

FRONT-END, LÓGICA E
PROJETO EM UM ÚNICO LUGAR

LÓGICA PARA PROGRAMAR

EDSON DE OLIVEIRA



07

LISTA DE FIGURAS

Figura 1 – Representação de um algoritmo estruturado.....	10
Figura 2 – Percepção dos subalgoritmos.....	11
Figura 3 – Nomeação dos subalgoritmos.....	12
Figura 4 – Separação dos subalgoritmos.....	13
Figura 5 – Funcionamento dos subalgoritmos	14
Figura 6 – Atribuição dos parâmetros	23
Figura 7 – Nomenclatura dos identificadores.....	33
Figura 8 – Erro de compilação por falta de parâmetros	37
Figura 9 – Erro de compilação por excesso de parâmetros.....	41
Figura 10 – Imagem ilustrativa de recursividade.....	51

LISTA DE QUADROS

Quadro 1 – Dia de aula	18
Quadro 2 – Procedimentos: dia de aula.....	20
Quadro 3 – Programa principal: dia de aula com subalgoritmos	20

Edson

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Exemplo de chamada de subalgoritmo	15
Código-fonte 2 – Funcionamento da função max()	16
Código-fonte 3 – Função maior3n()	17
Código-fonte 4 – Sintaxe de procedimento	21
Código-fonte 5 – Criação do subalgoritmo saudacao()	22
Código-fonte 6 – Criação do subalgoritmo saudacao2() com parâmetros	24
Código-fonte 7 – Sintaxe de criação de função	25
Código-fonte 8 – Aplicação da função valor_de_pi()	26
Código-fonte 9 – Solução área do círculo	27
Código-fonte 10 – Solução maiorValor()	28
Código-fonte 11 – Solução anosDeldade()	29
Código-fonte 12 – Solução situacaoAluno()	30
Código-fonte 13 – Solução notaValida()	31
Código-fonte 14 – Função media2n() em Linguagem C	32
Código-fonte 15 – Função media2n() em Python	32
Código-fonte 16 – Função statusAluno() em C++	34
Código-fonte 17 – Função media2n()	35
Código-fonte 18 – Solução media2n()	35
Código-fonte 19 – Solução saudacao() antes de utilizar parâmetros default	36
Código-fonte 20 – Solução saudacao() com parâmetros default	37
Código-fonte 21 – Programa principal com dois parâmetros default	37
Código-fonte 22 – Programa principal com o segundo parâmetro default	38
Código-fonte 23 – Programa principal com o primeiro parâmetro default	38
Código-fonte 24 – Programa principal com os dois parâmetros default	39
Código-fonte 25 – Solução antes de utilizar parâmetros *args	40
Código-fonte 26 – Solução utilizando parâmetros em excesso	41
Código-fonte 27 – Solução utilizando parâmetro *args	42
Código-fonte 28 – Solução notaValida utilizando procedimento	43
Código-fonte 29 – Solução notaValida utilizando função	44
Código-fonte 30 – Aplicação da função notaValida() como função	45
Código-fonte 31 – Solução com função pai e filho	47
Código-fonte 32 – Solução com subalgoritmo encadeado	49
Código-fonte 33 – Solução com subalgoritmo iterativo	52
Código-fonte 34 – Solução com subalgoritmo recursivo	53
Código-fonte 35 – Definição de condição de parada e argumento	54
Código-fonte 36 – Função emocao()	54
Código-fonte 37 – Função emocao() em um arquivo	55
Código-fonte 38 – Função emocao() importada	55
Código-fonte 39 – Função emocao() importada juntamente com outras	56
Código-fonte 40 – Função emocao() importada com todas as outras	56
Código-fonte 41 – Solução múltiplos algoritmos no programa principal	57
Código-fonte 42 – Funções no arquivo biblioteca.py	58
Código-fonte 43 – Execução do arquivo principal.py	58
Código-fonte 44 – Correção do desafio	72

LISTA DE COMANDOS DE PROMPT DO SISTEMA OPERACIONAL

Comando de prompt 1 – Programa de raiz quadrada.....	16
Comando de prompt 2 – Funcionamento da função max().....	16
Comando de prompt 3 – Funcionamento da função maior3n().....	17
Comando de prompt 4 – Execução do subalgoritmo saudacao()	22
Comando de prompt 5 – Execução da atribuição dos parâmetros	23
Comando de prompt 6 – Execução do subalgoritmo saudacao2() com parâmetros	24
Comando de prompt 7 – Execução da função valor_de_pi()	26
Comando de prompt 8 – Execução da solução área do círculo.....	27
Comando de prompt 9 – Execução da Solução maiorValor()	29
Comando de prompt 10 – Execução da solução anosDeldade().....	30
Comando de prompt 11 – Execução da solução situacaoAluno()	30
Comando de prompt 12 – Execução da solução notaValida()	31
Comando de prompt 13 – Execução da solução media2n() – válida para os três códigos-fontes anteriores	35
Comando de prompt 14 – Execução da solução saudação() utilizando dois parâmetros	38
Comando de prompt 15 – Execução da solução saudação() com o segundo parâmetro default	38
Comando de prompt 16 – Execução da solução saudação() com o primeiro parâmetro default	39
Comando de prompt 17 – Execução da solução saudação() com os dois parâmetros default.....	39
Comando de prompt 18 – Introdução a parâmetros *args	41
Comando de prompt 19 – Utilizando parâmetros *args	42
Comando de prompt 20 – Solução notaValida() utilizando procedimento	43
Comando de prompt 21 – Solução notaValida() utilizando função	44
Comando de prompt 22 – Solução notaValida() utilizando função	45
Comando de prompt 23 – Solução com função pai e filho.....	47
Comando de prompt 24 – Execução com notas válidas – subalgoritmo encadeado	49
Comando de prompt 25 – Execução com a primeira nota inválida – subalgoritmo encadeado.....	49
Comando de prompt 26 – Execução com a segunda nota inválida – subalgoritmo encadeado.....	50
Comando de prompt 27 – Execução da solução iterativa.....	52
Comando de prompt 28 – Execução da solução da forma recursiva.....	53
Comando de prompt 29 – Utilizando arquivo de função – forma 1	55
Comando de prompt 30 – Utilizando arquivo de função – forma 2	55
Comando de prompt 31 – Utilizando arquivo de função – forma 3	56
Comando de prompt 32 – Forma iterativa e recursiva	59
Comando de prompt 33 – Os quatro planos de testes do desafio	66

SUMÁRIO

1 LÓGICA PARA PROGRAMAR	8
1.1 É melhor fazer tudo ou por partes?	8
1.2 Visualização da técnica de utilização de subalgoritmo por analogia.....	8
2 SUBALGORITMOS	9
2.1 Definição – subalgoritmos	9
2.2 Vantagens da utilização de subalgoritmos.....	14
2.3 Subalgoritmos nativos e próprios	15
2.3.1 Subalgoritmos nativos	15
2.3.2 Subalgoritmos próprios	17
3 PROCEDIMENTO	18
3.1 Apresente-me tecnicamente o procedimento.....	20
3.2 Procedimento sem parâmetros	21
3.3 Procedimento com parâmetro(s).....	22
4 FUNÇÃO	24
4.1 Função sem parâmetros	26
4.2 Função com parâmetro(s)	27
4.4 Tipo de retorno da função	29
5 VAMOS APRENDER MAIS SOBRE PARÂMETROS?	31
5.1 Formalizando os parâmetros e os tipos das funções.....	31
5.2 Diferença entre parâmetro real e parâmetro formal	32
5.3 “Tipando” os parâmetros	34
5.4 Utilizando parâmetros default.....	36
5.5 Utilizando parâmetros *args	40
6 FUNÇÃO OU PROCEDIMENTO? EIS A QUESTÃO.....	42
7 AGRUPAMENTO DE SUBALGORITMOS	45
7.1 Tal pai, tal filho	46
7.2 Subalgoritmos encadeados, isso existe?	48
7.3 Subalgoritmos recursivos – recursividade.....	50
7.3.1 Solução iterativa	51
7.3.2 Solução recursiva	52
7.3.2.1 Condição de parada / argumento – recursividade	53
8 IMPORTANDO FUNÇÕES DE ARQUIVOS	54
8.1 Importando somente o arquivo.....	54
8.2 Importando o arquivo com os subalgoritmos listados	55
8.3 Importando o arquivo com todos os subalgoritmos.....	56
8.4 Exemplo de aplicação de subalgoritmos em um arquivo	57
9 DESAFIO – CÁLCULOS DAS MÉDIA FIAP PRESENCIAL.....	59
9.1 Definição do desafio	60
9.1.1 Plano de teste “Reprovado” de forma direta	62
9.1.2 Plano de teste “Reprovado” por exame	64
9.1.3 Plano de teste “Aprovado” por exame	65
9.1.4 Plano de teste “Aprovado” direto	66
10 ANEXO – RESOLUÇÃO DO DESAFIO	67

ATIVIDADE – PRATICANDO LÓGICA COM PYTHON.....	73
GLOSSÁRIO	79

FIAP

1 LÓGICA PARA PROGRAMAR

1.1 É melhor fazer tudo ou por partes?

Em nossas vidas, seja no cotidiano, aprendizado ou profissionalmente, estamos acostumados a aprender partindo de algo simples para algo complexo. Em nossa infância, tínhamos dificuldade de saber em qual pé colocar o tênis e um desafio ainda maior era amarrar o cadarço; hoje, a nossa dificuldade é ter de trabalhar para comprar um par de tênis. Sim, tudo evolui! Inclusive a complexidade da programação.

Conforme o aprendizado vai evoluindo, o desenvolvimento de um algoritmo passa a ser cada vez mais complexo. Seguindo essa mesma curva crescente, as técnicas de programação e de aprendizado também evoluem e a técnica que abordaremos neste capítulo é a de utilização de subalgoritmos (funções e procedimentos).

1.2 Visualização da técnica de utilização de subalgoritmo por analogia

Considere o trabalho da secretaria de uma escola. No início do ano, a maior demanda de serviço está na matrícula dos alunos. Dentre os procedimentos que essa tarefa requer, está a organização dos prontuários dos alunos, com os documentos, para que a matrícula seja efetivada.

Em uma escola de ensino médio, há dezoito turmas que se dividem em primeiros, segundos e terceiros anos; todos esses anos são oferecidos nos três períodos (matutino, vespertino e noturno), e cada turno tem a mesma quantidade de turmas/anos: dois primeiros, dois segundos e dois terceiros anos.

Cada turma tem, em média, 30 alunos, assim, teremos 540 alunos se matriculando (lembre-se, 540 prontuários) no fim de janeiro. Se eu me contentar em manipular 540 prontuários, a organização e a manipulação ficarão difíceis; por exemplo, um determinado aluno foi entregar na secretaria um documento que estava devendo e, para atualizar o prontuário desse aluno, a atendente precisou procurar em meio a 540 prontuários.

Um especialista em organização de processos veio à escola e decidiu criar dezoito caixas etiquetadas com o Ano, o Turno e a Turma. Em cada uma dessas caixas, foram guardados os prontuários dos alunos de uma mesma turma. Assim, quando houvesse a necessidade de procurar um prontuário, o procedimento ficaria mais fácil e ágil, com a atendente procurando inicialmente a caixa da turma.

Analogamente, assim é o funcionamento de um subalgoritmo. É muito mais fácil e ágil ordenar 30 prontuários em uma caixa do que 540 em um armário.

2 SUBALGORITMOS

2.1 Definição – subalgoritmos

O prefixo *sub* significa “abaixo de” e, de certa forma, é uma divisão de algo como em capítulo e sub-capítulos. Se algoritmos são linhas de código que resolvem um problema, então subalgoritmos são sub-conjuntos de um algoritmo.

Imagine um algoritmo escrito com diversas linhas, cuja função é efetuar operações financeiras.



Figura 1 – Representação de um algoritmo estruturado
Fonte: Elaborado pelo autor (2022)

Dentre essas linhas, algumas são relacionadas a *entradas de dados do cliente*, outras são relacionadas a *cálculos específicos*, e outras são relacionadas à *exibição das informações processadas*.

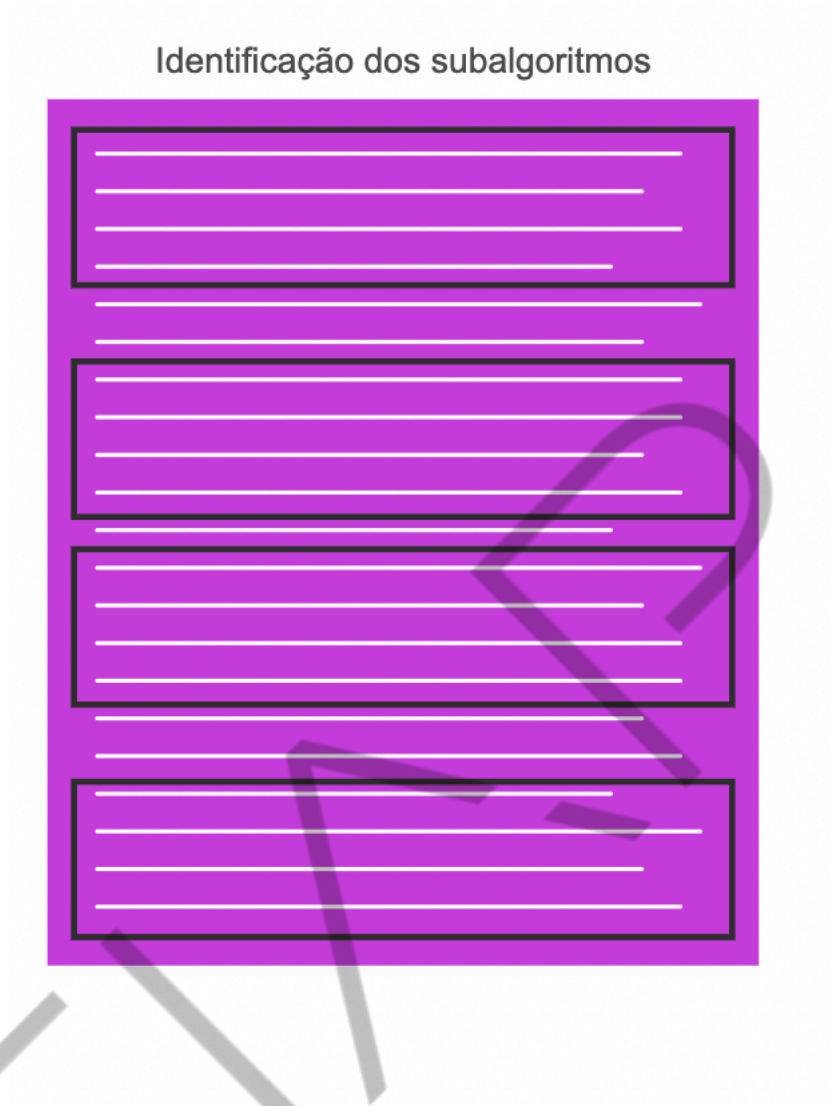


Figura 2 – Percepção dos subalgoritmos
Fonte: Elaborado pelo autor (2022)

Então, por que não criar um subalgoritmo chamado *lerDadosCliente*, outro chamado *calcularSaldo* e um terceiro de nome *exibirExtrato*? Assim, dividimos o grande problema financeiro em partes, e cada uma delas terá um nome, que poderá ser invocado a qualquer momento no programa principal como se fosse um comando.

