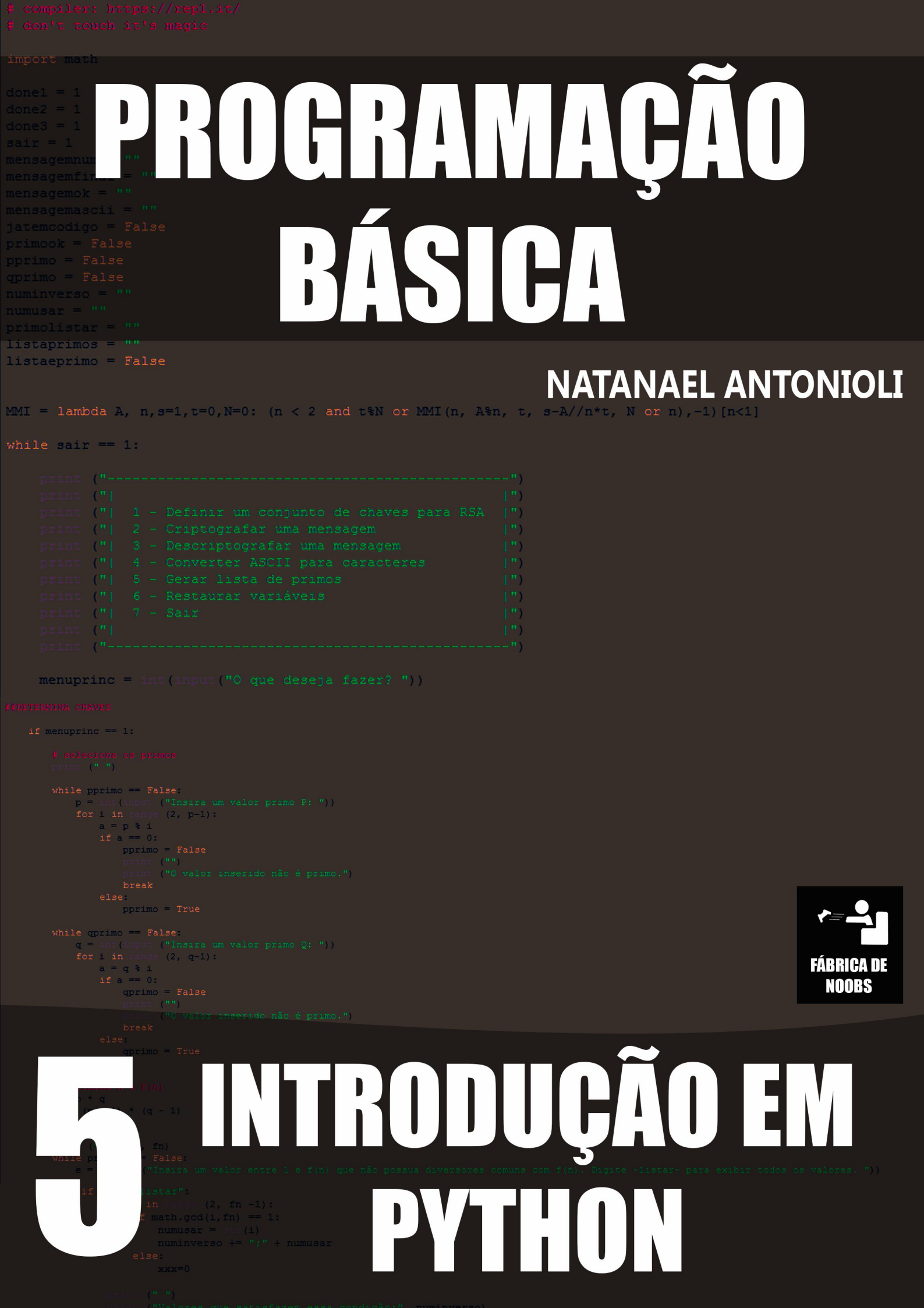
PROGRAMAÇÃO BÁSICA

NATANAEL ANTONIOLI



5 INTRODUÇÃO EM PYTHON

I. Introdução

Python é uma linguagem de programação de alto nível, lançada em 1991. Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation.

A linguagem foi projetada com o ideal de priorizar a legibilidade do código sobre a velocidade ou expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros.

