Desenvolvimento de Sistemas

Sub-rotinas: declaração de funções e procedimentos, parâmetros, retorno e chamada

Antes de prosseguir, é recomendada a leitura prévia dos conteúdos a seguir, pois é a partir deles que o curso evoluirá. Sem essa base, poderá haver dificuldades na leitura e nas práticas propostas neste conteúdo.

- Lógica de Programação: introdução à lógica; conceito de algoritmo.
- Algoritmos: naturais e estruturados; representações visuais, linguagem algorítmica; comandos de entrada, processamento e saída de dados.
- Manipulação de dados: tipos de dados, variáveis e constantes; expressões e operadores.
- Condicionais: lógica booliana, estrutura condicional simples e composta.
- Repetições: estruturas de repetição condicional pré-teste, pós-teste e com variável de controle.

Agora, se você já concluiu a leitura dos conteúdos citados, então chegou a hora de aprofundar-se nas sub-rotinas. Fazem parte desse conteúdo o desenvolvimento de novos conceitos e de práticas em programação, além da revisão de conceitos e de práticas passadas. Portanto, para um melhor aproveitamento do conteúdo, você já deve ter o Portugol Webstudio aberto em seu computador. Assim você poderá copiar e testar cada exemplo conforme avança no conteúdo e ver na prática como cada algoritmo se comporta.

Declaração de funções e procedimentos

Funções e procedimentos são duas das principais ferramentas para um desenvolvedor tornar o processo de desenvolvimento de sistemas mais eficiente. Ambas funcionam com o princípio de escrever um pequeno pedaço de código que pode ser chamado várias vezes e em qualquer lugar de um programa. Essa prática, além de reduzir o tamanho do programa, também melhora bastante a legibilidade do código.

Imagine que você tenha que desenvolver um algoritmo que calcule a soma de dois números e depois exiba na tela o resultado. Com os conhecimentos que você obteve, você deve imaginar que o seu algoritmo ficará mais ou menos assim:

```
programa {
    funcao inicio() {
        // Declaramos as variáveis
        inteiro a, b, resultado

        // Passamos os valores para a e b
        a = 1
        b = 2

        // Realizamos a soma e guardamos o valor em resultado
        resultado = a + b

        // Exibimos na tela
        escreva("O resultado é: " + resultado)
    }
}
```

Agora pense que você precisa fazer exatamente a mesma operação com números diferentes em **a** e **b**. Para facilitar o processo e otimizar seu tempo, é comum pensar em “copiar e colar” o código. Nesse caso, você terá o seguinte resultado:

```
programa {
    funcao funcao inicio() {
        // Declaramos as variáveis
        inteiro a, b, resultado

        // Passamos os valores para a e b
        a = 1
        b = 2

        // Realizamos a soma e guardamos o valor em resultado
        resultado = a + b

        // Exibimos na tela
        escreva("O resultado é: " + resultado)

        // Passamos os valores para a e b
        a = 10
        b = 20

        // Realizamos a soma e guardamos o valor em resultado
        resultado = a + b

        // Exibimos na tela
        escreva("O resultado é: " + resultado)
    }
}
```

Agora caso você tenha que fazer exatamente o mesmo novamente, em programação, é muito comum repetir um determinado trecho de código em outras partes de um sistema. O código acima é um bom exemplo dessa situação.

Em alguns cenários, o uso de laços de repetições pode ser uma boa solução. Porém, essa solução não permitirá reaproveitamento de código em outras partes do sistema, o que leva, novamente, ao “copiar e colar”. Essa é uma das principais vantagens de usar funções e procedimentos. Se você precisar executar a mesma tarefa várias vezes dentro do sistema, você pode escrever o algoritmo apenas uma vez e chama-lo várias vezes.

Na verdade, você já tem feito isso. Lembra das funções **leia()** e **escreva()** que você já usou várias vezes em seus algoritmos? Esses são exemplos de funções prontas no Portugol e são chamadas de **funções internas**. A função **leia()** é responsável por ler os dados que o usuário digita no teclado, enquanto a função **escreva()** recebe em seus parâmetros um texto para exibir na tela. Ambas as funções têm uma implementação bastante complexa e muitas linhas de código, com várias verificações, tratamentos e conversões para garantir que tudo funcione corretamente como se espera. Assim é o código-fonte de muitas funções internas.

Já imaginou se tivesse que reescrever esses blocos de código toda vez que você quisesse escrever ou ler algo no programa? Como essas funções já existem implementadas, basta fazer a chamada delas sempre que precisar, sem repetir o código-fonte.