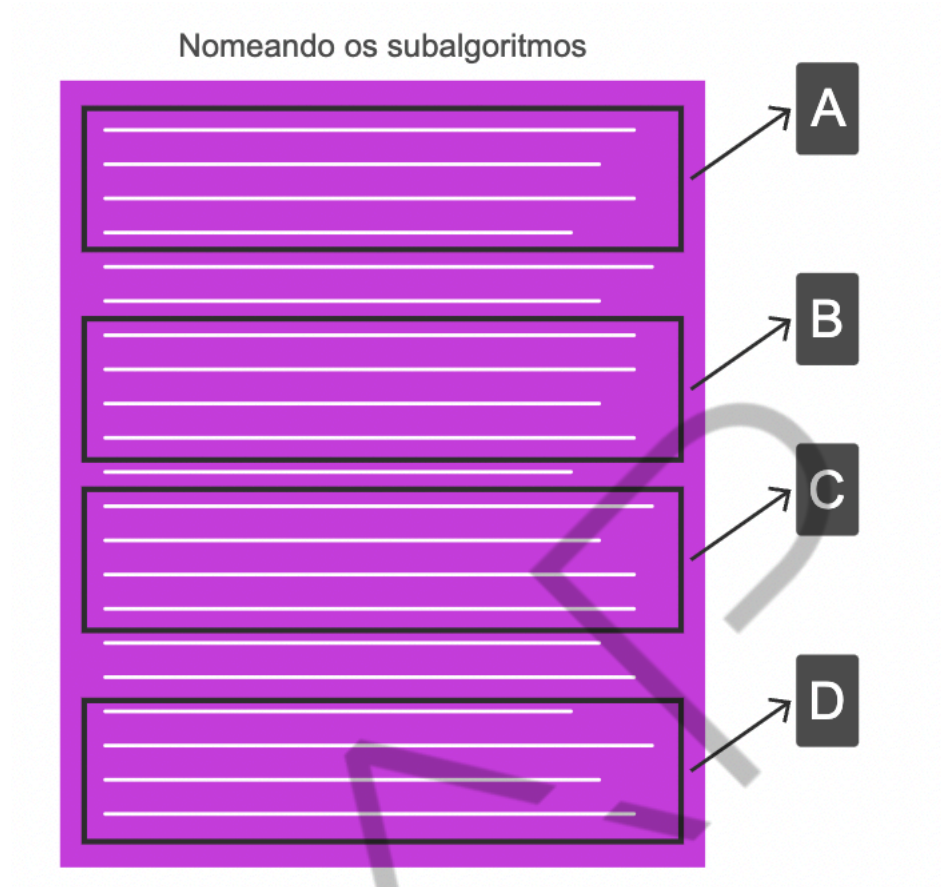


Figura 3 – Nomeação dos subalgoritmos
Fonte: Elaborado pelo autor (2022)

Depois de definidos os subalgoritmos, o ideal é separá-los do programa principal, colocando-os em outro arquivo.

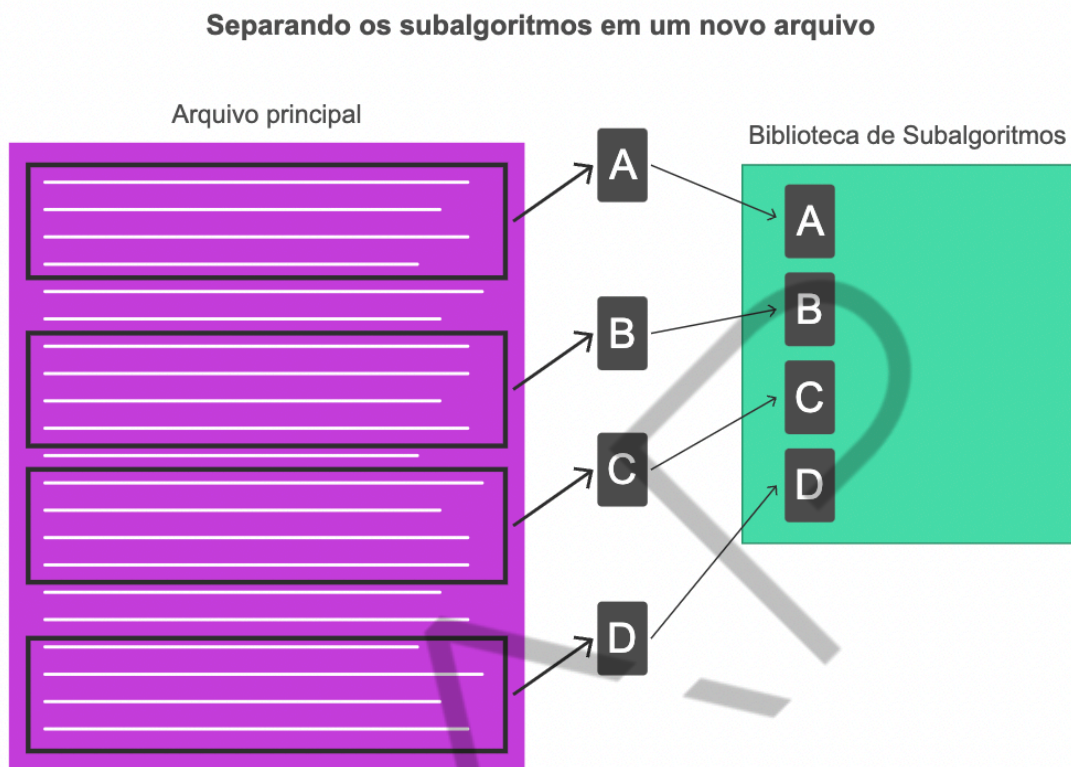


Figura 4 – Separação dos subalgoritmos
Fonte: Elaborado pelo autor (2022)

Basicamente esta é a definição de subalgoritmo: identificar problemas menores, separá-los, nomeá-los e utilizá-los a qualquer momento dentro de uma (ou até outra) solução.

Funcionamento do Algoritmo com Subalgoritmo

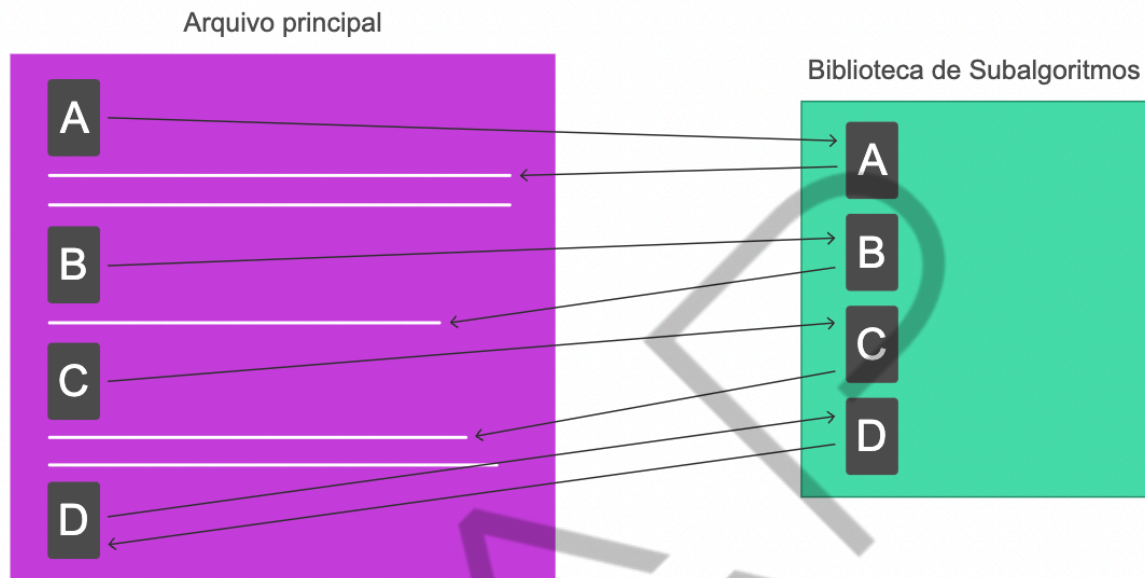


Figura 5 – Funcionamento dos subalgoritmos
Fonte: Elaborado pelo autor (2022)

O mecanismo de funcionamento de um programa com subalgoritmos é o programa principal chamar o subalgoritmo pelo nome, o fluxo do programa sair momentaneamente do principal para executar o subalgoritmo invocado, para, depois, o fluxo voltar ao programa principal na linha da chamada.

2.2 Vantagens da utilização de subalgoritmos

Daí surge a pergunta: “Aprendi a criar algoritmos e consigo resolver os problemas propostos, então por que que devo migrar e aprender subalgoritmos?”.

A resposta que eu te daria – e que você talvez não entenderia – é que subalgoritmos são o princípio de programação orientada a objetos. Como ainda não chegamos a esse nível, vou citar algumas vantagens (considere as palavras algoritmo e programa como sendo sinônimos):

- Separação de linhas de código com objetivos específicos.

- Modularização do programa.
- Reutilização do código no mesmo programa (ou até em outro).
- Objetividade na criação do programa.
- Maior acesso a identificadores locais (como variáveis e parâmetros).
- Manutenção do código.
- Evita a escrita de códigos redundantes.

2.3 Subalgoritmos nativos e próprios

Como assim subalgoritmos próprios ou nativos? Calma, vamos explicar!

2.3.1 Subalgoritmos nativos

Todas as linguagens de programação vêm com centenas de subalgoritmos (vou adiantar um conceito: funções, procedimentos ou métodos) prontos. A estes damos o nome de subalgoritmos nativos, ou seja, estão contidos na instalação da linguagem e estão disponíveis para utilização. Vejamos alguns exemplos:

O programa abaixo pede ao usuário que digite um número e, logo em seguida, ele mostra a sua raiz quadrada:

```
1. import math
2.
3. print("Calculando a raiz quadrada de um número")
4. numero = input("Digite um número: ")
5. numero = float(numero)
6. raizQuadrada = math.sqrt(numero)
7. print(f"A raiz quadrada de {numero} é {raizQuadrada}")
```

Código-fonte 1 – Exemplo de chamada de subalgoritmo
Fonte: Elaborado pelo autor (2022)

Na linha 6, o subalgoritmo (função) `math.sqrt()` é invocado para calcular a raiz quadrada do número passado por parâmetro. Repare que, na linha 1, foi

importada a biblioteca `import math` (falaremos posteriormente sobre biblioteca) onde está contido o subalgoritmo (função) `sqrt()`.

Vejam a execução deste programa:

```
Calculando a raiz quadrada de um número
Digite um número: 16
A raiz quadrada de 16.0 é 4.0
```

Comando de prompt 1 – Programa de raiz quadrada
Fonte: Elaborado pelo autor (2022)

Agora vejamos outro exemplo. Há uma função no Python que retorna o maior valor entre valores fornecidos por parâmetro:

```
print("Exibindo o maior entre 3 números com função nativa")
print(f"Maior valor entre 45, 83 e 33: {max(45,83,33)}")
```

Código-fonte 2 – Funcionamento da função `max()`
Fonte: Elaborado pelo autor (2022)

Dentro da função `max()`, foram inseridos 3 valores aleatórios e ela informou qual é o maior valor. Vejam a execução:

```
Exibindo o maior entre 3 números
Maior valor entre 45, 83 e 33: 83
```

Comando de prompt 2 – Funcionamento da função `max()`
Fonte: Elaborado pelo autor (2022)

Repare que, nesse caso, não foi necessário importar uma biblioteca para o funcionamento da função `max()` porque, nesse caso, o interpretador reconheceu a função sem a necessidade de adicionar uma biblioteca extra.

Nesses dois exemplos, utilizamos subalgoritmos (funções) nativos da linguagem Python. Alguns precisam importar bibliotecas para o seu funcionamento e outros não.

2.3.2 Subalgoritmos próprios

Enquanto os subalgoritmos nativos vêm instalados na máquina, os subalgoritmos próprios são aqueles que o programador (nesse caso, nós) desenvolve no algoritmo de acordo com a necessidade; na verdade este é o sentido deste capítulo: desenvolver os nossos subalgoritmos!

