

Guia de estudos *Data Science*

Introdução ao Python

Neuron/DSAI



GRUPO DE ESTUDOS EM *Data Science* NEURON/DSAI

FACEBOOK.COM/NEURONDSAI

Este material, em conjunto com os Notebooks do Jupyter que o acompanham, é de propriedade do grupo Neuron/DSAI e, legalmente, não pode ser reproduzido para fins comerciais (sujeito a processos judiciais). Qualquer dúvida ou sugestão entre em contato conosco pela nossa página do Facebook ou e-mail: *neuron.conteudo@gmail.com*.

Autores: Leonardo Ernesto, Samuel Henrique, Felipe Maia Polo, Matheus Faleiros, Pablo Leonardo, Murilo Henrique Soave, Carlos Henrique Lavieri e Mateus Padua

Edição: 24 de agosto de 2018



Sumário

1	Introdução	5
1.1	O grupo	5
1.2	O conteúdo	5
1.3	Os autores	6
1.4	Gestão 2018	8
2	Programação em Python	9
2.1	Introdução ao Python	9
2.1.1	O que é Python?	9
2.2	Instalação Python	9
2.2.1	Instalação no Windows	9
2.2.2	Instalação no Linux	10
2.2.3	Instalação no Mac	10
2.3	Instalando PIP	11
2.4	Jupyter Notebook	11
2.4.1	Instalação no Windows	12
2.4.2	Instalação no Linux/Mac	12
2.4.3	Conhecendo o Jupyter Notebook	12
2.5	Vamos programar?	13
2.5.1	Variáveis	13
2.5.2	Manipulação de Strings	14
2.5.3	Operadores	15
2.5.4	Condicionais	16

2.5.5	Manipulação de arquivos	17
2.5.6	Loops	18
2.5.7	Listas	19
2.5.8	Funções	23
2.5.9	Como usar o PIP e import de bibliotecas	24
2.5.10	Numpy	24
2.5.11	Pandas	26
2.6	Exercícios complementares ao conteúdo de Introdução ao Python	27
2.7	Exercícios de Desenvolvimento em Python	27
2.7.1	Iniciante	27
2.7.2	Intermediário	28
2.7.3	Avançado	29





1. Introdução

1.1 O grupo

O grupo Neuron surge com o principal intuito de cobrir um *gap* existente na realidade dos brasileiros e até mesmo das principais universidades brasileiras: o estudo, discussão e desenvolvimentos de projetos relacionados ao *Data Science*, que engloba programação, análise de dados e o emprego das tecnologias de Aprendizado de Máquina com fins práticos ligados tanto à pesquisa quanto aos negócios. Como atividade principal, o grupo Neuron abrigará um grupo de estudos em *Data Science* a partir do ano de 2018 localizado no campus da Universidade de São Paulo (USP) em Ribeirão Preto, São Paulo. No que tange às atividades extras, traremos periodicamente profissionais e estudiosos da área para conversar com nossos membros e daremos apoio a alunos que queiram desenvolver projetos próprios. Além disso, fechamos parceria com o grupo HAIT, mais importante grupo de estudos em Inteligência Artificial formado por alunos das melhores universidades japonesas. Essa parceria possibilitará grande troca de experiência e *networking* entre os membros dos dois grupos, sendo que o desenvolvimento de projetos entre alunos de diferentes nacionalidades será o ponto alto. Espero que aproveite seu tempo no Neuron. Grande abraço,

Felipe Maia Polo - Presidente 2018

1.2 O conteúdo

O conteúdo do Neuron foi desenvolvido com a finalidade de maximizar o seu aprendizado em *Data Science*, utilizando a linguagem de programação Python com o enfoque em Aprendizado de Máquina. Os assuntos que serão abordados foram divididos em quatro grandes tópicos: Introdução a Programação em Python, Fundamentos de Matemática, *Machine Learning* e *Deep Learning*. Esses tópicos serão divididos em subtópicos, para melhor explicação dos mesmos. Os conteúdos serão abordados em aulas expositivas com exercícios práticos, que serão disponibilizados antes de cada aula. Também contaremos com palestras de profissionais das diversas áreas, com o intuito de propiciar aos membros do grupo um conhecimento amplo sobre as aplicações das tecnologia no estudos em *Data Science*. Todo o conteúdo foi feito com muito esforço e dedicação com a

colaboração do Felipe Maia Polo e de toda equipe de Conteúdo. Espero que o conteúdo seja completo e de fácil compreensão. Atenciosamente,

Leonardo Ernesto – Vice-Presidente 2018

1.3 Os autores

Este espaço é dedicado para que você conheça um pouco dos nossos autores.

Felipe Maia Polo

Estudante de Economia pela Faculdade de Economia, Administração e Contabilidade de Ribeirão Preto da Universidade de São Paulo. Tem experiência com Estatística, Econometria e Aprendizado de Máquina. Trabalhou no Laboratório de Estudos e Pesquisas em Economia Social (LEPES) focando majoritariamente em avaliação de políticas públicas na área da educação. Concluiu parte da graduação na Universidade de Tóquio, onde teve contato com grupos voltados ao ensino e treinamento de pessoas em *Data Science*, estes que foram grandes inspirações para a concretização do Grupo de Estudos em *Data Science* Neuron/USP.

Leonardo Ernesto

Estudante de Informática Biomédica pela Faculdade de Medicina de Ribeirão Preto e pela Faculdade de Filosofia Ciências e Letras de Ribeirão Preto na Universidade de São Paulo. Exerce o cargo de diretor de assuntos acadêmicos no Centro Estudantil da Informática Biomédica - CEIB, o qual está sob sua responsabilidade desde 2016 e já foi representante Discente na Comissão Coordenadora da IBM em 2016. É aluno de iniciação científica desde 2014 no Laboratório de Tráfego Intracelular de Proteínas, onde desenvolve pesquisa de interação de proteínas. Possui amplo conhecimento nas áreas de Bioinformática, Inteligência Artificial, Reconhecimento de Padrões e *Data Mining*.

Matheus Faleiros

Formado em Informática Biomédica e mestrando do programa Interunidades em Bioengenharia da Universidade de São Paulo. Trabalha com classificação e reconhecimento de padrões em imagens médicas. Possui conhecimento em técnicas de *Machine Learning* e *Deep Learning*.

Samuel Henrique

Estudante de Informática Biomédica pela Faculdade de Medicina de Ribeirão Preto e pela Faculdade de Filosofia Ciências e Letras de Ribeirão Preto na Universidade de São Paulo. Estagiário no Laboratório de Inteligência em Saúde (LIS). Conhecimento avançado na linguagem Python 3.6, algumas de suas bibliotecas e Programação Orientada a Objetos.

Pablo Leonardo

Graduando em engenharia de produção pela faculdade Pitágoras. Graduando em licenciatura plena em matemática pela Universidade Federal do Maranhão. Amante de neuromatemática e matemática computacional.

