UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES

DANILA BRANCO GUEDES

LINGUAGEM DE PROGRAMAÇÃO PYTHON E ARDUINO COMO FERRAMENTA PARA MOTIVAR ESTUDANTES INICIANTES EM PROGRAMAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2018

DANILA BRANCO GUEDES

LINGUAGEM DE PROGRAMAÇÃO PYTHON E ARDUINO COMO FERRAMENTA PARA MOTIVAR ESTUDANTES INICIANTES EM PROGRAMAÇÃO

Trabalho de Conclusão de Curso de Graduação, apresentado ao Curso Superior de Tecnologia em Sistemas de Telecomunicações, do Departamento Acadêmico de Eletrônica - DAELN, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. M.Sc. Omero Francisco Bertol

CURITIBA
2018

TERMO DE APROVAÇÃO

DANILA BRANCO GUEDES

LINGUAGEM DE PROGRAMAÇÃO PYTHON E ARDUINO COMO FERRAMENTA PARA MOTIVAR ESTUDANTES INICIANTES EM PROGRAMAÇÃO

Este trabalho de conclusão de curso foi apresentado no dia 26 de julho de 2018, como requisito parcial para obtenção do título de Tecnóloga em Sistemas de Telecomunicações, outorgado pela Universidade Tecnológica Federal do Paraná. A aluna foi arguida pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Profa. Dra. Tânia Lucia Monteiro
Coordenadora de Curso
Departamento Acadêmico de Eletrônica

_____
Prof. M.Sc. Sérgio Moribe
Responsável pela Atividade de Trabalho de Conclusão de Curso
Departamento Acadêmico de Eletrônica

BANCA EXAMINADORA

_____
Prof. Dr. Edenilson José da Silva
UTFPR

_____
Prof. Dr. Joilson Alves Junior
UTFPR

_____
Prof. M.Sc. Omero Francisco Bertol
Orientador - UTFPR

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

AGRADECIMENTOS

Agradeço à Deus pela vitalidade e força concedida nessa longa caminhada.

Agradeço também a todos os professores que me acompanharam durante a graduação, em especial ao professor orientador Omero Francisco Bertol, responsável pela orientação deste trabalho.

Agradeço ao meu esposo Donizet Vieira, que de maneira especial e carinhosa sempre me apoiou nas dificuldades.

Agradeço a minha Família pois sempre acreditaram que a graduação era possível.

RESUMO

GUEDES, Danila Branco. **Linguagem de Programação Python e Arduino como Ferramenta para Motivar Estudantes Iniciantes em Programação**. 2018. 79 f. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Sistemas de Telecomunicações), Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2018.

Ao longo de vários anos a programação vem tomando espaço, e aumentando a utilização da tecnologia como um todo, com isso vislumbra-se uma grande oportunidade nessa área de conhecimento. Dessa forma, os objetivos desse trabalho de conclusão de curso são os seguintes: motivar e expandir o conhecimento de estudantes iniciantes na área de programação; demonstrar a facilidade de se trabalhar com o a linguagem de programação Python; demonstrar a praticidade de se trabalhar com a tecnologia do Arduino. Para se alcançar tais objetivos, foi realizado um estudo de caso utilizando a comunicação serial entre Python e Arduino.

Palavras chave: Linguagem de Programação Python. Arduino. Comunicação Serial.

ABSTRACT

GUEDES, Danila Branco. **Programming Language Python and Arduino as a Tool to Motivate Beginners Students in Programming**. 2018. 79 f. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Sistemas de Telecomunicações), Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2018.

Over the course of several years, programming has been taking up space, increasing the use of technology as a whole, with a great opportunity in this area of knowledge. Thus, the objectives of this course completion work are: to motivate and expand the knowledge of beginning students in the programming area; demonstrate the ease of working with the Python programming language; demonstrate the practicality of working with Arduino technology. In order to reach such objectives, a case study was carried out using the serial communication between Python and Arduino.

Keywords: Python Programming Language. Arduino. Serial Communication.

LISTA DE FIGURAS

LISTA DE LISTAGENS

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Nesta seção será apresentado o assunto tema deste trabalho de conclusão de curso, a motivação, o objetivo geral, os objetivos específicos, a justificativa e a estrutura do trabalho.

1.1 CONTEXTUALIZAÇÃO

A programação vem ao longo dos anos mudando a vida das pessoas, a maneira como todos vivem e convivem uns com os outros. Nossas atividades estão cada vez mais programadas e conectadas, o uso da tecnologia no trabalho, nas escolas, universidades cresce a cada dia, a vida cotidiana está conectada na sua grande parte ao uso de celulares, porém pequena parte da população sequer tem conhecimento sobre o que há por trás de uma tela de um aparelho celular. Em 2016 no Brasil haviam mais de 116 milhões de pessoas conectadas à internet, o equivalente a 64,7% da população com idade acima de 10 anos, segundo o Instituto Brasileiro de Geografia e Estatística (IBGE) (GOMES, 2018).

A tecnologia cresce, tudo se conecta, porém o número de programadores não cresce no mesmo volume. Nos próximos dez anos haverá mais de 1,4 milhão de vagas em ciência da computação e apenas 400 mil profissionais qualificados para essas vagas, segundo o Diário de programador (LUAN, 2013).

Apesar da grande demanda, muitos estudantes enfrentam dificuldades no aprendizado da programação, isso ocorre devido, a uma base muito pequena no ensino médio, ou até mesmo nenhuma base de programação nesse período. Tendo o primeiro contato com a programação apenas na graduação, e a isso se aplica várias dificuldades de entendimento sobre o assunto.

Pensando nessas dificuldade esse trabalho de conclusão de curso tem o objetivo procurar demonstrar a facilidade e a simplicidade da programação na linguagem de Programação Python, apoiando assim o estudo da programação.

Além da linguagem de programação Python nesse trabalho de conclusão de curso tem-se o objetivo de auxiliar os estudantes a conhecer o Arduino, uma placa de fácil acesso, preço acessível, e não necessita de grande conhecimento em eletrônica para a sua utilização.

Para se alcançar os objetivos propostos neste trabalho de conclusão de curso, será apresentado um estudo de caso que realiza a comunicação serial utilizando a linguagem de programação Python e o Arduino.

1.2 DELIMITAÇÃO DO ESTUDO

Motivar e expandir o conhecimento de estudantes iniciantes na área de programação; demonstrar a facilidade de se trabalhar com o a linguagem de programação Python e demonstrar a praticidade de se trabalhar com a tecnologia do Arduino.

1.3 PROBLEMA

A solução de vários problemas hoje está atrelado ao desenvolvimento de programas, sistemas, facilitando e auxiliando as atividades de todos. Porém há uma parcela de estudantes que enfrentam dificuldade de aprendizado nas disciplinas de programação.

1.4 OBJETIVOS

No desenvolvimento deste trabalho de conclusão de curso tem-se os seguintes objetivo geral e objetivos específicos.

1.4.1 Objetivo Geral

Motivar e expandir o conhecimento de estudantes iniciantes na área de programação, demonstrando a facilidade da utilização da linguagem de programação Python e a praticidade de se trabalhar com a tecnologia do Arduino.

1.4.2 Objetivos Específicos

Para atender ao objetivo geral neste trabalho de conclusão de curso os seguintes objetivos específicos serão abordados:

- Contextualizar e apresentar a linguagem de programação Python.

- Demonstrar por meio de códigos, os elementos básicos da programação: variáveis, tipos numéricos, operadores aritméticos, operadores relacionais, operadores lógicos, operador de atribuição e entrada e saída de dados.
- Utilizando a linguagem de programação Python apresentar as estruturas de controle: a) estruturas de controle com desvio de fluxo: comando condicional "*if*"; comando condicional "*if-else*" e comando "*if-elif*"; b) estruturas de controle com fluxo repetitivo: comando "*while*" e comando "*for*".
- Contextualizar e apresentar a placa Arduino: características e dados técnicos.
- Apresentar as estruturas de controle, elementos básicos da programação, funções e bibliotecas.
- Desenvolver e apresentar o estudo de caso demonstrando a comunicação serial entre a linguagem de programação Python e placa Arduino.

1.5 ESTRUTURA DO TRABALHO

Esta monografia de trabalho de conclusão de curso está dividida em 4 (quatro) seções:

1. Tutorial sobre Linguagem de Programação Python para Iniciantes: será apresentado uma introdução sobre a linguagem de programação Python, o processo de instalação da ferramenta, o funcionamento do modo interativo, ambientes de desenvolvimento, características básicas da linguagem, criação de um arquivo de código-fonte e os fundamentos da programação.
2. Tutorial sobre Arduino para Iniciantes: serão abordadas as informações importantes sobre a placa Arduino, sua descrição, características, dados técnicos, alimentação, entrada e saída de dados, comunicação, programação, opções de placa e, a apresentação da placa no desenvolvimento do projeto.
3. Integração entre a Linguagem de Programação Python e Arduino: tendo como base os procedimentos metodológicos e recursos, serão descritos o estudo de caso desenvolvido nesse trabalho de conclusão de curso, apresentado a comunicação serial entre a linguagem de programação Python e a plataforma Arduino.
4. Conclusão: serão retomados a pergunta de pesquisa e os seus objetivos e apontado como foram solucionados, respondidos, atingidos, por meio do trabalho realizado.

2 TUTORIAL SOBRE LINGUAGEM DE PROGRAMAÇÃO PYTHON PARA INICIANTES

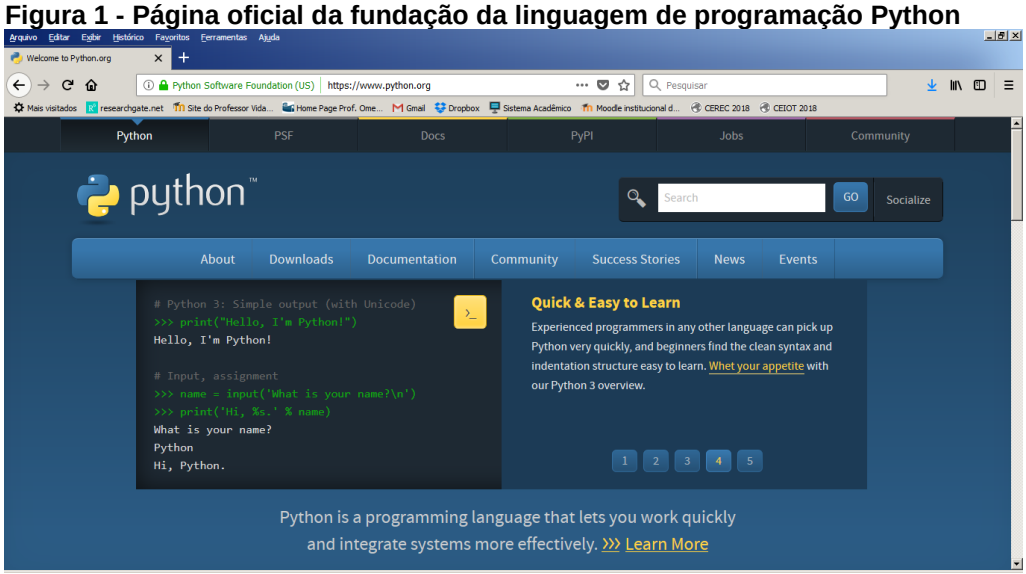
Nesta seção será apresentada a linguagem de programação Python mostrando as suas caraterísticas básicas, como realizar o processo de instalação, o modo de execução interativo, o ambiente de desenvolvimento IDLE e os elementos básicos para a programação de computadores utilizando a linguagem.

Os aspectos abordados sobre a linguagem de programação Python foram principalmente, levantados usando como referências: a) Livro “Introdução à Programação com Python: algoritmos e lógica de programação para iniciantes” de Nilo Ney Coutinho Menezes (MENEZES, 2014); e b) Material de aula para a linguagem de programação Python do Prof. Omero Francisco Bertol (BERTOL, 2018).

2.1 PROCESSO DE INSTALAÇÃO

Para realizar a instalação da linguagem de programação Python, precisa-se de um programa interpretador que aceita os comandos Python e os executa sentença por sentença. O interpretador é responsável por traduzir os programas (códigos-fonte) em um formato que pode ser executado pelo computador. O interpretador Python já vem instalado em ambientes operacionais como o Mac OS® e Linux®, porém não se encontra ativo no Microsoft Windows®, neste caso, deverá ser realizado o *download* do instalador, de forma recomendada, na página oficial da fundação que detém os direitos de propriedade intelectual da linguagem, mostrada na Figura 1.

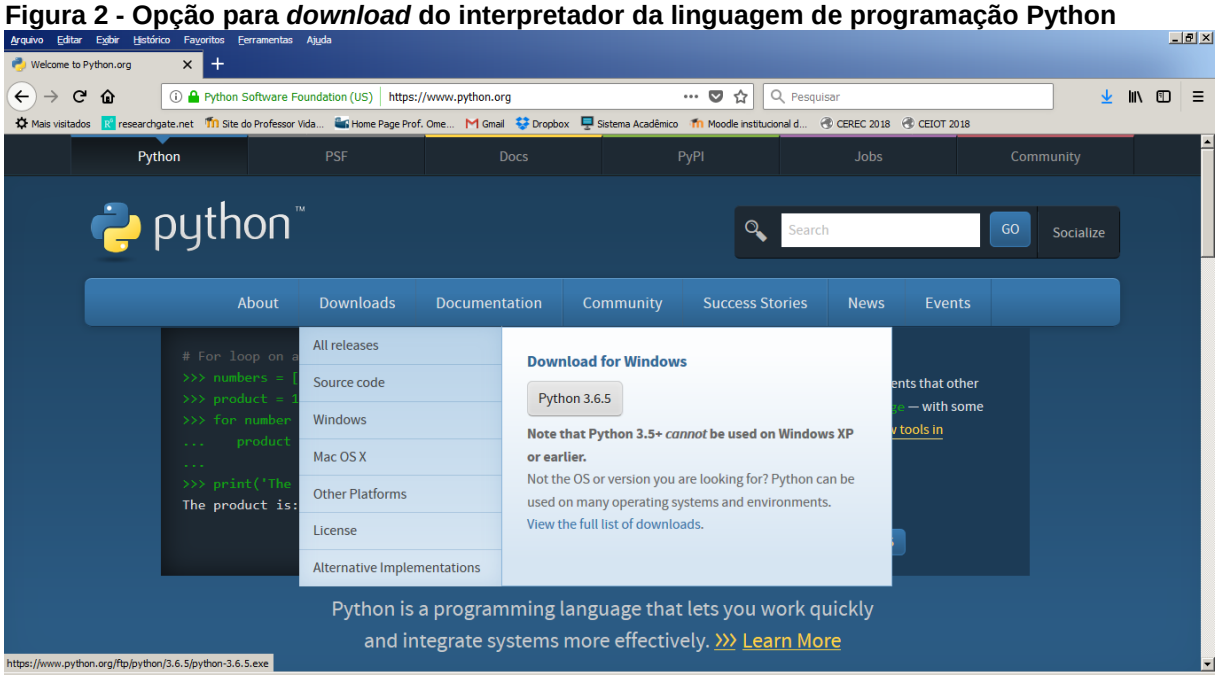
Figura 1 - Página oficial da fundação da linguagem de programação Python



Fonte: Python (2018).

Para realizar a transferência do instalador deve-se no menu, opção "*Downloads*", escolher o ambiente operacional adequado (por exemplo, Windows) e a versão mais atual da linguagem de programação Python (por exemplo, Python 3.6.5), como mostra a Figura 2.

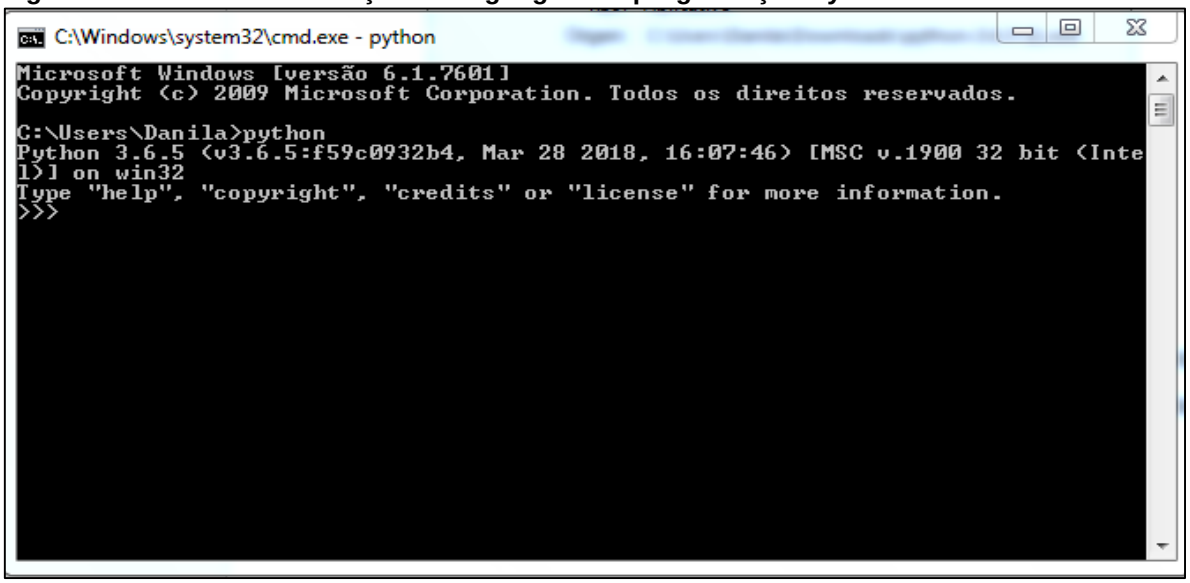
Figura 2 - Opção para *download* do interpretador da linguagem de programação Python



Fonte: Python (2018).

Concluído o processo de instalação recomenda-se verificar se o interpretador Python realmente foi instalado no computador. Esta verificação pode ser realizada abrindo-se o *prompt* (comando "cmd") do ambiente operacional Microsoft Windows®, por exemplo, e digitando "*python*", como mostra a Figura 3.

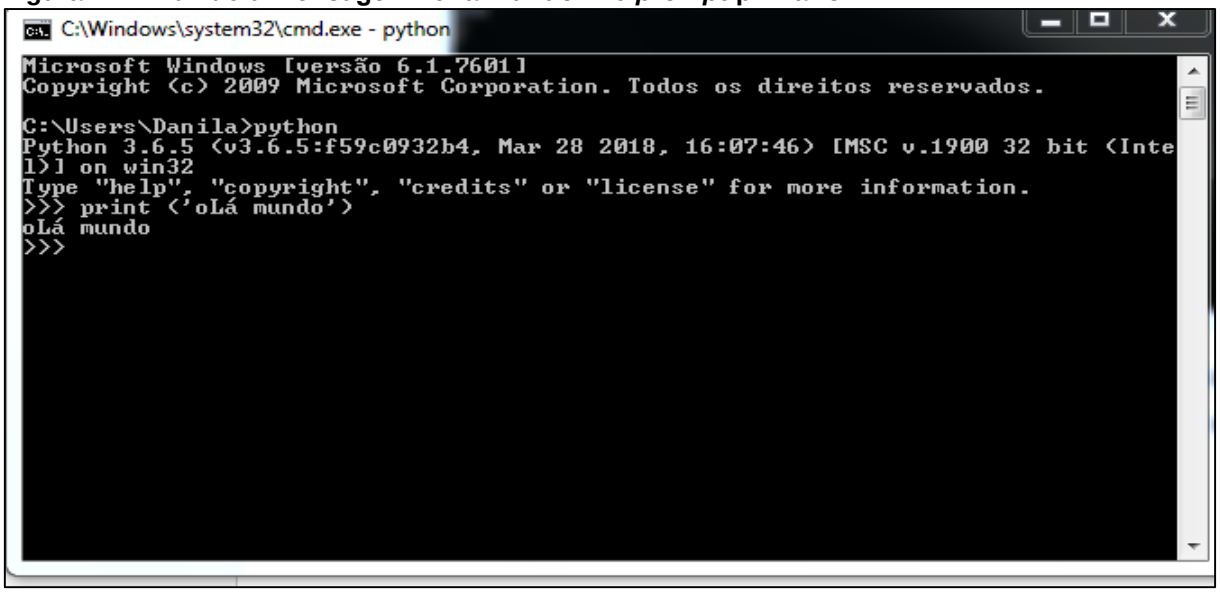
Figura 3 - Testando a instalação da linguagem de programação Python



Fonte: Autoria própria.