É amplamente divulgada como uma das melhores – se não a melhor – linguagem de programação para iniciantes, em função da sua sintaxe limpa, intuitiva e comunidade ampla, sempre disposta à tirar dúvidas de usuários novos.

Essa apostila é voltada para você, que procura aprender o básico da linguagem Python.

Meus agradecimentos ao Naccib e ao Oddy, que revisaram este material.

Boa leitura!

II. Obtenção

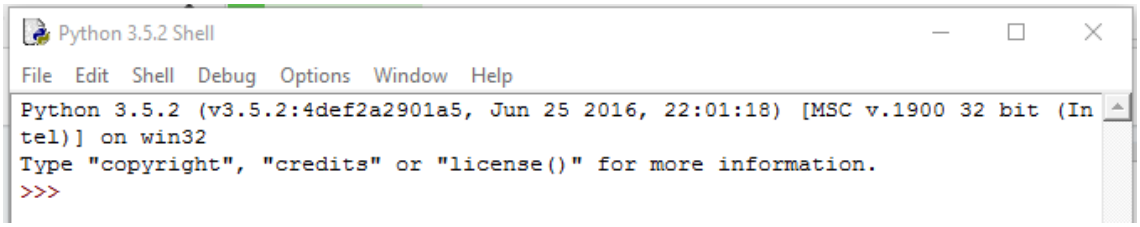
Se você é usuário de Linux ou OS X, o compilador Python já vem instalado por padrão, e scripts escritos na linguagem são executados sem maiores problemas.

Para usuários de Windows, o interpretador deve ser baixado em https://www.python.org/downloads/.

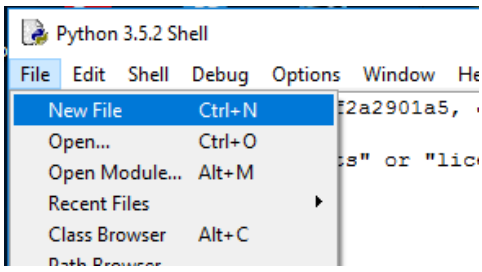
A linguagem possui duas versões: Python 2.x, desenvolvido em 2008, e Python 3.x, desenvolvido em 2015. Essa última possui diversos aspectos de seu núcleo adaptados para usuários iniciantes, e novos programas são escritos nelas.

Sendo assim, nesta apostila, usaremos como base a linguagem Python 3.x. Porém, um usuário de uma versão consegue, sem maiores problemas, adaptar sua escrita para a outra.

Uma vez instalado o interpretador, sua simples execução exibirá uma espécie de terminal, que pode ser usado para experiências simples com a linguagem, como operações matemáticas.



Clicando na aba File, pode-se criar um novo arquivo, que será lido e executado em Python.



Durante o curso, trabalharemos escrevendo em arquivos como esse. Quando usarmos diretamente o terminal do interpretador, isso será detonado por >>>.

Podemos executar um script já escrito através do menu Run. Cada execução implicará no salvamento do arquivo.

III. Variáveis

Em Python, não é necessário declarar variáveis antes de seu uso, nem definir seu tipo – isso já é feito automaticamente. Elas podem ser de três tipos:

- *int* para um número inteiro
- *float* para um número decimal
- *string* para uma sequência de caracteres.

Por exemplo, podemos declarar uma variável *a* com valor *6*, efetuando:

```
>>> a = 6
```

Caso desejarmos definir o valor da variável como resultado de uma operação matemática, como *2 * 3*, podemos fazer:

```
>>> a = 2 * 3
```

O mesmo vale para números decimais, como nos dois exemplos abaixo. Lembre-se

```
>>> b = 1.5
```

```
>>> b = 0.5 + 1
```

Já para strings, precisamos colocar o texto desejado entre aspas, simples ou duplas:

```
>>> c = "fabrica de noobs"
```

```
>>> c = 'fabrica de noobs'
```

Podemos inserir uma quebra de linha, dentro de uma variável, através do comando *:\n* no local em que a quebra precisa ser inserida. Por exemplo:

```
>>> d = "curso desenvolvido por :\n fabrica de noobs"
```

O valor de uma variável pode ser mostrado através do comando *print*, que deve seguir a estrutura *print (variável)*. No exemplo abaixo, exibidos todas as variáveis trabalhadas nesse capítulo.

```
>>> print (a)
```

```
6
```

```
>>> print (b)
```

```
1.5
```

```
>>> print (c)
```

```
fabrica de noobs
>>> print (d)
curso desenvolvido por :
 fabrica de noobs
```