Sem nos atermos ao significado dos códigos, e sim à exemplificação de criação de subalgoritmos, vamos a um exemplo.

Vamos construir um subalgoritmo com o mesmo efeito do `max()`, mas será desenvolvido com o nosso “raciocínio” e ele se chamará **maior3n**:

```
# --- DEFINIÇÃO DO SUBALGORITMO
def maior3n(n1, n2, n3):
    maior = n1
    if n2 > maior:
        maior = n2
    if n3 > maior:
        maior = n3
    return maior

# --- PROGRAMA PRINCIPAL
print("Exibindo o maior entre 3 números com função própria")
print(f"Maior valor entre 45, 83 e 33: {maior3n(45,83,33)}")
```

Código-fonte 3 – Função maior3n()
Fonte: Elaborado pelo autor (2022)

Repare que a execução é idêntica:

```
Exibindo o maior entre 3 números com função própria
Maior valor entre 45, 83 e 33: 83
```

Comando de prompt 3 – Funcionamento da função maior3n()
Fonte: Elaborado pelo autor (2022)

Perceberam a diferença? Então vamos aprender a construir os nossos subalgoritmos!

3 PROCEDIMENTO

Há dois tipos de subalgoritmos: função e procedimento. Neste capítulo, falaremos exclusivamente sobre Procedimento.

Procedimento é uma sequência de códigos que objetivam executar o mesmo processo. Vamos usar um problema do nosso cotidiano para exemplificar:

Eu, Edson de Oliveira, sou professor, e quando vou dar aula presencial para uma turma na FIAP, eu sigo o algoritmo abaixo:

ALGORITMO DIA_DE_AULA_NA_FIAP

- Entro no meu carro
- Dirijo até a FIAP
- Entro no prédio 2
- Estaciono o carro no estacionamento
- Vou até o prédio 1
- Subo até a sala dos professores
- Assino o ponto
- Vou até o prédio 2, no laboratório onde lecionarei
- Abro o laboratório
- Permito a entrada dos alunos
- Ligo a minha máquina
- Insiro o meu usuário
- Copio os slides do dia
- Abro os slides do dia
- Abro uma IDE do Python
- Abro o Teams
- Insiro o meu usuário
- Seleciono a turma
- Crio uma *live*
- Saúdo a turma presencial
- Explico o conteúdo do dia
- Passo um exemplo
- Passo exercícios sobre o conteúdo
- Corrijo os principais exercícios
- Faço a chamada
- Termina a aula

Quadro 1 – Dia de aula
Fonte: Elaborado pelo autor (2022)

Se eu seguir esse algoritmo, conseguirei dar a minha aula do dia. Contudo, com certas sequências de passos, eu posso criar subalgoritmos. Antes de prosseguir, vamos transformar esse algoritmo em subalgoritmos:

```
PROCEDIMENTO IR_ATÉ_FIAP()  
- Entro no meu carro  
- Dirijo até a FIAP  
- Entro no prédio 2  
- Estaciono o carro no estacionamento  
  
PROCEDIMENTO ASSINAR_PONTO()  
- Vou até o prédio 1  
- Subo até a sala dos professores  
- Assino o ponto  
  
PROCEDIMENTO IR_ATÉ_LAB(904)  
- Vou até o prédio 2 e ao laboratório onde  
lecionarei  
- Abro o laboratório  
- Permito a entrada dos alunos  
  
PROCEDIMENTO PREPARAR_O_PC()  
- Ligo a minha máquina  
- Insiro o meu usuário  
- Copio os slides do dia  
- Abro os slides do dia  
- Abro uma IDE do Python  
- Abro o Teams  
- Insiro o meu usuário  
- Seleciono a turma  
- Crio uma live  
  
PROCEDIMENTO MINISTRAR_AULA(ASSUNTO)  
- Saúdo a turma presencial  
- Explico o conteúdo do dia  
- Passo um exemplo  
- Passo exercícios sobre o conteúdo  
- Corrijo os principais exercícios  
- Faço a chamada  
- Termina a aula
```

Quadro 2 – Procedimentos: dia de aula
Fonte: Elaborado pelo autor (2022)

Depois de separar os passos em comum em procedimentos, determinamos um nome para cada procedimento. Então o algoritmo com procedimento implementado ficará assim:

```
ALGORITMO DIA_DE_AULA_NA_FIAP  
IR_ATE_FIAP()  
ASSINAR_PONTO()  
IR_ATE_LAB(904)  
PREPARAR_O_PC()  
MINISTRAR_AULA(ASSUNTO)
```

Quadro 3 – Programa principal: dia de aula com subalgoritmos
Fonte: Elaborado pelo autor (2022)

Viram como o meu algoritmo principal ficou mais enxuto?

Visualizando as vantagens da construção com subalgoritmos, imaginemos que eu passe a ter outro meio de transporte para ir à FIAP: basta atualizar o procedimento IR_ATE_FIAP() [manutenção do código] com os novos meios de transporte. Outra vantagem: imagine que eu esteja ocioso na FIAP e queira ir a um laboratório livre (o 204) só para estudar e/ou montar uma aula, então eu posso utilizar o procedimento IR_ATE_LAB(204) [reutilização do código] para fazer essa tarefa que é diferente de dar aula.

Confie! Utilizar sub--algoritmos torna a construção do algoritmo mais dinâmica e ajuda [muito] na codificação.

3.1 Apresente-me tecnicamente o procedimento

Cada linguagem de programação possui uma forma de tratar o procedimento. O Delphi utiliza a palavra reservada `procedure` no início da criação do procedimento; a linguagem C, Java e C# utilizam o retorno `void` para definir que o subalgoritmo é um procedimento. Mas estamos aqui para aprender os procedimentos em Python.

Vejam os a sua sintaxe:

```
def <nome_do_procedimento/função> ([<parâmetros>]) :  
    <corpo_do_procedimento>
```

Código-fonte 4 – Sintaxe de procedimento
Fonte: Elaborado pelo autor (2022)

Entendendo as terminologias dos sinais da sintaxe:

- Os sinais <> significam “Substitua pelo que está entre <>”; na sintaxe acima, está <nome_do_procedimento>, assim substitua por um nome de procedimento como “ASSINAR_PONTO”.
- O sinal de / significa “Uma coisa ou outra”; na sintaxe acima, está procedimento/função, ou seja, estamos criando um procedimento ou uma função.
- Os sinais de [] significam “Opcional”; na sintaxe acima, está [<parâmetros>], assim, colocar parâmetros é opcional.

3.2 Procedimento sem parâmetros

Como ainda não estamos passando parâmetros, falaremos sobre isso no próximo tópico.

Normalmente os sistemas criam uma mensagem de boas-vindas todas as vezes que um usuário acessa um sistema. Que tal criarmos um procedimento que faça isso? Vamos imaginar que, assim que eu logar no sistema, ele exiba a mensagem: “*Bom dia, Edson de Oliveira! Seja bem-vindo à FIAP*”. Ao invés de toda hora dar um print com essa mensagem, vamos criar um procedimento chamado “saudação”.

```
# DEFINIÇÃO DO PROCEDIMENTO  
def saudacao():  
    print("Bom dia, Edson de Oliveira! Seja bem-vindo à  
    FIAP!")  
  
# CHAMADA DO PROCEDIMENTO NO PROGRAMA PRINCIPAL  
saudacao()
```

Código-fonte 5 – Criação do subalgoritmo `saudacao()`
Fonte: Elaborado pelo autor (2022)

Veja a execução:

```
Bom dia, Edson de Oliveira! Seja bem-vindo à FIAP!
```

Comando de prompt 4 – Execução do subalgoritmo `saudacao()`
Fonte: Elaborado pelo autor (2022)

Nesse caso, depois da palavra `def` colocamos o nome do procedimento, que é `saudacao`. Como não utilizamos parâmetros, então não colocamos nada entre os parênteses. Depois, criamos o corpo do procedimento, que contém apenas a mensagem de boas-vindas. Por fim, invocamos o procedimento no programa principal. Teste na sua máquina e veja se funciona!

O Python permite que nomes de variáveis e/ou subalgoritmos tenham acentuação, diferentemente de outras linguagens. Particularmente prefiro não utilizar, logo, o meu procedimento *saudação* foi denominado *saudacao*.

3.3 Procedimento com parâmetro(s)

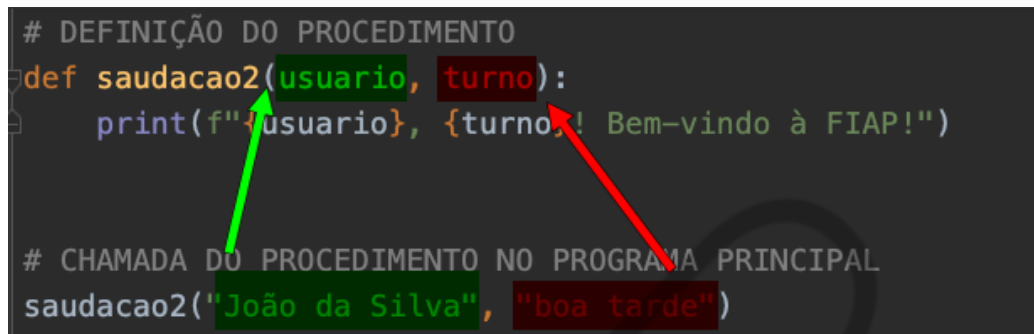
Sobre o exemplo anterior, você deve ter se perguntado: “Esse procedimento é mecânico porque exibiu ‘Bom dia’ ao ‘Edson de Oliveira’”. Então, essa mensagem só serviria para o *Edson* durante a parte da *manhã*. E se fosse em outro período para outra pessoa?”. Boa pergunta! Para resolvermos esse problema, precisamos utilizar parâmetros.

Parâmetros são informações passadas pelo programa principal e que são transportadas ao subalgoritmo até os seus parâmetros formais. Este usa os parâmetros para algum fim em seu corpo.

Neste novo exemplo, utilizaremos parâmetros. Vamos chamar esse novo procedimento de `saudacao2`.

Dentro do mesmo programa cada subalgoritmo deve ter um nome diferente para não haver duplicidade, assim como ocorre com variáveis.

Construindo e executando o procedimento *saudacao2*:



```
# DEFINIÇÃO DO PROCEDIMENTO
def saudacao2(usuario, turno):
    print(f"{usuario}, {turno}! Bem-vindo à FIAP!")

# CHAMADA DO PROCEDIMENTO NO PROGRAMA PRINCIPAL
saudacao2("João da Silva", "boa tarde")
```

Figura 6 – Atribuição dos parâmetros
Fonte: Elaborado pelo autor (2022)

Nesse exemplo, estamos utilizando parâmetros. Veja a execução:

```
Boa tarde, João da Silva! Seja bem-vindo à FIAP!
```

Comando de prompt 5 – Execução da atribuição dos parâmetros
Fonte: Elaborado pelo autor (2022)

A informação “João da Silva” foi transportada para o parâmetro usuário, enquanto a informação “Boa tarde” foi transportada para o parâmetro turno. Utilizando parâmetros, o procedimento ficou customizável, ou seja, ele passou a servir para qualquer nome ou turno passados por parâmetro, diferentemente do exemplo anterior.

Vamos melhorar ainda mais! Para isso, deixaremos o subalgoritmo “inteligente” e ele decidirá se é “bom dia”, “boa tarde” ou “boa noite”? Como isso? Ao invés de passar o parâmetro turno, vamos passar o parâmetro hora; dependendo da hora, ele vai decidir a mensagem que será exibida. Vamos ao procedimento:

```
# DEFINIÇÃO DO PROCEDIMENTO
def saudacao2(usuario, hora):
    if hora < 12:
        turno = "Bom dia"
    elif hora < 18:
        turno = "Boa tarde"
    else:
        turno = "Boa noite"
    print(f"{usuario}, {turno}! Seja bem-vindo à
FIAP!")

# CHAMADA DO PROCEDIMENTO NO PROGRAMA PRINCIPAL
saudacao2("Marcelo Marques", 19)
```

Código-fonte 6 – Criação do subalgoritmo saudacao2() com parâmetros
Fonte: Elaborado pelo autor (2022)

Reparem que agora o parâmetro *turno* foi trocado pelo parâmetro *hora*, e *turno* se tornou uma variável local no procedimento.

Variável local é aquela definida dentro do corpo de um subalgoritmo.

Vejam como ficou a execução:

```
Boa noite, Marcelo Marques! Seja bem-vindo à FIAP!
```

Comando de prompt 6 – Execução do subalgoritmo saudacao2() com parâmetros
Fonte: Elaborado pelo autor (2022)

Dessa forma, vimos a importância da utilização de parâmetros em subalgoritmos.

4 FUNÇÃO

Função tem o mesmo significado que procedimento, ou seja, também é uma sequência de códigos cujo objetivo é executar o mesmo processo, mas função tem o

diferencial de **retornar um valor** para o programa chamador (pode ser um algoritmo ou outro subalgoritmo).

Vamos exemplificar com algo corriqueiro para o professor em uma aula: fazer chamada. Considere que, para isso, os passos são:

```
PROCEDIMENTO EFETUAR_CHAMADA (TURMA)
- Abrir o portal
- Digitar o login e a senha
- Selecionar a turma
- Fazer a chamada
```

Até o momento nenhuma novidade, porque fizemos um procedimento. Porém eu pergunto: E se quiséssemos saber quantos alunos faltaram? Então o procedimento não servirá mais, porque o problema mudou de figura e ele passa a ser resolvido pelo conceito de função, uma vez que ele precisa de uma resposta, ou um retorno – a quantidade de alunos faltantes –, ao algoritmo que o invocou.