Na tela, apresentada na Figura 3, é possível verificar a versão do Python: Python 3.6.5, e também pode-se implementar o comando: *print ('oLá mundo')*, no *prompt* primário, indicado pelos símbolos ">>>", que define como saída/resultado a mensagem "*oLá mundo*", apresentada na Figura 4.

Figura 4 - Exibindo a mensagem "oLá mundo" no *prompt* primário



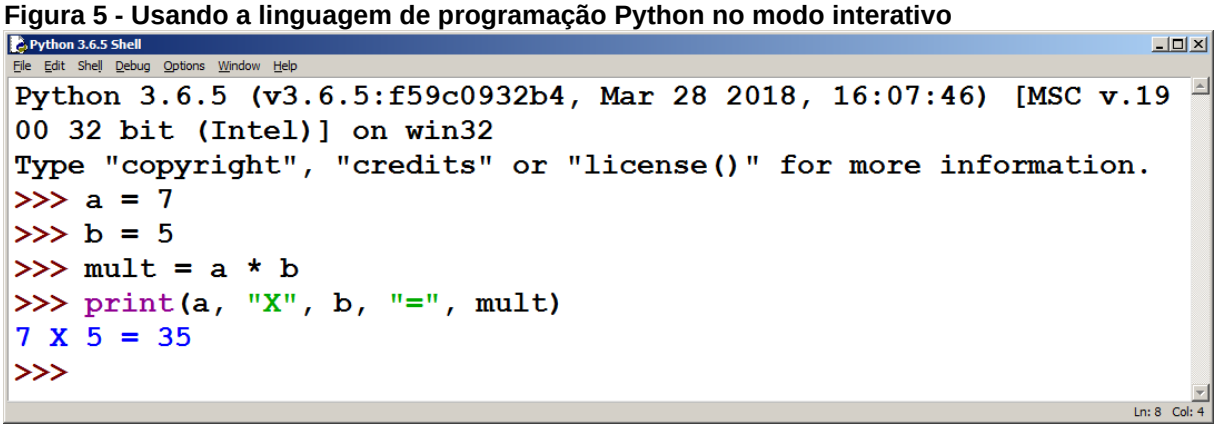
Fonte: Autoria própria.

2.2 MODO INTERATIVO

Quando os comandos/sentenças implementados na linguagem de programação Python são verificados e interpretados a partir do console (Figura 4), diz-se que o interpretador está em modo interativo. Nesse modo a implementação de comandos é realizada por meio do "*prompt* primário", identificado por três símbolos de "maior que" (>>>).

A Figura 5 mostra a implementação de comandos no modo interativo, por meio do *prompt* primário, para realizar a operação aritmética de multiplicação de dois valores atribuídos para a variável "a": >>> *a = 7* e para a variável "b": >>> *b = 5*. O resultado da operação é calculado e atribuído para a variável "mult": >>> *mult = a * b*. O resultado é apresentado no console por meio de um comando de saída "print": >>> *print(a, "X", b, "=", mult)*.

Figura 5 - Usando a linguagem de programação Python no modo interativo

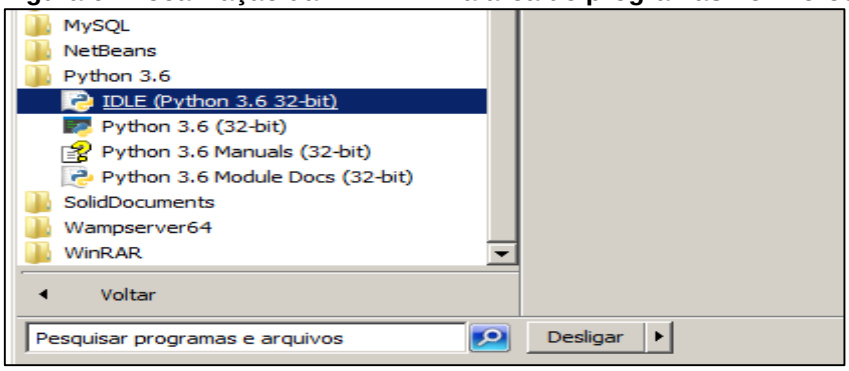


Fonte: Autoria própria.

2.3 AMBIENTE DE DESENVOLVIMENTO

Na implementação de programas (códigos-fonte) na linguagem de programação Python sugere-se a utilização de um ambiente integrado de desenvolvimento (ou do inglês, *Integrated Development Environment*, abreviado com a sigla IDE). Juntamente com a instalação do interpretador Python (Figura 2) é instalado a IDE denominada de IDLE, como mostra a Figura 6.

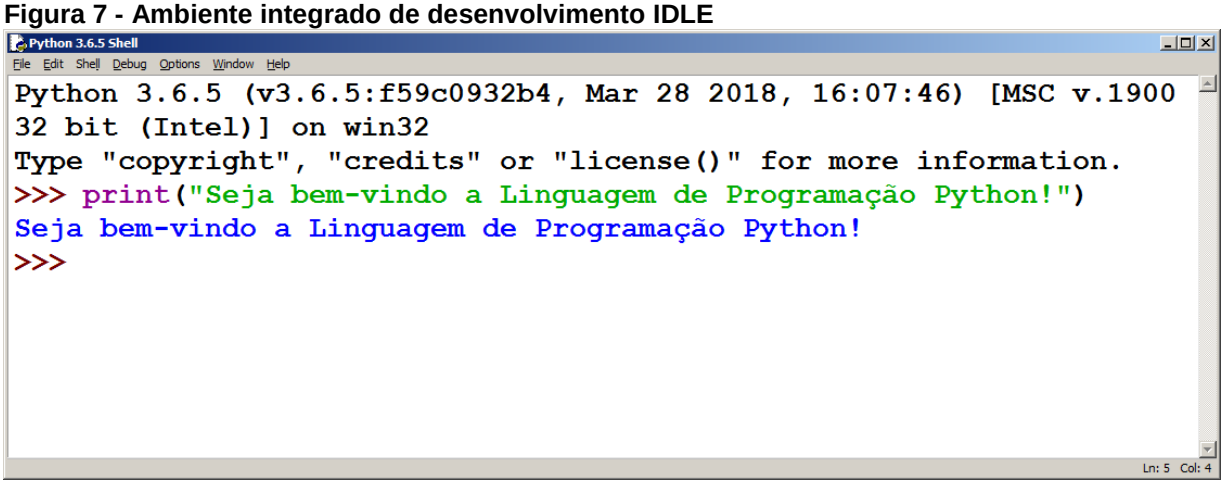
Figura 6 - Localização da IDE IDLE na área de programas no Microsoft Windows®



Fonte: Autoria própria.

A Figura 7 mostra a IDE IDLE aberta no modo interativo e por meio do *prompt* primário, indicado pelos símbolos “>>>”, é possível realizar a implementação do comando de saída “*print*” para mostrar no dispositivo padrão de saída, terminal de vídeo, a mensagem de boas-vindas: “Seja bem-vindo a Linguagem de Programação Python!”.

Figura 7 - Ambiente integrado de desenvolvimento IDLE



Fonte: Autoria própria.

2.3.1 Ambiente Integrado de Desenvolvimento PyCharm

A ferramenta PyCharm[1] é uma alternativa de IDE para a linguagem de programação Python. Essa ferramenta possui diversos recursos que facilitam o desenvolvimento de *software*, principalmente, quando comparado com o IDLE, IDE padrão do Python. O PyCharm é desenvolvido pela empresa da República Tcheca JetBrains, sendo escrito na linguagens Java e Python, e está disponível para vários sistemas operacionais, como Microsoft Windows®, Linux® e Mac OS®.

Existem várias licenças disponíveis para o uso do IDE PyCharm: a) *Professional Edition*, que é paga (gratuita para alguns casos específicos em uma versão *open source*), e b) *Community Edition*, distribuída sob licença Apache.

Entre os recursos presentes no PyCharm que podem ser destacados (PYCHARM, 2018), tem-se:

- *Debugger* gráfico.
- Unidade de testes integrada.
- Integração com sistemas de controle de versão, como *Git*, *Mercurial* e *Subversion*.
- Análise de código.
- *Code Completion*.
- Sintaxe e erros destacados.
- Refatoração.
- Suporte a desenvolvimento com *Django*.

[1] Ferramenta de Desenvolvimento PyCharm. Disponível em: <https://www.jetbrains.com/pycharm/>. Acesso em: 02 abr. 2018.

2.4 CARACTERÍSTICAS BÁSICAS DA LINGUAGEM

Python é uma linguagem de programação orientada a objetos, o que significa que ela possui recursos, classes e objetos, por exemplo, que dão suporte ao paradigma de programação orientada a objetos.

O projeto de desenvolvimento da linguagem de programação Python é *open source* (código aberto) e administrado pelo Python *Software Foundation*[2]. Python é multiplataforma[3] e uma linguagem interpretada, ou seja, que não gera código executável (*.exe), o código-fonte é interpretado durante a execução do programa.

Python é uma linguagem *case sensitive,* diferencia caracteres minúsculos dos respectivos caracteres maiúsculos, por exemplo: a) na implementação de sentenças, o comando para mostrar informações no dispositivo padrão de saída é sintaticamente correto quando escrito como "*print*", "*Print*" gera um erro de sintaxe[4]; b) nomes de identificadores, variável 'a' é diferente da variável 'A'.

Na linguagem de programação Python dois aspectos relacionados a tipagem de dados são importantes de destacar: a) tipagem forte; e b) tipagem dinâmica.

a) Tipagem forte: não há conversões automáticas de tipos, como mostrado no trecho de código da Listagem 1.

Listagem 1 - Demostrando a tipagem forte

```
>>> a = "7"

>>> b = 3

>>> type(a), type(b)
(<class 'str'>, <class 'int'>)

>>> result = a + b
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    result = a + b
TypeError: must be str, not int

>>>
```

Fonte: Autoria própria.

[2] Disponível em: <https://www.python.org/>. Acesso em: 28 abr. 2018v.
[3] Aquele que pode funcionar em várias plataformas (dispositivos) diferentes.
[4] Sentenças da linguagem (instruções, expressões) escritas incorretamente.

b) Tipagem dinâmica: não há declaração de variáveis, o tipo de uma variável é definido pelo valor atribuído a ela em tempo de execução. Assim, a variável pode mudar de tipo durante a execução do programa, como demonstra a Listagem 2.

Listagem 2 - Demonstrando a tipagem dinâmica

```
>>> a = 4

>>> type(a)
<class 'int'>

>>> a = 0.5
>>> type(a)
<class 'float'>

>>> a = "Estou aprendendo Python"
>>> type(a)
<class 'str'>

>>>
```

Fonte: Autoria própria.

No exemplo da Listagem 2, foi atribuído a variável “a” o valor *4* que representa um número inteiro (*class ‘int’*). A seguir, o valor atribuído para a variável “a” foi um valor com ponto flutuante, *0.5* (*class ‘float’*). Por último, o valor atribuído a variável “a” foi a sequência de caracteres *“Estou aprendendo Python”* (*class ‘str’*). Assim, nestes exemplos pode-se perceber as mudanças de tipo da variável em decorrência do valor a ela atribuído.

Todos os comandos digitados no *prompt* primário, indicado pelos símbolos “>>>”, quando submetidos são interpretados e executados, como pode ser observado na Listagem 3.

Listagem 3 - Demonstrando a execução de comandos Python no *prompt* primário

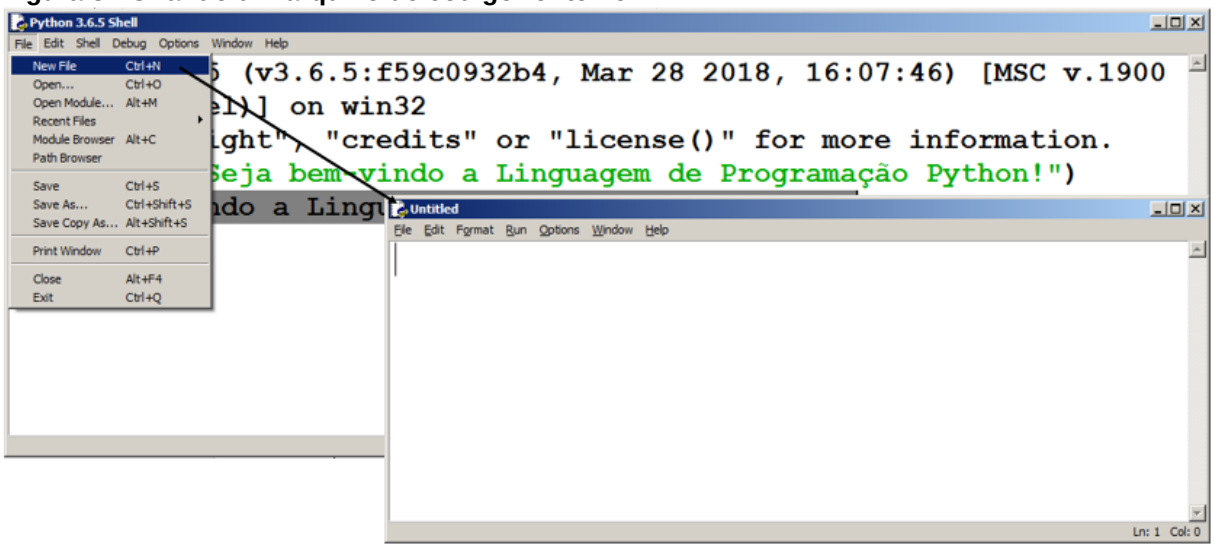
```
>>> print("Hello World!!!")
Hello World!!!
>>> print("Bem-vindo a Linguagem de Programação Python!")
Bem-vindo a Linguagem de Programação Python!
>>> a = 7
>>> b = 3
>>> result = a + b
>>> print("a =", a)
a = 7
>>> print("b =", b)
b = 3
>>> print("soma =", result)
soma = 10
```

Fonte: Autoria própria.

2.5 CRIAÇÃO DE UM ARQUIVO-FONTE

No aprendizado de uma linguagem de programação necessita-se de atividades práticas de desenvolvimento de programas (códigos-fonte). Para criar um arquivo de código-fonte na IDE IDLE deve-se escolher na barra de menu a opção "*File* | *New File*", como mostra a Figura 8.

Figura 8 - Criando um arquivo de código-fonte no IDLE



Fonte: Autoria própria.

Após a criação do arquivo de código-fonte pode-se escrever um programa na linguagem de programação Python, no caso apresentado na Listagem 4, tem-se um primeiro programa com o objetivo de apresentar uma mensagem de boas-vindas no dispositivo de saída.

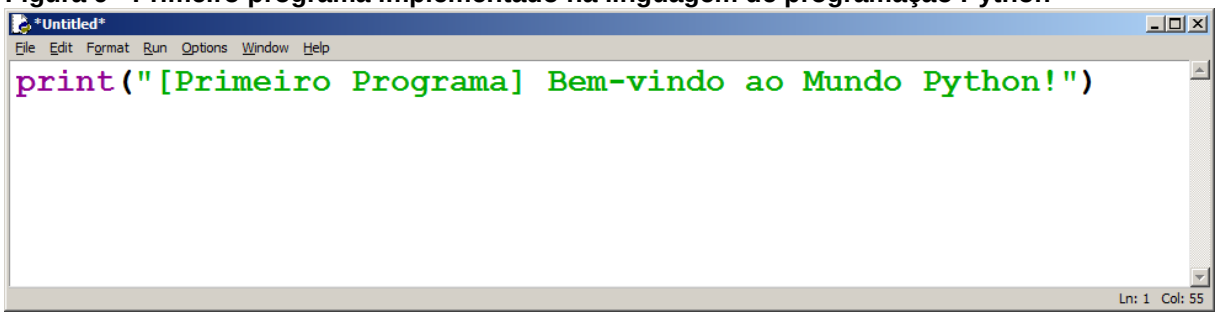
Listagem 4 - Primeiro programa na linguagem de programação Python

```
print("[Primeiro Programa] Bem-vindo ao Mundo Python!")
```

Fonte: Autoria própria.

A Figura 9 apresenta a conclusão do processo de implementação do primeiro programa (Listagem 4) e a seguir deve-se passar para a próxima etapa que corresponde a execução do programa. Por meio da opção do menu "*Run* | *Run Module F5*", o código-fonte implementado será interpretado e executado pelo interpretador da linguagem de programação Python.

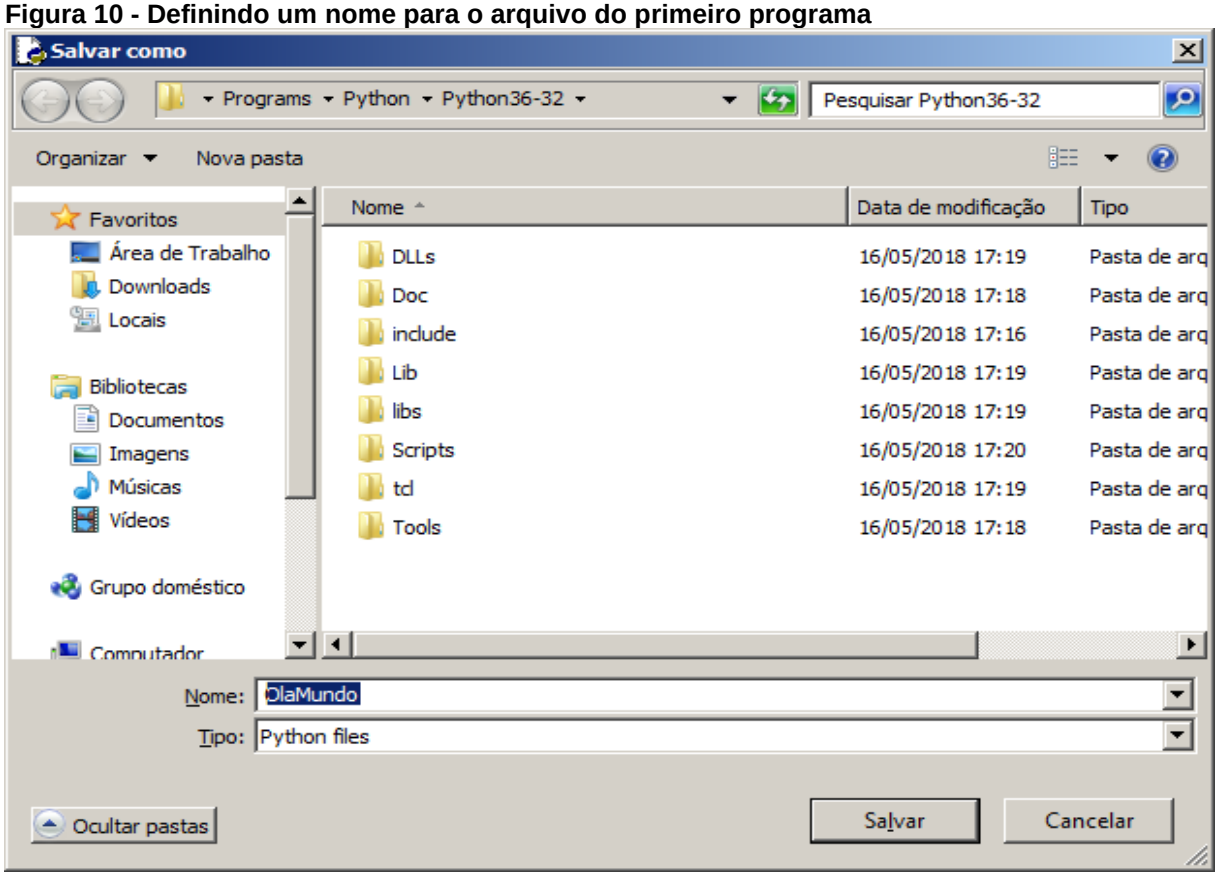
Figura 9 - Primeiro programa implementado na linguagem de programação Python



Fonte: Autoria própria.

Na primeira tentativa de execução de um programa será solicitado um nome, por exemplo “OlaMundo”, para identificar o código-fonte como pode ser observado na Figura 10.