Murilo Henrique Soave

Estudante de Economia Empresarial e Controladoria na Faculdade de Economia, Administração e Contabilidade da Universidade de São Paulo. Possui conhecimento avançado em Python e C, além de experiência na programação de sistemas embarcados. Atualmente, desenvolve trabalho de iniciação científica vinculado ao Instituto Federal de São Paulo.

Carlos Henrique Lavieri Marcos Garcia

Estudante de Economia na Faculdade de Economia, Administração e Contabilidade de Ribeirão Preto (FEA-RP/USP), com interesse em ciência de dados e Aprendizado de Máquina.

Mateus Padua

Bacharel em Ciência da Computação pela UniSEB COC. Trabalha como desenvolvedor WEB e Engenheiro de Software a 18 anos. Possui conhecimento avançado em Python, Django, JavaScript, OOP, Banco de dados, AWS. Sempre em busca de novos desafios.



1.4 Gestão 2018

Presidente
Felipe Maia Polo

Vice-Presidente
Leonardo Ernesto

Conteúdo

<i>Diretor:</i>	Samuel Henrique	<i>Membro:</i>	Matheus Faleiros
<i>Membro:</i>	Pablo Leonardo	<i>Membro:</i>	Murilo Soave
<i>Membro:</i>	Carlos Henrique	<i>Membro:</i>	Charles Chen
<i>Membro:</i>	Thiago Carvalho	<i>Membro:</i>	Bruno Comitre
<i>Membro:</i>	Mateus Padua		

Projeto

<i>Diretor:</i>	Leonardo Ernesto	<i>Membro:</i>	Eduardo Junqueira
-----------------	------------------	----------------	-------------------

Desenvolvimento de pessoas

<i>Diretor:</i>	Gustavo Ribeiro	<i>Membro:</i>	Edvaldo Santos
<i>Membro:</i>	Henrique Nogueira		

Financeiro

<i>Diretora:</i>	Natasha Freitas	<i>Membro:</i>	Beatriz Machado
<i>Membro:</i>	Murilo Soave		

Marketing

<i>Diretor:</i>	Jonathan Batista	<i>Membro:</i>	Cassio Vilela
<i>Membro:</i>	Willy Oliveira	<i>Membro:</i>	Gustavo Santos

Relações Públicas

<i>Diretor:</i>	Felipe Maia Polo	<i>Membro:</i>	Eduardo Heitor
-----------------	------------------	----------------	----------------

neuron



2. Programação em Python

2.1 Introdução ao Python

2.1.1 O que é Python?

O python é uma linguagem de alto nível (definimos como linguagem de alto nível uma linguagem que é criada com o intuito de ser de fácil entendimento na escrita e na leitura, com comandos mais similares à linguagem natural) que é simultaneamente uma linguagem compilada e interpretada.

Talvez você esteja se perguntando: mas o que são linguagens compiladas e interpretadas?

Linguagens interpretadas são aquelas onde o seu código fonte(o texto escrito com certos comandos) é dado como entrada para um interpretador. Esse interpretador lê as linhas uma por vez e para cada linha interpreta o que ela significa.

Linguagens compiladas são aquelas em que o código fonte é direcionado a um compilador e este compilador transforma seu programa em código de máquina(0 e 1).

2.2 Instalação Python

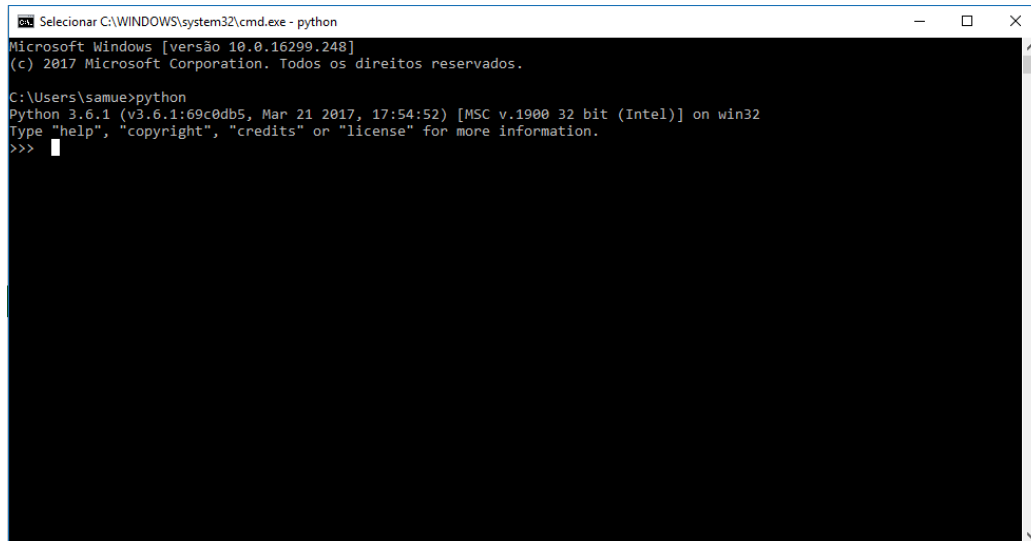
2.2.1 Instalação no Windows

- Para instalar o Python no sistema operacional Windows, acesse o site www.python.org. Depois clique em *Download* e na página que for aberta clique *Download Python*, este download será o da última versão disponibilizada, as demais versões se encontram disponíveis logo abaixo.
- Ao terminar o download clique em executar, coloque tudo em *default*, que seria a configuração padrão já inserida pelo instalador. Clique em finalizar e pronto: o interpretador do Python estará instalado no computador.

Agora vamos conferir se o python foi instalado com sucesso:

- Clique na tecla 'Windows' e 'r' simultaneamente ou clique com o botão direito em cima do menu iniciar e depois clique em executar;
- No Executar digite 'cmd' e clique no Enter;

