Laboratório Python Avançado 2.0

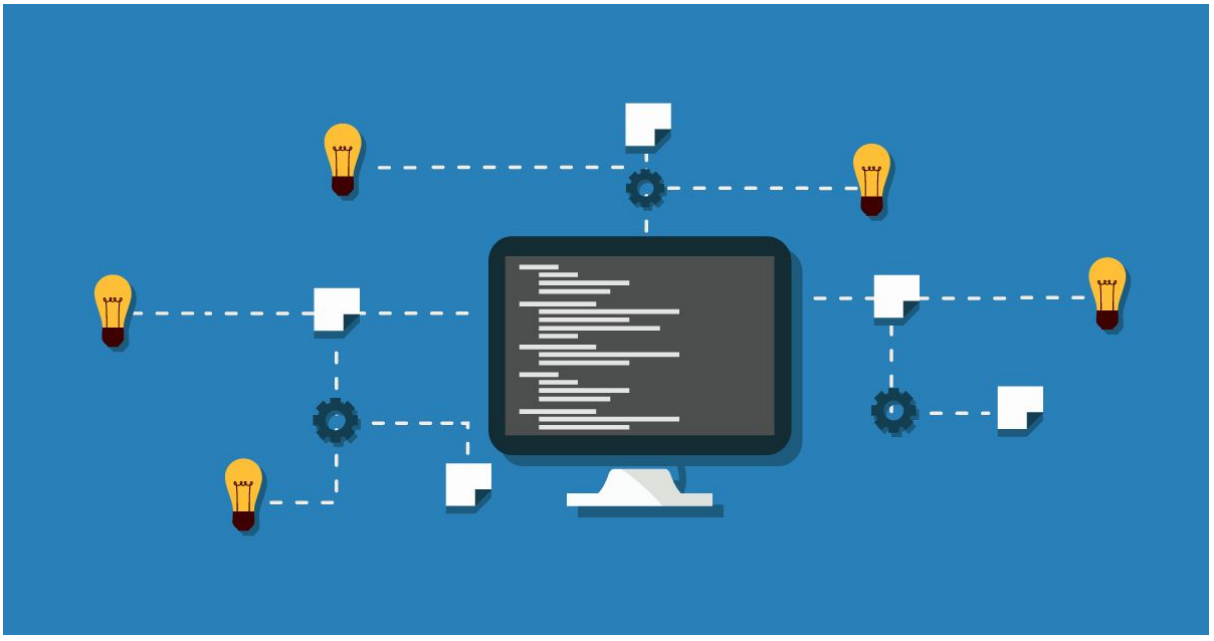
Lucas Jr. Ribas - 160052289
Thiago Chaves Monteiro de Melo -

6 - Algoritmos Estruturados

Módulo VI - Python - Algoritmos Estruturados

6.1 O que é?

Atualmente a necessidade de aumentar o desempenho na realização de tarefas é cada vez maior, com isso a automatização das mesmas se torna mais comum em várias áreas. Pense que você tenha que realizar uma tarefa, ao começar você divide tudo em passos e segue tudo em ordem do começo ao fim de maneira que se ocorrer de pular um passo ou inverter sua ordem, a tarefa não sairá com um resultado esperado, ou seja, uma forma procedural de realizar tarefas. Na programação isso é uma forma de que o programador possui sobre a estruturação e execução do programa, ou seja é um paradigma de programação, sendo assim uma de suas estruturas.



Para entendermos melhor vamos pensar em algumas situações hipotéticas, em que você precise calcular a média de produtos vendidos em sua loja na última semana, fazer um inventário do seu estoque de comida, ou até mesmo criar uma base de dados dos seus

livros preferidos e buscar de forma fácil qual livro falta para ler, ou quantas páginas faltam por exemplo, as possibilidades são infinitas.

6.2 Botando em prática

6.2.1 Procedural

Como parte do título já nos dá uma pista sobre qual linguagem utilizaremos, então já vamos direto para o nosso primeiro exemplo de algoritmo estruturado na linguagem Python.

Para essa aula utilizaremos o sistema operacional Linux Ubuntu.