Mas se já existe o código das funções pronto, não é uma solução válida “copiar e colar” o código? Por mais que essa solução funcione para atingir um determinado objetivo, como funcionou no exemplo acima, ela carrega consigo vários problemas. O primeiro problema é o tempo necessário para copiar e colar os trechos de código em outras partes do algoritmo. Depois disso, o código ficará extremamente extenso e difícil de ler. E se precisar fazer um ajuste? Em vez de, por exemplo, somar 2 números, precisar subtrair 2 números. Será necessário ir a cada bloco de código e fazer a alteração manualmente ou apagar tudo e repetir o processo de “copiar e colar” do zero. Esses são apenas alguns problemas que ocorrem ao se tentar construir algoritmos sem o uso de funções e procedimentos adequados.

Além de reduzir o tempo de codificação, o uso de funções e procedimentos também tem impacto positivo na manutenção de um sistema. Se uma parte do sistema não estiver funcionando corretamente, por exemplo, você conseguirá encontrar a falha facilmente entre suas funções e procedimentos e corrigi-la sem afetar outras funções que compõem o *software*.

Pode-se resumir as vantagens do uso de funções e procedimentos como:

- Permitir o reaproveitamento de código já construído.
- Evitar que um trecho de código seja repetido muitas vezes dentro do mesmo programa sem necessidade.
- Permitir a manutenção e a alteração de trechos de código de forma mais rápida e eficiente, pois só será preciso mudar o código dentro da função/procedimento que desejar corrigir.
- Tornar os blocos de código mais curtos e mais fáceis de serem entendidos.
- Separar o programa em pequenas partes que possam ser compreendidas de forma isolada.

Funções e procedimentos estão presentes em praticamente todas as linguagens de programação e oferecem muitas vantagens ao desenvolvedor, por isso são tão usados com tanta frequência. Portanto, independentemente da linguagem de programação que você utilizar para desenvolver seus projetos de *software* (exemplo: Java, C#, PHP etc.), o uso de funções e procedimentos estará presente no seu dia a dia como desenvolvedor.

Em poucas linguagens de programação, há uma distinção clara entre as duas, muitas preferem tratar tudo como funções, como na linguagem SQL, por exemplo. Na prática, funções e procedimentos funcionam da mesma forma e suas implementações são essencialmente as mesmas.

Conceitualmente, um procedimento difere de uma função pela ausência de um retorno de valor. Portanto um **procedimento** é um algoritmo que será executado, enquanto uma **função** é um algoritmo que será executado e produzirá um resultado final concreto que poderá ser usado por outro algoritmo. De qualquer forma, ambos exercem uma função dentro de um programa e têm exatamente a mesma implementação.

Em linguagens de programação orientada a objetos, como Java e C#, funções e procedimentos são chamados de métodos. Isso ocorre por conta dos conceitos estabelecidos de orientação a objetos. Esse assunto será abordado ainda neste curso.

Procedimento

Também chamado de *procedure* (“procedimento” em inglês). Na prática, procedimentos são muito similares a uma função, porém não têm um retorno (em breve, o curso abordará o que são e como funcionam os retornos). **Um procedimento é um algoritmo que será executado**. Você pode vê-lo com um bloco de código que será construído e poderá ser chamado posteriormente para ser usado em outras partes de um algoritmo ou um sistema. Os procedimentos são muito úteis para realizar tarefas especificas mais de uma vez sem precisar escrever novamente todo o seu passo a passo.

No que diz respeito à sua estrutura, os procedimentos têm a seguinte sintaxe:

```
funcao nomeProcedimento() {
    ...
}
```

Primeiramente, é necessário declarar um procedimento. Para isso, usa-se a palavra-chave “**funcao**” (sem acentos ou caracteres especiais). Apesar de a palavra reservada ser “**funcao**”, ainda é a declaração de um procedimento. Para o Portugol e muitas linguagens de programação, não faz sentido reservar uma outra palavra-chave apenas para procedimento, pois, na prática, seu funcionamento é exatamente como uma função. Portanto, a palavra-chave usada para declarar tanto funções quanto procedimentos é “**funcao**”.