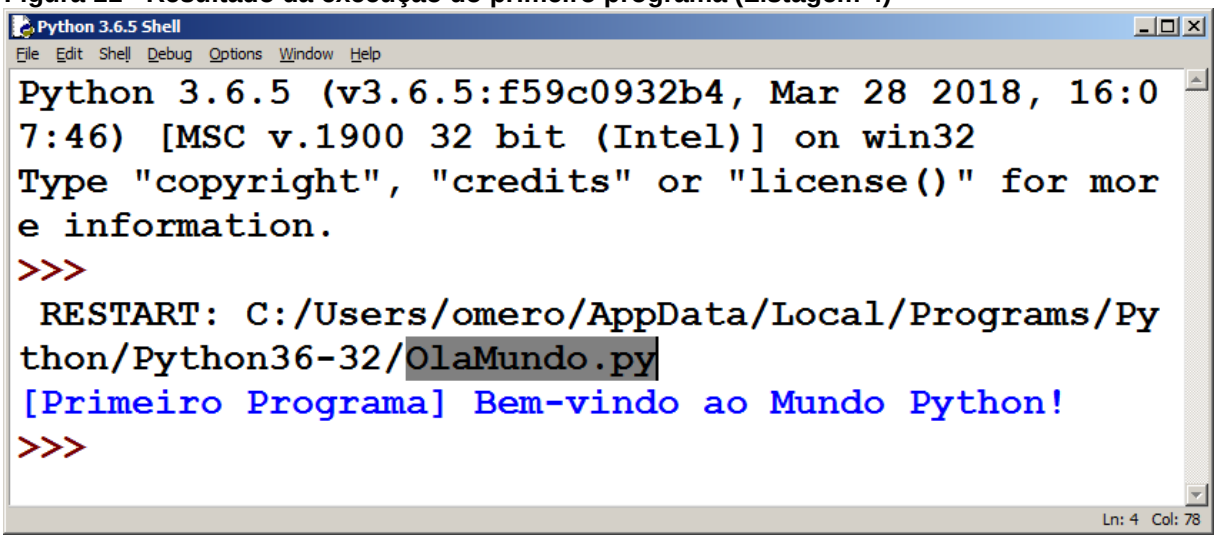
Figura 10 - Definindo um nome para o arquivo do primeiro programa



Fonte: Autoria própria.

O resultado da execução do primeiro programa implementado na linguagem de programação Python, identificado com o nome “OlaMundo.py”, pode ser observado na Figura 11.

Figura 11 - Resultado da execução do primeiro programa (Listagem 4)



Fonte: Autoria própria.

2.6 ELEMENTOS BÁSICOS DA PROGRAMAÇÃO

Na linguagem de programação Python existem cuidados necessários para que um programa execute sem erros, há alguns itens que devem ser considerados com muita atenção, entre eles:

a. Letras maiúsculas e minúsculas são diferentes (linguagem *case sensitive*).
b. Aspas são de grande importância no código, quando abre-se aspas, deve-se fechá-las, quando ocorrer o esquecimento disso, o seu programara não irá funcionar (erro de sintaxe[5]).
c. Parênteses quando usados devem ser aberto, símbolo “(“, e apresentando o correspondente fechamento, símbolo “)”.
d. Talvez a característica mais importante da linguagem de programação Python se apresentada na necessidade da identação de todo o código-fonte. Identar é colocar avanços ou recuos de texto, na forma de parágrafos, nas instruções em relação a margem ou ao início de uma cláusula de instrução para formar um bloco de código.

2.6.1 Variáveis

Variáveis são utilizadas em programação para armazenar valores e para dar nome a uma área de memória do computador onde são armazenados dados.

[5] Ocorre quando uma sentença da linguagem está mal formulada (escrita incorretamente).

Variável possui três atributos: a) nome, b) tipo de dado associado à mesma e, c) a informação por ela armazenada. A linguagem de programação Python possui vários tipos de dados, porém os mais comuns são números inteiros (*class* "*int*"), números de ponto flutuante (*class* "*float*"), valores booleanos (*class "bool"*)[6] e o tipo *Strings*. Variáveis do tipo *Strings* armazenam cadeias de caracteres como nomes e textos em um modo geral, delimitados pelos elementos sintáticos: aspas simples ou aspas duplas. Um trecho de código exemplificando a utilização dos tipos básicos em Python pode ser observado na Listagem 5.

Listagem 5 - Variáveis dos tipos *String*, *int* e *float*

```
>>> nome = "João"
>>> idade = 34
>>> peso = 83.500
>>> luzApagada = True

>>> print(nome, idade, peso, luzApagada)
(João, 34, 83.5, True)

>>> type(nome), type(idade), type peso)
(<class "str">, <class "int">, <class "float">)

>>> type(luzApagada)
<class "boll">
>>>
```

Fonte: Autoria própria.

2.6.2 Tipos Numéricos

Uma variável é definida como numérica quando a mesma armazena números inteiros (*class* "*int*") ou de ponto flutuante (*class* "*float*"). Números inteiros são aqueles valores negativos e positivos que não possuem parte decimal, como exemplo: 1, 2 ,7, 20, -10 e 100000. Já os números de ponto flutuante ou decimais são os que possuem uma parte decimal separada sintaticamente da parte inteira por um "ponto", por exemplo: 1.0, 2.5, 20.9, -10.8 e 100000.1.

Python utiliza, como na maioria das linguagens de programação, por exemplo: *Pascal*, *C* e *Java*, o "ponto" e não a vírgula para separar a parte inteira da parte fracionária de um número, isso se dá pelo fato de que a maioria da linguagens foram criadas utilizando o padrão da língua inglesa. Um trecho de código exemplificando a

[6] "True" representa o valor "verdadeiro, afirmação" e "False" o valor "falso, negação".

utilização dos tipos numéricos (inteiro e ponto flutuante) em Python pode ser observado na Listagem 6.

Listagem 6 - Exemplificando a utilização dos tipos numéricos

```
>>> idade = 55
>>> peso = 67.9

>>> print(idade, peso)
(55, 67.9, True)

>>> type(idade), type(peso)
(<class "int">, <class "float">)

>>>
```

Fonte: Autoria própria.

2.6.3 Operadores Aritméticos

Na linguagem de programação Python os operadores aritméticos, apresentados na Tabela 1, realizam operações matemáticas elementares, como por exemplo: soma (+), subtração (-), multiplicação (*) e divisão real (/). Há também operadores para exponenciação (**), obtenção da parte inteira de uma divisão (//) e extração do módulo ou resto da divisão (%).

Tabela 1 - Tabela dos operadores aritméticos na linguagem de programação Python

Operação	Operador
Adição	+
Subtração	-
Multiplicação	*
Divisão real	/
Divisão Inteira	//
Resto da divisão (módulo)	%
Exponenciação	**

Fonte: Autoria própria.

Na Listagem 7 pode-se observar um trecho de código[7], implementado no modo interativo, exemplificando a utilização dos operadores aritméticos na linguagem de programação Python.

[7] O símbolo “#” identifica a utilização de comentários de uma linha.

Listagem 7 - Exemplificando a utilização dos operadores aritméticos na linguagem de programação Python

```
>>> 5 + 5 # adição
10

>>> 75 - 30 # subtração
45

>>> 4 * 9 # multiplicação
36

>>> 18 / 9 # divisão real
2.0

>>> 9 // 2 # divisão inteira
4

>>> 9 % 2 # resto da divisão (módulo)
1

>>> 2 ** 5 # exponenciação: 2 elevado a 5
32

>>>
```

Fonte: Autoria própria.

2.6.4 Operadores Relacionais

Os operadores relacionais, apresentados na Tabela 2, são utilizados para relacionar valores e apresentando como resultado um valor do tipo lógico, podendo ser: a) *True* para indicar verdadeiro (afirmação) ou, b) *False* no caso do resultado falso (negação).

Tabela 2 - Tabela dos operadores relacionais na linguagem de programação Python

Operador	Operação	Símbolo Matemático
==	Igual a	=
>	Maior que	>
<	Menor que	<
!=	Diferente de	≠
>=	Maior ou igual a	≥
<=	Menor ou igual a	≤

Fonte: Autoria própria.

Na Listagem 8 pode-se observar um trecho de código, implementado no modo interativo, exemplificando a utilização dos operadores relacionais na linguagem de programação Python.

Listagem 8 - Exemplificando a utilização dos operadores relacionais na linguagem de programação Python

```
>>> x = 20
>>> y = 10
>>> z = 10
>>> x == z # operador relacional de igualdade
False
>>> y == z
True
>>> x > z # operador relacional maior que
True
>>> x > y, y > x
(True, False)
>>> x <= y,  x <= z # operador relacional menor ou igual
(False, False)
>>> y >= x # operador relacional maior ou igual
True
>>> x != y, x != z # operador relacional diferente
(True, True)
>>>
```

Fonte: Autoria própria.

2.6.5 Operadores Lógicos

A linguagem de programação Python suporta três operadores lógicos básicos, são eles: *not* (não, negação), *and* (e lógico), *or* (ou lógico). Esses operadores podem ser traduzidos como: não (¬negação), e (∧ conjunção) e, ou (v disjunção).

Cada operador obedece a um conjunto simples de regras, expresso na tabela verdade do operador. A tabela verdade, apresentada na Tabela 3, demonstra o resultado de uma operação com um ou dois valores lógicos e operandos. Quando um operador utiliza apenas um operando, é chamado de operador unitário. Quando é utilizado dois operandos, é chamado de operador binário. O operador *not* é um operador unitário, *or* e *and* são operadores binários, que necessitam envolver dois operandos na expressão.

Tabela 3 - Tabela verdade dos operadores lógicos *not*, *and* e *or*

A	B	not A	not B	A and B	A or B
V	V	F	F	**V**	V
V	F	F	V	F	V
F	V	V	F	F	V
F	F	V	V	F	**F**

Fonte: Autoria própria.

Na Listagem 9 pode-se observar um trecho de código, implementado no modo interativo, exemplificando a utilização dos operadores lógicos na linguagem de programação Python.

Listagem 9 - Exemplificando a utilização dos operadores lógicos

```
>>> A = True
>>> B = False
>>> not A
False
>>> not B
True
>>> A and B
False
>>> A or B
True
>>>
```

Fonte: Autoria própria.

2.6.6 Operador de Atribuição

A atribuição de valores é a passagem (ou alocação) de informação a uma determinada variável. Toda variável, por sua definição, pode receber valores ou então, pode ter seu valor alterado. A linguagem de programação Python tem definido que o sinal, conhecido pelo símbolo de igualdade (=), será o operador de atribuição. A notação para atribuição de valores define que a variável que receberá o valor estará do lado esquerdo do sinal de atribuição “=”, enquanto o valor a ser atribuído, estará do lado direito. Assim, é correto afirmar que a atribuição de valores sempre seguira o sentido da direita para a esquerda, como pode ser observado na Listagem 10.

Listagem 10 - Exemplificando a utilização do operador de atribuição

```
>>> nome = "Maria"
>>> numero = 10
>>> sexo = "F"
>>> salario = 1800.00
>>> flag = False
>>> print(nome, numero, sexo, salario, flag)
Maria, 10, F, 1800.00, False
>>> type(nome), type(numero), type(sexo)
(<class "str">, <class "int">, <class "str">)
>>> type(salario), type(flag)
(<class "float">, <class "bool">)
>>>
```

Fonte: Autoria própria.

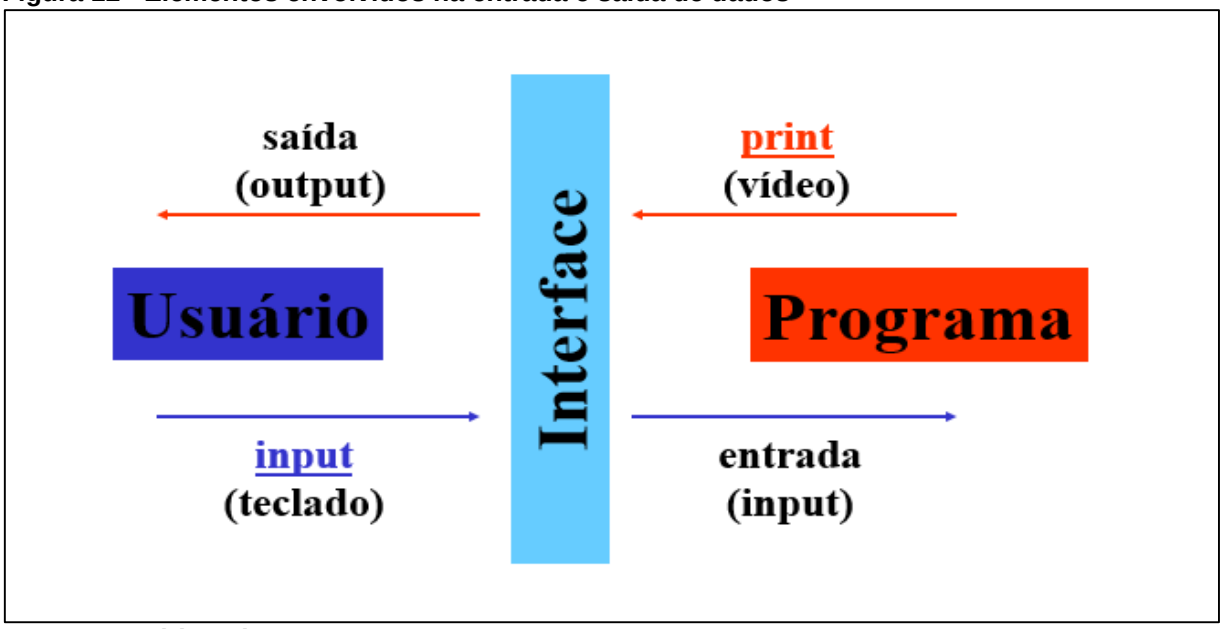
2.6.7 Entrada e Saída de Dados

Entrada de dados (*input*), representado principalmente pelo comando "*input*", é o meio pela qual as informações, dados ou valores são enviadas do usuário para o computador, isso é possível por meio de dispositivos de entrada de dados, como por exemplo: um teclado de computador, discos magnéticos, fitas e cartão perfurado.

Saída de dados (*output*), representado principalmente pelo comando "*print*", é o meio pelo qual o computador transfere as informações para o usuário, ou aos níveis secundários de memória, isso pode ocorrer no monitor de vídeo, impressora, fitas e discos magnéticos.

Para exemplificar a utilização dos comandos de entrada (*input*) e saída (*print*) é necessário definir que a sintaxe (escrita do comando), é a maneira como os comandos são escritos, de forma que esses comandos possam ser entendidos pelo tradutor (compilador, interpretador) de programas. Quando ocorre um erro de sintaxe o comando não é reconhecido pelo interpretador e gera um erro de compilação. Já a semântica (lógica do comando) é o conjunto de ações que serão exercidas pelo computador durante a execução do comando. A "*interface*" do usuário é o meio de comunicação entre usuário e programa. Este elementos de definição podem ser observados na Figura 12.

Figura 12 - Elementos envolvidos na entrada e saída de dados



Fonte: Bertol (2017).

2.6.7.1 Comandos de entrada e saída de dados

Comandos de entrada de dados permite que o usuário do programa possa fornecer valores de entrada por meio do teclado, por exemplo, e também observar resultados gerados no dispositivo de saída, como por exemplo o terminal de vídeo.

A entrada de dados é implementada utilizando a função "*input*()", por exemplo: *nome = input ("Qual é o seu nome? ")*, e a saída de dados por meio da função "*print*()", por exemplo: *print("Bom Aprendizado em Python Sr(a). ", nome)*.

Um trecho de código implementado na linguagem de programação Python para exemplificar a utilização dos comandos de entrada (*input*) e saída (*print*) de dados é mostrado na Listagem 11.

Listagem 11 - Exemplificando a implementação dos comandos de entrada e saída de Dados

```
nome = input("Qual é o seu nome? ")

print("Bom Aprendizado em Python Sr(a).", nome)
```

Fonte: Autoria própria.

No Quadro 1 pode-se observar a execução do trecho de código (Listagem 11) que exemplifica a implementação dos comandos de entrada e saída de dados.

Quadro 1 - Resultado da execução da Listagem 9: entrada e saída de dados

```
Qual é o seu nome? Maria
Bom Aprendizado em Python Sr(a). Maria
>>>
```

Fonte: Autoria própria.

2.6.8 Estrutura de Controle

Um programa de computador é uma sequência de instruções organizadas de forma a produzir a solução de um determinado problema, o que representa uma das habilidades básicas da programação, que é a sequenciação.

As instruções de um programa são executadas em sequência, isso é conhecido como fluxo sequencial de execução. Porém, em inúmeras ocasiões é necessário executar as instruções de um programa em uma ordem diferente da estritamente sequencial. Muitas situações são caracterizadas pela necessidade da repetição de instruções individuais ou conjuntos de instruções, e também pelo desvio do fluxo de execução.

2.6.8.1 Estruturas de controle com desvio de fluxo

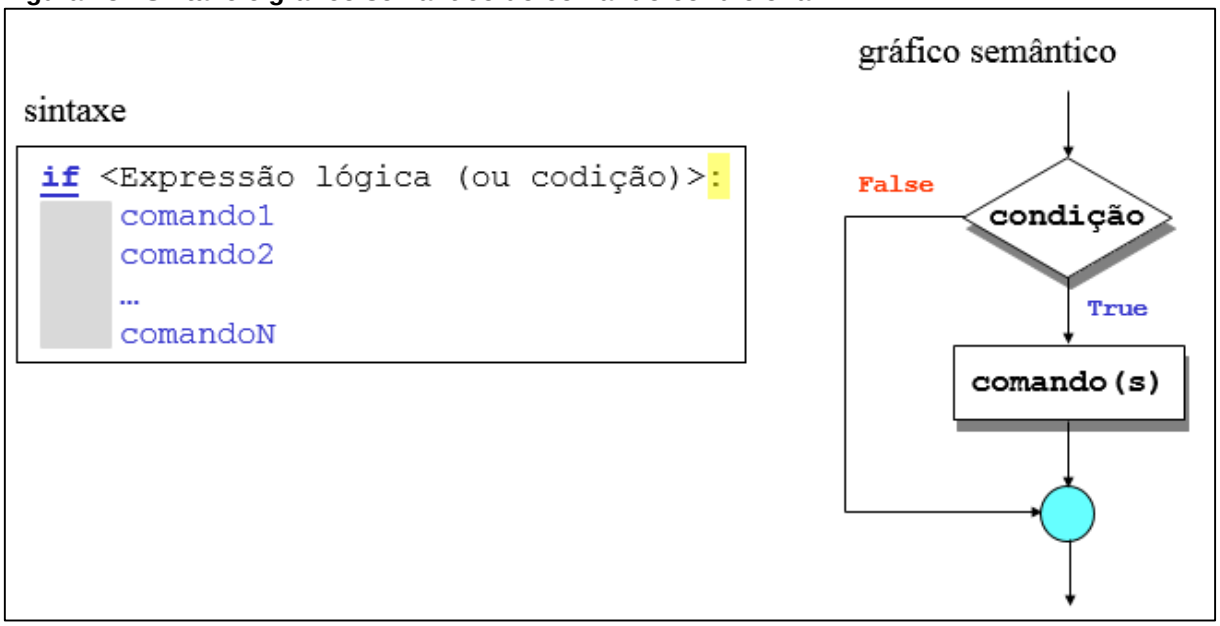
Não serão todas as vezes que as linhas de comando serão todas executadas. Precisa-se decidir quais linhas de comando do programa devem ser executadas com base no resultado de uma condição.

As decisões ajudam a selecionar quando uma parte do programa deve ser ativada e outras vezes deve ser ignorada. Na linguagem de programação *Python* existem basicamente 3 (três) formas de implementação de estruturas de controle com desvio de fluxo, são elas: a) "*if*": estrutura de decisão na forma simples; b) "*if-else*": estrutura de decisão na forma composta; e, c) "*if-elif*": estrutura de decisão na forma encadeada.

2.6.8.1.1 Comando condicional "if"

No comando condicional "*if*", com a sintaxe e o gráfico semântico[8] apresentados na Figura 13, o fluxo de execução passa pela sequência de comandos (comando1; comando2; ... comandoN) se o resultado da "Expressão lógica (ou codição)" definir como resultado o valor verdadeiro (*True*). Caso o resultado apresente o valor falso (*False*), nenhum comando será executado.

Figura 13 - Sintaxe e gráfico semântico do comando condicional "*if*"



Fonte: Bertol (2017).

[8] Apresenta o fluxo lógico de execução do comando.

No código, apresentado na Listagem 12, foram implementados 5 (cinco) comandos condicionais "*if*" para verificar diversas relações com um valor inteiro de entrada (variável "n").

Listagem 12 - Exemplificando a implementação do comando condicional "*if*"

```
n = int(input("Informe um número inteiro: "))

print() # deixa uma linha em branco na saída

if (n == 10):
    print(n, "é igual a 10")

if (n != 10):
    print(n, "é diferente de 10")

if (n < 10):
    print(n, "é menor que 10")

if (n > 10):
    print(n, "é maior que 10")

if ((n >= 10) and (n <= 21)):
    print(n,"PERTENCE AO INTERVALO" \
          " DE 10 ATÉ 21")
```

Fonte: Autoria própria.