Verifique se já tem o Python instalado, provavelmente já possui alguma versão do Python instalada por padrão. Para conferir, digite em um terminal:
`$ which python`

ou
`$ which python3`

que deve retornar algo como `/usr/bin/python`. Isso significa que o Python está instalado nesse endereço.
Caso contrário, se retornar algo como `which: no python in (/usr/local/sbin:/usr/local/bin:/usr/bin:/usr...)` você precisa instalar pelos repositórios ou gerenciador de pacotes de sua distribuição.

Instalação por Gerenciadores de Pacotes

Os gerenciadores de pacotes mais comuns são apt-get (Debian, Ubuntu) e yum (RedHat, CentOS). Caso sua distribuição utilize um gerenciador de pacotes diferente, acesse a **página de downloads do Python**.

Apt-get

Para instalar o Python 2.7, digite em um terminal:
`$ sudo apt-get install python2.7`

Para instalar o Python 3.5, digite em um terminal:
`$ sudo apt-get install python3.5`

(Opcional) Para instalar o gerenciador de pacotes pip, digite em um terminal:
`$ sudo apt-get install python-pip`

Yum

Para instalar o Python 2.7, digite em um terminal:

```
$ sudo yum install python27
```

Para instalar o Python 3.5, digite em um terminal:

```
$ sudo yum install python35
```

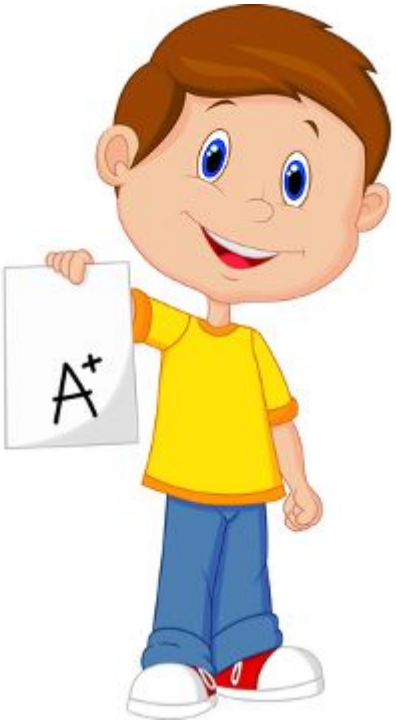
(Opcional) Para instalar o gerenciador de pacotes pip, digite em um terminal:

```
$ yum -y install python-pip
```

Usando um exemplo simples sem precisar de estruturas mais aprofundadas da linguagem vamos calcular a média de nota de um aluno qualquer.

Abrindo um editor de texto qualquer escreveremos a seguinte sintaxe e salvaremos em alguma pasta como “.py” seu_programa.py por exemplo.

```
n1 = float (input (‘Primeira nota do aluno: ’) )
n2 = float (input (‘Segunda nota do aluno: ’) )
media = (n1 + n2) / 2
print(‘A média entre {:.1f} e {:} é igual a {}’.format( n1, n2, media ))
```



Note que na primeira linha eu leio do teclado a nota 1 do aluno e na segunda linha a nota 2, após isso eu resolvo a média entre as duas notas e mostro o resultado na tela. De maneira simples e rápida da para entender os conceitos de programação estruturada. Basicamente o que compoem um algoritmo estruturado é o uso de subrotinas, laços de repetição,

condicionais e estruturas de blocos. Sendo algum desses assuntos vistos em módulos anteriores.

6.2.2 Orientado a objeto

A orientação a objeto é uma nova forma de programar, sem seguir a forma procedural como antes, você não mais precisa necessariamente seguir ao pé da letra a ordem de fazer as tarefas, ou seja é uma maneira mais modular de programar.

Nesse paradigma você cria uma idéia na sua cabeça e tenta organizar essa idéia em núcleos lógicos. Se resolver criar um jogo eletrônico, por exemplo de corrida, você irá precisar de um objeto carro, com todos os seus atributos, roda, cor, potência e etc.

Python não é a linguagem mais recomendada para aprender programação orientada a objeto, apesar de ser baseada, ela é uma linguagem muito aberta e não te restringe a programar sempre usando esse paradigma.

Classes

Uma classe define uma estrutura de dados que contenha instância de atributos, instância de métodos e classes aninhadas.

