desenvolve competências transversais importantes, como a **capacidade de ouvir**, **cooperar**, **gerir conflitos** e **cumprir prazos**. Estas competências são fundamentais não só no contexto escolar, mas também em situações profissionais futuras.

☑ Not@ que:

O trabalho colaborativo é uma oportunidade para aprenderes não só conteúdos relacionados com algoritmos, mas também a trabalhar em equipa de forma organizada e responsável. Trabalhar em grupo significa que todos os elementos devem participar ativamente e contribuir para o sucesso do projeto.

O primeiro passo é a organização do grupo. Devem discutir o problema escolhido, definir objetivos claros e dividir as tarefas de forma equilibrada. Esta divisão ajuda a gerir melhor o tempo e evita que o trabalho fique concentrado apenas em uma ou duas pessoas.

A comunicação é essencial ao longo de todo o projeto. É importante partilharem ideias, ouvir as opiniões dos colegas e justificar as próprias decisões. Quando surgirem dúvidas ou opiniões diferentes, o grupo deve procurar chegar a um consenso através do diálogo e do respeito mútuo. Errar faz parte do processo de aprendizagem e discutir erros em grupo ajuda a melhorar a solução.

Cada aluno deve assumir a responsabilidade pela sua parte do trabalho e cumprir os prazos definidos pelo grupo. O sucesso do projeto depende do empenho individual e do esforço coletivo. A colaboração implica compromisso, responsabilidade e entreajuda.

Checklist da execução de um projeto

Uma ***checklist*** é uma ferramenta que te ajuda a organizar o trabalho, a não te esqueceres de passos importantes e a verificar se o projeto está bem feito.

Antes de iniciares o teu projeto:

– lê toda a *checklist* com atenção;

– garante que percebes cada ponto.

Usa a *checklist* como um guia, que podes consultar ao longo do teu trabalho. A *checklist* ajuda a saber o que já fizeste e o que ainda falta.

Checklist

Escolha e definição do problema

☑ O problema está claramente definido?

☑ O teu grupo compreende o problema a resolver?

☑ Os objetivos do projeto foram definidos?

☑ O público-alvo foi definido?

Análise do problema

☑ Foram identificadas as entradas (dados necessários)?

☑ Foram identificadas as saídas (resultados esperados)?

☑ Foram definidas regras, condições ou restrições importantes?

Desenvolvimento do algoritmo

☑ O algoritmo resolve corretamente o problema?

☑ Os passos estão organizados de forma lógica e sequencial?

☑ Foram realizados testes e corrigidos erros?

☑ Foram introduzidos comentários?

☑ Qualquer pessoa consegue compreender o algoritmo ao lê-lo?

Testes com utilizadores

☑ O algoritmo foi testado com vários utilizadores?

☑ Foram identificados erros ou pontos a melhorar?

☑ O *feedback* dos utilizadores foi registado?

Documentação do projeto

☑ Todas as etapas do trabalho estão documentadas?

☑ O funcionamento do algoritmo está explicado passo a passo?

☑ Foram registadas dificuldades e decisões tomadas pelo grupo?

Trabalho colaborativo

☑ As tarefas foram bem distribuídas entre os elementos do grupo?

☑ Todos os alunos participaram ativamente no projeto?

☑ O teu grupo comunicou e tomou decisões em conjunto?

Reflexão final

☑ O grupo analisou os resultados dos testes?

☑ Foram propostas melhorias para o algoritmo?

☑ O teu grupo refletiu sobre o que aprendeu com o projeto?

Esta *checklist* deve ser usada como guia ao longo de todo o projeto e também antes da entrega final, garantindo que todos os pontos foram cumpridos.

Testa os teus conhecimentos

Sozinho ou em grupo, escolhe um problema e desenvolve um projeto de construção de uma solução: um miniprograma de computador (utiliza uma das ferramentas com que aprendeste a trabalhar). Podes escolher qualquer situação do teu dia a dia ou alguma das sugestões que se apresentam de seguida.

Lembra-te que desenvolver um projeto não se resume apenas à escrita de um conjunto de instruções. Envolve um processo estruturado que começa na identificação e análise do problema real, passa depois para o planeamento da solução e pela construção do algoritmo, pela documentação de todos os passos e, finalmente, pela validação da tua solução através de testes com utilizadores. Cada uma destas fases desempenha um papel essencial no sucesso do teu projeto.

Projeto 1. Jogo de andebol

Descrição:

O teu programa deve permitir registar os golos de um jogo de andebol. A cada golo registado, o algoritmo deve perguntar se o golo é da equipa visitada ou da visitante e apresentar o resultado global do jogo.