No Quadro 2 pode-se observar a execução do código da Listagem 12 que exemplifica a implementação de comandos condicionais "if".

Quadro 2 - Resultado da execução da Listagem 12: comando condicional "*if*"

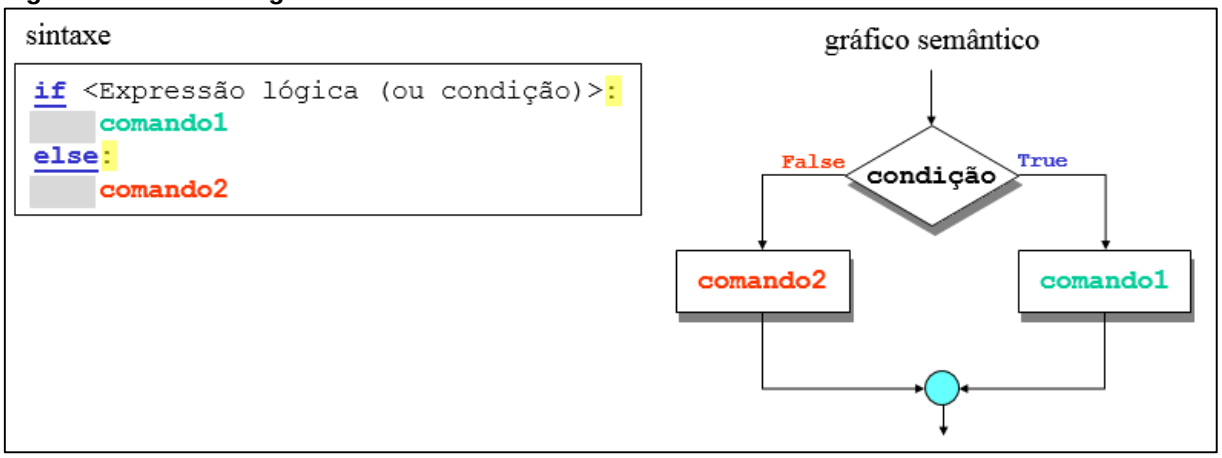
```
Informe um número inteiro: 17

17 é diferente de 10
17 é maior que 10
17 PERTENCE AO INTERVALO DE 10 ATÉ 21

>>>
```

Fonte: Autoria própria.

2.6.8.1.2 Comando condicional "if-else"

No comando condicional "*if-else*", com a sintaxe e o gráfico semântico apresentados na Figura 14, o fluxo de execução será capaz de selecionar um entre dois caminhos distintos para execução. O "comando1" será executado sempre que a "Expressão lógica (ou condição)" apresentar como resultado o valor verdadeiro (*True*). Caso o resultado apresente o valor falso (*False*), o "comando2" será executado.

Figura 14 - Sintaxe e gráfico semântico do comando condicional "*if-else*"



Fonte: Bertol (2017).

No código, apresentado na Listagem 13, foi implementado um comando condicional "*if-else*" para verificar se um valor inteiro de entrada (variável "a") corresponde a um número par ou ímpar.

Listagem 13 - Exemplificando a implementação do comando condicional "*if-else*"

```
a = int(input("Informe um número inteiro: "))

print()

if ((a % 2) == 0):
   print(a, "é um número \"par\".")
else:
   print(a, "é um número \"ímpar\".")
```

Fonte: Autoria própria.

No Quadro 3 pode-se observar a execução do código da Listagem 13 que exemplifica a implementação do comando condicional "*if-else*".

Quadro 3 - Resultado da execução da Listagem 13: comando condicional "*if-else*"

```
Informe um número inteiro: 27
27, é um número "ímpar".
>>>

Informe um número inteiro: 28
28, é um número "par".
>>>
```

Fonte: Autoria própria.

2.6.8.1.3 Comando condicional "if-elif"

A linguagem de programação Python oferece o comando condicional "*if-elif*" como uma alternativa para a implementação de múltiplos "*if-else*"*s* aninhados (ou encadeados), exemplificados na Listagem 14.

Listagem 14 - Exemplificando a implementação de estruturas condicionais "*if-else*"s aninhadas

```
a = int(input("Informe um valor: "))

b = int(input("Informe outro valor: "))

print()

if (a < b):
    print(a, "é menor que", b)
else:
    if (a == b):
       print(a, "é igual a", b)
    else:
       print(a, "é maior que", b)
```

Fonte: Autoria própria.

O comando condicional "*if-elif*" é um comando que implementa estruturas condicionais "*if-else*" encadeadas com o mesmo resultado semântico. O código-fonte, apresentado na Listagem 15, da mesma forma como foi implementado na Listagem 14, verifica a relação existente entre dois valores inteiros (variáveis "a" e "b").

Listagem 15 - Primeiro exemplo da implementação do comando condicional "*if-elif*"

```
a = int(input("Informe um valor: "))

b = int(input("Informe outro valor: "))

print()

if (a < b):
   print(a, "é menor que" , b)
elif (a == b):
   print(a, "é igual a", b)
else:
   print(a , "é maior que", b)
```

Fonte: Autoria própria.

No Quadro 4 pode-se observar a execução do código da Listagem 15 que exemplifica a implementação do comando condicional "*if-elif*". O resultado semântico apresentado com a execução do código da Listagem 15 (Quadro 4) é o mesmo que será obtido quando for executado o código da Listagem 14.

Quadro 4 - Resultado da execução da Listagem 15: comando condicional "*if-elif*"

```
Informe um valor: 5
Informe outro valor: 10

5 é menor que 10
>>>
Informe um valor: 5
Informe outro valor: 5

5 é igual a 5
>>>
Informe um valor: 10
Informe outro valor: 5

10 é maior que 5
>>>
```

Fonte: Autoria própria.

Na Listagem 16, tem-se outro exemplo da utilização do comando condicional *"if-elif*", neste caso, implementado para verificar e mostrar o nome de um mês do ano. O dado de entrada (*input*) é atribuído para a variável "mes".

Listagem 16 - Segundo exemplo da implementação do comando condicional "*if-elif*"

```
mes = int(input("Informe o mês: "))
print()
if (mes == 1):
    print("JANEIRO")
elif (mes == 2):
    print("FEVEREIRO")
elif (mes == 3):
    print("MARÇO")
elif (mes == 4):
    print("ABRIL")
elif (mes == 5):
    print("MAIO")
elif (mes == 6):
    print("JUNHO")
elif (mes==7):
    print("JULHO")
elif (mes == 8):
    print("AGOSTO")
elif (mes == 9):
    print("SETEMBRO")
elif (mes == 10):
    print("OUTUBRO")
elif (mes == 11):
    print("NOVEMBRO")
else:
    print("DEZEMBRO")
```

Fonte: Autoria própria.

No Quadro 5 pode-se observar a execução do código da Listagem 16 que exemplifica a implementação do comando condicional "*if-elif*".

Quadro 5 - Resultado da execução da Listagem 16: comando condicional "*if-elif*"

```
Informe o mês: 10

OUTUBRO
>>>

Informe o mês: 2

FEVEREIRO
>>>

Informe o mês: 11

NOVEMBRO
>>>

Informe o mês: 12

DEZEMBRO
>>>

Informe o mês: 1

JANEIRO
>>>
```

Fonte: Autoria própria.

2.6.8.2 Estruturas de controle com fluxo repetitivo

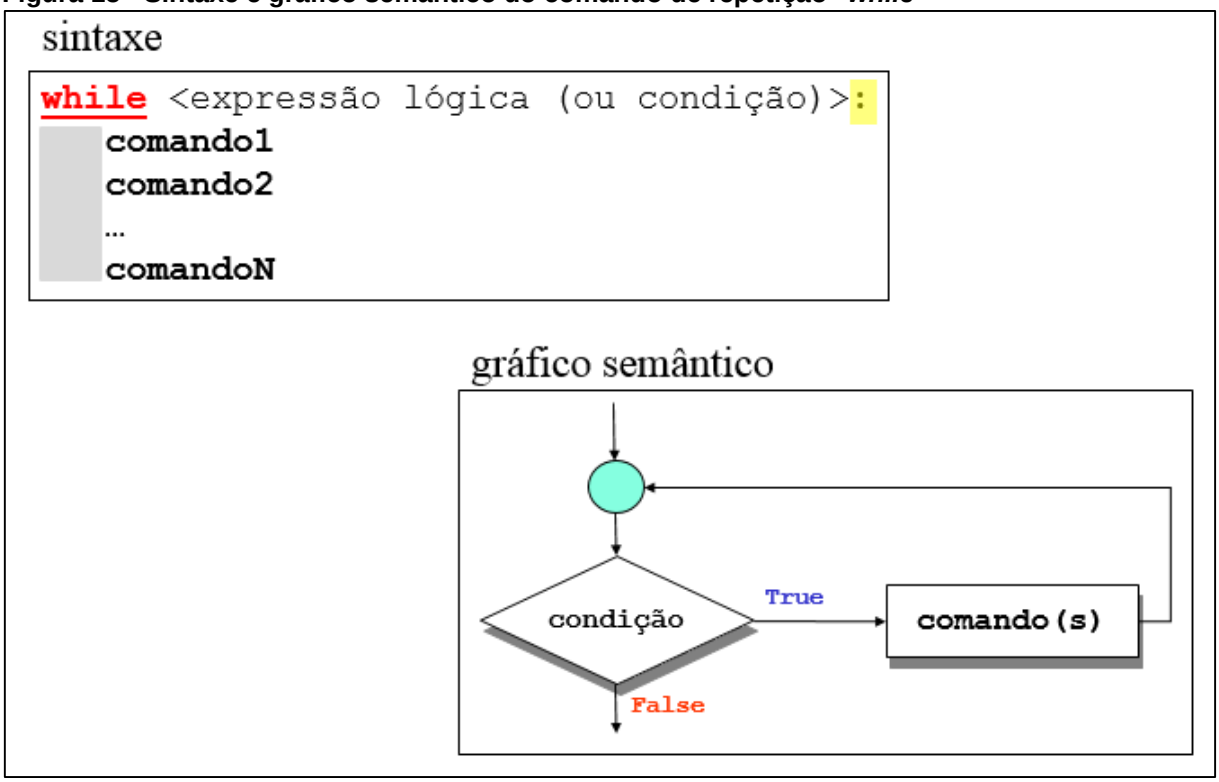
As estruturas de repetição, comandos "*while*" e "*for*", são estruturas de controle de desvio do fluxo que determinam que um conjunto de comandos possa ser executado/repetido um determinando números de vezes.

2.6.8.2.1 Comando de repetição "while"

O comando de repetição "*while*", com a sintaxe e o gráfico semântico apresentados na Figura 15, é uma estrutura de repetição condicional no qual o controle do número de repetição de execução do comando é realizado por meio da avaliação de uma "expressão lógica (ou condição)" associada: "enquanto a condição for verdadeira (*True*) o conjunto de comandos: comando1, comando2, ..., comandoN; é executada". Essas estruturas são adequadas para permitir a execução repetida de um ou mais comados por um número indeterminado de vezes, esse número não é

conhecido durante a programação, porém se torna conhecido durante a execução do mesmo, pode ser um valor fornecido pelo usuário, obtido de um arquivo, ou até mesmo de cálculos realizados por meio de outros dados.

Figura 15 - Sintaxe e gráfico semântico do comando de repetição "*while*"



Fonte: Bertol (2017).

No código, apresentado na Listagem 17, foi implementado um comando de repetição "*while*" que repete a execução dos comandos: "*print*(i)" e "i = i + 1"; enquanto a condição "i < 10" for verificada como verdadeira.

Listagem 17 - Exemplificando a implementação do comando de repetição "*while*"

```
i = 0
while (i < 10):
  print(i)
  i = i + 1
```

Fonte: Autoria própria.

No Quadro 6 pode-se observar a execução do código da Listagem 17 que exemplifica a implementação do comando de repetição "*while*". Nesta implementação, a variável "i", chamada de variável de controle, começa com o valor "0" e no processo repetitivo ela avança de 1 em 1 com a instrução "i = i + 1" até o limite final da repetição definido pelo valor "10". Os valores da variável "i" são mostrados pelo comando de saída "*print*(i)" a cada repetição.

Quadro 6 - Resultado da execução da Listagem 17: comando de repetição "*while*"

```
0
1
2
3
4
5
6
7
8
9
>>>
```

Fonte: Autoria própria.

2.6.8.2.2 Comando de repetição "for"

A repetição de uma das tarefas corriqueiras da programação, é utilizada para realizar muitas operações como contagens, totalizações, a obtenção de múltiplos dados, impressões, entre outras. O comando de repetição "*for*" executa um ciclo (um passo) para cada elemento do objeto que está sendo repetido[9]. Há ocasiões que necessita-se de uma variável de controle que possa ser incrementada ou decrementada a cada ciclo (ou passo), a forma mais simples para que isso seja realizado, é gerando uma lista com a função "*range*()".

O comando de repetição "*for*", com a sintaxe apresentada na Figura 16, sempre irá percorrer listas, seja uma lista de valores numéricos, uma cadeia de caracteres (*String*), ou uma lista de objetos distintos. A "variável" declarada na sentença do comando "for" se comporta como um item, definido pelo operador "in", da lista a partir do primeiro elemento lista até o último.

Figura 16 - Sintaxe do comando de repetição "*for*"

```
for <variável> in <lista>:
    comando1
    comando2
    ...
    comandoN
```

Fonte: Bertol (2018).

No código, apresentado na Listagem 18, foi implementado um comando de repetição "*for*" com a função "*range*()". A configuração "*range*(1, 11)", determina que a

[9] Cada repetição efetuada é o que se chama "iteração".

sentença “*print*("%d X %2d = %2d" % (n, i, n*i))” será executada 10 vezes montando a tabuada de “n”.

Listagem 18 - Primeiro exemplo da implementação do comando de repetição “*for*”

```
# Tabuada de um número Inteiro
n = int(input("Informe um nro para o cálculo da tabuada: "))
print()
for i in range(1, 11):
  print("%d X %2d = %2d" % (n, i, n*i))
```

Fonte: Autoria própria.

No Quadro 7, pode-se observar a execução do código da Listagem 18 que exemplifica a implementação do comando de repetição “*for*”. Nesta implementação, existe uma entrada de dados, atribuída a variável “n”, na qual deve-se informar o número que será usado para fazer a tabuada. A variável de controle “i” utilizada no comando “*for*”, começa com o valor “1” e no processo repetitivo ela avança de 1 em 1 até o limite final “10”. Estes limites são determinados pela implementação “*range* (1, 11)”. Os valores das variáveis “n” e “i” e da expressão “n * i” são mostrados pelo comando de saída “*print*” a cada repetição montando assim a tabuada de “n”.

Quadro 7 - Resultado da execução da Listagem 18: comando de repetição “*for*”

```
Informe um nro para o cálculo da tabuada: 9
9 X  1 =  9
9 X  2 = 18
9 X  3 = 27
9 X  4 = 36
9 X  5 = 45
9 X  6 = 54
9 X  7 = 63
9 X  8 = 72
9 X  9 = 81
9 X 10 = 90
>>>

Informe um nro para o cálculo da tabuada: 7
7 X  1 =  7
7 X  2 = 14
7 X  3 = 21
7 X  4 = 28
7 X  5 = 35
7 X  6 = 42
7 X  7 = 49
7 X  8 = 56
7 X  9 = 63
7 X 10 = 70
>>>
```

Fonte: Autoria própria.

A função “*range* ()” implementada em estruturas de repetição “*for*“ apresenta a seguinte sintaxe: *range*([*start*,]*stop*[, *step*]). Onde: a) *start*: valor inicial da lista (omitido assume o valor *default* 0); b) *stop*: valor final da lista (corresponde ao "valor especificado-1"); e c) *step*: passo (omitido assume o valor default 1). Por exemplo:

- *range* (10): sequência resultado = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- *range* (6): sequência resultado = 0, 1, 2, 3, 4, 5.
- *range* (5, 11): sequência resultado = 5, 6, 7, 8, 9, 10.
- *range* (10, 21): sequência resultado = 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20.
- *range* (0, 16, 3): sequência resultado = 0, 3, 6, 9, 12, 15.
- *range* (5, 31, 5): sequência resultado = 5, 10, 15, 20, 25, 30.
- *range* (10, -1, -1): sequência resultado = 10, 9, 8, 7, 6, 5, 4, 3, 2, 1.

No segundo exemplo da implementação do comando “*for*”, apresentado na Listagem 19, foi implementado um comando de repetição “*for*” para executar um ciclo (um passo) para cada um dos elementos (ou itens) do objeto lista identificado pela variável “x”.

Listagem 19 - Segundo exemplo da implementação do comando de repetição “*for*”

```
x = [7, 2, 0, 1, 8, 6, 9, 8, 4]

print("x = [ ", end="")

for item in x:
  print(item, end=" ")

print("]")
```

Fonte: Autoria própria.

No Quadro 8, pode-se observar a execução do código da Listagem 19 que exemplifica a implementação do comando de repetição “*for*”. Nesta implementação, a cada passo de execução, a variável “item” recebe um elemento que pertence (operador “in”) ao objeto lista “x”. Por meio do comando de saída “*print*” o valor da variável “item”, juntamente com um espaço em branco (*end*=” “), é mostrado no dispositivo padrão de saída.

Quadro 8 - Resultado da execução da Listagem 19: comando de repetição “*for*”

```
x = [ 7 2 0 1 8 6 9 8 4 ]
>>>
```

Fonte: Autoria própria.

No terceiro exemplo da implementação do comando "*for*", apresentado na Listagem 20, foi implementado um comando de repetição "for" para executar um ciclo (um passo) para cada um dos elementos (ou itens) do objeto *String*[10] identificado pela variável "s".

Listagem 20 - Terceiro exemplo da implementação do comando de repetição "*for*"

```
s = "UTFPR"

for caractere in s:
  print("{", caractere, "}", sep="")
```

Fonte: Autoria própria.

No Quadro 9, pode-se observar a execução do código da Listagem 20 que exemplifica a implementação do comando de repetição "for". Nesta implementação, a cada passo de execução, a variável "caractere" recebe um elemento que pertence (operador "in") ao objeto *String* "s". Por meio do comando de saída "*print*" o valor da variável "caractere", entre chaves "{"e "}" e sem espaço de separação (*sep*="") é mostrado no dispositivo padrão de saída.

Quadro 9 - Resultado da execução da Listagem 20: comando de repetição "*for*"

```
{U}
{T}
{F}
{P}
{R}
>>>
```

Fonte: Autoria própria.

Nesta seção foi apresentado as maneiras de se trabalhar com a linguagem de programação Pyhton, foram demonstrados de maneira clara nesse tutorial o processo de instalação, e todas as informações importantes estão nessa seção.

No aprendizado é necessário que se conheça os elementos básicos como foi mostrado na seção. Foi apresentado vários exemplos de código que são possíveis de se fazer e implementar, para que estes sejam repetidos e estudados pelos iniciantes, para que desse modo o estudante possa repetir os códigos, entender a linguagem e por fim fazer rodar o código, com todas essas opções mostradas nessa seção pode-se garantir o início de um aprendizado na linguagem de programação Python.

[10] Cadeia de caracteres: sequência de zero ou mais caracteres ASCII (do inglês ***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange*; "Código Padrão Americano para o Intercâmbio de Informação").

3 TUTORIAL SOBRE ARDUINO PARA INICIANTES

Nesta seção será apresentada a plataforma Arduino, sua descrição, as suas caraterísticas básicas, os dados técnicos, alimentação, memória, entrada e saída de dados, comunicação, os elementos de sua programação, as placas disponíveis, a placa que será utilizado no estudo de caso deste trabalho de conclusão de curso, e sua instalação.

Os aspectos abordados sobre Arduino foram principalmente, levantados usando como referências: a) Site oficial do Arduino (ARDUINO, 2018), b) Guia do Arduino (MULTILOGICA-SHOP, 2018).

O Arduino é uma plataforma eletrônica de código aberto baseada em *hardware* e *software* fáceis de usar. Todas as placas do Arduino são completamente *open source*, capacitando os usuários a construí-las independentemente e, eventualmente, adaptá-las às suas necessidades específicas. O *software* também é de código aberto e está crescendo com as contribuições dos usuários em todo o mundo (ARDUINO, 2018).

O *software* Arduino é fácil de usar para iniciantes, mas flexível o suficiente para usuários avançados. O Arduino é uma ferramenta fundamental para aprender coisas novas. Qualquer pessoa, crianças, amadores, artistas, programadores podem começar a aprender apenas seguindo as instruções passo a passo de um *kit* ou compartilhando ideias online com outros membros da comunidade Arduino (ARDUINO, 2018).