Em nomeProcedimento, define-se o nome do procedimento. Você pode usar o nome que quiser, desde que obedeça às mesmas regras de nomenclatura da declaração de variáveis. Essas regras foram abordadas no conteúdo de “Algoritmos”, no trecho “Blocos de declarações”, e também se aplicam às linguagens de programação das próximas Unidades Curriculares do curso. Você já viu, mas vale relembrar. Essas regras são:

- Podem ser usados letras, números e o caractere “_” (*underscore*).
- Sempre começar por letras ou pelo caractere “_”.
- Podem ser usadas letras maiúsculas ou minúsculas. Cuidar no Portugol, pois ele é *case sensitive*, ou seja, difere letra maiúscula de letra minúscula. Exemplo: **senac** é diferente de **Senac**.
- Não podem ser usados símbolos com: $, #, ?, !, &, >, <, +,-.
- Nunca usar espaços em branco e caractere com - (hífen).
- Não usar palavras reservadas. As palavras reservadas são identificadores predefinidos que têm significados especiais para o interpretador do algoritmo e variam de acordo com a linguagem utilizada. No Portugol, observe as seguintes palavras:

Programa	funcao	Inicio
Inteiro	real	Cadeia
Escreva	leia	Se
Senão	const	Inclua

Logo após o nome do procedimento, inserir um “abre e fecha” parênteses seguido de um “abre e fecha” chaves. Em comparação com a declaração de uma variável, a principal novidade é o uso de parênteses e chaves após o nome ser definido. As chaves são usadas para indicar onde o código começa e termina e delimitam o escopo da função ou do procedimento. Já os parênteses são usados para passar parâmetros. Você aprenderá o que são, para que servem e como usar os parâmetros em breve. Por enquanto, as funções não receberão parâmetros, mas ainda assim os parênteses serão usados, pois eles fazem parte da estrutura-base de uma função.

O nome nomeProcedimento() foi usado apenas para ilustrar como funciona a declaração de um procedimento. Em uma situação real, não se deve usar caracteres especiais nem acentos.

Veja agora na prática como declarar procedimentos:

Exemplo de procedimento de somar

```
funcao somar()
{
  inteiro numero1 = 3
  inteiro numero2 = 4

  inteiro resultado = numero1 + numero2

  escreva("\nO resultado da soma é: " + resultado )
}
```

Exemplo de procedimento de subtrair

```
funcao subtrair()
{
  inteiro numero1 = 9
  inteiro numero2 = 7

  inteiro resultado = numero1 - numero2

  escreva("\nO resultado da subtração é: " + resultado)
}
```

Definir uma função ou um procedimento não é o suficiente para executá-los. Para que a execução ocorra, é necessário fazer uma **chamada**. Você certamente já sabe como fazê-la. Lembre-se de que as **funções internas** já foram mencionadas. Você tem usado essas funções fazendo a chamada delas.

Exemplo de chamada de funções internas

```
programa
{
funcao inicio ()
{
  cadeia nome

  escreva("Digite seu nome: ")
  leia(nome)
}
}
```

Exemplo de declaração e chamada de procedimentos

Agora, use os procedimentos de declaração como exemplo e chame-os no algoritmo principal. O resultado final será o seguinte:

```
programa
{
funcao inicio()
{
  somar()
  subtrair()
}

funcao somar()
{
inteiro numero1 = 3
inteiro numero2 = 4

inteiro resultado = numero1 + numero2

escreva("\nO resultado da soma é: " + resultado )
}

funcao subtrair()
{
inteiro numero1 = 9
inteiro numero2 = 7

inteiro resultado = numero1 - numero2

escreva("\nO resultado da subtração é: " + resultado)
}

}
```

A chamada de funções e procedimentos é bem simples de fazer, mas lembre-se de que, ao chamar uma função ou procedimento, um bloco de código no algoritmo está sendo chamado. Se houver mais passos para seguir após a chamada da função, então o algoritmo seguirá normalmente.

Chegou a hora de realizar alguns desafios!

Faça um procedimento que troque os valores de x por y. Se x receber 50 e y 10, no final deve ser escrito na tela que o valor de x é 10 e de y é 50.

Faça um procedimento que receba 3 valores inteiros e escreva na tela a ordem crescente desses números.

Função

Também chamada de *function* (“função” em inglês). Toda função criada precisa obrigatoriamente ter um retorno. Dessa forma, uma função é definida como um algoritmo que será executado e, no final, retornará uma informação que poderá ser usada ou não. Ainda neste conteúdo, você entenderá detalhadamente como funciona o retorno em funções, mas antes precisa conhecer como é a estrutura de uma função.

Exemplo de declaração de função:

```
funcao tipoRetornado nomeFunção() {
    ...
}
```

Primeiro, você precisa dizer que está declarando uma função. Como você quer uma “função”, use a palavra-chave “**funcao**” (sem acentos nem caracteres especiais) exatamente como na declaração de um procedimento.

Por se tratar de uma função, é necessário retornar algum dado no final, mas não basta inserir um “retorno” no final do código. Como a função poderá ser usada em outros contextos do código, haverá situações em que o programa precisará saber com quais tipos de dados ele está lidando. E é por isso que precisa ficar explícito que o método retornará, por exemplo, um inteiro ou um texto.

