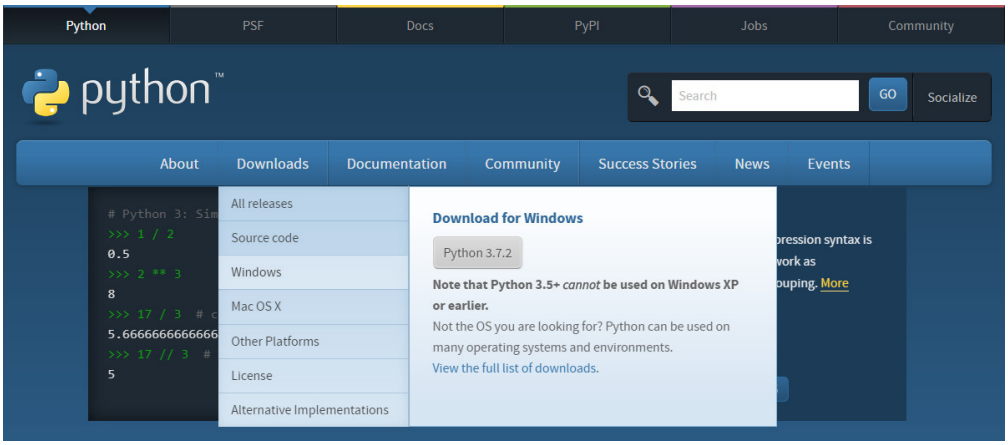
Sumário

Instalação

Neste capítulo aprenderemos como instalar a linguagem Python e sua IDE no Sistema Operacional Windows.

Linguagem Python

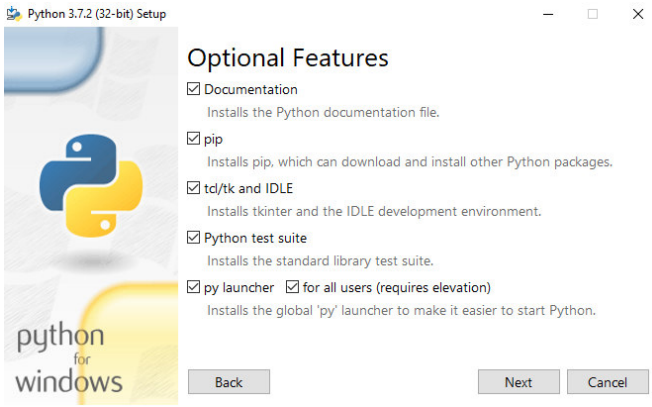
A linguagem de programação Python pode ser adquirida acessando sua aplicação através do endereço *www.python.org*.



Selecione o menu *Downloads* e baixe a versão Python 3.7.2. Execute o arquivo baixado e inicie sua instalação.

Na primeira tela marque a opção ***Add Python 3.7 to PATH*** e escolha a opção ***Customize installation***.

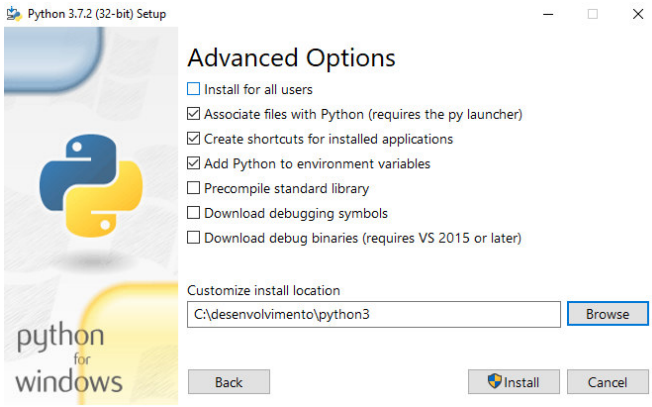
A seguir permaneça selecionado todas as opções e clique em ***Next***.



Cada opção marcada possui o seguinte significado:

- **Documentation:** instala a documentação da linguagem;
- **pip:** instala o pip que é usado para baixar bibliotecas desenvolvidas em Python;
- **tcl/tk and IDLE:** instala a biblioteca Python criação de interface gráfica;
- **Python test suite:** instala a biblioteca Python responsável pelo tratamento de teste de *software*;
- **py launcher:** instala o launcher para aplicativos com a extensão py.

Adiante deixe marcada as opções recomendadas. Em **Customize install location** crie o seguinte caminho de instalação: *c:\desenvolvimento\python3*. Clique no botão ***Install*** e acompanhe o término de sua instalação.



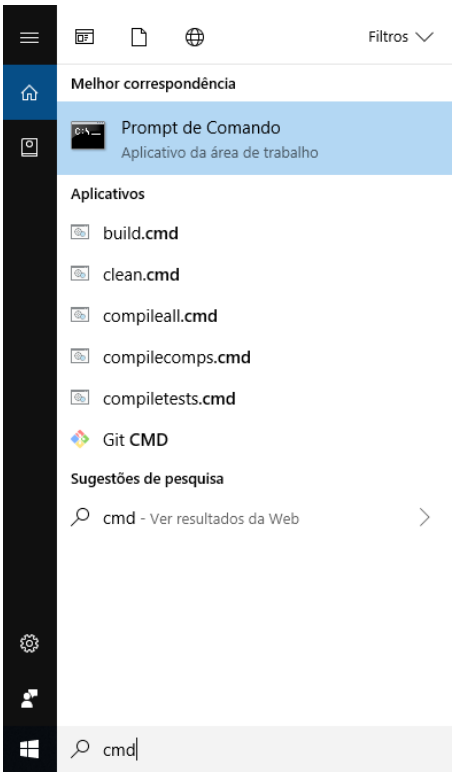
Primeiro programa

Agora vamos testar se a instalação ocorreu conforme planejado, criaremos nossa primeira aplicação. Abra o Prompt de Comando do Windows e digite o seguinte comando:

```
python -V
```

Neste comando será exibido a versão instalada em sua máquina. Para entrar no ambiente de desenvolvimento digite o comando:

```
python
```



Utilize o comando ***print*** para imprimir algo na tela. Teste a seguinte linha de comando:

```
print("Olá mundo")
```

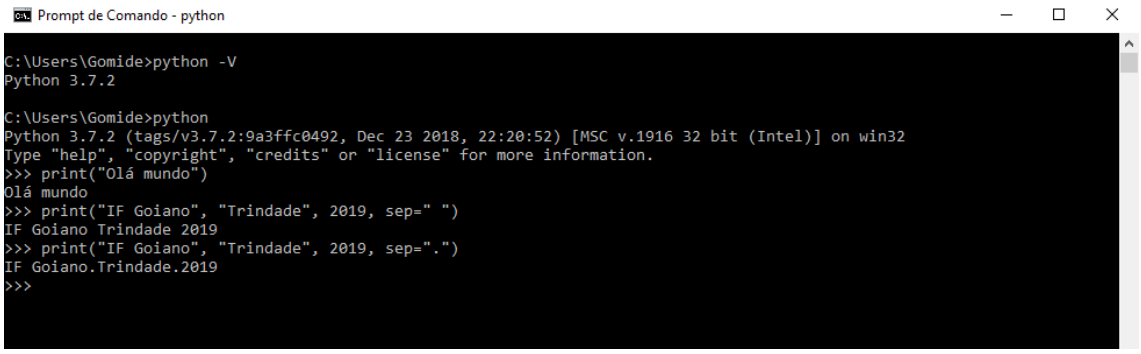
O comando apresentado trata-se de função. Por definição, toda função deve possuir parênteses. No caso apresentado a função ***print*** recebeu um conteúdo do tipo texto. A seguir entre com os seguintes comandos:

```
print("IF Goiano", "Trindade", 2019, sep=" ")
```

```
print("IF Goiano", "Trindade", 2019, sep=".")
```

A função ***print*** permite a impressão de diversos valores. Repare que cada valor deve ser separado por vírgula. É permitido apresentar valores de outros tipos que não seja texto. O último termo: `sep=" "`, define qual o caractere que irá ser utilizado como separador dos valores apresentados.

A seguir é apresentada a saída esperada para os comandos digitados.



Agora digite os seguintes comandos:

```
instituicao = "IF Goiano"
type(instituicao)
ano = 2019
type(ano)
print(instituicao, "Trindade", ano, sep=" ")
```

Repare que o comando `type(instituicao)` e `type(ano)` imprime respectivamente as saídas `<class 'str'>` e `<class 'int'>`. Essa saída refere-se aos tipos das variáveis `instituicao` e `ano`.

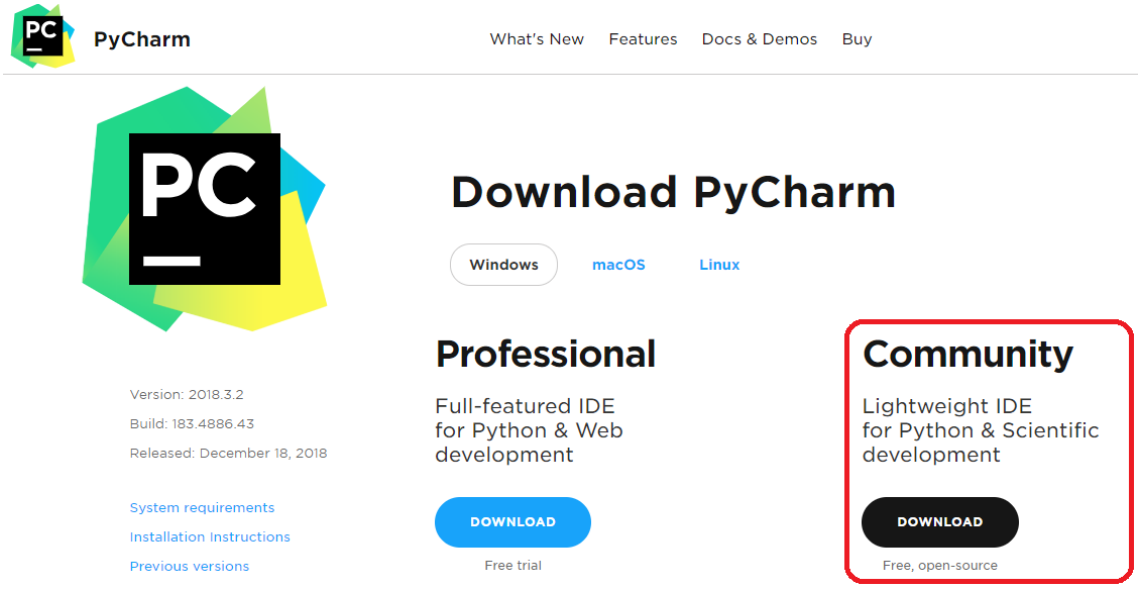
Instalando a IDE PyCharm

Desenvolver programas utilizando o Prompt de Comando pode ser uma tarefa extremamente difícil dependendo da complexidade da aplicação. Devido a esse problema os desenvolvedores utilizam aplicações próprias para o desenvolvimento de *softwares*. Essas aplicações são conhecidas pela sigla IDE, *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado.

Neste curso trabalharemos com a IDE *PyCharm*, desenvolvida pela empresa Jet Brains. Para realizar o *Download* desta ferramenta acesse o endereço:

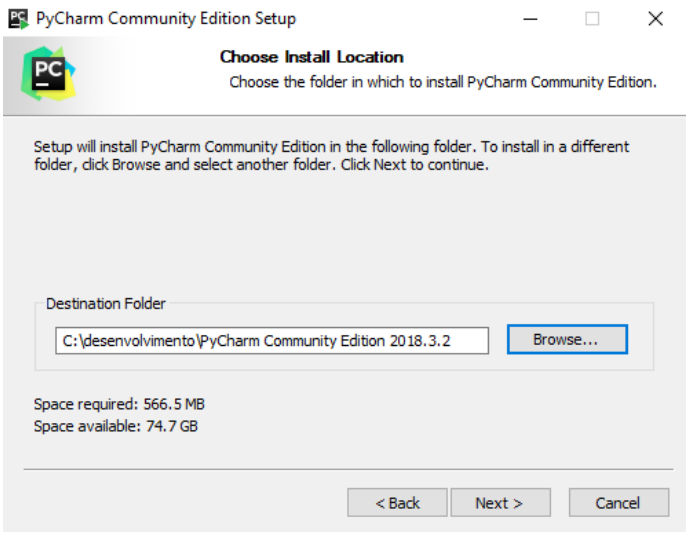
www.jetbrains.com/pycharm/download

e baixe a versão *Windows Community*.



Execute o arquivo, na tela que define o ***Destination Folder*** instale a aplicação no seguinte endereço:

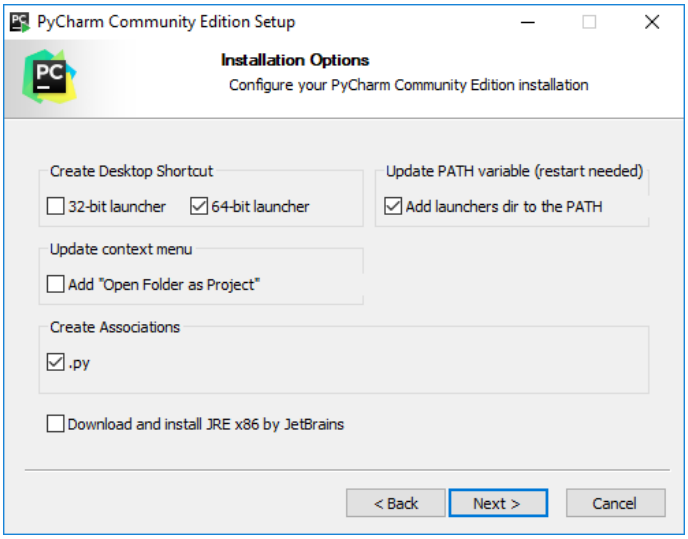
c:\desenvolvimento\PyCharm Community Edition 2018.3.2



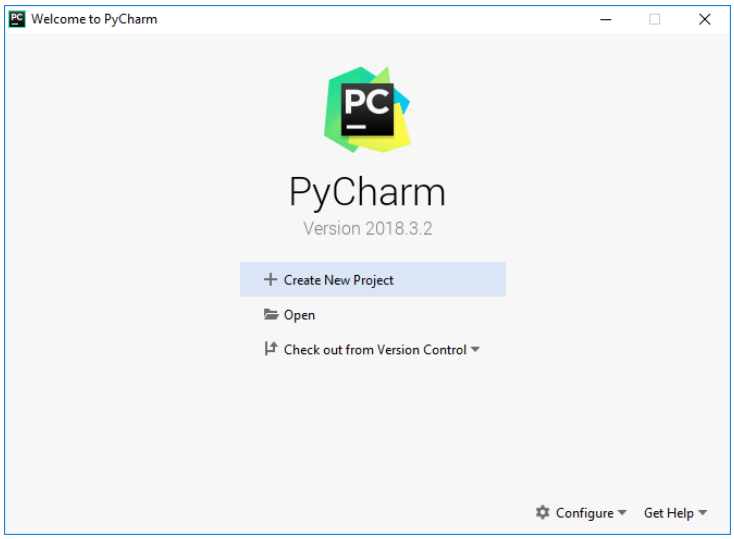
A seguir, em ***Installation Options***, marque as seguintes opções:

- 64-bit launcher;
- Add launcher dir to the PATH;
- .py

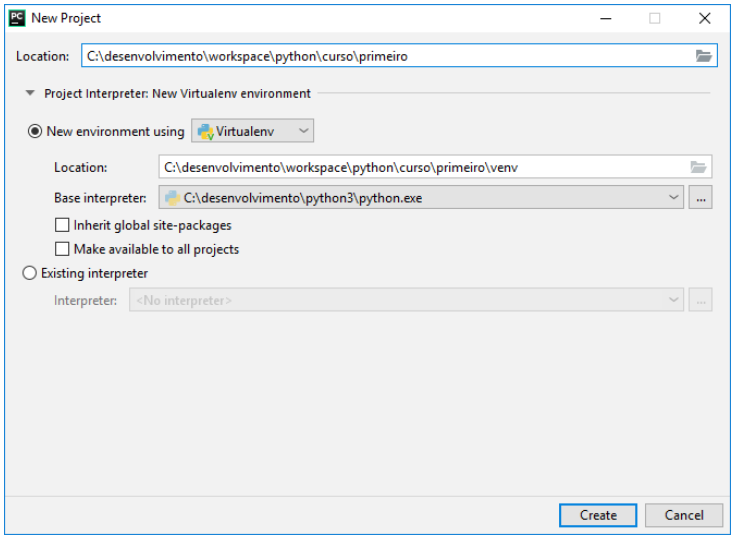
e clique no botão ***Next***.