- Com isso abrimos o *prompt* de comando do Windows. Para verificar se o Python foi realmente instalado basta digitar "python" e dar Enter. Logo após, a seguinte mensagem deve aparecer no *prompt*:



```
Selecionar C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows [versão 10.0.16299.248]
(c) 2017 Microsoft Corporation. Todos os direitos reservados.

C:\Users\samue>python
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figura 2.1: Verificando o sucesso da instalação do Python

Caso não consiga executar o Python no prompt de comando, o *path* (caminho da pasta na qual estão os arquivos de Python) não deve estar configurado. Basta seguir os passos a seguir:

- Vá em Computador, clique com o botão direito, e depois clique em Propriedades;
- Agora, clique em Configurações Avançadas de Sistema e, a seguir, em Variáveis de Ambiente;
- Desça até encontrar a variável Path, clique em Editar;
- Na janela aberta, no campo Valor de Variável, vá ao final da lista que aparece, acrescente um ponto e vírgula “;” e adicione o endereço/caminho de onde a pasta do Python está instalada em seu computador.

Abaixo estão alguns links caso ocorra problemas ou dúvidas no processo de instalação:

- <https://www.youtube.com/watch?v=gDxt7mncWys>
- https://tutorial.djangogirls.org/pt/python_installation/
- <http://excript.com/python/como-instalar-o-python-windows.html>

2.2.2 Instalação no Linux

- Abra o terminal do Linux.
- Digite o comando **sudo apt-get install python3**.
- Digite python3 no terminal para conferir se a instalação foi realizada corretamente.

2.2.3 Instalação no Mac

No Mac é comum o Python já vir instalado de fábrica. Para conferir isso, basta digitar **which python3** no terminal do Mac, onde deve aparecer uma mensagem parecida com essa `/usr/bin/python`. Caso isto não aconteça, basta digitar **brew install python3** no terminal.

2.3 Instalando PIP

O PIP é uma biblioteca do Python, que tem como finalidade nos ajudar a instalar outras bibliotecas com maior facilidade e rapidez. Sua instalação¹ pode ser realizada de forma simples e é feita da mesma maneira em qualquer sistema operacional, como veremos a seguir:

- Primeiro acesse o site <https://pip.pypa.io/en/stable/installing/#id8>;
- Clique com o botão direito no **get-pip.py**[2];
- Depois clique em "Salvar link como...", e selecione para salvar na sua pasta de Downloads;

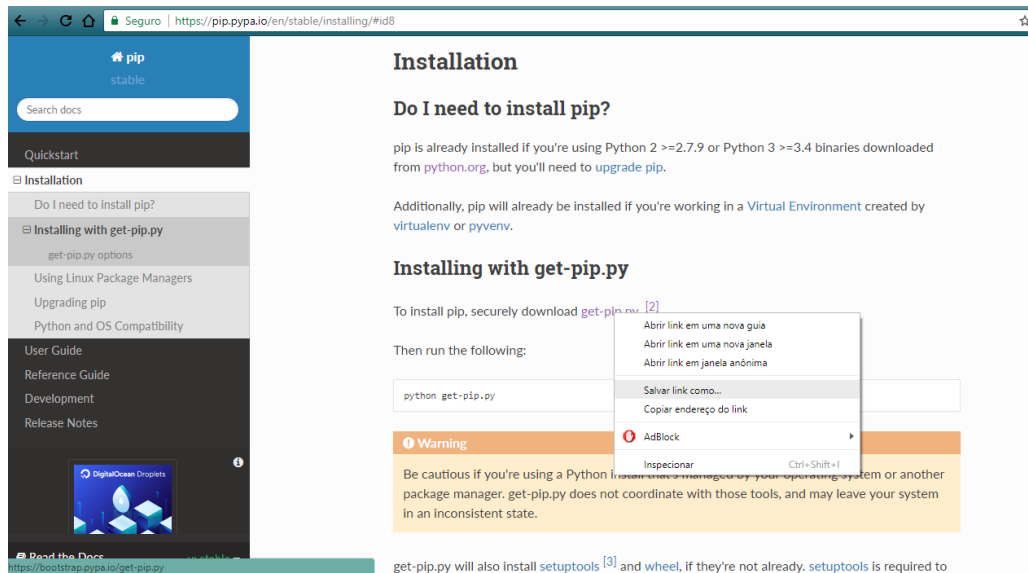


Figura 2.2: Site PIP

- Agora vamos abrir novamente o prompt de comando e digitar **cd Downloads**;
- Por fim, na pasta Downloads basta digitar **python get-pip.py** no prompt de comando.

Links adicionais caso ocorra problemas na instalação:

- <https://www.makeuseof.com/tag/install-pip-for-python/>.
- <https://stackoverflow.com/questions/4750806/how-do-i-install-pip-on-windows>.

2.4 Jupyter Notebook

A plataforma de programação que usaremos nesse curso é a Jupyter Notebook (<http://jupyter.org/index.html>). Essa plataforma será o local onde iremos escrever e rodar nosso código em Python. O Jupyter Notebook apresenta uma interface intuitiva, na qual podemos rodar nossos códigos por blocos de comandos, facilitando o uso. Dessa maneira, é possível ver se determinado pedaço do código realiza a ação que desejamos. Como informação adicional, é legal dizer que o Jupyter roda diretamente no seu navegador com servidor local.

¹Esta parte é focada no Windows. Se você estiver em um Linux ou Mac, por favor abrir primeiro o link disponibilizado mais a frente.

2.4.1 Instalação no Windows

Para instalar o Jupyter Notebook, basta acessar o link: <https://www.anaconda.com/download/> e clicar em "Download". Em seguida, ir seguindo os passos de instalação sem alterar a configuração padrão, apenas certifique-se de estar instalando o anaconda em uma pasta vazia. Após o Anaconda estar instalado em seu computador, basta ir até a pasta onde o programa se encontra e clicar no **Anaconda Prompt**. Em seguida, no prompt digitar **jupyter notebook** e dar Enter, então abrirá uma tela semelhante a Figura 2.3.

2.4.2 Instalação no Linux/Mac

Para instalar o Jupyter Notebook, basta acessar o link: <https://www.anaconda.com/download/> e clicar em "Download". Em seguida no prompt de comando, digite os seguintes comandos: **python3 -m pip install --upgrade** e em seguida **pip python3 -m pip install jupyter**. Para abrir o programa basta digitar **jupyter notebook** no prompt de comando. Com isso teremos o Jupyter instalado, caso aparecer algum erro olhe alguns dos links abaixo para ajudá-lo:

- <http://jupyter.readthedocs.io/en/latest/install.html>
- <http://jupyter.org/install>
- <https://www.youtube.com/watch?v=qRixkfuYp88>

2.4.3 Conhecendo o Jupyter Notebook

Na plataforma iremos encontrar todas as pastas do nosso computador. Primeiramente, é importante criar uma pasta onde você trabalhará. Após criar essa pasta no seu computador, ache o caminho dela no Jupyter Notebook e depois clique em **new** no canto superior e em seguida Python 3. Assim será criado seu notebook com Python.

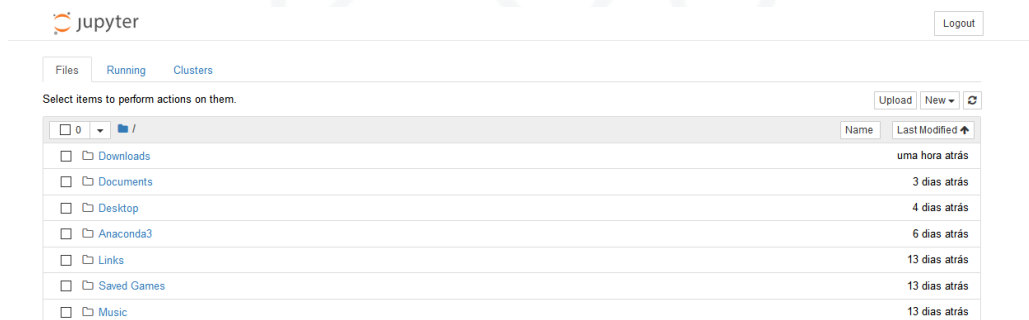


Figura 2.3: Layout da tela inicial do Jupyter Notebook

Com o processo de instalação finalizada, vamos entender melhor o que é um notebook: ele é a plataforma onde faremos nossa programação em Python. Nele que poderemos separar nosso código em células de comando e executá-las separadamente. Conforme você for programando no Jupyter Notebook perceberá que é possível mesclar código (Python) com textos (Markdown/LaTeX), que podem deixar os notebooks criados mais autoexplicativos.

Para criar seu primeiro código no Jupyter Notebook, basta ir no notebook que foi criado anteriormente, digitar o código a ser executado, podendo ser apenas uma linha de comando quanto várias linhas, em seguida basta clicar em **Run** no canto superior para ter seu código compilado, e conferir o resultado. Caso ocorra erros no compilamento do código, aparecerá uma descrição dos erros.

