INTRODUÇÃO À PROGRAMAÇÃO E ESTRUTURAS DE DADOS FUNDAMENTAIS COM PYTHON PARA PROGRAMADORES C

Antônio Rodrigo Delepiane De Vit
Sidnei Renato Silveira

SYNAPSE Editora
Compartilhando conhecimento

INTRODUÇÃO À PROGRAMAÇÃO E ESTRUTURAS DE DADOS FUNDAMENTAIS COM PYTHON PARA PROGRAMADORES C

DOI: doi.org/10.36599/editpa-ipedfp

Antônio Rodrigo Delepiane De Vit
Sidnei Renato Silveira

SYNAPSE Editora

Compartilhando conhecimento

Todo o texto bem como seus elementos, metodologia, dados apurados e a correção são de inteira responsabilidade dos autores. Estes textos não representam de forma alusiva ou efetiva a posição oficial da Synapse Editora.

A Synapse Editora não se responsabiliza por eventuais mudanças ocorridas nos endereços convencionais ou eletrônicos citados nesta obra.

Os livros editados pela Synapse Editora, por serem de acesso livre, *Open Access*, é autorizado o download da obra, bem como o seu compartilhamento, respeitando que sejam referenciados os créditos autorais. Não é permitido que a obra seja alterada de nenhuma forma ou usada para fins comerciais.

O Conselho Editorial e pareceristas convidados analisaram previamente todos os manuscritos que foram submetidos à avaliação pelos autores, tendo sido aprovados para a publicação.

SYNAPSE Editora
Compartilhando conhecimento
2023

D278i De Vit, Antônio Rodrigo Delepiane

Introdução à programação e estruturas de dados fundamentais com Python para programadores C.
Autores: Antônio Rodrigo Delepiane De Vit; Sidnei Renato Silveira
Belo Horizonte, MG: Synapse Editora, 2023, 61 p.

Formato: PDF
Modo de acesso: World Wide Web
Inclui bibliografia

ISBN: 978-65-88890-30-1
DOI: doi.org/10.36599/editpa-ipedfp

1. Python 2. Linguagem C de programação, 3. Computação, 4. Análise de sistemas.

I. Introdução à programação e estruturas de dados fundamentais com Python para programadores C.
II. Antônio Rodrigo Delepiane De Vit
Sidnei Renato Silveira

CDD: 005 - 005.10218
CDU: 004 - 004.43

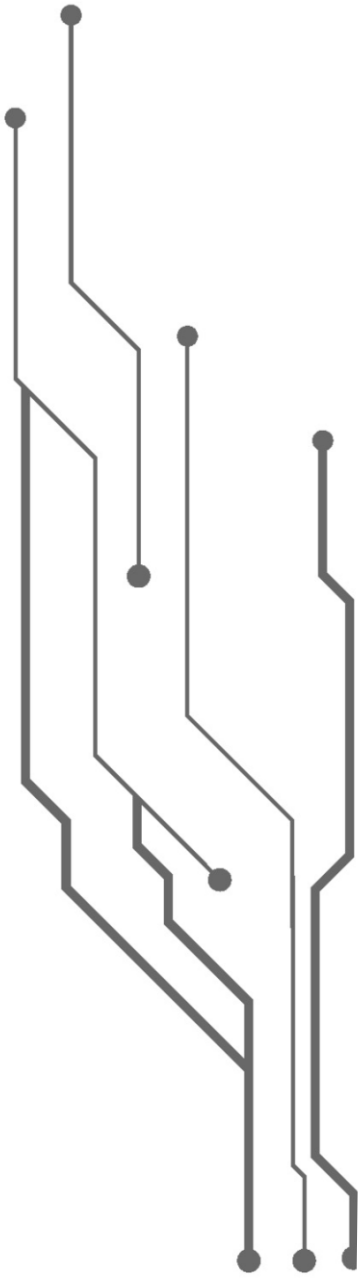
SYNAPSE EDITORA
Belo Horizonte – Minas Gerais
CNPJ: 20.874.438/0001-06
Tel: + 55 31 98264-1586
www.editorasynapse.org
editorasynapse@gmail.com

SYNAPSE Editora
Compartilhando conhecimento
2023

PREFÁCIO

Este livro apresenta os conceitos de programação de computadores e estruturas de dados fundamentais, utilizando as linguagens de programação C e Python. A intenção desta obra é de que a mesma sirva como material didático para as disciplinas introdutórias de programação e estruturas de dados para alunos que não possuem conhecimentos em programação e, também, para programadores e estudantes que conhecem a linguagem C e querem aprofundar seus conhecimentos na linguagem de programação Python. Os exemplos práticos foram desenvolvidos utilizando a IDE (Integrated Development Environment) Eclipse com a adição da feature PyDev versão 8.2.0 (Python IDE for Eclipse) para o desenvolvimento dos códigos-fonte em Python e, para os códigos-fonte em linguagem C, da feature Eclipse Embedded C/C++ versão 6.1.2. A instalação destas features deve ser realizada na IDE Eclipse, no menu Help, na opção Eclipse Marketplace.

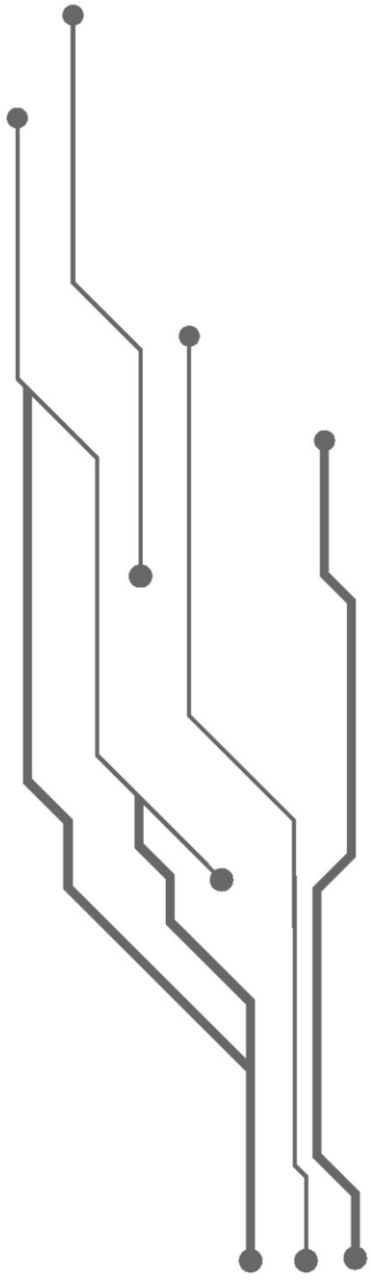
Aproveite!

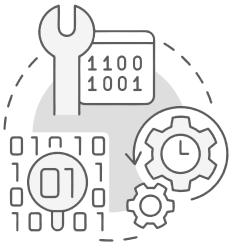


SYNAPSE Editora
Compartilhando conhecimento
2023

SUMÁRIO

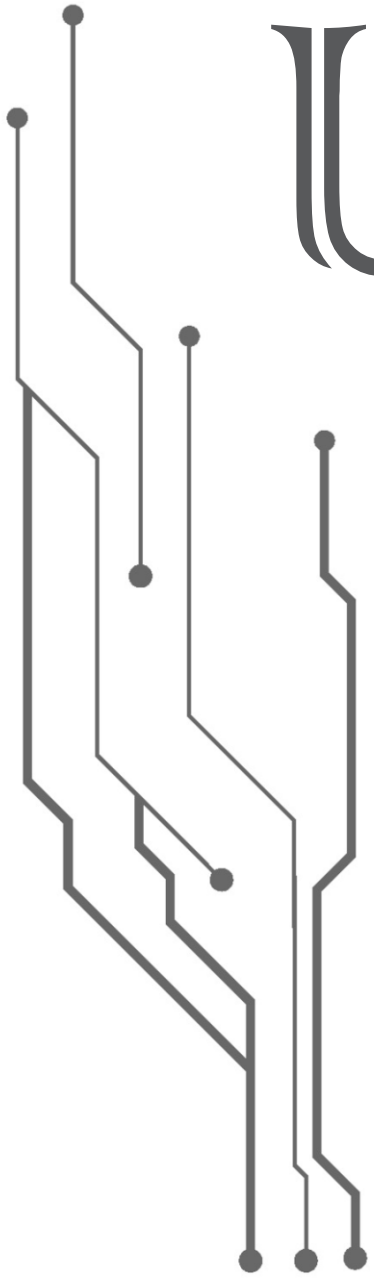
DOI: doi.org/10.36599/editpa-ipedfp





CAPÍTULO 1

Introdução à Programação



Um sistema computacional é dividido, basicamente, em duas partes: 1) hardware (parte física) e 2) software (parte lógica). O hardware abrange os componentes e o funcionamento interno do computador, enquanto que o software envolve o modo como se utiliza o computador. O hardware e o software são interdependentes, ou seja, de nada adianta um equipamento de última geração (hardware) sem que haja um bom programa (software) sendo executado nele, nem um software de última geração sendo executado em um computador ultrapassado (o que, na maioria das vezes, torna-se impossível) (SILVA et al., 2010; PARREIRA et al. 2017).

O processamento de dados realizado pelo computador (processamento de dados eletrônico) possui muitas vantagens sobre o processamento manual, principalmente pela sua velocidade de execução. Para que um computador possa desempenhar tarefas (processar dados) faz-se necessário (FALKEMBACH; SILVEIRA, 2005; PARREIRA et al. 2017; SILVA et al., 2010):

1. Descrever antecipadamente as operações que serão efetuadas, prevendo-se todos os casos possíveis para o problema que deve ser resolvido. Esta especificação é realizada por meio de um **algoritmo**. Um algoritmo é um conjunto de regras que permite realizar mecanicamente todas as operações particulares correspondentes a uma determinada tarefa. O algoritmo é uma decomposição de um problema do

usuário em operações elementares que podem ser executadas pelo computador utilizando alguma notação que programadores possam entender;

2. Traduzir o algoritmo obtido para uma linguagem de programação (programa fonte ou código-fonte). Uma linguagem de programação é composta por um conjunto bem definido de símbolos permitidos, regras de escrita e regras de comportamento (SEBESTA, 2018);

3. Traduzir para o formato interno do computador essa sequência de operações, por meio de um programa especial chamado compilador. Um compilador é um programa que traduz instruções em linguagem de programação para o formato interno do computador (SEBESTA, 2018);

4. Solicitar ao computador para que o programa seja executado, por meio de um comando do sistema operacional.

Uma segunda forma, mais completa, de apresentar os passos necessários para a resolução de problemas utilizando o computador é a seguinte (FALKEMBACH; SILVEIRA, 2005; PARREIRA *et al.* 2017; SILVA *et al.*, 2010):

1. Definição do Problema: consiste na descrição da situação a ser resolvida, por meio de um enunciado que deve ser claro e completo, fornecendo todas as informações necessárias para a sua resolução;

2. Análise do problema: consiste em obter, a partir da definição, informações, ou seja, subsídios para construir um modelo para a solução do problema e formalizar este modelo por meio de mecanismos, tais como os algoritmos. Quanto mais complexo for o problema mais se recomenda a utilização desta técnica;

3. Programação ou Codificação: consiste em transcrever, em uma Linguagem de Programação, as instruções referentes ao modelo da solução, criando o código-fonte;

4. Edição: consiste em transferir para o computador, mais especificamente para a memória RAM (*Random Access Memory*), as instruções do código-fonte;

5. Compilação: consiste na interpretação das instruções do código-fonte por meio da verificação da sintaxe do programa. O processo de compilação gera o Programa Objeto (SILVEIRA *et al.*, 2021);

6. Ligação (linkedição): a partir do Programa Objeto, gera-se o Programa Executável (SILVEIRA *et al.*, 2021);

7. Análise dos resultados (fase de testes): consiste em verificar se os resultados correspondem à expectativa, pois o fato do computador

apresentar um resultado indica, inicialmente, que o programa não possui erros de sintaxe, mas não significa que não apresenta erros que envolvem a lógica de programação, ou seja, o programa pode estar escrito corretamente de acordo com as regras da linguagem de programação mas não produzir os resultados esperados;

8. Documentação: consiste em descrever os procedimentos que foram utilizados na resolução do problema, para utilizá-los na solução de problemas similares.

Uma linguagem de programação é um simbolismo que permite a comunicação entre o programador e o computador. A linguagem natural (como a Língua Portuguesa) não é útil para esta finalidade, pelo fato de que a sua sintaxe é muito complexa, assim como sua semântica (SILVEIRA *et al.*, 2021).

Uma linguagem de programação é, portanto, mais restrita, e será definida por meio de um vocabulário autorizado, determinando sua sintaxe e sua semântica. Em uma linguagem de programação uma frase é chamada de instrução e corresponde à descrição de uma ou várias operações elementares do computador (SILVEIRA *et al.*, 2021).

Atualmente, existem mais de 2000 linguagens de programação diferentes, a maioria delas utilizadas apenas em situações muito particulares. Para facilitar o estudo, estas linguagens são divididas em classes diferentes, de acordo com suas características principais. Estas diferentes classes são chamadas de paradigmas de programação (SEBESTA, 2018; SILVEIRA *et al.*, 2021).

1.1 Conceitos Fundamentais

Para entendermos o que é um algoritmo, vamos partir de um exemplo. Queremos escrever um programa que calcule a média do rendimento acadêmico de um aluno. Supondo que, para ser aprovado, um aluno precisa atingir a média 7,0 (média aritmética simples) para ser aprovado (sem que seja necessário realizar o exame final), sendo que existem duas notas parciais, que iremos chamar de P1 e P2. Sendo assim, a fórmula para o cálculo da média é a seguinte:

$$\text{Média} = \frac{P1 + P2}{2}$$

A média é igual à nota atribuída à 1ª parte da avaliação (P1) mais a nota atribuída à 2ª parte da avaliação (P2), sendo que este resultado é dividido por 2. Se a média for maior ou igual a 7,0 o aluno está aprovado; caso contrário, ainda poderá realizar a avaliação final (exame final).

A intenção é construirmos um algoritmo que automatize o cálculo da média. Um algoritmo é a série de passos que permite que o cálculo da média seja executado pelo computador. Basicamente, teríamos os seguintes passos:

1. obter as notas de P1 e P2;
2. aplicar a fórmula da média;
3. verificar a média calculada.

Estes passos estão descritos em linguagem natural. Cada um pode descrever os passos utilizando as palavras e a forma que achar melhor. Em programação isto não é possível pois, como vimos anteriormente, uma linguagem de programação é composta por um conjunto bem definido de símbolos permitidos e regras de escrita (sintaxe) e regras de comportamento (semântica), ou seja, para escrevermos um programa que automatize o cálculo da média, precisaremos seguir estas regras, caso contrário o programa não funcionará, ou seja, não será executado (SILVA *et al.*, 2010; PARREIRA *et al.* 2017).

Uma das primeiras regras que precisamos entender é a utilização de informações variáveis e constantes. As informações variáveis precisam ficar armazenadas em posições da memória do computador ("gavetas") para que possam ser encontradas pelo programa. As variáveis são as informações que mudam no decorrer do tempo. No caso do programa que calcula a média, as informações variáveis são as notas de P1 e P2 e o resultado (média calculada). A Figura 1 apresenta, de forma gráfica, como as variáveis P1, P2 e média poderiam estar dispostas na memória do computador. Na figura foram representadas algumas gavetas vazias (podem existir inúmeras gavetas vazias, sendo limitadas à capacidade da memória utilizada) (SILVA *et al.*, 2010; PARREIRA *et al.* 2017).

Figura 1 – Representação gráfica das variáveis na memória do computador (Adaptado de PARREIRA *et al.*, 2017)

As informações constantes não variam durante a execução do programa. Por exemplo, no cálculo da média, independentemente da nota obtida pelo aluno, o resultado é sempre dividido por 2, pois estamos trabalhando com duas avaliações parciais para compor a média do aluno.

As informações variáveis, além de precisarem de um espaço de memória, precisam ter um tipo de dado associado, ou seja, o computador precisa saber

qual é o tipo de informação que cada uma das variáveis pode manipular. No caso da média de um aluno, o tipo de dado são os números reais, pois as notas são expressas com casas decimais. No início do algoritmo, o programador precisa declarar as variáveis que serão utilizadas no seu programa. O processo de declaração de variáveis consiste em definir um nome para as variáveis e o tipo de dado associado.

O nome de uma variável é um identificador (também conhecido como ID – *identifier*) e não muda durante a execução de um programa. Para definir o nome de uma variável devem ser seguidas algumas regras:

- não deve começar por um número;
- não deve conter espaços em branco;
- não deve conter caracteres especiais (+ - * / % $ # @ !), exceto o sublinhado (_), que é utilizado para separar os nomes das variáveis;
- não deve conter nenhum caracter de acentuação;
- não pode ser uma palavra reservada da linguagem de programação (um comando da linguagem, por exemplo, é uma palavra reservada).

Seguem alguns exemplos de **nomes de variáveis válidos**:

- Endereco (sem o ç que é um caracter especial)
- Idade
- Nome
- Nome1
- Nome_do_Aluno
- P1
- Nota_P1

Seguem alguns exemplos de **nomes de variáveis inválidos**:

- Nome do Aluno (contém espaços em branco)
- 1P (inicia por um número)
- Endereço (contém o caracter cedilha)
- if (é uma palavra reservada da linguagem, neste caso é uma palavra reservada das linguagens de programação C e Python que indica um comando ou instrução de seleção que estudaremos mais adiante).