Após ter sua instalação concluída, abra o programa. Escolha a opção ***Create New Project***.



Será disponibilizado uma tela para definir o caminho do projeto e qual a máquina virtual a ser utilizada pelo projeto.

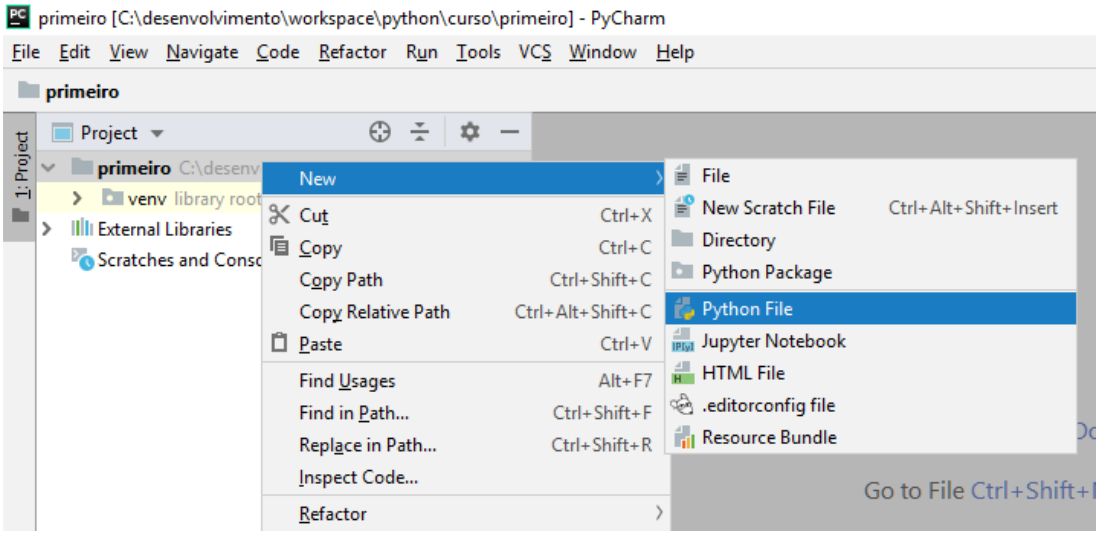


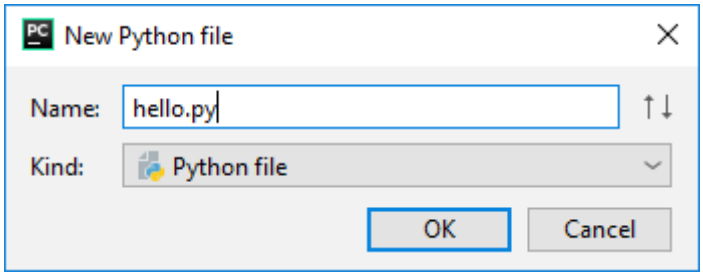
Em ***Location*** crie o seguinte caminho:

c:\desenvolvimento\workspace\python\curso\primeiro

Na opção ***Project Interpreter*** deixe marcado a máquina virtual recomendada. Clique no botão ***Create***.

Com o projeto criado clique com o botão direito no nome do projeto e escolha a opção ***New → Python File***. Nomeie o arquivo como *hello.py*.





Digite o seguinte código:

```
hello.py
1  nome = "Denecley"
2  sobrenome = "Alvim"
3  print("Bem vindo", nome, sobrenome, sep=" ")
```

Para executar clique no menu ***Run → Run*** ou pressione as teclas ***Alt + Shift + F10***. O resultado será exibido no canto inferior da tela do aplicativo.

Fundamentos

Desenvolvida por Guido van Rossum, a linguagem Python surgiu em 1991. Esta linguagem possui as seguintes características:

- Interpretada: não utiliza os recursos diretamente da máquina;
- Alto nível: baseada em outras linguagens;
- Script: suporta criação de *scripts*;
- Paradigma orientado a objetos: utiliza os conceitos de classe e objetos;
- Tipagem dinâmica: variáveis não possuem tipos definidos.

A estrutura da linguagem de programação determina que todo código esteja devidamente identado. As estruturas e blocos não são limitados com um início e fim.

Comentários

Um comentário permite que o desenvolvedor escreva algo no programa que não será interpretado pela linguagem. Este comentário pode ser de uma única linha, exemplo:

```
# Isso é apenas um comentário
```

Ou de múltiplas linhas:

```
'''
Fui a feira e digitei
os dados dos produtos
que adquiri
'''
```

O comentário de múltiplas linhas deve estar envolvido por três aspas simples no início e encerramento.

Variáveis

Em Python uma variável deve possuir apenas um nome e um valor. O tipo da variável é definido através do valor recebido, ou seja, a linguagem oferece a criação de variáveis não tipadas, que não possuem um tipo pré-definido.

Sendo assim uma variável ser um número ou um texto. Caso seja um número, esta poderá conter valores:

- **int** → armazena valores do inteiros;
- **float** → armazena valores reais;
- **complex** → armazena valores que possuem uma parte real e outra imaginária. Normalmente esses tipos são empregados em áreas como Engenharia Elétrica e Estatística.

```
>>> 51 #int
51
>>> 21.7 #float
21.7
>>> 5 + 4j #complex
(5+4j)
```

Um número também pode ser gerado por meio das funções: `int(); float(); e complex()`. Exemplo:

```
>>> int(8.9)
8
>>> int('34')
34
>>> float(3)
3.0
>>> float('24.34')
24.34
>>> complex(3, 5)
(3+5j)
```

Variáveis que possuem valores do tipo texto ser envolvidos por aspas simples ou duplas. Exemplo:

```
>>> 'Starcraft'
'Starcraft'
>>> "Isso é apenas um teste"
'Isso é apenas um teste'
```

Para nomear uma variável, e permitido utilizar qualquer caractere que não seja símbolo chave da linguagem. Uma variável pode possuir qualquer nome, com exceção das palavras reservadas da linguagem.

Comandos de Entrada e Saída

O desenvolvedor poderá imprimir uma saída através do comando ***print***. Este comando pode ser utilizado da seguinte forma:

- Imprimindo apenas o texto desejado. Exemplo:

```
print("Estou imprimindo este texto!")
```

- Imprimindo uma sequência de dados (variáveis e textos). Exemplo:

```
print("Olá", nome)
```

- Imprimindo uma sequência de dados que são delimitados por um caractere ou texto. Exemplo:

```
print("Hoje", "teremos", sep = "$")
```

Em python deve-se utilizar o comando *input* para capturar a entrada do usuário. Este comando retornará sempre um texto. Exemplo:

```
nome = input("Digite seu nome")
```

Operadores

A linguagem oferece quatro categorias de operadores: atribuição; aritméticos; relacionais e lógicos.

Operador de atribuição

Este operador tem como objetivo definir o conteúdo e o tipo de uma variável. Ele é representado pelo símbolo igual (=).

Operadores Aritméticos

Correspondem aos operadores utilizados na realização de cálculos matemáticos. São eles:

Descrição	Símbolo
Adição	+
Subtração	-
Divisão	/
Multiplicação	*
Resto	%

Operadores Relacionais

Correspondem aos operadores utilizados na criação de sentenças relacionais. Esses operadores são bastante utilizados nas estruturas condicionais e de repetição. São eles:

Descrição	Símbolo
Maior	>
Menor	<
Maior ou igual	>=
Menor ou igual	<=
Igualdade	==
Diferente	!=

Operadores Lógicos

Participam da construção de uma sentença lógica. Também são utilizados nas estruturas condicionais e de repetição. São eles:

Descrição	Símbolo
E lógico	and
OU lógico	or
NÃO lógico	not

Cada operador possui a seguinte tabela verdade:

and		
P	**Q**	**P and Q**
True	True	True

True	False	False
False	True	False
False	False	False

or		
P	**Q**	**P and Q**
True	True	True
True	False	True
False	True	True
False	False	False

not	
P	**not P**
True	False
False	True

Lista de exercícios A

1) Desenvolva um programa em Python no qual o usuário deverá digitar o cateto adjacente e cateto oposto relativo a um triângulo retângulo. O programa deverá calcular a hipotenusa deste triângulo e suas respectivas razões trigonométricas (seno, cosseno, e tangente). Imprima os valores do seno, cosseno e tangente.

Cálculo da hipotenusa:

$h^2 = ca^2 + co^2$

Cálculo do seno, cosseno e tangente

$$seno = \frac{co}{h} \qquad \cos seno = \frac{ca}{h} \qquad \tan gente = \frac{co}{ca}$$

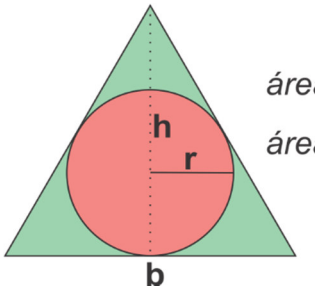
, onde:

- h = hipotenusa;
- ca = cateto adjacente;
- co = cateto oposto.

2) Desenvolva um programa em Python que gere um número aleatório de quatro algarismos (M – Milhar, C – Centena, D – Dezena, U - Unidade) variando entre 1000 e 9999. O programa deverá calcular e imprimir a seguinte expressão matemática:

$$x = \frac{U + randi(M * 10)}{\sqrt{C} + 2^{D}}$$

3) Desenvolva um programa em Python no qual o usuário digitará o raio de um círculo, a base e a altura de um triângulo. O programa deverá calcular a área do círculo e do triângulo e imprimir a diferença da área do triângulo pela área do círculo.



área do círculo = π.r²

área do triângulo = b . h / 2

4) O IF Goiano – Campus Trindade deseja verificar qual foi a pontuação obtida por um determinado candidato inscrito no processo eleitoral para coordenador de curso. A equipe de desenvolvedores decidiu ajudar nesta tarefa por meio da criação de um programa Python. Neste algoritmo é necessário que o usuário entre com: o número de votos válidos do segmento docente ao candidato; o total de votos válidos dos docentes; o número de votos válidos do segmento discente ao candidato; e o número de votos válidos dos discentes. O programa deverá realizar o seguinte cálculo:

$$Vt = \frac{2}{3} \cdot \frac{VDo}{Do} + \frac{1}{3} \cdot \frac{VDi}{Di}$$, sendo:

- Vt: Votos válidos obtidos pelo candidato;
- VDo: Votos válidos do segmento docente ao candidato;
- Do: Votos válidos dos docentes;
- VDi: Votos válidos do segmento discente ao candidato;
- Di: Votos válidos dos discentes.

Ao final do cálculo, o valor deverá ser exibido na tela.

5) Desenvolva um programa em Python no qual o usuário digitará um ângulo. O programa deverá calcular e imprimir o seno, o cosseno e a tangente deste ângulo.

6) Desenvolva um programa em Python que gere um número aleatório variando entre 0 e 100. Imprima o valor gerado.

7) Desenvolva um programa em Python que gere 3 números aleatórios variando respectivamente entre: 0 e 50; 10 e 40; 100 e 999. Imprima os valores gerados.

8) Desenvolva um programa em Python no qual o usuário digitará um valor inicial e um valor final. O programa deverá gerar um número aleatório dentro do intervalo escolhido pelo usuário. Imprima o número gerado.

9) Desenvolva um programa em Python no qual o usuário digitará dois números (n1 e n2). O programa deverá calcular e imprimir a seguinte fórmula:

$$resultado = \left(\sqrt{n1} - \sqrt{n2}\right)^3$$

Dicas:

- Calcular a potência de um número. Exemplo:

2 ** 5 = 32

- Calcular a raiz quadrada de um número. Exemplo:

4 ** (1/2) = 2

- Gerar números aleatórios. Exemplo:

```
import random
random.randint(1, 9)
```

- Gerar PI

```
import math
math.pi
```

- Obter seno, cosseno e tangente

```
import math
```

```
x = math.radians(85) #converte o ângulo de grau para radiano
math.sin(x) #seno
math.cos(x) #cosseno
z = math.tan(x) #tangente
y =math.degrees(z) #converte o ângulo de radiano para grau.
```

Estruturas condicionais

Uma estrutura condicional visa verificar uma sentença relacional e a partir desta verificação ocorre a tomada de decisão. Essa ação ocorrer quando a sentença relacional for verdadeira (True). Também é possível realizar ações o resultado da sentença relacional seja falso (False).

A linguagem oferece três tipos de estruturas condicionais: IF, ELIF; e ELSE.