Veja como ficaria o conceito da função:

```
FUNÇÃO CONTAR_FALTANTES (TURMA)
- Selecionar a turma
- Contar a quantidade de alunos faltantes
- RETORNAR a quantidade de faltantes
```

Depois de executada essa função, o valor retornado servirá para algum fim no programa principal.

Tecnicamente, a função possui a seguinte sintaxe no Python:

```
def <nome_da_função>([<parâmetros>]):
    <corpo_da_função>
    return <valor_retornado>
```

Código-fonte 7 – Sintaxe de criação de função
Fonte: Elaborado pelo autor (2022)

Comparando com procedimento, a função tem o aditivo do comando **return** em algum ponto de seu corpo, para nele retornar o valor ao algoritmo solicitante.

4.1 Função sem parâmetros

Apesar de pouco usual, é possível criarmos uma função que não passe parâmetros. Vamos imaginar que algum algoritmo necessite realizar cálculos que envolvam a área de um círculo. Uma referência bastante utilizada nessas equações é o valor de PI (3,14159).

Podemos então criar uma função chamada `valor_de_pi()` e toda vez que ela for invocada, retornará o número fracionário correspondente ao valor de PI.

Vejam a construção da função e sua utilização:

```
# DEFINIÇÃO DA FUNÇÃO
def valor_de_pi() :
    return 3.14159

# CHAMADA DA FUNÇÃO NO PROGRAMA PRINCIPAL
print(f"PI = {valor_de_pi()}")
```

Código-fonte 8 – Aplicação da função `valor_de_pi()`
Fonte: Elaborado pelo autor (2022)

Visão do prompt ao executar o código acima:

```
PI = 3.14159
```

Comando de prompt 7 – Execução da função `valor_de_pi()`
Fonte: Elaborado pelo autor (2022)

Repare que uma função é escrita acompanhada de algo no programa principal (nesse exemplo de um `print`) na linha da sua chamada enquanto um procedimento é escrito isoladamente em uma linha.

Utilizando a função `valor_de_pi()`, também podemos calcular a área de um círculo. Para efetuar esse cálculo, é necessário saber o raio do círculo. Então vamos criar uma solução para esse problema:

```
# DEFINIÇÃO DA FUNÇÃO
def valor_de_pi():
    return 3.14159

# CHAMADA DA FUNÇÃO NO PROGRAMA PRINCIPAL DENTRO DE UM CÁLCULO
raio = float(input("Digite o raio: "))
area_circulo = valor_de_pi() * raio ** 2
print(f"Área do círculo com raio {raio} é {area_circulo}")
```

Código-fonte 9 – Solução área do círculo

Fonte: Elaborado pelo autor (2022)

Nesse caso, a função `valor_de_pi()` foi utilizada como parte do valor na equação da área do círculo.

Vejam a execução:

```
Digite o raio: 14
Área do círculo com raio 14.0 é 615.75164
```

Comando de prompt 8 – Execução da solução área do círculo

Fonte: Elaborado pelo autor (2022)

4.2 Função com parâmetro(s)

Esta é a forma mais usual de utilizarmos funções: passando parâmetros.

Lembram da função própria `math.sqrt(16)` que retornava a raiz quadrada de 16? Então, ela é uma função que passa um parâmetro, nesse caso, o 16, e retorna 4, a sua raiz quadrada, contudo ela é uma função nativa do Python. Ela pode até servir como auxílio, mas, nesta Fase, o objetivo é aprendermos a construir as nossas próprias funções.

Para efeito de comparação, vamos construir uma função chamada `maiorValor()`, que terá o mesmo efeito da função `max()` – retorna o maior dentre os valores fornecidos –, para que possamos verificar no mesmo exemplo uma função nativa e uma própria.

Veja o código abaixo:

```
1. # ----- DEFINIÇÃO DOS SUBALGORITMOS
2. def maiorValor(n1, n2, n3):
3.     maior = n1
4.     if n2 > maior:
5.         maior = n2
6.     if n3 > maior:
7.         maior = n3
8.     return maior
9.
10. # ----- PROGRAMA PRINCIPAL
11. print("Digite 3 números: ")
12. n1 = float(input(""))
13. n2 = float(input(""))
14. n3 = float(input(""))
15. print(f"Resp. c/ a função 'max()'.....: {max(n1, n2, n3)}")
16. print(f"Resp. c/ a função 'maiorValor()'...: {maiorValor(n1, n2, n3)}")
```

Código-fonte 10 – Solução maiorValor()
Fonte: Elaborado pelo autor (2022)

Das linhas 2 a 8, temos a construção da função `maiorValor()`. Na linha 8, está o retorno da função com o maior valor. Na linha 15, temos a chamada da função nativa `max()`, e na linha 16, a chamada da função própria `maiorValor()`.

Vejam que, na execução, ambas retornaram o mesmo valor:

```
Digite 3 números:
23
65
12
Resposta com função nativa 'max()'.....: 65.0
Resposta com função própria 'maiorValor()'...: 65.0
```

Comando de prompt 9 – Execução da Solução maiorValor()
Fonte: Elaborado pelo autor (2022)

4.4 Tipo de retorno da função

Sabemos que toda função retorna um valor e esse valor é de um tipo. Apesar do procedimento não retornar valor, podemos dizer que ele retorna `None` (nada).

Nas funções construídas acima, todas retornaram um valor `float`. Então você me pergunta: “Existem funções com retorno de outros tipos?”. Sim!

Vamos passar alguns exemplos para demonstrar o funcionamento de funções de outros tipos:

Retorno inteiro (int) – considere que seja passado como parâmetro o ano de nascimento de alguém e a função retorne a idade (que é um número inteiro) da pessoa:

```
# ----- DEFINIÇÃO DOS SUBALGORITMOS
def anosDeIdade(anoNascimento) -> int:
    idade = 2022 - anoNascimento
    return idade

# ----- PROGRAMA PRINCIPAL
ano = int(input("Digite o ano do seu nascimento:"))
print(f"Você tem {anosDeIdade(ano)} anos.")
```

Código-fonte 11 – Solução anosDeIdade()
Fonte: Elaborado pelo autor (2022)

Veja a execução:

```
Digite o ano do seu nascimento: 1974
Você tem 48 anos.
```

Comando de prompt 10 – Execução da solução anosDeldade()
Fonte: Elaborado pelo autor (2022)

Retorno String (str) – considere que seja passado como parâmetro uma média da FIAP, e a função deverá retornar uma string “Aprovado” ou “Reprovado” de acordo com a média (6).

Veja como fica:

```
# ----- DEFINIÇÃO DOS SUBALGORITMOS
def situacaoAluno(m) -> str:
    if m < 6:
        return "Reprovado"
    else:
        return "Aprovado"

# ----- PROGRAMA PRINCIPAL
media = float(input("Digite a média anual:"))
print(f"Você está {situacaoAluno(media)} com média {media}.")
```

Código-fonte 12 – Solução situacaoAluno()
Fonte: Elaborado pelo autor (2022)

Segue a execução:

```
Você está aprovado com média 9.0.
```

Comando de prompt 11 – Execução da solução situacaoAluno()
Fonte: Elaborado pelo autor (2022)

Retorno Booleano (bool) – considere que será passado como parâmetro uma eventual nota. A função vai verificar se essa nota é válida ou não, retornando True ou False, respectivamente.

Segue o código:

```
# ----- DEFINIÇÃO DOS SUBALGORITMOS
def notaValida(n) -> bool:
    if n >= 0 and n <= 10:
        return True
    else:
        return False

# ----- PROGRAMA PRINCIPAL
nota = float(input("Digite uma nota: "))
if notaValida(nota):
    print(f"A nota {nota} é válida.")
else:
    print(f"A nota {nota} é inválida.")
```

Código-fonte 13 – Solução notaValida()
Fonte: Elaborado pelo autor (2022)

Segue a execução:

```
Digite uma nota: 9
A nota 9.0 é válida.
```

Comando de prompt 12 – Execução da solução notaValida()
Fonte: Elaborado pelo autor (2022)

5 VAMOS APRENDER MAIS SOBRE PARÂMETROS?

Parâmetros são recursos importantes para a utilização de subalgoritmos, seja procedimento ou função (ou futuramente métodos), até então empregamos o básico de parâmetros. Neste capítulo, apresentaremos outros recursos importantes que os parâmetros podem nos fornecer para otimizar a construção de nossas soluções.

5.1 Formalizando os parâmetros e os tipos das funções

Em quase todas as linguagens de programação, os parâmetros e os retornos são explicitamente definidos, por exemplo, se em linguagem C quiséssemos criar uma função que retorne a média de dois números, ficaria assim:

```
float media2n(float n1, float n2)
{
    return (n1 + n2) / 2;
}
```

Código-fonte 14 – Função media2n() em Linguagem C
Fonte: Elaborado pelo autor (2022)

Repare que, antes do nome da função, há um **float** que simboliza o tipo de dado que será retornado pela função. Antes de cada parâmetro, é utilizado o **float**, que significa o tipo do parâmetro.

Em linguagem C, é obrigatório utilizar esses conceitos, caso contrário, ocorrerá um erro de compilação.

No Python, não há a obrigatoriedade da definição dos tipos, eles servem mais para documentação da função do código (como o UML prega). Caso você queira utilizar tipagem nos subalgoritmos em Python, ficará assim:

```
def media2n(n1: float, n2: float) -> float:
    return (n1 + n2) / 2
```

Código-fonte 15 – Função media2n() em Python
Fonte: Elaborado pelo autor (2022)

Os **floats** ao lado de n1 e n2 indicam que os parâmetros devem ser do tipo float, enquanto o último **float**, ao lado de ->, significa que a função retornará um dado float.

Diferentemente da Linguagem C, o Python aceita que se coloque os parâmetros entre vírgulas e se defina o tipo uma vez para todos, como segue:

```
def media2n(n1, n2: float) -> float:
```

5.2 Diferença entre parâmetro real e parâmetro formal

Quando escrevemos os algoritmos juntamente com os subalgoritmos, utilizamos diversas nomenclaturas que acabam gerando confusão. Cada uma tem um significado.

Vamos esclarecer essas nomenclaturas com um exemplo:

```
# ----- DEFINIÇÃO DOS SUBALGORITMOS
def maiorValor(n1: float, n2: float, n3: float) -> float:
    maior = n1
    if n2 > maior:
        maior = n2
    if n3 > maior:
        maior = n3
    return maior

# ----- PROGRAMA PRINCIPAL
print("Digite 3 números: ")
num1 = float(input(""))
num2 = float(input(""))
num3 = float(input(""))
print(f"Resposta com a função própria 'maiorValor()': {maiorValor(num1, num2, num3)}")
```

Figura 7 – Nomenclatura dos identificadores

Fonte: Elaborado pelo autor (2022)

Nesse contexto, temos um programa principal e uma função sendo executados.

Segue o significado de cada termo:

- **Parâmetros formais**: são aqueles definidos na escrita dos subalgoritmos, entre parênteses, ao lado do nome da função; no exemplo, os parâmetros formais são `n1`, `n2` e `n3` (em verde). Eles representam como os parâmetros serão reconhecidos dentro da função, independentemente dos valores fornecidos pelo programa principal via parâmetros reais.
- **Tipo de retorno da função**: uma função sempre retorna um valor, esse valor é definido ao lado de `->` (em azul). Caso seja um procedimento, podemos colocar `None`, ou seja, não retorna nada.
- **Variável local**: ela (em amarelo) só existe quando utilizada dentro do escopo do subalgoritmo e enquanto o subalgoritmo é executado. No programa principal, a variável local deixa de existir.
- **Variável global**: ela (em vermelho) existe em todos os pontos (escopo) do algoritmo ou subalgoritmo.
- **Parâmetros reais**: são aqueles colocados entre parênteses na função no escopo do algoritmo principal (em branco), ou seja, são aqueles que “REALmente” serão processados pela solução.

5.3 “Tipando” os parâmetros

Em outras linguagens de programação (como C, C++, Java, Delphi...), os parâmetros formais são “tipados” obrigatoriamente – como o Word grifou em vermelho a palavra “tipados”, por não a conhecer, considere o verbo “tipar” como a arte de dar um tipo a um identificador –, enquanto em outras linguagens (como Python e JavaScript) não há essa necessidade.

Particularmente em Python, essa tipagem serve mais para uma boa documentação do código-fonte (similar ao UML). Assim, quem o ler entenderá qual é o tipo dos elementos da função, para o seu bom andamento.

Vou utilizar o mesmo exemplo em duas linguagens para que você faça um comparativo. Considere que sejam passadas duas notas por parâmetro e a função deve retornar, depois do cálculo da média, uma string dizendo se o aluno está “Aprovado” ou “Reprovado”.

Vejamos como funciona em C++:

```
string statusAluno(float nota1, float nota2)
{
    float media;
    media = (n1 + n2) / 2;
    if (media >= 6)
        return ("Aprovado");
    else
        return ("Reprovado");
}
```

Código-fonte 16 – Função statusAluno() em C++
Fonte: Elaborado pelo autor (2022)

Usei o conceito de variável local para melhor expressar a situação em C++. Veja que todos os parâmetros estão acompanhados obrigatoriamente de um tipo (inclusive a função) e, uma vez desrespeitado esse tipo na passagem real dos parâmetros, um erro poderá ser gerado.

Segue a mesma função em Python **sem** definir os tipos:

```
def media2n(nota1, nota2):  
    media = (nota1 + nota2) / 2  
    if media >= 6:  
        return "Aprovado"  
    else:  
        return "Reprovado"
```

Código-fonte 17 – Função media2n()
Fonte: Elaborado pelo autor (2022)

Essa função criada em Python tem a mesma execução da criada em C++, porém não define os tipos.