Além do nome, na declaração de variáveis precisamos definir o tipo de dado que cada variável irá manipular. Os tipos de dados podem ser, por exemplo: números inteiros, números reais, cadeias de caracteres, entre outros. Estudaremos os tipos de dados das linguagens de programação C e Python mais adiante neste livro.

Antes de iniciarmos a codificação de um algoritmo precisamos, por meio da análise do problema, identificar as variáveis que serão necessárias (espaços de memória que serão necessários para armazenar os dados manipulados

pelo algoritmo). Estes dados podem ser divididos em dados de entrada e dados de saída.

Os dados de entrada são todas as informações que um programa precisa para chegar ao resultado. Estas informações são fornecidas por meio dos periféricos de entrada. O periférico de entrada mais comum é o teclado (o usuário digita as informações solicitadas pelo programa). Atualmente, com o uso de *smartphones* e o avanço dos dispositivos móveis, a tela do celular é muito utilizada como periférico de entrada, por meio da função *touch screen* ou tela de toque. Os dados de saída são os resultados obtidos por meio da execução de um programa (SILVA *et al.,* 2010; PARREIRA *et al.* 2017). Quando falamos em dispositivos móveis, os programas são mais conhecidos como aplicativos ou pela sigla *app*.

No caso do algoritmo para calcular a média, os dados de entrada são as notas de P1 e P2 e o resultado é a média do aluno. Sendo assim, são necessários três espaços de memória para armazenar os dados. As três variáveis manipulam dados numéricos.

A declaração das variáveis para o algoritmo da média poderia ser a seguinte:

```
Var

P1, P2, Media: Real
```

A palavra reservada *Var* indica a seção de declarações de variáveis em um algoritmo, seguindo a sintaxe do *VisuAlg*, um ambiente visual para o estudo de algoritmos (PARREIRA *et al*., 2017) e estamos utilizando-o aqui neste texto apenas como exemplo. Cada ambiente e/ou linguagem de programação tem suas regras de sintaxe próprias. Sendo assim, a palavra *var* pode ou não ser utilizada em outros ambientes. Lembre-se que as variáveis não podem conter acentos.

Algumas observações importantes com relação às variáveis (SILVA *et al.,* 2010; PARREIRA *et al.* 2017):

1) O uso do nome de uma variável, dentro de uma expressão (fórmula), não altera o seu valor. Para alterar o valor de uma variável na memória é preciso utilizar um comando da linguagem de programação que é o comando de atribuição (comando que indica que desejamos atribuir ou definir um valor para a variável, ou seja, que queremos modificar o valor que está armazenado na memória cujo espaço é representado pelo nome desta variável);

2) O uso do nome de uma variável, dentro de uma expressão (fórmula) significa que está sendo recuperado o seu conteúdo na memória;

3) Uma variável recebe o valor que lhe está sendo atribuído, perdendo o valor anteriormente armazenado (se guardamos,

anteriormente, o valor 7 na variável P1 e, por meio de um comando de atribuição, damos ordem para que seja guardado o valor 8, a variável P1 manterá apenas o último valor, o 8, ou seja, uma variável armazena apenas um valor de cada vez, não sendo possível resgatar um histórico dos valores armazenados. Caso precisemos armazenar mais de um valor utilizando um mesmo nome de variável, precisaremos utilizar uma estrutura de dados, tais como os vetores – ou matrizes unidimensionais, que serão estudados mais adiante neste livro).

Esta declaração de variáveis informa ao computador que o programa precisa de 3 espaços de memória (como se fossem 3 gavetas), que serão reservados durante a execução do programa. Este processo é denominado de alocação de memória. Reservar os espaços (gavetas) significa que outros programas e outras variáveis não poderão usar aqueles mesmos espaços, para que não haja o problema de conflito entre os dados, ou seja, dados de diferentes programas compartilhando as mesmas gavetas.

Para que possamos estudar como um algoritmo (programa) é executado pelo computador vamos estudar o funcionamento básico do computador. A Figura 2 apresenta um esquema básico de funcionamento do *hardware,* baseado na arquitetura de *Von Neumann.*

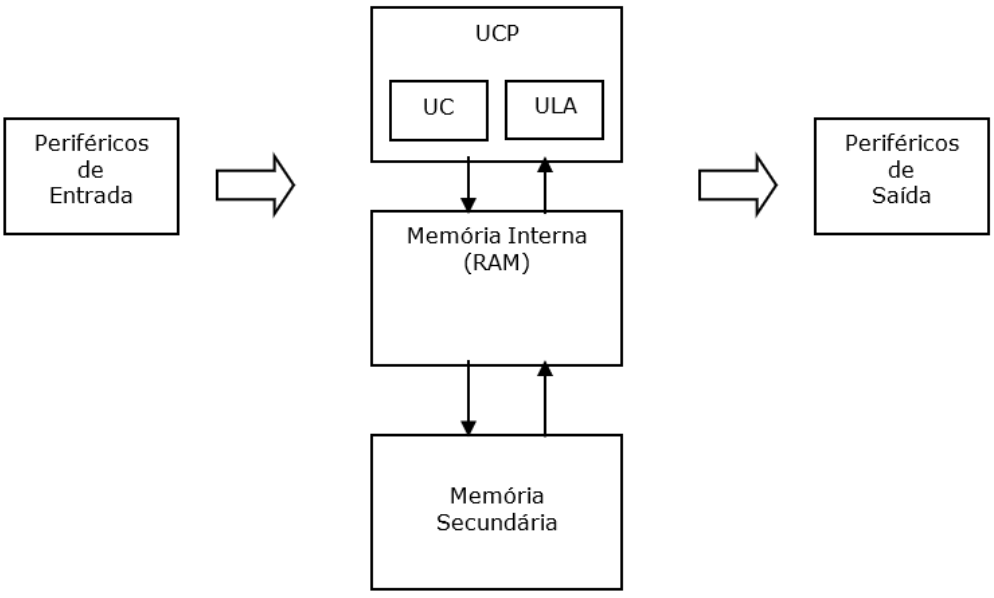


Figura 2 – Esquema Básico de Funcionamento do Computador de acordo com a Arquitetura de *Von Neumann*
(Fonte: Os autores)

Conforme apresenta a Figura 2, temos as 3 fases do processamento de dados (entrada, processamento e saída) representadas, respectivamente: 1) pelos periféricos de entrada; 2) Unidade Central de Processamento e Memória e 3) periféricos de saída.

Os **periféricos de entrada** são todos os dispositivos que permitem que uma informação do meio externo seja enviada para o computador. Por exemplo, quando utilizamos um caixa eletrônico em um banco, podemos enviar informações de diferentes formas. Quando inserimos o cartão magnético, as informações são lidas por meio de uma leitora de cartões. Quando digitamos nossa senha, podemos usar o teclado ou o vídeo (*touch screen).*

O **processamento** é executado por meio de duas unidades principais, que são a UCP (Unidade Central de Processamento ou CPU – *Central Process Unit*) e a Memória. A UCP tem dois componentes principais, que são a UC (Unidade de Controle) e a ULA (Unidade Lógico-Aritmética). A UC é responsável por acompanhar o fluxo de execução do algoritmo/programa, indicando qual é o próximo comando a ser executado. A ULA, por sua vez, é responsável por interpretar as expressões aritméticas e lógicas. A memória é responsável por armazenar o algoritmo (ou programa), bem como as variáveis utilizadas. Quando o programa é colocado em execução, os valores das variáveis são enviados à UCP para serem utilizados nas expressões e são devolvidos à memória quando os resultados são calculados. Por isso é que existem setas indicando a mão dupla entre a Memória e a UCP. Com relação à Memória, a figura ilustra as memórias interna e secundária. A memória interna (ou Memória RAM *Random Access Memory*) armazena as instruções do programa que está sendo executado, bem como as variáveis que estão sendo utilizadas. Esta memória é volátil, ou seja, o programa, bem como as variáveis, só ficam armazenados enquanto o programa estiver sendo executado. O código-fonte do programa precisa ser armazenado (salvo) em uma memória secundária, para que possa ser executado outras vezes. Como exemplos de memórias secundárias temos o HD (*Hard Disk* ou Disco Rígido) e o *pendrive*.

O resultado do processamento é apresentado por meio de **periféricos de saída.** Os periféricos de saída mais comuns são o vídeo ou tela e a impressora. Por exemplo, após executar o algoritmo que calcula a média do aluno, a média é armazenada na memória. Se o programador não incluir no algoritmo/programa um comando para apresentar a média na tela, o usuário final (quem irá utilizar o seu programa) não visualizará o resultado.

Detalhando ainda mais a execução de um algoritmo, vimos que envolve 3 etapas (SILVA *et al.,* 2010; PARREIRA *et al.* 2017):

1) Entrada de dados: nesta etapa, os dados são transmitidos de um meio externo para o computador. Normalmente, esta etapa é realizada por meio de um periférico de entrada. O periférico de entrada mais comum é o teclado mas, existem outros, tais como: *scanner*, leitora de códigos de barras, *touch screen*;

2) Processamento dos dados: esta etapa consiste na execução dos cálculos do programa, por meio da utilização dos dados armazenados na memória e da UCP. A memória interna e a UCP se comunicam, permitindo que os dados sejam enviados à ULA, processados e devolvidos para a memória, já com os resultados;

3) Saída de dados: esta etapa permite que os resultados obtidos sejam apresentados aos usuários, através de periféricos de saída. Os periféricos de saída mais comuns são o vídeo e a impressora.

No exemplo do algoritmo para calcular a média, estas 3 etapas ficariam assim distribuídas (SILVA *et al.*, 2010; PARREIRA *et al.* 2017):

1) Por meio de um periférico de entrada, como o teclado, o usuário informaria as notas de P1 e P2; estas notas seriam armazenadas na memória interna, nos espaços reservados;

2) Por meio da ULA, a média seria calculada de acordo com a fórmula adequada;

3) Por meio de um periférico de saída, como o vídeo, o usuário receberia o resultado, ou seja, a sua média.

Neste ponto cabe uma pergunta: Como é que o computador sabe que deve armazenar as notas de P1 e P2 na memória, calcular a média e depois mostrar o resultado? Estas ordens (comandos ou instruções) estão descrita s no nosso algoritmo (programa). Este programa deve ser armazenado na memória do computador e, quando colocado em execução, aciona os dispositivos do *hardware*, necessários para o funcionamento do programa. Sendo assim, a memória, além de armazenar os dados (de entrada e de saída) também armazena o programa (a série de passos que deve ser seguida para que se chegue à solução do problema).

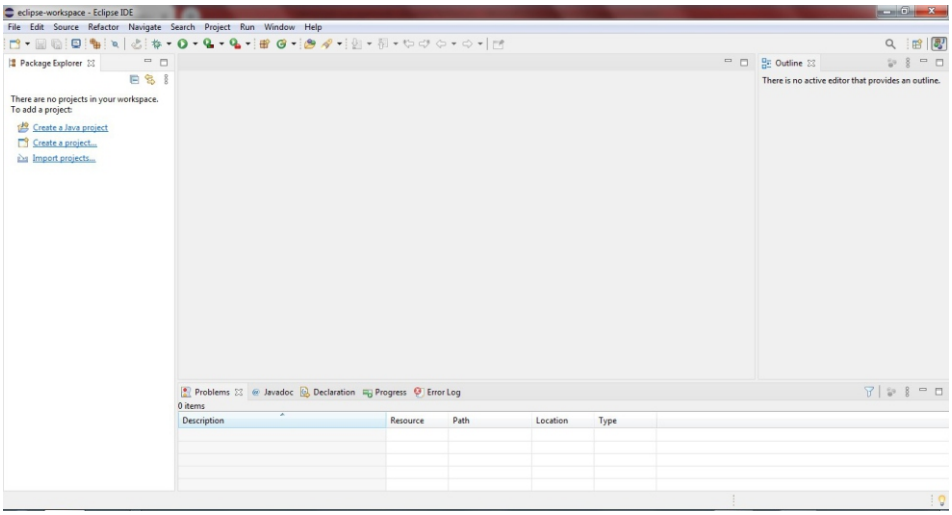
CAPÍTULO 2

Introdução à Programação em Python

Neste capítulo iremos estudar alguns conceitos introdutórios sobre a linguagem de programação Python, apresentando os principais comandos da linguagem, comparando-os aos comandos similares da linguagem de programação C.

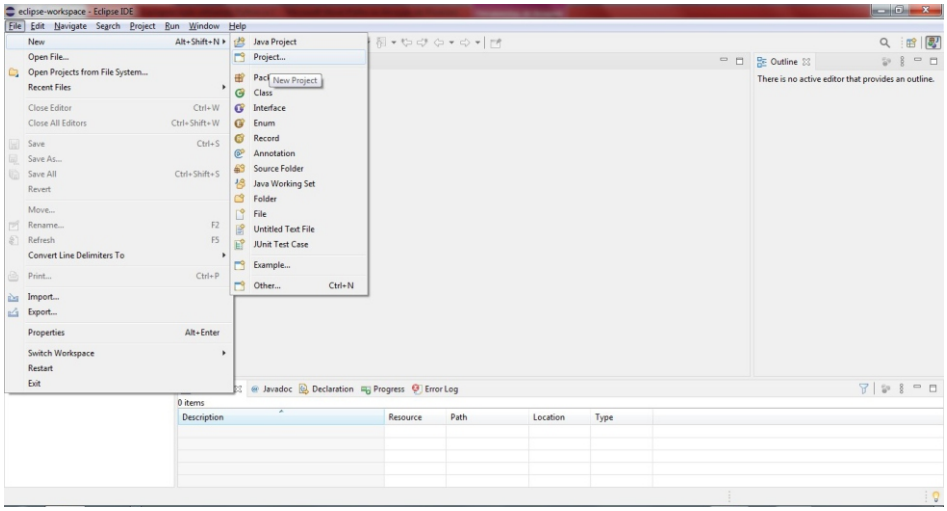
Vamos iniciar o estudo introdutório da linguagem de programação *Python* criando um primeiro programa, utilizando a IDE *Eclipse*, apresentada na Figura 3. Tradicionalmente, quando iniciamos o estudo de uma nova linguagem de programação (já conhecendo os conceitos de algoritmos e lógica de programação), criamos um programa denominado *Hello World* ou *Olá Mundo*. Para criarmos este programa vamos utilizar o comando print, para mostrarmos uma mensagem na tela.

Figura 3: Interface da IDE Eclipse (Fonte: Os autores)



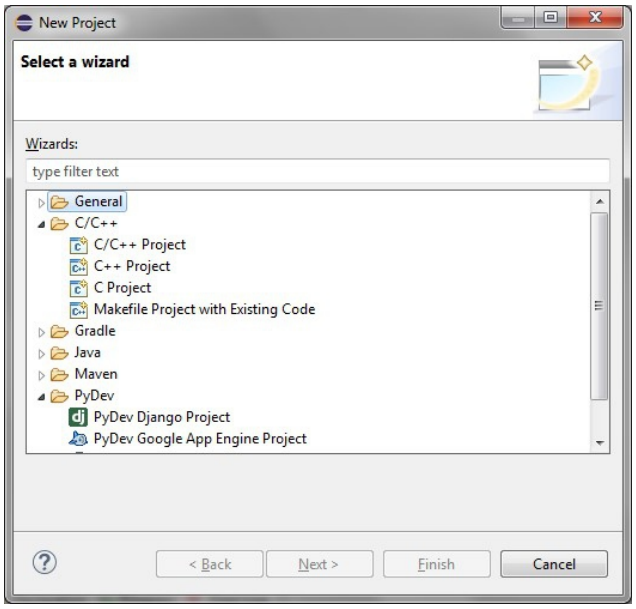
No Eclipse vamos selecionar a opção New -> Project no menu File, como mostra a Figura 4.

Figura 4: Menu *File* opção *New -> Project* (Fonte: Os autores)



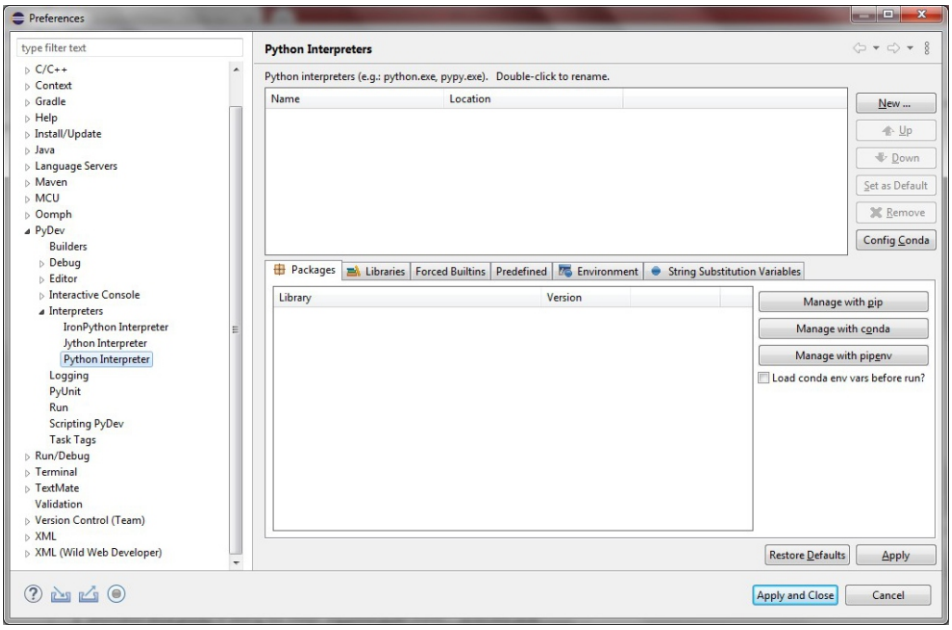
Como instalamos na nossa IDE as *features* para o desenvolvimento de programas nas linguagens C e *Python*, veremos as opções de diferentes projetos, tais como *C Project* e *PyDev Project* (como mostra a Figura 5).