A estrutura IF

Possui a seguinte sintaxe:

```
if condicao:
      comandos
```

onde:

- `if` → `corresponde ao nome da estrutura de condição;`
- `condicao` → `sentença relacional a ser testada;`
- `comandos` → `todos os comandos a serem executados pela estrutura.`

A estrutura ELIF

Utilizado existe a necessidade de testar várias sentenças relacionais. Para cada teste condicional utiliza-se um ELIF. Essa estrutura corresponde ao if em cascata. Observe sua sintaxe:

```
if condicaoA:
      comandoA
elif condicaoB:
```

```
      comandoB
elif condicaoC:
      comandoC
```

onde:

- `if` → corresponde ao nome da estrutura de condição;
- `elif` → utilizado para testar outra condição além da principal;
- `condicaoA; condicaoB; condicaoC` → sentenças relacionais a serem testadas;
- `comandoA; comandoB; comandoC` → todos os comandos a serem executados nos diferentes blocos.

A estrutura ELSE

Utilizando quando alguma sequencia de comandos será executada, mesmo todas as condições sendo falsas. Essa estrutura deve vir acompanhada de uma estrutura IF. Segue sua sintaxe:

```
if condicao:
      comandoA
else:
      comandoB
```

onde:

- `if` → corresponde ao nome da estrutura de condição;
- `else` → utilizado para executar um comando quando a condição contida no if for falsa;
- `condicao` → sentença relacional a ser testada;
- `comandoA; comandoB` → todos os comandos a serem executados nos diferentes blocos.

Dependendo do problema proposto, é possível integrar todas estruturas condicionais em uma só estrutura, sendo que o ELSE sempre será a última estrutura alocada.

Lista de exercícios B

1) Desenvolva um programa em Python no qual o usuário digitará uma letra (A até E) referente a um conceito de nota. O programa deverá converter a letra digitada em uma nota de acordo com a tabela abaixo:

CONCEITO	NOTA
A	95
B	85
C	75
D	65
E	30

Qualquer letra digitada diferente do que foi proposto na tabela acima, será impresso a seguinte mensagem "Conceito inválido". Imprima a nota convertida.

2) Desenvolva um programa em Python no qual o usuário digitará um valor referente a *x*. O programa deverá calcular e imprimir o valor de *f(x)*.

$$f(x) = \begin{cases} 1 & se \quad x \leq 1 \\ 2 & se \quad 1 < x \leq 2 \\ x^2 & se \quad 2 < x \leq 3 \\ x^3 & se \quad x > 3 \end{cases}$$

3) Desenvolva um programa em Python no qual o usuário digitará um valor numérico variando entre 1 a 7. O programa deverá converter este número em dia da semana de acordo com a tabela a seguir.

VALOR	SAÍDA
1	DOMINGO
2	SEGUNDA-FEIRA
3	TERÇA-FEIRA
4	QUARTA-FEIRA
5	QUINTA-FEIRA
6	SEXTA-FEIRA
7	SÁBADO

4) Desenvolva um programa em Python que leia um peso na Terra e o número de um planeta e imprima o valor do seu peso neste planeta. A relação de planetas é dada a seguir juntamente com o valor das gravidades relativas à Terra:

#	GRAVIDADE RELATIVA	PLANETA
1	0,37	MERCÚRIO
2	0,88	VÊNUS
3	0,38	MARTE
4	2,64	JÚPITER
5	1,15	SATURNO
6	1,17	URANO

Para o cálculo do peso considere a seguinte fórmula:

$$p = m \cdot g$$, sendo que:

- *p*: peso;
- *m*: massa;
- *g*: gravidade.

Calcule e imprima o peso na Terra e no planeta escolhido.

5) A escola "APRENDER" faz o pagamento de seus professores por hora/aula. Faça um programa em Python que calcule e exiba o salário de um professor. Sabe-se que o valor da hora/aula segue a tabela abaixo:

Professor Nível 1	R$12,00 por hora/aula
Professor Nível 2	R$17,00 por hora/aula
Professor Nível 3	R$25,00 por hora/aula

6) Uma empresa concederá um aumento de salário aos seus funcionários, variável de acordo com o cargo, conforme a tabela abaixo. Faça um programa em Python que leia o salário e o cargo de um funcionário e calcule o novo salário. Se o cargo do funcionário não estiver na tabela, ele deverá, então, receber 40% de aumento. Mostre o salário antigo, o novo salário e a diferença salarial.

CÓDIGO	CARGO	PERCENTUAL
101	Gerente	10%
102	Engenheiro	20%
103	Técnico	30%

7) Construa um programa em Python para determinar a situação (APROVADO/EXAME/REPROVADO) de um aluno, dado a sua freqüência (FREQ) (porcentagem de 0 a 100%) e sua nota (NOTA) (nota de 0.0 a 10.0), sendo que:

Condição	Situação
Frequência até 75%	REPROVADO
Frequência entre 75% e 100% e Nota até 3.0	REPROVADO
Frequência entre 75% e 100% e Nota de 3.0 até 7.0	EXAME
Frequência entre 75% e 100% e Nota entre 7.0 e 10.0	APROVADO

Estruturas de repetição

Estruturas de repetição são utilizadas quando existe a necessidade de executar o mesmo bloco de comando um número finito de vezes. Em Python podemos utilizar duas estruturas de repetição: WHILE; e FOR.

A estrutura WHILE

O WHILE executa um bloco de comando N vezes até que sua condição seja satisfeita. Essa estrutura possui a seguinte sintaxe:

```
while condicao:
      comandos
```

onde:

- `while` → palavra-chave que define a estrutura;
- `condicao` → sentença relacional que define a quantidade de vezes que os comandos serão repetidos;
- `comandos` → conjunto de instruções a serem executadas.

Exemplo:

```
lifeRock = 5000;
iron = 0;
batida = 200;
while lifeRock >= 0:
    iron += 5
    lifeRock -= batida
print('Pegou {0} iron'.format(iron))
```

A estrutura FOR

Utilizado para rastrear uma coleção de dados. Sintaxe:

```
for variavel in lista:
      comandos
```

onde:

- `for` → palavra-chave da estrutura;
- `variavel` → variável que receberá cada elemento da lista;
- `lista` → lista de elementos;
- `comandos` → conjunto de instruções.

Exemplo:

```
compras = ['Cupim', 'Picanha', 'Carvão', 'Coração de galinha',
'Pão de alho']
print('Lista de produtos para o churrasco')
for n in compras:
    print(n)
```

Lista de Exercícios C

01) Desenvolva um programa em Python no qual o usuário digitará dois valores inteiros (**A** e **B**). O programa deverá gerar 100 números aleatórios. Cada valor gerado deverá estar dentro do intervalo digitado pelo usuário (**A** e **B**). Imprima o maior valor gerado pelo programa.

02) Desenvolva um programa em Python no qual o usuário digitará o nome e as notas de 35 alunos. Cada aluno possui cinco notas. O programa deverá imprimir a média aritmética de cada aluno e o nome do aluno que possui maior média.

03) Desenvolva um programa em Python no qual o usuário digitará os seguintes dados de um produto: o nome de um alimento; sua categoria (P – Proteína; C – Carboidrato; G – Gordura); e seu valor calórico. O usuário digitará quantos alimentos quiser. A digitação encerrará quando o valor calórico digitado for igual a zero. Imprima o valor calórico total de cada categoria.

04) Desenvolva um programa em Python no qual o usuário digitará os seguintes dados de um produto: o nome de um alimento; sua categoria (P – Proteína; C – Carboidrato; G – Gordura); e seu valor calórico. O usuário digitará quantos alimentos quiser. A digitação encerrará quando o valor calórico digitado for igual a zero. Imprima:

- Qual categoria que possui maior valor calórico;
- Qual o nome do alimento de maior valor calórico.

05) Desenvolva um programa em Python que calcule a velocidade média de 100 veículos. O programa deverá calcular e imprimir a porcentagem de veículos cuja velocidade média ultrapassou a 80 km/h. Segue a fórmula da velocidade média:

$$Vm = \frac{d}{t}$$, sendo que:

- *Vm* → velocidade média;
- *d* → distância;
- *t* → tempo.

06) Desenvolva um programa em Python que simule a jogada de um dado de seis lados 2500 vezes. O programa deverá calcular e imprimir a porcentagens de números: uns; dois; três; quatros; cincos e seis que foram tirados durante as 2500 jogadas.

07) Desenvolva um programa em Python que no qual o usuário digitará a quantidade de Quilowatt-hora (kWh) gasto em uma casa durante 12 meses. O programa deverá apresentar a bandeira de cada mês. Sendo a bandeira definida por:

kWh	Bandeira
Até 172 kWh	Verde
Entre 173 e 280 kWh	Amarela
Mais que 281 kWh	Vermelha

08) Um náufrago encontra-se preso em uma ilha. Ele possui uma quantidade X de comida e Y de bebida. Sabe-se que por dia ele consome 5% da comida total e 3% da bebida total. Desenvolva um programa em Python no qual o usuário digitará o número de comida (X) e bebida (Y) que o náufrago possui inicialmente. O programa deverá apresentar em quantos dias ele demorou para consumir toda comida e bebida.

09) Um cadeado possui 10 fechaduras. Deseja-se criar um programa que verifique quantas tentativas uma pessoa gastará até que abra o cadeado. Desenvolva um programa em Python que gere aleatoriamente a fechadura correta. O usuário deverá escolher a fechadura que deseja abrir até que tenha êxito. Imprima quantas vezes o usuário tentou abrir o cadeado.

Arrays

Um *array* corresponde a uma estrutura de dados do tipo lista que tem como objetivo armazenar uma coleção de dados. O *array* pode armazenar qualquer tipo de valor: caracteres; inteiros; reais. Ela também pode armazenar inclusive outra *array*.

Para definir uma lista, declare uma variável e faça com que ela receba colchetes. Exemplo:

```
x = []
```

Neste momento a variável **x** armazena um *array* sem elementos. Caso seja necessário inicializar a lista com elementos, adicione os valores separados por vírgula. Exemplo:

```
x = [5, 6, 7, 2, 4]
```

O vetor **x** agora possui cinco elementos.

5	6	7	2	4
↑	↑	↑	↑	↑
0	1	2	3	4

Cada elemento possui um índice (*index*) dentro da lista, sendo que o primeiro índice é o zero (0).

Para acessar um elemento da lista, utilize o nome da variável e entre colchetes o índice desejado.

```
x[3]
# Será obtido o valor 2
```

Neste caso será obtido o valor 2 que se encontra no índice 3.

Para substituir um valor, faça com que a variável no índice desejado receba o novo valor.

```
x[2] = 35
```

5	6	35	2	4
↑	↑	↑	↑	↑
0	1	2	3	4

Será atribuído ao índice 2 o valor 35.

Caso seja necessário descobrir quantos elementos existem na lista, utilize a função **len()**.

```
len(x)
```

No caso citado, será retornado o valor 5.

Principais funções do array

Além dos recursos apresentados, a lista (*array*) contém um conjunto de funções que permite o desenvolvedor manipular seus elementos. Neste tópico será apresentado as principais funções de um *array*.

Segue o conjunto de funções:

- **`list.append(el)`** → Adiciona um elemento no final da lista. Exemplo:

```
x.append(69)
```

5	6	35	2	4	69
↑	↑	↑	↑	↑	↑
0	1	2	3	4	5

- **`list.insert(i, x)`** → Adiciona um elemento na posição desejada. Os demais elementos após o índice escolhido, são remanejados à direita da lista. Exemplo:

```
x.insert(2, 24)
```

5	6	24	35	2	4	69
↑	↑	↑	↑	↑	↑	↑
0	1	2	3	4	5	6

- **`list.remove(x)`** → Remove o primeiro elemento da lista cujo valor é citado na função. Exemplo:

```
x.remove(4)
```

5	6	24	35	2	69
↑	↑	↑	↑	↑	↑
0	1	2	3	4	5

- **`list.pop([i])`** → Retira o item da lista de acordo com a posição especificada e retorna o valor removido. Caso não seja especificado o índice, será removido o último item da lista. Exemplo:

```
x.pop(1)
# O valor 6 será retornado. O mesmo será excluído da lista.
```

5	24	35	2	69
↑	↑	↑	↑	↑
0	1	2	3	4

- **`list.index(x[, start[, end]])`** → Retorna o índice do valor desejado. Exemplo:

```
x.index(35)
# O valor retornado será 2
```

- **`list.count(x)`** → Retorna a quantidade de elementos iguais ao valor de entrada. Exemplo:

```
x.count(69)
# O valor retornado será 1
```

- **`list.reverse()`** → Inverte os valores da lista. Exemplo:

```
x.reverse()
```

69	2	35	24	5
↑	↑	↑	↑	↑
0	1	2	3	4

- **`list.sort(key=None, reverse=False)`** → Ordena os elementos da lista em ordem crescente. Se o parâmetro **reverse** receber **True** o método de ordenação ocorre em decrescente. Exemplo:

```
x.sort()
```

2	5	24	35	69
↑	↑	↑	↑	↑
0	1	2	3	4

```
x.sort(reverse=True)
```

69	35	24	5	2
↑	↑	↑	↑	↑
0	1	2	3	4

- **`list.clear()`** → Remove todos os elementos da lista. Exemplo:

```
x.clear()
```


Caso deseje conhecer e aprofundar nas funções disponíveis, acesse o site:

https://docs.python.org/3/tutorial/datastructures.html

Alguns exemplos práticos

Exemplo 01) Faça um programa em Python que carregue uma lista de 15 elementos com valores aleatórios variando entre 100 e 1000. O programa deverá calcular e mostrar:

a) O maior elemento da lista;
b) O menor elemento da lista.

```
import random
x = []
i = 0
while i <= 14:
    x.append(random.randint(100, 1000))
    i += 1
x.sort()
print(x)
print(x[0])
print(x[14])
```

Exemplo 02) Em uma corrida há 10 corredores. Desenvolva um programa em Python que leia os valores do número do corredor e o seu respectivo tempo na corrida. Além disso, o programa deverá imprimir a classificação e o tempo de corrida, do primeiro ao décimo colocado, identificando o número de inscrição do corredor referente àquela colocação.

```
corredor = []
i = 0
while i <= 2:
    inscricao = input('Digite a inscrição: ')
    tempo = int(input('Digite se tempo na corrida: '))
    dados = [inscricao, tempo]
    corredor.append(dados)
    i += 1
corredor.sort(key = lambda corredor:corredor[1])
count = 1
for i in corredor:
```

```
    print(count, '° lugar: ', i[0], ' -> Tempo: ', i[1], sep='')
    count += 1
```

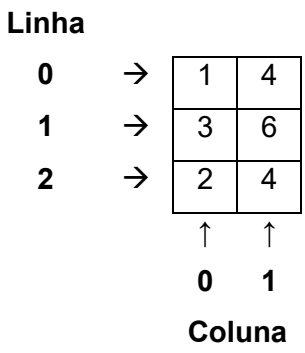
Exemplo 03) Desenvolva um programa em Python no qual o usuário digitará uma palavra. Imprima a palavra original e invertida. Compare se a palavra inversa é igual ao original.

```
palavra = input('Digite uma palavra: ')
x = list(palavra)
x.reverse()
palavraInvertida = ''.join(x)
print('Palavra: ', palavra)
print('Palavra invertida: ', palavraInvertida)
if palavra == palavraInvertida:
    print('PALAVRAS IGUAIS')
else:
    print('PALAVRAS DISTINTAS')
```

Arrays multidimensionais

Arrays multidimensionais, também conhecida como matriz, corresponde a uma lista que recebe uma coleção de dados. Essa coleção de dados é representada por outro *array*. Exemplo:

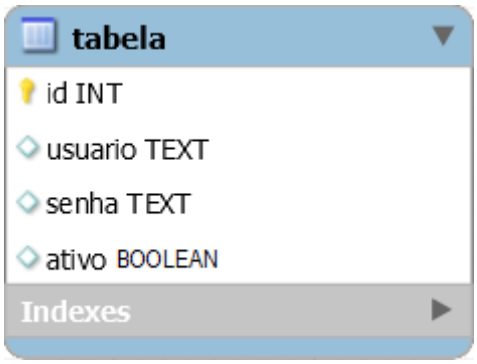
```
x = [[1, 4], [3, 6], [2, 4]]
```



Para acessar um dado desta estrutura utilize dois colchetes adicionando o endereço da linha e coluna.

```
x[1][0]
'''
O valor retornado será 3 que está presente na linha 1 e coluna 0
'''
```

O desenvolvedor também poderá criar um *array* multidimensional com dados de diversos tipos. Considere a imagem a seguir:



Considere que na imagem exibida, a lista **tabela** deverá possuir quatro colunas por dado. Cada coluna possui uma definição e um tipo de dado:

- id → receberá um inteiro correspondendo ao código de um usuário;
- usuário → receberá um texto correspondendo ao nome da conta de um usuário;
- senha → receberá um texto correspondendo a senha da conta do usuário;
- ativo → receberá um boolean correspondendo se a conta do usuário vigente encontra-se ativa.