Podemos imprimir mais de uma variável em uma única vez utilizando, dentro dos parênteses, uma vírgula entre cada variável. Por exemplo:

```
>>> print (a, c)
6 fabrica de noobs
```

Além disso, é possível converter um tipo de variável para outro. No exemplo abaixo, queremos a parte inteira do número *5.777*.

```
>>> int (5.777)
5
```

Caso queiramos imprimir, além das variáveis, algum tipo de texto, podemos fazê-lo com o uso de aspas dentro do comando *print*. Por exemplo:

```
>>> a = 17
>>> print ("O valor de a é",a)
O valor de a é 17
```

IV. Operações Matemáticas

O interpretador Python pode servir também como uma excelente calculadora matemática, que abrange desde as 4 operações fundamentais até números complexos. Ao realizar as operações, podemos utilizar tanto variáveis quando valores numéricos.

Uma adição pode ser feita através do operador +. Por exemplo:

```
>>> 2+3
5
>>> 1.5+2.5
4.0
```

Já uma subtração, pode ser feita com o operador -, como no exemplo:

```
>>> 1-3
-2
>>> 5-1
4
```

Multiplicação deve ser feita com o operador *, como mostrado abaixo.

```
>>> 3*5
15
>>> 3*-5
-15
```

E divisão deve ser feita com o operador / e // para divisão inteira. Por exemplo:

```
>>> 6/2
3.0
>>> 7/3
2.3333333333333335
>>> int (7/3)
2
```

Podemos obter o resto de uma divisão através do operador %, como no exemplo abaixo.

```
>>> 7%3
```

1

A exponenciação é feita utilizando o operador **. Por exemplo:

```
>>> 2**6
64
```

Não existe um comando específico para radiciação em Python. Ela deve ser feita elevando-se o radicando ao inverso do radicador. Por exemplo, para a raiz quadrada e cúbica:

```
>>> 2**0.5
1.4142135623730951
>>> 2**0.33
1.2570133745218284
```

Ao definirmos uma operação matemática, será seguida a mesma ordem convencionada na álgebra: divisão e multiplicação, adição e subtração. Caso quisermos uma ordem diferente, devemos especificá-la com uso dos parênteses. Não há problemas em colocar vários parênteses, um dentro do outro. Veja abaixo:

```
>>> 2+2/2
3.0
>>> (2+2)/2
2.0
```

Neste último exemplo, definimos algumas variáveis e realizamos operações matemáticas com elas.

```
>>> a = 3
>>> b = 2.5
>>> c = 9
>>> (a+c)/b
4.8
```

V. Entrada de Dados

É fundamental que, para rodar um programa, coletemos informações inseridas pelo usuário.

Em Python, isso pode ser feito através do comando input. Ele apresenta a estrutura *input("Pergunta")*, na qual a mensagem destinada ao usuário é inserida entre aspas e parênteses.

Para salvar o valor recebido em uma variável, podemos utilizar a seguinte sintaxe:

```
>>> a = input ("Qual a sua idade?")
Qual a sua idade?
```

Assim que o usuário inserir o valor, ele será armazenado na variável *a*. Se, em seguida, exibirmos a variável em questão, receberemos:

```
>>> print (a)
20
```

Para evitar entradas erradas, podemos condicionar a entrada de dados para determinado modelo. Por exemplo, se quisermos que um valor seja transformado em inteiro após ser recebido, podemos utilizar:

```
>>> a = int(input("Qual a sua idade?"))
Qual a sua idade?
```

Essa conversão pode ser útil em operações matemáticas, evitando erros posteriores.