3.1 DESCRIÇÃO

O microcontrolador da placa Arduino é programado mediante a linguagem de programação Arduino, baseada em Wiring, e o ambiente de desenvolvimento (IDE) está baseado em Processing. Os projetos desenvolvidos com Arduino podem ser executados mesmo sem a necessidade de estar conectados a um computador, apesar de que também podem ser feitos comunicando-se com diferentes tipos de *software* (como Flash, Processing ou MaxMSP). As placas podem ser feitas à mão ou compradas montadas de fábrica.

O *download* do *software* pode ser feito de forma gratuita e os desenhos da placa estão disponíveis sob uma licença aberta, com isso toda a sociedade é livre para adaptá-lo às suas necessidades (ARDUINO, 2018).

3.2 CARACTERÍSTICAS

A figura 17 mostra, como é o Arduino Uno, com Largura de 68,58mm e comprimento 53,34 mm, respectivamente, 2,7" x 2,1", com os conectores *USB* e de alimentação (itens [A] e [B] na Figura 17) estendendo-se além destas dimensões. Quatro orifícios para parafusos permitem que a placa seja fixada a uma superfície ou encapsulamento.

Figura 17 - Plataforma Arduino Uno

Fonte: Arduino (2018). Disponível em: <https://store.arduino.cc/usa/arduino-uno-rev3>. Acesso em: 28 mai. 2018.

3.3 DADOS TÉCNICOS

O Arduino é uma placa com base no microcontrolador *ATmega168* ou *ATmega328*, na Tabela 4 são apresentadas suas características técnicas. Ele é composto por 14 pinos de entrada ou saída digital (item [C] na Figura 17), 6 desses pinos podem ser usados como saídas PWM (sigla de *Pulse With Modulation*), 6 entradas analógicas, um oscilador de cristal 16MHz, controlador USB (item [A] na Figura 17), uma tomada de alimentação (item [B] na Figura 17), um conector *ICSP*, e um botão de *reset*. Pode-se conectá-lo a um computador com um cabo USB (sigla de *Universal Serie Bus*), ligá-lo com um adaptador AC para DC ou até mesmo uma bateria, sendo essas as principais formas de utilização.

Tabela 4 - Tabela com as características técnicas do Arduino

Microcontrolador	**ATmega328 ou ATmega168**
Tensão operacional	5V
Tensão de alimentação (recomendada)	7-12V
Tensão de alimentação (limite)	6-20V
Pinos I/O digitais	14 (6 podem ser saídas PWM)
Pinos PWM I/O digitais	6
Pinos de entrada analógica	6
Corrente contínua por pino I/0	20 mA
Corrente contínua para o pino 3.3V	50 mA
Memória *flash*	32 KB (2KB usados para *bootloader*) /16KB
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Frequência de *clock*	16 MHz
Comprimento	68,6 mm
Largura	53,4 mm
Peso	25 g

Fonte: Arduino (2018). Disponível em: <https://store.arduino.cc/usa/arduino-uno-rev3>. Acesso em: 12 mai. 2018.

3.4 ALIMENTAÇÃO

O Arduino tem a opção de ser alimentado pela conexão USB (item [A] na Figura 17) ou por qualquer fonte de alimentação externa. A fonte de alimentação é selecionada automaticamente. A alimentação externa (não USB) pode ser tanto de uma fonte ou de uma bateria. A fonte pode ser conectada com um *plug* de 2,1 mm (centro positivo) no conector de alimentação. Cabos vindos de uma bateria podem ser inseridos nos pinos GND (terra) e Vin (entrada de tensão) do conector de alimentação. A placa pode operar com uma alimentação externa de 6 a 20V. Entretanto, se a alimentação for inferior a 7V o pino 5V pode fornecer menos de 5V e a placa pode ficar instável. Se a alimentação for superior a 12V o regulador de tensão pode superaquecer e avariar a placa. A alimentação recomendada é de 7 a 12V. Os pinos de alimentação são:

- *Vin*: entrada de alimentação para a placa Arduino quando uma fonte externa for utilizada. O fornecimento da alimentação pode ser utilizado por esse pino ou, se usar o conector de alimentação, acessar a alimentação por este pino.

- *5V*: fonte de alimentação utilizada para o microcontrolador e para outros componentes da placa. Pode ser proveniente do pino "*Vin*" por meio de um regulador *on-board* ou ser fornecida pelo USB ou outra fonte de 5V.
- *3V*: alimentação de 3,3V fornecida pelo circuito integrado FTDI (controlador USB). Corrente máxima é de 50 mA.
- *GND (ground)*: pino terra.

3.5 MEMÓRIA

O *ATmega328* tem 32 KB de memória *flash* (onde é armazenado o *software*), além de 2 KB de SRAM (onde ficam as variáveis) e 1 KB de EEPROM (esta última pode ser lida e escrita por meio da biblioteca EEPROM e guarda os dados permanentemente, mesmo que desliguemos a placa). A memória SRAM é apagada toda vez que se desliga o circuito.

3.6 ENTRADA E SAÍDA

O Arduino é composto por 14 pinos digitais (item [C] da Figura 17) e os mesmos podem ser usados como entrada ou saída usando as funções de *pinMode*(), *digitalWrite*(), e *digitalRead*(). São operados com 5V. O valor máximo que cada pino pode fornecer ou receber de corrente é de 40 mA, há um resistor *pull-up* interno (desconectado por padrão) de 20-50 kΩ.

Alguns pinos têm funções específicas, entre eles:

- *Serial*: 0 (RX) e 1 (TX). Usados para receber (RX) e transmitir (TX) dados seriais *TTL*. Estes pinos são conectados aos pinos correspondentes do chip serial *FTDI USB*-to-*TTL*.
- *PWM*: 3, 5, 6, 9, 10, e 11. Fornecem uma saída analógica PWM de 8-*bit* com a função *analogWrite().*
- *SPI*: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estes pinos suportam comunicação *SPI*, que embora compatível com o *hardware*, não está incluída na linguagem de programação do Arduino.
- *LED*: há um LED já montado e conectado ao pino digital 13. O Arduino tem 6 entradas analógicas, cada uma delas está ligada a um conversor analógico-digital de 10 *bits*, ou seja, transformam a leitura analógica em um valor dentre 1024

possibilidades (exemplo: de 0 a 1023). Por padrão, elas medem de 0 a 5V, embora seja possível mudar o limite superior usando o pino AREF e um pouco de código de baixo nível.

Adicionalmente alguns pinos têm funcionalidades especializadas, são eles:

- *I2C*: 4 (SDA) e 5 (SCL). Suportam comunicação I2C (TWI) usando a biblioteca *Wire*.
- *AREF*: referência de itens para entradas analógicas. Usados com *analogReference().*
- *Reset*: deve-se enviar o valor *LOW* (baixo) para reiniciar o microcontrolador. Utilizados para adicionar um botão de *reset* aos *Shields* (placas que podem ser plugadas ao Arduino para estender suas capacidades) que bloqueiam o que há na placa.

3.7 COMUNICAÇÃO

O Arduino possui dois tipos de portas: a) analógicas e b) digitais; sendo esta última dividida entre binárias e PWM (sigla de *Pulse Width Modulation*). Existe esta distinção de portas, que devem ser designadas de acordo com o resultado esperado com os componentes ligados a elas.

As portas digitais trabalham com apenas dois valores de tensão: 0V e 5V. É extremamente importante saber que os componentes ligados a estas portas só podem trabalhar com estas duas tensões, seja enviando ou recebendo dados.

As principais funções na linguagem de programação Arduino para entradas e saída nas portas digitais são:

a) *digitalRead(pino)*: lê o valor de um “pino” digital especificado, retornando um valor que pode ser *HIGH* ou *LOW* (Listagem 21).

A Listagem 21 apresenta um código Arduino com a função *digitalRead()* que liga e desliga um LED conforme um botão de pressão (*push-button*) é acionado ou não.

Listagem 21 - Exemplo de código Arduino com a função *digitalRead()*

```
int ledPin = 13; // LED conectado pino digital 13

int inPin = 7; // push-button[11] conectado ao pino digital 7

int val; // variável para armazenar o valor de leitura


void setup() {
// Configura pino digital 13 como saída
  pinMode(ledPin, OUTPUT);

// Configura pino digital 7 como entrada
  pinMode(inPin, INPUT);
}


// Liga (push-button pressionado) ou
// desliga (push-button "não" pressionado) o LED
void loop() {
// faz a leitura do pino de entrada
  val = digitalRead(inPin);

// configura o LED para o valor do botão:
// HIGH: configura o LED ligado
// LOW: configura o LED desligado
  digitalWrite(ledPin, val);
}
```

Fonte: FBS Eletrônica (2018).

b) *digitalWrite(pino, valor)*: aciona um “valor”, *HIGH* ou *LOW*, em um “pino” digital (Listagem 22).

c) *pinMode(pino, modo)*: configura um “pino” específico para se entrada ou saída digital (parâmetro “modo”) (Listagem 22). Onde “modo” pode ser *INPUT*, *OUTPUT* ou *INPUT_PULLUP*; que correspondem respectivamente a entrada, saída e entrada com *pull-up* ativado.

A Listagem 22 apresenta um código Arduino com as funções *digitalWrite()* e *pinMode()* que liga: *digitalWrite(ledPin, HIGH)*; e desliga: *digitalWrite(ledPin, LOW)*; um LED em intervalos de 1 segundo: *delay(1000)*.

[11] “push-button” (botão de pressão) é um botão/pulsador utilizado comumente para dar ordem de acionamento.

Listagem 22 - Exemplo de código Arduino com as funções *digitalWrite()* e *pinMode()*

```
int ledPin = 13; // LED conectado ao pino digital 13

void setup() {
// configura o pino digital como saída
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH); // configura o LED ligado
  delay(1000); // espera por 1 segundo

  digitalWrite(ledPin, LOW); // configura o LED desligado
  delay(1000); // espera por 1 segundo
}
```

Fonte: FBS Eletrônica (2018).

As portas PWM (do inglês *Pulse Width Modulation*, ou Modulação por Largura de Pulso) se diferenciam das portas digitais binárias pois podem trabalhar não apenas com as tensões 0V e 5V, com uma escala que vai de 0 a 255 entre essas tensões, onde o 0 quer dizer 0V e 255 quer dizer 5V. As portas PWM permitem obter resultados analógicos com meios digitais e são capazes de controlar a potência de saída de um sinal. Pode se controlar, por exemplo, a potência em um LED, permitindo aumentar ou diminuir sua intensidade luminosa.

Portas Analógicas são utilizadas para entrada de dados. Diferentes das portas digitais, permitem não apenas ler os valores 0V e 5V, mas qualquer valor entre eles dentro de uma escala de 0 a 1023, onde o 0 representa 0V e o 1023 representa 5V. Isso porque os conversores ADC (signal de *Analog Digital Converter*) são de 10 *bit* (2^{10}: 1024 valores) e fornecem uma precisão de 0.005V ou 5mV. Essas portas são utilizadas, por exemplo, para ler os valores de um sensor.

3.7.1 Comunicação Serial

A comunicação serial é utilizada para comunicação entre uma placa Arduino e um computador ou outros dispositivos, possibilitando o envio de mensagens entre ambos. As mensagens podem ser enviadas por meio do teclado, ou até mesmo de algum programa instalado no computador, permitindo monitorar e controlar uma aplicação do Arduino.

A comunicação neste caso pode ser estabelecida com a utilização de métodos da classe “Serial” que encapsulam algumas funcionalidades para a comunicação serial, entre eles tem-se:

a. *begin(speed)*: configura a taxa de transferência em *bits* por segundo (parâmetro “speed”) para transmissão serial. A velocidade padrão para a placa *Arduino* Uno, utilizada nos estudos de casos deste trabalho de conclusão de curso, é igual a 9600.
b. *end()*: desativa a comunicação serial, permitindo que uma nova comunicação serial possa ser reestabelecida.
c. *available()*: retorna o número de *bytes* (caracteres) disponíveis para leitura da porta serial.
d. *parseFloat()*: retorna o próximo número válido de ponto flutuante (*float*) do *buffer* serial.
e. *parseInt()*: retorna o próximo inteiro válido no *buffer* de recebimento serial.
f. *print()*: imprime dados na porta serial. Essa função pode assumir várias formas: a) números são impressos usando um caractere *ASCII* para cada dígito; b) *floats* são similarmente impressos como dígitos *ASCII*, padronizados com duas casas decimais; c) *bytes* são enviados como um único caractere; d) caracteres e *strings* são enviados normalmente, pois já estão em formato *ASCII*. Por exemplo:

```
Serial.print(78)                imprime "78"
Serial.print(1.23456)           imprime "1.23"
Serial.print('N')               imprime "N"
Serial.print("Hello world.")    imprime "Hello world."
```

g. *println()*: imprime dados na porta serial como texto *ASCII* seguido pelo caracter de retorno do carro (*Enter*, *ASCII* 13, ou '\r') e um caracter de nova linha (*ASCII* 10, ou '\n').
h. *read()*: lê dados recebidos na porta serial.
i. *write(val)*: escreve dados binários na porta serial. Esses dados são enviados como um *byte* ou séries de *bytes*; para enviar os caracteres representando os dígitos de um número, por exemplo, deve-se utilizar a função “*print*()”.

A Listagem 23, apresenta a implementação do código Arduino que estabelece uma comunicação serial para ler um número inteiro da porta serial para verificar e imprimir uma mensagem indicando se o valor lido é um número “par” ou “ímpar”.

Listagem 23 - Exemplo de código Arduino para estabelecer uma comunicação serial: classe "Serial"

```
int n;

void setup(){
  Serial.begin(9600);
  Serial.println("Informe um número inteiro: ");
}

void loop() {
  if (Serial.available() > 0) {
     n = Serial.read();

     if ((n % 2) == 0) // verifica se "n" é par
        Serial.println("Par");
     else Serial.println("Ímpar");

     Serial.end(); // encerra a comunicação
  }
}
```

Fonte: Autoria própria.

3.8 PROGRAMAÇÃO

No ambiente do Arduino para implementar programas utiliza-se uma linguagem própria de referência que tem como base a linguagem de programação *C++.* Mantendo a sintaxe clássica da linguagem de base, como por exemplo, na declaração de variáveis, tipos de operadores, estrutura de dados vetores, definição e utilização de ponteiros, estruturas de controle e em muitas outras características da linguagem.

A linguagem de programação Arduino pode ser dividida em três partes principais (ARDUINO PROGRAMMING LANGUAGE, 2018): a) estruturas de controle, b) elementos básicos da programação: valores (variáveis e constantes), e c) funções.

3.8.1 Estruturas de Controle

As estruturas de controle do fluxo de execução na linguagem de programação Arduino seguem o padrão de outras linguagens de programação, tais como as linguagens: C, C++, Java, entre outras. As referências teóricas a essas estruturas de controle, são elas: comandos condicionais e comandos repetitivos, podem ser encontradas na documentação da linguagem de programação Arduino (ARDUINO PROGRAMMING LANGUAGE, 2018).

3.8.1.1 Comando condicional “if”

O comando “*if*”[12] verifica uma <expressão lógica> e executa o comando a seguir ou um bloco de comandos delimitados por chaves ({}), se o resultado da verificação apresentar um resultado verdadeiro (*true*).

- Sintaxe:

```
if (<expressão lógica>) {
   // comando ou comandos
}
```

- Exemplo:

```
if (Serial.available() > 0) {
   leitura = Serial.read();
   if (leitura == 'L') {
      // Liga a porta 13 se o valor recebido for 'L'
      digitalWrite(led1, HIGH);
      Serial.println("LED ligado com sucesso.");
   }
   if (leitura == 'D') {
      // Desliga a porta 13 se o valor recebido for 'D'
      digitalWrite(led1, LOW);
      Serial.println("LED desligado com sucesso.");
   }
   if (leitura == 'F') {
      Serial.end(); // Fim da conexão
   }
}
```

3.8.1.2 Comando condicional “*if-else*”

No comando condicional “*if-else*”[13], é permitido múltiplas verificações agrupada. Uma cláusula “*else*” (se presente) será executada se a <expressão lógica> apresentar um resultado verdadeiro. Um “*else*” pode anteceder outro teste “*if*”, tal que múltiplos, testes mutualmente exclusivos podem ser executados ao mesmo tempo.

[12] Fonte: Arduino Programming Language (2018). Referência sobre o comando condicional “if”. Disponível em: <https://www.arduino.cc/reference/pt/language/structure/control-structure/if/>. Acesso em: 15 jun. 2018.
[13] Fonte: Arduino Programming Language (2018). Referência sobre o comando condicional “if-else”. Disponível em: <https://www.arduino.cc/reference/pt/language/structure/control-structure/else/>. Acesso em: 15 jun. 2018.

- Sintaxe:

```
if (<expressão lógica 1>) {
   // comando ou comandos A
}
else if (<expressão lógica 2>) {
        // comando ou comandos B
      }
      else {
        // comando ou comandos C
      }
```

- Exemplo:

```
if (Serial.available() > 0) {
   leitura = Serial.read();
   if (leitura == 'L') {
      // Liga a porta 13 se o valor recebido for 'L'
      digitalWrite(led1, HIGH);
      Serial.println("LED ligado com sucesso.");
   }
   else if (leitura == 'D') {
           // Desliga a porta 13 se o valor recebido for 'D'
           digitalWrite(led1, LOW);
           Serial.println("LED desligado com sucesso.");
        }
        else if (leitura == 'F') {
                Serial.end(); // Fim da conexão
             }
}
```

3.8.1.3 Comando de repetição "*while*"

Um comando de repetição "*while*"[14] define um fluxo de execução que vai se repetir continuamente, e infinitamente, até que a <expressão lógica> dentro dos parênteses (), se torne falsa (*false*). Alguma situação deve ocorrer para mudar o valor da variável testada, ou o *loop* "*while*" nunca irá encerrar. Isso pode ser no seu código, por exemplo, uma variável incrementada, ou uma condição externa, como a leitura de um sensor.

[14] Fonte: Arduino Programming Language (2018). Referência sobre o comando de repetição "while". Disponível em: <https://www.arduino.cc/reference/pt/language/structure/control-structure/while/>. Acesso em: 15 jun. 2018.

- Sintaxe:

```
while (<expressão lógica>) {
  // comando ou comandos que serão executados repetidamente
}
```

- Exemplo:

```
int i; // variável de controle

i = 0; // início da repetição (start)

while (i < 10) { // fim da repetição (stop)
  // faz algo de forma repetitiva 10 vezes

  i = i + 1; // passo (step)
}
```

3.8.1.4 Comando de repetição "*for*"

O comando de repetição "*for*"[15] é usado para repetir um bloco de código envolvido por chaves um determinado número de vezes (laço contado). Um contador de incremento (*i*++) é geralmente utilizado para terminar o *loop*.

- Sintaxe:

```
for(<inicialização>; <expressão lógica>; <incremento>) {
  // comando ou comandos que serão executados repetidamente
}
```

- Exemplo:

```
int i; // variável de controle

for(i=0; i<10; i++) {
  // faz algo de forma repetitiva 10 vezes
}
```

3.8.2 Elementos Básicos da Programação

Os elementos básicos da programação na linguagem de programação Arduino se referem aos tipos de dados básicos e aos operadores: aritméticos, relacionais, lógicos, para ponteiros e *bitwise*. As referências teóricas a esses elementos podem

[15] Fonte: Arduino Programming Language (2018). Referência sobre o comando de repetição "for". Disponível em: <https://www.arduino.cc/reference/pt/language/structure/control-structure/for/>. Acesso em: 15 jun. 2018.

ser encontradas na documentação da linguagem de programação Arduino (ARDUINO PROGRAMMING LANGUAGE, 2018).

3.8.2.1 Tipos de dados básicos

Na programação de computadores os tipos de dados (Tabela 5) definem que valores uma variável pode assumir em memória e também os tipos de operações, por exemplo: operações aritméticas, operações relacionais, operações lógicas, entre outras; que podem ser realizadas com a variável.