A classe é o que faz com que Python seja uma linguagem de programação orientada a objetos. Classe é definida como um agrupamento de valores sua gama de operações. As classes facilitam a modularidade e abstração de complexidade. O usuário de uma classe manipula objetos instanciados dessa classe somente com os métodos fornecidos por essa classe.

Frequentemente classes diferentes possuem características comuns. As classes diferentes podem compartilhar valores comuns e podem executar as mesmas operações. Em Python tais relacionamentos são expressados usando derivação e herança.

```
class Pessoa:
    def __init__(self, nome,
idade):
        self.nome = nome
        self.idade = idade

    def setNome(self, nome):
        self.nome = nome

         def  setIdade(self,
idade):
        self.idade = idade

    def getNome(self):
        return self.nome

    def getIdade(self):
        return self.idade
```

6.3 Exemplos

Segue mais alguns exemplos procedurais abaixo:

```
# Calcula a área de um círculo.

def find Área(r):
    PI = 3.142
    return PI * (r*r);

# mostrar na tela
print("Area is %.6f" % findArea(5));

##########################################

# Mostra na tela todos os números primos em um intervalo.

start = 11
end = 25

for val in range(start, end + 1):

# Se num é divisível por qualquer número
# entre 2 e val, não é primo

   if val > 1:
       for n in range(2, val):
           if (val % n) == 0:
               break
       else:
           print(val)

##########################################

# Verifica se x é um quadrado perfeito
import math

# Uma função de utilidade que retorna true se x for quadrado
perfeito

def isPerfectSquare(x):
    s = int(math.sqrt(x))
    return s*s == x
```

```
# Retorna verdadeiro se n é um número de Fibinacci, senão falso
def isFibonacci(n):

# n é Fibinacci se um dos 5 * n * n + 4 ou 5 * n * n - 4 ou ambos
é um quadrado perfeito.

    return isPerfectSquare(5*n*n + 4) or isPerfectSquare(5*n*n -
4)

# Uma função de utilidade para testar as funções acima
for i in range(1,11):
     if (isFibonacci(i) == True):
         print i,"is a Fibonacci Number"
     else:
         print i,"is a not Fibonacci Number "
```

Exemplos Orientado a objeto:

```
  import datetime
  import math
  class humanos(object):
         def __init__(self,nome,dtnasc,peso,altura):
                      self.nome=nome
                      self.dtnasc=dtnasc
                      self.peso=peso
                      self.altura=altura
         def getidade(self):
                    calc=
datetime.date.today()-self.dtnasc
                    return int (calc.days/365.25)
     def getimc(self):
                    return
self.peso/math.pow(self.altura/100., 2)
 pessoa = humanos("Pessoa",datetime.date(1985,4,1),
98,178)
 print (pessoa.nome)
 print (pessoa.getidade(),"Anos de idade")
 print ("IMC é de:",round(pessoa.getimc(),2))
```

6.4 Lista de exercícios - em vídeo

Para já ir aquecendo, simule um comércio e calcule a média de lucro diário de sua loja hipotética. Utilize a linguagem Python para resolução do exercício.

Praticando programação estruturada:

https://www.youtube.com/playlist?list=PLHz_AreHm4dm6wYOIW20Nyg12TAjmMGT-
112 exercícios dos mais variados tipos de estruturas de programação, inclusive strings que é o assunto de nosso próximo módulo. Até lá ;)

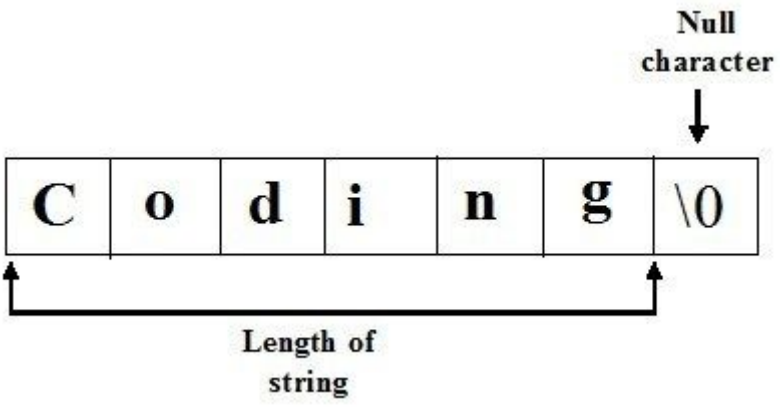
Curso Python orientado a objetos:
https://www.youtube.com/watch?v=uEEuSYkM9o4&list=PLp95aw034Wn_WtEmlepaDrw8FU8R5azcm