Por padrão, a entrada *input* já condiciona o resultado à uma string. Porém, podemos recebe-la (ou converter determinado valor para srting) utilizando o comando *str*.

```
>>> b = str(input("Qual o seu nome?"))
Qual o seu nome?
```

Veja abaixo um programa capaz de receber o nome e idade do usuário e, em seguida, exibi-la na tela.

```
a = str(input("Qual o seu nome? "))
b = int(input("Qual a sua idade? "))
print ("Meu nome é ", a, " e eu tenho", b, " anos.")
```

Tal código, após sua compilação, irá apresentar o seguinte resultado, mostrado abaixo.

```
Qual o seu nome? João
Qual a sua idade? 25
Meu nome é  ão  e eu tenho 25  anos.
```

Não é possível somarmos valores de texto, mas o python permite que estes sejam mesclados ou exibidos diversas vezes na tela. Por exemplo:

```
>>> a = "João"

>>> b = " da Silva"

>>> a + b

'João da Silva'
```

Para mesclarmos, podemos utilizar o sinal de multiplicação seguido do número de vezes em que a palavra deve ser repetida. Veja abaixo, como continuação do código anterior:

```
>>> a * 3

'JoãoJoãoJoão'
```

Por fim, observe o código abaixo, que utiliza os conceitos estudados neste capítulo.

```
a = str(input("Qual o seu nome? "))

b = str(input ("Qual seu sobrenome?"))

c = int(input("Qual a sua idade? "))

nome_completo = a + " " + b

print ("Meu nome é ", nome_completo, "e tenho ", c, " anos")
```

Veja agora sua compilação.

```
Qual o seu nome? João
Qual seu sobrenome?Silva
Qual a sua idade? 17
Meu nome é  João Silva e tenho  17  anos
>>>
```

VI. Tomadas de Decisão e Comparadores

Comparadores, como o nome diz, nos permitem comparar dois valores. Em Python, suas correspondências são:

==	Igual
!=	Diferente
<	Menor que
>	Maior que
<=	Menor ou igual
>=	Maior ou igual
not	Negação (inverso)
and	Ambas
or	Ao menos

Podemos usar os 6 primeiros no terminal para verificar igualdades, por exemplo:

```
>>> 5>7
False
>>> 2+2==4
True
>>> 5+5!=10
False
>>> 5>5
False
>>> 5>=5
True
```

Tomadas de decisão permitem que o programa execute determinada parte do código caso algo seja ou não verificado. Em Python, isso é feito através dos comandos *if*, *else*, *elif*, *while* e *for*.

Os comandos *if* e else correspondem, respectivamente, à *se* e *se não*, enquanto que o *elif* pode ser usado em uma situação em que contenham várias condições, sendo uma verificada após a outra.

Por exemplo, o código abaixo pede para que o usuário insira o resultado de uma operação matemática, e informa se ele está certo ou errado.

```
resultado = int(input("Quanto é 20 - 13?"))
```

```
if resultado == 7:
   print ("Resposta certa!")
else:
   print ("Resposta errada!")
```

Veja agora um código que classifica o usuário em criança, adolescente e adulto, conforme sua idade:

```
idade = int(input("Insira a sua idade: "))
if idade <= 10:
   print ("Criança.")
elif idade <= 17:
   print ("Adolescente.")
else:
   print ("Adulto.")
```

Note que usamos o *elif* na segunda condição. Observe o que aconteceria, se ao invés dele, usássemos o *if*:

```
Insira a sua idade: 8
Criança.
Adolescente.
```

Logo, concluímos que o comando *elif* faz com que, se verdadeiro, todas as condições verdadeiras anteriores sejam ignoradas.

Através dos comparadores *and* e *or*, podemos verificar se duas condições ocorrem simultaneamente ou apenas uma delas acontece.

Por exemplo:

```
idade1 = int(input("Insira a idade do primeiro usuário: "))
idade2 = int(input("Insira a idade do segundo usuário: "))
if (idade1 >= 18) and (idade2 >= 18):
   print ("Ambos podem dirigir.")
elif (idade1 >=18) and (idade2 < 18):
   print ("Somente o primeiro usuáiro pode dirigir")
elif (idade1 < 18) and (idade2 >= 18):
   print ("Somente o segundo usuáiro pode dirigir")
elif (idade1 < 18) and (idade2 < 18):
```

```
    print ("Ninguém pode dirigir")
```

Observe sua compilação para alguns valores:

```
Insira a idade do primeiro usuário: 10
Insira a idade do segundo usuário: 32
Somente o segundo usuáiro pode dirigir

Insira a idade do primeiro usuário: 35
Insira a idade do segundo usuário: 17
Somente o primeiro usuáiro pode dirigir

Insira a idade do primeiro usuário: 19
Insira a idade do segundo usuário: 20
Ambos podem dirigir.

Insira a idade do primeiro usuário: 5
Insira a idade do segundo usuário: 6
Ninguém pode dirigir
```