Para o caso citado, pode-se criar um *array* multidimensional da seguinte forma:

```
tabela = [
    [69,  'denecley@gmail.com', '******', True],
    [124, 'pepy@hotmail.com', 'pepylegal', True],
    [548, 'pedro.paulo@hotmail.com', '123321', False],
    [96,  'primopato@gmail.com', 'primim2018', True],
    [56,  'eng.let@gmail.com', 'alunos2019', False]
]
```

Linha					
0	→	69	denecley@gmail.com	******	True
1	→	124	pepy@hotmail.com	pepylegal	True
2	→	548	pedro.paulo@hotmail.com	123321	False
3	→	96	primopato@gmail.com	primim2018	True
4	→	56	eng.let@gmail.com	alunos2019	False
		↑	↑	↑	↑
		0	**1**	**2**	**3**

Coluna

E se quiséssemos listar todas as contas desativadas?

Para a pergunta acima podemos resolver o problema da seguinte forma:

```
for linha in tabela:
    if not linha[3]:
        print(linha)
```

ou

```
list(filter(lambda linha: not linha[3], tabela))
```

O comando *list* converte uma série de dados em uma lista, *array*. O comando *filter* retira todos os dados de uma lista a partir de um critério de filtro. Para utilizar o filtro use a sintaxe:

```
lambda valor: condicao
```

onde:

- `lambda` → `palavra-chave utilizada para criar uma variável de filtro;`
- `valor` → `nome da variável de filtro. Essa variável receberá cada valor da lista;`
- `condicao` → `critério de seleção.`

Agora, como podemos capturar todas as contas do *gmail*?

```
list(filter(lambda linha: 'gmail' in linha[1], tabela))
```

Lista de exercícios D

1) Crie um programa em Python que leia uma lista de 40 posições. Contar e escrever quantos valores pares ele possui.

2) Crie um programa em Python que leia duas listas de 20 posições e calcule uma outra lista contendo, nas posições pares os valores do primeiro e nas posições impares os valores do segundo.

3) Faça um programa em Python que determine quantos números maior que 100 há em uma lista de inteiros com 20 elementos. Preencha a lista com números aleatórios variando entre 1 a 200.

4) Crie um programa em Python que leia a pontuação de 10 provas de um concurso e os nomes dos respectivos participantes. Grave os dados em uma lista. Apresente um ranking dos colocados que obtiveram mais de 70 pontos.

5) Crie um programa em Python que leia quinze números do usuário e armazene em uma lista a raiz quadrada de cada número. Caso o valor digitado seja menor que zero o número –1 deve ser atribuído ao elemento da lista. Após isso, o programa deve imprimir todos os valores armazenados.

6) Crie um programa em Python que que receba a altura de 10 atletas. Armazene a altura de todos os atletas em uma lista. Esse programa deve imprimir a altura daqueles atletas que tem altura maior que a média.

7) Escrever um programa em Python que contenha uma lista G(13) que é o gabarito de um teste de loteria esportiva, contendo os valores: A, B ou C para cada posição de G. Ler, a seguir, o número do cartão de um apostador e outra lista resposta R (13). Verificar o número de acertos e escrever o número do apostador e seu número de acertos. Se tiver 13 acertos, acrescentar a mensagem: "GANHADOR, PARABENS".

8) Faça um programa em Python que:

a) Leia uma frase;
b) Conte quantos brancos existem na frase;
c) Conte quantas vezes a letra A aparece;
d) Imprima o que foi calculado nos itens b e c.

Dica: Acesse o site

https://docs.python.org/2/library/string.html

9) Faça um programa em Python que, dados duas listas A e B de 20 elementos, efetue as respectivas operações indicadas por uma outra lista C de 20 elementos, também fornecido pelo usuário, contendo as quatro operações aritméticas em qualquer combinação. Armazene em

uma lista resultado o valor da operação realizada. Por exemplo, suponha que A[1] = 10, B[1] = 3 e C[1] = '*', então D[1] = 30.

10) Faça um programa em Python que leia uma lista de 3 elementos e uma lista multidimensional de 3 x 3 elementos. Em seguida o programa deve fazer a multiplicação da lista simples pelas colunas da lista multidimensional. Exiba a nova lista.

11) Crie um programa em Python no qual o usuário digitará uma matriz com dimensões M × N. Para a matriz digitada, imprima na tela os números pares contidos na matriz, considerando separá-los por linha. Veja o exemplo para uma matriz 4× 4:

Entrada:

1	2	3	4
5	6	7	8
10	11	23	21
1	2	3	1

Saída:

2 4 6 8 10 2

12) Crie um programa em Python que leia uma matriz quadrada 10x10, de números inteiros. Verificar quantos números da diagonal principal são ímpares, escrever também, a soma e a média desses elementos.

13) Implemente um programa, utilizando a Linguagem Python, que leia 30 nomes e sobrenomes de pessoas. A cada nome e sobrenome lido, verifique se as palavras possuem o mesmo tamanho.

- Se o nome e o sobrenome possuírem o mesmo tamanho, concatene as palavras e escreva mensagem mostrando o nome completo;
- Se o nome e o sobrenome possuírem tamanhos diferentes, escreva uma mensagem mostrando que não possuem o mesmo tamanho.

14) Fazer um programa em Python que leia ao final da manhã o fechamento do caixa de uma loja, ou seja, o seu rendimento ao final da manhã. O mesmo deverá ser feito ao final da tarde. Este levantamento deve ser feito todos os dias da semana (de segunda-feira a sexta-feira). Ao final da semana, após feitas todas as leituras, descobrir e escrever o dia e o turno que teve maior rendimento. Obs.: Utilizar o conceito de matriz para resolver este exercício.

Funções

Neste capítulo aprenderemos o que é e qual a finalidade de uma função. Considere a seguinte situação:

```
i = 0
while i < 3:
    print('Escolha três números')
    n1 = int(input('Número 1: '))
    j = 1
    fat = 1
    while j <= n1:
        fat *= j
        j += 1
    print('Fatorial: {0}'.format(fat))
    n2 = int(input('Número 2: '))
    j = 1
    fat = 1
    while j <= n2:
        fat *= j
        j += 1
    print('Fatorial: {0}'.format(fat))
    n3 = int(input('Número 3: '))
    j = 1
    fat = 1
    while j <= n3:
        fat *= j
        j += 1
    print('Fatorial: {0}'.format(fat))
    i += 1
```

Neste exemplo o desenvolvedor pede ao usuário que digite três números três vezes. Para cada número é calculado seu fatorial (exemplo: *4! = 4 x 3 x 2 x 1*) e impresso na tela. Repare que este código possui uma repetição das seguintes instruções:

```
j = 1
fat = 1
while j <= n1:
   fat *= j
   j += 1
print('Fatorial: {0}'.format(fat))
```

E se o desenvolvedor quisesse calcular o fatorial de 10 números 3 vezes? Consegue imaginar a quantidade de linhas e códigos repetidos que seriam gerados?

Para evitar situações como essas as linguagens de programação oferecem um recurso denominado função.

A função corresponde a um bloco de código que poderá ser invocado quantas vezes forem necessárias. Ela define um problema. O uso de funções tem como objetivo:

- Minimizar a quantidade de código escrito;
- Facilitar a compreensão do código-fonte;
- Distribuir a resolução do problema em pequenos módulos.

Então como ficaria o código acima utilizando tal recurso?

```
def fatorial(n):
    j = 1
    fat = 1
    while j <= n:
        fat *= j
        j += 1
    print('Fatorial: {0}'.format(fat))

i = 0
while i < 3:
    print('Escolha três números')
    n1 = int(input('Número 1: '))
    fatorial(n1)
    n2 = int(input('Número 2: '))
    fatorial(n2)
```

```
    n3 = int(input('Número 3: '))
    fatorial(n3)
    i += 1
```

No primeiro cenário o código escrito possuía 25 linhas, com a aplicação de função houve a redução para 17 linhas, havendo uma economia de 32%. Repare também que o trecho:

```
j = 1
fat = 1
while j <= n1:
   fat *= j
   j += 1
print('Fatorial: {0}'.format(fat))
```

foi escrito apenas uma vez, dentro da função ***fatorial***. No código principal a função foi chamada pelo seu nome e o valor de entrada entre parênteses.

```
fatorial(n1)
fatorial(n2)
fatorial(n3)
```

Características

A função possui as seguintes características:

- Só pode ser invocada após sua criação;
- Pode receber ou não valores externos, parâmetros;
- Pode retornar ou não um resultado. Quando deseja-se retornar um valor é necessário utilizar o comando **return**.

Para criar a função utilize esta sintaxe:

```
def nomeFuncao(parametros):
      comandos
      return valor
```

onde:

- def → comando utilizado para definir uma função;
- nomeFuncao → o nome da função. Evite caracteres especiais e espaço;
- parametros → define a entrada de valores externos. Se a função não tiver, basta abrir e fechar parênteses. Caso tenha mais de um, será necessário separá-los por vírgula;
- comandos → conjunto de instruções de visam resolver o problema;
- return → comando utilizado para retornar um valor. O return permite retornar um único valor;
- valor → valor a ser retornado.

É importante ressaltar que um código poderá possuir quantas funções forem necessárias.

Exemplo 01

Desenvolva uma função em Python que imprima uma quantidade ***x*** de linhas:

```
*
**
***
****
*****
```

O usuário deverá escolher quantas linhas deseja imprimir no formato apresentado.

Resolução:

```
def impressao(linhas):
    for i in range(1, linhas+1):
        print('*' * i)
```

Exemplo 02

Desenvolva uma função em Python que receba uma lista contendo o nome e sobrenome. Esta função deverá retornar outra lista contendo o nome completo das pessoas que iniciam com ‘A’.

Resolução:

```
def filtro(lista):
    f = list(filter(lambda item : item[0][:1] == 'A', lista))
    r = []
    for i in f:
        r.append(i[0] + ' ' + i[1])
    return r
```

Exemplo 03

Desenvolva uma função em Python que recebe uma lista de pedidos. Cada elemento possui um dos seguintes valores:

CREME	NAPOLITANO	CHOCOLATE	FLOCOS

a função deverá retornar o nome do pedido mais frequente.

Resolução:

```
def pedido(lista):
    itens = ['CREME', 'CHOCOLATE', 'FLOCOS', 'NAPOLITANO']
    maior = 0
    sabor = ''
    for item in itens:
        q = len(list(filter(lambda i : i == item, lista)))
        if q >= maior:
            maior = q
            sabor = item
    return sabor
```

Exemplo 04

Desenvolva uma função em Python retorne os elementos da diagonal principal de uma matriz de ordem *N*.

Resolução:

```
def dPrincipal(matriz):
    index = 0
    diagonal = []
    for linha in matriz:
        diagonal.append(linha[index])
        index += 1
    return diagonal
```

Exemplo 05

Desenvolva uma função em Python que receba um número correspondente a quantidade de jogos da MEGA SENA a serem gerados. A função deverá retornar todos os palpites sorteados pelo computador. Uma cartela contém números de 1 a 60 e uma jogada possui 6 números.

Resolução:

```
def cartelas(quantidade):
    saida = []
    for i in range(1, quantidade + 1):
        jogo = []
        count = 0
        while True:
            num = random.randint(1, 60)
            if num not in jogo:
                jogo.append(num)
                count += 1
            if count == 6:
                break
        jogo.sort()
```

```
            saida.append(jogo)
    return saida
```

Lista de exercícios E

1) Desenvolva uma função em Python que receba um número inteiro. A função deverá retornar um boolean indicando: true se o número for primo; e false se o número não for primo. Utilize a seguinte assinatura.