7 - Strings

Módulo VII - Python - Strings

7.1 O que é?

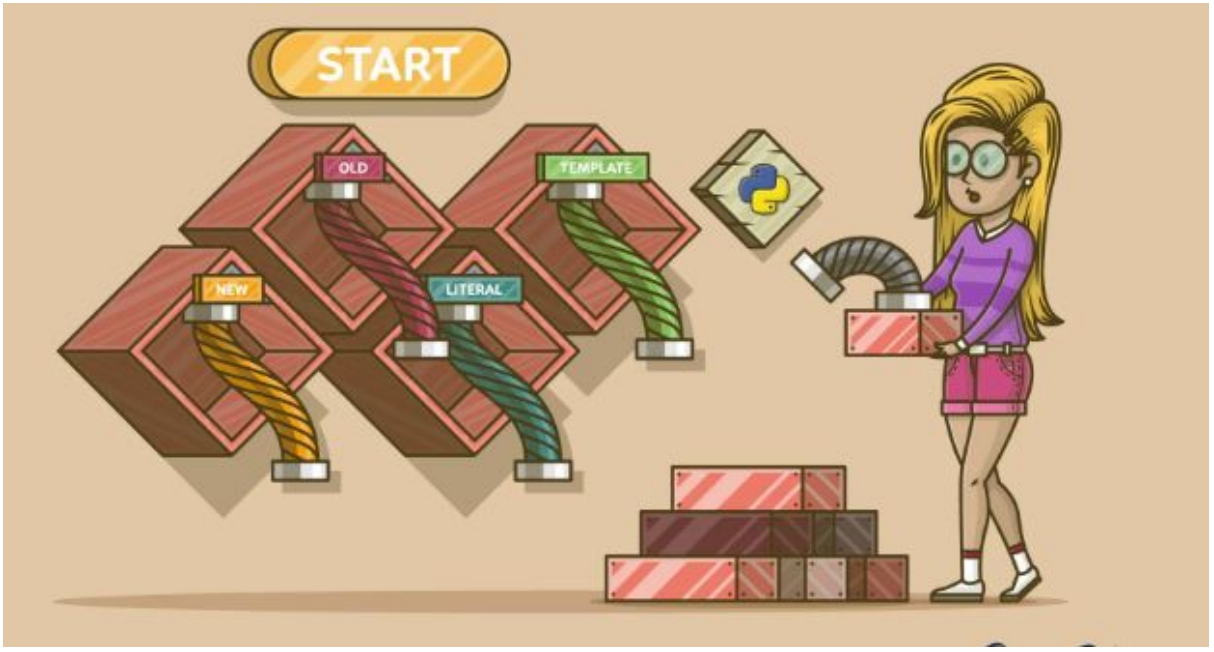
O que pensamos quando ouvimos a palavra string? A primeira resposta é buscarmos sua tradução já que ela não aparenta ser do nosso idioma, e assim nos deparamos com a palavra “corda”, mas como assim?! não faz nenhum sentido. Strings em programação são cadeias de caracteres, ou seja uma sequência de caracteres utilizada para representar palavras, frases ou textos de um programa de computador. As cadeias de caracteres podem ser expressas tanto na forma literal, como através de algum tipo de variável. Quando expressos através de variáveis, o conteúdo da cadeia geralmente pode ser alterado pela inclusão/exclusão de elementos ou pela substituição de seus elementos por outros elementos, formando uma nova cadeia. Assim, uma cadeia de caracteres é vista como sendo um tipo de dado e normalmente é implementada através de um arranjo de bytes que armazena os elementos da cadeia em sequência, utilizando alguma codificação preestabelecida.