Veja mais alguns exemplos de como declarar uma função:

Exemplo de função de somar com inteiros

No exemplo a seguir, a declaração de uma função realizará a soma de dois números (numero1 e numero2). Perceba que, no final deste algoritmo, é retornada a variável resultado. Por se tratar de uma variável do tipo inteiro, o retorno da função é do tipo inteiro. Por essa razão, na primeira linha, o tipo é definido como inteiro.

```
funcao inteiro somarInteiro()
{
inteiro numero1 = 3
inteiro numero2 = 4

inteiro resultado = numero1 + numero2

retorne resultado
}
```

Exemplo de função de somar com real

No exemplo a seguir, a situação é similar à situação do exemplo anterior. Porém, em vez de somar dois números inteiros, são somados dois números reais.

Consequentemente, o tipo de dado do resultado será real e o retorno também.

```
funcao real somarReal()
{
real numero1 = 3.4
real numero2 = 4.99


real resultado = numero1 + numero2

retorne resultado
}
```

Conforme mencionado no uso de procedimentos, a chamada de funções é feita exatamente da mesma forma.

Exemplo de declaração e chamada de funções

```
programa
{
funcao inicio()
{
  somarInteiro()
  somarReal()
}

funcao inteiro somarInteiro()
{
 inteiro numero1 = 3
 inteiro numero2 = 4

 inteiro resultado = numero1 + numero2

 retorne resultado
}


funcao real somarReal()
{
  real numero1 = 3.4
  real numero2 = 4.99


  real resultado = numero1 + numero2

  retorne resultado
}
}
```

Agora que você já conhece as funções e os procedimentos, poderá compará-los lado a lado.

Exemplo de função

```
funcao inteiro somar()
{
   inteiro numero1 = 2
   inteiro numero2 = 2

   inteiro resultado = numero1 + numero2

   retorne resultado
}
```

Exemplo de procedimento

```
funcao somar()
{
   inteiro numero1 = 2
   inteiro numero2 = 2

   inteiro resultado = numero1 + numero2

   escreva(resultado)
}
```

Na prática, como você pôde observar, a principal diferença entre uma função e um procedimento está no retorno de dados no final. Como uma função deve retornar um dado, é necessário apontar o tipo de dado que será retornado antes mesmo de definir o nome da função, mas você sabe o que realmente significa “retornar um dado”?

Retorno

No conteúdo “Algoritmos”, você aprendeu como funciona a entrada, o processamento e a saída de dados. De forma resumida, é possível definir que:

- **Entrada**: refere-se a etapa na qual os dados serão passados.
- **Processamento**: nessa etapa, os dados serão processados.
- **Saída**: após o processamento, é gerado um resultado final, e é na saída que esse resultado é retornado.

Retorno (também conhecido como *return* em inglês) é a saída de dados de um algoritmo. Logo, um retorno encerra a execução de uma função. Na prática, há funções que retornarão valores, e essas funções serão chamadas no programa principal. No Portugol, para retornar um dado, usa-se a palavra-chave **retorne**.

Veja um exemplo a seguir:

```
programa
{
    funcao inicio()
    {
        escreva("Vamos chamar a função...\n")
        somar()
        escreva("A função foi chamada e executada!\n")
    }

    funcao inteiro somar()
    {
        inteiro numero1 = 2
        inteiro numero2 = 2

        inteiro resultado = numero1 + numero2

        retorne resultado
    }
}
```

Nesse exemplo, há a função **somar()**, que retornará o valor de resultado. No algoritmo principal, foi realizada a chamada dessa mesma função entre as duas funções **escreva()**. Você precisa verificar que o retorno do resultado terminou a execução da função **somar()**, e não da função **inicio()**. Portanto, lembre-se de que, ao chamar uma função, você está chamando um bloco de código no algoritmo. Se tiver mais passos para seguir após a chamada da função, então o algoritmo seguirá normalmente. Logo, uma instrução de retorno faz com que a execução saia da função atual e continue no ponto de código seguinte imediatamente.

Se você executar esse algoritmo, terá o seguinte resultado no terminal:

```
Vamos chamar a função...
A função foi chamada e executada!

Programa finalizado.
```

Talvez você esteja se perguntando “por que o resultado da função **somar()** não apareceu?”. Retornar um dado não significa “exibir na tela”. Até então, você tem feito muito o uso da função **escreva()** para mostrar na tela alguma informação, seja um texto ou um número, por exemplo. Porém, a instrução **escreva()** apenas mostra ao usuário humano uma informação do que está acontecendo no computador. O computador não pode usar essa informação.