Vamos reescrever essa função em Python definindo tipos para os parâmetros, variável e função:

```
def media2n(nota1: float, nota2: float) -> str:  
    media: float  
    media = (nota1 + nota2) / 2  
    if media >= 6:  
        return "Aprovado"  
    else:  
        return "Reprovado"  
  
print(f"Situação: {media2n(7, 8)}")
```

Código-fonte 18 – Solução media2n()
Fonte: Elaborado pelo autor (2022)

Execução:

```
Situação: Aprovado
```

Comando de prompt 13 – Execução da solução media2n() – válida para os três códigos-fontes anteriores

Fonte: Elaborado pelo autor (2022)

Nessa nova versão da função **float**, são definidos os tipos do parâmetro; **str** é o tipo que a função retornará, e **float** é o tipo da variável local utilizada na função. Lembrando que, em Python a tipagem serve apenas como documentação, utilizando ou não, respeitando o tipo de dado ou não, o subalgoritmo continuará funcionando.

A partir daqui, “tiparemos” todos os subalgoritmos para um melhor entendimento de sua concepção.

5.4 Utilizando parâmetros default

Até o momento, sempre que utilizamos parâmetros nos subalgoritmos, respeitamos “religiosamente” a quantidade de parâmetros fornecidos e recebidos (que deve ser a mesma) e os tipos dos parâmetros.

Então, você me pergunta: “E se desrespeitarmos essas equivalências, vai dar ruim?”. Sim! e Não!

Sim! Se você utilizar os conceitos vistos até aqui.

Não! Se você utilizar parâmetros default.

Para exemplificarmos o conceito de parâmetros default, vamos utilizar um procedimento já conhecido por nós.

O procedimento “saudação” passa como parâmetro o nome da pessoa e a hora, lembram?

Primeiramente vamos executar o procedimento sem parâmetros default:

```
# ----- DEFINIÇÃO DOS SUBALGORITMOS
def saudacao(usuario: str, hora: int) -> None:
    if hora < 12:
        turno = "Bom dia"
    elif hora < 18:
        turno = "Boa tarde"
    else:
        turno = "Boa noite"
    print(f"{usuario}, {turno}! Seja bem-vindo à FIAP!")

# ----- PROGRAMA PRINCIPAL
saudacao("Marcelo Marques")
```

Código-fonte 19 – Solução saudacao() antes de utilizar parâmetros default
Fonte: Elaborado pelo autor (2022)

Reparem que o procedimento espera dois parâmetros, mas somente “Marcelo Marques” é passado. A consequência disso é:

```
Traceback (most recent call last):  
  File "/Users/edsondeoliveira/Desktop/fontes/python/estudo/ES0/capitulo4.py", line 12, in <module>  
    saudacao("Marcelo Marques")  
TypeError: saudacao() missing 1 required positional argument: 'hora'
```

Figura 8 – Erro de compilação por falta de parâmetros
Fonte: Elaborado pelo autor (2022)

Houve um erro de compilação na linha 12, que diz: “Faltando 1 parâmetro”.

Para solucionar esse problema, podemos utilizar parâmetros default. Como fazemos isso? No parâmetro formal, atribua a cada parâmetro um valor padrão. Assim, caso não seja fornecido um ou mais dos valores, ele usará o padrão (com marcação escura). Veja:

```
# ----- DEFINIÇÃO DOS SUBALGORITMOS  
def saudacao(usuario = 'Edson', hora = 8) -> None:  
    if hora < 12:  
        turno = "Bom dia"  
    elif hora < 18:  
        turno = "Boa tarde"  
    else:  
        turno = "Boa noite"  
    print(f"{usuario}, {turno}! Seja bem-vindo à FIAP!")
```

Código-fonte 20 – Solução saudacao() com parâmetros default
Fonte: Elaborado pelo autor (2022)

Também funcionaria se fosse colocado:

```
def saudacao(usuario: str = 'Edson', hora: int = 8) -> None:
```

Se passarmos dois parâmetros reais no procedimento, ambos serão exibidos como sempre ocorreu, nada vai mudar:

```
# ----- PROGRAMA PRINCIPAL  
# Passando os dois parâmetros  
saudacao("Marcelo Marques", 13)
```

Código-fonte 21 – Programa principal com dois parâmetros default
Fonte: Elaborado pelo autor (2022)

A execução ficou:

```
Boa tarde, Marcelo Marques! Seja bem-vindo à FIAP!
```

Comando de prompt 14 – Execução da solução saudação() utilizando dois parâmetros
Fonte: Elaborado pelo autor (2022)

Se passarmos somente o primeiro parâmetro, na execução do procedimento o segundo parâmetro assumirá o valor default 8 (que representa “Bom dia”):

```
# ----- PROGRAMA PRINCIPAL
# Passando somente o primeiro parâmetro
saudacao("Camila Guedes")
```

Código-fonte 22 – Programa principal com o segundo parâmetro default
Fonte: Elaborado pelo autor (2022)

A execução ficou assim:

```
Bom dia, Camila Guedes! Seja bem-vinda à FIAP!
```

Comando de prompt 15 – Execução da solução saudação() com o segundo parâmetro default
Fonte: Elaborado pelo autor (2022)

É possível passarmos somente o segundo parâmetro? Sim, é possível, mas nesse caso a escrita fica um pouco diferente:

```
# ----- PROGRAMA PRINCIPAL
# Passando o segundo parâmetro
saudacao(hora = 19)
```

Código-fonte 23 – Programa principal com o primeiro parâmetro default
Fonte: Elaborado pelo autor (2022)

Para identificarmos um parâmetro específico – neste caso, o segundo parâmetro (também serviria para o primeiro) –, devemos atribuir a ele um conteúdo. Então temos que colocar o nome do parâmetro que será modificado e ao qual será atribuído o valor desejado, enquanto o outro permanecerá com o valor default.

Vejam a execução do código acima:

```
Boa noite, Edson! Seja bem-vindo à FIAP!
```

Comando de prompt 16 – Execução da solução saudação() com o primeiro parâmetro default
Fonte: Elaborado pelo autor (2022)

Seria possível não passar parâmetros? Sim! Se não passarmos parâmetros reais e o subalgoritmo prever parâmetros default, não haverá erro de compilação. Os dois parâmetros considerados na execução serão default.

Segue o código:

```
# ----- PROGRAMA PRINCIPAL
# Não passando parâmetros
saudacao()
```

Código-fonte 24 – Programa principal com os dois parâmetros default
Fonte: Elaborado pelo autor (2022)

A execução ficará desta forma:

```
Bom dia, Edson! Seja bem-vindo à FIAP!
```

Comando de prompt 17 – Execução da solução saudação() com os dois parâmetros default
Fonte: Elaborado pelo autor (2022)

Os parâmetros default funcionam dessa forma. Eles são uma boa solução para os casos em que pode ocorrer de não termos todos os parâmetros reais.

5.5 Utilizando parâmetros *args

Quando construímos subalgoritmos são sistemáticos para que possa ser devidamente executado. Seja o compilador ou o interpretador, antes de ele executar o subalgoritmo, ele confere da seguinte forma o subalgoritmos chamado:

- O nome do subalgoritmo é o mesmo?
- A quantidade de parâmetros é a mesma?
- Os tipos dos parâmetros são os mesmos?

O mecanismo de chamada do subalgoritmo funciona dessa forma, certo? Sim, mas Não!

O Sim é a resposta para o que vimos até o tópico anterior e o Não é o que veremos a partir desse tópico ao conhecermos os parâmetros *args.

Primeiramente, o nome *args vem de argumentos, um sinônimo de parâmetros. Esse nome não é obrigatório, e sim uma convenção (boa prática). Poderíamos chamá-lo de *casa ou *lápiz, por exemplo.

Para entendermos esse conceito, vamos utilizar um exemplo simples a partir de uma função “normal” (que vimos até então). Considere que precisamos de uma função que calcule a soma de três números, ficaria assim:

```
# ----- DEFINIÇÃO DOS SUBALGORITMOS
def somaNumeros(n1: float, n2: float, n3: float) -> float:
    return n1 + n2 + n3

# ----- PROGRAMA PRINCIPAL
somatoria = somaNumeros(45,23,11)
print(f"Soma = {somatoria}")
```

Código-fonte 25 – Solução antes de utilizar parâmetros *args
Fonte: Elaborado pelo autor (2022)

Execução:

```
Soma = 79
```


Comando de prompt 18 – Introdução a parâmetros *args
Fonte: Elaborado pelo autor (2022)

Até aí nenhum problema, foram somados os números 45, 23 e 11, resultando 79 como esperado. O problema é: E se eu precisar somar mais (ou menos) de três números? Haverá problema, veja como o algoritmo se comportará:

```
# ----- DEFINIÇÃO DOS SUBALGORITMOS
def somaNumeros(n1: float, n2: float, n3: float) -> float:
    return n1 + n2 + n3

# ----- PROGRAMA PRINCIPAL
somatoria = somaNumeros(45, 23, 11, 56, 98)
print(f"Soma = {somatoria}")
```

Código-fonte 26 – Solução utilizando parâmetros em excesso
Fonte: Elaborado pelo autor (2022)

Reparem que, agora, acrescentamos os números 56 e 98 na chamada da função via parâmetros reais, e o que aconteceu?

```
Traceback (most recent call last):
  File "/Users/edsondeoliveira/Desktop/fontes/python/estudo/ES0/capitulo4.py", line 6, in <module>
    somatoria = somaNumeros(45, 23, 11, 56, 98)
TypeError: somaNumeros() takes 3 positional arguments but 5 were given
```

Figura 9 – Erro de compilação por excesso de parâmetros
Fonte: Autor (2022)

Um erro de compilação que basicamente diz: “A função tem 3 parâmetros e foram passados 5”.

Utilizando o parâmetro *args, resolvemos esse problema.

Veja a nova construção com essa utilização:

```
# ----- DEFINIÇÃO DOS SUBALGORITMOS
def somaNumerosComArgs(*args) -> float:
    soma = 0
    for num in args:
        soma += num
    return soma

# ----- PROGRAMA PRINCIPAL
somatoria = somaNumerosComArgs(45, 23, 11, 56, 98)
print(f"Soma = {somatoria}")
```

Código-fonte 27 – Solução utilizando parâmetro *args
Fonte: Elaborado pelo autor (2022)

Reparem que o parâmetro formal da função ficou com *args. A partir de agora, não importa a quantidade de valores que a função passe nos parâmetros reais, a soma será efetuada.

Veja a execução:

```
Soma = 233
```

Comando de prompt 19 – Utilizando parâmetros *args
Fonte: Elaborado pelo autor (2022)

Esta é uma grande vantagem, dependendo do contexto da função. Caso o número de parâmetros esteja bem definido, utilize a forma clássica de construção de funções; caso o número de parâmetros seja volúvel, utilize *args.

Há outro tipo de parâmetro similar chamado **kwargs, mas não o abordaremos neste capítulo porque precisamos saber como funcionam os “dicionários”. Então, aguarde o próximo capítulo.

6 FUNÇÃO OU PROCEDIMENTO? EIS A QUESTÃO

O maior problema para o aluno quando ele está aprendendo subalgoritmos é saber quando ele deve utilizar Funções ou Procedimentos.

Muitas situações se resolvem tanto com Procedimento quanto com Função, mas sempre um dos dois é o mais adequado.

Antes de prosseguir com a explicação, vou dar um exemplo de construção de um problema utilizando tanto função quanto procedimento.

Considere que seja passado como parâmetro uma eventual nota e o subalgoritmo deverá verificar se essa nota é válida (entre 0 e 10 inclusive).

Primeira resolução, utilizando procedimento:

```
# ----- UTILIZANDO PROCEDIMENTO
def notaValida(nota: float) -> None:
    if nota >= 0 and nota <= 10:
        print(f"A nota {nota} é válida.")
    else:
        print(f"A nota {nota} é inválida.")

# ----- PROGRAMA PRINCIPAL
# Chamando o procedimento
num = 7
    notaValida(num)
```

Código-fonte 28 – Solução notaValida utilizando procedimento
Fonte: Elaborado pelo autor (2022)

A execução será:

```
A nota 7 é válida.
```

Comando de prompt 20 – Solução notaValida() utilizando procedimento
Fonte: Elaborado pelo autor (2022)

Agora, vamos resolver o mesmo problema com função:

```
# ----- UTILIZANDO FUNÇÃO
def notaValida(nota: float) -> bool:
    # esta expressão analisará a condição e resultará True ou False
    return nota >= 0 and nota <= 10

# ----- PROGRAMA PRINCIPAL
num = 7
resposta = notaValida(num)
if resposta == True: # Ou simplesmente 'if resposta:'
    print(f"A nota {num} é válida.")
else:
    print(f"A nota {num} é inválida.")
```

Código-fonte 29 – Solução notaValida utilizando função
Fonte: Elaborado pelo autor (2022)

A execução será a mesma:

A nota 7 é válida.

Comando de prompt 21 – Solução notaValida() utilizando função
Fonte: Elaborado pelo autor (2022)

Nesse caso, qual seria a melhor resolução? A resolução com função ou a resolução com procedimento?

A resolução com função! No caso da função, ficou registrado na variável resposta o valor lógico True ou False sobre a nota ser válida ou não.