Começamos a complicar, mas o que é byte? Um byte é um dos tipos de dados integrais em computação. É usado com frequência para especificar o tamanho ou quantidade da memória ou da capacidade de armazenamento de um certo dispositivo, independentemente do tipo de dados. Quando o tipo de dado é uma String sua composição é estabelecida de forma que 1 caracter equivale a 1 byte, que são 8 bits, e assim uma string vai sendo composta, byte a byte.

7.2 Botando em prática

Bom, vimos a teoria que compoem uma String, agora vamos botar em prática. Imagine que você quer descobrir quantas letras tem o seu texto ou uma frase qualquer que você armazenou em uma string, ou substituir uma palavra sem precisar reescrever-la, esses são apenas alguns exemplos do que podemos fazer com strings.



Vamos colocar-los em prática:

Aqui criamos uma variável do tipo String e atribuímos a ela uma frase qualquer
```
test = 'This is just a simple string.'
```

Usamos a função ‘len’ para descobrir o número de caracteres incluindo espaços.
```
>>> len(test)
29
```

Aqui substituímos a palavra ‘Simple’ por ‘Short’
```
>>> test = test.replace('simple', 'short')
>>> test
'This is just a short string.'
```

7.3 Mais e mais exemplos

Contando quantas vezes aparece a letrar R
```
>>> test.count('r')
2
```

Achando posição da letra ou palavra
```
>>> test.find('r')
18
>>> test[18]
'r'
```

Dividir strings
```
>>> test.split()
['This', 'is', 'just', 'a', 'short', 'string.']
```

Escolhendo o ponto onde será separado
```
>>> test.split('a')
['This is just ', ' short string.']
```

Juntando strings
```
>>> ' some '.join(test.split('a'))
'This is just  some  short string.'
```

Deixando todas as letras maiúsculas
```
>>> test.upper()
'THIS IS JUST A SHORT STRING.'
```

Deixando tudo minúsculo
```
>>> test.lower()
'this is just a short string.'
```

Deixando apenas a primeira letra da string maiúscula
```
>>> test.lower().capitalize()
'This is just a short string.'
```

Deixando a primeira letra de cada palavra maiúscula
```
>>> test.title()
'This Is Just A Short String.'
```

Invertendo, o que é minúsculo se torna maiúsculo e o que é maiúsculo se torna minúsculo
```
>>> test.swapcase()
'tHIS IS JUST A SHORT STRING.'
```

Testando se toda a string é maiúscula
```
>>> 'UPPER'.isupper()
True
>>> 'UpPEr'.isupper()
False
```

Testando se toda a string é minúscula.

```
>>> 'lower'.islower()
True
>>> 'Lower'.islower()
False
```

Testando se todas as primeiras letras das palavras da string são maiúsculas, no caso se ela é um Título

```
>>> 'This Is A Title'.istitle()
True
>>> 'This is A title'.istitle()
False
```

Testando se a string é alfa-numérica, se não tem caracteres especias

```
>>> 'aa44'.isalnum()
True
>>> 'a$44'.isalnum()
False
```

Testando se tem só letras na string

```
>>> 'letters'.isalpha()
True
>>> 'letters4'.isalpha()
False
```

Testando se tem só números na string

```
>>> '306090'.isdigit()
True
>>> '30-60-90 Triangle'.isdigit()
False
```

Testando se tem só espaços na string

```
>>> '   '.isspace()
True
>>> ''.isspace()
False
```

Adicionando espaços nas extremidades da string

```
>>> 'A string.'.ljust(15)
'A string.'
```

Adicionando espaços no lado esquerdo

```
>>> 'A string.'.rjust(15)
'      A string.'
```

Centraliza a string dentre espaços

```
>>> 'A string.'.center(15)
'   A string.   '
```

7.4 Lista de exercícios - em vídeo

Comece com essa idéia, armazene uma frase em uma string e escolha 8 exemplos de manipulação de strings vistos acima para brincar com sua frase. Use a linguagem Python.

Exercícios em vídeo da mesma playlist do módulo 6

Ex 25 e 26.