Exibir uma informação na tela não afetará de forma alguma a execução de uma função. Será apenas mais uma instrução para ser executada (no caso, processada) para o usuário se beneficiar de alguma maneira. Uma boa forma de aplicar esse recurso é na depuração de um algoritmo para verificar o que o computador está fazendo sem interromper o programa. A seguir, um exemplo prático:

```
programa
{
    funcao inicio()
    {
        escreva("Vamos chamar a função...\n")
        somar()
        escreva("A função foi chamada e executada!\n")
    }

    funcao inteiro somar()
    {
        inteiro numero1 = 2
        inteiro numero2 = 2
        escreva("As variáveis numero1 e numero2 foram declaradas...\n")

        inteiro resultado = numero1 + numero2
        escreva("A variável resultado recebeu o valor da soma...\n")

        retorne resultado
    }
}
```

Nesse algoritmo, foi realizado o seguinte:

1. Criação de um bloco de código para declarar as variáveis numero1 e numero2 e atribuir o valor 2 para ambas.
2. Escrita de uma mensagem na tela para dizer que a etapa anterior foi concluída.
3. Declaração da variável resultado e atribuição do valor da soma de numero1 e numero2, que é 4.
4. Novamente, escrita de uma mensagem na tela para dizer que a etapa anterior foi concluída.
5. Por fim, retorno do valor da variável resultado.

Em caso de remoção das etapas 2 e 4, o algoritmo fará exatamente as mesmas ações. Porém, por mais que o computador saiba o que está acontecendo, o usuário não terá esse feedback.

Já o **retorne()** retorna um dado. Esse dado não é visto pelo usuário humano, mas pode ser usado pelo computador de alguma forma. Portanto, nem sempre retornar um dado será suficiente. Às vezes, é necessário dizer o que fazer com esse dado. Geralmente, esse dado é atribuído como valor de uma variável para ser usado em algum momento no algoritmo ou é escrito na tela para o usuário. Veja alguns exemplos para essa situação:

```
programa
{
    funcao inicio()
    {
        escreva("Vamos chamar a função...\n")
        inteiro resultadoDaSoma = somar()
        escreva("A função foi chamada e executada!\n")
        escreva("O resultado da soma é: " + resultadoDaSoma)
    }

    funcao inteiro somar()
    {
        inteiro numero1 = 2
        inteiro numero2 = 2
        escreva("As variáveis numero1 e numero2 foram declaradas...\n")

        inteiro resultado = numero1 + numero2
        escreva("A variável resultado recebeu o valor da soma...\n")

        retorne resultado
    }
}
```

Outro exemplo para escrever esse algoritmo, sem atribuir o retorno da função **somar()** a uma variável, seria da seguinte maneira:

```
programa
{
    funcao inicio()
    {
        escreva("O resultado da soma é: " + somar())
    }

    funcao inteiro somar()
    {
        inteiro numero1 = 2
        inteiro numero2 = 2

        inteiro resultado = numero1 + numero2

        retorne resultado
    }
}
```

Perceba que, nesse exemplo, foram removidas as chamadas da função escreva() para deixar o algoritmo mais consistente. No final da execução, há o seguinte resultado:

```
O resultado da soma é: 4
Programa finalizado.
```

Como o retorno representa a saída de dados, naturalmente só poderá ter um único retorno em uma função.

Até então, você está vendo algoritmos simples para ilustrar o uso de funções e procedimentos. Quando for iniciado o uso de laços de repetição, por exemplo, a forma de retornar dados necessitará de mais atenção a fim de evitar erros de lógica.

Observe os exemplos a seguir:

Exemplo de retorno usando **enquanto**

```
programa
{
funcao inicio()
{
  escreva("O resultado da soma é: " + somar())
}

funcao inteiro somar()
{
  inteiro x = 0

  enquanto (x < 10)
  { // Inicio do nosso laço de repetição

      // Aumentamos +1 o valor de x
      x = x+1

  } // Fim do nosso laço de repetição

  retorne x
}
}
```

No exemplo acima, há uma função que começa com a declaração do valor de x que é igual a zero. Logo após, há um laço de repetição para repetir 10 vezes a operação de aumentar o valor de x em +1. Só é retornado o valor depois que o laço de repetição termina. Se houver algum descuido e for inserido o “retorne” dentro do **enquanto()**, na execução do algoritmo, o algoritmo não funcionará como esperado, pois o uso do “retorne” indica o fim do algoritmo. Logo, o bloco de códigos no **enquanto()** só seria executado uma única vez.