*def **isPrimo**(numero)*

2) Desenvolva uma função em Python que receba uma matriz de texto e mais um atributo também do tipo texto. A função deverá retornar uma lista contendo todos os textos que contenha o valor de entrada. Segue assinatura do método:

*def **buscar**(valores, entrada)*

Exemplo:

valores:

Cachorro velho não aprende truque novo	Meu sapo jururu
Super Mario Brother	Show no rio

entrada: "rio"

Retorno:

Super Mario Brother	Show no rio

3) Desenvolva uma função em Python que receba quatro inteiros: o primeiro indica a quantidade de linhas; o segundo indica a quantidade de colunas; o terceiro o intervalo inicial; e o quarto o intervalo final. A função deverá retornar uma matriz de acordo com o tamanho especificado e preenchida com o valores aleatórios de acordo com o intervalo estipulado. Segue assinatura da função:

*def **matrizAleatoria**(linha, coluna, intervalInicial, intervalFinal)*

Exemplo:

Entrada:

matrizAleatoria(2, 3, 30, 60)

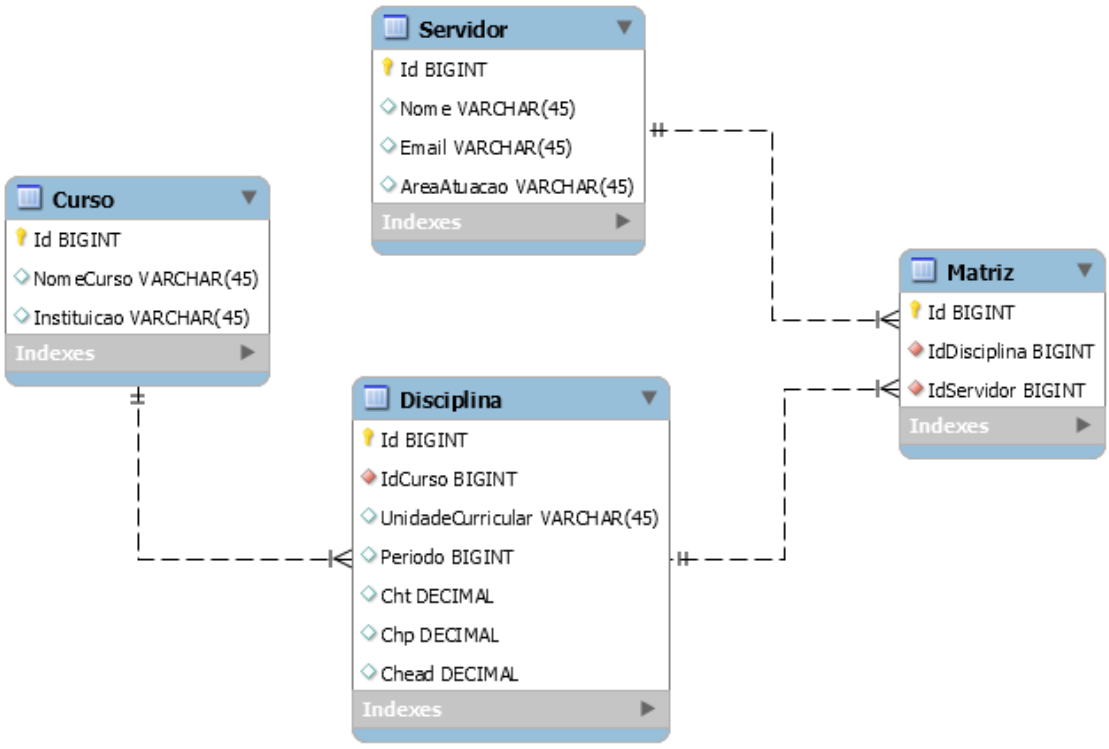
Saída:

45	32	56
31	50	41

Estrutura de Dados

Neste capítulo aprenderemos como montar um modelo conceitual a partir de uma estrutura de dados. Também veremos o significado de dicionário de dados e como podemos fazer para gerenciar esse dicionário.

Considere a seguinte modelagem de dados:



No modelo apresentado, os dados encontram-se distribuídos nas seguintes tabelas: Curso; Servidor; Disciplina; e Matriz.

- ***Curso*** → A tabela Curso é responsável por guardar o nome de qualquer curso referente a qualquer instituição de ensino;
- ***Servidor*** → A tabela Servidor é responsável por guardar as informações dos servidores que atuam como professores. Cada servidor possui um nome, um e-mail e sua área de atuação;
- ***Disciplina*** → A tabela Disciplina é responsável por guardar as informações de todas as disciplinas ofertadas em um curso. Cada disciplina possui o curso ao qual é ofertada, seu nome, o período de oferta, a carga horária total, a carga horária presencial e sua carga horária à distância;

- ***Matriz*** → A tabela Matriz é responsável por relacional cada disciplina com seu devido professor.

Criaremos uma estrutura de dados que permitirá ao desenvolvedor acessar o modelo relacional apresentado. Os arquivos que contém o modelo e os dados estão separados em duas categorias: ***data*** para arquivos *csv* que armazenam os dados; e meta-data para os arquivos *txt* que armazenam a lógica estrutural de como os dados são organizados.

Por exemplo a tabela Curso possui sua estrutura armazenada em *data/meta-data/Curso.txt*. Esse arquivo define as características da tabela Curso:

```
Id    bigint.
NomeCurso  varchar.
Instituicao     varchar.
```

O arquivo que possui os dados encontra-se em *data/data/Curso.csv*. Segue o formato dos dados:

```
1;Engenharia Elétrica;IF Goiano – Campus Trindade
2;Engenharia Civil;IF Goiano – Campus Trindade
3;Engenharia da Computação;IF Goiano – Campus Trindade
4;Pós-graduação em Humanidades;IF Goiano – Campus Trindade
```

Cada dado é separado por ponto-e-vírgula (;), o que corresponde a uma coluna da tabela construída.

Para começarmos o trabalho, o passo inicial é encontrar as estruturas de dados de cada tabela e criarmos um dicionário. O dicionário a ser criado será baseado nos arquivos que definem a estrutura e que estão guardados na pasta *meta-dados*.

Dicionário de dados

Também conhecido como *mapa* ou *array associativo*, o dicionário de dados tem como função receber uma estrutura de dados abstrata e relacioná-lo nas diversas entradas de dados.

Para criar um dicionário de dados obedeça uma das seguintes sintaxes.

```
dicionario = {
    'Curso': []
}
```

Onde:

- `dicionario` → `nome do dicionário de dados;`
- `curso` → `nome da estrutura da tabela a ser buscada.`

Também pode ser escrito da seguinte forma:

```
dicionario = dict(Curso = [])
```

ou

```
dicionario = dict()
dicionario['Curso'] = []
```

Conhecendo o modo de construção, precisaremos agora criar um dicionário que recebe todas as informações dos arquivos contidos no caminho *meta-dados*. O dicionário deverá ser construído no seguinte formato:

```
dicionario = {
    'Curso': [
        ('Id', 'bigint'),
        ('NomeCurso', 'varchar'),
        ('Instituicao', 'varchar')
    ]
}
```

A chave (*key*) será composta pelos nomes das entidades (*Curso.txt, Matriz.txt, Servidor.txt, Disciplina.txt*) e seu conteúdo pelos campos que definem cada entidade.

Para montar o dicionário dinamicamente, precisaremos carregar todos os arquivos contidos na pasta *meta-data*. Para capturar o nome dos arquivos contidos em uma pasta, utilize o seguinte código-fonte:

```
import os
```

```
for file in os.listdir('data/meta-data'):
    print(file)
```

O código apresentado importa um módulo denominado '***os'***. Este módulo permitirá carregar os arquivos contidos em uma pasta através da função ***listdir***, que retorna a lista contendo o nome dos arquivos. A saída esperada para esse código será:

```
Curso.txt
Disciplina.txt
Matriz.txt
Servidor.txt
```

Embora tenhamos os nomes dos arquivos contidos em *meta-data*, repare que estes possuem o nome com sua extensão. Precisamos retirar a extensão ***.txt***. Podemos retirar através da seguinte função:

```
def retirarExtensao(arquivo):
    return arquivo.split('.')[0]
```

Tendo o caminho e nome dos arquivos, precisamos agora criar uma função que permita carregar todo conteúdo. Criaremos então a seguinte função:

```
def lerMetaData(arquivo):
    dados = open('data/meta-data/' + arquivo, 'rt')
    metaData = []
    for linha in dados:
        i = linha.split('\t')
        metaData.append((i[0], i[1].split('.')[0]))
    dados.close()
    return metaData
```

A função open abre o arquivo, nela deve ser encaminhado o caminho do arquivo e o tipo de abertura, no caso é aberto apenas para leitura ***rt***. Um arquivo pode ser aberto de acordo com os caracteres:

- 'r' abre para leitura (padrão);

- 'w' abre para escrita;
- 'x' abre para criação. Se já existir um arquivo com mesmo nome, ocorrerá um erro;
- 'a' abre para escrita, adicionando o conteúdo para o fim do arquivo se existir;
- 'b' modo binário;
- 't' modo texto (padrão);
- '+' abre o arquivo de disco atualizando (lendo e escrevendo);
- 'U' modo nova linha (desatualizado).

A variável data recebe todo conteúdo do arquivo na função *open*. Ao encerrá-lo é necessário utilizar a função *close()*.

Juntando todo código produzido chegamos no seguinte resultado:

```
import os

def retirarExtensao(arquivo):
    return arquivo.split('.')[0]

def lerMetaData(arquivo):
    dados = open('data/meta-data/' + arquivo, 'rt')
    metaData = []
    for linha in dados:
        i = linha.split('\t')
        metaData.append((i[0], i[1].split('.')[0]))
    dados.close()
    return metaData

dicionario = {}
for file in os.listdir('data/meta-data'):
    tabela = retirarExtensao(file)
    dicionario[tabela] = lerMetaData(file)

for chave, valor in dicionario.items():
    print('Entidade: {}'.format(chave))
    print('Atributos: ', valor)
```

As informações contidas nesta entidade é definida pelo nome de tupla. Uma tupla é criada com parênteses e ao contrário da lista (colchetes), a tupla é imutável, isso significa que uma vez que o conteúdo do definido, não será possível alterá-lo.

```
Entidade: Curso
Atributos: [
      ('Id', 'bigint'),
      ('NomeCurso', 'varchar'),
      ('Instituicao', 'varchar')
]
```

No caso de *Curso* temos três tuplas. No nosso exemplo, uma tupla é constituída de dois dados: nome do campo; e tipo do campo.

Exemplo: Relacionando meta-dados com dados

Neste exemplo relacionaremos o meta-dados *Curso* com seus dados.

```
# Lê todos os meta-dados do arquivo
# Retorna uma lista
def lerMetaData(arquivo):
    dados = open('data/meta-data/' + arquivo, 'rt')
    metaData = []
    for linha in dados:
        i = linha.split('\t')
        metaData.append((i[0], i[1].split('.')[0]))
    dados.close()
    return metaData

# Separa uma linha de dados do CSV
# Retorna os dados no formato de lista
def lerDado(tupla, linha):
    linha = []
    for l in range(0, len(tupla)):
        linha.append(i.split(';')[l])
    return linha
```

```
# Cria o dicionário e extrai as informações
# em uma tupla
dicionario = {}
dicionario['Curso'] = lerMetaData('Curso.txt')
tupla = []
for t in dicionario['Curso']:
    tupla.append(t[0])

# Relaciona os dados com as tuplas
dados = []
d = open('data/data/Curso.csv', 'rt')
for i in d:
    linha = lerDado(tupla, i)
    # Cria um dicionário para cada linha de dado
    x = {}
    for key, value in zip(tupla, linha):
        x[key] = value
    dados.append(x)

d.close()
```

Lista de exercícios F

1) Desenvolva um código Python que crie as tuplas da entidade Servidor. Relacione cada tupla com seus dados.

2) Desenvolva um código Python que crie as tuplas da entidade Disciplina. Relacione cada tupla com seus dados.

3) Com base no exercício 2, desenvolva um código Python que visualize apenas as disciplinas do 1º e 3º período.

4) Desenvolva um código Python que exiba os seguintes dados:

Nome do Curso	Unidade Curricular	Período	Carga Horária Total
NomeCurso	UnidadeCurricular	Periodo	Cht

Neste exercício será necessário relacionar as entidades Curso e Disciplina.

5) Crie o seguinte aplicativo em Python:

Escolha uma opção:

1) Listar cursos

2) Listar disciplinas

3) Listar servidores

4) Listar disciplina por período

5) Listar disciplina por curso

6) Listar servidor por nome

7) Sair

Python e SQLite

Neste capítulo aprenderemos como conectar a linguagem Python com banco de dados SQLite. Veremos como realizar as principais rotinas de manipulação em banco de dados: consulta; inserção; atualização; e remoção.

Um Sistema de Gerenciamento de Banco de Dados (SGDB) trata-se de uma estrutura utilizada para armazenar dados. Além disso, ela permite que esses dados sejam armazenados de acordo com um modelo. O SGDB também disponibiliza recursos como integridade e segurança a base de dados.

Para conectarmos a linguagem de programação com o banco de dados, não será necessário importar bibliotecas externas. A linguagem Python já possui um recurso de conexão nativa com o banco de dados SQLite.

Para testarmos a conexão com banco de dados, crie um projeto chamado ***BancoDados***. No projeto criado construa um ***Python File*** e nomeie como ***conexao.py***.

Baixe o arquivo ***database.db*** presente na disciplina do Ambiente Virtual (Moodle). Coloque este arquivo na mesma pasta que ***conexao.py***. Digite o seguinte código:

```
01  import sqlite3
02  conn = sqlite3.connect('database.db')
03  cursor = conn.cursor()
04  cursor.execute('SELECT * FROM produto')
05  print(cursor.fetchall())
06  conn.close()
```

Linha 01: Importa o recurso responsável por conectar a linguagem com banco de dados;
Linha 02: O comando ***connect*** realiza a conexão com o banco de dados selecionado. Esta função recebe apenas o nome do banco de dados, caso o arquivo esteja junto com o ***.py***. Se o banco de dados estiver em outra pasta, será necessário passar todo caminho. Exemplo: *c:\pasta\subpasta\banco.db*;
Linhas 03 a 05: Veremos o significado desses comandos nos próximos tópicos;
Linha 06: O comando ***close*** finaliza a conexão. Necessário para fechar o banco de dados.

Manipulando o banco de dados

Neste tópico aprenderemos como manipular dados dentro do banco. Para isso considere o modelo a seguir.

No modelo apresentado temos três tabelas: produto; compra; compra_produto. Cada tabela possui a seguinte função:

- **produto:** armazena todos os produtos. Registra-se a descrição do produto e seu preço unitário;
- **compra:** armazena todas as compras realizadas. Registra-se a data de realização da compra e o local em que foi realizada a compra;
- **compra_produto:** armazena todos os produtos que foram comprados em uma determinada compra. Registra-se o código da compra, o código do produto e quantos produtos foram comprados (quantidade).

Comando INSERT

O comando INSERT é utilizado quando deseja-se inserir um dado no banco de dados. O INSERT possui a seguinte sintaxe:

INSERT INTO tabela (campos) VALUES (dados);

Onde:

- INSERT INTO: comando da instrução;
- tabela: nome da tabela que receberá o dado;
- campos: nome dos campos separados por vírgula (,);
- VALUES: comando da instrução;

- dados: dados relativo a cada campo. Cada dado deverá ser separado por vírgula (,). Dados do tipo texto devem estar entre aspas simples.