Agora, observe um código que recebe a nota do usuário em 4 bimestres, calcula uma média, verifica sua aprovação ou reprovação, o parabeniza em caso de bom desempenho. Compile-o e observe os resultados.

```
nota1 = int(input("Insira a primeira nota: "))

nota2 = int(input("Insira a segunda nota: "))

nota3 = int(input("Insira a terceira nota: "))

nota4 = int(input("Insira a quarta nota: "))


media = (nota1 + nota2 + nota3 + nota4)/4


if media >= 6:
    print ("Aprovado! Sua média é:", media)
    if media >=9:
        print("Parabéns!")
else:
    print ("Reprovado! Sua média é:", media)
```

Por fim, os comandos *while* e *for* significam, respectivamente, *enquanto* e *para*. Eles são usados em situações nas quais determinada sequência precisa ser executada enquanto outra for verdadeira, ou durante determinado número de vezes.

Abaixo, incrementamos o código que verificava se a resposta do cálculo matemático estava certa ou errada, fazendo com que a execução só termine no momento em que a resposta estiver certa.

Note que precisamos definir um valor arbitrário diferente da resposta para a variável, a fim de impedir que o código nos retorne um erro.

```
resultado = 0
while resultado != 7:
    resultado = int(input("Quanto é 20 - 13?"))
    if resultado == 7:
        print ("Resposta certa!")
    else:
        print ("Resposta errada!")
print("Parabéns, siga para a próxima questão!")
```

O comparador *for* pode ser usado para repetir uma operação determinado número de vezes. Para cada for, utilizamos um contador (dado por uma variável), que assume valores dentro de um intervalo.

Por exemplo, o comando abaixo conta até 10.

```
for x in range(0, 11):
    print (x)
```

Em sua execução, a variável *x* assume valores de *0* até *10*, e, para cada um deles, imprime seu próprio valor.

```
0
1
2
3
4
5
6
7
8
9
10
>>>
```

Podemos incrementá-lo, de forma que o programa eleve um valor inserido pelo usuário aos valores de 1 até 10.

```
a = int (input("Insira o valor: "))
```

```
for x in range(0, 11):
    print (a ** x)
print("Operação terminada!")
```

Eis sua compilação:

```
Insira o valor: 2
1
2
4
8
16
32
64
128
256
512
1024
Operação terminada!
```

VII. Listas

Listas são sequências de variáveis. Após sua definição, elas podem ser modificadas.

Para criar uma lista, devemos nomeá-la (assim como uma variável) e inserir seus elementos entre colchetes, separando-os por vírgulas. Por exemplo:

```
>>> a = [1,2,3]
>>> print (a)
[1, 2, 3]
```

O comando *len* pode ser usado para recebermos o tamanho de uma lista.

```
>>> len (a)
3
```

O comando *append* pode ser usado para adicionar um valor específico ao final da lista, enquanto que o comando *extend* faz o mesmo para uma lista inteira.

```
>>> a.append(4)
>>> print (a)
[1, 2, 3, 4]
>>> a.extend([5,6,7])
>>> print (a)
[1, 2, 3, 4, 5, 6, 7]
```

Podemos inserir um valor em determinada posição da lista, substituindo-o pelo antigo, dessa forma:

```
>>> a[2]=2.5
>>> print (a)
[1, 2, 2.5, 4, 5, 6, 7]
```

E deletá-lo utilizando o comando del:

```
>>> del a[2]
>>> print (a)
[1, 2, 4, 5, 6]
```

Listas também podem ser mescladas com o uso do operador de soma:

```
>>> a=[1,2,3]
>>> b=[4,5,6]
>>> c = a+b
>>> print (c)
[1, 2, 3, 4, 5, 6]
```

Observe o programa abaixo, capaz de receber quantas notas o autor desejar e imprimi-las em uma lista.

```
listanotas = []
numnotas = int(input("Quantas notas deseja cadastrar? "))
for i in range (0,numnotas):
    nota = int(input("Insira a nota: "))
    listanotas.append(nota)
print ("Processo concluído! Suas notas são: ", listanotas)
```

Veja agora sua execução:

```
Quantas notas deseja cadastrar? 7
Insira a nota: 6
Insira a nota: 0
Insira a nota: 4
Insira a nota: 10
Insira a nota: 9
Insira a nota: 4
Insira a nota: 5
Processo concluído! Suas notas são:  [6, 0, 4, 10, 9, 4, 5]
>>>
```

VIII. Dicionários

Um dicionário é um conjunto de elementos, chamados chaves, e suas correspondências, denominadas conteúdos ou índices.