```
In [92]: a = "Olá Mundo"
In [93]: a
Out[93]: 'Olá Mundo'
```

Figura 2.4: Primeiro programa no Jupyter Notebook

Para alterar um bloco do Jupyter para aceitar textos, basta alterar a célula na caixa superior onde está escrito **Code** para **Markdown**

Podemos também baixar nossos códigos ou textos, clicando em **File** e depois clicar em **Download as**, onde irá aparecer todos os formatos de arquivo no qual podemos baixar o notebook.

Para abrir um notebook externo, como por exemplo, o notebook da aula que será dada, basta navegar entre as pastas até o local onde está salvo o arquivo no seu computador, e então dar dois cliques no arquivo, em seguida o arquivo será aberto.

Links adicionais para aprender a utilizar melhor o Jupyter Notebook:

- <https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook>.
- <https://www.youtube.com/watch?v=xuahp9g3Dic>.

2.5 Vamos programar?

2.5.1 Variáveis

Uma variável é um nome qualquer atribuído a uma posição da memória cujo espaço é reservado a um determinado conjunto de dados. Ou seja, é um local onde registraremos informações que serão utilizadas posteriormente. No Python, a sua declaração se dá da seguinte maneira:

```
var = value
```

Sendo, neste caso, `var` o nome da variável e `value` a informação atribuída a posição da memória que apelidamos de “var”. Seguem alguns exemplos de variáveis declaradas:

```
num = 12
nf = 12.2
w = "world"
```

O nome atribuído a uma variável é arbitrário, desde que esteja de acordo com as regras abaixo:

- A primeira letra da variável deve ser uma letra ou uma underline;
- Não pode haver espaços em seu nome;
- Elas são “case sensitive”. Ou seja, há distinção entre letras maiúsculas e minúsculas. Se declarada como `aux`, não pode ser referenciada como `Aux` ou qualquer outra distorção do gênero;

De maneira geral, as variáveis são divididas em strings, integers e floats. Essa divisão ocorre em virtude da natureza dos dados que serão registrados na memória. Strings são variáveis que reúnem sequências de caracteres, integers números pertencentes ao conjunto dos inteiros e floats números pertencentes ao conjunto dos reais. Como pode ser percebido nos exemplos acima, o tipo das variáveis, no Python, é definido de maneira implícita, sem necessidade de referenciar o seu tipo no momento em que a variável é declarada.

```
>> type(12)
<class 'int'>
>> type(12.2)
<class 'float'>
>> type("world")
<class 'str'>
```

As variáveis podem ser criadas de duas maneiras, definindo-as diretamente - como nos exemplos acima - ou pegando-as como retorno de uma função. Vamos empregar esse segundo método tomando a função `input()` - que tem como finalidade registrar informações submetidas pelo usuário - como exemplo:

```
x = input("Digite o seu nome: ")
print("O valor digitado foi {} e o seu tipo é {}".format(x, type(x)))
```

output:

```
Digite o seu nome: Neuron
O valor digitado foi Neuron e o seu tipo é <class 'str'>.
```

Além disso, é possível transformar o tipo da variável por meio de funções. Por exemplo:

```
idade = input("Digite a sua idade: ")
print(type(idade))
print(type(int(idade)))
```

output:

```
<class 'str'>
<class 'int'>
```

No caso acima, uma variável do tipo string foi convertida para integer através da função `int()`. Por fim, é importante ter em mente que os nomes dados às variáveis devem ser intuitivos, tendo uma relação com a informação que registrará. Por exemplo, se a variável armazenará um nome, faz mais sentido chamá-la de “nome” em vez de `variavel1`.

2.5.2 Manipulação de Strings

Como adiantado brevemente no capítulo sobre variáveis, string nada mais é do que uma sequência de caracteres. No python, uma string pode ser identificada pelo característico meio de declará-la:

```
_string = "Isso e uma string"
type(_string)
```

O objetivo desse capítulo é introduzir as técnicas básicas de manipulação de string, uma vez que essas assumirão um papel fundamental em situações que dependam de processamento do texto. O Python é uma das linguagens mais poderosas quando o assunto é strings, ele vem com um conjunto variado de funções built-in voltadas a esse tipo de dado, segue abaixo as principais:

- `len()`: Retorna a quantidade de caracteres sequenciados dentro de determinada string.

```
len("hello world")
```

- `string.find()`: Encontra a posição que determinada sequência de caracteres está na string, vale lembrar que a contagem começa a partir do 0 e não do 1. Caso a sequência buscada não esteja presente na string, a função retornará -1.

```
X = "hello world"
print(X.find("world"))
print(X.find("horse"))
```

- `string.split()`: Divide a string em partes e retorna-as em uma lista, a separação ocorre por um divisor definido como parâmetro no momento em que a função é chamada, caso não seja definido um separador, a divisão ocorrerá a cada espaço.

```
x = "data science"
print(x.split())
print(x.split("a"))
```

Além das funções acima, é possível fazer recortes na string e trabalhar somente com uma parte específica:

```
m = "Hello world"
m0 = m[1:8]
print(m0)
```

output:

```
ello wo
```

No exemplo acima, o 1, entre os colchetes, é a posição inicial do recorte e 8 a final. Ou seja, os caracteres posicionados entre 1 e 8 serão selecionados. Outro recurso essencial é o concatenamento de strings, e é feito da seguinte forma:

```
a = "xy"
b = "z1"

c = a + b

print(c)
```

2.5.3 Operadores

Para realizarmos as operações no python usamos os operadores, eles nos permitem realizar operações matemáticas e operações lógicas, abrindo assim um grupo enorme de possibilidade. Os operadores matemáticos que nos permitem fazer somas, subtrações, divisões entre outras operações. Já os operadores lógicos são aqueles no qual é feita uma comparação e a resposta do linguagem sera Verdadeiro(true) ou Falso(false).

Operadores Matematicos