Lembre-se sempre de que o retorno deve ser feito no fim de uma função. Caso haja mais passos para seguir no algoritmo após o retorne, eles serão ignorados ou, dependendo da tecnologia utilizada, poderão gerar erros de compilação que impedirão o programa de ser executado.

Apesar de só poder ter um retorno no final da função, isso não significa que possa haver situações que levem a outras possibilidades de retorno. Imagine um cenário em que você tenha que desenvolver um algoritmo para verificar qual número é

maior entre os valores de x e y. Se x for maior que y, retorna x. Caso y seja maior que x, retorna y. Essas são duas possibilidades de retorno diferentes. Veja no próximo exemplo como ficaria esse algoritmo.

Exemplo de retorno usando **SE** e **SENAO**

```
programa {
  funcao inicio() {
    escreva(“O número maior é: “ + numeroMaior())
  }

  funcao inteiro numeroMaior()
  {
      inteiro x = 5
      inteiro y = 10

      se (x >= y) {
          retorne x
      } senao {
          retorne y
      }
  }
}
```

Perceba que, no final desse algoritmo, há duas possibilidades de retorno. Mas, em nenhum cenário, há 2 retornos acontecendo simultaneamente. Dessa forma, mais tratamentos e refinamentos podem ser acrescentados às funções que levam a cenários de retornos diferentes.

Chegou a hora de realizar alguns desafios!

Você deve fazer uma função que recebe um valor inteiro e verifica se é positivo ou negativo. A função deve retornar o valor lógico “verdadeiro” caso o valor seja positivo ou “falso” caso seja negativo.

Desenvolva uma função que recebe um valor inteiro como parâmetro e retorna o valor lógico “verdadeiro” caso o valor seja primo ou “falso” caso não seja.

Sabendo que a gorjeta do garçom é 10% do valor da refeição, faça uma função na qual você informa o valor da refeição e o valor total é retornado já com a gorjeta do garçom.

Parâmetros

A fim de tornar mais amplo o uso de funções, existe um recurso chamado **parâmetros**, que permite definir sobre quais dados uma função deve operar.

A função **escreva()**, por exemplo, recebe como parâmetro o que será escrito na tela, permitindo que seu comportamento seja definido a partir desse valor.

Exemplo de uso da função **escreva()**:

```
programa {
    funcao inicio() {
        escreva("Olá, mundo!")
    }
}
```

Os dados solicitados na **declaração** de uma função são chamados de **parâmetros formais**, mas podem ser chamados apenas de **parâmetros**. Já os valores passados na **chamada** de uma função são chamados de **argumentos**, mas também podem ser chamados de **parâmetros atuais/reais**.

Uma função pode não ter nenhum ou vários parâmetros. Até aqui, foram declarados funções e procedimentos sem nenhum parâmetro. Agora você verá na prática como fazer a declaração e a chamada de uma função que receberá um valor como parâmetro.

Outra particularidade interessante no uso de parâmetros é a independência em relação ao tipo de dado que será retornado. Pode haver uma função com parâmetros do tipo inteiro e reais, por exemplo, mas, no final do algoritmo, ter um retorno do tipo cadeia. Isso mostra que, em alguns casos, são necessários tipos de dados diferentes para se chegar a um determinado resultado. Um bom exemplo desse cenário é o cálculo de Índice de Massa Corporal (IMC). Caso você não conheça, o IMC é um cálculo que usa seu peso, sua idade e sua altura para medir o seu índice de massa corporal. O resultado classificará sua massa corporal em magreza, normal, sobrepeso ou obesidade.

O cálculo realizado é:

IMC = peso / (altura x altura)

Os resultados obtidos podem ser:

IMC	Classificação
Menor que 18,5	Magreza
Entre 18,5 e 24,9	Normal
Entre 25 e 29,9	Sobrepeso
Entre 30 e 39,9	Obesidade
Maior que 40	Obesidade Grave

Para ilustrar esse cenário, você verá uma função com o cálculo de IMC usando os parâmetros idade, altura e peso. No final, retornará uma cadeia com o valor do IMC e a classificação que representa esse valor.

```
programa
{
    funcao inicio()
    {
        cadeia resultado = calcularImc(24, 1.68, 67.2)
        escreva(resultado)
    }

    funcao cadeia calcularImc(inteiro idade, real altura, real peso)
    {
        cadeia resultado = ""
        real imc = peso/(altura * altura)

        se (imc <= 0)
        {
            resultado = "IMC inválido!"
        }

        se (imc > 0 e imc < 18.5)
        {
            resultado = "você está abaixo do peso."
        }

        se (imc >= 18.5 e imc < 25)
        {
            resultado = "você está no peso normal."
        }

        se (imc >= 25 e imc < 30)
        {
            resultado = "você está acima do peso."
        }

        se (imc >= 30 e imc < 35)
        {
            resultado = "você está com sobrepeso."
        }

        se (imc >= 35 e imc < 40)
        {
            resultado = "você está com obesidade."
        }

        se (imc >= 40)
        {
            resultado = "você está com obesidade grave."
        }

        retorne "Seu IMC é " + imc + " e " + resultado
```

```
    }
}
```

Chegou a hora de praticar!