Figura 5: Escolha do Tipo de Projeto na IDE *Eclipse* (Fonte: Os autores)



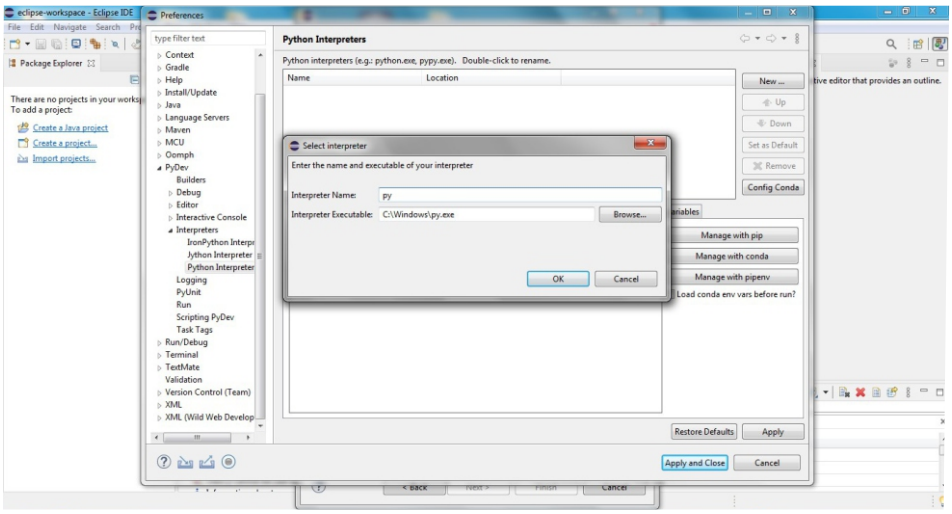
Vamos fazer o primeiro exemplo utilizando a linguagem de programação *Python*, a partir da criação de um novo projeto do tipo *PyDev Project*. Antes de criar o projeto é preciso configurar o interpretador de *Python* que será utilizado, por meio da opção *Python Interpreters* do menu *Windows -> Preferences* no *Eclipse*. Essa opção é apresentada na Figura 6.

Figura 6: Configuração do Interpretador *Python* (Fonte: Os autores)



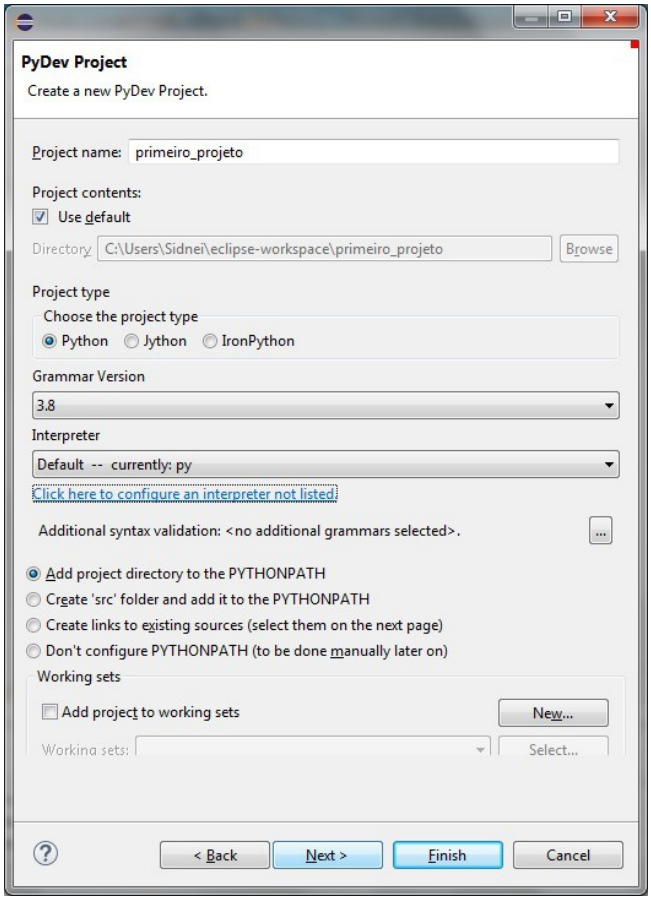
Ao selecionar esta opção na janela apresentada na Figura 6 devemos clicar no botão *New* e selecionar a pasta onde se encontra o arquivo executável do interpretador *Python* que está instalado no nosso computador, como mostra a Figura 7.

Figura 7: Localização (Pasta) do Interpretador *Python* (Fonte: Os autores)



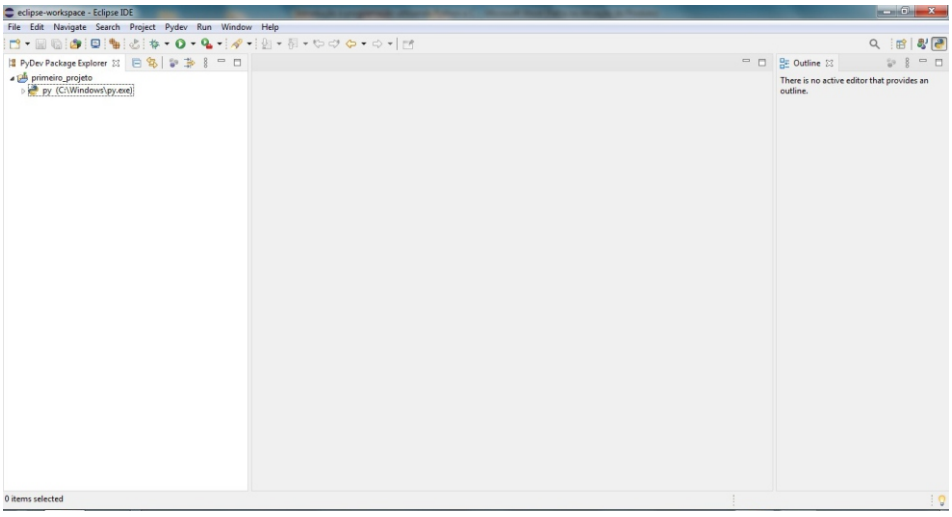
Após selecionarmos o tipo de projeto (conforme Figura 5) devemos clicar no botão *Next* e digitarmos um novo para o nosso projeto e clicarmos no botão *Finish*, como mostra a Figura 8 (escolhemos o nome *primeiro_projeto*).

Figura 8: Definição do Nome do Projeto (Fonte: Os autores)



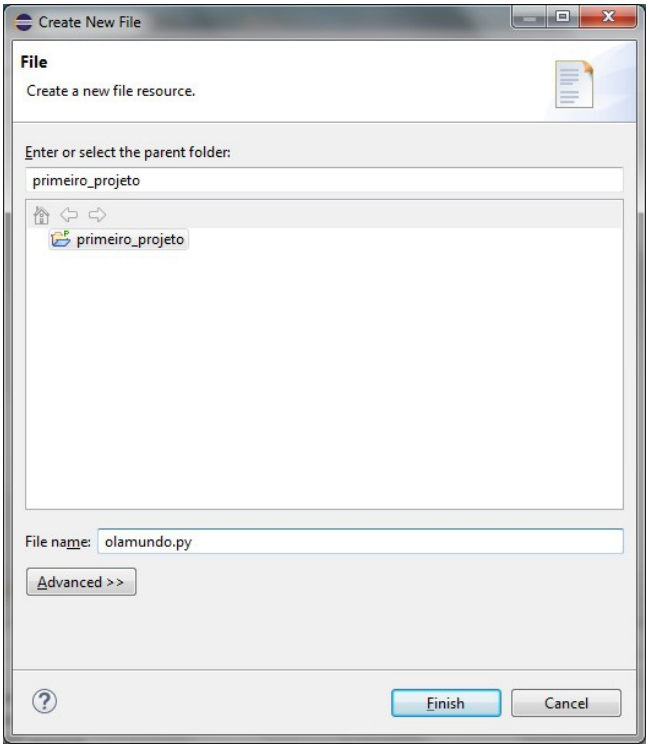
Após a criação do projeto que denominamos de *primeiro_projeto* veremos a tela apresentada na Figura 9.

Figura 9: Tela Inicial do Primeiro Projeto (Fonte: Os autores)



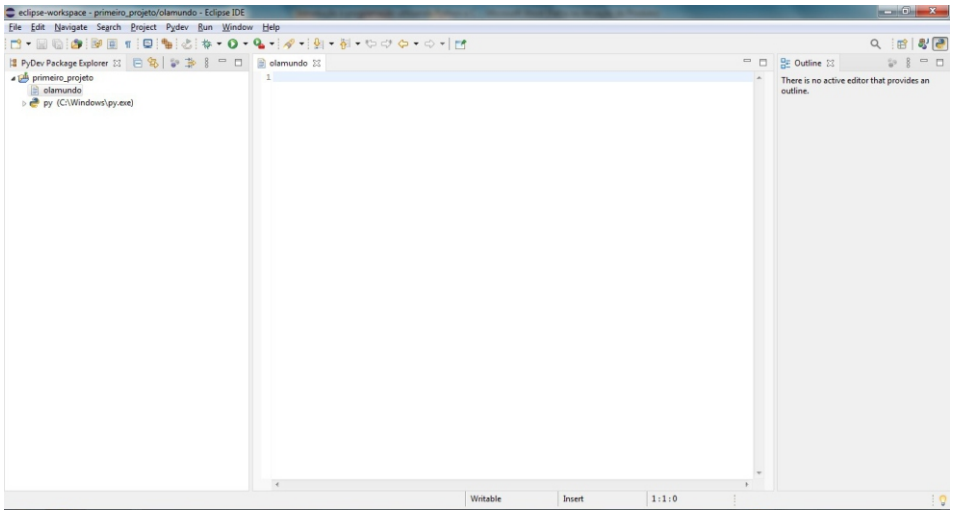
Agora, por meio da opção *New -> File* do menu *File*, vamos inserir um arquivo que conterá o código-fonte em *Python* do nosso primeiro programa. A Figura 10 mostra a janela que será aberta ao selecionarmos esta opção. Nesta janela devemos selecionar o nome do nosso projeto (*primeiro_projeto*) e definirmos o nome do nosso arquivo (utilizamos o nome *olamundo.py* como exemplo, sendo que *py* é a extensão dos arquivos contendo o código-fonte em *Python*).

Figura 10: Criação de um novo arquivo (Fonte: Os autores)



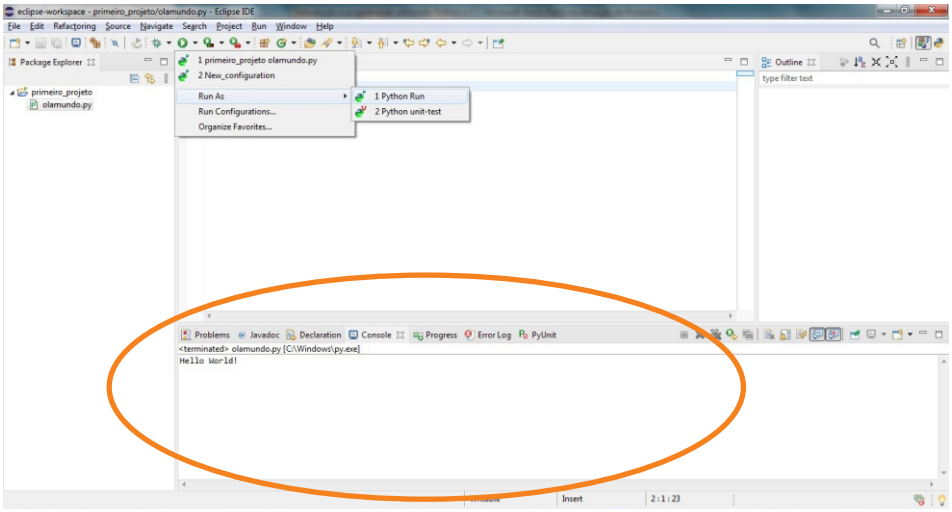
Agora temos um espaço na tela onde iremos digitar o código-fonte do nosso primeiro programa em *Python* (Figura 11).

Figura 11: Novo arquivo criado no projeto (Fonte: Os autores)



Vamos digitar o comando ***print("Hello World!")*** e executar o programa, usando a opção *Run As -> Python Run* do menu *Run*, como mostra a Figura 12. O resultado do comando *print* será apresentado na janela de console, na parte inferior da tela (destacada na Figura 12).

Figura 12: Execução do primeiro programa (Fonte: Os autores)



Comparando o código-fonte para mostrar a mensagem *Hello World* utilizando as linguagens de programação *Python* e C, como mostra o quadro 1, vemos que a linguagem *Python* é mais simples. Na Linguagem C precisamos incluir a biblioteca *Standard IO (Input-Output)* para utilizarmos o comando *printf* por meio da instrução *#include <stdio.h>* e, além disso, o programa precisa ter uma rotina principal (*main*), além do bloco de comandos delimitado pelas chaves.

Quadro 1 – Comparação dos Códigos-Fonte

Python	C
print("Hello World!")	*#include <stdio.h>* *main () {* *printf("Hello World!");* *}*

Fonte: Os autores

2.1 Características Importantes da Linguagem de Programação *Python*

Algumas características importantes, para iniciarmos o estudo da linguagem de programação *Python*, compreendem:

- não é necessário declarar as variáveis. Basta atribuir um valor para uma variável que a mesma será automaticamente declarada (declaração implícita) e assumirá o tipo de dado de acordo com o que for atribuído à mesma;

  Por exemplo:

  Valor =10

A variável *Valor* (ou identificador *Valor)* recebe o valor inteiro 10 e a variável passa a ter o tipo de dados inteiro atrelado à mesma

Nome = "José"

A variável *Nome* recebe a *string* (cadeia de caracteres) *José*

- a linguagem *Python* é *case sensitive*, ou seja, há diferenciação em utilizar letras maiúsculas ou minúsculas. Por exemplo: os identificadores *Valor* e *valor* não são iguais, pois *Valor* inicia com letra maiúscula e *valor* com letra minúscula. Neste caso são dois identificadores diferentes;

- a indentação faz parte da sintaxe da linguagem. Por exemplo, na instrução abaixo, há um erro de sintaxe, pois o comando *print* não está corretamente indentado, ou seja, não faz parte do comando *if*

If x>8:

print("x é maior do que 8")

O código-fonte correto é (o comando *print* tem que estar *dentro* do *if)*:

If x>8:

print("x é maior do que 8")

Após digitar o comando *if* e pressionar a tecla *Enter* o *Python* já irá indentar a próxima linha.

- nas expressões condicionais que compõem as instruções *if, for* e *while*, o uso de parênteses é opcional.

O Quadro 2 apresenta a comparação das principais características entre *Python* e C.

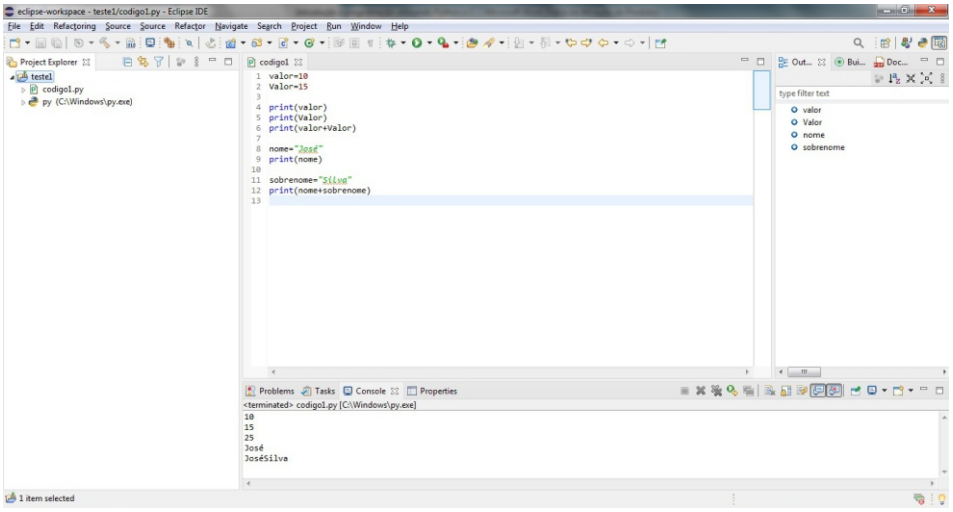
Quadro 2 – Comparação das Principais Características

Python	**C**
Não é necessário declarar as variáveis	A declaração de variáveis é obrigatória
A linguagem é *case sensitive* (diferencia letras maiúsculas e minúsculas)	A linguagem é *case sensitive*
A indentação faz parte da sintaxe	A indentação não faz parte da sintaxe (são utilizados os símbolos {} para os blocos de comandos)
O uso do ponto -e-vírgula (;) no final de uma instrução é opcional. Entretanto, se existir mais de uma instrução em uma mesma linha do código-fonte, o ponto -e-vírgula passa a ser obrigatório. Por exemplo: *print("Soma",2+2);print("Multiplicação",2*2)*	O uso do ponto -e-vírgula (;) no final de uma instrução é obrigatório

Fonte: Os autores

A Figura 13 mostra alguns exemplos de atribuição de valores a variáveis e impressão dos valores das mesmas, bem como de alguns resultados de expressões.