Se temos esse valor armazenado, conseguimos tratar essa falha com um laço dizendo: “Continue solicitando a nota até que ela seja válida”, veja:

```
# ----- UTILIZANDO FUNÇÃO
def notaValida(nota: float) -> bool:
    # será analisada a expressão e resultará True ou False
    return nota >= 0 and nota <= 10

# ----- PROGRAMA PRINCIPAL
num = float(input("Nota: "))
while not notaValida(num) :
    print(f"{num} é uma nota Inválida!")
    num = float(input("\nDigite uma nota entre 0 e 10: "))
print(f"Agora você digitou uma nota válida: {num}")
```

Código-fonte 30 – Aplicação da função `notaValida()` como função
Fonte: Elaborado pelo autor (2022)

Veja a execução:

```
Nota: 12
12.0 é uma nota inválida!
Digite uma nota entre 0 e 10: 11
11.0 é uma nota inválida!
Digite uma nota entre 0 e 10: -2
-2.0 é uma nota inválida!
Digite uma nota entre 0 e 10: -2
-2.0 é uma nota inválida!
Agora você digitou uma nota válida: 8.0
```

Comando de prompt 22 – Solução `notaValida()` utilizando função
Fonte: Elaborado pelo autor (2022)

Por termos utilizado uma função, conseguimos efetuar a consistência da nota. Se fosse um procedimento, não teríamos um valor (resposta True ou False) retornado e a sequência do algoritmo seria comprometida caso a nota estivesse errada.

7 AGRUPAMENTO DE SUBALGORITMOS

Profissionalmente falando, o desenvolvimento dos subalgoritmos não é necessariamente singular (construção de subalgoritmos independentes), mas sim

plural, um pode precisar do complemento de outro para auxiliar na resolução de um problema. Neste capítulo, falaremos sobre isso.

7.1 Tal pai, tal filho

Em genética, os filhos vêm dos pais, essa afirmação é óbvia!

Com subalgoritmos, podemos pensar como na genética, ou seja: alguns subalgoritmos existem pela atuação/necessidade de outros, enquanto certos subalgoritmos precisam de outros para ajudar a resolver o seu problema.

Um exemplo típico é o cálculo da média dos checkpoints (provas) dos cursos presenciais da FIAP. De três provas, é calculada a média das duas maiores notas, logo a menor nota deve ser descartada.

Vamos o código:

```
# ----- DEFINIÇÃO DOS SUBALGORITMOS
# Função FILHO: verifica e retorna somente o menor valor
def menor3notas(n1: float, n2: float, n3: float) -> float:
    menor = n1
    if n2 < menor:
        menor = n2
    if n3 < menor:
        menor = n3
    return menor

# Função PAI: utiliza outra função dentro dela
def mediaCheckPoints(n1: float, n2: float, n3: float) -> float:
    return (n1 + n2 + n3 - menor3notas(n1, n2, n3)) / 2

# ----- PROGRAMA PRINCIPAL
cp1 = float(input("Checkpoint 1: "))
cp2 = float(input("Checkpoint 2: "))
cp3 = float(input("Checkpoint 3: "))
print(f"Média dos checkpoints: {mediaCheckPoints(cp1, cp2,
cp3) :.1f}")
```

Código-fonte 31 – Solução com função pai e filho
Fonte: Elaborado pelo autor (2022)

Observe a execução:

```
Checkpoint 1: 4
Checkpoint 2: 8
Checkpoint 3: 9
Média dos checkpoints: 8.5
```

Comando de prompt 23 – Solução com função pai e filho
Fonte: Elaborado pelo autor (2022)

Dentro da função pai (`mediaCheckPoints`), há uma chamada para a função filho (`menor3notas`), e a função pai está sendo chamada dentro do programa principal.

Não há limite para a chamada de funções entre subalgoritmos, pode ter neto, tataravô etc. O que importa é que as chamadas sigam um roteiro lógico.

7.2 Subalgoritmos encadeados, isso existe?

Sabe aquele momento em programação no qual utilizamos os famosos *if* encadeados? Usamos essas situações quando precisamos tomar decisões dentro de respostas de outras decisões. Confuso, não?

Nem tanto! O significado de encadeado é estar ligado ou submetido a algo maior.

Pensando dessa forma, podemos cair em uma situação em que um subalgoritmo seja criado apenas para complementar outro “superior”. Eu já vi situações de chamadas de subalgoritmos em outros, como o que vimos (pais e filhos), mas criar um subalgoritmo e, dentro dele, criar um segundo, particularmente, só vi no Python.

Vamos aprender essa proeza.

Considere uma situação em que um subalgoritmo (neste exemplo, uma função) deseja calcular a média de duas notas (entre 0 e 10, inclusive) e só efetuará essa média caso as duas notas passadas por parâmetro sejam válidas; se um ou dois dos parâmetros não for uma nota válida, será retornado um flag com o valor -1 para informar o erro, caso contrário, será retornada a média das duas notas válidas.

Flag é um conceito de variável que controla ações no programa; dependendo do seu estado, ela força o fluxo do programa a tomar ações diferentes, independentemente das estruturas lógicas de decisão montadas.

Vamos elaborar uma rotina e as funções que atendam a necessidade do parágrafo anterior:


```
# DEFINIÇÃO DOS SUBALGORITMOS

# Definição da função superior
def media2notas(n1: float, n2: float) -> float:
    # Definição da função encadeada
    def notaValida(nota: float) -> bool:
        return nota >= 0 and nota <= 10

    # continuação da escrita da função superior
    if notaValida(n1) and notaValida(n2):
        return (n1 + n2) / 2
    else:
        return -1 # retornará -1 caso um dos parâmetros não seja
uma nota válida

# PROGRAMA PRINCIPAL
nota1 = float(input("Nota 1:"))
nota2 = float(input("Nota 2:"))
retorno = media2notas(nota1, nota2)
if retorno == -1:
    print("Nota(s) inválida(s)!")
else:
    print(f"Média = {retorno}")
```

Código-fonte 32 – Solução com subalgoritmo encadeado
Fonte: Elaborado pelo autor (2022)

Compare as execuções:

```
Nota 1: 8
Nota 2: 9
Média = 8.5
```

Comando de prompt 24 – Execução com notas válidas – subalgoritmo encadeado
Fonte: Elaborado pelo autor (2022)

```
Nota 1: 98
Nota 2: 9
Nota(s) inválida(s)!
```

Comando de prompt 25 – Execução com a primeira nota inválida – subalgoritmo encadeado
Fonte: Elaborado pelo autor (2022)

```
Nota 1: -2
```

```
Nota 2: 45
```

```
Nota(s) inválida(s)!
```

Comando de prompt 26 – Execução com a segunda nota inválida – subalgoritmo encadeado
Fonte: Elaborado pelo autor (2022)

Caso uma (ou mais) das notas seja(m) inválida(s), exibirá a mensagem “Nota(s) inválida(s)!”. Se as notas forem válidas, retornará o valor da média.

No desenvolvimento da rotina acima, começamos com a definição da função Superior e “do nada”, dentro dela, criamos uma função encadeada que a ajuda a resolver um segundo problema (verificar se a nota é válida); depois de construída a encadeada, volta à definição da função superior.

Todavia, a função encadeada só funcionará dentro da superior; se quisermos utilizá-la fora, ela não será encontrada. De certa forma, essa ação é um encapsulamento – somente a função que a envolve a enxerga.

Caso você tenha a necessidade de usar as duas de forma independente, crie-as como “pai e filho”, da maneira como vimos em um tópico anterior.

7.3 Subalgoritmos recursivos – recursividade

Até o momento, no tópico “Agrupamento de Algoritmos”, vimos as funções que são chamadas dentro de outras (pai e filho), função encadeada (uma construída dentro da outra); será que há uma forma de chamar a mesma função dentro de si? Sim! Isso se chama recursividade ou subalgoritmo recursivo.



Figura 10 – Imagem ilustrativa de recursividade
Fonte: Estatística com R (2015)

O conceito de recursividade é o seu espelhamento, como ilustra a imagem acima. Tecnicamente falando, um subalgoritmo recursivo é aquele que invoca a si dentro dele mesmo uma quantidade finita de vezes.

Para resolvermos um subalgoritmo, há duas formas: a iterativa e a recursiva.

7.3.1 Solução iterativa

Solução iterativa é aquela que nós utilizamos até o momento na criação dos subalgoritmos. O subalgoritmo é criado para resolver um problema claramente definido utilizando as instruções aprendidas.

Considere uma função iterativa que passe como parâmetro um número que representa o início da contagem e outro que representa o fim da contagem, ao ser executada a função, serão exibidos os números desse intervalo.

Veja a resolução iterativa:

```
# DEFINIÇÃO DO SUBALGORITMO ITERATIVO
def contagemIterativa(início: float, limite: float) -> None:
    while início <= limite:
        print(início)
        início += 1
    else:
        print("Fim!")

# PROGRAMA PRINCIPAL
print("Executando a função iterativa...")
contagemIterativa(1, 5)
```

Código-fonte 33 – Solução com subalgoritmo iterativo
Fonte: Elaborado pelo autor (2022)

Segue a execução:

```
Executando a função iterativa...
1
2
3
4
5
Fim!
```

Comando de prompt 27 – Execução da solução iterativa
Fonte: Elaborado pelo autor (2022)

Até aqui, nenhuma novidade; aprendemos a criar subalgoritmos assim.

7.3.2 Solução recursiva

A novidade vem agora, vamos utilizar o subalgoritmo recursivo na resolução de um problema.

Considere o mesmo problema do tópico anterior (exibir os números de um intervalo).

Segue o código:

```
# DEFINIÇÃO DO SUBALGORITMO RECURSIVO
def contagemRecursiva(inicio: float, limite: float) -> None:
    if inicio > limite:
        print("Fim!")
    else:
        print(inicio)
        contagemRecursiva(inicio + 1, limite)

# PROGRAMA PRINCIPAL
print("Executando a função recursiva...")
contagemRecursiva(1, 5)
```

Código-fonte 34 – Solução com subalgoritmo recursivo
Fonte: Elaborado pelo autor (2022)

Repare que a execução é idêntica.

```
Executando a função recursiva...
1
2
3
4
5
Fim!
```

Comando de prompt 28 – Execução da solução da forma recursiva
Fonte: Elaborado pelo autor (2022)

A função é invocada pelo programa principal, capta os números passados por parâmetro e dentro da função, na negativa da condição if, a função se “autochama”, novamente inserindo no parâmetro um valor +1 maior no primeiro parâmetro, para que a **condição de parada** (veja a seguir) seja satisfeita e termine a recursão.

7.3.2.1 Condição de parada / argumento – recursividade

Um subalgoritmo recursivo o executa finitas vezes, gerando um “looping finito”. Todavia, se um critério não for bem estabelecido, ele pode gerar um “looping infinito” e ocorrerá uma falha (BUG) semelhante a de um laço infinito.

Para estabelecermos esse critério, utilizamos uma **condição de parada** dentro do corpo do subalgoritmo recursivo e, para que essa condição seja atingida, devemos passar um **argumento** que possibilite que a condição seja alcançada.

Veja a localização desses termos no exemplo anterior:

```
def contagemRecursiva(inicio, limite):  
    if inicio > limite:  
        print("Fim!")  
    else:  
        print(inicio)  
        contagemRecursiva(inicio + 1, limite)
```

Código-fonte 35 – Definição de condição de parada e argumento
Fonte: Elaborado pelo autor (2022)

8 IMPORTANDO FUNÇÕES DE ARQUIVOS

Creio que, depois de ter visto todo esse conteúdo, você talvez esteja se perguntando: “Colocar todas as funções criadas dentro do mesmo arquivo vai deixar o arquivo-fonte extenso, poderíamos colocar as funções em outro arquivo?”.

Sim! É isso o que veremos neste tópico.

Vejamos algumas formas de trabalhar com arquivos de funções.

8.1 Importando somente o arquivo

Considere que um arquivo chamado subalgoritmos.py tenha o procedimento:

```
def emocao() -> None:  
    print("Eba! Estou aprendendo a usar biblioteca.")
```

Código-fonte 36 – Função emocao()
Fonte: Elaborado pelo autor (2022)

No arquivo principal, denominado Principal.py, considere o código:

```
import subalgoritmos  
  
subalgoritmos.emocao()
```

Código-fonte 37 – Função emocao() em um arquivo
Fonte: Elaborado pelo autor (2022)

Está pronto! Veja a execução:

```
Eba! Estou aprendendo a usar biblioteca.
```

Comando de prompt 29 – Utilizando arquivo de função – forma 1
Fonte: Elaborado pelo autor (2022)

Dessa forma, é necessário repetir o nome do arquivo (em nosso caso, subalgoritmos) com as funções antes do nome do procedimento (emoção) que vamos utilizar, separado por ponto: `subalgoritmos.emocao()`.

8.2 Importando o arquivo com os subalgoritmos listados

Vejam uma segunda forma.

```
from subalgoritmos import emocao  
  
emocao()
```

Código-fonte 38 – Função emocao() importada
Fonte: Elaborado pelo autor (2022)

Veja a execução:

```
Eba! Estou aprendendo a usar biblioteca.
```

Comando de prompt 30 – Utilizando arquivo de função – forma 2
Fonte: Elaborado pelo autor (2022)

Para tirarmos o nome do arquivo na chamada da função, podemos usar o `from` e o nome do subalgoritmo (*emoção*) logo após o `import`.

A vantagem dessa forma é que não precisamos repetir o nome do arquivo antes da chamada da função. A desvantagem é que devemos importar o subalgoritmo pelo nome: E se em um arquivo houver muitos subalgoritmos?

A resposta para essa pergunta está no próximo tópico.

8.3 Importando o arquivo com todos os subalgoritmos

Lembra que você perguntou: “Caso a biblioteca tenha diversas funções, devemos colocar o nome de todas?”. Sim e não.

Sobre a resposta “Sim”: você pode usar esse recurso separando o nome dos subalgoritmos por vírgula depois do `import`:

```
# listagem com todos os subalgoritmos que você queira utilizar
from subalgoritmos import emocao, subalg1, subalg2
```

Código-fonte 39 – Função `emocao()` importada juntamente com outras
Fonte: Elaborado pelo autor (2022)

Sobre a resposta “Não”: você pode utilizar uma terceira forma:

```
from subalgoritmos import *

emocao()
```

Código-fonte 40 – Função `emocao()` importada com todas as outras
Fonte: Elaborado pelo autor (2022)

Veja a execução:

```
Eba! Estou aprendendo a usar biblioteca.
```

Comando de prompt 31 – Utilizando arquivo de função – forma 3
Fonte: Elaborado pelo autor (2022)

O asterisco (como o do saudoso sistema operacional DOS) significa “Tudo”, ou seja, a importação valerá para todos os subalgoritmos contidos no arquivo biblioteca. Esta é a forma mais confortável.