Escreva um procedimento que receba um número inteiro digitado pelo usuário e verifique se esse número está entre o intervalo de 1 a 10. Se o número estiver den-tro do intervalo, escreva na tela “valor válido”, e se o número não estiver dentro do intervalo, escreva na tela “valor inválido”.

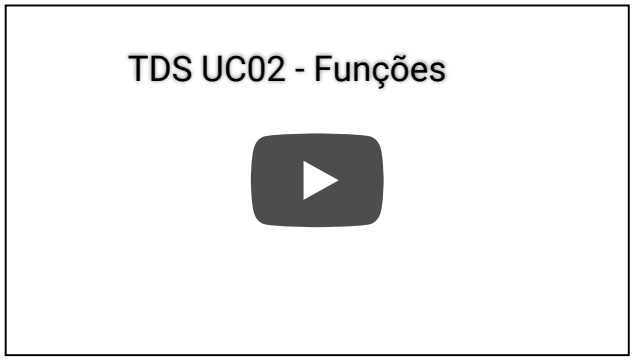
Faça um procedimento que receba um valor inteiro como parâmetro e escreva na tela todos os números a partir desse valor até 100. Exemplo: se o valor passado for 89, deve ser escrito na tela “89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100”.

Faça uma função que receba dois valores como parâmetro: um número-base e uma potência. Após isso, calcule a potência e retorne o resultado. Não use a biblioteca Util.

Desenvolva um procedimento que leia 4 notas e informe a média aritmética. Deverá imprimir “Aprovado” se a média for maior ou igual a 7, caso contrário, deverá imprimir “Reprovado”.

Desenvolva uma função que receba o dia, o mês e o ano de nascimento de uma pessoa e retorne a idade que essa pessoa tem.

Assista ao vídeo a seguir que apresenta exemplos de algoritmos com sub-rotinas.



Bibliotecas

Imagine que você precisa calcular a raiz quadrada de um valor. Porém, você não encontra nenhuma função interna para ajudá-lo nessa tarefa. Então você precisa desenvolver uma função que faça esse cálculo. Após um tempo organizando a lógica de programação e desenvolvendo a função, você tem ela pronta! Mas por mais que você tenha desenvolvido a sua própria solução para calcular a raiz quadrada, outras pessoas estão enfrentando problemas e não estão conseguindo chegar ao mesmo resultado. Então você decide compartilhar seu projeto com outras pessoas. Assim surgem as bibliotecas.

Bibliotecas, em programação, são soluções desenvolvidas e compartilhadas por meio de funções e procedimentos. Assim outros desenvolvedores podem utilizá-las, permitindo que o desenvolvimento seja mais rápido e fácil. Se apenas com funções e procedimentos já é possível economizar muito tempo de desenvolvimento e deixar os códigos mais limpos, as bibliotecas oferecem isso tudo em dobro para facilitar mais o desenvolvimento de sistemas.

Uma das principais vantagens das bibliotecas é que você pode utilizá-las em diversos projetos. Logo, você terá funções e procedimentos que poderão ser chamados e usados em diferentes situações.

Para usar uma biblioteca em algoritmos do Portugol, é necessário incluí-la no código. Observe a seguinte estrutura:

```
inclua biblioteca nomeBiblioteca --> nomeParaChamar
```

inclua biblioteca

Em “*inclua biblioteca*”, o algoritmo fará a importação de algo externo para o seu projeto, no caso, uma biblioteca.

nomeBiblioteca

Em “*nomeBiblioteca*”, você especifica o nome da biblioteca que deseja usar. No Portugol, há três bibliotecas diferentes: Matematica, Texto e Util. Cada biblioteca tem suas próprias funções e procedimentos com a finalidade de resolver um determinado problema.

nomeParaChamar

Em “*nomeParaChamar*”, você define um nome para facilitar a chamada da biblioteca no algoritmo. O algoritmo saberá que sempre que o *nomeParaChamar* for chamado significa que a biblioteca está sendo chamada. Seria um apontado da biblioteca por meio de uma variável.