Figura 13: Exemplos em *Python* (Fonte: Os autores)



Analisando os exemplos da Figura 13, temos:

- atribuição do número inteiro 10 à variável *valor* (variável criada com letras minúsculas);

- atribuição do número inteiro 15 à variável *Valor* (note que são variáveis diferentes, pois o 2º nome valor está com a letra V em maiúsculo);

- impressão dos conteúdos das variáveis *valor* e *Valor* (resultados apresentados na janela de console, na parte inferior da tela);

- impressão da soma dos conteúdos das variáveis *valor* e *Valor;*

- atribuição do nome *José* à variável *nome;*

- atribuição do sobrenome *Silva* à variável *sobrenome;*

- impressão da soma (concatenação) das *strings nome* e *sobrenome.*

2.2 Operadores Aritméticos e Expressões Aritméticas

Os operadores aritméticos básicos da linguagem *Python* são: + (adição), - (subtração), * (multiplicação) e / (divisão). Estes operadores são os mesmos da linguagem de programação C. Podemos utilizar o comando *print* para mostrar o resultado de operações aritméticas. Por exemplo, vamos testar os comandos abaixo na IDE *Eclipse*:

print(2+2)

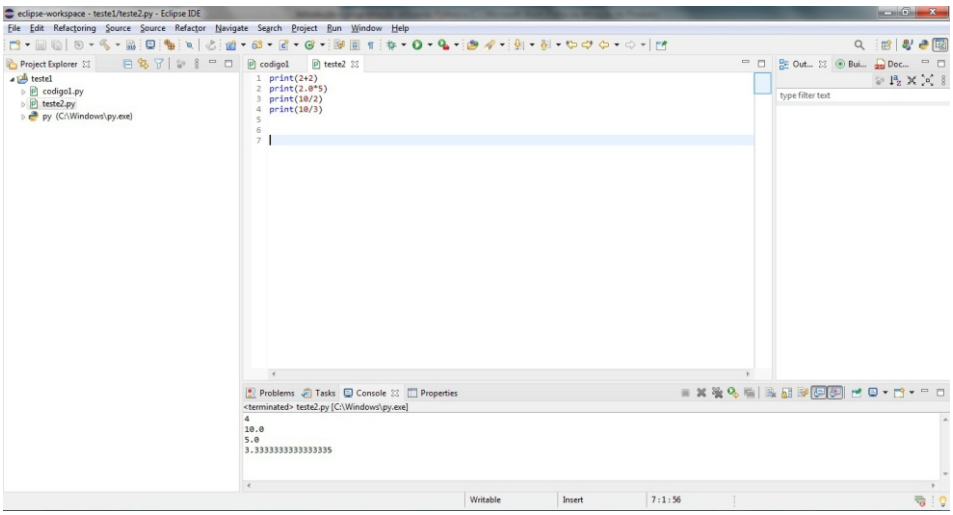
*print(2.0*5)*

print(10/2)

print(10/3)

A Figura 14 apresenta os comandos e o resultado da execução (na janela de console) na IDE *Eclipse.*

Figura 14: Exemplos de Operações Aritméticas (Fonte: Os autores)



Podemos colocar uma mensagem junto com as operações aritméticas, por exemplo:

*print("Resultado",2*4+5)*

print("Resultado",2(4+5))*

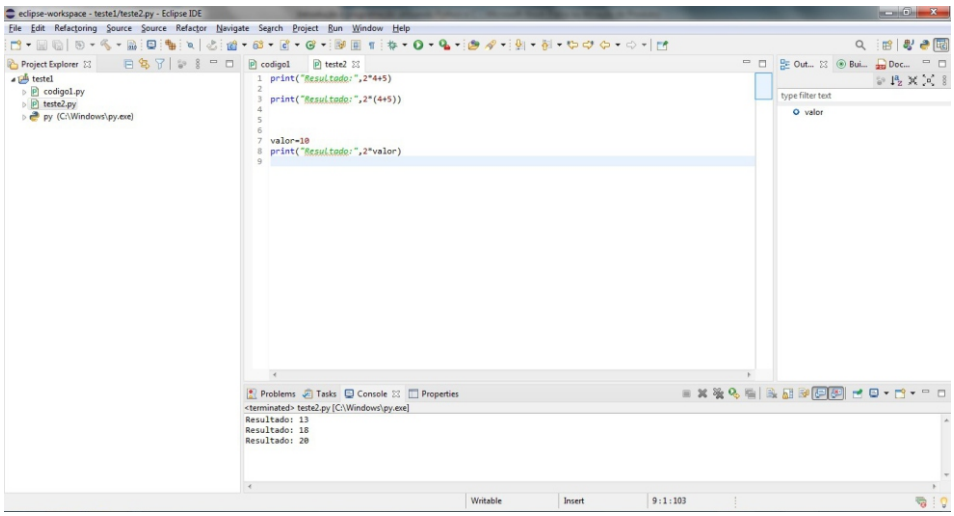
Também podemos utilizar variáveis na impressão dos resultados das expressões aritméticas. Por exemplo:

valor=10

*print("Resultado:",2*valor)*

A Figura 15 apresenta os resultados dos exemplos executados na IDE *Eclipse*.

Figura 15: Exemplos de Operações Aritméticas (Fonte: Os autores)



Quadro 3 – Comparação das Principais Características

Python	**C**
print(2+2)	*#include <stdio.h>* *main () {* *printf("%i",2+2);* *}*
*print(2.0*5)*	*#include <stdio.h>* *main () {* *printf("%f",2.0*5);* *}*
print(10/2)	*#include <stdio.h>* *main () {* *printf("%i",10/2);* *}*

Continua

Continuação

Python	**C**
print(10/3)	*#include <stdio.h>* *main () {* *printf("%d",10/3);* *}*
*print("Resultado",2*4+5)*	*#include <stdio.h>* *main () {* *printf("Resultado:%d",2*4+5);* *}*
print("Resultado",2(4+5))*	*#include <stdio.h>* *main () {* *printf("Resultado:%d",2*(4+5));* *}*

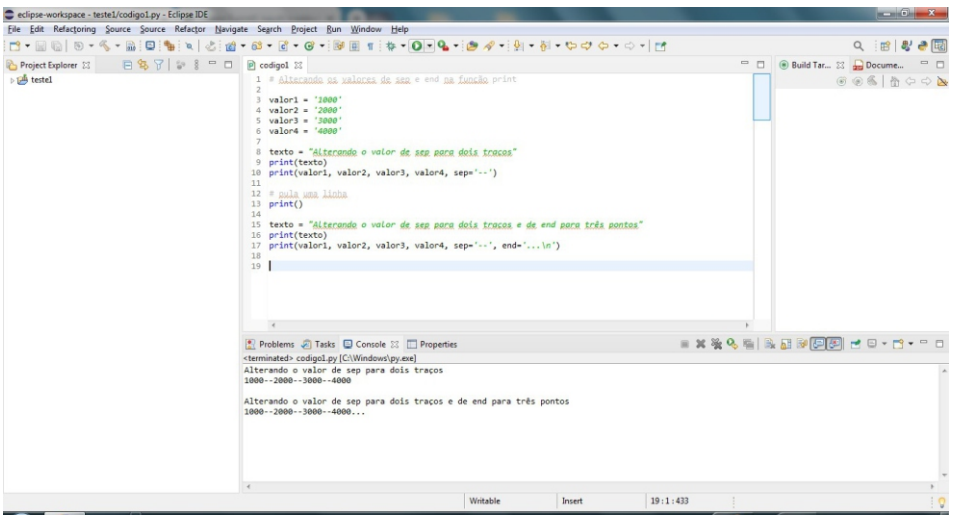
Fonte: Os autores

Analisando os exemplos de código-fonte em linguagem C, apresentados no Quadro 2, vemos que é preciso formatar a saída de dados do comando *printf* utilizando o símbolo % seguido do tipo de dados que será impresso, tal como *d* para um valor decimal inteiro, ou *f* para um valor fracionário (*float* ou *double*). Este mesmo tipo de formatação também pode ser utilizado na linguagem *Python*, como veremos em um exemplo mais adiante nesta seção.

Podemos utilizar os argumentos *sep* e *end* para formatar a saída de um comando *print*. O argumento *sep* coloca um espaço em branco (valor padrão) entre os argumentos que serão impressos, ou outro símbolo que podemos definir. O valor padrão do argumento *end* é uma nova linha "\n". Sendo assim, o comando *print* adiciona uma nova linha depois da impressão mas podemos definir outro valor, tal como uma tabulação por exemplo, utilizando "\t".

A Figura 16 apresenta o resultado da execução de um exemplo alterando o valor de *sep* para dois traços (--) e o valor de *end* para três pontos (...).

Figura 16: Utilização dos argumentos sep e end com o comando print (Fonte: Os autores)



Vamos fazer um exemplo utilizando o símbolo % para formatar a saída do comando *print*. Vamos fazer o cálculo da raiz quadrada, utilizando uma função matemática disponível na biblioteca *Math.* A biblioteca *Math* permite a utilização de diversas funções matemáticas, tais como os cálculos de raiz quadrada, seno e cosseno, por exemplo. Para utilizar esta biblioteca é preciso importá-la, por meio do comando *import math*.

Por exemplo, supondo que queiramos calcular a raiz quadrada de um número, podemos utilizar o código-fonte a seguir.

```
import math

numero=input('Digite um número:')

numero=int(numero)

print('A raiz quadrada de ',numero,' é igual a ', math.sqrt(numero))
```

Supondo que o usuário digite o valor 10 na entrada de dados (*input*), será mostrado o seguinte resultado:

```
A raiz quadrada de 10 é igual a 3.1622776601683795
```

Podemos formatar a saída do comando *print*, definindo o número de casas (algarismos) antes e depois da vírgula (ponto decimal). Por exemplo:

```
print('A raiz quadrada de ',numero, ' é igual a %3.2f.' %math.sqrt(numero))
```

onde *%3.2f.* significam 3 algarismos antes do ponto decimal, 2 após o ponto decimal e *f* significa um número do tipo *float*. A impressão do resultado ficará assim:

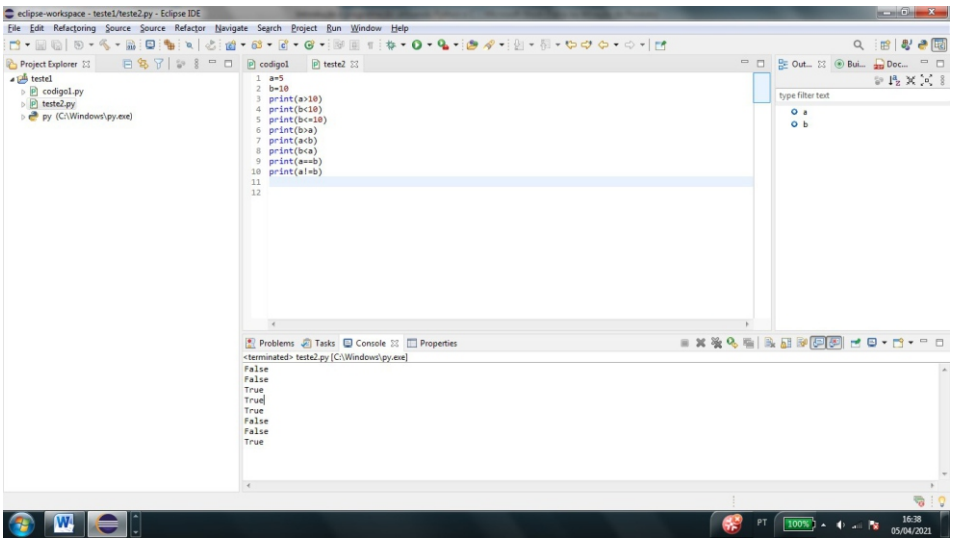
```
A raiz quadrada de 10 é igual a 3.16.
```

2.3 Operadores Relacionais e Lógicos

Os operadores relacionais da linguagem *Python* são: > (maior que), >= (maior ou igual), < (menor que), <= (menor ou igual), == (igual), e != (diferente), sendo os mesmos da linguagem de programação C (SILVA *et al.*, 2010). Vejamos alguns exemplos de aplicação na Figura 17. Na janela de console temos apresentados os resultados da avaliação das respectivas expressões lógicas (*True* ou *False*).

De acordo com o código-fonte, temos a variável *a* com o valor 5 e a variável *b* com o valor 10. Para cada expressão lógica utilizamos o comando *print* para mostrar o resultado da sua avaliação. Por exemplo: *a>10* resulta *False* pois a expressão lógica é falsa, já que o valor da variável *a* é menor do que 10.

Figura 17: Exemplos de Aplicação dos Operadores Relacionais em *Python* (Fonte: Os autores)



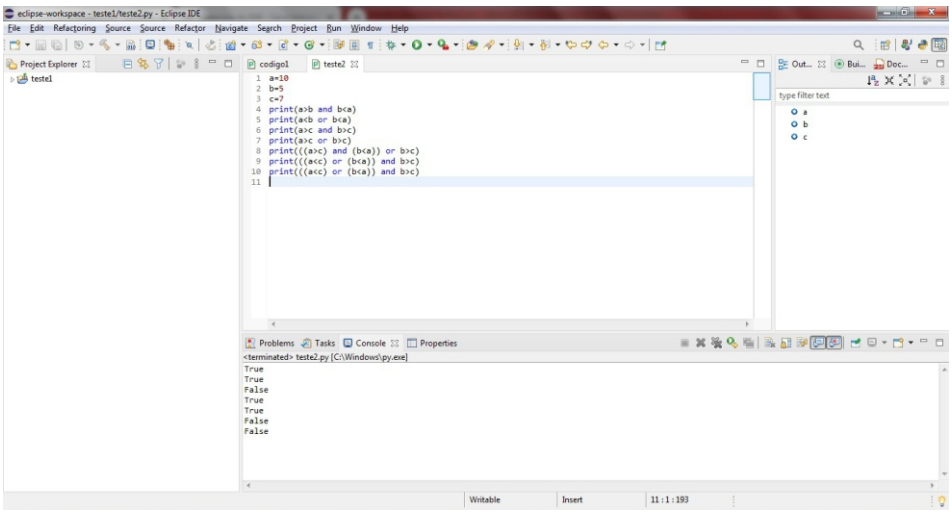
Devemos lembrar que o símbolo de igualdade (=) não significa uma operação de comparação e, sim, de atribuição de um valor a uma variável. Por exemplo:

a=10 # o valor 10 está sendo atribuído à variável *a*

a==10 # estamos perguntando (comparando), se o valor da variável *a* é igual a 10

Os operadores lógicos mais comuns são o *and* e o *or* e seu funcionamento é igual ao de outras linguagens de programação. Quando o *and* (e) é utilizado, todas as condições devem ser satisfeitas para que a expressão lógica retorne verdadeiro (*True*). Utilizando o operador lógico *or* (ou), pelo menos uma das condições deve ser satisfeita para que a expressão lógica retorne *True*. Veja os exemplos da Figura 18, em que utilizamos o comando *print* para mostrar o resultado da avaliação das expressões lógicas.

Figura 18: Exemplos de Aplicação dos Operadores Relacionais e Lógicos em *Python* (Fonte: Os autores)



Analisando os exemplos da Figura 18 e os resultados da avaliação das expressões lógicas, apresentadas na janela de console, temos (sendo que *a* é igual a 10, *b* é igual a 5 e c tem armazenado o valor 7):

- *a>b and b<a*: a é maior que b – *True* e b<a – *True*, *True and True* resulta *True*, de acordo com a tabela verdade (SILVA *et al.*, 2010);

- *a>b or b<a*: a é maior que b – *True* e b<a *True*, *True or True* resulta *True;*

- *a>c and b>c*: a é maior que c – *True* e b é maior que c – *False*, *True and False* retorna *False;*

- *a<c or b<c*: a é maior que C – *True* e b é menor do que c – *True*, *True or True* retorna *True*;

- *(a>c and b<a) or b>c*: neste exemplo usamos os parênteses para delimitar (ou definir a ordem de avaliação, pois as expressões entre parênteses serão avaliadas primeiro) as expressões, ou seja, a resposta da primeira expressão com *and* será comparada com a segunda expressão com *or*: a é maior que C – *True* e b é menor do que a também resulta *True – True and True* retorna *True.* A expressão b é maior que c retorna *False*. Finalmente, *True or False* retorna *True;*

- *(a<c or b<a) or b>c*: neste exemplo usamos os parênteses para delimitar as expressões, ou seja, a resposta da primeira expressão com *or* será comparada com a segunda expressão, também usando *or*: a é menor que C – resulta *False* e b é menor do que a, *True. False or True* retorna *True.* A expressão b é maior que c retorna *False*. Finalmente, *True or False* retorna *True;*

- *(a<c or b<a) and b>c*: neste exemplo usamos os parênteses para delimitar as expressões, ou seja, a resposta da primeira expressão com *or* será comparada com a segunda expressão, usando *and*: a é menor que C (*False*) e b é menor do que a (*True*) – *False or True* retorna *True.* A expressão b é maior que c retorna *False*. Finalmente, *True and False* retorna *False*.

O Quadro 4 apresenta a comparação dos exemplos de expressões lógicas entre *Python* e C.