Projeto 2. Tabuada interativa

Descrição:

O teu programa deve apresentar no ecrã a tabuada de um número escolhido pelo utilizador.

Deve listar os valores que são multiplicados em cada momento e o respetivo produto.

Projeto 3. Estimativa da conta de eletricidade

Descrição:

O teu programa deve ajudar as famílias a preverem o valor da fatura de eletricidade.

Deve ler um consumo em kWh e calcular o custo mensal.

Se quiseres, podes ajudar as famílias no cálculo da estimativa de kWh perguntando sobre a quantidade de eletrodomésticos e aparelhos eletrónicos e número de horas de uso.

Testa os teus conhecimentos

Projeto 4. Lista de compras

Descrição:

O teu programa pode ajudar uma pessoa a saber quanto vai gastar no supermercado.

Deve pedir ao utilizador que introduza os produtos a comprar, as quantidades e os preços. No final, deve apresentar o valor total, detalhando todas as compras.

Projeto 5. Cálculo dos gastos em transportes

Descrição:

O teu programa deve calcular os gastos mensais de uma pessoa em transportes.

Deve pedir ao utilizador que introduza o número e o preço de cada viagem e calcular o custo total por mês.

Pode diferenciar os dias da semana e fins de semana para uma estimativa mais exata.

Projeto 6. Custo em combustível de uma viagem

Descrição:

O teu programa deve calcular uma estimativa do consumo de um carro numa viagem.

Deve ler a distância a percorrer, o tipo de combustível e o consumo médio do carro, o preço do combustível por litro e, depois, calcular o custo total da viagem.

Podes optar por seres tu a consultar previamente os preços dos combustíveis ou solicitar ao utilizador que introduza o preço por litro.

Projeto 7. Gestão da mesada

Descrição:

O teu programa deve ajudar uma pessoa a controlar os gastos pessoais.

Deve permitir que o utilizador introduza o valor da mesada e registar as suas despesas.

Em cada interação, deve mostrar o total gasto e o saldo ainda disponível.

Projeto 8. Simulador de poupança

Descrição:

Desenvolve um programa que determine quanto tempo é necessário poupar para poder comprar um objeto especial.

O teu programa deve ler o preço do objeto a comprar e permitir várias simulações (em função do valor que é possível poupar em cada mês, informar quantos meses vão ser necessários).

Testa os teus conhecimentos

Projeto 9. Agenda de estudo

Descrição:

O teu programa deve ajudar um aluno a organizar o tempo de estudo.

Deve questionar sobre as horas de estudo necessárias por disciplina e calcular o tempo necessário total, fazendo uma distribuição do tempo de estudo por dias.

Projeto 10. Calculadora da pegada ecológica

Descrição:

Desenvolve um programa que promova a sensibilização ambiental.

O programa deve questionar o utilizador sobre alguns hábitos do dia a dia e depois calcular o seu impacto ambiental.

Procura na Internet dados e informações sobre a pegada ecológica para elaborares as tuas perguntas.

Projeto 11. Alerta meteorológico

Descrição:

Desenvolve um programa que analise dados meteorológicos introduzidos pelo utilizador e emita alertas automáticos com base em critérios previamente definidos.

Projeto 12. Jogo interativo

Descrição:

Desenvolve um jogo que permita aprender de forma interativa.

O teu programa deve apresentar perguntas e calcular a pontuação obtida por um jogador.

Será mais simples usar perguntas de opção múltipla em que o utilizador apenas tem de introduzir qual a opção (letra ou número) que acha estar correta. Para cada resposta, o programa deve dar um *feedback*, informando se a resposta está certa ou se está errada.

Programação 10.º ano

Criação intelectual
Dina Tavares
Helena Reis
Rita Cadima

Design
Porto Editora

Créditos fotográficos
© Stock.Adobe.com
Depositphotos.com
Pedro Moita – pp.19, 93, 94, 102 (sup.), 105, 111, 112, 114, 137, 163, 167, 169, 170, 172 (inf.), 173, 174.

Edição
2026

Este manual segue o programa experimental da disciplina, publicado pelo Ministério da Educação.

Cabo Verde

Brasão

Bandeira

Hino Nacional

Cântico da Liberdade

Canta, irmão
Canta, meu irmão
Que a liberdade é hino
E o homem a certeza.

Com dignidade, enterra a semente
No pó da ilha nua;
No despenhadeiro da vida
A esperança é do tamanho do mar
Que nos abraça,
Sentinela de mares e ventos
Perseverantes
Entre estrelas e o Atlântico
Entoa o cântico da liberdade.

Canta, irmão
Canta, meu irmão
Que a liberdade é hino
E o homem a certeza!

Este livro foi aprovado pelo Ministério da Educação para utilização obrigatória nas escolas