```
>> a = 5
>> b = 6
>> a+b
>> a-b
>> a/b
```

Operadores Logicos

```
>> 5 > 6
>> 5 = 6
>> 5 < 6
>> 5 != 6
```

2.5.4 Condicionais

Um aspecto intrínseco ao desenvolvimento de softwares é o processo decisório. No python, este processo se dá por meio da avaliação de expressões booleanas(expressões cujo resultado é TRUE ou FALSE). Ou seja, apresentada uma condição, se a expressão booleana der true, o programa seguirá um caminho, caso dê false, seguirá outro. A estrutura decisória é apresentada da seguinte forma: se uma expressão for verdadeira:

<executará um bloco de código>

se outra expressão for verdadeira:

<executará um outro bloco>

senão:

<executará um bloco de código>

Adiante alguns exemplos para ilustrar o funcionamento das condições:

```
a = 10
```

```
if(a<=10):
    print("Condição verdadeira")
```

No caso acima, caso a variável seja igual ou inferior a 10, o texto “Condição verdadeira” será impresso. Além do if, há também o elif que é empregado em situações onde há várias possíveis condições e é empregado da seguinte forma:

```
a = 20
```

```
b = 20
```

```
if a<b:
    print("{} é menor do que {}".format(a,b))
elif a==b:
    print("O numero {} e igual a {}".format(a,b))
```

output:

O número 20 é igual a 20.

Vale ressaltar que somente um bloco dos elencados no if será executado, no caso, o primeiro que se adequar à condição imposta, por exemplo:

```
aux = 19

if aux > 13:
    print("O valor armazenado na variavel é maior do que 13 .")

elif aux > 14:
    print("O valor armazenado na variavel é maior do que 14.")
else:
    print("A variável tem armazenado um valor inferior a 13.")
```

O programa retornará o texto “O valor armazenado na variável é maior do que 13” mesmo a variável tendo consigo um valor que se adéqua a duas condições justamente por essa característica.

2.5.5 Manipulação de arquivos

A manipulação de arquivos é algo muito prático no python, ela é usada quando queremos usar um arquivo exógeno (.txt, por exemplo) como entrada de dados do código, ou seja, o dado não vem do usuário e não está colocado no código fonte e vem de um lugar externo. Para vermos o conteúdo de um arquivo é necessário seguir alguns passos com funções do python. Primeiramente abrimos o arquivo e o armazenamos em uma variável - para isso utilizamos a função open, sendo que sua sintaxe fica da seguinte forma:

```
var= open("nome_do_arquivo.txt", tipo_manipulacao)
```

Após a abertura, sempre que quisermos manipular o arquivo usaremos a variável como referencia do arquivo. Na sintaxe da função, caso o arquivo esteja na mesma pasta do código basta colocar o nome dele, já se estiver em outro diretório, basta colocar o caminho até ele. Caso o arquivo passado para a função não existir, ela mesma criará o arquivo. É possível fazer dois tipos de manipulação, ou seja, escrever e ler um arquivo - elas são marcadas pela letras “w” (de write) e “r” (de read). portanto ao colocar no tipo de manipulação uma das duas letras escolhemos o tipo de manipulação.

Com o arquivo aberto podemos realizar as operações com ele. Primeiramente vamos escrever nele:

```
var=open("nome_do_arquivo.txt", "w")
```

Podemos adicionar textos ao arquivo de duas formas diferentes, podemos escrever uma string com tudo que quisermos e depois adicionar essa string no final da seguinte forma:

```
aux= "Neuron é animal!"
var.write(aux)
```

Ou adicionar linha por linha com:

```
var.writelines("Neuron")
var.writelines("é")
var.writelines("animal!")
```

Neste segundo caso é necessário informar quando se deve pular linha ou seja colocar um “\n”, ao final de cada linha, ou qualquer manipulação da string que seja desejada. Essa função também aceita listas (que serão abordadas em breve) como entrada.

Em relação à função de ler o arquivo, a usamos para ler e pegar os dados do arquivos (para printar na tela o conteúdo, por exemplo). Para isso usamos a função:


```
var=open("nome_do_arquivo.txt","r")
texto = var.read()
```

Para ver seu conteúdo, basta dar print nesta variável que recebeu os dados do arquivo. Podemos também pegar os dados do arquivo linha por linha, para isso usamos:

```
texto = var.readline()
```

Ela transforma cada linha do arquivo em um item de lista.

Ao final de qualquer operação, temos sempre que fechar o arquivo que foi aberto, pois é ao fechar o arquivo que consolidamos as ações feitas. Isso é feito com:

```
var.close()
```

Em breve abordaremos uma maneira muito mais eficiente de trabalhar com arquivos.

2.5.6 Loops

Em Python é possível iterar sobre os itens de qualquer tipo de sequência, as mutáveis (listas, dicionários, ...) e também as imutáveis (tuplas, range, strings, ...) na ordem em que eles aparecem na sequência de maneira simples, isto é, repetir o mesmo bloco de comandos enquanto uma condição está sendo satisfeita. Existem dois tipos principais de loops no Python: o *for* e o *while*. O *for* repete o bloco de comando até que um limite estabelecido seja atingido (e.g. "para i começando de 0 até i=4, faça alguma coisa"), a sintaxe é feita da seguinte forma:

```
for <variavel> in <sequencia>:
    <bloco de código>
```

Exemplo:

```
for item in [1, 2, 3, 4, 5, 6]:
    print (item**2)
```

1
4
9
16
25
36

Exemplo 2:

```
for x in range(1, 10):
    if x % 2 == 0: # repare que o X assume os valores dentro do range
        print x
```

2
4
6
8

O *while*, por sua vez, faz as iterações até que uma condição não seja satisfeita mais (e.g. "enquanto a condição for satisfeita, continue no loop") e sua sintaxe é da seguinte maneira:

```
while <condição>:  
    <bloco de código>
```

Exemplo:

```
i = 0  
while ( i < 3):  
    print ("i vale: ",i)  
    i = i + 1  
print("Saiu do loop")  
  
i vale: 0  
i vale: 1  
i vale: 2  
Saiu do loop
```

2.5.7 Listas

Listas em Python são estruturas de dados ordenadas que podem armazenar qualquer tipo de valor, sendo eles inteiros, strings, floats, funções, classes e outros. É importante dizer que cada valor é identificado por um índice.

- Podemos dizer que as Listas são parecidas com os arrays de outras linguagens de programação;
- As listas não armazenam realmente os objetos dentro dela, mas sim a referência de memória desses objetos;

Declaramos uma lista da seguinte maneira:

```
letras = ['a', 'b', 'c']
```

Ou usando a função built-in list passando uma tupla como argumento:

```
letras = list(('a', 'b', 'c'))
```

O comprimento da lista pode ser obtido através do comando "len":

```
len(letras)
```

Para acessar os elementos colocados em uma lista basta colocar o índice que o elemento se encontra na lista - esses índices são referentes a ordem que são inseridos na lista e a contagem desses índices no python começa a partir do 0, portanto:

```
letras[0] # retorna o primeiro item da lista  
          'a'  
letras[2] # retorna o terceiro item da lista  
          'c'  
letras[-1] # retorna o último elemento de lista,  
            'c' #repare que a contagem dos índices de trás para frente inicia em -1
```

Podemos também pegar partes dessa listas, "cortes" das partes que desejarmos, chamamos isso de Slicing, que podemos fazer da seguinte forma:

```
letras[1:3] # retorna elementos a partir do índice 1 até o índice 2

['b', 'c']

letras[:2] # retorna elementos a partir do índice 0 até o índice 1

['a', 'b']

letras[-2:] # retorna elementos a partir do índice -2(penúltimo item) até o fim.