Exemplo 01: Deseja-se inserir na tabela produto os seguintes dados:

Descrição	Preço unitário
Cupim Bovino	12.89
Costelinha Suína Sadia ou Perdigão	10.99

Os comandos seriam:

```
INSERT INTO produto (descricao, precounit) VALUES (‘Cupim Bovino’, 12.89);

INSERT INTO produto (descricao, precounit) VALUES (‘Costelinha Suína Sadia ou Perdigão’, 10.99);
```

Para cada conjunto de dados foi realizado um INSERT. No entanto também é possível inserir da seguinte forma:

```
INSERT INTO produto (descricao, precounit) VALUES
(‘Cupim Bovino’, 12.89),
(‘Costelinha Suína Sadia ou Perdigão’, 10.99);
```

Para realizarmos o comando INSERT dentro da linguagem Python, teremos que escrever o seguinte código:

```
01  import sqlite3
02  conn = sqlite3.connect('database.db')
03  cursor = conn.cursor()
04  cursor.execute(
05  'INSERT INTO produto (descricao, precounit) VALUES (?, ?), (?, ?)',
06  ('Cupim Bovino', 12.89, 'Costelinha Suína Sadia ou Perdigão', 10.99))
07  conn.commit()
08  conn.close()
```

Linha 03: A função ***cursor*** é responsável por realizar qualquer manipulação com o banco de dados. Sempre que houver a necessidade de trabalhar com uma tabela, será necessário chamar esta função;
Linhas 04, 05 e 06: O comando ***execute*** recebe qual comando deverá ser executado. Neste caso executaremos o seguinte comando:

```
'INSERT INTO produto (descricao, precounit) VALUES (?, ?), (?, ?)'
```

O símbolo de interrogação (?) denota que será necessário passar um parâmetro como referência. No caso será necessário passar quatro parâmetros:

```
('Cupim Bovino', 12.89, 'Costelinha Suína Sadia ou Perdigão', 10.99)
```

```
 'Cupim Bovino'   12.89      'Costelinha Suína Sadia ou Perdigão'      10.99
      ↑           ↑                        ↑                         ↑
      01          02                       03                        04
```

Linha 07: O comando ***commit*** grava a instrução passada em ***execute***. Este comando é necessário para efetuar a operação no banco.

Exemplo 02: Deseja-se inserir na tabela produto o seguinte dado:

Descrição	Preço unitário
Filé de Tilápia Geneseas	7.79

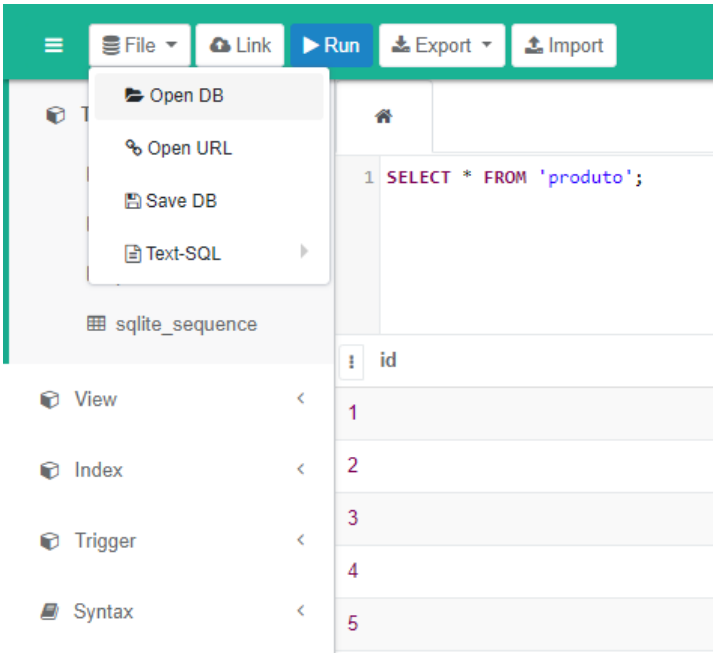
```
INSERT INTO produto (descricao, precounit) VALUES ('Filé de Tilápia Geneseas', 7.79)
```

```
01  import sqlite3
02  conn = sqlite3.connect('database.db')
03  cursor = conn.cursor()
04  cursor.execute(
05  'INSERT INTO produto (descricao, precounit) VALUES (?, ?)',
06  ('Filé de Tilápia Geneseas', 7.79))
07  conn.commit()
08  conn.close()
```

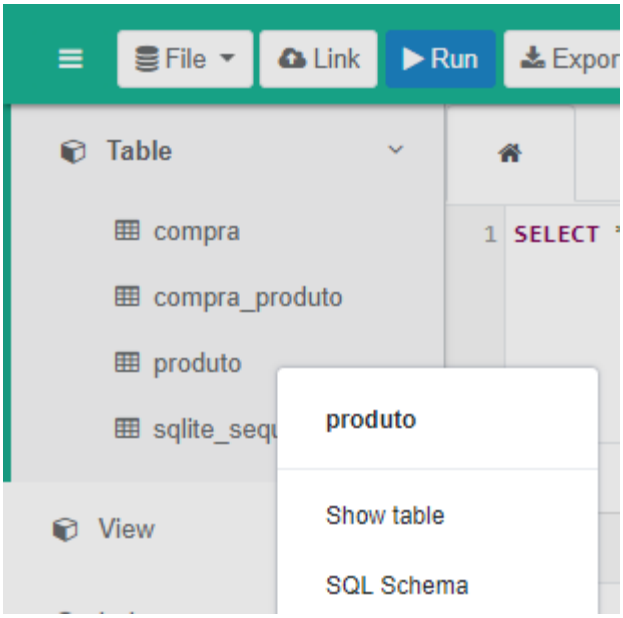
Para testar se os dados estão sendo armazenados em ***database.db***, utilize o seguinte site:

https://sqliteonline.com/

Para abrir o banco clique em ***File → Open DB***.



Clique com botão direito em ***produto*** e escolha ***Show table***.



Comando UPDATE

O comando UPDATE é responsável por editar um dado que já existe na tabela. Ele possui a seguinte sintaxe:

UPDATE tabela SET campo = novoValor WHERE campoChave = valor

Onde:

- UPDATE SET: comando da instrução;
- tabela: nome da tabela a ser utilizada;
- campo: nome do campo a ser editado. Se houver mais campos a serem alterados utilize vírgula (,) para separá-los;
- novoValor: novo valor do campo;
- WHERE: condição de atualização;
- campoChave: código da tabela;
- valor: valor do código.

Exemplo 03: Mude o preço unitário do produto:

Id	Descrição	Preço unitário
21	Cupim Bovino	12.89

para R$ 21.99.

```
UPDATE produto SET precounit = 21.99 WHERE id = 21
```

Para realizarmos o comando UPDATE dentro da linguagem Python, teremos que escrever o seguinte código:

```
01  import sqlite3
02  conn = sqlite3.connect('database.db')
03  cursor = conn.cursor()
04  cursor.execute(
05      'UPDATE produto SET precounit = ? WHERE id = ?', (21.99, 21))
06  conn.commit()
07  conn.close()
```

Exemplo 04: Mude a descrição e o preço unitário do produto:

Id	Descrição	Preço unitário
23	Filé de Tilápia Geneseas	7.79

para

Id	Descrição	Preço unitário
23	Camarão DellMare	8.99

```
UPDATE produto SET descricao = 'Camarão DellMare', precounit = 8.99 WHERE id = 23
```

```
01  import sqlite3
02  conn = sqlite3.connect('database.db')
03  cursor = conn.cursor()
04  cursor.execute(
05      'UPDATE produto SET descricao = ?, precounit = ? WHERE id = ?',
06  ('Camarão DellMare', 8.99, 23))
07  conn.commit()
08  conn.close()
```

Comando DELETE

O comando DELETE é utilizado para remover um dado de uma tabela do banco de dados. Ele possui a seguinte sintaxe:

DELETE FROM tabela WHERE campoChave = valor

Onde:

- DELETE FROM: comando da instrução;
- tabela: nome da tabela a ser manipulada;
- WHERE: condição de remoção;
- campoChave: código da tabela;
- valor: valor do código.

Exemplo: Delete o registro 23.

```
DELETE FROM produto WHERE id = 23
```

Para realizarmos o comando DELETE dentro da linguagem Python, teremos que escrever o seguinte código:

```
01  import sqlite3
02  conn = sqlite3.connect('database.db')
03  cursor = conn.cursor()
04  cursor.execute('DELETE FROM produto WHERE id = ?', (23,))
05  conn.commit()
06  conn.close()
```

Comando SELECT

O comando SELECT é utilizado na realização de consultas as tabelas do banco de dados. Ele possui a seguinte sintaxe básica:

SELECT campos FROM tabelas;

Onde:

- SELECT: seleciona os campos a serem exibidos;
- campos: campos escolhidos para exibição;
- FROM: seleciona as tabelas utilizadas;
- tabelas: tabelas escolhidas.

Exemplo 05: Visualize a descrição de todos os produtos cadastrados.

```
SELECT descricao FROM produto
```

```
01  import sqlite3
02  conn = sqlite3.connect('database.db')
03  cursor = conn.cursor()
04  cursor.execute('SELECT descricao FROM produto')
05  for linha in cursor.fetchall():
06    print(linha[0])
07  conn.commit()
08  conn.close()
```

Exemplo 06: Visualize todos os dados da tabela produto.

```
SELECT * FROM produto
```

```
01  import sqlite3
02  conn = sqlite3.connect('database.db')
03  cursor = conn.cursor()
04  cursor.execute('SELECT * FROM produto')
05  for linha in cursor.fetchall():
06    print('{0} - R$ {1}'.format(linha[1], linha[2]))
07  conn.commit()
08  conn.close()
```

Para visualizar todos os campos de um tabela utilize o símbolo asterisco (*). No caso será emitido uma lista contendo: (id, descricao, precounit).

Posição 0	id
Posição 1	drescricao
Posição 2	precounit

Comando WHERE

O comando WHERE é utilizado para testar uma determinada condição. Ele pode ser construído dentro dos comandos: INSERT; UPDATE; ou WHERE.

Exemplo 07: Exiba todos os produtos cujo o preço unitário é maior que R$ 500,00.

```
SELECT * FROM produto WHERE precounit > 500
```

```
01  import sqlite3
02  conn = sqlite3.connect('database.db')
03  cursor = conn.cursor()
04  cursor.execute('SELECT * FROM produto WHERE precounit > 500')
05  for linha in cursor.fetchall():
06    print('{0} - R$ {1}'.format(linha[1], linha[2]))
07  conn.commit()
08  conn.close()
```

Exemplo 08: Exiba todos os produtos cuja descrição inicia com a letra ‘F’.

```
SELECT * FROM produto WHERE descricao like 'F%'
```

```
01  import sqlite3
02  conn = sqlite3.connect('database.db')
03  cursor = conn.cursor()
04  cursor.execute("SELECT * FROM produto WHERE descricao LIKE ?", ('F%',))
05  for linha in cursor.fetchall():
06    print('{0} - R$ {1}'.format(linha[1], linha[2]))
07  conn.commit()
08  conn.close()
```

Trabalhando com mais de uma tabela

Vamos supor que deseja-se exibir todos os produtos realizados em uma determinada compra. Neste caso precisaríamos:

- da tabela compra para identificar qual a compra realizada;
- da tabela compra_produto para identificar quais são os produtos daquela compra; e
- da tabela produto para exibir a descrição.

O código seria o seguinte:

```
SELECT
      descricao
FROM
      produto, compra, compra_produto
WHERE
      compra.id = compra_produto.idcompra AND
      produto.id = compra_produto.id AND
      compra.id = 1
```

Interpretando o código acima, selecione a descrição do produto da compra cujo id é igual a 1. Precisamos relacionar as tabelas, e para isso é necessário comparar os campos chaves da tabela principal com suas respetivas referências:

```
compra.id = compra_produto.idcompra AND
produto.id = compra_produto.id
```

Segue o código em Python que resolveria esse problema.

```
01  import sqlite3
02  conn = sqlite3.connect('database.db')
03  cursor = conn.cursor()
04  cursor.execute(
05  'SELECT descricao FROM produto, compra, compra_produto ' +
06  'WHERE compra.id = compra_produto.idcompra AND ' +
07  'produto.id = compra_produto.id AND compra.id = 1')
08  for linha in cursor.fetchall():
09    print('{0}'.format(linha[0]))
10  conn.commit()
11  conn.close()
```

Exemplo 09: Apresente o valor total a ser pago da compra cujo id é igual a 1.

Para pegarmos o total de uma compra, será necessário multiplicarmos as colunas ***precounit*** e ***quantidade*** de cada item. Para isso precisamos realizar a seguinte consulta:

```
SELECT
      Descricao, precounit, quantidade
FROM
      produto, compra, compra_produto
WHERE
      compra.id = compra_produto.idcompra AND
      produto.id = compra_produto.id AND
      compra.id = 1
```

A seguir temos o código que resolve o problema.

```
01  import sqlite3
02  conn = sqlite3.connect('database.db')
03  cursor = conn.cursor()
04  cursor.execute(
05  'SELECT descricao, precounit, quantidade FROM produto, compra,
06  compra_produto WHERE compra.id = compra_produto.idcompra AND ' +
07  'produto.id = compra_produto.id AND compra.id = 1')
08  total = 0
09  for linha in cursor.fetchall():
10    print('{0} (R$ {1}) QUANT: {2}'.format(linha[0], linha[1], linha[2]))
11    total += linha[1] * linha[2]
12  print('Total a pagar: R$ {0}'.format(total))
```

```
13    conn.commit()
14    conn.close()
```

Linhas 08 e 11: cria uma variável e na sequência acumula a soma da multiplicação do preço unitário com a quantidade de cada produto.

Lista de exercício G

1) Com base no banco de dados apresentado, crie o seguinte sistema.

O usuário entrará em um menu que possuirá as seguintes opções:

Escolha uma opção:

1) Criar um novo produto
2) Criar uma nova compra
3) Adicionar um produto na compra
4) Consultar os produtos por nome
5) Consultar dados da compra
6) Excluir uma compra
7) Sair

Cada opção realiza a seguinte operação:

1 → O usuário irá inserir os dados do novo produto e gravar no banco de dados;

2 → O usuário poderá criar uma nova compra

3 → A partir de uma compra o usuário poderá adicionar um item, produto, na lista de compras

4 → A partir da descrição o usuário poderá consultar os produtos cadastrados

5 → A partir de uma compra, será listado todos os dados do produto e sua quantidade, assim como o total a pagar

6 → O usuário sairá do sistema

Tratamento de Exceção

Ao escrever um código, o desenvolvedor poderá cometer três tipos de erro: erro lógico; erro de sintaxe; erro de execução.

O erro lógico ocorre quando o desenvolvedor comete um equívoco nas regras de negócio da aplicação, esse tipo de erro normalmente é descoberto quando usuários que utilizam o sistema descobrem a incoerência do problema com a solução apresentada.

O erro de sintaxe ocorre no momento que o código está sendo escrito. Esse tipo de erro é o mais simples de ser detectado tendo em vista que ocorre durante a criação do projeto.