8.4 Exemplo de aplicação de subalgoritmos em um arquivo

Vamos utilizar um exemplo com mais de um subalgoritmo dentro do arquivo biblioteca.

Considere que o código abaixo está escrito em um arquivo de nome “principal.py”:

```
Principal.py
# DEFINIÇÃO DOS SUBALGORITMOS
def contagemIterativa(inicio: int, limite: int) -> None:
    while inicio <= limite:
        print(inicio)
        inicio += 1
    else:
        print("Fim!")

def contagemRecursiva(inicio: int, limite: int) -> None:
    if inicio > limite:
        print("Fim!")
    else:
        print(inicio)
        contagemRecursiva(inicio+1, limite)

# PROGRAMA PRINCIPAL
print("Executando a função iterativa...")
contagemIterativa(1, 5)
print("Executando a função recursiva...")
contagemRecursiva(1, 5)
```

Código-fonte 41 – Solução múltiplos algoritmos no programa principal
Fonte: Elaborado pelo autor (2022)

Agora crie um arquivo chamado **"Biblioteca.py"** na mesma pasta do arquivo anterior, recorte a parte da # Definição dos Subalgoritmos e cole neste arquivo.

O novo arquivo (biblioteca.py) deverá ficar assim:

```
biblioteca.py
# DEFINIÇÃO DOS SUBALGORITMOS
def contagemIterativa(inicio: int, limite: int):
    while inicio <= limite:
        print(inicio)
        inicio += 1
    else:
        print("Fim!")

def contagemRecursiva(inicio: int, limite: int):
    if inicio > limite:
        print("Fim!")
    else:
        print(inicio)
        contagemRecursiva(inicio+1, limite)
```

Código-fonte 42 – Funções no arquivo biblioteca.py
Fonte: Elaborado pelo autor (2022)

No programa principal.py, escreva o código:

```
from biblioteca import *

print("Execução do procedimento contagemIterativa...")
contagemIterativa(3, 8)
print("Execução do procedimento contagemRecursiva...")
contagemRecursiva(5,10)
```

Código-fonte 43 – Execução do arquivo principal.py
Fonte: Elaborado pelo autor (2022)

Veja a execução:

```
Execução do procedimento contagemIterativa...
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8
```

```
Execução do procedimento contagemRecursiva...
```

```
5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
10
```

Comando de prompt 32 – Forma iterativa e recursiva
Fonte: Elaborado pelo autor (2022)

Com esse exemplo, finalizamos o conteúdo de subalgoritmos.

9 DESAFIO – CÁLCULOS DAS MÉDIAS FIAP PRESENCIAL

Vou propor um desafio!

Para ver se você realmente compreendeu o conteúdo do Capítulo 4 “Subalgoritmos”!

Para passar por este desafio, você deverá criar vários subalgoritmos a fim de desenvolver a solução. No final do capítulo, você encontrará uma solução desenvolvida por mim.

Vamos ao desafio!

9.1 Definição do desafio

Teremos como desafio calcular a média anual de um aluno da FIAP no modelo presencial.

A FIAP utiliza o modelo anual e separa as médias em duas grandes médias, denominadas Médias Semestrais, para encontrar a Média Anual.

Vamos ao detalhamento do primeiro semestre:

- O aluno submete-se a três checkpoints (provas CP1, CP2 e CP3). A média dos checkpoints (**MCP1Sem**) equivale à média simples das duas maiores notas.
- O aluno participa do Challenge (projeto), que é composto de duas notas denominadas Sprint1 e Sprint2 (S1 e S2), a média dos sprints (**MS1Sem**) é calculada com base na média simples das notas S1 e S2.
- O aluno submete-se a uma prova semestral chamada Global Solution (**GS1Sem**).

A média do primeiro semestre (M1Sem) é calculada com a ponderação: **MCP1Sem** valendo 20%, **MS1Sem** valendo 20% e a **GS1Sem** valendo 60%.

Segue a fórmula:

$$M1Sem = \frac{MCP1Sem \cdot 2 + MS1Sem \cdot 2 + GS1Sem \cdot 6}{10}$$

Detalhamento do segundo semestre (quase igual):

- O aluno se submete a três checkpoints (provas CP4, CP5 e CP6), a média dos checkpoints (**MCP2Sem**) equivale à média simples das duas maiores notas.
- O aluno participa do Challenge (projeto), que é composto de duas notas denominadas Sprint3 e Sprint4 (S3 e S4), a média dos sprints (**MS2Sem**) é calculada com base na média simples das notas S3 e S4.
- O aluno se submete a uma prova semestral chamada Global Solution (**GS2Sem**).

A média do segundo semestre (**M2Sem**) é calculada com a ponderação: **MCP2Sem** valendo 20%, **MS2Sem** valendo 20% e a **GS2Sem** valendo 60%.

Segue a fórmula:

$$M2Sem = \frac{MCP2Sem \cdot 2 + MS2Sem \cdot 2 + GS2Sem \cdot 6}{10}$$

Com as **M1Sem** e **M2Sem** calculadas, agora podemos calcular a média anual (**MA**). Ela é calculada pela proporção: **M1Sem** valendo 40% e **M2Sem** valendo 60%.

Segue a fórmula:

$$MA = \frac{M1Sem \cdot 4 + M2Sem \cdot 6}{10}$$

Uma vez calculada a Média Anual (**MA**), deverá exibir os Status, sendo estes como descritos a seguir:

- Se a **MA** for maior ou igual a 6, o Status será “Aprovado” (exibir juntamente com a MA atingida).
- Se a **MA** for menor do que 4, o Status será “Reprovado” (exibir juntamente com a MA atingida).
- Se a **MA** estiver entre 4 e 5.9, o aluno poderá participar por meio do processo de exame. Nesse caso, o programa deverá pedir a nota de exame (**NE**) ao aluno e calcular a nova Média Final (**MF**) considerando a média simples entre **MA** e **NE**, logo:
 - Fórmula: $MF = \frac{MA + NE}{2}$

Para desenvolver essa solução plenamente, você deve atender às seguintes informações e requisitos:

- Todas as notas digitadas pelo usuário devem ser válidas (entre 0 e 10).
- Caso uma nota não esteja nesse intervalo, advertir o usuário e pedir para ele digitar novamente a mesma nota.
- Toda média calculada deve ser exibida na tela.
- Utilize a boa prática: cada subalgoritmo deve resolver apenas um problema.

- Ao final da execução do calculo da Média Anual, perguntar se o aluno deseja continuar executando o programa digitando [S]im ou [N]ão. A digitação do S ou N também deve ser verificada e somente essas duas letras serão aceitas.

Seguem a execução da proposta da solução e os planos de testes:

9.1.1 Plano de teste “Reprovado” de forma direta

Neste teste, vamos digitar também valores inválidos (inconsistentes):

```
----- P R I M E I R O   S E M E S T R E
Digite o checkpoint 1: 2
Digite o checkpoint 2: 34
A nota 34.0 não é válida.
Digite uma nota entre 0 e 10: 3
Digite o checkpoint 3: 2
Média dos checkpoints: 2.5
Digite o sprint 1: 5
Digite o sprint 2: 3
Média dos sprints: 4.0
Digite a prova semestral: 2
Média do primeiro semestre: 2.50
----- S E G U N D O   S E M E S T R E
Digite o checkpoint 1: 4
Digite o checkpoint 2: 5
Digite o checkpoint 3: 2
Média dos checkpoints: 4.5
Digite o sprint 1: -5
A nota -5.0 não é válida.
Digite uma nota entre 0 e 10: 3
Digite o sprint 2: 2
Média dos sprints: 2.5
Digite a prova semestral: 98
A nota 98.0 não é válida.
Digite uma nota entre 0 e 10: 3
Média do segundo semestre: 3.20

Média final: 2.9 - Reprovado!

Calcular a média de outro aluno? [S]im ou [N]ão: w
ERRO! Digite [S]im ou [N]ão: S
```

9.1.2 Plano de teste “Reprovado” por exame

```
----- P R I M E I R O   S E M E S T R E
Digite o checkpoint 1: 3
Digite o checkpoint 2: 4
Digite o checkpoint 3: 5
Média dos checkpoints: 4.5

Digite o sprint 1: 5
Digite o sprint 2: 6
Média dos sprints: 5.5

Digite a prova semestral: 5
Média do primeiro semestre: 5.00

----- S E G U N D O   S E M E S T R E
Digite o checkpoint 1: 4
Digite o checkpoint 2: 3
Digite o checkpoint 3: 4
Média dos checkpoints: 4.0

Digite o sprint 1: 5
Digite o sprint 2: 6
Média dos sprints: 5.5

Digite a prova semestral: 4
Média do segundo semestre: 4.30

Média final: 4.6 - Exame

Digite a nota do exame: 6
Reprovado em exame com média 5.29
Calcular a média de outro aluno? [S]im ou [N]ão: S
```


9.1.3 Plano de teste “Aprovado” por exame

```
----- P R I M E I R O   S E M E S T R E
Digite o checkpoint 1: 3
Digite o checkpoint 2: 4
Digite o checkpoint 3: 5
Média dos checkpoints: 4.5

Digite o sprint 1: 2
Digite o sprint 2: 3
Média dos sprints: 2.5

Digite a prova semestral: 4
Média do primeiro semestre: 3.80

----- S E G U N D O   S E M E S T R E
Digite o checkpoint 1: 5
Digite o checkpoint 2: 6
Digite o checkpoint 3: 3
Média dos checkpoints: 5.5

Digite o sprint 1: 4
Digite o sprint 2: 5
Média dos sprints: 4.5

Digite a prova semestral: 4
Média do segundo semestre: 4.40

Média final: 4.2 - Exame

Digite a nota do exame: 8.5
Aprovado em exame com média 6.33
Calcular a média de outro aluno? [S]im ou [N]ão: S
```

9.1.4 Plano de teste “Aprovado” direto

```
----- P R I M E I R O   S E M E S T R E
Digite o checkpoint 1: 6
Digite o checkpoint 2: 7
Digite o checkpoint 3: 8
Média dos checkpoints: 7.5

Digite o sprint 1: 9
Digite o sprint 2: 10
Média dos sprints: 9.5

Digite a prova semestral: 9
Média do primeiro semestre: 8.80
----- S E G U N D O   S E M E S T R E
Digite o checkpoint 1: 8
Digite o checkpoint 2: 7
Digite o checkpoint 3: 6
Média dos checkpoints: 7.5

Digite o sprint 1: 7
Digite o sprint 2: 8
Média dos sprints: 7.5

Digite a prova semestral: 9
Média do segundo semestre: 8.40

Média final: 8.6 - Aprovado!
Calcular a média de outro aluno? [S]im ou [N]ão: N
```

Comando de prompt 33 – Os quatro planos de testes do desafio
Fonte: Elaborado pelo autor (2022)

Depois de desenvolver a sua solução, efetue esses quatro testes para ver se a sua solução chega aos mesmos valores. Se sim, você passou no desafio. :)

10 ANEXO – RESOLUÇÃO DO DESAFIO

Cada um desenvolve a solução de uma forma, o importante é que todos os requisitos sejam atendidos. Segue uma correção que fiz utilizando os conceitos apresentados neste capítulo:

```
# ----- DEFINIÇÃO DOS SUBALGORITMOS
"""
Verifica se uma nota é válida ou não
PROTÓTIPO: notaValida(nota: Float): Boolean
"""
def notaValida(nota: float) -> bool:
    return nota >= 0 and nota <= 10

"""
Exibe uma mensagem de nota inválida
PROTÓTIPO: msgNotaInvalida(nota: Float): void
"""
def msgNotaInvalida(nota: float) -> None:
    print(f"A nota {nota} não é válida. \nDigite uma nota
entre 0 e 10: ", end='')

"""
Retorna o menor entre três valores
PROTÓTIPO: menor3n(n1, n2, n3: Float): float
"""
def menor3n(n1, n2, n3: float) -> float:
    menor = n1
    if n2 < menor:
        menor = chk2
    if n3 < menor:
        menor = chk3
    return menor

"""
Calcular a média dos CheckPoints
PROTÓTIPO: mediaCheckpoints(n1, n2, n3: Real): Real
"""
def mediaCheckpoints(n1, n2, n3: float) -> float:
    return (n1 + n2 + n3 - menor3n(n1, n2, n3)) / 2

"""
Calcular a média de 2 valores
```

```
PROTÓTIPO: media2n(n1, n2: Real): Real
"""
def media2n(n1, n2: float) -> float:
    return (n1 + n2) / 2

"""
Calcular a porcentagem de um valor
PROTÓTIPO: porcentagemValor(valor, percentual: Real): Real
"""
def porcentagemValor(valor, percentual: float) -> float:
    return valor * percentual

"""
Calcular a soma de 3 numeros
PROTÓTIPO: soma3n(n1, n2, n3: Real): Real
"""
def soma3n(n1, n2, n3: float) -> float:
    return n1 + n2 + n3

"""
Calcular a soma de 2 numeros
PROTÓTIPO: soma2n(n1, n2: Real): Real
"""
def soma2n(n1, n2: float) -> float:
    return n1 + n2

"""
Exibe se está aprovado ou reprovado
PROTÓTIPO: exhibeStatus(media: Real): String
"""
def exhibeStatusFunc(media: float) -> str:
    if media >= 6:
        return "Aprovado!"
    elif media < 4:
        return "Reprovado!"
    else:
        return "Exame"

"""
Exibe se está aprovado ou reprovado
PROTÓTIPO: exhibeStatus(media: Real): void
"""
def exhibeStatusProc(media: float) -> None:
    if media >= 6:
        print("Aprovado!\n")
    elif media < 4:
```

```

        print("Reprovado!\n")
    else:
        print("Exame\n")

"""
Exibe a mensagem de Aprovado ou Reprovado depois do exame
PROTÓTIPO: msgAprovReprovExame(mf: Real): void
"""
def msgAprovReprovExame(mf: float) -> None:
    if mf < 6:
        print(f"Reprovado em exame com média {mf}")
    else:
        print(f"Aprovado em exame com média {mf}")

"""
Verifica se foi digitado S de Sim
PROTÓTIPO: continua(op: string): Lógico
"""
def continua(op: str) -> bool:
    return op == "S" or op == "s"

"""
Verifica se foi digitado S ou N de Sim ou Não
PROTÓTIPO: opcaoInvalida(op: string): Lógico
"""
def opcaoInvalida(op):
    return op != "s" and op != "S" and op != "n" and op != "N"

# ----- PROGRAMA PRINCIPAL
opcao = 's'

while continua(opcao):

    print("----- P R I M E I R O   S E M E S
T R E")
    # Leitura dos checkpoints
    chk1 = float(input("Digite o checkpoint 1:"))
    while not notaValida(chk1):
        msgNotaInvalida(chk1)
        chk1 = float(input())

    chk2 = float(input("Digite o checkpoint 2:"))
    while not notaValida(chk2):
        msgNotaInvalida(chk2)
        chk2 = float(input())

    chk3 = float(input("Digite o checkpoint 3:"))