Tabela 5 - Tipos de dados básicos na linguagem de programação Arduino

(continua)

Tipo	Descrição	Exemplos
bool	com o tipo bool pode-se armazenar dois valores booleanos: *true* ou *false*.	bool running = true; bool ligado = false;
char	tipo de dado que ocupa 1 *byte* na memória e possibilita assim o armazenamento de um caractere. Caracteres literais delimitados por aspas simples, por exemplo: 'A', ‘a’, ‘0’, ‘9’, ‘%’, ‘&’, ‘?’ e ‘+’.	char liga = 'L'; char desliga = ‘D’; char fim = ‘F’;

float	tipo de dado para números de ponto flutuante (número reais). floats são armazenados em 32 *bits* (4 *bytes*) de memória definindo um intervalo de valores de -3.4028235E+38 a 3.4028235E+38.	int x = 5;
		int y = 2;
		float z;
		
		z = x / (float)y;

double	para números de ponto flutuante e em geral definem um espaço de memória de 8 *bytes*.	int x = 5;
		int y = 2;
		double z;
		
		z = x / (double)y;

byte	armazena valores numéricos de 1 *byte* (2^8 = 256) sem sinal, de 0 a 255.	byte mes = 12;
int	(*integer* ou inteiros) são o tipo de dados primário para armazenamento de números. Em geral utilizam 2 *bytes* para armazenamento definindo valores de -32.768 a 32.767.	int ledPin = 13; int analogPin = 3;

Tabela 5 - Tipos de dados básicos na linguagem de programação Arduino

(conclusão)

Tipo	Descrição	Exemplos
unsigned int	(inteiros sem sinal) são o mesmo que o tipo int armazenando apenas valores positivos de 0 a 65.535.	unsigned int ledPin = 13;
word	uma variável declarada do tipo word armazena um número inteiro sem sinal de 4 *bytes*, de 0 a 65535. O mesmo que uma variável unsigned int.	word ledPin = 13;
long	define variáveis de tamanho extendido para armazenamento de números, armazenam 32 *bits* (4 bytes), de -2.147.483.648 a 2.147.483.647.	long speedOfLight; speedOfLight = 186000L;
unsigned long	são o mesmo que o tipo long armazenando apenas valores positivos de de 0 a 4.294.967.295.	unsigned long time; time = millis();

void	A palavra chave[16] void é usada apenas em declarações de funções. Ela indica que é esperado que a função "não" retorne nenhuma informação para a função da qual foi chamada.	void setup() {
		  // ...
		}
		
		void loop() {
		  // ...
		}

Fonte: Autoria própria.

3.8.2.2 Operadores aritméticos

Na linguagem de programação *Arduino* os operadores aritméticos, apresentados na Tabela 6, realizam operações matemáticas elementares, como por exemplo: soma (+), subtração (-), multiplicação (*), divisão inteira ou real (/) e extração do módulo ou resto da divisão (%).

Tabela 6 - Tabela dos operadores aritméticos na linguagem de programação Arduino

Operação	Operador
Adição	**+**
Subtração	**-**
Multiplicação	*****
Divisão real	**/**
Resto da divisão (módulo)	**%**
Exponenciação	**
Operador de atribuição	**=**

Fonte: Autoria própria.

[16] Palavras chave são palavras reservadas para uso restrito pela linguagem de programação.

3.8.2.3 Operadores relacionais

Os operadores relacionais, apresentados na Tabela 7, são utilizados para relacionar valores e apresentando como resultado um valor do tipo lógico, podendo ser: *true* para indicar verdadeiro (afirmação) ou, *false* no caso do resultado falso (negação).

Tabela 7 - Tabela dos operadores relacionais na linguagem de programação Arduino

Operador	Operação	Símbolo Matemático
==	Igual a	=
>	Maior que	>
<	Menor que	<
!=	Diferente de	≠
>=	Maior que ou igual a	≥
<=	Menor que ou igual a	≤

Fonte: Autoria própria.

3.8.2.4 Operadores lógicos

A linguagem de programação *Arduino*, assim como na linguagem de programação *Python*, suporta três operadores lógicos básicos, são eles: *not* (não, negação), && (e lógico), || (ou lógico).

3.8.2.5 Operadores para ponteiros

Os ponteiros são tipo de dados que representam endereços de memória. Diz-se que um ponteiro “aponta” para uma variável quando contém o endereço da mesma.

Na linguagem de programação *Arduino*, assim como no linguagem de programação *C*, disponibiliza dois operadores para ponteiros: a) & (referência): se “x” é uma variável, então &x representa o endereço da variável “x”; b) * (de referência): se “p” é um ponteiro, então *p representa o valor contido no endereço apontado por “p”.

3.8.2.6 Operadores *bitwise*

Os operadores *bitwise*, apresentados na Tabela 8, são utilizados quando precisa-se realizar operações a nível de *bits* com números inteiros, trabalhando assim com a respectiva representação binária do número**.**

Tabela 8 - Tabela dos operadores bitwise na linguagem de programação Arduino

Operador	Operação	Exemplos
&	O operador & *bit-a-bit* atua em cada *bit* dos operandos independentemente, de acordo com a seguinte regra: se ambas entradas são 1, a saída resultante é 1, do contrário o resultado é 0.	0 0 1 1 var1 0 1 0 1 var2 --------- 0 0 0 1 (var1 & var2)
<<	O operador << (deslocamento à esquerda) faz os *bits* do operando à esquerda do operador serem deslocados à esquerda pelo número de posições especificadas pelo operando à direita do operador. Sintaxe: *variável << numero_de_bits;*	int a = 5; *// em binário (2 bytes): 000000000000101* int b = a << 3; *// em binário: 0000000000101000, ou 40 em decimal*
>>	O operador >> (deslocamento à direita) faz os *bits* do operando à esquerda do operador serem deslocados à direita pelo número de posições especificadas pelo operando à direita do operador. Sintaxe: *variável >> numero_de_bits;*	int a = 40; *// em binário: 000000000101000* int b = a >> 3; *// em binário: 0000000000000101, ou 5 em decimal*
^	Uma operação ^ (ou exclusivo) resulta em 1 apenas se os bits de entrada são diferentes. Se iguais, o resultado é zero.	0 0 1 1 var1 0 1 0 1 var2 --------- 0 1 1 0 (var1 ^ var2)
|	O resultado da operação | (ou) entre dois *bits* é 1 se qualquer um ou ambos os bits de entrada são 1, do contrário é 0.	0 0 1 1 var1 0 1 0 1 var2 --------- 0 1 1 1 (var1 | var2)
~	O operador ~ (não) muda cada *bit* para seu valor oposto: 0 se torna 1, e 1 se torna 0.	0 1 var1 ----- 1 0 ~var1

Fonte: Autoria própria.

3.8.3 Funções

As funções, implementadas e disponíveis em bibliotecas, são referências essenciais para controlar a placa *Arduino* e realizar cálculos. A documentação de referência da linguagem de programação *Arduino* (ARDUINO PROGRAMMING LANGUAGE, 2018) apresenta as funções em categorias, entre elas tem-se:

A. Funções Temporizadoras

a. *delay(ms)*: provoca uma pausa na execução do programa por uma quantidade especificada de tempo (em milissegundos: valor do tipo de dados "*unsigned long*", parâmetro "ms"). Cada segundo equivale a 1000 milissegundos. Está função não retorna valor como resposta a sua execução. Um exemplo de código *Arduino*

demonstrando a implementação da função “*delay*” pode ser observado na Listagem 22.

b. *delayMicroseconds(us)*: provoca uma pausa na execução do programa por uma quantidade especificada de tempo (em microssegundos: valor do tipo de dados “*unsigned int*”, parâmetro “*us*”). Há mil microssegundos em um milissegundo, e um milhão de microssegundos em um segundo. Está função não retorna valor como resposta a sua execução.
c. *micros()*: retorna o número de microssegundos (como um valor do tipo de dados “*unsigned long*”) passados desde que a placa *Arduino* começou a executar o programa atual.
d. *millis()*: retorna o número de milissegundos (como um valor do tipo de dados “*unsigned long*”) passados desde que a placa *Arduino* começou a executar o programa atual.

B. *Bit* e *bytes*

O *bit*, sigla para *binary digit*, na informática é a menor unidade de informação que pode ser armazenada ou transmitida. Um *bit* pode assumir somente 2 valores, como 0 ou 1. Por sua vez, um *byte* é uma sequência de 8 *bits* e portanto pode ter 2^8, ou seja, 256 valores: de 00000000 a 11111111. A seguir tem-se as funções “*bit*” e “*bytes*” na linguagem de programação *Arduino*:

a. *bit(n)*: retorna o valor do *bit* especificado (o *bit* 0 é igual a 1, *bit* 1 igual a 2, *bit* 2 igual a 4, *bit* 3 igual a 8, ou seja, 2 elevado ao *bit*: 2^n).
b. *bitClear(x, n)*: limpa (escreve um 0) em um *bit* (parâmetro “n”) de uma variável numérica (parâmetro “x”). Está função não retorna valor como resposta a sua execução.
c. *bitRead(x, n)*: retorna o “n”-ésimo *bit* do número armazenado no parâmetro “x”.
d. *bitSet(x, n)*: ativa (escreve um 1) em um *bit* (parâmetro “n”) de uma variável numérica (parâmetro “x”). Está função não retorna valor como resposta a sua execução.
e. *bitWrite(x, n, b)*: escreve o *bit* “b” (0 ou 1) em um *bit* especificado (parâmetro “n”) de um valor numérico (parâmetro “x”). Está função não retorna valor como resposta a sua execução.
f. *highByte(x)*: retorna o *byte* mais significativo do valor do parâmetro “x”.
g. *lowByte(x)*: retorna o *byte* menos significativo do valor do parâmetro “x”.

C. Entradas e Saídas Analógicas

a. *analogRead(pino)*: lê o valor de um "pino" analógico especificado. A placa *Arduino* possui um conversor analógico-digital de 6 canais (8 canais nos Mini e Nano, 16 no *Mega*). Isso significa que este irá mapear tensões entre 0 e 5 volts para valores inteiros entre 0 e 1023. Retorna um valor do tipo de dados "int" no intervalo de 0 até 1023.

A Listagem 24 apresenta um código Arduino que estabelece uma comunicação serial lendo o valor do pino analógico de entrada: *val = analogRead(analogPin)*; e imprimindo na porta serial o valor (*val*) na porta serial: *Serial.println(val)*.

Listagem 24 - Exemplo de código Arduino com a função "analogRead()"

```
// terminal do meio de um potênciometro[17]
// conectado ao pino analógico 3
int analogPin = 3;
int val; // variável de armazenamento do valor lido

void setup() {
  Serial.begin(9600); // configura a porta serial
}

void loop() {
  val = analogRead(analogPin); // lê pino de entrada
  Serial.println(val); // imprime o valor na porta serial
}
```

Fonte: Arduino Programming Language (2018). Disponível em: <https://www.arduino.cc/reference/pt/language/functions/analog-io/analogread/>. Acesso em: 09 jun. 2018.

b. *analogReference(tipo)*: configura a tensão de referência para a entrada analógica (o valor máximo do intervalo de entrada).

c. *analogWrite(pino, valor)*: aciona uma onda *PWM* (*Pulse Width Modulation*) em um "pino". Pode ser usada para variar o brilho de um *LED*, por exemplo, ou acionar um motor a diversas velocidades.

D. Funções Matemáticas

a. *abs(x)*: calcula o módulo (ou valor absoluto) de um número (parâmetro "x"). Retorna "x" se "x" é maior ou igual a 0 (zero), caso contrário, retorna "-x".

[17] Potenciômeto é um componente eletrônico que cria uma limitação para o fluxo de corrente elétrica que passa por ele.

b. *constrain(x, a, b)*: restringe um número (parâmetro “x”) a ficar dentro do intervalo especificado pelos parâmetros “a” e “b”. Retorna “x” se “x” pertence ao intervalo de “a” até “b”. Retorna “a” se “x” é menor do que “a”. Retorna “b” se “x” é maior do que “b”.
c. *map(value, fromLow, fromHigh, toLow, toHigh)*: remapeia um número (parâmetro “*value*”) de um intervalo (parâmetros “*fromLow*” e “*fromHigh*”) para outro (parâmetros “*toLow*” e “*toHigh*”).
d. *max(x, y)*: retorna o maior valor entre os valores dos parâmetros “x” e “y”.
e. *min(x, y)*: retorna o menor valor entre os valores dos parâmetros “x” e “y”.
f. *pow(base, exponente)*: retorna o valor de um número (parâmetro “base”) elevado a uma potência (parâmetro “exponente”).
g. *sq(x)*: retorna o quadrado (*square*) de um número (parâmetro “x”).
h. *sqrt(x)*: retorna a raiz quadrada (*square root*) de um número (parâmetro “x”).

E. Funções Trigonométricas

a. *cos(rad)*: retorna o cosseno de um ângulo (em radianos: valor “*float*”, parâmetro “*rad*”). O resultado será entre -1 e 1.
b. *sin(rad)*: retorna o seno de um ângulo (em radianos: valor do tipo de dados “*float*”, parâmetro “*rad*”). O resultado será entre -1 e 1.
c. *tan(rad)*: retorna a tangente de um ângulo (em radianos: valor do tipo de dados “*double*”, parâmetro “*tan*”). O resultado estará entre -infinito e +infinito (limitado pelo tamanho do tipo de dados “*double*”).

F. Caracteres

a. *isAlpha(thisChar)*: retorna *true* (verdadaeiro) se “*thisChar*” é um caractere alfabético e *false* (falso) em caso contrário.
b. *isAlphaNumeric(thisChar)*: retorna *true* se “*thisChar*” é um caractere alfabético ou um número e *false* em caso contrário.
c. *isAscii(thisChar)*: retorna *true* se “*thisChar*” é um caractere *ASCII* (sigla de *American Standard Code for Information Interchange*) e *false* em caso contrário.
d. *isControl(thisChar)*: retorna *true* se “*thisChar*” é um caractere de controle[18] e *false* em caso contrário.

[18] Os caracteres de controle são caracteres ASCII não-convencionais. Esses "caracteres" são indicados por meio de uma barra invertida seguida de um dígito ou letra. Por exemplo:
\0 caractere nulo (null)

e. *isDigit(thisChar)*: retorna *true* se "*thisChar*" é um dígito (ou número) e *false* em caso contrário.
f. *isGraph(thisChar)*: retorna *true* se "*thisChar*" é um caractere imprimível com algum conteúdo e *false* em caso contrário.
g. *isHexadecimalDigit(thisChar)*: retorna *true* se "*thisChar*" é um digito hexadecimal (A-F, 0-9) e *false* em caso contrário.
h. *isLowerCase(thisChar)*: retorna *true* se "*thisChar*" é um caractere alfabético minúsculo e *false* em caso contrário.
i. *isPrintable(thisChar)*: retorna *true* se "*thisChar*" é um caractere imprimível (isto é, qualquer caractere que produz uma saída, como letras e números, até mesmo um espaço) e *false* em caso contrário.
j. *isPunct(thisChar)*: retorna *true* se "*thisChar*" é um caractere de pontuação (isto é, uma vírgula, um ponto-e-vírgula, um ponto de exclamação, entre outros) e *false* em caso contrário.
k. *isSpace(thisChar)*: retorna *true* se "*thisChar*" é um caractere de espaço e *false* em caso contrário.
l. *isUpperCase(thisChar)*: retorna *true* se "*thisChar*" é um caractere alfabético maiúsculo e *false* em caso contrário.
m. *isWhitespace()*: retorna *true* se "*thisChar*" é um espaço em branco (isto é, o próprio caractere de espaço (' '), *formfeed* ('\f'), nova linha ('\n'), retorno ('\r'), *tab* horizontal *tab* ('\t') e *tab* vertical ('\v')) e *false* em caso contrário.

G. Números Aleatórios

a. *random(max)*: retorna um número aleatório entre 0 e max-1.
b. *random(min, max)*: retorna um número aleatório entre min e max-1.

A Listagem 25 apresenta um código Arduino implementando as duas versões da função "*random* ()". Os números aleatórios gerados pelas sentenças: a) *randNumber = random (1000);* e b) *randNumber = random (15, 26)*; são mostrados na porta serial por meio do comando: *Serial.println(randNumber)*.

\t tabulação horizontal (tab)
\n fim de linha (newline)
\r carriage return

Listagem 25 - Exemplo de código Arduino com a função "random ()"

```
long randNumber;
int i = 1;

void setup(){
  Serial.begin(9600);
}

void loop() {
  // imprime um número aleatório entre 0 e 999
  randNumber = random(1000);
  Serial.print(i);
  Serial.print(": ");
  Serial.println(randNumber);

  // imprime um segundo valor aleatório entre 15 e 25
  randNumber = random(15, 26);
  Serial.print(i);
  Serial.print(": ");
  Serial.println(randNumber);

  Serial.println("------------------");

  i++; // numeração sequencial dos números gerados

  delay(1000);
}
```

Fonte: Autoria própria.

H. Funções de Conversão

a. *byte(x)*: retorna a conversão de um valor (parâmetro "x") para o valor correspondente no tipo de dados "*byte*".

b. *char(x)*: retorna a conversão de um valor (parâmetro "x") para o valor correspondente no tipo de dados "*char*".

c. *float(x)*: retorna a conversão de um valor (parâmetro "x") para o valor correspondente no tipo de dado "float".

d. *int(x)*: retorna a conversão de um valor (parâmetro "x") para o valor correspondente no tipo de dado "*int*".

e. *long(x)*: retorna a conversão de um valor (parâmetro "x") para o valor correspondente no tipo de dado "*long*".

f. *word(x)*: retorna a conversão de um valor (parâmetro "x") para o valor correspondente no tipo de dado "*word*".

3.8.4 Bibliotecas

A utilização de bibliotecas disponíveis no Arduino abre um leque de programação, extremamente maior do que apenas utilizar as estruturas, valores e funções. Porém para a utilização de qualquer uma das bibliotecas, ela deverá estar previamente instalada na máquina de desenvolvimento. Entre as bibliotecas de referência tem-se[19]:

- *EEPROM*: leitura e escrita de “armazenamento” permanente.
- *Ethernet*: conexão a uma rede *Ethernet* usando o *Arduino Ethernet Shield*.
- *Firmata*: para se comunicar com os aplicativos no computador usando o protocolo *Firmata*.
- *LiquidCrystal*: controle de telas de cristal líquido (*LCD*).
- *Servo*: controle de servo motores.
- *SPI*: comunicação com dispositivos que utilizam barramento *Serial Peripheral*.
- *SoftwareSerial*: comunicação serial em qualquer um dos pinos digitais.
- *Stepper*: controle de motores de passo.
- *Wire*: dois *Wire Interface* (*TWI*/*I2C*) envio e recebimento de dados por meio de uma rede de dispositivos ou sensores.

No Arduino também existem disponíveis bibliotecas específicas, que são muito importantes para apoiar o desenvolvimento, entre elas tem-se:

- Comunicação (redes e protocolos):
  - *Messenger*: processamento de mensagens de texto a partir do computador.
  - *NewSoftSerial*: versão melhorada da biblioteca *SoftwareSerial*.
  - *OneWire*: dispositivos de controle que usam o protocolo *One Wire*.
  - *PS2Keyboard*: ler caracteres de um *PS2* teclado.
  - *Simple Message System*: envio de mensagens entre *Arduino* e a máquina.
  - *SSerial2Mobile*: enviar mensagens de texto ou *e-mails* usando um telefone celular.
  - *Webduino*: biblioteca que cria um servidor *web* (para uso com o *Arduino Ethernet Shield*).
  - *X10*: envio de sinais X10 nas linhas de energia *AC*.

[19] Fonte: Standard Libraries. Disponível em: <https://www.arduino.cc/en/Reference/Libraries>. Acesso em: 09 jun. 2018.

  - *XBee*: comunicação via protocolo XBee.
  - *SerialControl*: controle remoto por meio de uma conexão serial.
- Sensoriamento:
  - *Capacitive Sensing*: transformar dois ou mais pinos em sensores capacitivos.
  - *Debounce*: leitura de ruídos na entrada digital.
- Geração de frequência e de áudio:
  - *Tone*: gerar ondas quadradas de frequência de áudio em qualquer pino do microcontrolador.
- Temporização:
  - *DateTime*: biblioteca de atualização da data e hora atuais do *software*.
  - *Metro*: acionamento de tempo em intervalos regulares.
  - *MsTimer2*: utiliza o temporizador para desencadear uma ação a cada **N** milissegundos.
- Utilidades:
  - *TextString (String)*: manipulação de *strings.*
  - *PString*: classe leve para imprimir em *buffers*.
  - *Streaming*: um método para simplificar as declarações de impressão.

3.9 PLACAS ARDUINO

O Arduino tem uma gama enorme de placas, módulos, dependendo qual será a sua utilização pode-se escolher a placa que se adeque ao que será desenvolvido.

3.9.1 Níveis das Placas

Existem vários níveis, são eles: nível de entrada, recursos avançados, *internet* das coisas (*Internet of Things*), educação e *wearable*.