8 - Funções & Bibliotecas

8.1 O que é?

Funções são agrupamentos de instruções com objetivo de cumprir determinada tarefa. O principal uso de funções é diminuir ao mínimo possível a repetição de código já escrito anteriormente e assim tornar o programa mais simples e fácil de ler. Além disso, funções tornam possível o encapsulamento de determinadas tarefas. Isso quer dizer que pode-se criar rotinas onde suas implementações são escondidas do usuário já que para esse, somente importa o resultado final dessa função.

Com isso em mente, a linguagem python disponibiliza uma série de bibliotecas que não são nada menos que um agrupamento de funções prontas para ajudar o programador de tarefas simples até tarefas bastante complexas.

8.2 Uso básico de funções

Para criar uma nova função em python deve-se utilizar a palavra-chave "def" seguida do nome da função, abre e fecha parênteses e ":". Em seguida deve vir um bloco de código indentado que será a definição do que essa função fará:

```
def diz_oi():
  print("Olá Estudantes!")
```

Tendo essa função pronta, todas as vezes que quisermos utilizar dela durante o código, somente é necessário escrever

```
diz_oi()
```

e aquele trecho de código anteriormente definido será executado.

Funções são extremamente necessárias para deixa um código compreensivo e limpo, por isso o uso delas é recomendado sempre que possível.

https://www.w3schools.com/python/python_functions.asp

https://www.youtube.com/watch?v=ekHovQkbmVc

https://www.youtube.com/watch?v=ezfr9d7wd_k

Exercício:

1. Crie uma função que pegue um nome do terminal e escreva no console "olá, " seguido do nome lido

8.3 Bibliotecas

Existem inúmeras bibliotecas python atualmente e todas elas podem ser baixadas facilmente usando a ferramenta "pip". Para utilizar-se de alguma biblioteca baixada basta digitar a palavra-chave "import" seguida do nome da biblioteca que deseja. Dessa forma todas as funções criadas dentro dessa biblioteca poderão ser usadas livremente durante o código.

```
> import math
> math.floor(3.14)
3
> math.ceil(3.14)
4
```

Como instalar bibliotecas:

https://www.youtube.com/watch?v=GdhqnOQczuc

Bibliotecas que não precisam ser baixadas:

https://pt.wikibooks.org/wiki/Python/Bibliotecas_padrão

Exercício:

Procure funções da biblioteca "math" e teste elas.

Faça o mesmo para a biblioteca "time".

8.4 Argumentos e retorno

Uma função pode também receber argumentos, que funcionam como variáveis dentro delas e podem mudar seu funcionamento a cada chamada que é feita. Além disso, uma função também pode retornar algum valor como resposta no lugar em que ela foi chamada.

```
def soma_numeros(primeiro_numero, segundo_numero):
  resultado = primeiro_numero + segundo_numero
  return resultado
```

Em seguida, para usar essa função basta:

```
a = soma_numeros(2, 3)
b = soma_numeros(78, 89.3)

print( "A vale: {}".format(a) )
print( "B vale: {}".format(b) )
```

É possível ver no exemplo que a função recebe dois argumentos: "primeiro_numero" e "segundo_numero". Da mesma forma que foi mostrado, para usar argumentos na declaração de uma função, basta apenas colocar os nomes deles seguidos por "," dentro dos parênteses.

Além disso vemos que no exemplo `a = soma_números(2, 3)`, isso significa que a variável "a" receberá o retorno da função após ela ser executada com os argumentos 2 e 3. O retorno de uma função é determinado pelo que vem em seguida da palavra-chave "return", e essa mesma para a execução da função. Como o nome mesmo sugere, o que acontece é o retorno da execução do programa para o lugar onde a função foi chamada.

Um exemplo de função útil muito utilizada é a função randint da biblioteca random. Ela pega dois números como argumento e retorna um número inteiro entre esses dois.

```
import random
rnd = random.randint(0,9)
print(rnd)
```

https://panda.ime.usp.br/pensepy/static/pensepy/05-Funcoes/funcoes.html

https://www.dcc.ufrj.br/~fabiom/mab225/07func.pdf

https://www.youtube.com/watch?v=etjJ_4Eqrk8

Exercícios:

1. Crie uma função que receba 4 argumentos, x1, y1, x2 e y2, calcule a distância entre os pontos (x1, y1) e (x2, y2) e retorne o resultado do cálculo.
2. Crie uma função que receba receba um número inteiro e retorne se ele é um número primo ou não.