Para ilustrar como trabalhar com bibliotecas, serão exploradas as principais funcionalidades da biblioteca Matematica. Caso tenha interesse em conhecer as outras bibliotecas e todas as suas funções, você pode conferir no próprio Portugol Webstudio.

Matematica

Essa biblioteca é composta por funções prontas para realizar cálculos matemáticos mais complexos. Suas principais funções são:

raiz()

Biblioteca que recebe dois parâmetros. O primeiro é o valor para se calcular a raiz e o segundo é o tipo de raiz (quadrada ou cúbica, por exemplo).

Veja a seguir um exemplo de como usar essa biblioteca:

```
programa
{
inclua biblioteca Matematica --> mat

funcao inicio()
{
    real numero
    real raiz

    numero = 4.0
// Para chamarmos a função que queremos usar, usamos mat.NomeDaFunção(parametros)

    raiz = mat.raiz(numero, 2.0) // Obtém a raíz quadrada do número

    escreva("A raíz quadrada de ", numero , " é: ", raiz, "\n")

    numero = 27.0
    raiz = mat.raiz(numero, 3.0) // Obtém a raíz cúbica do número

    escreva("A raíz cúbica de ", numero , " é: ", raiz, "\n")
}
}
```

potencia()

Função que recebe dois parâmetros (número e base) para realizar o cálculo.

Veja a seguir um exemplo de como usar essa biblioteca:

```
programa
{
inclua biblioteca Matematica --> mat

funcao inicio()
{
    real base, quadrado, cubo, resultado

    escreva("Informe um número: ")
    leia(base)

    // Eleva o número informado ao quadrado
    quadrado = mat.potencia(base, 2.0)
    escreva("\n", base, " ao quadrado é igual a: ", quadrado)

    // Eleva o número informado ao cubo
    cubo = mat.potencia(base, 3.0)
    escreva("\n", base, " ao cubo é igual a: ", cubo, "\n")
}
}
```

logaritmo()

Função que recebe dois parâmetros (número e base) para realizar o cálculo.

Veja a seguir um exemplo de como usar essa biblioteca:

```
programa
{
inclua biblioteca Matematica --> mat

funcao inicio()
{
    real numero, base, expoente

    base = 2.0
    numero = 32.0

    expoente = mat.logaritmo(numero, base)
    escreva("O logaritmo de ", numero, " na base ", base, " é igual a: ", expoente,
"\n")

    base = 25.0
    numero = 625.0

    expoente = mat.logaritmo(numero, base)
    escreva("O logaritmo de ", numero, " na base ", base, " é igual a: ", expoente,
"\n")
}
}
```

arredondar()

Função útil para fazer o arredamento de valores reais. A função recebe dois parâmetros: o número a ser arredondado (valor do tipo real) e o número de casas decimais desejado para o arredondamento (valor do tipo inteiro).

Veja a seguir um exemplo de como usar essa biblioteca:

```
programa
{
inclua biblioteca Matematica --> mat

funcao inicio()
{
    real numero = 1.25673245
    real arredondamento

    // Arredonda o número para 3 casas decimais, isto é,
    // 3 casas depois da vírgula
    arredondamento = mat.arredondar(numero, 3)
    escreva("O número com 3 casas decimais é: ", arredondamento, "\n")


    // Arredonda o número para 2 casas decimais, isto é,
    // 2 casas depois da vírgula
    arredondamento = mat.arredondar(numero, 2)
    escreva("O número com 2 casas decimais é: ", arredondamento, "\n")


    // Arredonda o número para 1 casa decimal, isto é,
    // 1 casa depois da vírgula
    arredondamento = mat.arredondar(numero, 1)
    escreva("O número com 1 casa decimal é: ", arredondamento, "\n")

    // Arredonda o número removendo todas as casas decimais
    arredondamento = mat.arredondar(numero, 0)
    escreva("O número sem casas decimais é: ", arredondamento, "\n")
}
}
```

Essas são apenas algumas funções disponíveis na biblioteca Matematica. Como você pôde perceber, tudo que você precisa fazer, na prática, é:

1. Incluir a biblioteca em uma variável.

   Exemplo: inclua biblioteca Matematica --> **mat**.

2. Chamar a função usando nomeDaVariavel.função(parametros)

   Exemplo: mat.potencia(numero, base).

Bibliotecas seguem rigorosamente as boas práticas para serem mais acessíveis para os desenvolvedores. Portanto, o processo para usar outras bibliotecas é exatamente igual. Aproveite para explorar mais o uso de bibliotecas no Portugol Webstudio e aplicar o que aprendeu em outras bibliotecas.