O erro de execução ocorre quando o usuário solicitação a chamada de uma ação e esta por sua vez não consegue ser completada devido a um erro em alguma instrução do código. A seguinte instrução é um exemplo clássico de um erro de execução:

```
x = 1/0
```

Nessa instrução o sistema não acusa erro e permite que o código seja executado. No entanto ao executá-lo, será exibida a seguinte mensagem:

```
Traceback (most recent call last):
  File "C:/desenvolvimento/ numero01.py", line 1, in <module>
    x = 1/0
ZeroDivisionError: division by zero
```

Neste capítulo veremos como tratar erros de execução através do tratamento de exceções. As palavras chaves ***try/except*** são utilizadas na construção do tratamento de exceção. Utilizando o tratamento de exceção, o mesmo código poderia ser escrito da seguinte forma:

```
try:
    x = 1/0
except BaseException:
    print('Erro de divisão por zero')
print('Continuação...')
```

Embora ocorra o erro, a execução não é interrompida.

É possível declarar mais de um ***except*** para tratar diferentes tipos de erro. Exemplo:

```
import zipfile
try:
    banco_zip = zipfile.ZipFile("saida.zip")
    banco_zip.extractall(path="banco")
    banco_zip.close()
except FileNotFoundError:
    print("Arquivo inexistente")
except PermissionError:
    print("Erro de permissao")
```

Neste caso o código trata dois tipos de exceção: primeiro caso não encontre o arquivo (`FileNotFoundError`); e em segundo, caso não tenha permissão para extração do arquivo (`PermissionError`).

Hierarquia de exceções

Para cada erro é possível especificar seu tipo, por exemplo, quando um número é divido por zero o erro invocado será `ZeroDivisionError`. Em Python existe uma hierarquia de erros.

- `BaseException`
  - `SystemExit`
  - `KeyboardInterrupt`
  - `GeneratorExit`
  - `Exception`
    - `StopIteration`
    - `ArithmeticError`
      - `FloatingPointError`
      - `OverflowError`
      - `ZeroDivisionError`
    - `AssertionError`
    - `AttributeError`
    - `BufferError`
    - `EOFError`
    - `ImportError`
    - `LookupError`

        - IndexError
        - KeyError
    - MemoryError
    - NameError
        - UnboundLocalError
    - OSError
        - BlockingIOError
        - ChildProcessError
        - ConnectionError
            - BrokenPipeError
            - ConnectionAbortedError
            - ConnectionRefusedError
            - ConnectionResetError
        - FileExistsError
        - FileNotFoundError
        - InterruptedError
        - IsADirectoryError
        - NotADirectoryError
        - PermissionError
        - ProcessLookupError
        - TimeoutError
    - ReferenceError
    - RuntimeError
        - NotImplementedError
    - SyntaxError
        - IndentationError
            - TabError
    - SystemError
    - TypeError
    - ValueError
        - UnicodeError
            - UnicodeDecodeError
            - UnicodeEncodeError
            - UnicodeTranslateError
    - Warning
        - DeprecationWarning
        - PendingDeprecationWarning
        - RuntimeWarning
        - SyntaxWarning
        - UserWarning
        - FutureWarning
        - ImportWarning
        - UnicodeWarning
        - BytesWarning
        - ResourceWarning

Caso haja dúvida que qual exceção utilizar, é possível invocar o tratamento de erro primário `BaseException`. Cada classe de exceção possui sua particularidades.

Ação de limpeza

Ao escrever um código que contenha o tratamento de exceção, pode ocorrer do desenvolver precisar limpar alguma variável, fechar algum arquivo ou procedimento que esteja aberto. Considerando esses casos, existe um comando opcional que pode ser utilizado juntamente com `try/except`. Este comando é conhecido como `finally`.

O comando `finally` sempre executará o código, independente se o erro ocorreu ou não. Exemplo:

```
def lerMetaData(arquivo):
    try:
        dados = open('data/meta-data/' + arquivo, 'rt')
        metaData = []
        for linha in dados:
            i = linha.split('\t')
            metaData.append((i[0], i[1].split('.')[0]))
    except FileNotFoundError:
        print('Arquivo não encontrado.')
    finally:
        dados.close()
    return metaData
```

Comando raise

Este comando é utilizado quando o desenvolvedor precisa provocar um erro controlado. Siga a sintaxe:

```
raise Erro
```

onde:

- `raise` → palavra-chave utilizada para provocar o erro;
- `Erro` → erro a ser invocado.

Exemplo:

```
def verificar(sabor):
```

```
    if sabor not in ('CREME', 'CHOCOLATE', 'NAPOLITANO'):
        raise BaseException

try:
    sabor = input('Escolha o sabor de picolé: ')
    verificar(sabor)
    print('Boa escolha')
except BaseException:
    print('Não trabalhamos com esse sabor.')
```

Lista de exercício H

1) A partir do código apresentado:

```
x = [1, 3, 5, 6, 7, 8]
i = 0
while i <= len(x):
    print(x[i])
    i += 1
```

crie um tratamento de exceção para evitar a ocorrência do erro.

2) Crie um programa em Python que receba n números e some esses números enquanto a soma não for superior a 100. O programa deverá imprimir o valor somado (antes de atingir o número maior que 100) e deverá informar quantos números foram somados e qual a média. Refaça seu programa utilizando as seguintes regras:

a. Utilize tratamento de exceção para lidar com a entrada de dados;

b. Quando a soma for superior a 100, o programa deverá gerar uma exceção.

3) Escreva um programa em Python que preencha valores de um *array* com 10 posições. O usuário irá informar os valores a serem inseridos e suas respectivas posições no *array*. O programa deve tratar as exceções quando for informada uma posição inexistente do *array* e quando o valor informado não for número.

4) Suponha que o usuário realize transações em sua conta corrente. Desenvolva um código no qual o usuário digitará seu saldo inicial. Ele poderá realizar um Débito ou Crédito. Lance uma exceção caso o saldo do cliente fique negativo. O valor do saldo não deverá ficar negativo.

Arquivos

Neste capítulo você aprenderá como manipular arquivos utilizando a linguagem Python. Aprenderemos a ler e gravar dados em arquivos de texto.

Abrindo arquivo

Para realizar a leitura devemos utilizar uma função Python denominada *open()*. Ela permite que seja realizada a abertura do arquivo. Um Python 3 podemos ter arquivos: binários; binários bufferizados e de texto. Neste material manipularemos apenas arquivos de texto. Observe o seguinte exemplo:

```
01  arquivo = open('documentos/dados.csv', 'r')
02  print(arquivo)
03  arquivo.close()
```

Na **Linha 01** a variável *arquivo* recebe um conteúdo do tipo TextIOWrapper cujo nome do arquivo é *dados.csv* e o modo de abertura é do tipo leitura '*r*'.
A **Linha 02** apresenta o conteúdo da variável *arquivo*.

<_io.TextIOWrapper name='documentos/dados.csv' mode='r' encoding='cp1252'>

Além do nome (name) e do modo de leitura (mode) a variável apresenta o tipo de codificação de caracteres (encoding), neste caso **cp1252**.
Na **Linha 03** a variável é fechada, *close()*. Sempre que um arquivo for aberto, é importante que este também seja encerrado.

A seguir temos uma lista contendo os possíveis modos de abertura de um arquivo:

Caractere	Descrição
r	abre em modo leitura
w	abre para escrita
x	abre para criação, falhando se o arquivo existir
a	abre para escrita, adicionando o conteúdo no fim do arquivo caso exista
b	modo binário
t	modo texto
+	abre um arquivo de disco para atualização (lendo e escrevendo)

Para maiores informações consulte a documentação:

https://docs.python.org/3/library/

Lendo arquivo

Para realizar a leitura dos dados é necessário utilizar a função *read()*. Essa função pode receber um parâmetro do tipo inteiro indicando a quantidade de *bytes* a serem lidos. Caso a função seja invocada sem parâmetro, *read()*, será lido todo arquivo. Se a função for invocada com um parâmetro do tipo inteiro, *read(50)*, será lido apenas a quantidade de bytes que formam estipuladas no parâmetro.

Considere o exemplo:

```
01  arquivo = open('documentos/dados.csv', 'r')
02  print(arquivo.read())
03  arquivo.close()
04  print('-'*50)
05
06  arquivo = open('documentos/dados.csv', 'r')
07  print(arquivo.read(14))
08  arquivo.close()
```

A saída deste código será:

```
Denecley Alvim;(62)985814578;denecley.alvim@gmail.com
Astrobaldo;(64)923587458;astrogildo@hotmail.com
Denysis;(62)987892645;dede2019@gmail.com
--------------------------------------------------
Denecley Alvim
```

A **Linha 02** imprime todo o conteúdo do arquivo. A **Linha 07** imprime apenas os primeiros 14 bytes.

O final de cada informação é delimitado pela quebra de linha “\n”.

Caso deseje ler cada linha individualmente, pode-se trabalhar das seguintes maneiras:

- Criando um laço que receba o arquivo; ou

- Utilizando a função *readlines()*.

Exemplo:

- Criando um laço que receba o arquivo

```
01  arquivo = open('documentos/dados.csv', 'r')
02  for i in arquivo:
03      print(i)
04  arquivo.close()
```

- Utilizando a função *readlines()*

```
01  arquivo = open('documentos/dados.csv', 'r')
02  lista = arquivo.readlines()
03  print(lista)
04  arquivo.close()
```

Escrevendo arquivo

Um arquivo pode ser aberto para edição de duas formas:

- `w` → abre o arquivo e apaga todo conteúdo previamente escrito;
- `a` → abre o arquivo e mantem o conteúdo anterior. Novos dados são inseridos no fim do arquivo.

Exemplo:

```
01  arquivo = open('documentos/dados.csv', 'a')
02
03  nome = input('Entre com o nome: ')
04  telefone = input('Entre com o telefone: ')
05  email = input('Entre com o e-mail: ')
06
07  envio = "{0};{1};{2};\n".format(nome, telefone, email)
08
09  arquivo.write(envio)
10  arquivo.close()
```

Neste código o arquivo será aberto e manterá todos os dados anteriores. O usuário digitará o: nome; telefone; e email. Os dados serão gravados no seguinte formato:

`nome;telefone;email;\n`

Para efetivar a gravação utilize a função `write()`. Essa função espera receber uma *String* (texto).

Lista de exercício I

1) Em Python, crie um arquivo chamado **real.csv**.

2) Em Python, faça com que o usuário digite o valor de X. Grave cada valor digitado no arquivo **real.csv**. O usuário digitará quantos números quiser. Ele encerrará quando digitar o valor zero (0).

3) Em Python, imprima o conteúdo do arquivo **real.csv**.

4) Em Python, leia o arquivo **real.csv** e calcule $y = 0.9^x$ para cada *x*. Grave os valores calculados (y) em um arquivo **imaginario.csv**.

5) Em Python, leia os arquivos: **real.csv**; e **imaginario.csv**. Imprima os valores de *x* em relação a *y* no seguinte formato:

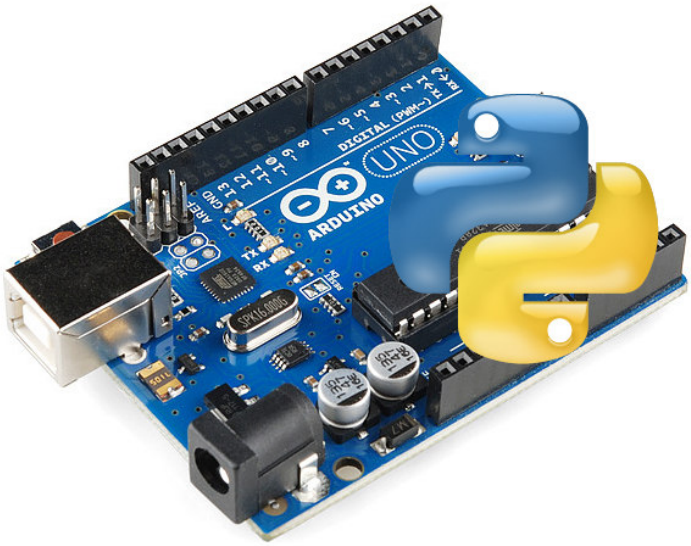
y = x

6) Em Python, crie o seguinte sistema:

Menu: 1) Criar Endereço 2) Deletar Endereço 3) Atualizar Endereço 4) Listar Endereço 5) Listar Endereço por Bairro 6) Sair	Um endereço é constituído pelos seguintes dados: • rua; • complemento; • bairro; • cidade; • cep.

Python e Arduino

Segundo BRENDON (2016), O arduino é uma plataforma de prototipagem eletrônica que nos últimos anos vem ganhando muito espaço entre os entusiastas dos mais diversos segmentos. Simples e barato, desenvolver com arduino é um processo bastante prático e divertido, facilitando a inserção de qualquer leigo no mundo da programação e eletrônica.



Neste capítulo veremos como a linguagem Python pode ser usada para conectar com o Arduino. Primeiramente é necessário a instalação de uma biblioteca denominada pyserial. Sua instalação poderá ser feita através do comando:

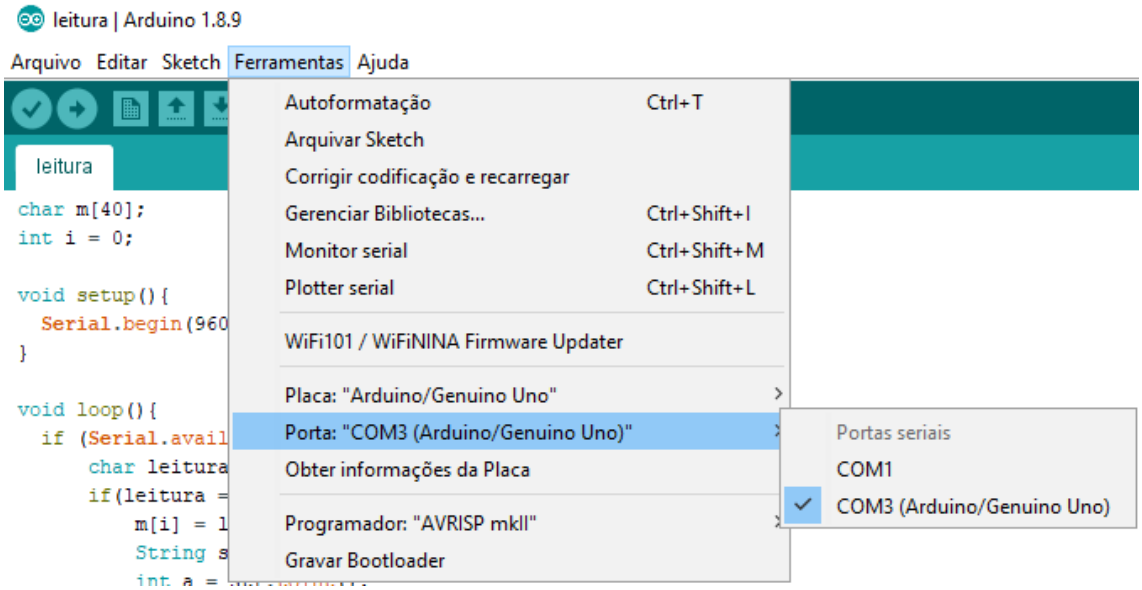
```
01    pip install pyserial
```

Essa biblioteca fornecerá a comunicação entre a linguagem de programação e o dispositivo. O código a seguir realiza a conexão com a placa:

```
01    import serial
02    conexao = serial.Serial('COM3', 9600)
03    print(conexao.isOpen())
04    conexao.close()
```

O comando `serial.Serial` estabelece a conexão, neste método são passados a porta no qual o Arduino encontra-se e a velocidade de conexão. O valor 9600

corresponde a velocidade padrão. A função `isOpen()` retornará `True` se a conexão foi estabelecida com sucesso.