9 - Armazenamento de dados e arquivos

9.1 Introdução

Como vocês já devem ter notado, após a execução de um programa python, todos os dados que estão salvos nas variáveis são perdidos. Dependendo da funcionalidade do programa, isso pode não importar, porém muitas vezes é necessário que exista uma persistência da informação para uso futuro.

Para resolver esse problema é possível usar o armazenamento de dados em arquivos. Assim, pode-se ter, em um arquivo de texto ou binário, as informações relevantes para o uso do programa salvas em memória permanente.

Créditos: https://realpython.com/working-with-files-in-python/

9.2 A função open

Para abrir, criar ou adicionar conteúdo em um arquivo de texto, usa-se a função "open" do python. Essa recebe 2 argumentos, o caminho para o arquivo desejado e o modo como ele deve ser aberto. Existem três tipos de modo diferentes, sendo eles "r" para leitura, "w" para escrita e, "a" para adicionar conteúdo. Para mais informações sobre esses modos acesse: https://stackoverflow.com/questions/1466000/python-open-built-in-function-difference-between-modes-a-a-w-w-and-r.

Aqui podemos ver um exemplo de uso da função open:

```
arquivo = open("meu_arquivo.txt", "w")
```

Essa função retorna um objeto que pode ser usando para escrever ou ler o conteúdo do arquivo, dependendo do segundo argumento passado.

Um detalhe importante sobre arquivos é que sempre, após o término do uso do arquivo, é necessário chamar o método "close" para salvar as mudanças feitas.

Para escrever no arquivo pode ser usado o método write. Exemplos de código para escrever em arquivos podem ser encontrados aqui:

https://www.guru99.com/reading-and-writing-files-in-python.html

https://www.w3schools.com/python/python_file_open.asp

https://www.tutorialspoint.com/python/python_files_io.htm

Agora, para a leitura de arquivos já existentes, existem várias opções possíveis. A mais direta é o método “read” que retorna o texto inteiro do arquivo. Tendo esse texto todo um uma como uma string, é possível usar as funções que manipulam strings para extrair a informação desejada do texto e trabalhar em cima dela. Além desse método, existem outros que tornam a leitura do arquivo mais dinâmica. Um desses métodos é o “readline” que lê apenas uma linha do arquivo e retorna uma string que contém o texto dessa linha. Outro méotodo possível é o “readlines” que nesse caso lê todo o arquivo e retorna uma lista contendo várias strings, sendo cada uma delas umas das linhas do arquivo.

Para mais informações sobre esses métodos, entre em:

https://www.w3schools.com/python/python_file_open.asp

https://stackabuse.com/reading-files-with-python/

Avançado:

https://www.pythonforbeginners.com/os/python-the-shutil-module

9.4 Exemplos

```
# Cria um arquivo novo (ou apaga o conteudo caso já exista) para escrever nele.

meu_arquivo = open("novo_arquivo.txt", "w")


# Escrevendo conteudo no arquivo novo_arquivo.txt

meu_arquivo.write("Esse arquivo contém meus dados de vida: ")

meu_arquivo.write("Meu nome é ...")

...


# Fecha o arquivo para salvar mudanças
```

```
meu_arquivo.close()
```

Exemplo 2:

```
import random

arq = open("numeros_aleatorios.txt", "w")


# Escrevendo conteudo no arquivo novo_arquivo.txt

for i in range(100):

  aleatorio = random.randint(0,100)

  arq.write("%d: %d" % (i , aleatorio))

...


# Fecha o arquivo para salvar mudanças

arq.close()
```

Após o final do código mostrado no exemplo 2, teremos salvos em memória, no arquivo "numeros_aleatorios" 100 números aleatorios, que podemos usar como quisermos em outros programas.

9.5 Exercícios

1. Faça um programa que funcione como um diário, onde toda vez que você rodar ele poderá escrever novas informações ou ler as antigas.
2. Escreva uma função que receba como paramêtro o nome de um arquivo e escreva no console todo seu conteúdo.