```

```
while not notaValida(chk3):
    msgNotaInvalida(chk3)
    chk3 = float(input())

# Calcula a media dos checkpoints do primeiro semestre
mediaChk = mediaCheckpoints(chk1, chk2, chk3)
print(f"Média dos checkpoints: {mediaChk:.1f}\n")

# Leitura dos Sprints
sprint1 = float(input("Digite o sprint 1:"))
while not notaValida(sprint1):
    msgNotaInvalida(sprint1)
    sprint1 = float(input())

sprint2 = float(input("Digite o sprint 2:"))
while not notaValida(sprint2):
    msgNotaInvalida(sprint2)
    sprint2 = float(input())

# Calculando a média dos Sprints
mediaSprint = media2n(sprint1, sprint2)
print(f"Média dos sprints: {mediaSprint:.1f}\n")

# Lendo a nota da prova semestral
provaSemestral = float(input("Digite prova semestral:"))
while not notaValida(provaSemestral):
    msgNotaInvalida(provaSemestral)
    provaSemestral = float(input())

# Ponderando os valores das médias
pontosChk = porcentagemValor(mediaChk, 0.2)
pontosSprints = porcentagemValor(mediaSprint, 0.2)
pontosSemestral = porcentagemValor(provaSemestral, 0.6)

# Cálculo da media do primeiro semestre
mediaPrimeiroSemestre = soma3n(pontosChk, pontosSprints,
pontosSemestral)
print(f"Média do primeiro semestre:
{mediaPrimeiroSemestre:.2f}\n")

# Pontos obtidos no primeiro semestre
pontosPrimeiroSemestre =
porcentagemValor(mediaPrimeiroSemestre, 0.4)

print("----- S E G U N D O   S E M E S T
R E")
# Leitura dos checkpoints
chk1 = float(input("Digite o checkpoint 1:"))
while not notaValida(chk1):
    msgNotaInvalida(chk1)
    chk1 = float(input())
```

```
chk2 = float(input("Digite o checkpoint 2:"))
while not notaValida(chk2):
    msgNotaInvalida(chk2)
    chk2 = float(input())

chk3 = float(input("Digite o checkpoint 3:"))
while not notaValida(chk3):
    msgNotaInvalida(chk3)
    chk3 = float(input())

# Calcula a media dos checkpoints do primeiro semestre
mediaChk = mediaCheckpoints(chk1, chk2, chk3)
print(f"Média dos checkpoints: {mediaChk:.1f}\n")

# Leitura dos Sprints
sprint1 = float(input("Digite o sprint 1:"))
while not notaValida(sprint1):
    msgNotaInvalida(sprint1)
    sprint1 = float(input())

sprint2 = float(input("Digite o sprint 2:"))
while not notaValida(sprint2):
    msgNotaInvalida(sprint2)
    sprint2 = float(input())

# Calculando a média dos Sprints
mediaSprint = media2n(sprint1, sprint2)
print(f"Média dos sprints: {mediaSprint:.1f}\n")

# Lendo a nota da prova semestral
provaSemestral = float(input("Digite prova semestral:"))
while not notaValida(provaSemestral):
    msgNotaInvalida(provaSemestral)
    provaSemestral = float(input())

# Ponderando os valores das médias
pontosChk = porcentagemValor(mediaChk, 0.2)
pontosSprints = porcentagemValor(mediaSprint, 0.2)
pontosSemestral = porcentagemValor(provaSemestral, 0.6)

# Cálculo da media do primeiro semestre
mediaSegundoSemestre = soma3n(pontosChk, pontosSprints,
pontosSemestral)
print(f"Média do segundo semestre:
{mediaSegundoSemestre:.2f}\n\n")

# Pontos obtidos no primeiro semestre
pontosSegundoSemestre =
porcentagemValor(mediaSegundoSemestre, 0.6)
```

```
# Cálculo da média final
mediaFinal = soma2n(pontosPrimeiroSemestre,
pontosSegundoSemestre)

print(f"Média final: {mediaFinal:.1f} - ", end="")

exibeStatusProc(mediaFinal)
if exibestatusFunc(mediaFinal) == "Exame":
    # Leitura dos Sprints
    notaExame = float(input("Digite a nota do exame:"))
    while not notaValida(notaExame):
        msgNotaInvalida(notaExame)
        notaExame = float(input())
    mfinal = media2n(mediaFinal, notaExame)
    msgAprovReprovExame(mfinal)

    opcao = input("Calcular a média de outro aluno? [S]im ou
[N]ão.")
    while opcaoInvalida(opcao):
        opcao = input("ERRO! Digite [S]im ou [N]ão:")

print('''
-----
Obrigado por utilizar o nosso programa!
-----
''')
```

Código-fonte 44 – Correção do desafio
Fonte: Elaborado pelo autor (2022)

ATIVIDADE – PRATICANDO LÓGICA COM PYTHON

Esta atividade não é avaliativa, caso queira ter um feedback do seu tutor, envie sua atividade para on@fiap.com.br.

O cálculo das raízes da equação do segundo grau é algo rotineiro para os alunos de ensino médio. Visando facilitar a conferência das respostas corretas a partir dos exercícios passados pelo professor, vamos construir uma solução (em Python) que calcule as raízes por Bhaskara a partir de valores fornecidos pelo usuário até que ele não queira mais utilizar a solução que você desenvolveu.

Antes, vamos lembrar como as raízes são calculadas por Bhaskara com um exemplo:

Expressão da Equação do segundo grau:

$$ax^2 + bx + c = 0$$

Para aplicar, vamos utilizar um exemplo que calcule duas raízes distintas:

Observe que temos uma equação do segundo grau completa. Primeiro vamos encontrar os coeficientes da equação, isto é, os valores de **a**, **b** e **c**.

- $x^2 - 5x + 6 = 0$
 - $a = 1$
 - $b = -5$
 - $c = 6$

Calculando o Delta:

Primeiro passo: ($\Delta = b^2 - 4ac$)

- $\Delta = (-5)^2 - 4.1.6 = 25 - 24 = 1 \ (\Delta > 0)$

Calculando as Raízes x_1 e x_2 :

Fórmula de Bhaskara:

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}$$

Para x_1 temos:

$$x_1 = \frac{-(-5) + \sqrt{1}}{2 \times 1} = \frac{5 + 1}{2} = \frac{6}{2} = 3$$

Para x_2 temos:

$$x_2 = \frac{-(-5) - \sqrt{1}}{2 \times 1} = \frac{5 - 1}{2} = \frac{4}{2} = 2$$

Raízes



Neste exemplo, obtivemos duas raízes distintas (x_1 e x_2) porque o delta resultou um valor positivo. Contudo, nem sempre a resolução é possível ou é desta forma. Então, considere os seguintes requisitos, restrições e orientações:

- Se o valor de **a** for 0 (zero), exibir a mensagem **“Esta equação não é do segundo grau, sim do primeiro”**. (Não tem como calcular)
- Se **b** ou **c** forem 0, exibir os valores encontrados (para delta, x_1 e x_2), em seguida a mensagem **“Equação do segundo grau INCOMPLETA”**.
- Se **b** e **c** não forem 0, exibir os valores encontrados (para delta, x_1 e x_2), em seguida a mensagem **“Equação do segundo grau COMPLETA”**.
- Se o delta resultar um valor negativo, não é possível calcular por Bhaskara com números Reais porque não há raiz quadrada negativa; então exibir o valor de delta e a mensagem **“Não é possível calcular x_1 e x_2 porque o delta é negativo”**.
- Se o delta resultar zero a equação admite o mesmo valor para as duas raízes, então exibir os valores encontrados (para delta, x_1 e x_2), em seguida a mensagem **“As raízes x_1 e x_2 tem o mesmo valor”**.
- Se o delta resultar um valor positivo (como no exemplo acima) as raízes se dão por duas soluções distintas para x_1 e x_2 , então exibir os valores encontrados (para delta, x_1 e x_2), em seguida a mensagem **“As raízes x_1 e x_2 tem valores distintos”**.

Agora que lembramos como se calcula a equação do segundo grau por Bhaskara, vamos construir uma solução em Python implementando Subalgoritmos (Funções e procedimentos com ou sem passagem de parâmetros) onde for possível e que resolva este problema (quando possível) a partir de 3 valores fornecidos pelo usuário. Exibir as frases em negrito descritas acima.

Uma ajuda:

Para calcular a raiz quadrada de um número, podemos utilizar a função `sqrt()`, veja uma aplicação:

```
# Acrescente esta biblioteca no início do Arquivo
import math

# y = math.sqrt(x) - calcula a raiz quadrada de x e atribui a y
raiz = math.sqrt(25) # a variável raiz valerá 5
```

Ufa, verificamos todas as possibilidades e eventualmente calculamos...

NÃO, NÃO ACABOU!

Ao final da execução do cálculo (OU NÃO) das raízes, perguntar ao usuário: **“Continuar executando o programa? [S]im ou [N]ão:”**. Considere “S” (ou “s” minúsculo) para continuar executando ou “N” (ou “n” minúsculo) para terminar a execução do programa. CASO O USUÁRIO **NÃO DIGITE** UMA DESTAS DUAS LETRAS (em maiúsculo ou minúsculo), adverti-lo com a mensagem **“Letra inválida! Digite [S]im ou [N]ão:”**, até que ele digite a letra correta.

Considerações:

- Você deve aplicar Subalgoritmos onde for possível.
- Cada Subalgoritmo deve resolver apenas UM problema.
- Exceto a função (método) `sqrt`, nenhuma outra função proprietária do Python pode ser utilizada.

TESTES:

PLANO DE TESTE							
este		b	c	delta	x1	x2	Mensagem
		2	3	-8	-	-	Delta = -8.0. Não é possível calcular x1 e x2 porque o delta é negativo.
1	2	3	16	-1	3		Delta = 16.0 x1 = -1.0 x2 = 3.0 Equação do segundo grau COMPLETA As raízes x1 e x2 tem valores distintos
		-	-	-	-	-	Esta equação não é do segundo grau, sim do primeiro.
1	0	2	8	-1,41	1,41		Delta = 8.0 x1 = -1.4142135623730951 x2 = 1.4142135623730951 Equação do segundo grau INCOMPLETA As raízes x1 e x2 tem valores distintos
1	2	0	4	0	2		Delta = 4.0 x1 = -0.0 x2 = 2.0 Equação do segundo grau INCOMPLETA As raízes x1 e x2 tem valores distintos
	2	1	0	-1	-1		Delta = 0.0 x1 = -1.0 x2 = -1.0 Equação do segundo grau COMPLETA As raízes x1 e x2 tem o mesmo valor
TESTAR A OBRIGATORIEDADE DE "S" OU "N" PARA CONTINUAR OU NÃO A EXECUÇÃO							

Esta atividade não é avaliativa (não vale pontos)

No TEAMS, dentro do canal “Fase 4”, tem uma conversa aberta chamada “Discussão sobre a Atividade não avaliativa de CTWP”. Postem lá o seu arquivo .py, baixem as versões dos colegas e discutam a solução que cada fez.

O professor também postará um gabarito.

Minhas crianças, desejo que tenham aprendido o conceito de Subalgoritmos e curtido este capítulo.

Até o próximo!

EDSON

REFERÊNCIAS

UFF. **Entendendo a recursão.** 2015. Disponível em:
<<http://www.estadisticacomr.uff.br/?p=98>>. Acesso em: 21 out. 2022.

EMASP

GLOSSÁRIO

Subalgoritmos	Partes menores de códigos.
Modularização	Programar utilizando subalgoritmos.
Subalgoritmos nativos	Os que vêm na instalação da linguagem.
Subalgoritmos próprios	Os criados pelo programador.
Procedimento	Definição de subalgoritmo que não retorna valor.
Função	Definição de subalgoritmo que retorna valor.
Parâmetros	Identificadores que transportam valores entre algoritmos e subalgoritmos.
Identificador	Tudo que é armazenado na memória do computador, tipo variável.
Default	Algo padrão.
*args	Tipo de parâmetro incontável.
Encadeado	Dependente.
Recursividade	Chamada sucessiva de si.
Import	Comando para acessar outro arquivo.