Enviando um byte

Para enviar um byte ao Arduino utilize a função `write`. E para receber um dado do Arduino utilize a função `readline`. Neste primeiro exemplo passaremos um byte para o Arduino e o mesmo nos fornecerá uma resposta. A seguir o código Arduino para receber o byte e retornar uma resposta:

```
01  char dado;
02  void setup(){
03    Serial.begin(9600);
04  }
05  void loop(){
06    if (Serial.available() > 0) {
07      dado = Serial.read();
08      Serial.println("CHEGOU");
09    }
10  }
```

Código Python:

```
01  import serial
02  import time
03  conexao = serial.Serial('COM3', 9600)
```

```
time.sleep(1.7)
conexao.write(b'6')
result = conexao.readline().decode('utf-8')
print('Retorno: {0}'.format(result))
conexao.close()
```

Enviando um conjunto de bytes (String)

A transmissão entre a linguagem Python e o arduino ocorre byte a byte. Cada pacote de dado equivale a um byte. Neste exemplo veremos como enviar um conjunto de bytes, uma String.

Código Arduino:

```
char m[40];
int i = 0;
void setup(){
  Serial.begin(9600);
}
void loop(){
  if (Serial.available()){
      char leitura = Serial.read();
      if(leitura == '\0'){
          m[i] = leitura;
          String str = m;
          Serial.println(str);
      }else{
          m[i] = leitura;
          i++;
      }
  }
}
```

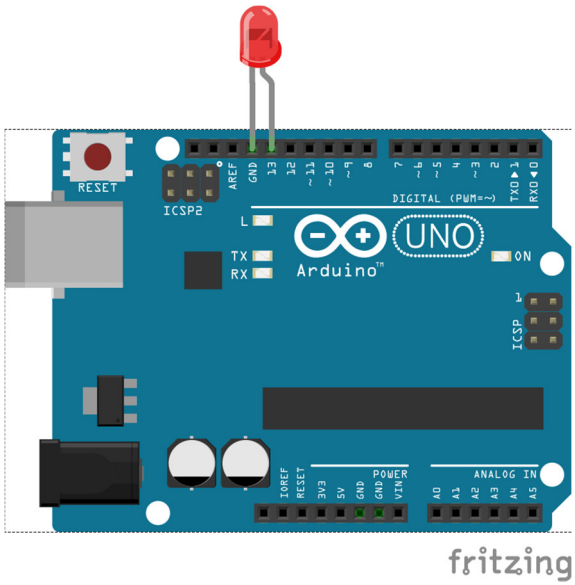
Código Python:

```
import serial
import time
conexao = serial.Serial('COM3', 9600)
time.sleep(1.7)
valor = 'GOMIDE\0';
for i in valor:
    conexao.write(i.encode())
```

```
result = conexao.readline().decode('utf-8')
print(result)
conexao.close()
```

Ligando e desligando um LED

Considere:



Código Arduino:

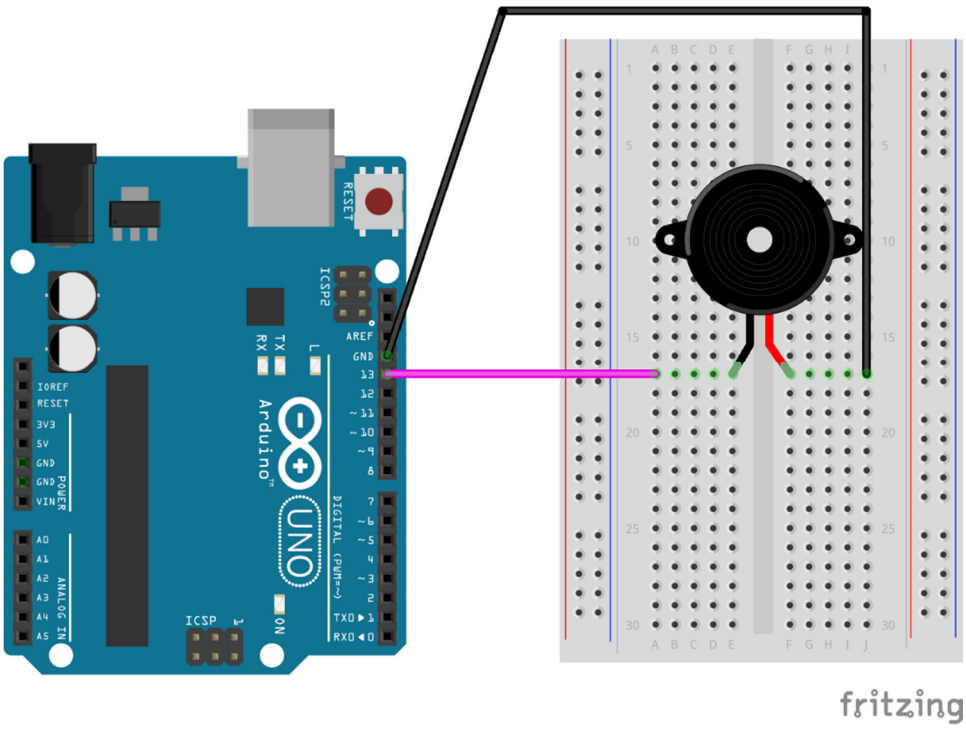
```
int led1 = 13;
char leitura;
void setup(){
  Serial.begin(9600);
  pinMode(led1, OUTPUT);
}
void loop(){
  if (Serial.available() > 0) {
    leitura = Serial.read();
    if (leitura == 'L') {
      digitalWrite(led1, HIGH);
      Serial.println("LED ligado com sucesso.");
    }
    else if (leitura == 'D') {
      digitalWrite(led1, LOW);
      Serial.println("LED desligado com sucesso.");
    }
  }
}
```

Código Python:

```
01  import serial
02  import time
03
04  conexao = serial.Serial('COM3', 9600)
05  time.sleep(1.7)
06  while True:
07      opcao = int(input("1 - LIGAR  2 - DESLIGAR  0 - SAIR: "))
08      if opcao == 0:
09          break
10      elif  opcao == 1:
11          conexao.write(b'L')
12      elif opcao == 2:
13          conexao.write(b'D')
14      result = conexao.readline().decode("utf-8")
15      print(result)
16  conexao.close()
```

Alterando a frequência do buzzer

Considere:



Código Arduino:

```
int frequencia = 0;
char m[40];
int i = 0;
void setup(){
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}
void loop(){
  if (Serial.available()){
      char leitura = Serial.read();
      if(leitura == '\0'){
          m[i] = leitura;
          String str = m;
          frequencia = str.toInt();
          resetar();
      }else{
          m[i] = leitura;
          i++;
      }
  }
  if(frequencia == 0)
    noTone(13);
  else
    tone(13, frequencia);
}
void resetar(){
    for(int i = 0; i < strlen(m); i++){
        m[i] = ' ';
    }
    i = 0;
}
```

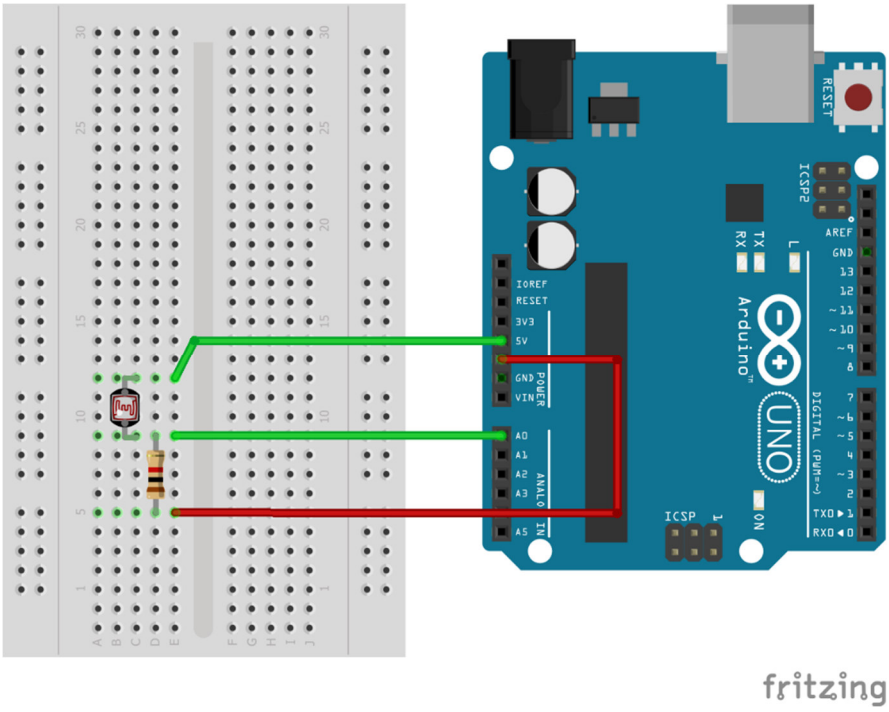
Código Python:

```
import serial
import time
conexao = serial.Serial('COM3', 9600)
time.sleep(1.7)
while True:
    opcao = input("Digite um valor (s - SAIR): ")
    if opcao == 's':
        break
```

```
    else:
        opcao += '\0'
        for i in opcao:
            conexao.write(i.encode())
conexao.close()
```

Recebendo dados do sensor de luminosidade

Considere:



Código Arduino:

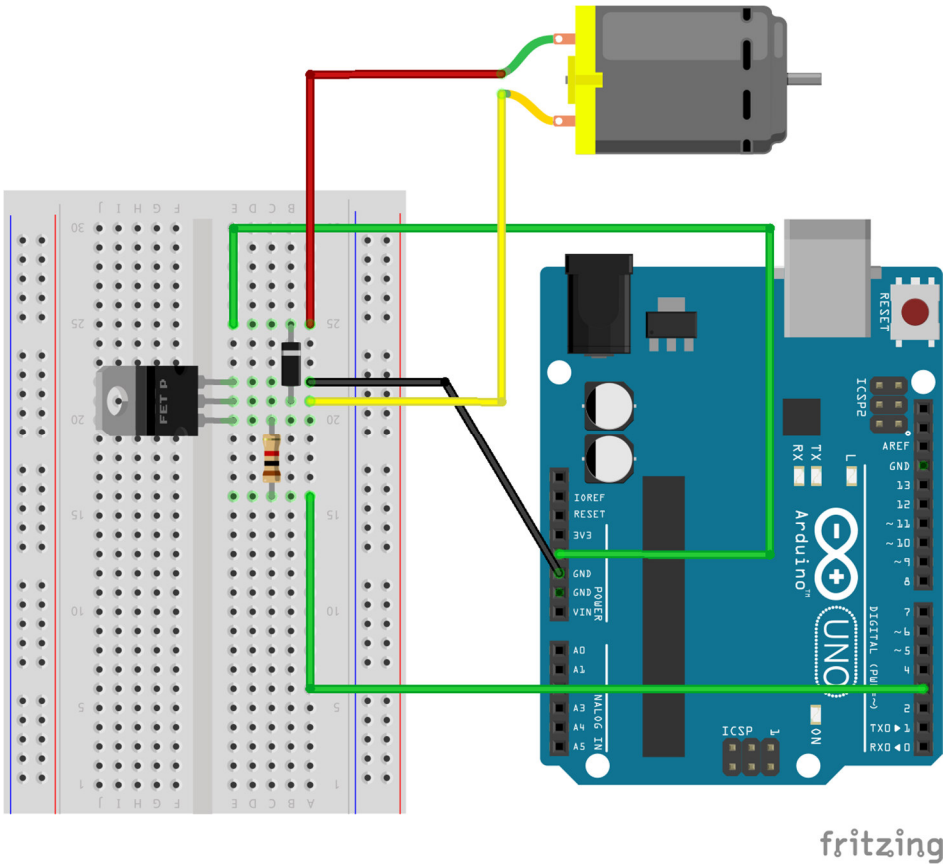
```
int pinoLDR = A0;
int leitura = 0;
void setup(){
  Serial.begin(9600);
  pinMode(pinoLDR, INPUT);
}
void loop(){
  leitura = analogRead(pinoLDR);
  Serial.println(leitura);
  delay(1000);
}
```

Código Python:

```
01  import serial
02  import time
03  conexao = serial.Serial('COM3', 9600)
04  time.sleep(1.7)
05  while True:
06      result = conexao.readline().decode("utf-8")
07      intresult = int(result)
08      if intresult < 45:
09          print('ESCURO')
10      else:
11          print('CLARO')
12  conexao.close()
```

Manipulando Motor DC

Considere:



Código Arduino:

```
01  int pwm = 3;
02  int velocidade = 0;
```

```
char m[40];
int i = 0;
void setup(){
  Serial.begin(9600);
  pinMode(pwm, OUTPUT);
}
void loop(){
  if (Serial.available()){
      char leitura = Serial.read();
      if(leitura == '\0'){
          m[i] = leitura;
          String str = m;
          velocidade = str.toInt();
          resetar();
      }else{
          m[i] = leitura;
          i++;
      }
  }
  analogWrite(pwm, velocidade);
}
void resetar(){
    for(int i = 0; i < strlen(m); i++){
        m[i] = ' ';
    }
    i = 0;
}
```

Código Python:

```
import serial
import time
conexao = serial.Serial('COM3', 9600)
time.sleep(1.7)
while True:
    opcao = input("Digite um valor entre [0-255] (s - SAIR): ")
    if opcao == 's':
        break
    else:
        opcao += '\0'
        for i in opcao:
            conexao.write(i.encode())
conexao.close()
```

Referência Bibliográfica

CRUZ, F. **Python: Escreva seus primeiros programas**. 1º ed. Casa do Código: São Paulo. 1998.

BRENDON, H. **Python e Arduino: Ganhando produtividade em seus projetos de internet das coisas**. 2016. Acessado em: junho de 2019. Disponível em: encurtador.com.br/mnr05.

MENEZES, N. **Introdução à programação com Python: algoritmos e lógica de programação para iniciantes**. 2º ed. Novatec: São Paulo. 2014.