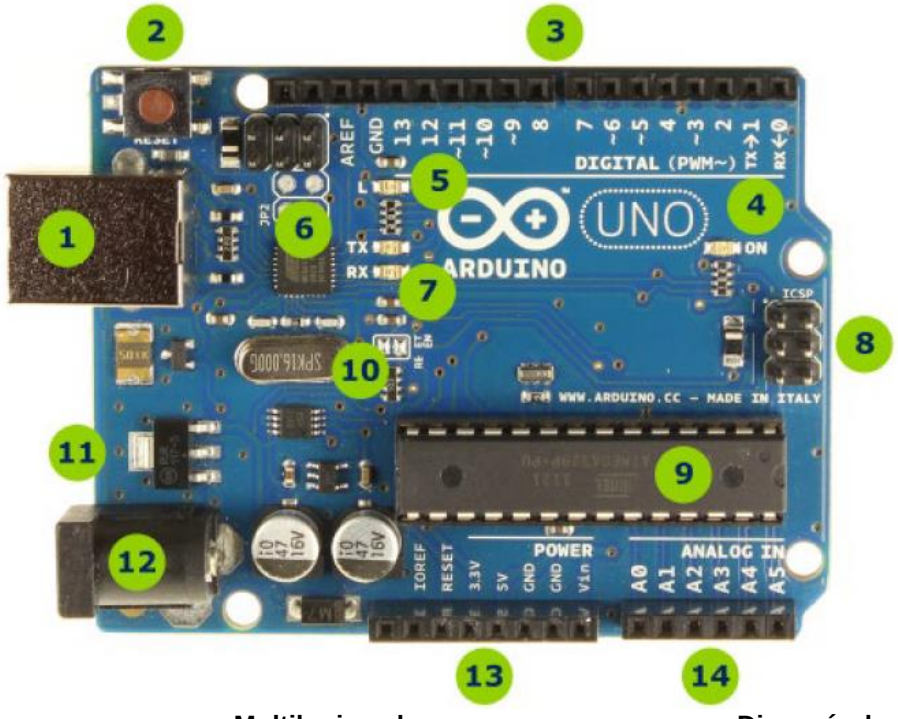
Nível de entrada é um nível iniciante, são produtos fáceis e os melhores módulos para aprender e mexer com eletrônicos e codificação. Nível de recursos aprimorados são para projetos mais complexos com funcionalidades mais avançadas ou para performance mais rápida. Nível *internet* das coisas são para fazer dispositivos conectados a *web*. Nível educação tem como principal objetivo capacitar os educadores com as ferramentas de *hardware* e *software* para criar experiências de aprendizagem mais prática e inovadora. Nível *Weareble* os desenvolvedores podem adicionar inteligência aos projetos.

Cada nível de aprendizado várias versões do Arduino, apenas no nível de entrada estão disponíveis oito versões diferentes. As placas disponíveis no nível de entrada são: Arduino Uno, Arduino Leonardo, Arduino 101, Arduino Esplora, Arduino Micro, Arduino Nano, Mzr2uno Adapter e Arduino Starter Kit.

3.9.2 Placa Utilizada no Trabalho de Conclusão de Curso

A placa Arduino Uno (Figura 18) é uma das principais placas para iniciantes com eletrônica e codificação. Uma placa robusta, e a mais utilizada de toda a família Arduino. Arduino Uno é um microcontrolador baseada no *ATmega328P*. Possui 14 pinos de entrada/saída digital (dos quais 6 podem ser usados como saídas PWM), 6 entradas analógicas, um cristal de quartzo de 16 MHz, uma conexão USB, um conector de energia, um conector ICSP e um botão de *reset*. Ele contém tudo o que é necessário para suportar o microcontrolador, basta conectá-lo a um computador com um cabo USB ou ligá-lo com um adaptador de CA-CC ou bateria para começar.

Figura 18 - Componentes da placa Arduino Uno



Fonte: Multilogica-shop.com. Disponível em: <https://multilogicashop.com/download_guia_arduino>. Acesso em: 28 mai. 2018.

Na Figura 18, estão enumerados os principais componentes e conectores da placa Arduino Uno, são eles:

1. Conector *USB* para o cabo tipo AB.
2. Botão de reset.

3. Pinos de entrada e saída digital e *PWM*.
4. *LED* verde de placa ligada.
5. *LED* laranja conectado ao pino 13.
6. *ATmega* encarregado da comunicação com o computador.
7. *LED TX* (transmissor) e *RX* (receptor) da comunicação serial.
8. Porta *ICSP* para programação serial.
9. Microcontrolador *ATmega* 328, cérebro do *Arduino*.
10. Cristal de quartzo 16Mhz.
11. Regulador de tensão.
12. Conector fêmea 2,1mm com centro positivo.
13. Pinos de tensão e terra.
14. Entradas analógicas.

3.10 PROCESSO DE INSTALAÇÃO

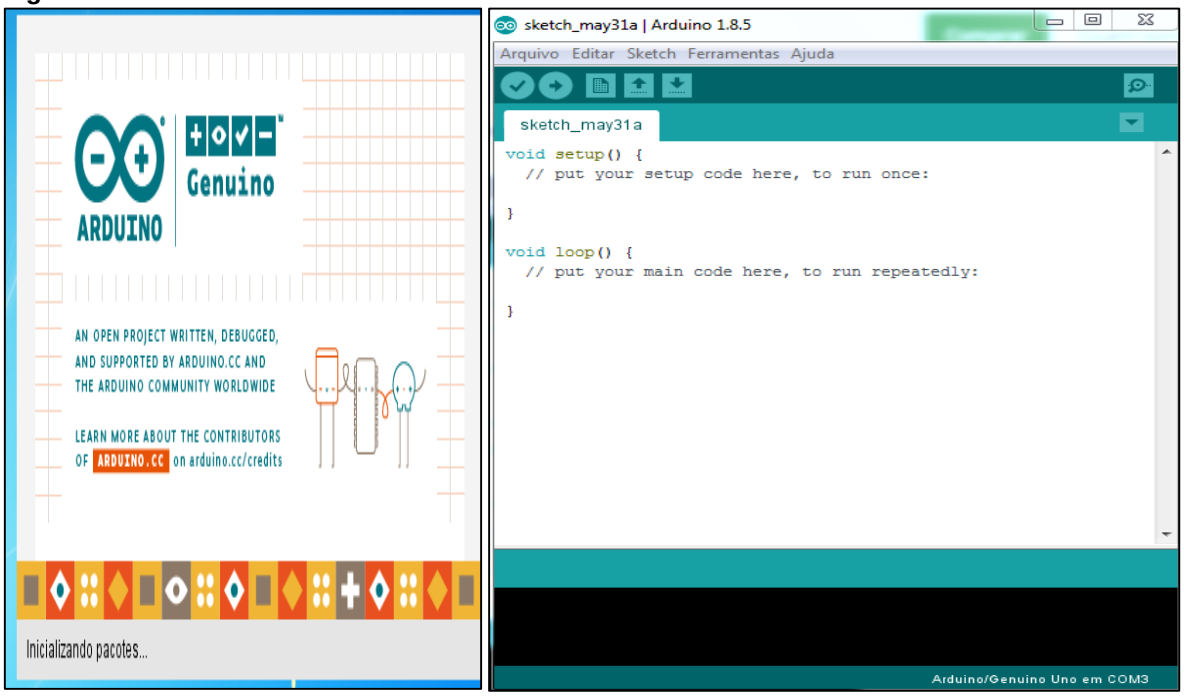
Para iniciar a utilização da placa Arduino e suas funcionalidades deve-se efetuar o *download* da IDE Arduino para a plataforma operacional desejada (Microsoft Windows®, Mac OS®, Linux®, por exemplo), diretamente na página oficial, disponível em: <https://www.arduino.cc/en/Main/Software>, acesso em: 09 jun. 2018.

Figura 19 - Página para Download Software Arduino

Fonte: Arduino (2018). Disponível em: <https://www.arduino.cc/en/Main/Software>. Acesso em: 31 mai. 2018.

Uma vez realizado o processo de instalação será possível abrir a IDE do Arduino e começar o desenvolvimento de projetos (Figura 20).

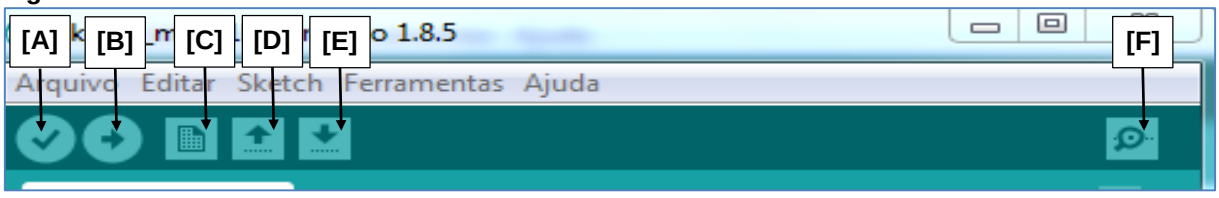
Figura 20 - IDE Arduino



Fonte: Autoria própria.

O Arduino possui uma IDE extremamente simples e objetiva, facilitando o processo de desenvolvimento. Além do espaço em branco central destinado ao desenvolvimento do programa, existem seis botões na parte superior (Figura 21): [A] Verificar: erros no código implementado; [B] Carregar: enviar (gravar) o programa na placa *Arduino*; [C] Novo: criar um novo código-fonte Arduino; [D] Abrir: código-fonte Arduino existente, [E] Salvar: código-fonte Arduino implementado; e [F] Monitor Serial: abre um monitor de dados da porta serial.

Figura 21 - Barra de botões da IDE Arduino



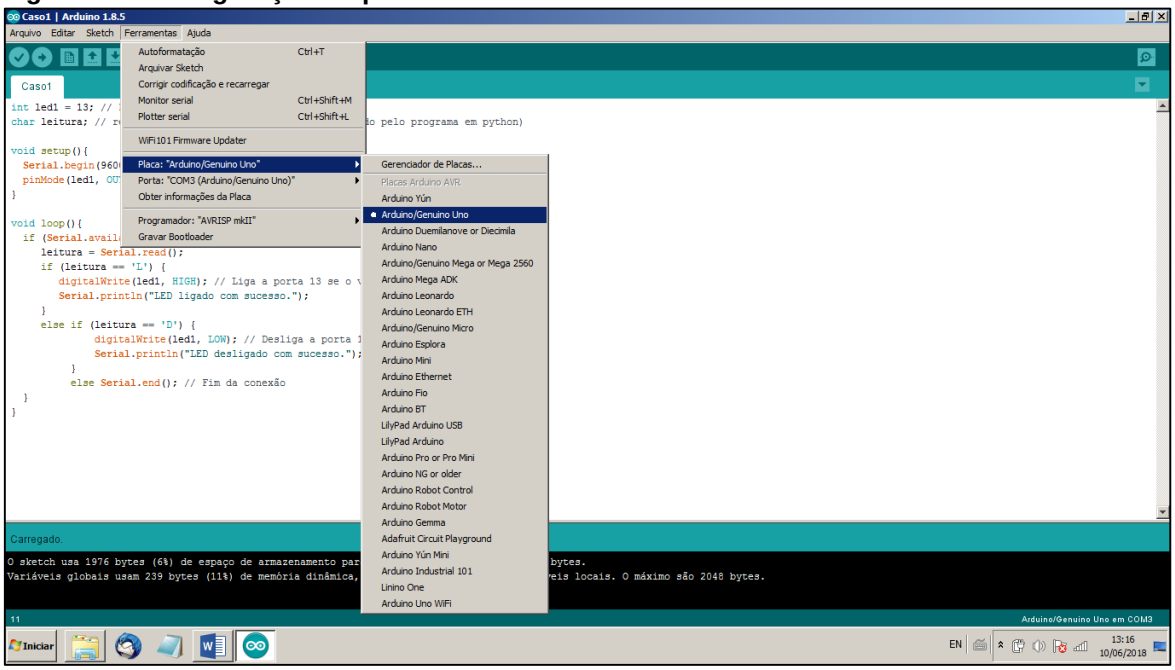
Fonte: Autoria própria.

Um código-fonte desenvolvido na linguagem de programação Arduino é chamado de *sketch*, traduzindo do inglês seria algo como “esboço” ou “rascunho”. Um *sketch* possui a extensão “.pde” (“*.py” na linguagem de programação Python).

Para implementar um novo programa no Arduino, primeiramente basta conectar a placa na porta USB (item [A] na Figura 17) e na *IDE Arduino* selecionar a placa

utilizada em “Ferramentas | Placa: “Arduino/ Genuino Uno” | Arduino / Genuino”, como mostra a Figura 22.

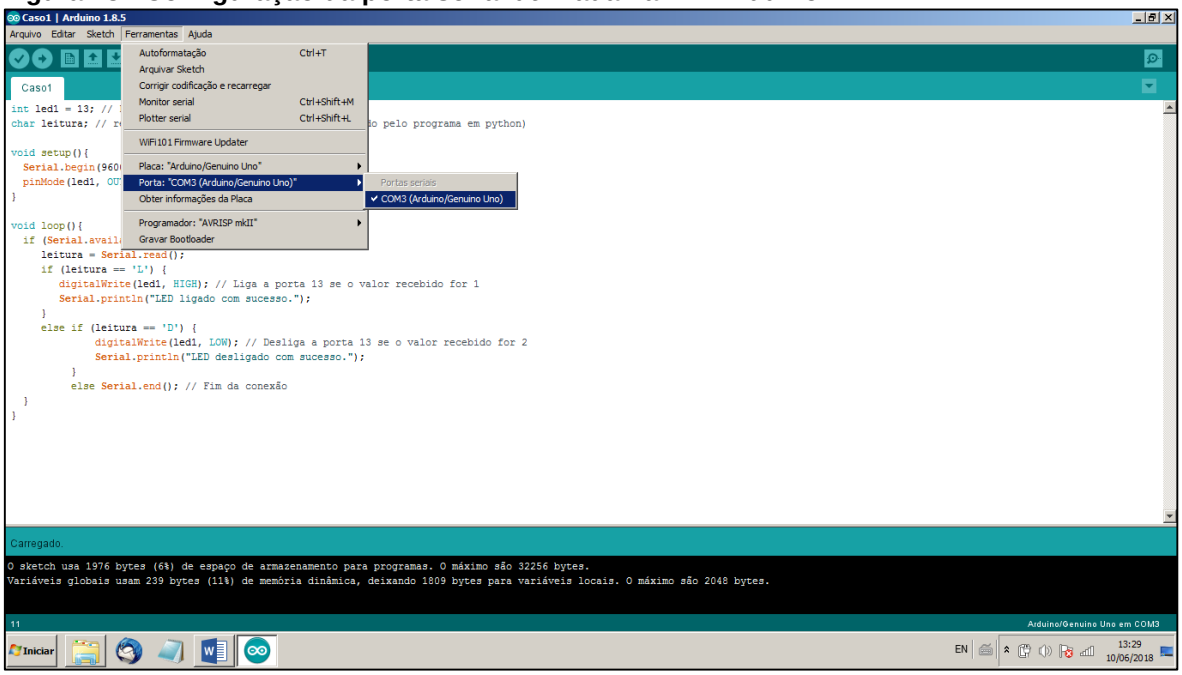
Figura 22 - Configuração da placa Arduino utilizada na IDE Arduino



Fonte: Autoria própria.

No próximo passo para o desenvolvimento do programa Arduino deve-se selecionar a Porta Serial (COM, sigla de *Communications*) associada a placa Arduino Uno em “Ferramentas | Porta: “COM 3 (Arduino/Genuino UNO)” | COM3 (Arduino/Genuino UNO)”, como mostra a Figura 23.

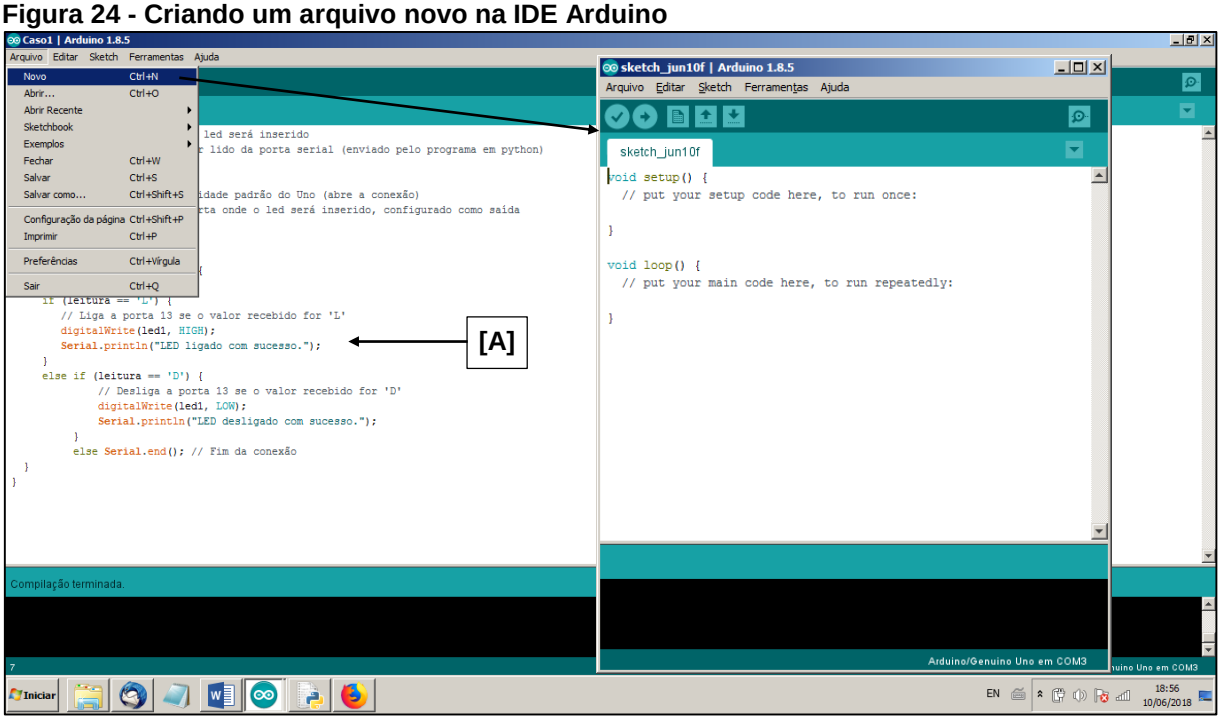
Figura 23 - Configuração da porta serial utilizada na IDE Arduino



Fonte: Autoria própria.

Para dar início ao processo de implementação do programa Arduino (*sketch*) deve-se criar um novo arquivo na IDE Arduino por meio da opção “Arquivo | Novo Ctrl+N”, como mostra a Figura 24.

Figura 24 - Criando um arquivo novo na IDE Arduino



Fonte: Autoria própria.

Finalizado o processo de implementação (item [A] na Figura 24) deve-se verificar se não existem problemas de sintaxe[20] no código-fonte (*sketch*) pressionando o botão “Verificar” (item [A] na Figura 21). Neste momento acontece a etapa denominda de “tempo de compilação” e durante a realização dessa fase as seguintes mensagens são exibidas na parte inferior da IDE Arduino: a) “Compilando sketch...”, e b) “Compilação terminada”.

Uma vez concluído com sucesso o processo de compilação o programa (*sketch*) deve ser carregado na placa Arduino pressionando o botão “Carregar” (item [B] na Figura 21). Neste momento acontece a etapa denominada de “tempo de execução” e durante a realização dessa fase as seguintes mensagens são exibidas na parte inferior da IDE Arduino: a) “Compilando sketch...”, b) “Carregando...”, e c) “Carregado.”.

Esta seção foi de enorme importância para mostrar ao estudante iniciante de o Arduino, foram apresentado as informações importantes como a instalação do

[20] Um erro de sintaxe ocorre quando as sentenças do programa estão mal formuladas.

software Arduino*,* com a descrição da placa o estudantes pode conhecer melhor o Arduino podendo perceber como essa placa é de fácil utilização. Foi demonstrado nessa seção os elementos básico do Arduino, construindo para o estudante uma base sólida para o seu ingresso na tecnologia Arduino.

Procurou-se demonstrar os códigos procurando facilitar o aprendizado de maneira simples e objetiva, para que haja uma melhor compreensão dos estudantes. Com o conhecimento de códigos de exemplo, o aluno poderá praticar, ajudando assim a sua experiência na plataforma Arduino.