Por exemplo, podemos criar um dicionário com números e suas respectivas formas em números romanos. Para isso, ele deverá seguir o modelo *dicionário = [chave1 : conteúdo1, chave2 : conteúdo2, chave N : conteúdo N]*, como mostrado abaixo.

```
>>> numeros = {1 : "I", 2 : "II"}

>>> numeros

{1: 'I', 2: 'II'}
```

Podemos adicionar um valor ao dicionário realizando:

```
>>> numeros[3] = "III"

>>> numeros

{1: 'I', 2: 'II', 3: 'III'}
```

Assim como podemos deletá-lo utilizando o comando *del*:

```
>>> del numeros [1]

>>> numeros

{2: 'II', 3: 'III'}
```

E consultar uma correspondência do dicionário utilizando:

```
>>> print (numeros [2])

II
```

Também é possível exibir somente as chaves ou valores através dos comandos *keys* e *values*, respectivamente.

```
>>> numeros.keys()

dict_keys([1, 2, 3])

>>> numeros.values ()

dict_values(['I', 'II', 'III'])
```

Ou ainda exibir uma correspondência lado a lado com o comando *items*.

```
>>> numeros.items()

dict_items([(1, 'I'), (2, 'II'), (3, 'III')])
```

IX. Funções

Imagine que, em um programa, determinado trecho (de várias linhas) precisa ser repetido diversas vezes. Se pudermos fazer com esse trecho seja "chamado" com um único comando, ganhamos tempo, espaço em disco, além de deixarmos o código como um todo mais elegante e menos sujeito a erros.

Essa é a utilidade das funções. Para definir uma função, utilizamos o comando *def*. Essa função deverá, obrigatoriamente, nos retornar algum valor, o qual é especificado com o comando *return*. Por exemplo:

```
def soma (x,y):
   return x + y
```

Sendo assim, a função precisa de dois valores (*x* e *y*), que deverão ser somados. Chamamos uma função utilizando seu nome, seguido destes valores:

```
>>> soma (3,5)
8
```

Outro exemplo:

```
def quadrado (x):
   return x**2
>>> quadrado (8)
64
```

Veja agora o exemplo abaixo, que cria uma função para realizar as 4 operações fundamentais com dois números.

```
soma = 0
subtracao = 0
multiplicacao = 0
divisao = 0
def operacoes (x,y):
   soma = (x + y)
   subtracao = x - y
   multiplicacao = x * y
   divisao = x / y
```

```
print ("A soma é ", soma)
print ("A subtracao é ", subtracao)
print ("A multiplicação é ", multiplicacao)
print ("A divisão é ", divisao)
```

E sua compilação:

```
>>> operacoes(5,3)
A soma é  8
A subtracao é  2
A multiplicação é  15
A divisão é  1.6666666666666667
```

X. Módulos

A linguagem Python já possui um conjunto de funções prontas para serem agregadas e utilizadas em seus programas. Essas funções estão agrupadas em estruturas denominadas módulos.

Para utilizar um módulo, precisamos importa-lo para o programa. Isso é feito através do comando *import*. No exemplo, importamos o módulo *math*.

```
>>> import math
```

Caso desejarmos saber mais sobre as funções disponíveis em um módulo, podemos usar o comando help:

```
>>> help (math)
```

Isso nos retornará uma lista de todas as funções presentes e como chamar cada uma delas.

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    acosh(...)
        acosh(x)

        Return the inverse hyperbolic cosine of x.

    asin(...)
        asin(x)

        Return the arc sine (measured in radians) of x.
```

Módulos podem ser uma ferramenta útil caso seja necessário realizar um conjunto específico de operações como por exemplo, valores trigonométricos. Estes já estão presentes no módulo *math*, e caso queria criar um programa com eles, não será necessário criar suas próprias funções.

Podemos encontrar uma lista de módulos em https://docs.python.org/3/py-modindex.html

Também é possível criar seu próprio módulo. Para isso, crie um arquivo com as funções que estarão presentes em seu módulo e salve-o com a extensão *.py*.

Em seguida, basta colocá-lo na mesma pasta de outro script para poder importa-lo.