Quadro 4 – Comparação das Expressões Lógicas Exemplificadas

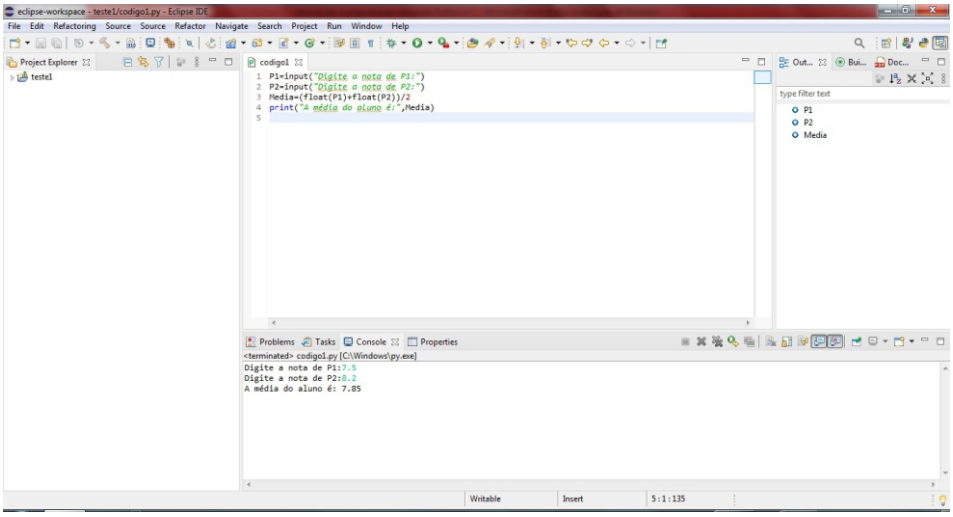
Python	**C**
a>b and b<a	(a>b && b<a)
a>b or b<a	(a>b || b<a)
a>c and b>c	(a>c && b>c)
a>c or b>c	(a>c || or b>c)
(a>c and b<a) or b>c	((a>c && b<a) || b>c)
(a<c or b<a) or b>c	((a<c || b<a) || b>c)
(a<c or b<a) and b>c	((a<c || b<a) && b>c)

Fonte: Os autores

2.4 Entrada e Saída de Dados

Já vimos nos exemplos deste capítulo que o comando *print* é um dos comandos utilizados em *Python* para a saída de dados, pois temos apresentado os resultados dos exemplos na janela de console da IDE *Eclipse*. Vamos relembrar o exemplo do algoritmo para calcular a média de um aluno, utilizado no capítulo 1. Para calcular a média, precisamos das duas notas do aluno (P1 e P2) e, para tal, precisamos utilizar um comando para a entrada de dados. Vamos utilizar, então, o comando *input* que permite a digitação dos valores na janela de console. Os valores digitados por meio do comando *input* são do tipo *string* em *Python*. Sendo assim, para utilizarmos as notas de P1 e P2 no cálculo da média, precisaremos convertê-las para o tipo numérico desejado. Como as notas de um aluno podem conter casas decimais, vamos utilizar o tipo *float*. Na Figura 19 apresentamos o código-fonte e a execução com um exemplo de cálculo da média para um aluno que obteve as notas 7.5 para P1 e 8.2 para P2.

Figura 19: Exemplo de entrada de dados com o comando *input* (Fonte: Os autores)



O código-fonte, apresentado de forma detalhada, é o seguinte:

P1=input("Digite a nota de P1:") # entrada de dados – permite que o usuário informe, na janela de console da IDE *Eclipse*, a nota obtida na 1ª avaliação e a armazena na variável P1 (o símbolo de igualdade significa a atribuição à variável)

P2=input("Digite a nota de P2:") # segunda entrada de dados – armazena a nota digitada na variável P2

Media=(float(P1)+float(P2))/2 # realiza o cálculo da média aritmética. Como P1 e P2, por terem sido informadas pelo comando *input*, são do tipo *string*, não podem ser utilizados em uma expressão aritmética com divisão. Sendo assim, utilizando a função *float*, convertemos os valores de P1 e P2 para um valor numérico com casas decimais. Se os

valores fossem apenas inteiros poderíamos utilizar a função *int*. Como a divisão por 2 só deve ocorrer após a soma das duas notas, colocamos a soma entre parênteses, para definir a ordem de execução do cálculo.

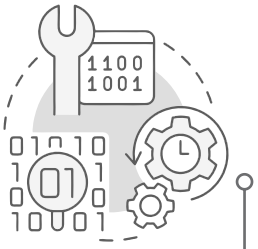
print(*"A média do aluno é:"*,Media) # saída de dados, onde apresentamos a mensagem "A média do aluno é:" e a média que foi calculada. Note que usamos a primeira letra maiúsculas nas variáveis. Lembre-se que a linguagem Python é *case sensitive.*

O Quadro 5 apresenta a comparação do exemplo para calcular a média codificada em *Python* e em C.

Quadro 5 – Comparação do Código-Fonte para Calcular a Média

Python	**C**
P1=input("Digite a nota de P1:") *P2=input("Digite a nota de P2:")* *Media=(float(P1)+float(P2))/2* *print("A média do aluno é:",Media)*	*#include <stdio.h>* *main() {* *float P1, P2, Media;* *scanf("%f", &P1);* *scanf("%f",&P2);* *Media=(P1+P2)/2;* *printf("A média do aluno é: %2.f \n", Media);* *}*

Fonte: Os autores



CAPÍTULO 3

Principais comandos da Linguagem Python

3.1 Expressões Condicionais e Comandos de Seleção

Por meio de expressões condicionais (ou condições), os desenvolvedores de *software* podem definir condições para que um determinado comando (ou blocos de comandos) sejam ou não executados. Por exemplo, se pensarmos no programa para calcular a média de um aluno podemos definir uma condição para mostrar se o aluno está ou não aprovado e, em caso de não estar aprovado, definir uma ação que deva ser tomada (fazer uma prova de recuperação ou uma exame final). Um dos comandos mais comuns, em diferentes linguagens de programação, utilizado para avaliar o resultado de expressões condicionais é o comando *se* (*if*) e existem diferentes formas de utilização, como veremos nesta seção.

3.1.1 Seleção Simples

Uma seleção simples ocorre quando verificamos apenas se a expressão condicional (ou condição) é verdadeira, sendo executado o comando (ou bloco de comandos) logo abaixo da condição. Por exemplo, no comando abaixo, se o resultado da avaliação da expressão condicional *condicao1* for verdadeiro, será executado o comando definido após os dois pontos. Caso contrário, se o resultado for falso, o fluxo de execução do programa continuará na próxima instrução, na linha seguinte à *comando*. A sintaxe do comando *if* em *Python* é apresentada a seguir. Lemos o comando da seguinte forma: *se a condicao1 for verdadeira, então comando será executado.*

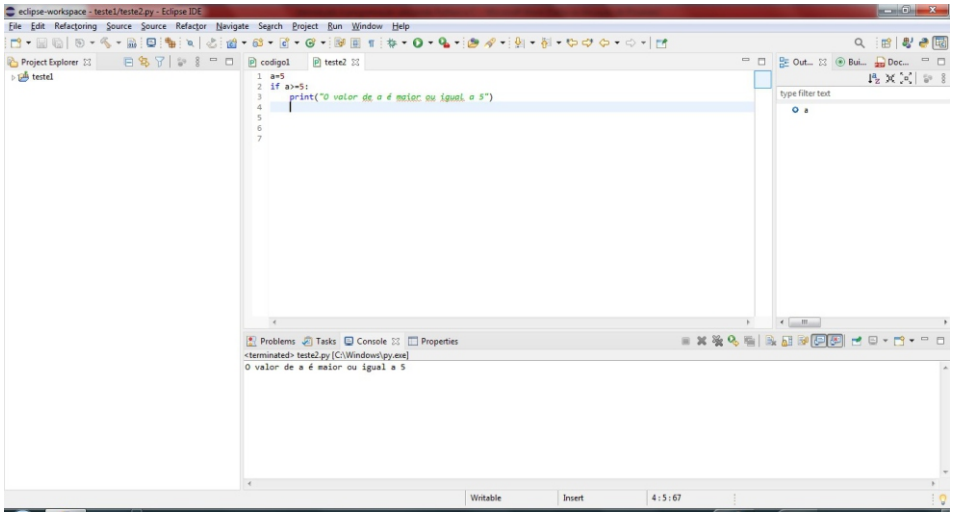
```
if condicao1:
    comando
```

Se o resultado da *condicao1* for *True* o *comando* da linha abaixo, desde que esteja corretamente indentado, será executado. Lembre-se que, em *Python,* a indentação faz parte das regras de sintaxe da linguagem. Por exemplo:

```
a=5
if a>=5:
  print("o valor de a é maior ou igual a 5")
```

A Figura 20 mostra o exemplo de seleção simples executado na IDE *Eclipse*. Como o resultado da expressão é *True*, já que a variável "a" tem o valor igual a 5, a mensagem do comando *print* é exibida na janela de console.

Figura 20: Exemplo de seleção simples (Fonte: Os autores)



O Quadro 6 apresenta a comparação do exemplo codificado em *Python* e em *C*.

Quadro 6 – Comparação do Código-Fonte para Calcular a Média

Python	**C**
a=5 *if a>=5:* *print("O valor de a é maior ou igual a 5")*	*#include <stdio.h>* *int a;* *main () {* *a=5;* *if (a>=5) {* *printf("O valor de a é maior ou igual a 5");* *}* *}*

Fonte: Os autores

3.1.2 Seleção Composta

Uma seleção composta ocorre quando verificamos se a expressão condicional (ou condição) é verdadeira ou falsa. Sendo assim, será executado o comando (ou bloco de comandos) logo abaixo da condição, caso a mesma seja verdadeira (*comando1* no exemplo abaixo) ou após o *else*, se a condição for falsa. Por exemplo, no comando abaixo, se o resultado da avaliação da expressão condicional *condicao1* for verdadeiro, será executado o comando definido após os dois pontos. Caso contrário, se o resultado for falso, o fluxo de execução do programa continuará na próxima instrução, na linha seguinte à *comando*. A sintaxe do comando *if* em *Python* é apresentada a seguir. Podemos ler essa instrução da seguinte forma: *se a condicao1 for verdadeira então o comando1 será executado, senão, caso a condicao1 seja falsa, o comando2 será executado*. Seguindo as regras de sintaxe da linguagem *Python*, devemos colocar dois pontos (:) no final da expressão condicional e depois do comando *else* também.

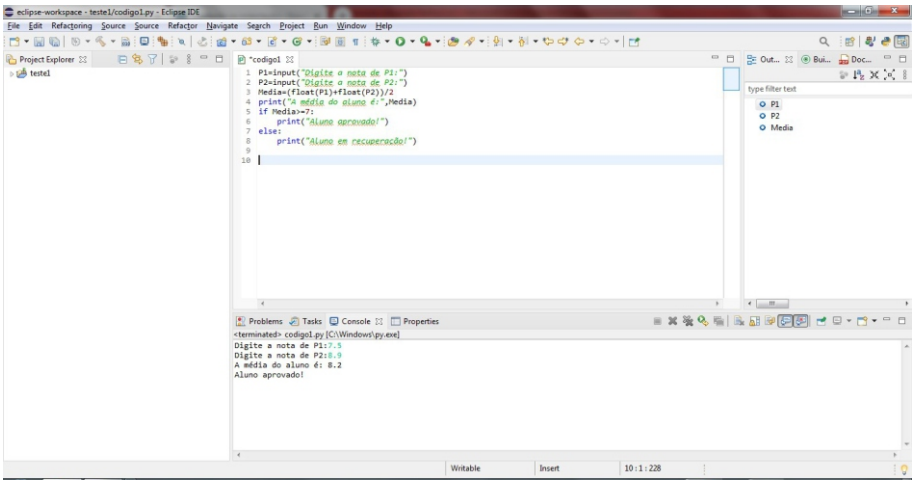
```
if condicao1:
    comando1
else:
    comando2
```

Por exemplo, se fizermos uma expressão condicional para avaliarmos se um aluno foi aprovado, utilizando a seleção composta, teríamos:

```
if media>=7:
     print("Aluno aprovado")
else:
     print("Aluno em recuperação")
```

Inserimos este comando de seleção composta no exemplo que fizemos no capítulo anterior (Figura 19). A Figura 21 apresenta o resultado da execução deste exemplo modificado.

Figura 21: Exemplo de seleção composta (Fonte: Os autores)



O Quadro 7 apresenta a comparação do exemplo da Figura 21 codificado em *Python* e em *C*.

Quadro 7 – Comparação do Código-Fonte para Calcular a Média

Python	**C**
P1=input("Digite a nota de P1:") *P2=input("Digite a nota de P2:")* *Media=(float(P1)+float(P2))/2* *print("A média do aluno é:",Media)* *if Media>=7:* *print("Aluno aprovado!")* *else:* *print("Aluno em recuperação!")*	*#include <stdio.h>* *main() {* *float P1, P2, Media;* *scanf("%f", &P1);* *scanf("%f",&P2);* *Media=(P1 +P2)/2;* *printf("A média do aluno é: %2.f \n", Media);* *if (Media>=7) {* *printf("Aluno aprovado!");* *}* *else {* *printf("Aluno em recuperação!");* *}* *}*

Fonte: Os autores

3.1.3 Seleção de Múltipla Escolha

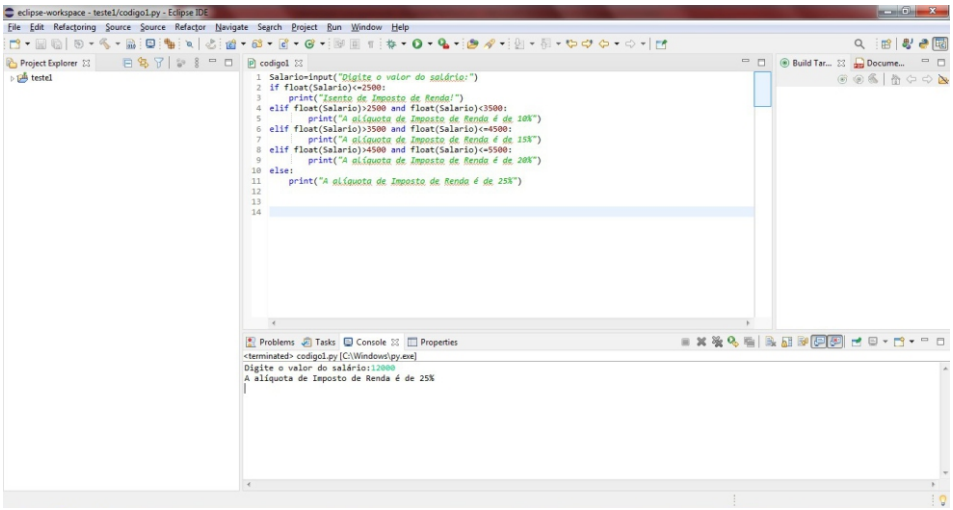
Uma seleção de múltipla escolha ocorre quando queremos verificar várias condições verdadeiras em um mesmo comando de seleção, ou seja, é utilizada para a verificação de vários casos. Em *Python* utilizamos, para uma seleção de múltipla escolha, os comandos *if* e *elif*, com a seguinte sintaxe:

```
if condicao1:
   comando1
elif condicao2:
    comando2
elif condicao3:
    comando3
else:
 comando4
```

Se a *condicao1* for verdadeira, será executado o *comando1.* Se a *condicao2* for verdadeira, será executado o *comando2* e assim por diante, testando as condições verdadeiras para todos os comandos *elif*. Caso nenhuma das condições seja verdadeira será executado o comando após *else*.

Por exemplo, vamos criar um programa que apresenta o percentual da alíquota de Imposto de Renda de acordo com o salário de uma pessoa, seguindo a tabela hipotética em que: até R$2.500,00 o contribuinte está isento de Imposto de Renda, acima de R$2.500,00 até R$3.500,00 a alíquota é de 10%, entre R$3.501,00 e R$4.500,00 a alíquota é de 15%, entre R$4.501,00 e R$5.500,00 a alíquota é de 20% e, acima de R$5.500,00 a alíquota é de 25%. Utilizando a seleção de múltipla escolha temos o seguinte código-fonte, apresentado na Figura 22 (note que não estamos fazendo o cálculo do valor do imposto a pagar e, sim, apenas mostrando qual o percentual de alíquota devido). Lembre-se que os valores informados por meio do comando *input* são do tipo *string*. Sendo assim, para testarmos o valor da variável *Salario* nas condições, fazemos a conversão da mesma para o tipo *float*.

Figura 22: Exemplo de seleção de múltipla escolha (Fonte: Os autores)



Na linguagem de programação C o comando *swith* pode ser utilizado para realizar uma seleção de múltipla escolha. Entretanto, este comando permite a utilização de valores do tipo inteiro (*int*) ou caracter (*char*), e o salário que utilizamos no exemplo é um valor do tipo *float*. Sendo assim, vamos fazer um outro exemplo para comparar a seleção de múltipla escolha em *Python* e C. Vamos supor que, por meio de uma nota, entre 1 e 5, podemos representar o conceito de um curso. O Quadro 8 apresenta o código-fonte deste exemplo de seleção de múltipla escolha, comparando as linguagens de programação *Python* e C. Como a variável *nota*, na linguagem *Python*, é informada por meio do comando *input*, o seu tipo de dados é *string*. Para utilizarmos na comparação dos conceitos, poderíamos manter a nota como *string* ou convertermos para um número inteiro (utilizando *int*) como fizemos no exemplo apresentado no Quadro 8.