4 INTEGRAÇÃO ENTRE A LINGUAGEM DE PROGRAMAÇÃO PYTHON E ARDUINO

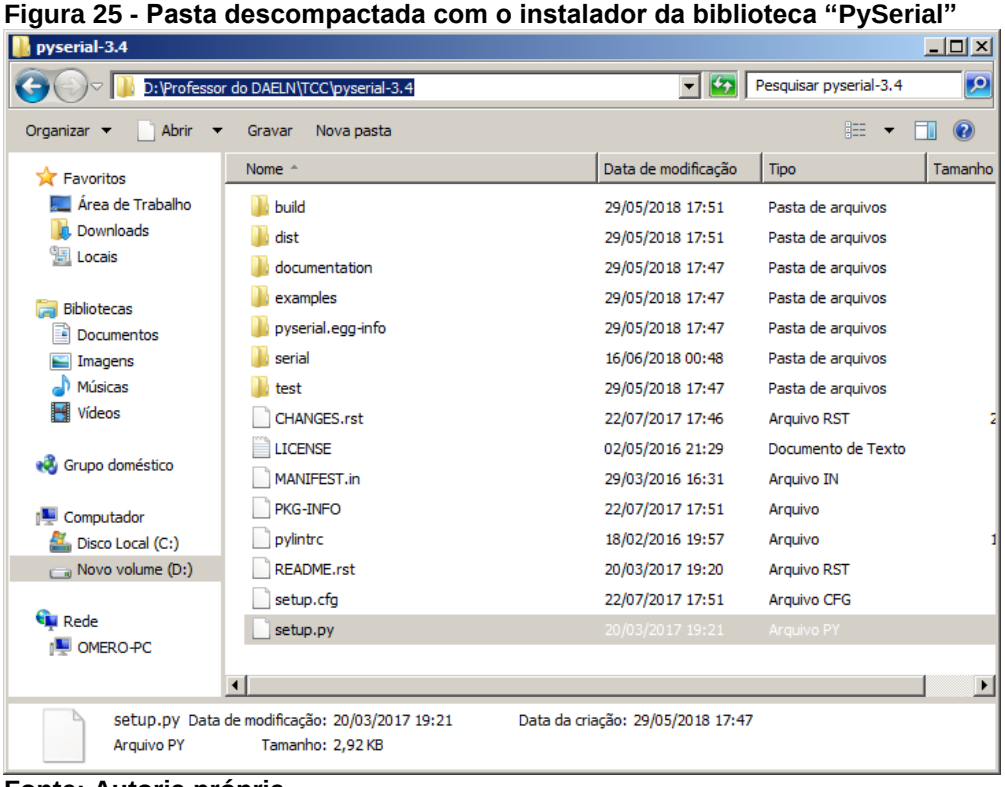
Nesta seção será demonstrada o estudo de caso desenvolvido efetuar a comunicação serial entre um aplicativo implementado na linguagem de programação Python e a placa Arduino.

4.1 BIBLIOTECA “PYSERIAL”

PySerial[21] é uma biblioteca de código[22] desenvolvida para facilitar a comunicação da linguagem de programação Python com dispositivos eletrônicos via porta serial (PYSERIAL, 2015).

Para instalar a biblioteca PySerial deve-se fazer primeiramente o *download* do instalador, “pyserial-3.4.tar.gz” para o ambiente operacional Microsoft Windows®, disponível em: <https://pypi.org/simple/pyserial/>, acesso em: 16 jun. 2018. A seguir deve-se descompactar o arquivo com o instalador, denominado de “*setup.py*”, em uma pasta qualquer (Figura 25).

Figura 25 - Pasta descompactada com o instalador da biblioteca “PySerial”



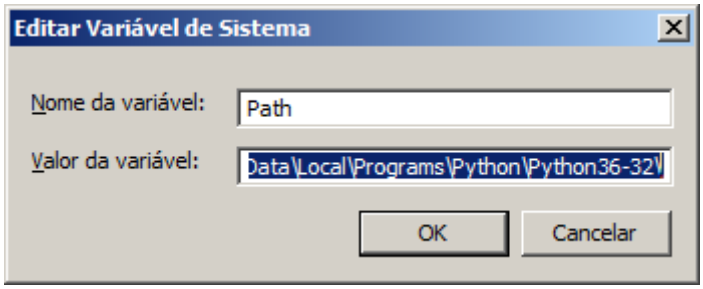
Fonte: Autoria própria.

[21] © Copyright 2001-2015, Chris Liechti.

[22] Biblioteca é um conjunto de códigos (classes, funções) utilizados no desenvolvimento de software.

O instalador “*setup.py*” da biblioteca “*PySerial*” trata-se de um programa desenvolvido na linguagem de programação Python e para executá-lo deve-se fazê-lo em linha de comando do ambiente operacional. No caso do Microsoft Windows®, inicialmente deve-se acrescentar o caminho do aplicativo “*Python.exe*”, por exemplo: “*C:\Users\omero\AppData\Local\Programs\Python\Python36-32*”, nas variáveis de ambiente do sistema (Figura 26).

Figura 26 - Acrescentando o caminho do aplicativo “Python.exe” nas variáveis de ambiente do sistema

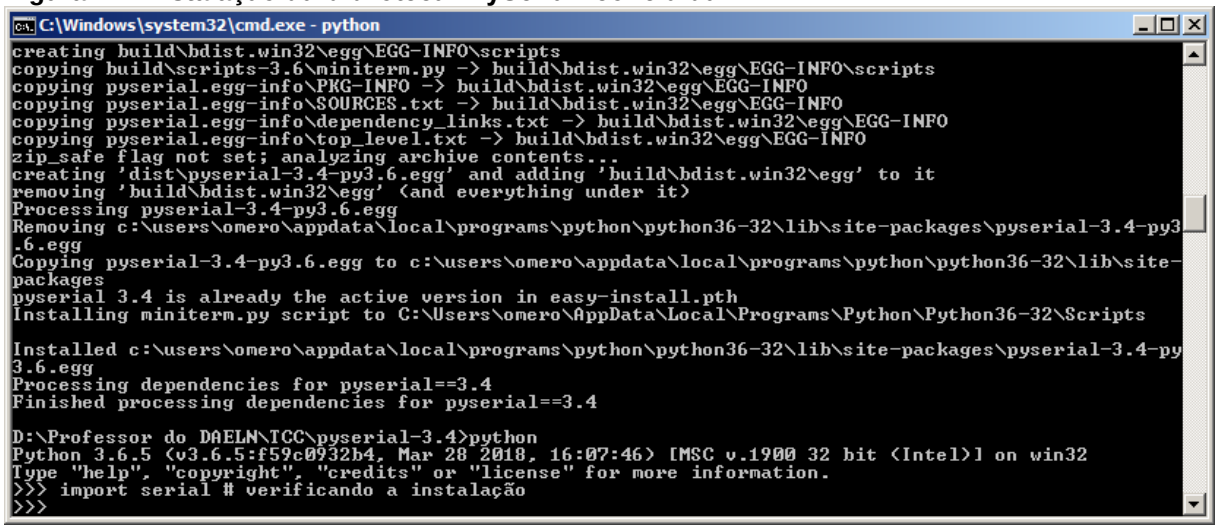


Fonte: Autoria própria.

Uma vez acrescentado o caminho do aplicativo “*Python.exe*” deve-se em linha de comando no ambiente operacional[23], executar o instalador “*setup.py*” com a instrução: “*python setup.py install*”.

Na Figura 27 pode-se observar o processo de instalação da biblioteca “*PySerial*” finalizado, juntamente com o comando na linguagem de programação Python para verificar se a instalação foi concluída com sucesso: *import serial*.

Figura 27 - Instalação da biblioteca “PySerial” concluída



Fonte: Autoria própria.

[23] No Microsof Windows® para iniciar a linha de comando basta digitar “**cmd**” (abreviação de “command line”) na opção “Executar” (Run) no botão “Iniciar”.

4.2 ESTUDO DE CASO: COMUNICAÇÃO SERIAL, INTEGRAÇÃO LINGUAGEM DE PROGRAMAÇÃO PYTHON E ARDUINO

No estudo de caso foi implementado um código na linguagem de programação Python (Listagem 26) para estabelecer uma comunicação serial[24] com a placa Arduino utilizando os métodos da classe "serial" disponível na biblioteca "*PySerial*" (PYSERIAL, 2015).

Listagem 26 - Estudo de caso: código implementado na linguagem de programação Python

```
import serial

# Configuração da conexão
conexao = serial.Serial('COM3', 9600)

print("__Menu Principal__")
print("[ 1 ] Ligar LED")
print("[ 2 ] Desligar LED")
print("[ 0 ] Encerrar\n")
while True:
  opcao = int(input("Opção desejada: "))
  if (opcao == 0):
     break
  elif ((opcao == 1) or (opcao == 2)):
    if (opcao == 1):
       # Escreve L no arduino (LED acende)
       conexao.write(b'L')
    else:
       # Escreve D no arduino (LED apaga)
       conexao.write(b'D')

    result = conexao.readline()
    result = result.decode("utf-8")
    print("Resultado:", result[:(len(result)-2)], "\n")

conexao.write(b'F') # Fim da conexão
conexao.close()
```

Fonte: Autoria própria.

Na aplicação desenvolvida na linguagem de programação Python (Listagem 26) a comunicação serial é estabelecida com o comando: *conexao = serial.Serial('COM3', 9600)*. A sentença: *conexao.write(b'L')*, escreve (*write*) o binário do caractere 'L' na placa Arduino para indicar que o *LED* deve ser ligado. Já no caso: *conexao.write(b'D')*, escreve o binário do caractere 'D' na placa *Arduino* para indicar

[24] A comunicação serial é o processo de enviar dados na forma de um bit (0 ou 1) de cada vez, sequencialmente, por meio de um canal de comunicação ou barramento.

que o LED deve ser desligado. O resultado da execução das opções de ligar e desligar o LED é recuperado (*read*) da placa Arduino com a ação: *result = conexao.readline()*. A comunicação serial é finalizada com as sentenças: a) *conexao.write(b'F')*, e b) *conexao.close()*.

A Listagem 27 apresenta o código implementado na linguagem de programação Arduino que complementa o estudo de caso desenvolvido.

Listagem 27 - Estudo de caso: código implementado na linguagem de programação Arduino

```
int led1 = 13; // Porta onde o led será inserido
// recebe o valor lido da porta serial
// (enviado pelo programa em python)
char leitura;

void setup(){
// Velocidade padrão do Uno (abre a conexão)
  Serial.begin(9600);
// Porta onde o led será inserido, configurado como saída
  pinMode(led1, OUTPUT);
}

void loop(){
  if (Serial.available() > 0) {
     leitura = Serial.read();
     if (leitura == 'L') {
     // Liga a porta 13 se o valor recebido for 'L'
        digitalWrite(led1, HIGH);
        Serial.println("LED ligado com sucesso.");
     }
     else if (leitura == 'D') {
           // Desliga a porta 13 se o valor recebido for 'D'
              digitalWrite(led1, LOW);
              Serial.println("LED desligado com sucesso.");
          }
          else Serial.end(); // Fim da conexão
  }
}
```

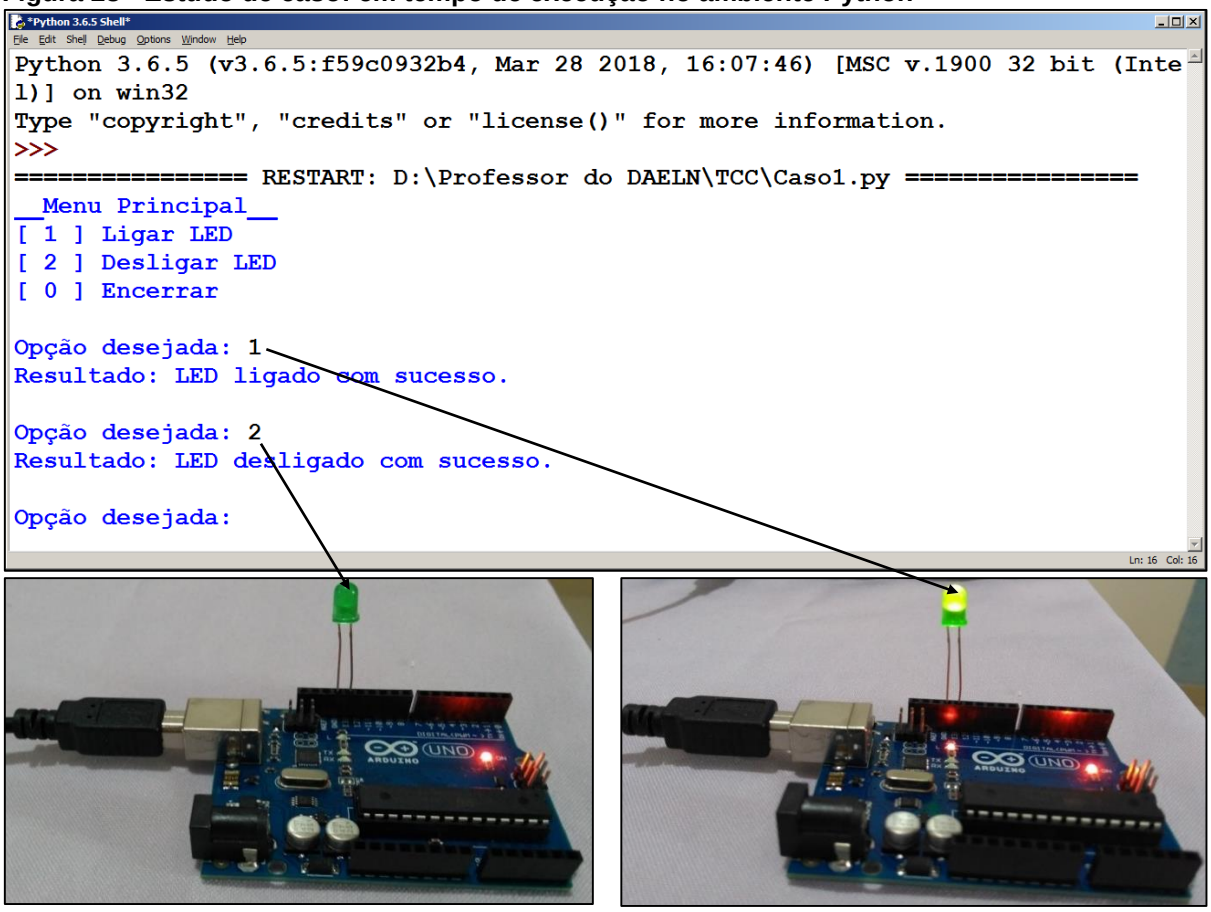
Fonte: Autoria própria.

Na aplicação desenvolvida na linguagem de programação Arduino (Listagem 27) a comunicação serial é estabelecida com o comando: *Serial.begin(9600)*. A expressão: *Serial.available() > 0*, verifica se existe um dado disponível no fluxo de entrada da comunicação. Se existe dado, a sentença: *leitura = Serial.read()*, atribui a variável “leitura” o valor recuperado (*read*). Se o valor da variável “leitura” for igual ao caractere ‘L’ então o LED será ligado: *digitalWrite(led1, HIGH)* e a mensagem “LED ligado com sucesso” será escrita (*write*) na placa Arduino: *Serial.println("LED ligado*

com sucesso."); em caso contrário o LED será desligado: *digitalWrite (led1, LOW)* e a mensagem “LED desligado com sucesso” será escrita na placa *Arduino*: *Serial.println ("LED desligado com sucesso.")*. A comunicação serial é finalizada com a sentença: *Serial.end()*.

Na Figura 26, pode-se observar a execução do código da Listagem 26, parte do estudo de caso desenvolvida na linguagem de programação Python, que oferece ao usuário final um menu principal as seguintes opções: 1) ligar o LED na placa Arduino, 2) desligar o LED na placa Arduino, e 0) encerrar a execução do programa.

Figura 28 - Estudo de caso: em tempo de execução no ambiente Python



Fonte: Autoria própria.

Nesse seção foi demostrado a integração entre a linguagem de programação Python e Arduino, o estudo de caso mostrou e explicou a praticidade de se trabalhar com essas tecnologias, mostrando ao estudante que a comunicação entre as duas tecnologias é simples e há muita facilidade, também foi possível observar, repetir e testar os códigos apresentados nessa seção, com isso o estudante tem o primeiro passos de muitos nesse aprendizado.

5 CONCLUSÃO

Neste trabalho de conclusão de curso foi mostrado de forma clara e objetiva os passos para aprender a linguagem de programação Python. Motivando assim um número maior de estudantes nessa área de atuação, auxiliando no início de todo esse processo de aprendizado.

Foi mostrado nesse trabalho de conclusão de curso a praticidade de aprender a plataforma Arduino, sua instalação foi mostrada de forma clara e eficiente, motivando o aluno a ser inserido aos poucos nesse aprendizado, foi apresentado os elementos e a base de todo Arduino facilitando assim o aprendizado do estudante.

O estudo de caso apresentado nesse trabalho de conclusão de curso foi de enorme importância, pode-se mostrar e detalhar a integração entre Python e Arduino, abrindo novas oportunidades para os iniciantes, mostrando a praticidade de trabalhar com essa integração.

A cada capítulo desse trabalho foi demonstrado que o mais leigo e sem conhecimento em eletrônica e programação pode com toda certeza ter a oportunidade de aprender o que foi proposto no início do trabalho. A cada exemplo poderá ser seguido e testado pelo aluno, trazendo então uma enorme importância para o que foi proposto.

Este trabalho de conclusão de curso, atuou na base do aprendizado de programação, motivando o aluno, a entender e compreender que todo início pode ser trabalhoso, porém precisa ser claro e objetivo para o estudante. O que pode-se concluir em todo esse trabalho de conclusão de curso que com uma base sólida e bem explicada, e seguindo os passos propostos pode-se aprender programação.

Por meio desse trabalho de conclusão de curso, acredita-se que um caminho de novas oportunidades foi aberto para os iniciantes na linguagem de programação Python e Arduino.

Espera-se ainda que por meio desse trabalho de conclusão de curso, os iniciantes de programação possam perceber a importância desse tema, e que esse trabalho possa auxiliar o início do aprendizado em programação.

REFERÊNCIAS

ARDUINO. **Arduino - Home**. Disponível em: <https://www.arduino.cc/>. Acesso em: 10 jun. 2018.

ARDUINO PROGRAMMING LANGUAGE. **Documentação de referência da linguagem Arduino: estruturas, valores (variáveis e constantes) e funções**. Disponível em: <https://www.arduino.cc/reference/pt/>. Acesso em: 07 jun. 2018.

BERTOL, Omero Francisco Bertol. **Material de aula para a linguagem de programação Python**. Curitiba: 2017. Disponível em: <http://acesso.materdei.edu.br/omero/Python/ppt.htm>. Acesso em: 19 mai. 2018.

CASS, Stephan. **The 2017 Top Programming Languages**. IEEE: 18 jul. 2018. Disponível em: <https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages>. Acesso em: 02 jun.2018.

FBS ELETRÔNICA. **Apostila Arduino: com aplicações baseada na placa Arduino Uno**. Disponível em: <http://www.valdick.com/files/ApostilaArduinoIntroducao.pdf>. Acesso em: 07 jun. 2018.

GOMES, Helton Simões. **Brasil tem 116 milhões de pessoas conectadas à internet, diz IBGE**. Globo.com: 21 fev. 2018. Disponível em: <https://g1.globo.com/economia/tecnologia/noticia/brasil-tem-116-milhoes-de-pessoas-conectadas-a-internet-diz-ibge.ghtml>. Acesso em: 15 jun. 2018.

LUAN, Paulo. **Demanda de programadores no mundo**. Diário de Programador: 29 mar. 2016. Disponível em: <http://pauloluan.github.io/blog/blog/2013/03/29/demanda-de-programadores-no-mundo/>. Acesso em: 01 Jun. 2018.

MENEZES, Nilo Ney Coutinho**. Introdução à Programação com Python: algoritmos e lógca de programação para iniciantes**. 2. ed. São Paulo: Novatec, 2014.

MULTILÓGICA-SHOP.**Guia Arduino Iniciantes.** Disponível em: < https://multilogica-shop.com/download_guia_arduino/>. Acesso em: 11 abr. 2018.

PYSERIAL. **Welcome to pySerial's documentation.** Copyright 2001-2015, Chris Liechti. Disponível em: <https://pythonhosted.org/pyserial/>. Acesso em: 14 jun. 2018.

PYTHON. **Python Software Foundation**. Copyright 2001-2018. Disponível em: <>. Acesso em: 18 jun. 2018.