['b', 'c']
```

Podemos também usar listas diretamente com os loops, tornando sua manipulação mais fácil:

```
for item in letras:
    print(item)

'a' # 1 volta
'b' # 2 volta
'c' # 3 volta
```

Ademais, é possível trabalhar com o índices dessas listas nos loops:

```
for index, item in enumerate(letras):
    print (index, item)

0 'a' # 1 volta
1 'b' # 2 volta
2 'c' # 3 volta
```

Podemos verificar se um elemento pertence a uma lista facilmente. Para isso, utilizamos o operador *in*, que testa se o elemento está dentro de uma determinada sequência. O operador serve tanto para listas, como para qualquer outro tipo de sequência, como por exemplo strings, dicionários e outras:

```
if 'b' in letras:
    print 'b esta na lista letras'

'b' esta na lista letras'
```

O python nos provê, de forma nativa, algumas funções para trabalharmos com as listas. Por exemplo, as funções de inserção que são frequentemente utilizadas:

```
letras = ['a', 'b', 'c']
letras.append('d')
letras

'a', 'b', 'c', 'd'
```



```
letras.append(['e', 'f'])
letras
```

```
'a', 'b', 'c', 'd', ['e', 'f']
```

Repare que foi adicionado uma lista na última posição:

```
letras.extend(['g', 'h'])
letras
```

```
'a', 'b', 'c', 'd', ['e', 'f'], 'g', 'h'
```

Veja que ao contrário de ".append" o método ".extend" decompõe a lista que foi passada como argumento e coloca um item por posição na lista original. Podemos também escolher onde desejamos inserir algo na lista:

```
letras.insert(1, 'w')
letras
```

```
'a', 'w', 'b', 'c', 'd', ['e', 'f'], 'g', 'h'
```

Outra função muito importante que o python nos fornece são as de remoção de elementos de lista, que podemos fazer isto da seguinte maneira:

```
letras = ['a', 'b', 'c', 'a']
letras.remove('a')
letras
```

```
['b', 'c', 'a'] # repare que apenas a primeira ocorrência de 'a' foi removida
```

Ou podemos remover o último elemento de uma lista com a função pop:

```
letras.pop() # remove o último elemento da lista e o retorna.
'a'

letras
['b', 'c']
```

Caso tente remover um item que não pertence a uma lista ocorrerá erro. Podemos também realizar operações com listas de maneira muito simples:

```
n = [1] + [2, 'a'] # somar duas listas, equivalente à [1].extend([2, 'a'])
n
```

```
[1, 2, 'a']
```

```
n += [3] # equivalente à n.extend([3])
```

```
[1, 2, 'a', 3]
```

```
n = ['a', 'b']
n * 3 # equivalente à ['a', 'b'] + ['a', 'b'] + ['a', 'b']

['a', 'b', 'a', 'b', 'a', 'b']
```

Uma importante característica de listas em python é que elas são **objetos mutáveis** - o que isso significa? Um Objeto mutável é aquele que pode ser alterado após a sua criação, ou seja, podemos alterar o estado do objeto sem termos que criar uma cópia do mesmo. Por exemplo:

```
n = ['a', 3.14]
id(n) # a função built-in retorna o endereço de memória do objeto.

1234567 # endereço de memória simbólico.

n.append('b')
n

['a', 3.14, 'b']

id(n)

1234567 # o endereço de memória não foi alterado, ou seja, é o mesmo objeto ainda.

n.reverse() # inverte a lista

['b', 3.14, 'a']

id(n)

1234567 # de novo o endereço de memória não foi alterado.
```

Podemos também referenciar uma lista, isto é, duas variáveis apontarem para a mesma lista, fazendo que as ações feitas em uma variável também vai refletir na outra.

```
n = ['a', 'b']
w = n
w, n

(['a', 'b'], ['a', 'b'])

id(n), id(w)

(1234567, 1234567) # repare que o endereço de memória é o mesmo para 'n' e 'w'

w.append('c')
w, n
```

```
(['a', 'b', 'c'], ['a', 'b', 'c']) # importante: 'n' também foi alterado, pois 'n' e 'w' apontam para a mesma memória

id(n), id(w)
```

```
(1234567, 1234567) # mesmo endereço de memória
```

Se tudo é referência então como criar uma cópia de uma Lista?

```
n = ['a', 'b']
w = n[:] # cria uma cópia de 'n'
id(n), id(w)

(1234567, 9876543)
```

Observe que o endereço da memória não é mais o mesmo, ou seja, temos duas variáveis, cada uma apontando para seu endereço de memória.

2.5.8 Funções

Um dos mais importantes conhecimentos que um programador deve ter é sobre o uso de funções. Elas são blocos de comandos que podem ser usados repetidas vezes em um programa, além de deixar o programa muito mais organizado. Ao longo desta apostila foi falamos muito de funções, sendo que as mencionadas anteriormente não são muito diferentes dessas que iremos abordar agora. A maior diferença é que as já faladas foram implementadas pelos desenvolvedores da linguagem e agora vamos te ensinar a fazer suas próprias funções!

As funções em python são caracterizadas pelo comando "def" seguido pelo nome da função e os argumentos que ela recebe, assim indicando o início de uma função:

```
def minha_funcao():
```

Chamamos de argumento de uma função os elementos que serão manipulados dentro de uma função, sendo esses elementos de qualquer tipo oferecido pela linguagem(listas, strings, etc):

```
def ola(meu_nome):
    print("ola "+meu_nome+"!")
```

Um outro aspecto importante que notamos no exemplo acima é que as funções também obedecem a indentação dos demais comandos como listas e loops. Geralmente, as funções não são blocos de comandos muito extensos, porém isso não é uma regra geral, apenas uma boa prática de programação.

As linhas de execução de uma função são interpretadas de maneira separada das demais linhas do código, isso significa que qualquer operação que for feita na função existirá apenas dentro da função. Para usarmos o resultado gerado por uma função usamos o comando "return" - ele permite fazer que esse resultado seja mandado para uma variável fora da função:

```
def media(x1, x2):
    return (x1+x2)/2
```

Para usarmos a função criada é bem simples: basta chamar o nome da função e passar os argumentos necessários para que ela funcione:

```
x = media(1,3)
print(x)
```

2

2.5.9 Como usar o PIP e import de bibliotecas

O pip é uma biblioteca do Python que tem como função facilitar a instalação de outras bibliotecas (ou pacotes). Para usá-lo no Python 3.6, devemos executar linhas de código do seguinte layout no Prompt de Comando ou na janela Executar (acessível apertando as teclas Win+R no Windows):

pip install nomeBiblioteca

Ou caso o Python não esteja adicionado como variável de ambiente no PATH do Windows:

python -m pip install nomeBiblioteca

Exemplo:

python -m pip install numpy

Alguns dos pacotes e bibliotecas podem ter um nome diferente do conhecido, por isso sempre procura sobre a biblioteca antes de instalar.

Para importarmos as bibliotecas que usaremos no Python, usamos o seguinte comando no Python:

```
import biblioteca
```

Exemplo:

```
import numpy
```

Para importar uma classe específica de uma biblioteca, usamos

```
from biblioteca import classe
```

Ou alternativamente,

```
import biblioteca.classe
```

Também podemos “renomear” as bibliotecas que importamos para facilitar o seu uso:

```
import biblioteca as nome
```

Exemplo:

```
import numpy as np
```

Para os tópicos a seguir e no restante do curso iremos usar as bibliotecas pandas e numpy.

2.5.10 Numpy

Numpy é um biblioteca do Python que nos permite trabalhar com vetores, matrizes e tensores de forma geral (arrays) com mais eficiência do que as ferramentas nativas da linguagem.

Quando trabalhamos com vetores e matrizes no Python com funções nativas usamos o conceito de listas, isso significa que o processo por trás é diferente do caso em que usamos o conceito de array do pacote Numpy, por exemplo. Geralmente quando optamos por trabalhar com arrays ao invés de listas, os processos ficam mais rápidos.

As diferenças por operações podem ser de frações de segundos o que pode parecer pouco, mas quando passamos a trabalhar com processos mais complexos, essa diferença passa a crescer exponencialmente. Pelo fato de se trabalhar com o pacote Numpy, vamos apresentá-lo. Iremos abordar aqui apenas algumas funções da biblioteca, no entanto, existem muitas mais que podem ser encontradas na documentação da biblioteca:

<https://docs.scipy.org/doc/numpy/reference/index.html>

Ao importar a biblioteca é normalmente usado as letras "np" para se abreviar e a maioria de suas funções são chamadas após essa abreviatura. Vamos criar um array para exemplificar:

```
x=np.array([1,2])
```

O array criado acima chamamos de array unidimensional ou vetor, pois todos os dados estão no mesmo segmento, porém podemos criar um array com quantas dimensões quisermos. Vamos criar um array bidimensional, basicamente uma matriz:

```
arrayB=np.array([[1,2,3],[4,5,6],[7,8,9]])
```

Existem outras maneiras de se criar matrizes com a Numpy, inclusive algumas matrizes bem específicas, como uma matriz diagonal, uma matriz identidade, etc:

```
MatrizIdentidade = np.eye(2) # argumento vai quantidade de 1s que queremos  
print(MatrizIdentidade)
```

Podemos também mudar o formato de um array como comando arrayShape, transformando um vetor em uma matriz, por exemplo:

```
arrayShape = np.array([1,2,3,4])  
print("Array com formato inicial: ",arrayShape)  
print("Dimensões do array: ",arrayShape.shape)
```

output:

```
Array com formato inicial: [1 2 3 4]  
Dimensões do array: (4,)
```

```
arrayShape.shape =(2,2)  
print("Array modificado: ")  
print(arrayShape)  
print("Dimensões do array: ",arrayShape.shape)
```

output:

```
Array modificado:  
[[1 2]  
 [3 4]]  
Dimensões do array: (2, 2)
```

Podemos também extrair dados dos arrays de maneira muito simples, dados como media, variancia, desvio padrão. Podemos conseguir também a inversa e a transposta de uma matriz de maneira bem simples, por exemplo.

Como mostramos, a biblioteca Numpy tem muitas funcionalidades e essas são apenas um pequeno pedaço de tudo que se é possível fazer.

2.5.11 Pandas

Conheceremos agora uma das bibliotecas mais importantes e mais usadas para fazer a manipulação dados no Python - o Pandas. O Pandas oferece diversas funções que facilitam a manipulação e visualização dos dados. O Pandas facilita a visualização de dados por causa da estrutura que ele é capaz de comportar as bases de dados - essa estrutura é chamada de DataFrame. Geralmente os DataFrames são construindo abrindo arquivos do formato ".csv" de organização dos dados. Os arquivos .csv são arquivos com informações separadas por vírgulas dispostas em colunas. A biblioteca Pandas entende automaticamente esses dados e os converte em Data Frame.

Formato de um .csv:

descrição1, descrição2, ..., descriçãoN

dado da descrição1, dado da descrição2, ..., dado da descriçãoN

dado da descrição1, dado da descrição2, ..., dado da descriçãoN

dado da descrição1, dado da descrição2, ..., dado da descriçãoN

Exemplo:

País,Capital,Área,População

Brasil,Brasília,8.516,200.4

Rússia,Moscou,143.5,17.100

Podemos criar um arquivo .csv de diversas maneiras e as mais fáceis são editar um bloco de notas e salvá-lo como .csv ou podemos ir no Google Planilhas/Excel e colocar cada dado em uma coluna e na hora de baixar ou salvar, escolher a opção "csv separado por vírgulas". No entanto, geralmente trabalharemos com arquivos csv já prontos, criados por terceiros.

Ao importar o pandas é comum usar "pd" como abreviação de pandas. Inicialmente, para se utilizar o pandas deve-se abrir o arquivo ".csv" dentro de um variável, para isso se usa o comando:

```
dataset = pd.read_csv("Nome_dataset.csv")
```

Com o arquivo referenciado por essa variável utilizamos os comandos da biblioteca em cima dela. Por exemplo, podemos utilizar o comando "head" para visualizar os primeiros dados do dataset:

```
dataset.head()
```

Podemos também obter dados muito úteis do dataset de maneira bem simples, como quantos dados existem por coluna, a média dos dados, a variância, o desvio padrão. Outra função essencial é que podemos fazer consultas no dataset e filtrar dados.

Pode-se encontrar mais informações sobre a biblioteca em sua documentação que é bem vasta:

<https://pandas.pydata.org/pandas-docs/stable/>

2.6 Exercícios complementares ao conteúdo de Introdução ao Python

1. Com a frase "Estou aprendendo Python com o Neuron" faça o seguinte: separe palavra por palavra armazenando-as em variáveis distintas e depois concatene todas printando a frase para o usuário.
2. Faça um código onde o usuário entre com a idade e diga se ele pode ou não tirar licença de motorista.
3. Crie uma função que diga se um número é ímpar ou par.
4. Crie uma lista apenas com números ímpares e outra somente com números pares e uma com números primos (range (0,50)). Use condicionais e faça o computador trabalhar por você.
5. Crie uma função que receba um número de 1 a 10 e retorne sua tabuada.
6. Printar uma matriz número por número.
7. Faça um jogo da forca, de forma que 2 pessoas joguem e uma delas entra com a palavra. Após isso a outra deverá adivinhar as letras, lembrando que com 6 erros a pessoa perde. O jogo deverá mostrar as letras que já foram tentadas, e conforme os acertos forem acontecendo, o programa deverá mostrar a letra no seu devido espaço até a palavra se formar.
8. Estude a função *random* do Python e após isso crie uma lista de 10 valores preenchidos de forma aleatória. Então:
 - Dê o maior valor da lista;
 - Dê o menor valor da lista;
 - Dê a soma dos valores da lista;
 - Dê o tamanho da lista;
 - Dê a lista em ordem crescente;
 - Dê a lista em ordem decrescente;
9. Peça para o usuário inserir uma frase e faça o seguinte com ela:
 - Tornar todas as letras maiúsculas;
 - Retire os espaços entre as palavras;
 - Realize as duas tarefas conjuntamente.
10. Pegue uma poesia qualquer, coloque em um arquivo .txt e dê print usando a função *readlines*.
11. Com o poema do exercício anterior, escolha uma palavra que aparece com grande frequência e faça um programa que conte quantas vezes essa palavra aparece no poema inteiro;
12. Em seguida, como continuação do exercício anterior, faça um programa que troque a palavra escolhida anteriormente por "Neuron" e depois reescreva o arquivo .txt com as modificações feitas.

2.7 Exercícios de Desenvolvimento em Python

Estes exercícios tem a finalidade de avaliar seu conhecimento e habilidades de programação em Python, resolva quais conseguir.

2.7.1 Iniciante

1. Escreva um algoritmo que armazene o valor 999 na variável *a* e o valor 555 na variável *b*. A seguir (utilizando apenas atribuições entre variáveis) troque os seus conteúdos fazendo com que o valor que está em *a* passe para *b* e vice-versa.
2. Escreva um algoritmo para ler um valor (do teclado) e escrever (na tela) o seu antecessor.
3. Ler um valor e escrever a mensagem **É MAIOR QUE 10!** se o valor lido for maior que 10,

caso contrário escrever **NÃO É MAIOR QUE 10!**

4. Ler dois valores e imprimir uma das três mensagens a seguir: ‘Números iguais’, caso os números sejam iguais ‘Primeiro é maior’, caso o primeiro seja maior que o segundo; ‘Segundo maior’, caso o segundo seja maior que o primeiro.
5. Ler três valores (considere que não serão informados valores iguais) e escrever o maior deles.
6. Ler três valores (considere que não serão informados valores iguais) e escrever a soma dos dois maiores.
7. Ler três valores (A, B e C) representando as medidas dos lados de um triângulo e escrever se formam ou não um triângulo. **OBS:** para formar um triângulo, o valor de cada lado deve ser menor que a soma dos outros dois lados.
8. Faça um algoritmo que leia a idade de uma pessoa expressa em anos, meses e dias e escreva a idade dessa pessoa expressa apenas em dias. Considerar ano com 365 dias e mês com 30 dias.
9. Faça um algoritmo para dizer se um número é par ou ímpar.
10. Faça um jogo onde é sorteado um número aleatório de 0 a 100(incluindo 100) e o jogador deve tentar acertar este número, a cada tentativa do jogador deverá ser informado do seguinte:
 - Parabéns você ganhou!(caso ele acerte o número)
 - O número é maior! (caso o número for maior)
 - O número é menor! (caso o número for menor)
 - Faça um menu de entrada para o jogo

Opcional: Faça que ao final do jogo seja possível jogar novamente sem ter que fechar o prompt. **Dica:** Estudar função Random e a sua seed.

2.7.2 Intermediário

1. Faça um algoritmo que preencha uma matriz e print número por número, depois faça uma matriz binária (apenas 0 e 1) e faça que print apenas os número 1 e onde seja 0 esteja vazio.
2. Faça uma função que print o Fibonacci de um número. **Opcional:** fazer sem recursão.
3. Concatenar duas listas em ordem crescente. Faça para listas de tamanhos iguais e diferentes
4. Faça um algoritmo que multiplique duas matrizes se possível (faça para matrizes não quadráticas).
5. Faça uma função que:
 - Leia arquivos txt
 - Edite arquivos txt
 - Salve arquivos txt
6. Dado uma lista de tamanho X , preenchidas de forma não ordenada, faça um algoritmo que faça uma busca por um valor , caso ele não esteja na lista mande uma mensagem falando isso. **Dica:** Primeiro ordene a lista.
7. Faça uma função que descubra se um número é primo.
8. A prefeitura de uma cidade deseja fazer uma pesquisa entre seus habitantes. Faça um algoritmo para coletar dados sobre o salário e número de filhos de cada habitante e após as leituras, escrever:
 - Média de salário da população
 - Média do número de filhos
 - Maior salário dos habitantes
 - Percentual de pessoas com salário menor que R\$ 150,00

Faça uma função para as letras A e B, outra para C e uma para D, faça um menu para escolher

qual deve ser usada, e deixe uma opção para o cadastro dos habitantes e faça que seja salvo em um arquivo.

2.7.3 Avançado

Usando conceito de Programação Orientada a Objetos resolva os exercícios a seguir:

1. Faça um algoritmo que resolva o jogo resta um. **Dica:** pesquise sobre busca em profundidade.
2. Dado uma matriz labirinto como na imagem abaixo, faça um algoritmo que diz se um labirinto tem ou não saída. **Opcional:** Fazer uma função que mostra o caminho de saída do labirinto.

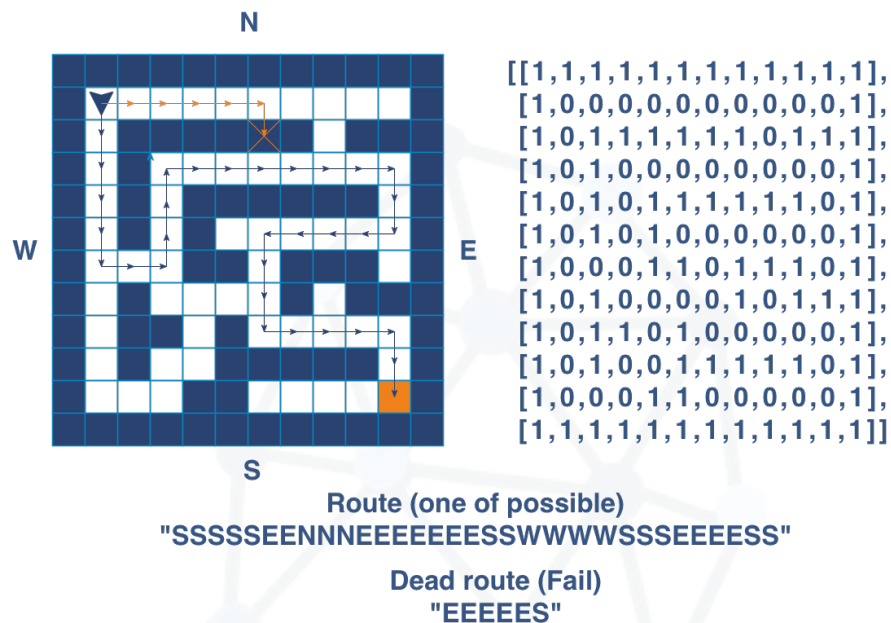


Figura 2.5: Labirinto