Quadro 8 – Comparação do Exemplo de Seleção de Múltipla Escolha

Python	**C**
nota=input("Digite a nota do curso:") *if int(nota)==1:* *print("Conceito Insuficiente")* *elif int(nota)==2:* *print("Conceito mínimo")* *elif int(nota)==3:* *print("Conceito médio")* *elif int(nota)==4:* *print("Conceito Bom")* *elif int(nota)==5:* *print("Conceito Muito Bom")* *else:* *print("Nota não classificada nos conceitos")*	*#include <stdio.h>* *int nota;* *main () {* *printf("Digite a nota do curso:");* *scanf("%i",&nota);* *switch(nota) {* *case 1:* *printf("Conceito insuficiente");* *break;* *case 2:* *printf("Conceito mínimo");* *break;* *case 3:* *printf("Conceito médio");* *break;* *case(4):* *printf("Conceito Bom");* *break;* *case(5):* *printf("Conceito Muito Bom");* *break;* *default:* *printf("Nota não classificada nos conceitos");* *}* *}*

Fonte: Os autores

3.2 Comandos de Repetição

3.2.1 Comando For

O comando *for* permite percorrer os itens de uma coleção (por exemplo, uma *string* ou uma estrutura de dados do tipo *lista*). Para cada um dos itens da coleção, será executado o comando declarado na estrutura de repetição (*loop*). Ao percorrer a lista de valores, a variável definida no comando *for* receberá, a cada iteração, um dos itens da coleção. A sintaxe do comando *for* em *Python* é:

for variavel in lista:

comandos

Por exemplo, utilizando o seguinte trecho de código:

```
nomes=['Ana','João','José','Maria']

for n in nomes:

    print(n)
```

Inicialmente criamos uma lista, denominada *nomes*, contendo quatro nomes (*Ana, João, José* e *Maria).* Após, utilizando o comando *for*, vamos percorrer toda a lista e, a cada iteração, um dos nomes, sequencialmente, será armazenado na variável *n*, que será impressa na tela. O resultado da execução deste trecho de código será a impressão dos nomes conforme estão armazenados na lista: *Ana, João, José* e *Maria.*

Podemos incluir o comando *else* ao final do comando *for.* Assim, um comando ou bloco de comandos será executado ao final da iteração. Por exemplo:

```
nomes=['Ana','João','José','Maria']

for n in nomes:

    print(n)

else:

   print("A lista de nomes foi impressa com sucesso!")
```

Também podemos usar o comando *for* para escrever todas as letras de um *string* (como se fosse soletrar as letras). Isso é possível porque uma *string* é uma lista de caracteres. Por exemplo:

```
for caracteres in 'Python':

    print('Letra:', caracteres)
```

Este trecho de código exibirá, como resultado:

```
Letra: P

Letra: y

Letra: t

Letra: h

Letra: o

Letra: n
```

Outra forma de utilização do comando *for* é por meio de intervalos de repetição, definidos por *range*. Por exemplo:

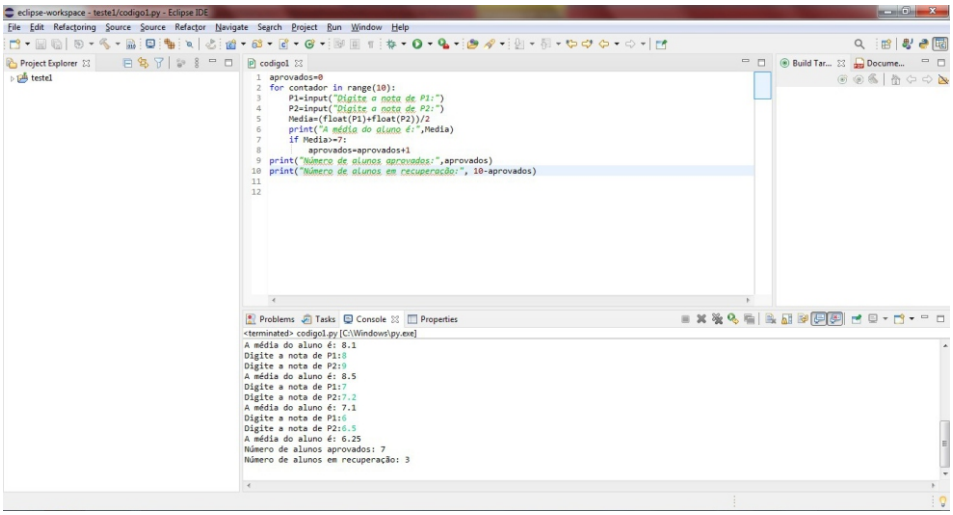
```
for i in range(3):
  print(i)
```

Este laço de repetição será executado três vezes, no intervalo de 0 a 2, sendo i igual a 0 na primeira interação.

Vamos fazer um exemplo a partir do programa para calcular a média de um aluno, implementado anteriormente. Vamos utilizar o comando *for* para calcular a média de um grupo de 10 alunos e, ao final, mostrar o número de alunos aprovados (média >=7) e o número de alunos em recuperação.

A Figura 23 apresenta esse exemplo na IDE *Eclipse*.

Figura 23: Exemplo com o comando *for* (Fonte: Os autores)



O Quadro 9 apresenta a comparação do exemplo da Figura 23 em *Python* com a linguagem de programação C.

Quadro 9 – Comparação do Exemplo com o Comando *For*

Python	**C**
aprovados=0 *for contador in range(10):* *P1=input("Digite a nota de P1:")* *P2=input("Digite a nota de P2:")* *Media=(float(P1)+float(P2))/2* *print("A média do aluno é:",Media)* *if Media>=7:* *aprovados=aprovados+1* *print("Número de alunos aprovados:",ap rovados)* *print("Número de alunos em recuperação:", 10 - aprovados)*	*#include <stdio.h>* *main() {* *float P1, P2, Media;* *int contador, aprovados;* *for (contador=1; contador<11; contador++) {* *printf("Digite a nota de P1:");* *scanf("%f", &P1);* *printf("Digite a nota de P2:");* *scanf("%f",&P2);* *Media=(P1+P2)/2;* *printf("A média do aluno é: %.2f \n", Media);* *if (Media>=7) {* *aprovados=aprovados+1;* *}* *}* *printf("Número de alunos a provados %d:",aprovados);* *printf("Número de alunos em recuperação %d:",10-aprovados);* *}*

Fonte: Os autores

3.2.2 Comando While

O comando *while* faz com que um trecho de código seja repetido enquanto uma condição for verdadeira. Quando o resultado da condição for falso, a execução da repetição é interrompida (saindo do *loop*) e passando para o próximo comando após o *while*. A sintaxe do comando *while* em *Python* é:

```
while (condição):
    comandos
```

Por exemplo:

```
contador=0
while (contador<5):
    print(contador)
    contador=contador+1
```

Neste exemplo de código estamos repetindo, por meio do *while*, o comando *print* e incrementando o contador 5 vezes, utilizando uma variável contadora (*contador*). Inicializamos a variável *contador* com o valor zero. Enquanto o valor do contador for menor do que 5 (enquanto o resultado desta condição for *True*), o trecho dentro do *while* será repetido. Lembre-se que a i ndentação do código-fonte faz parte das regras de sintaxe da linguagem *Python*. Sendo assim, os comandos dentro do *while* devem estar corretamente indentados para que sejam executados na repetição. O resultado da execução deste trecho é apresentado a seguir:

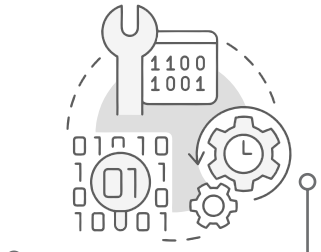
```
0
1
2
3
4
```

O Quadro 10 apresenta a comparação deste exemplo em *Python* com a linguagem de programação C.

Quadro 10 – Comparação do Exemplo com o Comando *While*

Python	**C**
contador=0 *while (contador<5):* *print(contador)* *contador=contador+1*	*#include <stdio.h>* *main() {* *int contador;* *contador=0;* *while (contador<5) {* *printf(contador);* *contador=contador+1;* *}* *}*

Fonte: Os autores



CAPÍTULO 4

Estruturas de Dados Fundamentais

Uma variável, em um programa pode ser simples ou composta. Nos três primeiros capítulos deste *e-book* utilizamos apenas variáveis simples, que possuem um identificador e um único valor. As variáveis compostas podem ser homogêneas ou heterogêneas, possuem um identificador e armazenam mais de um valor. As variáveis compostas são **estruturas de dados.** Uma estrutura de dados envolve, além da estrutura de armazenamento, as regras de definição, manipulação e de acesso aos dados contidos nas mesmas. Os programas, por meio de estruturas de dados, organizam logicamente os dados, permitindo sua visualização de diferentes formas (SILVA *et al.*, 2010).

As variáveis compostas homogêneas referem-se a uma estrutura de dados (ou conjunto de dados) que armazenam dados do mesmo tipo (inteiro, caractere, etc.) e esses dados são acessados por meio de um índice (valor numérico inteiro) que representa a posição do dado dentro da estrutura. As principais estruturas de dados homogêneas são os vetores (também conhecidos como matrizes unidimensionais ou matrizes de uma dimensão) e as matrizes bidimensionais.

4.1 Vetores (ou Listas em *Python*)

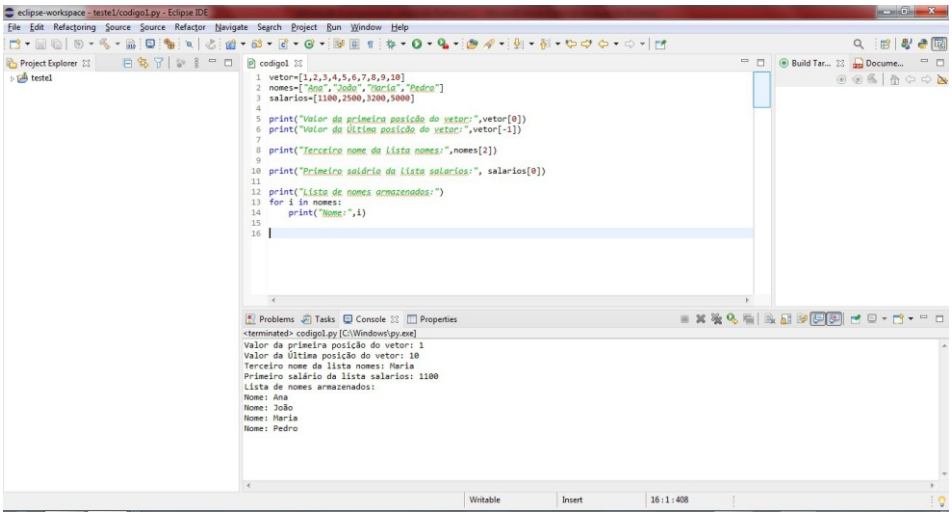
Um vetor (ou matriz unidimensional), na linguagem *Python*, é uma lista. As listas são definidas entre colchetes, sendo seus elementos separados por vírgula. Os elementos podem ser acessados por meio do índice que indica a posição do elemento desejado na lista, iniciando pelo índice zero. O índice -1 corresponde ao último item da lista. Sucessivamente, o índice -2 permite acessar o penúltimo item da lista e assim por diante.

Vamos fazer alguns exemplos criando e manipulando listas utilizando a IDE *Eclipse*. Vamos criar um novo programa em *Python* com os seguintes exemplos de código-fonte:

```
vetor=[1,2,3,4,5,6,7,8,9,10]

nomes=["Ana","João","Maria","Pedro"]

salarios=[1100,2500,3200,5000]


print("Valor da primeira posição do vetor:",vetor[0])

print("Valor da Última posição do vetor:",vetor[-1])

print("Terceiro nome da lista nomes:",nomes[2])

print("Primeiro salário da lista salarios:", salarios[0])

print("Lista de nomes armazenados:")

for i in nomes:

    print("Nome:",i)
```

A Figura 24 apresenta este código-fonte, bem como seus respectivos resultados de execução, utilizando a IDE *Eclipse.*

Figura 24: Exemplos de Listas/Vetores (Fonte: Os autores)



Vamos utilizar o exemplo para calcular a média de 10 alunos (conforme Quadro 9, do capítulo anterior). Vamos utilizar vetores (listas em *Python*) para armazenar, separadamente, as notas da 1ª avaliação dos 10 alunos, as notas da 2ª avaliação e as respectivas médias. Assim teremos três vetores, como mostra a Figura 25.

Figura 25 – Representação Gráfica dos Vetores (Fonte: Os autores)

Vetor P1 – Notas da 1ª avaliação (lembrando que os índices, em *Python*, iniciam em zero)

0	1	2	3	4	5	6	7	8	9

Vetor P2 – Notas da 2ª avaliação

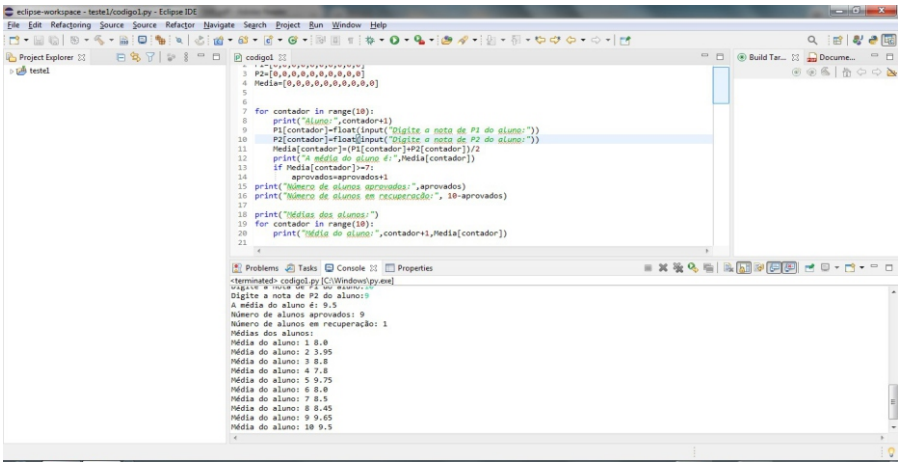
0	1	2	3	4	5	6	7	8	9

Vetor *Media* – Médias aritméticas das notas dos 10 alunos

0	1	2	3	4	5	6	7	8	9

A Figura 26 apresenta o exemplo de código-fonte em *Python*, utilizando os vetores apresentados anteriormente. Como vamos armazenar as médias no vetor *Media*, após o laço *for* da entrada de dados podemos acessar as médias individualmente (bem como as notas de P1 e de P2), o que não era possível no exemplo que fizemos no capítulo 3, já que estávamos utilizando variáveis simples.

Figura 26: Exemplo de Cálculo da Média com 3 vetores/listas
(Fonte: Os autores)



O Quadro 11 apresenta a comparação do exemplo do cálculo da média com vetores em *Python* e em Linguagem C.

Quadro 11 – Comparação do Exemplo com Vetores/Listas

Python	**C**
aprovados=0 *P1=[0,0,0,0,0,0,0,0,0,0]* *P2=[0,0,0,0,0,0,0,0,0,0]* *Media=[0,0,0,0,0,0,0,0,0,0]* *for contador in range(10):* *print("Aluno:",contador+1)* *P1[contador]=float(input("Digite a nota de P1 do aluno:"))* *P2[contador]=float(input("Digite a nota de P2 do aluno:"))* *Media[contador]=(P1[contador]+P2[contador])/2* *print("A média do aluno é:",Media[contador])* *if Media[contador]>=7:* *aprovados=aprovados+1* *print("Número de alunos aprovados:",aprovados)* *print("Número de alunos em recuperação:",* *10-aprovados)* *print("Médias dos alunos:")* *for contador in range(10):* *print("Média do aluno:",* *contador+1,Media[contador])*	*#include <stdio.h>* *main() {* *float P1[10], P2[10], Media[10];* *int contador, aprovados;* *for (contador=0; contador<10; contador++) {* *printf("Digite a nota de P1:");* *scanf("%f", &P1[contador]);* *printf("Digite a nota de P2:");* *scanf("%f",&P2[contador]);* *Media[contador]=(P1[contador]+P2[contador])/2;* *printf("A média do aluno é: %.2f\n",* *Media[contador]);* *if (Media[contador]>=7) {* *aprovados=aprovados+1;* *}* *}* *printf("Número de alunos aprovados %d:",aprovados);* *printf("Número de alunos em recuperação %d:",10-aprovados);* *printf("Média dos alunos:");* *for (contador=0; contador<10; contador++) {* *printf("Média do aluno: %d %.2f\n",contador,Media[contador]);* *}* *}*

Fonte: Os autores

4.2 Matrizes Bidimensionais (Listas em *Python*)

Uma matriz bidimensional, na linguagem *Python*, é uma lista com dois índices, geralmente denominados de *linha* e *coluna*, como uma tabela ou uma planilha do *Microsoft Excel*. As matrizes são definidas entre colchetes, sendo os elementos de cada linha inseridos entre colchetes (separados por vírgulas) e, cada linha também é separada por vírgulas.

Por exemplo, a seguinte atribuição em *Python* cria uma matriz A com 2 linhas e 2 colunas:

```
A=[[10,20],[30,40]]
```

A Figura 27 mostra a representação gráfica da matriz bidimensional A (2x2 – 2 linhas e 2 colunas), lembrando que os índices, em *Python*, iniciam no valor zero).

Figura 27: Representação Gráfica da Matriz A (Fonte: Os autores)

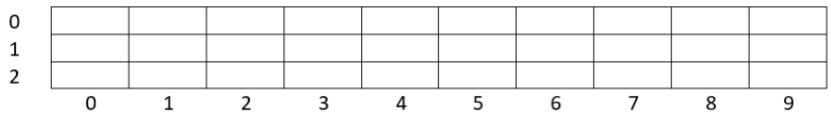
	Coluna 0	Coluna 1
Linha 0	10	20
Linha 1	30	40

Para percorrermos a matriz A precisamos, então, de dois índices (*linha* e *coluna*). O código-fonte abaixo mostra como percorrer e imprimir os valores contidos na matriz A em *Python:*

```
A=[[10,20],[30,40]]
for linha in range(2):
    for coluna in range(2):
        print("Valores armazenados na matriz A:",A[linha][coluna])
```

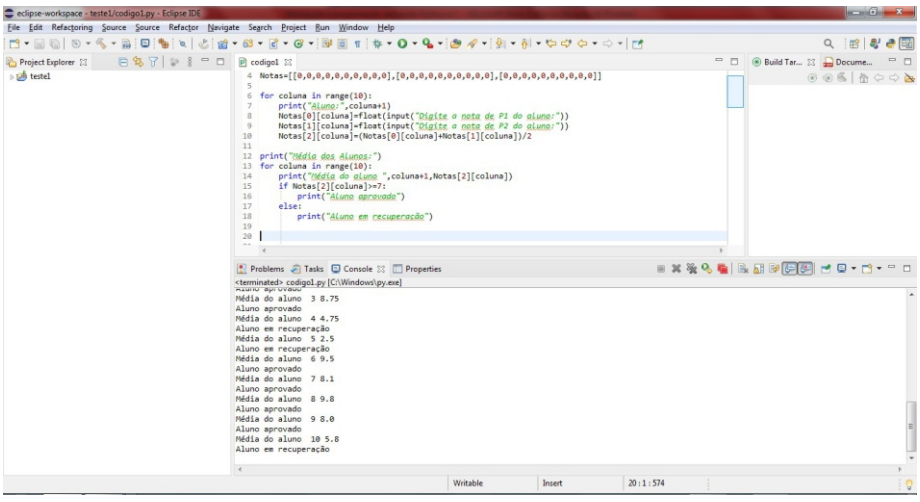
Vamos fazer o exemplo do cálculo da média utilizando uma matriz bidimensional para armazenar as notas de P1, de P2 e as médias. Ao invés de utilizarmos três vetores (matrizes unidimensionais) como fizemos na seção 4.1, vamos criar uma matriz bidimensional com 3 linhas (P1, P2 e Média) e 10 colunas (uma coluna para cada aluno). A linha *0* será utilizada para armazenar as notas de P1, a linha 1 para as notas de P2 e a linha 2 para armazenar as médias de cada aluno. A Figura 28 mostra a representação gráfica desta matriz 3x10 (3 linhas e 10 colunas).

Figura 28: Representação Gráfica da Matriz Bidimensional 3x10
(Fonte: Os autores)



A Figura 29 apresenta o exemplo construído na IDE *Eclipse*. Criamos uma matriz bidimensional com 3 linhas e 10 colunas denominada de *Notas*. Armazenamos, na linha 0, os valores de P1 (*Notas[0][coluna]*), os valores de P2 na linha 1 e o cálculo da média na linha 2.

Figura 29: Exemplo de Matriz Bidimensional para Calcular a Média dos Alunos
(Fonte: Os autores)

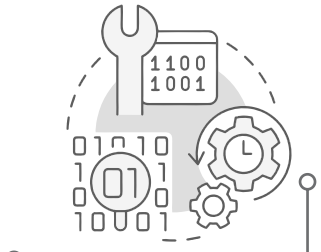


O Quadro 12 apresenta a comparação do exemplo do cálculo da média com uma matriz bidimensional com 3 linhas e 10 colunas em *Python* e em Linguagem C.

Quadro 12 – Comparação do Exemplo com Matriz Bidimensional

Python	**C**
Notas=[[0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0]] *for coluna in range(10):* * print("Aluno:",coluna+1)* * Notas[0][coluna]=float(input("Digite a nota de P1 do aluno:"))* * Notas[1][coluna]=float(input("Digite a nota de P2 do aluno:"))* * Notas[2][coluna]=(Notas[0][coluna]+Notas[1][coluna])/2* *print("Média dos Alunos:")* *for coluna in range(10):* * print("Média do aluno ",coluna+1,Notas[2][coluna])* * if Notas[2][coluna]>=7:* * print("Aluno aprovado")* * else:* * print("Aluno em recuperação")*	*#include <stdio.h>* *main() {* * float Notas[3][10];* * int coluna, aprovados;* * for (coluna=0; coluna<10; coluna++) {* * printf("Digite a nota de P1:");* * scanf("%f", &Notas[0][coluna]);* * printf("Digite a nota de P2:");* * scanf("%f",&Notas[1][coluna]);* *Notas[2][coluna]=(Notas[0][coluna]+Notas[1][coluna])/2;* * }* * printf("Média dos alunos:");* * for (coluna=0; coluna<10; coluna ++) {* * printf("Média do aluno %d %.2f\n",* *coluna+1,Notas[2][coluna]);* * if (Notas[2][coluna]>=7) {* * printf("Aluno aprovado\n");* * }* * {* * printf("Aluno em recuperação\n");* * }* * }* *}*

Fonte: Os autores



CAPÍTULO 5

Funções

Para que possamos resolver problemas complexos precisamos implementar algoritmos complexos. Para permitir um melhor entendimento dos algoritmos, problemas complexos podem ser divididos, utilizando-se a modularização do programa em pequenas partes, que podem ser denominadas de sub-rotinas, módulos ou subprogramas (SILVA *et al.*, 2010). As sub-rotinas permitem organizar melhor a funcionalidade do programa, evitar a repetição de trechos de código (já que podem ser executadas diversas vezes), facilitar a depuração do código e tornar o código mais legível (SILVA *et al.*, 2010). Na linguagem de programação *Python* as sub-rotinas são denominadas de funções e são declaradas por meio da palavra reservada *def*. A sintaxe para declarar uma função em *Python* é a seguinte:

```
def nome_da_função(argumentos):"""comentário – para que serve a função"""
```

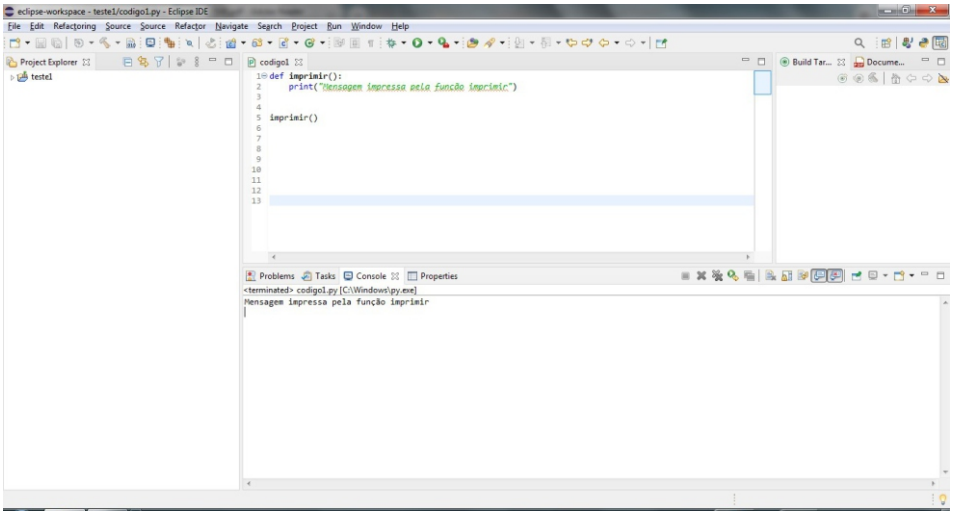
Temos, de acordo com a sintaxe, que definir o *nome_da_função* e os parâmetros que deverão ser informados no momento em que a mesma será executada (quando "chamamos" a função em nosso código-fonte). Se não existirem parâmetros deve-se deixar os parênteses vazios (). Os comentários são opcionais e devem ser iniciados e terminados com 3 aspas duplas.

Por exemplo, vamos definir uma função que mostra uma mensagem na tela. Vamos chamar a função de *imprimir:*

```
def imprimir():
    print("Mensagem impressa pela função imprimir")
```

Para utilizarmos (chamarmos) a função no nosso código-fonte, devemos referenciar o seu nome, com os parênteses: *imprimir()*. Vamos ver um exemplo na IDE *Eclipse*, como mostra a Figura 30. A função precisa estar definida antes de ser utilizada, ou seja, no código-fonte é preciso primeiro definir as funções (e o respectivo código de cada uma delas, após os dois pontos) para, posteriormente, podermos utilizá-las (chamá-las). Quando uma função é chamada ocorre um desvio de execução do programa para a linha/local onde a mesma está descrita. A partir de então, todos os comandos que fazem parte da função são executados. Ao final da execução o controle volta para o local onde previamente ocorreu o desvio, ou seja, o controle de execução volta para a rotina chamadora, que pode ser uma função ou a parte principal do programa (conhecido como programa principal, em C é representado pela função *main*).

Figura 30: Exemplo de Função em *Python* (Fonte: Os autores)



Vamos fazer um segundo exemplo de função, em que utilizaremos dois parâmetros: nome e idade:

```
def exemplo_funcao(nome, idade):
  print("Nome do usuário:",nome)
 print("Idade:",idade)
if idade<=18:
  print("Você é jovem!")
```

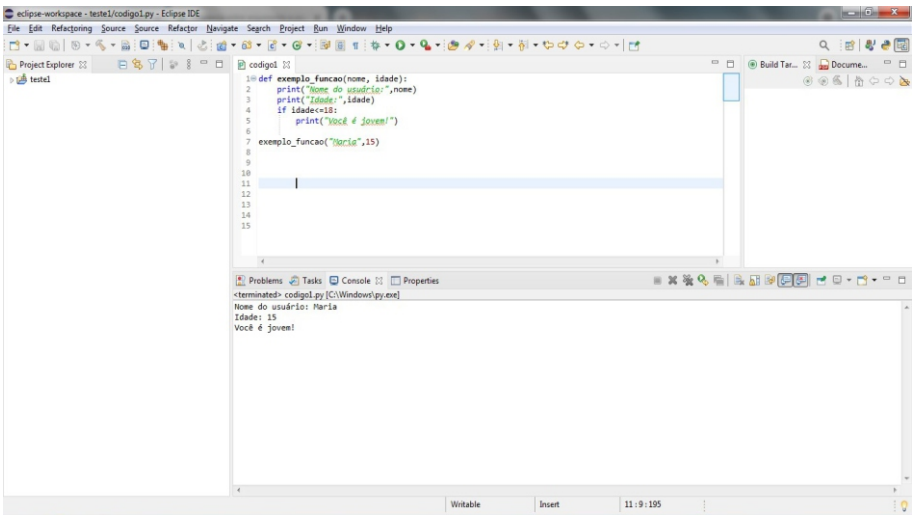
As variáveis *nome* e *idade* são os parâmetros da função. Para que possamos chamar esta função será preciso informar dois valores, um para a variável *nome* e outro para a variável *idade*. Quando definirmos os valores que serão "passados" para a função, esses valores são denominados de argumentos. Por exemplo:

```
exemplo_funcao("Maria",15)
```

Maria e 15 são os argumentos que serão informados ao chamarmos a função *exemplo_funcao.*

Vamos ver o exemplo na IDE *Eclipse*, como mostra a Figura 31.

Figura 31: Exemplo de Função com Parâmetros (Fonte: Os autores)



O Quadro 13 apresenta a comparação destes dois exemplos de funções em *Python* e em Linguagem C.

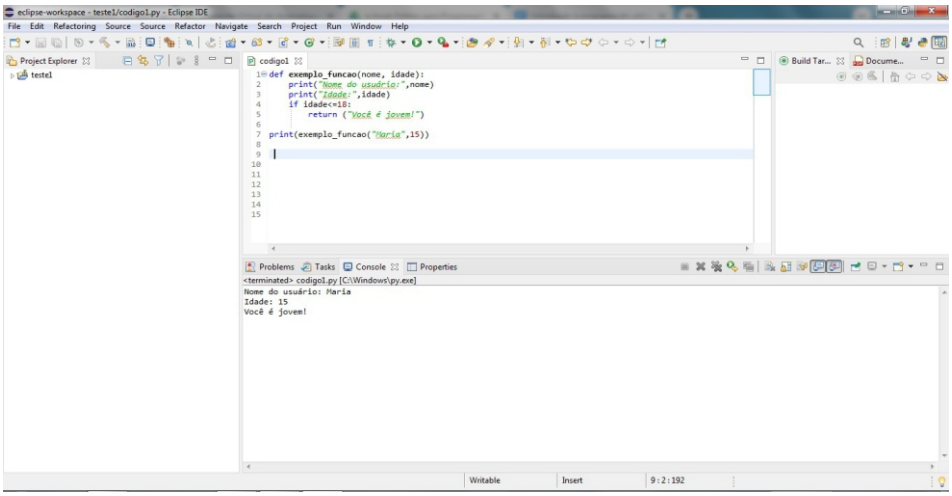
Quadro 13 – Comparação dos Exemplos de Funções

Python	**C**
#definição da função *def imprimir():* *print("Mensagem impressa pela função imprimir")* *# chamada da função* *imprimir()*	*#include <stdio.h>* *void Imprimir() {* *printf("Mensagem impressa pela função imprimir");* *}* *main() {* *Imprimir();* *}*
#definição da função *def exemplo_funcao(nome, idade):* *print("Nome do usuário:",nome)* *print("Idade:",idade)* *if idade<=18:* *print("Você é jovem!")* *#chamada da função* *exemplo_funcao("Maria",15)*	*#include <stdio.h>* *void Exemplo_funcao(char *Nome, int idade) {* *printf("Nome do usuário: %s \n",Nome);* *printf("Idade: %d \n",idade);* *if (idade<=18) {* *printf("você é jovem! \n");* *}* *}* *main() {* *Exemplo_funcao("Maria",15);* *}*

Fonte: Os autores

Nos exemplos apresentados, as funções exibem os valores de saída, por meio de comandos de impressão na tela. Entretanto, uma função não precisa exibir, necessariamente, um valor de saída. A função pode receber parâmetros, processar e devolver um resultado para a rotina chamadora. Vamos utilizar o mesmo exemplo anterior e, ao invés de imprimirmos a mensagem de resposta, vamos devolver o resultado para a rotina chamadora (o que costumamos denominar de valor de retorno da função - *return*). Inserimos, dentro da função, a palavra reservada *return*, para definir o que a função deve retornar (ou devolver) para a rotina chamadora. No exemplo, retornamos a mensagem "Você é jovem". Nesse caso, a rotina chamadora precisa estar adequada para receber o valor de retorno. O valor pode ser armazenado em uma variável ou impresso na tela. No exemplo da Figura 32, imprimimos o valor de retorno na tela, adicionando o comando print e, dentro dos parênteses, a chamada da função.

Figura 32: Exemplo de Função com Parâmetros e Retorno
(Fonte: Os autores)



O Quadro 14 apresenta a comparação do exemplo da função desenvolvida na Figura 32 em *Python* e em Linguagem C.

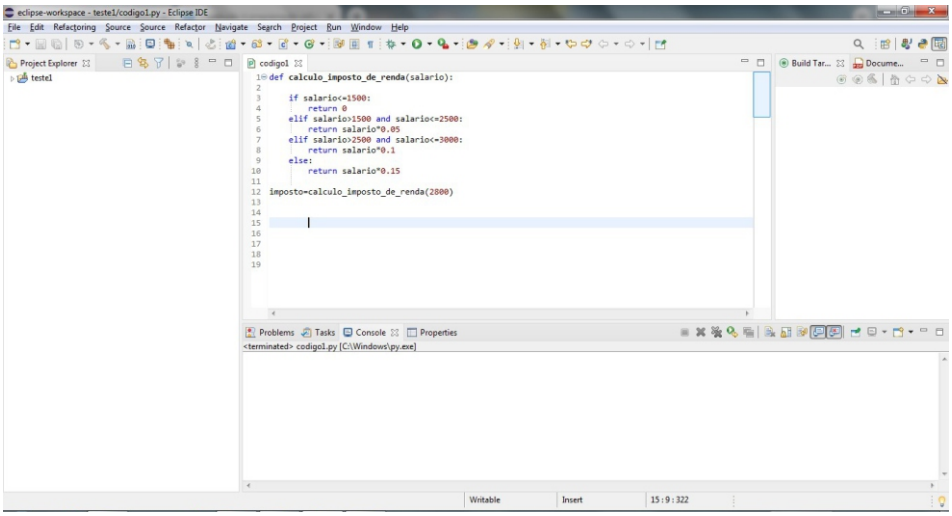
Quadro 14 – Comparação dos Exemplos de Funções

Python	**C**
#definição da função *def exemplo_funcao(nome, idade):* *print("Nome do usuário:",nome)* *print("Idade:",idade)* *if idade<=18:* *return ("Você é jovem!")* *#chamada da função* *print(exemplo_funcao("Maria",15))*	*#include <stdio.h>* *char* Exemplo_funcao(char *Nome, int idade) {* *printf("Nome do usuário: %s\n",Nome);* *printf("Idade: %d\n",idade);* *if (idade<=18) {* *return ("você é jovem!\n");* *}* *}* *main() {* *printf(Exemplo_funcao("Maria",15));* *}*

Fonte: Os autores

Podemos, também, armazenar o valor de retorno em uma variável. Vamos fazer um exemplo de função que receba o valor do salário de uma pessoa e devolva o valor do Imposto de Renda a ser pago, de acordo com as alíquotas: até R$1500,00 (isento), de R$1501,00 a R$2500,00 (5%), de R$2501,00 a R$3000,00 (10%) e acima de R$3000,00 (15%). No exemplo da Figura 33, a rotina chamadora armazena o valor de retorno da função na variável *imposto.* Entretanto, em nosso exemplo, não estamos mostrando o valor na tela, apenas armazenando-o na variável.

Figura 33: Exemplo de Função com Retorno armazenado em uma variável (Fonte: Os autores)



O Quadro 15 apresenta a comparação destes dois exemplos de funções em *Python* e em Linguagem C.

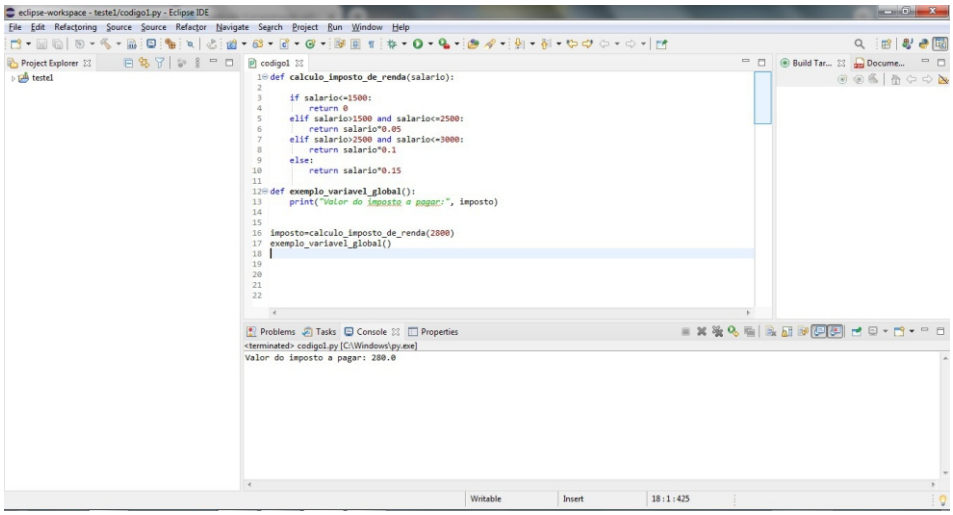
Quadro 15 – Comparação dos Exemplos de Funções

Python	**C**
def calculo_imposto_de_renda(salario): *if salario<=1500:* *return 0* *elif salario>1500 and salario<=2500:* *return salario*0.05* *elif salario>2500 and salario<=3000:* *return salario*0.1* *else:* *return salario*0.15* *imposto=calculo_imposto_de_renda(2800)*	*float imposto;* *float calculo_imposto_de_renda(float salario)* *{* *if (salario <=1500)* *return 0;* *if (salario>1500 and salario<2500)* *return salario*0.05;* *if (salario>2500 and salario<=3000)* *return salario*0.1;* *if (salario>3000)* *return(salario*0.15);* *}* *main () {* *imposto=calculo_imposto_de_renda(2800);* *}*

Fonte: Os autores

Quando incluímos funções em nossos programas precisamos estudar o escopo das variáveis, que pode ser global ou local. Uma variável definida no programa principal (rotina *main*, por exemplo, é uma variável local). Por sua vez, uma variável definida dentro de uma função, é uma variável local. No exemplo da Figura 33, a variável *imposto* é global, pois foi definida fora da função. Sendo assim, ela pode ser acessada em todo o programa, até mesmo em outras funções. Veja o exemplo da Figura 34.

Figura 34: Exemplo de Variável Global (Fonte: Os autores)



Vemos, no exemplo da Figura 34, que a variável *imposto* foi utilizada na função *exemplo_variável_global*, sem que precisasse ser definida dentro da função e nem mesmo seu valor tivesse sido passado por parâmetro. Apesar dessa possibilidade existir, por questões de legibilidade do código-fonte e, também, de segurança (para que uma função não modifique o valor de uma variável global - o que pode afetar todo o programa), não recomendamos o uso de variáveis globais dentro das funções.

O quadro 16 apresenta o código da Figura 34 em *Python* e na Linguagem C.

Quadro 16 – Comparação dos Exemplos de Funções

Python	**C**

`def calculo_imposto_de_renda(salario):`
 	
`  if salario<=1500:`
`  return 0`
`  elif salario>1500 and salario<=2500:`
`  return salario*0.05`
`  elif salario>2500 and salario<=3000:`
`  return salario*0.1`
`  else:`
`  return salario*0.15`
 	
`def exemplo_variavel_global():`
`  print("Valor do imposto a pagar:", imposto`
 	
`imposto=calculo_imposto_de_renda(2800)`
`exemplo_variavel_global()`
	`#include <stdio.h>`
	`float imposto;`
	
	`floatcalculo_imposto_de_renda(float salario)`
	`{`
	`if (salario <=1500)`
	`return 0;`
	`if (salario>1500 and salario<2500)`
	`return salario*0.05;`
	`if (salario>2500 and salario<=3000)`
	`return salario*0.1;`
	`if (salario>3000)`
	`return(salario*0.15);`
	`}`
	
	`voidexemplo_variavel_global()`
	`{`
	`printf("Valor do imposto a pagar: %f",imposto);`
	`}`
	
	`main () {`
	`imposto=calculo_imposto_de_renda(2800);`
	`exemplo_variavel_global();`
	`}`

Fonte: Os autores

Vamos ver o código do exemplo anterior (Figura 34 e Quadro 16) utilizando a passagem de parâmetros ao invés de uma variável global. O Quadro 17 mostra o exemplo do código-fonte em *Python* e na Linguagem C. No exemplo, utilizando o nome *imposto* (que é uma variável global) como nome do parâmetro da função. Entretanto, apesar de serem nomes iguais, ao passarmos *imposto* como parâmetro, estamos utilizando a passagem de parâmetros por valor. Sendo assim, a função recebe uma cópia do valor e armazena-o em outra posição da memória. Se fizermos alguma modificação no valor da variável *imposto*, dentro da função, isso só acontecerá no escopo local (dentro da função).

Quadro 17 – Comparação dos Exemplos de Funções

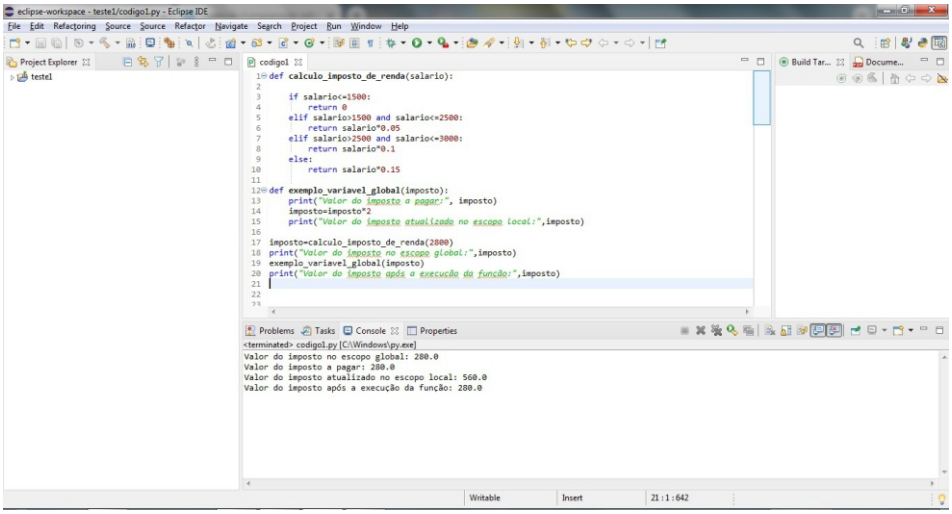
Python	**C**
def calculo_imposto_de_renda(salario): *if salario<=1500:* *return 0* *elif salario>1500 and salario<=2500:* *return salario*0.05* *elif salario>2500 and salario<=3000:* *return salario*0.1* *else:* *return salario*0.15* *def exemplo_variavel_global(imposto):* *print("Valor do imposto a pagar imposto)* *imposto=calculo_imposto_de_renda(280)* *exemplo_variavel_global(imposto)*	*#include <stdio.h>* *float imposto;* *float calculo_imposto_de_renda(float salario)* *{* *if (salario <=1500)* *return 0;* *if (salario>1500 and salario<2500)* *return salario*0.05;* *if (salario>2500 and salario<=3000)* *return salario*0.1;* *if (salario>3000)* *return(salario*0.15);* *}* *void exemplo_variavel_global(float imposto)* *{* *printf("Valor do imposto a pagar: %f",imposto)* *}* *main () {* *imposto=calculo_imposto_de_renda(2800);* *exemplo_variavel_global(imposto);* *}*

Fonte: Os autores

Utilizando o mesmo exemplo, vamos adicionar uma linha de código dentro da segunda função. Vamos alterar o valor do parâmetro *imposto*, multiplicando-o por dois. Ao executar o código (Figura 35) vemos que o valor aparece alterado dentro da função (escopo local) mas, ao voltar a execução para a rotina chamadora (escopo global), o valor não foi modificado.

O Quadro 18 apresenta o código-fonte do exemplo da Figura 35, comparado entre as linguagens Python e C.

Figura 35: Exemplo com escopo local e global (Fonte: Os autores)



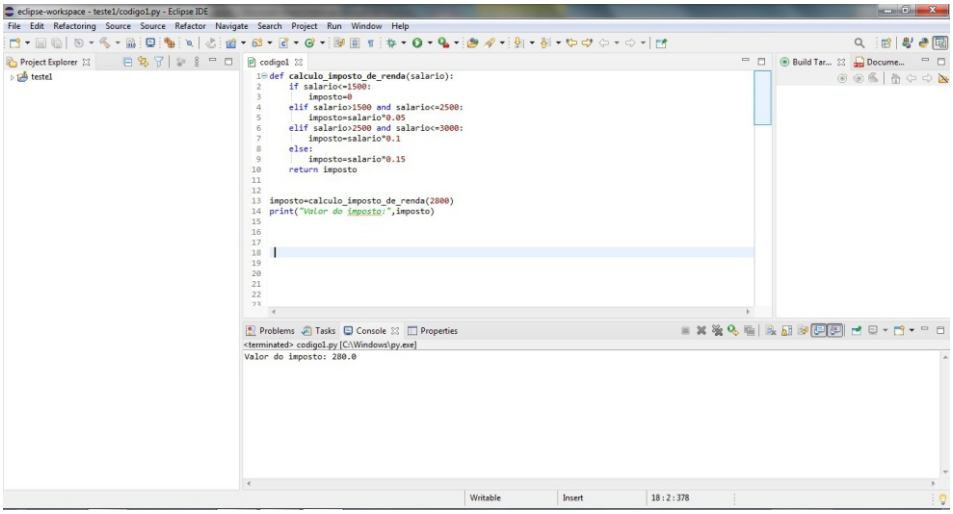
Quadro 18 – Comparação dos Exemplos com Escopo Local e Global

Python	**C**
def calculo_imposto_de_renda(salario): *if salario<=1500:* *return 0* *elif salario>1500 and salario<=2500:* *return salario*0.05* *elif salario>2500 and salario<=3000:* *return salario*0.1* *else:* *return salario*0.15* *def exemplo_variavel_global(imposto):* *print("Valor do imposto a pagar imposto)* *imposto=imposto*2* *print("Valor do imposto atualizado escopo local:",imposto)* *imposto=calculo_imposto_de_renda(2800* *print("Valor do imposto no esco global:",imposto)* *exemplo_variavel_global(imposto)* *print("Valor do imposto após a execução função:",imposto)*	*#include <stdio.h>* *float imposto;* *float calculo_imposto_de_renda(float salario)* *{* *if (salario <=1500)* *return 0;* *if (salario>1500 and salario<2500)* *return salario*0.05;* *if (salario>2500 and salario<=3000)* *return salario*0.1;* *if (salario>3000)* *return(salario*0.15);* *}* *void exemplo_variavel_global(float imposto)* *{* *printf("Valor do imposto a pagar: %f",imposto);* *imposto=imposto*2;* *printf("Valor do imposto atualizado no escopo local: %f",imposto);* *}* *main () {* *imposto=calculo_imposto_de_renda(2800);* *printf("Valor do imposto no escopo global: %f",imposto);* *exemplo_variavel_global(imposto);* *printf("Valor do imposto após a execução da função %f",imposto);* *}*

Fonte: Os autores

Existem outras formas de retornar o valor de uma função. Por exemplo, armazenando o resultado em uma variável local e retornando o valor da variável. Veja o exemplo da Figura 36, onde armazenamos o valor do imposto de renda a pagar em uma variável e depois utilizamos o valor da variável na instrução *return*.

Figura 36: Exemplo utilizando uma variável para armazenar o valor de retorno da função (Fonte: Os autores)



No Quadro 19 é possível verificarmos a comparação do código-fonte do exemplo da Figura 36, entre *Python* e C.

Quadro 19 – Comparação dos Exemplos com uma variável para retornar o resultado da função

Python	**C**
def calculo_imposto_de_renda(salario): *if salario<=1500:* *imposto=0* *elif salario>1500 and salario<=2500:* *imposto=salario*0.05* *elif salario>2500 and salario<=3000:* *imposto=salario*0.1* *else:* *imposto=salario*0.15* *return imposto* *imposto=calculo_imposto_de_renda(2800)* *print("Valor do imposto:",imposto)*	*#include <stdio.h>* *float imposto;* *float calculo_imposto_de_renda(float salario)* *{* *float imposto;* *if (salario <=1500)* *imposto=0;* *if (salario>1500 and salario<2500)* *imposto=salario*0.05;* *if (salario>2500 and salario<=3000)* *imposto=salario*0.1;* *if (salario>3000)* *imposto=salario*0.15;* *return imposto;* *}* *main () {* *imposto=calculo_imposto_de_renda(2800);* *printf("Valor do imposto: %f",imposto);* *}*

Fonte: Os autores

BIBLIOGRAFIA

REFERÊNCIAS BIBLIOGRÁFICAS

FALKEMBACH, Gilse Morgental; SILVEIRA, Sidnei Renato. Algoritmos e Programação I. Canoas: ULBRA, 2005. Caderno Universitário.

MATTHES, Eric. Curso Intensivo de Python: uma introdução prática e baseada em projetos de programação. São Paulo: Novatec, 2016.

PARREIRA, Fábio José; SILVEIRA, Sidnei Renato; BERTOLINI, Cristiano; SEVERO, Rosane Beatriz. Introdução a Algoritmos. Santa Maria: UAB/NTE/UFSM, 2017. Disponível em: https://repositorio.ufsm.br/bitstream/handle/1/15820/Licenciatura_Computacao_introducaoalgoritmos.pdf?sequence=1&isAllowed=y. Acesso em 21 out. 2019.

PYTHON.ORG. Welcome to Python.org. Disponível em: http://python.org. Acesso em 21 out. 2019.

ROSSUM, G. Tutorial Python. Disponível em: https://wiki.python.org.br/Tutorial_Python . Acesso em 20 out. 2019.

SILVA, Isabel Cristina Siqueira; FALKEMBACH, Gilse Morgental; SILVEIRA, Sidnei Renato. Algoritmos e Programação em Linguagem C. Porto Alegre: UniRitter, 2010.

SILVEIRA, Sidnei Renato; DE VIT; Antônio Rodrigo Delepiane; BERTOLINI, Cristiano; PARREIRA, Fábio José; CUNHA; Guilherme Bernardino; BIGOLIN, Nara Martini. Paradigmas de Programação: uma introdução. Belo Horizonte: Synapse, 2021. Disponível em: https://www.editorasynapse.org/wp-content/uploads/2021/03/paradigmas_programacao_uma_introducao_V0.pdf. Acesso em 30 mar. 2021.

INTRODUÇÃO À PROGRAMAÇÃO E ESTRUTURAS DE DADOS FUNDAMENTAIS COM PYTHON PARA PROGRAMADORES C

APRESENTAÇÃO DOS AUTORES

APRESENTAÇÃO DOS AUTORES

ANTÔNIO RODRIGO DELEPIANE DE VIT

Professor Adjunto do Departamento de Tecnologia da Informação da UFSM (Universidade Federal de Santa Maria) - Campus Frederico Westphalen/RS. Doutor em Ciência da Computação pela PUC-RS (Pontifícia Universidade Católica do Rio Grande do Sul). Mestre em Ciência da Computação pela UFRGS (Universidade Federal do Rio Grande do Sul) e Bacharel em Informática pela UNIJUÍ.

SIDNEI RENATO SILVEIRA

Professor Associado do Departamento de Tecnologia da Informação da UFSM (Universidade Federal de Santa Maria) - Campus Frederico Westphalen/RS. Doutor em Ciência da Computação pela UFRGS (Universidade Federal do Rio Grande do Sul). Mestre em Ciência da Computação pela UFRGS. Especialista em Gestão Educacional pelo SENAC. Especialista em Administração e Planejamento para Docentes pela ULBRA. Bacharel em Informática pela ULBRA.

https://www.facebook.com/Synapse-Editora-111777697257115

https://www.instagram.com/synapseeditora

https://www.linkedin.com/in/synapse-editora-compartilhando-conhecimento/

31 98264-1586

editorasynapse@gmail.com

SYNAPSE Editora

Compartilhando conhecimento